To the Graduate Council:

I am submitting herewith a thesis written by Bassel Atallah entitled "Automatic Test Facility Universal Remote". I have examined the final copy of this thesis for form and content and recommend that it will be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Daniel B. Koch

Daniel B. Koch, Major Professor

We have read this thesis
and recommend its acceptance:

Robert B. Barkhouse

T. Vaughn Blalock

Accepted for the council:

C. W. Minkel

Associate Vice Chancellor
and Dean of the Graduate School

AUTOMATIC TEST FACILITY UNIVERSAL REMOTE

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

Bassel Ghazi Atallah

August 1996

DEDICATION

I dedicate this work to my loving family. First and foremost, to my mother Leila Atallah and my father Ghazi Atallah, for their love and support. Secondly to my sister Aurore and my brother Wael, for their encouragement and belief in me.

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Daniel B. Koch, my professor and advisor for his great help, support and advice. Throughout this research, he was always by my side giving me positive feedback. I would like to thank Dr. Bodenheimer and Dr. Blalock for their time and effort in serving on my thesis committee. Finally, I would like to thank the CISP and ATF group members for their contribution and assistance throughout this project.

ABSTRACT

With the rapid advance of electronic technology, particularly in the area of consumer electronics, the switch from manual testing methods to automated testing methods is required. The transition outlined in this thesis occurred in the television industry at a national consumer electronics company. A new Automatic Test Facility (ATF) was designed; the only part missing was a universal remote control device. The purpose of the Automatic Testing Facility Universal Remote (ATFUR) is to provide infrared control for all TV functions, brands and models subject to testing.

This project consists of both hardware and software designs. Hardware includes an IR transmitter controlled by a Motorola microcontroller chip, housed in a separate enclosure and connected to a computer. Software consists of LabVIEW programs with a graphical control panel on the computer screen similar to the panel of a real remote control. Two versions of the software are available: manual and automated. Manual operation allows the user to operate the system with a mouse button, while automated operation runs by itself without human assistance.

The use of the ATFUR proved to be more efficient and less time consuming than using real remote controls. This device is user friendly and can be easily maintained.

TABLE OF CONTENTS

CHAPTER PAGE

1. INTRODUCTION....1

2. BACKGROUND....8

2.1 Location....8

2.2 Hardware Architecture....13

2.3 Incremental Visual Programming....17

3. ATFUR HARDWARE DEVELOPMENT....26

3.1 GPIB Interface....26

3.2 RS-232 Interface....37

3.3 Microcontroller....45

3.4 Components....65

3.5 IR Transmitter....94

4. LabVIEW SOFTWARE DEVELOPMENT....102

4.1 IR Format....103

4.2 Time Table....109

4.3 Code Table....115

4.4 TV Header....122

4.5 Manual Driver....129

4.6 Automated Driver....142

5. SYSTEM PERFORMANCE....151
5.1 Operation....151
5.2 Problems....156
6. DISCUSSION....161
6.1 Conclusion....161
6.2 Further Research....166
LIST OF REFERENCES....168
APPENDICES....172
Appendix A....173
Appendix B....224
Appendix C....232
Appendix D....259
Appendix E....263
Appendix F....265
VITA....268

LIST OF ILLUSTRATIONS

FIGURE		PAGE
Figure 1.1	Design Illustration	3
Figure 1.2	Input/Output Icon of ATFURMD	5
Figure 1.3	Input/Output Icon of ATFURAD	7
Figure 2.1	Layout of the Three Rooms	12
Figure 2.2	Instrument Connection	14
Figure 2.3	ATFURs Connection	15
Figure 2.4	IVP Flowchart	19
Figure 2.5	BB, SB, CB Example	22
Figure 2.6	Verification Checklist	23
Figure 3.1	GPIB Bus Handshaking Cycle	30
Figure 3.2	Front Panel of GPIB Write VI	33
Figure 3.3	Block Diagram of GPIB Write VI	34
Figure 3.4	GPIB-SCSI-A Box	36
Figure 3.5	GPIB-232CT-A Box	43
Figure 3.6	Configuration Setting	44
Figure 3.7	Parts Numbering	47
Figure 3.8	Block Diagram of the E2	49
Figure 3.9	Pin Assignment of the E2	50
Figure 3.10	Assembly Language Development Process	52
Figure 3.11	Subroutine Execution	59
Figure 3.12	LM7805CT Voltage Regulator	67

Figure 3.13 MAX232 Chip....68
Figure 3.14 DB9 Connector....70
Figure 3.15 MC34064....71
Figure 3.16 PNP Transistor....73
Figure 3.17 Characteristic Curves of NPN....76
Figure 3.18 NPN Transistor....77
Figure 3.19 IR LED Properties....80
Figure 3.20 Mechanical Data of IR LED....81
Figure 3.21 Block Diagram of the MC145157....84
Figure 3.22 MC145157 Pin Assignment....85
Figure 3.23 J-K Flip-Flop....88
Figure 3.24 Mechanical Dimensions of the ADAPT-11™....90
Figure 3.25 Prototyping Card....93
Figure 3.26 Hardware Box....93
Figure 3.27 IR Transmitter....95
Figure 3.28 SPICE Program....96
Figure 3.29 Voltages Graph....97
Figure 4.1 Logical "0" and "1" Structure....104
Figure 4.2 Example of RECS580 Code....105
Figure 4.3 NEC Logical "1"....107
Figure 4.4 NEC Logical "0"....107
Figure 4.5 Volume Up Waveform....108
Figure 4.6 Binary Codes for a Specific TV....110
Figure 4.7 Customed Commands for a Specific Model....111
Figure 4.8 Time Table GV....114

Figure 4.9 Code Table GV....121
Figure 4.10 TV Header GV....125
Figure 4.11 TVs Under Test....126
Figure 4.12 Universal Remote Control....132
Figure 4.13 BB of the ATFURMD....133
Figure 4.14 A Programmed GV....137
Figure 4.15 Error I/O Clusters....139
Figure 4.16 CB of the ATFURMD....143
Figure 4.17 BB of the ATFURAD....145
Figure 4.18 CB of the ATFURAD....147
Figure 5.1 ATFUR Project Schematic....152
Figure 5.2 IR Detector....157
Figure 6.1 Times Comparison....164

LIST OF TABLES

TABLE		Page
Table 2.1	RF Amplitude Levels	11
Table 3.1	IEEE 488 Pin Description	28
Table 3.2	GPIB DIP Switches Configuration	38
Table 3.3	Factory Default Setting for SW2	38
Table 3.4	LED Description	41
Table 3.5	Pin Assignment	42
Table 3.6	RS232 DIP Switches Configuration	45
Table 3.7	DB9 Pin Connection	70
Table 3.8	PNP Parameters	74
Table 3.9	Ratings at 25°C Temperature	81
Table 3.10	Header H1 Pinout	91
Table 3.11	Theoretical vs. Measured values	101
Table 4.1	Availability of Commands for Models	117
Table 4.2	Availability of Models for Brands	123
Table 4.3	LabVIEW Instrument Driver Templates	130
Table 4.4	ATFUR Commands	134
Table 4.5	ATFURMD Errors	141
Table 4.6	Outlets Distribution	149
Table 5.1	LED Wavelength vs. Voltage	159

LIST OF ACRONYMS AND ABBREVIATIONS

A	Amperes
AD	Automated Driver
ANSI	American National Standards Institute
ASCII	American Standard Code Information Interchange
ATE	Automated Test Equipment
ATF	Automatic Test Facility
ATFUR	Automatic Test Facility Remote Control
ATFURAD	Automatic Test Facility Remote Control Automated Driver
ATFURMD	Automatic Test Facility Remote Control Manual Driver
ATN	Attention
BB	Black Box
BDL	Box Definition Language
BJT	Bipolar Junction Transistor
CA	Competitive Analysis
CB	Clear Box
CISP(Group)	Communications, Information, and Signal Processing Group
CMOS	Complementary Metal Oxide Semiconductors
COP	Computer Operating Properly
CPU	Central Processing Unit
CTS	Clear To Send
DAV	Data Valid

DCD	Data Carrier Detect
DIP	Dual Inline Package
DCE	Data Communication Equipment
DMA	Direct Memory Access
DSR	Data Set Ready
DTE	Data Terminal Equipment
DTR	Data Terminal Ready
EEPROM	Electrically Erasable Programmable Read Only Memory
EIA	Electronic Industries Association
EOI	End Or Identify
EPROM	Erasable Programmable read Only memory
G	Graphical Language (National Instruments)
GPIB	General Purpose Interface Bus
GV	Global Variable
HCMOS	High-density Complementary Metal Oxide Semiconductor
HLL	High Level Language
HP	Hewlett Packard
HP-IB	Hewlett Packard Interface Bus
IEEE	Institute of Electrical and Electronics Engineers
IFC	Interface Clear
IML	Interface Management Line
IR	Infrared
IVP	Incremental Visual Processing
I/O	Input/Output

LAN	Local Area Network
LED	Light Emitting Diode
LVI	Low Voltage Inhibit
MC	Microcontroller
MCU	Microcontroller Unit
MD	Manual Driver
NDAC	Not Data Accepted
NI	National Instruments
NRFD	Not Ready For Data
OTP	One Time Programmable
PC	Personal Computer
PIP	Picture In Picture
PLCC	Plastic Leaded Chip Carrier
PROM	Programmable Read Only Memory
RAM	Random Access Memory
REN	Remote Enable
RF	Radio Frequency
RI	Ring Indicate
ROM	Read Only Memory
RTS	Ready To Send
SB	State Box
SCI	Serial Communication Interface
SCP	Software Cleanroom Process
SCSI	Small Computer System Interface
SG	Signal Ground

SP	Sensor Photodiode
SRQ	Service Request
TTL	Transistor Transistor Logic
TV	Television
UR	Universal Remote
UUT	Unit Under Test
VAC	Volts Alternating Current
VDC	Volts Direct Current
VI	Virtual Instrument

DISCLAIMER

The author of this thesis is not responsible in whole or in part for the misuse of the ATFUR devices by other individuals. These devices were designed for the ATF facility, and will be used by authorized personnel only. The author does not hold any liability for any damage this device can cause at any time or any place. If someone wishes to build this device from scratch according to the method presented in this project, he/she will do it at his/her own risk. All literature contained herein is provided for informational purposes only. The author does not make any warranty, or assume legal liability or responsibility for the functionality or misuse of the ATFUR. The author gives permission for the ATF facility to make as many units as needed to implement the whole project. Since the project was done for educational purposes only, this thesis is not a commercial product, and no financial profit can be made from it. Finally, the views and opinions of the author expressed herein do not necessarily state or reflect those of the CISP or ATF group members.

CHAPTER 1

INTRODUCTION

Due to the high level of competition in the television market, each company puts much time and effort into promoting its products. In order to supply the public with quality, working models, each company goes through a certain amount of testing before manufacturing the unit and making it available commercially. Because manual testing is inefficient and time consuming, manufacturers are adopting automated testing method in hopes of improving quality, and thereby increasing profit. Automation allows testing partially or totally without human supervision or control. A study done of a microprocessor-based testing system in 1989 showed that up to 200 TV sets could be tested in one day during manufacturing [1]. Moreover, automated testing requires less people to conduct a test. Only one operator needs to monitor the test, which means less human error and more effectiveness.

The Communications, Information, and Signal Processing (CISP) Group, at the University of Tennessee, Knoxville, is working on an Automatic Test Facility (ATF) project in which televisions are controlled and automatically tested. During a test, it is cumbersome to hold multiple remote controls in one hand to test different television sets. Therefore a computer controlled universal remote control, that contains all the codes of every available TV model, is needed. This research attempts to create such a device.

Early remote controls used ultrasound or radio frequency analog transmission and could perform only limited functions. Today, a remote control is standard with even the lowest cost basic electronic equipment. Remote controls use infrared (IR) light for digital data transmission. The same remote control standard will be used in this project.

The purpose of the Automatic Test Facility Universal Remote (ATFUR) is to provide infrared control for all television functions, brands, and models subject to testing. It will be connected to an RS-232/GPIB interface, and will be controlled by a Sun SPARCstation 10 workstation. An illustration of the design is found in Figure 1.1. Chapter 2 will discuss the connections between the instruments and the devices and describe the environment in which the tests will be conducted. The ATFUR project will consist of two major parts, hardware and software.

Hardware issues will be covered in more detail in Chapter 3. The unit consists of an infrared transmitter controlled by a Motorola microcontroller (MC68HC11). This chip is an advanced 8-bit microcontroller unit (MCU) in high density complementary metal oxide semiconductor (CMOS) technology. Major peripheral functions are provided on the chip [2]. It has 12 kbytes ROM, 512 bytes EEPROM, 512 bytes of RAM, SPI, SCI, A/D converter and 38 I/O pins. The timer system allows 3 or 4 input captures, 4 or 5 output compares, and real

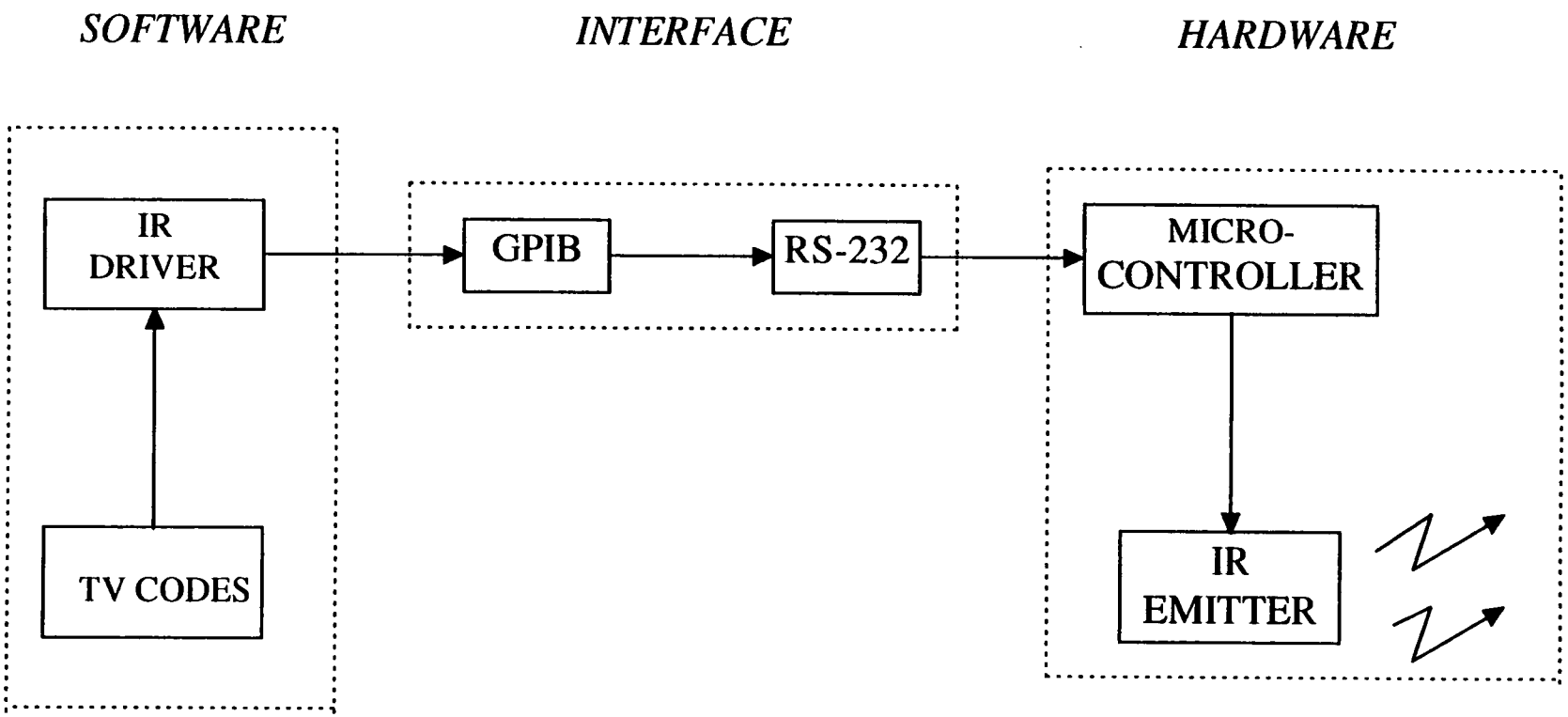


Figure 1.1 Design Illustration

time interrupt. The emulation of the chip will be done via the ADAPT-11 Module. The user can efficiently debug and evaluate the 68HC11 E series. Next, a brief description of each component used in this project will be presented. The rest of this chapter will discuss the source code written to program the MC68HC11 chip. Description of the GPIB and RS-232C interfaces used by the ATFUR devices are also included. Finally, an explanation of the design of the output stage transmitter will be presented.

Software issues will be covered in Chapter 4. In this project the Incremental Visual Programming (IVP) method will be used to develop the software [3]. Two versions of the software driver will be available, a manual (MD) and an automated (AD) version. With the manual version of the driver, the operator can use the computer mouse to click on any button on the screen to simulate an IR controller. The button will then send the appropriate command to the intended television via the IR transmitter. This driver will be beneficial for the "picture clarity test", for example, where the operator sends commands to televisions from his computer instead of figuring out which remote to use. The LabVIEW input/output icon of the manual version is shown in Figure 1.2.

The automated version of the driver will be connected to other LabVIEW drivers and tests. No user will be needed to operate it. This driver will be beneficial to the "RF sensitivity test" for example, where

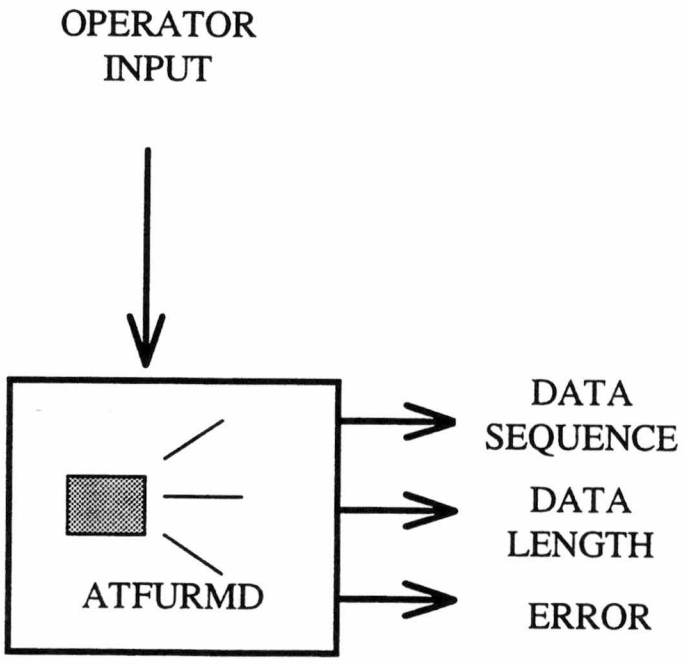


Figure 1.2 Input/Output Icon of ATFURMD

the test driver will send a command to the ATFUR such as "SONY, model SONY1, channel up". The input/output icon of the automated driver is shown in Figure 1.3.

The codes of all available televisions will be tabulated in Global Variables (GVs) which are built-in LabVIEW objects. They allow easy access to a given set of values throughout the LabVIEW application [4]. They can also be accessed by any other driver. For this reason, they are created only once in the design. In the event that new codes are generated for new televisions, they can be entered manually in the GV. This way, the microcontroller will not need to be reprogrammed.

Chapter 5 will cover the experimental and physical operation of the system, and the verification of its performance. It will also address the problems that may be encountered while using the ATFUR, and gives appropriate solutions. Conclusions and further development of the ATFUR units will be discussed in Chapter 6.

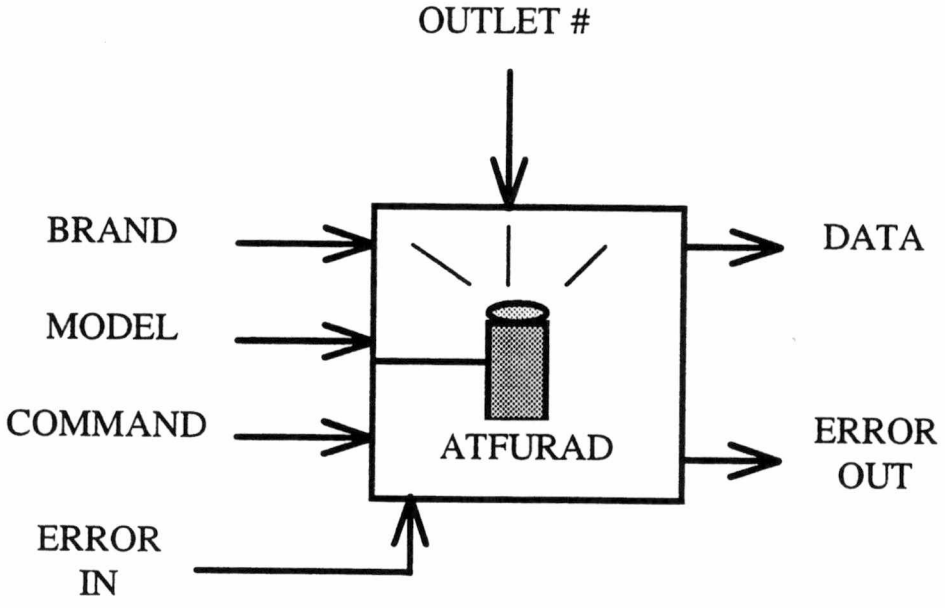


Figure 1.3 Input/Output Icon of ATFURAD

CHAPTER 2

BACKGROUND

This chapter will discuss in detail the location where the ATFUR units will be installed. They will be mounted in three different testing rooms at a national consumer electronics company. The hardware architecture of the whole ATF system will be explained, especially the parts vital for the ATFUR functionality and connectivity. In addition, the software design methodology adopted by the group members and known as IVP will be exposed in detail.

2.1 LOCATION

The ATFUR unit must be practical, reliable and inexpensive. The ideal high performance design should also produce a device that is easy to install, use, and maintain [5]. Performance of the ATFUR will be defined by its high output and fast speed IR transmission. It will be transmitting IR through light emitting diodes (LED) to be compatible with existing television sets. The ATFUR device will be used mainly in three different rooms which are described below.

Even though IR signals do not interfere with Radio Frequency (RF) signals, or propagate through walls, they still have limitations. These limitations are caused mainly by interference from multi-path

channels, ambient lights, and the limited transient response of the IR devices themselves. IR signals travel in a straight path and can be blocked easily by existing objects.

Almost all ATF tests will be conducted in three rooms. Different test devices will be used simultaneously; some of them will create signal interference such as RF signals. These signals can confuse the television receiver which will probably miss the signal sent. This kind of interference must be taken into serious consideration.

The first room is large enough to accommodate a lot of units under test (UUT). It must have a good lighting system to make observations as accurate as possible. This room will be used mainly for demonstrations, and comparisons of the different characteristics and performance levels of a variety of television models. For example, all the televisions will be tuned to the same channel. Participants in the study walk in, watch and examine all TVs. Then they will fill out a questionnaire rating picture clarity and so forth. Also, technicians will take measurements on specific television features, or test the performance of certain models. The ATFUR will be mounted on the ceiling, and will be radiating in an omni-directional pattern. Because some televisions have the IR receiver pushed or hidden inside the TV bezel, making it harder for the IR rays to hit them, the room will be flooded with IR signals to make sure that all televisions will receive the command and respond.

The second room is a special test area where technicians perform internal television measurements. This room contains racks of equipment that are used for measuring, monitoring and diagnosing various TV models. These racks consist of VHF and UHF modulators, CD player, dynamic ghost simulator, random noise generator, several transmitters, video generator, stereo and audio generator, signal generator, variable attenuators, voltage controller, single Phase Power analyzer and other signals coming from another room. In this room, few units under test can be tested at a time. Therefore, it needs a medium level of IR radiation. The ATFUR will be mounted on the wall above the window, and will be beaming straight to the television being tested.

The third room is an RF-free area. It is a specially designed receiver test area. This room is totally isolated from the outside in order to prevent RF interference. All televisions under testing are shielded from all external signal emissions. It has an X-terminal next to the door, which is a client to the Sun workstation, connected to the network via ethernet cable, and permitting remote operation from the demo and screen room. A mobile cart connected to the Sun GPIB bus is used for conducting tests in this room. This room can contain few unitsunder test. All tests dealing with RF-level control are conducted in it, to permit accurate low level RF measurements. An example is the sensitivity test, which measures the television's capability to overcome external reception interference [6]. Various levels of RF signal tests

ranging from 10μV to 1V will be conducted in this room. These levels are described in Table 2.1 [7]. The ATFUR will be installed next to the X-terminal, and will be sending directly to televisions being tested. Figure 2.1 shows the layout of the three rooms combined together.

Table 2.1 RF Amplitude Levels

CHARACTERISTIC	RF LEVEL
Sensitive	10 - 20 μV
Useable	50μV - 100 mV
Standard Cable	1 - 3 mV
Overload	100 mV - 1V

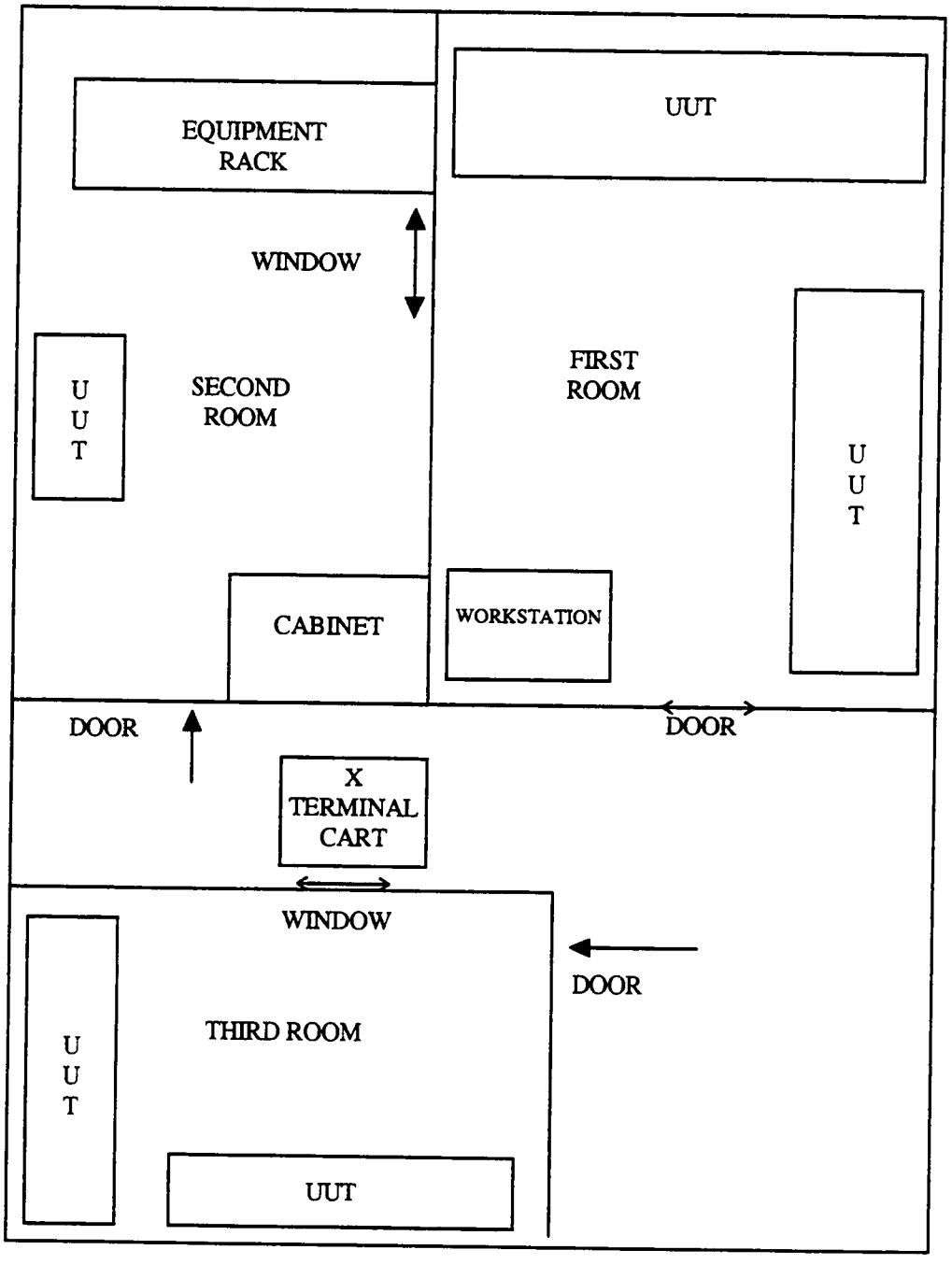


Figure 2.1 Layout of the Three Rooms

2.2 HARDWARE ARCHITECTURE

The hardware architecture of the system will consist mainly of a Sun SPARCstation 10 workstation, an X-terminal, GPIB-SCSI-A buses, and RS-232C interface boxes. A description of these items will be presented in Chapter 3. The Sun workstation is managed by the UNIX operating system; it is also the hub of a Local Area Network (LAN). The X-terminal is a server for the Sun workstation; LabVIEW can be run from it remotely. Team members can use it to develop drivers while competitive analysis (CA) engineers are using the Sun to conduct tests. The ATFUR will be connected to the RS-232 box that is connected to the GPIB bus controlled by the Sun workstation. This connection is illustrated in Figure 2.2.

Three similar ATFUR devices will be installed in each room according to Figure 2.3. More instruments and devices will be connected to this setup, but they are omitted in Figure 2.3 to make it easier to see the ATFUR connections. Two GPIB buses, three RS-232 boxes and one GPIB bus extender will be available to handshake with the remote controls. Bus 1 is connected to the GPIB box extender in order to communicate with the screen room. Bus 0 is communicating directly with the two other rooms since the Sun workstation is inside one of them. Installation of the equipment will be done only once

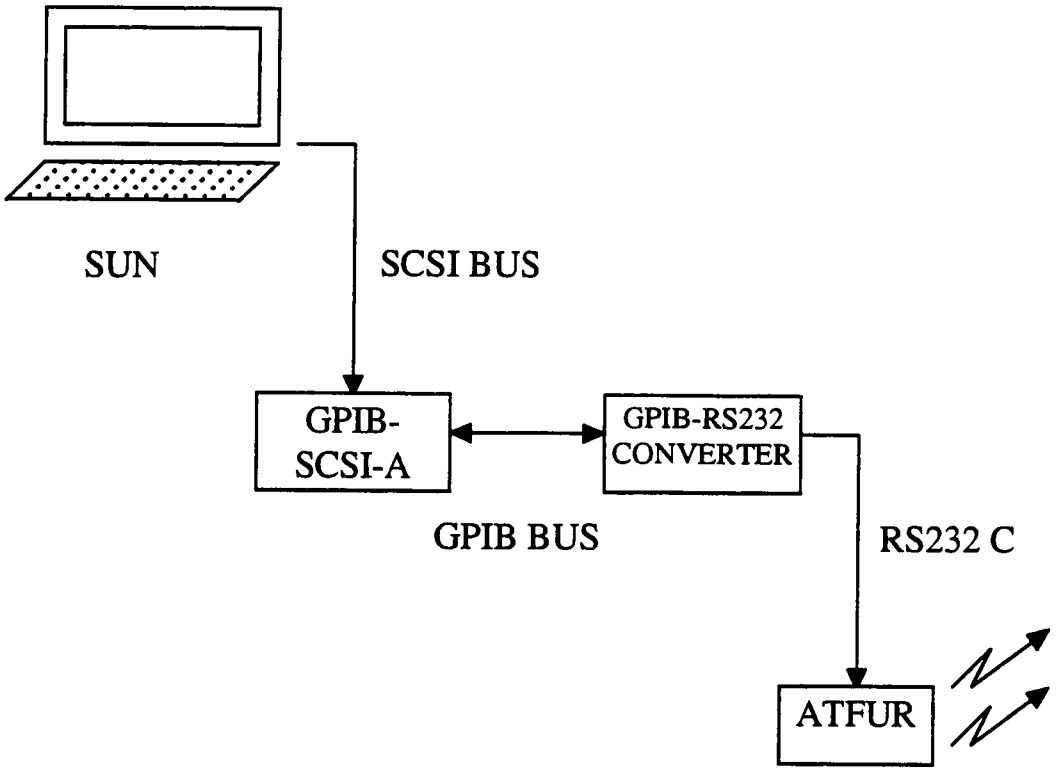


Figure 2.2 Instruments Connection

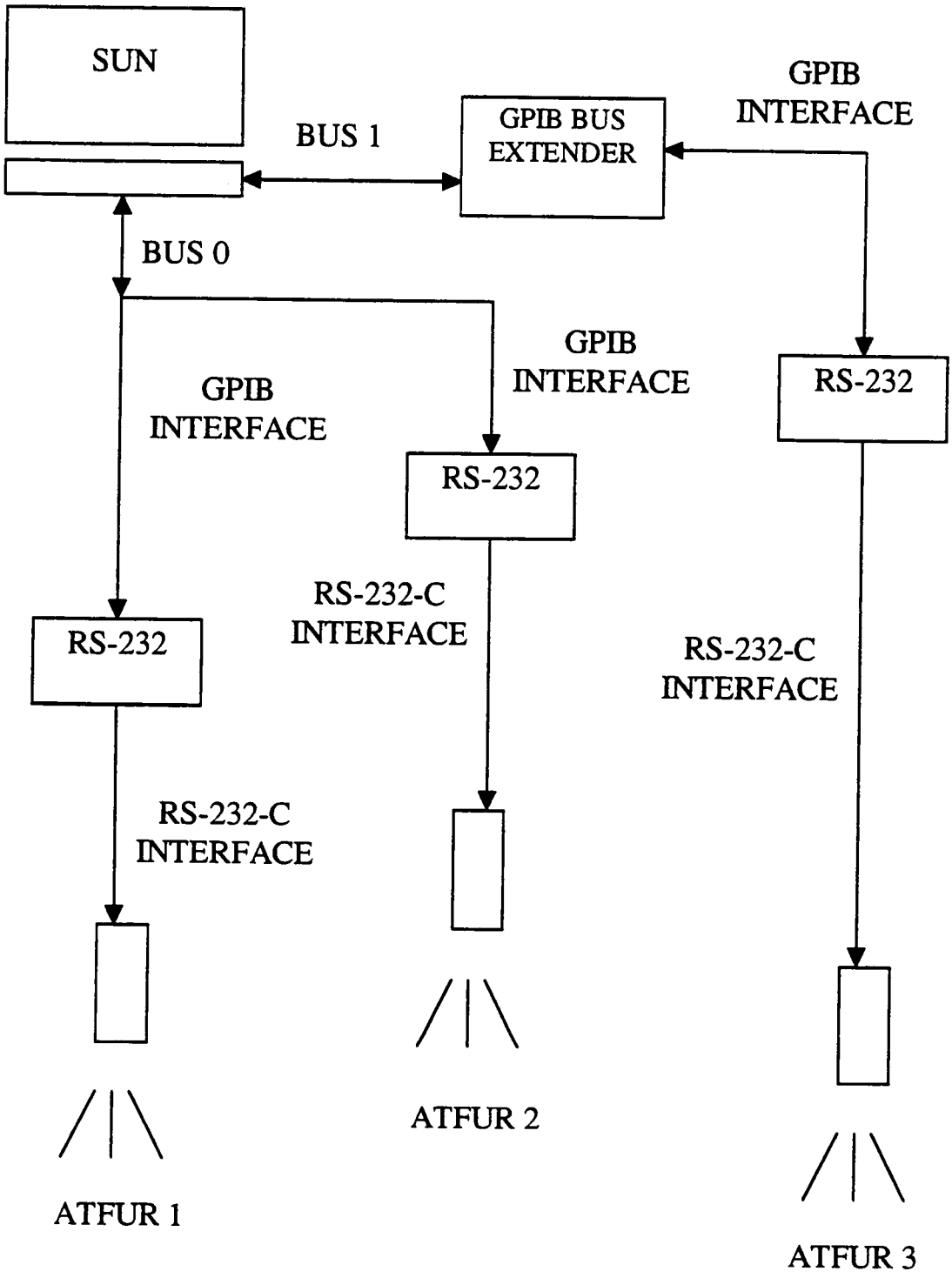


Figure 2.3 ATFURs Connection

unless the whole rack needs to be changed. Most changes will be in the TV devices being tested. The ATFUR, controlled from the Sun console using a mouse must be easy to use. The system architecture of the ATFUR setup does not need to be maintained because the only change will be in its hardware boxes. More about this will be discussed in Section 5.2. The ATFUR will consist of programs (drivers) that are written in LabVIEW, a graphical programming language.

2.3 INCREMENTAL VISUAL PROGRAMMING

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a powerful and flexible instrumentation and analysis software system that can be used on Sun workstations as well as on other platforms. It consists of a graphical programming language called "G". LabVIEW programs, called Virtual Instruments (VIs) are created using a block diagram approach. The software consisting of VIs is compiled into machine code [8].

A VI has three main parts: the front panel, the block diagram, and the icon/connector pane. The front panel consists of inputs and outputs (controls and indicators). The block diagram is the source code, where the program is written with a graphical interface. The icon/connector pane turns the VI into an object that can be used as a subroutine called a subVI in the block diagram of other VIs [8]. The ATFUR drivers will be LabVIEW VIs that can be used or called by other VIs. LabVIEW supports IEEE 488 (GPIB) which sends data from the VIs to the controlled devices.

In order to create these VIs with the LabVIEW environment the Incremental Visual Programming (IVP) method will be adopted. The CISP group members are working on an ATF testing project for a national consumer company. When the project is done, it will

implement tens of tests and will contain hundreds of VIs. To accomplish this task in an organized way, the Software Cleanroom Process (SCP) has been used. The code development in the SCP uses the concept of "box structures" in conjunction with the Box Description Language (BDL) to define elements and data codes. BDL was invented with textual programming languages such as (FORTRAN and Basic) in mind, but since the group is working with a graphical language, a new method was created. The IVP methodology keeps the methodical top-down design approach defined by BDL but in a manner more suitable for LabVIEW. Being a member of the CISP group, the IVP methodology has been used to develop project drivers.

IVP is a methodology that requires a three step process in the development of LabVIEW VIs. The first step is the Black Box (BB) consisting of controls and indicators. The second step is the State Box (SB), an explanation of the program execution. The last step is the Clear Box (CB), the final version of the driver. The IVP flow chart is shown in Figure 2.4. The IVP methodology has several advantages, one being the modularity of code. Each VI can be developed independently of any other VI. Another advantage is the ease with which each step can be understood. The designer can show progress to the customer before all VIs are completed. The customer in turn, can give early feedback to the designer. This way, writing costly, erroneous code can be avoided. This method is a team-based approach for the development of error free, quality VIs. Several group meetings were

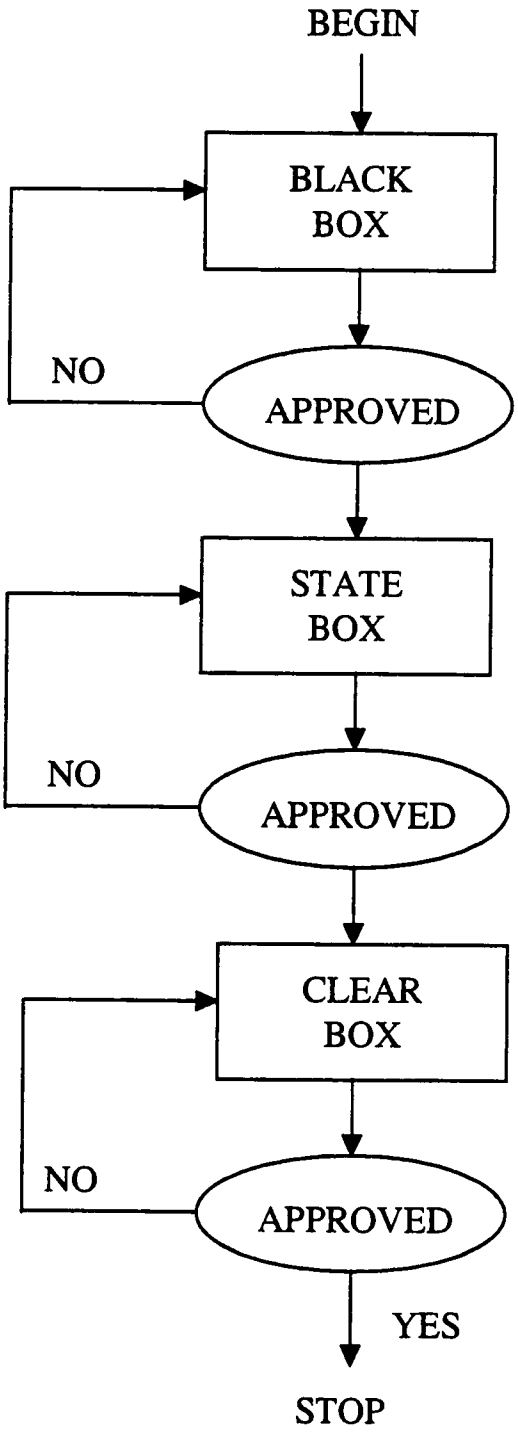


Figure 2.4 IVP Flowchart

held to come up with common conventions to make the code development as consistent as possible.

The BB VI is the first step in the code development. It describes the specifications of the design in terms of the input/output concept. It is made up of three sections: stimulus, response and procedure. Stimuli include controls and buttons. Responses are reactions caused by the software. Indicators and LEDs are examples of responses. The BB will be the only panel that the operator will see when conducting a test. According to the IVP conventions, the background of the front panel is pastel yellow while the block diagram is gray. The panel has a one-line title at the top in the "Helvetica" font. The "Get Info" block contains descriptions of each VI. The controls are on the left, top side, while indicators are on the right, top side.

After the designer explains the BB to the customer, who then approves it, the SB is the next step. The SB is created by copying the BB. This VI provides explanations of the program execution. It consists of all the structures (e.g., For Loop, Case Structure, Sequence) that are needed. A comprehensive description of the functionality of each is included. Items are not wired at this stage. Instead, free labels are used to guide the programmer at each point. The SB is only a description of the code. The background color of the front panel is olive green.

The CB is the final stage of the design. It is created by copying the SB. At this point, all structures and elements are connected (wired), and the driver is ready to be used. All the free labels are replaced by subVIs or actual functions of LabVIEW. For more complex VIs, some labels will be kept to describe different functions of the VI execution. The background color is aqua. The labels are bold and transparent in color. When the three step VI is finished, it is verified before actually being used. Each VI is evaluated for efficiency in execution and functionality. Finally, the VI is delivered to the customer after being verified. A simple VI that adds two numbers will be an example to illustrate the three steps of the IVP method. Note that the BB will be the same for all three steps. Those steps are shown in Figure 2.5.

After these steps are accomplished, the VI under development should meet all the requirements of a verification sheet, and should be tested in front of the group members for functionality before being certified to be used by the customer. Figure 2.6 illustrates the verification sheet adopted by the group members. The certification process is the real testing of the drivers in front of the technicians and engineers that will use them for real testing. All controls are activated to check for accuracy in responses. Each VI will be presented and explained in details. Finally a test approval form containing the names of the people who witnessed the certification process and of the engineers who approved it will be signed.

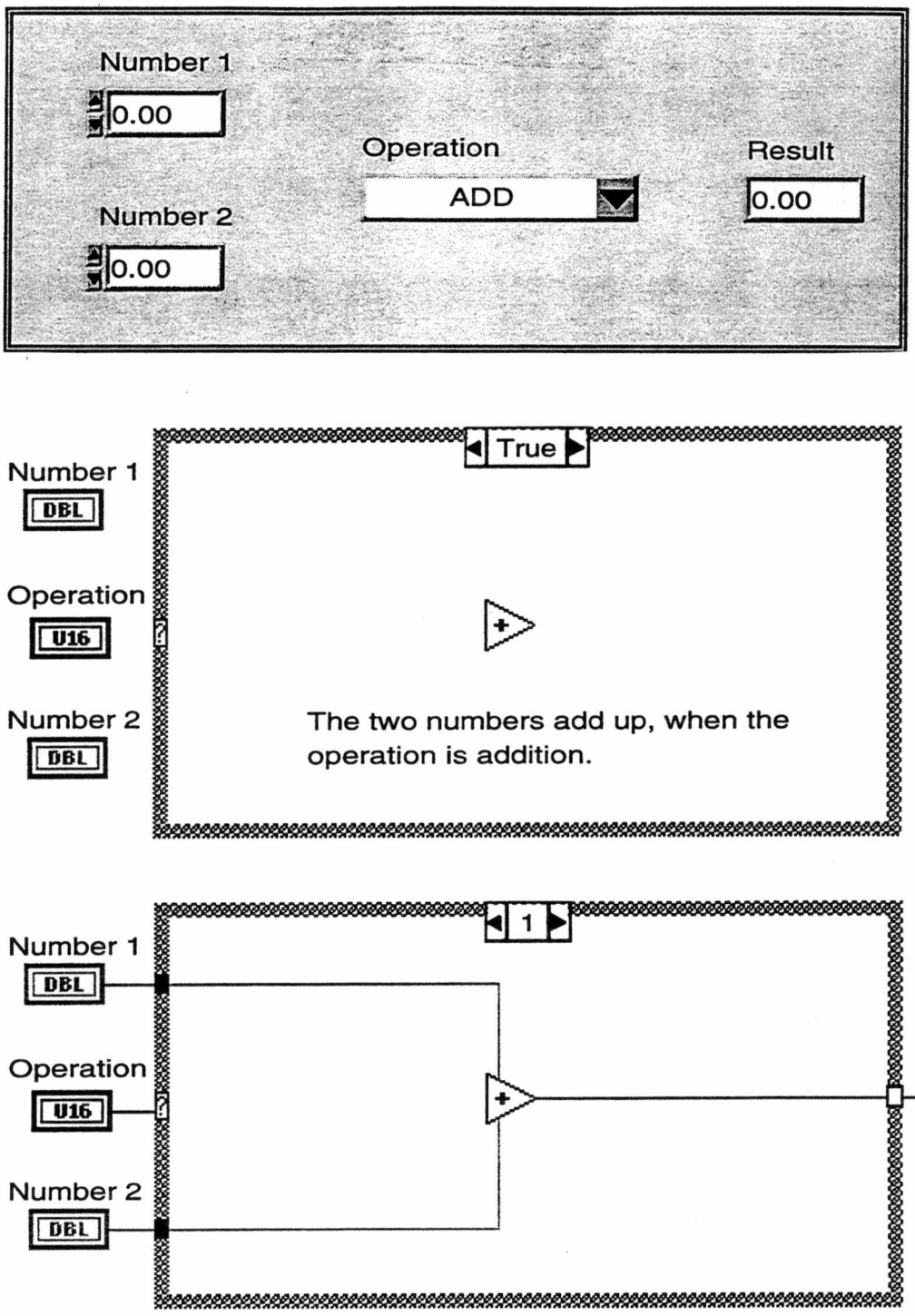


Figure 2.5 BB, SB, CB Example

ATF PROJECT
Verification Checklist
29 September 1995

Designer's Name: _____ VI Name: _____

No.	Item	Comments
1.	COLORS	
	a. BB: Pastel Yellow (R=255, G=225, B=204)	
	b. SB: Olive Green (R=205, G=255, B=102)	
	c. CB, etc.: Aqua (R=127, G=191, B=191)	
	d. i. ALL Block Diagrams=GRAY(229,229,229)	
	ii. ALL TEXT=BLACK	
2.	SPACE (BB, SB, CB)	
	a. Vertical Alignment	
	b. Horizontal Alignment	
	c. Plenty of Gray space on Block Diagram	
3.	DESIGN (CB)	
	a. Three steps completed	
	b. Is the Code accurate	
	c. Is the Code precise	
	d. Is the Code fully documented	
	e. Can the Code be easily maintained	
	f. GET INFO: Overview, Instructions, I/O, Author's name, Modifications, History	
	g. HELP Boxes should be descriptive	
	h. NO type conflict; i.e. no gray dots	
4.	MISCELLANEOUS	
	a. Wiring cleanliness	
	b. Descriptive Icons	

ADDITIONAL COMMENTS

_____

_____

_____

_____

_____

_____

Verifier #1_____
Verifier #2_____

Figure 2.6 Verification Checklist

At each increment of the design development, peer reviews took place. The objectives, characteristics and benefits of peer review are summarized below. The objectives of the peer review are:

- To find defects at the earliest possible point in the process.
- To ensure that the customer and the developers technically agree on the work assigned.
- To verify that the work meets predefined specifications and criteria.
- To ensure that work is ready to proceed to the next stage of development.

The characteristics of the peer review are :

- The peer review is a formal, structured process with verification checklists.
- They focus on identifying, discussing and solving the problems.

Finally, the benefits of peer review are:

- Reduce the number of errors.
- Prevent errors ahead of time by analyzing the requirements.
- Time improvement where the final code is properly working.
- Peer reviews help to motivate better work, designers may be motivated to work more carefully.
- Education; it is an effective means to educate team members and share techniques and ideas.

This chapter was an introduction to the environment in which the ATFUR units will be used. The location was chosen by the customer. The connections of the ATF setup was done by the company technicians. The IVP methodology was created by the CISP team members.

CHAPTER 3

ATFUR HARDWARE DEVELOPMENT

This chapter will discuss the ATFUR hardware development. It will cover in detail the GPIB and RS-232C interfaces used with the boxes that implement them. A description of the Motorola microcontroller chip is covered in Section 3.3. All parts and components used in this project such as LEDs, transistors, reset chip, etc. are listed under components. Each component is presented in an informative way. Finally, the design and operation of the transmitter output stage is presented in the last section.

3.1 GPIB INTERFACE

In 1965, Hewlett-Packard (HP) designed the Hewlett-Packard Interface Bus (HP-IB) to connect their line of programmable instruments to their dedicated controllers. Because of its high transfer rate, this interface bus quickly gained popularity [9]. It was later accepted as Institute of Electrical and Electronics Engineers (IEEE) Standard 488-1975. The main features of IEEE 488 can be summarized as follows:

- Controls up to 14 instruments.
- Transfer rates greater than 1 Mbyte/s.
- Worldwide and International Standard.

- 8-bit parallel protocol, ASCII command set.
- Hardware takes care of timing and handshaking.
- Round, flexible cable - 2m per device up to 20m.
- Designed for remote control of programmable instruments.
- Controls thousands of instrument from hundreds of suppliers.

IEEE 488 instruments have switches that set the address of the instrument and connectors that interface with the bus. Some instruments have a secondary address which gives access to different parts of the instrument [10]. By having two addresses, the instrument allows multiple addressing to perform different tasks.

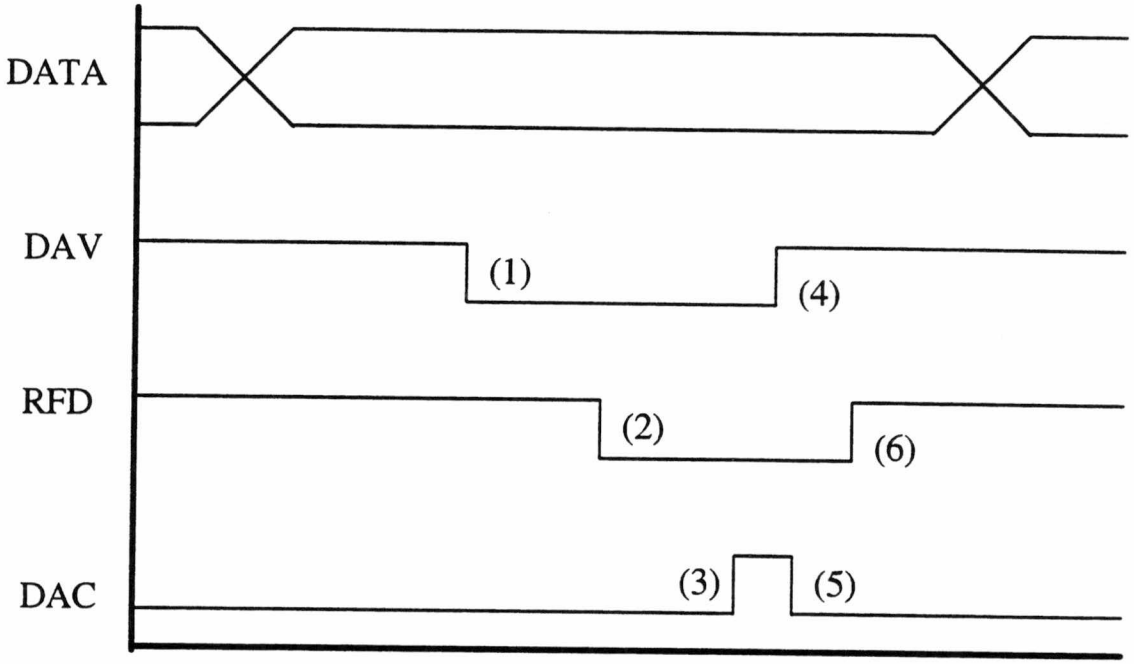
In order to implement this standard, 16 active signal lines and 8 ground returns are needed. Table 3.1 describes the functionality of these lines. Basically these pins are divided into four groups: 8 data lines, 5 interface management lines, 3 handshake lines and 8 ground returns. The eight data lines (1,2,3,4,13,14,15,16) carry both data and command messages. Most of data and commands are 7-bit ASCII or ISO code sets. The handshake lines (6,7,8) asynchronously control the transfer of message bytes among devices. This process is called a three-wire interlocked handshake and it guarantees that message bytes on the data lines are sent and received without transmission errors [11]. The NRFD line indicates when a device is ready or not ready to receive a message byte. The DAV line tells when the signal on data lines are valid and can be accepted safely by the devices. These two lines are

Table 3.1 IEEE 488 Pin Description

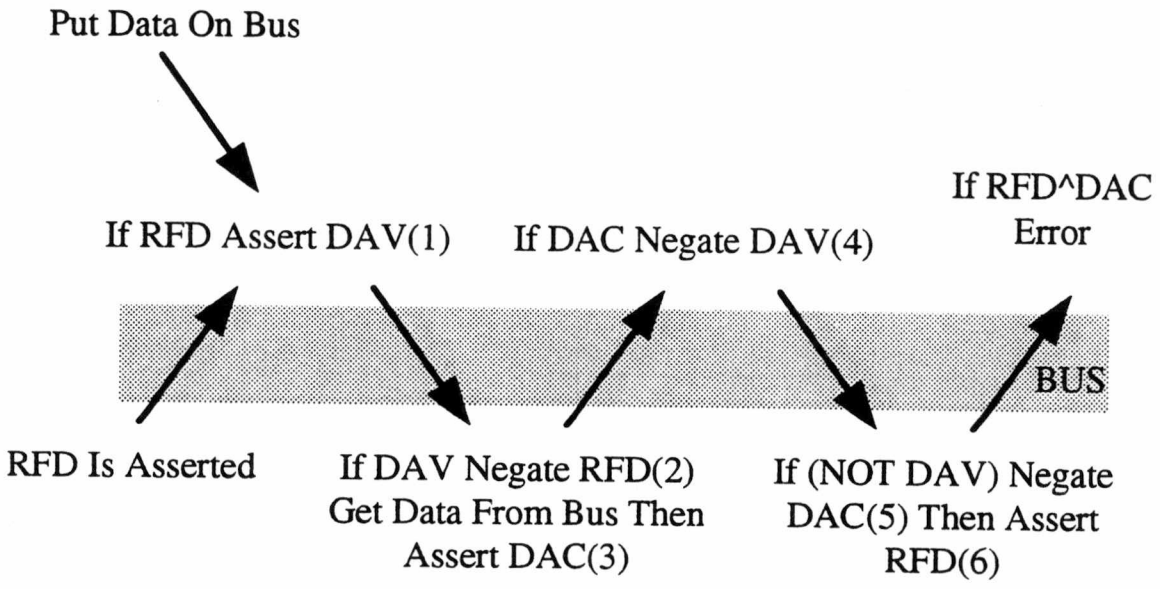
1	Data Bus
2	Data Bus
3	Data Bus
4	Data Bus
5	End Or Identify (EOI)
6	Data Valid (DAV)
7	Not Ready For Data (NRFD)
8	Not Data Accepted (NDAC)
9	Interface Clear (IFC)
10	Service Request (SRQ)
11	Attention (ATN)
12	Shield
13	Data Bus
14	Data Bus
15	Data Bus
16	Data Bus
17	Remote Enable (REN)
18	Ground
19	Ground
20	Ground
21	Ground
22	Ground
23	Ground
24	Logic Ground

used by all devices when receiving commands and by listeners when receiving data messages. The 5 Interface Management Lines (IML) (5,9,10,11,17) are used to manage the flow of information across the interface. The ATN line is driven high to let the data lines send commands, and is driven low to allow a talker to send data messages. The IFC line is used to initialize the bus and to allow the system controller to become the controller in charge. The REN line is used to place devices in remote or local program mode. The SRQ line is used by any device to request service from the controller. The EOI line is used by talkers to mark the end of a message string. It is also used by the controller to tell the devices to identify their response in a parallel poll.

The handshake cycle is like a clock cycle [9]. Each time a word is to be moved, the bus goes through a handshake cycle to move the word, as shown in Figure 3.1. The cycle involves negative logic DAV, sent by the transmitter of the data, and positive logic RFD and DAC, sent by the receiver of the data. If the receiver is able to take data, it asserts a high RFD. When the transmitter wants to send a data word, it first puts the word on the bus, and then begins the handshake cycle. It checks for the RFD signal. If it is asserted at the transmitter, the transmitter asserts a low DAV to indicate that the data is available. This is step 1 in Figure 3.1a and Figure 3.1b. When the receiver sees DAV asserted, it negates RFD in step 2 because it is no longer ready



a. Timing Diagram



b. Handshaking Sequence

Figure 3.1 GPIB Bus Handshaking Cycle

for data. When the data is picked up from the interface by a processor, the receiver asserts a high DAC to indicate that data has been accepted. This corresponds to step 3 in Figure 3.1. When the transmitter sees DAC asserted, it negates DAV in step 4 because it will soon stop sending data on the data bus. When the receiver sees DAV negated, it negates DAC in step 5. When it is ready to accept new data, it asserts a high RFD in step 6 to begin a new handshake cycle.

National Instruments (NI) expanded the use of the IEEE 488 bus to computers manufactured by companies other than Hewlett-Packard. Hence, the name General Purpose Interface Bus (GPIB) is now more widely used than HP-IB. Therefore the name GPIB refers to a link or interface system through which interconnected electronic devices communicate. NI also took care of the software part. It provided LabVIEW with built-in GPIB drivers. These drivers can be used in any application to communicate with other devices. The LabVIEW GPIB drivers communicate with the IEEE 488 instruments using NI GPIB products which are capable of transferring data at up to 1.5 MB/s [10]. LabVIEW currently has more than 450 ready-to-use instrument drivers for controlling the most popular GPIB instruments. These drivers are available at no charge when buying a software or product license. These drivers are ideal for controlling or just communicating with almost all instruments. They eliminate low-level programming, and provide high-level interfaces that can be used in any application. The common program used in this application is the "GPIB

Write VI". This VI is a virtual instrument used as a subVI for both drivers. More discussions about VIs will be available in chapter 4.

Most GPIB applications involve only writing and reading ASCII strings to and from an instrument in a two-step process: write a query to an instrument and read the instrument's reply. The GPIB Write VI will write to a device connected to a GPIB bus. The GPIB address of the device should first be selected. Then data strings should be connected to this VI. Finally the VI should be run and the data will be transferred automatically. Every device must have a unique GPIB address between 0 and 30. Address 0 is usually assigned to the GPIB controller. The instruments on the GPIB bus can use addresses 1 through 30. Then this VI will write data strings to the device at the specified GPIB address. Figures 3.2 and 3.3 show the front panel and the block diagram of this VI respectively.

A Sun SPARCstation 10 was used in this application and does not have a GPIB plug-in card; therefore LabVIEW was used to control GPIB instruments through the Small Computer System Interface (SCSI) port by using the NI GPIB-SCSI-A controller which converts the SCSI protocol to GPIB. This controller works with SunOS 4.1 or higher operating systems. The GPIB-SCSI-A is a high performance interface product that transparently handles data transfers between the SCSI bus and the GPIB bus [12].

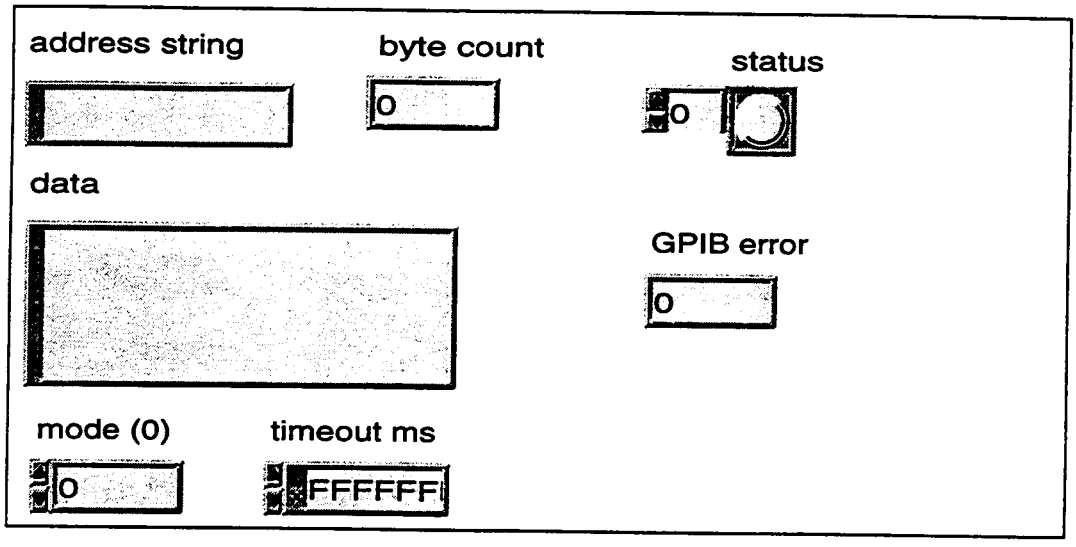


Figure 3.2 Front Panel of GPIB Write VI

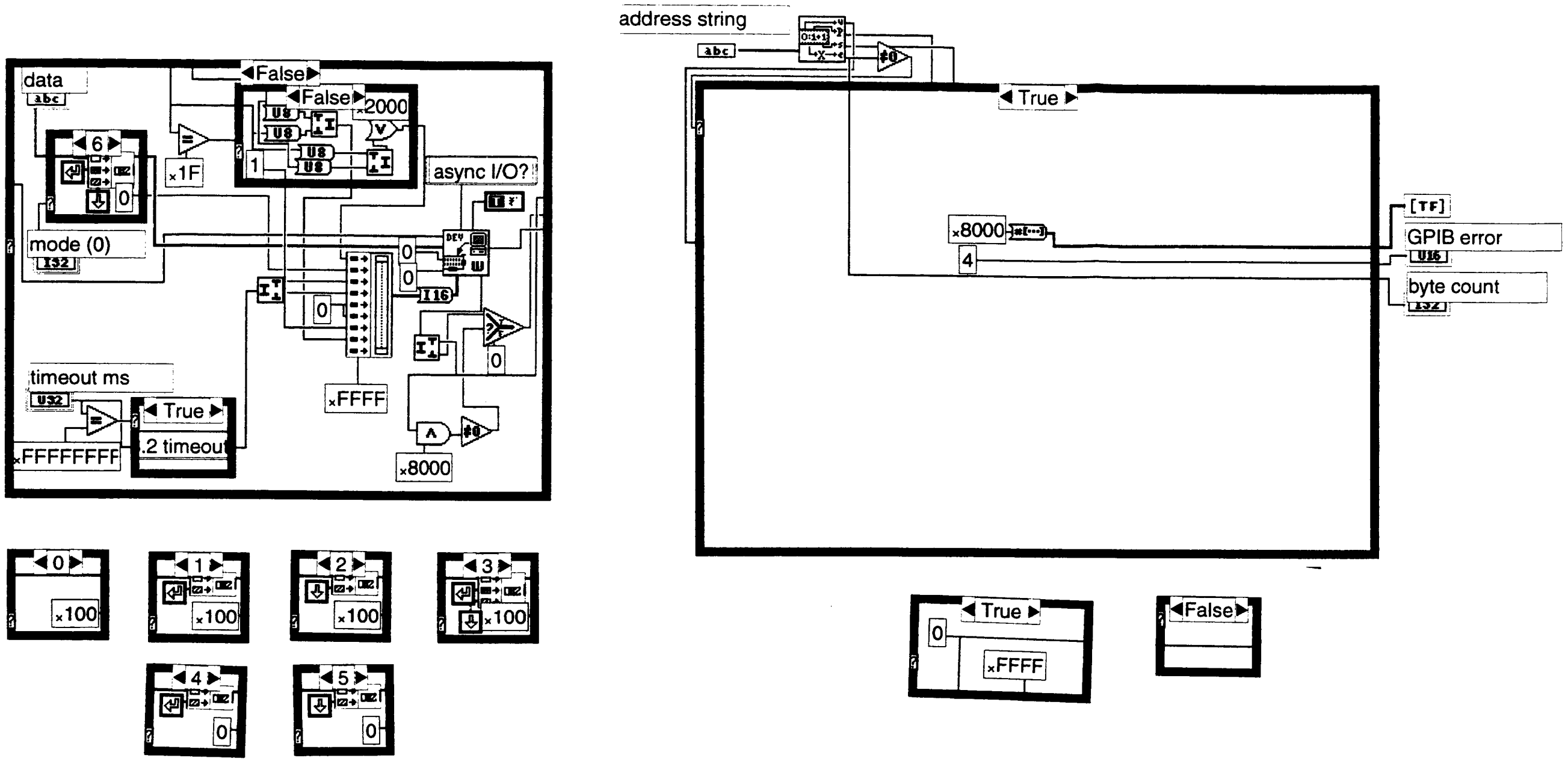


Figure 3.3 Block Diagram of GPIB Write VI

The GPIB-SCSI-A is an 8 bit microcomputer that operates as a full-function IEEE 488 controller. It can turn any computer with a SCSI port into a GPIB Talker, Listener, and Controller or can make any device on the SCSI bus look like a GPIB device. The role of this box can be compared to the role of a switching center of a city telephone system. The switching center (controller) monitors the communications network (GPIB). When the center notices that a party (device) wants to make a call (send data message), it connects the caller (talker) to the receiver (listener). The GPIB-SCSI-A has all the software and logic required to implement the physical and electrical characteristics of the ANSI/IEEE Standard 488. It is able to interpret and execute commands that can be sent to it over a SCSI port and perform all necessary GPIB to SCSI protocol conversions. Figure 3.4 shows the front and back panels of this controller.

The ATF project uses two GPIB buses. Both of them originate at the SCSI port of the Sun using two GPIB-SCSI-A boxes. The first bus known as "Bus 0" connects almost all the system's equipment. The second bus known as "Bus 1" connects a cart that can be used in all three rooms for remote testing. One of the ATFURs will be connected to Bus 0 with the GPIB address 23, while the other two will be connected to Bus 1 with the GPIB addresses 27 and 29 respectively. Each GPIB-SCSI-A box has two switches on the back SW1 and SW2. The dual inline package (DIP) switch at SW1 is used to configure the

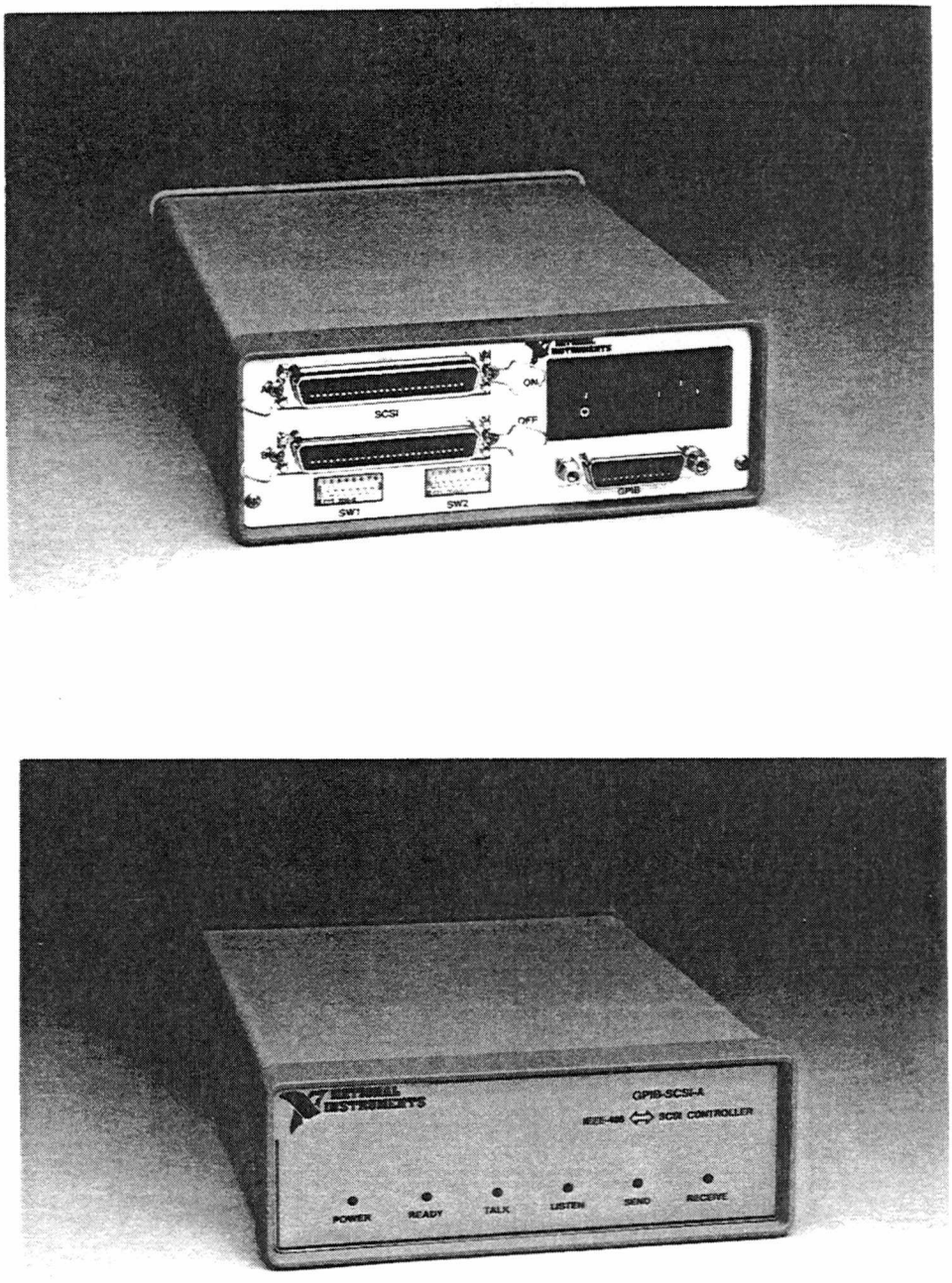


Figure 3.4 GPIB-SCSI-A Box

power on the primary GPIB address and the SCSI ID of the box. The first 3 switches are used to indicate the SCSI ID, the next 5 switches are used to set the GPIB primary address. The DIP switch at SW2 is used to configure the mode of operation of these boxes. these switch setting do not need to be changed when used with a Sun SPARCstation, because the box comes working with its factory default setting. Table 3.2 and 3.3 show the position of the switches while using the GPIB-SCSI-A boxes to control the ATFUR.

3.2 RS-232C INTERFACE

In the early days of telecommunication, optical signals, such as light, were the only methods of transmitting information across long distances. With increasing requirements on speed and growing amounts of information, more practical methods were developed. One milestone was the first wire-bound transmission using the famous Morse alphabet. Technology kept improving and fast interfaces were badly needed. The RS-232C bus was introduced in 1962 to connect computers to peripherals. The real name of this standard is an interface between Data Terminal Equipment (DTE) and Data Communication Equipment (DCE) employing serial binary data exchange [14]. It was adopted by the Electronic Industries Association (EIA) in 1969. This interface specifies how to get data from a personal computer (PC) communications adapter to the modem in a standard

Table 3.2 GPIB DIP Switches Configuration

BUS	SCSI	AD.	1	2	3	4	5	6	7	8
0	0	23	OFF	OFF	OFF	ON	OFF	ON	ON	ON
1	1	27	OFF	OFF	ON	ON	ON	OFF	ON	ON
1	1	29	OFF	OFF	ON	ON	ON	ON	OFF	ON

Table 3.3 Factory Default Setting for SW2

SWITCH	POSITION	INDICATION
8	OFF	Operating in SCSI mode
7	ON	GPIB-SCSI-A will neither notice nor report SCSI parity errors.
6	ON	GPIB-SCSI-A will buffer data during data transfer commands.
5	OFF	GPIB-SCSI-A will complete all data requests to the count specified.
4	ON	Double buffering is enable
1-3	OFF	Reserved and should remain OFF.

way and vice versa. It includes specifications for cabling, connector pinouts, signal levels, and timing.

RS-232 is a standard electrical interface for serial data communications. Serial means that each byte or character of data sent or received is sent one bit at a time. Each bit is either ON or OFF or MARK and SPACE. RS-232 comes in three different types (A, B and C), each one defining a different voltage range for the ON/OFF levels. In this project, the RS-232C interface was used.

The RS-232C standard is a very common standard, supported by all computer systems and most serial instruments. It defines a MARK bit as a voltage between -3V and -12V and a SPACE bit as a voltage between +3V and +12V. It uses a 25-pin D-style connector for most equipment or a 9-pin version on some systems. The standard defines functions for all lines even though just a few of them are actually used, such as: transmit data, receive data and ground. The rest of the lines are used for handshaking, such as Clear To Send (CTS) and Ready To Send (RTS). The advantages of this interface are simplicity and low cost. The main limitations are that RS-232C is electrically single-ended which can cause poor noise rejection. It also limits the maximum distance of connections to 100 meters, though this is frequently violated in practice [14]. When setting up RS-232C style serial communications systems some common problems are encountered:

- Swapped transmit and receive lines.

- Wrong speed, parity, or stop bits setting.
- Multiple applications trying to access the serial port.
- Failure to properly connect the hardware handshaking lines.

Since the ATF project uses the GPIB interface and some of the instruments and units under test need an RS-232C interface, the NI GPIB-232CT-A boxes are used. The GPIB-232CT-A is a high performance GPIB to serial interface transformer. It transforms a GPIB talker/listener/controller to an RS-232C port. It has all the software and logic required to implement the physical and electrical specifications of the IEEE 488 and the RS-232C standard. It is capable of interpreting and executing high level commands as well as performing GPIB to RS-232C protocol conversion. It has the NAT 4882 controller chip that implements all IEEE 488 functionality [13]. Table 3.4 shows the description of the LEDs on its front panel.

The standard 9-pin shielded D-subminiature male connector with a DTE interface configuration is connected to the ATFUR as well as other devices. Table 3.5 shows the pin assignments. Figure 3.5 shows the GPIB-232CT-A box.

Table 3.4 LED Description

LED	INDICATION
POWER	Indicates that power to the unit has been applied and the ON/OFF switch is in the ON position.
READY	Indicates that the power-on self test has passed successfully and the unit is ready to operate.
TALK	Indicates that the box is addressed as a GPIB talker.
LISTEN	Indicates that the box is addressed as a GPIB listener.
SRQ	Indicates that the GPIB signal line SRQ is asserted.
ATN	Indicates that the GPIB signal line ATN is asserted.

Table 3.5 Pin Assignment

PIN #	NAME	DESCRIPTION
1	-	No Connection
2	Receive Data (RxD)	Carries data from device to box.
3	Transmit Data (TxD)	Carries data from box to device.
4	Data Terminal Ready (DTR)	Powered ON and ready to use.
5	Ground	Reference point for all interface voltages.
6	-	No Connection
7	Request to Send (RTS)	When asserted box is ready to accept data. Not asserted means buffer is full.
8	Clear to Send (CTS)	When asserted device can receive data. Not asserted means data transmission is disabled.
9	-	No Connection



Figure 3.5 GPIB-232CT-A Box

When addressing the ATFUR devices, the GPIB-232CT-A should be configured as follows: the transfer rate should be 9600 baud, operates in G mode, sends 8 bits/character, should have no parity, and 1 stop bit/character. Figure 3.6 shows the position of the switches on the box to meet those specifications.

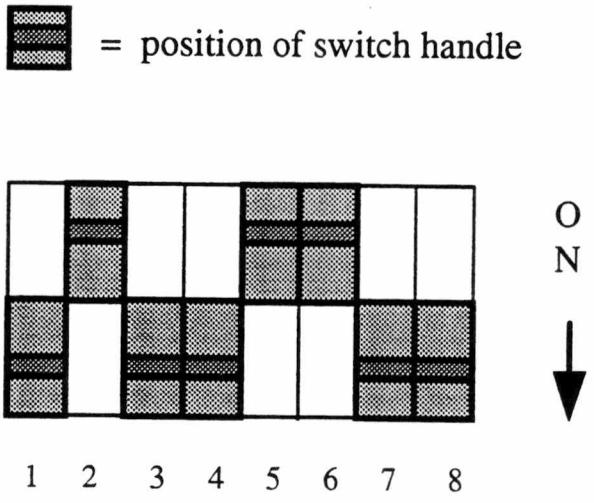


Figure 3.6 Configuration Setting

When the GPIB-232CT-A box is used in G mode, it uses 2 GPIB addresses. The first address, which is set by switches 1 through 5, is the primary GPIB address of the GPIB-232CT-A box. The second GPIB address, which is equal to primary address plus one, is the address of the serial device connected to the GPIB-232CT-A box. In this application, the primary GPIB addresses are set to: 23, 27 and 29 on each of the interface boxes. The secondary addresses are automatically 24, 28 and 30 respectively. Table 3.6 shows the position

of the dual inline package (DIP) of the switches to set the primary addresses.

Table 3.6 RS-232 DIP Switches Configuration

1	2	3	4	5	ADDRESS
ON	ON	ON	OFF	ON	23
ON	ON	OFF	ON	ON	27
ON	OFF	ON	ON	ON	29

3.3 MICROCONTROLLER

Microcontrollers are digital computers particularly designed to supervise, manage, monitor and control various processes in industry, business, defense, aerospace, and other areas of application [16]. With the advent of VLSI technology, microcontrollers are becoming essentially single chips microcomputers. As the technology advances, microcontroller chips attain higher density because of the fact that over a million transistors can be placed on a single chip. It is a one-chip solution [16]. A microcontroller chip contains a central processing unit (CPU), cache, main memory, input/output (I/O) interfaces, direct memory access (DMA) controller, interrupt handler, timers and other

subsystems necessary to implement an efficient microcontroller.

The Motorola MC68HC11 is the most capable of the Motorola 8-bit microcontroller units (MCU) in that it incorporates all of the capabilities of other microcontrollers. The general characteristics of the MC68HC11 are summarized below:

1. High-density CMOS technology (HCMOS).
2. Single voltage power supply operation.
3. Fully static operation (no minimum clock rate).
4. Watchdog timer (COP).
5. Eight bit arithmetic.
6. Sixteen bit timer/event time capture.
7. Eight bit pulse accumulator.
8. Synchronous and asynchronous hardware communication.
9. Low power operating modes under software control.
10. Memory mapped I/O.
11. Orthogonal addressing.
12. Real time interrupt.

Since the MC6811 is a CMOS device, therefore it is fully static. It can operate down to zero clock frequency (DC). In this mode, power is consumed if the clock is running even if the CPU is not executing any instructions due to the charging and discharging of capacitors through small resistance. In such a mode, the power consumption is reduced to

a minimum for critical low power applications, and this is needed for the ATFUR. Also being CMOS, the MC6811 has much higher immunity to noise, power fluctuations or spikes [9].

The MC6811 family of MCUs is composed of several members. Figure 3.7 explains how the product part numbers are constructed.

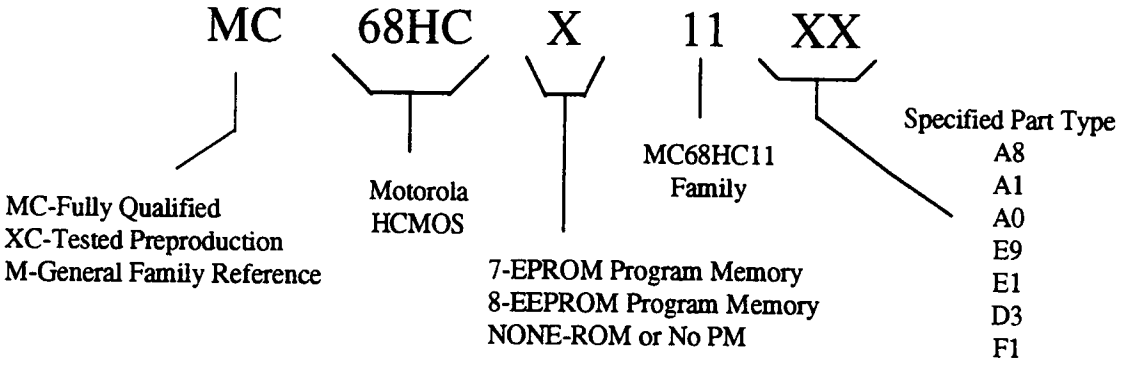


Figure 3.7 Parts Numbering

The main difference among these MCUs are the type and quantity of memory. In this project the MC68HC11E2 chip was chosen. It is an 8-bit microcontroller that has the following features:

- M68HC11 CPU.
- Power saving STOP and WAIT modes.
- Low voltage device (2.7-5.5 VDC).
- 256 Bytes of On chip RAM, data retained during standby.
- 2048 Bytes of On chip EEPROM with block protect for security.

- Asynchronous Nonreturn to Zero (NRZ) Serial Communication Interface (SCI).
- 52 pin Plastic Leaded Chip Carrier (PLCC).

Figure 3.8 shows the block diagram and Figure 3.9 shows the pin assignments of the E2 chip.

V_{DD} is the power supply that ranges from 2.7 to 5.5 Volts. V_{SS} is the ground. The XTAL and EXTAL lines provide the interface with a crystal that controls the internal clock generator circuitry. An 8 MHz crystal is used. The $\overline{\text{RESET}}$ line is connected to the MC34064 reset chip. V_{RL} and V_{RH} are reference voltages for the analog-to-digital converter circuitry. Port B is a general purpose output port, one of which is used to transmit the data to the IR LEDs. Ports A is configured for timer input capture (IC). The TxD and RxD are the transmit and receive data input/output signal for the SCI [19].

The MC68HC11E2 used in this project is a one time programmable (OTP) MC chip. The OTP is a Programmable Read Only Memory (PROM) device. Once a program is written into the device with a standard EPROM programmer, it can not be erased or modified. This is usually used for limited production runs before a ROM mask is done in order to test code. The package does not have a window for erasing. Therefore, the program should be debugged and tested before downloading it to the microcontroller.

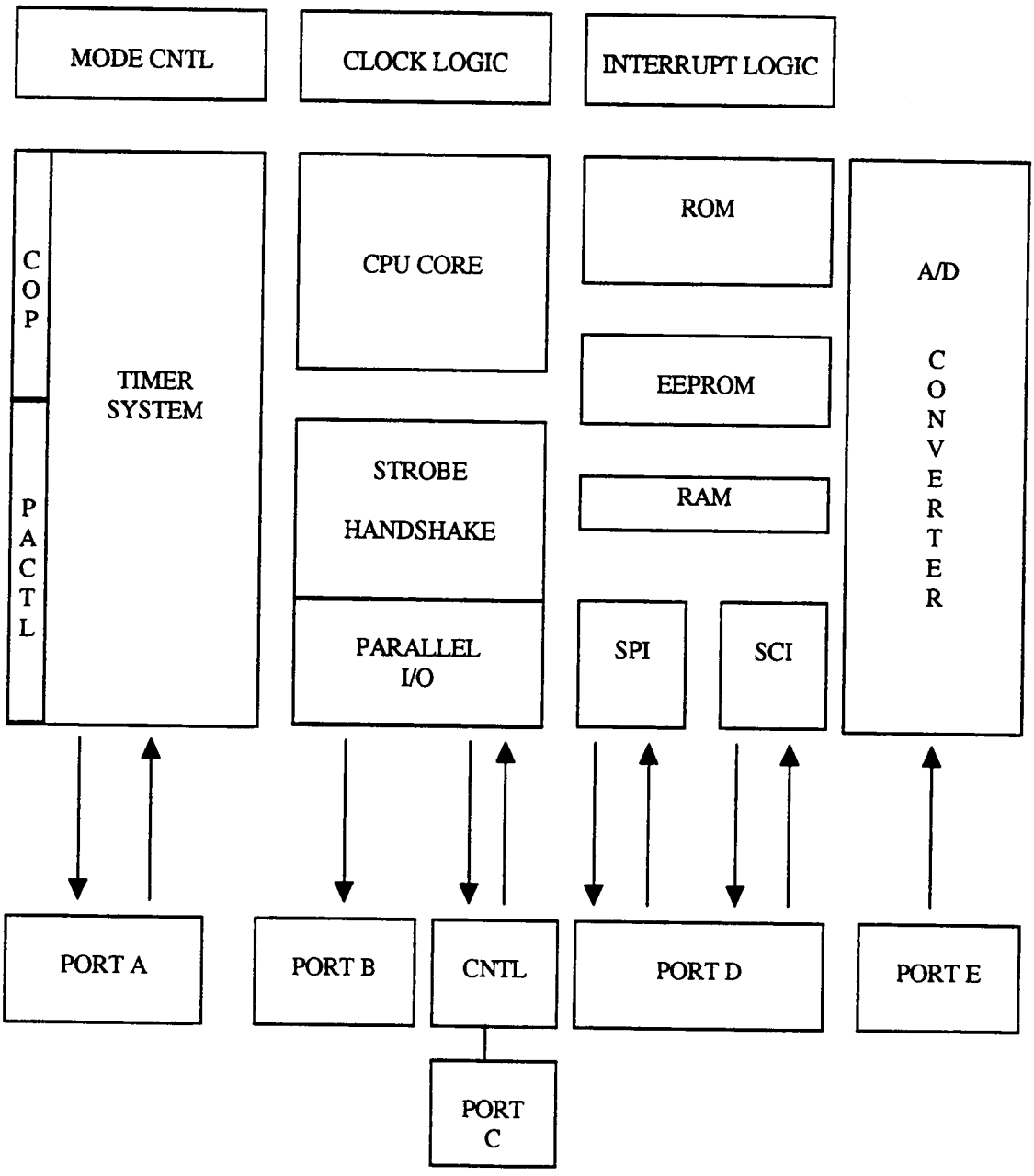
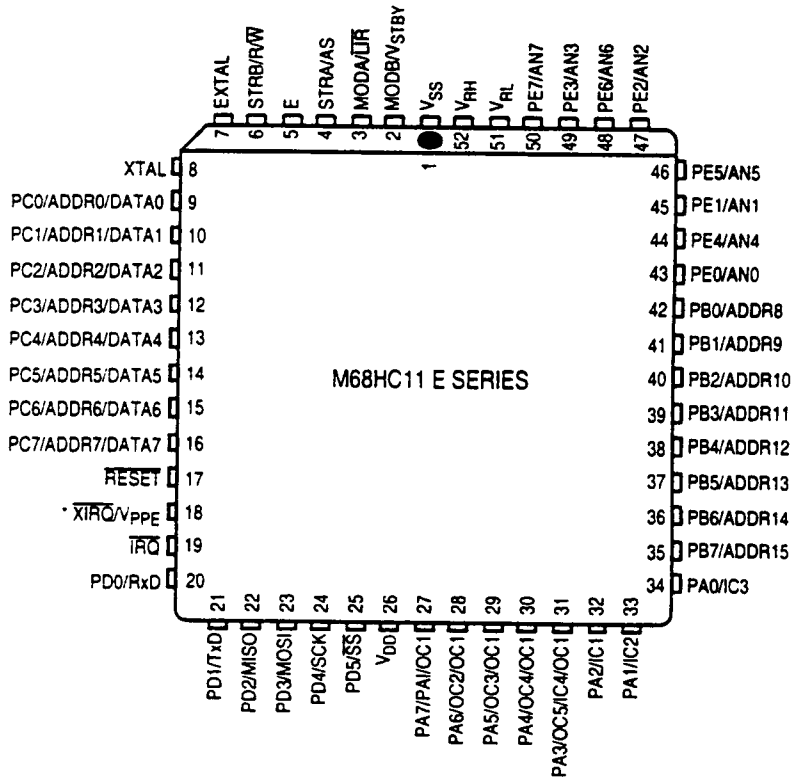


Figure 3.8 Block Diagram of the E2



* V_{PPE} APPLIES ONLY TO DEVICES WITH EPROM/OTPROM.

Figure 3.9 Pins Assignment of the E2

Microcontrollers, being a particular case of digital computers, have to be programmed so that they can perform an assigned task. The programming can be done with an assembly language program for a small simple process. For more complicated processes, a High Level Language (HLL) like "C" is indispensable. Machine language is the program representation which the microcontroller understands. It is not easy for humans to read, therefore assembly language is needed. It is a human readable form of machine language which makes it much easier for humans to work with [17]. Assembly language programs are usually fast and small. This is because the programmer is in charge of what goes into the program. A second reason for the use of assembly language is that all of the resources of the target computer system are readily available to the programmer. Thus, the input/output devices are directly accessible, the interrupt system can be manipulated, stacks can be modified, and so forth.

An assembly language program goes through a four-step development process. Figure 3.10 shows this process. First, a text editor is needed to create a new text file with the .ASM extension. This text file will be the source code. Second, the source code file is passed to the assembler program itself for translation to a relocatable object module file with .OBJ extension. As the assembler reads lines from the source code file, it will examine them, construct the binary machine instructions the source code represent, and then write those machine instructions to the object code file. Third, a linker transform the .OBJ

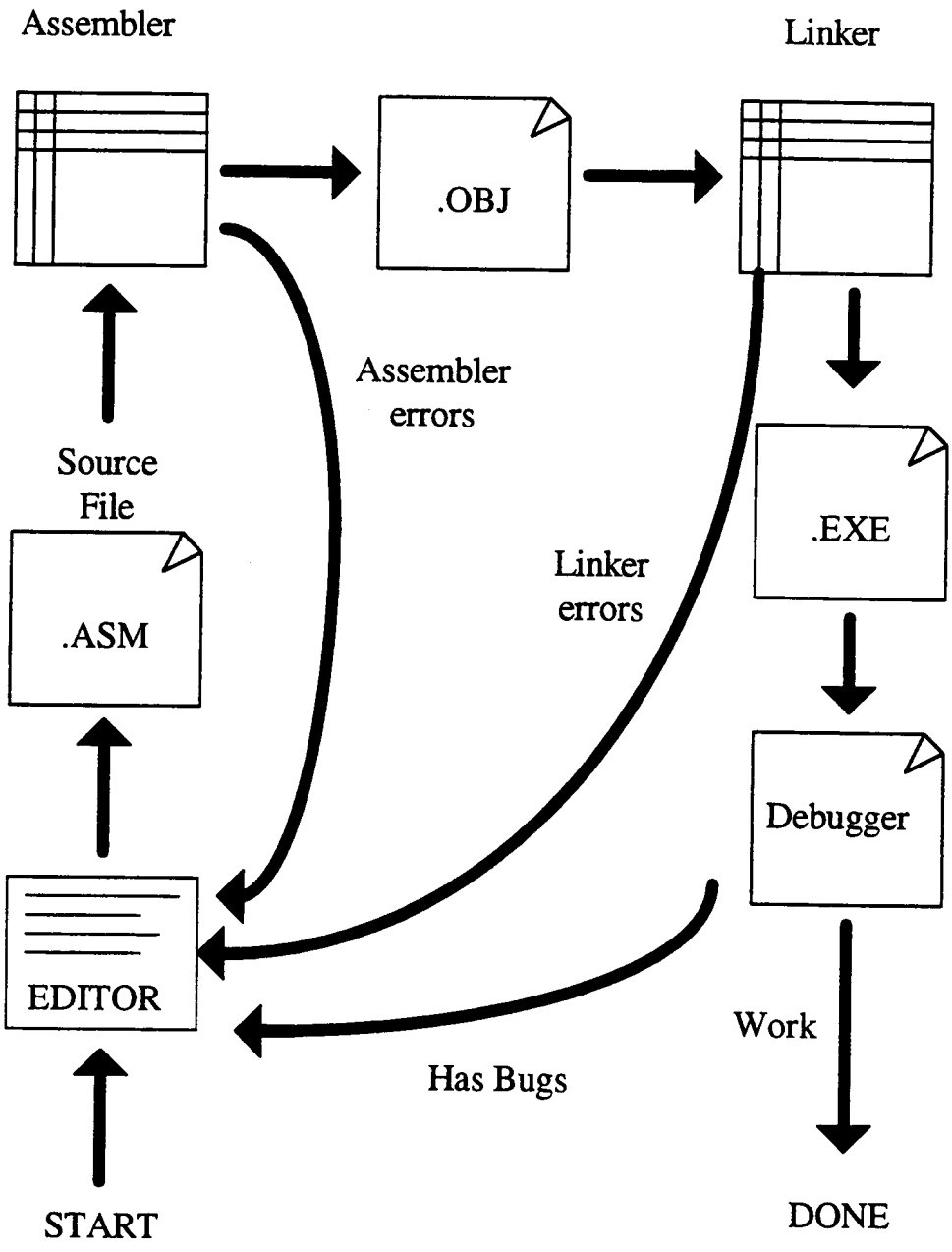


Figure 3.10 Assembly Language Development Process

extension file to an executable file with .EXE extension. It performs a number of operations in the .OBJ file. It weaves several .OBJ files into a single .EXE program file [18]. Finally, the .EXE program is debugged by using a debugger. A debugger is a utility program designed specifically to locate and identify bugs.

The software written in assembly language to program the microcontroller chip is partitioned into several headers and assembly source files. Building the software is directed by the MAKEFILE, written to be invoked under Borland Turbo C. It utilizes the 68HC11 freeware assembler called AS11NEW.EXE. The program CONVERT.EXE is used to convert the assembler S-record output to binary. The MAKEFILE directory contains all the dependencies and programs for making the MC6811 executable code. The address and operational constants for the code are included in the header file IR.INC. Definitions common to all 68HC11 variants are included in the header file 68HC11.INC. Definitions specific to the 68HC11E2 chip are included in the header file 811E2.INC These three header files can be found in Appendix A. Comments are present at each line after the comma to explain and define each line. To fully understand the commands, it should be referred to the Motorola MC68HC11 reference manual. It gives a good description and explanation of each command.

The IR.INC is the header file that includes the address and operational constants for the MC software. It specifies the starting address of the read only memory (RAM), internal register, EEPROM and vector table. A typical command for the ATFUR has the form described in page 150. Delimiters which must be defined like the slash which is equal to "/" as the start character of the signal, the colon which is equal to ":" as the start character of transmitting the burst and delay signals, and the semicolon which is equal to ";" as the end character of the signal. It also sets the receive queue size to 128 which means that the MC will only handles a total of 128 bursts and delays.

The 68HC11.INC file is the header file that includes all definitions common to all 68HC11 microcontrollers. It relies on the following two constants being defined before assembly: REG_BASE the base address of on-chip registers (defined in IR.INC as equal to $1000), and VEC_BASE the base address of vector table (defined in IR.INC as equal to $FFC0). This include file defines the registers as offsets from REG_BASE with a leading underscore and also defines the registers as the absolute address.

The 811E2.INC file is the header file that includes the definition specific to the 68HC811E2 microcontroller chip. It defines the RAM_SIZE and ROM_SIZE for the 68HC811E2, and adds register bits not included in the 68HC11.INC file. The RAM_SIZE is equal to 256 and ROM_SIZE is equal to 2048 (2*1024). It also requires BIT0-7

256 and ROM_SIZE is equal to 2048 (2*1024). It also requires BIT0-7 being defined as $01 through $80. This file was not combined with the previous one because these definitions corresponds only to the E2. This way if the chip is changed and replaced by a different microcontroller from the same family, only this file needs to be changed.

After explaining the purpose and functionality of the include files, comes a description of the main program IRMAIN.ASM also found in Appendix A. It defines Ports A, B, D and flag constants, defines the variables, and defines the main loop. This paragraph will try to translate the program commands into successive sentences. Appendix D contains the flow chart of the software development plan. This program turns on the LED activity signal, initializes the variables, the serial peripheral interface (SPI), and the phase lock loop (PLL) hardware. The activity LED and the PLL will be described later in this chapter. The time table entries are set to default for 1.000 ms and the repeat count sequence of the signal to 1. The serial communication interface (SCI) is set up for 9600 baud, 8 bits, 1 stop bit and no parity. This corresponds to the same parameters set on the GPIB-232CT-A box. The SPI is set up for 1MHz shift clock, CPHA0 and CPOL0. The /SS output (PD5) is used to strobe the data into the MC145157 R and N counter latches. The voltage controlled oscillator (VCO) is enabled and set up for the default frequency 37.5 kHz. The activity LED is turned off and the power-on message is sent out to the RS-232 port. The main loop strobes output PB7 (the ALIVE indicator) every time it is looking

for a beginning slash character, which is the beginning of the command sent by the LabVIEW driver. If it does not find a slash, it goes back, waits and looks for another character. It also checks for a semi-colon which indicates the end of the transmitted signal because if the RS-232 receive interrupt routine finds one it de-asserts CTS and thus must be re-asserted. This is done to hold of the serial port host until a pattern is transmitted so that interrupts do not interfere with the timing of the transmit pattern. Once a slash is found, multiple slashes are checked to assist in error recovery, and any redundant ones are discarded. After getting the slash, the program looks for the repeat count digit. If a digit is found it will be stored as the repeat count if it is a non-zero digit. If it is a zero, then it is considered an error and is reported. Then the program looks for a colon. Once a colon is found, the previous repeat count, frequency, and times intervals from the time table are used. The loop will look for the sequence that is an even number of bursts and delays. After getting the repeat count digit, the loop looks for the carrier frequency. If a colon is the first non-white space character that is found, then the default frequency and times are used and the loop looks for the ON/OFF sequence. Else, if an error is found in the frequency value, control goes back to the top of the main loop. Otherwise, the frequency is set by the ParseFrequency routine. After getting a valid frequency value, the loop looks for the time intervals. The times should be separated by a space so the ParseTime routine can identify them. Up to eight time intervals can be entered. The maximum time intervals obtained when customizing the codes were six or seven.

The eighth one was added in case it is needed when new TV models are available. If a colon is found then the loop will start looking for the sequence pattern. Else, if an error is found in a time interval value, control goes back to the top of the main loop. Otherwise the time intervals are stored in the current time entry and the index is incremented. Once all time entries are parsed, the loop looks for a colon. As long as the character is not a colon or a pattern digit (1 through 8), it keeps looking for one or the other, indicating that if all time intervals are specified no colon is necessary. Once the colon is found then the sequence transmit pattern is parsed. It starts with the first pattern digit (1 through 8) and continues through the last digit followed by a semi-colon. The receive queue index of the first pattern digit is saved in the TxStart variable for use in the Xmit routine. The last pattern digit must be followed directly by the semi-colon or an error is reported. The length of the sequence pattern is not checked, but it should not exceed 127 characters. If this happens, the input queue will wrap-around and the first characters will be over-written by the later part of the transmit pattern. Then the code sequence will be totally different from the intended one, and the TV will not respond. Therefore the user must be sure not to enter a transmit pattern longer than 127 characters. When the end of the pattern is found, its length is checked for an even number. If it is even then the Xmit routine is called to send the pattern. If no patterns are entered, the zero-length pattern will be valid and no error will be reported. After the Xmit routine returns an OK, the string is sent and control returns to the top of the main loop and

wait for a new command from the LabVIEW driver.

The program reports errors when they occur. To view the error messages, the hardware of the ATFUR must be connected directly to COM2 of a PC. The communication parameters should be set accordingly, and the user must type in the sequence. The errors will show on the PC screen when they occur. The common errors are: bad repeat count when the repeat count value is zero, bad pattern when no spaces are entered between the time intervals or when a character is missing, and odd pattern length when there is more bursts than delays or vice versa. Unfortunately the ATFUR VIs cannot show these errors because it does not read anything from the RS-232 or GPIB interfaces, it just writes to them. The program file of the IRMAIN.ASM can be found in Appendix A with the include files.

Once the main program has been explained, the special subroutine programs should be discussed. A subroutine is a specialized program structure. It provides a means where a segment of the program can be called from elsewhere. The original intent was to allow the segment to be written once and called many times [24]. Figure 3.11 illustrates the concept of a subroutine execution. The main program execution sequence is shown as the top horizontal line. At point A in the program the subroutine is called and the processor must branch to the beginning of the subroutine at point X. Upon the completion of the subroutine at point Y, the processor branches back to just after point A.

Later in the program at point B the subroutine is called again. Again the processor must execute the code between points X and Y. This

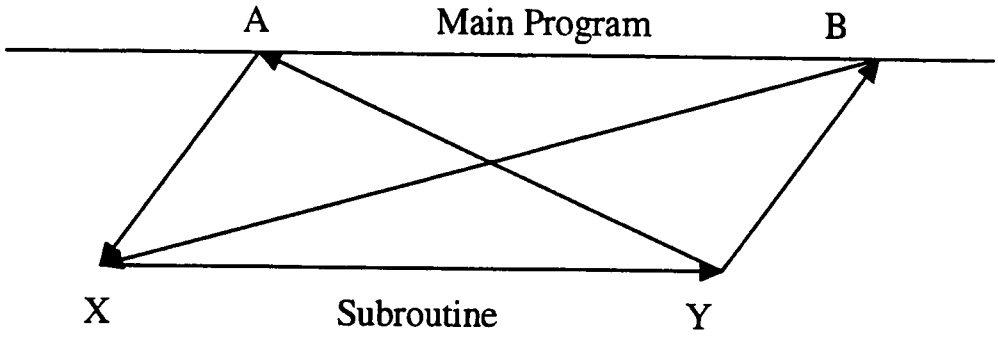


Figure 3.11 Subroutine Execution

time, however, the return to the main sequence will be just after point B. The first subroutines are the input parsing functions IRPARSE.ASM. The following paragraphs will try to explain the functionality of these subroutines in words rather than in assembly language. The ParseFreq subroutine calls the ParseNumber function to input a number with up to two digits before the decimal point, an optional decimal point, and one digit after the decimal point. The frequency value should be in the form (NN.N). If the number is returned from the ParseNumber subroutine without errors, its range is checked to be within the range MIN_FREQ to MAX_FREQ. If it is outside the range an error is reported, otherwise it calls SetFreq to set the frequency synthesizer to the possibly new frequency and returns with the carry condition code clear indicating success. The range of the

frequencies was determined from the available TV models. Usually, televisions operate between 35 and 45 kHz. To be on the safe side, the minimum frequency was set to 30kHz and the maximum frequency was set to 55.5kHz, which is a relatively wide range.

The ParseTime function subroutine calls the ParseNumber function to input a number with up to two digits before the decimal point, an optional decimal point, and three digits after the decimal point. The time value entered should be in the form (NN.NNN). If the number is returned from the ParseNumber without errors, its range is checked to be greater or equal to MIN_TIME. The minimum time is set to 00.080 ms. If it is outside the range an error is reported, otherwise it stores the entered time into the TimeTbl array according to the value in an index. The time is then converted from μs to half μs ticks by multiplying it when stored in the table. This way the Xmit subroutine does not have to convert the numbers during transmission. On success, the carry condition code is clear on return.

The ParseNumber subroutine is a generic fixed-point parsing routine to input up to the specified number of integer digits, an optional decimal point, then up to the specified number of fractional digits. If the maximum number of fractional digits is not parsed before a non-digit is found, the value is normalized by multiplying by ten for every fractional digit not parsed. For example, if 12.1 is parsed when the maximum FraDigits is 3, the result is multiplied by 100 so that the

decimal point is implicitly three spaces from the right. ParseNumber ignores leading white space, and when a non-digit or repeated decimal-point is encountered, parsing of digits is ended.

ParseNumber can optionally return with the carry condition code set and the colon first flag set indicating that the first non-white space character encountered was a colon. This is used to skip entry of frequency or time intervals and skipping the pattern parsing. The MultAccum and Mult10 routines are used to accumulate the decimal digits by ParseNumber. MultAccum multiplies Accum [0-2] by 10 and add decimal value in A. Mult10 multiplies Accum [0-2] by 10, adds Accum*2 to Accum*8 to get Accum*10. SkipWhite skips all white-spaces characters, such as spaces, tabs, carriage returns (CR), and line feeds (LF). IsDigit returns with the carry condition code clear if the value in the A accumulator is ASCII 0 through 9. IsPattern returns the same for ASCII values from 1 through 8. The PutByte and PutNibble routines send the value in a out as one or two ASCII hex nibbles. The Report*Err routines report various combinations of error conditions such as bad format, not a number, etc.

The next routine is the I/O subroutine named IRIO.ASM. The SCIInit initializes the RS-232 port to 9600 baud, 8 bits, 1 stop bit and no parity. It also enables the transmitter, receiver, receive interrupts, and enables CTS. The GetChar returns the next character in the input queue, waiting if the queue is empty. GetChar actually calls PeekChar

to get the character value then removes it from the queue. PeekChar returns the value of the next character in the receive queue without removing it, so that it can be re-parsed. While waiting for a character, the activity LED is toggled ON/OFF continuously, resulting in an effective 50% intensity. The PutChar sends the character in the A register out the RS-232 port. The PutStr sends out the zero-terminated string pointed to by the Y register out the RS-232 port. The PutCrLf sends a carriage return and linefeed out the RS-232 port. The SCISvc routine is the serial port receive interrupt service routine. If the character received is a semi-colon, indicating the end of a transmit pattern, CTS is cleared to hold off further transmissions from the host until the pattern can be transmitted. In any case, the character is placed into the input queue and the queue pointer and counter are incremented. The SPIInit initializes the SPI interface used to shift divider values into the MC145157 frequency synthesizer. Shift frequency is set to 1MHz, CPHA and CPOL are 0, and master mode is enabled. The /SS pin (PD5) is made an output to strobe the counter values into the MC145157. The PutSPI sends the byte in A out the SPI port to the MC145157. The PLLInit enables the VCO section of the MC14046 and sets the frequency synthesizer to the default frequency. The SetFreq sets the frequency synthesizer to the frequency in the D register. It sets the R and N divider values of the MC145157 to values looked up in the PLLTab table. The R and N values are packed with the R value in the upper 6 bits of the 16-bit entry, and the N value in the lower 10 bits. The SetDivisor sets the divisor of the MC145157 to the

value specified in the D register. The least significant bit (LSB) is the divider select bit, 1 specifies the R counter, 0 specifies the N counter. The next 14 bits are the counter, and the most significant bit (MSB) is a don't care.

Finally, the Xmit subroutine performs the transmission of the pattern, and repeats it the specified number of times. During transmission, the activity LED is turned off. During the first burst of each repetition of the transmit pattern, debugging output pin TRIGGER outputs so that they occur simultaneously. The transmission is started by scheduling an output capture (OC1) event a fixed time from the start of calling Xmit. Then each time an OC1 event occurs the next time interval is retrieved from the receive queue and looked up in the time interval table. The OC1 timer compare register is updated to schedule the next ON/OFF transition at the end of that interval, repeating until the end of the pattern is sent. Then, it either starts over at the beginning of the pattern if one or more repeats remain, or the transmission is complete and the subroutine returns after turning the activity LED back on. The only complication is when an interval is greater than 32.7675 ms (65535 ticks). In this case, an XmitLong operation is performed which essentially waits for the specified number of 65536 wrap-around to occur and then the outputs get updated on the following OC1 compare, extending the interval by 32.768 ms for every wrap-around of the timer. These subroutines can be found also in Appendix A.

The Frequency Synthesizer table is found in Appendix B. This table contains the R and N divider values for a given synthesizer frequency. The base address (PLLTAB) is offset by MIN_FREQ, with successive entries representing 100Hz increments. The limit to table length is 256 entries, so MAX_FREQ will always be MIN_FREQ + 25.5kHz. This table is processed by the GNU AWK from the file IRTAB.DAT, which is generated by the program F.C/F.EXE. This file builds the data table based on a given maximum deviation from ideal frequencies and optimizes the divider value. To generate a different IRTAB.DAT table, the F.EXE program should be run specifying a different maximum error. The maximum divider value for N is 1023 due to the data packing format, and the maximum value for R is 31. If either of these values exceed their maximum, the AWK program will indicate it in the last line which has the number of errors encountered. It will also generate a line that causes an assembly-time error in the file IRTAB.ASM, assuring that MAKEFILE will notice the problem.

As said earlier, the CONVERT.EXE was used to convert the assembler S-record output to binary. The CONVERT.EXE is a file conversion utility which takes Motorola S-record files as input and converts them to a memory image format. The following is a list of the parameters required and a description of each:

- Prom Size: the size of the output file in k bytes.
- Input file: The name of the S-record input file.
- Output File: the name to be assigned to the output file.

- Offset: The byte offset to be subtracted from the address in the input file.

After responding to all of the above requests, the input file is read and converted. No error checking is done in this conversion. The assembler used is given out free of charge by Motorola for evaluation of Motorola's microcontrollers. The file ASREF.MAN gives detailed documentation and description of this Motorola Freeware cross assembler available for PC compatibles. All other related files are found in Appendix A.

3.4 COMPONENTS

The purpose of this section is to briefly describe each component used in this project. It will familiarize the reader with the functionality of each device. This section will revolve around the power supply, the voltage regulator, the MAX232, the MC34064, the PNP and NPN transistors, the IR LEDs, the frequency synthesizer, J-K flip-flop and the ADAPT-11™ prototype board.

The power supply will be between 9-15 VDC provided by a power adapter that takes 110 VAC as input. It is adjustable and it will be set to give exactly 15 V. The AC adapter is a modular unit which is compact and easy to use. When using an AC adapter, some safety precautions should be taken into serious consideration. All connections

to the AC line must be inside a fully enclosed housing; if any connection is exposed it is considered to be a potential shock hazard.

A voltage regulator converts a voltage applied to its input into a fixed voltage. The voltage regulator used in this project is the famous LM7805CT. It is a three terminal device, where pin 1 is the input voltage, pin 2 is ground, and pin 3 is the output voltage. Figure 3.12 shows the equivalent device circuitry. The input voltage is limited to a maximum of 15 V. The output voltage is within the range 4.8 to 5.2 VDC at a temperature of 25°C.

The MAX232 is a Maxim chip that converts RS-232 levels (-10,+10)V to TTL (0,+5)V voltages and vice versa. Maxim defines the MAX232 as a +5V powered, dual RS-232 transmitter and receiver [20]. This chip needs only a single +5V supply for operation. The RS-232 drivers/receivers have a board charge pump converter which converts the +5V input power to the +/- 10V needed to generate the RS-232 output levels. Since nearly all RS-232 applications need both line drivers and receivers, this chip includes both drivers and receivers in one package. Both receivers and line drivers (transmitters) meet all EIA RS-232 specifications. Figure 3.13 shows the MAX232 chip and equivalent circuitry.

The transmitters or line drivers are inverting level translators which convert the CMOS or TTL input levels to RS-232 levels. The

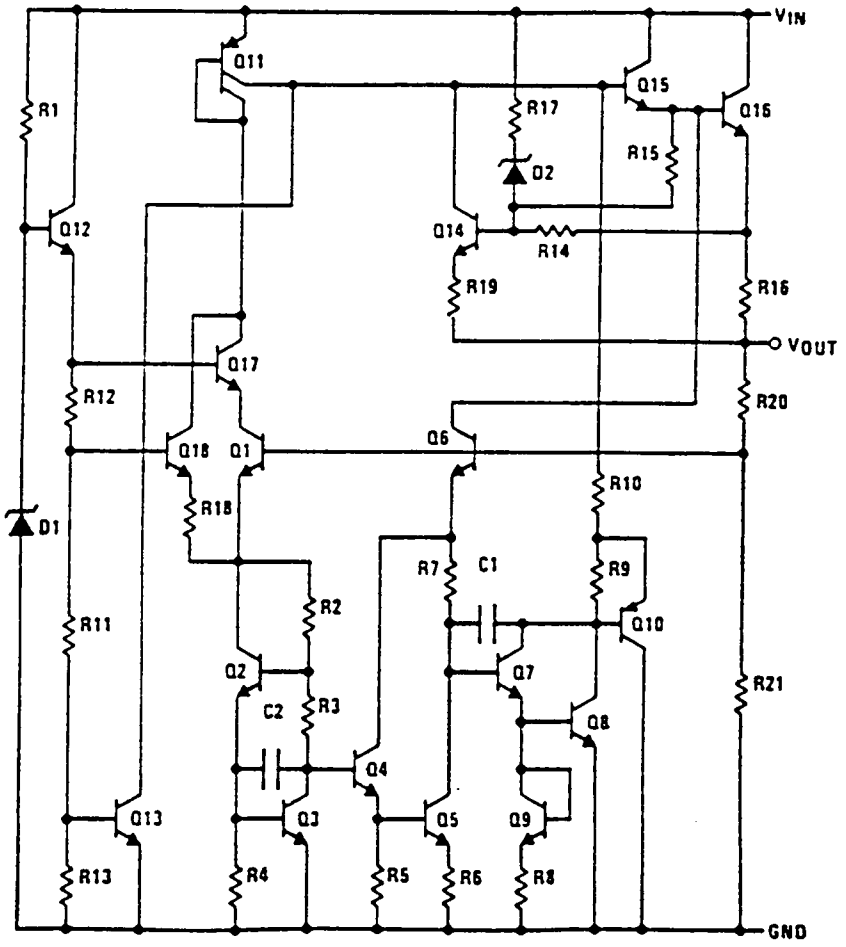
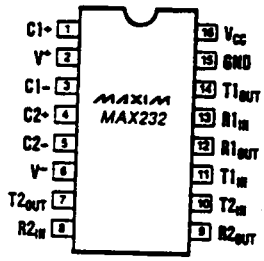


Figure 3.12 LM7805CT Voltage Regulator



16 Lead Small Outline
also available.

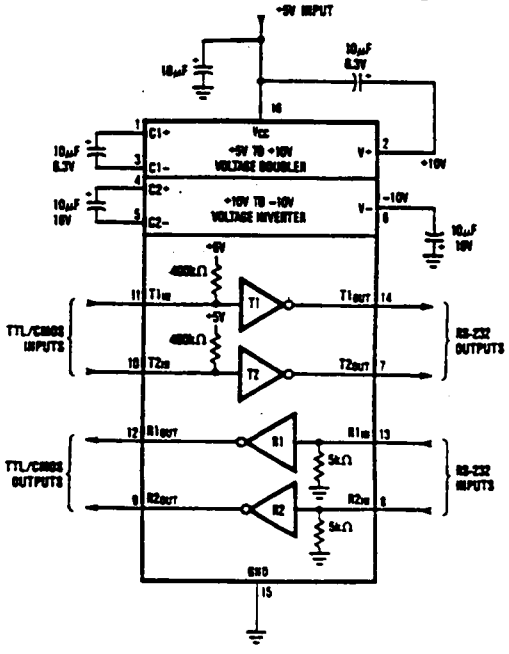


Figure 3.13 MAX232 Chip

input thresholds are both CMOS and TTL compatible, with a logic threshold of about 25% of Vcc. The inputs of unused drivers sections can be left unconnected. An internal 400KΩ input pull-up resistor to Vcc will pull the inputs high, forcing the unused transmitter outputs low [20].

The receivers convert the ± 5V to ± 15V RS-232 signals to +5V TTL/CMOS outputs. Since the RS-232 specifications define a voltage level greater than +3V as a 0, the receivers are inverting. The input thresholds of the receivers is set to 0.8V minimum and 2.4V maximum which are significantly tighter than the -3.0V minimum and +3.0V maximum required by the RS-232 specifications [20]. The 0.8V threshold ensures that the receivers will have a logic 1 output if they are not being driven.

In this application, pin 1 and 3 are tied together to a 10μF (6.3V) capacitor; pin 4 and 5 are tied together to a 10μF (16V) capacitor. Pin 11 and 12 are connected to the SCI of the MC. Pin 13 and 14 are connected to the DB9 pin female connector. Pin 15 is connected straight to ground, while pin 6 is connected to a 10μF (16V) capacitor and driven to ground. Finally, pin 2 is connected to 10μF (6.3V) capacitor and driven to the +5V supply. Table 3.7 shows the pin assignments of the DB9 connector. Figure 3.14 shows the physical layout of the DB9 connector.

Table 3.7 DB9 Pin Connection

PIN #	NAME	DESCRIPTION
1	DCD	Data Carrier Detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data Terminal Ready
5	SG	Signal Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicate

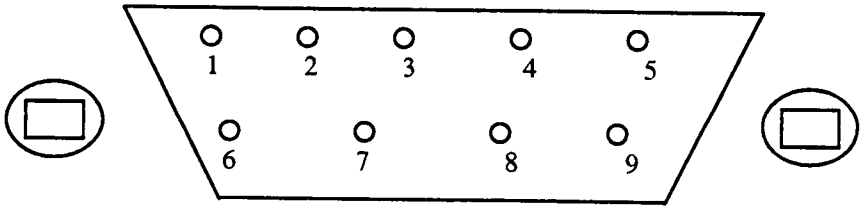


Figure 3.14 DB9 Connector

The MC34064 is a simple, inexpensive Low Voltage Inhibit (LVI) device that provides an open drain output to directly drive the $\overline{\text{RESET}}$ pin of the MC. This device is connected to V_{DD} to get power, to V_{SS} (ground) and to the $\overline{\text{RESET}}$ pin of the MC. A 4.7 KΩ pull-up resistor is required for the reset circuit. This device automatically controls the $\overline{\text{RESET}}$ pin, it drives it low whenever V_{DD} is below legal limits. Figure 3.15 shows the block diagram of this chip.

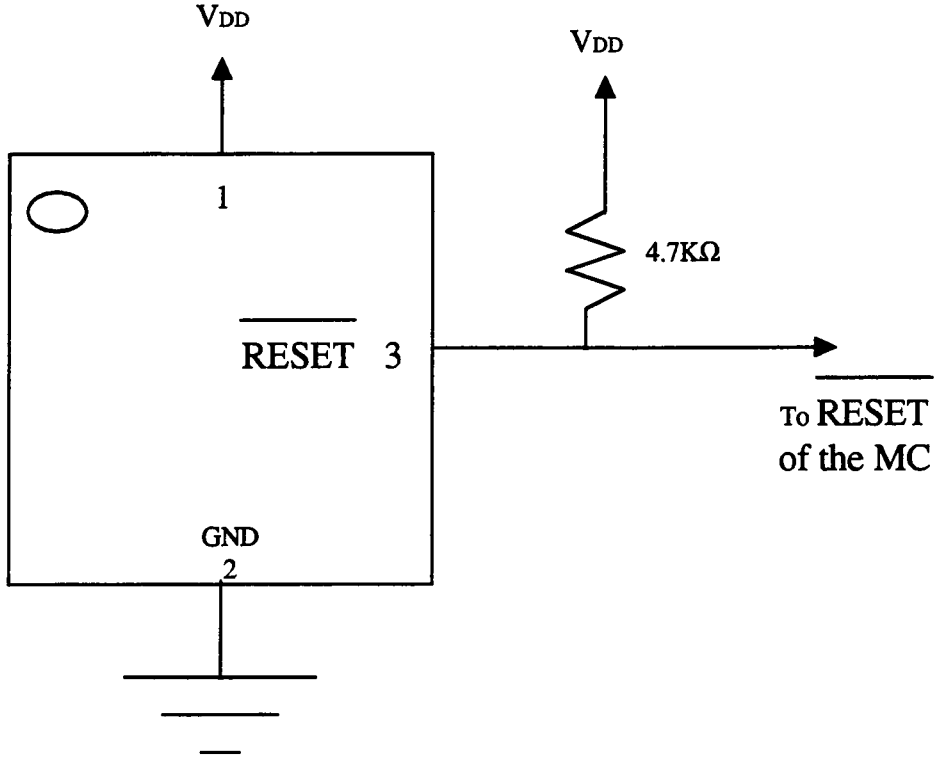


Figure 3.15 MC34064

As said earlier, it is extremely important that the $\overline{\text{RESET}}$ is controlled during power up and power down sequences in order for the MC to function properly. Until V_{DD} reaches 10% of the operating V_{DD} during a power on sequence, the $\overline{\text{RESET}}$ pin on the MC must be held low. When V_{DD} reaches 1V during power up, a valid clock is present for the MC to count 4064 E clock cycles during which the $\overline{\text{RESET}}$ pin is driven low. After this delay, the MC will release $\overline{\text{RESET}}$ for about 8 E clock cycles and then sample the voltage at the pin. If $\overline{\text{RESET}}$ is high, the MC will fetch its power on reset vector and begin program execution. If $\overline{\text{RESET}}$ is low, the MC will delay another 8 E clock cycles and then sample the pin again. This sequence proceeds until $\overline{\text{RESET}}$ is sampled at a logic 1, ideally the point at which V_{DD} is within 10% of its operating level. the $\overline{\text{RESET}}$ pin must also be controlled during power down or loss of power. When V_{DD} drops below 10% V_{DD}, $\overline{\text{RESET}}$ must be driven low to avoid unexpected program execution or corruption of RAM or internal EPROM.

Transistors are semiconductor devices with three leads. A very small current or voltage at one lead can control a much larger current flowing through the other two leads. This means that transistors can be used as switches. The three layers of a Bipolar transistor are the emitter, base and collector. The base is very thin and has fewer doping atoms than the other two layers. the three main key features of transistors are: The base emitter junction will not conduct until the forward voltage exceeds 0.6V. Too much current will cause a transistor

to heat up and operate improperly. Too much current also may damage or permanently destroy the semiconductor chip that forms a transistor. These features were taken into consideration through out this project, and appropriate current and voltage were supplied.

The PNP device is a silicon transistor known as a Bipolar Junction Transistor (BJT). In this application, the Q2N2907A transistor will be used. Figure 3.16 shows the device outline and Table 3.8 shows the parameters of this transistor.

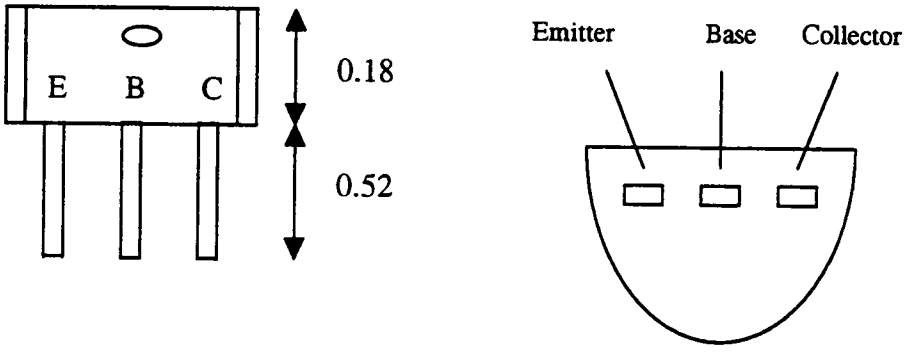


Figure 3.16 PNP Transistor

Table 3.8 PNP Parameters

NAME	PARAMETER	UNITS	VALUE
Is	Saturation Current	V	650E-18
XTI	Saturation Current Temperature coefficient	-	3
Eg	Energy Band Gap for Temperature coeff.	eV	1.11
Vaf	Forward Early Voltage	V	115.7
BF	Forward beta	-	231.7
NE	b-e Emission coefficient	-	1.829
Ise	b-e saturation Current	A	54.81f
Ikf	Forward High Current Beta Roll Off	A	1.079
XTB	Beta Temperature coefficient	-	1.5
BR	Reverse Beta	-	3.563
NC	b-c Emission coefficient	-	2
Cjc	b-c Zero Bias Junction Capacitor	F	14.76p
Vjc	b-c Built In Potential	V	0.75
Cje	b-e Zero Bias Junction Capacitor	F	19.82p
MJE	b-e Grading coefficient	-	0.3357
Vje	b-e Built In Potential	V	0.75
TR	Reverse b-c Transit Time	sec	111.3n
TF	Forward b-e Transit Time	sec	603.7p
Itf	High Current Parameter for TF	A	0.65
Vtf	Voltage Dependence of Vbc on TF	V	5
Rb	Base Resistance	Ω	10

The NPN Q2N2907A device is also a BJT which has the following characteristics at maximum ratings:

V_{CE}	Collector Emitter Voltage	160V
V_{CB}	Collector Base Voltage	160V
V_{EB}	Emitter Base Voltage	7V
I_C	Collector Current	1000 mA
P_D	Total Power Dissipation	900 mW

When the base of the NPN transistor is grounded (0 Volts), no current flows from the emitter to the collector, the transistor in this case is OFF. If the base is forward-biased by at least 0.6Volts, a current will flow from the emitter to the collector, the transistor in this case is ON. When operated in only these two modes, the transistor functions as a switch. Whenever it gets 0V it shuts off, and whenever it gets a positive voltage it opens and allows the current to flow through. Figure 3.17 shows the characteristic curves of this transistor and Figure 3.18 shows the mechanical outline.

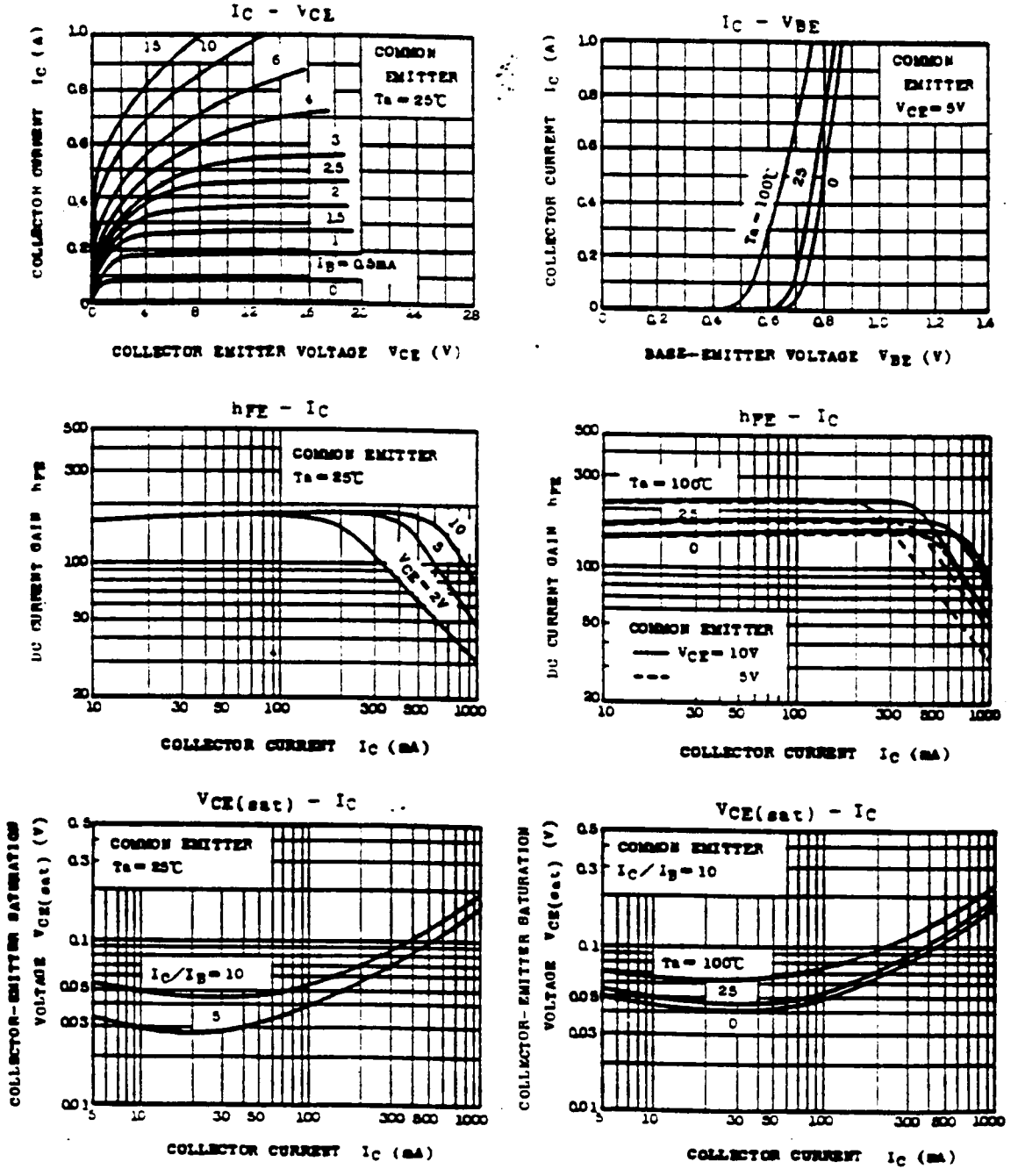


Figure 3.17 Characteristic Curves of NPN

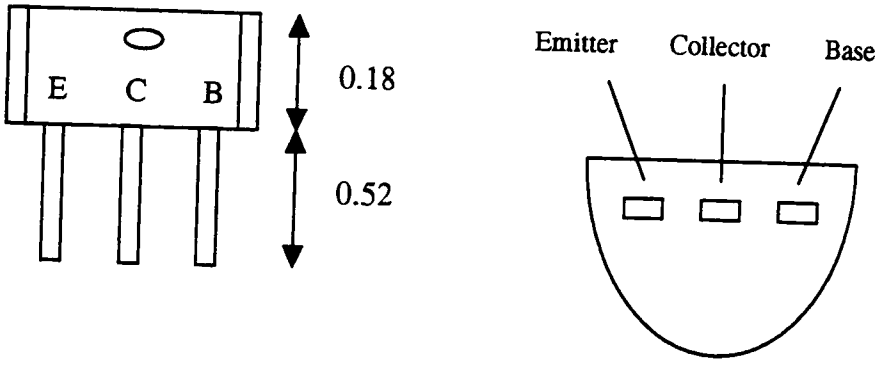


Figure 3.18 NPN Transistor

Before talking about the IR LED type, a general discussion of infrared radiation is required. Infrared radiation is the portion of the electromagnetic spectrum that extends from the long wavelength, or red, end of the visible light range to the microwave range. Invisible to the eye, it can be detected as a sensation of warmth in the skin. The infrared range is usually divided into three regions: near infrared with wavelength 0.78 to about 2.5 micrometers; the middle infrared with wavelength 2.5 to about 50 micrometers; and far infrared with wavelength 50 to 1000 micrometers [21].

The IR LED diodes are made from III-V compounds related to gallium arsenide (GaAs). They operate over a wide range of frequencies, and are used in many electronic applications. The IR LED is a p-n junction in which an applied voltage yields a flow of current, and the recombination of the carriers injected across the

junction results in the emission of light (heat). The IR LEDs are used as a source with glass optical fiber for a long range optical fiber transmission. The glass optical fiber suffers much less loss and have the lowest transmission loss in the infrared at wavelengths of 1.3 and 1.55 μm. When current flows through a diode, infrared radiation is emitted in two different spectral bands: incoherent radiation over a wide spectrum related to the blackbody emission curve corresponding to the temperature of the diode, and recombination radiation over a narrow spectrum whose wavelength is determined by the diode's material and its physical characteristics [22]. This emission is illustrated in the following equation:

$$dH/dt = - K A dT/dx \qquad (3\text{-}1)$$

where K = thermal conductivity

A = surface area

dH/dt = time rate of heat flow

dT/dx = temperature gradient

In other words, the heat transfer rate between an IR LED and a receiver that are kept at a given temperature gradient is directly proportional to the thermal conductance of the heat flow path, or inversely proportional to its thermal resistance. No two diodes operate identically, since each of them emits infrared radiation at a different power level. The measurement of IR radiation is done by measuring their power dissipation [22].

The forward voltage across an LED must exceed a threshold level before a current can cross the junction. In this project the threshold of LEDs used is 1.6V. The heat emitted by an LED has a narrow wavelength range because the electrons in it are all excited to the same level. When an LED begins to conduct, the voltage increases gradually while the current increases rapidly, because I is an exponential function of V. Too much current applied to the LED will overheat it and possibly separate the leads or melt the device. A safety test has been performed on the LEDs and used to check their maximum ability in handling a specific current. Since the heat emitted by the IR LEDs is directly proportional to current through the LED. Therefore they are ideal to transmit information by brief pulses of current. Figure 3.19 shows various properties in graph format and Figure 3.20 shows the mechanical data of the IR LEDs used.

After viewing the characteristic curves and the mechanical data of the IR LEDs used in this application, it is instructive to show the ratings of the IR LEDs at 25°C ambient temperature. Table 3.9 shows a summary.

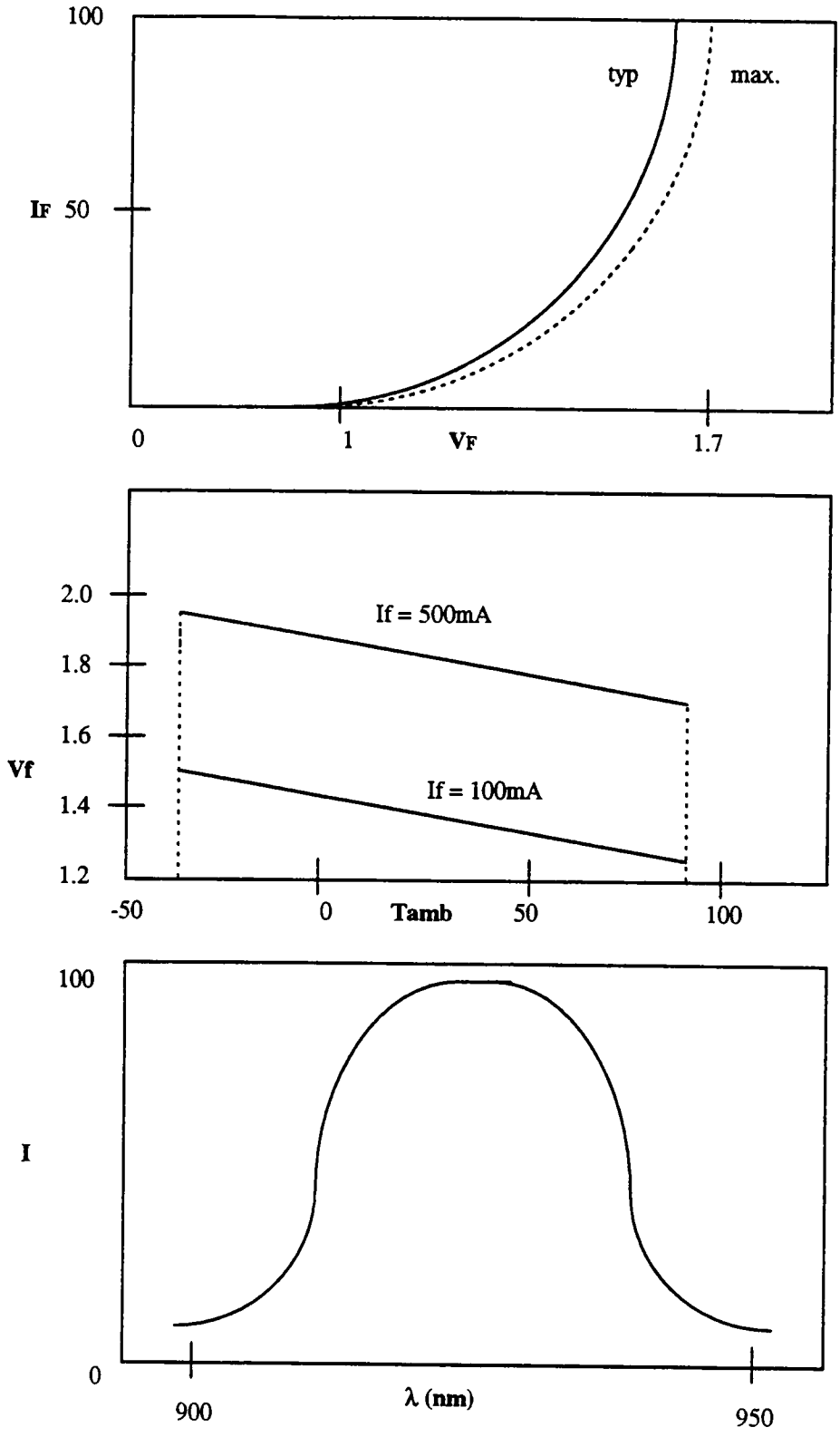


Figure 3.19 IR LED Properties

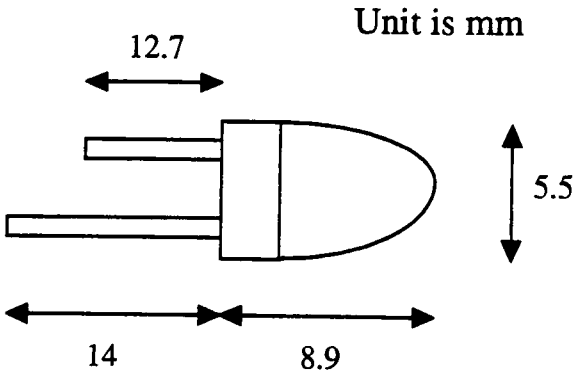


Figure 3.20 Mechanical Data of IR LED

Table 3.9 Ratings at 25°C Temperature

V_R	5 V
I_F	100 mA
I_{FSM}	2.5 A
P_D	215 mW
T_j	+ 100°C
T_{stg}	-55 to + 100°C
T_L	+ 260°C

One major issue was to provide the desired frequencies. At 2MHz the MC68HC11 has a basic timing resolution of 0.5μs. Therefore, any frequency will have a delta of ±0.5μs. For example, a 40kHz (25μs) frequency will have a maximum value of 40.8kHz (24.5μs) and a minimum value of 39.2kHz (39.2μs), which is not the 100Hz resolution specified in the format of the transmit command. Going to a phase locked loop running off a 100Hz master time reference is one solution. The MC68HC11 can easily provide the 100Hz along with a 10-bit divisor for the PLL.

A digital PLL circuit has a serial shift register receiving input pulses and producing time-delayed output pulses, a clock generator applying clock pulses to the shift register to drive it and set the phase shift of the output pulses, and a digitally-controlled potentiometer connected in series with the clock generator and being adjustable to change its resistance in increments in order to adjust the resistance of the clock generator and thereby set the frequency of the clock pulses applied to the shift register and the time delay produced by the shift register [23]. A feedback control arrangement receives the same digital input pulses as received by the shift register and detects the periods of the input pulses by counting to produce control pulses proportional to the detected periods. A ROM unit stores a look-up table of values representing an array of different counts of increments by which the potentiometer resistance can be adjusted. The ROM unit can be accessed by input signals identifying different addresses. The input

signals are composed of the control pulses and other external pulses defining different desired phase shifts of the output pulses. The ROM unit produces output signals that are used to adjust the resistance of the potentiometer by the counts of increments stored at the accessed addresses and thereby set the desired frequency of the clock pulses and the phase shift of the output pulses.

The MC145157 is a CMOS low power phase locked loop frequency synthesizer. When combined with an external low-pass filter and voltage-controlled oscillator, this device can provide all functions of a PLL frequency synthesizer operating up to the device's frequency limit. The MC145157 has a fully programmable 14-bit reference counter, as well as a fully programmable ÷N counter. The counters are programmed serially through a common data input and latched into appropriate counter latch, according to the last data bit entered [23]. The features of this device are summarized below:

- Fully programmable Reference and ÷N counters.
- ÷R range = 3 to 16383.
- ÷N range = 3 to 16383.
- Lock detect signal.
- Compatible with SPI on CMOS MCUs.
- 3.0 to 9.0 V supply range.

Figure 3.21 shows the block diagram of the MC145157, and Figure 3.22 shows the pin assignment.

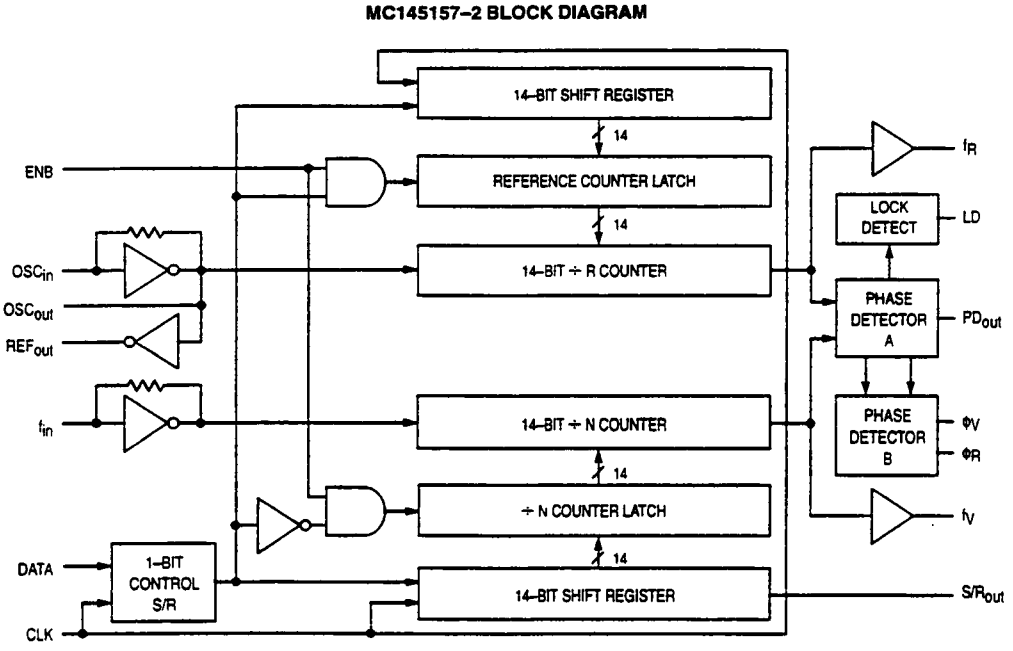


Figure 3.21 Block Diagram of the MC145157

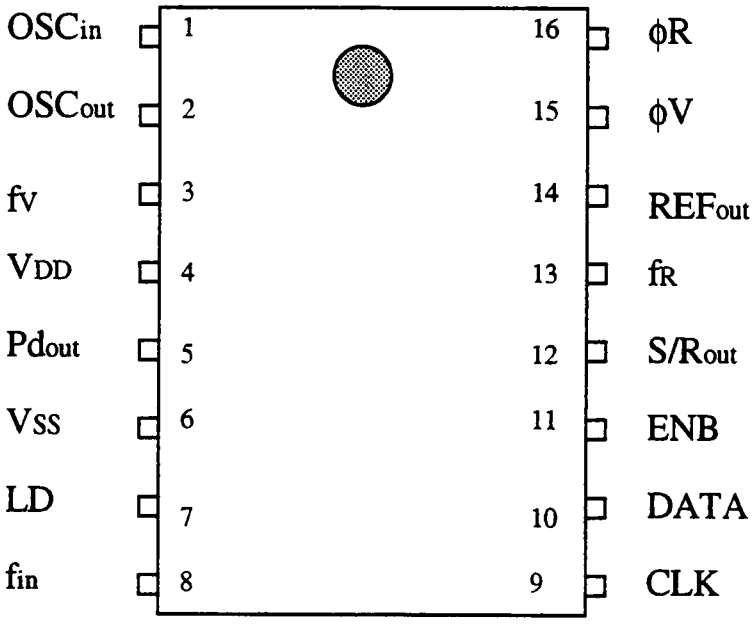


Figure 3.22 MC145157 Pin Assignment

The frequency synthesizer is used to generate the adjustable carrier frequency for the IR signals. It consists of an MC14046 voltage controlled oscillator (VCO) and the MC145157 phase lock loop (PLL) frequency synthesizer. The VCO components were selected for a minimum frequency of less than 20kHz when the VCO_{IN} is 0V, and a maximum frequency of more than 200kHz when VCO_{IN} is 5V. First C_1/R_2 are determined with VCO_{IN} = 0V. This must be done empirically since there is a wide tolerance in the VCO itself. Then with VCO_{IN} set to +5V, R_1 was chosen so that the desired maximum frequency was obtained. The values were determined to be 220pF for C_1, 10KΩ for R_1 and 1MΩ for R_2. These values were obtained from the following

equations:

$$f_{min} = 1/R_2 (C_1 + 32pF) \quad (3\text{-}2)$$

$$f_{max} = [1/R_1 (C_1 + 32pF)] + f_{min} \quad (3\text{-}3)$$

$$\Rightarrow C_1 = (1/f_{min}*R_2) - 32pF \quad \text{for } R_2 = 1M\Omega \quad (3\text{-}4)$$

$$\Rightarrow R_1 = (1/(C_1 + 32)\ 10E\text{-}12 * f_{max}) + f_{min} \quad (3\text{-}5)$$

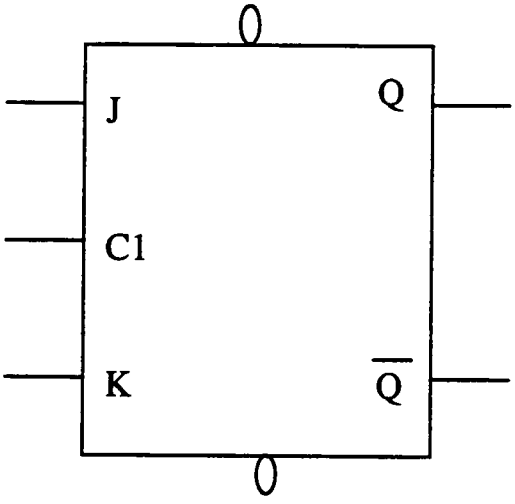
The output of the VCO is twice the desired carrier frequency, because it is divided by 2 in the output stage. To close the loop on the VCO frequency, its output along with the 2MHz E output from the microcontroller board are divided and compared by the MC145157. It contains two programmable dividers (N and R) which are set by the software via the 68HC11 SPI port. The 2MHz reference signal feeds the R (reference) divider, the VCO output feeds the N divider. The output of the two dividers is compared by the phase detector A. When the R divider output is at a higher frequency than the N divider output, it forces the phase detector output, PD_{out}, normally tri-state/high impedance, high. This feeds back through an R-C filter to the VCO_{IN} of the VCO, increasing its frequency. When the reverse is true, the VCO is divided by the N divider is a higher frequency than the 2MHz divided by the R divider, the PD_{out} signal is pulled low, decreasing the frequency of the VCO. When the frequencies are extremely close very high/low glitches occur on the PD_{out} signal, and the VCO is locked at the correct frequency.

Originally, the frequency synthesizer was planned to have the R divider fixed at 10000, resulting in its output being fixed at 200Hz and yielding an output of 100Hz per step in N divider value. But it was found that this low a synchronizing frequency was not very stable, and the loop response time would be very long. Therefore, a different scheme was derived such that the R and N divider ratio was optimized for maximum sync frequency for a given maximum deviation from the ideal frequency. The PD_{out} signal is filtered by R_3 (1MΩ) and C_2 (0.01μF). The VCO INH inhibit signal is driven by the 68HC11 PB2 output.

The J-K type flip-flop used is found in Figure 3.23. the JK = 00 results in no change, JK =01 clears or resets the output, and JK = 10 sets the output. The J-K type make use of the combination of 11 which causes the output to toggle or to change state from its prior value to the complement of that value. The letters J and K has no meaning [24]. With the transition table shown in Figure 3.23 the $\overline{PRE}$ and $\overline{CLR}$ inputs are dominant asynchronous, and active low. That is, whenever $\overline{PRE} = 0$ the flip-flop is set, regardless of the condition of J, K, and C1 inputs; and whenever $\overline{CLR} = 0$ the flip-flop is cleared.

The output of the frequency synthesizer goes to the clock input of the J-K flip-flop. When the carrier is to be transmitted the OC2 (PA6) output of the 68HC11 is set high, enabling the flip-flop to toggle at half the frequency synthesizer frequency. When OC2 goes low, the flip-flop

output goes low at the next clock and stays there, keeping the IR LEDs OFF. The output of the flip-flop is buffered by the IR output transmitter, which is driven into saturation for the peak current of the IR LEDs string. The other half of the flip-flop is unused and its inputs are tied high/low to minimize current drain.



C1	J	K	Q+
↓	0	0	Q-
↓	0	1	0
↓	1	0	1
↓	1	1	Q+

Figure 3.23 J-K Flip-Flop

The microcontroller was implemented via an ADAPT-11™ Module from Technological Arts. This board is a compact, carefully designed board that contain all the elements common to every 68HC11 design. Clock circuitry, reset circuitry, a mode select switch, a regulator, and an RS-232 interface are all provided on board. In

addition, all the microcontroller I/O and control lines are brought out to a versatile 50-pin header pattern on the printed circuit board. It also provides a socket for the 52-pin PLCC-packaged 68HC11 family members, and an 8MHz crystal. All these components have been presented earlier. The ADAPT-11™ has switches for in-circuitry programming of EEPROM for the 68HC811E2 chips [25]. The only modification necessary was to connect the 68HC11 PB0 signal (pin 42) to the unused RS232 buffer so that CTS to the host computer on the DB9 connector can be enabled and disabled while the ATFUR unit is transmitting. Connection to peripheral circuitry is via the dual-row 50-pin header. Figure 3.24 shows the mechanical dimensions of this module, and Table 3.10 shows the Header H1. The ADAPT-11™ board was used by following these steps:

- The board was plugged into the application board.
- Peripheral circuits were wired accordingly.
- Software was written in assembly language.
- Program was assembled and a Motorola S-record file was created.
- The board was attached to the PC through a serial cable.
- The board was put in BOOT mode, and the batch files were downloaded from the PC.
- Finally, the board was put in RUN mode, and it was executing the program.

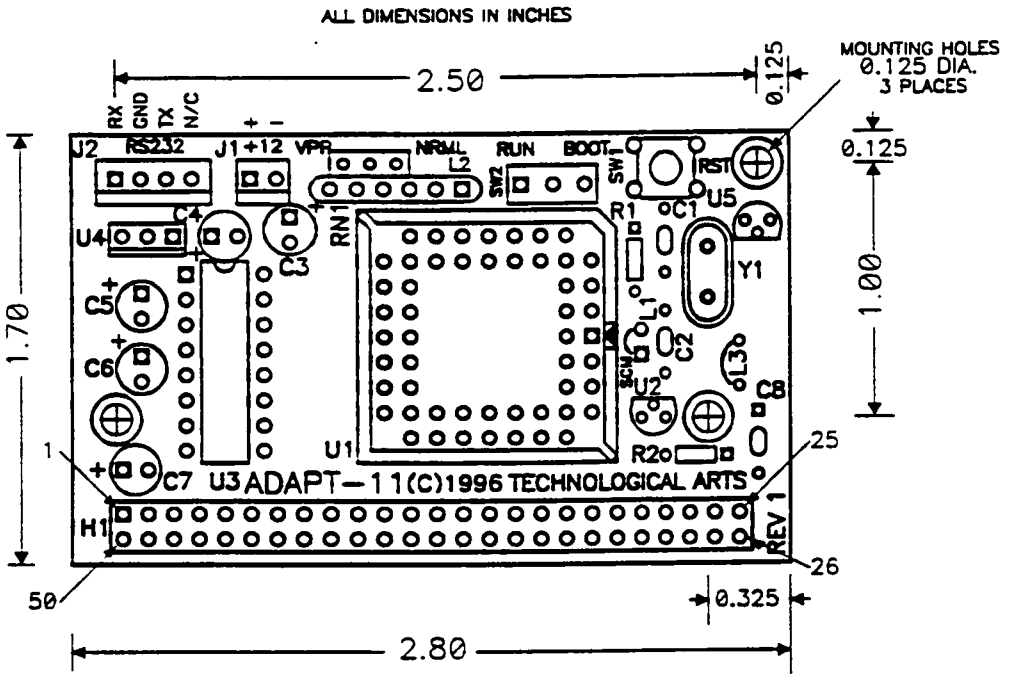


Figure 3.24 Mechanical Dimensions of the ADAPT-11™

Table 3.10 Header H1 Pinout

PIN	SIGNAL NAME	PIN	SIGNAL NAME
1	PD2	20	PB1 (A9)
2	PD3	21	PB0 (A8)
3	PD4	22	PE0 (AN0)
4	PD5	23	PE1 (AN1)
5	RESERVED	24	PE2 (AN2)
6	PA7	25	PE3 (AN3)
7	PA6	26	PE7 (AN7)
8	PA5	27	PE6 (AN6)
9	PA4	28	PE5 (AN5)
10	PA3	29	PE4 (AN4)
11	PA2	30	VRH
12	PA1	31	VRL
13	PA0	32	AS
14	PB7 (A15)	33	E
15	PB6 (A14)	34	R/W*
16	PB5 (A13)	35	PC0 (AD0)
17	PB4 (A12)	36	PC1 (AD1)
18	PB3 (A11)	37	PC2 (AD2)
19	PB2 (A10)	38	PC3 (AD3)

Table 3.10 (continued) Header H1 Pinout

39	PC4 (AD4)	45	XIRQ*
40	PC5 (AD5)	46	IRQ*
41	PC6 (AD6)	47	+5VDC
42	PC7 (AD7)	48	RESERVED
43	RESERVED	49	GROUND
44	RESET*	50	GROUND

A prototyping card was used to mount the circuitry. The board contains a grid of isolated, plated-through holes on 0.1" centers. It provides a 50-pin header for connecting the ADAPT-11™ microcontroller module. It has a +5V and GND bus signals around the periphery. Figure 3.25 shows the prototyping card. Then everything was enclosed in a plastic blue box. The only parts showing are: the LEDs, the RS232 port input, and the power supply plug. The LEDs were mounted on the wide side of the box to maximize their angle of transmission. Figure 3.26 shows the ATFUR hardware box.

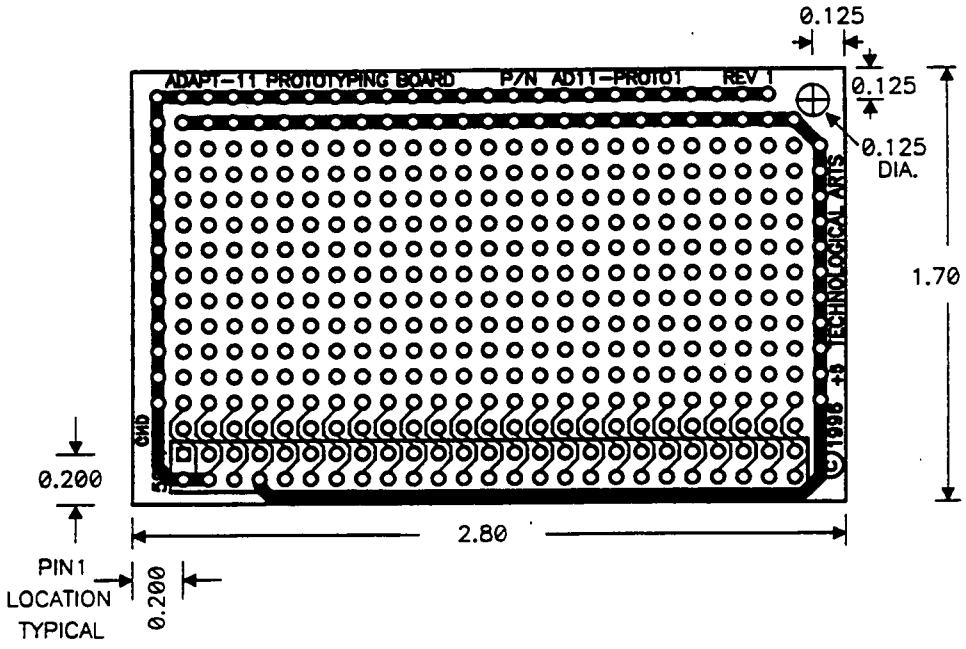


Figure 3.25 Prototyping Card

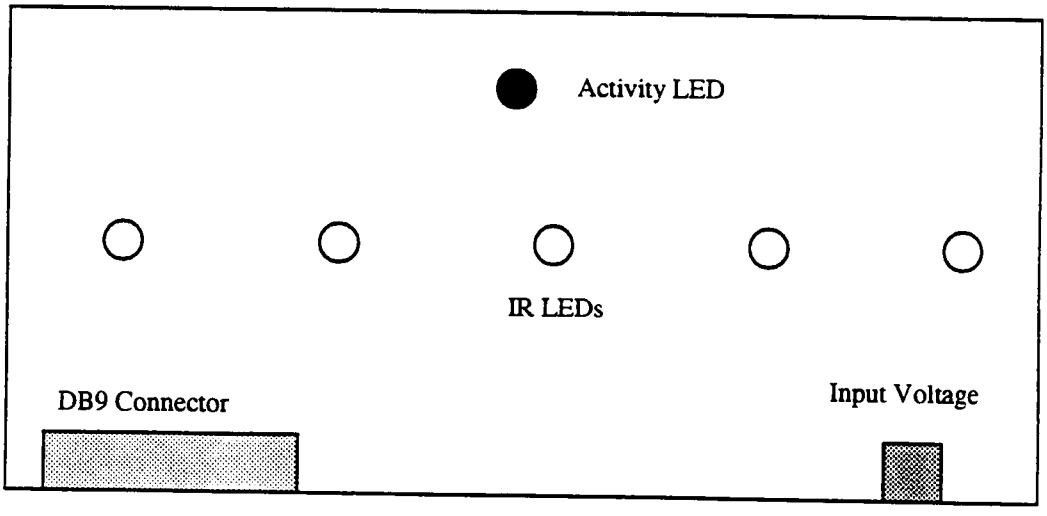


Figure 3.26 Hardware Box

3.5 IR TRANSMITTER

This section will discuss the output stage of the design, which is the IR transmitter. It consists of 6 IR diodes, 2 BJT transistors and a couple of resistors. The previous section handled all the information needed about these components.

Simulations using SPICE were conducted, to see values generated in order to make the output stage practically feasible. SPICE is a software environment that can implement an analog circuit simulation of any circuitry [26]. The first version was commercialized in 1971. Before the design can fully utilize SPICE, accurate device models must be available. Without device models of resistors, diodes, and BJTs, the simulator can not produce accurate results. The widespread acceptance of SPICE and the proliferation of SPICE-like circuit simulators have lead more and more component vendors to offer SPICE models of their own components. But the majority of component vendors still do not offer SPICE models [26]. For this reason, some devices used in this project were replaced by similar devices that have SPICE models available. Figure 3.27 shows The IR transmitter output stage. Figure 3.28 shows the SPICE program written to simulate the design. Charts of results are found in Figure 3.29.

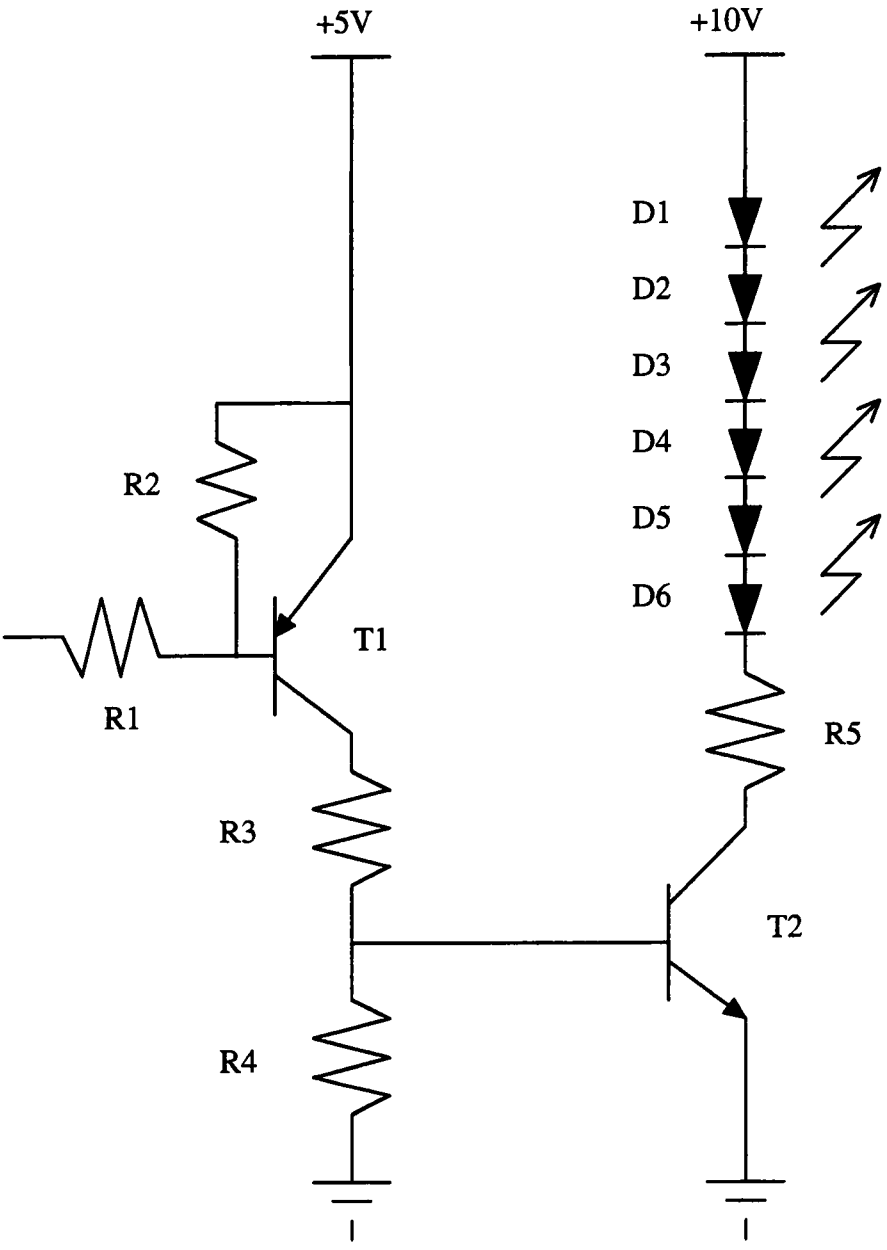


Figure 3.27 IR Transmitter

```
************ 03/13/96**********17:17:13*****************
*******************************************************
* This program simulates the circuit described in Figure 3.27 for
* the output stage of the remote control.
*******************************************************
* Circuit Description
.OP
.TF V(11) VIN1
.TF V(15) VIN2
.TRANS 20NS 1000NS
.PROBE
.LIB
VIN1 11 0 5V
VIN2 15 0 10V
Q1 9 10 11 Q2N2907A
Q2 7 8 0 Q2N2222
R1 0 10 3.3K
R2 10 11 47K
R3 9 8 68
R4 8 0 47K
R5 6 7 3
D1 15 1 D1N4148
D2 1 2 D1N4148
D3 2 3 D1N4148
D4 3 4 D1N4148
D5 4 5 D1N4148
D6 5 6 D1N4148
.END
*******************************************************
```

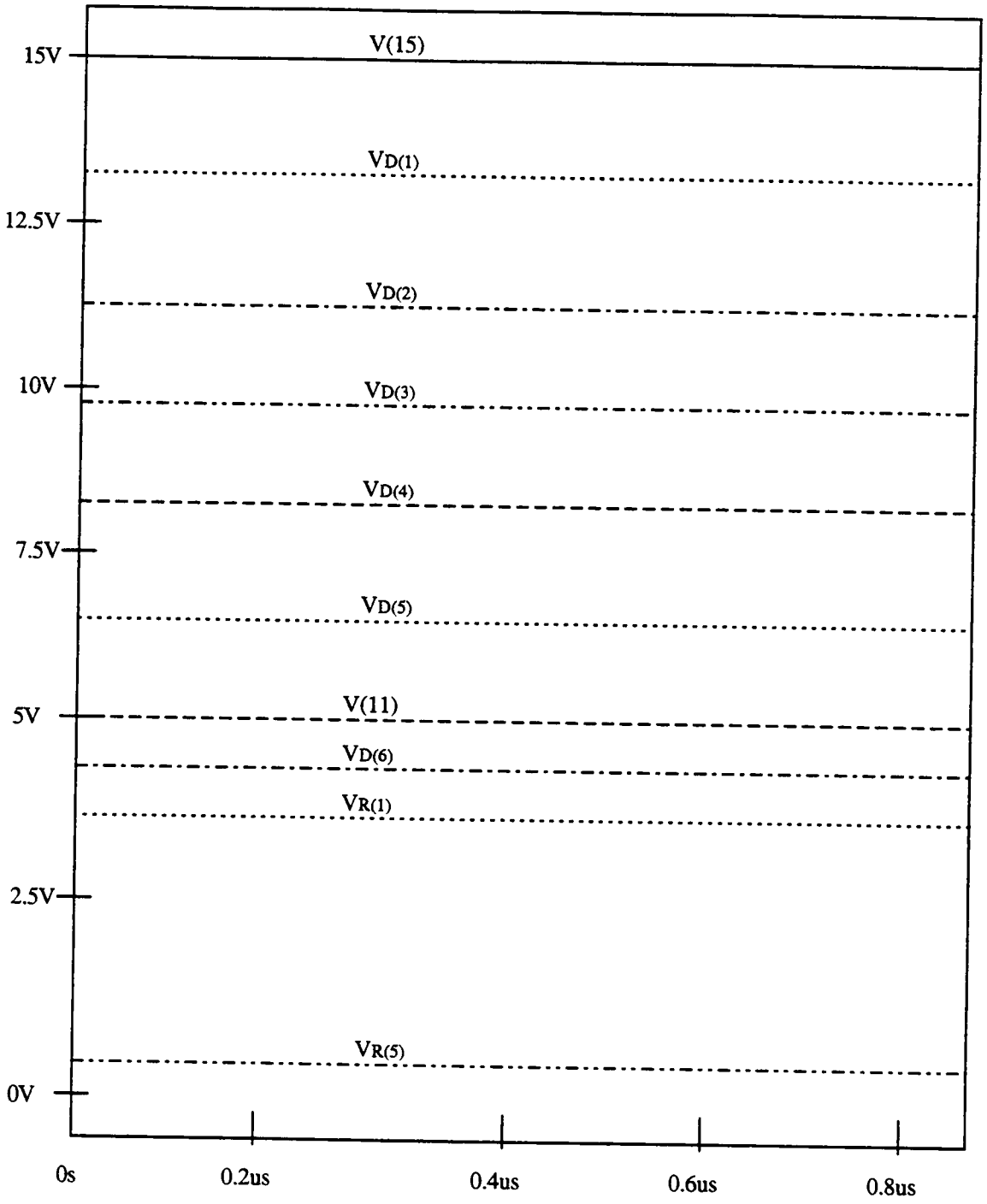


Figure 3.29 Voltage Graphs

The 47KΩ resistors are used to turn the transistors on and off whenever signals come from the MC. One is a pull-up resistor, the other is a pull-down resistor. The 3Ω resistor is a safety, if the current goes higher than what it is supposed to, the resistor will prevent the current from burning out other components. This way the LEDs are protected from excessive current. The 3Ω value was obtained from the following equation:

$$Rs = (\text{Supply Voltage} - \text{LED Voltage}) / \text{LED Current} \qquad (3\text{-}6)$$

The emitter of T2 is connected straight to ground to bring the base current up resulting in a higher current going through the collector and flowing to the diodes. The most important criteria was to make the current flowing through R1 as small as possible without affecting the throughput of the output stage. The MC can sink up to 1.6 mA current, the actual current of 1.3 mA was below this value, which means that the MC will not be damaged. The same exact circuit was built to test its operation. The resistor R1 was connected to ground because of the assumption that the MC is outputting a low signal. The 110 VAC power supply was connected to a load, the readings on a multimeter gave an output voltage of 15V and an output current of 750 mA. The readings had been taken twice for more accuracy and the results of each measurement matched. A 3Ω (4W) resistor was put in series with 6 IR diodes and a constant current of 600 mA was supplied constantly to the circuit. After a while the resistor overheated and destructed causing an

open circuit. This resistor was used to set the current flowing through the LEDs.

The next step was to test if under any circumstances the LEDs will overheat and cause a fire. A fuse box containing a 1A cutoff fuse replaced the 3Ω resistor. A current of 0.6A was supplied to the circuit. Evidently, the fuse did not blow since it can handle higher currents. The test lasted for about 30 minutes before one of the LEDs destructed causing an open circuit. The test was successful because none of the LEDs actually burst into flame. The test was repeated three times to make sure that no fire would take place. High current was supplied to see how components behave under stress, the current used in the application was only 300mA. In normal operation, when the transistor T2 is saturated the current flowing through the diodes will be only 150mA.

After testing for safety, the voltage across each node was measured to make sure that theoretical results held true and the system would operate as designed. For a PNP transistor the following equations were used to calaculate the currents:

$$I_C = \beta I_B \qquad (3\text{-}7)$$

$$I_E = \alpha I_C \qquad (3\text{-}8)$$

$$\alpha = \beta + 1 \qquad (3\text{-}9)$$

where I_C is the collector current, I_B is the base current and I_E is the emitter current. The beta (β) of T1 is around 50 and the beta of T2 is

around 15. Table 3.11 shows the comparison between theoretical and experimental results of voltages taken at the terminals of each device.

The percentage error was expected and was due to the following factors. One factor is the components themselves; these analog devices already have an error margin in their operation. While taking measurements, the voltage and current were supplied for a very short time; still this may have heated up some components which caused the different results. In general the error margin was acceptable and was within the expected range.

This chapter was a brief description of each component used in this project starting with analog devices and ending with the IR output stage. This chapter will make it easier to understand the operation of the hardware part in Chapter 5. The safety test conducted under stress conditions was performed in the presence of an engineer that verified the safety of the ATFUR devices.

Table 3.11 Theoretical Vs. Measured Values

VOLTAGE	THEOR.	MEAS.	ERROR %
V_{IN1}	15.0	14.60	2.66
V_{IN2}	5.00	4.90	2.00
V_{D1}	1.70	1.69	0.00
V_{D2}	1.70	1.60	5.88
V_{D3}	1.70	1.67	1.18
V_{D4}	1.70	1.68	1.18
V_{D5}	1.70	1.72	1.18
V_{D6}	1.70	1.65	2.94
$V_{6.6\Omega}$	4.20	4.40	4.76
V_{C2}	0.10	0.12	20.00
V_{E2}	0.00	0.00	0.00
V_{B2}	0.80	0.80	0.00
V_{C1}	4.00	4.20	5.00
V_{B1}	4.30	4.20	2.32
V_{E1}	5.00	5.00	0.00
$V_{3.3K\Omega}$	4.29	4.20	2.10
$V_{47K\Omega}$	0.97	0.75	7.14

CHAPTER 4

LabVIEW SOFTWARE DEVELOPMENT

Each computer application is a problem which must be solved. The solution requires a certain hardware configuration and certain programs to carry out necessary tasks [24]. The first step in writing a program is to analyze the problem that the software is intended to solve. This requires a careful consideration of the various inputs and the exact outputs required. Therefore, the programmer must come up with an proper and efficient algorithm that make the task work. Once the algorithm has been defined, the program must be developed. A flowchart will present the program in a clear, detailed manner. After the program has been described, it must be written out in a programming language; LabVIEW in this case. After finishing the coded program, it should be run and tested.

This chapter will discuss in detail the ATFUR software development. Starting from defining the specification of the drivers, and ending with running and testing them. The development method of this program was done according to the IVP methodology described in Chapter 2. This chapter will also provide a description of each major VI used to implement the overall driver. It will also give a general idea of the operation of each subVI. The first sections will cover the IR format, time table, code table, and TV header. The last two sections

will cover the automated and manual drivers, their description and operation.

4.1 IR Format

All remote controls that use IR to transmit signals follow the same principle. The signals are binary coded and vary in length, time wise and also bit wise. These signals are modulated with carrier frequencies around the 40 kHz range. The reason behind this modulation is to distinguish the signal from the noise present in ambient light. A well known source of noise are lamps, especially fluorescent ones.

The first issue encountered in this project was how to get written and accurate formats of the signals sent by remotes. Unfortunately, there is no single standard format that television manufacturers use. Each company has its own codes and formats. Even though companies are trying to agree on one international standard format, none is available yet. So far, there are at least two international standards which are used by remote controls to encode commands, the RC5 and the RECS80 codes.

The RC5 code has a uniform duration for all bits. There is always a transition in the middle of the time interval assigned to each bit. A binary "0" is encoded by a high to low transition and a binary "1" by a low to high transition. Figure 4.1 shows an example of a "0" and a "1". The transition time from high to low is the same as the transition from low to high. Therefore the bit time of a "0" is equal to the bit time of a "1".



Figure 4.1 Logical "0" and "1" Structure

The RECS80 code uses pulse length modulation. Each bit to be transmitted is encoded by a high level of duration T followed by a low level of duration 2T representing a logical "0" or 3T representing a logical "1". Figure 4.2 illustrates an example of a 010.

There are many other techniques used by famous brands. Panasonic uses a space coded signal which varies the length of the spaces. Philips uses a shift coded signal that alters the order of pulse spaces .

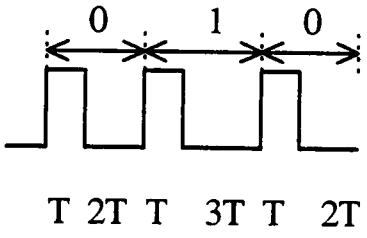


Figure 4.2 Example of RECS80 code

Since no company makes its codes available to the public, it is hard to get accurate codes. The best way to decode signals from the infrared control unit is to digitize and record them. The data could then be used directly or converted to a more easily interpreted form. The codes can be viewed with a photodiode connected directly to the emitter leads in the remote control unit.

Another way is to use an oscilloscope and record the codes manually. The user should hook up the remote control that he is trying to learn the codes from to an oscilloscope. Then push on a command button and record the signal displayed by the oscilloscope. Some remote controls send the code only once, therefore it is necessary to

repeatedly press each button to view the code.

Codes are generally divided into three parts, a header, an address and a command. The header is in general made of 2 bits and initializes the start of a sequence. The address selects which equipment to send to, and the command tells the equipment what to do. Each code is a long sequence, therefore the oscilloscope should be set properly in order to see the whole signal on the display screen. After creating a recording of the signals, an IR receiver such as GP1U52X IR receiver/demodulator module can be used. It detects 40 kHz modulated IR signals and produces a TTL or CMOS compatible signal [29]. This device should not be used to get the initial estimate of the signals, because the turn-on/turn-off times of this device are different and vary with signal strength, resulting in a distorted and inaccurate output waveform. However, if the estimated codes are provided by means of an oscilloscope, they can be easily recognized and verified by using the GP1U52X.

In this application, the codes for several televisions were supplied in binary format. A reconstruction of the signals was needed to configure the times and sequences needed to be sent to the MC. These formats were obtained by the second method described above and can be found in Appendix C. Codes were reconstructed from the available data. Each signal starts with a burst and ends with a delay. The time that a burst lasts was measured, then the time of the delay was

measured and so forth. After getting all of the burst/delay times, a time table was constructed. The fact that many bursts/delays have equal duration, narrowed the time table down to a maximum of 5 unique time constants. A numeric number was given to each time constant. For example time 1 is equal to *x* ms and time 2 is equal to *y* ms. Then, each burst/delay was assigned the number of its duration. For example, for a NEC television NAP model, a logical "1" is defined in the form shown in Figure 4.3.

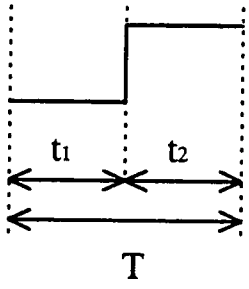


Figure 4.3 NEC Logical "1"

where t_1 = 2.5 ms and t_2 = 0.4 m. A logical "0" is defined by the form shown in Figure 4.4, where t_0 = 2.9 ms.

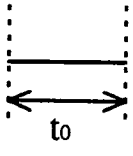


Figure 4.4 NEC Logical "0"

The corresponding time table will be:

→ time 1 = $t_0 + t_1$ = 3.3 ms for a "01"

→ time 2 = t_2 = 2.5 ms for a "0"

→ time 3 = $t_0 + t_0$ = 5.8 ms for a "00"

Those are the only combinations that can be found in any command sequence for this particular model. Examining the VOLUME UP command sequence, it is: 0101010101. The corresponding data structure looks like the signal in Figure 4.5.

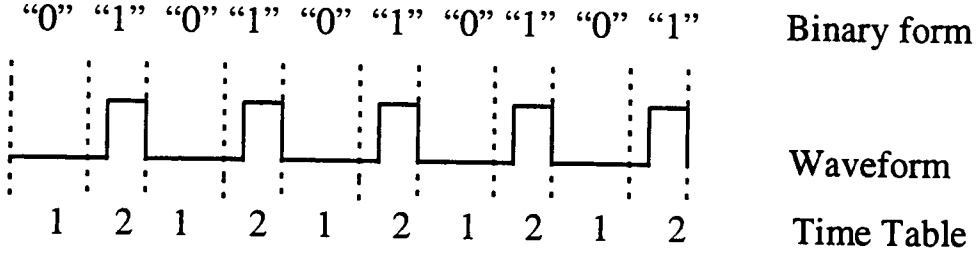


Figure 4.5 Volume Up Waveform

Most codes starts with a header and a custom part. The custom part ends sometimes with a burst which is why the first bit in this example is a delay and not a burst. The command that is sent to the MC is in the following format: 3.3 2.5 : 1212121212; (this is only an example). The 3.3 and 2.5 are time 1 and time 2 respectively. The "1" is a delay followed by "2" which is a burst followed by "1" which is a delay and so forth. The first delay will last time 1, the first burst will last time 2

and the second delay will last time 1 and so forth. More details about this are supplied in Section 4.2. Reconstruction of the command structure is a very long and precise process. It is very easy to get confused and make mistakes in translating the codes from the binary format to a time table. Figure 4.6 shows all the binary sequences for a particular TV model. Figure 4.7 shows 3 commands customized to the desired format.

4.2 TIME TABLE

National Instruments (NI) defines a Global Variable (GV) as a non-reentrant subVI with local memory that uses an uninitialized shift register to store data from one execution to the next. The memory associated with copies of these subVIs is shared and thus can be used to pass global data between them [28]. Each user can create his own global variable with front panel controls that suits only his needs. It can be read at any time by single or multiple users. All VIs calling it will access the same data. This includes multiple copies on the same diagram and communications between top-level VIs [28]. A GV can store multiple elements of different types. This way the designer can combine arrays with elements and clusters.

Brand Name: NEC Model: NAP

Carrier Information: Freq.: 37500 Hz Duty Cycle: 33% (time ON)

There is a sync pulse (head pulse) that repeats for each transmission.
Length of sync burst 9.00 ms
Length of delay after sync burst 4.50 ms

On-Time for a 0 :0.55 ms
Delay time for a 0 :1.70 ms

On-Time for a 1 :0.55 ms
Delay time for a 1 :0.55 ms

Total number of bits is 24. Custom: 16 Command: 8

Data Structure: Head Pulse + Custom Code + Data + $\overline{\text{Data}}$

Output Codes

Custom Code 1101111100100000

Power	11101111	Key 0	11110111	Cursor up	not valid
Vol Up	10111111	Key 1	01110111	Sleep	not valid
Vol Dn	00111111	Key 2	10110111	Ch Guide	00100111
Ch Up	11111111	Key 3	00110111	PIP	not valid
Ch Dn	01111111	Key 4	11010111	PIP Pos	not valid
Mute	10101111	Key 5	01010111	Swap	not valid
Alt Ch	10100111	Key 6	10010111	Freeze	not valid
Plus	11011111	Key 7	00010111	Preview	not valid
Minus	01101111	Key 8	11100111	Strobe	not valid
Cur Dn	00101111	Key 9	01100111	Menu	10100111

Figure 4.6 Binary Codes for a TV

Brand Name: NEC Model: NAP

Frequency: 37.5 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333433333333334343334343434343

Commands: Data + Data

Power: 33333334333333333434343334343434

Vol Up: 33343333333333333433343434343434

Vol Dn: 34343333333333333333343434343434

Chan Up: 33333333333333333434343434343434

Chan Dn: 34333333333333333334343434343434

Mute: 33343334333333334333343334343434

Alt Chan: 333433343433333334333433333343434

Plus: 33333433333333333434333434343434

Minus: 343333343333333333343433334343434

Cur Dn: 34343334333333333333433334343434

Key 0: 33333333433333334343433343343434

Figure 4.7 Customed Commands for a TV

Key 1: 34333333343333333334343343334343434

Key 2: 33343333343333333433343343334343434

Key 3: 34343333343333333333343343334343434

Key 4: 33333433343333333434333433343434

Key 5: 34333433343333333334333433343434

Key 6: 33343433343333333433333433343434

Key 7: 34343433343333333333333433343434

Key 8: 33333334343333333434343333343434

Key 9: 34333334343333333334343333343434

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: 34343334343333332222232222232323

PIP: Not Assigned

PIP Pos: 33343434333333332322222223232323

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: 33343334343333333433343333343434

Figure 4.7 (continued) Customed Commands for a TV.

The Time Table is a GV that consists of a table and two one-dimensional arrays as seen in Figure 4.8. The horizontal array corresponds to the rows of the table. It contains 16 indicators that list the TV models. The vertical array corresponds to the columns of the table. It contains 4 different indicators. The first indicator gives the carrier frequency. This frequency is different for each television model; some TVs operate on 37.5 kHz others on 40 kHz and so on. The second indicator gives the time sequence. It is made out of all times needed to be sent to the MC. Each time is represented by 6 characters. For example, the time sequence for a UR-16 model is 00.88 90 1.778 88.400. The first six characters correspond to time 1 which is equal to 0.889 ms, the next six characters correspond to time 2 which is equal to 1.778 ms, and the last six characters correspond to time 3 which is equal to 88.4 ms. Times 1, 2 and 3 are combined together to construct the time sequence for this particular model. The times are different for each model. The third indicator gives the custom code. The custom code is a combination of a binary "0" and "1", or in the IR language "burst" and "delay". The custom code is like a key to a door lock. To operate a television, its custom code should be sent first, so the TV will know that it is being addressed.

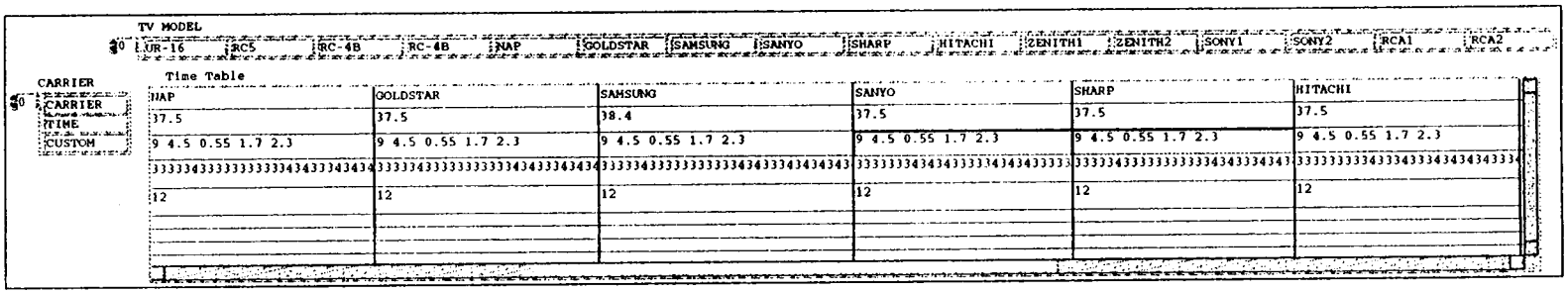


Figure 4.8 Time Table GV

Some televisions do not have custom codes, therefore the corresponding cell in the table is empty. The last indicator gives the TV header. The header is a short signal that initiates the addressed television. This is a kind of wake up signal to the TV used before sending the command sequence. Usually the header is formed by a burst followed by a delay.

More TV models can be entered into this GV as needed. Both the manual and automatic driver VIs use the time table GV. The manner in which this is done will be discussed later. Note that a scroll bar is used in the table, so it will be easier to navigate back and forth when entering the data. Each time new data is entered or any old data has been modified, the user should click on "make current values default", then save the VI. Otherwise the data will be lost. All of the information contained in this GV can be found in Appendix C Figure 4.9 shows the time table GV.

4.3 CODE TABLE

The Code Table is another global variable that is called by both drivers. It consists of three major components: a table and two one-dimensional arrays. The horizontal array contains television models and corresponds to the first row of the table. It contains 16 indicators; each corresponds to a unique television model. The vertical array

contains television commands and corresponds to the first column in the table. It contains 31 indicators; each corresponds to a unique television command. Not all commands are available for all models. Table 4.1 shows the availability of the command for each model.

The code table is similar to a two-dimensional array, where the rows are the commands and the columns are the models as seen in Figure 4.9. The first row and column contain names only to make it easier for the user to identify each cell. The cells are filled with the corresponding codes derived from Section 4.1. An empty cell means that the corresponding command is not available for that model.

The TV model array has 16 indices corresponding to 16 models. Each index corresponds to the location of a model. For example, index "1" corresponds to the RCA1 model and so forth. The up and down arrows are used to navigate through the models. The same thing applies for the commands array except that the index number will correspond to a command. For instance, index "2" corresponds to the VOLUME DOWN command. The table has a scroll bar in it and can hold as many as 100 models. It is used as a pointer table where a specific cell can be located by finding the intersection between the command and the model. This cell will contain the code that executes the command for this model. The array and the table can be expanded

Table 4.1 Availability of Commands for Models

	UR-16	RC5	RC-4B	RC-4A	NAP	GOLDST	SAMSUNG	SEARS
POWER	YES	YES	YES	YES	YES	YES	YES	YES
VOL UP	YES	YES	YES	YES	YES	YES	YES	YES
VOL DN	YES	YES	YES	YES	YES	YES	YES	YES
CH UP	YES	YES	YES	YES	YES	YES	YES	YES
CH DN	YES	YES	YES	YES	YES	YES	YES	YES
MUTE	YES	YES	YES	YES	YES	YES	YES	YES
ALT CH	YES	YES	YES	YES	YES	NO	NO	YES
PLUS	YES	YES	NO	NO	YES	NO	NO	YES
MINUS	YES	YES	NO	NO	YES	NO	NO	YES
CUR DN	YES	YES	YES	NO	YES	NO	NO	NO
0	YES	YES	YES	YES	YES	YES	YES	YES
1	YES	YES	YES	YES	YES	YES	YES	YES
2	YES	YES	YES	YES	YES	YES	YES	YES
3	YES	YES	YES	YES	YES	YES	YES	YES
4	YES	YES	YES	YES	YES	YES	YES	YES

Table 4.1 (continued) Availability of Commands for Models

5	YES	YES	YES	YES	YES	YES	YES	YES
6	YES	YES	YES	YES	YES	YES	YES	YES
7	YES	YES	YES	YES	YES	YES	YES	YES
8	YES	YES	YES	YES	YES	YES	YES	YES
9	YES	YES	YES	YES	YES	YES	YES	YES
MENU	YES	YES	YES	NO	YES	NO	NO	YES
CUR UP	YES	NO	YES	NO	NO	NO	NO	NO
SLEEP	YES	YES	YES	NO	NO	NO	NO	NO
CH GUI.	YES	YES	YES	NO	YES	NO	NO	YES
PIP	YES	YES	NO	NO	NO	NO	NO	YES
PIP POS	YES	YES	NO	NO	NO	NO	NO	YES
SWAP	YES	YES	NO	NO	NO	NO	NO	NO
FREEZE	YES	YES	NO	NO	NO	NO	NO	YES
PREVIEW	YES	YES	NO	NO	NO	NO	NO	NO
STROBE	YES	YES	NO	NO	NO	NO	NO	NO

Table 4.1 (continued) Availability of Commands for Models

	SHARP	HITACHI	ZENITH1	ZENITH2	SONY1	SONY2	RCA1	RCA2
POWER	YES	YES	YES	YES	YES	YES	YES	YES
VOL UP	YES	YES	YES	YES	YES	YES	YES	YES
VOL DN	YES	YES	YES	YES	YES	YES	YES	YES
CH UP	YES	YES	YES	YES	YES	YES	YES	YES
CH DN	YES	YES	YES	YES	YES	YES	YES	YES
MUTE	YES	YES	YES	YES	YES	YES	YES	YES
ALT CH	YES	YES	YES	YES	YES	YES	YES	YES
PLUS	NO	NO	YES	YES	YES	YES	YES	NO
MINUS	YES	NO	YES	YES	YES	YES	YES	NO
CUR DN	NO	YES	YES	YES	YES	YES	YES	NO
0	YES	YES	YES	YES	YES	YES	YES	YES
1	YES	YES	YES	YES	YES	YES	YES	YES
2	YES	YES	YES	YES	YES	YES	YES	YES
3	YES	YES	YES	YES	YES	YES	YES	YES
4	YES	YES	YES	YES	YES	YES	YES	YES

Table 4.1 (continued) Availability of Commands for Models

5	YES	YES	YES	YES	YES	YES	YES	YES
6	YES	YES	YES	YES	YES	YES	YES	YES
7	YES	YES	YES	YES	YES	YES	YES	YES
8	YES	YES	YES	YES	YES	YES	YES	YES
9	YES	YES	YES	YES	YES	YES	YES	YES
MENU	YES	NO	YES	YES	YES	YES	YES	NO
CUR UP	NO	NO	NO	NO	NO	NO	NO	NO
SLEEP	YES	NO	NO	NO	YES	NO	NO	NO
CH GUI.	YES	YES	YES	YES	YES	YES	YES	YES
PIP	YES	NO	YES	YES	YES	NO	YES	NO
PIP POS	NO	YES	YES	YES	NO	NO	NO	NO
SWAP	NO	NO	YES	YES	NO	YES	YES	NO
FREEZE	NO	NO	YES	YES	YES	YES	YES	NO
PREVIEW	NO	NO	NO	NO	NO	NO	NO	NO
STROBE	NO	NO	NO	NO	NO	NO	NO	NO

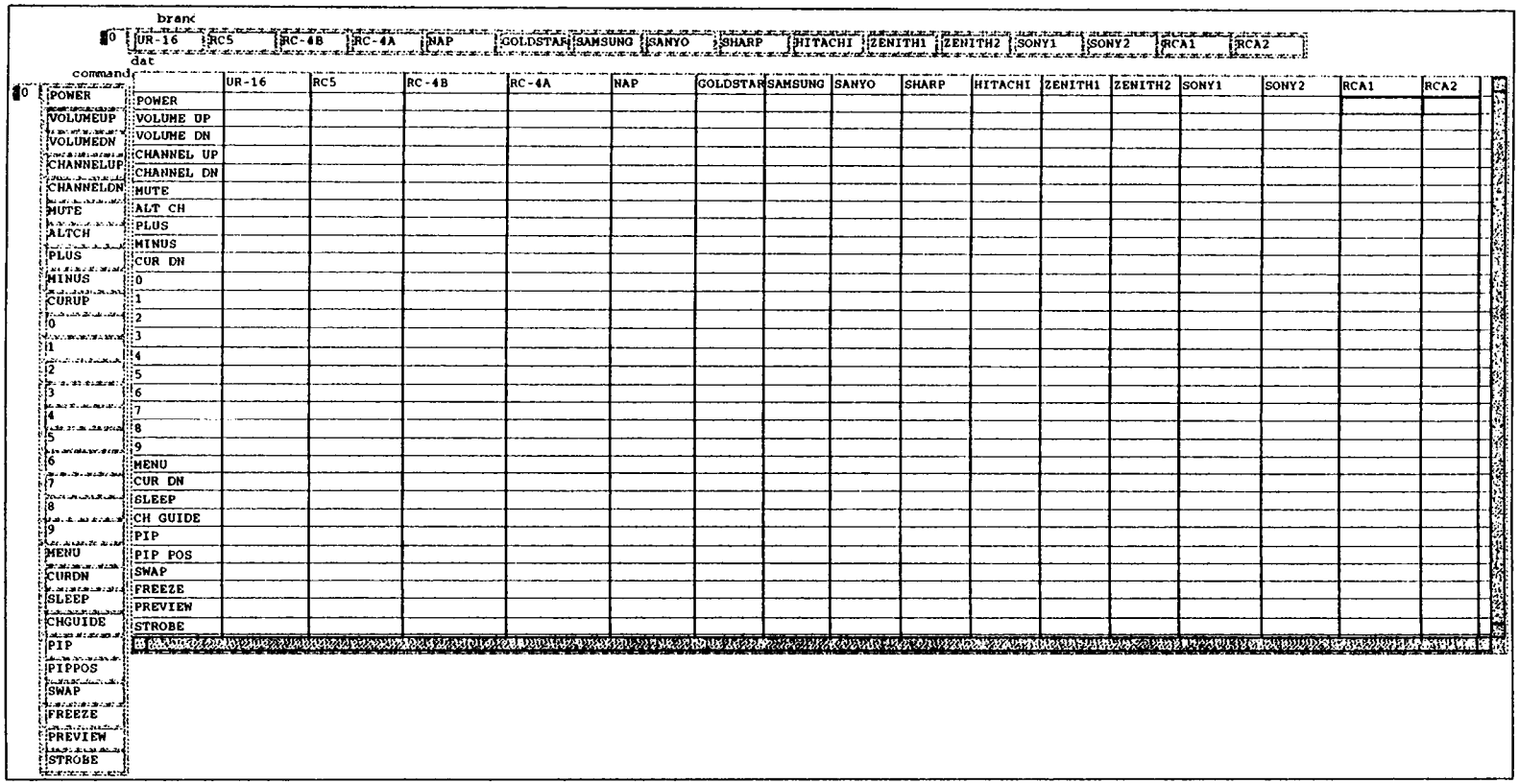


Figure 4.9 Code Table GV

and new models or commands can be added when needed. The user makes changes only in the GV, he does not need to modify anything in the drivers. That is what makes soft coding more suitable to use in such applications than hard coding. The code will stay the same and only the information in the table will change. After entering the data, the user should click on the table with the mouse's left button and choose "make current values default", then save the VI. Since the table is large, scroll bars were used to navigate back and forth to enter data; therefore not all the data will be shown in Figure 4.9.

4.4 TV HEADER

This function is implemented as a GV which is called by the automated driver only. It will supply the driver with the information needed to address the selected television. It consists of the TV brands which are listed in a menu ring that contains almost all the TV brands needed for testing. The brand is selected by pushing on the menu ring with the computer's mouse and highlighting the desired brand. The model is a string control that can be entered in ASCII format by the user. The model should be consistent with the brand or else, the driver will return a popup menu stating that this model is for a different TV brand. Table 4.2 shows the correspondence of a model to a brand. The screen size is a menu ring consisting of numerical numbers defining the television screen size in inches. It will not be used by the ATFURAD but it is needed to serve other tasks in the ATF project.

Table 4.2 Availability of Models for Brands.

TV BRAND	TV MODEL
MAGNAVOX1	UR-16 RC5
MAGNAVOX2	RC-4B RC-4A
NEC	NAP
GOLDSTAR	GOLDSTAR
SAMSUNG	SAMSUNG
SANYO	SANYO
SHARP	SHARP
HITACHI	HITACHI
ZENITH	ZENITH 1 ZENITH 2
SONY	SONY 1 SONY 2
RCA	RCA 1 RCA 2

The same applies to the version and to the serial number.

The TV Header functions are needed and called by other drivers for other applications. The outlet is a numeric number that indicates the number of the actual outlet to be used. This way the corresponding GPIB address will be sent by the driver. Only 32 outlets are available for use in the ATF facility; they are combined in a menu ring and are distributed in three rooms. All controls in the GV are found in a cluster that is contained in a one-dimensional array of clusters. This way each index of the array will correspond to only one TV set. For example, index "0" will contain the information needed for Sony, index "1" will contain the information needed for Magnavox and so forth. To go from one index to another, the user needs to push on the upper arrow to scroll up and on the lower arrow to scroll down. An estimated one hundred TV sets can be entered into this array. Each time new TV sets are entered into this GV, the user should choose "make current values default" from the "Operate" menu and then should chose "Save" from the "File" menu. Otherwise, all the changes will be lost and need to be entered again. Figure 4.10 shows the front panel of the TV header, and Figure 4.11 shows more details about the televisions under test. It contains the corresponding frequencies, models, availability of headers and custom codes, and time tables for each brand.

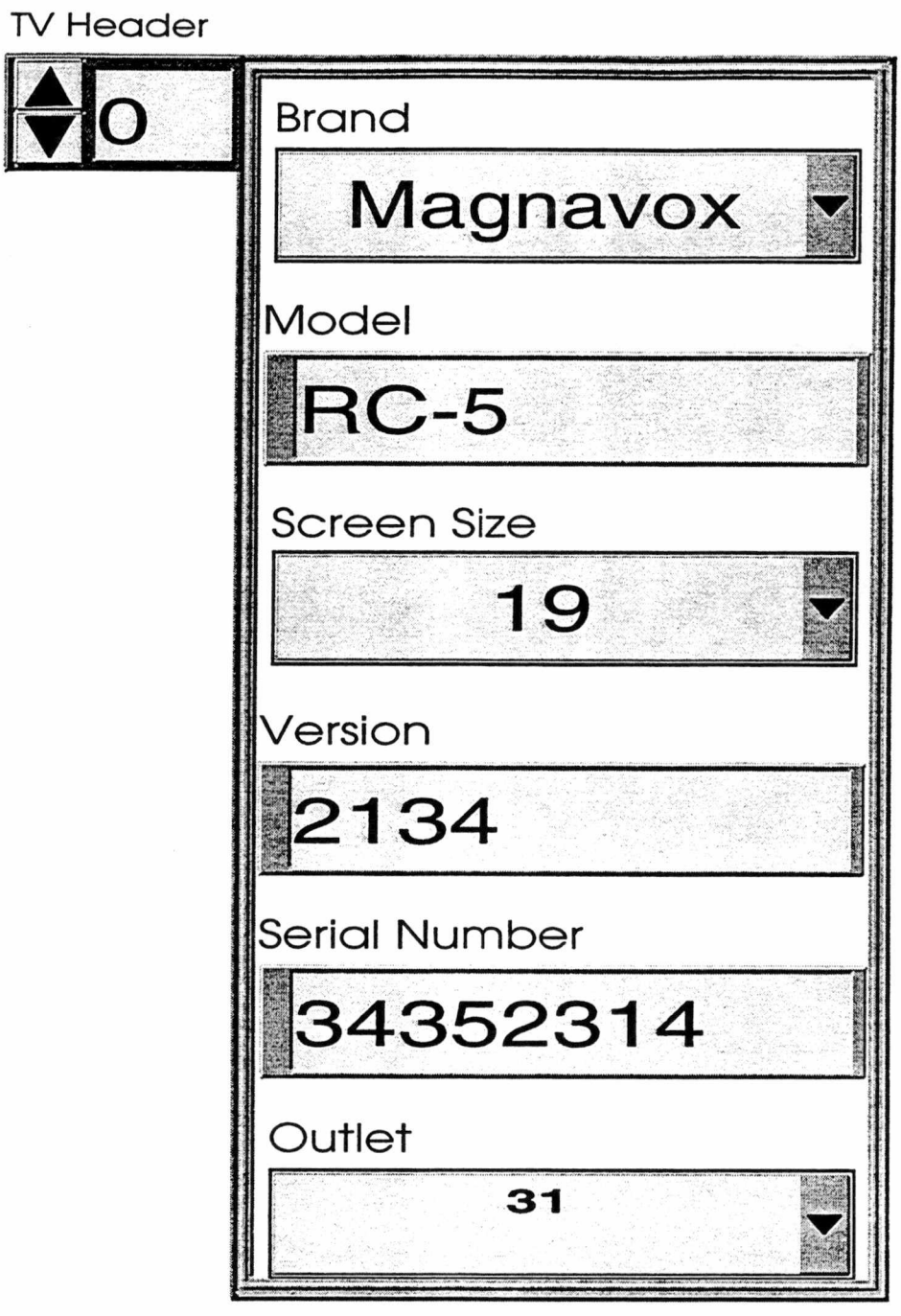


Figure 4.10 TV Header GV

BRAND	FREQUENCY (kHz)	MODEL	HEADER	CUSTOM CODE	TIMES (ms)
MAGNAVOX 1	35.7	UR-16	YES	YES	T1 = 1.778 T2 = 0.889 T3 = 88.0
MAGNAVOX 1	35.7	RC-5	YES	YES	T1 = 1.778 T2 = 0.889 T3 = 88.0
MAGNAVOX 2	45.5	RC-4B	NO	YES	T1 = 2.5 T2 = 0.4 T3 = 2.9 T4 = 5.4 T5 = 5.8 T6 = 25.0
MAGNAVOX 2	45.5	RC-4A	NO	YES	T1 = 2.5 T2 = 0.4 T3 = 2.9 T4 = 5.4 T5 = 5.8 T6 = 25.0

Figure 4.11 TVs Under Test

NEC	37.5	NAP	YES	YES	T1 = 9.0 T2 = 4.5 T3 = 0.55 T4 = 1.7
GOLDSTAR	37.5	GOLDSTAR	YES	YES	T1 = 9.0 T2 = 4.5 T3 = 0.55 T4 = 1.7
SAMSUNG	38.4	SAMSUNG	NO	NO	T1 = 1.1 T2 = 2.2 T3 = 3.3 T4 = 4.4 T5 = 27.0
SANYO	37.5	SANYO	YES	YES	T1 = 2.5.0 T2 = 0.4 T3 = 2.9 T4 = 5.4 T5 = 5.8 T6 = 25.0
SHARP	37.5	SHARP	YES	YES	T1 = 2.5 T2 = 0.4 T3 = 2.9 T4 = 5.4 T5 = 5.8

Figure 4.11 (continued) TVs Under Test

HITACHI	37.5	HITACHI	YES	YES	T1 = 2.5 T2 = 0.4 T3 = 2.9 T4 = 5.4 T5 = 5.8 T6 = 25.0
ZENITH	40.0	ZENITH	NO	NO	T1 = 0.5 T2 = 5.0 T3 = 4.0 T4 = 95.0
SONY	40.0	SONY 1	YES	YES	T1 = 2.4 T2 = 0.6 T3 = 1.2 T4 = 25.0
SONY	40.0	SONY 2	NO	NO	T1 = 0.5 T2 = 1.5 T3 = 3.5 T4 = 36.0

Figure 4.11 (continued) TVs Under Test

4.5 MANUAL DRIVER

Since it is cumbersome to hold multiple remote controls in one hand to operate different TV sets, a one-unit remote control solution was needed. The idea of a universal remote that is a multi-function IR remote control transmitter and utilizes around 30 command buttons driven by software manifested itself. All existing remotes will be replaced by the Automatic Test Facility Universal Remote Manual Driver (ATFURMD). It is a LabVIEW VI that controls all available televisions from a Sun workstation. After learning about the functions, the operating mode, and obtaining data for different brands, the design of the ATFURMD took place.

A LabVIEW instrument driver is a collection of VIs that control a programmable instrument [8]. Each routine handles a specific operation such as reading data, writing data, or configuring an instrument. A well written driver makes it easy to control an instrument. NI offers a free reference, Application Notes 006 *Developing a LabVIEW Instrument Driver.* This is a well organized programming guide. It gives an overview of the preferred way to write a driver from scratch. The driver can be as smart as the designer wants it to be. NI also provides Instrument Driver Templates that can be used in any application. Table 4.3 lists these templates.

Table 4.3 LabVIEW Instrument Driver Templates

PREFIX INIT	This is the first VI called when accessing an instrument driver. It configures the communications interface, manages, handles and sends a default setup command to the instrument.
PREFIX CLOSE	This is the last VI called when controlling an instrument driver. It terminates the software connection to the instrument and deallocates system resources.
PREFIX RESET	This VI places the instrument in its default state.
PREFIX SELF TEST	If an instrument has self-test capability, this VI will cause it to initiate that self-test and returns the result.
PREFIX REVISION QUERY	This VI outputs the revision of the driver and the firmware revision of the instrument. If the instrument firmware revision cannot be queried, the literal "Not Available" should be returned.
PREFIX ERROR QUERY	This VI queries the instrument and returns the instrument specific error information. This VI can only be used if the instrument has error query capability.
PREFIX ERROR MESSAGE	This VI translates the error status information received by the Error Query VI.
PREFIX MESSAGE BASED	This VI is the starting point for the development of the developer specified (GPIB) instrument driver VIs. This VI has all required instrument driver controls and instructions.
PREFIX REGISTER-BASED	This VI is the starting point for the development of the developer (VXI) instrument driver VIs. This VI has all required instruments driver controls and instructions.

Knowledge of how the instrument operates is required. It can be done by reading manuals about the instrument or simply experimenting with it. In this project, there was no available manual to be read, therefore a general idea was acquired by gathering pieces of information from several engineers knowledgeable about the operation of TV remote controls.

According to the IVP method, the BB was created first. It was inspired by Figure 4.12. The BB of this driver is shown in Figure 4.13. All Boolean controls labeled with different function names were gathered in a cluster. A cluster is conceptually the same as a "record" in Pascal or a "struct" in the C language [4]. Clusters are normally used to group related data elements that are used in multiple places on a diagram. They can contain either controls or indicators but not a mixture of both. In this application the cluster contains only controls that act the same as buttons on a regular remote control. Table 4.4 shows the functionality of each button. After defining the command buttons, two menu rings were used to identify the television brand. The menu ring labeled Brand is the primary selector, it needs to be selected first. The second menu ring is a programmable one. The Brand selector contains the available brands to be tested. When a brand is selected, the second menu ring will only display the available models or IR format for this particular brand. For example, if Sony is chosen from the primary menu ring, the second ring will only show Sony1 and Sony2

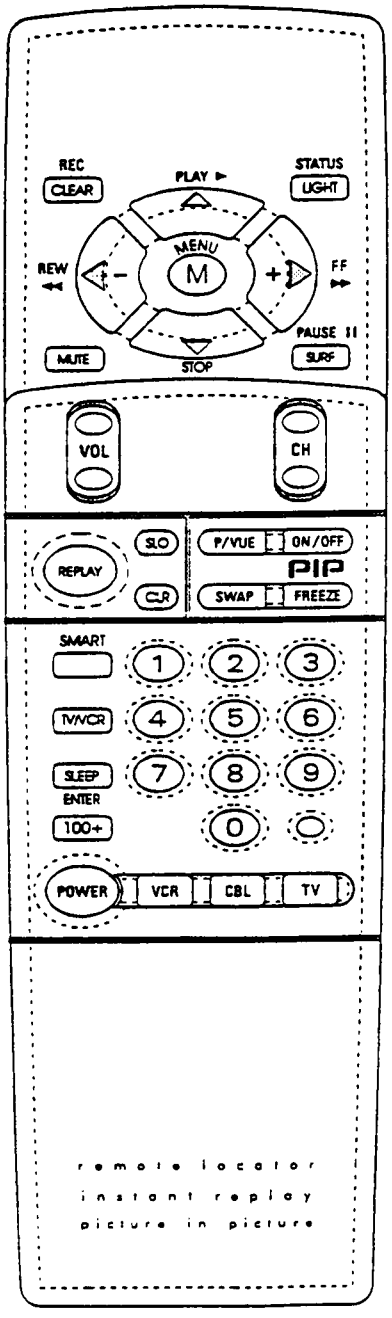


Figure 4.12 Universal Remote Control

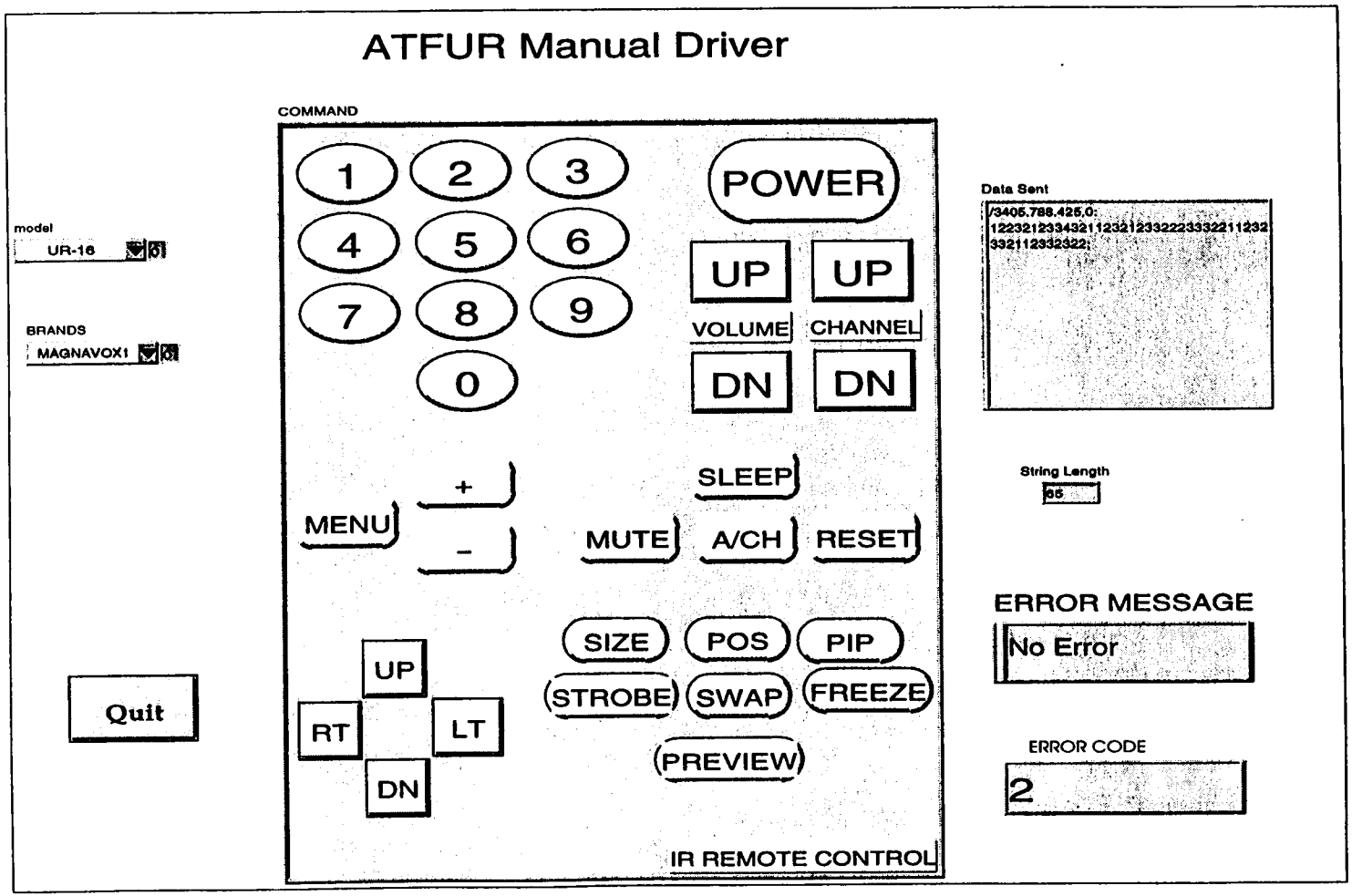


Figure 4.13 BB of the ATFURMD.

Table 4.4 ATFUR Commands

COMMAND	DESCRIPTION
Cursor Up	Press this button when in MENU mode to move the high-lighted cursor UP the menu.
Cursor Dn	Move the MENU highlight in a downward direction. Opposite direction to UP.
Minus Key	Press this button to decrement, toggle or sequence a high-lighted function on a TV menu page.
Plus Key	Press this button to increment, toggle or sequence a high-lighted function on a TV menu page.
Menu	Press this button to display a page of the menu. Each successive press will display the next menu.
Volume Up	Press this button to increase the TV volume.
Volume Dn	Press this button to decrease the TV volume.
Mute	Press this button to toggle the TV sound OFF or ON.
Alternate Ch	Press this button to toggle between the current and the last viewed channel displayed on TV.
Channel Up	Press this button to scan channels in ascending order.
Channel Dn	Press this button to scan channels in descending order.
Digit 0-9	Press the appropriate buttons to randomly access channels selected by the system selection keys.
PIP	Press this button to display PIP in one of 4 corners of TV screen.
Swap	Press this toggle button to swap main picture to PIP picture and back.

Table 4.4 (continued) ATFUR Commands

Freeze	Press this button to freeze and release the PIP picture.
Position	Press this button to move the PIP display to any one of the several predetermined PIP locations.
Strobe	Press this button to display several PIP captions of one channel at different time intervals.
Size	Press this button to select between one of several display sizes of the PIP picture.
Power	Press this button to toggle the power to the TV OFF or ON.
Left	Press this button to highlight a left selection in the menu mode.
Right	Press this button to highlight a right selection in the MENU mode.
Reset	Press this button to reset the TV to the factory conditions.

which are the available models for the specified brand. The user then can make a selection without being confused by matching the right format with the right brand. This automation was accomplished with the aid of a GV that contains a two-dimensional array. Each column of this array contains the available formats for only one TV brand. The primary menu ring will be an index for the array while the second index is disabled. When the "brand" menu ring sends the number 3 to the array, the array will respond by sending all models available in column #3. Figure 4.14 shows the programmed global variable.

In addition to these controls rings, a QUIT button was added to stop the execution of the VI at any time. When this button is pushed, it sends a boolean value of false that is complimented by the NOT function forcing the for-loop to end and causing the VI to stop.

On the right hand side, two indicators were added; one shows the total data sent to the MC and the other shows the length of this sequence. This way the user can identify whether the driver is sending data or not. It is not necessary to input the outlet number because this driver will be only used in the "Demo Room", therefore only one GPIB address is needed.

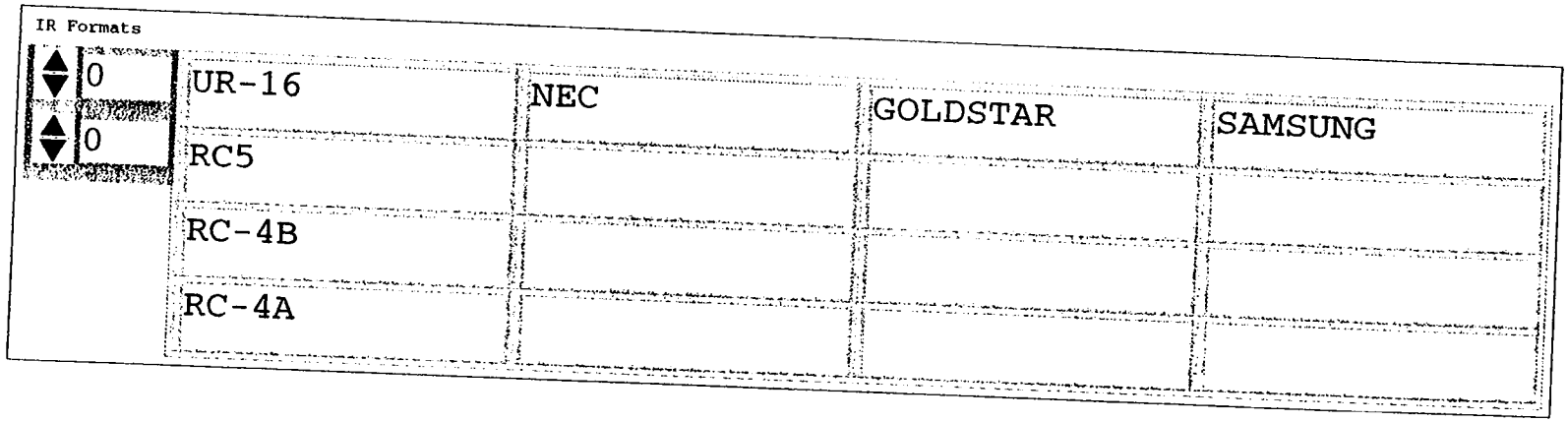


Figure 4.14 A Programmed GV

Since this driver is a software application, the possibility of generating unexpected errors should be considered. The error can be caused from several sources:

- A system configuration (unproper connections)
- A programming error
- A LabVIEW bug
- A human's input error (wrong data)

Fortunately, NI has developed some error handler drivers that tell the user that an error occurred, which instrument generated it, the meaning of the error, and what should be done to fix it. These error handlers give the user the option to terminate the application in which the error occurred, or simply pass it from the driver to the calling VI. Since the ATFURMD is writing to an instrument, the error input/output Cluster VI will be used. This error cluster is usually used by high-level I/O operations. Figure 4.15 shows the error VI. It is made up of a cluster that contains a boolean indicator which returns a false value when an error is detected, a digital indicator which is the error code, and a string which tell which VI generated the error.

Two error clusters appear on each side of the front panel. The one on the left is the error-in and the one on the right is the error-out. The error-in was not needed because this driver will be used by itself, it will not be a subVI for other drivers. The error-in cluster must be present for compatibility with the group conventions. The error-out will

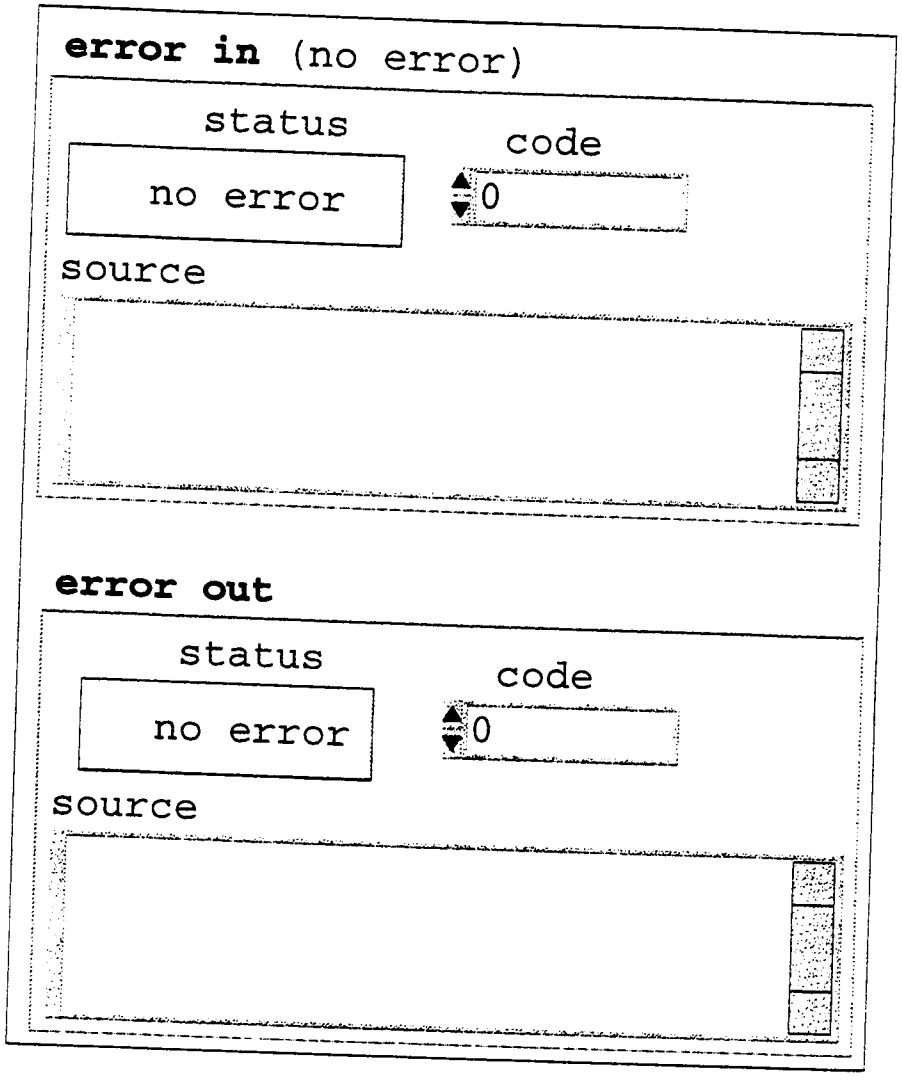


Figure 4.15 Error I/O Clusters

show any errors that this VI generates. These errors can be GPIB errors as well as other errors. Table 4.5 lists all the errors that this VI can generate.

The way the ATFURMD works is easy to follow; this driver is user friendly. The operator opens this driver on the Sun workstation and uses the mouse button to operate it. He must choose a TV brand from the primary menu ring first, then choose a model from the second menu ring. After the selection is made, the button POWER should be pushed to turn the TV on. After this particular button is pushed, the driver gives feedback to the user. A popup menu appears asking the user if the TV responded; the user can reply by pushing the button "yes" or "no". If the button "yes" is pushed this means that the driver is addressing the correct television and the user can start sending commands to the unit under test. If the button "no" is pushed, the driver advises the user to choose another model for the chosen brand. Then the identification process is repeated until the "yes" button is pushed. In addition, if the user sends a command that is not available for the chosen model a popup menu will appear stating that this command is not available for this model. By receiving feedback, the user will not get frustrated or confused when the TV does not respond.

Each control in the cluster is identified by a numeric number. When a button is pushed, a "false" value is returned and the number that created this "false" value is identified. Then it is incremented and

Table 4.5 ATFURMD Errors

ERROR CODE	ERROR NAME
6803	No Error
6804	Function requires GPIB to be CIC
6805	No listeners detected on write
6806	Controller not addressed correctly
6807	Invalid argument(s)
6808	Function requires GPIB to be SC
6809	I/O operation aborted
6810	Hardware not present
6811	DMA error
6812	DMA bus time-out
6813	New I/O attempted
6814	Function not implemented
6815	File system error
6816	Shareable board exclusively owned
6817	GPIB bus error
6818	Serial POLL Byte lost
6819	SRQ line can not be cleared
6820	Unrecognized command

sent as an index to the multiple GVs. The brand will also send an index to the GVs. The intersection of these two indices in each table will locate a cell, and the information contained in each of these cells will be concatenated, enclosed with a start and stop character, and sent to the MC via the GPIB subVI.

The CB of this driver is shown in Figure 4.16. Comments have been added to the driver to make it easier for other users to maintain and update it. As mentioned before, this VI acts exactly like a real universal remote. It keeps running continuously so that the user can send as many commands as he wants before stopping the VI. This driver saves the user a lot of time and effort when controlling multiple televisions. With a little bit of modification, this driver will be able to control VCRs and Cable TV boxes. More controls need to be installed in the cluster to add more functionality to this VI.

4.6 AUTOMATED DRIVER

This section will describe the content and the operation of the Automatic Test Facility Universal Remote Automated Driver (ATFURAD). This driver will be connected to other drivers in order to conduct a test. No user will be needed to operate it; it is fully automated.

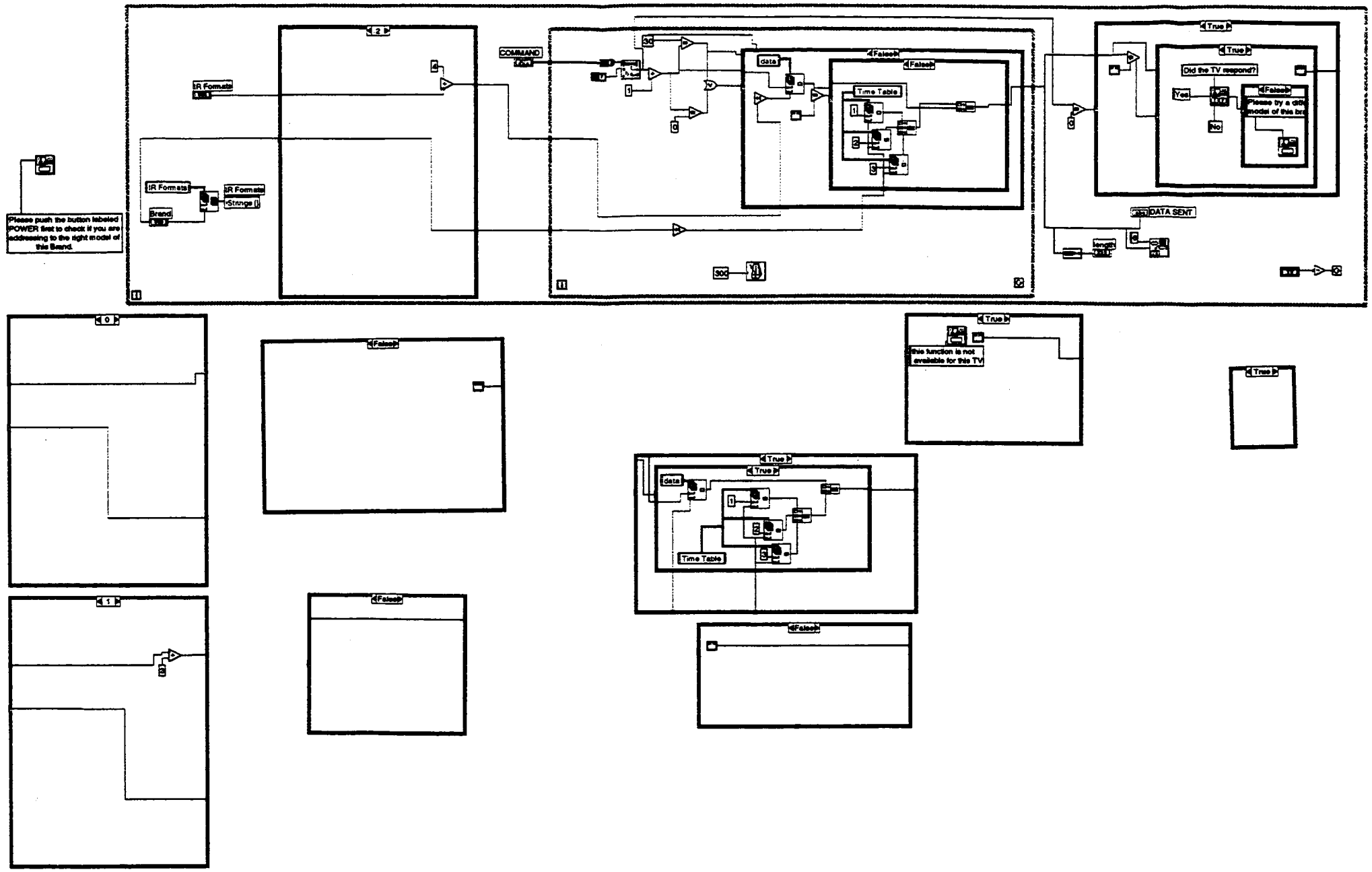


Figure 4.16 CB of the ATFURMD

The Black Box of the ATFURAD is found in Figure 4.17. It contains control inputs and indicator outputs. The inputs are ASCII string character controls which represent the command entered and a digital control which serves as an index to the TV global variable that has all the information needed to identify the television under test, such as TV brand, IR format, and outlet number. The outputs are a combination of string, array, and digital indicators. The array displays the Time Table sent to the MC; it contains the times in ms. One digital indicator shows the carrier frequency in kHz, another shows the length of the overall string sent to the MC. The rest are ASCII strings that display the location in which the ATFURAD will execute the commands, a header, a custom code, a command when available, and an indicator showing the total data sent to the hardware unit via the GPIB Bus. The data will be sent in the following order. The number of times the sequence should be repeated is sent first. In this case it is set to three times because some televisions cannot receive the sequence from the first time. The next six characters correspond to the carrier frequency. This is followed by a set of thirty characters maximum which represent times from the time table. The next two characters represent the header if applicable; some televisions do not have headers or custom codes. The next set is the custom code if available, followed by the command set if valid and ending with a ";" indicating the end of transmission. Refer to Table 4.1 for the availability of the command for a specific TV model. This VI calls three GV: the TV Header, Time Table and TV Codes.

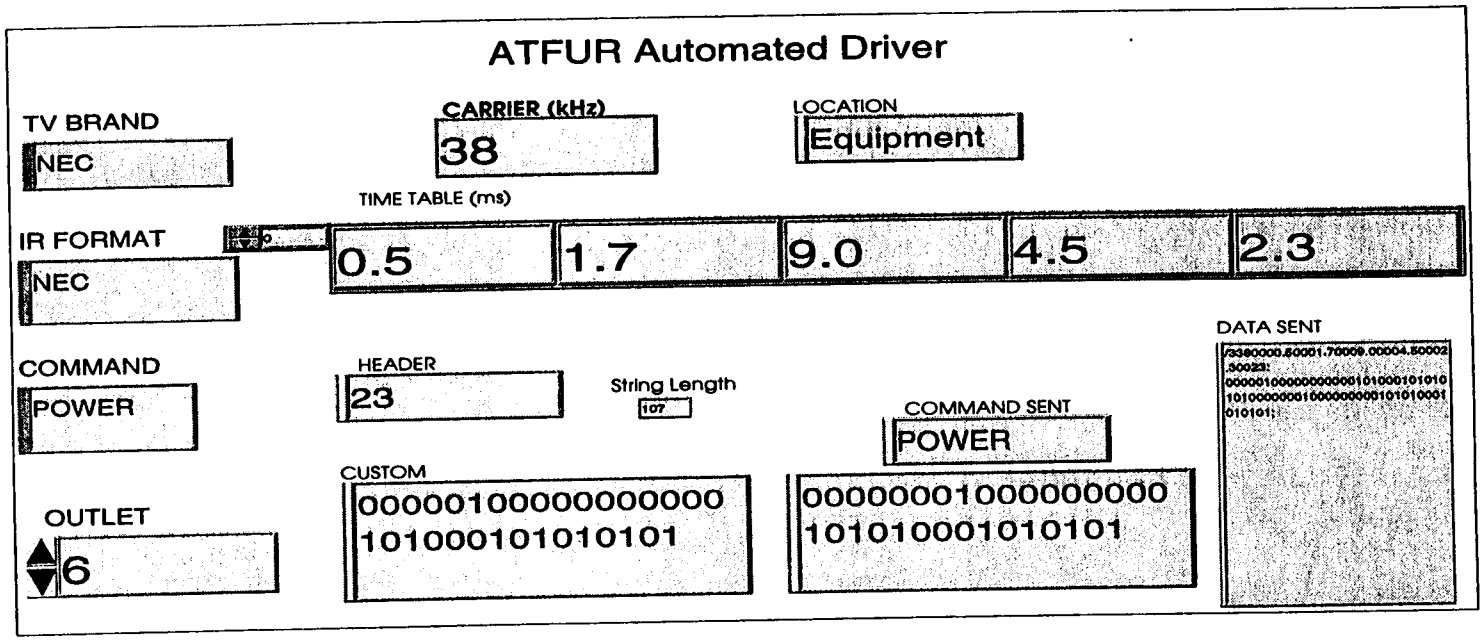


Figure 4.17 BB of the ATFURAD

The Clear Box of the ATFURAD consists of LabVIEW tools, such as cases, for-loops, array tools, and so forth. The data is sent from the software to the hardware with the aid of the GPIB driver developed by National Instruments and provided for LabVIEW users. ATFURAD will be talking to the MC through the traditional IEEE 488 GPIB bus. Figure 4.18 shows the CB of the ATFURAD.

The operation of this driver is easy to follow. As stated earlier, this VI will be a subVI for a test and will be connected to other VIs. The ATFURAD will receive an index number and a command. The command will enter a search loop in order for the VI to find its location from the code table. The command will be looked up from an array of commands. It will be compared to each element of this array, if not equal to the current element it will be compared to the next and so on until a match occurs. When a match occurs the number of iterations of the for-loop plus one (because the first column of the table is a title) will be the row index of the Time Table and the TV Code table. This number is the location of the command in each of those tables. The loop will stop after 30 iterations maximum because the size of the commands array is thirty, and the search does not need to start all over again. Therefore the loop will stop and the number "9999" will be sent as an index to the tables. This number will present a pop-up menu stating that this command is not available for this model. The index number will lead to the TV Header GV where all the information needed is found. The GV will be unbundeled by names.

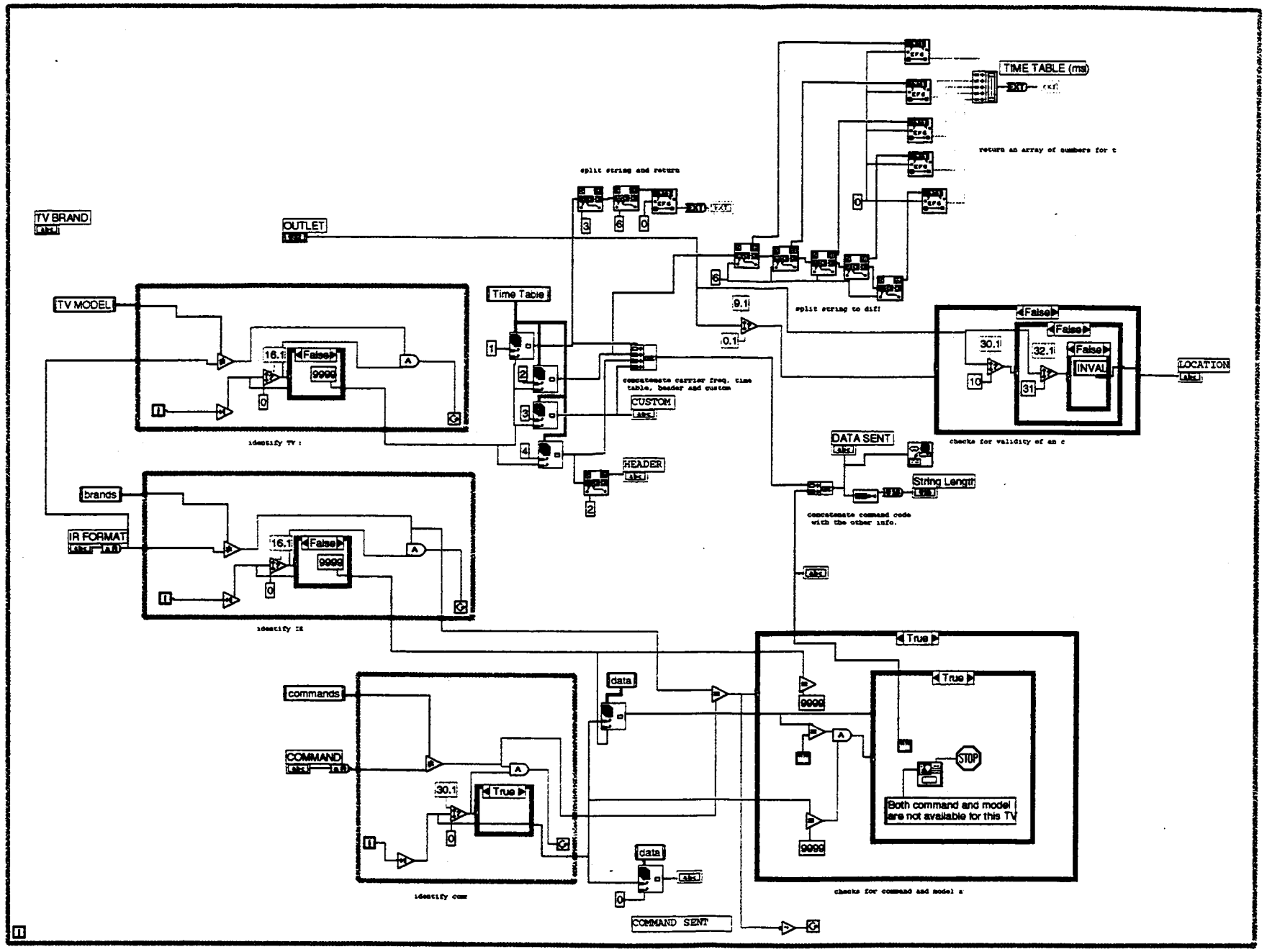


Figure 4.18 CB of the ATFURAD

The TV model and IR format will be compared to an array of TV models and IR formats respectively. The loop will stop when a match occurs, and the number of iterations plus one will be the column index for both tables. The search will last only for a maximum of 16 iterations maximum because this is the number of televisions entered in the table. When reaching the number 16 the search will stop and the number "9998" will be sent. This number will present a pop-up menu stating that this model is not available for this TV. After getting a warning message the VI will stop unless a new command or index number is given. The outlet number will correspond to the room location and the driver will send the corresponding address to the GPIB. In this way the ATFURAD will address only one hardware unit. If a number larger than 32 is given, a pop-up menu appears stating that this outlet number is invalid and no address will be given to the GPIB. Table 4.6 shows the distribution of the outlets. After the identification process, the row and column indices will locate the appropriate command code for the specified TV model and the correct frequency and time sequence. Then all of the data strings will be concatenated and sent to the GPIB.

In order for the user to see better what is being sent each data block has been formatted accordingly. For example the time table has been split to different parts, and each part can be viewed by itself. The exact codes from the tables has been sent to the bus without being affected by the formatting process. Each part of this driver has been

Table 4.6 Outlets Distribution

OUTLETS RANGE	LOCATION
1-8	DEMO ROOM
9-28	TEST ROOM
29-32	SCREEN ROOM

provided with the necessary description and information. This way the system operator will not have a difficult time modifying or adding changes to it.

This driver also contains two error clusters to indicate if any errors have been caused by other drivers or by this particular one. A detailed description of these error clusters has been presented in the previous section. The error-in cluster describes the error status before this VI executes. If error-in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error-out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error-out. The error-out cluster describes the error status after this VI executes. If an error occurred before this VI was called, error-out is the same as error in. Otherwise, error-out shows the error, if any, that occurred in

this VI. In addition to the errors listed in the previous section, this driver generates the following two errors:

6801 This command is not available for this model

6802 This model is not for this brand

This chapter has described the reconstruction of the IR formats, all global variables used in this application and both LabVIEW drivers that were used to control television units under test. The drivers were verified by the customer and the group members, comments have been made, and drivers have been certified for actual use in the ATF project.

CHAPTER 5

SYSTEM PERFORMANCE

This chapter will try to summarize the operation of the overall design. Figure 5.1 and appendix E show the schematic of the ATFUR project. In addition, this chapter will address the problems that may be encountered while using the ATFUR, and provides appropriate solutions.

5.1 OPERATION

The operation of the ATFUR starts by turning the Sun workstation on. The LabVIEW software should be started and the user has to call the ATFUR VIs. LabVIEW will load all the subVIs and functions present in the block diagram of the ATFUR driver. Next, the right GPIB addresses should be set on the GPIB and RS-232 boxes. The RS-232 box should be configured to work at 9600 baud rate, 8 bits, 1 stop bit, and no parity. Refer to Chapter 3 to learn how to set this environment. Finally, the power supply should be connected to the ATFUR box. The ATFUR is now ready to operate, and the user is ready to conduct a test.

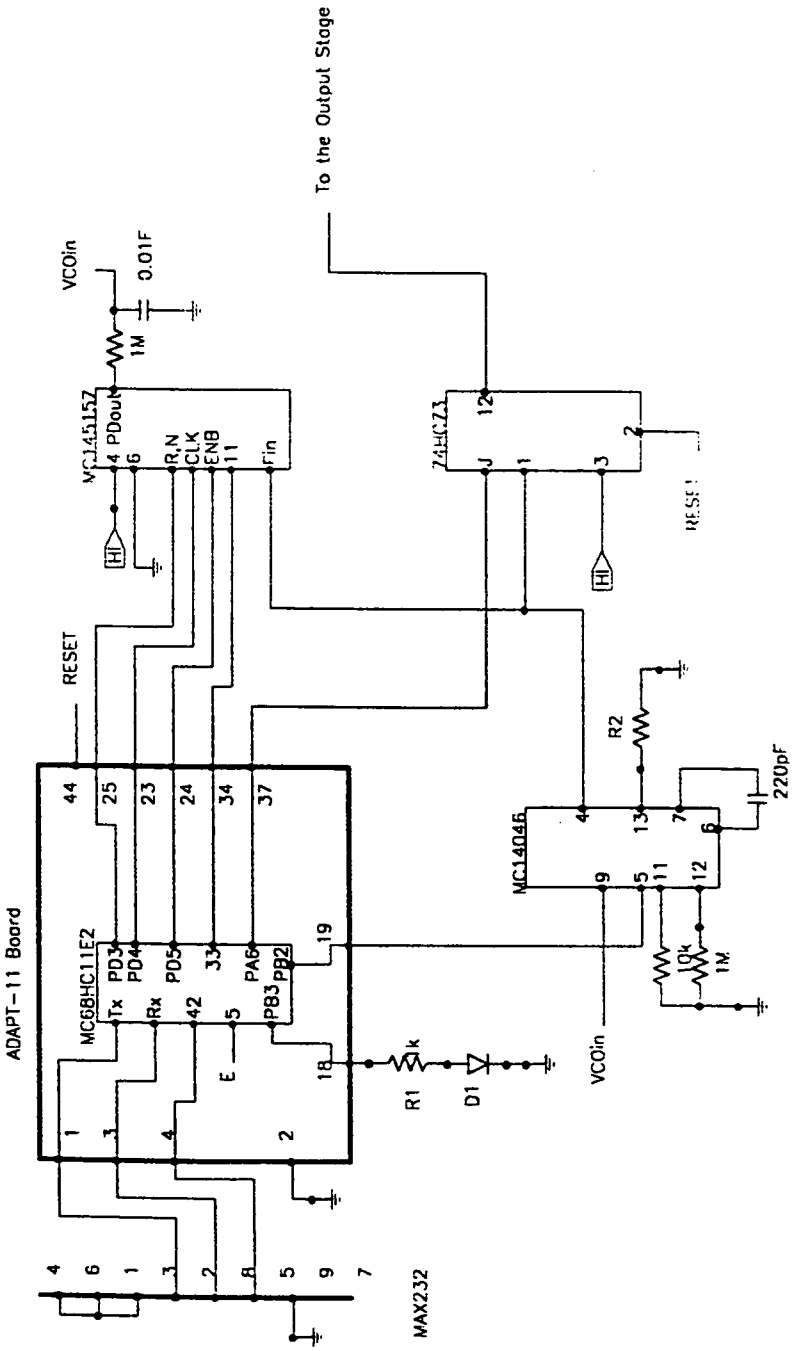


Figure 5.1 ATFUR Project Schematic

The user has to choose the brand and model of the TV which is going to be tested. This is done by clicking on both brand and IR format menu rings, and highlighting the desired selection. If the ATFURMD is used, the VI will advise the user to push the POWER button first to make sure that he is addressing the right television model. Therefore, the user must push the POWER button by using the Sun workstation mouse. Once this button is pushed, the data will be concatenated in a single string and sent to the GPIB VI. The brand and model selection will get the repeat sequence which is set to 3, the carrier frequency, time intervals in ms, the header and the custom code if applicable. The model and the command button will generate the output code. All these parameters will be concatenated in a single string according to the following example:

/3 NN.N T1 T2 T3 T4 T5 : 5233111132432222511532 4;

This string will be sent to the GPIB-SCSI-A box via the GPIB Write VI located in the block diagram of the ATFUR driver.

Next, the GPIB-232CT-A box will receive this string from the GPIB-SCSI-A box. At this point the signal will be represented by ±10V voltages. The 232CT box will send the data to the MAX232 chip, which transforms these voltages from ± 10V to 0 and 5V. The microcontroller will receive this data on the Tx/Rx pins. All I/O ports are initialized. The MC will interpret the data by looking for a "/" character first, than for the repeat digit and so forth. Once receiving a digit, the MC will store it in the variable 'Repeats' in order to be used

later by the Xmit subroutine. Then the function ParseFreq is called to parse a frequency if present. If the frequency is below 30kHz, the function will return an error. If the frequency is within the defined range, the frequency synthesizer chip will be setup for the correct frequency. Then the software goes into a loop looking for time intervals. When a colon is found the look up process stops. The ParseTime function stores the value in the TimeTbl array and looks for another time value. When the colon character is found, this function exits and the software will look for the burst/delay patterns. First non-white character found is saved in the variable TxStart, so it can be used by the Xmit routine. Then all the following transmit time index characters (1-8) are counted until a semi-colon character is found. Finally, the Xmit subroutine is called to transmit the pattern. All These tasks are fully explained in Chapter 3. While receiving the data, the activity LED will be blinking. This visible red LED indicator is driven by the microcontroller PB3 output, and is current limited by the 1kΩ resistor. Once the microcontroller receives the entire data sequence ending with a semi-colon without any errors, it will send it to the output stage transmitter. The BJT transistors will saturate and the IR LEDs will turn on and send the square wave signal to the television. Right after this and by the means of a popup menu, the ATFUR VI will ask the user if the TV has responded. If the user answers with YES, then the VI will keep running and the user can push any other button, and the code will be sent in the same manner the POWER code was sent but with different value in the sequence. If the user answers with NO, the

ATFUR will advise him to choose a new brand from the brand's menu ring. Then the same procedure is repeated until the user is able to get feedback from the TV.

The ATFURAD will operate the same, but without human intervention. This VI will receive its instructions automatically from another VI, but it will process them in the same manner and format as the manual version. This VI will not generate any popup menus except when a mismatch between the model and brand, or a mismatch between a model and a command occurs. It will still generate errors when they occur and it will pass them to the next VI. In case these errors are GPIB errors, the VI will not execute, and will report these errors. The user can then identify the errors and try to fix them before running the VIs again.

Sometimes it is hard to find the exact model of a TV because it is not labeled. For example, an NEC TV may not respond to any command even though the NEC IR format is being sent. This does not mean that the software is not generating the sequence, nor that the microcontroller is not generating it, but simply this is not the right model. Therefore, the user should have some knowledge and experience in identifying which IR format goes with which brand. The user can always look at the output of the VIs and check if they are sending any codes. Arrays and indicators are available on both drivers to show the transmission of the data.

5.2 PROBLEMS

This section will discuss possible problems that may be encountered while using the ATFUR units. These problems are often of a physical nature. Even though the ATFUR is operating on low voltage under non-stressful conditions, electronic failure can take place. In case the remote control is not functioning it should be determined whether the problem is indeed in the remote itself or the TV. The easiest way to do this is to check the commands-models table to see if the command to be executed is available for this model. Next, a manual execution of the command should be done. For instance, pushing the power button on the TV to check if it is working. If the TV responds, then it is certain that the problem lies in the remote control itself.

The most common type of problems within the remote control is to have a bad LED, or a bad component such as a resistor or a capacitor. Most remote control units use a pulse code modulated carrier to send commands. A typical carrier frequency is 40 kHz with each pulse consisting of multiple cycles of this carrier. A suggested solution is to use an IR detector to diagnose the problem. Figure 5.2 shows the schematic of an IR detector. It may be used to test the remote control unit. The IR detector, developed by the author, is inexpensive to build. The components are affordable and easy to get. It works on a 9V power supply, an IR sensor photodiode (SP), a BJT transistor and a visible LED. The values of R1, R2 and R3 are 3.3KΩ,

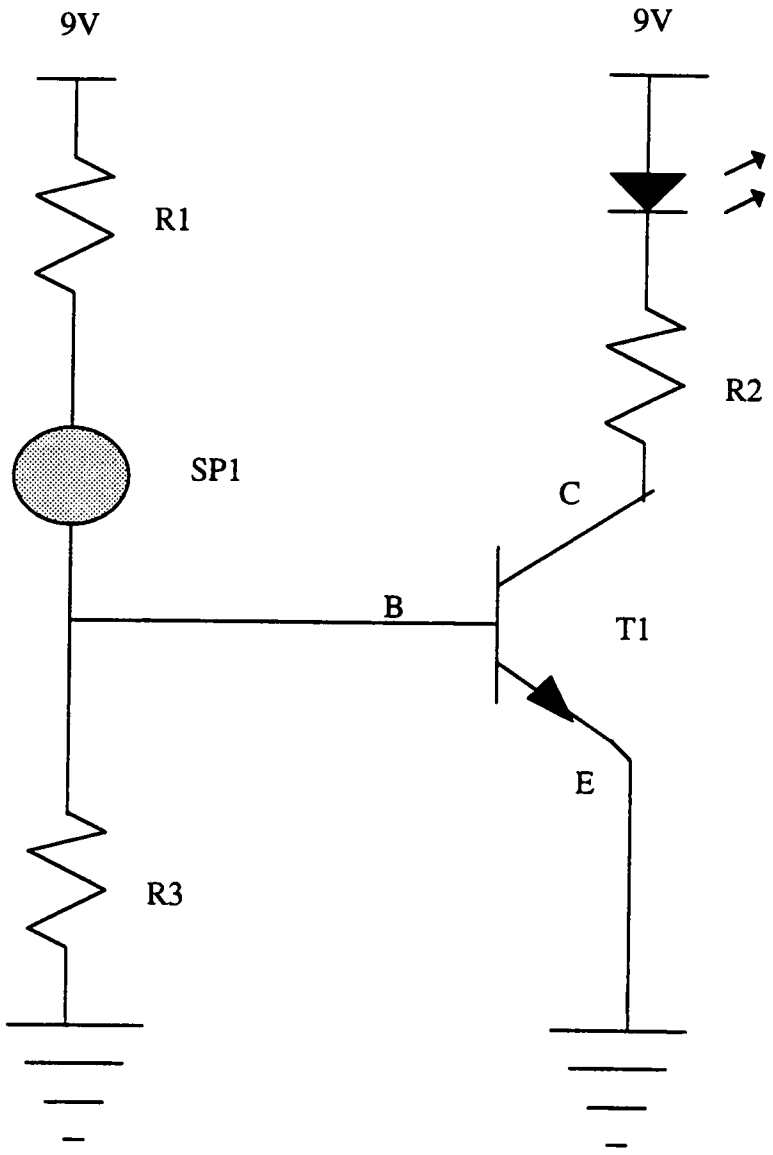


Figure 5.2 IR Detector

500Ω and 27KΩ respectively. The BJT transistor is a typical 2N3904. If the remote control is sending IR signals, the SP1 will detect them and the visible LED will be lit. If any of the IR LEDs in the ATFUR is bad or burned out, none of the other LEDs will be able to send signals because they will be in an open circuit situation. Therefore, the SP1 will not detect any signals and the visible LED will not light. The IR detector can be used also to view the actual signals. The collector of the BJT can be connected to an oscilloscope and the IR signals can be viewed on the screen of the scope. If the problem is a burned LED, it should be replaced by exactly the same model. Since the forward voltage and the wavelength of an LED are directly related, it is not possible to substitute different LEDs without changing the voltage and current. Table 5.1 shows the relationship between the LED wavelength and the correspondent voltage [27].

Another problem may be in the IC chips. Since the chips used in this design are sensitive devices, variation of voltage or current within the circuitry can damage them. Sometimes inserting the positive / negative wires backwards may harm the ICs. A scope may be used to check each pin on the ICs to determine if it is good. The alternative solution to that is to identify which part is not functional and replace it. The MC chip must be replaced by a similar and pre-programmed one. This problem is not very common.

Table 5.1 LED Wavelength vs. Voltage

WAVELENGTH (Nm)	VOLTAGE (V)
880	2.0 - 2.5
900	1.2 - 1.6
940	1.3 - 1.7

Finally, the problem may not be in the hardware but in the software. The user should watch the length and the content of the outputted string and verify its accuracy. The assigned GPIB address should be checked to see if it matches the RS-232 box address plus one. The outlet number should be also checked to see if it exists in the room where the ATFUR is being used. The power connections should be checked to make sure that every device is powered up and running.

The above are problems that may be encountered when operating the ATFUR devices throughout time. Extreme caution should be taken when diagnosing or fixing a problem. Safety precautions should not be ignored while fixing the device or even when operating it. Before the hardware box is opened for diagnosing purposes, power should be shut off. While operating the device, a proper ground should be used. All wires carrying AC voltages must be properly and thoroughly insulated. If the ATFUR location is to be changed in the future, adequate safety

should be the number one concern and not convenience at all times.

CHAPTER 6

DISCUSSION

6.1 CONCLUSION

Software quality is the result of good design and the systematic elimination of random errors. The ATF project team wanted to deliver good software quality. Therefore automated testing was adopted to implement the project. Automated testing has many advantages over manual testing. It saves time, cost, increases accuracy and removes a bottleneck. *Save time:* automated tests can be run unattended once created. The same operator can perform more tests in the same time. *Save cost:* testing defects and errors can be detected earlier and the cost to fix them at that stage will be low. It also reduces the risk of delivering defective products with added support costs. *Increase accuracy:* manual testing is subject to the simple variability of human nature. Automated testing performs quality tests consistently and relentlessly. *Remove a bottleneck:* testing is a major bottleneck, automated testing allow efficient testing as code is written, so developers do not trade off quality for speed of the design.

In the ATF project, automated tests were executed frequently to ensure the accuracy of the results obtained. These tests were quick and

efficient. Quick because of the fact that only one operator was needed to run the software, make a few connections between devices and change the televisions under test. Efficient because of the fact that the results obtained were very similar to standard known values.

LabVIEW was a wonderful software environment to use. Each team member was able to develop his/her own drivers for different tests. All the drivers were connected at the end to make the final hierarchy of test drivers. LabVIEW allowed the codes to be very user friendly because of its graphical simplicity. It also provided means to transfer data to and from instruments and devices. Those means are summarized by the pre-developed GPIB VIs. The IVP methodology used in conjunction with LabVIEW deserves a lot of credit. It offered a very good approach for developing test drivers. It allowed customers to keep track of the development process and give comments when needed. It also offered modularity of codes, all the VIs looks as if they were done by only one programmer. Finally, it allowed team members to verify each others work at each increment in a way similar to a peer review process, resulting in efficient and error -free code.

The peer review process was very beneficial in many ways. Defects were found at the earliest possible point in the process of development. Work was verified to check if it met the predefined criteria or specifications. Peer review ensured that each work was ready to proceed to the next stage of development according to the IVP

methodology. The black box was reviewed and agreed upon first, then the development on the state box followed. In addition, by conducting peer reviews each team member gained knowledge about others work and was able to comment on it.

The ATFUR was developed so that the operator uses only one remote control to change several TV channels. All available television codes were stored in global variables. The driver has total access to these GVs to get the brand, model and command codes for televisions under test. The manual driver was used to control televisions from the Sun console. This driver can be used to power up all TVs and set them to different channels and states. The automated driver is used in conjunction with other VIs. It saves the operator a lot of time during any test. It will not be necessary to stop LabVIEW, find the corresponding remote control, set the channels and run LabVIEW. Instead the ATFUR will receive these commands from another VI and execute them. Each of these units will be installed in a room, which allows multiple use of the devices at the same time. This is possible by using the two GPIB buses. Figure 6.1 shows times required to send only one command to a specified model by different means. By using either one of the ATFUR versions, the operator is saving time. To send a single command, it will take about 66 seconds to find the correct remote control and push the button on a real remote. Using the ATFURMD it will take around 36 seconds to choose the brand and model and push on the command button on the Sun console. Using the

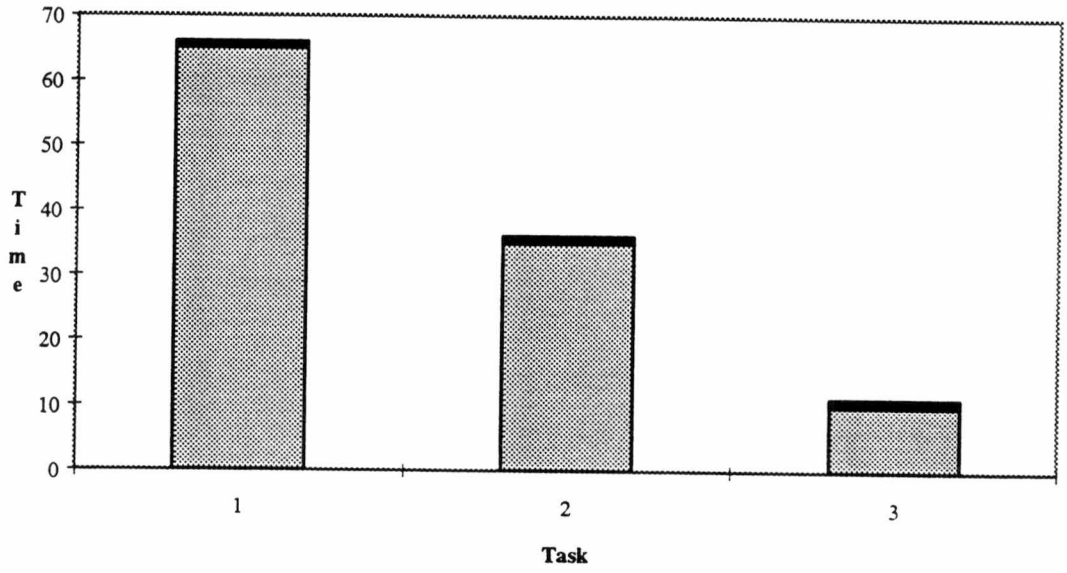


Figure 6.1 Times Comparison

ATFURAD it will take only 11 seconds for locating the code in the GVs and sending it. During one day, one command button will be pushed more than 100 times, multiplying the times in Figure 6.1 by 100 will tell how much time can be saved by using the ATFURs.

The ATFUR has its disadvantages also. The devices will be set up in one location and will not be moved. IR signals will not hit the bezel of some TV receivers at the desired angle to make it respond. Therefore, the operator needs to move the television and put it in a way that faces the unit. Another inconvenience will be in the stack codes.

The global variable tables need to be maintained on a permanent basis. In case a failure occurs while running these GVs, data can be lost or mixed up. The operator then needs to enter all of the data again or use a backup copy. Each time new TV codes are available, they need to be customized to the format explained in Section 4.1. This is the only format that the ATFUR can execute. The new codes need to be entered in the appropriate GVs. This process needs to be checked and double checked to avoid the occurrence of mistakes when customizing the codes or when entering it. The last inconvenience is the GPIB addresses. GPIB and RS-232 boxes are used by all the ATF project. Each device or instrument has a different GPIB address. Therefore the addresses on the boxes are going to change each time a specific device is used. The operator needs to set the appropriate addresses again when borrowing a box from a different device. This can be a real hassle unless only one operator will conduct the tests, or each device will be supplied with its own GPIB/RS-232 box.

Finally, this project was a very informative experience for me. I was able to learn about software development as well as hardware development. I understood how a microcontroller works and the way they can be programmed. I also understood how software drivers can control hardware equipment.

6.2 FURTHER DEVELOPMENT

Even though the ATFUR is fully functional and efficient, some improvements and additions can be made to it. At this stage, the ATFUR only has the ability to send codes to televisions. It does not have any clue if the intended television has received the code or not. A receiver can be implemented in the hardware box to make the ATFUR smarter. The IR receiver will get a feedback from the TV and act accordingly. In case the TV did not receive the code, the ATFUR will keep sending it automatically until it gets an acknowledgment from the TV. The second step will be in developing a LabVIEW VI that converts the binary codes of the signals to the custom form that can be used directly without converting them manually. This is not hard to do, the structure of a "0" and "1" will be given to the VI followed by the binary sequence. The VI will output the times needed and the duration of each burst and delay. By doing that, the ATFUR will be able to control VCRs and cable boxes with their codes. This VI will eliminate human error when customizing or entering the codes. The last stage of improvement will be in making the ATFUR learn the codes of each TV from the TV itself, and store it in the global variables directly. The ATFUR will then act like any universal remote (UR) control that learns the codes of any television. The only difference between the ATFUR and a normal UR will be in the location of the codes. The ATFUR will still be soft coded with the codes being in GVs, while the UR will be hard coded with the codes being in the MC chip. With this

implementation, the ATFUR will need to be initialized only once when a new television is introduced.

Another way to improve the ATFUR is to make it work from a PC. In case PCs are introduced to the ATF facility, the ATFUR should be operated straight from the serial port COM1 or COM2 of the computer. The LabVIEW GPIB Write VI can be replaced by two other VIs: Serial Port Init and Serial Port Write. The Serial Port Init VI will initialize the PC serial port with the right setting in order to talk to the ATFUR. Parity bits, stop bits, bits/character and baud rate should be defined. The Serial Port Write VI will take the data from the global variable's tables and put them on the PC buffer, where they are transmitted to the ATFUR hardware box.

These improvements will be nice to have as features in the ATFUR. The unit then will become even more efficient and more user friendly. It will also be utilized in multiple applications later on, such as controlling a CD changer, a laser disk player or any kind of equipment that is controlled by infrared signals.

LIST OF REFERENCES

LIST OF REFERENCES

[1] S. Sadiq-Hussain, and K. Al-Shakarchi, "*Microprocessor-Based System for Color TV Testing During Manufacture*", Computer in Industry, Vol. 11, February 1989, pp. 349-355.

[2] *Motorola M68HC11 Reference Manual*, Rev. 3, 1991, Motorola Inc.

[3] Daniel B. Koch, S. Shah, K. Singh, E. Wruchen, and G. Ragsdale, "*On Using The Software Cleanroom Process In Automated Test Equipment Applications*", Proceedings of the IEEE Southeastern'96, pp.703-706, Tampa, Florida, April 11-14, 1996.

[4] *LabVIEW for Sun User manual*, December 1993 Edition, National Instruments Corporation.

[5] Thomas A. Freeburry, "*Enabling Technologies For Wireless In Building Network Communications-Four Technical Challenges, Four Solutions*", IEEE Communications Magazine, pp. 58-64, April 1991.

[6] G. L. Ragsdale, "*Automated Nonlinear Distortion Measurements*", University of Tennessee, Knoxville, TN, 1995.

[7] Keeratpal Singh, Personal Correspondence, *RF Amplitude Levels*, January 1996.

[8] *LabVIEW Basics Course Manual*, January 1994 Edition, National Instruments Corporation.

[9] G. J. Lipovski, "*Single - and Multiple - Chip Microcomputer Interfacing*", Prentice-Hall Inc.,1988.

[10] *1996 Instrumentation Reference And Catalogue*, 1995, National Instruments corporation.

[11] F. Mazda, "*Electronic Instruments And Measurement Techniques*", Cambridge University Press, New York, 1987.

[12] *Getting Started With Your GPIB-SCSI-A And The NI-488.2M Software For The Sun SPARCstation*, September 1992 Edition, National Instruments Corporation.

[13] *GPIB-232CT-A User Manual*, October 1993 Edition, National Instruments Corporation.

[14] David W. Schwaderer, "*Digital Communications Programming On The IBM PC*", John Wiley & Sons Inc., pp. 34, 1985.

[15] Gary W. Johnson, "*LabVIEW Graphical Programming*", McGraw-Hill Inc., 1994.

[16] Kenneth Hintz and Daniel Tabak, "*Microcontrollers Architecture, Implementation & Programming*", McGraw-Hill Inc., 1992.

[17] John Socha and Peter Norton, "*Assembly Language For The PC*", Brady Publishing, Third Edition, 1992.

[18] Jeff Duntemann, "*Assembly Language*", John Wiley & Sons Inc., 1992.

[19] *Motorola M68HC11 E Series Technical Data*, Revision 1, 1995, Motorola Inc.

[20] *Maxim MAX220-MAX249*, Revision 5, 1995, Maxim Integrated Products.

[21] *Britannica On-line*, World Wide Web, Encyclopedia Britannica, 1995.

[22] Riccardo Vanzetti, "*Infrared Emission By Electronic Equipment*", John Wiley & Sons Inc., pp. 169-180, February 1972.

[23] *Motorola Semiconductor technical Data*, Revision 2, 1995, Motorola Inc.

[24] Ralph Hovath, "*An Introduction to Microprocessors with the MC6809*", McGraw-Hill Inc., 1990.

[25] Technological Arts, "*Modular Prototypes Systems*", World Wide Web, 1996.

[26] Ron Kielkowski, "*SPICE Practical Device Modeling*", McGraw-Hill Inc., 1995.

[27] Forrest M. Mims, "*Getting Started in Electronics*", Third edition, Radio Shack, 1983.

[28] *LabVIEW Function Reference Manual*, December 1993 Edition, National Instruments Corporation.

[29] D. T. Horn, "*Home Remote-control & Automation Projects*", Mc-Graw Hill Inc., 1986.

APPENDICES

APPENDIX A

All programs, subroutines and utilities for programming the MC68HC11E2 are found in this appendix.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; IR.INC -- InfraRed Transmitter Configuration File
; Defines project-specific constants.
; Bassel Atallah
; 06/01/96
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RAM_BASE    EQU $0000   ;RAM starting address
REG_BASE    EQU $1000   ;internal register starting address
ROM_BASE    EQU $F800   ;EEPROM starting address
VEC_BASE    EQU $FFC0   ;vector table starting address
RXQSIZE     EQU 128     ;Receive queue size
NUM_TIMES   EQU 8       ;Maximum number of burst/delay times

;;;;; CHARACTER CONSTANTS ;;;;;

TAB         EQU $09
LF          EQU $0A
CR          EQU $0D
SPACE       EQU $20
SLASH       EQU '/      ;start of command character
COLON       EQU ':      ;start of transmit pattern character
SEMICOLON   EQU $3B     ;end of command character
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; IRMAIN.ASM -- InfraRed Transmitter Mainline Program.
; Bassel Atallah
; 06/01/96
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;; CONSTANTS ;;;;;

; PORT A
XMITENA    EQU BIT6   ;PA6/OC2/OC1 transmitter carrier enable
TRIGGER    EQU BIT5   ;PA5/OC3/OC1 external monitor trigger

; PORT B
CTS        EQU BIT0   ;PB0 - CTS RS232 handshake (low = clear to
send)
PLLINH     EQU BIT2   ;PB2 - PLL inhibit bit (hi = inhibit)
ACTIVE     EQU BIT3   ;PB3 - activity LED (hi = on)
ALIVE      EQU BIT7   ;PB7 - heartbeat, pulses when looking for cmd
start

; PORT D
SYNENB     EQU BIT5   ;PD5 - frequency synthesizer load enable (hi
= load)
SCK        EQU BIT4   ;PD4 - shift clock
MOSI       EQU BIT3   ;PD3 - master data out
```

```
; Flags
COLON1ST   EQU BIT0   ;colon was first non-white character in
parse function
ENAN       EQU BIT1   ;Not a Number error from ParseNumber
EFMT       EQU BIT2   ;Error in number format (too many int digits)
EHIGH      EQU BIT3   ;numeric value too high
ELOW       EQU BIT4   ;numeric value too low

; Xmit parameters
DFLT_FREQ  EQU 375    ;37.5kHz

;;;;; VARIABLES ;;;;;
ORG    RAM_BASE
nil        RMB    1
Flags      RMB    1
Accum      RMB    3
AccB       RMB    3
IntDigits  RMB    1
FracDigits RMB    1
DigitCount RMB    1
Minimum    RMB    2
ErrPtr     RMB    2
ErrChar    RMB    1
Freq       RMB    2
Index      RMB    1
```

```
TimeTbl    RMB    NUM_TIMES*3
Repeats    RMB    1
RepeatCount RMB    1
TxLength   RMB    1
TxCount    RMB    1
TxStart    RMB    1
TxIndex    RMB    1
LongMSB    RMB    1
RxCount    RMB    1
RxIn       RMB    1
RxOut      RMB    1
RxQueue    RMB    RXQSIZE

;;;;; MAINLINE ;;;;;

        ORG    $10000-ROM_SIZE
        fcc    / Copyright (c) 1996 /
        fcc    / Bassel Atallah /
        fcc    / All Rights Reserved. /
main    lds    #RAM_SIZE-1    ;initialize stack pointer
        ldx    #REG_BASE      ;point X at registers and NEVER change
it
        bset   _PORTB,X ACTIVE ;turn power/activity LED on
        ldaa   #1             ;set default number of repeats
        staa   <Repeats
```

```
        ldaa   #NUM_TIMES     ;number of intervals
        staa   nil
        ldd    #1000*2        ;initialize time table for 1.000ms intervals
        ldy    #TimeTbl       ;address of interval time table

InitLp  clr    0,y            ;clear MSB of interval
        std    1,y            ;set LSBs for 1 ms
        iny                   ;point to next entry (3-byte values)
        iny
        iny
        dec    <nil           ;done yet?
        bne    InitLp         ;if not init another
        jsr    SCIInit        ;initialize serial port
        jsr    SPIInit        ;initialize SPI port
        jsr    PLLInit        ;initialize PLL hardware
        cli                   ;enable interrupts
        bclr   _PORTB,X ACTIVE ;turn off power/activity LED while
sending msg
        ldy    #PONStr        ;send power-on string out RS232 port
        jsr    PutStr
        bset   _PORTB,X ACTIVE ;turn power/activity LED back on

MainLoop   ;main operating loop
        clr    <Index         ;clear interval parsing index
        clr    <TxLength      ;clear xmit pattern length
```

```
GetStart   ;look for slash at beginning of command
     bset   _PORTB,X ALIVE  ;tickle output bit while looking for
slash
     bclr   _PORTB,X ALIVE
     clr    <Flags          ;clear state flags
     jsr    GetChar         ;get character
     cmpa   #SLASH          ;is it a slash?
     beq    CheckStart      ;if so look for repeat count
     cmpa   #SEMICOLON      ;was it a semi-colon?
     bne    GetStart        ;if not keep looking for slash
     bclr   _PORTB,X CTS    ;if so enable CTS again
     bra    GetStart

CheckStart ;check for repeated slash to help recovery from errors
     jsr    SkipWhite       ;skip any space characters
     cmpa   #SLASH          ;got another slash?
     bne    GetRepeat       ;if not check for repeat/colon
     jsr    GetChar         ;flush slash
     bra    CheckStart      ;and check for another

GetRepeat  ;get repeat count or colon
     jsr    IsDigit         ;is non-white character a digit?
     bcc    GotRepeat       ;if so check it
     cmpa   #COLON          ;else was it a colon?
     beq    GotColon        ;if so get xmit pattern
```

```
BadRepeat   ;report bad repeat count
        ldy    #BadRepeatStr   ;print error string
        jsr    ReportError
        bra    MainLoop        ;start over at beginning

GotRepeat   ;got repeat count
        jsr    GetChar         ;flush repeat character
        suba   #'0
        beq    BadRepeat       ;zero repeat not allowed, report error
        staa   Repeats         ;save number of repeats

GetFreq     ;get transmit carrier frequency
        jsr    ParseFreq       ;get frequency
        brset  <Flags COLON1ST GotColon    ;if colon get xmit pattern
        bcs    MainLoop        ;else if error start over

GetTimes    ;get time interval(s)
        jsr    ParseTime       ;get time interval
        brset  <Flags COLON1ST GotColon    ;if colon get xmit pattern
        bcs    MainLoop                    ;else if error start over
        inc    <Index          ;increment interval parse index
        ldaa   <Index          ;see if we've filled time interval table
        cmpa   #NUM_TIMES
        blo    GetTimes        ;if not get another
```

```
GetColon   ;get colon separator or first xmit pattern digit
     jsr    SkipWhite      ;skip space characters
     cmpa   #COLON         ;is non-white character a colon?
     beq    GotColon       ;if so get xmit pattern
     jsr    IsPattern      ;else is it a pattern digit?
     bcc    GotPatStart    ;if so it is the start of the xmit pattern
     jsr    GetChar        ;else ignore character
     bra    GetColon       ;and keep looking for colon

GotColon   ;got colon
     jsr    GetChar        ;flush colon character

GetPatStart ;get start of transmit pattern
     jsr    SkipWhite      ;skip space characters
     jsr    IsPattern      ;is non-white character pattern digit?
     bcc    GotPatStart    ;if so it is the start of the xmit pattern

BadPattern  ;report bad pattern character
     jsr    GetChar        ;flush erroneous character
     ldy    #BadPatternStr ;print error string
     jsr    ReportError
     jmp    MainLoop       ;and start all over again

GotPatStart ;got start of transmit pattern
     ldaa   <RxOut         ;save indx of first pattern character
```

```
        staa   <TxStart
        jsr    GetChar        ;flush first pattern character

GetPattern  ;get rest of pattern until semicolon
        inc    <TxLength      ;count pattern character
        jsr    GetChar        ;flush pattern character
        jsr    IsPattern      ;is it a pattern digit?
        bcc    GetPattern     ;if so continue
        cmpa   #SEMICOLON     ;else is it a semicolon?
        bne    BadPattern     ;if not bad pattern

CheckPattern ;see if pattern valid length
        brclr  <TxLength BIT0 GoodPattern  ;if not odd length then it's
good
        ldy    #OddPatternStr  ;else print error string
        jsr    ReportError
        jmp    MainLoop       ;and start all over again

GoodPattern ;set up and send pattern
        ldaa   <Repeats       ;set number of times to repeat pattern
        staa   <RepeatCount
        jsr    Xmit           ;send xmit pattern
        ldy    #OKStr         ;print pattern OK string
        jsr    PutStr
        bclr   _PORTB,X CTS   ;re-enable RS232 CTS signal for next
```

```
command
        jmp     MainLoop        ;get next command string

;;;;; STRINGS ;;;;;

PONStr          fcc 'IRX V1.00'
                fcb CR,LF,0

OKStr           fcc 'OK'
                fcb CR,LF,0

BadRepeatStr    fcc 'BAD REPEAT COUNT'
                fcb 0

BadPatternStr   fcc 'BAD PATTERN'
                fcb 0

OddPatternStr   fcc 'ODD PATTERN LENGTH'
                fcb 0
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; IRIO.ASM -- InfraRed Transmitter I/O Routines.
; Bassel Atallah
; 06/01/96
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RXQWRAP EQU -RXQSIZE
MIN_LONG    EQU 200     ;uncertainty period for long (greater than
;65535 tick) intervals

;;;;; SERIAL I/O ROUTINES ;;;;;

SCIInit   ;initialize SCI (RS232) port for 9600 baud
      ldaa   #SCP1|SCP0      ;select 9600 baud
      staa   _BAUD,X
      clr    <RxIn           ;clear software variables
      clr    <RxOut
      clr    <RxCount
      ldaa   #RIE|TE|RE      ;enable RX and TX, RX interrupt
      staa   _SCCR2,X
      bclr   _PORTB,X CTS    ;enable RS232 CTS
      rts
GetChar   ;return character from serial port in A
      bsr    PeekChar        ;wait until character available
      bsr    PutChar         ;echo character back
```

```
        dec   <RxCount       ;remove from queue
        inc   <RxOut         ;increment queue out index
        bclr  <RxOut RXQWRAP  ;wrap modulo queue size
        rts

PeekChar   ;return next serial port character in A without removing
; from queue
        ldaa  _PORTB,X       ;toggle power/activity LED while waiting
        eora  #ACTIVE
        staa  _PORTB,X
        ldaa  <RxCount       ;wait until character available
        beq   PeekChar
        bset  _PORTB,X ACTIVE ;turn power/activity LED back on
        pshb                 ;save registers
        pshy
        ldab  <RxOut         ;get queue out index
        ldy   #RxQueue       ;add to address of queue
        aby
        ldaa  0,y            ;get character value
        puly                 ;restore registers
        pulb
        rts
PutChar    ;send character in A out serial port
        brclr _SCSR,X TDRE PutChar    ;wait for TX ready
        staa  _SCDR,X        ;send character
```

```
        rts

PutStr    ;send zero-terminated string pointed to by Y out serial port
        psha              ;save registers
        pshy
putslp  ldaa   0,y        ;get character from string
        beq    puts99     ;quit if zero
        bsr    PutChar    ;else send character
        iny               ;increment string pointer
        bra    putslp     ;get next character
puts99  puly              ;restore registers
        pula
        rts

PutCrLf   ;send CR and LF out serial port
        psha              ;save register
        ldaa   #CR        ;send carriage return
        jsr    PutChar
        ldaa   #LF        ;send linefeed
        jsr    PutChar
        pula              ;restore register
        rts
;;;;; SCI Interrupt Service Routine ;;;;;
```

```
SCISvc    ;service SCI interrupt (only RX interrupt enabled)
      ldaa   _SCSR,X        ;read status register to clear RX flag
      ldaa   _SCDR,X        ;read data register
      cmpa   #SEMICOLON     ;check for semicolon, if so disable
; RS232 CTS
      bne    SSNotC         ;if not just enqueue character
      bset   _PORTB,X CTS   ;disable CTS

SSNotC  ldab   <RxIn        ;get queue insertion index
      ldy    #RxQueue       ;add to address of queue
      aby
      staa   0,y            ;store received character
      inc    <RxCount       ;increment received character count
      inc    <RxIn          ;increment insertion index
      bclr   <RxIn RXQWRAP  ;wrap it modulo queue size
      rti                   ;return from interrupt

;;;;; SPI Routines ;;;;;

SPIInit   ;initialize SPI as master, 1MHz SCK frequency, CPHA =
CPOL = 0
      bset   _DDRD,X SYNENB|SCK|MOSI ;enable /SS, SCK and
MOSI outputs
      bclr   _PORTD,X SYNENB        ;clear /SS output
      ldaa   #SPE|MSTR              ;set SPI mode
```

```
        staa    _SPCR,X
        rts

PutSPI      ;send byte in A out SPI port
        staa    _SPDR,X         ;send data byte
        brclr   _SPSR,X SPIF *  ;wait until done sending
        rts

;;;;; PLL Routines ;;;;;

PLLInit     ;initialize PLL hardware to default frequency in
DFLT_FREQ
        bclr    _PORTB,X PLLINH ;enable PLL
        ldd     #DFLT_FREQ
        bsr     SetFreq
        rts

SetFreq     ;set PLL frequency to value in D (in hundreds of Hertz)
          ;must be in the range 300 (30.0 kHz) to 450 (45.0 KHz)
        std     <Freq
        subd    #MIN_FREQ       ;subtract table offset frequency
        lsld                    ;multiply by 2 to make word offset
        addd    #PLLTAB         ;add address of table
        xgdy                    ;put address into Y register
        ldd     0,y             ;get packed R and N counter values
```

```
        psha            ;save N (upper 6 bits)
        anda  #%00000011    ;get R from lower 10 bits of D
        sec             ;shift 1 into LSB for R value
        rolb
        rola
        jsr   SetDivisor    ;set R value
        pulb            ;restore N
        andb  #%11111100    ;extract from upper 6 bits
        lsrb            ;leave shifted 1 bit, 0 in LSB for N value
        clra            ;clear MSB of 16 bit D register
        jsr   SetDivisor
        rts

SetDivisor  ;set divider in MC145157 frequency synthesizer to value in
; D
          ;LSB = 1 for R counter, 0 for N counter.  Next 14 bits are
          ;counter value.  MSB is a don't care.
        bsr   PutSPI        ;send high byte of value out SPI
        tba
        bsr   PutSPI        ;send low byte of value and control bit
        bset  _PORTD,X SYNENB ;strobe into divider
        bclr  _PORTD,X SYNENB
        rts
```

```
;;;;; Transmit Routine ;;;;;

Xmit    ;transmit pattern RepeatCount times
        bclr   _PORTB,X ACTIVE ;turn off power/activity LED
        ldd    _TCNT,X         ;set start of first burst to occur
        addd   #100            ;after the following setup operation
        std    _TOC1,X
        bclr   _TFLG1,X $FF^OC1F      ;clear OC1 flag
        bset   _OC1M,X XMITENA|TRIGGER ;OC1 updates
XMITENA, TRIGGER outputs
        bset   _OC1D,X XMITENA|TRIGGER ;set XMITENA,
TRIGGER outputs high on
                               ;TOC1 match
        brclr  _TFLG1,X OC1F * ;wait for start of first burst

XmitPattern ;send pattern from beginning
        ldaa   <TxLength       ;set number of times in pattern
        staa   <TxCount
        ldaa   <TxStart        ;set pointer to start of pattern
        staa   <TxIndex

XmitInterval ;handle one interval of pattern
        bclr   _TFLG1,X $FF^OC1F   ;clear OC1 flag
        ldab   <TxIndex        ;get index to xmit interval character
        inc    <TxIndex        ;increment index for next time
```

```
        bclr   <TxIndex RXQWRAP ;wrap modulo queue size
        ldy    #RxQueue       ;get address of queue
        aby                   ;add index
        ldab   0,y            ;get interval character
        subb   #'1            ;subtract ASCII numeral offset
        ldy    #TimeTbl       ;get time interval table address
        aby                   ;offset by index in B times 3
        aby                   ;(time table entries are 3-byte values)
        aby
        ldab   0,y            ;get MSB of interval
        beq    XmitShort      ;if MSB zero do short interval
        stab   <LongMSB       ;save long interval MSB
        ldd    1,y            ;get LSBs of interval
        cmpd   #MIN_LONG      ;within interval of uncertainty?
        bhs    XmitLong       ;if not handle normally
        inc    _TOC1,X        ;bump TOC1 by 256 to wait until end of
                              ;uncertainty interval
        brclr  _TFLG1,X OC1F * ;wait for OC1F
        dec    _TOC1,X        ;restore TOC1

XmitLong  ;handle normal long interval
        addd   _TOC1,X        ;add LSBs of long interval to TOC1
        std    _TOC1,X
```

```
XmitLongLp  ;wait for MSB of long interval to expire
        brclr  _TFLG1,X OC1F * ;wait for OC1 flag
        bclr   _TFLG1,X $FF^OC1F   ;clear flag
        dec    <LongMSB        ;decrement long MSB
        bne    XmitLongLp      ;if not zero wait for another wrap-around
        bra    XmitSetup       ;set up outputs changes for end of interval

XmitShort   ;do short (less than 65536 ticks) interval
        ldd    _TOC1,X         ;get TOC1 value
        addd   1,y             ;add current interval
        std    _TOC1,X

XmitSetup   ;set up output states to take effect at end of current interval
        ldaa   _OC1D,X         ;set up output state for next TOC1
        eora   #XMITENA        ;assume we're inverting XMITENA
; output
        anda   #$FF^TRIGGER    ;  and clearing TRIGGER output
        dec    <TxCount        ;see if at last interval of pattern
        bne    XmitWait        ;if not we're set
        oraa   #TRIGGER        ;assume we're asserting TRIGGER again
        dec    <RepeatCount    ;see if at last repeat of pattern
        bne    XmitWait        ;if not we're set
        anda   #$FF^XMITENA^TRIGGER    ;turn all off after last
; interval
```

```
XmitWait   ;wait for current interval to expire
     staa  _OC1D,X        ;set output state at next TOC1 compare
     brclr _TFLG1,X OC1F * ;wait for interval to expire
     ldab  <TxCount       ;see if any intervals left in pattern
     bne   XmitInterval   ;if so do next interval
     ldab  <RepeatCount   ;see if any repeats left
     bne   XmitPattern    ;if so do another pattern

XmitDone   ;xmit complete
     bset  _PORTB,X ACTIVE ;turn power/activity LED back on
     rts

;;;;; VECTOR TABLE ;;;;;

     ORG   SCIVEC         ;SCI serial interrupt vector
     FDB   SCISvc
    ORG    RESETVEC       ;reset vector
     FDB   main
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; IRPARSE.ASM -- InfraRed Transmitter Parsing Functions.
; Bassel Atallah
; 06/01/96
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MIN_FREQ   equ 300        ;30.0KHz
MAX_FREQ   equ 555        ;55.5KHz
MIN_TIME   equ 80         ;00.080ms
MAX_TIME   equ 99999      ;99.999ms (don't check, must be <= 5
digits)

ParseFreq  ;parse frequency from input queue of the form [nn][.n]

    clr   <ErrChar      ;no character to report in case of error
    ldaa  #2            ;get number with <= 2 int and <= 1 frac digits
    ldab  #1
    ldy   #FreqString
    jsr   ParseNumber
    bcs   PFDone        ;if colon found/error return
      ;check frequency -- no need to check MSB since with 3 digits
      ;maximum value is 99.9 KHz
    ldd   <Accum+1      ;check LSBs
    cmpd  #MIN_FREQ
    blo   PFLow
```

```
        cmpd   #MAX_FREQ
        bls    PFOK
PFHigh  jsr    ReportHighErr   ;frequency high
        bra    PFDone
PFLow   jsr    ReportLowErr    ;frequency low
        bra    PFDone
PFOK    ;frequency OK
        jsr    SetFreq         ;set PLL to parsed frequency
        clc                    ;return without error
PFDone  rts

ParseTime   ;parse time from input queue of the form [nn][.nnn]
            ;note: number cannot be too high since 99.999 is a valid time
            ;value so no need to check for that condition

        ldaa   <Index          ;set delay number being parsed in case of
error
        adda   #'1
        staa   <ErrChar
        ldaa   #2              ;get number with <= 2 int and <= 3 frac digits
        ldab   #3
        ldy    #DelayString
        jsr    ParseNumber
        bcs    PTDone          ;if colon found/error return
        ldaa   <Accum          ;else check time range
```

```
        bne    PTOK          ;if MSB non-zero then must be >=65536
(OK)
        ldd    <Accum+1
        cmpd   #MIN_TIME     ;see if above minimum time
        bhs    PTOK

PTLow   jsr    ReportLowErr
        bra    PTDone

PTOK    ldab   <Index        ;get index of time interval being parsed
        ldy    #TimeTbl      ;get address of time table
        aby                  ;add index to it (3-byte entries)
        aby
        aby
        ldaa   <Accum        ;copy parsed value to table
        staa   0,y
        ldd    <Accum+1
        lsld                 ;save interval converted to 2MHz E ticks
        rol    0,y           ;(multiply by 2)
        std    1,y
        clc
PTDone  rts

ParseNumber ;parse decimal number from input queue, return in
Accum[0-2]
```

```
        ;pass number of integer digits in A, fractional in B, and
        ;pointer to put into ErrPtr in Y.  Returns with carry set
        ;COLON1ST flat set if colon first non-white character found.
        ;returns carry set and ENAN flag set if no digits found.
        staa   <IntDigits      ;save arguments
        stab   <FracDigits
        sty    <ErrPtr
        clr    <DigitCount     ;clear number of digits found
        clr    <Accum          ;clear accumulator
        clr    <Accum+1
        clr    <Accum+2
        bclr   <Flags ENAN|EHIGH|ELOW|COLON1ST ;clear flags
        jsr    SkipWhite       ;skip space characters
        cmpa   #COLON          ;is first non-white a colon?
        bne    PNLp1           ;if not continue
        bset   <Flags COLON1ST ;else flag it
        bra    PNError         ;and return with carry set

PNLp1   ;parse integer digits
        cmpa   #'.             ;is character decimal point?
        beq    ParseDP         ;if so parse fractional digits
        jsr    IsDigit         ;is character a digit?
        bcs    PNValid         ;if not then see if we have a valid number
        inc    <DigitCount     ;count digit
        jsr    GetChar         ;flush it
```

```
        jsr   MultAccum     ;multiply accumulator by 10 and add digit
        jsr   PeekChar      ;check next digit
        dec   IntDigits     ;any integer digits left?
        bne   PNLp1         ;if so check for another
        cmpa  #'.           ;else must be decimal point
        beq   ParseDP       ;if so parse it
        jsr   IsDigit       ;see if digit
        bcs   PNValid       ;if not see if we have a valid number
        bset  <Flags EFMT   ;if so too many int digits
        ldy   #FMTString    ;and report it
        bra   PNReportErr

ParseDP ;got decimal point
        jsr   GetChar       ;flush decimal point

ParseFrac ;parse fractional digits
        jsr   PeekChar      ;check next character
        jsr   IsDigit       ;is it a digit?
        bcs   PNValid       ;if not see if we have a valid number
        inc   <DigitCount   ;else count digit
        jsr   GetChar       ;flush it
        jsr   MultAccum     ;multiply accumulator by 10 and add digit
        dec   <FracDigits   ;any fractional digits left?
        bne   ParseFrac     ;if so check for another
```

```
PNValid   ;see if we have a valid number
     ldaa  <DigitCount    ;any digits seen?
     bne   Normalize      ;if so then check for normalization
     bset  <Flags ENAN    ;flag not a number error
     ldy   #NANString     ;and report it
     bra   PNReportErr

Normalize  ;check for normalization
     ldaa  <FracDigits    ;got all possible fractional digits?
     beq   PNOK           ;if so then we're done

NormLp    ;perform normalization
     jsr   Mult10         ;shift left 1 decimal place (multiplying by 10)
     dec   <FracDigits    ;done normalizing?
     bne   NormLp         ;if not shift again

PNOK   clc                ;return successful status (carry clear)
     bra   PNDone

PNReportErr ;report error parsing number
     jsr   NumericError   ;report error

PNError sec               ;return colon found/error status (carry set)

PNDone  rts
```

```
MultAccum  ;multiply Accum[0-2] by 10 and add decimal value in A
        bsr    Mult10        ;multiply previous accumulated digits by 10
        suba   #'0           ;subtract ASCII offset
        adda   <Accum+2      ;add to LSB of accumulator
        staa   <Accum+2
        bcc    MADone        ;if not carry we're done
        inc    <Accum+1      ;else propagate to middle byte
        bne    MADone        ;if no rollover we're done
        inc    <Accum        ;else propagate to MSB

MADone  rts

Mult10     ;multiply Accum[0-2] by 10
          ;added Accum*2 to Accum*8 to get Accum*10
        psha             ;save registers
        pshb
        ldd    <Accum+1   ;Accum = AccB = Accum * 2
        lsld
        rol    <Accum
        std    <Accum+1
        std    <AccB+1
        ldaa   <Accum
        staa   <AccB
        ldd    <Accum+1   ;Accum = Accum * 4 (original Accum * 8)
        lsld
```

```
        rol    <Accum
        lsld
        rol    <Accum
        addd   <AccB+1     ;Accum = Accum * 2 + Accum * 8 = Accum
* 10
        std    <Accum+1
        ldaa   <Accum
        adca   <AccB
        staa   <Accum
        pulb               ;restore registers
        pula
        rts

SkipWhite  ;skip whitespace characters, return first non-white in A
        jsr    PeekChar    ;check next character in rx queue
        cmpa   #SPACE      ;space character?
        beq    SkipIt      ;if so skip it
        cmpa   #TAB        ;tab character?
        beq    SkipIt      ;if so skip it
        cmpa   #CR         ;carriage return character?
        beq    SkipIt      ;if so skip it
        cmpa   #LF         ;linefeed character?
        bne    SkipDone    ;if not then quit skipping
```

```
SkipIt  ;skip space character
        jsr    GetChar        ;flush character
        bra    SkipWhite      ;check next character

SkipDone rts

IsDigit    ;return carry clear if A is decimal digit, else set
        cmpa   #'0            ;less than ASCII 0?
        blo    IDNot          ;if so then not digit
        cmpa   #'9            ;greater than ASCII 9?
        bhi    IDNot          ;if so then not digit
        clc                   ;else digit, return carry clear
        bra    IDDone

IDNot   sec                   ;not digit, return carry set

IDDone  rts

IsPattern   ;return carry clear if A is an xmit pattern digit, else set
        cmpa   #'1            ;less than ASCII 1?
        blo    IPNot          ;if so then not valid pattern digit
        cmpa   #NUM_TIMES+'0  ;greater than highest pattern digit?
        bhi    IPNot          ;if so then not valid pattern digit
        clc                   ;valid pattern digit, return carry clear
        bra    IPDone
```

```
IPNot   sec                 ;not valid pattern digit, return carry set

IPDone  rts

PutByte     ;send A as two hex nibbles out RS232 port (debugging
routine)
        psha                ;save value
        lsra                ;shift high nibble to low nibble
        lsra
        lsra
        lsra
        jsr    PutNibble    ;send high nibble
        pula                ;restore value
        jsr    PutNibble    ;send low nibble
        rts

PutNibble   ;send low nibble of A in hex out RS232 port (debugging
routine)
        psha                ;save value
        anda   #$0F         ;mask low nibble
        oraa   #'0          ;add ASCII 0 offset
        cmpa   #'9          ;greater than ASCII 9?
        bls    PNSend       ;if not then ready to send
        adda   #7           ;add extra offset for ASCII A through F
```

```
PNSend jsr   PutChar       ;send character
       pula                ;restore value
       rts

ReportHighErr  ;report error as "ERROR: <ErrPtr>[ErrChar]
HIGH<CR><LF>"
       bset  <Flags EHIGH  ;set error flag
       ldy   #HighString   ;get pointer to " HIGH" string
       bsr   NumericError  ;report error
       rts

ReportLowErr   ;report error as "ERROR: <ErrPtr>[ErrChar]
LOW<CR><LF>"
       bset  <Flags ELOW   ;set error flag
       ldy   #LowString    ;get pointer to " LOW" string
       bsr   NumericError  ;report error
       rts

NumericError ;report error as "ERROR:
<ErrPtr>[ErrChar]<userstring>"
       pshy                ;save user string pointer
       ldy   <ErrPtr       ;get error string pointer
       bsr   PrintError    ;print "ERROR: " and error string
       ldaa  <ErrChar      ;get error character
       beq   NECont        ;if zero skip it
```

```
        jsr    PutChar     ;else print it

NECont  puly               ;restore user string pointer
        jsr    PutStr      ;print it
        jsr    PutCrLf     ;print carriage return and linefeed
        rts

ReportError ;report error as "ERROR: <userstring><CR><LF>"
        bsr    PrintError  ;print "ERROR: " and userstring
        jsr    PutCrLf     ;print carriage return and linefeed
        sec                ;set carry to flag error
        rts

PrintError  ;print "ERROR: <userstring>"
        pshy               ;save user string pointer
        ldy    #ErrorStr   ;print "ERROR: "
        jsr    PutStr
        puly               ;restore user string pointer
        jsr    PutStr      ;print it
        rts

;;;;; STRINGS ;;;;;
DelayString     fcc 'TIME'
                fcb 0
FreqString      fcc 'FREQUENCY'
```

```
            fcb 0
ErrorStr    fcc 'ERROR: '
            fcb 0
HighString  fcc ' HIGH'
            fcb 0
LowString   fcc ' LOW'
            fcb 0
FMTString   fcc ' BAD FORMAT'
            fcb 0
NANString   fcc ' NOT A NUMBER'
            fcb 0

; Upper 6 bits are N divisor, lower 10 are the R divisor

PLLTAB
        FDB 3172    ; 30000 Hz (30000 Hz, R  100, N  3)
        FDB 5286    ; 30100 Hz (30120 Hz, R  166, N  5)
        FDB 9514    ; 30200 Hz (30201 Hz, R  298, N  9)
        FDB 3171    ; 30300 Hz (30303 Hz, R   99, N  3)
        FDB 8455    ; 30400 Hz (30418 Hz, R  263, N  8)
        FDB 9511    ; 30500 Hz (30508 Hz, R  295, N  9)
        FDB 3170    ; 30600 Hz (30612 Hz, R   98, N  3)
        FDB 7396    ; 30700 Hz (30701 Hz, R  228, N  7)
        FDB 9508    ; 30800 Hz (30821 Hz, R  292, N  9)
        FDB 3169    ; 30900 Hz (30927 Hz, R   97, N  3)
```

```
FDB 4225  ; 31000 Hz (31007 Hz, R 129, N 4)
FDB 7393  ; 31100 Hz (31111 Hz, R 225, N 7)
FDB 21121 ; 31200 Hz (31201 Hz, R 641, N 20)
FDB 12671 ; 31300 Hz (31331 Hz, R 383, N 12)
FDB 6335  ; 31400 Hz (31413 Hz, R 191, N 6)
FDB 7390  ; 31500 Hz (31531 Hz, R 222, N 7)
FDB 8445  ; 31600 Hz (31620 Hz, R 253, N 8)
FDB 11611 ; 31700 Hz (31700 Hz, R 347, N 11)
FDB 7388  ; 31800 Hz (31818 Hz, R 220, N 7)
FDB 3166  ; 31900 Hz (31914 Hz, R 94, N 3)
FDB 4221  ; 32000 Hz (32000 Hz, R 125, N 4)
FDB 7386  ; 32100 Hz (32110 Hz, R 218, N 7)
FDB 18991 ; 32200 Hz (32200 Hz, R 559, N 18)
FDB 14769 ; 32300 Hz (32332 Hz, R 433, N 14)
FDB 6329  ; 32400 Hz (32432 Hz, R 185, N 6)
FDB 4219  ; 32500 Hz (32520 Hz, R 123, N 4)
FDB 3164  ; 32600 Hz (32608 Hz, R 92, N 3)
FDB 7382  ; 32700 Hz (32710 Hz, R 214, N 7)
FDB 13708 ; 32800 Hz (32828 Hz, R 396, N 13)
FDB 8435  ; 32900 Hz (32921 Hz, R 243, N 8)
FDB 7380  ; 33000 Hz (33018 Hz, R 212, N 7)
FDB 5271  ; 33100 Hz (33112 Hz, R 151, N 5)
FDB 9487  ; 33200 Hz (33210 Hz, R 271, N 9)
FDB 35837 ; 33300 Hz (33300 Hz, R 1021, N 34)
FDB 12647 ; 33400 Hz (33426 Hz, R 359, N 12)
```

```
FDB 6323  ; 33500 Hz (33519 Hz, R 179, N 6)
FDB 4215  ; 33600 Hz (33613 Hz, R 119, N 4)
FDB 3161  ; 33700 Hz (33707 Hz, R  89, N 3)
FDB 7375  ; 33800 Hz (33816 Hz, R 207, N 7)
FDB 17909 ; 33900 Hz (33932 Hz, R 501, N 17)
FDB 5267  ; 34000 Hz (34013 Hz, R 147, N 5)
FDB 10533 ; 34100 Hz (34129 Hz, R 293, N 10)
FDB 9479  ; 34200 Hz (34220 Hz, R 263, N 9)
FDB 7372  ; 34300 Hz (34313 Hz, R 204, N 7)
FDB 15796 ; 34400 Hz (34403 Hz, R 436, N 15)
FDB 25271 ; 34500 Hz (34532 Hz, R 695, N 24)
FDB 8423  ; 34600 Hz (34632 Hz, R 231, N 8)
FDB 5264  ; 34700 Hz (34722 Hz, R 144, N 5)
FDB 7369  ; 34800 Hz (34825 Hz, R 201, N 7)
FDB 11579 ; 34900 Hz (34920 Hz, R 315, N 11)
FDB 7368  ; 35000 Hz (35000 Hz, R 200, N 7)
FDB 15787 ; 35100 Hz (35128 Hz, R 427, N 15)
FDB 5262  ; 35200 Hz (35211 Hz, R 142, N 5)
FDB 13680 ; 35300 Hz (35326 Hz, R 368, N 13)
FDB 13679 ; 35400 Hz (35422 Hz, R 367, N 13)
FDB 6313  ; 35500 Hz (35503 Hz, R 169, N 6)
FDB 12625 ; 35600 Hz (35608 Hz, R 337, N 12)
FDB 3156  ; 35700 Hz (35714 Hz, R  84, N 3)
FDB 11571 ; 35800 Hz (35830 Hz, R 307, N 11)
FDB 6311  ; 35900 Hz (35928 Hz, R 167, N 6)
```

FDB 9466 ; 36000 Hz (36000 Hz, R 250, N 9)

FDB 10517 ; 36100 Hz (36101 Hz, R 277, N 10)

FDB 5258 ; 36200 Hz (36231 Hz, R 138, N 5)

FDB 11567 ; 36300 Hz (36303 Hz, R 303, N 11)

FDB 11566 ; 36400 Hz (36423 Hz, R 302, N 11)

FDB 8411 ; 36500 Hz (36529 Hz, R 219, N 8)

FDB 10513 ; 36600 Hz (36630 Hz, R 273, N 10)

FDB 13666 ; 36700 Hz (36723 Hz, R 354, N 13)

FDB 6307 ; 36800 Hz (36809 Hz, R 163, N 6)

FDB 10511 ; 36900 Hz (36900 Hz, R 271, N 10)

FDB 38888 ; 37000 Hz (37000 Hz, R 1000, N 37)

FDB 15764 ; 37100 Hz (37128 Hz, R 404, N 15)

FDB 8407 ; 37200 Hz (37209 Hz, R 215, N 8)

FDB 5254 ; 37300 Hz (37313 Hz, R 134, N 5)

FDB 11558 ; 37400 Hz (37415 Hz, R 294, N 11)

FDB 3152 ; 37500 Hz (37500 Hz, R 80, N 3)

FDB 12607 ; 37600 Hz (37617 Hz, R 319, N 12)

FDB 22061 ; 37700 Hz (37702 Hz, R 557, N 21)

FDB 9454 ; 37800 Hz (37815 Hz, R 238, N 9)

FDB 8403 ; 37900 Hz (37914 Hz, R 211, N 8)

FDB 10503 ; 38000 Hz (38022 Hz, R 263, N 10)

FDB 13653 ; 38100 Hz (38123 Hz, R 341, N 13)

FDB 6301 ; 38200 Hz (38216 Hz, R 157, N 6)

FDB 10501 ; 38300 Hz (38314 Hz, R 261, N 10)

FDB 25201 ; 38400 Hz (38400 Hz, R 625, N 24)

FDB 22049 ; 38500 Hz (38532 Hz, R 545, N 21)

FDB 9449 ; 38600 Hz (38626 Hz, R 233, N 9)

FDB 6299 ; 38700 Hz (38709 Hz, R 155, N 6)

FDB 13647 ; 38800 Hz (38806 Hz, R 335, N 13)

FDB 10497 ; 38900 Hz (38910 Hz, R 257, N 10)

FDB 8397 ; 39000 Hz (39024 Hz, R 205, N 8)

FDB 7347 ; 39100 Hz (39106 Hz, R 179, N 7)

FDB 4198 ; 39200 Hz (39215 Hz, R 102, N 4)

FDB 7346 ; 39300 Hz (39325 Hz, R 178, N 7)

FDB 8395 ; 39400 Hz (39408 Hz, R 203, N 8)

FDB 10493 ; 39500 Hz (39525 Hz, R 253, N 10)

FDB 4197 ; 39600 Hz (39604 Hz, R 101, N 4)

FDB 11541 ; 39700 Hz (39711 Hz, R 277, N 11)

FDB 8393 ; 39800 Hz (39801 Hz, R 201, N 8)

FDB 16785 ; 39900 Hz (39900 Hz, R 401, N 16)

FDB 3147 ; 40000 Hz (40000 Hz, R 75, N 3)

FDB 13636 ; 40100 Hz (40123 Hz, R 324, N 13)

FDB 7342 ; 40200 Hz (40229 Hz, R 174, N 7)

FDB 5244 ; 40300 Hz (40322 Hz, R 124, N 5)

FDB 4195 ; 40400 Hz (40404 Hz, R 99, N 4)

FDB 16779 ; 40500 Hz (40506 Hz, R 395, N 16)

FDB 8389 ; 40600 Hz (40609 Hz, R 197, N 8)

FDB 9437 ; 40700 Hz (40724 Hz, R 221, N 9)

FDB 4194 ; 40800 Hz (40816 Hz, R 98, N 4)

FDB 9436 ; 40900 Hz (40909 Hz, R 220, N 9)

FDB 8387 ; 41000 Hz (41025 Hz, R 195, N 8)

FDB 16773 ; 41100 Hz (41131 Hz, R 389, N 16)

FDB 15724 ; 41200 Hz (41208 Hz, R 364, N 15)

FDB 5241 ; 41300 Hz (41322 Hz, R 121, N 5)

FDB 7337 ; 41400 Hz (41420 Hz, R 169, N 7)

FDB 11529 ; 41500 Hz (41509 Hz, R 265, N 11)

FDB 27249 ; 41600 Hz (41600 Hz, R 625, N 26)

FDB 28295 ; 41700 Hz (41731 Hz, R 647, N 27)

FDB 11527 ; 41800 Hz (41825 Hz, R 263, N 11)

FDB 7335 ; 41900 Hz (41916 Hz, R 167, N 7)

FDB 5239 ; 42000 Hz (42016 Hz, R 119, N 5)

FDB 4191 ; 42100 Hz (42105 Hz, R 95, N 4)

FDB 13620 ; 42200 Hz (42207 Hz, R 308, N 13)

FDB 8381 ; 42300 Hz (42328 Hz, R 189, N 8)

FDB 7333 ; 42400 Hz (42424 Hz, R 165, N 7)

FDB 17808 ; 42500 Hz (42500 Hz, R 400, N 17)

FDB 13617 ; 42600 Hz (42622 Hz, R 305, N 13)

FDB 12569 ; 42700 Hz (42704 Hz, R 281, N 12)

FDB 11521 ; 42800 Hz (42801 Hz, R 257, N 11)

FDB 10473 ; 42900 Hz (42918 Hz, R 233, N 10)

FDB 4189 ; 43000 Hz (43010 Hz, R 93, N 4)

FDB 5236 ; 43100 Hz (43103 Hz, R 116, N 5)

FDB 7330 ; 43200 Hz (43209 Hz, R 162, N 7)

FDB 11518 ; 43300 Hz (43307 Hz, R 254, N 11)

FDB 26176 ; 43400 Hz (43402 Hz, R 576, N 25)

FDB 36644 ; 43500 Hz (43532 Hz, R 804, N 35)

FDB 13610 ; 43600 Hz (43624 Hz, R 298, N 13)

FDB 8375 ; 43700 Hz (43715 Hz, R 183, N 8)

FDB 11515 ; 43800 Hz (43824 Hz, R 251, N 11)

FDB 9421 ; 43900 Hz (43902 Hz, R 205, N 9)

FDB 7327 ; 44000 Hz (44025 Hz, R 159, N 7)

FDB 3140 ; 44100 Hz (44117 Hz, R 68, N 3)

FDB 13606 ; 44200 Hz (44217 Hz, R 294, N 13)

FDB 7326 ; 44300 Hz (44303 Hz, R 158, N 7)

FDB 24070 ; 44400 Hz (44401 Hz, R 518, N 23)

FDB 13604 ; 44500 Hz (44520 Hz, R 292, N 13)

FDB 12557 ; 44600 Hz (44609 Hz, R 269, N 12)

FDB 11510 ; 44700 Hz (44715 Hz, R 246, N 11)

FDB 13602 ; 44800 Hz (44827 Hz, R 290, N 13)

FDB 15694 ; 44900 Hz (44910 Hz, R 334, N 15)

FDB 9416 ; 45000 Hz (45000 Hz, R 200, N 9)

FDB 6277 ; 45100 Hz (45112 Hz, R 133, N 6)

FDB 9415 ; 45200 Hz (45226 Hz, R 199, N 9)

FDB 14645 ; 45300 Hz (45307 Hz, R 309, N 14)

FDB 39749 ; 45400 Hz (45400 Hz, R 837, N 38)

FDB 28241 ; 45500 Hz (45531 Hz, R 593, N 27)

FDB 12551 ; 45600 Hz (45627 Hz, R 263, N 12)

FDB 8367 ; 45700 Hz (45714 Hz, R 175, N 8)

FDB 6275 ; 45800 Hz (45801 Hz, R 131, N 6)

FDB 9412 ; 45900 Hz (45918 Hz, R 196, N 9)

FDB 11503 ; 46000 Hz (46025 Hz, R 239, N 11)

FDB 16731 ; 46100 Hz (46109 Hz, R 347, N 16)

FDB 11502 ; 46200 Hz (46218 Hz, R 238, N 11)

FDB 12547 ; 46300 Hz (46332 Hz, R 259, N 12)

FDB 11501 ; 46400 Hz (46413 Hz, R 237, N 11)

FDB 4182 ; 46500 Hz (46511 Hz, R 86, N 4)

FDB 9409 ; 46600 Hz (46632 Hz, R 193, N 9)

FDB 5227 ; 46700 Hz (46729 Hz, R 107, N 5)

FDB 11499 ; 46800 Hz (46808 Hz, R 235, N 11)

FDB 13589 ; 46900 Hz (46931 Hz, R 277, N 13)

FDB 11498 ; 47000 Hz (47008 Hz, R 234, N 11)

FDB 9407 ; 47100 Hz (47120 Hz, R 191, N 9)

FDB 11497 ; 47200 Hz (47210 Hz, R 233, N 11)

FDB 15677 ; 47300 Hz (47318 Hz, R 317, N 15)

FDB 11496 ; 47400 Hz (47413 Hz, R 232, N 11)

FDB 19856 ; 47500 Hz (47500 Hz, R 400, N 19)

FDB 3135 ; 47600 Hz (47619 Hz, R 63, N 3)

FDB 20899 ; 47700 Hz (47732 Hz, R 419, N 20)

FDB 11494 ; 47800 Hz (47826 Hz, R 230, N 11)

FDB 8359 ; 47900 Hz (47904 Hz, R 167, N 8)

FDB 6269 ; 48000 Hz (48000 Hz, R 125, N 6)

FDB 9403 ; 48100 Hz (48128 Hz, R 187, N 9)

FDB 15671 ; 48200 Hz (48231 Hz, R 311, N 15)

FDB 10447 ; 48300 Hz (48309 Hz, R 207, N 10)

FDB 20893 ; 48400 Hz (48426 Hz, R 413, N 20)

FDB 13580 ; 48500 Hz (48507 Hz, R 268, N 13)

FDB 7312 ; 48600 Hz (48611 Hz, R 144, N 7)

FDB 15668 ; 48700 Hz (48701 Hz, R 308, N 15)

FDB 24023 ; 48800 Hz (48832 Hz, R 471, N 23)

FDB 9400 ; 48900 Hz (48913 Hz, R 184, N 9)

FDB 5222 ; 49000 Hz (49019 Hz, R 102, N 5)

FDB 11488 ; 49100 Hz (49107 Hz, R 224, N 11)

FDB 16709 ; 49200 Hz (49230 Hz, R 325, N 16)

FDB 11487 ; 49300 Hz (49327 Hz, R 223, N 11)

FDB 13575 ; 49400 Hz (49429 Hz, R 263, N 13)

FDB 5221 ; 49500 Hz (49504 Hz, R 101, N 5)

FDB 13574 ; 49600 Hz (49618 Hz, R 262, N 13)

FDB 9397 ; 49700 Hz (49723 Hz, R 181, N 9)

FDB 13573 ; 49800 Hz (49808 Hz, R 261, N 13)

FDB 26101 ; 49900 Hz (49900 Hz, R 501, N 25)

FDB 3132 ; 50000 Hz (50000 Hz, R 60, N 3)

FDB 19835 ; 50100 Hz (50131 Hz, R 379, N 19)

FDB 11483 ; 50200 Hz (50228 Hz, R 219, N 11)

FDB 8351 ; 50300 Hz (50314 Hz, R 159, N 8)

FDB 6263 ; 50400 Hz (50420 Hz, R 119, N 6)

FDB 5219 ; 50500 Hz (50505 Hz, R 99, N 5)

FDB 4175 ; 50600 Hz (50632 Hz, R 79, N 4)

FDB 7306 ; 50700 Hz (50724 Hz, R 138, N 7)

FDB 19830 ; 50800 Hz (50802 Hz, R 374, N 19)

FDB 11480 ; 50900 Hz (50925 Hz, R 216, N 11)

FDB 5218 ; 51000 Hz (51020 Hz, R 98, N 5)

FDB 16697 ; 51100 Hz (51118 Hz, R 313, N 16)

FDB 17740 ; 51200 Hz (51204 Hz, R 332, N 17)

FDB 28174 ; 51300 Hz (51330 Hz, R 526, N 27)

FDB 9391 ; 51400 Hz (51428 Hz, R 175, N 9)

FDB 12521 ; 51500 Hz (51502 Hz, R 233, N 12)

FDB 8347 ; 51600 Hz (51612 Hz, R 155, N 8)

FDB 3130 ; 51700 Hz (51724 Hz, R 58, N 3)

FDB 10433 ; 51800 Hz (51813 Hz, R 193, N 10)

FDB 15649 ; 51900 Hz (51903 Hz, R 289, N 15)

FDB 9389 ; 52000 Hz (52023 Hz, R 173, N 9)

FDB 11475 ; 52100 Hz (52132 Hz, R 211, N 11)

FDB 13561 ; 52200 Hz (52208 Hz, R 249, N 13)

FDB 9388 ; 52300 Hz (52325 Hz, R 172, N 9)

FDB 12517 ; 52400 Hz (52401 Hz, R 229, N 12)

FDB 21904 ; 52500 Hz (52500 Hz, R 400, N 21)

FDB 3129 ; 52600 Hz (52631 Hz, R 57, N 3)

FDB 29203 ; 52700 Hz (52730 Hz, R 531, N 28)

FDB 14601 ; 52800 Hz (52830 Hz, R 265, N 14)

FDB 10429 ; 52900 Hz (52910 Hz, R 189, N 10)

FDB 7300 ; 53000 Hz (53030 Hz, R 132, N 7)

FDB 17728 ; 53100 Hz (53125 Hz, R 320, N 17)

FDB 14599 ; 53200 Hz (53231 Hz, R 263, N 14)

FDB 26069 ; 53300 Hz (53304 Hz, R 469, N 25)

FDB 18769 ; 53400 Hz (53412 Hz, R 337, N 18)

```
        FDB 16683  ; 53500 Hz (53511 Hz, R  299, N 16)
        FDB 17725  ; 53600 Hz (53627 Hz, R  317, N 17)
        FDB 13554  ; 53700 Hz (53719 Hz, R  242, N 13)
        FDB 12511  ; 53800 Hz (53811 Hz, R  223, N 12)
        FDB 11468  ; 53900 Hz (53921 Hz, R  204, N 11)
        FDB 28148  ; 54000 Hz (54000 Hz, R  500, N 27)
        FDB 19807  ; 54100 Hz (54131 Hz, R  351, N 19)
        FDB 9382   ; 54200 Hz (54216 Hz, R  166, N  9)
        FDB 17721  ; 54300 Hz (54313 Hz, R  313, N 17)
        FDB 8339   ; 54400 Hz (54421 Hz, R  147, N  8)
        FDB 23974  ; 54500 Hz (54502 Hz, R  422, N 23)
        FDB 13550  ; 54600 Hz (54621 Hz, R  238, N 13)
        FDB 11465  ; 54700 Hz (54726 Hz, R  201, N 11)
        FDB 21887  ; 54800 Hz (54830 Hz, R  383, N 21)
        FDB 14591  ; 54900 Hz (54902 Hz, R  255, N 14)
        FDB 11464  ; 55000 Hz (55000 Hz, R  200, N 11)
        FDB 7295   ; 55100 Hz (55118 Hz, R  127, N  7)
        FDB 9379   ; 55200 Hz (55214 Hz, R  163, N  9)
        FDB 13547  ; 55300 Hz (55319 Hz, R  235, N 13)
        FDB 20841  ; 55400 Hz (55401 Hz, R  361, N 20)
        FDB 58353  ; 55500 Hz (55500 Hz, R 1009, N 56)
;For maximum error = 33.0 minimum fsync = 1958.8 @ 33300 Hz
;MAKETAB -- 0 errors
```

```
/* PLL error analysis program */


#include <stdio.h>
#include <math.h>

#define MAXERR  1.00

#define FBEGIN  30000
#define FEND    55500

#define FREF    2.0e6
#define DIVMAX  16384



void   main( int  argc, char  *argv[] )
{
   float  fsync, factual, ferr, minsync, maxerr;
   unsigned  fideal, divR, divN, minfreq;


   if( argc < 2 )
   {
      for( maxerr = 1.0; maxerr < 100.0; maxerr += 1.0 )
      {
         minsync = FREF;

         for( fideal = FBEGIN; fideal <= FEND; fideal += 100 )
         {
            for( divN = 3; divN <= DIVMAX; divN++ )
            {
               fsync = (float) fideal * 2.0 / divN;
```

```
                divR = FREF / fsync;
                factual = FREF / (float) divR * (float) divN / 2.0;
                ferr = (float) fideal - factual;

                if( fsync < minsync )
                {
                  minsync = fsync;
                  minfreq = fideal;
                }

                if( fabs(ferr) <= maxerr ) break;
              }
            }

            printf( "For maximum error = %4.1f minimum fsync = %6.1f @
%u Hz\n", maxerr, minsync, minfreq );
          }
        }
        else if( argc < 3 && sscanf(argv[1], "%f", &maxerr) )
        {
          minsync = FREF;

          printf( "Ideal\tDiv R\t fSync\tDiv N\tActual\t  Error\n" );

          for( fideal = FBEGIN; fideal <= FEND; fideal += 100 )
          {
            for( divN = 3; divN <= DIVMAX; divN++ )
            {
              fsync = (float) fideal * 2.0 / (float) divN;
              divR = FREF / fsync;
              factual = FREF / (float) divR * (float) divN / 2.0;
              ferr = (float) fideal - factual;
```

```
            if( fsync < minsync )
            {
              minsync = fsync;
              minfreq = fideal;
            }

            if( fabs(ferr) <= maxerr )  break;
          }

          printf( "%u\t%5u\t%7.1f\t%5u\t%7.1f\t%7.1f\n", fideal, divR,
fsync, divN, factual, ferr );
        }

      printf( "For maximum error = %4.1f minimum fsync = %6.1f @ %u
Hz\n", maxerr, minsync, minfreq );
    }
    else
    {
      fprintf( stderr, "usage: %s [maxerror]\n", argv[0] );
      exit( 1 );
    }
}
```

```
#########################################################
########################################
#
#  MAKEFILE -- InfraRed Project Makefile.
#
#
#  BASSEL ATALLAH Initial version
#  Ver 1.00
#


SRC=ir.inc 68hc11.inc 811e2.inc irmain.asm irparse.asm irtab.asm
irio.asm


default: p.bin


p.bin: ir.s19 convert.ir
   convert < convert.ir


ir.s19: $(SRC)
   as11new $(SRC) -l c > ir.lst


# irtab.dat should be one of the tables output from the PLL error analysis
pgm
irtab.asm: irtab.dat maketab.awk
   gawk -f maketab.awk irtab.dat > irtab.asm
```

```
#########################################################
#########################################
#
#  MAKETAB.AWK -- AWK program to create PLL data table.
#
#
#  Convert output of frequency analysis program into ASM source for
#  PLL frequency synthesizer look-up table.
#
#  BASSEL ATALLAH
#  Ver 1.00
#
#########################################################
#########################################

BEGIN {
   MAX_R = 1023
   MAX_N = 63
   errors = 0

   print "; Data table created by MAKETAB.AWK Ver 1.00"
   print ";"
   print "; Upper 6 bits are N divisor, lower 10 are the R divisor"
   print
   print "PLLTAB"
}

{
  if( int($1) ) {
     R = int($2)
     N = int($4)
```

```
      printf "\tFDB %u\t; %s Hz (%5u Hz, R %4u, N %2u)\n", N * 1024
+ R, $1, $5, R, N

      if( R > MAX_R )
      {
        printf "ERROR @ %uHz: R > %u (%u)\n", $1, MAX_R, R
        ++errors
      }
      if( N > MAX_N )
      {
        printf "ERROR @ %uHz: N > %u (%u)\n", $1, MAX_N, N
        ++errors
      }
    }

}

/maximum/   { print ";" $0 }


END {
  print ";MAKETAB -- " errors " errors"

  exit errors
}
```

```
@echo off
cls
echo                    TECHNOLOGICAL ARTS

echo                 ADAPT-11 Dowloading Program
echo                 ===========================
echo          1) Make sure ADAPT-11 is connected to COM1:
echo          2) Place switch in BOOT position. (Other switch NRML.)
echo          3) Make sure ADAPT-11 power is on.
echo          4) Press ADAPT-11 RESET button.

echo Press <ctrl>^C to abort, or...
pause
echo          Downloading now... (takes approximately 80 seconds)
@echo off
mode com1: 1200,N,8,1
copy BOOT8.BIN com1: /b
mode com1: 300,n,8,1
copy P.BIN com1: /b

echo          ADAPT-11 downloading complete.

echo          Place ADAPT-11 switch in RUN mode.

echo          Press ADAPT-11 RESET button.

echo          ADAPT-11 is now running your program!
•
```

APPENDIX B

The frequency synthesizer table is found in this appendix.

Ideal	Div R	fSync	Div N	Actual	Error
30000	100	20000.0	3	30000.0	0.0
30100	166	12040.0	5	30120.5	-20.5
30200	298	6711.1	9	30201.3	-1.3
30300	99	20200.0	3	30303.0	-3.0
30400	263	7600.0	8	30418.2	-18.2
30500	295	6777.8	9	30508.5	-8.5
30600	98	20400.0	3	30612.2	-12.2
30700	228	8771.4	7	30701.8	-1.8
30800	292	6844.4	9	30821.9	-21.9
30900	97	20600.0	3	30927.8	-27.8
31000	129	15500.0	4	31007.8	-7.8
31100	225	8885.7	7	31111.1	-11.1
31200	641	3120.0	20	31201.2	-1.2
31300	383	5216.7	12	31331.6	-31.6
31400	191	10466.7	6	31413.6	-13.6
31500	222	9000.0	7	31531.5	-31.5
31600	253	7900.0	8	31620.6	-20.6
31700	347	5763.6	11	31700.3	-0.3
31800	220	9085.7	7	31818.2	-18.2
31900	94	21266.7	3	31914.9	-14.9
32000	125	16000.0	4	32000.0	0.0
32100	218	9171.4	7	32110.1	-10.1
32200	559	3577.8	18	32200.4	-0.4
32300	433	4614.3	14	32332.6	-32.6
32400	185	10800.0	6	32432.4	-32.4
32500	123	16250.0	4	32520.3	-20.3
32600	92	21733.3	3	32608.7	-8.7
32700	214	9342.9	7	32710.3	-10.3
32800	396	5046.2	13	32828.3	-28.3
32900	243	8225.0	8	32921.8	-21.8
33000	212	9428.6	7	33018.9	-18.9
33100	151	13240.0	5	33112.6	-12.6
33200	271	7377.8	9	33210.3	-10.3
33300	1021	1958.8	34	33300.7	-0.7
33400	359	5566.7	12	33426.2	-26.2
33500	179	11166.7	6	33519.6	-19.6

33600	119	16800.0	4	33613.4	-13.4
33700	89	22466.7	3	33707.9	-7.9
33800	207	9657.1	7	33816.4	-16.4
33900	501	3988.2	17	33932.1	-32.1
34000	147	13600.0	5	34013.6	-13.6
34100	293	6820.0	10	34129.7	-29.7
34200	263	7600.0	9	34220.5	-20.5
34300	204	9800.0	7	34313.7	-13.7
34400	436	4586.7	15	34403.7	-3.7
34500	695	2875.0	24	34532.4	-32.4
34600	231	8650.0	8	34632.0	-32.0
34700	144	13880.0	5	34722.2	-22.2
34800	201	9942.9	7	34825.9	-25.9
34900	315	6345.5	11	34920.6	-20.6
35000	200	10000.0	7	35000.0	0.0
35100	427	4680.0	15	35128.8	-28.8
35200	142	14080.0	5	35211.3	-11.3
35300	368	5430.8	13	35326.1	-26.1
35400	367	5446.2	13	35422.3	-22.3
35500	169	11833.3	6	35503.0	-3.0
35600	337	5933.3	12	35608.3	-8.3
35700	84	23800.0	3	35714.3	-14.3
35800	307	6509.1	11	35830.6	-30.6
35900	167	11966.7	6	35928.1	-28.1
36000	250	8000.0	9	36000.0	0.0
36100	277	7220.0	10	36101.1	-1.1
36200	138	14480.0	5	36231.9	-31.9
36300	303	6600.0	11	36303.6	-3.6
36400	302	6618.2	11	36423.8	-23.8
36500	219	9125.0	8	36529.7	-29.7
36600	273	7320.0	10	36630.0	-30.0
36700	354	5646.2	13	36723.2	-23.2
36800	163	12266.7	6	36809.8	-9.8
36900	271	7380.0	10	36900.4	-0.4
37000	1000	2000.0	37	37000.0	0.0
37100	404	4946.7	15	37128.7	-28.7
37200	215	9300.0	8	37209.3	-9.3
37300	134	14920.0	5	37313.4	-13.4

37400	294	6800.0	11	37415.0	-15.0
37500	80	25000.0	3	37500.0	0.0
37600	319	6266.7	12	37617.6	-17.6
37700	557	3590.5	21	37702.0	-2.0
37800	238	8400.0	9	37815.1	-15.1
37900	211	9475.0	8	37914.7	-14.7
38000	263	7600.0	10	38022.8	-22.8
38100	341	5861.5	13	38123.2	-23.2
38200	157	12733.3	6	38216.6	-16.6
38300	261	7660.0	10	38314.2	-14.2
38400	625	3200.0	24	38400.0	0.0
38500	545	3666.7	21	38532.1	-32.1
38600	233	8577.8	9	38626.6	-26.6
38700	155	12900.0	6	38709.7	-9.7
38800	335	5969.2	13	38806.0	-6.0
38900	257	7780.0	10	38910.5	-10.5
39000	205	9750.0	8	39024.4	-24.4
39100	179	11171.4	7	39106.1	-6.1
39200	102	19600.0	4	39215.7	-15.7
39300	178	11228.6	7	39325.8	-25.8
39400	203	9850.0	8	39408.9	-8.9
39500	253	7900.0	10	39525.7	-25.7
39600	101	19800.0	4	39604.0	-4.0
39700	277	7218.2	11	39711.2	-11.2
39800	201	9950.0	8	39801.0	-1.0
39900	401	4987.5	16	39900.2	-0.2
40000	75	26666.7	3	40000.0	0.0
40100	324	6169.2	13	40123.5	-23.5
40200	174	11485.7	7	40229.9	-29.9
40300	124	16120.0	5	40322.6	-22.6
40400	99	20200.0	4	40404.0	-4.0
40500	395	5062.5	16	40506.3	-6.3
40600	197	10150.0	8	40609.1	-9.1
40700	221	9044.4	9	40724.0	-24.0
40800	98	20400.0	4	40816.3	-16.3
40900	220	9088.9	9	40909.1	-9.1
41000	195	10250.0	8	41025.6	-25.6
41100	389	5137.5	16	41131.1	-31.1

41200	364	5493.3	15	41208.8	-8.8
41300	121	16520.0	5	41322.3	-22.3
41400	169	11828.6	7	41420.1	-20.1
41500	265	7545.5	11	41509.4	-9.4
41600	625	3200.0	26	41600.0	0.0
41700	647	3088.9	27	41731.1	-31.1
41800	263	7600.0	11	41825.1	-25.1
41900	167	11971.4	7	41916.2	-16.2
42000	119	16800.0	5	42016.8	-16.8
42100	95	21050.0	4	42105.3	-5.3
42200	308	6492.3	13	42207.8	-7.8
42300	189	10575.0	8	42328.0	-28.0
42400	165	12114.3	7	42424.2	-24.2
42500	400	5000.0	17	42500.0	0.0
42600	305	6553.8	13	42622.9	-22.9
42700	281	7116.7	12	42704.6	-4.6
42800	257	7781.8	11	42801.6	-1.6
42900	233	8580.0	10	42918.5	-18.5
43000	93	21500.0	4	43010.8	-10.8
43100	116	17240.0	5	43103.4	-3.4
43200	162	12342.9	7	43209.9	-9.9
43300	254	7872.7	11	43307.1	-7.1
43400	576	3472.0	25	43402.8	-2.8
43500	804	2485.7	35	43532.3	-32.3
43600	298	6707.7	13	43624.2	-24.2
43700	183	10925.0	8	43715.8	-15.8
43800	251	7963.6	11	43824.7	-24.7
43900	205	9755.6	9	43902.4	-2.4
44000	159	12571.4	7	44025.2	-25.2
44100	68	29400.0	3	44117.6	-17.6
44200	294	6800.0	13	44217.7	-17.7
44300	158	12657.1	7	44303.8	-3.8
44400	518	3860.9	23	44401.5	-1.5
44500	292	6846.2	13	44520.5	-20.5
44600	269	7433.3	12	44609.7	-9.7
44700	246	8127.3	11	44715.4	-15.4
44800	290	6892.3	13	44827.6	-27.6
44900	334	5986.7	15	44910.2	-10.2

45000	200	10000.0	9	45000.0	0.0
45100	133	15033.3	6	45112.8	-12.8
45200	199	10044.4	9	45226.1	-26.1
45300	309	6471.4	14	45307.4	-7.4
45400	837	2389.5	38	45400.2	-0.2
45500	593	3370.4	27	45531.2	-31.2
45600	263	7600.0	12	45627.4	-27.4
45700	175	11425.0	8	45714.3	-14.3
45800	131	15266.7	6	45801.5	-1.5
45900	196	10200.0	9	45918.4	-18.4
46000	239	8363.6	11	46025.1	-25.1
46100	347	5762.5	16	46109.5	-9.5
46200	238	8400.0	11	46218.5	-18.5
46300	259	7716.7	12	46332.0	-32.0
46400	237	8436.4	11	46413.5	-13.5
46500	86	23250.0	4	46511.6	-11.6
46600	193	10355.6	9	46632.1	-32.1
46700	107	18680.0	5	46729.0	-29.0
46800	235	8509.1	11	46808.5	-8.5
46900	277	7215.4	13	46931.4	-31.4
47000	234	8545.5	11	47008.5	-8.5
47100	191	10466.7	9	47120.4	-20.4
47200	233	8581.8	11	47210.3	-10.3
47300	317	6306.7	15	47318.6	-18.6
47400	232	8618.2	11	47413.8	-13.8
47500	400	5000.0	19	47500.0	0.0
47600	63	31733.3	3	47619.0	-19.0
47700	419	4770.0	20	47732.7	-32.7
47800	230	8690.9	11	47826.1	-26.1
47900	167	11975.0	8	47904.2	-4.2
48000	125	16000.0	6	48000.0	0.0
48100	187	10688.9	9	48128.3	-28.3
48200	311	6426.7	15	48231.5	-31.5
48300	207	9660.0	10	48309.2	-9.2
48400	413	4840.0	20	48426.1	-26.1
48500	268	7461.5	13	48507.5	-7.5
48600	144	13885.7	7	48611.1	-11.1
48700	308	6493.3	15	48701.3	-1.3

48800	471	4243.5	23	48832.3	-32.3
48900	184	10866.7	9	48913.0	-13.0
49000	102	19600.0	5	49019.6	-19.6
49100	224	8927.3	11	49107.1	-7.1
49200	325	6150.0	16	49230.8	-30.8
49300	223	8963.6	11	49327.4	-27.4
49400	263	7600.0	13	49429.7	-29.7
49500	101	19800.0	5	49504.9	-4.9
49600	262	7630.8	13	49618.3	-18.3
49700	181	11044.4	9	49723.8	-23.8
49800	261	7661.5	13	49808.4	-8.4
49900	501	3992.0	25	49900.2	-0.2
50000	60	33333.3	3	50000.0	0.0
50100	379	5273.7	19	50131.9	-31.9
50200	219	9127.3	11	50228.3	-28.3
50300	159	12575.0	8	50314.5	-14.5
50400	119	16800.0	6	50420.2	-20.2
50500	99	20200.0	5	50505.1	-5.1
50600	79	25300.0	4	50632.9	-32.9
50700	138	14485.7	7	50724.6	-24.6
50800	374	5347.4	19	50802.1	-2.1
50900	216	9254.5	11	50925.9	-25.9
51000	98	20400.0	5	51020.4	-20.4
51100	313	6387.5	16	51118.2	-18.2
51200	332	6023.5	17	51204.8	-4.8
51300	526	3800.0	27	51330.8	-30.8
51400	175	11422.2	9	51428.6	-28.6
51500	233	8583.3	12	51502.1	-2.1
51600	155	12900.0	8	51612.9	-12.9
51700	58	34466.7	3	51724.1	-24.1
51800	193	10360.0	10	51813.5	-13.5
51900	289	6920.0	15	51903.1	-3.1
52000	173	11555.6	9	52023.1	-23.1
52100	211	9472.7	11	52132.7	-32.7
52200	249	8030.8	13	52208.8	-8.8
52300	172	11622.2	9	52325.6	-25.6
52400	229	8733.3	12	52401.7	-1.7
52500	400	5000.0	21	52500.0	0.0

52600	57	35066.7	3	52631.6	-31.6
52700	531	3764.3	28	52730.7	-30.7
52800	265	7542.9	14	52830.2	-30.2
52900	189	10580.0	10	52910.1	-10.1
53000	132	15142.9	7	53030.3	-30.3
53100	320	6247.1	17	53125.0	-25.0
53200	263	7600.0	14	53231.9	-31.9
53300	469	4264.0	25	53304.9	-4.9
53400	337	5933.3	18	53412.5	-12.5
53500	299	6687.5	16	53511.7	-11.7
53600	317	6305.9	17	53627.8	-27.8
53700	242	8261.5	13	53719.0	-19.0
53800	223	8966.7	12	53811.7	-11.7
53900	204	9800.0	11	53921.6	-21.6
54000	500	4000.0	27	54000.0	0.0
54100	351	5694.7	19	54131.1	-31.1
54200	166	12044.4	9	54216.9	-16.9
54300	313	6388.2	17	54313.1	-13.1
54400	147	13600.0	8	54421.8	-21.8
54500	422	4739.1	23	54502.4	-2.4
54600	238	8400.0	13	54621.8	-21.8
54700	201	9945.5	11	54726.4	-26.4
54800	383	5219.0	21	54830.3	-30.3
54900	255	7842.9	14	54902.0	-2.0
55000	200	10000.0	11	55000.0	0.0
55100	127	15742.9	7	55118.1	-18.1
55200	163	12266.7	9	55214.7	-14.7
55300	235	8507.7	13	55319.1	-19.1
55400	361	5540.0	20	55401.7	-1.7
55500	1009	1982.1	56	55500.5	-0.5

For maximum error = 33.0 minimum fsync = 1958.8 @ 33300 Hz

APPENDIX C

Customed commands and all related information for all TV models is found in this appendix.

Brand Name: MAGNAVOX1 Model: UR-16

Frequency: 35.7 kHz

Time Table Time 1 : 0.889 ms

Time 2 : 1.778 ms

Time 3 : 88.0 ms

Header: 1112

Custom Code: 1111111111

Commands:

Power: 1112112111

Vol Up: 12111111111

Vol Dn: 1211111121

Chan Up: 12111111111

Chan Dn: 1211111121

Mute: 1112112111

Alt Chan: 1211112111

Plus: 11122211

Minus: 122112111

Cur Dn: 1211112111

Key 0: 111111111111

Key 1: 11111111121

Key 2: 1111111221

Key 3: 11111112111

Key 4: 1111122111

Key 5: 111112221

Key 6: 1111121121

Key 7: 11111211111

Key 8: 1112211111

Key 9: 111221121

Cur Up: 1211112111

Sleep: 121121121

Ch Guide: 11111111121

PIP: 111112221

PIP Pos: 1221111111

Swap: 1112211111

Freeze: 111221121

Preview: 111211221

Strobe: 12211221

Menu: 122111121

Brand Name: MAGNAVOX1 Model: RC-5

Frequency: 35.7 kHz

Time Table Time 1 : 0.889 ms

Time 2 : 1.778 ms

Time 3 : 88.0 ms

Header: 1112

Custom Code: 1111111111

Commands:

Power: 1112112111

Vol Up: 1221111111

Vol Dn: 122111121

Chan Up: 12111111111

Chan Dn: 1211111121

Mute: 1112112111

Alt Chan: 121111221

Plus: 12222111

Minus: 122112111

Cur Dn: 1111122111

Key 0: 111111111111

Key 1: 11111111121

Key 2: 1111111221

Key 3: 11111112111

Key 4: 1111122111

Key 5: 111112221

Key 6: 1111121121

Key 7: 11111211111

Key 8: 1112211111

Key 9: 111221121

Cur Up: Not Assigned

Sleep: 121121121

Ch Guide: 11111111121

PIP: 111112221

PIP Pos: 1221111111

Swap: 1112211111

Freeze: 111221121

Preview: 111211221

Strobe: 12211221

Menu: 122111121

Brand Name: MAGNAVOX2 Model: RC-4B

Frequency: 45.5 kHz

Time Table Time 1 : 2.5 ms

Time 2 : 0.4 ms

Time 3 : 2.9 ms

Time 4 : 5.4 ms

Time 5 : 8.3 ms

Time 5 : 25.0 ms

Header: Not assigned

Custom Code:1212521252125212

Commands:

Power: 121242424242

Vol Up: 4242424242

Vol Dn: 421212424242

Chan Up: 424242421212

Chan Dn: 42121242421212

Mute: 4212121212411212

Alt Chan: 421212121212121212

Plus: Not Assigned

Minus: Not Assigned

Cur Dn: 1212421212121242

Key 0: 4212124212121212

Key 1: 42421212121242

Key 2: 424242121242

Key 3: 424212124242

Key 4: 4212121212121242

Key 5: 42121242121242

Key 6: 42121212124242

Key 7: 4242121212121212

Key 8: 42424212121212

Key 9: 42421212421212

Cur Up: Not Assigned

Sleep: 12124212124242

Ch Guide: 4212121212121212

PIP: Not Assigned

PIP Pos: Not Assigned

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: Not Assigned

Brand Name: MAGNAVOX2 Model: RC-4A

Frequency: 45.5 kHz

Time Table Time 1 : 2.5 ms

Time 2 : 0.4 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 333334333333333334343334343434343434

Commands:

Power: 1222133312211331

Vol Up: 22233333111132

Vol Dn: 11222233331112113

Chan Up: 11133211113222

Chan Dn: 1113323331222

Mute: 111223333112

Alt Chan: 11123322212323223

Plus: 1232222111223321

Minus: Not Assigned

Cur Dn: Not Assigned

Key 0: 122211231111

Key 1: 112221331232

Key 2: 222311231232

Key 3: 222331231221

Key 4: 223311111111

Key 5: 231111111111

Key 6: 221231231231

Key 7: 211123212221

Key 8: 212312311111

Key 9: 223121111111

Cur Up: Not Assigned

Sleep: 2212312312321212

Ch Guide: 33312312312223

PIP: Not Assigned

PIP Pos: Not Assigned

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: 23122222112312313

Brand Name: GOLDSTAR Model: GOLDSTAR

Frequency: 37.5 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333433333333333434333434343434

Commands:

Power: 33333334333333333434343334343434

Vol Up: 33343333333333333433343434343434

Vol Dn: 34343333333333333333343434343434

Chan Up: 33333333333333333434343434343434

Chan Dn: 34333333333333333334343434343434

Mute:34333334333333333334343334343434

Alt Chan: Not Assigned

Plus: Not Assigned

Minus: Not Assigned

Cur Dn: Not Assigned

Key 0: 33333333433333333434343433343434

Key 1: 34333333433333333334343433343434

Key 2: 33343333343333333433343433343434

Key 3: 34343333343333333333343433343434

Key 4: 33333433343333333434333433343434

Key 5: 343334333433333333334333433343434

Key 6: 333434333433333334333334333343434

Key 7: 343434333433333333333333433343434

Key 8: 333333343433333334343433333343434

Key 9: 343333343433333333343433333343434

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: Not Assigned

PIP: Not Assigned

PIP Pos: Not Assigned

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: Not Assigned

Brand Name: SAMSUNG Model: SAMSUNG

Frequency: 38.4 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333433333333333434333434343434

Commands:

Power: 234444234234

Vol Up: 444232343432

Vol Dn: 34343434422

Chan Up: 4343434222

Chan Dn: 43434343422

Mute: 34444233434322

Alt Chan: Not Assigned

Plus: Not Assigned

Minus: Not Assigned

Cur Dn: Not Assigned

Key 0: 42424242333

Key 1: 243442343433

Key 2: 43423433332

Key 3: 43434343422

Key 4: 23434434342

Key 5: 44442343422

Key 6: 43234424242

Key 7: 23423423333

Key 8: 44424342442

Key 9: 24242433334

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: Not Assigned

PIP: Not Assigned

PIP Pos: Not Assigned

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: 324244234222

Brand Name: SANYO Model: SANYO

Frequency: 37.5 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333334343433333434343333333434

Commands:

Power: 33343333343333333433343433343434

Vol Up: 33343434333333333433333334343434

Vol Dn: 343434343333333333333333334343434

Chan Up: 3334333433333333343343334343434

Chan Dn: 34343334333333333333343334343434

Mute: 333333343433333334343433333343434

Alt Chan: 3433333434333333333434333333434434

Plus: 333433343433333334333433333343434

Minus: 34343333343333333333343433343434

Cur Dn: Not Assigned

Key 0: 33333333333333333434343434343434

Key 1: 34333333333333333334343434343434

Key 2: 33343333333333333433343434343434

Key 3: 343433333333333333333343434343434

Key 4: 333334333333333334343334343434343434

Key 5: 343334333333333333334333434343434

Key 6: 333434333333333334333333434343434

Key 7: 343434333333333333333333434343434

Key 8: 333333343333333334343334343434

Key 9: 343333343333333333343334343434

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Gu ide:333334343333333334343333334343434

PIP: 34333334343333333334343333343434

PIP Pos: 333434343333333334333333334343434

Swap: Not Assigned

Freeze :343334343333333333343333334343434

Preview: Not Assigned

Strobe: Not Assigned

Menu: 34343334343333333333343333343434

Brand Name: NEC Model: NAP

Frequency: 37.5 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333433333333333434333434343434

Commands: Data + Data

Power: 33333334333333333434343334343434

Vol Up: 33343333333333333433343434343434

Vol Dn: 34343333333333333333343434343434

Chan Up: 33333333333333333434343434343434

Chan Dn: 34333333333333333334343434343434

Mute: 33343334333333333433343334343434

Alt Chan: 333433343433333334333433333343434

Plus: 33333433333333333434333434343434

Minus: 34333334333333333334343334343434

Cur Dn: 343433343333333333333343334343434

Key 0: 33333333343333333434343433343434

Key 1: 34333333343333333334343433343434

Key 2: 33343333343333333433343433343434

Key 3: 34343333343333333333343433343434

Key 4: 33333433343333333434333433343434

Key 5: 34333433343333333334333433343434

Key 6: 33343433343333333433333433343434

Key 7: 34343433343333333333333433343434

Key 8: 33333334343333333434343333343434

Key 9: 34333334343333333334343333343434

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: 34343334343333332222232222232323

PIP: Not Assigned

PIP Pos: 33343434333333332322222223232323

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: 33343334343333333433343333343434

Brand Name: SHARP Model: SHARP

Frequency: 37.5 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333333333333333434343434343434

Commands:

Power: 34343334333333333333343334343434

Vol Up :33343434333333333433333334343434

Vol Dn: 343434343333333333333333334343434

Chan Up :333334343333333334343333334343434

Chan Dn: 343334343333333333343333334343434

Mute: 34333333343333333334343433343434

Alt Chan: 343433333433333333333433343434

Plus: Not Assigned

Minus: 33333434343333333434333333343434

Cur Dn: Not Assigned

Key 0: 343333343333333333343343334343434

Key 1: 33333333333333333434343434343434

Key 2: 34333333333333333334343434343434

Key 3: 33343333333333333433343434343434

Key 4: 34343333333333333333343434343434

Key 5: 3333343333333333343433434343434

Key 6: 3433343333333333333434334343434

Key 7: 3334343333333333343333434343434

Key 8: 3434343333333333333333434343434

Key 9: 3333333433333333343434334343434

Cur Up: Not Assigned

Sleep: 343334333433333333343334333343434

Ch Guide: 34343433343333333333333433343434

PIP: 33333433343333333434334333343434

PIP Pos: Not Assigned

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: 33343433343333333433333433343434

Brand Name: HITACHI Model: HITACHI

Frequency: 37.5 kHz

Time Table Time 1 : 9.0 ms

Time 2 : 4.5 ms

Time 3 : 0.55 ms

Time 4 : 1.7 ms

Time 5 : 2.3 ms

Header: 12

Custom Code: 33333333343334333434343433343334

Commands:

Power: 34343433343333333333333433343434

Vol Up: 33343333343333333433343433343434

Vol Dn: 34333433343333333334333433343434

Chan Up: 343333343433333333343433333343434

Chan Dn: 33333334343333333434343333343434

Mute: 34343334333333333333343334343434

Alt Chan: 333433343333333334333433343434343434

Plus: Not Assigned

Minus: Not Assigned

Cur Dn: 343333343333333333343433343434343434

Key 0: 33333434333333333434333334343434

Key 1: 34333434333333333334333334343434

Key 2: 33343434333333333433333334343434

Key 3: 34343434333333333333333334343434

Key 4: 33333434343333333434333333343434

Key 5: 343334343433333333434333333343434

Key 6: 333434343433333334333333333343434

Key 7: 343434343433333333333333333343434

Key 8: 333334333333333334343334343434343

Key 9: 343334333333333333434333434343434

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: 333333333433333334343433343434

PIP: Not Assigned

PIP Pos: 343333333433333333434343343434

Swap: Not Assigned

Freeze: Not Assigned

Preview: Not Assigned

Strobe: Not Assigned

Menu: Not Assigned

Brand Name: ZENITH Model: ZENITH

Frequency: 40.0 kHz

Time Table Time 1 : 0.50 ms

Time 2 : 5.00 ms

Time 3 : 4.00 ms

Time 4 : 95.0 ms

Time 5 : 0.0 ms

Header: Not Assigned

Custom Code: Not Assigned

Commands:

Power: 1113121113111312111312111312121113

Vol Up: 11131211131113121211131111312121113

Vol Dn: 1113121113121113121113111312121113

Chan Up: 111312111311131212111312111311312

Chan Dn: 11131211131211131211131211131111312

Mute: 1113121113111312121113111312111312

Alt Chan: 11131113121113121211131111312111312

Plus: 1113111312111312111312121113111312

Minus: 11131211131211131211131211131121113

Cur Dn: 1113121113111312111312121113111312

Key 0: 11131113121211131211131211131121113

Key 1: 11131113121211131211131211131111312

Key 2: 11131113121211131211131111312121113

Key 3: 11131113121211131211131111312111312

Key 4: 11131113121211131111312121113121113

Key 5: 11131113121211131113121211131113 12

Key 6: 11131113121211131113121113121211 13

Key 7: 11131113121211131113121113121113 12

Key 8: 11131113121113121211131211131211 13

Key 9: 11131113121113121211131211131113 12

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: 11131113121113111312121211131211 13

PIP: 11131113121113121113121113121113 12

PIP Pos: 11131211131211131211131113121113 12

Swap: 11131211131211131113121113121211 13

Freeze: 11131211131113121113121113121113 12

Preview: Not Assigned

Strobe: Not Assigned

Menu: 11131113121113121113121113121211 13

Brand Name: SONY Model: SONY1

Frequency: 40.0 kHz

Time Table Time 1 : 2.40 ms

Time 2 : 0.60 ms

Time 3 : 1.20 ms

Time 4 : 25.0 ms

Time 5 : 0.00 ms

Header: 1

Custom Code: 2322222222

Commands:

Power: 232223222322222322222222

Vol Up: 222322222322222322222222

Vol Dn: 232322222322222322222222

Chan Up: 2222222223222223222222222

Chan Dn: 2322222223222223222222222

Mute: 222223222322222322222222

Alt Chan: 2223222323223222322222222

Plus: 232323222322222322222222

Minus: 2322232222232223222222222

Cur Dn: 222322322232223222222222

Key 0: 232222232222222322222222

Key 1: 222222222222222322222222

Key 2: 232222222222222322222222

Key 3: 222322222222222322222222

Key 4: 232322222222222322222222

Key 5: 22222322222222322222222

Key 6: 23222322222222322222222

Key 7: 22232322222222322222222

Key 8: 23232322222222322222222

Key 9: 22222232222222322222222

Cur Up: Not Assigned

Sleep: 2223232223223222322222222

Ch Guide: 23232223222222322222222

PIP: 232223232223222322222222

PIP Pos: Not Assigned

Swap: Not Assigned

Freeze: 2322222322223222322222222

Preview: Not Assigned

Strobe: Not Assigned

Menu: 222222232223222322222222

Brand Name: SONY

Model: SONY2

Frequency: 40.0 kHz

Time Table

Time 1 : 0.50 ms

Time 2 : 1.50 ms

Time 3 : 3.50 ms

Time 4 : 36.0 ms

Time 5 : 0.00 ms

Header: Not Assigned

Custom Code: Not Assigned

Commands:

Power: 121313131212

Vol Up: 131312121212

Vol Dn: 121213121212

Chan Up: 131212121212

Chan Dn: 121312121212

Mute: 131212131212

Alt Chan: 131312131212

Plus: 131213121212

Minus: 131312131212

Cur Dn: 131313131312

Key 0: 131212131312

Key 1: 121212121312

Key 2: 131212121312

Key 3: 121312121312

Key 4: 131312121312

Key 5: 121213121312

Key 6: 131213121312

Key 7: 121313121312

Key 8: 131313121312

Key 9: 121212131312

Cur Up: Not Assigned

Sleep: Not Assigned

Ch Guide: 131213131212

PIP: Not Assigned

PIP Pos: Not Assigned

Swap: 121212131212

Freeze: 131313121212

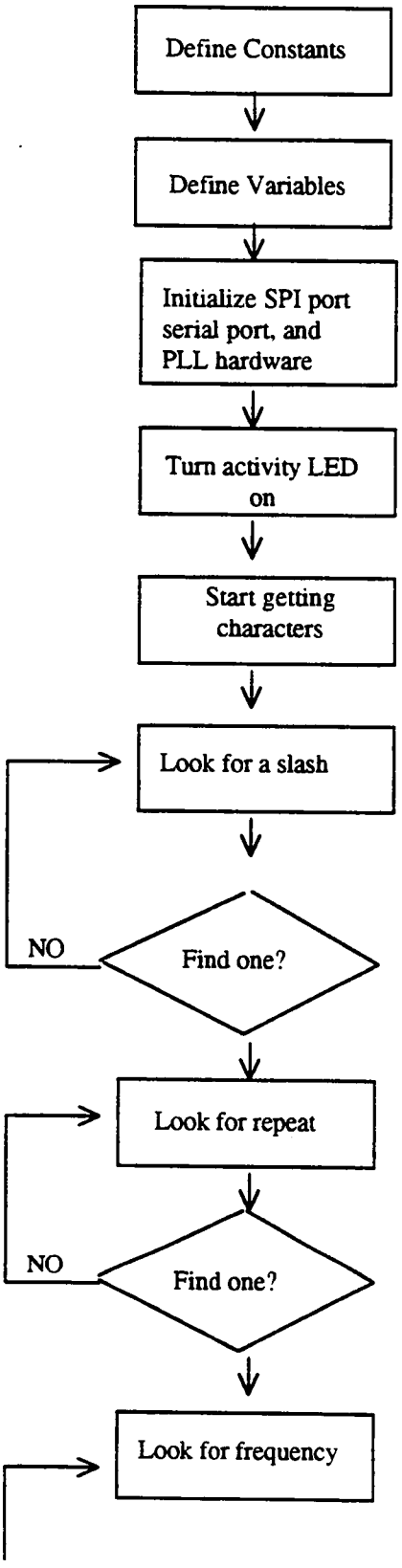
Preview: Not Assigned

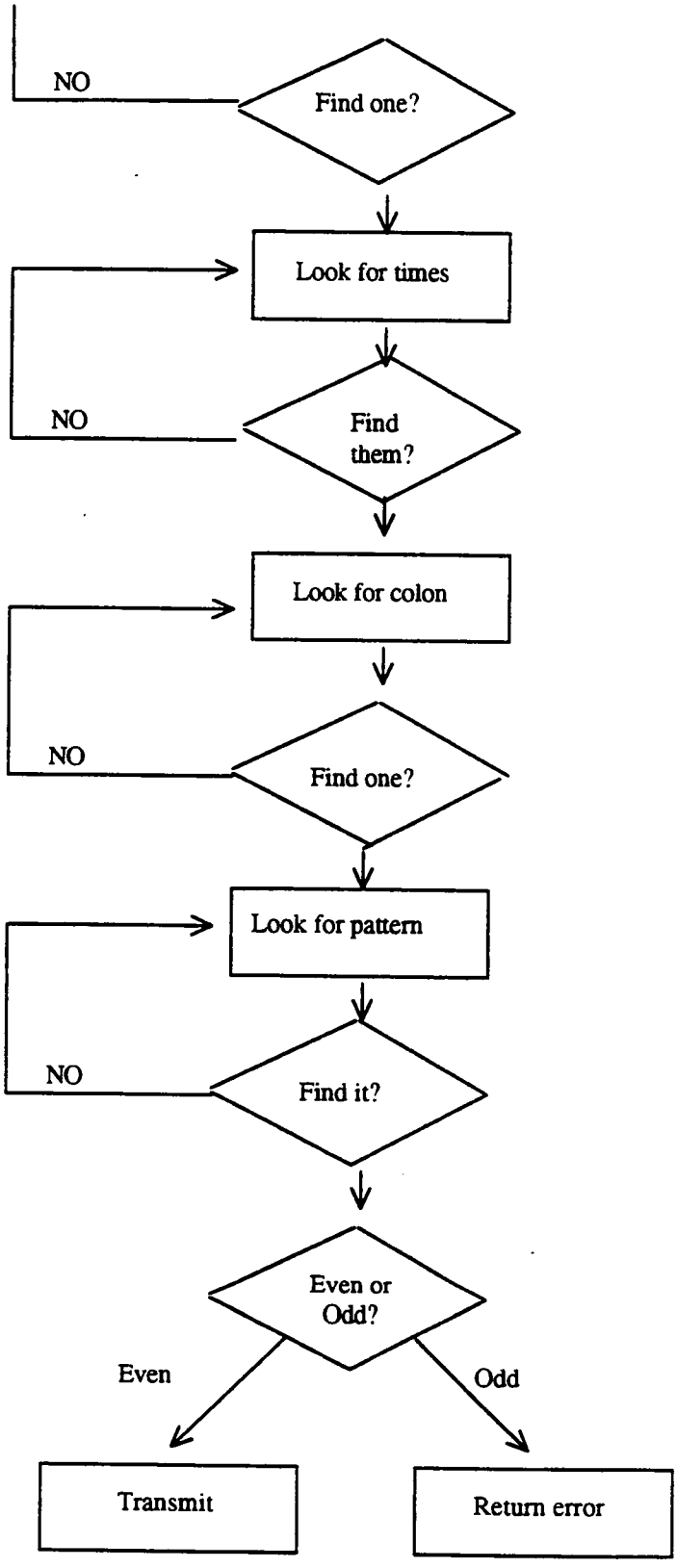
Strobe: Not Assigned

Menu: 121313121212

APPENDIX D

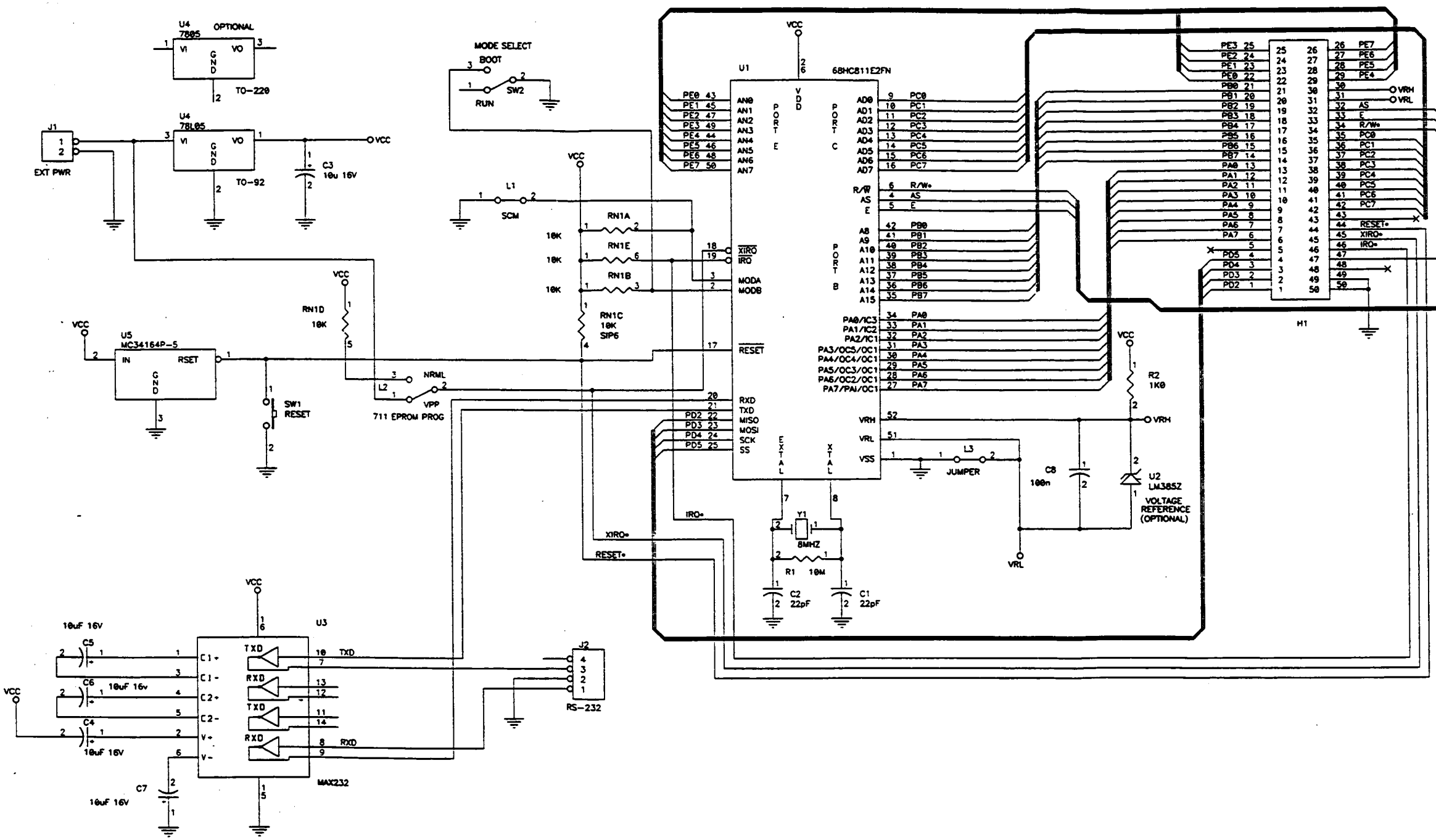
This appendix contains the flow chart of the MC68HC11E2 software to implement the desired task in this thesis.


Define Constants
Define Variables
Initialize SPI port serial port, and PLL hardware
Turn activity LED on
Start getting characters
Look for a slash
NO
Find one?
Look for repeat
NO
Find one?
Look for frequency


NO
Find one?
Look for times
NO
Find them?
Look for colon
NO
Find one?
Look for pattern
NO
Find it?
Even or Odd?
Even
Odd
Transmit
Return error

APPENDIX E

Schematic of the ADAPT-11™ board.


U4
7805
OPTIONAL
TO-220
U4
78L05
TO-92
J1
EXT PWR
C3
10u 16V
VCC
MODE SELECT
BOOT
RUN
SW2
L1
SCM
RN1A
RN1E
RN1B
RN1C
10K
SIP6
RN1D
U5
MC34164P-5
RSET
SW1
RESET
NRML
VPP
711 EPROM PROG
U1
68HC811E2FN
RESET
RXD
TXD
MISO
MOSI
SCK
SS
XIRQ
IRQ
MODA
MODB
VRH
VRL
VSS
L3
JUMPER
R2
1K0
U2
LM385Z
VOLTAGE
REFERENCE
(OPTIONAL)
C8
100n
Y1
8MHZ
R1 10M
C2
22pF
C1
22pF
IRQ*
XIRQ*
RESET*
H1
U3
MAX232
C5
C6
C4
C7
10uF 16V
J2
RS-232

APPENDIX F

This appendix contains letters of permission to reproduce some graphs.

NATIONAL INSTRUMENTS™
The Software is the Instrument®

6504 Bridge Point Parkway • Austin. TX 78730-5039

Sales/Service (512) 794-0100
other calls (512) 338-9119
Fax (512) 794-8411

May 10, 1996

Bassell Atallah
Software Test Engineer
Land Systems Division
General Dynamics
38500 Mound Road
M.Z. 436-20-84
Sterling Heights, MI 48310-3200

RE: Permission to Reproduce National Instruments Artwork

Dear Mr. Atallah,

Thank you for your elecronic mail requesting permission to reproduce illustrations of National Instruments products in your thesis manuscript. In accordance with your request, this letter grants you the limited right to include in your paper the enclosed photographs of the GPIB-SCSI-A and GPIB-232CT-A and the illustration of the GPIB-232CT-A under the condition that National Instruments is given copyright credit in approximately the form that follows:

> Reprinted with permission of the copyright owner, National Instruments Corporation

All other rights are reserved by National Instruments Corporation. I have enclosed photographic prints and a disk containing the illustration of the GPIB-232CT-A for use in your paper. Please send a copy of the final publication of your thesis to me for our files and for reprint consideration for our customer library. If you have any questions, please contact me at (512) 794-5464, fax (512) 794-5569.

Best Regards,

Roxanne Green

Roxanne Green
Publicity Manager
(512) 794-5464 (voice)
(512) 794-5569 (fax)
roxanne.green@natinst.com

Enclosures.

MOTOROLA

June 3, 1996

Bassel Atallah
General Dynamics - Land Systems Division
34514 Baypointe Drive
Sterling Heights, MI 48310

Dear Mr Atallah:

This is in response to your inquiry of 5/9/96 requesting permission to reproduce certain Motorola material. Permission is hereby granted for you to reproduce the following material for which Motorola Semiconductor Products Sector holds the copyright:

Figure 2-1 in the MC68HC11E9 Data Book

We understand that this material will be used in a thesis you are writing.

This permission conveys to you non-exclusive North American and world rights in the English language, to reprint this material in the current and future editions of this work. We also grant permission for reproduction in Braille, large print, and in recorded form for the blind and handicapped. This permission extends to usage in the associated study guide, instructors manual, and transparency masters.

With each usage of this material, please include the following source attribution statement:

"Copyright of Motorola,
Used by Permission"

Thank you for your interest in Motorola products and technology.

Regards,

Karen Bosco

Karen Bosco
Manager, Technical Publications
(602) 244-6584
(602) 244-6560 (FAX)

Motorola, Semiconductor Products Sector
Technical Publications Department, MD A201
5005 East McDowell Road, Phoenix, Arizona 85008

VITA

Bassel Atallah was born on the 8th day of April 1972, in Beirut Lebanon. He went to "College De La Sagesse" a French system school, and graduated in 1989. He enrolled in a dual degree program at the "American Lebanese University" and transferred to "Georgia Institute Of Technology" in 1992. He was awarded a degree in Bachelor of Engineering with honors, and a degree in Bachelor of Arts in 1994. Finally, He enrolled at the University Of Tennessee at Knoxville in 1994, and pursued the Master of Science degree. He was engaged in a research assistantship in the CISP group until he graduated in 1996.