To the Graduate Council:

I am submitting herewith a thesis written by Michael D. Womac entitled "Radiation Effects on the Performance of a Mixed Mode Integrated Circuit." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

E. J. Kennedy, Major Professor

We have read this thesis
and recommend its acceptance

T. V. Blalock

J. M. Rochelle

Accepted for the Council:

Associate Vice Chancellor and
Dean of The Graduate School

RADIATION EFFECTS ON THE PERFORMANCE OF A MIXED MODE INTEGRATED CIRCUIT

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Michael D. Womac
May 1996

DEDICATION

This thesis is dedicated to my wife

Mrs. D. Ann Womac

who has given me the opportunity to pursue my education.

ABSTRACT

The basis of this thesis is centered around the performance characterization of an ASIC (Application Specific Integrated Circuit) called the ATA chip (AMU TAC ADC) before and after irradiation. A variety of tests are performed to describe the capabilities of this device, including Integral Non-Linearity, Differential Non-linearity, noise, and gain. Discussions of how tests are performed and how the hardware was implemented is given. In addition to performance concerns, a method of reducing various types of input pedestals is introduced and evaluated. Also, some statistics resulting from a fabrication run by Orbit Semiconductor 1.2μ N-Well CMOS process of about 2700 devices is presented.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
THE ATA CHIP 2
ANALOG MEMORY UNIT 3
TIME-TO-AMPLITUDE CONVERTER 4
ANALOG TO DIGITAL CONVERTER 5
IC PROCESS 6
PEDESTAL CORRECTION METHOD 7
CONTENT OF THE THESIS 7

CHAPTER 2 RADIATION EFFECTS ON CMOS 8
TYPES OF RADIATION 9
BASIC EFFECTS OF RADIATION IN SEMICONDUCTORS 10
RADIATION EFFECTS IN TRANSISTORS 12
BIPOLAR 13
CMOS 13

CHAPTER 3 PERFORMANCE CHARACTERIZATION DESCRIPTION 18
LINEARITY 18
INTEGRAL NON-LINEARITY 19
DIFFERENTIAL NON-LINEARITY 21
NOISE 25
GAIN 26

CHAPTER 4 TEST DESCRIPTIONS 27
FAILURE TESTING DESCRIPTIONS 27
AMU/ADC TEST DESCRIPTION 29
TAC/ADC TEST DESCRIPTION 30
FIRMWARE PROGRAM DESCRIPTIONS 32
AMU 32
TAC 34
COMPUTER PROGRAM 37
PERFORMANCE TESTING DESCRIPTIONS 38
FIRMWARE PROGRAM MODIFICATION DESCRIPTIONS 40
TAC 41
COMPUTER PROGRAM DESCRIPTIONS 43
INL PROGRAM DESCRIPTIONS 43
DNL PROGRAM DESCRIPTIONS 46

CHAPTER 5 RESULTS 48
FAILURE ANALYSIS RESULTS 48
INL AMU/ADC RESULTS 52
INL TAC/ADC RESULTS 54
DNL AMU/ADC RESULTS 57
DNL TAC/ADC RESULTS 62
NOISE TAC/ADC RESULTS 70
GAIN: AMU/ADC RESULTS 70
GAIN: TAC/ADC RESULTS 70

CHAPTER 6 PEDESTAL CORRECTION: A NOISE REDUCTION METHOD 75
SOURCES OF COHERENT NOISE 76
THEORY 76
TYPES OF TESTS 77
THE METHOD 78
BACKGROUND NOISE RESULTS 79

CHAPTER 7 CONCLUSION 82
YIELD 82
LINEARITY 82
NOISE 84
GAIN 84
PEDESTAL CORRECTION 85
FINAL THOUGHTS 85

REFERENCES 86

APPENDICES
APPENDIX A 90
APPENDIX B 97
APPENDIX C 144
APPENDIX D 164
APPENDIX E 184
APPENDIX F 261

VITA 270

TABLE OF FIGURES

CHAPTER 1
FIGURE 1 - 1. BASIC CALORIMETER CONFIGURATION. 2
FIGURE 1 - 2. ATA CHIP BLOCK DIAGRAM. 3
FIGURE 1 - 3. OVERALL AMU/ADC SECTION BLOCK DIAGRAM. 3
FIGURE 1 - 4. DETAILED AMU PIPE SCHEMATIC DIAGRAM. 4
FIGURE 1 - 5. TAC/ADC SECTION OF THE ATA CHIP. 5
FIGURE 1 - 6. ATA ADC SECTION SIMPLIFIED BLOCK DIAGRAM. 6

CHAPTER 2
FIGURE 2 - 1. PLOT OF PARTICLE REACTIONS VERSES PHOTON ENERGY AND ATOMIC NUMBER [SOURCE: SROUR2]. 11
FIGURE 2 - 2. SIMPLE DIAGRAM OF A CRYSTAL STRUCTURE WITH DIFFERENT TYPES OF IRREGULARITIES [SOURCE: SROUR4]. 12
FIGURE 2 - 3. LOCATION DIAGRAM OF NOT AND NIT IN THE SI AND SIO2 CONFIGURATION [SOURCE: SZE1]. 14
FIGURE 2 - 4. SCHEMATIC REPRESENTATION OF ELECTRON-HOLE GENERATION FOR POSITIVE BIAS DUE TO IONIZING RADIATION. 15
FIGURE 2 - 5. VT SHIFTS AS A FUNCTION OF RADIATION DOSE [SOURCE: SROUR5]. 17

CHAPTER 3
FIGURE 3 - 1. ILLUSTRATION OF INL ERROR. 19
FIGURE 3 - 2. CHANNEL WIDTH ANALOGY 22
FIGURE 3 - 3. TIMING DIAGRAM SHOWING SIGNAL RELATIONSHIPS OF THE ATA CHIP ADC AND THE RAMP GENERATOR. 23
FIGURE 3 - 4. SIGNAL RELATIONSHIPS BETWEEN COMPUTER AND PULSE GENERATOR. 24

CHAPTER 4
FIGURE 4 - 1. TEST FIXTURE AS USED IN THE FAILURE ANALYSIS TESTING. 28
FIGURE 4 - 2. AMU/ADC CONFIGURATION BLOCK DIAGRAM. 29
FIGURE 4 - 3. EXAMPLE OF A PULSE SEQUENCE SENT TO THE TAC FOR FUNCTIONALITY VERIFICATION. 30
FIGURE 4 - 4. TIMING DIAGRAM FOR THE AMU WRITE SECTION OF THE FIRMWARE. 32
FIGURE 4 - 5. TIMING FOR READ PORTION OF AMU TEST. THIS SHOWS HANDSHAKING FOR THE COMPUTER, FIRMWARE AND ATA UNDER TEST 33
FIGURE 4 - 6. TEST FIXTURE AS USED FOR CHARACTERIZATION TESTING 40
FIGURE 4 - 7. TIMING DIAGRAM FOR TAC FIRMWARE. 41
FIGURE 4 - 8. SHOWS THE AMU TESTING CONFIGURATION FOR LIN3.EXE PROGRAM 44

CHPATER 5
FIGURE 5 - 1. CHIP #4 PRE-RADIATION INL RESULTS. 53
FIGURE 5 - 2. MID-SCALE INL TAKEN AT 2.5V FOR ALL CHIPS PRIOR TO IRRADIATION. 53
FIGURE 5 - 3. CHIP #4 SHOWING RESULTS PRIOR TO AND AFTER IRRADIATION. 55
FIGURE 5 - 4. CHIP #9 PRE AND POST-RADIATION AMU/ADC INL RESULTS. 55
FIGURE 5 - 5. CHIP #9 TAC/ADC PRE-RADIATION INL RESULTS 56
FIGURE 5 - 6. MID-SCALE PRE-RADIATION INL, TAKEN WITH THE INPUT PULSE SEQUENCE SEPARATED AT 120 NS, FOR ALL CHIPS. 56
FIGURE 5 - 7. CHIP #4 PRE AND POST-RADIATION TAC/ADC INL RESULTS. 58
FIGURE 5 - 8. CHIP #9 PRE AND POST-RADIATION TAC/ADC INL RESULTS. 58
FIGURE 5 - 9. CHIP #4 PRE-RADIATION DNL RESULTS. 59

FIGURE 5 - 10. CHIP #9 PRE-RADIATION DNL RESULTS SHOWING THE DIFFERENT EFFECTS OF THE CASE BEING GROUNDED. (A) UNGROUNDED CASE, (B) GROUNDED CASE.....59
FIGURE 5 - 11. CHIP #4 POST-RADIATION DNL RESULTS.....61
FIGURE 5 - 12. CHIP #9 POST-RADIATION DNL RESULTS.....62
FIGURE 5 - 13. CHIP #4 PRE-RADIATION DNL RESULTS.....63
FIGURE 5 - 14. CHIP #9 PRE-RADIATION DNL RESULTS.....63
FIGURE 5 - 15. CHIP #4 POST-RADIATION DNL RESULTS.....64
FIGURE 5 - 16. CHIP #9 POST-RADIATION DNL RESULTS.....64
FIGURE 5 - 17. CHIP #4 PRE-RADIATION NOISE RESULTS INDICATING NOISE INTRODUCED AT THE INPUT.....66
FIGURE 5 - 18. CHIP #7 NOISE RESULTS BEFORE FILTERING INPUT OF THE AMU/ADC.....66
FIGURE 5 - 19. CHIP #7 NOISE RESULTS AFTER FILTERING THE INPUT TO THE AMU/ADC. ..67
FIGURE 5 - 20. CHIP #4 PRE-RADIATION NOISE RESULTS.....68
FIGURE 5 - 21. CHIP #9 PRE-RADIATION NOISE RESULTS.....68
FIGURE 5 - 22. CHIP #4 POST-RADIATION NOISE RESULTS.....69
FIGURE 5 - 23. CHIP #9 POST-RADIATION NOISE RESULTS.....69
FIGURE 5 - 24. CHIP #4 PRE-RADIATION NOISE RESULTS.....71
FIGURE 5 - 25. CHIP #9 PRE-RADIATION NOISE RESULTS.....71
FIGURE 5 - 26. CHIP #4 POST-RADIATION NOISE RESULTS.....72
FIGURE 5 - 27. CHIP #9 POST-RADIATION NOISE RESULTS.....72
FIGURE 5 - 28. AMU/ADC GAIN FOR ALL CHIPS BEFORE IRRADIATION.....73
FIGURE 5 - 29. AMU/ADC GAIN - PRE & POST RADIATION COMPARISON (FOR IRRADIATED CHIPS ONLY).....73
FIGURE 5 - 30. TAC/ADC GAIN (PRE-RADIATION FOR ALL CHIPS)).....74
FIGURE 5 - 31. TAC/ADC GAIN - PRE & POST RADIATION COMPARISON (FOR IRRADIATED CHIPS ONLY).....74

CHAPTER 6
FIGURE 6 - 1. SINGLE PIPE-LINE SUBTRACTION METHOD.....75
FIGURE 6 - 2. SCHEMATIC DIAGRAM OF POWER SUPPLY INJECTED NOISE.....78

Table of Acronyms

ADC:	Analog to Digital converter
AMU:	Analog Memory Unit
TAC:	Time to Amplitude Converter
ATA:	AMU, TAC, ADC
DAQ:	Data Acquisition System
DAC:	Digital to Analog Converter
CPLD:	Complex Programmable Logic Device
INL:	Integral Non-Linearity
FSR:	Full Scale Range
DNL:	Differential Non-Linearity
CERN:	European Center of Nuclear Research
DUT:	Device Under Test
CPU:	Central Processing Unit
FSC:	Full Scale Count
F/F:	Flip-Flop
MSB:	Most Significant Bit
CMOS:	Complimentery Metal Oxide Semiconductor
Si:	Silicon
SiO_2:	Silicon Di-Oxide
AKA:	Also Known As
PMT:	Photomulitiplier Tube
RMS:	Root Mean Squared

Chapter 1 Introduction

In 1994, an experiment called the WA98 experiment was conducted at the European Center for Nuclear Research (which has the French acronym of CERN) in Geneva, Switzerland. The purpose of this experiment was to measure the radiation of direct photons to investigate quark-gluon plasmas (QCG). The formation of quark-gluon plasmas due to heating from the release of kinetic energy is predicted by the Standard Model of physics. According to the 'Big Bang' theory the universe existed as a QCG at about 1 - 3 μS after the 'bang'.

A QCG is formed by the collision of sub-atomic particles called hadrons. Hadrons are composed of quarks and gluons. When enough hadrons collide with sufficient energy a quark-gluon plasma results consisting of unbound quarks and gluons. Since this plasma is very hot, it radiates high-energy photons (black-body radiation). The energies of the photons are measured by the lead-glass calorimeter. The photon energy is absorbed in the lead-glass and the energy is re-emitted as visible light that is detected by photomultiplier tubes (PMT). The amount of light emitted is proportional to the amount of energy lost. So, if the glass is thick enough, the total amount of energy can be measured. The measurement of the energy is accomplished by a calorimeter (i.e. a device to measure heat or energy). This calorimeter is in the form of a wall of lead glass and the electronic system developed at ORNL. A basic depiction of the calorimeter electronics can be seen in the following Figure 1 - 1.

The electronics for this system is comprised of a two chip set. First there is the

pre-amplifier chip, which prepares the timing and analog signals for the second ATA (AMU, TAC, ADC) chip. Next, the ATA chip is used to receive voltages and timing sequences that are converted to digital representations for processing. The ATA chip will be the focus of this thesis.

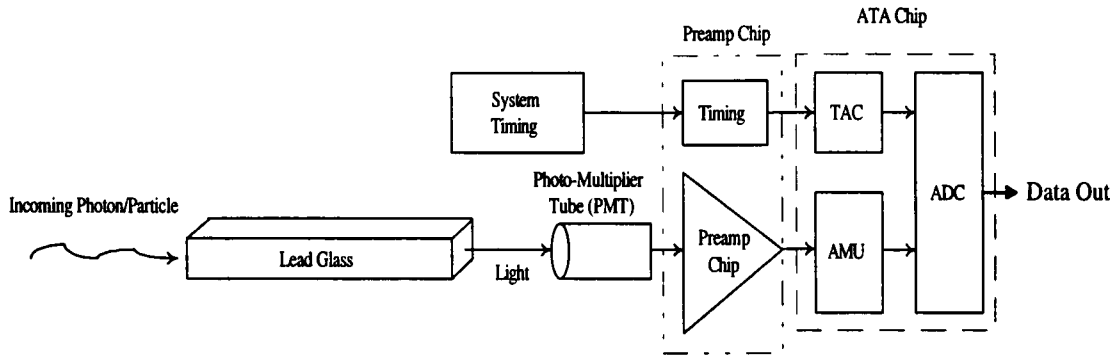


Figure 1 - 1. Basic Calorimeter configuration.

The ATA Chip

The ATA chip is comprised of 8 Time to Amplitude Converters (TACs), a 16x16 Analog Memory Unit array (AMU), and a 24 channel Analog to Digital Converter (ADC) (there is one channel dedicated for each TAC and AMU pipe [a column of AMU cells]). A block diagram of the ATA chip can be seen in Figure 1 - 2.

The first chip of the two chip set, the preamp chip, receives its input from a photo multiplier tube and prepares it for the input to the ATA chip. The ATA chip takes the voltages that are present at the input of the AMU and the timing pulses received by the TAC and converts the outputs of the AMU's and the TAC's via the dedicated ADC's to the resulting digital data. This data is sent to the data processing center and used for analysis.

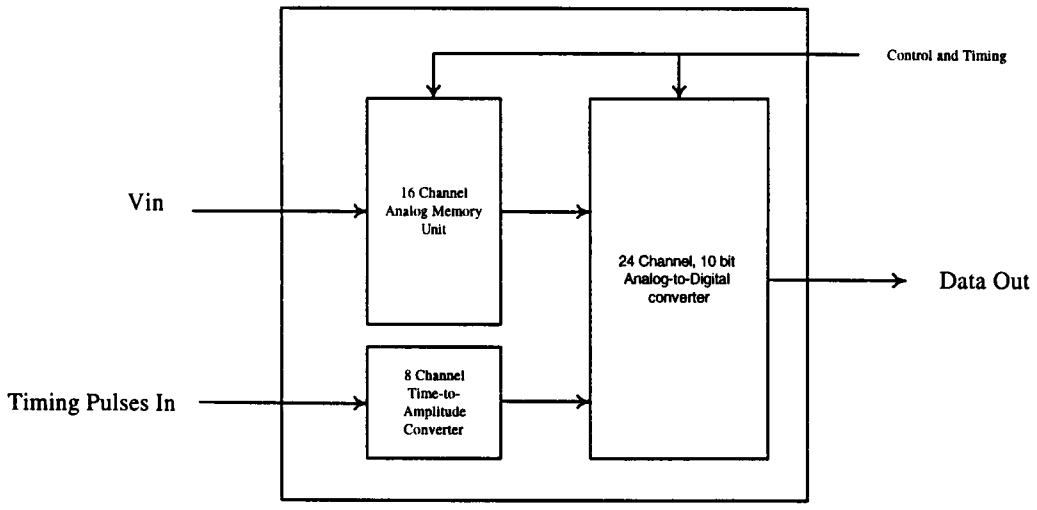


Figure 1 - 2. ATA chip block diagram.

Analog Memory Unit

A block diagram for the AMU/ADC section can be seen in Figure 1 - 3.

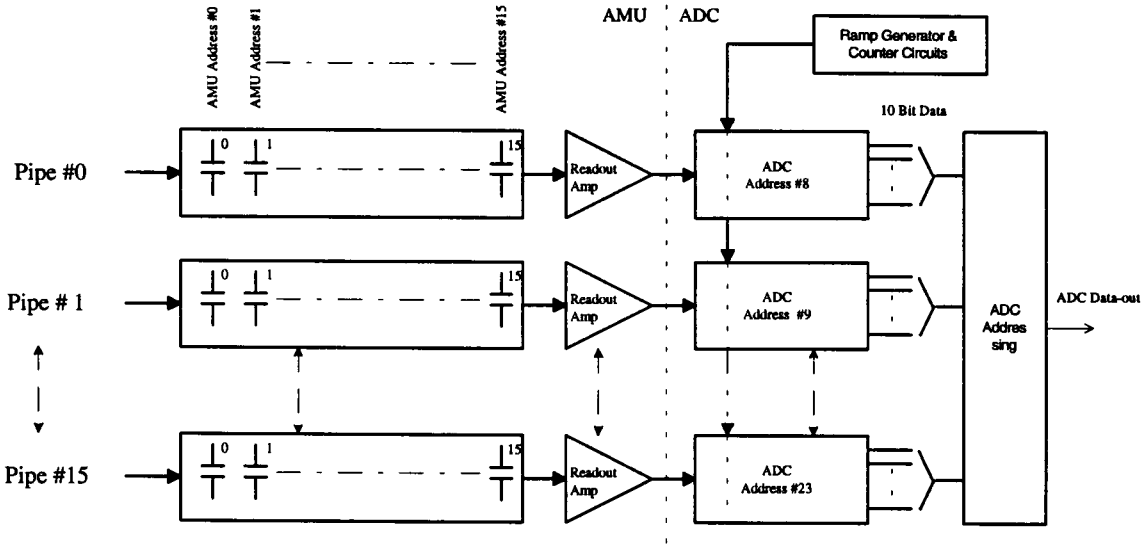


Figure 1 - 3. Overall AMU/ADC section block diagram.

This section of circuitry is a capacitor array of memory cells that form a 16x16 matrix.

This matrix is organized in such a fashion that each row is common to an operational

amplifier. This is called a pipe. Each pipe is connected to a dedicated 10 bit ADC channel. In Figure 1 - 4, a more detailed diagram for one AMU pipe can be seen. Each column is addressed via a four bit address decoder that allows writing of the entire 16 cells common to that write address simultaneously. Likewise, each row is addressed by separate addressing circuits that allows a connection for each cell to its operational amplifier. That is, all 16 amplifiers have connection to one cell within its pipe at a time.

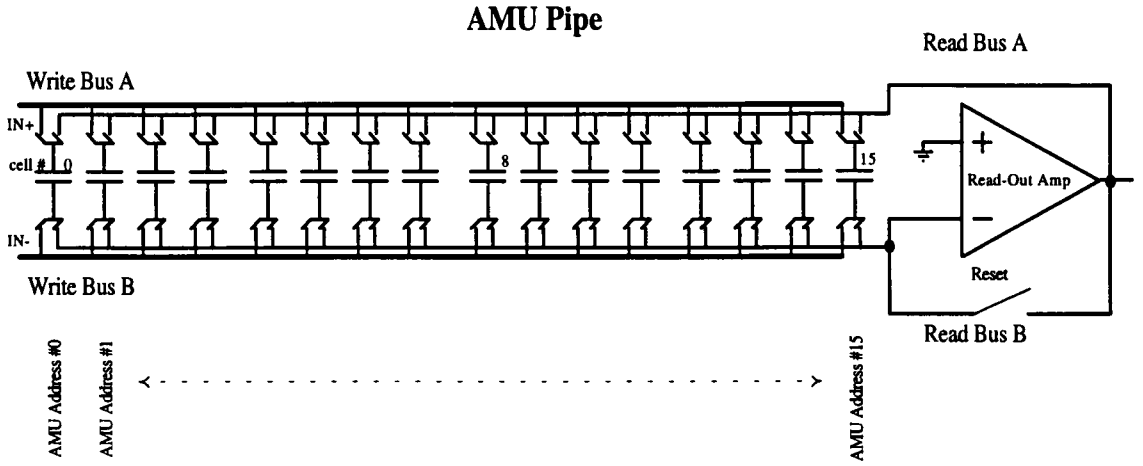


Figure 1 - 4. Detailed AMU Pipe schematic diagram.

Time-to-Amplitude Converter

A functional block diagram of the TAC/ADC section of the ATA chip can be seen in Figure 1 - 5. This section contains eight channels of time-to-amplitude conversion. Each channel has a dedicated ADC channel. As with the AMU/ADC section, the ADC's are addressed one at a time for output to the data bus. The TAC ADC's addressing starts with address zero and goes to address seven. Address eight is where the AMU ADC's addressing begins.

For the time-to-amplitude conversion, a start pulse is received (refer to Figure 1 - 5). This initiates a capacitor to begin charging by an amplifier. This amplifier provides charge for the capacitor at a constant rate. The rate at which the capacitor is charged allows the TAC to convert a time difference of approximately 0 to 200 ns to a voltage range between 0V and 5V. The magnitude of this voltage is proportional to the amount of time between the start and the stop pulse. Hence, time-to-amplitude conversion occurs. This voltage is then present at the output amplifier until it is cleared by an input clear command. During the hold time, the resulting voltage can be converted.

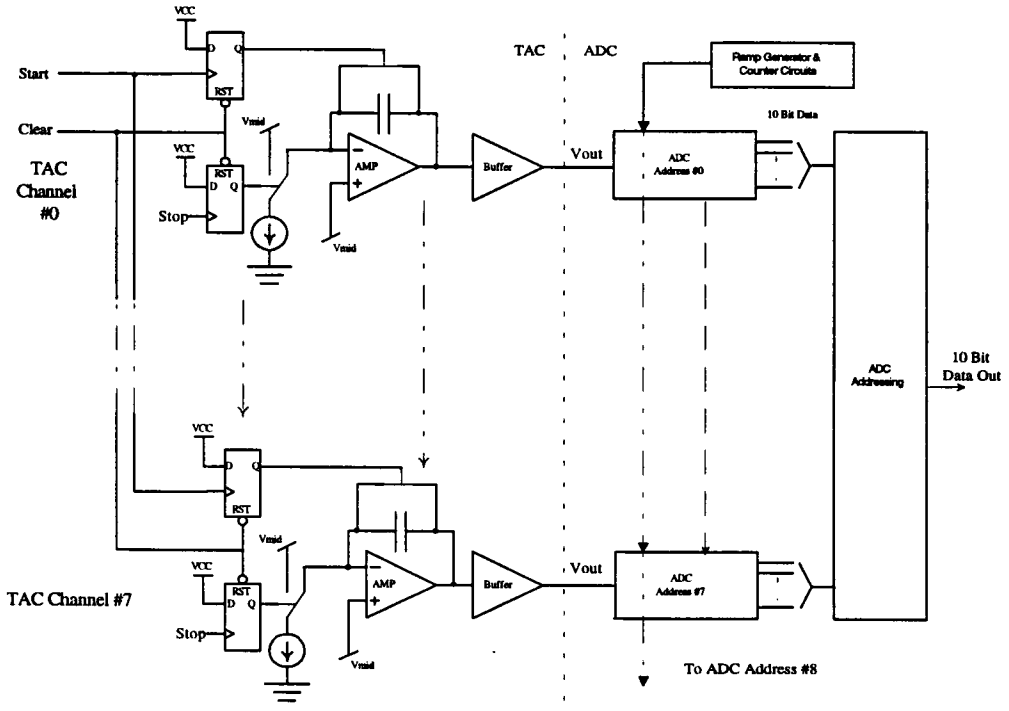


Figure 1 - 5. TAC/ADC section of the ATA chip.

Analog to Digital Converter

The ADC is a 10 bit Wilkinson type [WILKINSON1]. The ADC section contains 24 channels of conversion capability. The dynamic range is 4.5V, and the device is

designed to run at a clock frequency of 80 MHz. A simplified block diagram can be seen in Figure 1 - 6.

This ADC configuration allows the conversion of all 24 channels simultaneously. The conversion is initiated by sending a ramp trigger signal to the ATA chip. This signal starts an integrator that takes approximately 15 μS (depending on clock frequency) to charge to the maximum voltage of 4.5V. The resulting ramp voltage is common to all channels, and is thus compared to the respective channels' input voltage. The comparator is fired at the point in which the voltages are equal. The latch is activated to hold the count that is provided by the common 10-bit counter. This count is the converted code that will be delivered to represent the input voltage present at each respective channel.

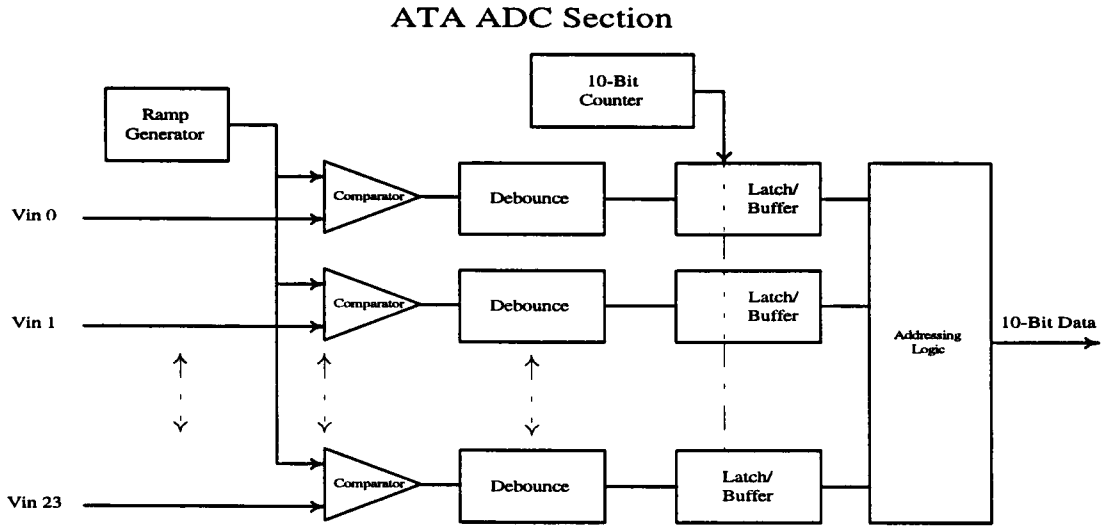


Figure 1 - 6. ATA ADC Section simplified block diagram.

IC Process

The ATA chip was processed by Orbit Semiconductor in Sunnyvale, California using their 1.2 μm CMOS N-Well process. The individual die size is 4.97 mm X 3.74

mm. The ATA chip was fabricated on the same wafer as the pre-amp chip. The die were alternately placed on the same wafer to reduce masking costs (only one set of masks were made for both devices instead of two mask sets for two chips). There were approximately 2700 chips fabricated for both types. There were approximately 150 die per wafer from a total of 25 wafers fabricated.

Pedestal Correction Method

Physicists have concerns with electronic induced signals being received as being part of the particle measurement. Anything that unexpectedly adds to the measurement signal is termed a pedestal. So, it would be convenient to be able to measure the pedestal and reduce it by a subtraction process. The pedestal correction method is a technique used to reduce any correlated noise in the AMU. This will be introduced in Chapter 6.

Content of the Thesis

The remainder of this thesis is divided into six more chapters. Chapter 2 introduces the effects of radiation on CMOS circuits. Chapter 3 defines the elements that have been characterized for the ATA integrated circuit. Chapter 4 describes the methods used for characterization. Chapter 5 presents the results of the experiment. Chapter 6 introduces a method of noise reduction for the AMU and the results of the noise reduction method. Finally, Chapter 7, the conclusion, shows a summary of the thesis.

Chapter 2 Radiation Effects on CMOS

The purpose of this chapter is to briefly describe some of the effects that CMOS circuits experience from the damage caused by radiation. The devices being tested were processed using a non-radiation hardened (or "soft" process) 1.2μm N-Well process by Orbit Semiconductor. The ATA chip is to be used as a test vehicle for the radiation effects on this "soft" process, to obtain radiation data for future designs to be implemented. This chapter will describe some basic concepts and effects of radiation on silicon (Si), silicon dioxide (SiO_2), and most importantly CMOS integrated circuits. Much of the information that follows is adapted from [SROUR].

When radiation particles collide with Si and SiO_2, damage is done to the materials. Since there are two different types of materials, there will be different types and different amounts of damage done. This damage can have adverse effects on circuit dc bias (the most important effect) and also on actual circuit performance. There are many different effects that can be observed. Those discussed in this chapter are listed in the following section.

As radiated particles with sufficient energy collide with the crytal structures of the Si and SiO_2,, electron-hole generation results. If this is done in SiO_2, a net positive charge remains due to the loss of an electron (i.e. a positive changed ion remains) in the SiO_2 molecule. This positive charge is the element which causes the most important degradation factor in CMOS, namely threshold voltage (V_t) shift. Due to the change in

device V_t, the circuit dc bias will change. A discussion of what radiation is, and how it causes this and

other problems will be briefly given.

Types of Radiation

Nuclear radiation has several different forms. Listed below is a list of some of the most common types:

- Photons, have the form of:
  - X-Rays
  - γ Rays
- Charged Particles, have the form of:
  - Electrons
  - Protons
  - α particles
  - ions
- Neutrons

There are a variety of different sources for each type of radiation. For this thesis, only gamma (γ) radiation will be discussed, since this was the most significant radiation effect expected in the WA80 experiment. The source used to emit γ radiation for this experiment was a CO^{60} gamma field, located at Argonne National Laboratory in Chicago, Illinois.

A CO^{60} source provides photons with energies of 1.16 MeV and 1.33 MeV. Depending on the amount of energy a photon has will determine how the particles react upon impact. There are three types of interactions [SROUR2]:

- Photoelectric effect

- Compton Scattering
- Positron-Electron Pair Production

The photoelectric effect dominates when the energy of an incoming photon is completely absorbed upon a collision with an atom. From that collision, an electron is released from the atom. As photon energies are increased, another mechanism begins to dominate which is known as Compton scattering. Compton scattering effects are a result from a photon that has more than enough energy to release an electron from an atom. The incident photon gives up a portion of its energy to release the electron from the impacted atom and continues to travel until it is absorbed. The third type of interaction, positron-electron pair production, occurs when a photon strikes its destination at a very high energy and causes a positron and electron to be emitted (a positron is a particle that has the same rest mass of an electron but opposite in charge).

With the source of radiation used in this experiment, the dominant interaction mechanism is Compton scattering. In Figure 2 - 1, a graph relating photon energy, type of interaction, and atomic Z (or atomic number) of the targeted material is shown. It can be seen that with a CO^{60} source which has energies of 1.16 MeV and 1.33 MeV that Compton scattering mechanism dominates.

Basic Effects of Radiation in Semiconductors

For electronic materials, there are two dominant effects which cause damage to silicon material [SROUR3]. They are:

- Displacement
- Ionization

A brief discussion of each type will be described in the following paragraphs.

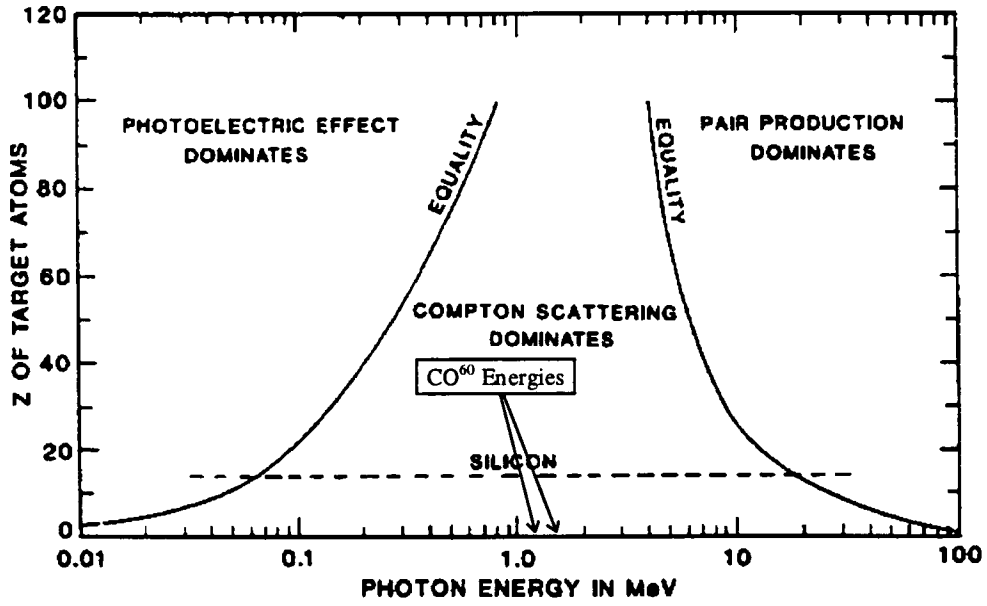


Figure 2 - 1. Plot of Particle Reactions verses Photon Energy and Atomic Number [Source: SROUR2].

Displacement refers to the removal of atoms from the organized lattice structure. There are several different types of displacements. A displaced atom in the crystal results in a vacancy, which can be just one vacancy or it may be multiple vacancies (i.e. divacancy, trivacancy ect.). The displaced atom/s can be left at another location in the crystal structure (i.e. out of the regular crystal structure which is called an interstitial/di-interstitial atom/s) or completely removed depending on the amount of energy that the radiated particle has upon impact. See Figure 2 - 2 below for a diagram of crystal irregularities.

The amount of energy that would be needed to remove an atom from the crystal lattice is ~25eV [SROUR3]. So, if an impinging photon from a CO^{60} source has a

minimum of 25eV of energy at the time of collision, an atom should be displaced in the crystal, thereby creating an irregularity in the crystal structure. This phenomena

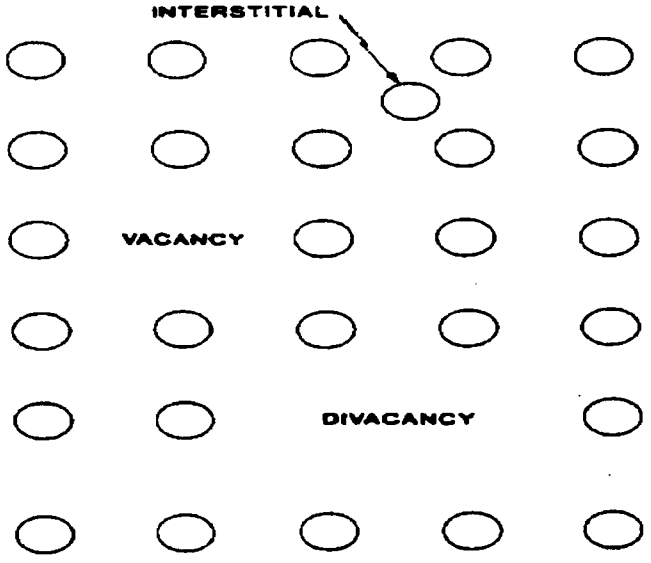


Figure 2 - 2. Simple diagram of a crystal structure with different types of irregularities [Source: SROUR4].

happens also with certain integrated circuit fabrication methods, such as ion implantation, and is an undesirable result of fabrication. In 'Introduction to Microelectronic Fabrication' [JAEGER1], a brief discussion of this type of lattice damage is discussed. For an intricate treatment of this topic see [MA2].

Ionization occurs when an atom has had an electron stripped from its outer shell leaving the atom with a net positive charge (i.e. a hole). This electron-hole pair generation is the main concern for this discussion on radiation effects on CMOS.

Radiation Effects in Transistors

Damage to transistors due to radiation is different for different types of transistors. The bipolar transistor is affected in a way that is dramatically different than its CMOS

counterpart. Due to the composition of the ATA chip being CMOS, only a brief discussion of radiation effects on bipolar transistors will be done. A more lengthy discussion of radiation effects on CMOS devices will be given.

Bipolar

A bipolar transistor is predominately effected by decreased current gain (usually defined as BETA = I_C/I_B) [MA1]. This is primarily due to the reduction of the lifetime of an electron-hole pair in the base region. If this happens, base current increases effectively decreasing the current gain. Equation (2 - 1) [GRAY1] describes the relationship between base current and carrier lifetime,

$$I_B = \frac{Q_e}{\tau_b} \qquad (2 - 1)$$

where Q_e is the amount of minority carrier charge in the base region and τ_b is the lifetime in the base region. Therefore, if τ_b decreases, base current increases (that is if collector current remains the same). Hence, the collector to base current ratio (BETA) will decrease, degrading the performance of the bipolar transistor.

CMOS

The main concern for this portion of the thesis is that of ionization effects pertaining to CMOS devices. Because of the two forms of silicon that comprise CMOS devices, there are two different materials for radiation to effect. SiO_2 is the material in which the most significant effects are located. Therefore, much different effect will be seen in CMOS devices. Listed below are three results of radiation damage in SiO_2 and Si:

- Trapped charge in the oxide ≡ N_{OT}
- Interface Traps ≡ N_{IT}
- Bulk Oxide Traps

Of the three types, N_{OT} is the most noticeable type of damage. The charge that is trapped in the oxide of the device is the mechanism which causes the threshold voltage to shift. Figure 2 - 3 shows the relative placements of the trapped oxide charge and the trapped charge at the interface.

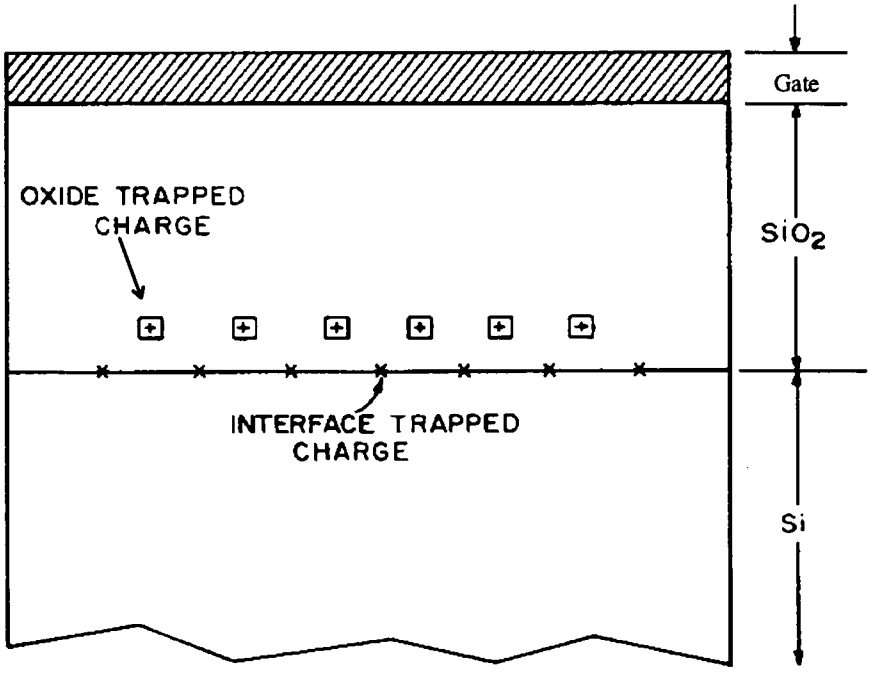


Figure 2 - 3. Location diagram of N_{OT} and N_{IT} in the Si and SiO_2 configuration [Source: SZE1].

At the time of impact with the SiO_2, electron-hole pairs are generated. These electrons rapidly leave the SiO_2 leaving behind the much slower holes (for an extensive treatment of this phenomena see [MA] chapter 3). A result of the inability of the holes to

leave the SiO_2 is that there will be no recombination of the holes with electrons because the electrons have left the SiO_2. Therefore, the holes become trapped with the resulting trapped charge in the SiO_2. In Figure 2 - 4 [BRITTON1], a diagram shows schematically how a photon impinging on SiO_2 might cause an electron-hole pair generation.

The placement of the interface trapped charge in the silicon in Figure 2 - 3 and Figure 2 - 4 is located in the bandgap of the energy diagram for silicon. This allows for recombination of holes at energies between the conduction and valence bands. Most likely, this will affect the transition of a MOS device into and out of the inversion region mode of operation due to electron-hole pairs recombining at an energy level less than that required to traverse the bandgap region in silicon.

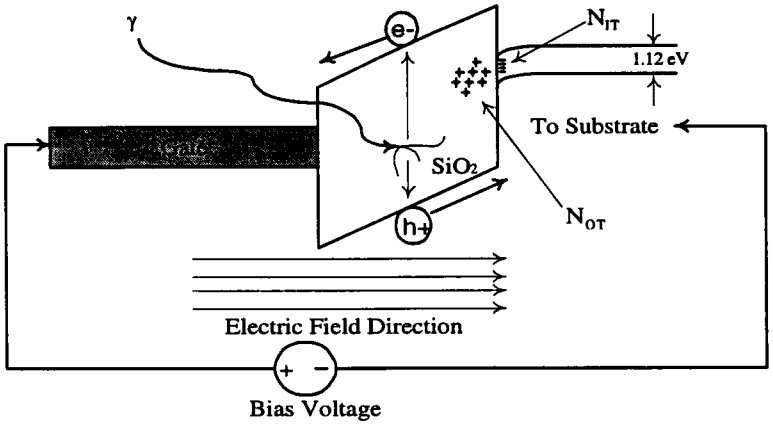


Figure 2 - 4. Schematic representation of electron-hole generation for positive bias due to ionizing radiation.

When the targeted transistor is under a positive bias, the electric field that is present assists in the movement of the particles. The holes are forced toward the Si and SiO_2 interface

and the electrons are quickly attracted toward the positive gate. As the holes become trapped in the oxide, a fixed charge results. This trapped charge causes a shift in the threshold voltage. A graphical representation of how the threshold voltage shifts with radiation dose is shown in Figure 2 - 5 (note, that this is not for the Orbit Semiconductor process used for the ATA chip fabrication). In this figure, the threshold voltage for the NMOS device begins to shift from a negative slope to a positive slope at some relatively high dose level. The phenomena that is occurring here is that the number of negatively charged interface states is beginning to dominate over the oxide trapped positive charge (in Figure 2-5 this occurs around 10^6 rads) [MA2][SROUR6]. If this turnaround did not occur, the enhanced-mode NMOS device would become a depletion mode device. The PMOS device, however, monotonically continues to shift negative, since interface traps in a PMOS device are acceptors (which are negative when occupied by an electron and neutral when unoccupied), which will cause the interface trap to be less effective. The reason for this, Srour says, "...the Fermi level at turn-on for p-channel transistors is below nearly all interface states in the bandgap and thus they would be neutral.".

The total gamma dose used for the ATA chip was about 60Krads at a dose rate of 18 Krads/hr. If the graph of Figure 2-5 were used to predict the Orbit process used for the ATA chip, this should have resulted in an approximate -0.5V shift. For the NMOS devices, this means the transistors will turn on at a lower gate voltage. The PMOS devices will correspondingly require a larger voltage to turn on after irradiation. For the ATA chip process, the designed V_t is ~1V. If a shift of this magnitude were to happen, it would most likely cause the device to become inoperable.

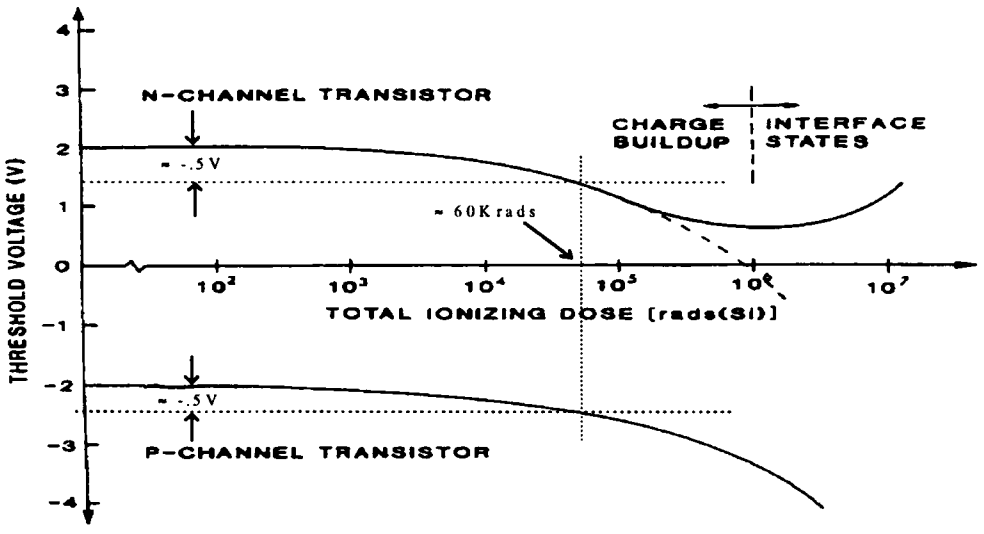


Figure 2 - 5. Vt shifts as a function of radiation dose [Source: SROUR5].

Therefore, dc biasing becomes a problem. For example, if an integrated circuit process provides a NMOS V_t of +2V (as shown in Figure 2 - 5), after 60 Krad irradiation it will be conducting at +2V (at ~1.5V), which could cause adverse effects, if not failure. Conversely for the PMOS devices, one would now need ~-2.5V in order to cause channel inversion. The p-channel transistors may or may not come on or could operate in the linear mode of MOS device operation, that is look like a resistor instead of linear-mode transistor. So, depending on where the devices are, and how even the radiation is spread across the devices, many different effects could result.

Chapter 3 Performance Characterization Description

One of the main reasons for financial support of this thesis was to characterize and document the performance of the ATA integrated circuit. For this circuit, the parameters that are of interest are the following:

- Linearity
- Noise
- Gain

Linearity

Linearity is of importance for many types of mechanical and electrical devices. It describes the desired ability of a device to have an output that is a linear function of the input. Every device will have its own unique characteristic in which linearity is important. The concern of this thesis with respect to linearity pertains to how linearity is important to a Data Acquisition System (DAQ). A DAQ is generally comprised of an Analog to Digital Converter (ADC), some digital process, and a Digital to Analog Converter (DAC). Two different types of linearity which are of interest for DAQ's are:

- Integral Non-Linearity (INL)
- Differential Non-Linearity (DNL).

DAQ's have unique types of problems due to the existence of conversions between the continuous realm of analog electronics to the discrete dimension of digital electronics. The component of a DAQ that is used in the ATA chip is the ADC. For the purposes used here, the data processing section is the IBM computer of the test fixture. A brief discussion of what the errors are and how they are measured for the case of interest will be discussed.

Integral Non-Linearity

The Measurement

INL is the measure of the deviation from a best fit straight line. An example can be seen in Figure 3 - 1.

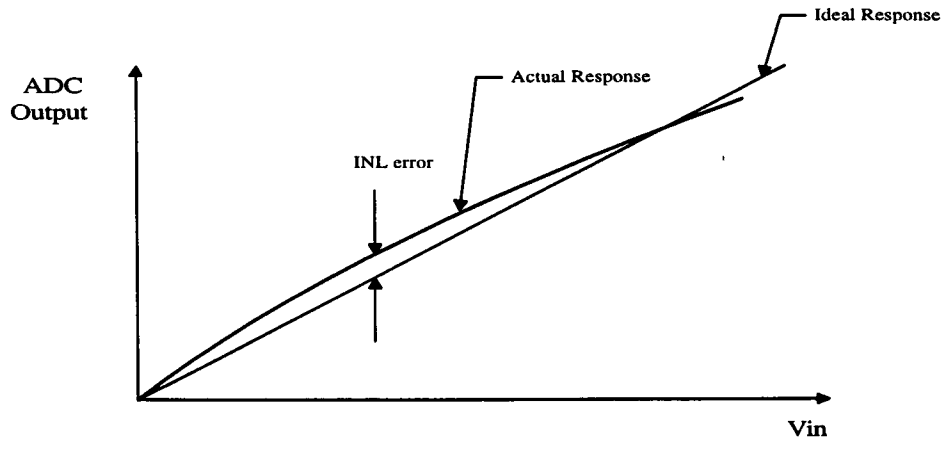


Figure 3 - 1. Illustration of INL error.

The AMU/ADC INL measurement

For the AMU/ADC, INL is measured over a 0V to 5V range in increments of 0.1V. The voltage source is provided by a National Instruments DAQ board (model ATMIO-16X) which has two 16 bit programmable DACs. One of the DACs is connected to the input of the AMU. A computer program named '*Lin3.exe*' (see Appendix C) is used to control the DAC and the conversions of the AMU/ADC sections of the ATA chip. This program selects the first voltage starting at zero and converts it 1000 times. A mean is calculated and stored in memory. The DAC is then incremented by 0.1V and the process is repeated. This continues until the entire voltage range has been covered for each of the 16 AMU pipes. The results are stored in memory and then used to calculate a

Least Squares Fit to the mean data. As a by-product of this program and these calculations (i.e. the following parameters were found in the process of calculating the INL), the slope and y-intercept of the best fit line, the standard deviation (i.e. based on a Gaussian distribution), and variance are stored.

The TAC/ADC INL measurement

The TAC/ADC section's INL is measured by providing the TAC with 10 series of pulse sequences which cover the entire range of the TAC for each of the 8 channels. A pulse sequence consists of a start bit and stop bit. The first set of start and stop pulses are separated by 20 nS. The next set has 40 nS of separation and this continues up to a maximum of 200 nS of separation. This provides 10 points in which to calculate a Least Squares Fit.

A computer program named '*inl.exe*' (see Appendix D) is used for control of the CPLD (Complex Programmable Logic Device) and the conversions of the TAC/ADC sections of the ATA chip. From the use of this program, each pulse sequence is sampled 2000 times. A mean is calculated for use of the Least Squares Fit and the INL calculation. Again, the slope and y-intercept of the best fit line, the standard deviation, and variance are stored to disk.

The INL Calculation

The resulting best fit lines of the AMU/ADC and TAC/ADC data is used as the base line (or ideal line) in which to calculate the INL error. The equation used to calculate the INL can be seen in equation (3 - 1) below:

$$INL = \frac{Measured - Best_Fit}{2^n} \times 100\% \qquad (3 - 1)$$

Where n = 10 for this system. The INL is calculated for each point taken. The results are stored to disk under a unique file name identified by the chip serial number and file extension identifying the INL test. Then the data file can be plotted. The INL is displayed in percent of Full Scale Range (% of FSR) which provides one with the amount of deviation from the best fit line with respect to the overall range of the DAQ.

Differential Non-linearity

The Measurement

DNL is a measure of the discrete sensitivity of the DAQ. This has different meanings for DACs and ADCs. DNL for DACs refers to the continual increasing of the output voltage for every successive increment of the input digital count. This is commonly termed as monotonicity. However, when one is describing DNL for an ADC, it is a depiction of the analog width of each digital channel. An analogy that describes this can be seen in Figure 3 - 2.

The DNL measurement for the ADC is often the most difficult to attain. This is due to the large number of conversions that is required to yield any useful results. For example, in order to achieve < 1% accuracy, the number of samples required is determined by equation (3 - 2) below:

$$\% \text{ accuracy} < \frac{\sqrt{N}}{N} \qquad (3 - 2)$$

where N is the number of samples taken.

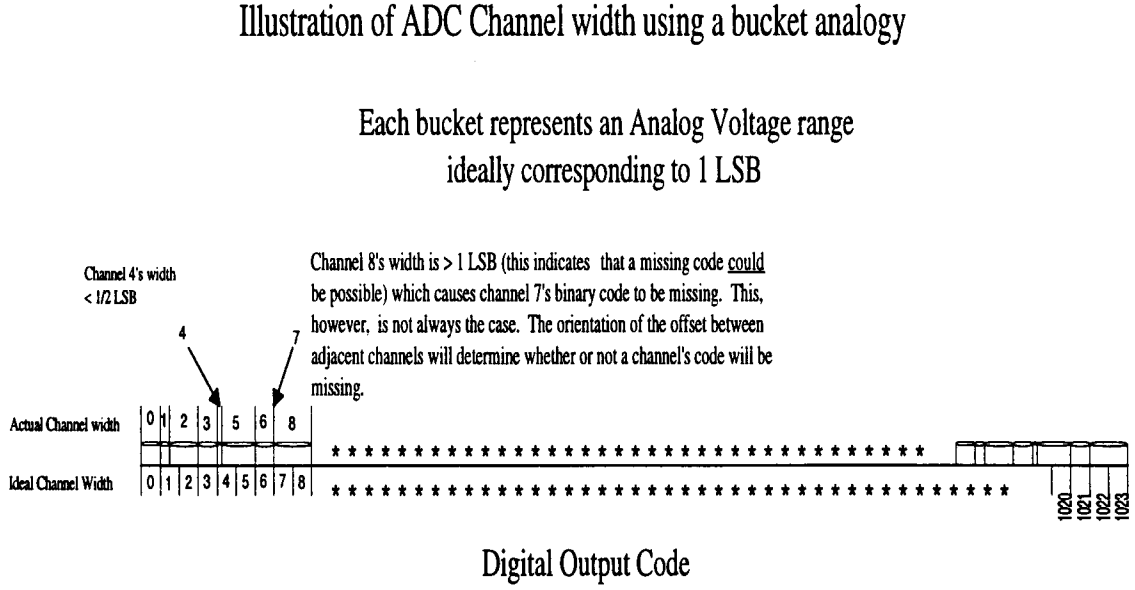


Figure 3 - 2. Channel width analogy.

This equation allows one to ensure that the statistical variance is much smaller than the magnitude of the device variation which in this case is DNL. For 1% accuracy, N must be greater than 10^4 samples in one channel. The number of samples required for the desired accuracy is then multiplied by the total number of channels for the DAQ (i.e. 2^{10} for this case or 1024 channels). Therefore, the total number of samples required is 1024×10^4 per channel or pipe (note: there are 16 AMU pipes and 8 TAC channels) which means there must be in excess of 10^7 samples per TAC channel or AMU pipe taken to achieve a measurement with an accuracy to within 1% of the actual value. Hence, the time requirements to take the data necessary for an accurate measurement could be an important consideration.

The AMU/ADC DNL measurement

The method used to measure the DNL for the ATA chip is accomplished by use

of a signal generator connected to the input of the AMU. The computer program named *'sigdnl.exe'* (see appedix C) has been written to control the ATA test fixture for the acquiring of data for the DNL test. A diagram showing the ramp input signal with relative ADC conversion timing signals embodied in the DNL test can be seen in Figure 3 - 3.

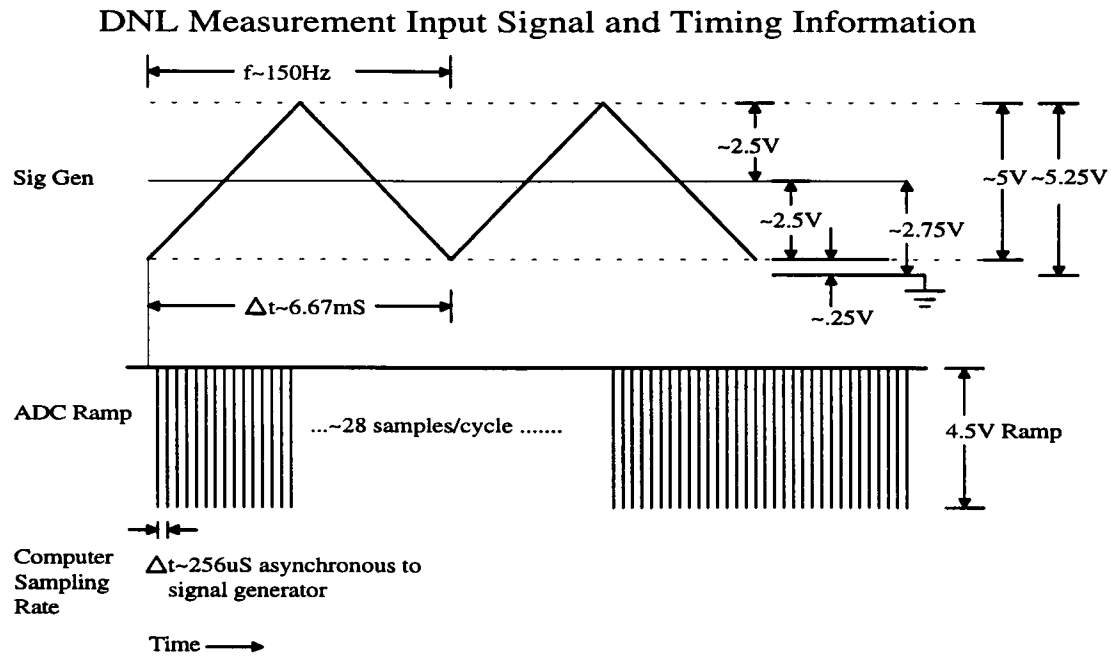


Figure 3 - 3. Timing diagram showing signal relationships of the ATA chip ADC and the ramp generator.

The signal generator and the computer conversions are running asynchronously with the computer sampling at a much higher rate than the frequency of the input signal. When the program begins, the computer does conversions as fast as possible. While the program counts the number of conversions, it monitors the elapsed time. From this information, the program estimates when the desired number of counts per channel have been reached.

The TAC/ADC DNL measurement

The TAC/ADC DNL measurement is achieved in much the same manner as the AMU/ADC measurement. A computer program named '*tdnl.exe*' (see Appendix D) has been written to control this test. The input to the TAC is provided by the CPLD and a 1 MHz asynchronous pulse generator. The computer provides the start pulse and the asynchronous pulse generator provides the stop pulse to channel 3 of the TAC/ADC. The randomness, caused by the computer and the pulse generator running asynchronously, allows for the entire range of the TAC/ADC to be covered. A diagram showing the concept of the signal relationships can be seen in Figure 3 - 4.

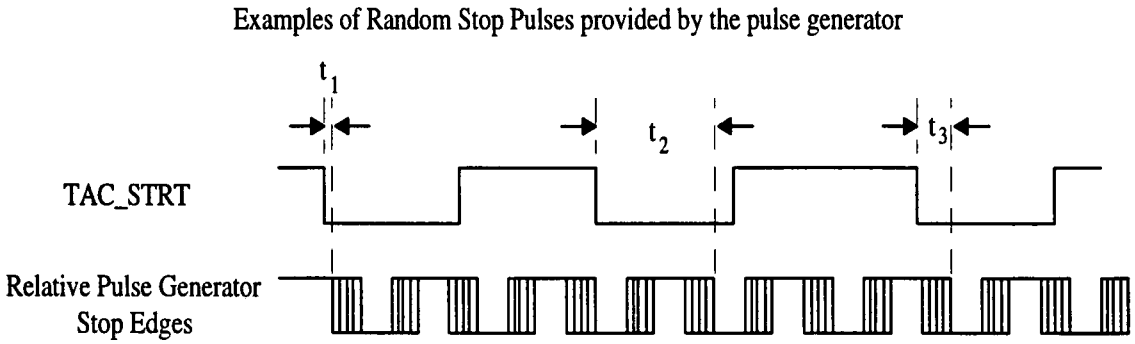


Figure 3 - 4. Signal relationships of the computer and the pulse generator.

The computer performs one conversion each pass. Upon receipt of the resultant data, the corresponding ADC channel is incremented. This is done for an estimated time that allows for enough samples to be taken to attain the desired accuracy. The most accurate measurement attained for the computer conversion time is ~72 μS. This means that for ~30K counts per ADC channel a time of ~36 minutes is required.

The DNL Calculation

For the data taken from the AMU/ADC and TAC/ADC sections, the results are

calculated using equation (3 - 3) below:

$$DNL = \frac{N - \overline{N}}{\overline{N}} \times 100\% \qquad (3 - 3)$$

Where N is the number of counts received in a specific channel and $\overline{N}$ is the mean of the counts taken over the operating range of the ATA chip (i.e. the designed operating range covers from ~1.5V to ~3.5V which is channel 689 through channel 226 respectively). The results are then saved to disk with a unique chip identification number and file extension. The resulting data is then plotted.

Noise

For a DAQ, noise can not be seen by a relatively simple measurement (as is possible with an analog circuit). This must be done by conversion with a mathematical means. A way in which this can be accomplished is to accumulate sufficient data, assume a Gaussian distribution, and then calculate the standard deviation. This will represent the rms noise present in the system.

The standard deviation has been calculated from the 1000 points of data for the AMU/ADC and 2000 points of data for the TAC/ADC. This was done from the data collected in the INL programs. The reason for the different number of points is due to the limitation of the computer programs (/programmer) to handle a larger volume of data. The equation used to calculate the standard deviation can be seen in equation (3 - 4).

$$\sigma^2 = \overline{X^2} - \overline{X}^2 \qquad (3 - 4)$$

where:

$\sigma^2 \equiv$ Standard Deviation

$\overline{X^2} \equiv$ Mean Squared

$\overline{X}^2 \equiv$ Square of the Mean

The routine used to calculate this yields the mean and the variance. These resulting sets of data have been used in other measurements.

Gain

The gain measurement for both the AMU/ADC and TAC/ADC sections of the ATA chip are attained from the slope of the least squares best fit line. Ideally, this should be unity. For the ATA chip with the ramp adjusted at 4.5V, the ideal gain should be ~.9V/V. This is due the actual range of the ADC ramp being 0V to 4.5V. Essentially, the input range is reduced from 0V-5V to 0V-4.5V which yields the .9V/V gain (=4.5V/5V).

Chapter 4 Test Descriptions

This section describes in detail the method used to accomplish the various tests. There are two categories of tests that are performed:

- Functionality Tests
- Performance Tests

First, the functionality of the devices had to be determined. The functionality tests were performed on the chips that are being used in the experiment at CERN. This has been accomplished by making simple inputs to the different subcircuits and checking for the proper output responses.

Second, performance of the devices have been characterized. Ten randomly selected chips were tested. These tests were much more thorough. The purpose of these tests are to enable the designers to understand the effects of combining these mixed mode circuits together on one chip and to enable improvements to be made on future revisions. The following sections will describe the methods used.

Failure Testing Descriptions

Failure analysis testing came about due to the need of functional verification. There were approximately 2600 ATA chips fabricated for the Lead Glass experiment at CERN. These chips needed to be tested in a short amount of time and by different operators, meaning that the interface had to be reliable and user friendly.

The test had to verify that the entire chip was working over the entire designed dynamic range. The AMU/ADC sections had to be useable over a range of 1.5V to 3.5V for each of the 16 pipes. The TAC/ADC had to be able to recognize a pulse separation

range from ~0 to 200nS. If any portion of the test did not yield a result within the desired tolerance, the chip would be discarded as not useable.

The Test

The testing of the ATA chip consists of the test fixture in Figure 4 - 1.

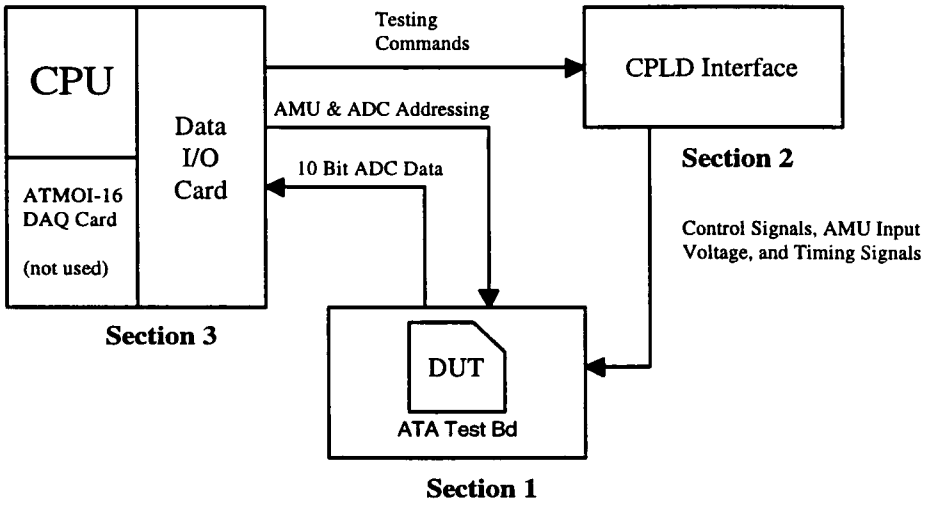


Figure 4 - 1. Test fixture as used in the failure analysis testing.

Section 1, the ATA test board, is a circuit board (created during the development of the ATA chip) that provides access of the analog inputs to the AMU, timing inputs to the TAC, and ADC sections of the ATA chip. Section 2 is the CPLD (Complex Programmable Logic Device) interface of the test fixture which provides control signals to the ATA chip. This is needed due to the inability of the data processing section (AKA Section 3) to provide addressing with speeds on the order of 20 nS (which is required by the ATA chip). For this reason, the CPLD is clocked at 50 MHz. Section 3, the data processing section, is comprised of a IBM 486DX66 computer with a digital I/O card (providing information transfer to and from sections 1 and 2 of the test fixture).

The method in which the tests were completed can be seen in the following discussion.

AMU/ADC Test Description

There are 16 AMU pipes with dedicated ADC's for each pipe. For a better understanding of the AMU/ADC configuration, see Figure 4 - 2:

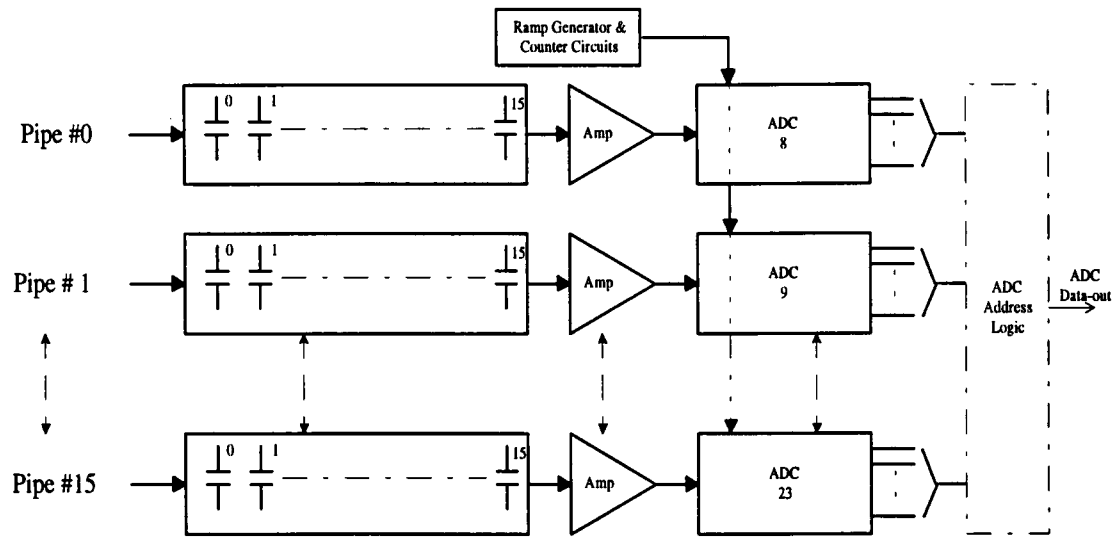


Figure 4 - 2. AMU/ADC configuration block diagram.

The ADC dedicated to the AMU/ADC section of the chip have ADC addresses of 8 to 23. It can be seen from Figure 4 - 2 that there can be many types of failures which cause the chip to be discarded. However, only a fraction of the possibilities can be singled out. This is primarily due to the inaccessibility of the individual circuits. Some of the information attained can be used to determine what section of the ATA chip caused the failure. For instance, if all of the AMU/ADC's and all of the TAC/ADC's fail (i.e. everything fails), the probable cause is a failure in the ramp generator/counter circuits. Similarly, if all of the data is bad for an AMU/ADC pipe, either the read-out amplifier failed or the dedicated ADC channel failed.

The AMU/ADC test consists of providing known voltages and verifying that the correct ADC conversion is received. This is done for every cell of each pipe. The ADC results are then read into memory and compared to the ideal value with a user determined tolerance. If the result is not within the accepted range, an error bit is set in the corresponding element of the AMU section of the error file. Then an error message is sent to the screen notifying the user of a failure.

TAC/ADC Test Description

There are 8 TAC channels with dedicated ADC's for each channel. The TAC/ADC configuration is much the same as the AMU/ADC configuration. There is one TAC channel per ADC channel. The ADCs dedicated to the TAC/ADC section of the chip have addresses 0 -7. The test consists of providing 2 sequences of pulses for each individual channel. An example of what is meant by a pulse sequence can be seen in Figure 4 - 3:

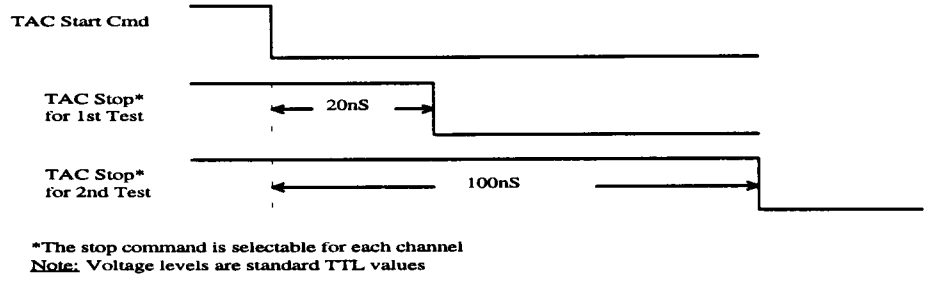


Figure 4 - 3. Example of a pulse sequence sent to the TAC for functionality verification.

The first sequence is a 20 nS pulse sequence. The 20 nS pulse sequence is sent, by the CPLD, first to TAC channel 0 of the DUT. The ADC is then told to convert the resulting voltage from the TAC. The ADC output is then stored and the results are

compared to the ideal value with a user determined tolerance. If the result is not within the accepted range of values for the particular pulse sequence, an error flag is set in the TAC section of the error file and an error message is displayed to screen. This is repeated for the remaining TAC channels.

At the completion of the tests, the raw ADC data and the error data is stored to disk with a filename identified by the chip serial number as discussed earlier. These files were then used for the failure analysis.

Firmware Program Descriptions

In this section, a description of the CPLD firmware programs is given. The AMU and the TAC firmware will be discussed. The firmware program written for this set of tests is called 'DAQ.JED' (see Appendix F).

In the following description, many signals are given. In order to aid in the understanding of the signal flow, it is necessary to provide a description of text coding. This allows one to understand the origin and destination of the various signals.

Legend for Signal Flow:

Underline - Bold - Uppercase	→	Computer ⇒ CPLD Interface
Bold - Lowercase	→	Internal CPLD signals
Bold - Uppercase	→	Outputs of the CPLD ⇒ ATA Test Bd.
Underlined only	→	ATA Test BD ⇒ Computer
Italics	→	Computer ⇒ ATA Test Bd.

AMU

The AMU section of the CPLD firmware program has two major sections. The first section is the write section, which controls the address lines to the AMU. In Figure 4

- 4, is a timing diagram for the write section of the firmware. This figure shows two complete writing cycles for the analog memory section.

The CPLD starts the addressing upon receipt of a pulse named '**WRITE**'. This pulse is received from the computer. This sets off a chain of events within the CPLD resulting in the addressing of the AMU capacitor array. Below is a description of the

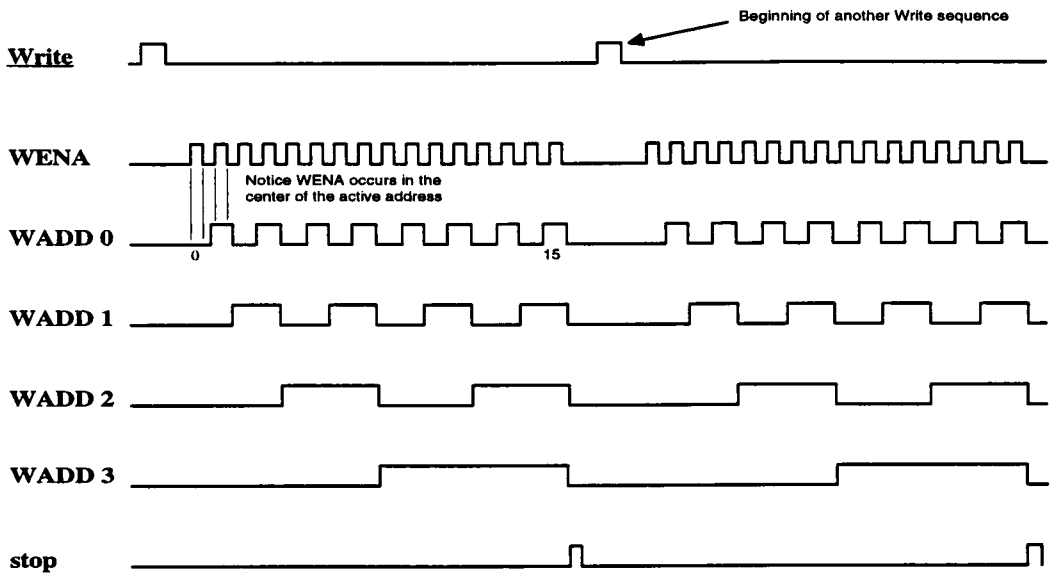


Figure 4 - 4. Timing diagram for the AMU Write Section of the firmware.

signals and events within the CPLD that allow the desired chain of events to occur:

- ***'WRITE'*** is received and latched in by 3 F/F's to a signal called **'go'**. This signal is basically an enable used to control the write address counter - **'WADD[0..3]'** and the write enable - **'WENA'**.
- Once the **'go'** signal is active, the **'WADD [0..3]'** counter starts counting and **'WENA'** starts becoming active during the middle of each write address to assure the address lines are settled. This is done by **'CLK2'** that is 180° out of phase with **'CLK1'** enabling **'WENA'** to become active at the desired time.

- The **'stop'** signal is used to stop the counter (also known as write address lines) at a predefined count. This can be changed to any count. The clock period times the count set (i.e. **'WADD[0..3]'**) determines the write period for the AMU. The current clock frequency is 50 MHz (⇒ clock period = 20nS and the write period is 320 nS).
- After the counter has stopped, the counter is reset and waits for the next **'WRITE'** signal to start another chain of events.

The second section of the AMU portion of the CPLD firmware program is the Read section. This section is responsible for providing the control signals necessary to read the AMU voltages to the AMU ADC's. The timing for this section is best illustrated with the computer and ATA signals included. These timing signals for the AMU read cycle can be seen in Figure 4 - 5.

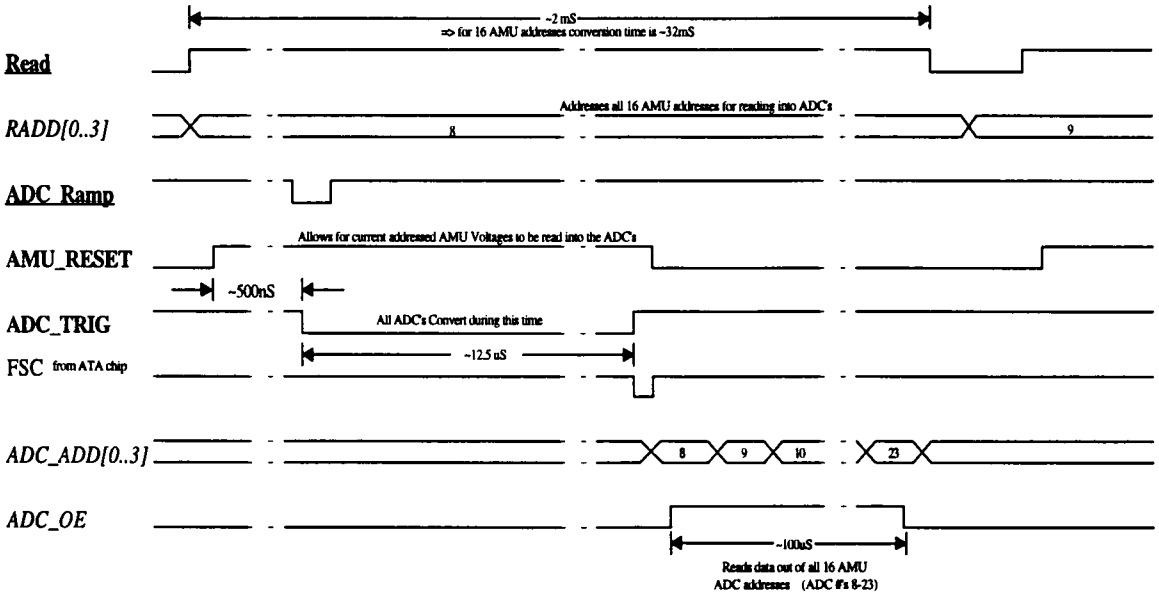


Figure 4 - 5. Timing for Read portion of AMU test. This shows handshaking for the computer, firmware and ATA under test.

Following is a description of the signals and events within the CPLD that allow the read cycle chain of events to occur:

- ***'READ'*** signal is received and latched in by 3 flip flops to a signal called **'rgo'**. This signal is basically an enable used to control the output signals sent to the test board. The **'READ'** signal starts the chain of events.
- **'AMU_RESET'** is used to start the selected ADC in the ATA chip to convert the analog voltage presently addressed by the computer on the **'RADD[0..3]'** lines. The RENA signal is a result of the **'AMU_RESET'** signal. **'AMU_RESET'** is converted to RENA by delaying it through one inverted gate on ATA TEST board.
- **'delay'** is used to turn off the **'rcntr[0..4]'** counter. This counter provides the ability to delay the **'ADC_TRIG'** for up to 640 nS in 20 nS increments (based upon a 50 MHz clock).
- The **'ADC_TRIG'** signal is used to start the ADC conversion process on the ATA Test Board. This signal occurs after 500 nS which is determined by the number of counts that is selected in the Boolean equation of the **'ADC_TRIG'** definition. The selection of the delay is done by the **'rcntr[0..4]'** bits. Once this signal is sent, the responsibility of the read section of the firmware code is complete until another **'READ'** signal is received. After the **'ADC_TRIG'** signal is sent, the computer controls the interfacing required in order to read data out of the ADC's.

TAC

The basic function for the CPLD in the TAC portion of the firmware program is to provide two pulse sequences starting with 20 nS separation and ending with 100 nS of pulse separation. The CPLD is controlled by a computer program named *'chip2.exe'*. Listed below are the sequence of events which provide the necessary information to understand the interfacing between the computer, CPLD Interface Board, and the ATA Test Board.

Sequence of Events:

1. **Computer sends CHAN_SEL[0..7], TST[0..2], and PULSE[0..3] to the Mach 230 CPLD**. Below is a description of the control bits:

CHAN_SEL[0..7] BITS ARE INDIVIDUALLY SELECTABLE

Each bit represents a channel that is to receive the testing pulse set. Otherwise, the unselected channel/s times out and provides an amplitude that the ADC converts to full scale count. TAC Channel 0 is the LSB and TAC Channel 7 is the MSB in the CHAN_SEL[0..7] bits.

TST[0..2] BITS ARE BINARY CODE SELECTABLE

These bits select the set/s of pulse sequences (in conjunction with the PULSE[0..3] bits) to send to the ATA chip. The pulse sequences are dedicated to the particular test (different tests can be added or redefined in the Palasm code for the Mach 230 CPLD). Table 4 - 1 gives the test code definitions.

Table 4 - 1. Test code definitions.

Test Code	Resulting pulse enables	Corresponding Pulse Code/s Required
TEST = 0	200 nS *	Pulse 9
TEST = 1	20 nS & 100 nS	Pulse 0 & 9
TEST = 2	40,80,120,160, & 200 nS	Pulse 1,3,5,7,9
TEST = 3 - 7	NOT DEFINED	

PULSE[0..3] BITS ARE BINARY CODE SELECTABLE

The control bits labeled *PULSE[0..3]* are used to provide the Mach 230 CPLD with the ability to select the appropriate sequence of pulses. The pulse sequences are dedicated to a particular test in the CPLD as indicated in the TST[0..2] bits. These can be changed in the Palasm code (i.e. the language used to program the CPLD) to any desired combination. Table 4 - 2 gives the pulse code definitions.

Table 4 - 2. Pulse code definitions used in failure analysis verification.

Pulse Code	Delay
PULSE = 0	20 nS
PULSE = 1	40 nS
PULSE = 2	60 nS
PULSE = 3	80 nS
PULSE = 4	100 nS
PULSE = 5	120 nS
PULSE = 6	140 nS
PULSE = 7	160 nS
PULSE = 8	180 nS
PULSE = 9	200 nS
PULSE = 10 - 15	NOT DEFINED

2. The computer sends the *TAC_STRT* command to the Mach 230 CPLD. This signal starts a chain of events within the CPLD that will provide the appropriate signals to test the desired feature. Within the CPLD there are a series of F/F's that provide consecutive 20 nS delays (with a 50 MHz clock). Each channel has access to each delay when selected by the PULSE[0..7]. With the appropriate test and channel/s selected, the TAC_STRT command is sent from the computer, the Mach 230 CPLD issues a 'TAC_START' pulse to the DAQ test board, and then the selected amount of time later a stop pulse is sent to the selected channel/s.

3. The *FSC* bit (Full Scale Conversion; active LOW) is monitored from the DAQ chip. Once received, the computer will provide the first ADC address and OE to the DAQ Test Bd. to read out the data. The address count will be incremented and the process will continue until all of the addresses have been read from the 8 different TACs.

4. **The computer will check to see if there are any more pulse sequences for the selected test.** If so, the computer will go to step two and continue until the test set is complete. If not, the computer will exit and go back to the main menu.

Note: The Mach 230 CPLD is provided with the control bits and a TAC_STRT command (negative edge triggered) which starts a 'chain reaction' to furnish the DAQ chip with the pertinent signals for testing. The computer handles the control for multiple pulse sequences, addressing needed for the ADC conversion, and obtains the resulting data.

Computer Program

A program named 'chip2.exe' (see Appendix B) has been written to control and perform the tests. In addition to testing all sections of the ATA chip, the program controls the firmware programs, calibrates the input voltage source, provides several ramp adjustment routines, tests the data for failures, and records the results. The program provides the user with a user friendly interface. The program prompts the user for which tests to run and a desired tolerance to test the data against. After the final parameter has been defined, the program executes giving the user a pass/fail indicator while storing the data to disk.

Upon completion of the tests for both sections, a data file and an error file are generated and stored to disk. Both the AMU/ADC and the TAC/ADC section results are stored in the same file. This reduces the number of files that are needed to be manipulated for analysis. The files are stored with a distinct filename for each chip. This is done in the following format:

- '**c***chipid*#**.dat**' for raw data

- '**c***chipid#***.err**' for location of failure

The raw data file are created so that one can see the severity of the failure. The error file allows one to locate the failure quickly by eye and also easily by computer check. This file is very valuable when writing programs to determine statistics for the number and types of failures (see Appendix E for data collection programs). The data file and the error file are mirror images of each other. By virtue of the location, one is able to identify the test that failed and the specific function of the ATA chip that caused the test failure/s. The data and error file are divided up into three categories:

- TAC/ADC section
- 1.5V AMU/ADC section test
- 3.5V AMU/ADC section test

In Figure A-1 (in Appendix A), the structure of the files can be seen.

Performance Testing Descriptions

Device Logistics

The performance tests are performed on 10 randomly selected chips five of which are in plastic packages and five in ceramic packages. The chips are numbered in the following fashion:

- Plastic packages - numbered 0-4
- Ceramic packages - numbered 5-9

These five chips were irradiated simultaneously under normal operating conditions, using a Co^{60} radiation source at Argonne National Laboratory. The inputs to the AMU are connected to 2.5V (i.e. chip ground). The TAC has its channel inputs grounded. The ADC outputs were not connected.

Types of Tests

There are numerous tests that could be performed in the characterization of these devices. However, there are limiting constraints which dictate a relatively narrow scope of testing mainly time and funding. The most important measurements that are of concern to the designers are the following:

1. Linearity
    - INL
    - DNL
2. Noise
3. Matching
    - Between Package types
    - Between Devices
    - Within Devices (i.e. channel comparisons)

All comparisons are made both pre- and post-irradiation.

How are the Tests Performed?

The tests are performed using a test fixture, firmware, and 'C/C++' programs which control the testing. The test fixture and firmware will be discussed in the following section.

Test Fixture

The test fixture that has been developed for the testing of the ATA chip was originally designed for functionality verification. By performing minor modifications it was possible to use the fixture for the performance analysis. The test fixture is comprised of 3 sections as shown in Figure 4 - 6.

The only difference in the test fixture from the failure analysis section is in section 3. An ATMIO-16X DAQ card has been added to provide computer controlled input voltages to the AMU and a pulse generator was used for TAC testing.

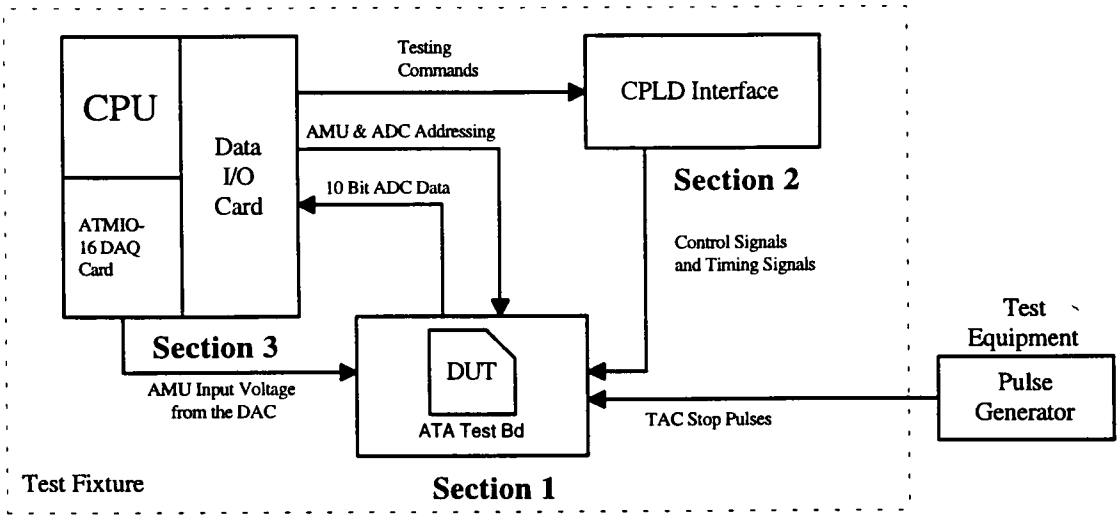


Figure 4 - 6. Test Fixture as used for Characterization Testing

Firmware Program Modification Descriptions

In this section, a description of the CPLD firmware programs for both the AMU and the TAC, as used for analyzing the performance of the ATA chip, will be discussed. For the performance testing, the AMU/ADC tests are performed by the original firmware program named 'DAQ.JED' (see Appendix F). For a description of the original firmware programs, see the functionality firmware discussion. In order to accommodate the needed control for the evaluation of the TAC, a new program has been written (TAC.JED; see Appendix F). This has been done to eliminate the need for the test codes.

Again, it is necessary to provide a description of text coding used below. This allows one to understand the origin and destination of the various signals.

Legend for Signal Flow:

Underline - Bold - Uppercase	→	Computer ⇒ CPLD Interface
Bold - Lowercase	→	Internal CPLD signals
Bold - Uppercase	→	Outputs of the CPLD ⇒ ATA Test Bd.
Underlined only	→	ATA Test BD ⇒ Computer
Italics	→	Computer ⇒ ATA Test Bd.

TAC

The performance and functionality testing of the TAC have different firmware programs. Functionality testing firmware is described in the failure analysis section. The firmware program 'TAC.JED' used for the performance characterization testing is described in this section.

The basic function for the CPLD in the TAC portion of this firmware program is to provide a pulse sequences starting with 20 nS separation and ending with 200 nS of pulse separation in intervals of 20 nS. This is the limiting factor when measuring INL resolution.

The CPLD is controlled by a computer program. A basic timing diagram for the TAC portion of the firmware can be seen in Figure 4 - 7.

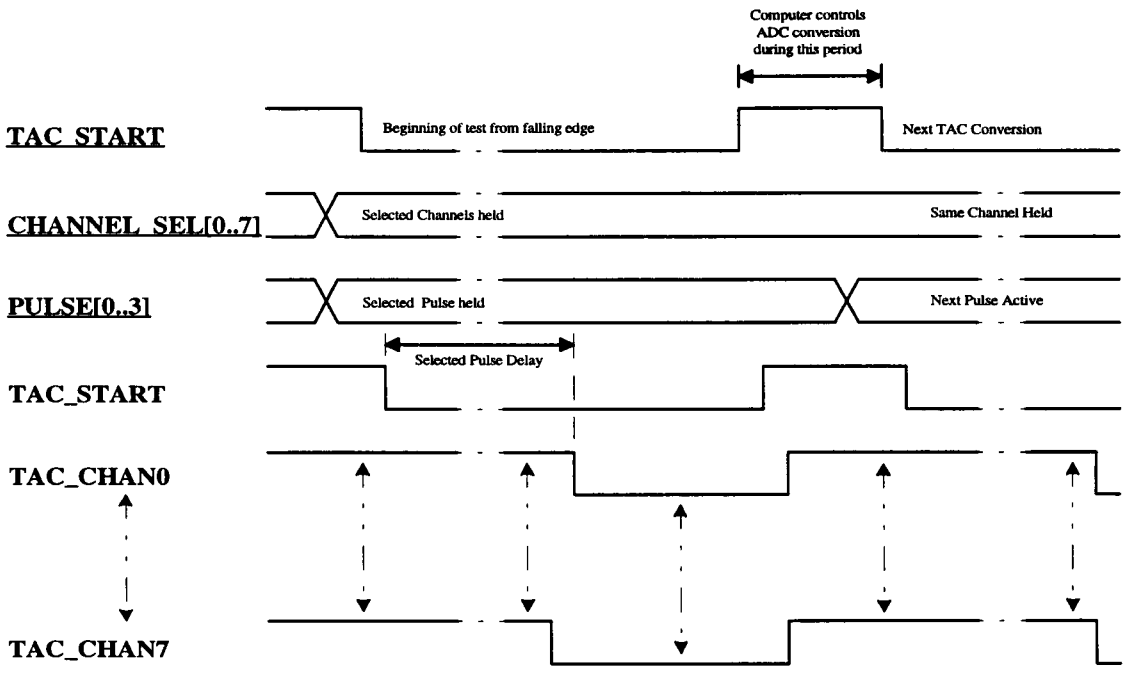


Figure 4 - 7. Timing diagram for TAC firmware.

Listed below are the sequence of events which provide the necessary information to understand the interfacing between the computer, CPLD Interface Board, and the ATA Test Board.

- **Sequence of Events:**

1. Computer sends 'CHAN_SEL[0..7]' and 'PULSE[0..3]' to the Mach 230 CPLD. Below is a description of the control bits:

CHAN_SEL[0..7] BITS ARE INDIVIDUALLY SELECTABLE

Each bit represents a channel that is to receive the testing pulse set. Otherwise, the unselected channel/s times out and provides an amplitude that the ADC converts to full scale count. Channel 0 is the LSB and Channel 7 is the MSB in the '**CHAN_SEL[0..7]**' bits.

PULSE[0..3] BITS ARE BINARY CODE SELECTABLE

The control bits labeled '**PULSE[0..3]**' are used to provide the Mach 230 CPLD with the ability to select the appropriate sequence of pulses. This can be seen in Table 4 - 3.

Table 4 - 3. TAC firmware program pulse code definitions.

Pulse Code	Delay
PULSE = 0	20 nS
PULSE = 1	40 nS
PULSE = 2	60 nS
PULSE = 3	80 nS
PULSE = 4	100 nS
PULSE = 5	120 nS
PULSE = 6	140 nS
PULSE = 7	160 nS
PULSE = 8	180 nS
PULSE = 9	200 nS
PULSE = 10 - 15	NOT DEFINED

2. **The computer sends the 'TAC_STRT' command to the Mach 230 CPLD.** This signal starts a chain of events within the CPLD that provide the appropriate signals to test the desired feature. The CPLD is configured to provide 10 different pulse delays. Each channel has access each delay which is selectable by the **'PULSE[0..7]'** control bits. With the channel/s selected, the **'TAC_STRT'** command is received from the computer. The Mach 230 CPLD issues a **'TAC_START'** pulse to the ATA Test BD and then a selected amount of time later a stop pulse is sent to the selected channel/s.

3. **The *FSC* bit (Full Scale Conversion; active LOW) is monitored from the ATA chip.** Once received, the computer will provide the first ADC address and OE to the ATA Test BD to read out the data. The address count will be incremented and this will continue until all of the addresses have been read in from the 8 different TACs.

4. **The computer is now ready to do another operation or repeat the process.**

Computer Program Descriptions

INL Program Descriptions

The computer is controlled by programs written specifically for the tasks needed. The tests were performed with the following executable programs:

- LIN3.EXE (AMU INL program; see Appendix C)
- INL.EXE (TAC INL program; see Appendix D)

These programs were written in 'C' and 'C++'. The reason for the two different languages are due to availability and convenience. The test fixture was initially controlled using programs complied using a 'C' compiler. At a later time, a 'C++' compiler was available. The 'C++' compiler provided some luxuries not available to a 'C' compiler.

LIN3.EXE (AMU INL program)

This program provides the proper control signals and routines to characterize the INL for the AMU of the ATA chip. The program uses command line arguments that determine the following parameters:

- Chip Identification #
- The number of samples to be taken at each voltage point
- The increment in which to step the DAC (minimum increment of 1/80V)
- The upper limit in which to use data for the Least Squares Fit calculations
- The lower limit in which to use data for the Least Squares Fit calculations
- The lowest voltage is the voltage at which to calculate the Least Squares Fit data

These parameters allow flexibility in the use of the program. In the program input parameters above, notice that the voltage step has a minimum increment value. This is due to the difficulty in accessing memory above 640 KB in DOS applications.

The program begins by initializing many parameters. The AMU is supplied with the first voltage (which is allowed to settle) and is told to convert. The selected input voltage is converted the number of times determined by the command line parameter discussed above. Each conversion is done to the center cell of each AMU pipe. In Figure 4 - 8, a diagram of the AMU shows the cell configuration.

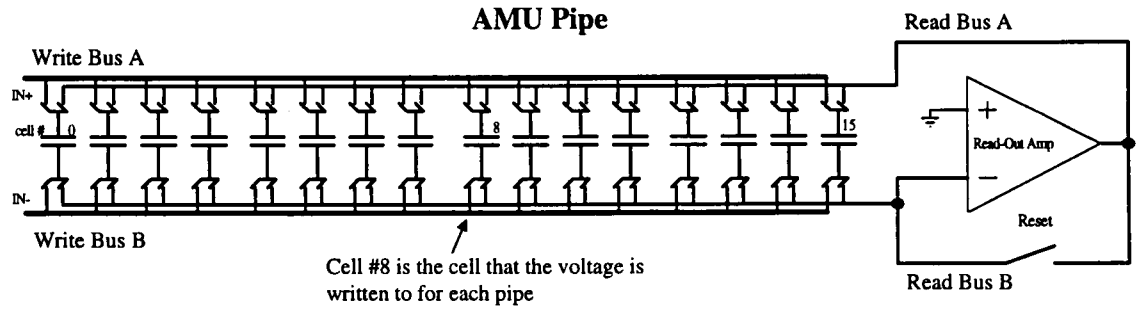


Figure 4 - 8. The AMU testing configuration for LIN3.EXE program.

The data for each conversion is stored in an array in memory. After all of the samples are taken at the current voltage point, the program computes the standard deviation of this voltage level and then stores this value in memory. The DAC is incremented and the process is repeated until the voltage range from 0V to 5V has been covered. The program then exits the loop and computes a least squares fit to the resultant mean data. All of the above calculations are stored to disk under the chip identification number. A flow diagram can be seen in Figure A-2 (in Appendix A).

INL.EXE (TAC INL program)

This program provides the appropriate control signals and routines to characterize the INL for the TAC section of the ATA chip. The program has command line arguments that allow flexibility. The following is a list of the parameters that are changeable at the command line:

- Chip identification number
- Number of times to sample the given pulse sequence
- Upper limit of the data to be used for the Least Squares Fit calculations.
- Lower limit of the data to be used for the Least Squares Fit calculations.

A program flow diagram can be seen in Figure A-3 (Appendix A). The program basically functions the same as the 'Lin3.exe' except for the mode of providing the input signal. This is done by use of the firmware with the TAC.JED (see Appendix G) program loaded. This program allows the TAC to send the pulses in increments of 20 nS by simply sending the pulse code of the desired pulse sequence and the channels in which the stop bits are to be sent. The incrementing of the pulse selection code is put in a loop much the same way

as the voltage level in 'Lin3.exe'. The same type of data is again calculated and stored to disk.

DNL Program Descriptions

The DNL tests for both the AMU and the TAC are discussed in the following section. The tests were performed with the following executable programs:

- SIGDNL.EXE (AMU DNL program; see Appendix C)
- TDNL.EXE (TAC DNL program; see Appendix D)

SIGDNL.EXE (AMU DNL program)

This program provides necessary control signals and routines in which to characterize the DNL for the AMU. The command line arguments allow the following program parameters to be supplied:

- Chip identification number
- The number of samples to be taken for each channel
- Frequency of the input ramp generator

This program does not have to provide any voltages to accomplish the measurement due to a separate ramp generator that is running asynchronously to the computer program. The program runs in a simple loop that minimizes loop time to allow for quick acquisition time. It then allocates a matrix that is 16x1024 elements in size. The 16 rows are for each pipe of the AMU and the 1024 columns are for each ADC channel.

For each conversion, the resulting ADC data is used as the index for the ADC channel and that channel is incremented. This means that the channel has been used. Ideally, each channel should be used exactly the same number of times for a given integral number of passes. However, due to nonlinearities in the ATA chip, this is not possible.

The result is that the end channels accumulate a much greater number of counts than the middle (useable) channels. Therefore, only the channels within the designed operating range are used for the DNL measurement.

The program runs a number of passes that will give an estimated number of counts in each channel. This is calculated by multiplying the number of passes times the amount of time for each pass. At the end of the estimated time, the resulting 16x1024 matrix is stored to disk for use in the plotting software. A flow diagram can be seen in Figure A-4 (in Appendix A).

TDNL.EXE (TAC DNL program)

This program is almost exactly the same as the DNL program for the AMU except for the manner in which the input is provided. In this case, TAC channel #3 is supplied a start bit from the computer program and a stop bit from an asynchronous pulse generator running at ~1 MHz (see Figure 4 - 3). This has been experimentally determined to provide the best DNL results. See Figure A-5 (in Appendix A) for a program flow diagram.

Chapter 5 Results

Failure Analysis Results

There are two sets of results from the failure analysis. The first set of results are the results from the main production run (total number = 2565 ATA chips). The second set of results come from two wafers of chips that were kept for emergency use and/or testing (total number = 149). The overall results of the failure analysis testing for the production run (i.e. chip #'s 5000-7564) can be seen in Table 5 - 1.

Table 5 - 1. Chip #'s 5000-7564 failure analysis results.

ATA CHIP SUBCIRCUIT STATISTICS Chip Area	Fail	Out of 938 Bad Chips	Out of 2565 Chips
AMU only caused chip failure: (32.2%)	260	27.7%	10.1%
TAC/ADC only caused chip failure (22.9%)	193	20.6%	7.52%
ADC RAMP or COUNTER CKTS/P.S. Failures: (23.2%)	214	22.8%	8.34%
Both sections (AMU & TAC) have at least one failure:	271	28.9%	10.6%
Total of Bad Chips with Data files:	938	100%	36.6%
Good chips that tested bad(*due to prg errors* in chip.exe):	57	6.08%	
Total number of good chips that tested bad:	995		
Total # of chips tested bad (some chips drew too much current) :	938		36.6%
Total # of chips that failed every test or had short circuits:	232		9.04%
	1170		45.6%
OVERALL STATISTICS			
Total # of chips found good:	1395	***Yield***	**54.4%**
Total Number of Bad chips	1170		45.6%
Total:	2565		100%

The results for chip #'s 7565-7713 can be seen in Table 5 - 2. It can be seen from the two different sets of results that there is some correlation between the percentage of circuit section failures and relative chip areas. In Table 5 - 1, the relative chip areas are

proportional to the percentage of failures. This is most apparent for the production run due to the number of devices tested (i.e. there is not as much correlation in Table 5 - 1.

Table 5 - 2. Chip #'s 7565-7713 failure analysis results.

ATA CHIP SUBCIRCUIT STATISTICS Chip Area		Fail	Out of 73 Bad Chips	Out of 149 Total # Chips
AMU only caused chip failure:	(32.2%)	18	24.7%	12.08%
TAC/ADC only caused chip failure	(22.9%)	11	15.1%	7.38%
ADC RAMP or COUNTER CKTS/P.S. Failures:	(23.2%)	24	32.9%	16.11%
Both sections (AMU & TAC) have at least one failure:		20	27.4%	13.42%
Total of Bad Chips with Data files:		73	100%	48.99%
Total # of chips tested bad:		60		40.27%
Total # of chips that failed every test or had short circuits:		13		8.72%
		73		48.99%
OVERALL STATISTICS				
Total # of chips found good:		76	***Yield***	**51.01%**
Total Number of Bad chips		73		48.99%
Total:		149		100%

Since the ADC could not be separated from the AMU or the TAC, there are only a few failures that can be positively identified from analyzing the resultant data. An AMU pipe and the corresponding ADC channel can be determined as the source of a failure if all of the data within the column in the data file is out of tolerance (see Figure A-1 in Appendix A). Likewise for the TAC channel and corresponding ADC channel, it can be found to be bad if it is out of tolerance for both the 20 nS and 100 nS pulse sequences. In a similar manner, an AMU address/es can be determined to be bad if a row/s are out of tolerance in the AMU/ADC section of the data. The ADC counter

circuitry, ADC ramp circuitry, and/or a biasing problem is most likely indicated by bad data in all section of the test. Other than these few indications, there is no way to identify a failures from the available data.

It should be stated that there did not seem to be a large number of failures in a particular section of the ATA chip. This implies that there were no major design problems such as large offsets in operational amplifiers, ect.. The yield predicted by Orbit Semiconductor along with probable defects caused by packaging will produce a result within a reasonable range of the yield experienced. Therefore, the most likely reason/s for the failures is fabrication processing defects and/or defects caused by the packaging process.

Defect Density

The defect density allows one to calculate the probability that a die will be good. This is limited by the state of technology of integrated circuit processing. For the Orbit Semiconductor 1.2 micron process, the claimed defect density is 15 defects/in^2. There are several models of estimating yield. Listed below are three models that have been developed to predict yield [GEIGER1].

$$P = \exp(-A*D) \qquad (5 - 1)$$

$$P = \exp(\sqrt{(A*D)}) \qquad (5 - 2)$$

$$P = ((1 - \exp(-A*D))/A*D)^2 \qquad (5 - 3)$$

where:

P ≡ percentage of good devices

A ≡ Area per device (in square inches)

D ≡ Defect Density

For the case for the ATA chip, the area for the ATA chip is .0288 in^2 and the predicted defect density by Orbit Semiconductor is 15 defects/in^2.

In Table 5 - 3, a compilation of the three models and their results are presented. The actual yield for the ATA chip was 54.2%.

Table 5 - 3. Defect Density estimations using different models

Equation	Yield	% Difference from Experimental Results
Poisson Model (5 - 1)	64.96%	19.85%
Seed Model (5 - 2)	51.85%	4.34%
Murphy Model (5 - 3)	65.93%	21.7%

Using a Poisson distribution, which is used for small die, the percentage of devices that is predicted good is off by 19.85%. This would be expected since the ATA chip is a large die. The Seed model and the Murphy model are used for predicting large die yield. As can be seen in Table 5 - 1, the Seed model is the closest to the actual yield. The Murphy model results are actually worse than the Poisson model. Other factors that contribute to the device failures that are not predicted by the above models are:

- cutting of the die
- packaging

None of these factors however, should cause the results of the models to be grossly off as in the case of the Murphy model. Therefore, the Murphy model seems not be a good model for the die the size of the ATA chip which has dimensions of 4.9mm by 3.73mm.

The 'brother' chip that has been fabricated on the same wafer with the ATA chip is the Preamp chip. This device experienced 58.7% yield. So, the estimation of the defect density predicted by Orbit is best described by the Seed model.

INL AMU/ADC Results

Pre-Radiation

In Figure 5 - 1, a typical pre-radiation result can be seen. The deviation from the best fit line, in percent of the full scale range (% FSR), as well as the dynamic range of the device can be seen in the INL plot. The design range for the ATA chip is 1.5V to 3.5V. As can be seen in Figure 5 - 1, the INL ranges from about +.15 % FSR to less than -.15% FSR over the entire designed dynamic range. At ~4.5V, the device can be seen to become erratic. This is most likely due to the onset of biasing problems due to the PMOS devices beginning to turn off.

In Figure 5 - 2, a plot of the INL at the mid-scale for all of the chips (pre-rad) is shown. In this plot, the mid-scale INL for all of the chips ranges from -.02 % FSR to -12% FSR. A closer look tends to show that the plastic packages (#'s 0-4) exhibit a lower INL than do the ceramic packages (#'s 5-9). Also, in most cases, there is a rise in the INL for each chip from pipe 7 to pipe 8 due to a test fixture induced problem that will be discussed in the AMU/ADC noise results section. This will be much more apparent in the noise measurement results.

Post-Radiation

The effects of radiation on the AMU/ADC are evident in both types of packaging. The ceramic packages, however, seem to exhibit a more pronounced effect. For example,

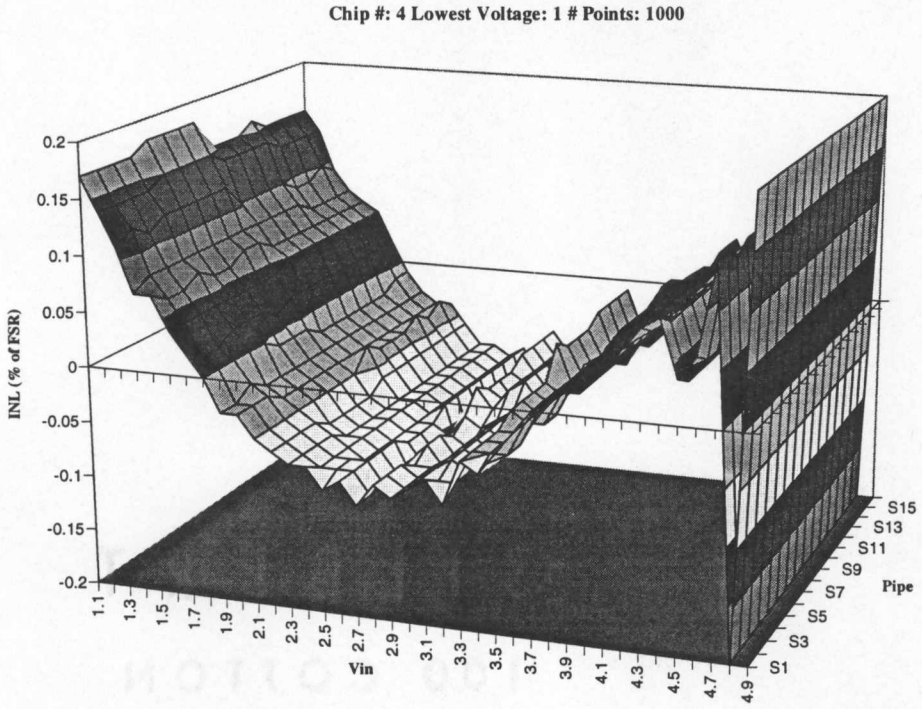


Figure 5 - 1. Chip #4 Pre-Radiation INL Results.

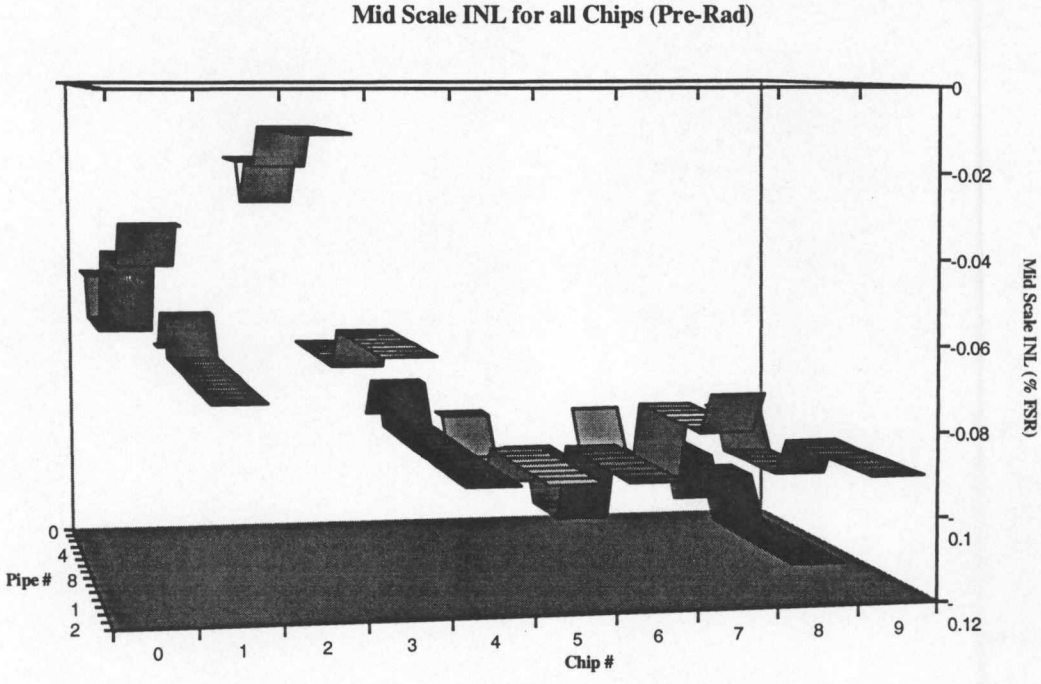


Figure 5 - 2. Mid-scale INL taken at 2.5V for all chips prior to irradiation.

in Figure 5 - 3, a comparison of the pre and post-radiation effects for a plastic packaged device can be seen. Notice the mid-scale increase from approximately -.15 % FSR to about -.35 % FSR. This is a typical result for the plastic packages. The ceramic packages exhibit an increase from approximately the same pre-radiation value to about -.6 % FSR. This can be seen in Figure 5 - 4. The drastic increase in the INL for the ceramic packaged devices seems to indicate that the ceramic packages (i.e. the gold lids) enhance the effects of the radiation damage [SROUR1]. The gold lid enhances the damaging radiation effects over the entire area of the chip which is apparent due to the same effect over all 16 pipes of the AMU/ADC. This circuitry takes up ~53.9 % of the chip area which indicates an even distribution of damage.

INL TAC/ADC Results

Pre-Radiation

A typical INL measurement for the TAC/ADC can be seen in Figure 5 - 5. For the basic INL measurement, there were no distinguishing features between the different packages. The INL for the TAC/ADC section have approximately -.5 % to -1 % deviation from the best fit line at mid-scale, which is about a factor of ~6 greater than that of the INL for the AMU/ADC section. This is mostly due to the number of samples available for the calculation of the best fit line. The AMU/ADC had 50 points available for the least squares fit approximation whereas the TAC/ADC had only ten points. Thus, accuracy is degraded. A plot of the INL for the mid-scale values for all the chips can be seen in Figure 5 - 6.

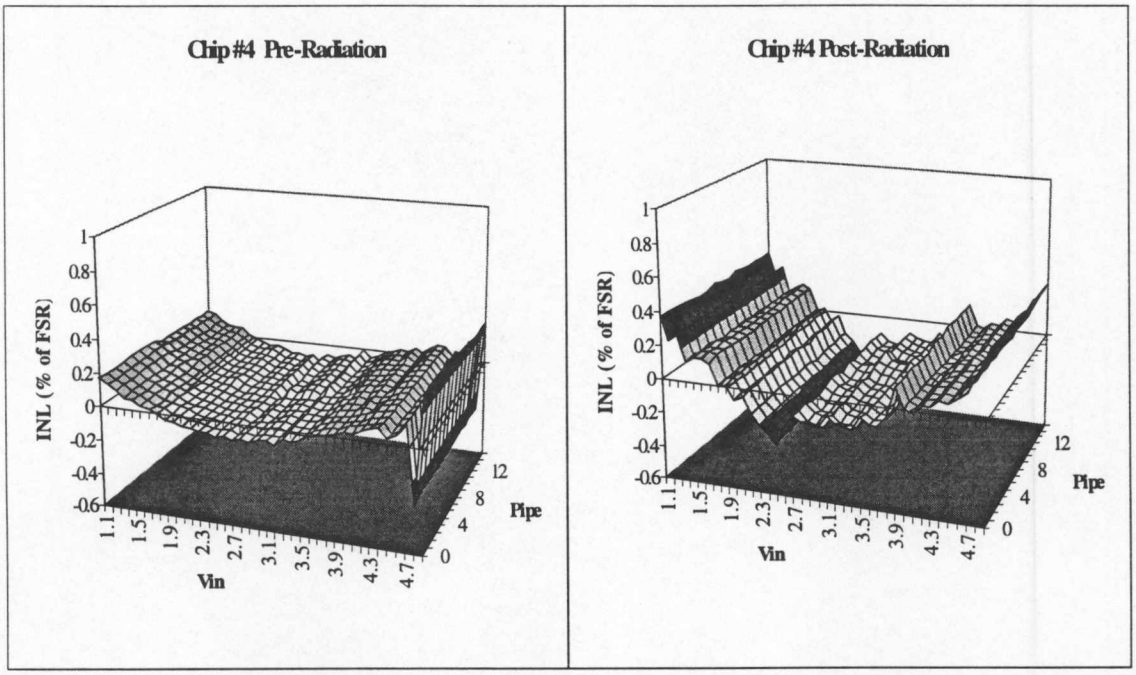


Figure 5 - 3. Chip #4 showing results prior to and after irradiation.

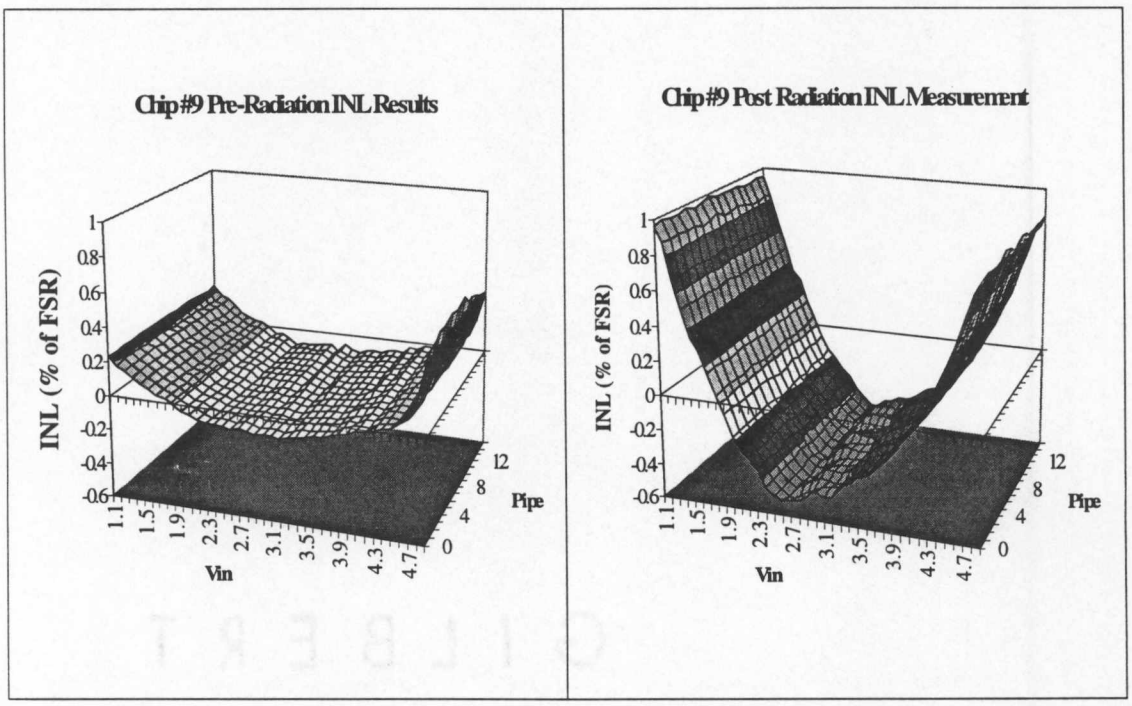


Figure 5 - 4. Chip #9 Pre and Post-radiation AMU/ADC INL results.

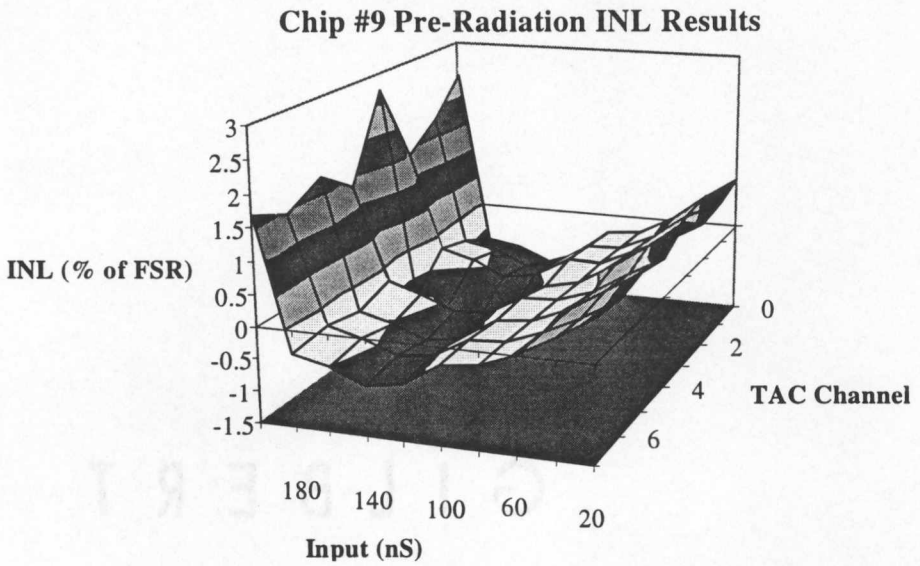


Figure 5 - 5. Chip #9 TAC/ADC Pre-Radiation INL results.

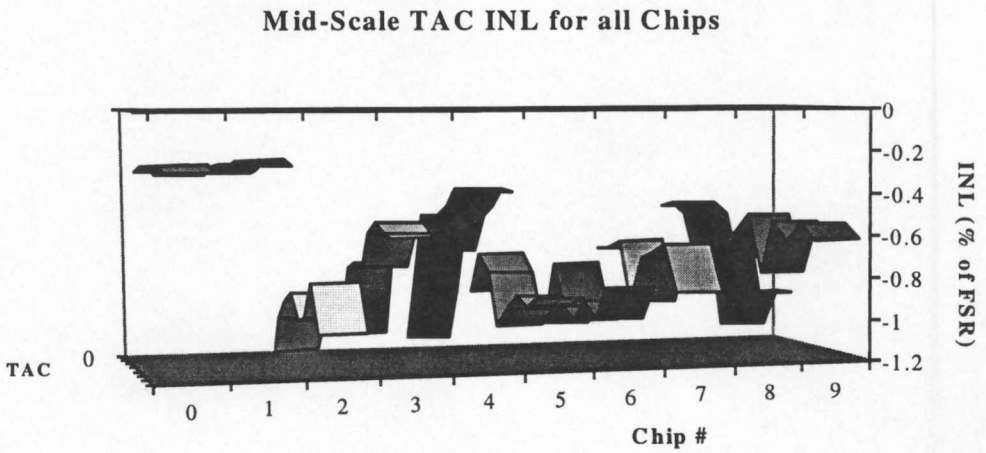


Figure 5 - 6. Mid-scale Pre-Radiation INL, taken with the input pulse sequence separated at 120 nS, for all chips.

Post-Radiation

The damage due to the radiation effects for the TAC/ADC are not very pronounced except for those chips in the ceramic packages. The plastic packages exhibited very small change. In Figure 5 - 7 the pre and post-radiation results for a plastic packaged device can be seen to be very close. For the ceramic packages, results are much more dramatic. The INL for the pre-radiation data is approximately one percent. After radiation, INL increases to approximately six percent. These results again support the evidence that the gold lid on the ceramic packaged devices enhances the radiation dose as seen in Figure 5 - 8.

DNL AMU/ADC Results

Pre-Radiation

Interesting results are noticed between plastic and ceramic packages. The results of a typical DNL plot for a plastic package can be seen in Figure 5 - 9. The design range is from channel ~220 to ~700. The DNL for this range is < 1/2 LSB. This is a typical result for the plastic packaged chips.

The ceramic packages, however, display a problem not associated with the plastic packages. In Figure 5 - 10, a plot of a ceramic packaged device is shown. In plot (a) of Figure 5 - 10, the effects of not grounding the case of the package can be seen. Notice the channel numbers of the channels with the highest DNL value are located in powers of two apart. The channel labeled 386 is the 2^9 bit added to the 2^8 bit and the channel labeled 770 is the 2^10 bit added to the 2^9 bit. These channels are exhibit the effects of the significant bit changes in the counter circuits of the ADC. Most likely, this noise

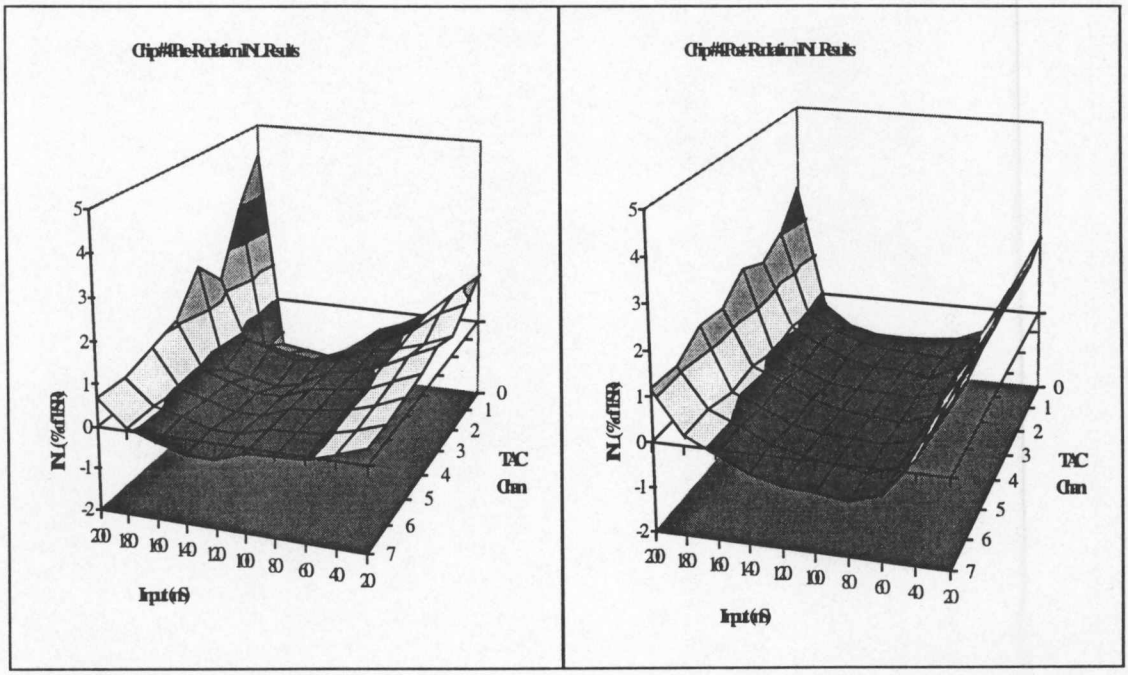


Figure 5 - 7. Chip #4 Pre and Post-Radiation TAC/ADC INL results.

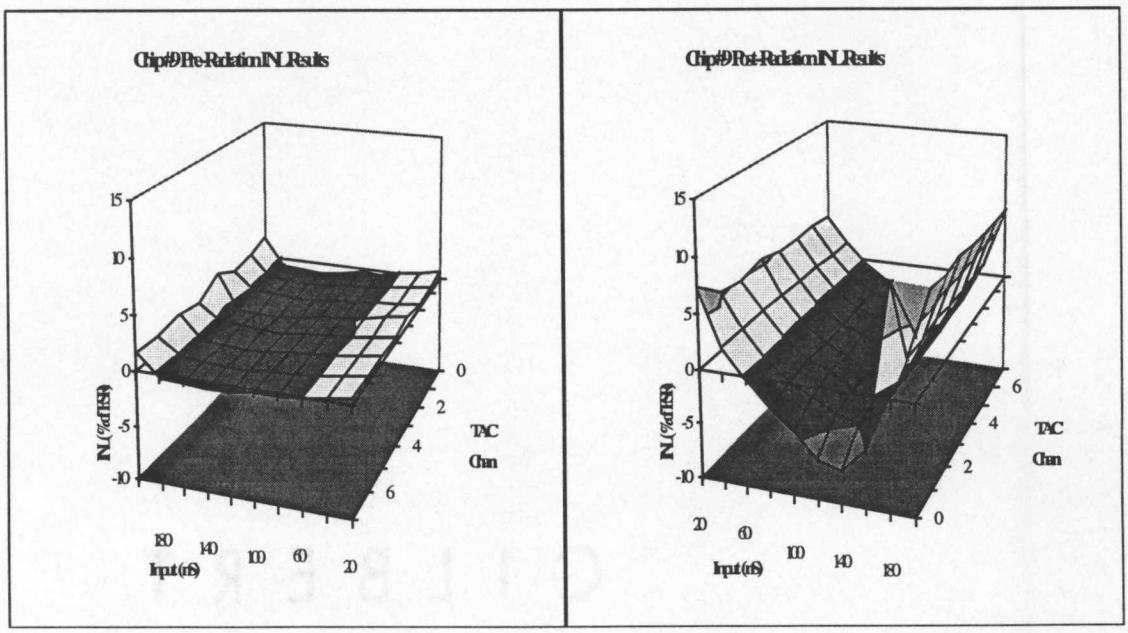


Figure 5 - 8. Chip #9 Pre and Post-Radiation TAC/ADC INL results.

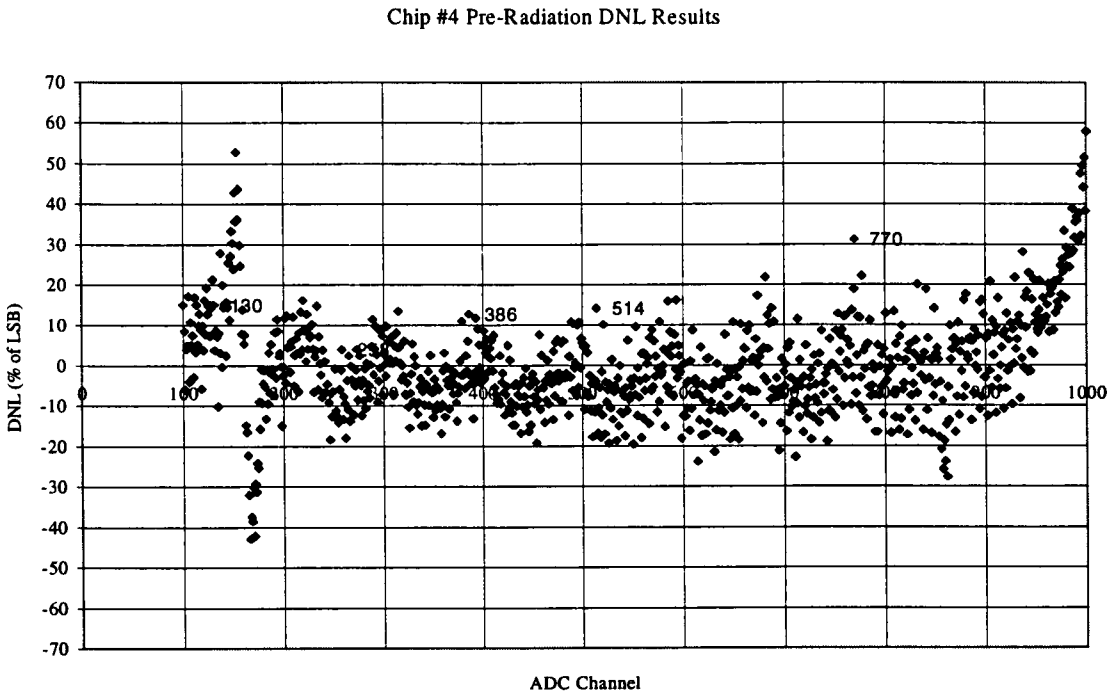
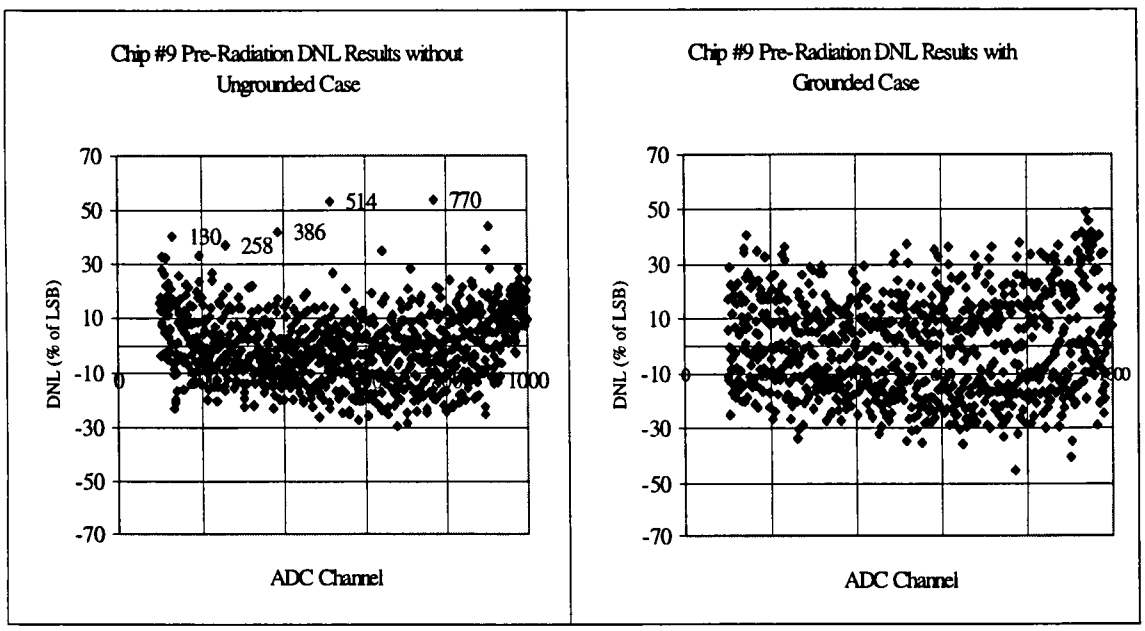


Figure 5 - 9. Chip #4 Pre-Radiation DNL Results.



(a) (b)

Figure 5 - 10. Chip #9 Pre-Radiation DNL Results showing the different effects of the case being grounded. (a) Ungrounded case, (b) grounded case.

disturbance is being introduced by transmission of the digital noise created in the ADC counter circuitry through the wire bonds.

In plot (b) of Figure 5 - 10, the results are shown with the case of the ceramic package grounded. Notice that the significant bit effects are no longer present. Also, notice that the DNL is now below the 1/2 LSB point for the entire operating range. This data supports the evidence that the digital noise is being transmitted through the wire bonds. This seems to imply that, upon grounding the lid, a shield is introduced that eliminates the transmitted signal from the wire bonds. Since the plastic packages do not seem to exhibit this problem, it also indicates that the metal surrounding the chip (when it is not grounded) tends to enhance the propagation of the noise over the chip area.

Post-Radiation

The effects of radiation on the chip are more dramatic when looking at the DNL results. The plastic packages as well as the ceramic packages both reveal an extensive amount of damage.

A plot of the post-radiation DNL results for a representative plastic package can be seen in Figure 5 - 11. This device is no longer within the 1/2 LSB of accuracy over the designed operating range. In the mid-scale region, the damage seems to be most apparent. The dynamic range, however, is still approximately the same as before.

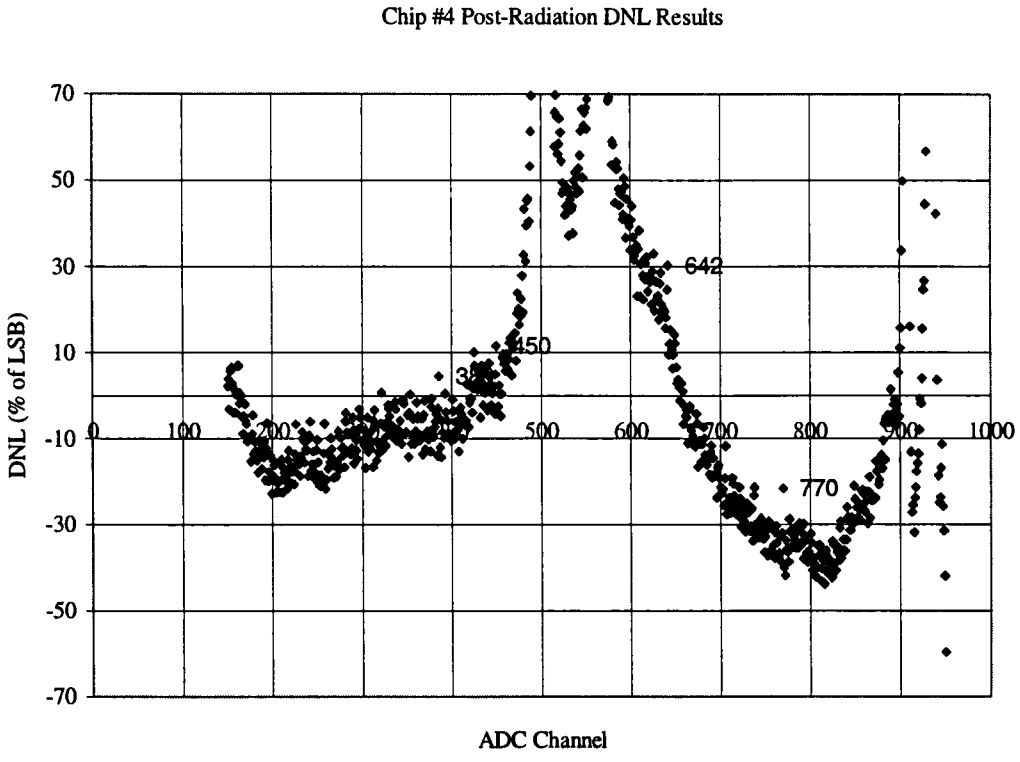


Figure 5 - 11. Chip #4 Post-Radiation DNL Results.

The ceramic packages likewise exhibit a peculiar DNL result. This can be seen in Figure 5 - 12. The dynamic range seems to have been limited to around the count of approximately 840 (i.e. ~ 100 channels; this equates to about a loss of .44V of dynamic range) which is still above the designed range of use. It is noted, however, that the ceramic packaged devices are also not within 1/2 LSB of accuracy for the designed operating range.

Although the devices are affected dramatically, they do not have any missing codes within the designed operating range after the irradiation.

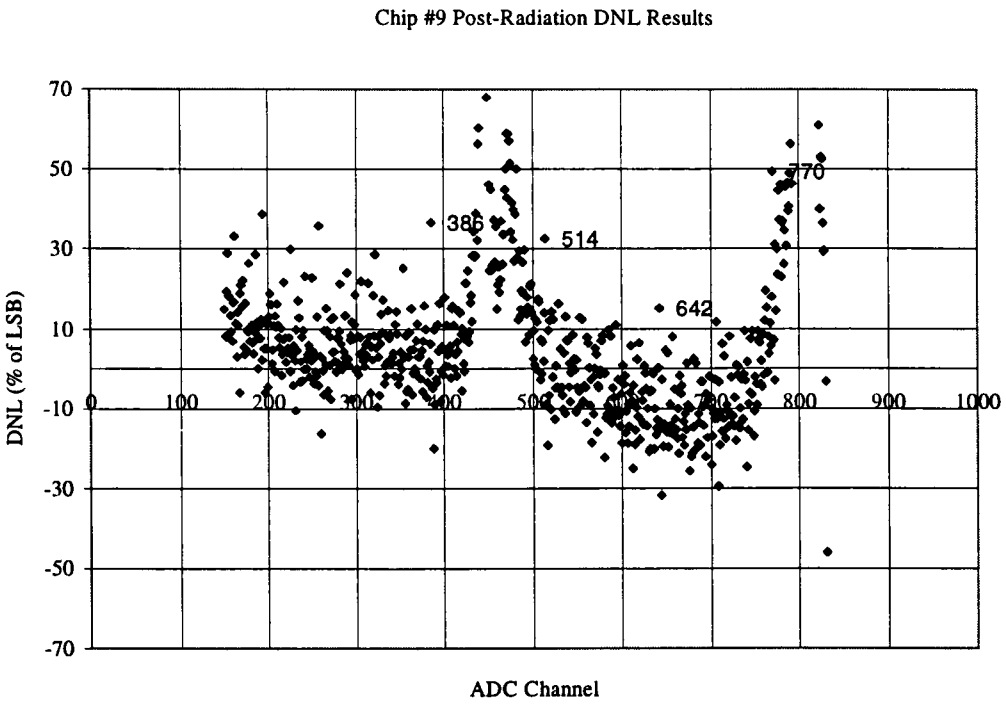


Figure 5 - 12. Chip #9 Post-Radiation DNL Results.

DNL TAC/ADC Results

Pre-Radiation

The DNL measurement results for the two packages are similar. There does not seem to be any distinguishing features. There is a periodic appearance to the DNL results as can be seen in **Error! Reference source not found.** and Figure 5 - 14 (both of the package types). The reason for this effect is not known. Mostly likely, this is a by-product of the method used for testing.

Noise AMU/ADC Results

The noise measurements of the AMU/ADC turned up an interesting result. The

Chip #4 Pre-Radiation DNL Results

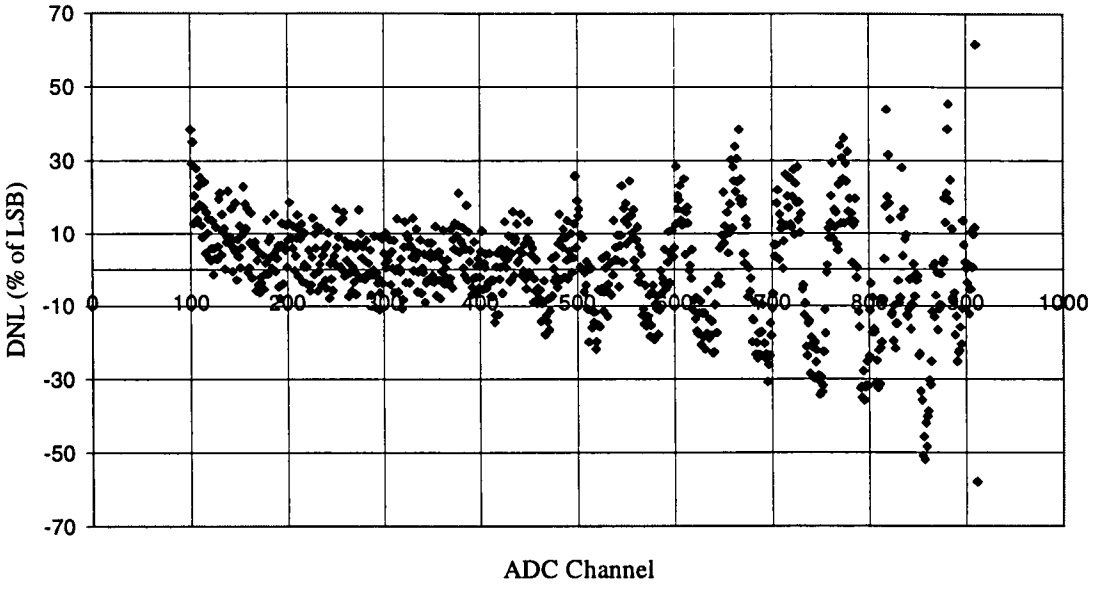


Figure 5 - 13. Chip #4 Pre-Radiation DNL Results.

Chip #9 Pre-Radiation DNL Results

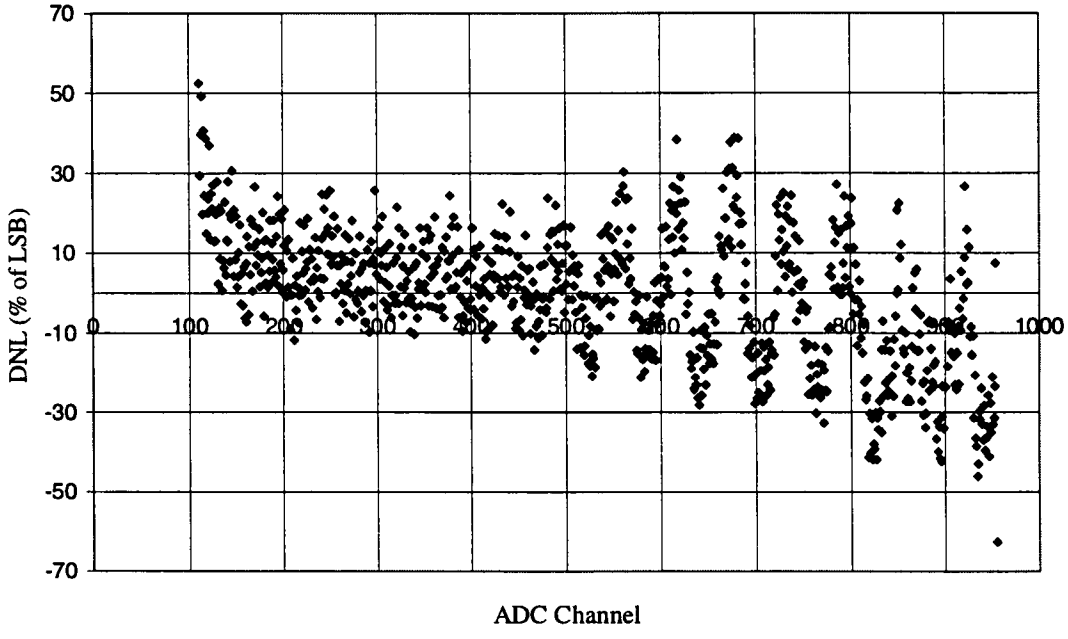


Figure 5 - 14. Chip #9 Pre-Radiation DNL Results.

Post-Radiation

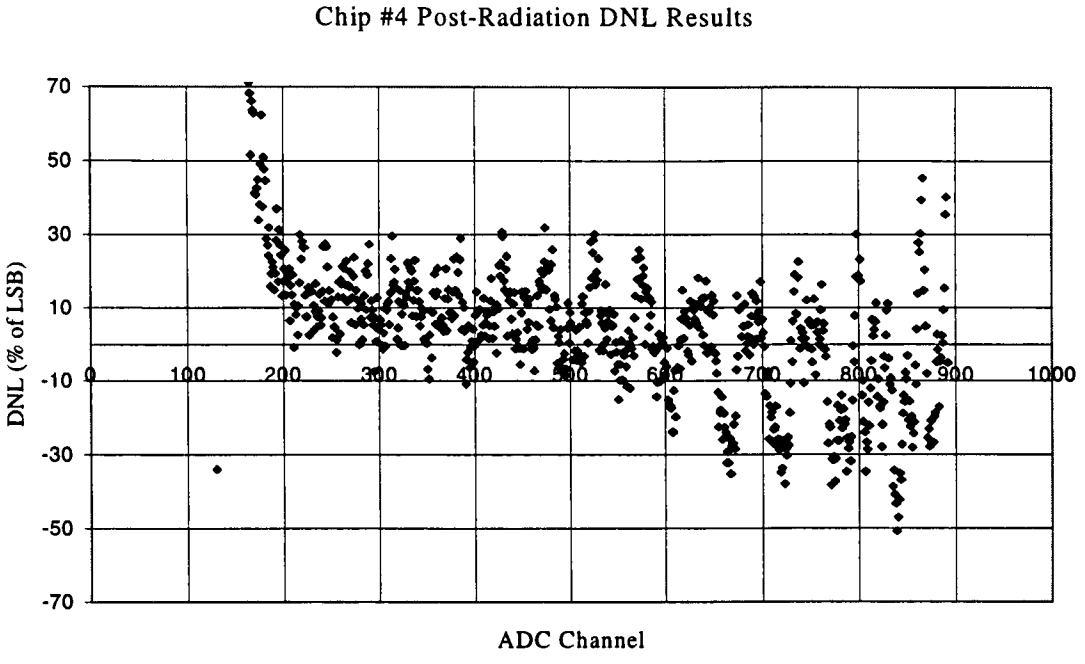


Figure 5 - 15. Chip #4 Post-Radiation DNL Results

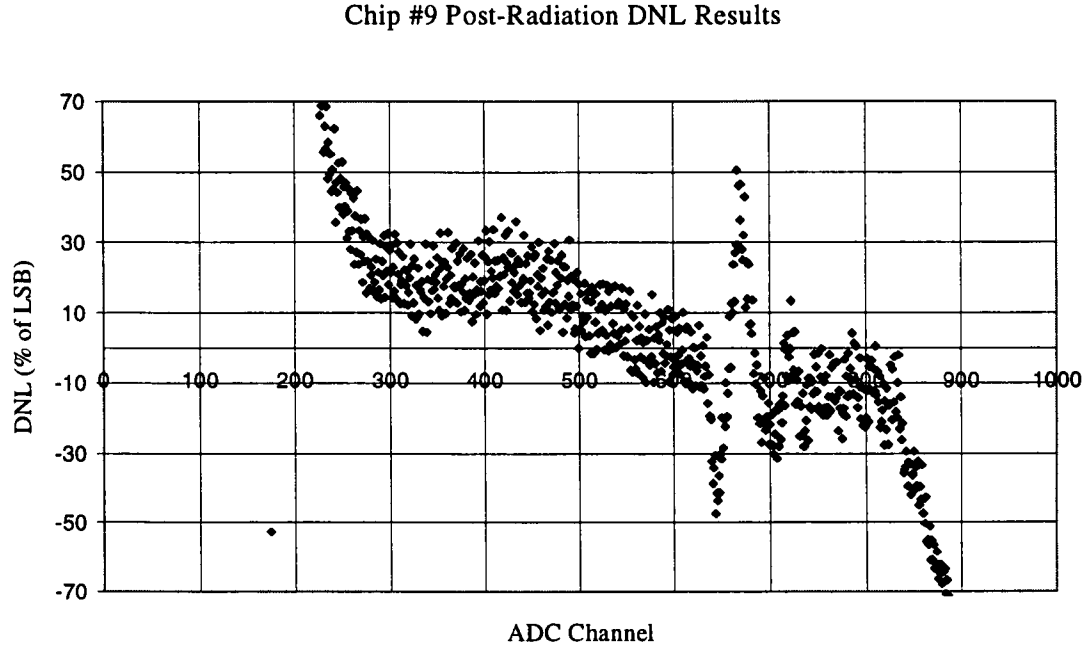


Figure 5 - 16. Chip #9 Post-Radiation DNL Results.

interesting result is the fact that pipes 8-15 have approximately double the noise of pipes 0-7. Notice in Figure 5 - 17 that one can see that noise increases for pipes 4, 8, and 12. The source of the noise has been found to be a lack of filtering at the input of the AMU. At the input to the AMU, there are four banks of four jumpers connected in series. The effects of noise injected at these four banks of jumpers can be seen in Figure 5 - 17 (i.e. Pipes: 0-4, 4-7, 8-11, 12-15). After realizing the source of the problem, filter capacitors were placed at the inputs of each bank of jumpers. The results were significantly better. Chip #7 has been used to verify the effectiveness of the filter capacitors. This is a ceramic device that has not been irradiated. Figure 5 - 18 shows chip #7 prior to the addition of the filter capacitors. Figure 5 - 19 shows the filtered results. The effects of noise from the banks of jumpers can still be seen but are much less noticeable. One could probably take out this noise by applying capacitive filters to the input of each pipe (i.e. all 16 inputs). Due to time constraints, this was not verified. These results are close enough to those that will be presented for the TAC/ADC to justify this decision. From this result, it is evident that consideration of the input connection configuration of the AMU is important. For the performance testing, capacitance is not important since the inputs are at DC levels. However if one needs to have other types of signals where there are fast rise and fall times, the capacitive filtering on the input could create problems. It is important that the connections at the input be as short as possible to reduce the effects of noise coupling to the AMU.

Pre-Radiation

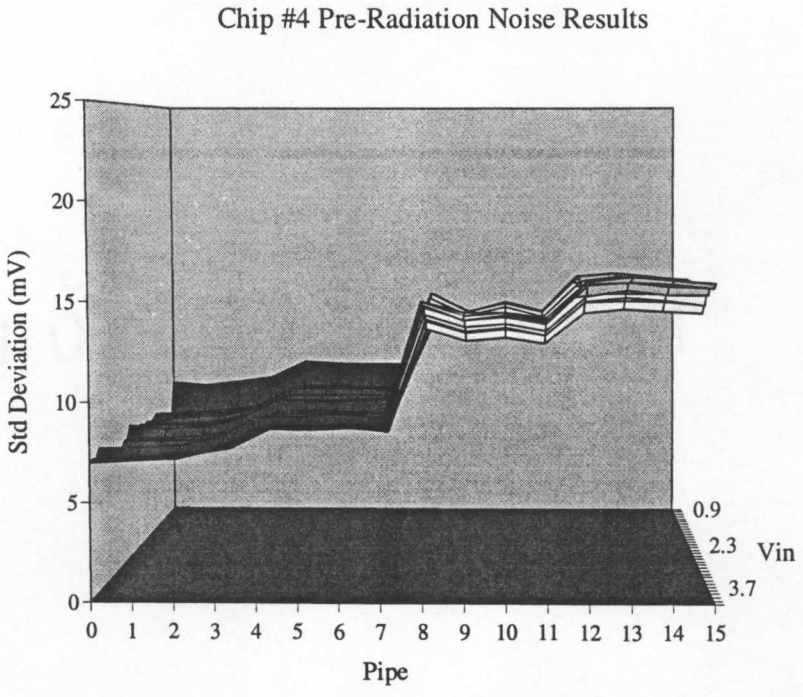


Figure 5 - 17. Chip #4 Pre-Radiation Noise Results indicating noise introduced at the input.

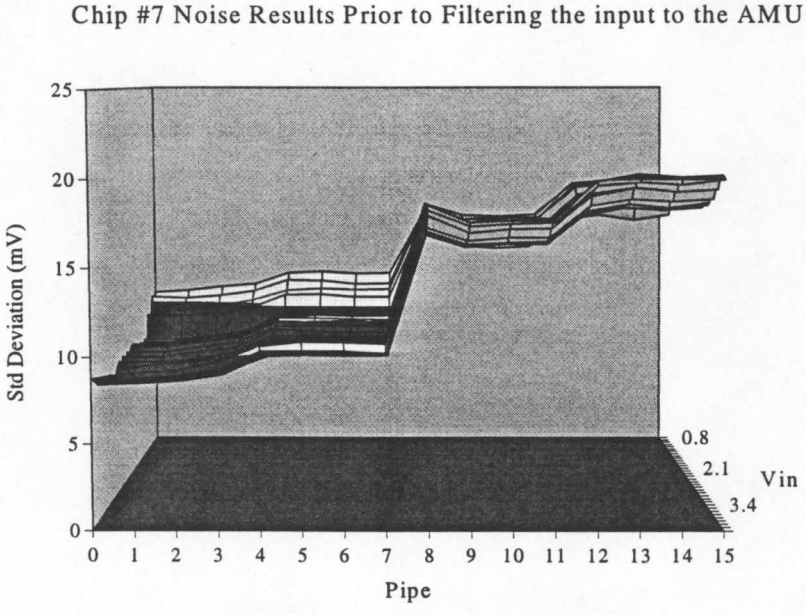


Figure 5 - 18. Chip #7 Noise Results Prior to Filtering the input to the AMU/ADC

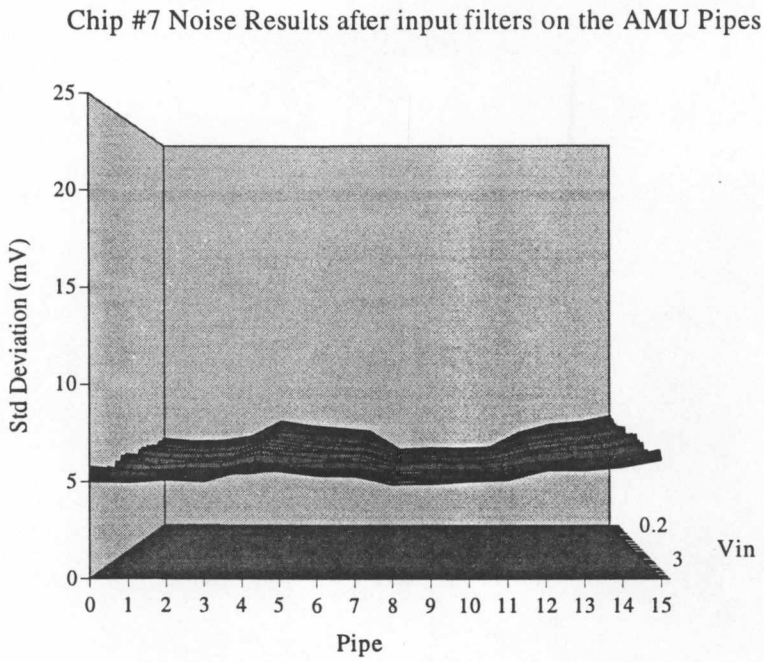


Figure 5 - 19. Chip #7 Noise Results after filtering the input to the AMU/ADC.

The effects of radiation had no noticeable results on the noise measurement. So, the original data that have the noise coupling effects of the input jumpers of the AMU will be shown. This data is prior to irradiation. Keep in mind that the actual noise magnitude will be in the range of 5-6 mV as shown in Figure 5 - 19 and that will be presented in the TAC/ADC noise results section.

In Figure 5 - 20 and Figure 5 - 21, the pre-radiation results for both packages can be seen. There are no surprising differences in the two packages. From this, one can conclude that noise considerations does not impact the packaging choices.

Figure 5 - 22 and Figure 5 - 23 show the post radiation results. Noise values have not increased. However, the dynamic range again can be seen to have been decreased.

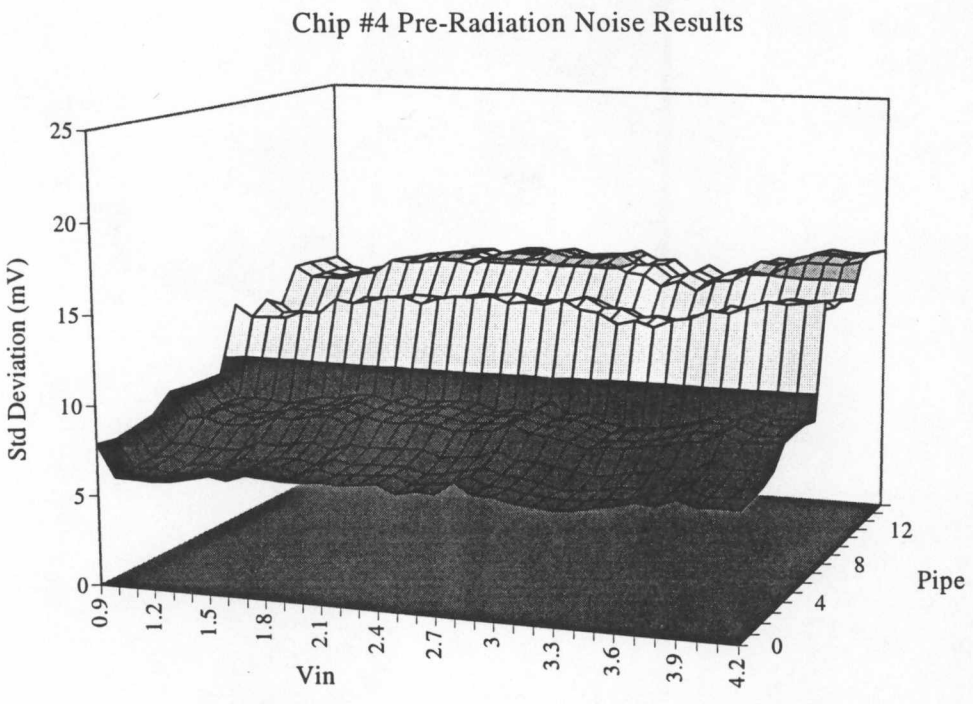


Figure 5 - 20. Chip #4 Pre-Radiation Noise Results.

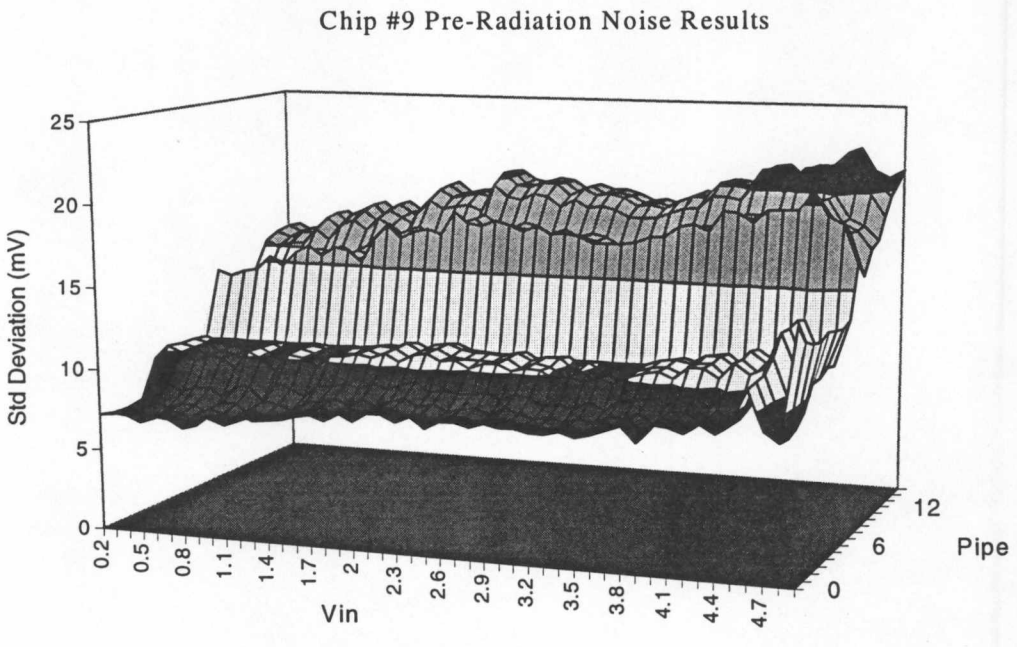


Figure 5 - 21. Chip #9 Pre-Radiation Noise Results.

Post-Radiation

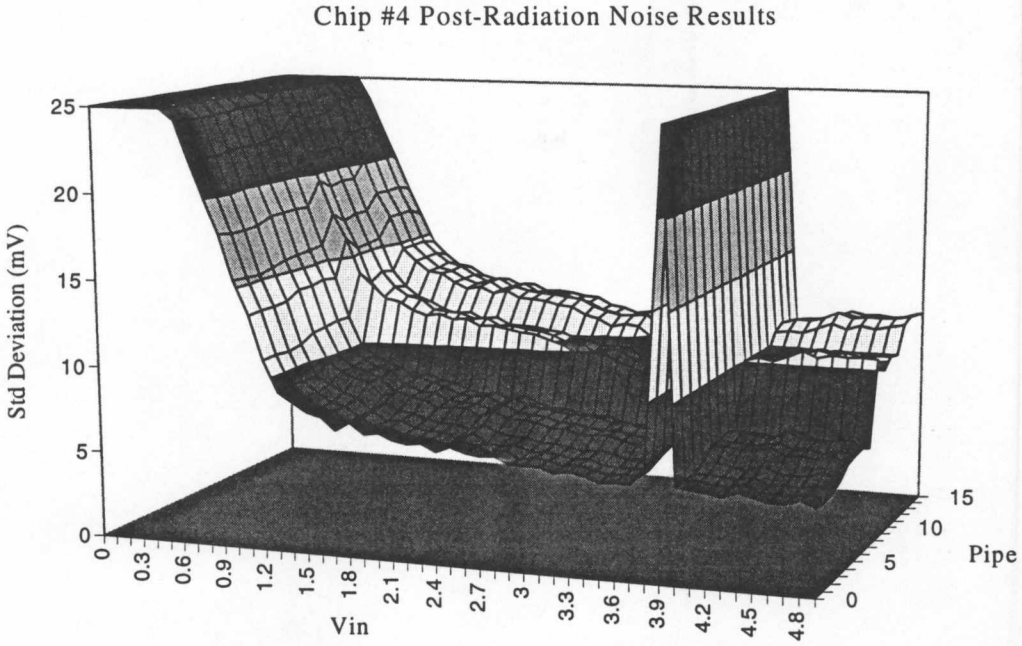


Figure 5 - 22. Chip #4 Post-Radiation Noise Results.

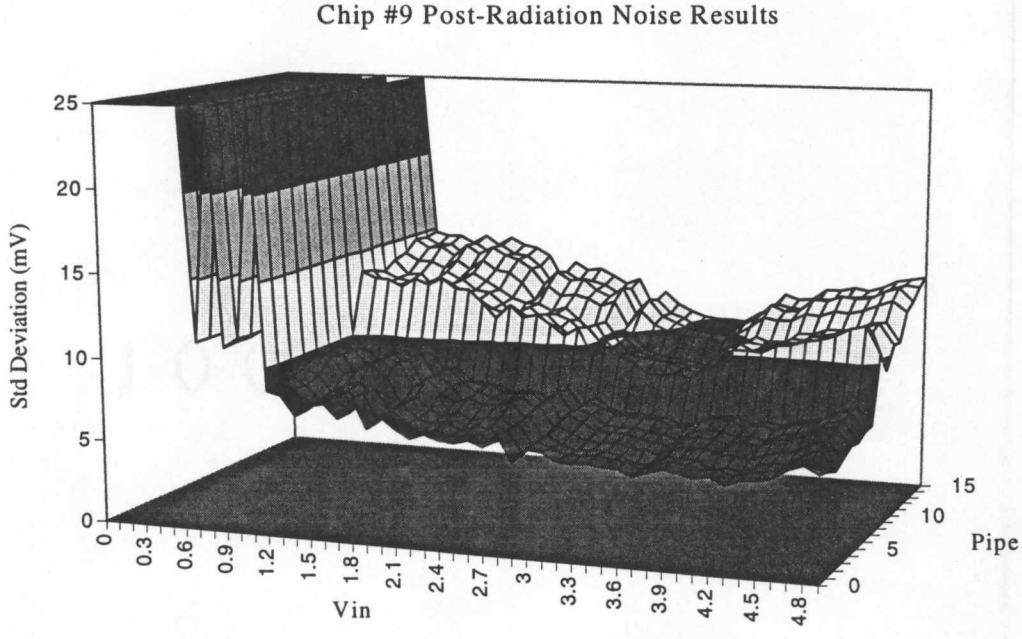


Figure 5 - 23. Chip #9 Post-Radiation Noise Results.

Noise TAC/ADC Results

The noise measurements of the TAC/ADC yield almost the same noise values of ~5 mV. The following two figures show that plastic and ceramic packages results have similar results. The pre-radiation results can be seen in Figure 5 - 24 and Figure 5 - 25.

The effects of radiation on the ceramic packaged devices were large. The plastic packaged device did not change much (see Figure 5 - 26). The TAC circuitry is much more sensitive to the effects of radiation than the AMU circuitry. The post-radiation plot for the ceramic device can be seen in Figure 5 - 27.

Gain: AMU/ADC Results

The AMU/ADC's gain is ideally 0.9 V/V. The ceramic packages seem to come closest to the ideal value. There are no exciting results revealed in this measurement. The results can be seen in Figure 5 - 28 for pre-radiation and Figure 5 - 29.

The effects of radiation are noticeable. The greatest change is for chip #9 and is ~10%. All others are less than ~ 5 %. Again, the ceramic packages are affected the most.

Gain: TAC/ADC Results

The TAC/ADC's gain is not as close to the ideal value as is the AMU/ADC. This, again, is most likely due to the number of points available to calculate the best fit line. Therefore the accuracy will be less (i.e. ~5 % from the ideal of .9 V/V). The pre and post results yield the same conclusions. There is a ~15% to 20% decrease after irradiation. The pre and post-radiation plots can be seen in Figure 5 - 30 and Figure 5 - 31 respectively.

Pre-Radiation

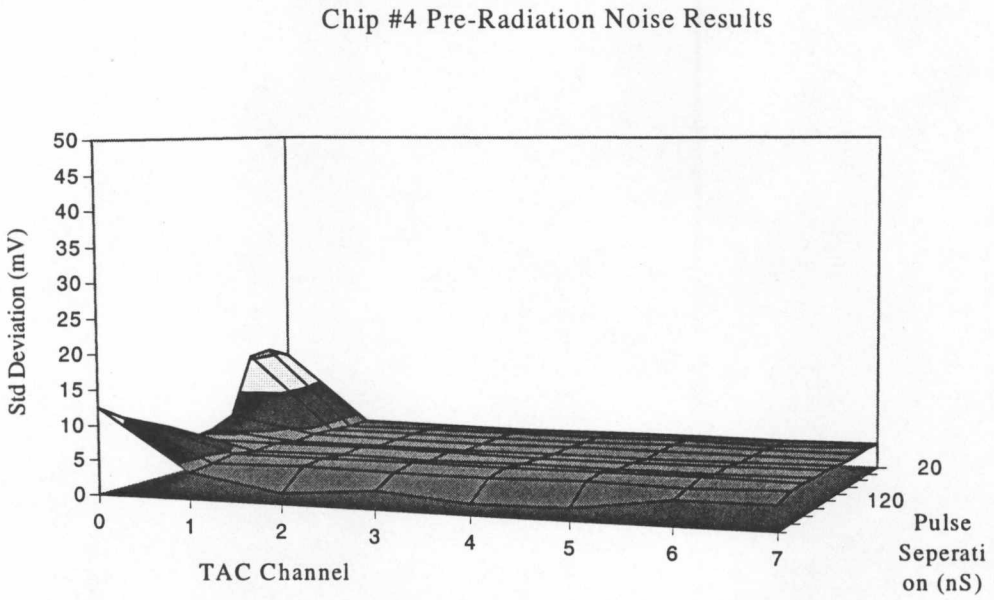


Figure 5 - 24. Chip #4 Pre-Radiation Noise Results.

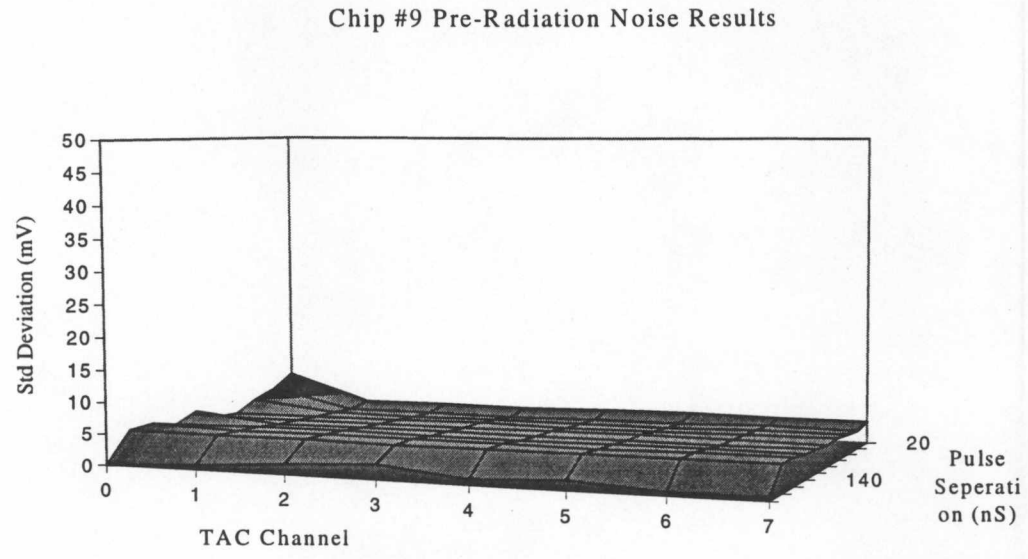


Figure 5 - 25. Chip #9 Pre-Radiation Noise Results.

Post-Radiation

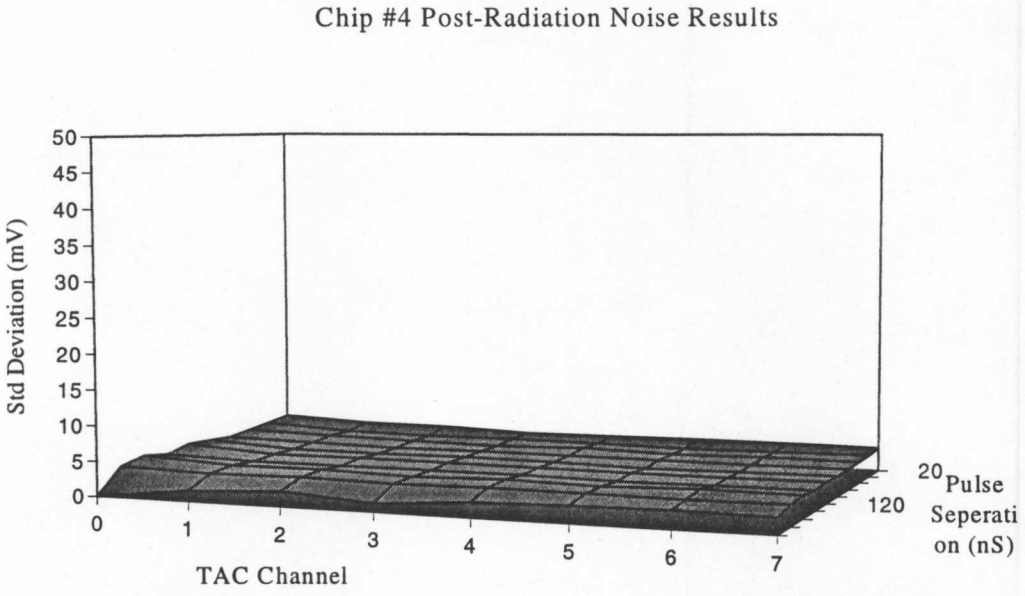


Figure 5 - 26. Chip #4 Post-Radiation Noise Results

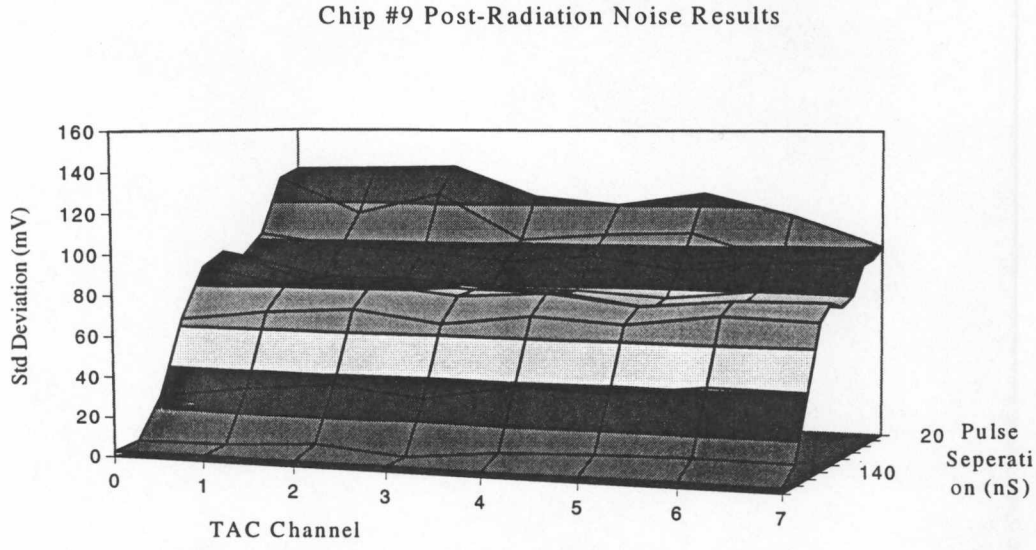


Figure 5 - 27. Chip #9 Post-Radiation Noise Results.

Pre-Radiation

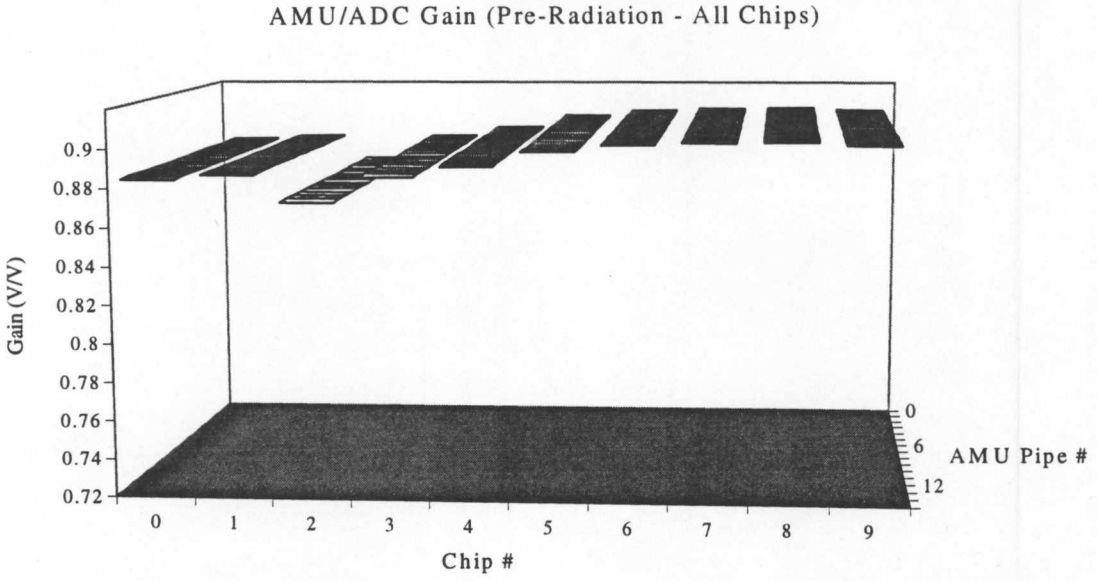


Figure 5 - 28. AMU/ADC Gain for all Chips before irradiation.

Post-Radiation

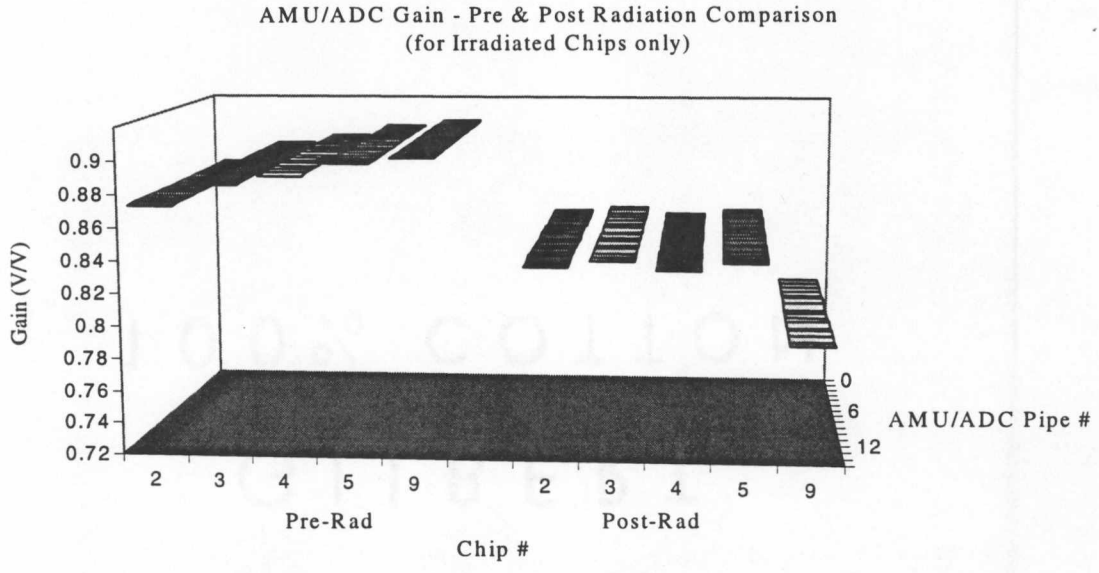


Figure 5 - 29. AMU/ADC Gain - Pre & Post Radiation Comparison (for Irradiated Chips only)

Pre-Radiation

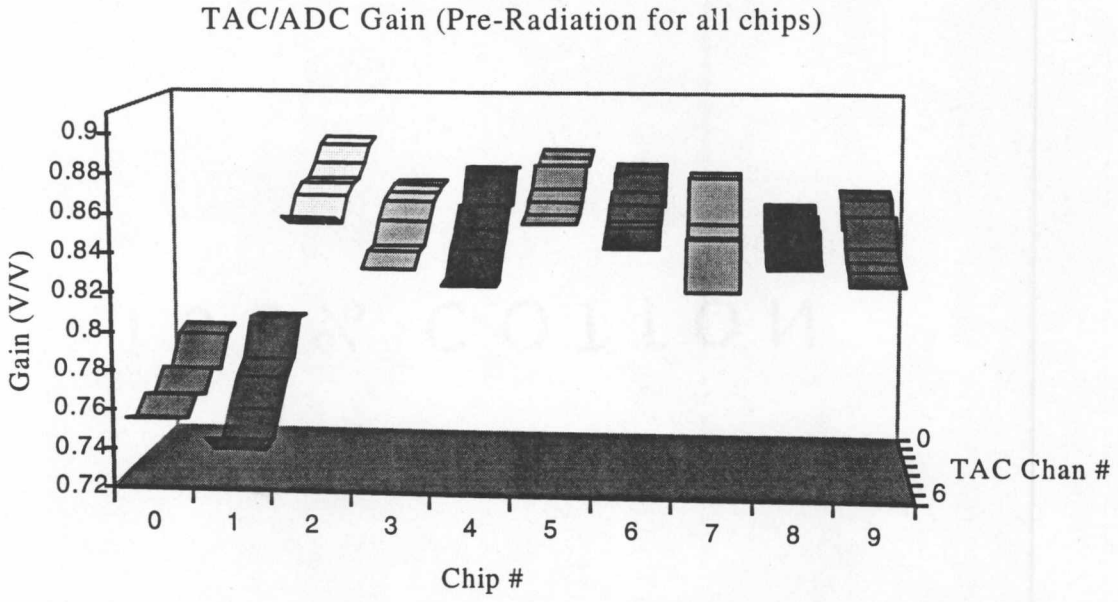


Figure 5 - 30. TAC/ADC Gain (Pre-Radiation for all Chips))

Post-Radiation

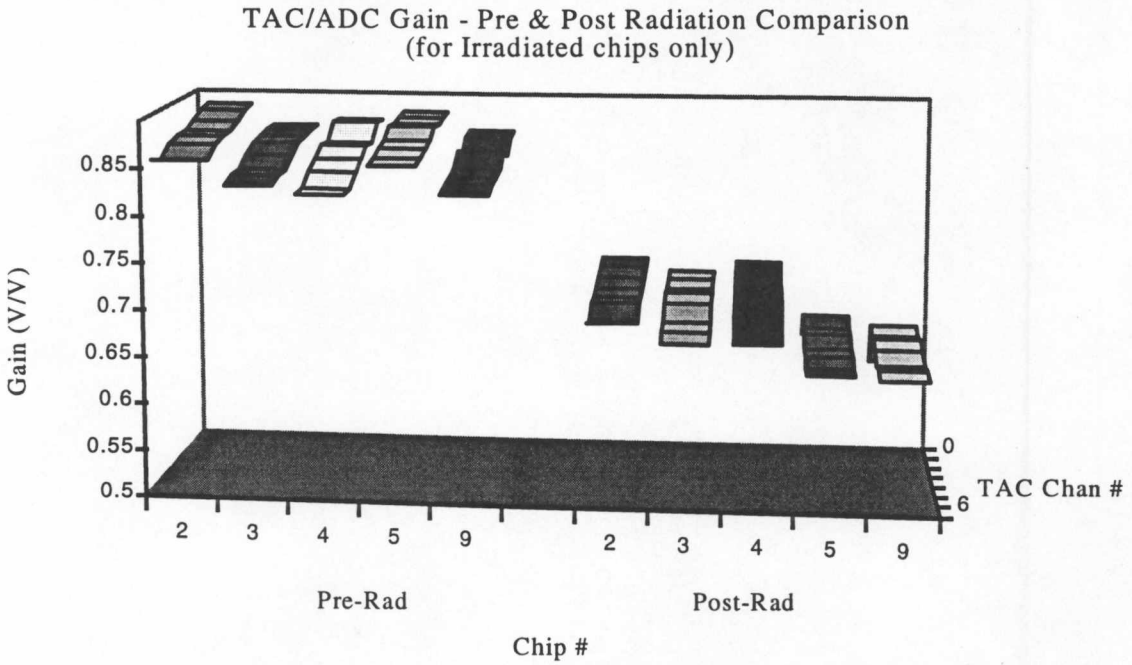


Figure 5 - 31. TAC/ADC Gain - Pre & Post Radiation Comparison (for Irradiated chips only)

Chapter 6 Pedestal Correction: A Noise Reduction Method

Pedestal Correction is a method that is being evaluated with several different configurations to reduce the effects of coherent noise in analog memory circuitry. One configuration is to use a differential subtraction method [CUNITZ1]. The problem with this method is that the size of the analog memory has to be doubled. An alternative to this method is to use a single pipeline as a reference to the pipelines [BRITTON1]. This will provide a quasi-differential subtraction from the other pipelines. Figure 6 - 1 shows a graphical representation of the single pipeline subtraction method.

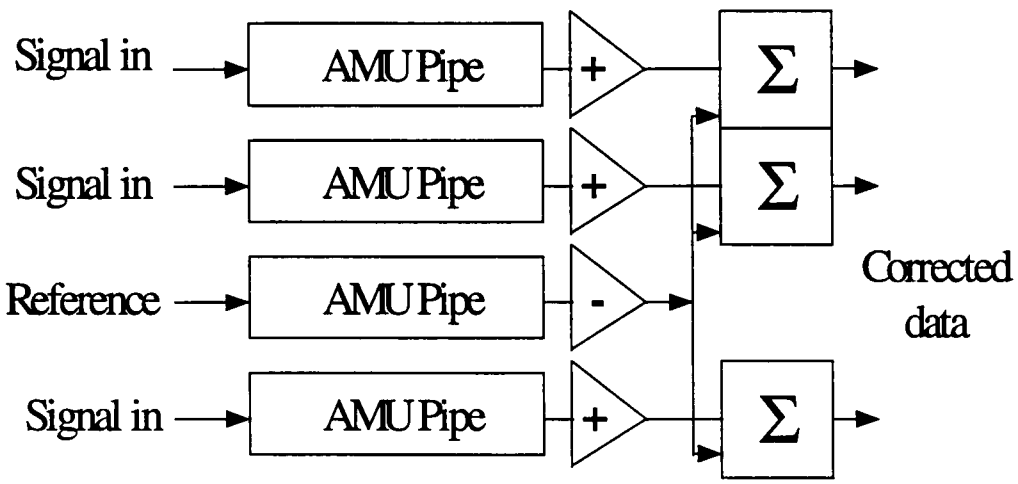


Figure 6 - 1. Single pipe-line subtraction method.

As reported in [BRITTON1], unknowns for this method are:

- Correlation between adjacent channels
- Correlation as a function of radiation dose

In this chapter, these unknowns will be explored for the 16-channel analog memory configuration.

Sources of Coherent Noise

Applications which use the analog memory circuitry are exposed to several sources of coherent noise. Particularly for the ATA chip itself, the internal digital addressing logic induce a potential coherent noise source across the device. The associated ADC circuitry is another potential source. As seen in chapter 5, the ADC counter proves to be the most noticeable internal noise source (depending on chip packaging). Exterior to the ATA chip, two large noise sources in the RHIC enviroment are the digital noise through the power supplies and synchronized noise due to the master clock.

Theory

If the predominate noise in the system is identical in all of the analog memory devices, cancellation should be achieveable. There are uncertainties which must be addressed (The following is from [BRITTON1]):

1. Unequal noise pickup in each channel. - If the noise pickup is different in each channel, then cancellation will not occur. In fact this could actually worsen the result rather than improve it.
2. Unequal gains in the input devices to the analog memory. - Same effect as above.
3. Uncorrelated noise between channels. - This could be a result from processing mis-matching from channel to channel. Also, each channel will have individual inherent noise associated with each channel amplifier. The net effect of two uncorrelated noise sources is predicted in equation (6 - 1).

$$e_{subtracted} = \sqrt{e^2_{channel} + e^2_{reference}} \qquad (6 - 1)$$

4. Unequal pedestal contributions from channel to channel. - This area of concern results from noise generated within the device under test. For this case, if the devices have any imbalance, in the amount of noise that is generated from channel to channel, the effects of this correction method will be to aggravate the noise situation rather than improve it.

This thesis will investigate only the effects of item 4. (The effects of item 1 has probably to a small degree impacted the data taken due to the test fixture configuration.) The intent is to measure the effectiveness of the pedestal correction cancellation technique before and after irradiation.

Types of Tests

Two types of testing were accomplished. The first set of tests consists of using the pedestal subtraction with the inherent noise of the device and its surroundings, which will be called background noise. The second set of tests targeted coherent noise for the entire device. This technique has been applied by checking its cancellation effectiveness on the circuit background noise and then, to check for the coherent noise, injecting noise in the power supply terminals of the chip. The background noise test configuration had no special alteration to the orginal test fixture as discussed in chapter 4 (refer to Figure 4 - 6). In order to inject noise into the power supply terminals of the DUT, the circuit in Figure 6 - 2 had to be added to the test fixture. A power transistor was used to provide enough current drive for the chip. A random noise generator was used at the

input to the base of the power transistor. The random noise generator was offset to 5.7 volts to bias the transistor properly so the transistor circuit would act as a 5V power supply for the chip.

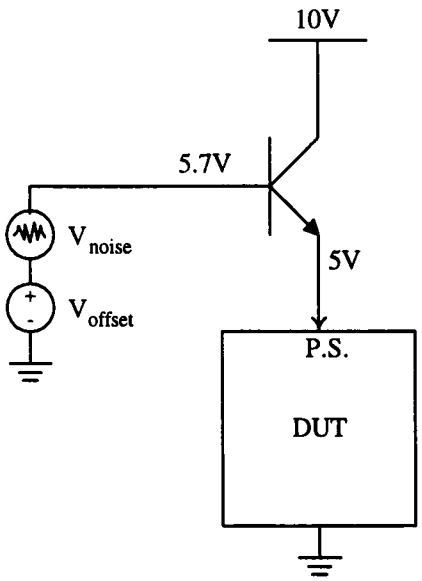


Figure 6 - 2. Schematic diagram of power supply injected noise.

The magnitude of V_{noise} source was tested at several values starting at 100mV up to 700mV in increments of 100mV. Effects of the noise source was not noticeable until its magnitude was about 400mV. It became overwhelming to the DUT (i.e. functionality was degraded) at about 700mV. Data taken at 500mV was chosen for the test results. These results will be presented later in this chapter.

The Method

The technique used for this memory configuration was much the same as the other configurations. For this case, the following approach was used:

- a reference voltage is connected to the input of the designated reference pipe and the remaining pipes were connected to a common voltage source

- the entire memory matrix (16x16) is written with their respective inputs
- a read cycle is done for the entire matrix and is stored
- the reference is subtracted from all others and the results stored
- the standard deviation for the raw and the subtracted data are calculated, converted to rms volts, and stored by a computer program called 'subt.exe' (see Appendix G).

These steps were done for reference voltages of 1.5V, 2.5V, and 3.5V. For each reference voltage, a three voltage series of 1.5V, 2.5V, and 3.5V was provided to the input of the remaining pipes. This yielded three sets of data for each reference voltage for a total of nine data sets per reference pipe. There were a total of three pipes chosen for references. The pipes are # 0, 7, and 15. These were chosen to give results for the edges (pipe # 0 and #15) and the middle of the memory matrix (pipe #7). This gave a total of 27 sets of data.

It should be noted that these tests are not just providing results of the AMU alone but also the added effects of the ADC (which prove to be negligible). As discussed earlier, the configuration of this chip does not allow one to separate the AMU for the ADC, so the effects of both are seen.

Background Noise Results

Data for these measurements are presented in Figure 6-3. This figure shows the resulting data for the reference pipe voltage equal to the common input voltage. The data from all different combinations of the voltages and the common input voltages showed the best results are when $Vref = Vin$.

The pre-radiation and post-radiation results are very similar. The subtracted results for both cases is about 2mV regardless of the raw noise. This shows the consistency of the pedestal subtraction method prior to and after irradiation. In the pre-radiation results, the reduction of noise for the background noise data is not pronounced in all cases as the injected noise data that will be presented in the following section. This is primarily due to the influence of incoherent noise that is available at the inputs of the 16 pipes of the AMU.

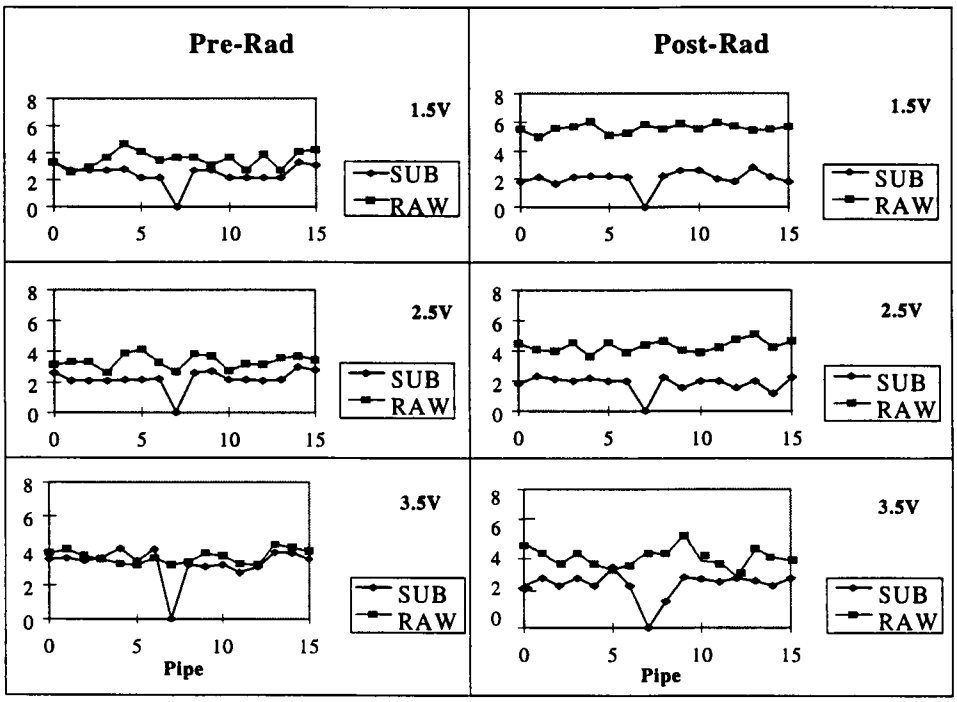
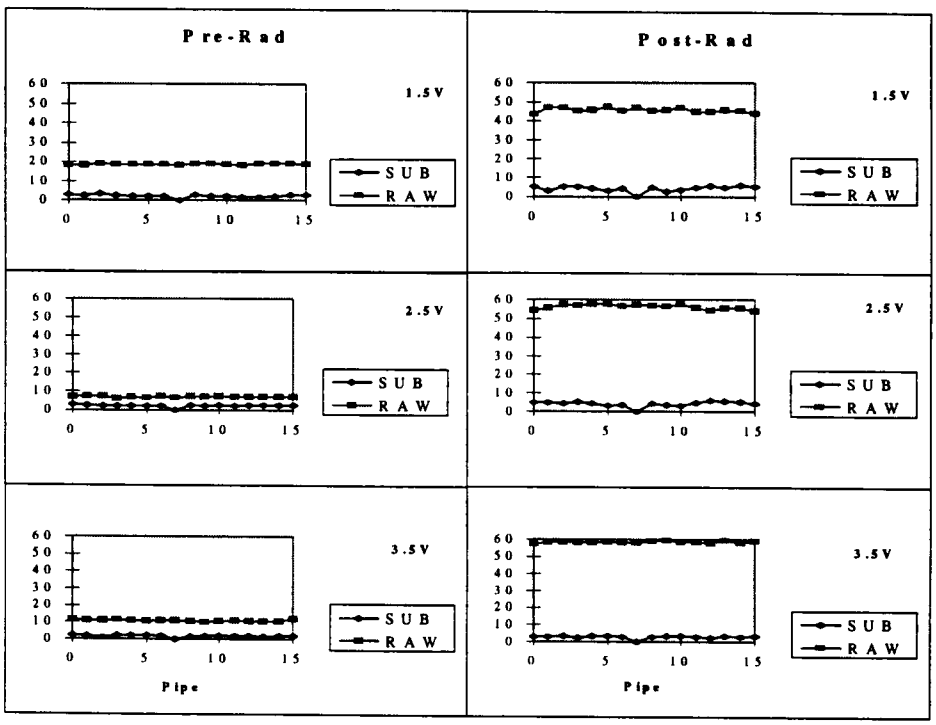


Figure 6-3. Background Noise Results (in mV) with *Vref* = *Vin* & channel #7 as reference pipe (a) for the pre-rad data (b) and for the post-rad data.

Injected Noise Results

This test was completed by injecting 500mV of noise into the power supply of the chip with a random noise generator. With such a large magnitude, the background noise will be absorbed and the benefits of the pedestal subtraction method can be seen more easily. The results are presented in Figure 6-4.

The magnitude of the raw noise has increased. However, the subtracted results yield the same 2mV corrected result. This again shows the correction method significantly reduces the magnitude of the noise. So, if the devices have either coherent noise or just background noise, the subtracted data will yield a constant result of ~2mV of pedestal noise if $V_{in} = V_{ref}$.



(a) (b)

Figure 6-4 Injected Noise Results (in mV) with *Vref* = *Vin* & channel #7 as reference pipe (a) for the pre-rad data (b) and for the post-rad data.

Chapter 7 Conclusion

This thesis described the performance of the ATA chip, reported the yield of the Orbit Semiconductor 1.2μ N-Well CMOS process for a large area chip, presented radiation effects on the circuitry and relevancy to packaging selection, and presented the pedestal subtraction technique. In this chapter, a summary of the findings will be presented.

Yield

Yield of the Orbit Semiconductor 1.2μ N-Well CMOS process was ~ 54%. This was done for a 2700 die run. The yield estimated by Orbit Semiconductor would apparently best be done by the Seed model which predicted 51.84% yield.

Linearity

The linearity of the ATA chip is sufficient to allow a ten bit resolution over the designed operating range. In Table 7 - 1 and Table - 2, a summary of the INL results can be seen. The only item of interest comes from the effects of radiation damage. The ceramic packages (due to the gold lids) exhibit greater damage than do the plastic packaged devices. The results for both the AMU/ADC and the TAC/ADC support this. Variations between channels and packaging (not irradiated) were minimal.

The DNL is less than 1/2 LSB. See Table 7 - 3 and Table 7 - 4 for a summation of the DNL results. The ceramic packaging with the gold lids tend to expose potential problems. The ADC counter noise is prevalent in this packaging. This can be eliminated by adding a ground connection to the package lid. The DNL results show a degradation

Integral Nonlinearity

Table 7 - 1. AMU/ADC INL Results.

Packaging	Pre-Radiation INL (% FSR)	Pre-Radiation Dynamic Range	Post-Radiation INL (% FSR)	Post-Radiation Dynamic Range
Plastic	.15% ↔ -.12%	1.1V → 4.3V	.4% ↔ -.35%	1.1V → 4.3V
Ceramic	.15% ↔ -.12%	1.1V → 4.3V	.9% ↔ -.6%	1.1V → 4.3V

Table 7 - 2. TAC/ADC INL Results.

Packaging	Pre-Radiation INL(% FSR)	Pre-Radiation Dynamic Range	Post-Radiation INL(% FSR)	Post-Radiation Dynamic Range
Plastic	1% ↔ -1%	20 nS → 200 nS	1% ↔ -1%	20 nS → 200 nS
Ceramic	1% ↔ -1%	20 nS → 200 nS	6% ↔ -5%	20 nS → 200 nS

Differential Nonlinearity

Table 7 - 3. AMU/ADC DNL Results.

Packaging	Pre-Radiation DNL (% LSB)	Pre-Radiation Dynamic Range	Post-Radiation DNL(% LSB)	Post-Radiation Dynamic Range
Plastic	< 1/2 LSB	.5V→ 4.2V	> 1/2 LSB	1.5V→ 3.5V
Ceramic	< 1/2 LSB	.5V→ 4.2V	> 1/2 LSB	1.5V→ 3.5V

Table 7 - 4. TAC/ADC DNL Results.

Packaging	Pre-Radiation DNL(% FSR)	Pre-Radiation Dynamic Range	Post-Radiation DNL(% FSR)	Post-Radiation Dynamic Range
Plastic	< 1/2 LSB	20 nS → 200 nS	< 1/2 LSB	45 nS → 170 nS
Ceramic	< 1/2 LSB	20 nS → 200 nS	< 1/2 LSB	45 nS → 145 nS

of dynamic range after irradiation. The channel to channel results were not presented for this measurement due to the cumbersome amount of data. Again, there were no abnormalities with channel to channel comparisons with this measurement.

Noise

The noise has been found to be ~5 mV RMS for TAC/ADC and AMU/ADC sections (after filter capacitor applied to the input of the AMU). A potentially significant problem resulting from the input connection configurations of the AMU/ADC must be given special consideration. The result is that noise could adversely affect signals of interest. If filtering is required for the AMU inputs to reduce noise pickup, the resulting time constant from the input capacitor/s could be significant enough to cause detrimental effects for a fast implementation of the ATA chip. Otherwise, the matching for the devices from chip to chip and channel to channel is relatively constant. The AMU/ADC has been measured to have a noise magnitude of ~2 mV. This cannot be measured with a 4.4 mV LSB resolution.

Gain

The gain for the AMU/ADC section of the circuitry is ~0.9 V/V. The TAC/ADC is about 0.86 V/V. The TAC/ADC gain is less because the number of points available for the tests did not provide as much resolution as did the AMU/ADC section. This could be one of the possible reasons for the differences in the measurement.

The chip to chip variations and the channel to channel variations are not significant. However, remember that the manual adjustments associated with the ramp

that were require for every chip, will be a portion of the noticeable differences in the chip to chip gain results.

Pedestal Correction

As presented in chapter 6, the pedestal correction technique reduced the effects of coherent power supply injected noise and to a lesser degree reduced the background noise. The method was not effective for cases where the input voltages are different from the reference voltage. So, if $V_{in} \neq V_{ref}$, the pedestal correction technique most likely will not reduce the noise but rather increase it. This was the case for before and after irradiation. For post radiated devices the method yields a reduction to ~2mV, even if the injected noise approaches 100mV.

Final Thoughts

The ATA chip performed over the designed operating range of 1.5V to 3.5V. The chip to chip and channel to channel variations are minimal. Packaging is, as expected, an important consideration. This is especially true if the devices are going to be located near radioactive sources. The primary consideration for this selection is if metallic lids are going to be used because gold lids enhance the radiation dose received by the die.

It is suggested that if a large chip fabricated by Orbit Semiconductor with the 1.2μ N-Well CMOS process is used, one should plan on ordering twice as many chips as needed.

The pedestal correction method works great if one is able to match the magnitude of the DC value of the reference voltage to that of the input signal. This would most

likely require a dynamic reference voltage source that would be able to follow the input signal so the best possible cancellation would be achievable.

REFERENCES

Chapter 1

[WILKINSON1] D. H. Wilkinson, Proc. Cambridge Philos. Soc. 46, Pt. 3, 508, 1950.

Chapter 2

[SROUR] J. R. Srour, et al., Radiation Effects On and Dose Enhancement of Electronic Materials, Noyes Publications, Park Ridge, NJ, 1984, pp. 8

[SROUR2] Ibid Srour, p.11.

[MA1] T. P. Ma and P. V. Dressendorfer, Ionizing Radiation Effects In MOS Devices and Circuits, John Wiley and Sons, New York, 1989, pp. 35.

[GRAY1] P. R. Gray and R. G. Meyer, Analysis and Design of Analog Integrated Circuits, 3rd Edition, John Wiley and Sons, New York, 1993 pp. 13.

[SROUR3] Ibid Srour, p.17.

[SZE1] S. M. SZE, Semiconductor Devices - Physics and Technology, John Wiley and Sons, New York, 1985, pp.197.

[SROUR4] Ibid Srour, pp. 19.

[SROUR5] Ibid Srour, pp. 64.

[JAEGER1] R. C. Jaeger, Introduction to Microelectronic Fabrication, Addison-Wesley Publishing Company, Massachusetts, 1988, pp. 101.

[MA2] Ibid T. P. Ma and P. V. Dressendorfer, Chapter 7.

[SROUR6] Ibid Srour, pp. 63.

[BRITTON1] C. L. Brittion, Studies on High-Speed Analog Memories for Physics Experiments with Emphasis on Radiation Hardening, PhD. Dissertaion for the University of Tennesse, Knoxville.

Chapter 5

[SROUR1] J. R. Srour, et al., Radiation Effects On and Dose Enhancement of Electronic Materials, Noyes Publications, Park Ridge, NJ, 1984, pp. 91

[GEIGER1] R. L. Geiger, P. E. Allen, N. R. Strader, VLSI Design Techniques For Analog and Digital Circuits, McGraw -Hill, Inc., New York, 1990, pp. 22-23

Chapter 6

[CUNITZ1] H. Cuntiz, et al., "A High-Rate, Preceision Calorimetry Readout", Proc. of the International Conference on Electronics for Future Colliders, May 11-12, 1994, pp. 19-30.

[BRITTON1] C. L. Britton, C. L. Britton, Jr., A. L. Wintenberg, G. R. Young, T. C. Awes, M. Womac, E. J. Kennedy, R. S. Smith, "Post-Radiation Memory Correction Using Differential Subtraction for Phenix", IEEE Transactions on Nuclear Science (to be published).

APPENDICES

APPENDIX A

FIGURES

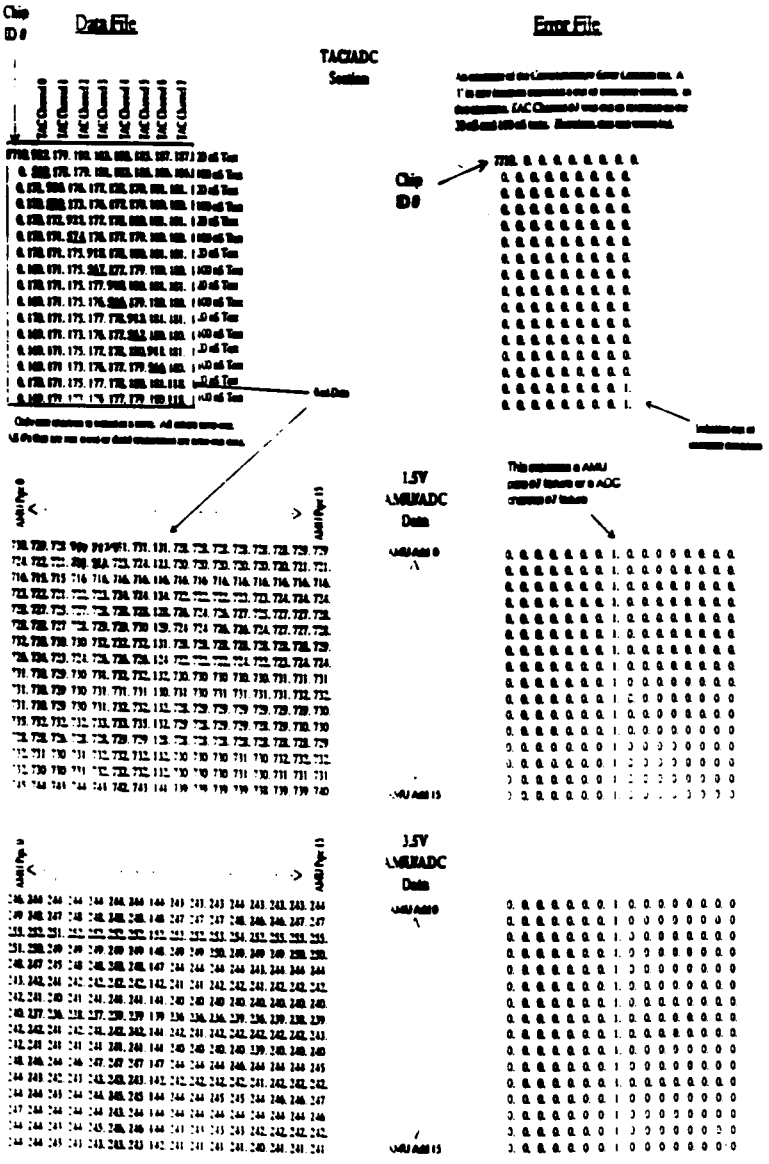


Figure A - 1. File structures for data and error files created from the 'chip2.exe program.

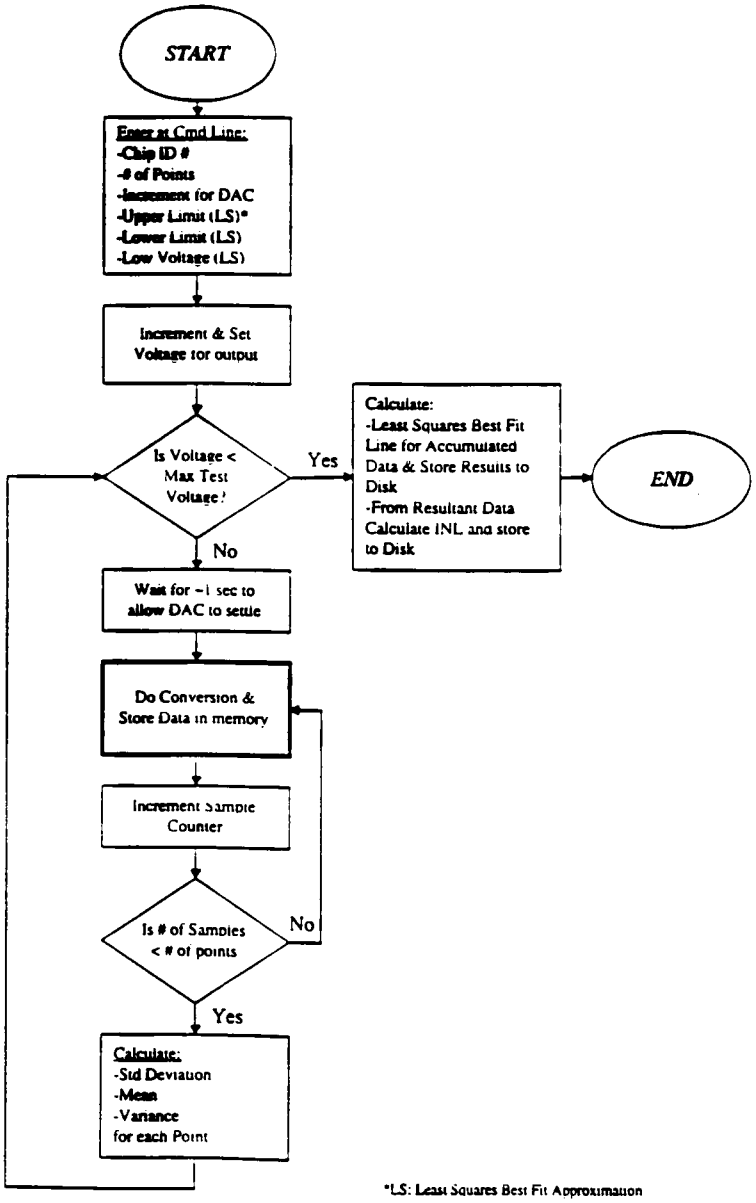


Figure A - 2. Flow Diagram for 'Lin3.exe'.

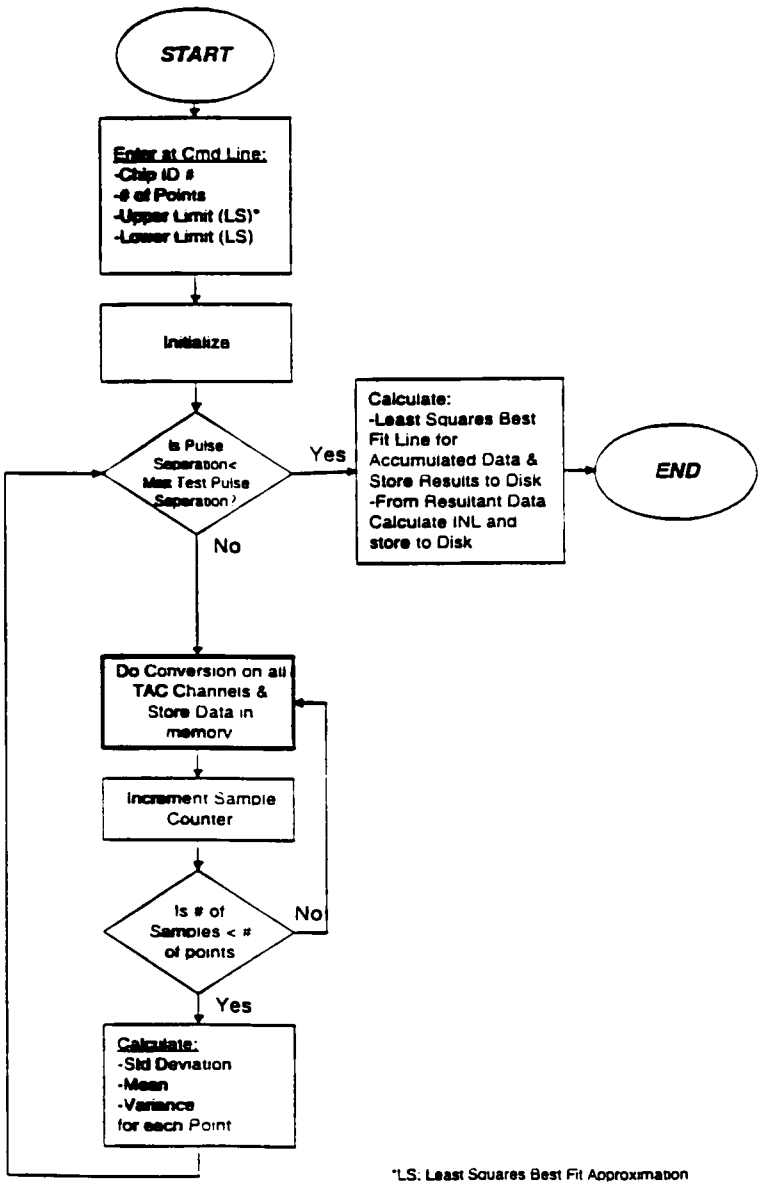


Figure A - 3. Flow Diagram for 'inl.exe' for the TAC/ADC.

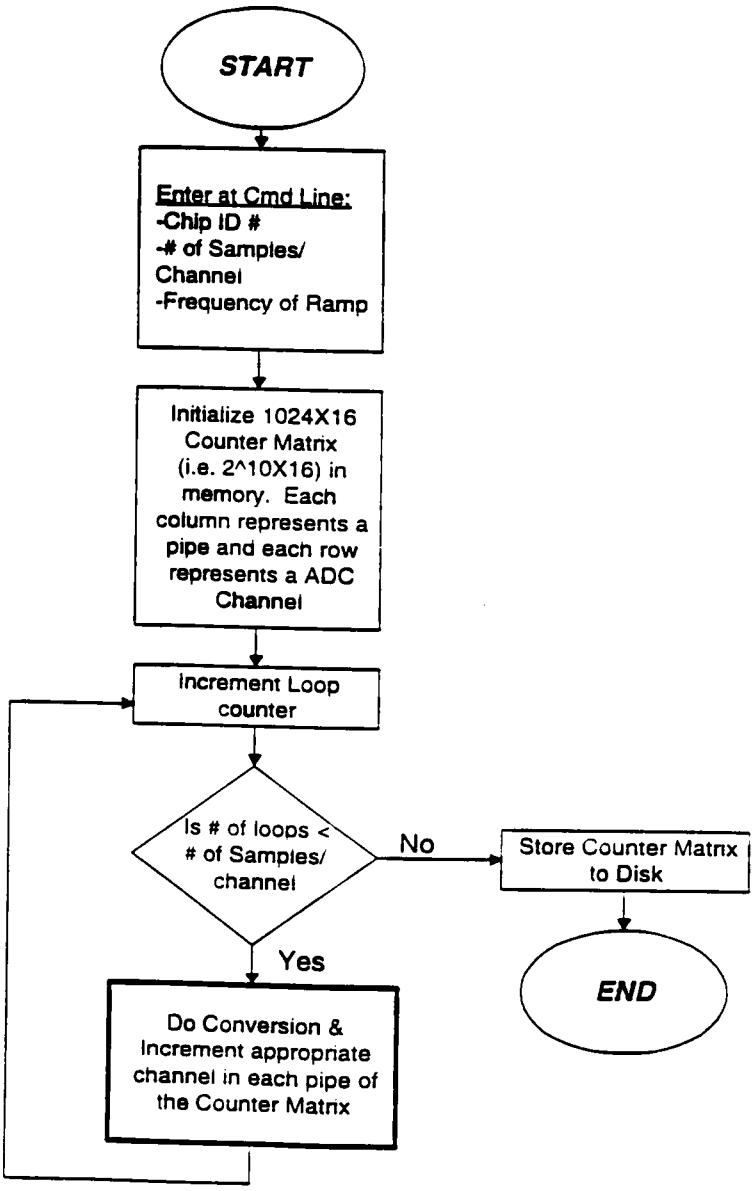


Figure A - 4. Flow Diagram for 'Sigdnl.exe'.

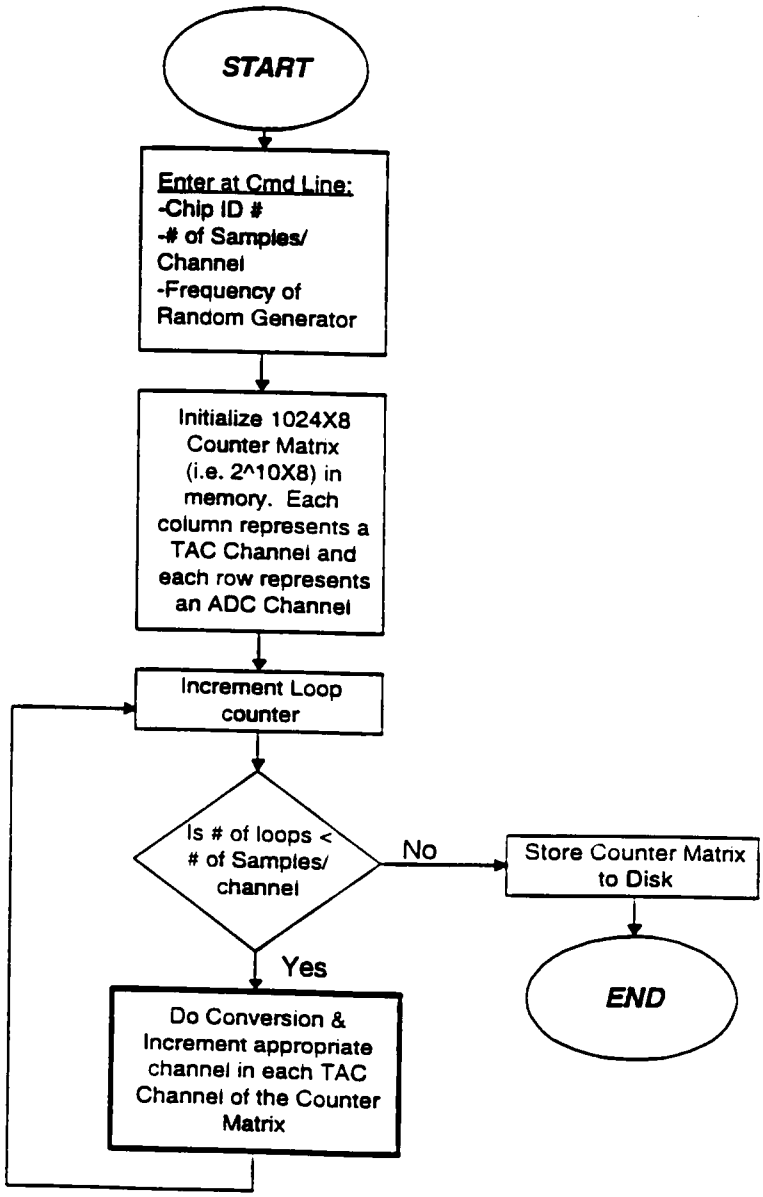


Figure A - 5. Flow Diagram for tdnl.exe'.

APPENDIX B

```
PROJ    =CHIP2
DEBUG   =1
CC      =qcl
CFLAGS_G        = /AS /Ml /Ze
CFLAGS_D        = /Zi /Zr /Gi$(PROJ).mdt /Od
CFLAGS_R        = /O /Ot /DNODEBUG
CFLAGS  =$(CFLAGS_G) $(CFLAGS_D)
LFLAGS_G        = /CP:0xfff /SE:0x80 /ST:0x800
LFLAGS_D        = /CO
LFLAGS_R        =
LFLAGS  =$(LFLAGS_G) $(LFLAGS_D)
RUNFLAGS        =
OBJS_EXT =
LIBS_EXT =      scrlib.lib

.asm.obj: ; $(AS) $(AFLAGS) -c $*.asm

all:    $(PROJ).EXE

maintest.obj:   maintest.c $(H)

tacadjst.obj:   tacadjst.c $(H)

ramp.obj:       ramp.c $(H)

adjust.obj:     adjust.c $(H)

read_adc.obj:   read_adc.c $(H)

tacselec.obj:   tacselec.c $(H)

chiptest.obj:   chiptest.c $(H)

testchip.obj:   testchip.c $(H)

retest.obj:     retest.c $(H)

file_ops.obj:   file_ops.c $(H)

amu2.obj:       amu2.c $(H)

chip2.obj:      chip2.c $(H)

$(PROJ).EXE:    maintest.obj tacadjst.obj ramp.obj adjust.obj read_adc.obj tacsel
ec.obj \
        chiptest.obj testchip.obj retest.obj file_ops.obj amu2.obj chip2.obj $(OB
JS_EXT)
        echo >NUL @<<$(PROJ).crf
maintest.obj +
tacadjst.obj +
ramp.obj +
adjust.obj +
read_adc.obj +
tacselec.obj +
chiptest.obj +
testchip.obj +
retest.obj +
file_ops.obj +
amu2.obj +
chip2.obj +
$(OBJS_EXT)
$(PROJ).EXE

$(LIBS_EXT);
<<
        ilink -a -e "qlink $(LFLAGS) @$(PROJ).crf" $(PROJ)
```



```
run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)
```

```
int J1_data, J2_data[45][25], J3_data, store_bad_channel[45][25];
```

16:

10/10/1994 16:09 Filename: CONSTANT.H Page 1

```
                /******** Constants ********/

#define upper_limit     0x03ff
#define tac             0
#define amu             1
#define amu_voltage_1   0x2000 /* sent to connector J1 */
#define amu_voltage_3   0xdfff /* sent to connector J1 */
#define dataclr         0x0000
#define data_mask       0x03ff
#define dont_care       0xffff

#define io_base_address 0x02a0
#define J1lower_byte    0x0000
#define J1upper_byte    0x0001
#define J2lower_byte    0x0000
#define J2upper_byte    0x0001
#define J3lower_byte    0x0002
#define J3upper_byte    0x0003

/* mach code masks */

#define reset           0x8000 /* sent to connector J1 */
#define read            0x0800 /* sent to connector J1 */
#define write           0x0400 /* sent to connector J1 */
#define adc_ramp        0x0200 /* sent to connector J1 */
#define adc_trig        0x0200 /* sent to connector J1 */
#define clr_adc_trig    0xfdff /* sent to connector J1 */
#define adc_oe          0x1000 /* sent to connector J1 */
#define tac_strt        0x8000 /* sent to connector J3 */


#define pulse_clr_mask  0xfe1f /* sent to connector J1 */

#define pulse_20ns      0x0000 /* sent to connector J1 */
#define pulse_40ns      0x0020 /* sent to connector J1 */
#define pulse_60ns      0x0040 /* sent to connector J1 */
#define pulse_80ns      0x0060 /* sent to connector J1 */
#define pulse_100ns     0x0080 /* sent to connector J1 */
#define pulse_120ns     0x00a0 /* sent to connector J1 */
#define pulse_140ns     0x00c0 /* sent to connector J1 */
#define pulse_160ns     0x00e0 /* sent to connector J1 */
#define pulse_180ns     0x0100 /* sent to connector J1 */
#define pulse_200ns     0x0120 /* sent to connector J1 */

#define chan0           0x0080 /* sent to connector J3 */
#define chan1           0x0100 /* sent to connector J3 */
#define chan2           0x0200 /* sent to connector J3 */
#define chan3           0x0400 /* sent to connector J3 */
#define chan4           0x0800 /* sent to connector J3 */
#define chan5           0x1000 /* sent to connector J3 */
#define chan6           0x2000 /* sent to connector J3 */
#define chan7           0x4000 /* sent to connector J3 */

#define test0           0x0000 /* sent to connector J3 */
#define test1           0x0010 /* sent to connector J3 */
#define test2           0x0020 /* sent to connector J3 */
#define test3           0x0030 /* sent to connector J3 */
#define test4           0x0040 /* sent to connector J3 */
#define test5           0x0050 /* sent to connector J3 */
#define test6           0x0060 /* sent to connector J3 */
#define test7           0x0070 /* sent to connector J3 */
```

```
extern void cls();
extern void locate(int,int,int);
```

```
/*
   MAINTEST.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           29 JUN 94
*/

/*****************************************************************************/
/********                        Headers                            ********/
/*****************************************************************************/

#include <conio.h>
#include "constants.h"
#include <stdlib.h>
#include <graph.h>
#include <stdio.h>
#include "variables.h"

/*****************************************************************************/
/********                       Variables                           ********/
/*****************************************************************************/

extern int x, sequence_mode;
extern int terror_bit, test_again, drive_nam[20];
extern long chip_id;

/*****************************************************************************/
/********                       Functions                           ********/
/*****************************************************************************/

void chiptest( int );


/*****************************************************************************/
/********                        Code                               ********/
/*****************************************************************************/




/*****************************************************************************/

/*
   Function Name: chiptest
   Requires: int type, chip # to be tested
   Returns:  void
Purpose:  to setup test parameters and initiate testing, save results, and
          rerun test
*/

/*****************************************************************************/


void chiptest( int chip_id ) /* later to rcv & pass the chip id data */
{
        int tac_test_select_exit;
        int r, c;

/*****************************************************************************/
/*                      Setup Parameters                                     */
/*****************************************************************************/

        J2_data[0][0] = chip_id;


        cls();
        Tac_timeout_generator();
```



```
        cls();
        input_tolerance();

        cls();
        tac_test_select_exit = tac_test_select();
        if( tac_test_select_exit == 1 )
        {
                return;
        }

        cls();
        test_display( J3_data );
        pulse_display( J1_data );
        tac_channel_select();

        cls();
        test_display( J3_data );
        pulse_display( J1_data );
        channel_display( J3_data );

/*****************************************************************************/
/*                      Testing of the chip                                  */
/*****************************************************************************/


        test_chip();


/*****************************************************************************/
/*****************************************************************************/


        cls();

        store_bad_channel[0][0] = chip_id;

        for( r = 0 ; r < 1; r++ )
        {
                for( c = 1; c < 16; c++ )
                {
                        store_bad_channel[r][c] = 0;
                }
        }


        save_data( chip_id );/* stores data to file on selected disk and file */

        retest();             /* Decides whether or not another test is rerun */

/*****************************************************************************/
/*                      Retesting Option                                     */
/*****************************************************************************/

        while( test_again ) /* test_again flag is set by retest() rerunning test*
/
        {
                run_test_again(); /* Sets up conditions and runs another test */
                retest();         /* Decides whether or not another test is rerun
 */
        }
}
```

```
/*
  TACADJST.C  p/o chip.c which is the main file of the program
  Written by:   Michael Womac
*/


/*****************************************************************************/
/********                        Headers                            ********/
/*****************************************************************************/

#include <stdio.h>
#include "scrlib.h"
#include <conio.h>
#include <graph.h>
#include <process.h>
#include "constant.h"
#include "variables.h"

/*****************************************************************************/
/********                        Variables                          ********/
/*****************************************************************************/

extern int a, sequence_mode;
int screen_update;
int adc_address;
int fsc;
int fac_timeout_count;
int where;

/*****************************************************************************/
/********                        Functions                          ********/
/*****************************************************************************/

void tac_adjust( int );
void Pulse_select( double *pulse, double *testnum );
void fac_timeout_generator( void );
void retac_adj( void );

/*****************************************************************************/
/********                        Code                               ********/
/*****************************************************************************/



/*****************************************************************************/

/*
   Function Name: tacadjust
   Requires:   void
   Returns:    void
   Purpose:    selects channels 0 & 4 to receive 200 ns pulse sequence
               repetively and display the data read while adjustment is being
               done.
*/

/*****************************************************************************/


void tac_adjust( int where )
{
        /*
                where = 1 => called from retest
                where = 0 => called from adjust
        */
```



```
        double k;
        double Pulse, Testnum;
        int pn;
        char buffer[80];

        Pulse_select( &Pulse, &Testnum);

        if( Pulse == pulse_20ns )
        {
                pn = 20;
        }
        if( Pulse == pulse_100ns )
        {
                pn = 100;
        }
        if( Pulse == pulse_200ns )
        {
                pn = 200;
        }
        if( Pulse == 999 )
        {
                return;
        }


        cls();
                /*locate(2,25,0);
        printf("TAC Adjustment Generator");*/

        _settextposition(3,27);
        _settextcolor( 3 );
        sprintf( buffer, "TAC Adjustment Generator");
        _outtext( buffer );

        J1_data = Pulse;
                locate(5,20,0);
        printf("Test 0");
                locate(6,20,0);
        printf("Channel 1");
                locate(7,20,0);
        printf("%dnS Pulse Separation", pn);

        if( Pulse == pulse_20ns )
        {
                pn = 20;

                _settextposition(12,15);
                _settextcolor( 14 );
                sprintf( buffer, "Adjust Pot #2 until Decimal value is - 700 to -
800");
                _outtext( buffer );

                _settextposition(13,25);
                _settextcolor( 14 );
                sprintf( buffer, "for 20 nS pulse seperation");
                _outtext( buffer );

        }
        if( Pulse == pulse_100ns )
        {
                pn = 100;

                _settextposition(12,15);
                _settextcolor( 14 );
                sprintf( buffer, "Adjust Pot #2 until Decimal value is - 400 to -
500");
```

```
                _outtext( buffer );

                _settextposition(13,25);
                _settextcolor( 14 );
                sprintf( buffer, "for 100 nS pulse seperation");
                _outtext( buffer );

        }
        if( Pulse == pulse_200ns )
        {
                pn = 200;

                _settextposition(12,15);
                _settextcolor( 14 );

                sprintf( buffer, "Adjust Pot #2 until Decimal value is - 50 to -2
50");
                _outtext( buffer );

                _settextposition(13,25);
                _settextcolor( 14 );
                sprintf( buffer, "for 200 nS pulse seperation");
                _outtext( buffer );

        }
        if( Pulse == 999 )
        {
                return;
        }


                locate(16,40,0);
        printf("Decimal   Hex");
        /*
           These next 2 output cmds make active the proper control bits that
           are needed prior to the start and trigger cmds to start the
           conversions and reading of the data
        */
        outpw( io_base_address + J1lower_byte, reset ); /* reset Mach part */
        outpw( io_base_address + J3lower_byte, J3_data );
        outpw( io_base_address + J1lower_byte, J1_data );
        J3_data = Testnum + 0x0080 + tac_strt;
                locate(7,1,0);
         displaycursor(0);
        while( !kbhit() )
        {
                        /*
                           This output cmd outputs the tac_strt cmd along
                           with the data test0 & chan4 that is already
                           active
                        */
                        outpw( io_base_address + J1lower_byte, J1_data );
                        outpw( io_base_address + J3lower_byte, J3_data );
                        outpw( io_base_address + J3lower_byte, dataclr );
                        /*
                           This output cmd outputs the acd_oe, adc_trig, and
                           the J1 data (pulse_200ns) that is already active.
                           Also, the adc_trig is 'and'ed with the clr_adc_
                           trig signal in order to clear it out.
                           The FSC input is monitored to check for the ADC
                           being done.  If so, read out data if not, wait.
                        */
                        outpw( io_base_address + J1lower_byte,
                                        J1_data + adc_oe + adc_trig );
                        outpw( io_base_address + J1lower_byte,
                                        ( J1_data + adc_oe + adc_trig )
```



```
                                        & clr_adc_trig );
                        /*
                          This flow control checks for full scale conversion
                          to be low.  If it is high, it will continue to
                          check it.  If it is low, it will continue in the
                          program
                        */
                        do
                        {
                                fsc = (inpw(io_base_address + J2lower_byte)
                                        & 0x8000);
                        } while( fsc );
                        sequence_mode = 0;
                        read_adc( 0, dont_care, 1, dont_care, dont_care,
                                dont_care );
                        locate(18,20,0);  /* previously was locate(18,20,0); */
                        printf( " ADC Channel IAC #1 = %d     %x ",
                                        J2_data[1][ 1 ],
                                  J2_data[1][ 1 ] );


                                _settextposition(24,26);
                                _settextcolor( 23 );
                                sprintf( buffer, "Press any key when ready" );
                                _outtext( buffer );



/*              locate(13,20,0);
                        printf( " ADC Channel IAC #2 = %d     %x ",
                                        J2_data[2][ 1 ],
                                        J2_data[2][ 1 ] );
                        locate(14,20,0);
                        printf( " ADC Channel IAC #3 = %d     %x ",
                                        J2_data[3][ 1 ],
                                        J2_data[3][ 1 ] );
                        locate(15,20,0);
                        printf( " ADC Channel IAC #4 = %d     %x ",
                                        J2_data[4][ 1 ],
                                        J2_data[4][ 1 ] );
                        locate(16,20,0);
                        printf( " ADC Channel IAC #5 = %d     %x ",
                                        J2_data[5][ 1 ],
                                        J2_data[5][ 1 ] );
                        locate(17,20,0);
                        printf( " ADC Channel IAC #6 = %d     %x ",
                                        J2_data[6][ 1 ],
                                        J2_data[6][ 1 ] );
                        locate(18,20,0);
                        printf( " ADC Channel IAC #7 = %d     %x ",
                                        J2_data[7][ 1 ],
                                        J2_data[7][ 1 ] );
                        locate(19,20,0);
                        printf( " ADC Channel IAC #8 = %d     %x ",
                                        J2_data[8][ 1 ],
                                        J2_data[8][ 1 ] );
*/
        for( k = 1; k < 1000000; k++ );

        }
        outpw( io_base_address + J3lower_byte, dataclr );
        _displaycursor(1);

        if( where == 0 )
        {
                adjust();
```

```
        )
)

/*************************************************************************/

/*
        Function Name: Pulse_select
        Requires:   two pointers to the variables to send the constant values
        Returns:    void
        Purpose:    select the desired pulse sequence to send to the IAC
*/

/*************************************************************************/


void Pulse_select( double *pulse, double *testnum)
{

        char sel, buffer[80];

        cls();
                /*locate(10,25,0);
        printf("Select Delay to be sent to the IAC");*/

          _settextposition(11,16);
          _settextcolor( 11 );
          sprintf( buffer, "Select Delay to be sent to the IAC" );
          _outtext( buffer );

                locate(12,25,0);
        printf("    20 nS");
                locate(13,25,0);
        printf("    100 nS    - use for IAC Bias adjustment");
                locate(14,25,0);
        printf("    200 nS");
                locate(15,25,0);
        printf("    Exit");


         _settextposition(13,26);
         _settextcolor( 15 );
        sprintf( buffer, "1." );
        _outtext( buffer );

        _settextposition(14,26);
        _settextcolor( 15 );
        sprintf( buffer, "2." );
        _outtext( buffer );

        _settextposition(15,26);
        _settextcolor( 15 );
        sprintf( buffer, "3." );
        _outtext( buffer );

        _settextposition(16,26);
        _settextcolor( 15 );
        sprintf( buffer, "4." );
         outtext( buffer );
        fflush(stdin);
                locate(18,25,0);
        printf("Selection: ");
        sel = getch();

        if(sel == '1')
        {
                *pulse = pulse_20ns;
```



```
                *testnum = test1;
        }
        if(sel == '2')
        {
                *pulse = pulse_100ns;
                *testnum = test1;
        }
        if(sel == '3')
        {
                *pulse = pulse_200ns;
                *testnum = test0;
        }
        if(sel == '4')
        {
                *pulse = 999.
        }


}

void Iac_timeout_generator( void )
{
        int timeout_sum;
        int i;
        int timeout_avg;
        int sequence_mode_tmp;
        char buffer[80];


        sequence_mode_tmp = sequence_mode;


        cls();
                /*locate(10,25,0);
        printf("IAC Timeout Generator");*/


        _settextposition(11,25);
        _settextcolor( 13 );
        sprintf( buffer, "IAC Adjustment Generator");
        _outtext( buffer );


        J1_data = 0;
        /*
                These next 2 output cmds make active the proper control bits that

                are needed prior to the start and trigger cmds to start the
                conversions and reading of the data
        */
        outpw( io_base_address + J1lower_byte, reset ); /* reset Mach part */
        outpw( io_base_address + J3lower_byte, J3_data );
        outpw( io_base_address + J1lower_byte, J1_data );
        J3_data = iac_strt;
                locate(7,1,0);
        _displaycursor(0);
                        /*
                                This output cmd outputs the iac_strt and along
                                with the data that is already active
                        */
                        outpw( io_base_address + J1lower_byte, J1_data );
                        outpw( io_base_address + J3lower_byte, J3_data );
                        outpw( io_base_address + J3lower_byte, datasir );
                        /*
```

```
                        this output cmd outputs the acd_oe, adc_trig, and

                        the J1 data that is already active.
                        Also, the adc_trig is 'and'ed with the clr_adc_
                        trig signal in order to clear it out.
                        The fSC input is monitored to check for the ADC
                        being done.  If so, read out data if not, wait.
                */
                outpw( io_base_address + J1lower_byte,
                                J1_data + adc_oe + adc_trig );
                outpw( io_base_address + J1lower_byte,
                                ( J1_data + adc_oe + adc_trig )
                                & clr_adc_trig );
                /*
                  This flow control checks for full scale conversion
                  to be low.  If it is high, it will continue to
                  check it.  If it is low, it will continue in the
                  program
                */
                do
                {
                        fsc = (inpw(io_base_address + J2lower_byte)
                                & 0x8000);
                } while( fsc );
                sequence_mode = 0;
                read_adc( 0, dont_care, 1, dont_care, dont_care,
                                dont_care );
                locate(12,20,0);
                printf( " ADC Channel TAC #1 = %d     %x ",
                                J2_data[1][ 1 ],
                                J2_data[1][ 1 ] );
                locate(13,20,0);
                printf( " ADC Channel TAC #2 = %d     %x ",
                                J2_data[2][ 1 ],
                                J2_data[2][ 1 ] );
                locate(14,20,0);
                printf( " ADC Channel TAC #3 = %d     %x ",
                                J2_data[3][ 1 ],
                                J2_data[3][ 1 ] );
                locate(15,20,0);
                printf( " ADC Channel TAC #4 = %d     %x ",
                                J2_data[4][ 1 ],
                                J2_data[4][ 1 ] );
                locate(16,20,0);
                printf( " ADC Channel TAC #5 = %d     %x ",
                                J2_data[5][ 1 ],
                                J2_data[5][ 1 ] );
                locate(17,20,0);
                printf( " ADC Channel TAC #6 = %d     %x ",
                                J2_data[6][ 1 ],
                                J2_data[6][ 1 ] );
                locate(18,20,0);
                printf( " ADC Channel TAC #7 = %d     %x ",
                                J2_data[7][ 1 ],
                                J2_data[7][ 1 ] );
                locate(19,20,0);
                printf( " ADC Channel TAC #8 = %d     %x ",
                                J2_data[8][ 1 ],
                                J2_data[8][ 1 ] );


        timeout_sum = 0;
        timeout_avg = 0;

        for( i = 1; i < 9; i++ )
        {
                timeout_sum += J2_data[i][1];
```



```
        }

        timeout_avg = timeout_sum / 8;

                locate(22,25,0);
        printf("                        %d", timeout_avg );

        _settextposition(23,25);
        _settextcolor( 15 );
        sprintf( buffer, "TAC Timeout Average = " );
        _outtext( buffer );


        tac_timeout_count = timeout_avg;


        _settextposition(25,25);
        _settextcolor( 23 );
        sprintf( buffer, "Press any key when ready" );
        _outtext( buffer );

        getch();

        outpw( io_base_address + J3lower_byte, dataclr );
        _displaycursor(1);

        sequence_mode = sequence_mode_tmp;

}


void retac_adj( void )
{

        double k;
        int ppn;
        char buffer[80];
        int J3_data_tmp;
        int J1_data_tmp;
        int sequence_mode_tmp;

        ppn = 100;

        sequence_mode_tmp = sequence_mode;

        J1_data_tmp = J1_data;
        J3_data_tmp = J3_data;

        cls();

        _settextposition(3,27);
        _settextcolor( 3 );
        sprintf( buffer, "TAC Adjustment Generator");
        _outtext( buffer );

        J1_data_tmp = pulse_100ns;
                locate(5,20,0);
        printf("Test 0");
                locate(6,20,0);
        printf("Channel 1");
                locate(7,20,0);
        printf("100nS Pulse Separation");
```

```
                _settextposition(12,15);
                _settextcolor( 14 );
                sprintf( buffer, "Adjust Pot #2 until Decimal value is - 400 to -
500");
                _outtext( buffer );

                _settextposition(13,25);
                _settextcolor( 14 );
                sprintf( buffer, "for 100 nS pulse seperation");
                _outtext( buffer );



                locate(16,40,0);
        printf("Decimal   Hex");
        /*
                These next 2 output cmds make active the proper control bits that

                are needed prior to the start and trigger cmds to start the
                conversions and reading of the data
        */
        outpw( io_base_address + J1lower_byte, reset ); /* reset Mach part */
        outpw( io_base_address + J3lower_byte, J3_data_tmp );
        outpw( io_base_address + J1lower_byte, J1_data_tmp );
        J3_data_tmp = test1 + 0x0080 + tac_strt;
                locate(7,1,0);
         displaycursor(0);
        while( !kbhit() )
        {
                        /*
                                This output cmd outputs the tac_strt cmd along
                                with the data test0 & chan4 that is already
                                active
                        */
                        outpw( io_base_address + J1lower_byte, J1_data_tmp );
                        outpw( io_base_address + J3lower_byte, J3_data_tmp );
                        outpw( io_base_address + J3lower_byte, dataclr );
                        /*
                                This output cmd outputs the acd_oe, adc_trig, and

                                the J1 data (pulse_200ns) that is already active.

                                Also, the adc_trig is 'and'ed with the clr_adc_
                                trig signal in order to clear it out.
                                The FSC input is monitored to check for the ADC
                                being done.  If so, read out data if not, wait.
                        */
                        outpw( io_base_address + J1lower_byte,
                                        J1_data_tmp + adc_oe + adc_trig );
                        outpw( io_base_address + J1lower_byte,
                                        ( J1_data_tmp + adc_oe + adc_trig )
                                        & clr_adc_trig );
                        /*
                          This flow control checks for full scale conversion
                          to be low.  If it is high, it will continue to
                          check it.  If it is low, it will continue in the
                          program
                        */
                        do
                        {
                                fsc = (inpw(io_base_address + J2lower_byte)
                                        & 0x8000);
                        } while( fsc );
                        sequence_mode = 0;
                        read_adc( 0, dont_care, 1, dont_care, dont_care,
                                dont_care );
```



```
                        locate(18,20,0);  /* previously was locate(12,20,0); */
                        printf( " ADC Channel JAC #1 = %d     %x ",
                                        J2_data[1][ 1 ],
                                        J2_data[1][ 1 ]);


                                _settextposition(24,26);
                                _settextcolor( 23 );
                                sprintf( buffer, "Press any key when ready" );
                                _outtext( buffer );


        for( k = 1; k < 500000; k++ );

        }


        outpw( io_base_address + J3lower_byte, dataclr );

        _displaycursor(1);

        sequence_mode = sequence_mode_tmp;

}
```

```
/*
  Ramp.c  p/o chip.c which is the main file of the program
  Written by:   Michael Womac
*/

/*****************************************************************************/
/********                        Headers                            ********/
/*****************************************************************************/

#include <stdio.h>
#include "scrlib.h"
#include <conio.h>
#include <graph.h>
#include <process.h>
#include "constants.h"
#include "variables.h"

/*****************************************************************************/
/********                        Variables                          ********/
/*****************************************************************************/

extern int x;

/*****************************************************************************/
/********                        Functions                          ********/
/*****************************************************************************/

void ramp( void );


/*****************************************************************************/
/********                        Code                               ********/
/*****************************************************************************/



/*****************************************************************************/

/*
   Function Name: ramp
   Requires:  void
   Returns:   void
   Purpose:   to send necessary commands cause a continuous ramp from the adc
*/

/*****************************************************************************/


void ramp( void )
{
        int io_data;
        char buffer[80];
        cls();
                locate(5,29,0);
        printf("Ramp Generator");
                locate(9,16,0);
        printf("Place or make sure the O'scope probe is on TP1");
        J1_data = adc_ramp;
                /*locate(1,1,0);
        printf("J1 data = %x", J1_data);*/
        outpw( io_base_address + J1lower_byte, reset ); /* reset Mach part */

        _settextposition(16,26);
        _settextcolor( 23 );
```



```
        sprintf( buffer, "Press any key when ready" );
        _outtext( buffer );

        while( !kbhit() )
        {
                io_data = outpw( io_base_address + J1lower_byte, J1_data );
                io_data = outpw( io_base_address + J1lower_byte, defaultr );
        }

}
```

```
/*
  Adjust.c  p/o chip.c which is the main file of the program
  Written by:   Michael Womac
/*

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdio.h>
#include "scrlib.h"
#include <conio.h>
#include <graph.h>
#include <process.h>
#include "constants.h"
#include "variables.h"

/*************************************************************************/
/********                      Variables                        ********/
/*************************************************************************/

char choice, routine, ch;

/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/

void adjust( void );
void amu_voltage_adj( void );
void select_adjust_method( void );

/*************************************************************************/
/********                      CODE                             ********/
/*************************************************************************/


/************************************************************************/

/*
   Function Name: adjust
   Requires: void
   Returns:  void
   Purpose:  menu selecting adjustment routine
*/

/************************************************************************/


void adjust( void )
{

        char buffer[80];

        cls();
        _setbkcolor(8); /*black*/
        _settextposition(8,23);
        _settextcolor( 5 ); /*magenta*/
        sprintf( buffer, "Entering Adjustment Routines" );
        _outtext( buffer );
                locate(8,25,0);
        printf("  ");
                locate(10,20,0);
        printf("    ADC Ramp Adjustment Routine ");
                locate(12,20,0);
        printf("    TAC Adjustment Routine ");
```



```
                locate(14,20,0);
        printf("    AMU Reference Voltage Adjustment Routine ");
                locate(16,20,0);
        printf("    Return to Main Menu ");

        _settextposition(11,22);
        _settextcolor( 15 );
        sprintf( buffer, "1." );
        _outtext( buffer );

        _settextposition(13,22);
        _settextcolor( 15 );
        sprintf( buffer, "2." );
        _outtext( buffer );

        _settextposition(15,22);
        _settextcolor( 15 );
        sprintf( buffer, "3." );
        _outtext( buffer );

        _settextposition(17,22);
        _settextcolor( 15 );
        sprintf( buffer, "4." );
        _outtext( buffer );



                locate(18,20,0);
        printf(" Selection: ");
        /* scanf("%d", &ch); */
        fflush(stdin);
        ch = getch();
        printf("%d", ch);
        switch(ch)
        {
                case '1':
                        cls();
                        select_adjust_method();
                        adjust();
                        break;
                case '2':
                        cls();
                        tac_adjust(0);
                        adjust();
                        break;
                case '3':
                        cls();
                        amu_voltage_adj();
                        adjust();
                        break;
                case '4':
                        cls();
                        return;
                        break;
                default:
                        adjust();
                        break;
        }
        return;
}

void amu_voltage_adj( void )
{
        char level;
        char buffer[80];
```

```
        int amu_3;
        int the_end;
        the_end = 0;
        while( !the_end )
        {
                cls();
                        locate(10,25,0);
                printf("    Adjust High Level ");
                        locate(12,25,0);
                printf("    Adjust Low Level ");
                        locate(14,25,0);
                printf("    Exit ");


                _settextposition(9,18);
                _settextcolor( 2 );
                sprintf( buffer, "AMU Reference Voltage Adjustment Select" );
                _outtext( buffer );


                _settextposition(11,26);
                _settextcolor( 15 );
                sprintf( buffer, "1." );
                _outtext( buffer );

                _settextposition(13,26);
                _settextcolor( 15 );
                sprintf( buffer, "2." );
                _outtext( buffer );

                _settextposition(15,26);
                _settextcolor( 15 );
                sprintf( buffer, "3." );
                _outtext( buffer );

                        locate(16,25,0);
                printf("Selection: ");
                fflush(stdin);
                level = getch();
                printf("%d", level);
                switch(level)
                {
                        case '1':
                                cls();

                                amu_3 = ((0xdfff)&(inpw( io_base_address + J1lowe
r byte )));


                                _settextposition(12,19);
                                _settextcolor( 12 );
                                sprintf( buffer, "Set up Multimeter for voltage m
easurement" );
                                _outtext( buffer );

                                _settextposition(13,25);
                                _settextcolor( 12 );
                                sprintf( buffer, "on J1 & GND of DAQ Test board"
);
                                _outtext( buffer );

                                _settextposition(16,24);
                                _settextcolor( 10 );
                                sprintf( buffer, "Now ready to adjust High level"
 );
                                _outtext( buffer );
```



```
                                outpw( io_base_address + J1lower_byte, amu_3 );
                                        /*locate(16,25,0);
                                printf("Press any key when ready");*/

                                _settextposition(20,26);
                                _settextcolor( 23 );
                                sprintf( buffer, "Press any key when ready" );
                                _outtext( buffer );

                                fflush(stdin);
                                getch();

                                break;

                        case '2':
                                cls();

                                        /*locate(11,18,0);
                                printf("Set up multimeter for voltage measurement
");
                                        locate(13,23,0);
                                printf("Now ready to adjust Low Level");*/



                                _settextposition(12,19);
                                _settextcolor( 12 );
                                sprintf( buffer, "Set up Multimeter for voltage m
easurement" );
                                _outtext( buffer );

                                _settextposition(13,25);
                                _settextcolor( 12 );
                                sprintf( buffer, "on J1 & GND of DAQ Test board"
);
                                _outtext( buffer );

                                _settextposition(16,24);
                                _settextcolor( 10 );
                                sprintf( buffer, "Now ready to adjust Low level"
);
                                _outtext( buffer );


                                outpw( io_base_address + J1lower_byte, amu_voltag
e_1 );

                                        /*locate(16,25,0);
                                printf("Press any key when ready");*/

                                _settextposition(20,26);
                                _settextcolor( 23 );
                                sprintf( buffer, "Press any key when ready" );
                                _outtext( buffer );

                                fflush(stdin);
                                getch();

                                break;


                        case '3':
                                the_end = 1;
                                break;
```

```
                        default:
                                cls();
                                amu_voltage_adj();
                                break;
                }

        }


}


void
select_adjust_method( void )
{
        int theend;
        char buffer[80];
        char method;

        theend = 0;
        while( !theend )
        {
                cls();
                        locate(10,15,0);
                printf("    Adjust using continuous ramps on the O'scope");
                        locate(12,15,0);
                printf("    Adjust using 1.5V & 3.5V AMU inputs and adjusting ");

                        locate(13,15,0);
                printf("    ADC output values to within ideal conversion range.")
;
                        locate(15,15,0);
                printf("    Exit ");

                _settextposition(9,22);
                _settextcolor( 2 );
                sprintf( buffer, "Select Ramp Adjustment Method" );
                _outtext( buffer );


                _settextposition(11,16);
                _settextcolor( 15 );
                sprintf( buffer, "1." );
                _outtext( buffer );

                _settextposition(13,16);
                _settextcolor( 15 );
                sprintf( buffer, "2." );
                _outtext( buffer );

                _settextposition(16,16);
                _settextcolor( 15 );
                sprintf( buffer, "3." );
                _outtext( buffer );

                        locate(17,20,0);
                printf("Selection: ");
                fflush(stdin);
                method = getch();
                printf("%d", method);
                switch(method)
                {
                        case '1':
                                cls();
```



```
                                ramp();

                                break;


                        case '2':
                                cls();

                                amu2();

                                break;


                        case '3':
                                theend = 1;
                                break;

                        default:
                                cls();
                                select_adjust_method();
                                break;
                }

        }
        return ;
}
```

```
/*
  READ_ADC.C  p/o chip.c which is the main file of the program
  written by:   Michael Womac
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdio.h>
#include "scrlib.h"
#include <conio.h>
#include <graph.h>
#include <process.h>
#include "constant.h"
#include <stdlib.h>
#include <math.h>
#include "variables.h"

/*************************************************************************/
/********                      Variables                        ********/
/*************************************************************************/

extern int amu_read, all_odd_even;
extern int sequence_mode, num_of_pulses, terror_bit;
extern int tac_timeout_count;
extern long chip_id;


int tac_adc_limit[6];
int amu_adc_limit[2];
int adc_address, adc_limit, tolerance, amu_tolerance;
int select, amu_pipe, amu_or_tac;
int adc_begin_address, adc_end_address;
int tac_test_col, toler_ans, next_tac_pulse;
int amu_voltage, tac_loop, tac_test_num, tac_loop_num, tac_col;
int amu_offset, b20ns_offset, b100ns_offset, b200ns_offset;

/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/

void read_adc( int, int, int, int, int, int );
void test_limits( int, int, int );
void test_data( int, int, int );
void input_tolerance( void );

/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/



/*************************************************************************/

/*
   function Name: read_adc
   Requires:    select:  select = 0 => read tac adc's
                         select = 1 => read amu adc's

                amu_pipe:       is a variable used to determine which column
                                is data to be stored in.  It is a amu column #

                tac_pipe:       likewise, is column to store data to. For
                                sequence_mode = 0, all 8 tacs are addressed
```



```
                                for each column given.

                tac_loop_num:   is used when sequence_mode = 1.  This loop is
                                the all, odd, & even passes for the selected
                                test. all pulse sequences will be tested for
                                all channels, odd channels, and even channels.

                tac_col:        ?

                next_tac_pulse: used to determine which pulse is currently
                                being sent to the tac.

                all of these selection variables  determine the starting and
                ending column #'s for the data to be written into the J2 data
                matrix.
   Returns:
   Purpose:
*/

/*************************************************************************/


void read_adc( int select, int amu_pipe, int tac_pipe, int tac_loop_num,
               int tac_col, int next_tac_pulse )
{
        int amu_address, column_strt, column_stop, sequence_pipe;
        int J2_begin_address;

        /*********************************************************************/

        /*
           This section of flow control is used to determine whether or not
           the tac adc's are read or the amu adc's are read

           Single Mode => set of pulses only sent once
        */

        /*********************************************************************/

        if( select == 0 )  /* TAC ADC's */
        {
          if( sequence_mode == 0 )  /* sequence_mode is external variable */
          {
                adc_begin_address = 0;
                adc_end_address = 8;

                for( adc_address = adc_begin_address;
                adc_address < adc_end_address; adc_address += 1 )
                {

        /*********************************************************************/

        /*
                  Address data & ADC OE is output to the DAQ chip
                  here along with any necessary codes previously
                  defined
        */

        /*********************************************************************/

                outpw( io_base_address + J1lower_byte,
                       adc_address + adc_oe);

        /*********************************************************************/
```

```
/*
  Data is input here from the DATA I/O 8d & 16 bit data that is
  collected is 'and'ed with the input data with a mask which will
  allow only the adc_out data
*/

/*****************************************************************/

                        J2_data[ adc_address + 1 ][ tac_pipe ] =
                              (inpw( io_base_address + J2lower_byte )
                              & data_mask );

        }

      outpw(io_base_address + J1lower_byte, adc_address & 0xefff);

   }

/*****************************************************************/

/*
     Sequence Mode => set of pulses sent 3 times with
     all, odd, even channels selected one at a time.
*/

/*****************************************************************/

  if( sequence_mode == 1 )
  {

/*****************************************************************/
/*                      Tac test  #1                             */
/*****************************************************************/


     if( num_of_pulses == 1 && tac_loop_num == 1 )
     {
        column_strt = 0;      column_stop = 1;
     }
     if( num_of_pulses == 1 && tac_loop_num == 2 )
     {
        column_strt = 1;      column_stop = 2;
     }
     if( num_of_pulses == 1 && tac_loop_num == 3 )
     {
        column_strt = 2;      column_stop = 3;
     }
         if( num_of_pulses == 1 && tac_loop_num == 4 )
         {
       column_strt = 3;      column_stop = 4;
         }
         if( num_of_pulses == 1 && tac_loop_num == 5 )
         {
       column_strt = 4;      column_stop = 5;
         }
         if( num_of_pulses == 1 && tac_loop_num == 6 )
         {
       column_strt = 5;      column_stop = 6;
         }
         if( num_of_pulses == 1 && tac_loop_num == 7 )
         {
       column_strt = 6;      column_stop = 7;
         }
         if( num_of_pulses == 1 && tac_loop_num == 8 )
         {
       column_strt = 7;      column_stop = 8;
```



```
         }

/*****************************************************************/
/*                      Tac test  #2                             */
/*****************************************************************/

/*****************************************************************/
/*                      next pulse = 1                           */
/*                      Tac test  #2                             */
/*****************************************************************/

     if( num_of_pulses == 2 && tac_loop_num == 1 && tac_col ==0
        && next_tac_pulse == 1)
     {
        column_strt = 0;      column_stop = 1;
     }
     if( num_of_pulses == 2 && tac_loop_num == 2 && tac_col == 0
        && next_tac_pulse == 1)
     {
        column_strt = 2;      column_stop = 3;
     }
     if( num_of_pulses == 2 && tac_loop_num == 3 && tac_col == 0
        && next_tac_pulse == 1)
     {
        column_strt = 4;      column_stop = 5;
     }
       if( num_of_pulses == 2 && tac_loop_num == 4 && tac_col == 0
     && next_tac_pulse == 1)
       {
     column_strt = 6;      column_stop = 7;
       }
       if( num_of_pulses == 2 && tac_loop_num == 5 && tac_col == 0
     && next_tac_pulse == 1)
       {
     column_strt = 8;      column_stop = 9;
       }
       if( num_of_pulses == 2 && tac_loop_num == 6 && tac_col == 0
     && next_tac_pulse == 1)
       {
     column_strt = 10;      column_stop = 11;
       }
       if( num_of_pulses == 2 && tac_loop_num == 7 && tac_col == 0
     && next_tac_pulse == 1)
       {
     column_strt = 12;      column_stop = 13;
       }
       if( num_of_pulses == 2 && tac_loop_num == 8 && tac_col == 0
     && next_tac_pulse == 1)
       {
     column_strt = 14;      column_stop = 15;
       }


/*****************************************************************/
/*                      next pulse = 2                           */
/*                      Tac test  #2                             */
/*****************************************************************/

     if( num_of_pulses == 2 && tac_loop_num == 1 && tac_col == 1
        && next_tac_pulse == 2)
     {
        column_strt = 1;      column_stop = 2;
     }
     if( num_of_pulses == 2 && tac_loop_num == 2 && tac_col == 1
        && next_tac_pulse == 2)
     {
```

```
            column_strt = 3;      column_stop = 4;
          }
          if( num_of_pulses == 2 && tac_loop_num == 3 && tac_col == 1
            && next_tac_pulse == 2)
          {
            column_strt = 5;      column_stop = 6;
          }

              if( num_of_pulses == 2 && tac_loop_num == 4 && tac_col == 1
            && next_tac_pulse == 2)
              {
            column_strt = 7;      column_stop = 8;
              }
              if( num_of_pulses == 2 && tac_loop_num == 5 && tac_col == 1
            && next_tac_pulse == 2)
              {
            column_strt = 9;      column_stop = 10;
              }
              if( num_of_pulses == 2 && tac_loop_num == 6 && tac_col == 1
            && next_tac_pulse == 2)
              {
            column_strt = 11;      column_stop = 12;
              }
              if( num_of_pulses == 2 && tac_loop_num == 7 && tac_col == 1
            && next_tac_pulse == 2)
              {
            column_strt = 13;      column_stop = 14;
              }
              if( num_of_pulses == 2 && tac_loop_num == 8 && tac_col == 1
            && next_tac_pulse == 2)
              {
            column_strt = 15;      column_stop = 16;
              }


            adc_begin_address = 0;
            adc_end_address = 8;

            for( adc_address = adc_begin_address;
            adc_address < adc_end_address; adc_address += 1 )
            {

/*****************************************************************/

/*
  Address data & ADC OE is output to the DAQ chip
  here along with any necessary codes previously
  defined
*/

/*****************************************************************/

        outpw( io_base_address + J1lower_byte,
                        J1_data + adc_address + adc_oe);

                for( sequence_pipe = column_strt;
                     sequence_pipe < column_stop; sequence_pipe ++ )
                {
/*****************************************************************/

/*
  Data is input here from the DATA I/O Bd & 16 bit data that is
  collected is 'and'ed with the input data with a mask which will
  allow only the adc_out data
*/
```



```
/*****************************************************************/

                        J2_data[ adc_address + 1 ][ sequence_pipe ] =
                            (inpw( io_base_address + J2lower_byte )
                            & data_mask );
                    }
                }
            }
        }



/*****************************************************************/
/*                        AMU TESTING                            */
/*****************************************************************/


        if( select == 1 )              /* AMU ADC's */
        {

            if( (J1_data & 0x2000) == 0x2000 )
            {
                    adc_begin_address = 8;
                    adc_end_address = 25;
                    J2_begin_address = 8;
            }
            if( (J1_data & 0x2000) == 0 )
            {
                    adc_begin_address = 8;
                    adc_end_address = 25;
                    J2_begin_address = 25;
            }
                for( adc_address = adc_begin_address;
                        adc_address < adc_end_address; adc_address++ )
                {
/*****************************************************************/

/*
  Address data & ADC OE is output to the DAQ chip    e along
          with any necessary codes previously defined
*/

/*****************************************************************/

                    outpw( io_base_address + J1lower_byte,
                                    adc_address + adc_oe );

/*****************************************************************/

/*
  Data is input here from the DATA I/O Bd & 16 bit data that
  is collected is 'and'ed with the input data with a mask
  which will allow only the adc_out data
*/

/*****************************************************************/
                    if( J2_begin_address == 25 )
                    {
                            J2_data[ adc_address + 17 ][ amu_pipe  ] =
                                        (inpw( io_base_address + J2lower
_byte )&  data_mask );
                    }
                    if( J2_begin_address == 8 )
                    {
```

```
                                J2_data[ adc_address + 1 ][ amu_pipe ] =
                                                (inpw( io_base_address + J2lower
 byte )& data_mask );
                        )
                    )
            )
)



/*************************************************************************/

/*
   Function Name:
   Requires:
            tac_test_num:       test number that tac was tested under, which
                                determines which columns to store data in.

            amu_input_voltage:  selects the amu for data to be saved under
                                for either the 1.5 or 3 volt input.
   Returns:
   Purpose: selects the parameters used by test_limits functions which
            actually makes the comparason of the data against the appropriate
            error band.
*/

/*************************************************************************/


void test_data( int tac_test_num, int amu_input_voltage, int tac_loop )
{
        int i,j;
        amu_adc_limit[1] = 716; /*+ amu_offset;*/ /* 1.5V input */
        amu_adc_limit[2] = 307; /*+ amu_offset;*/ /* 3.5V input */

        if( tac_test_num < 30 && tac_test_num > 0 )
        {
                tac_test_num  = 15;
        }

        /*************************************************************/

        /*
            if more tests are added, this selection logic will have to be
            redone.
        */

        /*************************************************************/

        if( tac_test_num > 30 )
        {
                tac_test_num -= 30;
        }

/*        for(i = 0; i < 41; i ++ )
        {
                for(j = 0; j < 17; j ++ )
                {
                J2_data[i][j] = 0x03ff;
                }
        }
*/

        if( tac_test_num == 0 )
        {
```



```
                tac_adc_limit[1] = 100; /*10 + b200ns_offset;*/ /* 200 nS */
                test_limits( 1, 1, tac_loop );
        }
        if( tac_test_num == 1 )
        {
                tac_adc_limit[1] = 700; /*+ b20ns_offset;*/  /*  20 nS */
                tac_adc_limit[2] = 500; /* - b100ns_offset;*/ /* 100 ns */
                test_limits( 1, 2, tac_loop );
        }
/*      if( tac_test_num == 2 )
        ( */
                /* tac_adc_limit[1] = 800;*/ /*  40 nS */
                /*tac_adc_limit[2] = 700;*/ /*  80 nS */
                /*tac_adc_limit[3] = 500;*/ /* 120 nS */
                /*tac_adc_limit[4] = 300;*/ /* 160 nS */
                /*tac_adc_limit[5] = 200;*/  /* 200 nS */
        /*      test_limits( 1, 5, tac_loop );
        ) */
        if( amu_input_voltage == 0 || amu_input_voltage == 1)
        {
                test_limits( 0, dont_care, tac_loop );
        }
}
/*************************************************************************/

/*
function: test_limits
Requires:
        amu_or_tac = 0          =>      Amu being tested
        amu_or_tac = 1          =>      Tac being tested
        tac_test_col            =>      First column to be tested
        tac_loop                =>      Determines where the sequence test is

              at.
        tac_loop = 1            =>      Channel 0  being tested
        tac_loop = 2            =>      Channel 1  being tested
        tac_loop = 3            =>      Channel 2  being tested
        tac_loop = 4            =>      Channel 3  being tested
        tac_loop = 5            =>      Channel 4  being tested
        tac_loop = 6            =>      Channel 5  being tested
        tac_loop = 7            =>      Channel 6  being tested
        tac_loop = 8            =>      Channel 7  being tested

Returns :
Purpose : To test the J2 data against the code for the pulse being tested
          +/- the user inputed tolerance.
*/


/*************************************************************************/

void test_limits( int amu_or_tac, int tac_test_col, int tac_loop )
{
    int row, column, column_start, column_end;
        int row1, column1;
        int row2, column2;
        int row3, column3;
        int row4, column4;
        int row5, column5;
        int row6, column6;
        int row7, column7;
        int row8, column8;

      /*************************************************************/
      /*   Testing TAC channel 0 -> Tac test 00 -> 200nS Pulse only      */
      /*************************************************************/
```

```
        if( amu_or_tac == 1 && tac_loop == 1 && tac_test_col == 1 )
        {
                for( row1 = 1; row1 < 9; row1 ++ )
                {
                  for( column1 = 0; column1 < 1 ; column1 ++ )
                {
                        if( row1 == 1 )
                        {
                                if( (J2_data[row1][column1] < tac_adc_limit[ 1 ]
                                        - tolerance) || (J2_data[row1][column1] >

                                        tac_adc_limit[ 1 ] + tolerance ))
                                {
                                        store_bad_channel[ row1 ][ column1 ] = 1;

                                        terror_bit = 1;
                                }
                        }
                        if( row1 != 1 )
                        {
                                if( (J2_data[row1][column1] > Tac_timeout_count +
200)   )
                                {
                                        store_bad_channel[ row1 ][ column1 ] = 1;

                                        terror_bit = 1;
                                }
                        }
                  }
                }
        }
        /*****************************************************************/
        /*   Testing TAC channel 1 -> Tac test #0 -> 200nS Pulse only    */
        /*****************************************************************/

        if( amu_or_tac == 1 && tac_loop == 2 && tac_test_col == 1 )
        {
                for( row2 = 1; row2 < 9; row2 ++ )
                {
                  for( column2 = 1; column2 < 2 ; column2 ++ )
                  {
                        if( row2 == 2 )
                        {
                                if( (J2_data[row2][column2] < tac_adc_limit[ 1 ]
                                        - tolerance) || (J2_data[row2][column2] >

                                        tac_adc_limit[ 1 ] + tolerance ))
                                {
                                        store_bad_channel[ row2 ][ column2 ] = 1;

                                        terror_bit = 1;
                                }

                        }
                        if( row2 != 2 )
                        {
                                if(  (J2_data[row2][column2] > Tac_timeout_count
+ 200)   )
                                {
                                        store_bad_channel[ row2 ][ column2 ] = 1;

                                        terror_bit = 1;
                                }
                        }
```



```
                  }
                }
        }

        /*****************************************************************/
        /*   Testing TAC channel 2 -> Tac test #0 -> 200nS Pulse only    */
        /*****************************************************************/

        if( amu_or_tac == 1 && tac_loop == 3 && tac_test_col == 1 )
        {
                for( row3 = 1; row3 < 9; row3 ++ )
                {
                  for( column3 = 2; column3 < 3 ; column3 ++ )
                  {
                        if( row3 == 3  )
                        {
                                if( (J2_data[row3][column3] < tac_adc_limit[ 1 ]
                                        - tolerance) || (J2_data[row3][column3] >

                                        tac_adc_limit[ 1 ] + tolerance ))
                                {
                                        store_bad_channel[ row3 ][ column3 ] = 1;

                                        terror_bit = 1;
                                }
                        }
                        if( row3 != 3 )
                        {
                                if( (J2_data[row3][column3] > Tac_timeout_count +
200)   )
                                {
                                        store_bad_channel[ row3 ][ column3 ] = 1;

                                        terror_bit = 1;
                                }
                        }
                  }
                }
        }

        /*****************************************************************/
        /*   Testing TAC channel 3 -> Tac test #0 -> 200nS Pulse only    */
        /*****************************************************************/

        if( amu_or_tac == 1 && tac_loop == 4 && tac_test_col == 1 )
        {
                for( row4 = 1; row4 < 9; row4 ++ )
                {
                  for( column4 = 3; column4 < 4 ; column4 ++ )
                  {
                        if( row4 == 4 )
                        {
                                if( (J2_data[row4][column4] < tac_adc_limit[ 1 ]
                                        - tolerance) || (J2_data[row4][column4] >

                                        tac_adc_limit[ 1 ] + tolerance ))
                                {
                                        store_bad_channel[ row4 ][ column4 ] = 1;

                                        terror_bit = 1;
                                }
                        }
                        if( row4 != 4 )
                        {
                                if( (J2_data[row4][column4] > Tac_timeout_count +
200)   )
```

```
                        {
                                store_bad_channel[ row4 ][ column4 ] = 1;

                                terror_bit = 1;
                        }
                }
        }
}

/*****************************************************************/
/*   Testing TAC channel 4 -> Tac test #0 -> 200nS Pulse only    */
/*****************************************************************/

if( amu_or_tac == 1 && tac_loop == 5 && tac_test_col == 1 )
{
        for( row5 = 1; row5 < 9; row5 ++ )
        {
          for( column5 = 4; column5 < 5 ; column5 ++ )
          {
                if( row5 == 5 )
                {
                        if( (J2_data[row5][column5] < tac_adc_limit[ 1 ]
                                - tolerance) || (J2_data[row5][column5] >
                                tac_adc_limit[ 1 ] + tolerance ))
                        {
                                store_bad_channel[ row5 ][ column5 ] = 1;

                                terror_bit = 1;
                        }
                }
                if( row5 != 5 )
                {
                        if( (J2_data[row5][column5] > Tac_timeout_count +
200)    )
                        {
                                store_bad_channel[ row5 ][ column5 ] = 1;

                                terror_bit = 1;
                        }
                }
          }
        }
}

/*****************************************************************/
/*   Testing TAC channel 5 -> Tac test #0 -> 200nS Pulse only    */
/*****************************************************************/

if( amu_or_tac == 1 && tac_loop == 6 && tac_test_col == 1 )
{
        for( row6 = 1; row6 < 9; row6 ++ )
        {
          for( column6 = 5; column6 < 6 ; column6 ++ )
          {
                if( row6 == 6 )
                {
                        if( (J2_data[row6][column6] < tac_adc_limit[ 1 ]
                                - tolerance) || (J2_data[row6][column6] >
                                tac_adc_limit[ 1 ] + tolerance ))
                        {
                                store_bad_channel[ row6 ][ column6 ] = 1;

                                terror_bit = 1;
```



```
                        }
                }
                if( row6 != 6 )
                {
                        if( (J2_data[row6][column6] > Tac_timeout_count +
200)    )
                        {
                                store_bad_channel[ row6 ][ column6 ] = 1;

                                terror_bit = 1;
                        }
                }
          }
        }
}

/*****************************************************************/
/*   Testing TAC channel 6 -> Tac test #0 -> 200nS Pulse only    */
/*****************************************************************/

if( amu_or_tac == 1 && tac_loop == 7 && tac_test_col == 1 )
{
        for( row7 = 1; row7 < 9; row7 ++ )
        {
          for( column7 = 6; column7 < 7 ; column7 ++ )
          {
                if( row7 == 7 )
                {
                        if( (J2_data[row7][column7] < tac_adc_limit[ 1 ]
                                - tolerance) || (J2_data[row7][column7] >
                                tac_adc_limit[ 1 ] + tolerance ))
                        {
                                store_bad_channel[ row7 ][ column7 ] = 1;

                                terror_bit = 1;
                        }
                }
                if( row7 != 7 )
                {
                        if( (J2_data[row7][column7] > Tac_timeout_count +
200)    )
                        {
                                store_bad_channel[ row7 ][ column7 ] = 1;

                                terror_bit = 1;
                        }
                }
          }
        }
}

/*****************************************************************/
/*   Testing TAC channel 7 -> Tac test #0 -> 200nS Pulse only    */
/*****************************************************************/

if( amu_or_tac == 1 && tac_loop == 8 && tac_test_col == 1 )
{
        for( row8 = 1; row8 < 9; row8 ++ )
        {
          for( column8 = 7; column8 < 8 ; column8 ++ )
          {
                if( row8 == 8 )
                {
                        if( (J2_data[row8][column8] < tac_adc_limit[ 1 ]
                                - tolerance) || (J2_data[row8][column8] >
```

```
                                tac_adc_limit[ 1 ] + tolerance ))
                        {
                                store_bad_channel[ row8 ][ column8 ] = 1;
                                terror_bit = 1;
                        }
                }
                if( row8 != 8 )
                {
                        if( (J2_data[row8][column8] > Tac_timeout_count +
200)  )
                        {
                                store_bad_channel[ row8 ][ column8 ] = 1;
                                terror_bit = 1;
                        }
                }
            }
        }


        /*****************************************************************/
        /*                      Tac test #1                              */
        /*****************************************************************/



        /*****************************************************************/
        /*  Testing TAC channel 0 -> Tac test #1 -> 20nS & 100nS Pulse   */
        /*****************************************************************/


        if( amu_or_tac == 1 && tac_loop == 1 && tac_test_col == 2 )
        {
                for( row = 1; row < 9; row ++ )
                {
                  for( column = 0; column < tac_test_col ; column ++ )
                  {
                        if( row == 1 && column == 0 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                        - 500) || (J2_data[row][column] >
                                        tac_adc_limit[ 1 ] + 500 ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                        if( row == 1 && column == 1 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                        - tolerance) || (J2_data[row][column] >
                                        tac_adc_limit[ 2 ] + tolerance ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }

                        if( row != 1 && column != 0 || row != 1 && column != 1)
                        {
                                if( (J2_data[row][column] > Tac_timeout_count + 2
00)  )
                                {
```



```
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                    }
                }
        }


        /*****************************************************************/
        /*  Testing TAC channel 1 -> Tac test #1 -> 20nS & 100nS Pulse   */
        /*****************************************************************/


        if( amu_or_tac == 1 && tac_loop == 2 && tac_test_col == 2 )
        {
                for( row = 1; row < 9; row ++ )
                {
                  for( column = 2; column < 4 ; column ++ )
                  {
                        if( row == 2 && column == 2 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                        - 500) || (J2_data[row][column] >
                                        tac_adc_limit[ 1 ] + 500 ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                        if( row == 2 && column == 3 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                        - tolerance) || (J2_data[row][column] >
                                        tac_adc_limit[ 2 ] + tolerance ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }

                        if( row != 2 && column != 2 || row != 2 && column != 3)
                        {
                                if( (J2_data[row][column] > Tac_timeout_count + 2
00)  )
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                    }
                }
        }


        /*****************************************************************/
        /*  Testing TAC channel 2 -> Tac test #1 -> 20nS & 100nS Pulse   */
        /*****************************************************************/


        if( amu_or_tac == 1 && tac_loop == 3 && tac_test_col == 2 )
        {
                for( row = 1; row < 9; row ++ )
                {
                  for( column = 4; column < 6 ; column ++ )
                  {
```

```
                    if( row == 3 && column == 6 )
                    {
                            if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                    - 500) || (J2_data[row][column] >
                                    tac_adc_limit[ 1 ] + 500 ))
                            {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                            }
                    }
                    if( row == 3 && column == 5 )
                    {
                            if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                    - tolerance) || (J2_data[row][column] >
                                    tac_adc_limit[ 2 ] + tolerance ))
                            {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                            }
                    }

                    if( row != 3 && column != 4 || row != 3 && column != 5)
                    {
                            if( (J2_data[row][column] > Tac_timeout_count + 2
00)   )
                            {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                            }
                    }
                }
            }
        }

        /*****************************************************************/
        /*   Testing TAC channel 3 -> Tac test #1 -> 20nS & 100nS Pulse    */
        /*****************************************************************/


        if( amu_or_tac == 1 && tac_loop == 4 && tac_test_col == 2 )
        {
                for( row = 1; row < 9; row ++ )
                {
                  for( column = 6; column < 8 ; column ++ )
                  {
                        if( row == 4 && column == 6 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                        - 500 ) || (J2_data[row][column] >
                                        tac_adc_limit[ 1 ] + 500 ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                        if( row == 4 && column == 7 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                        - tolerance) || (J2_data[row][column] >
                                        tac_adc_limit[ 2 ] + tolerance ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
```



```
                    if( row != 4 && column != 6 || row != 4 && column != 7)
                    {
                            if( (J2_data[row][column] > Tac_timeout_count + 2
00)   )
                            {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                            }
                    }
                }
            }
        }

        /*****************************************************************/
        /*   Testing TAC channel 4 -> Tac test #1 -> 20nS & 100nS Pulse    */
        /*****************************************************************/


        if( amu_or_tac == 1 && tac_loop == 5 && tac_test_col == 2 )
        {
                for( row = 1; row < 9; row ++ )
                {
                  for( column = 8; column < 10 ; column ++ )
                  {
                        if( row == 5 && column == 8 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                        - 500 ) || (J2_data[row][column] >
                                        tac_adc_limit[ 1 ] + 500 ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                        if( row == 5 && column == 9 )
                        {
                                if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                        - tolerance) || (J2_data[row][column] >
                                        tac_adc_limit[ 2 ] + tolerance ))
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }

                        if( row != 5 && column != 8 || row != 5 && column != 9)
                        {
                                if( (J2_data[row][column] > Tac_timeout_count + 2
00)   )
                                {
                                        store_bad_channel[ row ][ column ] = 1;
                                        terror_bit = 1;
                                }
                        }
                    }
                }
        }

        /*****************************************************************/
        /*   Testing TAC channel 5 -> Tac test #1 -> 20nS & 100nS Pulse    */
        /*****************************************************************/


        if( amu_or_tac == 1 && tac_loop == 6 && tac_test_col == 2 )
        {
                for( row = 1; row < 9; row ++ )
```

```
                {
                  for( column = 10; column < 12 ; column ++ )
                  {
                        if( row == 6 && column == 10 )
                        {
                              if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                    - 500 ) || (J2_data[row][column] >
                                    tac_adc_limit[ 1 ] + 500 ))
                              {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                              }
                        }
                        if( row == 6 && column == 11 )
                        {
                              if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                    - tolerance) || (J2_data[row][column] >
                                    tac_adc_limit[ 2 ] + tolerance ))
                              {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                              }
                        }

                        if( row != 6 && column != 10 || row != 6 && column != 11)
                        {
                              if( (J2_data[row][column] > Tac_timeout_count + 2
00)   )
                              {
                                    store_bad_channel[ row ][ column ] = 1;
                                    terror_bit = 1;
                              }
                        }
                  }
                }

                /*****************************************************************/
                /*   Testing TAC channel 6 -> Tac test #1 -> 20nS & 100nS Pulse    */
                /*****************************************************************/


                if( amu_or_tac == 1 && tac_loop == 7 && tac_test_col == 2 )
                {
                      for( row = 1; row < 9; row ++ )
                      {
                        for( column = 12; column < 14 ; column ++ )
                        {
                              if( row == 7 && column == 12 )
                              {
                                    if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                          - 500 ) || (J2_data[row][column] >
                                          tac_adc_limit[ 1 ] + 500 ))
                                    {
                                          store_bad_channel[ row ][ column ] = 1;
                                          terror_bit = 1;
                                    }
                              }
                              if( row == 7 && column == 13 )
                              {
                                    if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                          - tolerance) || (J2_data[row][column] >
                                          tac_adc_limit[ 2 ] + tolerance ))
                                    {
                                          store_bad_channel[ row ][ column ] = 1;
```



```
                                          terror_bit = 1;
                                    }
                              }

                              if( row != 7 && column != 12 || row != 7 && column != 13)
                              {
                                    if( (J2_data[row][column] > Tac_timeout_count + 2
00)   )
                                    {
                                          store_bad_channel[ row ][ column ] = 1;
                                          terror_bit = 1;
                                    }
                              }
                        }
                      }

                /*****************************************************************/
                /*   Testing TAC channel 7 -> Tac test #1 -> 20nS & 100nS Pulse    */
                /*****************************************************************/


                if( amu_or_tac == 1 && tac_loop == 8 && tac_test_col == 2 )
                {
                      for( row = 1; row < 9; row ++ )
                      {
                        for( column = 14; column < 16 ; column ++ )
                        {
                              if( row == 8 && column == 14 )
                              {
                                    if( (J2_data[row][column] < tac_adc_limit[ 1 ]
                                          - 500 ) || (J2_data[row][column] >
                                          tac_adc_limit[ 1 ] + 500 ))
                                    {
                                          store_bad_channel[ row ][ column ] = 1;
                                          terror_bit = 1;
                                    }
                              }
                              if( row == 8 && column == 15 )
                              {
                                    if( (J2_data[row][column] < tac_adc_limit[ 2 ]
                                          - tolerance) || (J2_data[row][column] >
                                          tac_adc_limit[ 2 ] + tolerance ))
                                    {
                                          store_bad_channel[ row ][ column ] = 1;
                                          terror_bit = 1;
                                    }
                              }

                              if( row != 8 && column != 14 || row != 8 && column != 15)
                              {
                                    if( (J2_data[row][column] > Tac_timeout_count + 2
00)   )
                                    {
                                          store_bad_channel[ row ][ column ] = 1;
                                          terror_bit = 1;
                                    }
                              }
                        }
                      }
                }
```

```
        /*****************************************************************/
        /*                          Testing AMU                          */
        /*****************************************************************/


    if( amu_or_tac == 0 )
    {
      if( (J1_data & 0x2000) == 0x2000 ) /* test low input voltage nf 1.5V */
      {
                        for( row = 9; row < 25; row ++ )
                        {
                                for( column = 0; column < 16; column ++ )
                                {
                                        if( (  J2_data[row][column] < amu_adc_lim
it[1] -  amu_tolerance)
                                                ||( J2_data[row][column] > amu_ad
c_limit[1] + amu_tolerance) )
                                        {
                                                store_bad_channel[row][column] =
1;
                                                terror_bit = 1;
                                        }
                                }
                        }
      }
      if( (J1_data & 0x2000) == 0 ) /* Test for 3V input voltage */
      {
                        for( row = 25; row < 41; row ++ )
                        {
                                for( column = 0; column < 16; column ++ )
                                {
                                  if( ((J2_data[row][column]) < (amu_adc_limit[2]
 - amu_tolerance ))
                                  || ((J2_data[row][column]) > ( amu_adc_limit[2]
 + amu_tolerance )) )
                                        {
                                                store_bad_channel[row][column] =
1;
                                                terror_bit = 1;
                                        }
                                }
                        }
              }
    }
}



/*****************************************************************/

/*
   Function Name: input_tolerance
   Requires:  void
   Returns:   void
   Purpose:   provide user input for tolerance from the ideal value
*/

/*****************************************************************/

void input_tolerance( void )
{
        char buffer[80];
```



```
        cls();
                /*locate(10,5,0);
        printf("Input tolerance value for the TAC: ");*/

                _settextposition(11,4);
                _settextcolor( 12 );
                sprintf( buffer, "Input tolerance value for the TAC (norm = 100)
: " );
                _outtext( buffer );


        scanf("%d", &tolerance );

                /*locate(12,5,0);
        printf("Input the AMU tolerance: ");*/

                _settextposition(13,4);
                _settextcolor( 12 );
                sprintf( buffer, "Input tolerance value for the AMU (norm = 100)
: " );
                _outtext( buffer );


        scanf("%d", &amu_tolerance );

        b20ns_offset  = tac_timeout_count;
        b100ns_offset = tac_timeout_count;
        b200ns_offset = tac_timeout_count;
                /*locate(14,25,0);
        printf("Are these values correct? (Y/N): ");*/

                _settextposition(15,16);
                _settextcolor( 12 );
                sprintf( buffer, "Are these values correct? (y/n) " );
                _outtext( buffer );


        toler_ans = getch();
        if( toler_ans == 'N' || toler_ans == 'n' )
        {
                input_tolerance( );
        }

}
```

```
/*
   TACSELEC.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           5 JUL 94
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <conio.h>
#include "constants.h"
#include <stdlib.h>
#include <stdio.h>
#include <graph.h>
#include "variables.h"
/*************************************************************************/
/********                      Variables                        ********/
/*************************************************************************/

extern int J1_temp, J3_temp;
extern int choice;
extern long chip_id;


int num_of_pulses, pulse_number;
int pulse[10];
int i, J3data, J1data;
int J3_data_old;
int channel_hex_code, repeat_channel_select;
int col;
int selected_channel;
int sequence_mode;
int row, column, channel_num;
char chn0, chn1, chn2, chn3, chn4, chn5, chn6, chn7;
char channel, sequence;
char channel_correct, answer;

/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/

int tac_test_select( void );
void tac_channel_select( void );
void print_tac_variables( void );
void print_tac_channels( void );
void select_channel( char, int, int, int, int);
void channel_display( int );
void test_display ( int );
void print_tac_channel( void );
void pulse_display( int );

/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/




/*************************************************************************/

/*
   Function Name: tac_test_select
   Requires:  void
```



```
   Returns:   int, used for setting a flag to exit out of chiptest
   Purpose:   to provide selection menu & set appropriate global variables
              select tests
*/

/*************************************************************************/


int tac_test_select( void )
{
        char buffer[80];

        _setbkcolor(8); /*black*/
        _settextposition(8,18);
        _settextcolor( 15 );
        sprintf( buffer, "MODIFY CHIP TAC TEST PARAMETERS MENU" );
        _outtext( buffer );
        _settextposition(11,10);
        _settextcolor( 11 );
        sprintf( buffer, "TAC Test" );
        _outtext( buffer );
                locate(12,12,0);
        printf("   Test 0 => Continuous 200 nS Pulse Separation ");
                locate(14,12,0);
        printf("   Test 1 => 20 nS & 100 nS Pulse Separation (Preferred)");
                locate(16,12,0);
        printf("   Exit");


        _settextposition(13,12);
        _settextcolor( 15 );
        sprintf( buffer, "1." );
        _outtext( buffer );

        _settextposition(15,12);
        _settextcolor( 15 );
        sprintf( buffer, "2." );
        _outtext( buffer );

        _settextposition(17,12);
        _settextcolor( 15 );
        sprintf( buffer, "3." );
        _outtext( buffer );




                locate(20,12,0);
      printf("Selection: ");
      choice = getch();
      switch(choice)
      {
      case '1':
              J1_data = 0;
              J3_data = 0;
              J1_data = pulse_200ns;
              J3_data = test0;
              num_of_pulses = 1;
              pulse_number = 1;
              break;
      case '2':
              J1_data = 0;
              J3_data = 0;
              J1_data = pulse_20ns;
              J3_data = test1;
              num_of_pulses = 2;
```

```
                pulse_number = 1;
                pulse[2] = pulse_100ns;
                break;
/*      case '3':
                J1_data = 0;
                J3_data = 0;
                J1_data = pulse_40ns;
                J3_data = test2;
                num_of_pulses = 5;
                pulse_number = 1;
                pulse[2] = pulse_80ns;
                pulse[3] = pulse_120ns;
                pulse[4] = pulse_160ns;
                pulse[5] = pulse_200ns;
                break;  */
        case '3':
                return( 1 );
                break;
        default:
                chiptest(  chip_id );
                break;
        }
}


/*************************************************************************/

/*
   Function Name: tac_channel_select
   Requires: void
   Returns:  void
   Purpose:  selects channels for tac to be selected and sets appropriate
             bits.
*/
   ,
/*************************************************************************/


void tac_channel_select( void )
{
        int wrong_input;
        char buffer[80];

                locate(5,28,0);
        printf("TAC Channel Selection");
                locate(6,10,0);
        printf("To select:          Place a 'x' by the desired channel/channels")
;
                locate(7,10,0);
        printf("                    or place a 'x' by SEQUENCE MODE (Preferred)")
;
                locate(8,10,0);
        printf("Not to select:      Press Any Key");

/*************************************************************************/
/*
        The following printf is placed above in the code what it will be
        printed in under on the screen so that the cursor will appear at
        the select mode.
*/
/*************************************************************************/

                locate(11,15,0);
        printf("(sequentially check each channel)");
```



```
                /*locate(10,15,0);
        printf("Sequence Mode: ");*/

        _settextposition(11,15);
        _settextcolor( 15 );
        sprintf( buffer, "Sequence Mode: " );
        _outtext( buffer );


                sequence = 0;
                sequence = getch();
                switch( sequence )
                {
                   case 'x':
                   case 'X':
                        sequence_mode = 1;
                        J3_data |=  chan0;
                        J3_temp = J3_data;
                        J1_temp = J1_data;
                                locate(10,30,0);
                        printf("%c", 'X');
                        return;
                        break;
                   default:
                        sequence_mode = 0;
                        break;
                }

                locate(14,15,0);
        printf("Channel 0: ");
        chn0 = getch();
        select_channel( chn0, 14, 15, 0, chan0);
                locate(15,15,0);
        printf("Channel 1: ");
        chn1 = getch();
        select_channel( chn1, 15, 15, 1, chan1);
                locate(16,15,0);
        printf("Channel 2: ");
        chn2 = getch();
        select_channel( chn2, 16, 15, 2, chan2);
                locate(17,15,0);
        printf("Channel 3: ");
        chn3 = getch();
        select_channel( chn3, 17, 15, 3, chan3);
                locate(18,15,0);
        printf("Channel 4: ");
        chn4 = getch();
        select_channel( chn4, 18, 15, 4, chan4);
                locate(19,15,0);
        printf("Channel 5: ");
        chn5 = getch();
        select_channel( chn5, 19, 15, 5, chan5);
                locate(20,15,0);
        printf("Channel 6: ");
        chn6 = getch();
        select_channel( chn6, 20, 15, 6, chan6);
                locate(21,15,0);
        printf("Channel 7: ");
        chn7 = getch();
        select_channel( chn7, 21, 15, 7, chan7);

/*************************************************************************/
/*              Check to see if selection/s is/are correct               */
/*************************************************************************/

        while( wrong_input )
```

```
        {
                locate(23,15,0);
          printf("Is this correct? [Y/N]: ");
                locate(23,45,0);
          printf("J1 Data = %x     J3 Data = %x", J1_data, J3_data );
                locate(23,38,0);
          channel_correct = getch();
          switch( channel_correct )
          {
                case 'y':
                case 'Y':
                        J3_temp = J3_data;
                        J1_temp = J1_data;
                        wrong_input = 0;
                        break;
                case 'n':
                case 'N':
                        wrong_input = 0;
                        print_tac_channels();
                        repeat_channel_select = 1;
                        J3_data = J3_data_old;
                        tac_channel_select();
                        break;
                default:
                        repeat_channel_select = 1;
                        J3_data = J3_data_old;
                        wrong_input = 1;
                        break;
          }
        }
}


/*************************************************************************/

/*
   Function Name: print_tac_channels
   Requires:  void
   Returns:   void
   Purpose:   displays selected tac channels on screen
*/

/*************************************************************************/

void print_tac_channels( void )
{
                locate(14,15,0);
        printf("Channel 0:  ");
                locate(15,15,0);
        printf("Channel 1:  ");
                locate(16,15,0);
        printf("Channel 2:  ");
                locate(17,15,0);
        printf("Channel 3:  ");
                locate(18,15,0);
        printf("Channel 4:  ");
                locate(19,15,0);
        printf("Channel 5:  ");
                locate(20,15,0);
        printf("Channel 6:  ");
                locate(21,15,0);
        printf("Channel 7:  ");
}
```



```
/*************************************************************************/

/*
   Function Name: select_channel
   Requires:
           channel:            is a character that is used to select the
                                                                channel i
f an 'x' is inputed.
                row & column:       are used for location of printf statements

           channel_num:        used for selection

           channel_hex_code:   used for data to be output to the J3
                                                                connector
 to select the channels

   Returns:  void
   Purpose:  used for selection
*/

/*************************************************************************/


void select_channel( char channel, int row, int column, int channel_num,
                     int channel_hex_code )
{
        if( (channel == 'x' || channel == 'X')
            && ( repeat_channel_select == 0 ) && ( channel_num == 0) )
        {

/*************************************************************************/

/*
  Have to be careful with the following code such that the channel data is
  not added back onto itself because this will give erroneous(? sp) results.
  Thus the reason for all of the selection 'if' statements.
*/

/*************************************************************************/

                J3_data_old = J3_data;  /* Stores the orginal J3 value */
                J3_data |=  channel_hex_code;

/*************************************************************************/

/*
  This locate 'column + 11' is to place the cursor at the end of the item.
*/

/*************************************************************************/

                locate( row, column + 11,0);
                printf("%c", 'X');
                _exit;
        }
        else if( (channel == 'x' || channel == 'X')
            && ( repeat_channel_select == 0 ) && ( channel_num > 0) )
        {

/*************************************************************************/

/*
  This section repetively adds J3 back to itself.  This only adds the channel
```

```
  info and nothing else.
*/

/*************************************************************************/

                J3_data += channel_hex_code;

/*************************************************************************/

/*
  This locate 'column + 11' is to place the cursor at the end of the item.
*/

/*************************************************************************/

                locate( row, column + 11,0);
                printf("%c", 'X');
                _exit;
        }
        else if( (channel == 'x' || channel == 'X')
                && ( repeat_channel_select == 1 ) && (channel_num == 0) )
        {

/*************************************************************************/

/*
  Initializes the J3 data to the orignal value.  So that J3 data is not
  compounded with the '+=' operator when the channels are changed more than
  once.
*/

/*************************************************************************/

                J3_data = J3_data_old;
                J3_data += channel_hex_code;
                locate( row, column + 11,0);
                printf("%c", 'X');
                _exit;
        }
        else if( (channel == 'x' || channel == 'X')
                && ( repeat_channel_select == 1 ) && (channel_num > 0) )
        {


/*************************************************************************/

/*
  This section adds the channel data to the J3_data after initializing it in
  the previous 'if' statement.  This is because with only the previous 'if'
  statement, the resultant J3_data is only the last channel inputed due to
  the initializing of J3_data with the J3_data_old
*/

/*************************************************************************/

                J3_data += channel_hex_code;
                locate( row, column + 11,0);
                printf("%c", 'X');
                _exit;
        }
        else if( channel == ' ' )
        {
                return;
        }
}
```



```

/*************************************************************************/

/*
   Function Name: channel_display
   Requires:  int type, J3 data
   Returns:   void
   Purpose:   displays channel #'s on the screen at appropriate columns
*/

/*************************************************************************/


void channel_display( int J3data )
{
        int channel_data = 0;
        char buffer[80];

        if( sequence_mode == 1 )
        {
                channel_data = 0x7f80;
        }
        if( sequence_mode != 1 )
        {
                channel_data = J3_data & 0x7f80;
        }
                /*locate(0,54,0);
        printf("Channels Selected");*/

        _settextposition(1,55);
        _settextcolor( 10 );
        sprintf( buffer, "Channels Selected" );
        _outtext( buffer );

        if( ( channel_data & 0x0080 ) == 0x0080 )
        {
                        locate(1,55,0);
                printf("0");
        }
        if( ( channel_data & 0x0100 ) == 0x0100 )
        {
                        locate(1,57,0);
                printf("1");
        }
        if( ( channel_data & 0x0200 ) == 0x0200 )
        {
                        locate(1,59,0);
                printf("2");
        }
        if( ( channel_data & 0x0400 ) == 0x0400 )
        {
                        locate(1,61,0);
                printf("3");
        }
        if( ( channel_data & 0x0800 ) == 0x0800 )
        {
                        locate(1,63,0);
                printf("4");
        }
        if( ( channel_data & 0x1000 ) == 0x1000 )
        {
                        locate(1,65,0);
                printf("5");
        }
        if( ( channel_data & 0x2000 ) == 0x2000 )
```

```
        {
                        locate(1,67,0);
                printf("6");
        }
        if( ( channel_data & 0x4000 ) == 0x4000 )
        {
                        locate(1,69,0);
                printf("7");
        }
}


/*************************************************************************/

/*
   Function Name: test_display
   Requires:  int type, J3 data
   Returns:   void
   Purpose:   displays test that was previously selected
*/

/*************************************************************************/

void test_display( int J3data )
{
        int test_data = 0;
        char buffer[80];

        test_data = J3data & 0x0070;
        if( !test_data )
        {
                /*   locate(0,1,0);
                printf("Test: 0");
                        locate(2,1,0);
                printf("200 nS Pulse only!");*/

                _settextposition(2,1);
                _settextcolor( 15 );
                sprintf( buffer, "Test: 0" );
                _outtext( buffer );

                _settextposition(3,1);
                _settextcolor( 15 );
                sprintf( buffer, "200nS Pulse only!" );
                _outtext( buffer );

        }
        else if( test_data == 0x0010 )
        {
                /*   locate(0,1,0);
                printf("Test: 1");
                        locate(2,1,0);
                printf("20 nS & 100 nS Pulse");*/

                _settextposition(2,1);
                _settextcolor( 15 );
                sprintf( buffer, "Test: 1" );
                _outtext( buffer );

                _settextposition(3,1);
                _settextcolor( 15 );
                sprintf( buffer, "20nS & 100nS Pulse" );
                _outtext( buffer );
```



```
        }
        else if( test_data == 0x0020 )
        {
                        /*locate(0,1,0);
                printf("Test: 2");
                        locate(2,1,0);
                printf("40 nS, 80nS, 120 nS, 160nS & 200 nS");*/

                _settextposition(2,1);
                _settextcolor( 15 );
                sprintf( buffer, "Test: 2" );
                _outtext( buffer );

                _settextposition(3,1);
                _settextcolor( 15 );
                sprintf( buffer, "40nS, 80nS, 120nS, 160nS & 200nS Pulse" );
                _outtext( buffer );

        }
}


/*************************************************************************/

/*
   Function Name: pulse_display
   Requires:  int type, J1 data
   Returns:   void
   Purpose:   to display the pulse sequences that will be sent to the tec
*/

/*************************************************************************/

void pulse_display( int J1data )
{
        int pulse_data = 0;
        char buffer[80];

        pulse_data = J1data & 0x01e0;
        if( pulse_data == 0 )
        {
        /*      locate(3,1,0);
                printf("First pulse is: 20 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 20 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x0120 ) == 0x0020 )
        {
                /*   locate(3,1,0);
                printf("First pulse is: 40 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 40 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x0040 ) == 0x0040 )
        {
                        /*locate(3,1,0);
```

```
                printf("First pulse is: 60 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 60 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x0060 ) == 0x0060 )
        {
                        /*locate(3,1,0);
                printf("First pulse is: 80 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 80 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x0080 ) == 0x0080 )
        {
                        /*locate(3,1,0);
                printf("First pulse is: 100 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 100 nS Pulse" );
                _outtext( buffer );


        }
        else if( ( pulse_data & 0x00a0 ) == 0x00a0 )
        {
                        /*locate(3,1,0);
                printf("First pulse is: 120 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 120 nS Pulse" );
                _outtext( buffer );

        }
        else if( (pulse_data & 0x00c0 ) == 0x00c0 )
        {
                        /*locate(3,1,0);
                printf("First pulse is: 140 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 140 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x00e0 ) == 0x00e0 )
        {
                        /*locate(3,1,0);
                printf("First pulse is: 160 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 160 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x0120 ) == 0x0100 )
        {
```



```
                        /*locate(3,1,0);
                printf("First pulse is: 180 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 180 nS Pulse" );
                _outtext( buffer );

        }
        else if( ( pulse_data & 0x0120 ) == 0x0120 )
        {
                        /*locate(3,1,0);
                printf("First pulse is: 200 nS Pulse");*/

                _settextposition(4,1);
                _settextcolor( 15 );
                sprintf( buffer, "First pulse is: 200 nS Pulse" );
                _outtext( buffer );

        }
}
```

```
/*
   CHIPTEST1.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           5 JUL 94 - AUG 94
   VER: 1.0
   DATE: 9 FEB 95
   VER: 2.0  MODIFIED READING AND WRITING SEQUENCES

*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <conio.h>
#include "constants.h"
#include <stdlib.h>
#include <stdio.h>
#include <graph.h>
#include "variables.h"

/*************************************************************************/
/********                     Variables                         ********/
/*************************************************************************/

extern int x;
extern int adc_address;
extern int num_of_pulses, pulse_number, pulse[10];
extern int pulse_2, pulse_3, pulse_4, pulse_5, pulse_6;
extern int fsc;
int J1data, J1dat;
int J3data;
int tac_loop_num, multi_column;
int amu_read;

/*************************************************************************/
/********                     Functions                         ********/
/*************************************************************************/

void next_tac_test( int, int, int, int );
void trigger_adc( int );
void check_fsc( void );
void read_amu( int, int, int, int );
void tac_test( int, int );
void write_amu( int );

/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/



/*************************************************************************/

/*
   Function Name: tac_test
   Requires: int type, J1 & J3 data
   Returns:  void
   Purpose:  test tac with the initial pulse sequence of any tac test selected
*/

/*************************************************************************/


void tac_test( int J1data, int J3data )
```



```
{
        int tmp1, tmp3a, tmp3p;
        char buffer[80];

        /*   locate(6,5,0);
        printf("TAC Test Started");*/

        _settextposition(7,6);
        _settextcolor( 15 );
        sprintf( buffer, "-> TAC Test Started" );
        _outtext( buffer );

/*************************************************************************/

/*
  Sets up Mach part prior to sending the tac_strt
  cmd so that no timing conflicts occur.
*/

/*************************************************************************/

        tmp3a = outpw( io_base_address + J3lower_byte, J3data );
        tmp1 = outpw( io_base_address + J1lower_byte, J1data );
        tmp3p = outpw( io_base_address + J3lower_byte, J3data + tac_strt );


/*************************************************************************/
/*                     Clears tac strt command                           */
/*************************************************************************/

        outpw( io_base_address + J3lower_byte, J3data );

                /* Clears all the commands */
        outpw( io_base_address + J1lower_byte, dataclr );
        outpw( io_base_address + J3lower_byte, dataclr );




                locate(7,5,0);
        printf("J1 = %x  J3 prior = %x  J3 sent = %x  Pulse #: 1   Pulse: %x",
                tmp1, tmp3a, tmp3p, pulse[1] );


        _settextposition(7,6);
        _settextcolor( 15 );
        sprintf( buffer, "-> TAC Test Started" );
        _outtext( buffer );

}


/*************************************************************************/

/*
   Function Name: next_tac_test
   Requires: 4 int types, J1 & J3 data tac_loop_num & multi column to pass
             to the read_adc function.
   Returns:  void
   Purpose:  to test the tac with more than one pulse sequence
*/

/*************************************************************************/
```

```
void next_tac_test( int J1data, int J3data, int tac_loop_num,
                    int multi_column )
{
        int num_of_puls;
        int tmp1, tmp3a, tmp3p;
        char buffer[80];


        J1data = J1data & pulse_clr_mask;
        for( num_of_puls = 1; num_of_puls < num_of_pulses;
             num_of_puls ++ )
        {
                int next_pulse;
                next_pulse = pulse[ num_of_puls + 1 ];
        tmp3a = outpw( io_base_address + J3lower_byte, J3data );
        tmp1 = outpw( io_base_address + J1lower_byte, J1data + next_pulse );
        tmp3p = outpw( io_base_address + J3lower_byte, J3data + tac_strt );

                /* Clears tac_strt command */
        outpw( io_base_address + J3lower_byte, J3data );

                /* Clears all the commands */
        outpw( io_base_address + J1lower_byte, dataclr );
        outpw( io_base_address + J3lower_byte, dataclr );


        /*   locate(6,5,0);
        printf("TAC Test Started");*/

        _settextposition(7,6);
        _settextcolor( 15 );
        sprintf( buffer, "-> TAC Test Started" );
        _outtext( buffer );

                locate(7,5,0);
        printf("J1 = %x  J3 prior = %x  J3 sent = %x  Pulse #: %d  Pulse: %x",
                tmp1, tmp3a, tmp3p, num_of_puls +1, next_pulse );


        /*   locate(9,5,0);
        printf("-> Triggered ADC");
                locate(9,25,0);
        printf("Waiting for FSC");
                locate(9,50,0);
        printf("Reading TAC ADC's");*/

        _settextposition(10,6);
        _settextcolor( 3 );
        sprintf( buffer, "-> Triggered ADC" );
        _outtext( buffer );

        trigger_adc( 1 );

        _settextposition(10,26);
        _settextcolor( 4 );
        sprintf( buffer, "Waiting for FSC" );
        _outtext( buffer );

        check_fsc();

        _settextposition(10,51);
        _settextcolor( 2 );
        sprintf( buffer, "Reading TAC ADC's" );
        _outtext( buffer );
```



```
        read_adc( tac, dont_care, num_of_puls, tac_loop_num, multi_column,
                  num_of_puls + 1 );
        outpw( io_base_address + J1lower_byte, dataclr );

        }

}


/*****************************************************************************/

/*
   Function Name: read_amu
   Requires: 4 int types, J1 & J3 data and row num's for data to be printed
   Returns:  void
   Purpose:  reads the 16 amu addresses, triggers the adc, reads out columns
             of the adc for each amu address.
*/

/*****************************************************************************/


void read_amu( int J1data, int J3data, int one, int two )
{
        int amu_pipe;
        char buffer[80];

        for( amu_pipe = 0; amu_pipe < 16; amu_pipe ++ )
        {

                if( amu_pipe == 3 || amu_pipe == 7 || amu_pipe == 11 )
                {
                        write_amu(J1_data);
                }

                outpw( io_base_address + J3lower_byte, amu_pipe );
                outpw( io_base_address + J1lower_byte,
                        J1data + read );

                trigger_adc( 0 );

                check_fsc();

                _settextposition(one,6);
                _settextcolor( 3 );
                sprintf( buffer, "-> Triggered ADC" );
                _outtext( buffer );

                _settextposition(one,26);
                _settextcolor( 4 );
                sprintf( buffer, "Waiting for FSC" );
                _outtext( buffer );


                        /*locate(one,50,0);
                printf("-> Reading AMU ADC's");*/

                _settextposition( one, 51);
                _settextcolor( 14 );
                sprintf( buffer, "-> Reading AMU ADC's" );
                _outtext( buffer );

                read_adc( amu, amu_pipe, dont_care, dont_care, dont_care,
                          dont_care );
```

```
                outpw( io_base_address + J1lower_byte,
                        datac1r );
        }
}


/*************************************************************************/


/*
   Function Name: write_amu
   Requires: int type, J1 data
   Returns:  void
   Purpose:  to set write command to the connector J1 and then clear it
*/

/*************************************************************************/


void write_amu( int J1data )
{

        outpw( io_base_address + J1lower_byte,
                J1data );                               /* sets up voltage prior to wri
te */

        outpw( io_base_address + J1lower_byte,
                J1data + write );
        outpw( io_base_address + J1lower_byte,
                J1data );                                /* clears write command */

}


/*************************************************************************/

/*
   Function Name: trigger_adc
        Requires:  int to determine whether or not the amu is being read so read
                                signal does not get reset.
   Returns:   void
   Purpose: sends adc trigger signal to the J1 connector for the chip under
            test.
*/

/*************************************************************************/


void trigger_adc( int amu_read )
{
        if( amu_read == 0 )  /* trigger sent during amu read cycle */
        {
                outpw( io_base_address + J1lower_byte,
                        adc_oe + adc_trig + read);
                outpw( io_base_address + J1lower_byte,
                        adc_oe + read );
        }
        if( amu_read == 1 )  /* trigger sent during tac cycle */
        {
                outpw( io_base_address + J1lower_byte,
                        adc_oe + adc_trig );
                outpw( io_base_address + J1lower_byte,
               adc_oe );
        }
```



```
}


/*************************************************************************/

/*
   Function Name: check_fsc
   Requires: void
   Returns:  void
   Purpose:  to check the fsc bit on the J2 connector (bit 16) and wait
             until it is active.
*/

/*************************************************************************/


void check_fsc( void )
{
        do
        {
                fsc = (inpw(io_base_address + J2lower_byte)
                        & 0x8000);
        } while( fsc );
}
```

```
/*
   TESTCHIP1.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           29 JUN 94 - AUG 94
   VER: 1.0
   DATE: 9 FEB 95
   VER: 2.0  MODIFIED READING AND WRITING SEQUENCES

*/

/*****************************************************************/
/********                    Headers                      ********/
/*****************************************************************/

#include <stdio.h>
#include <conio.h>
#include "constants.h"
#include <stdlib.h>
#include <graph.h>

/*****************************************************************/
/********                   Variables                     ********/
/*****************************************************************/

extern int J1_data, J2_data[41][17], J3_data, num_of_pulses;
extern int sequence_mode, terror_bit;

int amu_voltage, all_odd_even, tac_test_loop;

/*****************************************************************/
/********                   Functions                     ********/
/*****************************************************************/

void test_chip( void );


/*****************************************************************/
/********                     Code                        ********/
/*****************************************************************/



/*---------------------------------------------------------------*/

/*
   Function Name: test_chip
   Requires:  void
   Returns:   void
   Purpose:   provides hand shaking & commands to test the daq chip

*/

/*---------------------------------------------------------------*/


void test_chip( void )
{
        int tmp, first, multi_col, k;
        int i,j;
        double z;
        char buffer[80];

        /* Start Testing of Chip */

/*---------------------------------------------------------------*/
```



```
/*****************************************************************/

/*
                Sequence of Events for TAC Testing:

        1. Is this a seqence? If so, at all_odd_even = 4.

        2. Check tac_test_loop.  If 2 or 3 change J3_data to
           appropriate value.

        3. Send parameters chosen by outputing them to the
           appropriate connectors.

        4. Send the tac_strt command.

        5. Trigger the ADC and wait for FSC.

        6. Is this the last time around the loop? If not, go to step 2.

        7. Read in the TAC ADC's into an array.
*/

/*****************************************************************/
/*****************************************************************/

        outpw( io_base_address + J1lower_byte, reset ); /* reset Roch part */
        outpw( io_base_address + J1lower_byte, datac1r );
        outpw( io_base_address + J3lower_byte, datac1r );

        trigger_adc( 1 ); /*setup tac clr so that first conversion for tac be rea
dy*/

        /* check to see if sequence mode */
        if( sequence_mode == 1)
        {
                all_odd_even = 9; /* This number sets the loop count for the tac
*/
        }
        else
        {
                all_odd_even = 1;
        }
        if( all_odd_even == 1 )
        {
                tac_test( J1_data, J3_data );


                _settextposition(7,6);
                _settextcolor( 15 );
                sprintf( buffer, "-> TAC Test Started" );
                _outtext( buffer );


          _settextposition(10,6);
                _settextcolor( 3 );
                sprintf( buffer, "-> Triggered ADC" );
                _outtext( buffer );

                trigger_adc( 1 );

                        /*locate(9,25,0);
                printf("Waiting for FSC");*/

                _settextposition(10,26);
                _settextcolor( 4 );
                sprintf( buffer, "Waiting for FSC" );
```

```
                _outtext( buffer );

                check_fsc();

                        /*locate(9,50,0);
                printf("Reading TAC ADC's");*/

                _settextposition(10,51);
                _settextcolor( 2 );
                sprintf( buffer, "Reading TAC ADC's" );
                _outtext( buffer );


                read_adc( tac, dont_care, 0, dont_care, dont_care,
                          dont_care );
                if( num_of_pulses > 1 )
                {
                        next_tac_test( J1_data, J3_data, dont_care,
                                       dont_care );
                }

/*************************************************************************/

/*
  J3 'anded' with 0x0070 masks off all data except the test data.
  A '1' is sent to the test_limits() function to meet the logic
  condition for the 'if' statement to set the proper data storage.
*/

/*************************************************************************/

                test_data( (J3_data & 0x0070), dont_care, 1 );
        }
        else
        {
                /*J3_data = J3_data & 0x807f;
                J3_data += chan0;*/
                for( tac_test_loop = 1; tac_test_loop < all_odd_even;
                     tac_test_loop ++ )
                {
                        if( tac_test_loop == 2 )
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan1;
                        }
                        if( tac_test_loop == 3 )
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan2;
                        }
                        if( tac_test_loop == 4 )
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan3;
                        }
                        if( tac_test_loop == 5 )
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan4;
                        }
                        if( tac_test_loop == 6 )
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan5;
                        }
                        if( tac_test_loop == 7 )
```



```
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan6;
                        }
                        if( tac_test_loop == 8 )
                        {
                                J3_data = J3_data & 0x807f;
                                J3_data += chan7;
                        }

                        tac_test( J1_data, J3_data);

                        _settextposition(10,6);
                        _settextcolor( 3 );
                        sprintf( buffer, "-> Triggered ADC" );
                        _outtext( buffer );

                        trigger_adc( 1 );

                                /*locate(9,25,0);
                        printf("Waiting for FSC");*/

                        _settextposition(10,26);
                        _settextcolor( 4 );
                        sprintf( buffer, "Waiting for FSC" );
                        _outtext( buffer );

                        check_fsc();

                                /*locate(9,50,0);
                        printf("Reading TAC ADC's");*/

                        _settextposition(10,51);
                        _settextcolor( 2 );
                        sprintf( buffer, "Reading TAC ADC's" );
                        _outtext( buffer );


                        read_adc( tac, dont_care, dont_care, tac_test_loop,
                                  0, 1 );
                        if( num_of_pulses > 1 )
                        {
                                next_tac_test( J1_data, J3_data,
                                               tac_test_loop, 1 );
                        }

/*************************************************************************/

/*
  J3 'anded' with 0x0070 masks off all data except the test data
*/

/*************************************************************************/

                        test_data( ( J3_data & 0x0070), dont_care,
                                   tac_test_loop );
                }
        }

/*************************************************************************/
/*************************************************************************/

/*
                Sequence of Events for AMU Testing:
```

```
        1. Write to the 16x16 AMU matrix with desired voltages voltages

        2. Read the first/next pipe (column) of the AMU's into the ADC

        3. Perform the read_address function (with the AMU constant) to
                read in the current AMU address.  Start with address 8 end with
                address 23.

        4. Increment the AMU address by one and goto step 1 until all 16
           AMU's have been read and converted.
*/

/*************************************************************************/
/*************************************************************************/

                /*locate(11,5,0);
        printf("-> Writing 1.5 volts to the AMU");*/


                _settextposition(12,6);
                _settextcolor( 15 );
                sprintf( buffer, "-> Writing 1.5 Volts to the AMU" );
                _outtext( buffer );


/*************************************************************************/
/*              Setup 1.5 volts for amu to storage                       */
/*************************************************************************/

        amu_voltage = 1; /* set up to read in 1.5V input */
        outpw( io_base_address + J1lower_byte, amu_voltage_1 );
        J1_data |= amu_voltage_1;

        for(z=1; z < 60000; z++ );


        /* Start AMU testing for first voltage of 1.5 volts*/

                _settextposition(14,6);
                _settextcolor( 14 );
                sprintf( buffer, "-> Reading the AMU" );
                _outtext( buffer );

        write_amu( J1_data );



        /*
          The #'s 13 & 14 are to place the text at the appropriate rows
        */
        read_amu( J1_data, J3_data, 13, 14 );

        test_data( dont_care, amu_voltage, dont_care );

/*************************************************************************/
/*              Setup 3.5 volts for amu to storage                       */
/*************************************************************************/

        amu_voltage = 0; /* set up for reading 3V input voltage */

        /*
          Clear the AMU voltage bit.  This sets up 3 volts to be applied
          to the AMU.
        */
```



```
        J1_data = J1_data & amu_voltage_3;
        outpw( io_base_address + J1lower_byte, J1_data );

/*************************************************************************/
/*              Start AMU testing for second voltage of 3.5 volts        */
/*************************************************************************/

                /*locate(17,5,0);
        printf("-> Writing 3.5 volts to the AMU");*/

                _settextposition(18,6);
                _settextcolor( 15 );
                sprintf( buffer, "-> Writing 3.5 Volts to the AMU" );
                _outtext( buffer );


        for(z=1; z < 60000; z++ );

        write_amu( J1_data );
                /*locate(19,5,0);
        printf("-> Reading the AMU");*/

                _settextposition(20,6);
                _settextcolor( 14 );
                sprintf( buffer, "-> Reading the AMU" );
                _outtext( buffer );


        read_amu( J1_data, J3_data, 19, 20 );
        _displaycursor(0);
        while( !kbhit() )
        {
                        /*locate(22,30,0);
                printf("Press any Key");*/

                _settextposition(23,31);
                _settextcolor( 23 );
                sprintf( buffer, "Press any Key" );
                _outtext( buffer );

        }
        _displaycursor(1);



        test_data( dont_care, amu_voltage, dont_care);



/*
                        used to display data matrix on the screen
*/

/*   j=0;
        clio();
        for(i=0; i<41; i++ )
        {
                printf("\n[%d]=",i);
                for(j=0; j<16; j++ )
                {
                  printf(" %d", J2_data[i][j]);

                        if( i == 7 && j == 15 )
                        {
```

```
                        getch();
                    }
                    if( i == 23 && j == 15 )
                    {
                        getch();
                    }

                }
        }
*/

        getch();
        if( terror_bit == 0 )
        {
                cls();

                _settextposition(12,30);
                _settextcolor( 31 );
                sprintf( buffer, "CHIP PASSES!" );
                _outtext( buffer );

                _settextposition(23,25);
                _settextcolor( 19 );
                sprintf( buffer, "Press any Key to continue" );
                _outtext( buffer );

                getch();
                terror_bit = 0;
                cls();
        }



        if( terror_bit == 1 )
        {
                cls();
                fflush(stdin);


                 settextposition(12,16);
                 settextcolor( 28 );
                sprintf( buffer, "Chip fails! See data error file for details." )
;
                _outtext( buffer );

                _settextposition(23,25);
                _settextcolor( 31 );
                sprintf( buffer, "Press any Key to continue" );
                _outtext( buffer );


                getch();
                terror_bit = 0;
                cls();
        }
}
```

```
/*
   RETEST.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           6 JUL 94
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "constant>.h"
#include <stdlib.h>
#include <graph.h>
#include "variables.h"

/*************************************************************************/
/********                      Variables                        ********/
/*************************************************************************/

extern int J1_temp, J3_temp;
extern int test_again;
extern char again;
extern char temp_file[20], file_name[25], file_name1[25], drive_nam[20];
extern long chip_id;
extern int lac_timeout_count;

char again;
int the_end = 0;

/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/

void retest( void );
void clear_J2( void );
void run_test_again( void );
void print_J2( void );
void print_bad_data( void );

/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/




/*************************************************************************/

/*
   Function Name: retest
   Requires:   void
   Returns:    void
        Purpose:   select whether a test will be run again & set/reset data in th
e                                 appropriate conditions.
*/

/*************************************************************************/


void retest( void )
{
        int wrong_key, J1_tmp, J3_tmp;
```



```
        char same_drive, same_parameters, buffer[80];
        the_end = 0;

                print_J2();

                print_bad_data();


                getch();

        while( !the_end )
        {
                cls();
                        /*locate(10,20,0);


                printf("Do you want to test another chip [Y/N]: ");*/

                _settextposition(11,21);
                _settextcolor( 15 );
                sprintf( buffer, "Do you want to test another chip? [y/n]: " );
                _outtext( buffer );


                locate(0,0,0);
                again = 0;
                again = getch();
                switch( again )
                {
                        case 'y':
                        case 'Y':

                                cls();

                                        locate(10,22,0);
                                printf("Turn off power and remove tested chip");
                                        locate(20,28,0);
                                printf("Press any key when ready");
                                getch();

                                input_chip_num();

                                cls();

                                /*   locate(10,25,0);
                                printf("Place new chip in test fixture");
                                        locate(20,28,0);
                                printf("Press any key when ready");
                                getch();*/


                                cls();
                                _settextposition(12,10);
                                _settextcolor( 15 );
                                sprintf( buffer, "Ensure that chip is inserted an
d ready for ADC &amp adjustment" );
                                _outtext( buffer );
                                        locate(20,22,0);
                                printf("Press any key to begin adjustment");

                                getch();

                                /*asm2();*///*this is used for the averaging metho
d*/
```

```
                        ramp();


                        getch();

                        cls();

                        retac_adj();

                        cls();

                        tac_timeout_generator();

                        getch();

                        cls();

                        _settextposition(15,21);
                        _settextcolor( 15 );
                        sprintf( buffer, "Same Drive & Path name? (y/n):
" );
                        _outtext( buffer );


                        same_drive = getch();
                        switch( same_drive )
                        {
                            case 'n':
                            case 'N':
                                        input_drive( chip_id );
                                        break;
                            default:
                                        break;
                        }
                        while( wrong_key )
                        {

                                _settextposition(17,21);
                                _settextcolor( 15 );
                                sprintf( buffer, "Same Test parameters? (
y/n): " );

                                _outtext( buffer );

                          same_parameters = getch();
                          switch( same_parameters )
                          {
                                case 'y':
                                case 'Y':
                                        test_again = 1;
                                        wrong_key = 0;
                                        break;
                                case 'n':
                                case 'N':
                                        cls();

                                        J1_tmp = J1_data;
                                        J3_tmp = J3_data;

                                        clear_J2();

                                        J1_data = J1_tmp;
                                        J3_data = J3_tmp;

                                        chiptest( chip_id );
                                        wrong_key = 0;
                                        break;
```



```
                                default:
                                        wrong_key = 1;
                                        break;
                          }
                        }
                        the_end = 1;
                        break;
                case 'n':
                case 'N':
                                        J1_tmp = J1_data;
                                        J3_tmp = J3_data;

                                        clear_J2();

                                        J1_data = J1_tmp;
                                        J3_data = J3_tmp;

                        again = 0;
                        the_end = 1;
                        break;
                default:
                        the_end = 0;
                        break;
            }
        }
}


/*************************************************************************/

/*
   Function Name: clear_J2
   Requires:  void
   Returns:   void
   Purpose:   clears out the J2_data matrix
*/
/*************************************************************************/


void clear_J2( void )
{
        /* clears J2_data & store_bad_data matrix */
        int x, y;
        for( x = 0; x < 61; x++ )
        {
                for( y = 0; y < 18; y++ )
                {
                        J2_data[ x ][ y ] = 0;
                        store_bad_channel[x][y] = 0;
                }
        }
}


/*************************************************************************/

/*
   Function Name: print_J2
   Requires: void
   Returns:  void
   Purpose:  to display J2_matrix data on the screen
*/
```

```
/*************************************************************************/


void print_J2( void )
{
        int x, y = 0;
        cls();
        printf("J2 data matrix\n");
        for( x = 0; x < 22; x++ )
        {
                printf("%d  %d = ", x, y);
                for( y = 0; y < 17; y++ )
                {
                        printf("%d ", J2_data[ x ][ y ]);
                }
                printf("\n");
        }
        printf("\n\t\t press any key\n");
        getch();
        for( x = 22; x < 41; x++ )
        {
                printf("%d %d = ", x, y);
                for( y = 0; y < 17; y++ )
                {
                        printf("%d ", J2_data[ x ][ y ]);
                }
                printf("\n");
        }
        printf("\n\t\t press any key\n");
        getch();
}


/*************************************************************************/

/*
   Function Name: print_bad_data
   Requires: void
   Returns:  void
   Purpose:  displays store_bad_channel matrix on the screen
*/
/*************************************************************************/


void print_bad_data ( void )
{
        int x, y = 0;
        cls();
        printf("store bad channel matrix\n");
        for( x = 0; x < 22; x++ )
        {
                printf("%d  %d = ", x, y);
                for( y = 0; y < 17; y++ )
                {
                        printf("%d ", store_bad_channel[ x ][ y ]);
                }
                printf("\n");
        }
        printf("\n\t\t press any key\n");
        while( !kbhit() )
        {
        }
        for( x = 22; x < 41; x++ )
        {
```



```
                printf("%d %d = ", x, y);
                for( y = 0; y < 17; y++ )
                {
                        printf("%d ", store_bad_channel[ x ][ y ]);
                }
                printf("\n");
        }
        printf("\n\t\t press any key\n");

        fflush(stdin);

        getch();
}


/*************************************************************************/

/*

   Function Name: run_test_again
   Requires:  void
   Returns:   void
   Purpose:   initializes variables & flags, tests chip, and saves data
*/

/*************************************************************************/


void run_test_again( void )
{
        int r, c, J1_tmp, J3_tmp;
        cls();
                                                J1_tmp = J1_data;
                                                J3_tmp = J3_data;

                                                clear_J2();

                                                J1_data = J1_tmp;
                                                J3_data = J3_tmp;


        J1_data = J1_temp;
        J3_data = J3_temp;
        test_display( J3_data );
        pulse_display( J1_data );
        channel_display( J3_data );
        J2_data[0][0] = chip_id;
        test_chip();
        again = 0;

        store_bad_channel[0][0] = chip_id;

        for( r = 0 ; r < 1; r++ )
        {
                for( c = 1; c < 16; c++ )
                {
                        store_bad_channel[r][c] = 0;
                }
        }

        save_data( chip_id );
        test_again = 0;
}
```

```
/*
   FILE OPS.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           19 JULY 94 -
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <graph.h>
#include "variables.h"

/*************************************************************************/
/********                      Variables                        ********/
/*************************************************************************/

extern char file_name[25], file_name1[25], drive_nam[20];

long chip_ident;

/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/

void save_data( long );
void input_drive( void );

/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/


/************************************************************************/

/*
   Function Name: Save_data
   Requires: int type, chip # that is being tested
   Returns: void
   Purpose: reads out of memory the 41x17 arrays of J2 dat &
            store bad_channels. Then stores the information in a file.

*/

/************************************************************************/


void save_data( long chip_ident)
{
        FILE *fp;
        int x,y;
        char buffer[80];

        cls();

        sprintf( file_name, "%sc%d.dat", drive_nam, chip_ident );
        sprintf( file_name1, "%sc%d.err", drive_nam, chip_ident );

        /*sprintf( file_tac, "%sc%dt.dat", drive_nam, chip_ident );
        sprintf( file_tac_err, "%sc%dt.err", drive_nam, chip_ident );
        sprintf( file_amu1_5, "%sc%dal.dat", drive_nam, chip_ident );
        sprintf( file_amu1_5err, "%sc%dal.err", drive_nam, chip_ident );
        sprintf( file_amu3_5, "%sc%dau.dat", drive_nam, chip_ident );
        sprintf( file_amu3_5err, "%sc%dau.err", drive_nam, chip_ident );*/
```



```
        if ( (fp = fopen( file_name,"w")) !=NULL )
        {
                for ( y = 0; y < 16; y ++ )
                {
                        for( x = 0; x < 9; x ++ )
                        {
                                fprintf( fp, "%4.1d,", J2_data[x][y] );
                        }
                        fputc( '\n', fp );
                }
                fclose( fp );
        }
        if ( (fp = fopen( file_name,"a")) !=NULL )
        {
                for ( y = 0; y < 16; y ++ )
                {
                        for( x = 9; x < 25; x ++ )
                        {
                                fprintf( fp, "%4.1d,", J2_data[x][y] );
                        }
                        fputc( '\n', fp );
                }
                fclose( fp );
        }

        if ( (fp = fopen( file_name,"a")) !=NULL )
        {
                for ( y = 0; y < 16; y ++ )
                {
                        for( x = 25; x < 41; x ++ )
                        {
                                fprintf( fp, "%4.1d,", J2_data[x][y] );
                        }
                        fputc( '\n', fp );
                }
                fclose( fp );
        }
        else
        {
                        /*locate(17,25,0);
                printf("Cannot open file for c%d.dat", chip_ident );*/

                _settextposition(17,26);
                _settextcolor( 31 );
                sprintf( buffer, "Cannot open file!" );
                _outtext( buffer );


        }

        if ( (fp = fopen( file_name1,"w")) !=NULL )
        {
                for ( y = 0; y < 16; y ++ )
                {
                        for( x = 0; x < 9; x ++ )
                        {
                                fprintf( fp, "%4.1d,", store_bad_channel[x][y] );

                        }
                        fputc( '\n', fp );
                }
                fclose( fp );
        }
        if ( (fp = fopen( file_name1,"a")) !=NULL )
```

```
        {
                for ( y = 0; y < 16; y ++ )
                {
                        for( x = 9; x < 25; x ++ )
                        {
                                fprintf( fp, "%4.1d,", store_bad_channel[x][y] );

                        }
                        fputc( '\n', fp );
                }
                fclose( fp );
        }
        if ( (fp = fopen( file_name1,"a")) !=NULL )
        {
                for ( y = 0; y < 16; y ++ )
                {
                        for( x = 25; x < 41; x ++ )
                        {
                                fprintf( fp, "%4.1d,", store_bad_channel[x][y] );

                        }
                        fputc( '\n', fp );
                }
                fclose( fp );
        }
        else
        {
                        locate(16,25,0);
                printf("Cannot open file for c%d.err", chip_ident );
        }
                locate(14,19,0);
        printf("the following files were saved to: " );
                locate(16,30,0);
        printf( "%sc%d.dat", drive_nam, chip_ident );
                locate(17,30,0);
        printf( "%sc%d.err", drive_nam, chip_ident );


         settextposition(25,25);
        _settextcolor( 12 );
        sprintf( buffer, "Press any key to continue");
        _outtext( buffer );

        getch();
}

/*************************************************************************/

/*
   Function Name: input_drive
   Requires:      integer for the # of the chip
   Returns:       void
   Purpose:       Stores the drive, path, and filename in the global
                  variable drive_nam that is used in the save_data function.
*/

/*************************************************************************/

void input_drive( void )
{
        char buffer[80];

        /*printf("\n\t\t");
        printf("Input 'Drive:\Path name\' : ");*/
```



```
                _settextposition(13,15);
                _settextcolor( 12 );
                sprintf( buffer, "Input 'Drive:\Pathname\' : ");
                _outtext( buffer );

        scanf("%s", drive_nam);
}
```

```
/*
   AMU.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           10 AUG 94 -
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "constants.h"
#include <stdlib.h>
#include <graph.h>
#include "variables.h"

/*************************************************************************/
/********                      Variables                        ********/
/*************************************************************************/

int num_of_pulses;
int sequence_mode, terror_bit;
int amu_read, all_odd_even;
int sequence_mode, num_of_pulses, terror_bit;
int num_of_pulses, pulse_number, pulse[10];
int pulse_2, pulse_3, pulse_4, pulse_5, pulse_6;
int fsc;
int tac_timeout_count;/*not used in this program used for the chip prg. */
int amu_voltage, all_odd_even, tac_test_loop;

/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/









/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/




/*************************************************************************/

/*
   Function Name: main
   Requires:   void
   Returns:    void
   Purpose:    to convert a amu voltage and display the data

*/

/*************************************************************************/


void amu2( void )
{
        int tmp, first, multi_col, i, j;
        double amu_1_5, avg1_5, amu_3_5, avg3_5;
```



```
        double q;
        char cont;
        double k;

        outpw( io_base_address + J1lower_byte, 0x2000 );
        /* Start Testing of Chip */
        while( !kbhit() )
        {
/*************************************************************************/

/*
                Sequence of Events for AMU Testing:

        1. Write to the 16x16 AMU matrix with desired voltages voltages

        2. Read the first pipe (column) of the AMU's into the ADC

        3. Perform the read_address function (with the AMU constant to
           read in the 8 thru 23 AMU ADC's.

        4. Increment the AMU address by one and goto step 2 until all 16
           AMU's have been read and converted.
*/

/*************************************************************************/
/*************************************************************************/


/*************************************************************************/
/*              Setup 1.5 volts for amu to storage                       */
/*************************************************************************/

        cls();
        amu_voltage = 1; /* set up to read in 1.5V input */

        outpw( io_base_address + J1lower_byte, reset );
        outpw( io_base_address + J1lower_byte, amu_voltage_1 );

        J1_data |= amu_voltage_1;  /* outputs 1.5 volts to the amu */

        /* Start AMU testing for first voltage of 1.5 volts*/

        for( q = 0; q < 40000; q++);

        write_amu( J1_data );

        /*
          The #'s 13 & 14 are to place the text at the appropriate rows
        */


        read_amu( J1_data, J3_data, 100, 140 );


        J1_data &= 0xdfff;      /* outputs 3.5 volts to the amu */

        outpw( io_base_address + J1lower_byte, J1_data );

        /* Start AMU testing for first voltage of 3.5 volts*/

        /*
          The #'s 13 & 14 are to place the text at the appropriate rows
        */

        for( q = 0; q < 40000; q++);
```

```
        write_amu( J1_data );

        read_amu( J1_data, J3_data, 130, 140 );


        j=0;
        cls(),

/*   for(i=10; i<20; i++ )  uncomment to use full screen display for amu data
        {
                printf("\n[%d]= ",i);
                for(j=0; j<16; j++ )
                {
                  settextcolor(15);
                  printf(" %d", J2_data[i][j]);
                }
        }

        printf("\n\n");

        for(i=26; i<36; i++ )
        {
                printf("\n[%d]= ",i);
                for(j=0; j<16; j++ )
                {
                  printf(" %d", J2_data[i][j]);
                }
        }*/

        amu_1_5 = 0;
        amu_3_5 = 0;

        for(i=9; i< 24; i++ )
        {
                for(j=0; j<16; j++ )
                {
                  amu_1_5 += J2_data[i][j];
                }
        }
                                2
        avg1_5 = amu_1_5/256;

        for(i=26; i< 41; i++ )
        {
                for(j=0; j<16; j++ )
                {
                  amu_3_5 += J2_data[i][j];
                }
        }

        avg3_5 = amu_3_5/256;

                locate(6,13,0);
        printf("Adjust Pot #1 so that the 1.5V input is from -690 to -760");
                locate(8,22,0);
        printf("and 3.5V input is from -240 to -320");

                locate(18,18,0);
        printf("The AMU average value for 1.5 V input is: %3.0lf", avg1_5 );

                locate(20,18,0);
        printf("The AMU average value for 3.5 V input is: %3.0lf", avg3_5 );

        for( k = 1; k < 500000; k++ );
```



```
  }
  return;
}
```

```
/*
   CHIP.C
   WRITTEN BY:     MICHAEL WOMAC
   DATE:           21 JUN 94 - AUG 94
   VER: 1.0
   DATE: 9 FEB 95
   VER: 2.0  MODIFIED READING AND WRITING SEQUENCES
*/

/*****************************************************************************/
/********                        Headers                            ********/
/*****************************************************************************/

#include <stdio.h>
#include "scrlib.h"
#include <conio.h>
#include <graph.h>
#include <process.h>
#include "constants.h"
#include <stdlib.h>
#include "variables.h"

/*****************************************************************************/
/********                     Global Variables                      ********/
/*****************************************************************************/

int J1_temp, J3_temp;
int choice, routine, test_again;
int terror_bit;

long chip_id;

char again, file_name[25], file_name1[25];
char drive_nam[20];

/*****************************************************************************/
/********                     Structures                            ********/
/*****************************************************************************/


/*****************************************************************************/
/********                     Functions                             ********/
/*****************************************************************************/

int m_menu( void );
void input_chip_num( void );

/*****************************************************************************/
/********                     Code                                  ********/
/*****************************************************************************/


/*****************************************************************************/

/*
   Function Name: main
   Requires:  void
   Returns:   void
   Purpose:   start program
*/

/*****************************************************************************/

main()
{
```



```
        int endflag = 0;
        cls();
        while(!endflag)
        {
                choice = 0;
                choice = m_menu();
                switch(choice)
                {
                        case '1':
                                cls();
                                adjust();
                                break;
                        case '2':
                                cls();
                                input_chip_num();
                                input_drive();
                                cls();
                                chiptest( chip_id );
                                break;
                        case '3':
                                cls();
                                endflag = 1;
                                break;
                }
        }
_setvideomode( _DEFAULTMODE );
}



/*****************************************************************************/

/*
   Function Name: m_menu
   Requires: void
   Returns:  int type, the selection made
   Purpose:  provide selection for the main menu
*/

/*****************************************************************************/


int m_menu( void )
{

        char buffer[80];
        cls();
        _settbkcolor(0); /*black*/
        _settextposition(8,35);
        _settextcolor( 15 ); /*bright white*/
        sprintf( buffer, "Main Menu" );
        _outtext( buffer );

                locate(10,20,0);
        printf("     ADC &amp and TAC Adjustment Routine ");
                locate(12,20,0);
        printf("     Perform Chip Test Routine ");
                locate(14,20,0);
        printf("     Exit to DOS ");

        _settextposition(11,22);
        _settextcolor( 15 );
        sprintf( buffer, "1." );
        _outtext( buffer );
```

```
        _settextposition(13,22);
        _settextcolor( 15 );
        sprintf( buffer, "2." );
        _outtext( buffer );

        _settextposition(15,22);
        _settextcolor( 15 );
        sprintf( buffer, "3." );
        _outtext( buffer );


                locate(18,20,0);
        printf(" Selection: ");
        routine = getch();
        return(routine);

}


/*************************************************************************/

/*
   Function Name: input_chip_id
   Requires:      void
   Returns:       void
   Purpose:       Stores the chip identification # in the global variable
                  chip_id.
*/

/*************************************************************************/

void input_chip_num( void )
{
        char buffer[80];
        char yesno;

        cls();

        _settextposition(8,15);
        _settextcolor( 3 );
        sprintf( buffer,"Chip Identification Number can be Scanned in using bar "
);
        _outtext( buffer );

        _settextposition(9,19);
        _settextcolor( 3 );
        sprintf( buffer, "code reader or by entering number via keyboard " );
        _outtext( buffer );

        _settextposition(12,15);
        _settextcolor( 12 );
        sprintf( buffer, "Enter Chip Identification Number: " );
        _outtext( buffer );

        scanf("%ld", &chip_id );


                locate(14,14,0);
        printf("Apply power to test fixture and check current limit" );

                locate(16,14,0);
        printf("Chip id =  %ld   Is this correct? (y/n): ", chip_id );

        yesno = getch();
```



```
        if( yesno == 'n' || yesno == 'N' )
        {
                input_chip_num();
        }


}
```

APPENDIX C

LIN3.MAK AND ASSOCIATED PROGRAMS

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = LIN3
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = /d DEBUG
R_RCDEFINES = /dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\PERFORM\LINPRGS\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = AMUINT3.C
FIRSTCPP = LIN3.CPP
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AH /Od /D "_DEBUG" /D "_DOS" /FR /Fd"LIN3.PD
B"
CFLAGS_R_DEXE = /nologo /Gs /G2 /W3 /AH /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE /CO
LFLAGS_R_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE
LIBS_D_DEXE = oldnames llibce stdaq_c
LIBS_R_DEXE = oldnames llibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = AMUINT3.SBR \
               LIN3.SBR \
               LEAST.SBR


AMUINT3_DEP = c:\perform\linprgs\constant.h


LIN3_DEP = c:\perform\linprgs\nidaq.h \
        c:\perform\linprgs\nidaqerr.h \
        c:\perform\linprgs\nidaqcns.h \
        c:\perform\linprgs\constant.h


LEAST_DEP =
```



```
all:    $(PROJ).EXE $(PROJ).BSC

AMUINT3.OBJ:    AMUINT3.C $(AMUINT3_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c AMUINT3.C

LIN3.OBJ:       LIN3.CPP $(LIN3_DEP)
        $(CPP) $(CFLAGS) $(CPPCREATEPCHFLAG) /c LIN3.CPP

LEAST.OBJ:      LEAST.CPP $(LEAST_DEP)
        $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c LEAST.CPP

$(PROJ).EXE::   AMUINT3.OBJ LIN3.OBJ LEAST.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
AMUINT3.OBJ +
LIN3.OBJ +
LEAST.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
c:\nidaqwin\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
                /******** Constants ********/

#define upper_limit      0x03ff
#define tac              0
#define amu              1
#define amu_voltage_1    0x2000 /* sent to connector J1 */
#define amu_voltage_3    0xdfff /* sent to connector J1 */
#define dataclr          0x0000
#define data_mask        0x03ff
#define dont_care        0xffff

#define io_base_address 0x02a0
#define J1lower_byte     0x0000
#define J1upper_byte     0x0001
#define J2lower_byte     0x0000
#define J2upper_byte     0x0001
#define J3lower_byte     0x0002
#define J3upper_byte     0x0003

/* math code masks */

#define reset            0x8000 /* sent to connector J1 */
#define read             0x0800 /* sent to connector J1 */
#define write            0x0400 /* sent to connector J1 */
#define adc_ramp         0x0200 /* sent to connector J1 */
#define adc_trig         0x0200 /* sent to connector J1 */
#define clr_adc_trig     0xfdff /* sent to connector J1 */
#define adc_oe           0x1000 /* sent to connector J1 */
#define tac_strt         0x8000 /* sent to connector J3 */


#define pulse_clr_mask   0xfe1f /* sent to connector J1 */

#define pulse_20ns       0x0000 /* sent to connector J1 */
#define pulse_40ns       0x0020 /* sent to connector J1 */
#define pulse_60ns       0x0040 /* sent to connector J1 */
#define pulse_80ns       0x0060 /* sent to connector J1 */
#define pulse_100ns      0x0080 /* sent to connector J1 */
#define pulse_120ns      0x00a0 /* sent to connector J1 */
#define pulse_140ns      0x00c0 /* sent to connector J1 */
#define pulse_160ns      0x00e0 /* sent to connector J1 */
#define pulse_180ns      0x0100 /* sent to connector J1 */
#define pulse_200ns      0x0120 /* sent to connector J1 */

#define chan0            0x0080 /* sent to connector J3 */
#define chan1            0x0100 /* sent to connector J3 */
#define chan2            0x0200 /* sent to connector J3 */
#define chan3            0x0400 /* sent to connector J3 */
#define chan4            0x0800 /* sent to connector J3 */
#define chan5            0x1000 /* sent to connector J3 */
#define chan6            0x2000 /* sent to connector J3 */
#define chan7            0x4000 /* sent to connector J3 */

#define test0            0x0000 /* sent to connector J3 */
#define test1            0x0010 /* sent to connector J3 */
#define test2            0x0020 /* sent to connector J3 */
#define test3            0x0030 /* sent to connector J3 */
#define test4            0x0040 /* sent to connector J3 */
#define test5            0x0050 /* sent to connector J3 */
#define test6            0x0060 /* sent to connector J3 */
#define test7            0x0070 /* sent to connector J3 */
```

```
/*
  amuint.c
  Modified funtions from original Chip2.exe code to do performance
  testing on the amu
  Date: May 95
  Written by: Michael Womac
*/


/*************************************************************************/
/********                      Headers                            ********/
/*************************************************************************/

#include <conio.h>
#include <graph.h>
#include <stdlib.h>
#include <stdio.h>
#include <graph.h>
#include "constant.h"

/*************************************************************************/
/********                      Variables                          ********/
/*************************************************************************/

//global
extern int  J1_data,
                        J3_data,
                        amuInputData[][16];

int fsc;
int J1data, J1dat;
int J3data;
int amu_read;


/*************************************************************************/
/********                      Functions                          ********/
/*************************************************************************/

void trigger_adc( int );
void check_fsc( void );
void read_amu( int, int, int);
void write_amu( int );
void print_J2( void );

/*************************************************************************/

/*
   Function Name: read_amu
   Requires: 4 int types, J1 & J3 data and row num's for data to be printed
   Returns:  void
   Purpose:  reads the 16 amu addresses, triggers the adc, reads out columns
             of the adc for each amu address.
*/

/*************************************************************************/


void read_amu( int J1data, int J3data, int amuaddress )
{
        int amu_pipe;

      //Sends desired AMU address to ATA chip
                outpw( io_base_address + J3lower_byte, amuaddress );
```



```
        write_amu( J1data );

                outpw( io_base_address + J1lower_byte,
                        J1data + read );

        //      _settextposition(100,6);
        //      _settextcolor( 3 );
        //      sprintf( buffer, "-> Triggered ADC" );
        //      _outtext( buffer );

                trigger_adc( 0 );


        //      _settextposition(100,26);
        //      _settextcolor( 4 );
        //      sprintf( buffer, "Waiting for FSC" );
        //      _outtext( buffer );

                check_fsc();

        //      _settextposition( 100, 51);
        //      _settextcolor( 14 );
        //      sprintf( buffer, "--> Reading AMU ADC's" );
        //      _outtext( buffer );

        for( amu_pipe = 0; amu_pipe < 16; amu_pipe ++ )
        {
      //reads the amu adc's 8 - 23 (or amu pipes)
                outpw( io_base_address + J1lower_byte,
                                        (amu_pipe+8) + adc_oe );

          amuInputData[ 0 ][ amu_pipe ] =
                                (inpw( io_base_address + J2lower_byte )& data_ma
sk );
        }
        outpw( io_base_address + J1lower_byte,
                        datoclr );

}


/*************************************************************************/


/*
   Function Name: write_amu
   Requires: int type, J1 data
   Returns:  void
   Purpose:  to set write command to the connector J1 and then clear it
*/
/*************************************************************************/


void write_amu( int J1data )
{
        outpw( io_base_address + J1lower_byte,
                J1data );                               /* sets up voltage prior to wri
te */

        outpw( io_base_address + J1lower_byte,
                J1data + write );
        outpw( io_base_address + J1lower_byte,
                J1data );                               /* clears write command */
```

```
}


/*************************************************************************/

/*
   Function Name: trigger_adc
        Requires:  int to determine whether or not the amu is being read so read
                                signal does not get reset.
   Returns:   void
   Purpose: sends adc trigger signal to the J1 connector for the chip under
            test.
*/

/*************************************************************************/


void trigger_adc( int amu_read )
{
        if( amu_read == 0 )  /* trigger sent during amu read cycle */
        {
                outpw( io_base_address + J1lower_byte,
                        adc_oe + adc_trig + read);
                outpw( io_base_address + J1lower_byte,
                        adc_oe + read );
        }
        if( amu_read == 1 )  /* trigger sent during tac cycle */
        {
                outpw( io_base_address + J1lower_byte,
                        adc_oe + adc_trig );
                outpw( io_base_address + J1lower_byte,
               adc_oe );
        }
}


/*************************************************************************/

/*
   Function Name: check_fsc
   Requires: void
   Returns:  void
   Purpose:  to check the fsc bit on the J2 connector (bit 16) and wait
             until it is active.
*/

/*************************************************************************/


void check_fsc( void )
{
        do
        {
                fsc = (inpw(io_base_address + J2lower_byte)
                        & 0x8000);
        } while( fsc );
}
```

```
/*
   LIN3.CPP
   WRITTEN BY:     MICHAEL WOMAC
   DATE:  24 May 95 - 16 Jun 95
   Ver. 1.0
*/

/*************************************************************************/

/*
   Function Name: Lin3.exe
   Requires:  command line: 'chip number #points MaxVoltage'
   Returns: void
   Purpose: This prg uses the AT-MIO-16X DAQ board DACO.  It provides an
   input voltage for the Pb Glass chip test fixture and all hand
   shake signals needed to interface for a write, read, and adc
   conversion.  The prg outputs a ramp voltage in steps determined
   by the increment at the command line (min increment = 1/80V, this is due
   to DOS memory problems). The data for each voltage increment is then sent
   to the variance function to have the variance, mean, std
   calculated. The voltage range is 0-5v. The results are stored in a file
   with "c'chipnum'.ext" and ext of .mn, var, or .std for mean, variance,
   std respectively. Also, the INL is stored under a filename of 'c#total.inl.
   The raw data is stored under a file with ext of .dat.  All files are
   stored in \perform\linprgs\data subdirectory.

   Tests Cell is cell #8 for all 16 pipes

*/

/*************************************************************************/



/*************************************************************************/
/********                      Headers                            ********/
/*************************************************************************/

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include <math.h>

//current directory header file
#include "nidaq.h"      /* function definitions */
#include "nidaqerr.h"   /* error code definitions */
#include "nidaqcns.h"   /* NI-DAQ constants definitions */
#include "constants.h"

/*************************************************************************/
/********                      Defines                            ********/
/*************************************************************************/

#define MAX_NUM_POINTS_PER_VOLTAGE 1000 // # of iterations per voltage
#define MAX_NUM_OF_SAMPLES 500 //determines the max # of voltage samples

/*************************************************************************/
/********                      Structures                         ********/
/*************************************************************************/
/*************************************************************************/
/********                      Global Variables                   ********/
```



```
/*************************************************************************/

 double
              sumX = 0,
              sumX2 = 0,
              sumY[16] = {0},
              sumXY[16] = {0},
              voltage;

       int
              huge ADCresults[MAX_NUM_POINTS_PER_VOLTAGE][16] = {0},
              NumofPoints_per_voltage,
              point,
              voltage_level,
              chip,
              status;


/*************************************************************************/
/********                      Functions                          ********/
/*************************************************************************/

extern "C"
{
              int     JJ_data,
                      JJ_data,
                      amuInputData[16][16];
              void read_amu( int, int, int);
              void write_amu( int );
              void print_J2( void );
}

void least_squares( int, int, int, int, int, int );
void variance( int, int, int );

/*************************************************************************/
/********                      Code                               ********/
/*************************************************************************/


void main( int argc, char **argv )
{

       //Note: the limit in the increment is due to least squares allocating the

       //mean_data matrix
   if( argc != 7 )
   {
       cout << "Usage: chipnum #points increment upper lower low_voltage" << end
l;
      cout << "chipnum: is an integer value for chip id" << endl
              << "#points:  is how many points to be taken at each voltage incr
ement"
              << endl
              << "increment: is step in which voltage is incremented"
              << endl << "i.e. 2 => 1/2V, 4 => 1/4V, 10 => 1/10V" << endl
              << "NOTE: increment must be <= 80 (or 1/80V)!!!" << endl
              << "upper: upper limit to begin inl testing (in ADC channels)" <<
 endl
              << "lower: lower limit to end inl testing (in ADC channels)" << e
ndl
              << "low_voltage: lowest voltage to calculate best fit line "
              << endl;
       exit(1);
   }
```

```
        int deviceNumber, chan, num_voltage_points, upper, lower, low_voltage;
        int pipe, increment, amuaddress;
        double r = 0, voltage_increment;
        char filename[80];

        FILE *fp;

        // Initializations
        point = 0;
        J1_data = 0x2000;
        J3_data = 0;
        chan = 0;
        deviceNumber = 1; //selects appropriate AIMID card
    amuaddress = 8;   //selects amu address to be tested

        //Input Parameters
        low_voltage = atoi( argv[6] );
        lower = atoi( argv[5] );
        upper = atoi( argv[4] );
        increment = atoi( argv[3] );
        NumofPoints_per_voltage = atoi(argv[2]);
        chip = atoi(argv[1]);

        // Save raw data to disk
        sprintf( filename, "c:\\perform\\linprgs\\data\\dc%d.dat", chip );
        fp = fopen(filename, "w");

        fprintf( fp, "Chip #: %d\n", chip );
        fprintf( fp, "# Points: %d", NumofPoints_per_voltage );


        //AI-MIO-16x configure
        AO_Configure( deviceNumber, 0, 1, 0, 5, 0 );

        //Pb chip test setup
        outpw( io_base_address + J1lower_byte, 0x2000 );
        outpw( io_base_address + J1lower_byte, reset );

        //defines number points that will be taken over the voltage range 0-5v
        num_voltage_points = 5*increment;

        for( voltage_level = 0; voltage_level < num_voltage_points ;
                        voltage_level ++)
        {
                voltage_increment = 5.0/num_voltage_points; //Determines voltage

                                                        //increment

                //Determines voltage, sums the voltage, and sums the square of
                //the voltage for the Least Squares function (which is the x axis
    of
                //the best fit line
                voltage = voltage_level * voltage_increment;
                sumX += voltage;
                sumX2 += pow( sumX, 2);


                //store_voltage[voltage_level] = voltage;

                status = AO_VWrite( deviceNumber, 0, voltage ); //Write voltage

                while( r < 50000)  //Allows DAC settling time (also a RC timecons
    tant)
```



```
                {
                        r++;
                }
                r=0;

                if( status != 0)
                {
                        cout << "Error in writting voltage!" << endl;
                        cout << "AI-MIO Error Code: %d" << status << endl;
                        exit;
                }

                cout << "Voltage: " << voltage << endl;

                //initialize point so that always store data starting at [0][0]
                point = 0;
                while( point < NumofPoints_per_voltage )
                {

                        write_amu( J1_data ); //Write to AMU

                        read_amu( J1_data, J3_data, amuaddress ); //Read AMU to A
    DC

                        for( pipe=0; pipe<16; pipe++)
                        {
                                ADCresults[point][pipe] = amuInputData[0][pipe];
                        }
                        point ++;
                }
                point = 0;


            //Run Tests on a single Data point
            Variance( NumofPoints_per_voltage, voltage_level, num_voltage_points )
    ;

                fprintf( fp, "\nVoltage: %f\n", voltage );
                for( pipe = 0; pipe < 16; pipe ++ )
                {
                        fprintf( fp, "Pipe: %d: ", pipe );
                        for( point = 0; point < NumofPoints_per_voltage; point ++
     )
                        {
                                fprintf( fp, "%d ", ADCresults[point][pipe] );
                        }
                        fprintf( fp, "\n");
                }
            fprintf( fp, "\n");
       }

        fclose( fp );
        least_squares( chip, num_voltage_points, NumofPoints_per_voltage, upper,
                                                lower, low_voltage );
}


void Variance( int NumofPoints_per_voltage, int voltage_level,
                                        int num_voltage_points )
{
        int
                        n = NumofPoints_per_voltage, num,
```

```
                        pipe;

        double
                        vsumX[16] = {0},
                        vsumX2[16] = {0},
                        vpipemean[16] = {0},
                        vmeanX2[16] = {0},
                        vmeanXsq[16] = {0},
                        vpipestd[16] = {0},
                        vpipevar[16] = {0};


    char filename[80];

        FILE *fpipemn, *fpipermn, *fpipevar, *fpipestd;

        //File pointer initializations
        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.mn", chip );
        if( voltage_level == 0 )
        {
                fpipemn = fopen(filename, "w");
                fprintf( fpipemn , "Chip #: %d\n", chip );
                fprintf( fpipemn, "\n# Voltages: %d", num_voltage_points );
                fprintf( fpipemn, "\n# Points per Voltage: %d", NumofPoints_per_v
oltage );
            fprintf( fpipemn, "\n     , 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15,");
        }
        else
        {
                fpipemn = fopen(filename, "a" );
        }

      , //prepares a file for least squares function
        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.rmn", chip );
        if( voltage_level == 0 )
        {
                fpipermn = fopen(filename, "w");
                fprintf( fpipermn, "%d, %d, %d,", chip, NumofPoints_per_voltage,
                                        num_voltage_points );

        }
        else
        {
                fpipermn = fopen(filename, "a" );
        }


        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.var", chip );
        if( voltage_level == 0 )
        {
                fpipevar = fopen(filename, "w");
                fprintf( fpipevar, "Chip #: %d\n", chip );
                fprintf( fpipevar, "\n# Voltages: %d", num_voltage_points );
                fprintf( fpipevar, "\n# Points per Voltage: %d", NumofPoints_per_
voltage );
            fprintf( fpipevar, "\n    , 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15");
        }
        else
        {
                fpipevar = fopen(filename, "a" );
        }

        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.std", chip );
        if( voltage_level == 0 )
        {
```



```
                fpipestd = fopen(filename, "w");
                fprintf( fpipestd, "Chip #: %d\n", chip );
                fprintf( fpipestd, "\n# Voltages: %d", num_voltage_points );
                fprintf( fpipestd, "\n# Points per Voltage: %d", NumofPoints_per_
voltage );
            fprintf( fpipestd, "\n    , 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15");
        }
        else
        {
                fpipestd = fopen(filename, "a" );
        }

    //Do Calculations for each Data point
        /*
          sumX: sum of each column (16 columns for each data point)
          sumX2: (sumX)^2
          mean: is average of each pipe
          meanXsq: is square of the mean
          meanX2: is mean squared
          variance(AKA sigma^2)=meansquared squareofthemean
          stdev(AKA sigma)=squareroot of sigma^2
        */

        //Columns used for calculations and stored in fpipemn & fpipevar
        for( pipe = 0; pipe < 16; pipe ++ )
        {



                for( num = 0; num < NumofPoints_per_voltage; num ++ )
                {
                        vsumX[pipe]+=ADCresults[num][pipe];

                        vsumX2[pipe]+=pow( ADCresults[num][pipe], 2);
                }

                vpipemean[pipe] = vsumX[pipe]/n;

                vmeanX2[pipe] = vsumX2[pipe]/n;

                vmeanXsq[pipe] = pow( vpipemean[pipe], 2);

                vpipevar[pipe] = vmeanX2[pipe] - vmeanXsq[pipe];

                vpipestd[pipe] = sqrt(vpipevar[pipe]);


        }

    //Store Mean for each Voltage
    fprintf( fpipemn, "\n%4.2f, ", voltage );
    fprintf( fpipermn, "\n%4.2f, ", voltage );
    for( pipe = 0; pipe < 16; pipe ++ )
    {
            fprintf( fpipemn, "%4.2f, ", vpipemean[pipe]);
            fprintf( fpipermn, "%4.2f, ", vpipemean[pipe]);
    }

        //Store Variance for each Voltage
        fprintf( fpipevar, "\nV: %4.2f, ", voltage );
    for( pipe = 0; pipe < 16; pipe ++ )
    {
        fprintf( fpipevar, "%4.2f, ", vpipevar[pipe]);
    }

        //Store Standard Deviation for each Voltage
```

```
        fprintf( fpipestd, "\n%4.2f, ", voltage );
    for( pipe = 0; pipe < 16; pipe ++ )
    {
        fprintf( fpipestd, "%4.2f, ", vpipestd[pipe]);
    }

    fclose( fpipestd );
    fclose( fpipevar );
    fclose( fpipemn );
    fclose( fpipermn );

}
```

```
/*
   least.CPP
   WRITTEN BY:     MICHAEL WAMAL
   DATE:  24 May 95 - 16 Jun 95
   Ver. 1.0
*/

/*************************************************************************/

/*
   Function Name: least_squares

   Requires:  int chip, int num_of_voltage_increments,
                                int NumofPoints_per_voltage, int upper, int low
er,
                                int low_voltage
              where - chip: is the chip ID #
                                num_of_voltage_increments: is the # of times th
e voltage
                                        will be incremented
                                NumofPoints_per_voltage: is the # of samples to
 take at
                                        each voltage
                                upper: is the upper limit at which the data for
 the best
                                        fit line is to be calculated
                                lower: is the lower limit at which the data for
 the best
                                        fit line is to be calculated
                                low_voltage: is voltage at which to start using
 data for best
                                        fit line.  Sort of simular to the
 upper limit but used
                                        as voltage instead of an ADC coun
ter.

   Returns: void

   Purpose: Calculate the Best fit line parameter from the data in memory
   created by lin3.cpp.  The best fit line is calculated and the result is
        used to calculate the INL for the DUT.  The results are stored to disk
        under various file extensions for each device.
*/

/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>

#define MaxWords 50  //Defines the max number of words in array for parsing
#define MaxBuffer 150 //Defines # of char's that are receive at one time from
                                                        //from the fgets() funct
ion.

double _far total_ini[400][17] = {0};
double _far mean_data[400][17] = {0};
double _far store_voltage[400] = {0};
double _far y[400][16] = {0};

void Parse_data( char *, int *, char ** );
```



```
void least_squares( int chip, int num_of_voltage_increments,
                                                        int NumofPoints_per_voltage, in
t upper, int lower,
                                                        int low_voltage )
{
        char    *File_num[MaxWords],
                   inputbuffer[MaxBuffer] = {0};

        int Count;

        int header_info[3], notwanted_data = 0, trigger = 0;

  char filename[80];

        int word_count, increment = 0, pipe;

        FILE *fpi;


                int
                        n = 16,
                        sample;

        double current_value = 0, maximum_value = 0;

        double
                        slopeterm1[16] = {0}, slopeterm2[16] = {0},
                        yinter_term1[16] = {0}, yinter_term2[16] = {0},
                        slope[16] = {0}, y_intercept[16] = {0},
                        sumX = 0,
                        sumY[16] = {0},
                        sumX2 = 0,
                        sumXY[16] = {0},
                        term3[16] = {0},
                        term4[16] = {0},
         ini[16] = {0},
         raw_ini[16] = {0};

        //File pointer initializations
        FILE *fpipeini, *fpiperawini, *ftotalini, *fpipelin, *fpipemb;

        //Setup files
        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.lin", chip );
        fpipelin = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.ini", chip );
        fpipeini = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\linprgs\\data\\c%dtotal.ini", chip );
        ftotalini = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.rni", chip );
        fpiperawini = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\linprgs\\data\\c%d.mb", chip );
        fpipemb = fopen(filename, "w");

        fprintf( fpipelin , "Chip #: %d Lowest Voltage: %d\n", chip, low_voltage
);
        fprintf( fpipelin, "# Points: %d\n", NumofPoints_per_voltage );

        fprintf( fpipeini , "Chip #: %d Lowest Voltage: %d\n", chip, low_voltage
);
        fprintf( fpipeini, "# Points: %d\n", NumofPoints_per_voltage );
```

```
        fprintf( ftotalinl , "Chip #: %d Lowest Voltage: %d\n", chip, low_voltage
 );
        fprintf( ftotalinl, "# Points: %d\n", NumofPoints_per_voltage );
        fprintf( fpiperawinl , "Chip #: %d Lowest Voltage: %d\n", chip, low_volta
ge );
        fprintf( fpiperawinl, "# Points: %d\n", NumofPoints_per_voltage );
        fprintf( fpipemb , "Chip #: %d Lowest Voltage: %d\n", chip, low_voltage )
;
        fprintf( fpipemb, "# Points: %d\n", NumofPoints_per_voltage );
        sprintf( filename, "c:\\perform\\lisprgs\\data\\c%d.rmn", chip );
        //Get Data from file
        if( (   fpl = fopen(filename, "r") ) != NULL )
        {
                while( (fgets(inputbuffer, MaxBuffer, fpl) != NULL) )
                {
                        //Seperates data into blocks or words to an array pointed
                        //to by an array of pointers called File_num.  The number
 of the
                        //words in the array is determined by MaxWords constant.
 The number
                        //of characters loaded into 'fgets' each pass is determin
ed by
                        //MaxBuffer constant.  Count is the number of words in th
e array
                        //pointed to by File_num.
                        Parse_data( inputbuffer, &Count, &(File_num[0]) );
                        //This allows the header info to be pulled into an array
 called
                        //header_info
                        if( Count == 3 )
                        {
                                for( word_count = 0; word_count < Count; word_cou
nt ++ )
                                {
                                        header_info[word_count] = atoi(File_num[w
ord_count]);
                                }
                        }
                        //This allows the mean data to be stored into an array ca
lled
                        //mean_data
                        if( Count > 3)
                        {
                                for( word_count = 0; word_count < Count; word_cou
nt ++ )
                                {
                                        mean_data[increment][word_count] = atof(F
ile_num[word_count]);
                                }
                        }
                        if( Count > 3 )
                        {
                                increment ++;
                        }
                }
        }
        fclose( fpl );
   for( increment = 0; increment < (num_of_voltage_increments); increment ++ )
```



```
   {
        if( mean_data[increment][0] > low_voltage )
        {
                if( notwanted_data == 0 && trigger == 0)
                {
                        notwanted_data = increment;
                        trigger = 1;
                }
                store_voltage[(increment-notwanted_data)] =
                                mean_data[increment][0];
                sumX += mean_data[increment][0];
                sumX2 += pow( mean_data[increment][0], 2);
                for( pipe = 1; pipe < 17; pipe ++ )
                {
                        //Note: must subtract 1 from the index for sumY a
nd sumXY because
                        //the mean_data matrix is [large][17] and sumY an
d sumXY are [16].
                        //Only want the last 16 columns of mean_data matr
ix and want them
                        //in the first 16 columns of sumX and sumXY there
fore subtract one
                        //from their indexes.
                        sumY[pipe-1] += mean_data[increment][pipe];
                        sumXY[pipe-1] += mean_data[increment][0]*mean_dat
a[increment][pipe];
                }
        }
   }


    //Calculate terms for Best Fit Line Equations
                for( pipe = 0; pipe < 16; pipe ++ )
                {


          slopeterm1[pipe] = (num_of_voltage_increments-notwanted_data)
                                                                *sumXY[pipe];
                  slopeterm2[pipe] = sumX*sumY[pipe];
                  yinter_term1[pipe] = sumX2*sumY[pipe];
                  yinter_term2[pipe] = sumXY[pipe]*sumX;
                  term3[pipe] = (num_of_voltage_increments-notwanted_data)*sumX2;
                  term4[pipe] = pow( sumX, 2);
                        slope[pipe] = (slopeterm1[pipe] - slopeterm2[pipe])/
                                                        (term3[pipe] - term4[pi
pe]);
                        y_intercept[pipe] = (yinter_term1[pipe] - yinter_term2[pi
pe])/
                                                        (term3[pipe] - term4[pi
pe]);
                }
                //Calculate the Best Fit line from the slope and y intercept valu
es
                for( pipe = 0; pipe < 16; pipe ++ )
                {
                        for( sample = 0; sample < (num_of_voltage_increments-notu
```

```
anted_data);
                                        sample ++ )
                                {
                                        y[sample][pipe] = slope[pipe] * store_voltage[sam
ple]
                                                                                + y_inte
rcept[pipe];
                                }
                        }

                        //Calculate the absolute INL per pipe
                        for( pipe = 0; pipe < 16; pipe ++ )
                        {
                                for( sample = 0; sample < num_of_voltage_increments-notwa
nted_data;
                                        sample ++ )
                                {
                                        //This if statement cuts off the end point data f
rom the INL
                                        //test and only checks the linear section
                                        if( mean_data[(sample+notwanted_data)][pipe+1] <
upper &&
                                                mean_data[(sample+notwanted_data)][pipe+
1] > lower )
                                        {
                                                current_value = fabs( mean_data[(sample +
notwanted_data)][pipe+1] -
                                y[sample][pipe] );
                                                if( maximum_value < current_value)
                                                {
                                                        inl[pipe] = maximum_value = curre
nt_value;
                                                }
                                        }
                                }
                           current_value = maximum_value = 0;
                        }


                        //Calculate the raw INL per pipe
                        for( pipe = 0; pipe < 16; pipe ++ )
                        {
                                for( sample = 0; sample < num_of_voltage_increments-notwa
nted_data;
                                        sample ++ )
                                {
                                        //This if statement cuts off the end point data f
rom the INL
                                        //test and only checks the linear section
                                        if( mean_data[(sample+notwanted_data)][pipe+1] <
upper &&
                                                mean_data[(sample+notwanted_data)][pipe+
1] > lower )
                                        {
                                                total_ini[sample][0] = mean_data[(sample
+ notwanted_data)][0];
                                                total_ini[sample][pipe+1] =
                                                mean_data[(sample + notwanted_data)][pipe
+1] - y[sample][pipe];
                                                current_value = fabs( mean_data[(sample +
notwanted_data)][pipe+1] -
                                y[sample][pipe] );
                                                if( maximum_value < current_value)
                                                {
```



```
                                                        maximum_value = current_value;
                                                        raw_ini[pipe] =
                                                        mean_data[(sample + notwanted_dat
a)][pipe+1] - y[sample][pipe];
                                                }
                                        }
                                }
                           current_value = maximum_value = 0;
                        }



                //Store INL Data to file

        fprintf( ftotalini, "Percent of FSR:\n" );
                for( sample = 0; sample < num_of_voltage_increments-notwanted_dat
a;
                        sample ++ )
                {
                        fprintf( ftotalini, "%4.2f, ", total_ini[sample][0] );
                        for( pipe = 1; pipe < 17; pipe ++ )
                        {

                fprintf( ftotalini, "%4.2f, ", (total_ini[sample][pipe]/1024)*100 );
                }
          fprintf( ftotalini, "\n" );
        }
        fprintf( ftotalini, "\n\n" );


        fprintf( ftotalini, "ADC Counts:\n" );
                for( sample = 0; sample < num_of_voltage_increments-notwanted_dat
a;
                        sample ++ )
                {
                        for( pipe = 0; pipe < 17; pipe ++ )
                        {

                fprintf( ftotalini, "%4.2f, ", total_ini[sample][pipe] );
                }
          fprintf( ftotalini, "\n" );
        }



                for( pipe = 0; pipe < 16; pipe ++ )
                        {
                   fprintf( fpipeini, "%4.2f, ", ini[pipe] );
                }

                for( pipe = 0; pipe < 16; pipe ++ )
                        {
                   fprintf( fpiperawini, "%4.2f, ", raw_ini[pipe] );
                }

                        //Store Best Fit line Data
                        fprintf( fpipelin, "\n");
                        for( pipe = 0; pipe < 16; pipe ++ )
                        {
                                if(pipe == 0 )
```

```
                {
                        fprintf( fpipelin, "Voltage:, " );
                        for( sample = 0; sample < num_of_voltage_incremen
ts-notwanted_data;
                                sample ++ )
                    {
                      fprintf( fpipelin, "%4.2f, ",
                                store_voltage[sample]);
                        }
                      fprintf( fpipelin, "\n" );
                    }
                    fprintf( fpipelin, "Pipe: %d, ", pipe);
                    for( sample = 0; sample < num_of_voltage_increments-notwa
nted_data;
                        sample ++ )
                    {
                    fprintf( fpipelin, "%4.2f, ", y[sample][pipe]);
                }
               fprintf( fpipelin, "\n" );
          }

          //Store Slope and Y-Intercept of Best fit line
          fprintf( fpipemb, "Slope per pipe (0 - 15)\n" );
          for( pipe = 0; pipe < 16; pipe ++ )
          {
                fprintf( fpipemb, "%4.2f, ", slope[pipe] );
          }

          fprintf( fpipemb, "\n\nY-Intercept per pipe (0 - 15)\n" );
          for( pipe = 0; pipe < 16; pipe ++ )
          {
                fprintf( fpipemb, "%4.2f, ", y_intercept[pipe] );
          }

          fclose( fpipemb );
          fclose( fpipeini );
          fclose( fpipelin );


}



/*************************************************************************/

/*
        Function Name:  Parse_data
        Requires:       pointer to a predefined character array (buffer)
                                                pointer to integer to count the
number of words stored
                                                        in the word array
                                                pointer to a pointer to a charac
ter array with desired
                                                                        size of p
arsed words
        Definitions:
                #define MaxWords 50  Defines the max number of words in array for
                                                                        parsing
                #define MaxBuffer 150 Defines # of ch[illegible] that are receive at one
                                                                        time fro
```



```
m from the fgets() function.

        Declarations:

                functions -     void Parse_data( char *, int *, char ** );

                Variables -     char    *File_num[MaxWords],
                                                                        inputbuffer[Ma
xBuffer] = {0};
                                                        int Count;

        Usage:

                while( (fgets(inputbuffer, MaxBuffer, fp) != NULL) )
                {
                        Parse_data( inputbuffer, &Count, &(File_num[0]) );
                        .
                        .
                        .

        Returns: void
        Purpose: seperates a buffer full (this case one line) of data into a
                                'word' array that is predefined in the calling pr
g by MaxWord.
                                The buffer length is determined by MaxBuffer cons
tant.
                                Once the buffer is seperated, the word array can
be converted
                                an 'int' or 'double' or can be used as a 'char' (
which is the
                                way the word array is defined).
*/
/*************************************************************************/



void Parse_data( char *buf, int *Count, char **Word)
{
        *Word = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (i.e. \0) character in the contents
 of the current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }
                /*
```

```
                        Stores wanted characters in 'word' array which uses count
s as its
                        indexing parameter
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        Word[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;
                }
        }
        *buf = '\0';
}
```

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = SIGDNL
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = /d_DEBUG
R_RCDEFINES = /dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\PERFORM\DNL\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = AMUINT3.C
FIRSTCPP = SIGDNL.CPP
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AH /Od /D "_DEBUG" /D "_DOS" /FR /Fd"SIGDNL.
PDB"
CFLAGS_R_DEXE = /nologo /Gs /G2 /W3 /AH /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /NOI /STACK:40000 /ONERROR:NOEXE /CO
LFLAGS_R_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE
LIBS_D_DEXE = oldnames llibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = SIGDNL.SBR \
               AMUINT3.SBR


SIGDNL_DEP = c:\perform\dnl\nidaq.h \
        c:\perform\dnl\nidaqerr.h \
        c:\perform\dnl\nidaqcns.h \
        c:\perform\dnl\constant.h


AMUINT3_DEP = c:\perform\dnl\constant.h


all:    $(PROJ).EXE $(PROJ).BSC
```



```
SIGDNL.OBJ:     SIGDNL.CPP $(SIGDNL_DEP)
        $(CPP) $(CFLAGS) $(CPPCREATEPCHFLAG) /c SIGDNL.CPP

AMUINT3.OBJ:    AMUINT3.C $(AMUINT3_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c AMUINT3.C

$(PROJ).EXE::   SIGDNL.OBJ AMUINT3.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
SIGDNL.OBJ +
AMUINT3.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
c:\nidaqwin\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
                /******** Constants ********/

#define upper_limit     0x03ff
#define tac             0
#define amu             1
#define amu_voltage_1   0x2000 /* sent to connector J1 */
#define amu_voltage_3   0xdfff /* sent to connector J1 */
#define dataclr         0x0000
#define data_mask       0x03ff
#define dont_care       0xffff

#define io_base_address 0x02a0
#define J1lower_byte    0x0000
#define J1upper_byte    0x0001
#define J2lower_byte    0x0000
#define J2upper_byte    0x0001
#define J3lower_byte    0x0002
#define J3upper_byte    0x0003

/* mach code masks */

#define reset           0x8000 /* sent to connector J1 */
#define read            0x0800 /* sent to connector J1 */
#define write           0x0400 /* sent to connector J1 */
#define adc_ramp        0x0200 /* sent to connector J1 */
#define adc_trig        0x0200 /* sent to connector J1 */
#define clr_adc_trig    0xfdff /* sent to connector J1 */
#define adc_oe          0x1000 /* sent to connector J1 */
#define tac_strt        0x8000 /* sent to connector J3 */


#define pulse_clr_mask  0xfe1f /* sent to connector J1 */

#define pulse_20ns      0x0000 /* sent to connector J1 */
#define pulse_40ns      0x0020 /* sent to connector J1 */
#define pulse_60ns      0x0040 /* sent to connector J1 */
#define pulse_80ns      0x0060 /* sent to connector J1 */
#define pulse_100ns     0x0080 /* sent to connector J1 */
#define pulse_120ns     0x00a0 /* sent to connector J1 */
#define pulse_140ns     0x00c0 /* sent to connector J1 */
#define pulse_160ns     0x00e0 /* sent to connector J1 */
#define pulse_180ns     0x0100 /* sent to connector J1 */
#define pulse_200ns     0x0120 /* sent to connector J1 */

#define chan0           0x0080 /* sent to connector J3 */
#define chan1           0x0100 /* sent to connector J3 */
#define chan2           0x0200 /* sent to connector J3 */
#define chan3           0x0400 /* sent to connector J3 */
#define chan4           0x0800 /* sent to connector J3 */
#define chan5           0x1000 /* sent to connector J3 */
#define chan6           0x2000 /* sent to connector J3 */
#define chan7           0x4000 /* sent to connector J3 */

#define test0           0x0000 /* sent to connector J3 */
#define test1           0x0010 /* sent to connector J3 */
#define test2           0x0020 /* sent to connector J3 */
#define test3           0x0030 /* sent to connector J3 */
#define test4           0x0040 /* sent to connector J3 */
#define test5           0x0050 /* sent to connector J3 */
#define test6           0x0060 /* sent to connector J3 */
#define test7           0x0070 /* sent to connector J3 */
```

```
/*
   sigdnl.CPP
   WRITTEN BY.     MICHAEL WUMAC
   DATE:  6 Jun 95 -
*/


/*****************************************************************************/

/*
   Function Name: dnl.exe
   Requires:  command line: 'chip number #points MaxVoltage'
   Returns: void
   Purpose: This prg uses a signal generator.  It provides an
   input voltage for the Pb Glass chip test fixture. The prg provides all
   hand shake signals needed to interface for a write, read, and adc
   conversion.  The prg outputs a ramp voltage in steps determined
   by the increment at the command line.  Each step is converted a number .
   of times determined by the command line. Tests cell #8 for all 16 pipes.

*/

/*****************************************************************************/



/*****************************************************************************/
/********                      Headers                            ********/
/*****************************************************************************/

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include <math.h>
#include <time.h>

//current directory header file
#include "nidaq.h"      /* function definitions */
#include "nidaqerr.h"   /* error code definitions */
#include "nidaqcns.h"   /* NI-DAQ constants definitions */
#include "constants.h"

/*****************************************************************************/
/********                      Defines                            ********/
/*****************************************************************************/


/*****************************************************************************/
/********                      Structures                         ********/
/*****************************************************************************/


/*****************************************************************************/
/********                      Global Variables                   ********/
/*****************************************************************************/

unsigned long _huge DNLresults[16][1024];

/*****************************************************************************/
/********                      Functions                          ********/
```



```
/*****************************************************************************/

extern "C"
{
                int     J1_data,
                        J3_data,
                        amuInputData[16][16];
                void read_amu( int, int, int);
                void write_amu( int );
                void print_J2( void );
}

/*****************************************************************************/
/********                      Code                               ********/
/*****************************************************************************/


void main( int argc, char **argv )
{

        //Note: the limit in the increment is due to least
        //squares allocating the mean_data matrix
   if( argc != 4 )
   {
        cout << "Usage: chipnum SamplesPerChan freq" << endl;
      cout << "chipnum: is an integer value for chip id" << endl
                              << "SamplesPerChan: desired # of samples per channels"

                              << "freq: input frequency of ramp from sig gen"
           << endl;
      exit(1);
   }

   char backup_time[20], start_time[20], end_time[20];
        int pipe, amuaddress, chip, chan, freq;
        long loop=0, counter = 0, samp, SamplesPerChan;

        long ramp_count = 0, last_ramp = 0; //determines how many ramps have elap
sed


        char filename[80];

        FILE *fp;

        clock_t start, finish, tmp, backup = 0;
        double duration, last_reported;


        // initializations
        //this will write to screen every ramp_multiple*9.5sec
        J1_data = 0x2000;
        J3_data = 0;
   amuaddress = 8;   //selects amu address to be tested

        //input Parameters
        freq = atoi(argv[3]);
        samp = atoi(argv[2]);
        chip = atoi(argv[1]);

        SamplesPerChan = 1024*samp;

        //Pb chip test setup
        outpw( io_base_address + J1lower_byte, 0x2000 );
        outpw( io_base_address + J1lower_byte, reset );
```

```
        start = clock();

        _strtime( start_time );

        while( loop < SamplesPerChan )
        {
                loop ++;
                counter ++;
                write_amu( J1_data ); //Write to AMU

                read_amu( J1_data, J3_data, amuaddress ); //Read AMU to ADC

                for( pipe=7; pipe<8; pipe++)
                {
                        DNLresults[pipe][amuInputData[0][pipe]] ++;
                }
                if( counter == SamplesPerChan/10 )
                {
                   finish = clock();
                   duration = (double)(finish - start)/CLOCKS_PER_SEC;
                   cout << "Ideal # of samples/chan: " << loop/1024 << endl;
                   cout << "Elapsed Time: " << duration << endl;
                        counter = 0;
                }

        }


        _strtime( end_time );

    finish = clock();
    duration = (double)(finish - start)/CLOCKS_PER_SEC;

        //Save channel results to disk
        sprintf( filename, "c:\\perform\\dnl\\data\\sig%d.dat", chip );
        fp = fopen(filename, "w");
        fprintf( fp, "Chip #: %d\n", chip );
        fprintf( fp, "Time Program ended: %s\n", end_time );
    fprintf( fp, "Total number of # of samples: %lu\n", loop );
    fprintf( fp, "Ideal number of # of samples/chan: %lu\n", loop/1024 );
    fprintf( fp, "Input Frequency: %d\n", freq );
    fprintf( fp, "Duration: %f in seconds\n", duration );
    fprintf( fp, "Duration: %f in minutes\n", duration/60 );
    fprintf( fp, "Duration: %f in hours\n", duration/3600 );
//      for( pipe = 0; pipe < 16; pipe ++ )
//      {
                fprintf( fp,"7, "); //%d, ", pipe );
//      }
        fprintf( fp, "\n");

        for( chan = 0; chan < 1024; chan ++ )
        {
                for( pipe = 7; pipe < 8; pipe ++ )
                {
                        fprintf( fp, "%lu, ", DNLresults[pipe][chan] );
                }
                fprintf( fp, "\n" );
        }
        fclose( fp );

    cout << "Approximate # of ramps: " << duration*freq << endl;
    cout << "Total number of # of samples: " << loop << endl;
    cout << "Ideal number of # of samples/chan: " << loop/1024 << endl;
    cout << "Time per Loop: " << duration/loop << " in seconds" << endl;
```



```
        cout << endl << "Start Time: " << start_time << endl;
        cout << "End Time: " << end_time << endl;
    cout << "Elapsed CPU Time: " << duration << " seconds" << endl;
        cout << "                  or " << duration/60 << " minutes" << endl;
        cout << "                  or " << duration/3600 << " hours" << endl;
}
```

```
/*
  amuint.c
  Modified functions from original Chip2.exe code to do performance
  testing on the amu
  Date: May 95
  Written by: Michael Womac
*/


/*****************************************************************************/
/********                        Headers                            ********/
/*****************************************************************************/

#include <conio.h>
#include <graph.h>
#include <stdlib.h>
#include <stdio.h>
#include <graph.h>
#include "constant.h"

/*****************************************************************************/
/********                        Variables                          ********/
/*****************************************************************************/

//global
extern int  J1_data,
                        J3_data,
                        amuInputData[1][16];

int fsc;
int J1data, J1dat;
int J3data;
int amu_read;


/*****************************************************************************/
/********                        Functions                          ********/
/*****************************************************************************/

void trigger_adc( int );
void check_fsc( void );
void read_amu( int, int, int);
void write_amu( int );
void print_J2( void );

/*****************************************************************************/

/*
   Function Name: read_amu
   Requires: 4 int types, J1 & J3 data and row num's for data to be printed
   Returns:  void
   Purpose:  reads the 16 amu addresses, triggers the adc, reads out columns
             of the adc for each amu address.
*/

/*****************************************************************************/


void read_amu( int J1data, int J3data, int amuaddress )
{
        int amu_pipe;

      //Sends desired AMU address to ATA chip
              outpw( io_base_address + J3lower_byte, amuaddress );
```



```
        //write_amu( J1data );

                outpw( io_base_address + J1lower_byte,
                        J1data + read );


                trigger_adc( 0 );


                check_fsc();

        for( amu_pipe = 7; amu_pipe < 8; amu_pipe ++ )
        {
      //reads the amu adc's 8 - 23 (or amu pipes)
                outpw( io_base_address + J1lower_byte,
                                (amu_pipe*8) + adc_oe );

           amuInputData[ 0 ][ amu_pipe ] =
                                (inpw( io_base_address + J2lower_byte )& data_ma
sk );
        }
        outpw( io_base_address + J1lower_byte,
                        datoclr );

}


/*****************************************************************************/


/*
   Function Name: write_amu
   Requires: int type, J1 data
   Returns:  void
   Purpose:  to set write command to the connector J1 and then clear it
*/

/*****************************************************************************/


void write_amu( int J1data )
{

        outpw( io_base_address + J1lower_byte,
                J1data );                               /* sets up voltage prior to write
 */

        outpw( io_base_address + J1lower_byte,
                J1data + write );
        outpw( io_base_address + J1lower_byte,
                J1data );                                       /* clears write command */

}


/*****************************************************************************/

/*
   Function Name: trigger_adc
        Requires:  int to determine whether or not the amu is being read so read
                                signal does not get reset.
   Returns:   void
```

```
   Purpose: sends adc trigger signal to the J1 connector for the chip under
            test.
*/

/*************************************************************************/

void trigger_adc( int amu_read )
{
        if( amu_read == 0 )  /* trigger sent during amu read cycle */
        {
                outpw( io_base_address + J1lower_byte,
                       adc_oe + adc_trig + read);
                outpw( io_base_address + J1lower_byte,
                       adc_oe + read );
        }
/*      if( amu_read == 1 )*/  /* trigger sent during tac cycle */
/*      {
                outpw( io_base_address + J1lower_byte,
                       adc_oe + adc_trig );
                outpw( io_base_address + J1lower_byte,
               adc_oe );
        }*/
}


/*************************************************************************/

/*
   Function Name: check_fsc
   Requires:  void
   Returns:   void
   Purpose:   to check the fsc bit on the J2 connector (bit 16) and wait
              until it is active.
*/

/*************************************************************************/

void check_fsc( void )
{
        do
        {
                fsc = (inpw(io_base_address + J2lower_byte)
                       & 0x8000);
        } while( fsc );
}
```

APPENDIX D

INL.MAK AND ASSOCIATED PROGRAMS

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = INL
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = /d_DEBUG
R_RCDEFINES = /dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\PERFORM\TAC\INL\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = TACPERF.C
FIRSTCPP = INL.CPP
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AM /Od /D "_DEBUG" /D "_DOS" /FR /Fd"INL.PDB
"
CFLAGS_R_DEXE = /nologo /Gs /G2 /W3 /AM /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /NOI /STACK:30000 /ONERROR:NOEXE /CO
LFLAGS_R_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE
LIBS_D_DEXE = oldnames llibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = INL.SBR \
               TACPERF.SBR \
               TLEAST.SBR


INL_DEP = c:\msvc\include\iostream.h \
        c:\msvc\include\ios.h \
        c:\msvc\include\streamb.h \
        c:\msvc\include\istream.h \
        c:\msvc\include\ostream.h \
        c:\msvc\include\stdio.h \
        c:\msvc\include\stdlib.h \
        c:\msvc\include\conio.h \
        c:\msvc\include\malloc.h \
        c:\msvc\include\graph.h \
```



```
        c:\msvc\include\math.h \
        c:\perform\tac\inl\constant.h


TACPERF_DEP = c:\msvc\include\stdio.h \
        c:\msvc\include\conio.h \
        c:\msvc\include\process.h \
        c:\perform\tac\inl\constant.h \
        c:\perform\tac\inl\variable.h


TLEAST_DEP = c:\msvc\include\stdio.h \
        c:\msvc\include\conio.h \
        c:\msvc\include\iostream.h \
        c:\msvc\include\ios.h \
        c:\msvc\include\streamb.h \
        c:\msvc\include\istream.h \
        c:\msvc\include\ostream.h \
        c:\msvc\include\math.h \
        c:\msvc\include\stdlib.h


all:    $(PROJ).EXE $(PROJ).BSC

INL.OBJ:        INL.CPP $(INL_DEP)
        $(CPP) $(CFLAGS) $(CPPCREATEPCHFLAG) /c INL.CPP

TACPERF.OBJ:    TACPERF.C $(TACPERF_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c TACPERF.C

TLEAST.OBJ:     TLEAST.CPP $(TLEAST_DEP)
        $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c TLEAST.CPP

$(PROJ).EXE::   INL.OBJ TACPERF.OBJ TLEAST.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
INL.OBJ +
TACPERF.OBJ +
TLEAST.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
/******** Constants ********/

#define upper_limit      0x03ff
#define amu_voltage_1    0x2000 /* sent to connector J1 */
#define amu_voltage_3    0xdfff /* sent to connector J1 */
#define dataclr          0x0000
#define data_mask        0x03ff
#define dont_care        0xffff

#define io_base_address 0x02a0
#define J1lower_byte     0x0000
#define J1upper_byte     0x0001
#define J2lower_byte     0x0000
#define J2upper_byte     0x0001
#define J3lower_byte     0x0002
#define J3upper_byte     0x0003

/* mach code masks */

#define reset            0x8000 /* sent to connector J1 */
#define adc_ramp         0x0200 /* sent to connector J1 */
#define adc_trig         0x0200 /* sent to connector J1 */
#define clr_adc_trig     0xfdff /* sent to connector J1 */
#define adc_oe           0x1000 /* sent to connector J1 */
#define tac_strt         0x8000 /* sent to connector J3 */


#define pulse_clr_mask   0xfe1f /* sent to connector J1 */

#define pulse_20ns       0x0000 /* sent to connector J1 */
#define pulse_40ns       0x0020 /* sent to connector J1 */
#define pulse_60ns       0x0040 /* sent to connector J1 */
#define pulse_80ns       0x0060 /* sent to connector J1 */
#define pulse_100ns      0x0080 /* sent to connector J1 */
#define pulse_120ns      0x00a0 /* sent to connector J1 */
#define pulse_140ns      0x00c0 /* sent to connector J1 */
#define pulse_160ns      0x00e0 /* sent to connector J1 */
#define pulse_180ns      0x0100 /* sent to connector J1 */
#define pulse_200ns      0x0120 /* sent to connector J1 */

#define chan0            0x0080 /* sent to connector J3 */
#define chan1            0x0100 /* sent to connector J3 */
#define chan2            0x0200 /* sent to connector J3 */
#define chan3            0x0400 /* sent to connector J3 */
#define chan4            0x0800 /* sent to connector J3 */
#define chan5            0x1000 /* sent to connector J3 */
#define chan6            0x2000 /* sent to connector J3 */
#define chan7            0x4000 /* sent to connector J3 */

#define test0            0x0000 /* sent to connector J3 */
#define test1            0x0010 /* sent to connector J3 */
#define test2            0x0020 /* sent to connector J3 */
#define test3            0x0030 /* sent to connector J3 */
#define test4            0x0040 /* sent to connector J3 */
#define test5            0x0050 /* sent to connector J3 */
#define test6            0x0060 /* sent to connector J3 */
#define test7            0x0070 /* sent to connector J3 */
```

```
int J1_data, J3_data;
int tac_data[8];
```

```
/*
   INL.CPP
   WRITTEN BY:     MICHAEL WOMAC
   DATE:  28 Jun 95
   Ver. 1.0
*/

/*************************************************************************/

/*
   Function Name: inl.exe
   Requires:
   Returns: void
   Purpose:

*/

/*************************************************************************/




/*************************************************************************/
/********                    Headers                          ********/
/*************************************************************************/

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include <math.h>

//current directory header file
#include "constant8.h"

/*************************************************************************/
/********                    Defines                          ********/
/*************************************************************************/


/*************************************************************************/
/********                    Structures                       ********/
/*************************************************************************/
/*************************************************************************/
/*************************************************************************/
/********                    Global Variables                 ********/
/*************************************************************************/


  int  far tac_results[8][2000];
  int chip;

/*************************************************************************/
/********                    Functions                        ********/
/*************************************************************************/

extern "C"
{
                int     J1_data,
                        J3_data,
                        tac_data[8];
                void  tac( int, int);
                void  read_adc( int );
```



```
}
void int( int, int, int, int, int );
void variance( int, int, int );

/*************************************************************************/
/********                    Code                             ********/
/*************************************************************************/


void main( int argc, char **argv )
{

   if( argc != 5 )
   {
        cout << "Usage: chip# #iterations upper lower" << endl;
        cout << "chip#: is the identification # of the chip" << endl;
                cout << "#iterations: # of times to test at each pulse" << endl;

                cout << "upper: upper limit for best fit line (in ADC channels)"
<< endl;
                cout << "lower: lower limit for best fit line (in ADC channels)"
<< endl;
        exit(1);
   }

    int iteration, chan, pulse, numof_iterations, upper, lower;
        char filename[80];

        FILE *fp;


        //input Parameters
        chip = atoi(argv[1]);
        numof_iterations = atoi(argv[2]);
        upper = atoi(argv[3]);
        lower = atoi(argv[4]);

        // Save raw data to disk
        sprintf( filename, "c:\\perform\\tac\\ini\\data\\tc%d.dat", chip );
        fp = fopen(filename, "w");

        fprintf( fp, "Chip #: %d\n", chip );
        fprintf( fp, "# of Pulses: 10           # of Iterations: %d\n", numof_ite
rations );

        for( pulse = 0; pulse < 10; pulse++ )
        {
                fprintf( fp, "Pulse #: %d\n", pulse );
                for( iteration = 0; iteration < numof_iterations; iteration ++ )
                {
                        for( chan = 0; chan < 8; chan ++ )
                        {
                                tac( chan, pulse );

                                fprintf( fp, "%d, ",
                                                tac_results[chan][iteration] = t
ac_data[chan] );
                        }
                        fprintf( fp, "\n" );
                }
                fprintf(fp, "\n\n" );

           //Run Tests on a single Data point
           variance( numof_iterations, iteration, pulse );
        }
```

```
        fclose( fp );
        inl( chip, 10, numof_iterations, upper, lower );

}


/*************************************************************************/

/*
   Function Name: Variance
   Requires:
                        Numof_IterationsPerSample
                        Iteration_number
                        SampleNumber
   Returns: void
   Purpose: To calculate the std, var, mean of the matrix tac_results which
                        is a global variable.
        Note:   This function can easily be modified to work with other prgs
                        by changing the matrix variable, the size of the summing
arrays,
                        editing the file parameters to the desired contexts

*/

/*************************************************************************/

void Variance( int Numof_IterationsPerSample, int Iteration_number,
                        int SampleNumber )
{

        int
                        n = Numof_IterationsPerSample, iteration, notcounted = 0,

                        chan;

        double
                vsumX2[8][10] = {0},
                v1sumX2[8][10] = {0},
                v2sumX2[8][10] = {0},
                vmean[8][10] = {0},
                vmeanX2[8][10] = {0},
                vmeanX2sq[8][10] = {0},
                vsumX[8][10] = {0},
                v1sumX[8][10] = {0},
                v2sumX[8][10] = {0},
                std[8][10] = {0},
                variance[8][10] = {0};


   char filename[80];

        FILE *fpvmean, *fpvariance, *fpstd;

        //File pointer initializations
        sprintf( filename, "c:\\perform\\tac\\inl\\data\\tc%d.mn", chip );
        if( SampleNumber == 0 )
        {
                fpvmean = fopen(filename, "w");
                fprintf( fpvmean , "\nChip #: %d", chip );
                fprintf( fpvmean, "\n# Pulses: 10" );
                fprintf( fpvmean, "\n# Iterations per Pulse: %d", Numof_Iteration
sPerSample );
                fprintf( fpvmean, "\n, 0, 1, 2, 3, 4, 5, 6, 7,");
        }
        else
```



```
        {
                fpvmean = fopen(filename, "a" );
        }


        sprintf( filename, "c:\\perform\\tac\\inl\\data\\tc%d.var", chip );
        if( SampleNumber == 0 )
        {
                fpvariance = fopen(filename, "w");
                fprintf( fpvariance, "\nChip #: %d", chip );
                fprintf( fpvariance, "\n# Pulses: 10" );
                fprintf( fpvariance, "\n# Iterations per Pulse: %d", Numof_Iterat
ionsPerSample );
                fprintf( fpvariance, "\n, 0, 1, 2, 3, 4, 5, 6, 7, ");
        }
        else
        {
                fpvariance = fopen(filename, "a" );
        }

        sprintf( filename, "c:\\perform\\tac\\inl\\data\\tc%d.std", chip );
        if( SampleNumber == 0 )
        {
                fpstd = fopen(filename, "w");
                fprintf( fpstd, "\nChip #: %d", chip );
                fprintf( fpstd, "\n# Pulses: 10" );
                fprintf( fpstd, "\n# Iterations per Pulse: %d", Numof_IterationsP
erSample );
                fprintf( fpstd, "\n, 0, 1, 2, 3, 4, 5, 6, 7, ");
        }
        else
        {
                fpstd = fopen(filename, "a" );
        }

  //Do Calculations for each Data point
        /*
          vsumX: sum of each column (16 columns for each data point)
          vsumX2: (vsumX)^2
          vmean: is average of each chan
          vmeanXsq: is square of the vmean
          vmeanX2: is vmean squared
          variance(AKA sigma^2)=meansquared-squareofthemean
          stdev(AKA sigma)=squareroot of sigma^2
        */

        //Columns used for calculations and stored in fpvmean & fpvariance
        for( chan = 0; chan < 8; chan ++ )
        {


                for( iteration = 0; iteration < Numof_IterationsPerSample;
                                iteration ++ )
                {
                        vsumX[chan][SampleNumber]+=tac_results[chan][iteration];

                        vsumX2[chan][SampleNumber]+=pow( tac_results[chan][iterat
ion], 2);
                }

                vmean[chan][SampleNumber] = vsumX[chan][SampleNumber]/n;

                vmeanX2[chan][SampleNumber] = vsumX2[chan][SampleNumber]/n;
```

```
                vmeanX2sq[chan][SampleNumber] = pow( vmean[chan][SampleNumber], 2
);

                variance[chan][SampleNumber] = vmeanX2[chan][SampleNumber]
                                                        - vmeanX
2sq[chan][SampleNumber];

                std[chan][SampleNumber] = sqrt(variance[chan][SampleNumber]);

        }


    for( chan = 0; chan < 8; chan ++ )
    {
                for( iteration = 0; iteration < Numof_IterationsPerSample;
                        iteration ++ )
                {
                        if( (tac_results[chan][iteration] > (vmean[chan][SampleNu
mber]+
                                6*std[chan][SampleNumber])) ||
                                (tac_results[chan][iteration] < (vmean[chan][Samp
leNumber]-
                                6*std[chan][SampleNumber]))  )
                        {
                                notcounted ++;
                        }
                        else
                        {
                v1sumX[chan][SampleNumber]+=tac_results[chan][iteration];

                                v1sumX2[chan][SampleNumber]+=
                                                        pow( tac_results[chan][it
eration], 2);
                        }

                }
                vmean[chan][SampleNumber] =
                                                v1sumX[chan][SampleNumber]/(n-not
counted);

                vmeanX2[chan][SampleNumber] =
                                                v1sumX2[chan][SampleNumber]/(n-no
tcounted);

                vmeanX2sq[chan][SampleNumber] = pow( vmean[chan][SampleNumber], 2
);

                variance[chan][SampleNumber] = vmeanX2[chan][SampleNumber]
                                                - vmeanX2sq[chan][SampleNumber];


                std[chan][SampleNumber] = sqrt(variance[chan][SampleNumber]);

                notcounted = 0;
        }

    for( chan = 0; chan < 8; chan ++ )
    {
                for( iteration = 0; iteration < Numof_IterationsPerSample;
                        iteration ++ )
                {
                        if( (tac_results[chan][iteration] > (vmean[chan][SampleNu
mber]+
                                6*std[chan][SampleNumber])) ||
                                (tac_results[chan][iteration] < (vmean[chan][Samp
leNumber]-
```



```
                                6*std[chan][SampleNumber]))  )
                        {
                                notcounted ++;
                        }
                        else
                        {
                v2sumX[chan][SampleNumber]+=tac_results[chan][iteration];

                                v2sumX2[chan][SampleNumber]+=
                                                        pow( tac_results[chan][it
eration], 2);
                        }

                }
                vmean[chan][SampleNumber] =
                                                v2sumX[chan][SampleNumber]/(n-not
counted);

                vmeanX2[chan][SampleNumber] =
                                                v2sumX2[chan][SampleNumber]/(n-no
tcounted);

                vmeanX2sq[chan][SampleNumber] = pow( vmean[chan][SampleNumber], 2
);

                variance[chan][SampleNumber] = vmeanX2[chan][SampleNumber]
                                                - vmeanX2sq[chan][SampleNumber];


                std[chan][SampleNumber] = sqrt(variance[chan][SampleNumber]);

                notcounted = 0;
        }


    //Store Mean for each Chan
    fprintf( fpvmean, "\n%d, ", SampleNumber);
    for( chan = 0; chan < 8; chan ++ )
    {
            fprintf( fpvmean, "%4.2f, ", vmean[chan][SampleNumber]);
    }

        //Store Variance for each Chan
        fprintf( fpvariance, "\nV: %d, ", SampleNumber );
    for( chan = 0; chan < 8; chan ++ )
    {
        fprintf( fpvariance, "%4.2f, ", variance[chan][SampleNumber]);
    }

        //Store Standard Deviation for each Chan
        fprintf( fpstd, "\n%d, ", SampleNumber );
    for( chan = 0; chan < 8; chan ++ )
    {
        fprintf( fpstd, "%4.2f, ", std[chan][SampleNumber]);
    }

    fclose( fpstd );
    fclose( fpvariance );
    fclose( fpvmean );

}
```

```
/*
  IACperf.C
  Written by:   Michael Womac
*/


/*************************************************************************/
/********                      Headers                            ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include "constant.h"
#include "variables.h"

/*************************************************************************/
/********                      Variables                          ********/
/*************************************************************************/


/*************************************************************************/
/********                      Functions                          ********/
/*************************************************************************/

void tac( int, int );
void read_adc( int );

/*************************************************************************/
/********                      Code                               ********/
/*************************************************************************/



/************************************************************************/

/*
   Function Name: tacperf
   Requires:  void
   Returns:   void
   Purpose:   selects channels 0 & 4 to receive 200 ns pulse sequence
              repetively and display the data read while adjustment is being
              done.
*/

/************************************************************************/


void tac( int chan, int pulse )
{

   int pulsecode[10]={ 0x0000, 0x0020, 0x0040, 0x0060, 0x0080,
                                                0x00a0, 0x00c0, 0x00e0, 0x0100,
0x0120 };
/*
sent to connector J1

pulse_20ns      0x0000
pulse_40ns      0x0020
pulse_60ns      0x0040
pulse_80ns      0x0060
pulse_100ns     0x0080
pulse_120ns     0x00a0
pulse_140ns     0x00c0
pulse_160ns     0x00e0
```



```
pulse_180ns     0x0100
pulse_200ns     0x0120
*/

      int chancode[17]={ 0x0080, 0x0100, 0x0200, 0x0400, 0x0800,
                                                0x1000, 0x2000, 0x4000,
0x5500, 0x2a80,                                 0x7f80, 0x0780, 0x7800,
0x0180, 0x0600,                                 0x1800, 0x6000 };


/*
command sent to connector J3

selected chan/s:     HEX Code:

0               chan0           0x0080
1               chan1           0x0100
2               chan2           0x0200
3               chan3           0x0400
4               chan4           0x0800
5               chan5           0x1000
6               chan6           0x2000
7               chan7           0x4000
8               odd                             0x5500
9               even                            0x2a80
10              all                     0x7f80
11              1st 4 (0-3)     0x0780
12              2nd 4   (4-7)           0x7800
13              1st 2   (0,1)           0x0180
14              2nd 2   (2,3)           0x0600
15              3rd 2   (4,5)           0x1800
16              4th 2 (6,7)             0x6000
*/

      /*
         These next 2 output cmds make active the proper control bits that
         are needed prior to the start and trigger cmds to start the
         conversions and reading of the data
      */

   /*  Important: DO NOT reset Mach part prior to IAC use */
   J1_data = pulsecode[pulse];
   J3_data = chancode[chan];
      outpw( io_base_address + J3lower_byte, J3_data );
      outpw( io_base_address + J1lower_byte, J1_data );



      /*
         These output cmds output the tac_strt cmd along
         with the cmds that are already active
      */

      outpw( io_base_address + J3lower_byte, J3_data + tac_strt );
      outpw( io_base_address + J3lower_byte, datacir );


      read_adc( chan );

      outpw( io_base_address + J3lower_byte, datacir );
}

/************************************************************************/
```

```
/*
   Function Name: read_adc
   Requires:

   Returns:
   Purpose:
*/

/*************************************************************************/


void read_adc( int chan )
{
   int fsc;

        /*
           This output cmd outputs the acd_oe, adc_trig, and
           the J1 data that is already active.
                Also, the adc_trig is 'and'ed with the clr_adc_
           trig signal in order to clear it out.
        */
        outpw( io_base_address + J1lower_byte,
                    J1_data + adc_oe + adc_trig );
        outpw( io_base_address + J1lower_byte,
                  ( J1_data + adc_oe + adc_trig )
                  & clr_adc_trig );
        /*
          This flow control checks for full scale conversion
          to be low.  If it is high, it will continue to
          check it.  If it is low, it will continue in the
          program
        */
        do
        {
                fsc = (inpw(io_base_address + J2lower_byte)
                                  & 0x8000);
        } while( fsc );


        /*
          Address data & ADC OE is output to the DAQ chip
          here along with any necessary codes previously
          defined
        */

        outpw( io_base_address + J1lower_byte,
                        chan + adc_oe);

        /*
                Data is input here from the DATA I/O Bd & 16 bit data
                that is collected is 'and'ed with the input data with
                a mask which will allow only the adc data
        */


        tac_data[ chan ] = (inpw( io_base_address + J2lower_byte )
                                          & data_mask );

}
```

```
/*
   tleast.CPP
   WRITTEN BY:     MICHAEL WOMAC
   DATE:  1 July 95
   Ver. 1.0
*/

/*************************************************************************/

/*
   Function Name: int

   Requires:  int chip, int NumofSamples,
                                int Numof_Iterations, int upper, int lower,
                                int line_endpoint
              where -> chip: is the chip ID #
                                NumofSamples: # of seperate points taken
                                Numof_Iterations: is the # times to test at
                                                  each point
                                upper: is the upper limit at which the data for
 the best
                                                  fit line is to be calculated
                                lower: is the lower limit at which the data for
 the best
                                                  fit line is to be calculated
                                line_endpoint: is the point at which to start u
sing data
                                                  for best fit line.

   Returns: void

   Purpose: Calculate the Best fit line parameter from the data in memory
   created by int.cpp.  The best fit line is calculated and the result is
        used to calculate the INL for the DUT.  The results are stored to disk
        under various file extensions for each device.
*/

/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>

#define MaxWords 50  //Defines the max number of words in array for parsing
#define MaxBuffer 150 //Defines # of char's that are receive at one time from
                                                        //from the fgets() funct
ion.

double _far total_int[10][9] = {0};
double _far mean_data[10][9] = {0};
double _far store_sample[10] = {0};
double _far y[10][8] = {0};

void Parse_data( char *, int *, char ** );


void int( int chip, int NumofSamples, int Numof_Iterations, int upper,
                int lower)
{
        char    *File_num[MaxWords],
                  inputbuffer[MaxBuffer] = {0};
```



```
        int Count, line = 0;

        int header_info[3];

  char filename[80];

        int word_count, increment = 0, chan;

        FILE *fp1;


                int
                        sample;

        double current_value = 0, maximum_value = 0;

        double
                        slopeterm1[8] = {0}, slopeterm2[8] = {0},
                        yinter_term1[8] = {0}, yinter_term2[8] = {0},
                        slope[8] = {0}, y_intercept[8] = {0},
                        sumX = 0,
                        sumY[8] = {0},
                        sumX2 = 0,
                        sumXY[8] = {0},
                        term3[8] = {0},
                        term4[8] = {0},
         int[8] = {0},
         raw_int[8] = {0};

        //File pointer initializations
        FILE *fpint, *fprawint, *ftotalint, *fplin, *fpmb;

        //Setup Files
        sprintf( filename, "c:\\perform\\tac\\int\\data\\tc%d.lin", chip );
        fplin = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\tac\\int\\data\\tc%d.int", chip );
        fpint = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\tac\\int\\data\\tc%dtotal.int", chip );
        ftotalint = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\tac\\int\\data\\tc%d.rni", chip );
        fprawint = fopen(filename, "w");

        sprintf( filename, "c:\\perform\\tac\\int\\data\\tc%d.mb", chip );
        fpmb = fopen(filename, "w");

        fprintf( fplin , "Chip #: %d\n", chip );
        fprintf( fplin, "# Points: %d\n", Numof_Iterations );

        fprintf( fpint , "Chip #: %d: %d\n", chip );
        fprintf( fpint, "# Points: %d\n", Numof_Iterations );

        fprintf( ftotalint , "Chip #: %d\n", chip );
        fprintf( ftotalint, "# Points: %d\n", Numof_Iterations );

        fprintf( fprawint , "Chip #: %d\n", chip );
        fprintf( fprawint, "# Points: %d\n", Numof_Iterations );

        fprintf( fpmb , "Chip #: %d\n", chip );
        fprintf( fpmb, "# Points: %d\n", Numof_Iterations );

        sprintf( filename, "c:\\perform\\tac\\int\\data\\tc%d.mr", chip );

        //Get Data from file
```

```
        if( (   fpl = fopen(filename, "r") ) != NULL )
        {
                while( (fgets(inputbuffer, MaxBuffer, fpl) != NULL) )
                {

                        //Seperates data into blocks or words to an array pointed
                        //to by an array of pointers called File_num.  The number
of the
                        //words in the array is determined by MaxWords constant.
 The number
                        //of characters loaded into 'fgets' each pass is determin
ed by
                        //MaxBuffer constant.  Count is the number of words in th
e array
                        //pointed to by File_num.
                        Parse_data( inputbuffer, &Count, &(File_num[0]) );

                        //This allows the header info to be pulled into an array
called
                        //header_info
                        if( Count == 5 )
                        {
                                for( word_count = 0; word_count < Count; word_cou
nt ++ )
                                {
                                        header_info[word_count] = atoi(file_num[w
ord_count]);
                                }
                        }
                        //This allows the mean data to be stored into an array ca
lled
                        //mean_data
                        if( line > 4)
                        {
                                for( word_count = 0; word_count < Count; word_cou
nt ++ )
                                {
                                        mean_data[increment][word_count] = atof(F
ile_num[word_count]);
                                }
                        }
                        if( line > 4 )
                        {
                                increment ++;
                        }
                        line ++;
                }
        }
        fclose( fpl );

    for( increment = 0; increment < (NumofSamples); increment ++ )
    {
                store_sample[(increment)] = (mean_data[increment][0]+1)*20;
                sumX += (mean_data[increment][0]+1)*20;
                sumX2 += pow( (mean_data[increment][0]+1)*20, 2);
                for( chan = 1; chan < 9; chan ++ )
                {

                        //Note: must subtract 1 from the index for sumY and sumXY
 because
                        //the mean_data matrix is [10][9] and sumY and sumXY are
[8].
                        //Only want the last 8 columns of mean_data matrix and wa
nt them
                        //in the first 8 columns of sumX and sumXY therefore subt
```



```
ract one
                        //from their indexes.

                        sumY[chan-1] += mean_data[increment][chan];
                        sumXY[chan-1] += (mean_data[increment][0]+1)*20
                                                *mean_data[increment][chan];
                }
    }


    //Calculate terms for Best Fit Line Equations
                for( chan = 0; chan < 8; chan ++ )
                {


                slopeterm1[chan] = (NumofSamples)*sumXY[chan];
                slopeterm2[chan] = sumX*sumY[chan];
                yinter_term1[chan] = sumX2*sumY[chan];
                yinter_term2[chan] = sumXY[chan]*sumX;
                term3[chan] = (NumofSamples)*sumX2;
                term4[chan] = pow( sumX, 2);

                        slope[chan] = (slopeterm1[chan] - slopeterm2[chan])/
                                                        (term3[chan] - term4[ch
an]);

                        y_intercept[chan] = (yinter_term1[chan] - yinter_term2[chan])/
                                                        (term3[chan] - term4[ch
an]);

                }

                //Calculate the Best Fit Line from the slope and y intercept valu
es
                for( chan = 0; chan < 8; chan ++ )
                {
                        for( sample = 0; sample < (NumofSamples);
                                sample ++ )
                        {
                                y[sample][chan] = slope[chan] * store_sample[samp
le]
                                                                + y_inte
rcept[chan];
                        }
                }

                //Calculate the absolute INL per chan
                for( chan = 0; chan < 8; chan ++ )
                {
                        for( sample = 0; sample < NumofSamples;
                                sample ++ )
                        {
                                //This if statement cuts off the end point data f
rom the int
                                //test and only checks the linear section
                                if( mean_data[sample][chan+1] < upper &&
                                        mean_data[sample][chan+1] > lower )
                                {
                                        current_value = fabs( mean_data[sample][c
han+1] -
                y[sample][chan] );
                                        if( maximum_value < current_value)
```

```
                                        {
                                                ini[chan] = maximum_value = curre
nt_value;
                                        }
                                }
                        }
                current_value = maximum_value = 0;
        }


        //Calculate the raw INL per chan
        for( chan = 0; chan < 8; chan ++ )
        {
                for( sample = 0; sample < NumofSamples;
                                sample ++ )
                {
                        //This if statement cuts off the end point data f
rom the inl
                        //test and only checks the linear section
                        if( mean_data[sample][chan+1] < upper &&
                                mean_data[sample][chan+1] > lower )
                        {
                                total_inl[sample][0] = mean_data[sample][
0];
                                total_inl[sample][chan+1] =
                                mean_data[sample][chan+1] - y[sample][cha
n];
                                current_value = fabs( mean_data[sample][c
han+1]
                                                - y[sample][cha
n] );

                                if( maximum_value < current_value)
                                {
                                        maximum_value = current_value;
                                        raw_inl[chan] =
                                        mean_data[sample][chan+1] - y[sam
ple][chan];

                                }
                        }
                }
        current_value = maximum_value = 0;
        }



        //Store INL Data to file

    fprintf( ftotalinl, "Percent of FSR:\n" );
        for( sample = 0; sample < NumofSamples;
                        sample ++ )
        {
                fprintf( ftotalinl, "%4.2f, ", total_inl[sample][0] );
                for( chan = 1; chan < 9; chan ++ )
                {

           fprintf( ftotalinl, "%4.2f, ", (total_inl[sample][chan]/1024)*100 );
          }
         fprintf( ftotalinl, "\n" );
        }

        fprintf( ftotalinl, "\n\n" );
```



```
    fprintf( ftotalinl, "ADC Counts:\n" );
        for( sample = 0; sample < NumofSamples;
                        sample ++ )
        {
                for( chan = 0; chan < 9; chan ++ )
                {

         fprintf( ftotalinl, "%4.2f, ", total_inl[sample][chan] );
          }
         fprintf( ftotalinl, "\n" );
        }



        for( chan = 0; chan < 8; chan ++ )
                {
           fprintf( fpinl, "%4.2f, ", inl[chan] );
        }

        for( chan = 0; chan < 8; chan ++ )
                {
           fprintf( fprawinl, "%4.2f, ", raw_inl[chan] );
        }

                //Store Best Fit Line Data
                fprintf( fplin, "\n");
                for( chan = 0; chan < 8; chan ++ )
                {
                        if(chan == 0 )
                    {
                                fprintf( fplin, "Pulse, " );
                                for( sample = 0; sample < NumofSamples;
                                                sample ++ )
                                {
                                        fprintf( fplin, "%4.2f, ",
                                                        store_sample[sample]);
                                }
                                fprintf( fplin, "\n" );
                        }
                        fprintf( fplin, "chan: %d, ", chan);
                        for( sample = 0; sample < NumofSamples;
                                        sample ++ )
                        {
                        fprintf( fplin, "%4.2f, ", y[sample][chan]);
                        }
                        fprintf( fplin, "\n" );
        }

                //Store Slope and Y-Intercept of Best fit line
                fprintf( fpmb, "Slope per chan (0 - 7)\n" );
                for( chan = 0; chan < 8; chan ++ )
                {
                        fprintf( fpmb, "%4.2f, ", slope[chan] );
                }

                fprintf( fpmb, "\n\nY-Intercept per chan (0 - 7)\n" );
                for( chan = 0; chan < 8; chan ++ )
                {
                        fprintf( fpmb, "%4.2f, ", y_intercept[chan] );
                }

                fclose( fpmb );
                fclose( fpinl );
                fclose( fplin );
```

```
)



/*************************************************************************/

/*
        Function Name:  Parse_data
        Requires:       pointer to a predefined character array (buffer)
                                        pointer to integer to count the
number of words stored
                                                in the word array
                                        pointer to a pointer to a charac
ter array with desired
                                                                size of p
arsed words
        Definitions:
                #define MaxWords 50  Defines the max number of words in array for
                                                                parsing
                #define MaxBuffer 150 Defines # of char's that are receive at one
                                                                time fro
m from the fgets() function.

        Declarations:

                functions -     void Parse_data( char *, int *, char ** );

                Variables -     char    *File_num[MaxWords],
                                                        inputbuffer[Ma
xBuffer] = {0};
                                                int Count;

        Usage:

                while( (fgets(inputbuffer, MaxBuffer, fp1) != NULL) )
                {
                        Parse_data( inputbuffer, &Count, &(File_num[0]) );
                        .
                        .
                        .

        Returns: void
        Purpose: seperates a buffer full (this case one line) of data into a
                                'word' array that is predefined in the calling pr
g by MaxWord.
                                The buffer length is determined by MaxBuffer cons
tant.
be converted
                                Once the buffer is seperated, the Word array can
                                an 'int' or 'double' or can be used as a 'char' (
which is the
                                way the Word array is defined).
*/
/*************************************************************************/
```



```
void Parse_data( char *buf, int *Count, char **Word)
{
        *Word = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (i.e. \0) character in the contents
 of the current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array which uses count
s as its
                        indexing parameter
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        Word[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                        (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;
                }
        }
        *buf = '\0';
}
```

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = TDNL
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\PERFORM\TAC\DNL\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = TACPERF.C
FIRSTCPP = TDNL.CPP
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AM /Od /D "_DEBUG" /D "_DOS" /FR /Fd"TDNL.PD
B"
CFLAGS_R_DEXE = /nologo /Gs /G2 /W3 /AM /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE /CO
LFLAGS_R_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE
LIBS_D_DEXE = oldnames llibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = TDNL.SBR \
               TACPERF.SBR


TDNL_DEP = c:\msvc\include\iostream.h \
        c:\msvc\include\ios.h \
        c:\msvc\include\streamb.h \
        c:\msvc\include\istream.h \
        c:\msvc\include\ostream.h \
        c:\msvc\include\stdio.h \
        c:\msvc\include\stdlib.h \
        c:\msvc\include\conio.h \
        c:\msvc\include\malloc.h \
        c:\msvc\include\graph.h \
        c:\msvc\include\math.h \
```



```
        c:\msvc\include\time.h \
        c:\perform\tac\dnl\variable.h \
        c:\perform\tac\dnl\constant.h


TACPERF_DEP = c:\msvc\include\stdio.h \
        c:\msvc\include\conio.h \
        c:\msvc\include\process.h \
        c:\perform\tac\inl\constant.h \
        c:\perform\tac\inl\variable.h


all:    $(PROJ).EXE $(PROJ).BSC

TDNL.OBJ:       TDNL.CPP $(TDNL_DEP)
        $(CPP) $(CFLAGS) $(CPPCREATEPCHFLAG) /c TDNL.CPP

TACPERF.OBJ:    ..\INL\TACPERF.C $(TACPERF_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c ..\INL\TACPERF.C

$(PROJ).EXE::   TDNL.OBJ TACPERF.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
TDNL.OBJ +
TACPERF.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
/******** Constants ********/

#define upper_limit     0x03ff
#define amu_voltage_1   0x2000 /* sent to connector J1 */
#define amu_voltage_3   0xdfff /* sent to connector J1 */
#define dataclr         0x0000
#define data_mask       0x03ff
#define dont_care       0xffff

#define io_base_address 0x02a0
#define J1lower_byte    0x0000
#define J1upper_byte    0x0001
#define J2lower_byte    0x0000
#define J2upper_byte    0x0001
#define J3lower_byte    0x0002
#define J3upper_byte    0x0003

/* mach code masks */

#define reset           0x8000 /* sent to connector J1 */
#define adc_ramp        0x0200 /* sent to connector J1 */
#define adc_trig        0x0200 /* sent to connector J1 */
#define clr_adc_trig    0xfdff /* sent to connector J1 */
#define adc_oe          0x1000 /* sent to connector J1 */
#define tac_strt        0x8000 /* sent to connector J3 */


#define pulse_clr_mask  0xfe1f /* sent to connector J1 */

#define pulse_20ns      0x0000 /* sent to connector J1 */
#define pulse_40ns      0x0020 /* sent to connector J1 */
#define pulse_60ns      0x0040 /* sent to connector J1 */
#define pulse_80ns      0x0060 /* sent to connector J1 */
#define pulse_100ns     0x0080 /* sent to connector J1 */
#define pulse_120ns     0x00a0 /* sent to connector J1 */
#define pulse_140ns     0x00c0 /* sent to connector J1 */
#define pulse_160ns     0x00e0 /* sent to connector J1 */
#define pulse_180ns     0x0100 /* sent to connector J1 */
#define pulse_200ns     0x0120 /* sent to connector J1 */

#define chan0           0x0080 /* sent to connector J3 */
#define chan1           0x0100 /* sent to connector J3 */
#define chan2           0x0200 /* sent to connector J3 */
#define chan3           0x0400 /* sent to connector J3 */
#define chan4           0x0800 /* sent to connector J3 */
#define chan5           0x1000 /* sent to connector J3 */
#define chan6           0x2000 /* sent to connector J3 */
#define chan7           0x4000 /* sent to connector J3 */

#define test0           0x0000 /* sent to connector J3 */
#define test1           0x0010 /* sent to connector J3 */
#define test2           0x0020 /* sent to connector J3 */
#define test3           0x0030 /* sent to connector J3 */
#define test4           0x0040 /* sent to connector J3 */
#define test5           0x0050 /* sent to connector J3 */
#define test6           0x0060 /* sent to connector J3 */
#define test7           0x0070 /* sent to connector J3 */
```

```
int J1_date, J3_date;
int tac_data[8];
```

```
/*
   idnl.CPP
   WRITTEN BY:     MICHAEL WOMAC
   DATE:  6 Jun 95 -
*/


/*************************************************************************/

/*
   Function Name: dnl.exe
   Requires:  command line: 'chip number #points MaxVoltage'
   Returns: void
   Purpose: This prg uses a random pulse generator.  It provides random
   stop bits for the TAC of the DUT. The prg provides all
   hand shake signals needed to interface for TAC
   conversion.

*/

/*************************************************************************/



/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include <math.h>
#include <time.h>
//#include "variable.h"
#include "constants.h"

/*************************************************************************/
/********                      Defines                          ********/
/*************************************************************************/


/*************************************************************************/
/********                      Structures                       ********/
/*************************************************************************/


/*************************************************************************/
/********                      Global Variables                 ********/
/*************************************************************************/


unsigned long _huge DNLresults[8][1024];
extern "C"
{
                int     J1_data,
                        J3_data,
                        tac_data[8];
                void  tac( int, int);
                void  read_adc( int );
}
```



```
/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/


/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/


void main( int argc, char **argv )
{

        //Note: the limit in the increment is due to least
        //squares allocating the mean_data matrix
   if( argc != 4 )
   {
        cout << "Usage: chipnum SamplesPerChan freq" << endl;
      cout << "chipnum: is an integer value for chip id" << endl
                        << "SamplesPerChan: desired # of samples per channels"

                        << "freq: Input frequency of ramp from sig gen"
           << endl;
      exit(1);
   }

   char backup_time[20], start_time[20], end_time[20];
        int chan = 0, adc_chan = 0, chip, freq;
        long loop=0, counter = 0, samp, SamplesPerChan;

        long ramp_count = 0, last_ramp = 0; //determines how many ramps have elap
sed


        char filename[80];

        FILE *fp;

        clock_t start, finish, tmp, backup = 0;
        double duration, last_reported;



        //Input Parameters
        freq = atoi(argv[3]);
        samp = atoi(argv[2]);
        chip = atoi(argv[1]);

        SamplesPerChan = 1024*samp;

        //Pb chip test setup
        outpw( io_base_address + J1lower_byte, 0x2000 );
        outpw( io_base_address + J1lower_byte, reset );

        start = clock();

        _strtime( start_time );

        while( loop < SamplesPerChan )
        {
                loop ++;
                counter ++;

            tac( 3,0);

                for( adc_chan=0; adc_chan<8; adc_chan++)
```

```
                {
                        DNLresults[3][tac_data[3]] ++;
                }
                if( counter == SamplesPerChan/10 )
                {
                   finish = clock();
                   duration = (double)(finish - start)/CLOCKS_PER_SEC;
                   cout << "Ideal # of samples/chan: " << loop/1024 << endl;
                   cout << "Elapsed Time: " << duration << endl;
                        counter = 0;
                }

        }


        _strtime( end_time );

   finish = clock();
   duration = (double)(finish - start)/CLOCKS_PER_SEC;

        //Save channel results to disk
        sprintf( filename, "c:\\perform\\tac\\dnl\\data\\t%d.dat", chip );
        fp = fopen(filename, "w");
        fprintf( fp, "Chip #: %d\n", chip );
        fprintf( fp, "Time Program ended: %s\n", end_time );
   fprintf( fp, "Total number of # of samples: %lu\n" loop );
   fprintf( fp, "Ideal number of # of samples/chan: %lu\n", loop/1024 );
   fprintf( fp, "Input Frequency: %dKHZ\n", freq );
   fprintf( fp, "Duration: %f in seconds\n", duration );
   fprintf( fp, "Duration: %f in minutes\n", duration/60 );
   fprintf( fp, "Duration: %f in hours\n", duration/3600 );
//      for( chan = 0; chan < 8; chan ++ )
//      {
                fprintf( fp,"%d, ", 3 ); //chan );
//      }
        fprintf( fp, "\n");

        for( adc_chan = 0; adc_chan < 1024; adc_chan ++ )
        {
//              for( chan = 0; chan < 8; chan ++ )
//              {
                        fprintf( fp, "%lu, ", DNLresults[3][adc_chan] );
//              }
                fprintf( fp, "\n" );
        }
        fclose( fp );

   cout << "Approximate # of ramps: " << duration*freq << endl;
   cout << "Total number of # of samples: " << loop << endl;
   cout << "Ideal number of # of samples/chan: " << loop/1024 << endl;
   cout << "Time per Loop: " << duration/loop << " in seconds" << endl;

        cout << endl << "Start Time: " << start_time << endl;
        cout << "End Time: " << end_time << endl;
   cout << "Elapsed CPU Time: " << duration << " seconds" << endl;
        cout << "                  or " << duration/60 << " minutes" << endl;
        cout << "                  or " << duration/3600 << " hours" << endl;
}
```

```
/*
  TACperf.C
  Written by:   Michael Womac
*/

/*************************************************************************/
/********                      Headers                            ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include "constant.h"
#include "variables.h"

/*************************************************************************/
/********                      Variables                          ********/
/*************************************************************************/


/*************************************************************************/
/********                      Functions                          ********/
/*************************************************************************/

void tac( int, int );
void read_adc( int );

/*************************************************************************/
/********                      Code                               ********/
/*************************************************************************/



/************************************************************************/

/*
   Function Name: tac
   Requires:  void
   Returns:   void
   Purpose:   to convert the desired chan with the chosen pulse seperation
   passed as input parameters from the calling function

*/

/************************************************************************/


void tac( int chan, int pulse )
{

   int pulsecode[10]={ 0x0000, 0x0020, 0x0040, 0x0060, 0x0080,
                                               0x00a0, 0x00c0, 0x00e0, 0x0100,
0x0120 );
/*
sent to connector J1

pulse_20ns      0x0000
pulse_40ns      0x0020
pulse_60ns      0x0040
pulse_80ns      0x0060
pulse_100ns     0x0080
pulse_120ns     0x00a0
pulse_140ns     0x00c0
pulse_160ns     0x00e0
```



```
pulse_180ns     0x0100
pulse_200ns     0x0120
*/

      int chancode[17]={ 0x0080, 0x0100, 0x0200, 0x0400, 0x0800,
                                                         0x1000, 0x2000, 0x4000,
0x5500, 0x2a80,                                          0x7f80, 0x0780, 0x7800,
0x0180, 0x0600,                                          0x1800, 0x6000 );


/*
command sent to connector J3

selected chan/s:     HEX Code:

0               chan0           0x0080
1               chan1           0x0100
2               chan2           0x0200
3               chan3           0x0400
4               chan4           0x0800
5               chan5           0x1000
6               chan6           0x2000
7               chan7           0x4000
8               odd                             0x5500
9               even                            0x2a80
10              all                     0x7f80
11              1st 4 (0-3)     0x0780
12              2nd 4   (4-7)           0x7800
13              1st 2   (0,1)           0x0180
14              2nd 2   (2,3)           0x0600
15              3rd 2   (4,5)           0x1800
16              4th 2 (6,7)             0x6000
*/
      /*
         These next 2 output cmds make active the proper control bits that
         are needed prior to the start and trigger cmds to start the
         conversions and reading of the data
      */

   /*  Important: DO NOT reset Mach part prior to TAC use */
   J1_data = pulsecode[pulse];
   J3_data = chancode[chan];
      outpw( io_base_address + J3lower_byte, J3_data );
      outpw( io_base_address + J1lower_byte, J1_data );



      /*
         These output cmds output the tac_strt cmd along
         with the cmds that are already active
      */

      outpw( io_base_address + J3lower_byte, J3_data + tac_strt );
      outpw( io_base_address + J3lower_byte, dataclr );


      read_adc( chan );

      outpw( io_base_address + J3lower_byte, dataclr );
}

/************************************************************************/
```

```
/*
   Function Name: read_adc
   Requires:

   Returns:
   Purpose:
*/

/*************************************************************************/


void read_adc( int chan )
{
   int fsc;


        /*
           This output cmd outputs the acd_oe, adc_trig, and
           the J1 data that is already active.
                Also, the adc_trig is 'and'ed with the clr_adc_
           trig signal in order to clear it out.
        */
        outpw( io_base_address + J1lower_byte,
                        J1_data + adc_oe + adc_trig );
        outpw( io_base_address + J1lower_byte,
                    ( J1_data + adc_oe + adc_trig )
                    & clr_adc_trig );
        /*
          This flow control checks for full scale conversion
          to be low.  If it is high, it will continue to
          check it.  If it is low, it will continue in the
          program
        */
        do
        {
                fsc = (inpw(io_base_address + J2lower_byte)
                                & 0x8000);
        } while( fsc );


        /*
          Address data & ADC OE is output to the DAQ chip
          here along with any necessary codes previously
          defined
        */

        outpw( io_base_address + J1lower_byte,
                        chan + adc_oe);

        /*
                Data is input here from the DATA I/O Bd & 16 bit data
                that is collected is 'and'ed with the input data with
                a mask which will allow only the adc data
        */


        tac_data[ chan ] = (inpw( io_base_address + J2lower_byte )
                                        & data_mask );

}
```

APPENDIX E

COLLECT.MAK AND ASSOCIATED PROGRAMS

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = COLLECT
DEBUG = 1
PROGTYPE = 0
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\WOMAC\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = COLLECT.C
FIRSTCPP =
RC = rc
CFLAGS_D_WEXE = /nologo /W3 /FR /GZ /Zi /D_DEBUG /Od /AM /GA
CFLAGS_R_WEXE = /nologo /W3 /FR /O1 /DNDEBUG /AM /GA
LFLAGS_D_WEXE = /NOLOGO /ONERROR:NOEXE /NOD /PACKC:61440 /CO /ALIGN:16 /STACK:102
40
LFLAGS_R_WEXE = /NOLOGO /ONERROR:NOEXE /NOD /PACKC:61440 /ALIGN:16 /STACK:10240
LIBS_D_WEXE = oldnames libw commdlg shell olecli olesvr mlibcew
LIBS_R_WEXE = oldnames libw commdlg shell olecli olesvr mlibcew
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
DEFFILE = COLLECT.DEF
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_WEXE)
LFLAGS = $(LFLAGS_D_WEXE)
LIBS = $(LIBS_D_WEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_WEXE)
LFLAGS = $(LFLAGS_R_WEXE)
LIBS = $(LIBS_R_WEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = COLLECT.SBR


COLLECT_DEP =

all:    $(PROJ).EXE $(PROJ).BSC

COLLECT.OBJ:    COLLECT.C $(COLLECT_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c COLLECT.C


$(PROJ).EXE::   COLLECT.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
COLLECT.OBJ +
```



```
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF
        $(RC) $(RESFLAGS) $B


run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
/*
        collect.C
        WRITTEN BY:     MICHAEL WOMAC
        DATE: 5 Jan 1995 - 9 Jan 1995
        Version: 1.0
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*************************************************************************/
/********                     Definitions                       ********/
/*************************************************************************/

#define word_size 16
#define line_size 256
#define amu_line_size 256

/**** MASKS ****/

#define chan0 0x0001
#define chan1 0x0002
#define chan2 0x0004
#define chan3 0x0008
#define chan4 0x0010
#define chan5 0x0020
#define chan6 0x0040
#define chan7 0x0080

#define pipe0 0x0001
#define pipe1 0x0002
#define pipe2 0x0004
#define pipe3 0x0008
#define pipe4 0x0010
#define pipe5 0x0020
#define pipe6 0x0040
#define pipe7 0x0080
#define pipe8 0x0100
#define pipe9 0x0200
#define pipe10 0x0400
#define pipe11 0x0800
#define pipe12 0x1000
#define pipe13 0x2000
#define pipe14 0x4000
#define pipe15 0x8000

/*************************************************************************/
/********                   Global Variables                    ********/
/*************************************************************************/


        /**** FILE POINTERS ****/

        FILE *fp_tac;
        FILE *fp_amu;
        FILE *fp_no_errt, *fp_no_errm;
        FILE *fp_a, *fp_i;
        FILE *fp_gd_chp;

        /**** CONSTANTS ****/
        int Count;
```



```
        int filecounter = 0;

        /* TAC */
        long Num_of_timeout = 0, Num_of_20ns = 0, Num_of_100ns = 0;
        long to[8];
        long p20ns[8];
        long p100ns[8];
    int tac_good = 0;
    int tac_only_failed = 0;
    int to_only_failed = 0;
    int p20_only_failed = 0;
    int p100_only_failed = 0;
    int tac_to_only_failure = 0;
    int tac_p20_only_failure = 0;
    int tac_p100_only_failure = 0;
        int multit_failed = 0;
        int tac_multi_failures_only = 0;

        /* AMU */
        long Num_1cells = 0, Num_3cells = 0, Num_1pipes = 0, Num_3pipes = 0;
        int one_cell_failure = 0;
        int one_five_failed = 0, three_five_failed = 0;
        int one_failed = 0;
        int both_failed = 0;
    int one_20ns_failed = 0;
    int one_100ns_failed = 0;
    int amu_good = 0;
    int amu_only_failed = 0;
    int both_sections_failed = 0;
    int both_good=0;
        int amu_matrix = 0;
        int hard_add= 0;
        int loopcntr=0;
        int achip = 0;
        int tchip = 0;
        long bad_pipes_per_chip_cntr = 0;
    int multia_failed = 0;
    int amu_multi_failures_only = 0;
    int both_multi_failures = 0;
    int good_chip = 0;
    int tot;

    /**** ADC ****/
    int adc_or_ps_failure = 0;
    int alreadycounted = 0;

        /**** CHARACTER ARRAYS ****/
        char *word[ 20 ], *words[20], *wordt[20];
        char line_of_data[ line_size ];
        char line_of_amu_data[ amu_line_size ];
        char chnuma[10], chnumt[10];

        /**** INTEGER ARRAYS ****/
        long parameter[20];
        long parametert[20];
        long p1_5[16];
        long p3_5[16];
        int num_bad_pipes_per_chip[17];
        long r1[16];
        long c1[16];
        long r3[16];
        long c3[16];
        long p1[16];
        long p3[16];
        long add1[16];
        long add3[16];
```

```
        int add1_failure[16];
        long currentadd1[17];

/*************************************************************************/
/********                    Structures                         ********/
/*************************************************************************/



/*************************************************************************/
/********                    Functions                          ********/
/*************************************************************************/

 void Parse_data( char *, int *, char ** );

/*************************************************************************/
/********                    Code                               ********/
/*************************************************************************/



/*************************************************************************/

/*
        Function Name:  main
        Requires:   void
        Returns:    void
        Purpose: To collect Pb chip statistical data from output files generated
                                by test.exe.
*/

/*************************************************************************/


void main( void )
{

                        /* Array initialization */
                        for(loopcntr = 0; loopcntr< 16; loopcntr ++)
                        {
                                p1_5[loopcntr] = 0;
                                p3_5[loopcntr] = 0;
                                to[loopcntr] = p20ns[loopcntr] = p100ns[loopcntr]
 = 0;
                                r1[loopcntr]=c1[loopcntr]=r3[loopcntr]=c3[loopcnt
r]=0;
                                p1[loopcntr]=p3[loopcntr]=add1[loopcntr]=add3[loo
pcntr]=0;

                        }

                        for(loopcntr = 0; loopcntr< 17; loopcntr ++)
                        {
                                num_bad_pipes_per_chip[loopcntr] = 0;
                                add1_failure[loopcntr]=0;
                        currentadd1[loopcntr]=0;
                        }

/*************************************************************************/
/*************************************************************************/
/*                      TAC SECTION                                      */
/*************************************************************************/
/*************************************************************************/


    fp_no_errs = fopen( "c:\\womac\\nochip#\\no_err.amu", "w" );
```



```
        /*
                opens file that contains data
        */
  fp_amu = fopen("c:\\womac\\results\\results.amu", "r");

        fp_no_errt = fopen( "c:\\womac\\nochip#\\no_err.tac", "w" );

        /*
                opens file that contains data
        */
        if(  (fp_tac = fopen("c:\\womac\\results\\results.tac", "r"))  != NULL)
        {
          /*
                get first line of data
          */
          while ( (fgets(line_of_data, line_size, fp_tac) != NULL) )
          {

                        Parse_data( line_of_data, &Count, &(word[0]) );

                        /*
                                converts character array into integer array
                        */

                        parameter[0] = atoi(word[0]); /* chip number */
                        parameter[1] = atoi(word[1]); /* total time out failures
 */
                        parameter[2] = atoi(word[2]); /* sum of 20ns failures */

                        parameter[3] = atoi(word[3]); /* sum of 100ns failures *
/
                        parameter[4] = atoi(word[4]); /* locates timout failures
 */
                        parameter[5] = atoi(word[5]); /* locates 20ns failures *
/
                        parameter[6] = atoi(word[6]); /* locates 100ns failures
*/
                        parameter[7] = atoi(word[7]); /* locates poss tac adc fa
ilures */

                        filecounter ++;

                        Num_of_timeout += parameter[1]; /* total time out failur
es */
                        Num_of_20ns += parameter[2]; /* sum of 20ns failures */
                        Num_of_100ns += parameter[3]; /* sum of 100ns failures *
/

                        if( parameter[2] == 1 )
                        {
                                one_20ns_failed ++;
                        }

                        if( parameter[3] == 1 )
                        {
                                one_100ns_failed ++;
                        }


                        /* locates timout failures */
                        if( (parameter[4] & chan0) == 1 )
                        {
                                to[0] ++;
```

```
}
if( (parametert[4] & chan1) == 2 )
{
        to[1] ++;
}
if( (parametert[4] & chan2) == 4 )
{
        to[2] ++;
}
if( (parametert[4] & chan3) == 8 )
{
        to[3] ++;
}
if( (parametert[4] & chan4) == 16 )
{
        to[4] ++;
}
if( (parametert[4] & chan5) == 32 )
{
        to[5] ++;
}
if( (parametert[4] & chan6) == 64 )
{
        to[6] ++;
}
if( (parametert[4] & chan7) == 128 )
{
        to[7] ++;
}

/* locates 20ns failures */
if( (parametert[5] & chan0) == 1 )
{
        p20ns[0] ++;
}
if( (parametert[5] & chan1) == 2 )
{
        p20ns[1] ++;
}
if( (parametert[5] & chan2) == 4 )
{
        p20ns[2] ++;
}
if( (parametert[5] & chan3) == 8 )
{
        p20ns[3] ++;
}
if( (parametert[5] & chan4) == 16 )
{
        p20ns[4] ++;
}
if( (parametert[5] & chan5) == 32 )
{
        p20ns[5] ++;
}
if( (parametert[5] & chan6) == 64 )
{
        p20ns[6] ++;
}
if( (parametert[5] & chan7) == 128 )
{
        p20ns[7] ++;
}

/* locates 100ns failures */
if( (parametert[6] & chan0) == 1 )
```



```
{
        p100ns[0] ++;
}
if( (parametert[6] & chan1) == 2 )
{
        p100ns[1] ++;
}
if( (parametert[6] & chan2) == 4 )
{
        p100ns[2] ++;
}
if( (parametert[6] & chan3) == 8 )
{
        p100ns[3] ++;
}
if( (parametert[6] & chan4) == 16 )
{
        p100ns[4] ++;
}
if( (parametert[6] & chan5) == 32 )
{
        p100ns[5] ++;
}
if( (parametert[6] & chan6) == 64 )
{
        p100ns[6] ++;
}
if( (parametert[6] & chan7) == 128 )
{
        p100ns[7] ++;
}


/*
        records chip numbers with not errors indicated
*/
if( parametert[1] == 0 && parametert[2] == 0 && parametert[3] == 0 &&
        parametert[0] != 0 )
{
        fprintf( fp_no_errt , "\n%d ", parametert[0] );
}
```

```
/*****************************************************************************/
/*****************************************************************************/
/*                          AMU SECTION                                      */
/*****************************************************************************/
/*****************************************************************************/

        fgets(line_of_amu_data, amu_line_size, fp_amu);

                /*
                */      get first line of data
```

```
 */
 */
 */
 */
es in a amu add */
es in a pipe */
es in a amu add */
es in a pipe */
es  */
ures  */
es */
es */
ilure */
/
```

```
Parse_data( line_of_amu_data, &Count, &(word[0]) );

/*
        converts character array into integer array
*/

parameter[0] = atoi(word[0]); /* chip number */
parameter[1] = atoi(word[1]); /* total 1.5v cell failures
parameter[2] = atoi(word[2]); /* total 3.5v cell failures
parameter[3] = atoi(word[3]); /* total 1.5v pipe failures
parameter[4] = atoi(word[4]); /* total 3.5v pipe failures
parameter[5] = atoi(word[5]); /* locates 1.5v cell failur
parameter[6] = atoi(word[6]); /* locates 1.5v cell failur
parameter[7] = atoi(word[7]); /* locates 3.5v cell failur
parameter[8] = atoi(word[8]); /* locates 3.5v cell failur
parameter[9] = atoi(word[9]); /* locates 1.5v pipe failur
parameter[10] = atoi(word[10]); /* locates 3.5v pipe fail
parameter[11] = atoi(word[11]); /* locates 1.5v addfailur
parameter[12] = atoi(word[12]); /* locates 3.5v addfailur
parameter[13] = atoi(word[13]); /* one complete matrix fa
parameter[14] = atoi(word[14]); /* both matrix's failed *


Num_1cells += parameter[1];
Num_3cells += parameter[2];
Num_1pipes += parameter[3];
Num_3pipes += parameter[4];

/* locates amu addrs with bad 1.5v cells */
if( (parameter[5] & pipe0) == 1 )
{
        r1[0] ++;
}
if( (parameter[5] & pipe1) == 2 )
{
        r1[1] ++;
}
if( (parameter[5] & pipe2) == 4 )
{
        r1[2] ++;
}
if( (parameter[5] & pipe3) == 8 )
{
        r1[3] ++;
}
if( (parameter[5] & pipe4) == 16 )
{
        r1[4] ++;
}
if( (parameter[5] & pipe5) == 32 )
{
```



```
        r1[5] ++;
}
if( (parameter[5] & pipe6) == 64 )
{
        r1[6] ++;
}
if( (parameter[5] & pipe7) == 128 )
{
        r1[7] ++;
}
if( (parameter[5] & pipe8) == 256 )
{
        r1[8] ++;
}
if( (parameter[5] & pipe9) == 512 )
{
        r1[9]++;
}
if( (parameter[5] & pipe10) == 1024 )
{
        r1[10] ++;
}
if( (parameter[5] & pipe11) == 2048 )
{
        r1[11] ++;
}
if( (parameter[5] & pipe12) == 4096 )
{
        r1[12] ++;
}
if( (parameter[5] & pipe13) == 8192 )
{
        r1[13] ++;
}
if( (parameter[5] & pipe14) == 16384 )
{
        r1[14] ++;
}
if( (parameter[5] & pipe15) == 32768 )
{
        r1[15] ++;
}


/* locates pipes with bad 1.5v cells */
if( (parameter[6] & pipe0) == 1 )
{
        c1[0] ++;
}
if( (parameter[6] & pipe1) == 2 )
{
        c1[1] ++;
}
if( (parameter[6] & pipe2) == 4 )
{
        c1[2] ++;
}
if( (parameter[6] & pipe3) == 8 )
{
        c1[3] ++;
}
if( (parameter[6] & pipe4) == 16 )
{
        c1[4] ++;
}
if( (parameter[6] & pipe5) == 32 )
```

```
{
        c1[5] ++;
}
if( (parameter[6] & pipe6) == 64 )
{
        c1[6] ++;
}
if( (parameter[6] & pipe7) == 128 )
{
        c1[7] ++;
}
if( (parameter[6] & pipe8) == 256 )
{
        c1[8]++;
}
if( (parameter[6] & pipe9) == 512 )
{
        c1[9] ++;
}
if( (parameter[6] & pipe10) == 1024 )
{
        c1[10] ++;
}
if( (parameter[6] & pipe11) == 2048 )
{
        c1[11] ++;
}
if( (parameter[6] & pipe12) == 4096 )
{
        c1[12] ++;
}
if( (parameter[6] & pipe13) == 8192 )
{
        c1[13] ++;
}
if( (parameter[6] & pipe14) == 16384 )
{
        c1[14] ++;
}
if( (parameter[6] & pipe15) == 32768 )
{
        c1[15] ++;
}

/* locates and adds with bad 3.5v cells */
if( (parameter[7] & pipe0) == 1 )
{
        r3[0] ++;
}
if( (parameter[7] & pipe1) == 2 )
{
        r3[1] ++;
}
if( (parameter[7] & pipe2) == 4 )
{
        r3[2] ++;
}
if( (parameter[7] & pipe3) == 8 )
{
        r3[3] ++;
}
if( (parameter[7] & pipe4) == 16 )
{
        r3[4] ++;
}
```



```
if( (parameter[7] & pipe5) == 32 )
{
        r3[5] ++;
}
if( (parameter[7] & pipe6) == 64 )
{
        r3[6] ++;
}
if( (parameter[7] & pipe7) == 128 )
{
        r3[7] ++;
}
if( (parameter[7] & pipe8) == 256 )
{
        r3[8] ++;
}
if( (parameter[7] & pipe9) == 512 )
{
        r3[9] ++;
}
if( (parameter[7] & pipe10) == 1024 )
{
        r3[10] ++;
}
if( (parameter[7] & pipe11) == 2048 )
{
        r3[11] ++;
}
if( (parameter[7] & pipe12) == 4096 )
{
        r3[12] ++;
}
if( (parameter[7] & pipe13) == 8192 )
{
        r3[13] ++;
}
if( (parameter[7] & pipe14) == 16384 )
{
        r3[14] ++;
}
if( (parameter[7] & pipe15) == 32768 )
{
        r3[15] ++;
}

/*  locates pipes with bad 3.5v cells */
if( (parameter[8] & pipe0) == 1 )
{
        c3[0] ++;
}
if( (parameter[8] & pipe1) == 2 )
{
        c3[1] ++;
}
if( (parameter[8] & pipe2) == 4 )
{
        c3[2] ++;
}
if( (parameter[8] & pipe3) == 8 )
{
        c3[3] ++;
}
if( (parameter[8] & pipe4) == 16 )
{
        c3[4] ++;
```

```
}
if( (parameter[8] & pipe5) == 32 )
{
        c3[5] ++;
}
if( (parameter[8] & pipe6) == 64 )
{
        c3[6] ++;
}
if( (parameter[8] & pipe7) == 128 )
{
        c3[7] ++;
}
if( (parameter[8] & pipe8) == 256 )
{
        c3[8] ++;
}
if( (parameter[8] & pipe9) == 512 )
{
        c3[9] ++;
}
if( (parameter[8] & pipe10) == 1024 )
{
        c3[10] ++;
}
if( (parameter[8] & pipe11) == 2048 )
{
        c3[11] ++;
}
if( (parameter[8] & pipe12) == 4096 )
{
        c3[12] ++;
}
if( (parameter[8] & pipe13) == 8192 )
{
        c3[13] ++;
}
if( (parameter[8] & pipe14) == 16384 )
{
        c3[14] ++;
}
if( (parameter[8] & pipe15) == 32768 )
{
        c3[15] ++;
}


/*  locates pipes with bad 1.5v pipes */
if( (parameter[9] & pipe0) == 1 )
{
        p1[0] ++;
        p1_5[0] ++;
}
if( (parameter[9] & pipe1) == 2 )
{
        p1[1] ++;
        p1_5[1] ++;
}
if( (parameter[9] & pipe2) == 4 )
{
        p1[2] ++;
        p1_5[2] ++;
}
if( (parameter[9] & pipe3) == 8 )
{
        p1[3] ++;
```



```
        p1_5[3] ++;
}
if( (parameter[9] & pipe4) == 16 )
{
        p1[4] ++;
        p1_5[4] ++;
}
if( (parameter[9] & pipe5) == 32 )
{
        p1[5] ++;
        p1_5[5] ++;
}
if( (parameter[9] & pipe6) == 64 )
{
        p1[6] ++;
        p1_5[6] ++;
}
if( (parameter[9] & pipe7) == 128 )
{
        p1[7] ++;
        p1_5[7] ++;
}
if( (parameter[9] & pipe8) == 256 )
{
        p1[8] ++;
        p1_5[8] ++;
}
if( (parameter[9] & pipe9) == 512 )
{
        p1[9] ++;
        p1_5[9] ++;
}
if( (parameter[9] & pipe10) == 1024 )
{
        p1[10] ++;
        p1_5[10] ++;
}
if( (parameter[9] & pipe11) == 2048 )
{
        p1[11] ++;
        p1_5[11] ++;
}
if( (parameter[9] & pipe12) == 4096 )
{
        p1[12] ++;
        p1_5[12] ++;
}
if( (parameter[9] & pipe13) == 8192 )
{
        p1[13] ++;
        p1_5[13] ++;
}
if( (parameter[9] & pipe14) == 16384 )
{
        p1[14] ++;
        p1_5[14] ++;
}
if( (parameter[9] & pipe15) == 32768 )
{
        p1[15] ++;
        p1_5[15] ++;
}
```

```
/*  locates pipes with bad 3.5v pipes */
if( (parameter[10] & pipe0) == 1 )
{
        p3[0] ++;
        p3_5[0] ++;
}
if( (parameter[10] & pipe1) == 2 )
{
        p3[1] ++;
        p3_5[1] ++;
}
if( (parameter[10] & pipe2) == 4 )
{
        p3[2] ++;
        p3_5[2] ++;
}
if( (parameter[10] & pipe3) == 8 )
{
        p3[3] ++;
        p3_5[3] ++;
}
if( (parameter[10] & pipe4) == 16 )
{
        p3[4] ++;
        p3_5[4] ++;
}
if( (parameter[10] & pipe5) == 32 )
{
        p3[5] ++;
        p3_5[5] ++;
}
if( (parameter[10] & pipe6) == 64 )
{
        p3[6] ++;
        p3_5[6] ++;
}
if( (parameter[10] & pipe7) == 128 )
{
        p3[7] ++;
        p3_5[7] ++;
}
if( (parameter[10] & pipe8) == 256 )
{
        p3[8] ++;
        p3_5[8] ++;
}
if( (parameter[10] & pipe9) == 512 )
{
        p3[9] ++;
        p3_5[9] ++;
}
if( (parameter[10] & pipe10) == 1024 )
{
        p3[10] ++;
        p3_5[10] ++;
}
if( (parameter[10] & pipe11) == 2048 )
{
        p3[11] ++;
        p3_5[11] ++;
}
if( (parameter[10] & pipe12) == 4096 )
{
        p3[12] ++;
```



```
        p3_5[12] ++;
}
if( (parameter[10] & pipe13) == 8192 )
{
        p3[13] ++;
        p3_5[13] ++;
}
if( (parameter[10] & pipe14) == 16384 )
{
        p3[14] ++;
        p3_5[14] ++;
}
if( (parameter[10] & pipe15) == 32768 )
{
        p3[15] ++;
        p3_5[15] ++;
}

/*  locates add failures in 1.5v section of amu */
if( (parameter[11] & pipe0) == 1 )
{
        add1[0] ++;
}
if( (parameter[11] & pipe1) == 2 )
{
        add1[1] ++;
}
if( (parameter[11] & pipe2) == 4 )
{
        add1[2] ++;
}
if( (parameter[11] & pipe3) == 8 )
{
        add1[3] ++;
}
if( (parameter[11] & pipe4) == 16 )
{
        add1[4] ++;
}
if( (parameter[11] & pipe5) == 32 )
{
        add1[5] ++;
}
if( (parameter[11] & pipe6) == 64 )
{
        add1[6] ++;
}
if( (parameter[11] & pipe7) == 128 )
{
        add1[7] ++;
}
if( (parameter[11] & pipe8) == 256 )
{
        add1[8] ++;
}
if( (parameter[11] & pipe9) == 512 )
{
        add1[9] ++;
}
if( (parameter[11] & pipe10) == 1026 )
{
        add1[10] ++;
}
if( (parameter[11] & pipe11) == 2048 )
{
        add1[11] ++;
```

```
}
if( (parameter[11] & pipe12) == 4096 )
{
        add1[12] ++;
}
if( (parameter[11] & pipe13) == 8192 )
{
        add1[13] ++;
}
if( (parameter[11] & pipe14) == 16384 )
{
        add1[14] ++;
}
if( (parameter[11] & pipe15) == 32768 )
{
        add1[15] ++;
}

/*  locates add failures in 3.5v section of amu */
if( (parameter[12] & pipe0) == 1 )
{
        add3[0] ++;
}
if( (parameter[12] & pipe1) == 2 )
{
        add3[1] ++;
}
if( (parameter[12] & pipe2) == 4 )
{
        add3[2] ++;
}
if( (parameter[12] & pipe3) == 8 )
{
        add3[3] ++;
}
if( (parameter[12] & pipe4) == 16 )
{
        add3[4] ++;
}
if( (parameter[12] & pipe5) == 32 )
{
        add3[5] ++;
}
if( (parameter[12] & pipe6) == 64 )
{
        add3[6] ++;
}
if( (parameter[12] & pipe7) == 128 )
{
        add3[7] ++;
}
if( (parameter[12] & pipe8) == 256 )
{
        add3[8] ++;
}
if( (parameter[12] & pipe9) == 512 )
{
        add3[9] ++;
}
if( (parameter[12] & pipe10) == 1024 )
{
        add3[10] ++;
}
if( (parameter[12] & pipe11) == 2048 )
{
        add3[11] ++;
```



```
}
if( (parameter[12] & pipe12) == 4096 )
{
        add3[12] ++;
}
if( (parameter[12] & pipe13) == 8192 )
{
        add3[13] ++;
}
if( (parameter[12] & pipe14) == 16384 )
{
        add3[14] ++;
}
if( (parameter[12] & pipe15) == 32768 )
{
        add3[15] ++;
}

/*  counts matrix failures where all of the cells are bad
in the 1.5v
        matrix and 3.5v matrix of the amu tests */

if( parameter[13] == 1 && parameter[14] == 0)
{
        both_failed ++;
}

/*  counts failures where only one matrix fails in either
the 1.5v
        matrix or the 3.5v matrix of amu ( not both ) */


if( parameter[13] == 1 && parameter[14] == 1)
{
        one_failed ++;
}

/* counts 1.5 matrix failures only */
if( parameter[13] == 1 && parameter[14] == 1 && parameter
[1] == 256)
{
        one_five_failed ++;
}

/* counts 3.5v matrix failures only */
if( parameter[13] == 1 && parameter[14] == 1 && parameter
[2] == 256)
{
        three_five_failed ++;
}


if( parameter[1] == 0 && parameter[2] == 0 && parameter[0
] != 0)
{
                fprintf( fp_no_errs , "\n%d ", parameter[
0] );
}

for(loopcntr = 0; loopcntr < 16; loopcntr ++ )
{
        bad_pipes_per_chip_cntr += pf_S[loopcntr];
}
```

```
if( bad_pipes_per_chip_cntr == 0 )
{
        num_bad_pipes_per_chip[0] ++;
}
if( bad_pipes_per_chip_cntr == 1 )
{
        num_bad_pipes_per_chip[1] ++;
}
if( bad_pipes_per_chip_cntr == 2 )
{
        num_bad_pipes_per_chip[2] ++;
}
if( bad_pipes_per_chip_cntr == 3 )
{
        num_bad_pipes_per_chip[3] ++;
}
if( bad_pipes_per_chip_cntr == 4 )
{
        num_bad_pipes_per_chip[4] ++;
}
if( bad_pipes_per_chip_cntr == 5 )
{
        num_bad_pipes_per_chip[5] ++;
}
if( bad_pipes_per_chip_cntr == 6 )
{
        num_bad_pipes_per_chip[6] ++;
}
if( bad_pipes_per_chip_cntr == 7 )
{
        num_bad_pipes_per_chip[7] ++;
}
if( bad_pipes_per_chip_cntr == 8 )
{
        num_bad_pipes_per_chip[8] ++;
}
if( bad_pipes_per_chip_cntr == 9 )
{
        num_bad_pipes_per_chip[9] ++;
}
if( bad_pipes_per_chip_cntr == 10 )
{
        num_bad_pipes_per_chip[10] ++;
}
if( bad_pipes_per_chip_cntr == 11 )
{
        num_bad_pipes_per_chip[11] ++;
}
if( bad_pipes_per_chip_cntr == 12 )
{
        num_bad_pipes_per_chip[12] ++;
}
if( bad_pipes_per_chip_cntr == 13 )
{
        num_bad_pipes_per_chip[13] ++;
}
if( bad_pipes_per_chip_cntr == 14 )
{
        num_bad_pipes_per_chip[14] ++;
}
if( bad_pipes_per_chip_cntr == 15 )
{
        num_bad_pipes_per_chip[15] ++;
}
if( bad_pipes_per_chip_cntr == 16 )
{
```



```
        num_bad_pipes_per_chip[16] ++;
}

for(loopcntr = 0; loopcntr < 16; loopcntr ++)
{
        p1_5[loopcntr] = 0;
        p3_5[loopcntr] = 0;
}

bad_pipes_per_chip_cntr = 0;


// ***********************************************************
//                  This section of code checks to see which
//                  chip failed
// ***********************************************************


if( parameter1[1] == 0 && parameter1[2] == 0 &&
        parameter1[3] == 0 )
{
        tec_good = 1;
}
if( parameter1[1] == 1 && parameter1[2] == 0 &&
        parameter1[3] == 0 )
{
        to_only_failed = 1;
}
if( parameter1[1] == 0 && parameter1[2] == 1 &&
        parameter1[3] == 0 )
{
        p20_only_failed = 1;
}
if( parameter1[1] == 0 && parameter1[2] == 0 &&
        parameter1[3] == 1 )
{
        p100_only_failed = 1;
}
if( parameter1[1] > 1 || parameter1[2] > 1 ||
        parameter1[3] > 1 )
{
        multit_failed = 1;
}

if( parameter[1] == 0 && parameter[2] == 0 )
{
        emu_good = 1;
}

if( parameter[1] != 0 || parameter[2] != 0 )
{
        emu_good = 0;
}
if( parameter[1] > 1 || parameter[2] > 1 )
{
        multia_failed = 1;
}


if( multit_failed == 1 && emu_good == 1 )
```

```
                    {
                        tac_multi_failures_only ++;
                    }
                    if( multia_failed == 1 && tac_good == 1 )
                    {
                        amu_multi_failures_only ++;
                    }
                    if( multit_failed == 1 && multia_failed == 1 )
                    {
                        both_multi_failures ++;
                    }
                    if( tac_good == 0 && amu_good == 1 )
                    {
                        tac_only_failed ++;
                    }

                    if( tac_good == 1 && amu_good == 0 )
                    {
                        amu_only_failed ++;
                    }

                    if( tac_good == 0 && amu_good == 0 )
                    {
                        both_sections_failed ++;
                    }

                    if( parameter[1] == 1 && parameter[2] == 1 && tac_good ==
1 )
                    {
                        one_cell_failure ++;
                    }

            if( to_only_failed == 1 && amu_good == 1 )
            {
                tac_to_only_failure ++;
            }

            if( p20_only_failed == 1 && amu_good == 1 )
            {
                tac_p20_only_failure ++;
            }

            if( p100_only_failed == 1 && amu_good == 1 )
            {
                tac_p100_only_failure ++;
            }

            if( tac_good == 1 && amu_good == 1 )
            {
                good_chip ++;
                                        /*
                                                counts chips with no coun
ted errors.
                                                                probably
chips with errors in 3.5v
                                                                section o
nly
                                        */
            }

                    if( parameter[14] == 1 && (parameter[5] == 255 ||
                        parameter[6] == 255) && alreadycounted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
```



```
                    if( parameter[13] == 1 && (parameter[5] == 255 ||
                        parameter[6] == 255) && alreadycounted == 0)
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
                    if( parameter[14] == 1 && parameter[5] == 255 && alreadycoun
ted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
                    if( parameter[14] == 1 && parameter[6] == 255 && alreadycoun
ted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
                    if( parameter[13] == 1 && parameter[5] == 255 && alreadycoun
ted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
                    if( parameter[13] == 1 && parameter[6] == 255 && alreadycoun
ted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
                    if( parameter[13] > 180 && parameter[5] == 255 && alreadycou
nted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }
                    if( parameter[13] > 180 && parameter[6] == 255 && alreadycou
nted == 0 )
                    {
                        adc_or_ps_failure ++;
                        alreadycounted = 1;
                    }




                  amu_good=tac_good=to_only_failed=p20_only_failed=p100_only_fail
ed=0;
                  multit_failed=multia_failed=alreadycounted=0;
                }

        }
        fclose( fp_no_errt );
        fclose(fp_amu);
    fclose( fp_no_errs );
        fclose(fp_tac);

        if( ( fp_tac = fopen( "c:\\uamac\\results\\tfinal.res", "w" )) != NULL )
        {
            fprintf( fp_tac,"\nTotal number of Timeout failures: %d", Num_of_
timeout );
            fprintf( fp_tac,"\nTotal number of 20ns pulse failures: %d", Num_
of_20ns );
            fprintf( fp_tac,"\nTotal number of 100ns pulse failures: %d", Num
```

```
_of_100ns );
                fprintf( fp_tac,"\nTotal number of chips that had only one 20ns p
ulse failure: %d", one_20ns_failed );
                fprintf( fp_tac,"\nTotal number of chips that had only one 100ns
pulse failure: %d", one_100ns_failed );

                fprintf( fp_tac,"\n\n");
                for(loopcntr = 0; loopcntr < 8; loopcntr ++)
                {
                  fprintf( fp_tac,"\nAt least 1 Chan %d Timeout failure: %d",
                                        loopcntr, to[loopcntr] );
                }


                fprintf( fp_tac,"\n\n");
                for(loopcntr = 0; loopcntr < 8; loopcntr ++)
                {
                        fprintf( fp_tac,"\nTAC/ADC %d 20ns failures: %d",
                                        loopcntr, p20ns[loopcntr] );
                }

                fprintf( fp_tac,"\n\n");
                for(loopcntr = 0; loopcntr < 8; loopcntr ++)
                {
                        fprintf( fp_tac,"\nTAC/ADC %d 100ns failures: %d",
                                        loopcntr, p100ns[loopcntr] );
                }

                fprintf( fp_tac,"\n\nFile count: %d", filecounter);

                }

        fclose(fp_tac);

        if( ( fp_tac = fopen( "c:\\womac\\results\\t.res", "w" )) != NULL )
        {
                fprintf( fp_tac,"\nTotal number of Timeout failures: %d", Num_of_
timeout );
                fprintf( fp_tac,"\nTotal number of 20ns pulse failures: %d", Num_
of_20ns );
                fprintf( fp_tac,"\nTotal number of 100ns pulse failures: %d", Num
_of_100ns );
                fprintf( fp_tac,"\nTotal number of chips that had only one 20ns p
ulse failure: %d", one_20ns_failed );
                fprintf( fp_tac,"\nTotal number of chips that had only one 100ns
pulse failure: %d", one_100ns_failed );

                fprintf( fp_tac,"\n\n");
                fprintf( fp_tac,"\n\nAt least 1 Chan 0->7 Timeout failure: \n");
                for(loopcntr = 0; loopcntr < 8; loopcntr ++)
                {
                  fprintf( fp_tac,"%d, ", to[loopcntr] );
                }


                fprintf( fp_tac,"\n\n");
                fprintf( fp_tac,"\n\nTAC/ADC 0->7 20ns failures: \n");
                for(loopcntr = 0; loopcntr < 8; loopcntr ++)
                {
                        fprintf( fp_tac,"%d, ", p20ns[loopcntr] );
                }

                fprintf( fp_tac,"\n\n");
                fprintf( fp_tac,"\n\nTAC/ADC 0->7 100ns failures: \n");
                for(loopcntr = 0; loopcntr < 8; loopcntr ++)
                {
```



```
                        fprintf( fp_tac,"%d, ", p100ns[loopcntr] );
                }

                fprintf( fp_tac,"\n\nFile count: %d", filecounter);

                }

        fclose(fp_tac);



        if( ( fp_amu = fopen( "c:\\womac\\results\\afinal.res", "w" )) != NULL )
        {
                fprintf( fp_amu,"\nTotal number of 1.5v cell failures: %d", Num_1
cells);
                fprintf( fp_amu,"\nTotal number of 3.5v cell failures: %d", Num_3
cells );
                fprintf( fp_amu,"\nTotal number of 1.5v pipe failures: %d", Num_1
pipes );
                fprintf( fp_amu,"\nTotal number of 3.5v pipe failures: %d", Num_3
pipes );

                fprintf( fp_amu,"\nBoth matrix's completely failed: %d", both_fai
led );

                fprintf( fp_amu,"\nOne matrix completely failed: %d", one_failed
);

                fprintf( fp_amu,"\n1.5v matrix completely failed: %d", one_five_f
ailed );

                fprintf( fp_amu,"\n3.5v matrix completely failed: %d", three_five
_failed );


                fprintf( fp_amu,"\n\nTotal number of failures due to only one bad
cell: %d",
                                        one_cell_failure );
                fprintf( fp_amu,"\n\nTotal number of failures due to only one tac
time out failure: %d",
                                        tac_to_only_failure );
                fprintf( fp_amu,"\n\nTotal number of failures due to only one tac
20ns pulse failure: %d",
                                        tac_p20_only_failure );
                fprintf( fp_amu,"\n\nTotal number of failures due to only one tac
100ns pulse failure: %d",
                                        tac_p100_only_failure );
                fprintf( fp_amu,"\n\nMultiple TAC failures only caused failure: %
d", tac_multi_failures_only );
                fprintf( fp_amu,"\n\nMultiple AMU failures only caused failure: %
d", amu_multi_failures_only );
                fprintf( fp_amu,"\n\nMultiple failures due to both sections cause
d failure: %d", both_multi_failures );
                fprintf( fp_amu,"\n\nADC or Power Supply caused failure: %d", adc
_or_ps_failure);
                fprintf( fp_amu,"\n\nAMU only caused failure: %d", amu_only_faile
d );
                fprintf( fp_amu,"\n\nTAC only caused failure: %d", tac_only_faile
d );
                fprintf( fp_amu,"\n\nBoth sections caused failure: %d", both_sect
ions_failed );
                fprintf( fp_amu,"\n\nGood Chips (due to errors chip.exe [3.5v mat
rix not cleared]): %d", good_chip );
```

```
                fprintf( fp_amu,"\n\nTotal chips failed: %d", tot=(amu_only_failed
+tac_only_failed+both_sections_failed) );
        fprintf( fp_amu,"\nTotal # of chips in bad file: %d", (tot+good_chip) );


                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\nTotal # of at least 1 occurance of a b
ad 1.5v cell in amu add %d: %d",
                                                loopcntr, r1[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\nTotal # of at least 1 occurance of a b
ad 1.5v cell in pipe %d: %d",
                                                loopcntr, c1[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\nTotal # of at least 1 occurance of a b
ad 3.5v cell in amu add %d: %d",
                                                loopcntr, r3[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\nTotal # of at least 1 occurance of a b
ad 3.5v cell in pipe %d: %d",
                                                loopcntr, c3[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\n1.5v Pipe %d failures: %d", loopcntr,
p1[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\n3.5v Pipe %d failures: %d", loopcntr,
p3[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                {
                        fprintf(fp_amu,"\nNumber of chips with %2.1d bad pipes is
: %d",
                                                loopcntr, num_bad_pipes_per_chip[
loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\nAMU add's %d failures in 1.5v amu sect
ion : %d",
                                                loopcntr, add1[loopcntr] );
```



```
                }

                fprintf( fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"\nAMU add's %d failures in 3.5v amu sect
ion : %d",
                                                loopcntr, add3[loopcntr] );
                }



                fprintf( fp_amu,"\n\nFile count: %d\n", filecounter);

        }
        fclose(fp_amu);

        if( ( fp_amu = fopen( "c:\\womac\\results\\a.res", "w" )) != NULL )
        {
                fprintf( fp_amu,"\nTotal number of 1.5v cell failures: %d", Num_1
cells);
                fprintf( fp_amu,"\nTotal number of 3.5v cell failures: %d", Num_3
cells );
                fprintf( fp_amu,"\nTotal number of 1.5v pipe failures: %d", Num_1
pipes );
                fprintf( fp_amu,"\nTotal number of 3.5v pipe failures: %d", Num_3
pipes );

                fprintf( fp_amu,"\nBoth matrix's completely failed: %d", both_fai
led );

                fprintf( fp_amu,"\nOne matrix completely failed: %d", one_failed
);

                fprintf( fp_amu,"\n1.5v matrix completely failed: %d", one_five_f
ailed );

                fprintf( fp_amu,"\n3.5v matrix completely failed: %d", three_five
_failed );

                fprintf( fp_amu,"\n\nTotal number of failures due to only one bad
cell: %d",
                                                one_cell_failure );
                fprintf( fp_amu,"\n\nTotal number of failures due to only one tac
time out failure: %d",
                                                tac_to_only_failure );
                fprintf( fp_amu,"\n\nTotal number of failures due to only one tac
20ns pulse failure: %d",
                                                tac_p20_only_failure );
                fprintf( fp_amu,"\n\nTotal number of failures due to only one tac
100ns pulse failure: %d",
                                                tac_p100_only_failure );
                fprintf( fp_amu,"\n\nMultiple TAC failures only caused failure: %
d", tac_multi_failures_only );
                fprintf( fp_amu,"\n\nMultiple AMU failures only caused failure: %
d", amu_multi_failures_only );
                fprintf( fp_amu,"\n\nADC or Power Supply caused failure: %d", adc
_or_ps_failure);
                fprintf( fp_amu,"\n\n\nAMU only caused failure: %d", amu_only_fai
led );
                fprintf( fp_amu,"\n\nTAC only caused failure: %d", tac_only_faile
d );
                fprintf( fp_amu,"\n\nBoth sections failure: %d", both_sections_fa
iled );
```

```
                fprintf( fp_amu,"\n\nGood Chips (due to errors chip.exe (3.5v mat
rix not cleared)]: %d", good_chip );
                fprintf( fp_amu,"\n\nTotal chips failed: %d", tot=(amu_only_faile
d+tac_only_failed+both_sections_failed) );
        fprintf( fp_amu,"\nTotal # of chips in bad file: %d", tot );

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nTotal # of at least 1 occurance of a bad 1.5v
cell in amu add \n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", r1[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nTotal # of at least 1 occurance of a bad 1.5v
cell in amu pipe \n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", c1[loopcntr] );
                }


                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nTotal # of at least 1 occurance of a bad 3.5v
cell in amu add \n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", r3[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nTotal # of at least 1 occurance of a bad 3.5v
cell in amu pipe \n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", c3[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\n1.5v Pipe failures: \n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", p1[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\n3.5v Pipe failures: \n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", p3[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nNumber of chips with 0->16 bad pipes is:\n");
                for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                {
                        fprintf(fp_amu,"%d, ", num_bad_pipes_per_chip[loopcntr] )
;
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nAMU add's 0->16 failures in 1.5v amu section :
\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
```



```
                        fprintf( fp_amu,"%d, ", add1[loopcntr] );
                }

                fprintf( fp_amu,"\n\n");
                fprintf( fp_amu,"\nAMU add's 0->16 failures in 3.5v amu section :
\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf( fp_amu,"%d, ", add3[loopcntr] );
                }



                fprintf( fp_amu,"\n\nFile count: %d\n", filecounter);

        }
        fclose(fp_amu);



        /* this routine checks no_err.amu & no_err.tac for same chip numbers
                so that chips that have no errors can be eliminated from the
                orginal badchip.num file.  This was caused by problems in chip.ex
e
                program
        */
        fp_a = fopen("c:\\womac\\nochip#\\no_err.amu", "r" );
        fp_gd_chp = fopen("c:\\womac\\goodchip\\good.chp", "w" );


        while( ( fgets(chnuma, 8, fp_a) ) != NULL )
        {
                Parse_data( chnuma, &Count, &(worda[0]) );

                achip = atoi(worda[0]);
                if( achip > 4999 && achip < 7600 )
                {
                        fp_t = fopen("c:\\womac\\nochip#\\no_err.tac", "r" );
                        while( ( fgets(chnumt, 8, fp_t) ) != NULL )
                        {
                                Parse_data( chnumt, &Count, &(wordt[0]) );

                                tchip = atoi(wordt[0]);
                                if( tchip == achip )
                                {
                                        fprintf( fp_gd_chp, "%d, ", tchip );
                                }
                        }
                        fclose( fp_t);
                }
        }
        fclose( fp_a );
        fclose( fp_gd_chp );

        printf("\nFiles generated by this program:");
        printf("\nc:\\womac\\results\\offinal.res");
        printf("\nc:\\womac\\results\\a.res");
        printf("\nc:\\womac\\results\\tfinal.res");
        printf("\nc:\\womac\\results\\t.res");
        printf("\nc:\\womac\\nochip#\\no_err.amu");
        printf("\nc:\\womac\\nochip#\\no_err.tac");
        printf("\nc:\\womac\\goodchip\\good.chp");
}
```

```
/*****************************************************************************/

/*
        Function Name:  Parse_data ( & Parse_J2_data )
        Requires:       pointer to a predefined character array (buffer)
                                                pointer to ?????
                                                pointer to a pointer to a charac
ter array with desired
                                                 size of parsed word
        Returns: void
        Purpose: seperates a buffer full (this case one line) of data into a
                                'word' array that is predefined in the calling pr
g.  The
                                determination of the buffer length is the charact
ers that
                                are checked for in the first while statement.  Th
e maximum
                                buffer size is actually determined in the calling
 prg.
*/

/*****************************************************************************/



void Parse_data( char *buf, int *Count, char **Word)
{
        *Word = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (i.e. \0) character in the contents
 of the current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        Word[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
```



```
toring
                                of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                        (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;
                }
        }
        *buf = '\0';
}
```

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = COLLECT2
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\WOMAC\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC =
FIRSTCPP =
RC = rc
CFLAGS_D_DEXE = /nologo /W3 /FR /G2 /Zi /D_DEBUG /Od /AM /D_DOS /Fd"COLLECT2.PDB"

CFLAGS_R_DEXE = /nologo /W3 /FR /G2 /DNDEBUG /Gs /Ox /AM /D_DOS
LFLAGS_D_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /CO /STACK:5120
LFLAGS_R_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /STACK:5120
LIBS_D_DEXE = oldnames mlibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = COLLECT2.SBR


COLLECT2_DEP =

all:    $(PROJ).EXE $(PROJ).BSC

COLLECT2.OBJ:   COLLECT2.C $(COLLECT2_DEP)
        $(CC) $(CFLAGS) $(CUSEPCHFLAG) /c COLLECT2.C

$(PROJ).EXE::   COLLECT2.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
COLLECT2.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
```



```
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
/*
        collect.C
        WRITTEN BY:     MICHAEL WOMAC
        DATE: 5 Jan 1995 - 9 Jan 1995
        Version: 1.0
*/

/*************************************************************************/
/********                      Headers                          ********/
/*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*************************************************************************/
/********                      Definitions                      ********/
/*************************************************************************/

#define word_size 16
#define line_size 256

/**** MASKS ****/


/*************************************************************************/
/********                      Global Variables                 ********/
/*************************************************************************/



/*************************************************************************/
/********                      Structures                       ********/
/*************************************************************************/



/*************************************************************************/
/********                      Functions                        ********/
/*************************************************************************/

void Parse_data( char *, int *, char ** );

/*************************************************************************/
/********                      Code                             ********/
/*************************************************************************/



/************************************************************************/

/*
        Function Name:  main
        Requires:   void
        Returns:    void
        Purpose: To collect Pb chip statistical data from output files generated
                                by test2.exe.
*/

/************************************************************************/


void main( void )
{
        /**** FILE POINTERS ****/
   FILE *fp_amu;
```



```
        /**** COUNTERS ****/
        int Count, x, y;
        int filecounter = 0;
        int rowcounter = 0;
     int loopcntr=0;
        long bad_cells_per_amuadd = 0 ;
        long bad_cells_per_pipe = 0 ;
        int bad_amu_add = 0;

        /**** CHARACTER ARRAYS ****/
        char *word[ 20 ];
        char line_of_data[ line_size ];

        /**** INTEGER ARRAYS ****/
        long parameter[20];
        long bad_cells_in_1vrow[16];
        long bad_cells_in_3vrow[16];
        long bad_cells_in_1vpipe[16];
        long bad_cells_in_3vpipe[16];
        long bad_cells_at_amuadd[16][17];
        long bad_cells_at_pipe[16][17];

        int row_failures[17];
        int pipe_failures[17];
        int chips_with_badamuadd[17];

                        /* Initialize Arrays */
                        for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                        {
                                bad_cells_in_1vrow[loopcntr] = 0;
                                bad_cells_in_3vrow[loopcntr] = 0;
                                bad_cells_in_1vpipe[loopcntr] = 0;
                                bad_cells_in_3vpipe[loopcntr] = 0;
                        }

                        for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                        {
                                row_failures[loopcntr] = 0;
                                pipe_failures[loopcntr] = 0;
                                chips_with_badamuadd[loopcntr] = 0;
                        }

           for(x = 0; x < 16; x++ )
           {
               for(y = 0; y < 17; y ++ )
               {
                        bad_cells_at_amuadd[x][y] = 0;
                        bad_cells_at_pipe[x][y] = 0;
               }
           }


/*************************************************************************/
/*************************************************************************/
/*                          AMU SECTION                                  */
/*************************************************************************/
/*************************************************************************/


        /*
                opens file that contains data
        */
        if(  (fp_amu = fopen("c:\\womac\\results\\results2.amu", "r"))  != NULL)
        {
```

```
                /*
                        get first line of data
                */
                while ( (fgets(line_of_data, line_size, fp_amu) != NULL) )
                {

                        Parse_data( line_of_data, &Count, &(word[0]) );

                        /*
                                Converts character array into integer array
                                Each pass through the above while loop grabs a li
ne
                                of data that is stored in word[0] thru word[15] a
nd
                                converted to intergers in parameter[0] thru param
eter[15]
                                Each group (line of data) is the number of bad ce
lls in
                                the corresponding amuadd/pipe.
                                Line #1 is # of bad cells in pipes 0-15 for 1.5v
data
                                Line #2 is # of bad cells in amuadd 0-15 for 1.5v
 data
                                Line #3 is # of bad cells in pipes 0-15 for 3.5v
data
                                Line #4 is # of bad cells in amuadd 0-15 for 3.5v
 data
                        */

                        parameter[0] = atoi(word[0]); /*  */
                        parameter[1] = atoi(word[1]); /*  */
                        parameter[2] = atoi(word[2]); /*  */
                        parameter[3] = atoi(word[3]); /*  */
                        parameter[4] = atoi(word[4]); /*  */
                        parameter[5] = atoi(word[5]); /*  */
                        parameter[6] = atoi(word[6]); /*  */
                        parameter[7] = atoi(word[7]); /*  */
                        parameter[8] = atoi(word[8]); /*  */
                        parameter[9] = atoi(word[9]); /*  */
                        parameter[10] = atoi(word[10]); /*  */
                        parameter[11] = atoi(word[11]); /*  */
                        parameter[12] = atoi(word[12]); /*  */
                        parameter[13] = atoi(word[13]); /*  */
                        parameter[14] = atoi(word[14]);
                        parameter[15] = atoi(word[15]);


                        if( rowcounter != 0 )
                        {
/*****************************************************************************/
/*                                                                           */
                                ARU PIPES
                        */
/*****************************************************************************/

                                /* # of bad cells in pipe 0->15 for 1.5v data */
                                if(rowcounter == 1 )
                                {
                                        for(loopcntr = 0; loopcntr < 16; loopcntr
 ++ )

                                        {
                                                bad_cells_in_1vpipe[loopcntr] +=
parameter[loopcntr];
                                                bad_cells_per_pipe += parameter[l
oopcntr];
```



```
                                        /*
failures due to

                                        This section while count the # of
                                        0->16 bad cells in each amuadd.
                                        */


                                        if( bad_cells_per_pipe == 0 )
                                        {
                                                pipe_failures[0] ++;
                                        }
                                        if( bad_cells_per_pipe == 1 )
                                        {
                                                pipe_failures[1] ++;
                                        }
                                        if( bad_cells_per_pipe == 2 )
                                        {
                                                pipe_failures[2] ++;
                                        }
                                        if( bad_cells_per_pipe == 3 )
                                        {
                                                pipe_failures[3] ++;
                                        }
                                        if( bad_cells_per_pipe == 4 )
                                        {
                                                pipe_failures[4] ++;
                                        }
                                        if( bad_cells_per_pipe == 5 )
                                        {
                                                pipe_failures[5] ++;
                                        }
                                        if( bad_cells_per_pipe == 6 )
                                        {
                                                pipe_failures[6] ++;
                                        }
                                        if( bad_cells_per_pipe == 7 )
                                        {
                                                pipe_failures[7] ++;
                                        }
                                        if( bad_cells_per_pipe == 8 )
                                        {
                                                pipe_failures[8] ++;
                                        }
                                        if( bad_cells_per_pipe == 9 )
                                        {
                                                pipe_failures[9] ++;
                                        }
                                        if( bad_cells_per_pipe == 10 )
                                        {
                                                pipe_failures[10] ++;
                                        }
                                        if( bad_cells_per_pipe == 11 )
                                        {
                                                pipe_failures[11] ++;
                                        }
                                        if( bad_cells_per_pipe == 12 )
                                        {
                                                pipe_failures[12] ++;
                                        }
                                        if( bad_cells_per_pipe == 13 )
                                        {
                                                pipe_failures[13] ++;
                                        }
                                        if( bad_cells_per_pipe == 14 )
                                        {
                                                pipe_failures[14] ++;
```

```
                                        }
                                        if( bad_cells_per_pipe == 15 )
                                        {
                                                pipe_failures[15] ++;
                                        }
                                        if( bad_cells_per_pipe == 16 )
                                        {
                                                pipe_failures[16] ++;
                                        }
                                        if( loopcntr == 0 )
                                        {
                                if( bad_cells_per_pipe == 0 )
                                                {
                                                        bad_cells_at_pipe
[0][0] ++;                                              }
                                                if( bad_cells_per_pipe ==
 1 )                                            {
                                                        bad_cells_at_pipe
[0][1] ++;                                              }
                                                if( bad_cells_per_pipe ==
 2 )                                            {
                                                        bad_cells_at_pipe
[0][2] ++;                                              }
                                                if( bad_cells_per_pipe ==
 3 )                                            {
                                                        bad_cells_at_pipe
[0][3] ++;                                              }
                                                if( bad_cells_per_pipe ==
 4 )                                            {
                                                        bad_cells_at_pipe
[0][4] ++;                                              }
                                                if( bad_cells_per_pipe ==
 5 )                                            {
                                                        bad_cells_at_pipe
[0][5] ++;                                              }
                                                if( bad_cells_per_pipe ==
 6 )                                            {
                                                        bad_cells_at_pipe
[0][6] ++;                                              }
                                                if( bad_cells_per_pipe ==
 7 )                                            {
                                                        bad_cells_at_pipe
[0][7] ++;                                              }
                                                if( bad_cells_per_pipe ==
 8 )                                            {
                                                        bad_cells_at_pipe
[0][8] ++;                                              }
```



```
                                                if( bad_cells_per_pipe ==
 9 )                                            {
                                                        bad_cells_at_pipe
[0][9] ++;                                              }
                                                if( bad_cells_per_pipe ==
 10 )                                           {
                                                        bad_cells_at_pipe
[0][10] ++;                                             }
                                                if( bad_cells_per_pipe ==
 11 )                                           {
                                                        bad_cells_at_pipe
[0][11] ++;                                             }
                                                if( bad_cells_per_pipe ==
 12 )                                           {
                                                        bad_cells_at_pipe
[0][12] ++;                                             }
                                                if( bad_cells_per_pipe ==
 13 )                                           {
                                                        bad_cells_at_pipe
[0][13] ++;                                             }
                                                if( bad_cells_per_pipe ==
 14 )                                           {
                                                        bad_cells_at_pipe
[0][14] ++;                                             }
                                                if( bad_cells_per_pipe ==
 15 )                                           {
                                                        bad_cells_at_pipe
[0][15] ++;                                             }
                                                if( bad_cells_per_pipe ==
 16 )                                           {
                                                        bad_cells_at_pipe
[0][16] ++;                                             }
                                        }
                                        if( loopcntr == 1 )
                                        {
                                if( bad_cells_per_pipe == 0 )
                                                {
                                                        bad_cells_at_pipe
[1][0] ++;                                              }
                                                if( bad_cells_per_pipe ==
 1 )                                            {
                                                        bad_cells_at_pipe
[1][1] ++;                                              }
                                                if( bad_cells_per_pipe ==
 2 )
```

```
[1][2] ++;
 3 )
[1][3] ++;
 4 )
[1][4] ++;
 5 )
[1][5] ++;
 6 )
[1][6] ++;
 7 )
[1][7] ++;
 8 )
[1][8] ++;
 9 )
[1][9] ++;
 10 )
[1][10] ++;
 11 )
[1][11] ++;
 12 )
[1][12] ++;
 13 )
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
```



```
[1][13] ++;
 14 )
[1][14] ++;
 15 )
[1][15] ++;
 16 )
[1][16] ++;
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
        }
        if( loopcntr == 2 )
        {
if( bad_cells_per_pipe == 0 )
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
                if( bad_cells_per_pipe ==
                {
                        bad_cells_at_pipe
                }
[2][0] ++;
 1 )
[2][1] ++;
 2 )
[2][2] ++;
 3 )
[2][3] ++;
 4 )
[2][4] ++;
 5 )
[2][5] ++;
 6 )
```

```
[2][6] ++;
 7 )
[2][7] ++;
 8 )
[2][8] ++;
 9 )
[2][9] ++;
 10 )
[2][10] ++;
 11 )
[2][11] ++;
 12 )
[2][12] ++;
 13 )
[2][13] ++;
 14 )
[2][14] ++;
 15 )
[2][15] ++;
 16 )
[2][16] ++;
                    }
          if( loopcntr == 3 )
                    {
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
```



```
          if( bad_cells_per_pipe == 0 )
                    {
                              bad_cells_at_pipe
[3][0] ++;
 1 )
[3][1] ++;
 2 )
[3][2] ++;
 3 )
[3][3] ++;
 4 )
[3][4] ++;
 5 )
[3][5] ++;
 6 )
[3][6] ++;
 7 )
[3][7] ++;
 8 )
[3][8] ++;
 9 )
[3][9] ++;
 10 )
[3][10] ++;
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
                    if( bad_cells_per_pipe ==
                    {
                              bad_cells_at_pipe
                    }
```

```
 11 )
[3][11] ++;
 12 )
[3][12] ++;
 13 )
[3][13] ++;
 14 )
[3][14] ++;
 15 )
[3][15] ++;
 16 )
[3][16] ++;
                    )
          if( loopcntr == 4 )
                    {
     if( bad_cells_per_pipe == 0 )
 0 )
[4][0] ++;
 1 )
[4][1] ++;
 2 )
[4][2] ++;
 3 )
[4][3] ++;
 4 )
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        )
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
```



```
[4][4] ++;
 5 )
[4][5] ++;
 6 )
[4][6] ++;
 7 )
[4][7] ++;
 8 )
[4][8] ++;
 9 )
[4][9] ++;
 10 )
[4][10] ++;
 11 )
[4][11] ++;
 12 )
[4][12] ++;
 13 )
[4][13] ++;
 14 )
[4][14] ++;
 15 )
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
```

```
                                                {
                                                        bad_cells_at_pipe
[4][15] ++;                                             }
                                                if( bad_cells_per_pipe ==
 16 )                                           {
                                                        bad_cells_at_pipe
[4][16] ++;                                             }
                                        }

                        if( loopcntr == 5 )
                                        {
                                if( bad_cells_per_pipe == 0 )
                                                {
                                                        bad_cells_at_pipe
[5][0] ++;                                              }
                                                if( bad_cells_per_pipe ==
 1 )                                            {
                                                        bad_cells_at_pipe
[5][1] ++;                                              }
                                                if( bad_cells_per_pipe ==
 2 )                                            {
                                                        bad_cells_at_pipe
[5][2] ++;                                              }
                                                if( bad_cells_per_pipe ==
 3 )                                            {
                                                        bad_cells_at_pipe
[5][3] ++;                                              }
                                                if( bad_cells_per_pipe ==
 4 )                                            {
                                                        bad_cells_at_pipe
[5][4] ++;                                              }
                                                if( bad_cells_per_pipe ==
 5 )                                            {
                                                        bad_cells_at_pipe
[5][5] ++;                                              }
                                                if( bad_cells_per_pipe ==
 6 )                                            {
                                                        bad_cells_at_pipe
[5][6] ++;                                              }
                                                if( bad_cells_per_pipe ==
 7 )                                            {
                                                        bad_cells_at_pipe
[5][7] ++;                                              }
                                                if( bad_cells_per_pipe ==
 8 )                                            {
```



```
                                                        bad_cells_at_pipe
[5][8] ++;                                              }
                                                if( bad_cells_per_pipe ==
 9 )                                            {
                                                        bad_cells_at_pipe
[5][9] ++;                                              }
                                                if( bad_cells_per_pipe ==
 10 )                                           {
                                                        bad_cells_at_pipe
[5][10] ++;                                             }
                                                if( bad_cells_per_pipe ==
 11 )                                           {
                                                        bad_cells_at_pipe
[5][11] ++;                                             }
                                                if( bad_cells_per_pipe ==
 12 )                                           {
                                                        bad_cells_at_pipe
[5][12] ++;                                             }
                                                if( bad_cells_per_pipe ==
 13 )                                           {
                                                        bad_cells_at_pipe
[5][13] ++;                                             }
                                                if( bad_cells_per_pipe ==
 14 )                                           {
                                                        bad_cells_at_pipe
[5][14] ++;                                             }
                                                if( bad_cells_per_pipe ==
 15 )                                           {
                                                        bad_cells_at_pipe
[5][15] ++;                                             }
                                                if( bad_cells_per_pipe ==
 16 )                                           {
                                                        bad_cells_at_pipe
[5][16] ++;                                             }
                                        }
                        if( loopcntr == 6 )
                                        {
                                if( bad_cells_per_pipe == 0 )
                                                {
                                                        bad_cells_at_pipe
[6][0] ++;                                              }
                                                if( bad_cells_per_pipe ==
 1 )                                            {
                                                        bad_cells_at_pipe
[6][1] ++;
```

01/27/1995 12:42 Filename: COLLECT2.C Page 15

```
}
if( bad_cells_per_pipe ==
 2 )
{
        bad_cells_at_pipe
[6][2] ++;
}
if( bad_cells_per_pipe ==
 3 )
{
        bad_cells_at_pipe
[6][3] ++;
}
if( bad_cells_per_pipe ==
 4 )
{
        bad_cells_at_pipe
[6][4] ++;
}
if( bad_cells_per_pipe ==
 5 )
{
        bad_cells_at_pipe
[6][5] ++;
}
if( bad_cells_per_pipe ==
 6 )
{
        bad_cells_at_pipe
[6][6] ++;
}
if( bad_cells_per_pipe ==
 7 )
{
        bad_cells_at_pipe
[6][7] ++;
}
if( bad_cells_per_pipe ==
 8 )
{
        bad_cells_at_pipe
[6][8] ++;
}
if( bad_cells_per_pipe ==
 9 )
{
        bad_cells_at_pipe
[6][9] ++;
}
if( bad_cells_per_pipe ==
 10 )
{
        bad_cells_at_pipe
[6][10] ++;
}
if( bad_cells_per_pipe ==
 11 )
{
        bad_cells_at_pipe
[6][11] ++;
}
if( bad_cells_per_pipe ==
 12 )
{
        bad_cells_at_pipe
[6][12] ++;
```

01/27/1995 12:42 Filename: COLLECT2.C Page 16

```
}
if( bad_cells_per_pipe ==
 13 )
{
        bad_cells_at_pipe
[6][13] ++;
}
if( bad_cells_per_pipe ==
 14 )
{
        bad_cells_at_pipe
[6][14] ++;
}
if( bad_cells_per_pipe ==
 15 )
{
        bad_cells_at_pipe
[6][15] ++;
}
if( bad_cells_per_pipe ==
 16 )
{
        bad_cells_at_pipe
[6][16] ++;
}
        }
    if( loopcntr == 7 )
        {
        if( bad_cells_per_pipe == 0 )
{
        bad_cells_at_pipe
[7][0] ++;
}
if( bad_cells_per_pipe ==
 1 )
{
        bad_cells_at_pipe
[7][1] ++;
}
if( bad_cells_per_pipe ==
 2 )
{
        bad_cells_at_pipe
[7][2] ++;
}
if( bad_cells_per_pipe ==
 3 )
{
        bad_cells_at_pipe
[7][3] ++;
}
if( bad_cells_per_pipe ==
 4 )
{
        bad_cells_at_pipe
[7][4] ++;
}
if( bad_cells_per_pipe ==
 5 )
{
        bad_cells_at_pipe
[7][5] ++;
}
if( bad_cells_per_pipe ==
```

```
 6 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][6] ++;                                      if( bad_cells_per_pipe ==
 7 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][7] ++;                                      if( bad_cells_per_pipe ==
 8 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][8] ++;                                      if( bad_cells_per_pipe ==
 9 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][9] ++;                                      if( bad_cells_per_pipe ==
 10 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][10] ++;                                     if( bad_cells_per_pipe ==
 11 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][11] ++;                                     if( bad_cells_per_pipe ==
 12 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][12] ++;                                     if( bad_cells_per_pipe ==
 13 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][13] ++;                                     if( bad_cells_per_pipe ==
 14 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][14] ++;                                     if( bad_cells_per_pipe ==
 15 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][15] ++;                                     if( bad_cells_per_pipe ==
 16 )
                                                {
                                                        bad_cells_at_pipe
                                                }
[7][16] ++;
                                        }
```



```
                                if( loopcntr == 8 )
                                                {
                                        if( bad_cells_per_pipe == 0 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][0] ++;                                                      if( bad_cells_per_pipe ==
 1 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][1] ++;                                                      if( bad_cells_per_pipe ==
 2 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][2] ++;                                                      if( bad_cells_per_pipe ==
 3 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][3] ++;                                                      if( bad_cells_per_pipe ==
 4 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][4] ++;                                                      if( bad_cells_per_pipe ==
 5 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][5] ++;                                                      if( bad_cells_per_pipe ==
 6 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][6] ++;                                                      if( bad_cells_per_pipe ==
 7 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][7] ++;                                                      if( bad_cells_per_pipe ==
 8 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][8] ++;                                                      if( bad_cells_per_pipe ==
 9 )
                                                                {
                                                                        bad_cells_at_pipe
                                                                }
[8][9] ++;                                                      if( bad_cells_per_pipe ==
 10 )
```

```
                                        {
                                                bad_cells_at_pipe
                                        }
(8)(10) ++;                             if( bad_cells_per_pipe ==
 11 )                                   {
(8)(11) ++;                                     bad_cells_at_pipe
 12 )                                   }
(8)(12) ++;                             if( bad_cells_per_pipe ==
 13 )                                   {
(8)(13) ++;                                     bad_cells_at_pipe
 14 )                                   }
(8)(14) ++;                             if( bad_cells_per_pipe ==
 15 )                                   {
(8)(15) ++;                                     bad_cells_at_pipe
 16 )                                   }
(8)(16) ++;                             if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                }

        if( loopcntr == 9 )
                                {
                if( bad_cells_per_pipe == 0 )
                                        {
                                                bad_cells_at_pipe
                                        }
(9)(0) ++;                              if( bad_cells_per_pipe ==
 1 )                                    {
(9)(1) ++;                                      bad_cells_at_pipe
 2 )                                    }
(9)(2) ++;                              if( bad_cells_per_pipe ==
 3 )                                    {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
```



```
                                                bad_cells_at_pipe
                                        }
(9)(3) ++;                              if( bad_cells_per_pipe ==
 4 )                                    {
(9)(4) ++;                                      bad_cells_at_pipe
 5 )                                    }
(9)(5) ++;                              if( bad_cells_per_pipe ==
 6 )                                    {
(9)(6) ++;                                      bad_cells_at_pipe
 7 )                                    }
(9)(7) ++;                              if( bad_cells_per_pipe ==
 8 )                                    {
(9)(8) ++;                                      bad_cells_at_pipe
 9 )                                    }
(9)(9) ++;                              if( bad_cells_per_pipe ==
 10 )                                   {
(9)(10) ++;                                     bad_cells_at_pipe
 11 )                                   }
(9)(11) ++;                             if( bad_cells_per_pipe ==
 12 )                                   {
(9)(12) ++;                                     bad_cells_at_pipe
 13 )                                   }
(9)(13) ++;                             if( bad_cells_per_pipe ==
 14 )                                   {
```

```
[9][14] ++;
 15 )
[9][15] ++;
 16 )
[9][16] ++;
                                    }

                            if( loopcntr == 10 )
                                    {
                                if( bad_cells_per_pipe == 0 )
[10][0] ++;
 1 )
[10][1] ++;
 2 )
[10][2] ++;
 3 )
[10][3] ++;
 4 )
[10][4] ++;
 5 )
[10][5] ++;
 6 )
[10][6] ++;
 7 )
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
```



```
[10][7] ++;
 8 )
[10][8] ++;
 9 )
[10][9] ++;
 10 )
[10][10] ++;
 11 )
[10][11] ++;
 12 )
[10][12] ++;
 13 )
[10][13] ++;
 14 )
[10][14] ++;
 15 )
[10][15] ++;
 16 )
[10][16] ++;
                                    }

                            if( loopcntr == 11 )
                                    {
                                if( bad_cells_per_pipe == 0 )
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
```

```
[11][0] ++;
 1 )
[11][1] ++;
 2 )
[11][2] ++;
 3 )
[11][3] ++;
 4 )
[11][4] ++;
 5 )
[11][5] ++;
 6 )
[11][6] ++;
 7 )
[11][7] ++;
 8 )
[11][8] ++;
 9 )
[11][9] ++;
 10 )
[11][10] ++;
 11 )
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
```



```
[11][11] ++;
 12 )
[11][12] ++;
 13 )
[11][13] ++;
 14 )
[11][14] ++;
 15 )
[11][15] ++;
 16 )
[11][16] ++;
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        )

                if( loopcntr == 12 )
                                        {
                        if( bad_cells_per_pipe == 0 )
                                        {
                                                bad_cells_at_pipe
                                        }
[12][0] ++;
 1 )
[12][1] ++;
 2 )
[12][2] ++;
 3 )
[12][3] ++;
 4 )
[12][4] ++;
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
```

```
 5 )
[12][5] ++;
 6 )
[12][6] ++;
 7 )
[12][7] ++;
 8 )
[12][8] ++;
 9 )
[12][9] ++;
 10 )
[12][10] ++;
 11 )
[12][11] ++;
 12 )
[12][12] ++;
 13 )
[12][13] ++;
 14 )
[12][14] ++;
 15 )
[12][15] ++;
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
```



```
 16 )
[12][16] ++;
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                )

        if( loopcntr == 13 )
                                (
                if( bad_cells_per_pipe == 0 )
                                        {
                                                bad_cells_at_pipe
[13][0] ++;                             }
 1 )                                    if( bad_cells_per_pipe ==
                                        {
[13][1] ++;                                     bad_cells_at_pipe
 2 )                                    }
                                        if( bad_cells_per_pipe ==
[13][2] ++;                             {
 3 )                                            bad_cells_at_pipe
                                        }
[13][3] ++;                             if( bad_cells_per_pipe ==
 4 )                                    {
                                                bad_cells_at_pipe
[13][4] ++;                             }
 5 )                                    if( bad_cells_per_pipe ==
                                        {
[13][5] ++;                                     bad_cells_at_pipe
 6 )                                    }
                                        if( bad_cells_per_pipe ==
[13][6] ++;                             {
 7 )                                            bad_cells_at_pipe
                                        }
[13][7] ++;                             if( bad_cells_per_pipe ==
 8 )                                    {
                                                bad_cells_at_pipe
[13][8] ++;                             }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                        if( bad_cells_per_pipe ==
                                        {
                                                bad_cells_at_pipe
                                        }
                                )
```

```
                        if( bad_cells_per_pipe == 9 )
                        {
                                bad_cells_at_pipe[13][9] ++;
                        }
                        if( bad_cells_per_pipe == 10 )
                        {
                                bad_cells_at_pipe[13][10] ++;
                        }
                        if( bad_cells_per_pipe == 11 )
                        {
                                bad_cells_at_pipe[13][11] ++;
                        }
                        if( bad_cells_per_pipe == 12 )
                        {
                                bad_cells_at_pipe[13][12] ++;
                        }
                        if( bad_cells_per_pipe == 13 )
                        {
                                bad_cells_at_pipe[13][13] ++;
                        }
                        if( bad_cells_per_pipe == 14 )
                        {
                                bad_cells_at_pipe[13][14] ++;
                        }
                        if( bad_cells_per_pipe == 15 )
                        {
                                bad_cells_at_pipe[13][15] ++;
                        }
                        if( bad_cells_per_pipe == 16 )
                        {
                                bad_cells_at_pipe[13][16] ++;
                        }
                }

        if( loopcntr == 14 )
                {
                if( bad_cells_per_pipe == 0 )
                        {
                                bad_cells_at_pipe[14][0] ++;
                        }
                        if( bad_cells_per_pipe == 1 )
                        {
                                bad_cells_at_pipe[14][1] ++;
                        }
                        if( bad_cells_per_pipe ==
```



```
2 )
                        {
                                bad_cells_at_pipe[14][2] ++;
                        }
                        if( bad_cells_per_pipe == 3 )
                        {
                                bad_cells_at_pipe[14][3] ++;
                        }
                        if( bad_cells_per_pipe == 4 )
                        {
                                bad_cells_at_pipe[14][4] ++;
                        }
                        if( bad_cells_per_pipe == 5 )
                        {
                                bad_cells_at_pipe[14][5] ++;
                        }
                        if( bad_cells_per_pipe == 6 )
                        {
                                bad_cells_at_pipe[14][6] ++;
                        }
                        if( bad_cells_per_pipe == 7 )
                        {
                                bad_cells_at_pipe[14][7] ++;
                        }
                        if( bad_cells_per_pipe == 8 )
                        {
                                bad_cells_at_pipe[14][8] ++;
                        }
                        if( bad_cells_per_pipe == 9 )
                        {
                                bad_cells_at_pipe[14][9] ++;
                        }
                        if( bad_cells_per_pipe == 10 )
                        {
                                bad_cells_at_pipe[14][10] ++;
                        }
                        if( bad_cells_per_pipe == 11 )
                        {
                                bad_cells_at_pipe[14][11] ++;
                        }
                        if( bad_cells_per_pipe == 12 )
                        {
                                bad_cells_at_pipe[14][12] ++;
                        }
                        if( bad_cells_per_pipe ==
```

```
 13 )
                                                {
                                                        bad_cells_at_pipe
[14][13] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 14 )
                                                {
                                                        bad_cells_at_pipe
[14][14] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 15 )
                                                {
                                                        bad_cells_at_pipe
[14][15] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 16 )
                                                {
                                                        bad_cells_at_pipe
[14][16] ++;
                                                }
                                        }

                        if( loopcntr == 15 )
                                        {
                                if( bad_cells_per_pipe == 0 )
                                                {
                                                        bad_cells_at_pipe
[15][0] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 1 )
                                                {
                                                        bad_cells_at_pipe
[15][1] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 2 )
                                                {
                                                        bad_cells_at_pipe
[15][2] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 3 )
                                                {
                                                        bad_cells_at_pipe
[15][3] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 4 )
                                                {
                                                        bad_cells_at_pipe
[15][4] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 5 )
                                                {
                                                        bad_cells_at_pipe
[15][5] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 6 )
```



```
                                                {
                                                        bad_cells_at_pipe
[15][6] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 7 )
                                                {
                                                        bad_cells_at_pipe
[15][7] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 8 )
                                                {
                                                        bad_cells_at_pipe
[15][8] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 9 )
                                                {
                                                        bad_cells_at_pipe
[15][9] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 10 )
                                                {
                                                        bad_cells_at_pipe
[15][10] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 11 )
                                                {
                                                        bad_cells_at_pipe
[15][11] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 12 )
                                                {
                                                        bad_cells_at_pipe
[15][12] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 13 )
                                                {
                                                        bad_cells_at_pipe
[15][13] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 14 )
                                                {
                                                        bad_cells_at_pipe
[15][14] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 15 )
                                                {
                                                        bad_cells_at_pipe
[15][15] ++;
                                                }
                                                if( bad_cells_per_pipe ==
 16 )
                                                {
                                                        bad_cells_at_pipe
[15][16] ++;
                                                }
                                        }
```

```
                    bad_cells_per_pipe = 0;
                                )
                        )


/*****************************************************************/
/*                                                  AMU ADDRESSES
                    */
/*****************************************************************/

                    /* # of bad cells in amuadd 0->15 for 1.5v data *
/
                    if(rowcounter == 2 )
                    {
                            for(loopcntr = 0; loopcntr < 16; loopcntr
 ++ )
                            {
                                    bad_cells_in_1vrow[loopcntr] += p
arameter[loopcntr];
                                    bad_cells_per_amuadd += parameter
[loopcntr];

                                    /*
                                    This section while count the # of
 failures due to
                                    0->16 bad cells in each pipe.
                                    */

                                    if( bad_cells_per_amuadd == 0 )
                                    {
                                            row_failures[0] ++;
                                    }
                                    if( bad_cells_per_amuadd == 1 )
                                    {
                                            row_failures[1] ++;
                                    }
                                    if( bad_cells_per_amuadd == 2 )
                                    {
                                            row_failures[2] ++;
                                    }
                                    if( bad_cells_per_amuadd == 3 )
                                    {
                                            row_failures[3] ++;
                                    }
                                    if( bad_cells_per_amuadd == 4 )
                                    {
                                            row_failures[4] ++;
                                    }
                                    if( bad_cells_per_amuadd == 5 )
                                    {
                                            row_failures[5] ++;
                                    }
                                    if( bad_cells_per_amuadd == 6 )
                                    {
                                            row_failures[6] ++;
                                    }
                                    if( bad_cells_per_amuadd == 7 )
                                    {
                                            row_failures[7] ++;
                                    }
                                    if( bad_cells_per_amuadd == 8 )
                                    {
                                            row_failures[8] ++;
```



```
                                    }
                                    if( bad_cells_per_amuadd == 9 )
                                    {
                                            row_failures[9] ++;
                                    }
                                    if( bad_cells_per_amuadd == 10 )
                                    {
                                            row_failures[10] ++;
                                    }
                                    if( bad_cells_per_amuadd == 11 )
                                    {
                                            row_failures[11] ++;
                                    }
                                    if( bad_cells_per_amuadd == 12 )
                                    {
                                            row_failures[12] ++;
                                    }
                                    if( bad_cells_per_amuadd == 13 )
                                    {
                                            row_failures[13] ++;
                                    }
                                    if( bad_cells_per_amuadd == 14 )
                                    {
                                            row_failures[14] ++;
                                    }
                                    if( bad_cells_per_amuadd == 15 )
                                    {
                                            row_failures[15] ++;
                                    }
                                    if( bad_cells_per_amuadd == 16 )
                                    {
                                            row_failures[16] ++;
                                    }

                                    /*
                                            Counts the Number of bad
amu address (i.e. 16 bad
                                            cells )
                                    */
                                    if( bad_cells_per_amuadd == 16 )
                                    {
                                            bad_amu_add ++;
                                    }

                                    if( loopcntr == 0 )
                                    {
                            if( bad_cells_per_amuadd == 0 )
                                                    {
                                                            bad_cells_at_amua
dd[0][0] ++;
                                                    }
                                                    if( bad_cells_per_amuadd
== 1 )
                                                    {
                                                            bad_cells_at_amua
dd[0][1] ++;
                                                    }
                                                    if( bad_cells_per_amuadd
== 2 )
                                                    {
                                                            bad_cells_at_amua
dd[0][2] ++;
                                                    }
                                                    if( bad_cells_per_amuadd
== 3 )
```

```
dd[0][3] ++;
== 4 )
dd[0][4] ++;
== 5 )
dd[0][5] ++;
== 6 )
dd[0][6] ++;
== 7 )
dd[0][7] ++;
== 8 )
dd[0][8] ++;
== 9 )
dd[0][9] ++;
== 10 )
dd[0][10] ++;
== 11 )
dd[0][11] ++;
== 12 )
dd[0][12] ++;
== 13 )
dd[0][13] ++;
== 14 )
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
```



```
                {
                        bad_cells_at_amua
                }
dd[0][14] ++;
== 15 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[0][15] ++;
== 16 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[0][16] ++;
                }
                if( loopcntr == 1 )
                {
        if( bad_cells_per_amuadd == 0 )
                {
                        bad_cells_at_amua
                }
dd[1][0] ++;
== 1 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[1][1] ++;
== 2 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[1][2] ++;
== 3 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[1][3] ++;
== 4 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[1][4] ++;
== 5 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[1][5] ++;
== 6 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
dd[1][6] ++;
== 7 )
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
```

```
dd[1][7] ++;
== 8 )
dd[1][8] ++;
== 9 )
dd[1][9] ++;
== 10 )
dd[1][10] ++;
== 11 )
dd[1][11] ++;
== 12 )
dd[1][12] ++;
== 13 )
dd[1][13] ++;
== 14 )
dd[1][14] ++;
== 15 )
dd[1][15] ++;
== 16 )
dd[1][16] ++;
dd[2][0] ++;
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                }
                                if( loopcntr == 2 )
                                {
                        if( bad_cells_per_amuadd == 0 )
                                        {
                                                bad_cells_at_amua
                                        }
```



```
== 1 )
dd[2][1] ++;
== 2 )
dd[2][2] ++;
== 3 )
dd[2][3] ++;
== 4 )
dd[2][4] ++;
== 5 )
dd[2][5] ++;
== 6 )
dd[2][6] ++;
== 7 )
dd[2][7] ++;
== 8 )
dd[2][8] ++;
== 9 )
dd[2][9] ++;
== 10 )
dd[2][10] ++;
== 11 )
dd[2][11] ++;
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
```

```
                                                    if( bad_cells_per_amuadd
== 12 )                                             {
dd[2][12] ++;                                               bad_cells_at_amua
                                                    }
== 13 )                                             if( bad_cells_per_amuadd
dd[2][13] ++;                                       {
                                                            bad_cells_at_amua
== 14 )                                             }
                                                    if( bad_cells_per_amuadd
dd[2][14] ++;                                       {
                                                            bad_cells_at_amua
== 15 )                                             }
                                                    if( bad_cells_per_amuadd
dd[2][15] ++;                                       {
                                                            bad_cells_at_amua
== 16 )                                             }
                                                    if( bad_cells_per_amuadd
dd[2][16] ++;                                       {
                                                            bad_cells_at_amua
                                                    }
                                            }
                                    if( loopcntr == 3 )
                                            {
                                        if( bad_cells_per_amuadd == 0 )
                                                    {
                                                            bad_cells_at_amua
dd[3][0] ++;                                        }
                                                    if( bad_cells_per_amuadd
== 1 )                                              {
dd[3][1] ++;                                                bad_cells_at_amua
                                                    }
== 2 )                                              if( bad_cells_per_amuadd
dd[3][2] ++;                                        {
                                                            bad_cells_at_amua
== 3 )                                              }
                                                    if( bad_cells_per_amuadd
dd[3][3] ++;                                        {
                                                            bad_cells_at_amua
== 4 )                                              }
                                                    if( bad_cells_per_amuadd
dd[3][4] ++;                                        {
                                                            bad_cells_at_amua
== 5 )                                              }
                                                    if( bad_cells_per_amuadd
```



```
                                                    {
dd[3][5] ++;                                                bad_cells_at_amua
                                                    }
== 6 )                                              if( bad_cells_per_amuadd
dd[3][6] ++;                                        {
                                                            bad_cells_at_amua
== 7 )                                              }
                                                    if( bad_cells_per_amuadd
dd[3][7] ++;                                        {
                                                            bad_cells_at_amua
== 8 )                                              }
                                                    if( bad_cells_per_amuadd
dd[3][8] ++;                                        {
                                                            bad_cells_at_amua
== 9 )                                              }
                                                    if( bad_cells_per_amuadd
dd[3][9] ++;                                        {
                                                            bad_cells_at_amua
== 10 )                                             }
                                                    if( bad_cells_per_amuadd
dd[3][10] ++;                                       {
                                                            bad_cells_at_amua
== 11 )                                             }
                                                    if( bad_cells_per_amuadd
dd[3][11] ++;                                       {
                                                            bad_cells_at_amua
== 12 )                                             }
                                                    if( bad_cells_per_amuadd
dd[3][12] ++;                                       {
                                                            bad_cells_at_amua
== 13 )                                             }
                                                    if( bad_cells_per_amuadd
dd[3][13] ++;                                       {
                                                            bad_cells_at_amua
== 14 )                                             }
                                                    if( bad_cells_per_amuadd
dd[3][14] ++;                                       {
                                                            bad_cells_at_amua
== 15 )                                             }
                                                    if( bad_cells_per_amuadd
dd[3][15] ++;                                       {
                                                            bad_cells_at_amua
== 16 )                                             }
                                                    if( bad_cells_per_amuadd
```

```
                                                {
                                                        bad_cells_at_amux
dd[3][16] ++;                                   }
                                        }
                                if( loopcntr == 4 )
                                        {
                                        if( bad_cells_per_amuxdd == 0 )
                                                {
                                                        bad_cells_at_amux
dd[4][0] ++;                                    }
                                                if( bad_cells_per_amuxdd
                                                {
                                                        bad_cells_at_amux
== 1 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][1] ++;                                    {
                                                        bad_cells_at_amux
== 2 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][2] ++;                                    {
                                                        bad_cells_at_amux
== 3 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][3] ++;                                    {
                                                        bad_cells_at_amux
== 4 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][4] ++;                                    {
                                                        bad_cells_at_amux
== 5 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][5] ++;                                    {
                                                        bad_cells_at_amux
== 6 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][6] ++;                                    {
                                                        bad_cells_at_amux
== 7 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][7] ++;                                    {
                                                        bad_cells_at_amux
== 8 )                                          }
                                                if( bad_cells_per_amuxdd
dd[4][8] ++;                                    {
                                                        bad_cells_at_amux
== 9 )                                          }
```



```
dd[4][9] ++;                                    }
                                                if( bad_cells_per_amuxdd
== 10 )                                         {
                                                        bad_cells_at_amux
dd[4][10] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 11 )                                         {
                                                        bad_cells_at_amux
dd[4][11] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 12 )                                         {
                                                        bad_cells_at_amux
dd[4][12] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 13 )                                         {
                                                        bad_cells_at_amux
dd[4][13] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 14 )                                         {
                                                        bad_cells_at_amux
dd[4][14] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 15 )                                         {
                                                        bad_cells_at_amux
dd[4][15] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 16 )                                         {
                                                        bad_cells_at_amux
dd[4][16] ++;                                   }
                                        }
                                if( loopcntr == 5 )
                                        {
                                        if( bad_cells_per_amuxdd == 0 )
                                                {
                                                        bad_cells_at_amux
dd[5][0] ++;                                    }
                                                if( bad_cells_per_amuxdd
                                                {
                                                        bad_cells_at_amux
== 1 )                                          }
                                                if( bad_cells_per_amuxdd
dd[5][1] ++;                                    {
                                                        bad_cells_at_amux
== 2 )                                          }
                                                if( bad_cells_per_amuxdd
dd[5][2] ++;                                    {
                                                        bad_cells_at_amux
```

```
== 3 )
dd[5][3] ++;
== 4 )
dd[5][4] ++;
== 5 )
dd[5][5] ++;
== 6 )
dd[5][6] ++;
== 7 )
dd[5][7] ++;
== 8 )
dd[5][8] ++;
== 9 )
dd[5][9] ++;
== 10 )
dd[5][10] ++;
== 11 )
dd[5][11] ++;
== 12 )
dd[5][12] ++;
== 13 )
dd[5][13] ++;
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
```



```
== 14 )
dd[5][14] ++;
== 15 )
dd[5][15] ++;
== 16 )
dd[5][16] ++;
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
                if( bad_cells_per_amuadd
                {
                        bad_cells_at_amua
                }
        if( loopcntr == 6 )
                {
                if( bad_cells_per_amuadd == 0 )
                        {
                        bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
                        {
                                bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
                        {
                                bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
                        {
                                bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
                        {
                                bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
                        {
                                bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
                        {
                                bad_cells_at_amua
                        }
                        if( bad_cells_per_amuadd
dd[6][0] ++;
== 1 )
dd[6][1] ++;
== 2 )
dd[6][2] ++;
== 3 )
dd[6][3] ++;
== 4 )
dd[6][4] ++;
== 5 )
dd[6][5] ++;
== 6 )
dd[6][6] ++;
```

```
== 7 )
dd[6][7] ++;
== 8 )
dd[6][8] ++;
== 9 )
dd[6][9] ++;
== 10 )
dd[6][10] ++;
== 11 )
dd[6][11] ++;
== 12 )
dd[6][12] ++;
== 13 )
dd[6][13] ++;
== 14 )
dd[6][14] ++;
== 15 )
dd[6][15] ++;
== 16 )
dd[6][16] ++;
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                }
                        if( loopcntr == 7 )
                        {
                                if( bad_cells_per_onuadd == 0 )
                                {
```



```
dd[7][0] ++;
== 1 )
dd[7][1] ++;
== 2 )
dd[7][2] ++;
== 3 )
dd[7][3] ++;
== 4 )
dd[7][4] ++;
== 5 )
dd[7][5] ++;
== 6 )
dd[7][6] ++;
== 7 )
dd[7][7] ++;
== 8 )
dd[7][8] ++;
== 9 )
dd[7][9] ++;
== 10 )
dd[7][10] ++;
== 11 )
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
                                                bad_cells_at_once
                                        }
                                        if( bad_cells_per_onuadd
                                        {
```

```
dd[7][11] ++;
== 12 )
dd[7][12] ++;
== 13 )
dd[7][13] ++;
== 14 )
dd[7][14] ++;
== 15 )
dd[7][15] ++;
== 16 )
dd[7][16] ++;

                if( loopcntr == 8 )
                        {
                    if( bad_cells_per_amuadd == 0 )
dd[8][0] ++;
== 1 )
dd[8][1] ++;
== 2 )
dd[8][2] ++;
== 3 )
dd[8][3] ++;
== 4 )
```

```
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                        }
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
```



```
dd[8][4] ++;
== 5 )
dd[8][5] ++;
== 6 )
dd[8][6] ++;
== 7 )
dd[8][7] ++;
== 8 )
dd[8][8] ++;
== 9 )
dd[8][9] ++;
== 10 )
dd[8][10] ++;
== 11 )
dd[8][11] ++;
== 12 )
dd[8][12] ++;
== 13 )
dd[8][13] ++;
== 14 )
dd[8][14] ++;
== 15 )
```

```
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
                            }
                            if( bad_cells_per_amuadd
                            {
                                bad_cells_at_amua
```

```
dd[8][15] ++;

== 16 )

dd[8][16] ++;

if( loopcntr == 9 )
{
    if( bad_cells_per_amuadd == 0 )
    dd[9][0] ++;

    == 1 )

    dd[9][1] ++;

    == 2 )

    dd[9][2] ++;

    == 3 )

    dd[9][3] ++;

    == 4 )

    dd[9][4] ++;

    == 5 )

    dd[9][5] ++;

    == 6 )

    dd[9][6] ++;

    == 7 )

    dd[9][7] ++;

    == 8 )

    dd[9][8] ++;
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
}
```



```
== 9 )

dd[9][9] ++;

== 10 )

dd[9][10] ++;

== 11 )

dd[9][11] ++;

== 12 )

dd[9][12] ++;

== 13 )

dd[9][13] ++;

== 14 )

dd[9][14] ++;

== 15 )

dd[9][15] ++;

== 16 )

dd[9][16] ++;

if( loopcntr == 10 )
{
    if( bad_cells_per_amuadd == 0 )
    dd[10][0] ++;

    == 1 )

    dd[10][1] ++;
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
    if( bad_cells_per_amuadd
    {
        bad_cells_at_amua
    }
}
```

```
== 2 )
dd[10][2] ++;
== 3 )
dd[10][3] ++;
== 4 )
dd[10][4] ++;
== 5 )
dd[10][5] ++;
== 6 )
dd[10][6] ++;
== 7 )
dd[10][7] ++;
== 8 )
dd[10][8] ++;
== 9 )
dd[10][9] ++;
== 10 )
dd[10][10] ++;
== 11 )
dd[10][11] ++;
== 12 )
dd[10][12] ++;
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
```



```
== 13 )
dd[10][13] ++;
== 14 )
dd[10][14] ++;
== 15 )
dd[10][15] ++;
== 16 )
dd[10][16] ++;
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
)
if( loopcntr == 11 )
        {
    if( bad_cells_per_amuadd == 0 )
dd[11][0] ++;
== 1 )
dd[11][1] ++;
== 2 )
dd[11][2] ++;
== 3 )
dd[11][3] ++;
== 4 )
dd[11][4] ++;
== 5 )
dd[11][5] ++;
        {
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
if( bad_cells_per_amuadd
{
        bad_cells_at_amua
}
```

```
== 6 )
dd[11][6] ++;
== 7 )
dd[11][7] ++;
== 8 )
dd[11][8] ++;
== 9 )
dd[11][9] ++;
== 10 )
dd[11][10] ++;
== 11 )
dd[11][11] ++;
== 12 )
dd[11][12] ++;
== 13 )
dd[11][13] ++;
== 14 )
dd[11][14] ++;
== 15 )
dd[11][15] ++;
== 16 )
dd[11][16] ++;
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
                                        if( bad_cells_per_amuadd
                                        {
                                                bad_cells_at_amua
                                        }
```



```
                                        }
                                }
                        if( loopcntr == 12 )
                                        {
                                if( bad_cells_per_amuadd == 0 )
                                                {
                                                        bad_cells_at_amua
                                                }
dd[12][0] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 1 )
dd[12][1] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 2 )
dd[12][2] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 3 )
dd[12][3] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 4 )
dd[12][4] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 5 )
dd[12][5] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 6 )
dd[12][6] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 7 )
dd[12][7] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 8 )
dd[12][8] ++;
                                                if( bad_cells_per_amuadd
                                                {
                                                        bad_cells_at_amua
                                                }
== 9 )
dd[12][9] ++;
                                                )
```

```
                                                        if( bad_cells_per_amuadd
== 10 )                                                 {
dd[12][10] ++;                                                  bad_cells_at_amua
                                                        }
== 11 )                                                 if( bad_cells_per_amuadd
                                                        {
dd[12][11] ++;                                                  bad_cells_at_amua
                                                        }
== 12 )                                                 if( bad_cells_per_amuadd
                                                        {
dd[12][12] ++;                                                  bad_cells_at_amua
                                                        }
== 13 )                                                 if( bad_cells_per_amuadd
                                                        {
dd[12][13] ++;                                                  bad_cells_at_amua
                                                        }
== 14 )                                                 if( bad_cells_per_amuadd
                                                        {
dd[12][14] ++;                                                  bad_cells_at_amua
                                                        }
== 15 )                                                 if( bad_cells_per_amuadd
                                                        {
dd[12][15] ++;                                                  bad_cells_at_amua
                                                        }
== 16 )                                                 if( bad_cells_per_amuadd
                                                        {
dd[12][16] ++;                                                  bad_cells_at_amua
                                                        }
                                                }

                                if( loopcntr == 13 )
                                                {
                        if( bad_cells_per_amuadd == 0 )
                                                        {
                                                                bad_cells_at_amua
dd[13][0] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 1 )                                                  {
                                                                bad_cells_at_amua
dd[13][1] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 2 )                                                  {
                                                                bad_cells_at_amua
dd[13][2] ++;                                           }
                                                        if( bad_cells_per_amuadd
```



```
== 3 )                                                  {
                                                                bad_cells_at_amua
dd[13][3] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 4 )                                                  {
                                                                bad_cells_at_amua
dd[13][4] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 5 )                                                  {
                                                                bad_cells_at_amua
dd[13][5] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 6 )                                                  {
                                                                bad_cells_at_amua
dd[13][6] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 7 )                                                  {
                                                                bad_cells_at_amua
dd[13][7] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 8 )                                                  {
                                                                bad_cells_at_amua
dd[13][8] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 9 )                                                  {
                                                                bad_cells_at_amua
dd[13][9] ++;                                           }
                                                        if( bad_cells_per_amuadd
== 10 )                                                 {
                                                                bad_cells_at_amua
dd[13][10] ++;                                          }
                                                        if( bad_cells_per_amuadd
== 11 )                                                 {
                                                                bad_cells_at_amua
dd[13][11] ++;                                          }
                                                        if( bad_cells_per_amuadd
== 12 )                                                 {
                                                                bad_cells_at_amua
dd[13][12] ++;                                          }
                                                        if( bad_cells_per_amuadd
== 13 )                                                 {
                                                                bad_cells_at_amua
dd[13][13] ++;                                          }
                                                        if( bad_cells_per_amuadd
```

```
== 14 )
                                                {
                                                        bad_cells_at_amux
                                                }
dd[13][14] ++;                                  if( bad_cells_per_amuxdd
                                                {
== 15 )                                                 bad_cells_at_amux
                                                }
dd[13][15] ++;                                  if( bad_cells_per_amuxdd
                                                {
== 16 )                                                 bad_cells_at_amux
                                                }
dd[13][16] ++;                                  )

                        if( loopcntr == 14 )
                                                {
                                if( bad_cells_per_amuxdd == 0 )
                                                {
                                                        bad_cells_at_amux
dd[14][0] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 1 )                                          {
                                                        bad_cells_at_amux
dd[14][1] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 2 )                                          {
                                                        bad_cells_at_amux
dd[14][2] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 3 )                                          {
                                                        bad_cells_at_amux
dd[14][3] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 4 )                                          {
                                                        bad_cells_at_amux
dd[14][4] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 5 )                                          {
                                                        bad_cells_at_amux
dd[14][5] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 6 )                                          {
                                                        bad_cells_at_amux
dd[14][6] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 7 )
```



```
                                                {
                                                        bad_cells_at_amux
dd[14][7] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 8 )                                          {
                                                        bad_cells_at_amux
dd[14][8] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 9 )                                          {
                                                        bad_cells_at_amux
dd[14][9] ++;                                   }
                                                if( bad_cells_per_amuxdd
== 10 )                                         {
                                                        bad_cells_at_amux
dd[14][10] ++;                                  }
                                                if( bad_cells_per_amuxdd
== 11 )                                         {
                                                        bad_cells_at_amux
dd[14][11] ++;                                  }
                                                if( bad_cells_per_amuxdd
== 12 )                                         {
                                                        bad_cells_at_amux
dd[14][12] ++;                                  }
                                                if( bad_cells_per_amuxdd
== 13 )                                         {
                                                        bad_cells_at_amux
dd[14][13] ++;                                  }
                                                if( bad_cells_per_amuxdd
== 14 )                                         {
                                                        bad_cells_at_amux
dd[14][14] ++;                                  }
                                                if( bad_cells_per_amuxdd
== 15 )                                         {
                                                        bad_cells_at_amux
dd[14][15] ++;                                  }
                                                if( bad_cells_per_amuxdd
== 16 )                                         {
                                                        bad_cells_at_amux
dd[14][16] ++;                                  }
                                        )

                        if( loopcntr == 15 )
                                                {
                                if( bad_cells_per_amuxdd == 0 )
                                                {
```

```
                                        bad_cells_at_amux
                                }
dd[15][0] ++;                           if( bad_cells_per_amuxdd
                                        {
== 1 )                                          bad_cells_at_amux
                                        }
dd[15][1] ++;                           if( bad_cells_per_amuxdd
                                        {
== 2 )                                          bad_cells_at_amux
                                        }
dd[15][2] ++;                           if( bad_cells_per_amuxdd
                                        {
== 3 )                                          bad_cells_at_amux
                                        }
dd[15][3] ++;                           if( bad_cells_per_amuxdd
                                        {
== 4 )                                          bad_cells_at_amux
                                        }
dd[15][4] ++;                           if( bad_cells_per_amuxdd
                                        {
== 5 )                                          bad_cells_at_amux
                                        }
dd[15][5] ++;                           if( bad_cells_per_amuxdd
                                        {
== 6 )                                          bad_cells_at_amux
                                        }
dd[15][6] ++;                           if( bad_cells_per_amuxdd
                                        {
== 7 )                                          bad_cells_at_amux
                                        }
dd[15][7] ++;                           if( bad_cells_per_amuxdd
                                        {
== 8 )                                          bad_cells_at_amux
                                        }
dd[15][8] ++;                           if( bad_cells_per_amuxdd
                                        {
== 9 )                                          bad_cells_at_amux
                                        }
dd[15][9] ++;                           if( bad_cells_per_amuxdd
                                        {
== 10 )                                         bad_cells_at_amux
                                        }
dd[15][10] ++;                          if( bad_cells_per_amuxdd
                                        {
== 11 )
```



```
                                        bad_cells_at_amux
                                }
dd[15][11] ++;                          if( bad_cells_per_amuxdd
                                        {
== 12 )                                         bad_cells_at_amux
                                        }
dd[15][12] ++;                          if( bad_cells_per_amuxdd
                                        {
== 13 )                                         bad_cells_at_amux
                                        }
dd[15][13] ++;                          if( bad_cells_per_amuxdd
                                        {
== 14 )                                         bad_cells_at_amux
                                        }
dd[15][14] ++;                          if( bad_cells_per_amuxdd
                                        {
== 15 )                                         bad_cells_at_amux
                                        }
dd[15][15] ++;                          if( bad_cells_per_amuxdd
                                        {
== 16 )                                         bad_cells_at_amux
                                        }
dd[15][16] ++;
                                }
                bad_cells_per_amuxdd = 0;
                        }
                if(rowcounter == 3 )
                {
                        for(loopcntr = 0; loopcntr < 16; loopcntr
 ++ )
                        {
                                bad_cells_in_3vpipe[loopcntr] +=
parameter[loopcntr];
                        }
                if(rowcounter == 4 )
                {
                        for(loopcntr = 0; loopcntr < 16; loopcntr
 ++ )
                        {
                                bad_cells_in_3vrow[loopcntr] += p
arameter[loopcntr];
                        }
                }
        }
        rowcounter ++;
        if(rowcounter > 4)
        {
                                if( bad_amux_add == 0 )
                                {
```

```
                        chips_with_badamuadd[0] +
*;
                }
                if( bad_amu_add == 1 )
                {
                        chips_with_badamuadd[1] +
*;
                }
                if( bad_amu_add == 2 )
                {
                        chips_with_badamuadd[2] +
*;
                }
                if( bad_amu_add == 3 )
                {
                        chips_with_badamuadd[3] +
*;
                }
                if( bad_amu_add == 4 )
                {
                        chips_with_badamuadd[4] +
*;
                }
                if( bad_amu_add == 5 )
                {
                        chips_with_badamuadd[5] +
*;
                }
                if( bad_amu_add == 6 )
                {
                        chips_with_badamuadd[6] +
*;
                }
                if( bad_amu_add == 7 )
                {
                        chips_with_badamuadd[7] +
*;
                }
                if( bad_amu_add == 8 )
                {
                        chips_with_badamuadd[8] +
*;
                }
                if( bad_amu_add == 9 )
                {
                        chips_with_badamuadd[9] +
*;
                }
                if( bad_amu_add == 10 )
                {
                        chips_with_badamuadd[10]
**;
                }
                if( bad_amu_add == 11 )
                {
                        chips_with_badamuadd[11]
**;
                }
                if( bad_amu_add == 12 )
                {
                        chips_with_badamuadd[12]
**;
                }
                if( bad_amu_add == 13 )
                {
                        chips_with_badamuadd[13]
```



```
**;
                }
                if( bad_amu_add == 14 )
                {
                        chips_with_badamuadd[14]
**;
                }
                if( bad_amu_add == 15 )
                {
                        chips_with_badamuadd[15]
**;
                }
                if( bad_amu_add == 16 )
                {
                        chips_with_badamuadd[16]
**;
                }

                        bad_amu_add = 0;
                        rowcounter = 0;
                        filecounter **;
                }
        }
    }
    fclose(fp_amu);

    if( ( fp_amu = fopen( "c:\\womac\\results\\afinal2.res", "w" )) != NULL )

    {
        for(loopcntr = 0; loopcntr < 16; loopcntr **)
        {
            fprintf(fp_amu,"\n[AMU 1.5v Section] Total number of cell
s bad in row %2.1d: %d",                        loopcntr, bad_cells_in_1vrow[loop
cntr] );
        }
        fprintf(fp_amu,"\n\n");
        for(loopcntr = 0; loopcntr < 16; loopcntr **)
        {
            fprintf(fp_amu,"\n[AMU 1.5v Section] Total number of cell
s bad in pipe %2.1d: %d",                       loopcntr, bad_cells_in_1vpipe[loo
pcntr] );
        }
        fprintf(fp_amu,"\n\n");
        for(loopcntr = 0; loopcntr < 16; loopcntr **)
        {
            fprintf(fp_amu,"\n[AMU 3.5v Section] Total number of cell
s bad in row %2.1d: %d",                        loopcntr, bad_cells_in_3vrow[loop
cntr] );
        }
        fprintf(fp_amu,"\n\n");
        for(loopcntr = 0; loopcntr < 16; loopcntr **)
        {
            fprintf(fp_amu,"\n[AMU 3.5v Section] Total number of cell
s bad in pipe %2.1d: %d",                       loopcntr, bad_cells_in_3vpipe[loo
pcntr] );
        }

        fprintf(fp_amu,"\n\n");
        for(loopcntr = 0; loopcntr < 17; loopcntr **)
        {
```

```
                        fprintf(fp_amu,"\n[AMU 1.5v Section] Total number AMU Pip
es with %2.1d bad cells: %d",
                                                loopcntr, pipe_failures[loopcntr]
 );
                )



                fprintf(fp_amu,"\n\n");
                for( x = 0; x < 16; x ++ )
                {
                  for(y = 0; y < 17; y ++ )
                  {
                        fprintf(fp_amu,
                        "\nAt AMU Pipe %2.1d, # of times %2.1d bad cells occurred
: %d",
                        x, y, bad_cells_at_pipe[x][y]);
                  }
                  fprintf(fp_amu,"\n\n");
                }


                fprintf(fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                {
                        fprintf(fp_amu,"\n[AMU 1.5v Section] Total number AMU Add
resses with %2.1d bad cells: %d",
                                                loopcntr, row_failures[loopcntr]
);
                }

                fprintf(fp_amu,"\n\n");
                for( x = 0; x < 16; x ++ )
                {
                  for(y = 0; y < 17; y ++ )
                  {
                        fprintf(fp_amu,
                        "\nAt AMU Add %2.1d, # of times %2.1d bad cells occurred:
%d",
                        x, y, bad_cells_at_amuadd[x][y]);
                  }
                  fprintf(fp_amu,"\n\n");
                }

                fprintf(fp_amu,"\n\n");
                for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                {
                        fprintf(fp_amu,"\nChips with %2.1d bad amu address: %d",
                                                loopcntr, chips_with_badamuadd[lo
opcntr] );
                }


                fprintf( fp_amu,"\n\nFile count: %d\n", filecounter);

        }
        fclose(fp_amu);


        if( ( fp_amu = fopen( "c:\\womac\\results\\a2.res", "w" )) != NULL )
        {
                fprintf(fp_amu,"\n\n");
                fprintf(fp_amu,"\nTotal Number of bad Cells in Pipe 0->15\n\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
```



```
                        fprintf(fp_amu,"%d, ", bad_cells_in_1vrow[loopcntr] );
                }

                fprintf(fp_amu,"\n\n");
                fprintf(fp_amu,"\nTotal Number of bad Cells in AMU Add 0->15\n\n"
);
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf(fp_amu,"%d, ", bad_cells_in_1vpipe[loopcntr] );
                }

/*              3.5v Data
                fprintf(fp_amu,"\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf(fp_amu,"%d, ", bad_cells_in_3vrow[loopcntr] );
                }
                fprintf(fp_amu,"\n");
                for(loopcntr = 0; loopcntr < 16; loopcntr ++)
                {
                        fprintf(fp_amu,"%d, ", bad_cells_in_3vpipe[loopcntr] );
                }
*/

                fprintf(fp_amu,"\n\n");
                fprintf(fp_amu,"\nTotal number AMU Pipes with 0->16 bad cells\n\n
");
                for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                {
                        fprintf(fp_amu,"%d, ", pipe_failures[loopcntr] );
                }


                fprintf(fp_amu,"\n\n");
                fprintf(fp_amu,"\nAt AMU Pipe 0->15(row), # of times 0->16 (col)
bad cells occurred\n\n");
                for( x = 0; x < 16; x ++ )
                {
                  /*fprintf(fp_amu, "%2.1d, ", x );*/
                  for(y = 0; y < 17; y ++ )
                  {
                        fprintf(fp_amu,"%d, ", bad_cells_at_pipe[x][y]);
                  }
                  fprintf(fp_amu,"\n");
                }



                fprintf(fp_amu,"\n\n");
                fprintf(fp_amu,"\nTotal number AMU Add with 0->16 bad cells\n\n")
;
                for(loopcntr = 0; loopcntr < 17; loopcntr ++)
                {
                        fprintf(fp_amu,"%d, ", row_failures[loopcntr] );
                }




                fprintf(fp_amu,"\n\n");
                fprintf(fp_amu,"\nAt AMU Add 0->15(row), # of times 0->16 (col) b
ad cells occurred\n\n");
                for( x = 0; x < 16; x ++ )
                {
                  /*fprintf(fp_amu, "%2.1d, ", x );*/
```

```
                for(y = 0; y < 17; y ++ )
                {
                        fprintf(fp_amu,"%d, ", bad_cells_at_amuadd[x][y]);
                }
                fprintf(fp_amu,"\n");
              }

             fprintf(fp_amu,"\n\n");
             fprintf(fp_amu,"\nChips with 0->16 bad amu addresses\n\n");
             for(loopcntr = 0; loopcntr < 17; loopcntr ++)
             {
                    fprintf(fp_amu,"%d, ", chips_with_badamuadd[loopcntr] );
             }


        }
        fclose(fp_amu);

        printf("\nFiles generated by this program:");
        printf("\nc:\\womac\\results\\afinal2.res");
        printf("\nc:\\omac\\results\\a2.res");

}



/*************************************************************************/

/*
        Function Name:  Parse_data ( & Parse_J2_data )
        Requires:       pointer to a predefined character array (buffer)
                                                pointer to ?????
                                                pointer to a pointer to a charac
ter array with desired
                                                 size of parsed word
        Returns: void
        Purpose: seperates a buffer full (this case one line) of data into a
                                'word' array that is predefined in the calling pr
g.  The
                                determination of the buffer length is the charact
ers that
                                are checked for in the first while statement.  Th
e maximum
                                buffer size is actually determined in the calling
 prg.
*/

/*************************************************************************/



void Parse_data( char *buf, int *Count, char **Word)
{
        *Word = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
        while ( (*buf != '\n') && (*buf != '\0') )
        {
```



```
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (i.e. \0) character in the contents
 of the current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        Word[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;
                }
        }
        *buf = '\0';
}
```

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = TEST2
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\WOMAC\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = TEST2.C
FIRSTCPP =
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AL /Od /D "_DEBUG" /D "_DOS" /FR /Fd"TEST2.P
DB"
CFLAGS_R_DEXE = /nologo /Gs /G2 /W3 /AM /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /CO /STACK:5120
LFLAGS_R_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /STACK:5120
LIBS_D_DEXE = oldnames llibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = TEST2.SBR \
               PARSE.SBR \
               AMU_TST2.SBR


TEST2_DEP = c:\womac\variable.h


PARSE_DEP = c:\womac\variable.h


AMU_TST2_DEP = c:\womac\variable.h


all:    $(PROJ).EXE $(PROJ).BSC
```



```
TEST2.OBJ:      TEST2.C $(TEST2_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c TEST2.C

PARSE.OBJ:      PARSE.C $(PARSE_DEP)
        $(CC) $(CFLAGS) $(CUSEPCHFLAG) /c PARSE.C

AMU_TST2.OBJ:   AMU_TST2.C $(AMU_TST2_DEP)
        $(CC) $(CFLAGS) $(CUSEPCHFLAG) /c AMU_TST2.C

$(PROJ).EXE::   TEST2.OBJ PARSE.OBJ AMU_TST2.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
TEST2.OBJ +
PARSE.OBJ +
AMU_TST2.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
/*
        test2.C
        WRITTEN BY:     MICHAEL WOMAC
        DATE: 10 Jan 1995
        Version: 1.0
*/

/*************************************************************************/
/********                    Headers                          ********/
/*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include "c:\womac\variable.h"


/*************************************************************************/
/********                    Definitions                      ********/
/*************************************************************************/

#define MaxWords 1          /* defines the max number of words in array */
#define MaxBuffer 7         /* defines the max size of words in array */
#define matrix_size 512     /* defines char matrix size of test_data matrix */

/*************************************************************************/
/********                    Global Variables                 ********/
/*************************************************************************/

char *J2 column[256];
int J2_data[48][17];

/*************************************************************************/
/********                    Structures                       ********/
/*************************************************************************/



/*************************************************************************/
/********                    Functions                        ********/
/*************************************************************************/

void convert_J2_to_integer( int );
extern void amu_tst2( int *, amu2 ** );
extern void Parse_data( char *, int *, char ** );
extern void Parse_J2( char *, int *, char ** );

/*************************************************************************/
/********                    Code                             ********/
/*************************************************************************/



/*************************************************************************/

/*
        Function Name:  main
        Requires:   filename.ext
        Returns:    void
        Purpose:    open & close files that are listed in file: badchip.num.
                    The filenames are then parsed into individual words.
                    File is then opened and data is read out a line at a time

                    and converted to integer text and stored in a matrix call
ed
```



```
o
                    J2_data.  The address of the data matrix is then passed t
                    the testing funtions for analysis.

*/

/*************************************************************************/


void main( int argc, char **argv )
{

        /**** CONSTANTS ****/
        int Count, J2_Count;
        int chip_number;

        /**** COUNTERS ****/
        int row = 0;            /* counts number of rows read out of data file */
        int file_counter = 0; /* counts number of files that are opened */
        int x;

        /**** FILE POINTERS ****/
        FILE *fp_badchip;
        FILE *fp_bad_data;
        FILE *fp_amu2;

        /**** CHARACTER ARRAYS ****/
        char *File_num[ MaxWords ];
        char filenum[ MaxBuffer ];
        char test_data[ matrix_size ];
        char filename[25];
    char file2[50];

        /**** STRUCTURE POINTERS ****/
        amu2 *amu2_results_add;
        amu2 *amu2_tmp;

        amu2_results_add = NULL;



        /**** CODE ****/
        if( argc != 2 )
        {
                puts("\nUsage: Filename.ext");
                exit(1);
        }

         sprintf( file2, "c:\\womac\\badchip#\\%s",  argv[1]);

         printf("loaded file: %s", argv[1] );
        /*
                opens file that contains bad chip numbers
        */
        if(  (fp_badchip = fopen( file2, "r")) != NULL)
        {
                /*
                        get next bad file number
                */
                while ( (fgets(filenum, MaxBuffer, fp_badchip) != NULL) )
                {
                        /*
                                puts chip number in a File_num[0] so it can be op
ened
                        */
```

```
                        Parse_data( filenum, &Count, &(file_num[0]) );

                        chip_number = atoi(file_num[0]);

                        /*
                                stores the file name to a variable called filenam
e
                        */
                        sprintf ( filename, "c:\\chiptst1\\c%s.dat", file_num[0]
 ) ;

                        /*
                                opens bad file for testing
                        */
                        if ( (fp_bad_data = fopen( filename , "r")) != NULL )
                        {
                                /*
                                        gets a line from the bad data fil
e and stores in the
                                        variable test_data. pointer fp_ba
d_data increments each
                                        pass so to get the next line
                                */
                                while( (fgets( test_data, matrix_size , f
p_bad_data))
                                                != NULL )
                                {
                                        /*
                                                seperates test_data strin
g which is one line out of the
                                                bad data file into J2_col
umn words
                                        */
                                        Parse_J2( test_data, &J2_Count, &
(J2_column[0]) );
                                        convert_J2_to_integer( row );
                                        row ++;
                                }
                        printf("\nfile count: %5.1d", file_counter);

                        amu_tst2( &chip_number, &amu2_results_add );

                }
                file_counter ++;
                row = 0;
                fclose( fp_bad_data );
        }
        fclose(fp_badchip);
        }
        else
        {
                printf("\nCan't open file");
                exit(1);
        }

        /*
                saves structures to corresponding files
        */

        if ( ( fp_amu2 = fopen( "c:\\womac\\results\\results2.amu", "w" ))  != NU
LL )
        {
```



```
                for( amu2_tmp = amu2_results_add; amu2_tmp != NULL;
                                amu2_tmp = amu2_tmp->amu2_Previous_add )
                {
                        fprintf( fp_amu2,"%d\n", amu2_tmp->chip );
                        for(x = 0; x < 16; x ++)
                        {
                                fprintf( fp_amu2,"%d, ", amu2_tmp->num_bad_cells_
1amupipe[x] );
                        }
                        fprintf( fp_amu2,"\n");
                        for(x = 0; x < 16; x ++)
                        {
                                fprintf( fp_amu2,"%d, ", amu2_tmp->num_bad_cells_
1amuadd[x] );
                        }
                        fprintf( fp_amu2,"\n");
                        for(x = 0; x < 16; x ++)
                        {
                                fprintf( fp_amu2,"%d, ", amu2_tmp->num_bad_cells_
3amupipe[x] );
                        }
                        fprintf( fp_amu2,"\n");
                        for(x = 0; x < 16; x ++)
                        {
                                fprintf( fp_amu2,"%d, ", amu2_tmp->num_bad_cells_
3amuadd[x] );
                        }
                        fprintf( fp_amu2,"\n");
                }

        }
        fclose(fp_amu2);

        printf("\nfile_counter: %d\n", file_counter);
        printf("\nfiles generated by this program:");
        printf("\nc:\\womac\\results\\results2.tac");
        printf("\nc:\\womac\\results\\results2.amu");

}


/*************************************************************************/

/*
        function Name: convert_J2_to_integer
        Requires:  current row of type integer
        Returns:   void
        Purpose:   converts J2_column character array into J2_data
                                integer array.
*/

/*************************************************************************/

void convert_J2_to_integer ( int row )
{
        int column;
        if ( row < 16 )
        {
                for( column = 0; column < 9; column ++ )
                {
```

```
                J2_data[row][column] = atoi( J2_column[ column ] );
            }
        }
        if ( row > 15 )
        {
            for( column = 0; column < 16; column ++ )
            {
                J2_data[row][column] = atoi( J2_column[ column ] ) ;
            }
        }
}
```

```
/*
        parse.c  ---- A Utility program -------
        Written by: Lloyd Clonts
        Modified by: Mike Womac
        Modified on: 5 Jan 1995
        Version: 2.0 (Lloyds version is version 1.0 )
*/


/*************************************************************************/
/********                    Headers                            ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "c:\womac\variable.h"

/*************************************************************************/
/********                    Definitions                        ********/
/*************************************************************************/

#define size_of_column 16

/*************************************************************************/
/********                    Global Variables                   ********/
/*************************************************************************/

extern char *J2_column[ size_of_column];

/*************************************************************************/
/********                    Structures                         ********/
/*************************************************************************/


/*************************************************************************/
/********                    Functions                          ********/
/*************************************************************************/


void Parse_J2( char *, int *, char ** );

/*************************************************************************/
/********                    Code                               ********/
/*************************************************************************/



/************************************************************************/

/*
        Function Name:  Parse_data ( & Parse_J2_data )
        Requires:       pointer to a predefined character array (buffer)
                                        pointer to ?????
                                        pointer to a pointer to a charac
ter array with desired
                                         size of parsed word
        Returns: void
        Purpose: seperates a buffer full (this case one line) of data into a
                                'word' array that is predefined in the calling pr
g.  The
                                determination of the buffer length is the charact
ers that
                                are checked for in the first while statement.  Th
e maximum
                                buffer size is actually determined in the calling
 prg.
```



```
*/
/************************************************************************/




void Parse_data( char *buf, int *Count, char **Word)
{
        *Word = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (i.e. \0) character in the contents
 of the current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        Word[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;
                }
        }
        *buf = '\0';
}
void Parse_J2( char *buf, int *Count, char **J2_column)
{
        *J2_column = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
```

```
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (\0) character in the contents of t
he current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        j2_column[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;

                }
        }

        /*
                stores a NULL character in last used address in the buffer
        */
        *buf = '\0';
}
```

```
/*
        amu_tst2.C
        WRITTEN-BY:     MICHAEL WOMAC
        DATE: 10 Jan 1995
        Version: 1.0
*/

/*************************************************************************/
/********                    Headers                            ********/
/*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "c:\womac\variable.h"
/*#include <alloc.h>*/ /* c++ compiler */
#include <malloc.h>   /* quick c compiler */

/*************************************************************************/
/********                    Definitions                        ********/
/*************************************************************************/


/*************************************************************************/
/********                    Global Variables                   ********/
/*************************************************************************/

 extern int J2_data[48][17];

/*************************************************************************/
/********                    Structures                         ********/
/*************************************************************************/


/*************************************************************************/
/********                    Functions                          ********/
/*************************************************************************/

void amu_tst2( int *, amu2 ** );
void save_amu2_data( amu2 **, int, int *, int *, int *, int * );

/*************************************************************************/
/********                    Code                               ********/
/*************************************************************************/



/************************************************************************/

/*
        Function Name: amu_tst2
        Requires:       pointer to chip_num
                                        pointer to pointer to amu2 struct
ure
        Returns:        void
        Purpose:    Count errors of Pb chip data files.  These errors
                                        are then recorded by calling save_amu2_de
ta function
*/

/************************************************************************/

void amu_tst2( int *chip_num, amu2 **amu2_results_add )
{
        /**** COUNTERS ****/
        int row, column;
```



```
        /**** CONSTANTS ****/
        int amu2_limit[2];
        int amu2_tolerance;

        /**** ARRAYS ****/
        int amu1_pipe_cells[ 17 ];
        int amu1_add_cells[ 17 ];
        int amu3_pipe_cells[ 17 ];
        int amu3_add_cells[ 17 ];


        /**** Initializations ****/

        amu2_limit[1] = 716; /* 1.5V input */
        amu2_limit[0] = 307; /* 3.5V input */

        amu2_tolerance = 100;


        for( row = 0; row < 17; row ++)
        {
                amu1_pipe_cells[ row ] = 0;
                amu1_add_cells[ row ] = 0;
                amu3_pipe_cells[ row ] = 0;
                amu3_add_cells[ row ] = 0;
        }

        /*************************************************************/
        /*                    Testing AMU                            */
        /*************************************************************/


                        for( column = 0; column < 16; column ++)
                        {
                                for( row = 16; row < 32; row++ )
                                {
                                        if( (  J2_data[row][column] <
                                                        amu2_limit[1] -  amu2_tol
erance)
                                                ||( J2_data[row][column] >
                                                        amu2_limit[1] + amu2_tole
rance) )
                                        {
                                                amu1_pipe_cells[ column ] ++;
                                                amu1_add_cells[ row-16 ] ++;
                                        }

                                }

                        }
/* ******************** 3.5 v section ********************************** */
                        for( column = 0; column < 16; column ++ )
                        {
                                for( row = 32; row < 48; row ++ )
                                {
                                  if(((J2_data[row][column])<(amu2_limit[0] - amu
2_tolerance))
                                  || ((J2_data[row][column])>(amu2_limit[0] + amu
2_tolerance)))
                                        {
                                                amu3_pipe_cells[ column ] ++;
                                                amu3_add_cells[ row-32 ] ++;
```

```
                                )
                        )
                )

                        /**** testing for AMU ADC failures ****/


        save_amu2_data( amu2_results_add, *chip_num, &(amu1_pipe_cells[0]),
                                        &(amu1_add_cells[0]), &(amu3_pipe_cells
[0]),
                                        &(amu3_add_cells[0]) );


}

void save_amu2_data( amu2 **amu2_results_add, int chip_num,
                                                int *amu1_pipe_cells, int *amu1
_add_cells,
                                                int *amu3_pipe_cells, int *amu3
_add_cells )
{
        int y;

        amu2 *amu2_tmp;

        /*
                allocates memory for one element based on the size of 'tac'
                (tac_Results structure).  It returns a pointer to the place for t
he
                next empty space for data storage if successful or a NULL otherwi
se.
        */
        if( (amu2_tmp = (amu2 *) calloc( 1, sizeof(amu2) ) ) == NULL)
        {
                printf("\nCan not allocate memory for amu2 data");
                printf("\n\nPress any key to continue");
                getch();
        }


        amu2_tmp -> chip = chip_num;

        for( y = 0; y < 17; y++ )
        {
                amu2_tmp -> num_bad_cells_1amupipe[ y ] = *amu1_pipe_cells;
                amu2_tmp -> num_bad_cells_3amupipe[ y ] = *amu3_pipe_cells;
                amu2_tmp -> num_bad_cells_1amuadd[ y ] = *amu1_add_cells;
                amu2_tmp -> num_bad_cells_3amuadd[ y ] = *amu3_add_cells;
                amu1_pipe_cells ++;
                amu3_pipe_cells ++;
                amu1_add_cells ++;
                amu3_add_cells ++;
        }

        /*
                stores the address of the previous record into the current record
s
                structure to allow for first in last out operation
        */
        amu2_tmp -> amu2_Previous_add = *amu2_results_add;

        /*
                updates pointer to address of current record
        */
        *amu2_results_add = amu2_tmp;
```



```
}
```

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = TEST
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\WOMAC\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = AMU_TEST.C
FIRSTCPP =
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AL /Od /D "_DEBUG" /D "_DOS" /FR /Fd"TEST.PD
B"
CFLAGS_R_DEXE = /nologo /Gs /G2 /W3 /AM /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /CO /STACK:5120
LFLAGS_R_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /STACK:5120
LIBS_D_DEXE = oldnames llibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = AMU_TEST.SBR \
               PARSE.SBR \
               TAC_TEST.SBR \
               TEST.SBR


AMU_TEST_DEP = c:\womac\variable.h


PARSE_DEP = c:\womac\variable.h


TAC_TEST_DEP = c:\womac\variable.h
```



```
TEST_DEP = c:\womac\variable.h


all:    $(PROJ).EXE $(PROJ).BSC

AMU_TEST.OBJ:   AMU_TEST.C $(AMU_TEST_DEP)
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c AMU_TEST.C

PARSE.OBJ:      PARSE.C $(PARSE_DEP)
        $(CC) $(CFLAGS) $(CUSEPCHFLAG) /c PARSE.C

TAC_TEST.OBJ:   TAC_TEST.C $(TAC_TEST_DEP)
        $(CC) $(CFLAGS) $(CUSEPCHFLAG) /c TAC_TEST.C

TEST.OBJ:       TEST.C $(TEST_DEP)
        $(CC) $(CFLAGS) $(CUSEPCHFLAG) /c TEST.C

$(PROJ).EXE::   AMU_TEST.OBJ PARSE.OBJ TAC_TEST.OBJ TEST.OBJ $(OBJS_EXT) $(DEFFIL
E)
        echo >NUL @<<$(PROJ).CRF
AMU_TEST.OBJ +
PARSE.OBJ +
TAC_TEST.OBJ +
TEST.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

```
/*
        amu_test.C
        WRITTEN BY:     MICHAEL WOMAC
        DATE: 27 Dec 1994 - 30 Dec 1994
        Version: 1.0
*/

/*************************************************************************/
/********                        Headers                          ********/
/*************************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "c:\womac\variable.h"
/*#include <alloc.h>*/ /* c++ compiler */
/*#include <malloc.h>*/   /* quick c compiler */

/*************************************************************************/
/********                        Definitions                      ********/
/*************************************************************************/


/*************************************************************************/
/********                        Global Variables                 ********/
/*************************************************************************/

 extern int J2_data[48][17];

/*************************************************************************/
/********                        Structures                       ********/
/*************************************************************************/


/*************************************************************************/
/********                        Functions                        ********/
/*************************************************************************/

void amu_test( int *, amu ** );
void save_amu_data( amu **, int, int, int, long, long, long, long, long,
                                                    long, long, long, long, long, i
nt, int);

/*************************************************************************/
/********                        Code                             ********/
/*************************************************************************/



/*************************************************************************/

/*
        Function Name: amu_test
        Requires:       pointer to chip_num
                                                pointer to pointer to amu structu
re
        Returns:        void
        Purpose:    Count and locate errors of Pb chip data files.  These errors
                                        are then recorded by calling save_amu_dat
a function
*/

/*************************************************************************/

void amu_test( int *chip_num, amu **amu_results_add )
```



```
{




        /**** COUNTERS ****/
        int row_column;
        int a1_5v_cell = 0, a3_5v_cell = 0; /* counts 1.5 & 3.5v cell failures */

        long pipes1_failed = 0;             /* counts 1.5v pipe failures */
        long pipes3_failed = 0;             /* counts 3.5v pipe failures */
        long cells_failed_this_pipe = 0;    /* counts cells failed per pipe */

        /**** CONSTANTS ****/
        int amu_limit[2];
        int amu_tolerance;
        int pipe_failure_criteria;
        int amu_add_failure_critia;

        /**** ARRAYS FOR ADC TESTING ****/
        long amu_1adj_failures[16];
        long amu_3adj_failures[16];
        long amu_[16];

        /**** COUNTERS FOR ADC TESTING ****/
        long amu_1add_failure = 0;
        long amu_3add_failure = 0;

        /**** CONSTANTS FOR ADC TESTING ****/
        int single_failure_only = 0;
        int hard_amu_add_failure  = 0;

        /*
                locators: are a binary word stored with hexadecimal bits indicati
ng
                                        a particular location to be bad by virtu
e of its bit
                                        assignment. Decoding will be done in ano
ther function/prg.
        */

        /**** LOCATORS ****/

        long badcell1_col_location = 0;   /* locates 1.5v column cell failuires *
/
        long badcell1_row_location = 0;   /* locates 1.5v row cell failuires */
        long badcell3_col_location = 0;   /* locates 3.5v column cell failuires *
/
        long badcell3_row_location = 0;   /* locates 3.5v row cell failuires */
        long badpipe1_location = 0;       /* locates 1.5v pipe failuires */
        long badpipe3_location = 0;       /* locates 3.5v pipe failuires */


        /** 1.5V data **/
        int c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15;

        int r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;


        /** 3.5V data **/
        int c30, c31, c32, c33, c34, c35, c36, c37, c38, c39, c300, c310, c320;
        int c330, c340, c350;
        int r30, r31, r32, r33, r34, r35, r36, r37, r38, r39, r300, r310, r320;
        int r330, r340, r350;
```

```
/* pipe lock out variables */

/** 1.5V pipe data **/
int c100, c110, c120, c130, c140, c150, c160, c170, c180, c190, c1000;
int c1100, c1200, c1300, c1400, c1500;

/** 3.5V pipe data **/
int c3000, c3100, c3200, c3300, c3400, c3500, c3600, c3700, c3800, c3900;

int c30000, c31000, c32000, c33000, c34000, c35000;

/**** Initializations ****/

/** 1.5V data **/
c0=c1=c2=c3=c4=c5=c6=c7=c8=c9=c10=c11=c12=c13=c14=c15=0;
r0=r1=r2=r3=r4=r5=r6=r7=r8=r9=r10=r11=r12=r13=r14=r15=0;

/** 3.5V data **/
c30=c31=c32=c33=c34=c35=c36=c37=c38=c39=c300=c310=c320=c330=c340=c350=0;
r30=r31=r32=r33=r34=r35=r36=r37=r38=r39=r300=r310=r320=r330=r340=r350=0;

/** 1.5v pipe data **/
c100=c110=c120=c130=c140=c150=c160=c170=c180=c190=c1000=c1100=c1200=0;
c1300=c1400=c1500=0;

/** 3.5v pipe data **/
c3000=c3100=c3200=c3300=c3400=c3500=c3600=c3700=c3800=c3900=c30000=0;
c31000=c32000=c33000=c34000=c35000=0;

amu_limit[1] = 716; /* 1.5V input */
amu_limit[0] = 307; /* 3.5V input */

amu_tolerance = 100;

pipe_failure_criteria = 1; /* Determines number of cells to fail a pipe *

amu_add_failure_critia = 15; /* "                    " to fail amu add */


amu_[0] = 0x1;
amu_[1] = 0x2;
amu_[2] = 0x4;
amu_[3] = 0x8;
amu_[4] = 0x10;
amu_[5] = 0x20;
amu_[6] = 0x40;
amu_[7] = 0x80;
amu_[8] = 0x100;
amu_[9] = 0x200;
amu_[10] = 0x400;
amu_[11] = 0x800;
amu_[12] = 0x1000;
amu_[13] = 0x2000;
amu_[14] = 0x4000;
amu_[15] = 0x8000;

for( row = 0; row < 16; row ++)
{
        amu_1add_failures[row]=0;
        amu_3add_failures[row]=0;
}

/*****************************************************************/
/*                        Testing AMU                            */
```



```
/*****************************************************************/

                for( column = 0; column < 16; column ++)
                {
                        for( row = 16; row < 32; row++ )
                        {
                                if( (  J2_data[row][column] <
                                                amu_limit[1] -  amu_toler
ance)
                                        ||( J2_data[row][column] >
                                                amu_limit[1] + amu_tolera
nce) )
                                {
                                        cells_failed_this_pipe ++;
                                        a1_5v_cell ++;
                                        if( column == 0 && c0 == 0 )
                                        {
                                                badcell1_col_location +=
0x1;
                                                c0 = 1;
                                        }
                                        if( column == 1 && c1 == 0 )
                                        {
                                                badcell1_col_location +=
0x2;
                                                c1 = 1;
                                        }
                                        if( column == 2 && c2 == 0 )
                                        {
                                                badcell1_col_location +=
0x4;
                                                c2 = 1;
                                        }
                                        if( column == 3 && c3 == 0 )
                                        {
                                                badcell1_col_location +=
0x8;
                                                c3 = 1;
                                        }
                                        if( column == 4 && c4 == 0 )
                                        {
                                                badcell1_col_location +=
0x10;
                                                c4 = 1;
                                        }
                                        if( column == 5 && c5 == 0 )
                                        {
                                                badcell1_col_location +=
0x20;
                                                c5 = 1;
                                        }
                                        if( column == 6 && c6 == 0 )
                                        {
                                                badcell1_col_location +=
0x40;
                                                c6 = 1;
                                        }
                                        if( column == 7 && c7 == 0 )
                                        {
                                                badcell1_col_location +=
0x80;
                                                c7 = 1;
                                        }
                                        if( column == 8 && c8 == 0 )
                                        {
```

```
                badcell1_col_location +=
0x100;          c8 = 1;
            }
            if( column == 9 && c9 == 0 )
            {
                badcell1_col_location +=
0x200;          c9 = 1;
            }
            if( column == 10 && c10 == 0 )
            {
                badcell1_col_location +=
0x400;          c10 = 1;
            }
            if( column == 11 && c11 == 0 )
            {
                badcell1_col_location +=
0x800;          c11 = 1;
            }
            if( column == 12 && c12 == 0 )
            {
                badcell1_col_location +=
0x1000;         c12 = 1;
            }
            if( column == 13 && c13 == 0 )
            {
                badcell1_col_location +=
0x2000;         c13 = 1;
            }
            if( column == 14 && c14 == 0 )
            {
                badcell1_col_location +=
0x4000;         c14 = 1;
            }
            if( column == 15 && c15 == 0 )
            {
                badcell1_col_location +=
0x8000;         c15 = 1;
            }
            if( row == 16 && r0 == 0 )
            {
                badcell1_row_location +=
0x1;            r0 = 1;
            }
            if( row == 17 && r1 == 0 )
            {
                badcell1_row_location +=
0x2;            r1 = 1;
            }
            if( row == 18 && r2 == 0 )
            {
                badcell1_row_location +=
0x4;            r2 = 1;
            }
            if( row == 19 && r3 == 0 )
            {
```



```
                badcell1_row_location +=
0x8;            r3 = 1;
            }
            if( row == 20 && r4 == 0 )
            {
                badcell1_row_location +=
0x10;           r4 = 1;
            }
            if( row == 21 && r5 == 0 )
            {
                badcell1_row_location +=
0x20;           r5 = 1;
            }
            if( row == 22 && r6 == 0 )
            {
                badcell1_row_location +=
0x40;           r6 = 1;
            }
            if( row == 23 && r7 == 0 )
            {
                badcell1_row_location +=
0x80;           r7 = 1;
            }
            if( row == 24 && r8 == 0 )
            {
                badcell1_row_location +=
0x100;          r8 = 1;
            }
            if( row == 25 && r9 == 0 )
            {
                badcell1_row_location +=
0x200;          r9 = 1;
            }
            if( row == 26 && r10 == 0 )
            {
                badcell1_row_location +=
0x400;          r10 = 1;
            }
            if( row == 27 && r11 == 0 )
            {
                badcell1_row_location +=
0x800;          r11 = 1;
            }
            if( row == 28 && r12 == 0 )
            {
                badcell1_row_location +=
0x1000;         r12 = 1;
            }
            if( row == 29 && r13 == 0 )
            {
                badcell1_row_location +=
0x2000;         r13 = 1;
            }
            if( row == 30 && r14 == 0 )
            {
```

```
                                        badcell1_row_location +=
0x4000;
                                        r14 = 1;
                                    }
                                    if( row == 31 && r15 == 0 )
                                    {
                                        badcell1_row_location +=
0x8000;
                                        r15 = 1;
                                    }
                                }
                            }
                            if ( cells_failed_this_pipe >= pipe_failure_crite
ria )
                            {
                                pipes1_failed ++;
                            }
                            if (cells_failed_this_pipe >= pipe_failure_criter
ia )
                            {
                                if( column == 0 && c100 == 0 )
                                {
                                    badpipe1_location += 0x1;
                                    c100 = 1;
                                }
                                if( column == 1 && c110 == 0 )
                                {
                                    badpipe1_location += 0x2;
                                    c110 = 1;
                                }
                                if( column == 2 && c120 == 0 )
                                {
                                    badpipe1_location += 0x4;
                                    c120 = 1;
                                }
                                if( column == 3 && c130 == 0 )
                                {
                                    badpipe1_location += 0x8;
                                    c130 = 1;
                                }
                                if( column == 4 && c140 == 0 )
                                {
                                    badpipe1_location += 0x10;
                                    c140 = 1;
                                }
                                if( column == 5 && c150 == 0 )
                                {
                                    badpipe1_location += 0x20;
                                    c150 = 1;
                                }
                                if( column == 6 && c160 == 0 )
                                {
                                    badpipe1_location += 0x40;
                                    c160 = 1;
                                }
                                if( column == 7 && c170 == 0 )
                                {
                                    badpipe1_location += 0x80;
                                    c170 = 1;
                                }
                                if( column == 8 && c180 == 0 )
                                {
                                    badpipe1_location += 0x100;
                                    c180 = 1;
                                }
                                if( column == 9 && c190 == 0 )
```



```
                                {
                                    badpipe1_location += 0x200;
                                    c190 = 1;
                                }
                                if( column == 10 && c1000 == 0 )
                                {
                                    badpipe1_location += 0x400;
                                    c1000 = 1;
                                }
                                if( column == 11 && c1100 == 0 )
                                {
                                    badpipe1_location += 0x800;
                                    c1100 = 1;
                                }
                                if( column == 12 && c1200 == 0 )
                                {
                                    badpipe1_location += 0x1000;
                                    c1200 = 1;
                                }
                                if( column == 13 && c1300 == 0 )
                                {
                                    badpipe1_location += 0x2000;
                                    c1300 = 1;
                                }
                                if( column == 14 && c1400 == 0 )
                                {
                                    badpipe1_location += 0x4000;
                                    c1400 = 1;
                                }
                                if( column == 15 && c1500 == 0 )
                                {
                                    badpipe1_location += 0x8000;
                                    c1500 = 1;
                                }
                            }
                            cells_failed_this_pipe = 0;
                        }
/* ******************** 3.5 v section ********************************** */
                        for( column = 0; column < 16; column ++ )
                        {
                            for( row = 32; row < 48; row ++ )
                            {
                              if(((J2_data[row][column])<(amu_limit[0] - amu_
tolerance))
                               || ((J2_data[row][column])>(amu_limit[0] + amu_
tolerance)))
                                {
                                    cells_failed_this_pipe ++;
                                    a3_5v_cell ++;
                                    if( column == 0 && c30 == 0 )
                                    {
                                        badcell3_col_location +=
0x1;
                                        c30 = 1;
                                    }
                                    if( column == 1 && c31 == 0 )
                                    {
                                        badcell3_col_location +=
0x2;
                                        c31 = 1;
                                    }
                                    if( column == 2 && c32 == 0 )
                                    {
                                        badcell3_col_location +=
```

```
0x4;                                    c32 = 1;
                                        }
                                        if( column == 3 && c33 == 0 )
                                        {
0x8;                                    badcell3_col_location +=
                                        c33 = 1;
                                        }
                                        if( column == 4 && c34 == 0 )
                                        {
0x10;                                   badcell3_col_location +=
                                        c34 = 1;
                                        }
                                        if( column == 5 && c35 == 0 )
                                        {
0x20;                                   badcell3_col_location +=
                                        c35 = 1;
                                        }
                                        if( column == 6 && c36 == 0 )
                                        {
0x40;                                   badcell3_col_location +=
                                        c36 = 1;
                                        }
                                        if( column == 7 && c37 == 0 )
                                        {
0x80;                                   badcell3_col_location +=
                                        c37 = 1;
                                        }
                                        if( column == 8 && c38 == 0 )
                                        {
0x100;                                  badcell3_col_location +=
                                        c38 = 1;
                                        }
                                        if( column == 9 && c39 == 0 )
                                        {
0x200;                                  badcell3_col_location +=
                                        c39 = 1;
                                        }
                                        if( column == 10 && c300 == 0 )
                                        {
0x400;                                  badcell3_col_location +=
                                        c300 = 1;
                                        }
                                        if( column == 11 && c310 == 0 )
                                        {
0x800;                                  badcell3_col_location +=
                                        c310 = 1;
                                        }
                                        if( column == 12 && c320 == 0 )
                                        {
0x1000;                                 badcell3_col_location +=
                                        c320 = 1;
                                        }
                                        if( column == 13 && c330 == 0 )
                                        {
                                        badcell3_col_location +=
```



```
0x2000;                                 c330 = 1;
                                        }
                                        if( column == 14 && c340 == 0 )
                                        {
0x4000;                                 badcell3_col_location +=
                                        c340 = 1;
                                        }
                                        if( column == 15 && c350 == 0 )
                                        {
0x8000;                                 badcell3_col_location +=
                                        c350 = 1;
                                        }
                                        if( row == 32 && r30 == 0 )
                                        {
0x1;                                    badcell3_row_location +=
                                        r30 = 1;
                                        }
                                        if( row == 33 && r31 == 0 )
                                        {
0x2;                                    badcell3_row_location +=
                                        r31 = 1;
                                        }
                                        if( row == 34 && r32 == 0 )
                                        {
0x4;                                    badcell3_row_location +=
                                        r32 = 1;
                                        }
                                        if( row == 35 && r33 == 0 )
                                        {
0x8;                                    badcell3_row_location +=
                                        r33 = 1;
                                        }
                                        if( row == 36 && r34 == 0 )
                                        {
0x10;                                   badcell3_row_location +=
                                        r34 = 1;
                                        }
                                        if( row == 37 && r35 == 0 )
                                        {
0x20;                                   badcell3_row_location +=
                                        r35 = 1;
                                        }
                                        if( row == 38 && r36 == 0 )
                                        {
0x40;                                   badcell3_row_location +=
                                        r36 = 1;
                                        }
                                        if( row == 39 && r37 == 0 )
                                        {
0x80;                                   badcell3_row_location +=
                                        r37 = 1;
                                        }
                                        if( row == 40 && r38 == 0 )
                                        {
                                        badcell3_row_location +=
```

```
0x100;
                                        r38 = 1;
                                    }
                                    if( row == 41 && r39 == 0 )
                                    {
                                        badcell3_row_location |=
0x200;
                                        r39 = 1;
                                    }
                                    if( row == 42 && r300 == 0 )
                                    {
                                        badcell3_row_location |=
0x400;
                                        r300 = 1;
                                    }
                                    if( row == 43 && r310 == 0 )
                                    {
                                        badcell3_row_location |=
0x800;
                                        r310 = 1;
                                    }
                                    if( row == 44 && r320 == 0 )
                                    {
                                        badcell3_row_location |=
0x1000;
                                        r320 = 1;
                                    }
                                    if( row == 45 && r330 == 0 )
                                    {
                                        badcell3_row_location |=
0x2000;
                                        r330 = 1;
                                    }
                                    if( row == 46 && r340 == 0 )
                                    {
                                        badcell3_row_location |=
0x4000;
                                        r340 = 1;
                                    }
                                    if( row == 47 && r350 == 0 )
                                    {
                                        badcell3_row_location |=
0x8000;
                                        r350 = 1;
                                    }
                                }
                            }
                            if ( cells_failed_this_pipe >= pipe_failure_crite
ria)
                            {
                                pipes3_failed ++;
                            }
                            if (cells_failed_this_pipe >= pipe_failure_criter
ia )
                            {
                                if( column == 0 && c3000 == 0 )
                                {
                                    badpipe3_location |= 0x1;
                                    c3000 = 1;
                                }
                                if( column == 1 && c3100 == 0 )
                                {
                                    badpipe3_location |= 0x2;
                                    c3100 = 1;
                                }
                                if( column == 2 && c3200 == 0 )
```



```
                                {
                                    badpipe3_location |= 0x4;
                                    c3200 = 1;
                                }
                                if( column == 3 && c3300 == 0 )
                                {
                                    badpipe3_location |= 0x8;
                                    c3300 = 1;
                                }
                                if( column == 4 && c3400 == 0 )
                                {
                                    badpipe3_location |= 0x10;
                                    c3400 = 1;
                                }
                                if( column == 5 && c3500 == 0 )
                                {
                                    badpipe3_location |= 0x20;
                                    c3500 = 1;
                                }
                                if( column == 6 && c3600 == 0 )
                                {
                                    badpipe3_location |= 0x40;
                                    c3600 = 1;
                                }
                                if( column == 7 && c3700 == 0 )
                                {
                                    badpipe3_location |= 0x80;
                                    c3700 = 1;
                                }
                                if( column == 8 && c3800 == 0 )
                                {
                                    badpipe3_location |= 0x100;
                                    c3800 = 1;
                                }
                                if( column == 9 && c3900 == 0 )
                                {
                                    badpipe3_location |= 0x200;
                                    c3900 = 1;
                                }
                                if( column == 10 && c30000 == 0 )
                                {
                                    badpipe3_location |= 0x400;
                                    c30000 = 1;
                                }
                                if( column == 11 && c31000 == 0 )
                                {
                                    badpipe3_location |= 0x800;
                                    c31000 = 1;
                                }
                                if( column == 12 && c32000 == 0 )
                                {
                                    badpipe3_location |= 0x1000;
                                    c32000 = 1;
                                }
                                if( column == 13 && c33000 == 0 )
                                {
                                    badpipe3_location |= 0x2000;
                                    c33000 = 1;
                                }
                                if( column == 14 && c34000 == 0 )
                                {
                                    badpipe3_location |= 0x4000;
                                    c34000 = 1;
                                }
                                if( column == 15 && c35000 == 0 )
                                {
```

```
                                        badpipe3_location += 0x8000;
                                        c35000 = 1;
                                }
                        }
                        cells_failed_this_pipe = 0;
                }

                        /**** Testing for AMU ADC failures ****/

        for( row = 16; row < 32; row ++ )
        {
                for( column = 1; column < 16; column ++ )
                {

                        if(((J2_data[row][column])<(amu_limit[1] - amu_tolerance)
)
                                || ((J2_data[row][column])>(amu_limit[1] + amu_
tolerance)))
                        {
                                amu_1add_failures[row-16] ++;
                        }
                }
        }

        for( row = 0; row < 16; row ++ )
        {
                if( (amu_1add_failures[row] >= amu_add_failure_critia) )
                {
                        amu_1add_failure += amu_[row];
                }
   }

        for( row = 32; row < 48; row ++ )
        {
                for( column = 1; column < 16; column ++ )
                {

                        if(((J2_data[row][column])<(amu_limit[0] - amu_tolerance)
)
                                || ((J2_data[row][column])>(amu_limit[0] + amu_
tolerance)))
                        {
                                amu_3add_failures[row-32] ++;
                        }
                }
        }

        for( row = 0; row < 16; row ++ )
        {
                if( (amu_3add_failures[row] >= amu_add_failure_critia) )
                {
                        amu_3add_failure += amu_[row];
                }
        }

        if( amu_1add_failure == 65535 || amu_3add_failure == 65535 )
        {
                hard_amu_add_failure = 1;
        }

        if( amu_1add_failure != 65535 && amu_3add_failure == 65535 )
        {
                single_failure_only = 1;
        }
        if( amu_1add_failure == 65535 && amu_3add_failure != 65535 )
        {
```



```
                single_failure_only = 1;
        }

        save_amu_data( amu_results_add, *chip_num, a1_5v_cell, a3_5v_cell,
                                                        pipes1_failed, pipes3_failed, bad
cell1_row_location,
                                                        badcell1_col_location, badpipe1_l
ocation,
                                                        badcell3_row_location, badcell3_c
ol_location,
                                                        badpipe3_location, amu_1add_failu
re, amu_3add_failure,
                                                        hard_amu_add_failure, single_fail
ure_only );

}

void save_amu_data( amu **amu_results_add, int chip_num, int cells_1,
                    int cells_3, long pipes_1, long pipes_3,
                    long badcell1_row, long badcell1_col, long badpipe1_loc,
                    long badcell3_row, long badcell3_col, long badpipe3_loc,
                    long amu_1add_failure, long amu_3add_failure,
                    int hard_amu_add_failure, int single_failure_only )
{
        amu  *amu_tmp;

        /*
                allocates memory for one element based on the size of 'foo'
                (foo_Results structure).  It returns a pointer to the place for t
he
                next empty space for data storage if successful or a NULL otherwi
se.
        */
        if( (amu_tmp = (amu *) calloc( 1, sizeof(amu) ) ) == NULL)
        {
                printf("\nCan not allocate memory for amu data");
                printf("\n\nPress any key to continue");
                getch();
        }



        amu_tmp -> chip = chip_num;
        amu_tmp -> cells_1_5 = cells_1;
        amu_tmp -> cells_3_5 = cells_3;
        amu_tmp -> pipes_1_5 = pipes_1;
        amu_tmp -> pipes_3_5 = pipes_3;
        amu_tmp -> badcell1_row = badcell1_row;
        amu_tmp -> badcell1_col = badcell1_col;
        amu_tmp -> badcell3_row = badcell3_row;
        amu_tmp -> badcell3_col = badcell3_col;
        amu_tmp -> badpipe1_loc = badpipe1_loc;
        amu_tmp -> badpipe3_loc = badpipe3_loc;
        amu_tmp -> amu_1add_failure = amu_1add_failure;
        amu_tmp -> amu_3add_failure = amu_3add_failure;
        amu_tmp -> hard_amu_add_failure = hard_amu_add_failure;
        amu_tmp -> single_failure_only = single_failure_only;

        /*
                stores the address of the previous record into the current record
s
                structure to allow for first in last out operation
        */
        amu_tmp -> amu_Previous_add = *amu_results_add;
```

```
        /*
                updates pointer to address of current record
        */
        *amu_results_addr = amu_tmp;
}
```

```
/*
        parse.c  ---- A Utility program -------
        Written by: Lloyd Clonts
        Modified by: Mike Womac
        Modified on: 5 Jan 1995
        Version: 2.0 (lloyds version is version 1.0 )
*/

/*************************************************************************/
/********                    Headers                            ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "c:\womac\variable.h"

/*************************************************************************/
/********                    Definitions                        ********/
/*************************************************************************/

#define size_of_column 16

/*************************************************************************/
/********                    Global Variables                   ********/
/*************************************************************************/

extern char *J2_column[ size_of_column];

/*************************************************************************/
/********                    Structures                         ********/
/*************************************************************************/


/*************************************************************************/
/********                    Functions                          ********/
/*************************************************************************/

void Parse_J2( char *, int *, char ** );

/*************************************************************************/
/********                    Code                               ********/
/*************************************************************************/



/*************************************************************************/

/*
        Function Name:  Parse_data ( & Parse_J2_data )
        Requires:       pointer to a predefined character array (buffer)
                                        pointer to ?????
                                        pointer to a pointer to a charac
ter array with desired
                                         size of parsed word
        Returns: void
        Purpose: seperates a buffer full (his case one line) of data into a
                        'word' array that is predefined in the calling pr
g.  The
                        determination of the buffer length is the charact
ers that
                        are checked for in the first while statement.  Th
e maximum
                        buffer size is actually determined in the calling
 prg.
```



```
*/
/*************************************************************************/



void Parse_data( char *buf, int *Count, char **Word)
{
        *Word = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (i.e. \0) character in the contents
 of the current
                                buffer address then it increments to the next buf
for address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        Word[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                (*buf != ',') )
                {
                        /*
                                increments buffer address
                        */
                        buf++;
                }
        }
        *buf = '\0';
}

void Parse_J2( char *buf, int *Count, char **J2_column)
{
        *J2_column = buf;
        *Count = 0;

        /*
                Defines what terminates parsing of data
        */
```

```
        while ( (*buf != '\n') && (*buf != '\0') )
        {
                /*
                        Defines characters not wanted in word array
                */
                while ( (*buf == ' ' ) || (*buf == ',') )
                {
                        /*
                                Stores a NULL (\0) character in the contents of t
he current
                                buffer address then it increments to the next buf
fer address
                        */
                        *(buf++)='\0';
                }

                /*
                        Stores wanted characters in 'word' array
                */
                if ( (*buf != '\n' ) && (*buf != '\0' ) )
                {
                        J2_column[ (*Count)++ ] = buf;
                }

                /*
                        Defines characters not wanted in array which allows the s
toring
                        of all other data
                */
                while ( (*buf != ' ') && (*buf != '\0') && (*buf != '\n' ) &&
                                (*buf != ',') )
                {
                        /*
                                Increments buffer address
                        */
                        buf++;

                }
        }

        /*
                Stores a NULL character in last used address in the buffer
        */
        *buf = '\0';
}
```

```
/*
        tac_test.C
        WRITTEN BY:     MICHAEL WOMAC
        DATE: 20 Dec 1994 - 30 Dec 1994
        Version: 1.0
*/

/*************************************************************************/
/********                                                         ********/
/********                      Headers                            ********/
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "c:\womac\variable.h"
#include <stdlib.h>

/*************************************************************************/
/********                                                         ********/
/********                      Definitions                        ********/
/*************************************************************************/


/*************************************************************************/
/********                                                         ********/
/********                      Global Variables                   ********/
/*************************************************************************/

  extern int J2_data[48][17];

/*************************************************************************/
/********                                                         ********/
/********                      Structures                         ********/
/*************************************************************************/


/*************************************************************************/
/********                                                         ********/
/********                      Functions                          ********/
/*************************************************************************/

void tac_test( int *, tac ** );
void save_tac_data( tac **, int, int, int, int, int, int, int);

/*************************************************************************/
/********                                                         ********/
/********                      Code                               ********/
/*************************************************************************/



/************************************************************************/

/*
        Function Name: tac_test
        Requires:   address of chip_num
                                        pointer to a pointer to the tac structure

        Returns:    void
        Purpose:    Count an locate errors of Pb chip data files.  These errors
                                        are then recorded by calling save tac dat
a
*/

/************************************************************************/

void tac_test( int *chip_num, tac **tac_results_add )
{
        /**** COUNTERS ****/
        int row, column, r;
        int tac_time_out = 0; /* counts timeout errors */
        int p20ns = 0;        /* counts 20ns pulse errors */
```



```
        int p100ns = 0;       /* counts 100ns pulse errors */
        int tac_TO_row_failure[16]; /* counts tac timeout row failures */

        /**** CONSTANTS ****/
        int tac_limit[2];
        int tchan[16];


        /*
                locators: are a binary word stored with hexadecimal bits indicati
ng
                                        a particular location to be bad by virtu
e of its bit
                                        assignment. Decoding will be done in ano
ther function/prg.
        */

        /**** LOCATORS ****/
        int tac_tchan_failures = 0; /* locates timeout failures */
        int tac_p20ns_failures = 0; /* locates 20ns failures only */
        int tac_p100ns_failures = 0; /* locates 100ns failures only */
        /*int tac_bias_failure = 0;*/ /* locates possible tac bias failures */

        /**** LOCK OUT variables ****/
        int p0, p1, p2, p3, p4, p5, p6, p7;
        int t0, t1, t2, t3, t4, t5, t6, t7;
        int p10, p11, p12, p13, p14, p15, p16, p17;

        /**** Initializations ****/
        p0=p1=p2=p3=p4=p5=p6=p7=0;
        t0=t1=t2=t3=t4=t5=t6=t7=0;
        p10=p11=p12=p13=p14=p15=p16=p17=0;



        tac_limit[1] = 900;
        tac_limit[0] = 500;

        tchan[0] = 0;
        tchan[1] = 0x1;
        tchan[2] = 0;
        tchan[3] = 0x2;
        tchan[4] = 0;
        tchan[5] = 0x4;
        tchan[6] = 0;
        tchan[7] = 0x8;
        tchan[8] = 0;
        tchan[9] = 0x10;
        tchan[10] = 0;
        tchan[11] = 0x20;
        tchan[12] = 0;
        tchan[13] = 0x40;
        tchan[14] = 0;
        tchan[15] = 0x80;


        for( r = 0; r < 17; r++)
        {
                tac_TO_row_failure[r] = 0;
}

        /************************************************************************/
        /*                          Tac test 01                                 */
        /************************************************************************/
```

```
/*****************************************************************/
/*   Testing TAC channel 0 -> Tac test #1 -> 20nS & 100nS Pulse   */
/*****************************************************************/


for( row = 0; row < 16; row ++ )
{
        for( column = 1, column < 9; column ++ )
        {
                /* 20 ns*/
                if( row == 0 && column == 1 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {
                                p20ns ++;
                                if ( p0 == 0 )
                                {

                                        tac_p20ns_failures += 0x1;
                                        p0 = 1;

                                }
                        }
                }
                /* 100 ns*/
                if( row == 1 && column == 1 )
                {
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column) >
                                tac_limit[ 0 ] + 100 ))
                        {
                                p100ns ++;
                                if ( p10 == 0 )
                                {

                                        tac_p100ns_failures += 0x1;
                                        p10 = 1;
                                }

                        }
                }

                if( row == 0 && column != 1 || row == 1 && column != 1 )
                {
                        if( (J2_data[row][column] > 250)   )
                        {
                                tac_time_out ++;
                                if ( t0 == 0 )
                                {
                                        tac_tchan_failures += 0x1;
                                        t0 = 1;
                                }
                        }
                }
        }



/*****************************************************************/
/*   Testing TAC channel 1 -> Tac test #1 -> 20nS & 100nS Pulse   */
/*****************************************************************/
```



```
        for( column = 1; column < 9 ; column ++ )
        {
                if( row == 2 && column == 2 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {
                                p20ns ++;
                                if( p1 == 0)
                                {

                                        tac_p20ns_failures += 0x2;
                                        p1 = 1;
                                }

                        }
                }
                if( row == 3 && column == 2 )
                {
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 0 ] + 100 ))
                        {
                                p100ns ++;
                                if( p11 == 0)
                                {

                                        tac_p100ns_failures += 0x2;
                                        p11 = 1;
                                }

                        }
                }

                if( row == 2 && column != 2 || row == 3 && column != 2)
                {
                        if( (J2_data[row][column] > 250)   )
                        {
                                tac_time_out ++;
                                if ( t1 == 0 )
                                {
                                        tac_tchan_failures += 0x2;
                                        t1 = 1;
                                }
                        }
                }
        }



/*****************************************************************/
/*   Testing TAC channel 2 -> Tac test #1 -> 20nS & 100nS Pulse   */
/*****************************************************************/


        for( column = 1; column < 9 ; column ++ )
        {
                if( row == 4 && column == 3 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {
```

```
                                p20ns ++;
                                if ( p2 == 0)
                                {

                                        tac_p20ns_failures += 0x4;
                                        p2 = 1;
                                }

                        }
                }
                if( row == 5 && column == 3 )
                {
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 0 ] + 100 ))
                        {
                                p100ns ++;
                                if ( p12 == 0)
                                {

                                        tac_p100ns_failures += 0x4;
                                        p12 = 1;
                                }
                        }
                }

                if( row == 4 && column != 3 || row == 5 && column != 3)
                {
                        if( (J2_data[row][column) > 250)    )
                        {
                                tac_time_out ++;
                                if ( t2 == 0 )
                                {
                                        tac_tchan_failures += 0x4;
                                        t2 = 1;
                                }
                        }
                }
        }


/*****************************************************************/
/*   Testing TAC channel 3 -> Tac test #1 -> 20nS & 100nS Pulse    */
/*****************************************************************/


        for( column = 1; column < 9 ; column ++ )
        {
                if( row == 6 && column == 4 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100 ) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {
                                p20ns ++;
                                if ( p3 == 0)
                                {

                                        tac_p20ns_failures += 0x8;
                                        p3 = 1;
                                }
                        }
                }
                if( row == 7 && column == 4 )
                {
```



```
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 0 ] + 100 ))
                        {
                                p100ns ++;
                                if ( p13 == 0)
                                {

                                        tac_p100ns_failures += 0x8;
                                        p13 = 1;
                                }
                        }
                }

                if( row == 6 && column != 4 || row == 7 && column != 4 )
                {
                        if( (J2_data[row][column) > 250)    )
                        {
                        tac_time_out ++;
                                if ( t3 == 0 )
                                {
                                        tac_tchan_failures += 0x8;
                                        t3 = 1;
                                }
                        }
                }
        }


/*****************************************************************/
/*   Testing TAC channel 4 -> Tac test #1 -> 20nS & 100nS Pulse    */
/*****************************************************************/

        for( column = 1; column < 9 ; column ++ )
        {
                if( row == 8 && column == 5 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100 ) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {
                                p20ns ++;
                                if ( p4 == 0)
                                {

                                        tac_p20ns_failures += 0x10;
                                        p4 = 1;
                                }
                        }
                }
                if( row == 9 && column == 5 )
                {
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 0 ] + 100 ))
                        {
                                p100ns ++;
                                if ( p14 == 0)
                                {

                                        tac_p100ns_failures += 0x10;
```

```
                                        p14 = 1;
                                }
                        }
                }

                if( row == 8 && column != 5 || row == 9 && column != 5 )
                {
                        if( (J2_data[row][column] > 250)   )
                        {

                                tac_time_out ++;
                                if ( t4 == 0 )
                                {
                                        tac_tchan_failures += 0x10;
                                        t4 = 1;
                                }
                        }
                }
        }


/*****************************************************************/
/*   Testing TAC channel 5 -> Tac test #1 -> 20nS & 100nS Pulse  */
/*****************************************************************/


        for( column = 1; column < 9 ; column ++ )
        {
                if( row == 10 && column == 6 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100 ) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {

                                p20ns ++;
                                if ( p5 != 0)
                                {

                                        tac_p20ns_failures += 0x20;
                                        p5 = 1;
                                }
                        }
                }
                if( row == 11 && column == 6 )
                {
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 0 ] + 100 ))
                        {

                                p100ns ++;
                                if ( p15 != 0)
                                {

                                        tac_p100ns_failures += 0x20;
                                        p15 = 1;
                                }
                        }
                }

                if( row == 10 && column != 6 || row == 11 && column != 6)

                {
                        if( (J2_data[row][column] > 250)   )
```



```
                        {

                                tac_time_out ++;
                                if ( t5 == 0 )
                                {
                                        tac_tchan_failures += 0x20;
                                        t5 = 1;
                                }
                        }
                }
        }


/*****************************************************************/
/*   Testing TAC channel 6 -> Tac test #1 -> 20nS & 100nS Pulse  */
/*****************************************************************/


        for( column = 1; column < 9 ; column ++ )
        {
                if( row == 12 && column == 7 )
                {
                        if( (J2_data[row][column] < tac_limit[ 1 ]
                                - 100 ) || (J2_data[row][column] >
                                tac_limit[ 1 ] + 100 ))
                        {

                                p20ns ++;
                                if ( p6 == 0)
                                {

                                        tac_p20ns_failures += 0x40;
                                        p6 = 1;
                                }
                        }
                }
                if( row == 13 && column == 7 )
                {
                        if( (J2_data[row][column] < tac_limit[ 0 ]
                                - 100) || (J2_data[row][column] >
                                tac_limit[ 0 ] + 100 ))
                        {

                                p100ns ++;
                                if ( p16 == 0)
                                {

                                        tac_p100ns_failures += 0x40;
                                        p16 = 1;
                                }
                        }
                }

                if( row == 12 && column != 7 || row == 13 && column != 7
)
                {
                        if( (J2_data[row][column] > 250)   )
                        {

                                tac_time_out ++;
                                if ( t6 == 0 )
                                {
                                        tac_tchan_failures += 0x40;
                                        t6 = 1;
                                }
```

```
                        }
                    }
                }


        /*****************************************************************/
        /*   Testing TAC channel 7 -> Tac test #1 -> 20nS & 100nS Pulse   */
        /*****************************************************************/


                for( column = 1; column < 9  ; column ++ )
                {
                        if( row == 14 && column == 8 )
                        {
                                if( (J2_data[row][column] < tac_limit[ 1 ]
                                        - 100 ) || (J2_data[row][column] >
                                        tac_limit[ 1 ] + 100 ))
                                {

                                        p20ns ++;
                                        if ( p7 == 0)
                                        {

                                                tac_p20ns_failures += 0x80;
                                                p7 = 1;
                                        }
                                }
                        }
                        if( row == 15 && column == 8 )
                        {
                                if( (J2_data[row][column] < tac_limit[ 0 ]
                                        - 100) || (J2_data[row][column] >
                                        tac_limit[ 0 ] + 100 ))
                                {

                                        p100ns ++;
                                        if ( p17 == 0)
                                        {

                                                tac_p100ns_failures += 0x80;
                                                p17 = 1;
                                        }
                                }
                        }

                        if( row == 14 && column != 8 || row == 15 && column != 8)

                        {
                                if( (J2_data[row][column] > 250)    )
                                {

                                        tac_time_out ++;
                                        if ( t7 == 0 )
                                        {
                                                tac_tchan_failures += 0x80;
                                                t7 = 1;
                                        }
                                }
                        }
                }
        }

        /**** Testing for TAC ADC failures ****/
```



```
        for( row = 0; row < 16; row ++ )
        {
                for( column = 1; column < 9; column ++ )
                {
                        if(
                                (row == 0 && column == 1) || (row == 1 && column
 == 1) ||
                                (row == 2 && column == 2) || (row == 3 && column
 == 2) ||
                                (row == 4 && column == 3) || (row == 5 && column
 == 3) ||
                                (row == 6 && column == 4) || (row == 7 && column
 == 4) ||
                                (row == 8 && column == 5) || (row == 9 && column
 == 5) ||
                                (row == 10 && column == 6) || (row == 11 && colu
mn == 6) ||
                                (row == 12 && column == 7) || (row == 13 && colu
mn == 7) ||
                                (row == 14 && column == 8) || (row == 15 && colu
mn == 8)
                                )
                                {
                                }
                        else
                                {
                                        if( J2_data[row][column] > 250 )
                                        {
                                                tac_T0_row_failure[row] ++;
                                        }
                                }
                }
    }

        for( row = 0; row < 16; row += 2 )
        {
                if( ((tac_T0_row_failure[row] > 5) && (tac_T0_row_failure[row+1]
> 5)) )
                {
                        /*      tac_bois_failure += tchan[row+1];*/
                }
    }



        save_tac_data( tac_results_add, *chip_num, tac_time_out, p20ns, p100ns,
                                                tac_tchan_failures, tac_p20ns_fai
lures,
                                                tac_p100ns_failures );
}

void save_tac_data( tac **tac_results_add, int chip_num, int timeout_failures,
                    int p20ns, int p100ns, int tac_tchan_failures,
                    int tac_p20ns_failures, int tac_p100ns_failures )

{
        tac *tmp;
        /*
                allocates memory for one element based on the size of 'tac'
                (tac_Results structure).  It returns a pointer to the place for t
he
                next empty space for data storage if successful or a NULL otherwi
se.
```

```
        */
        if( (tmp = (tac *) calloc( 1, sizeof(tac) ) ) == NULL)
        {
                printf("\nCan not allocate memory for tac data");
                printf("\n\nPress any key to continue");
                getch();
        }
        tmp -> total_chip_time_out_failures = timeout_failures;
        tmp -> chip = chip_num;
        tmp -> total_20ns_failures = p20ns;
        tmp -> total_100ns_failures = p100ns;
        tmp -> channel_timeout_failures = tac_tchan_failures;
        tmp -> channel_20ns_failures = tac_p20ns_failures;
        tmp -> channel_100ns_failures = tac_p100ns_failures;
        /*tmp -> tac_bais_failure = tac_bais_failure;*/

        /*
                stores the address of the previous record into the current record
                structure to allow for first in last out operation
        */
        tmp -> tac_Previous_add = *tac_results_add;

        /*
                updates pointer to address of current record
        */
        *tac_results_add = tmp;
}
```

```
/*
        test.C
        WRITTEN BY:     MICHAEL WOMAC
        DATE: 16 Dec 1994 - 27 Jan 1995
        Version: 1.0
*/

/*****************************************************************/
/********                    Headers                    ********/
/*****************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include "c:\womac\variable.h"


/*****************************************************************/
/********                   Definitions                 ********/
/*****************************************************************/

#define MaxWords 1          /* defines the max number of words in array */
#define MaxBuffer 7         /* defines the max size of words in array */
#define matrix_size 512     /* defines char matrix size of test_data matrix */

/*****************************************************************/
/********                Global Variables               ********/
/*****************************************************************/

char *J2_column[256];
int J2_data[48][17];


        /**** CONSTANTS ****/
        int Count, J2_Count;
        int chip_number;
        int inp_count;

        /**** COUNTERS ****/

        int row = 0;            /* counts number of rows read out of data file */
        int file_counter = 0; /* counts number of files that are opened */

        /**** FILE POINTERS ****/
        FILE *fp_badchip;
        FILE *fp_bad_data;
        FILE *fp_tac, *fp_amu;

        /**** CHARACTER ARRAYS ****/
        char *File_num[ MaxWords ];
        char filenum[ MaxBuffer ];
        char test_data[ matrix_size ];
        char filename[25];
        char inp_file[25];
        char file2[50];

        /**** STRUCTURE POINTERS ****/
        tac *tac_results_add;
        tac *tmp;
        amu *amu_results_add;
        amu *amu_tmp;

/*****************************************************************/
/********                   Structures                  ********/
/*****************************************************************/
```



```
/*****************************************************************/
/********                   Functions                   ********/
/*****************************************************************/

void convert_J2_to_integer( int );
extern void tac_test( int *, tac ** );
extern void amu_test( int *, amu ** );
extern void Parse_data( char *, int *, char ** );
extern void Parse_J2( char *, int *, char ** );

/*****************************************************************/
/********                     Code                      ********/
/*****************************************************************/


/*****************************************************************/

/*
        Function Name:  main
        Requires:   filename.ext
        Returns:    void
        Purpose:    open & close files that are listed in file: badchip.num.
                    The filenames are then parsed into individual words.
                    File is then opened and data is read out a line at a time

                    and converted to integer text and stored in a matrix call

ed
                    J2_data.  The address of the data matrix is then passed t

o
                    the testing funtions for analysis.

*/
/*****************************************************************/


void main( int argc, char **argv )
{


        /**** CODE ****/
        if( argc != 2 )
        {
                puts("\nUsage: Filename.ext");
                exit(1);
        }

         sprintf( file2, "c:\\womac\\badchip#\\%s", argv[1] );

        printf("loaded file: %s", argv[1] );
        /*
                opens file that contains bad chip numbers
        */
        if(  (fp_badchip = fopen( file2, "r")) != NULL)
        {
                /*
                        get next bad file number
                */
                while ( (fgets(filenum, MaxBuffer, fp_badchip) != NULL) )
                {
                        /*
                                puts chip number in a File_num[0] so it can be op
```

```
ened
                        */
                        Parse_data( filenum, &Count, &(File_num[0]) );

                        chip_number = atoi(File_num[0]);

                        /*
                                stores the file name to a variable called filenam
e
                        */
                        sprintf ( filename, "c:\\chiptst\\c%s.dat", File_num[0]
 ) ;

                        /*
                                opens bad file for testing
                        */
                        if ( ( (fp_bad_data = fopen( filename , "r")) != NULL )
                        {
                                /*
                                        gets a line from the bad data fil
e and stores in the
                                        variable test_data. pointer fp_ba
d_data increments each
                                        pass so to get the next line
                                */
                                while( (fgets( test_data, matrix_size , f
p_bad_data))
                                                != NULL )
                                {
                                        /*
                                                seperates test_data strin
g which is one line out of the
                                                bad data file into 32_col
umn words
                                        */
                                        Parse_32( test_data, &32_Count, &
(32_column[0]) );
                                        convert_32_to_integer( row );
                                        row **;
                                }

                                printf("\nfile count: %5.1d", file_counter);

                                tac_test( &chip_number, &tac_results_add );

                                amu_test( &chip_number, &amu_results_add );

                        }
                        file_counter **;
                        row = 0;
                        fclose( fp_bad_data );
                }
        fclose(fp_badchip);
        }
        else
        {
                printf("\nCan't open file");
                exit(1);
        }


        /*
                saves structures to corresponding files
        */
```



```
        if ( ( fp_tac = fopen( "c:\\womac\\results\\results.tac", "w" )) != NULL
 )
        {
                for( tmp = tac_results_add; tmp != NULL; tmp = tmp->tac_Previous_
add )
                {
                        fprintf( fp_tac,"%d ", tmp->chip );
                        fprintf( fp_tac,"%d, ", tmp->total_chip_time_out_failures
);
                        fprintf( fp_tac,"%d, ", tmp->total_20ns_failures);
                        fprintf( fp_tac,"%d, ", tmp->total_100ns_failures);
                        fprintf( fp_tac,"%d, ", tmp->channel_timeout_failures);
                        fprintf( fp_tac,"%d, ", tmp->channel_20ns_failures);
                        fprintf( fp_tac,"%d\n", tmp->channel_100ns_failures);
/*
                        fprintf( fp_tac,"%d\n ", tmp->tac_bias_failure);
*/
                }
        }
        fclose(fp_tac);


        if ( ( fp_amu = fopen( "c:\\womac\\results\\results.amu", "w" )) != NULL
 )
        {
                for( amu_tmp = amu_results_add; amu_tmp != NULL;
                                amu_tmp = amu_tmp->amu_Previous_add )
                {
                        fprintf( fp_amu,"%d ", amu_tmp->chip );
                        fprintf( fp_amu,"%d, ", amu_tmp->cell0_1_s );
                        fprintf( fp_amu,"%d, ", amu_tmp->cell0_2_s );
                        fprintf( fp_amu,"%d, ", amu_tmp->pipe0_1_s );
                        fprintf( fp_amu,"%d, ", amu_tmp->pipe0_2_s );
                        fprintf( fp_amu,"%ld, ", amu_tmp->badcell1_row );
                        fprintf( fp_amu,"%d, ", amu_tmp->badcell1_col );
                        fprintf( fp_amu,"%d, ", amu_tmp->badcell3_row );
                        fprintf( fp_amu,"%d, ", amu_tmp->badcell3_col );
                        fprintf( fp_amu,"%d, ", amu_tmp->badpipe1_loc );
                        fprintf( fp_amu,"%d, ", amu_tmp->badpipe3_loc );
                        fprintf( fp_amu,"%d, ", amu_tmp->amu_1add_failure );
                        fprintf( fp_amu,"%d, ", amu_tmp->amu_3add_failure );
                        fprintf( fp_amu,"%s, ", amu_tmp->hard_amu_add_failure );
                        fprintf( fp_amu,"%d\n", amu_tmp->single_failure_only );

                }

        }
        fclose(fp_amu);


        printf("\nfile_counter: %d\n", file_counter);
        printf("\nfiles generated by this program:");
        printf("\nc:\\womac\\results\\results.tac");
        printf("\nc:\\womac\\results\\results.amu");


}


/**********************************************************************/

/*
        Function Name: convert_32_to_integer
```

```
        Requires:   current row of type integer
        Returns:    void
        Purpose:    converts J2_column character array into J2_data
                            integer array.
*/

/*****************************************************************************/

void convert_J2_to_integer ( int row )
{
        int column;
        if ( row < 16 )
        {
                for( column = 0; column < 9; column ++ )
                {
                        J2_data[row][column] = atoi( J2_column[ column ] );
                }
        }
        if ( row > 15 )
        {
                for( column = 0; column < 16; column ++ )
                {
                        J2_data[row][column] = atoi( J2_column[ column ] ) ;
                }
        }
}
```

APPENDIX F

PALASM CODE FOR THE MACH 230 CPLD

```
;*************************************************************************
TITLE:          DAQ TST1 INTERFACING CARD
PATTERN:        DAQ_TEST.PDS
REVISION:       1.0
AUTHOR:         MICHAEL WOMAC
COMPANY:        MARTIN MARIETTA
DATE:           13 JUNE 1994

CHIP     DAQ_TEST MACH230

;*********************PIN DECLARATIONS*****************************

;CLOCKS
PIN 65 CLK1
PIN 62 CLK2
PIN 23 CLK3

;INPUTS
PIN 67 WRITE
PIN 55 READ
PIN 80 FSC
PIN 12 TAC_STRT
PIN 81 RESET ; MACH CHIP RESET
PIN 76, 75, 61, 60, 59, 58, 57, 56 CHAN_ENA[0..7]
PIN 82, 79, 78 TST[0..2]
PIN 20, 41, 83, 77 PULSE[0..3]
PIN 54 ADC_RAMP

;NOTE: AMD ADDRESS MUST BE SENT TO CHIP PRIOR TO THE READ PULSE TO THE
;MACH PART (from the computer).  LIKEWISE, THE ADC ADDRESS MUST BE SENT
;PRIOR TO THE ADC_OE

;OUTPUTS
PIN 25 WENA REGISTERED ;
PIN 26, 27, 28, 24 WADD[0..3] REGISTERED ;
;PIN 7 RENA REGISTERED ;
PIN 45 ADC_TRIG REGISTERED ;
PIN 46 AMD_RESET REGISTERED ;
PIN 40, 38, 36, 34, 3, 5, 7, 9 TAC_CHAN[0..7] COMBINATORIAL ;
PIN 66 TAC_START COMBINATORIAL ;


;  NODES FOR THE TAC SECTION

NODE 7  TAC_STRT_BUF REGISTERED ;
NODE 22 DELAY_20 COMBINATORIAL ;
NODE 24 DELAY_40 COMBINATORIAL ;
NODE 26 DELAY_60 COMBINATORIAL ;
NODE 28 DELAY_80 COMBINATORIAL ;
NODE 18 DELAY_100 COMBINATORIAL ;
NODE 20 DELAY_120 COMBINATORIAL ;
NODE 118 DELAY_140 COMBINATORIAL ;
NODE 120 DELAY_160 COMBINATORIAL ;
NODE 128 DELAY_180 COMBINATORIAL ;
NODE 114 DELAY_200 COMBINATORIAL ;
NODE 88, 102, 90, 104, 92, 99, 108, 116, 117, 119, 115 M_DELAY[0..10] REGISTERED
;

; --- NODES FOR THE WRITE SECTION

NODE 106 GO REGISTERED ;
NODE ?   GO2 REGISTERED;
NODE 100 WCNTRCLK REGISTERED ;
NODE 110, 96, 82 WSTART[0..2] REGISTERED ;
NODE 44 STOP REGISTERED ;
```



```
;---- NODES FOR THE READ SECTION

NODE 69, 71, 72, 67, 80 RCNTR[0..4] REGISTERED ;
NODE 84 RGO REGISTERED ;
NODE 86 RCNTRCLK REGISTERED ;
NODE 95, 35, 37 RSTART[0..2] REGISTERED ;
NODE 65 DELAY REGISTERED ;

;#########################################################################
###
;#########################################################################
###


EQUATIONS

;*************************************************************

;*************** EQUATIONS FOR THE TAC SECTION ***************

;*************************************************************


;---------------------------- TAC CLEAR SIGNAL ---------------


TAC_STRT_BUF.CLKF = CLK1
TAC_STRT_BUF.RSTF = RESET

/TAC_STRT_BUF    := /TAC_STRT


;--------- ------------------MULTIPLE  20nS DELAYS-------------

M_DELAY[0..10].CLKF = CLK1
M_DELAY[0..10] .RSTF = RESET

M_DELAY[0]       := /TAC_STRT
M_DELAY[1]       := M_DELAY[0]
M_DELAY[2]       := M_DELAY[1]
M_DELAY[3]       := M_DELAY[2]
M_DELAY[4]       := M_DELAY[3]
M_DELAY[5]       := M_DELAY[4]
M_DELAY[6]       := M_DELAY[5]
M_DELAY[7]       := M_DELAY[6]
M_DELAY[8]       := M_DELAY[7]
M_DELAY[9]       := M_DELAY[8]
M_DELAY[10]   := M_DELAY[9]

;TAC_START.CLKF = CLK1
;TAC_START.RSTF = RESET

TAC_START    = M_DELAY[0]

;DELAY_20.CLKF = CLK1
;DELAY_20.RSTF = RESET

/DELAY_20        =       M_DELAY[1]  * /TST[2] * /TST[1] * TST[0]
                       * /PULSE[3] * /PULSE[2] * /PULSE[1] * /PULSE[0]

;DELAY_40.CLKF = CLK1
;DELAY_40.RSTF = RESET
```

```
/DELAY_40       =       M_DELAY[2] * /TST[2] * TST[1] * /TST[0]
                        * /PULSE[3] * /PULSE[2] * /PULSE[1] * PULSE[0]

;DELAY_60.CLKF = CLK1
;DELAY_60.RSTF = RESET

/DELAY_60       =       M_DELAY[3] * /TST[2] * /TST[1] * /TST[0]
                        * /PULSE[3] * /PULSE[2] * PULSE[1] * /PULSE[0]

;DELAY_80.CLKF = CLK1
;DELAY_80.RSTF = RESET

/DELAY_80       =       M_DELAY[4] * /TST[2] * TST[1] * /TST[0]
                        * /PULSE[3] * /PULSE[2] * PULSE[1] * PULSE[0]

;DELAY_100.CLKF = CLK1
;DELAY_100.RSTF = RESET

/DELAY_100      =       M_DELAY[5] *  /TST[2] * /TST[1] * TST[0]
                        * /PULSE[3] * PULSE[2] * /PULSE[1] * /PULSE[0]

;DELAY_120.CLKF = CLK1
;DELAY_120.RSTF = RESET

/DELAY_120      =       M_DELAY[6] * /TST[2] * TST[1] * /TST[0]
                        * /PULSE[3] * PULSE[2] * /PULSE[1] * PULSE[0]

;DELAY_140.CLKF = CLK1
;DELAY_140.RSTF = RESET

/DELAY_140      =       M_DELAY[7] * /TST[2] * /TST[1] * /TST[0]
                        * /PULSE[3] * PULSE[2] * PULSE[1] * /PULSE[0]

;DELAY_160.CLKF = CLK1
;DELAY_160.RSTF = RESET

/DELAY_160      =       M_DELAY[8] * /TST[2] * TST[1] * /TST[0]
                        * /PULSE[3] * PULSE[2] * PULSE[1] * PULSE[0]

;DELAY_180.CLKF = CLK1
;DELAY_180.RSTF = RESET

/DELAY_180      =       M_DELAY[9] * /TST[2] * /TST[1] * /TST[0]
                        * PULSE[3] * /PULSE[2] * /PULSE[1] * /PULSE[0]

;DELAY_200.CLKF = CLK1
;DELAY_200.RSTF = RESET

/DELAY_200      =       (M_DELAY[10] * /TST[2] * /TST[1] * /TST[0]
                        * PULSE[3] * /PULSE[2] * /PULSE[1] * PULSE[0])
                    +   (M_DELAY[10] * /TST[2] * TST[1] * /TST[0]
                        * PULSE[3] * /PULSE[2] * /PULSE[1] * PULSE[0])

;-------------------------TAC CHANNELS 0 THRU 7 ---------------------

;TAC_CHAN[0..7].CLKF = CLK1
;TAC_CHAN[0..7].RSTF = RESET

TAC_CHAN[0]     =       CHAN_ENA[0] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)
```



```
TAC_CHAN[1]     =       CHAN_ENA[1] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

TAC_CHAN[2]     =       CHAN_ENA[2] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

TAC_CHAN[3]     =       CHAN_ENA[3] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

TAC_CHAN[4]     =       CHAN_ENA[4] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

TAC_CHAN[5]     =       CHAN_ENA[5] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

TAC_CHAN[6]     =       CHAN_ENA[6] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

TAC_CHAN[7]     =       CHAN_ENA[7] * (/DELAY_20 + /DELAY_40 + /DELAY_60 + /DELAY_
80
                                        + /DELAY_100 + /DELAY_120 + /DELAY_140
+ /DELAY_160
                                        + /DELAY_180 + /DELAY_200)

;=========================================================

;=============== EQUATIONS FOR THE WRITE SECTION ===============

;=========================================================


WSTART[0..2].CLKF = CLK2
WSTART[0..2].RSTF = RESET

WSTART[0]  :=  WRITE

WSTART[1]  :=  WSTART[0]

WSTART[2]  :=  WSTART[1]

;---------------------------
go.clkf = clk2
go.rstf = RESET

go              :=  WSTART[1] * /WSTART[2]
                +   go * /stop
```

```
go2.clkf = clk2
go.rstf = reset

go2          :=  WSTART[0] * /WSTART[1]
             #   go2 * /STOP
;------------ counter clock used for timing ---------------

WCNTRCLK.CLKF = CLK2
WCNTRCLK.RSTF = RESET

WCNTRCLK     :=  /WCNTRCLK

;-------     ---  STOP-----------------------------------------

STOP.CLKF = CLK2
STOP.RSTF = RESET

STOP         :=  WADD[0] * WADD[1] * WADD[2] * WADD[3] * wena
             #   WSTART[2] * STOP


;-------------- write enable ------ ----------

WENA.CLKF = CLK3
WENA.RSTF = RESET

WENA     :=  go2 * /WCNTRCLK

;---------------------WRITE ADDRESS----------------

WADD[0..3].CLKF = CLK2
WADD[0..3].RSTF = RESET

WADD[0]    := go * (WADD[0]:+:WCNTRCLK)

WADD[1]    := go * ( /WADD[1] * WADD[0] * WCNTRCLK
                   #  WADD[1] * /(WADD[0] * WCNTRCLK))

WADD[2]    := go * ( /WADD[2] * WADD[1] * WADD[0] * WCNTRCLK
                   #  WADD[2] * /(WADD[1] * WADD[0] * WCNTRCLK) )


WADD[3]    :=  go * ( /WADD[3] * WADD[2] * WADD[1] * WADD[0] * WCNTRCLK
                   #  WADD[3] * /(WADD[2] * WADD[1] * WADD[0] * WCNTRCLK) )


;*********************************************************************

;*************** EQUATIONS FOR THE READ SECTION ***************

;*********************************************************************


;----------------------------

RSTART[0..2].CLKF = CLK1
RSTART[0..2].RSTF = RESET

RSTART[0]  :=  READ

RSTART[1]  :=  RSTART[0]
```



```
RSTART[2]  :=  RSTART[1]

;---------------

Rgo.clkf = clk1
RGO.rstf = RESET

RGO :=  RSTART[2] * FSC
    #   RGO * RSTART[2]

;---------------

AMU_RESET.CLKF = CLK1
AMU_RESET.RSTF = RESET

/AMU_RESET = READ

;---------------

;RENA.CLKF = CLK1
;RENA.RSTF = RESET

;RENA := AMU_RESET

;-------DELAY USED TO STOP AND RESET COUNTER AFTER DESIRED DELAY--------


DELAY.CLKF = CLK1
DELAY.RSTF = RESET

DELAY    := RGO * ( /ADC_TRIG # DELAY)

;------------- counter clock used for timing ----------------

RCNTRCLK.CLKF = CLK1
RCNTRCLK.RSTF = RESET

RCNTRCLK     := /RCNTRCLK

;-------------internal clocks for timing --------------------

RCNTR[0..6].CLKF = CLK1
RCNTR[0..6].RSTF = RESET

RCNTR[0]     := RGO * /DELAY * ( RCNTR[0] :+: RCNTRCLK);

RCNTR[1]     := RGO * /DELAY * ( /RCNTR[1] * RCNTR[0] * RCNTRCLK
                                 #  RCNTR[1] * /( RCNTR[0] * RCNTRCLK) );

RCNTR[2]     := RGO * /DELAY * ( /RCNTR[2] * RCNTR[1] * RCNTR[0] * RCNTRCLK
                                 #  RCNTR[2] * /(RCNTR[1] * RCNTR[0] * RCNTRCLK)
 );

RCNTR[3]     := RGO * /DELAY * ( /RCNTR[3] * RCNTR[2] * RCNTR[1] * RCNTR[0] * RCN
TRCLK
                               #  RCNTR[3] * /(RCNTR[2] * RCNTR[1] * RCNTR[0] * RCN
TRCLK) );

RCNTR[4]     := RGO * /DELAY * ( /RCNTR[4] * RCNTR[3] * RCNTR[2] * RCNTR[1]*RCNTR
[0]*RCNTRCLK
                               #  RCNTR[4] * /(RCNTR[3] * RCNTR[2] * RCNTR[1] * RCN
TR[0] * RCNTRCLK) );

;-------------------------------------------------------------
```

```
ADC_TRIG.CLKF = CLK1
ADC_TRIG.RSTF = RESET

;THIS SHOULD BE ACTIVED AFTER A 500nS (COUNT OF 25 WITH 50MHz CLOCK) DELAY


/ADC_TRIG :=   AGO *  RCNTR[4] * RCNTR[3] * /RCNTR[2] * /RCNTR[1] * RCNTR[0] * FB
C
         *    ADC_RAMP


;-------------------------------------------------------------------------
;#########################################################################
;#########################################################################

SIMULATION

;=========================================================================

;================ SIMULATION FOR THE TAC SECTION ========================

;*************************************************************************


;TRACE_ON TAC_STRT TST[0..2] PULSE[0..3] TAC_START
;         TAC_CHAN[0..7] CHAN_ENA[0..7]
;         RESET CLK1 clk2 WRITE go stop WCNTRCLK wena WADD[0..3]

;TRACE_ON TAC_STRT DELAY_20 DELAY_40 DELAY_60 DELAY_80 DELAY_100
;         DELAY_120 DELAY_140 DELAY_160 DELAY_180 DELAY_200 H_DELAY[10]


SETF RESET TAC_STRT

FOR X := 1 TO 10 DO BEGIN
            SETF CLK1
            SETF /CLK1
END



SETF /RESET /TAC_STRT /TST[2] /TST[1] /TST[0]
SETF CHAN_ENA[0] /CHAN_ENA[1..7]
SETF PULSE[3] /PULSE[2] /PULSE[1] PULSE[0]
                                  ;200 nS

FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END

;-----------------------
SETF TAC_STRT

FOR X := 1 TO 20 DO BEGIN
        SETF CLK1
```



```
        SETF /CLK1
END

SETF /TAC_STRT /TST[2] /TST[1] /TST[0]      ;180 nS
SETF CHAN_ENA[1] /CHAN_ENA[0]
SETF PULSE[3] /PULSE[2] /PULSE[1] /PULSE[0]


FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END

;-----------------------
SETF TAC_STRT
                                  ;THIS SECTION OF THE SIMULATION RE-INITIALIZES TH
                                  ;THE REGISTERS
FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END


SETF  /TAC_STRT /TST[2] TST[1] /TST[0]      ;160 nS
SETF CHAN_ENA[2] /CHAN_ENA[1]
SETF /PULSE[3] PULSE[2] PULSE[1] PULSE[0]

FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END
;-----------------------
SETF TAC_STRT
                                  ;THIS SECTION OF THE SIMULATION RE-INITIALIZES TH
                                  ;THE REGISTERS
FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END

SETF /TAC_STRT /TST[2] /TST[1] /TST[0]      ;140 nS
SETF CHAN_ENA[3] /CHAN_ENA[2]
SETF /PULSE[3] PULSE[2] PULSE[1] /PULSE[0]

FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END
;-----------------------
SETF TAC_STRT
                                  ;THIS SECTION OF THE SIMULATION RE-INITIALIZES TH
                                  ;THE REGISTERS
FOR X := 1 TO 20 DO BEGIN
         SETF CLK1
         SETF /CLK1
END
```

```
SETF /TAC_STRT /TST[2] TST[1] /TST[0]       ;120 nS
SETF CHAN_ENA[4] /CHAN_ENA[3]
SETF /PULSE[3] PULSE[2] /PULSE[1] PULSE[0]


FOR X := 1 TO 20 DO BEGIN
        SETF CLK1
        SETF /CLK1
END

;---- ... ... ... ... ... ...
SETF TAC_STRT
                        ;THIS SECTION OF THE SIMULATION RE-INITIALIZES TH

                        ;THE REGISTERS
FOR X := 1 TO 20 DO BEGIN
        SETF CLK1
        SETF /CLK1
END

SETF /TAC_STRT /TST[2] /TST[1] TST[0]       ;100 nS
SETF CHAN_ENA[5] /CHAN_ENA[4]
SETF /PULSE[3] PULSE[2] /PULSE[1] /PULSE[0]


FOR X := 1 TO 20 DO BEGIN
        SETF CLK1
        SETF /CLK1
END

;-----------------------
SETF TAC_STRT
                        ;THIS SECTION OF THE SIMULATION RE-INITIALIZES TH

                        ;THE REGISTERS
FOR X := 1 TO 20 DO BEGIN
        SETF CLK1
        SETF /CLK1
END

SETF /TAC_STRT /TST[2] TST[1] /TST[0]       ;80 nS
SETF CHAN_ENA[6] /CHAN_ENA[5]
SETF /PULSE[3] /PULSE[2] PULSE[1] PULSE[0]


FOR X :  1 TO 20 DO BEGIN
        SETF CLK1
        SETF /CLK1
END

;-----------------------
SETF TAC_STRT
                        ;THIS SECTION OF THE SIMULATION RE-INITIALIZES TH

                        ;THE REGISTERS
FOR X := 1 TO 20 DO BEGIN
        SETF CLK1
        SETF /CLK1
END

SETF /TAC_STRT /TST[2] /TST[1] /TST[0]      ;60 nS
SETF CHAN_ENA[7] /CHAN_ENA[6]
SETF /PULSE[3] /PULSE[2] PULSE[1] /PULSE[0]
```



```
;  FOR X := 1 TO 20 DO BEGIN
;          SETF CLK1
;          SETF /CLK1
;  END
;
;  TRACE_OFF


;*************************************************************

;************** SIMULATION FOR THE WRITE SECTION **************

;*************************************************************

TRACE_ON  RESET CLK1 clk2 CLK3 WRITE go stop WCNTRCLK wena WADD[0..3]

SETF /CLK2 CLK3 /WRITE RESET
setf reset
setf /reset


FOR X := 1 TO 5 DO BEGIN
        SETF CLK2 /CLK3

        SETF /CLK2 CLK3
END

SETF WRITE

FOR X := 1 TO 37 DO BEGIN
        SETF CLK2 /CLK3

        SETF /CLK2 CLK3
END

SETF /WRITE

FOR X := 1 TO 5 DO BEGIN
        SETF CLK2 /CLK3

        SETF /CLK2 CLK3
END

SETF WRITE

FOR X := 1 TO 60 DO BEGIN
        SETF CLK2 /CLK3

        SETF /CLK2 CLK3
END

TRACE_OFF

;*************************************************************

;************** SIMULATION FOR THE READ SECTION **************

;*************************************************************


;  TRACE_ ON RCNTRCLK RGO RENA READ ARU_RESET DELAY
;             FSC RCN10[0..4] ADC_RAMP ADC_TRIG ARU_RESET
;
;
```

```
    SETF /READ FSC RESET /ADC_RAMP

    FOR X := 1 TO 5 DO BEGIN
            SETF CLK1

            SETF /CLK1
    END
;-------------------------------------------CHECK RESET FUNCTION

SETF /RESET ADC_RAMP

FOR X := 1 TO 10 DO BEGIN
        SETF CLK1

        SETF /CLK1
END

;-----------------------------------------#1 - CHECK FUNTIONALITY OF DEVICE

SETF READ /ADC_RAMP

FOR X := 1 TO 70 DO BEGIN
        SETF CLK1

        SETF /CLK1
END

SETF /FSC

FOR X := 1 TO 70 DO BEGIN
        SETF CLK1

        SETF /CLK1
END

SETF FSC

FOR X := 1 TO 100 DO BEGIN
        SETF CLK1

        SETF /CLK1

END

;-------------CHECK IF READ FUNTIONS CORRECTLY

SETF /READ

FOR X := 1 TO 10 DO BEGIN
        SETF CLK1

        SETF /CLK1
END

;--------CHECK TO SEE IF COUNTER RESETS AND THE RESPONSE TO A RESENT IN THE
;--------MIDDLE OF A SET OF EVENTS
```



```
SETF READ

FOR X := 1 TO 70 DO BEGIN
        SETF CLK1

        SETF /CLK1
END

SETF /FSC

FOR X := 1 TO 20 DO BEGIN
        SETF CLK1

        SETF /CLK1
END

;TRACE_OFF
```

```
;PALASM Design Description

;------------------------------ Declaration Segment ------------
TITLE    ADC conversion control
PATTERN  mikew.pds
REVISION 1
AUTHOR   James Walker
COMPANY  Martin Marietta
DATE     04/17/95

CHIP   chuck  MACH215


;------------------ Power, Ground, & Clocks ---------------

;PIN  1 GND ;
;PIN 12 GND ;
;PIN 23 GND ;
;PIN 34 GND ;
;PIN 22 VCC ;
;PIN 44 VCC ;


PIN 35 CLK40M ;

;-- -----------------INPUTS ------------------------

PIN 10  RESET       ; from SYSTEM global reset
PIN 11  TRIG        ; from SYSTEM trigger from system
PIN 13, FSC         ; from ADC

PIN 29 BUF_IN

;---------- --------- OUTPUTS ---------------- ---------------

PIN 28 BUF_OUT

PIN 25, 26, 19, 17, 15 CNT[0..4] REGISTERED ;

PIN 18      CLK1M           REGISTERED ;

PIN 7       RAMPEN_REG          ; to ADC
PIN 3       ICS_REG             ; to ADC
PIN 24      ADC_AZB             ; to ADC
PIN 5       ADC_AZ          REG ; to ADC
PIN 14      CMP_RST         REG ; to ADC


PIN 4, 6, 8 DELAY[1..3]     REG ; for ADC_AZ
PIN 2       AZ_ARM          REG ; flag
PIN 9       HOLD            REG

PIN 21, 20  TRIG_BUF[0..1)  REG
PIN 16      TAC_CLR         COMBINATORIAL



;------------------------ Boolean Equation Segment ------
EQUATIONS
```



```
BUF_OUT = /BUF_IN



;-------------------------------------------
; Divide by 40 circuit used to produce
; a 1 MHz clock for other MACHs.
; Added by Shane Frank

CNT[0..4].CLKF = CLK40M
CNT[0..4].RSTF = RESET

CNT[0].T    := 1

CNT[1].T    := CNT[0]

CNT[2].T    := CNT[0] * CNT[1] * /CNT[4]

CNT[3].T    := CNT[2] * CNT[0] * CNT[1]

CNT[4].T    := CNT[3] * CNT[0] * CNT[1] * CNT[2]
             + CNT[0] * CNT[1] * CNT[4]


;----------- 1MHz clock output ---------------

;This 1MHz clock is developed here and used in other
;places in this chip.  It is connected internally and
;thus requires the MACH-215 part.  The MACH-215 allows
;derivations of internal clocks with combinatorial logic.

CLK1M.CLKF  = CLK40M
CLK1M.RSTF  = RESET
CLK1M.T     := CNT[0] * CNT[1] * CNT[4]


;-----------------------------------------------------
; Latches the incoming trigger signal as a constant high.
; mode bit that stays high during the conversion process.

HOLD.CLKF = CLK40M
HOLD.RSTF = RESET

HOLD    := TRIG_BUF[0] * TRIG_BUF[1] * /HOLD
         + HOLD * /(/FSC * RAMPEN)

;-----------------------------------------------------

TRIG_BUF[0..1].CLKF = CLK40M
TRIG_BUF[0..1].RSTF = RESET

TRIG_BUF[0] := /TRIG

TRIG_BUF[1] := TRIG_BUF[0]

;-----------------------------------------------------
;changed /TAC_CLR = TRIG_BUF[0] * /TRIG_BUF[1] ;combinatorial

/TAC_CLR = /hold * /cmp_rst  ;combinatorial
```

```
;-------------------------------------------------------------

CMP_RST.RSTF = RESET;
CMP_RST.CLKF = CLK1M;

/CMP_RST    :=  HOLD

;-------  --- -------------------------------------------------
;Delay for setting pulse width for the ADC_AZ

DELAY[1..3].RSTF = RESET
DELAY[1..3].CLKF = CLK1M

DELAY[1]        := ADC_AZ

DELAY[2]        := DELAY[1]

DELAY[3]        := DELAY[2]

;-------------------------------------------------------------
ADC_AZ.RSTF = RESET
ADC_AZ.CLKF = CLK1M;

ADC_AZ      :=  AZ_ARM * /CMP_RST * HOLD

ADC_AZB     = /ADC_AZ               ; COMBINATORIAL

;-------------------------------------------------------------
; az_arm is essentially a flag that only allows the ADC's
; auto zero signal to occur once. AZ_ARM
; is armed-hi disarmed-low

AZ_ARM.RSTF = RESET
AZ_ARM.CLKF = CLK1M;

/AZ_ARM    :=  ADC_AZ  * DELAY[3]
           +   FSC * /AZ_ARM

;-------  ----------------------------------------------------

ICS.RSTF    = RESET
ICB.CLKF    = CLK1M;

/ICS        :=  HOLD * /ADC_AZ * /AZ_ARM * FSC


;-------  ----------------------------------------------------
RAMPEN.RSTF = RESET
RAMPEN.CLKF = CLK1M;

RAMPEN      := /ICS * FSC

;------  ---- ------------------- Simulation Segment ------------
SIMULATION
; simulation is in auxiliary file.
;-------------------------------------------------------------
```

VITA

Michael Womac was born in Athens, Tennessee on January 17, 1965. He attended schools in the McMinn County public school system where he graduated from McMinn County High School in 1982. He joined the Navy in 1984 and was honorably discharged in 1989. He began work in Vienna, Virginia and worked there until 1991. He then began pursuing his electrical engineering degree in the spring of 1991 and finished his bachelors in that discipline in the spring of 1995. The masters degree in the same area was received in 1996.