

To the Graduate Council:

I am submitting herewith a thesis written by Robert Henry Fugerer entitled "The Development of a Digital Signal Processor for Focal Plane Array Data Acquisition and Reduction." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Bruce W. Bomar
B. W. Bomar, Major Professor

We have read this thesis
and recommend its acceptance:

Roy D Joseph
Alfonso Pujol, Jr

Accepted for the Council:

C. W. Mink
Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at the University of Tennessee, Knoxville, I agree that the library shall make it available to borrowers under rules of the library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor in his absence, by the head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature

Robert H. Tignor

Date

11/29/88

THE DEVELOPMENT OF
A DIGITAL SIGNAL PROCESSOR
FOR FOCAL PLANE ARRAY
DATA ACQUISITION AND REDUCTION

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

Robert Henry Fugerer
December 1988

ACKNOWLEDGEMENTS

The author would like to thank several people who helped in the preparation of this thesis. Thanks are extended to Dr. Bruce Bomar, Dr. Roy Joseph, and Dr. Alfonso Pujol for their assistance and counsel in preparation for this thesis.

A sincere thanks is extended to Mr. Roger L. Johnson without whose help this work would not have been possible.

The author would like to thank Calspan Inc. and the Air Force for providing the opportunity to attend UTSI.

Finally, the author would like to thank his wife Susan for her patience and understanding during the writing of this thesis.

References to named commercial products in this thesis are not to be considered in any sense as an endorsement of the product by the United States Air Force or the Government.

The research reported herein was performed by the Arnold Engineering Development Center (AEDC), Air Force Systems Command. Work and analysis for this research was done by personnel of Calspan Inc., AEDC Division, operating contractor for the aerospace flight dynamics

testing facilities at the AEDC. Further reproduction
is authorized to satisfy needs of the U. S. Government.

ABSTRACT

The capability to acquire and reduce focal plane array (FPA) data in real time is a critical requirement to FPA testing. The initial data system developed at the AEDC for this application used buffer memory modules located in a Digital Equipment Corporation (DEC) computer system to acquire and store the raw FPA data. This system, though inexpensive, created a "processing bottleneck" since all of the data reduction was performed by a single LSI-11/73.

The initial data system was improved in throughput capability and overall system performance by the development of a parallel signal processing system which distributes the FPA data reduction task over several digital signal processors. The development of a digital signal processor which has the capability to acquire and reduce data from two FPA data channels in real time is presented. These boards maintain the DEC system interface and provide both input data buffering from the FPA and processed data buffering to the DEC computer. The ability to process data in real time is provided by a high speed fixed point integer signal processor. The signal processor maintains high computational accuracy since many of the noise

computations require processing of very small numbers.
A test to verify proper FPA processing was developed
and used to successfully check the signal processor.
In addition, the signal processing software was
benchmarked to verify real time operation.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
Statement of the problem	1
Focal plane array detectors	2
Focal plane array signal description	4
Data acquisition requirements	8
Signal processing requirements	9
Previous processing techniques	13
Thesis description and organization	14
2. PROCESSOR ARCHITECTURAL OVERVIEW	17
Introduction	17
Data buffers	19
Memory	21
Input/output ports	25
Control	26
Digital signal processor	28
3. PROCESSOR DESCRIPTION	30
Introduction	30
Data buffers	30
Memory	31
Data acquisition controller	35
QBUS controller	40
Input/output ports	45
Digital signal processor	51

CHAPTER	PAGE
3. (Continued)	
Power requirements	59
4. SYSTEM TIMING	60
Introduction	60
QBUS timing	60
Digital signal processor timing	62
Microsequencer timing	66
5. SYSTEM SOFTWARE	69
Introduction	69
QBUS interface software	69
FPA application software	70
6. PROCESSOR CHECKOUT	72
Introduction	72
FPA application software benchmarks	72
FPA noise computations	75
7. SUMMARY AND CONCLUSION	80
LIST OF REFERENCES	83
VITA	84

LIST OF FIGURES

FIGURE	PAGE
1.1. Hybrid focal plane array	3
1.2. FPA signal description and system storage . . .	5
1.3. FPA initial system configuration	15
2.1. FPA parallel processing upgrade	18
2.2. QBUS signal processor architectural overview . .	20
2.3. Host/QBSP control communication	27
3.1. 256Kx16 data memory decoding	33
3.2. QBSP address generator	34
3.3. QBSP data acquisition controller	37
3.4. FPA data acquisition algorithm	38
3.5. QBUS interface controller	41
3.6. QBSP base address selection	46
3.7. QBSP control status register contents	49
3.8. TMS320C25 digital signal processor	52
3.9. TMS320C25 memory configuration	53
3.10. TMS320C25 arithmetic logic unit	55
3.11. Q15 and Q12 arithmetic formats	58
4.1. QBUS interface timing	61
4.2. TMS320C25 four phase clock	63
4.3. TMS320C25 read/write timing	64
4.4. TMS320C25 startup sequence	68
6.1. FPA data acquisition code benchmarks	73

FIGURE (Continued)

PAGE

6.2. Noise measurement test configuration	77
6.3. Theoretical .vs. QBSP noise data	79

CHAPTER 1

INTRODUCTION

Statement of the Problem

The majority of systems proposed for the Strategic Defense Initiative (SDI) rely on infrared focal plane arrays (FPAs) to gather information for surveillance, acquisition, tracking, and kill assessment. An effort to advance the state of the art in FPA technology has been initiated at AEDC by establishing a test capability to perform radiometric calibration of these devices at the chip level. In addition, the AEDC is developing the capability to present real time mission simulations to FPA sensors. Radiometric calibration of FPA devices requires that specific noise and intensity data be computed for each detector element in the device. Since FPAs have many detector elements, the noise and intensity data must be computed as rapidly as possible to minimize test time. When performing a mission simulation test, the pixel intensities must be computed in real time.

Focal Plane Array Detectors

Focal plane arrays tested at the AEDC are two dimensional arrays of photodetectors (pixels) that are sensitive to the visible and mid-infrared wavelength radiation (0.4-20 microns) [1].* The photosensing array can be described as a group of electrostatically controlled capacitors which convert photon energy into electrical voltages. Once converted, these devices use charge coupling techniques to transfer the electrical voltages out of the device. Hybrid FPA devices contain a charge coupled device (CCD) multiplexer on one chip, and a photosensing array on another chip. These two chips are then bonded together to form a single unit. Figure 1.1 diagrams a typical hybrid FPA like those tested at the AEDC. The detectors have 1280 pixel elements organized into a rectangular array of 20 columns and 64 rows. There are several input signals required to interface with a hybrid FPA. Each pixel element has a unique column and row address which multiplexes the pixels' output into one of a number of data channels. One data channel is usually used for each column in the detector array. This reduces the

* Numbers in brackets refer to similarly numbered references in the List of References.

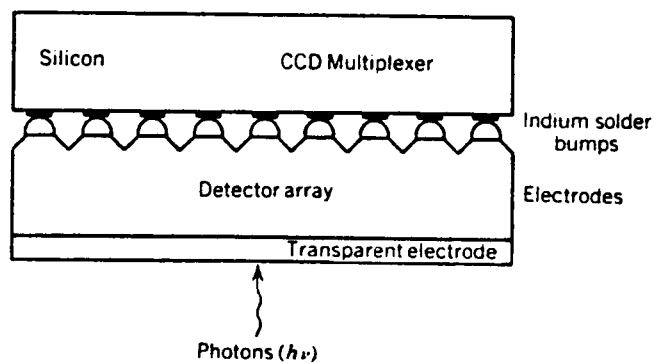
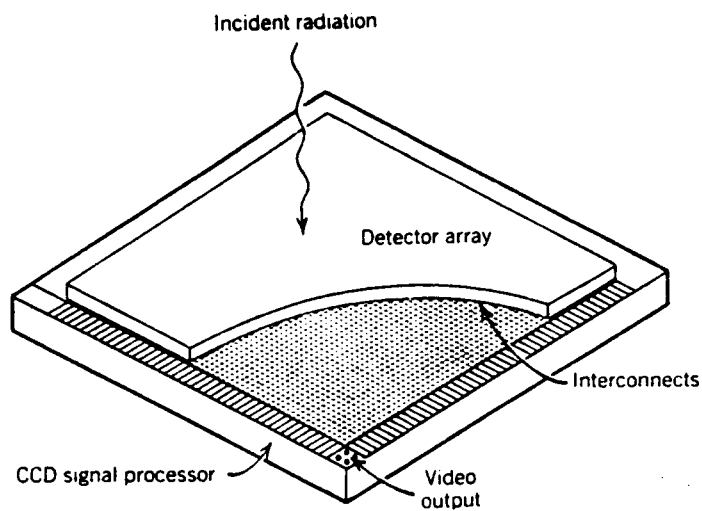


Figure 1.1. Hybrid Focal Plane Array.

number of addresses required by the FPA multiplexer. For a 20 column by 64 row FPA with 20 data channels, the interface electronics need only provide a row address. After applying a row address, the pixel voltages for all columns in the selected row will be present at the data channel outputs. Since photons entering the FPA detector create an increase in voltage at each pixel element, an input signal is needed to periodically reset each pixel. Typically this is accomplished by applying a row address, then applying the proper voltage to the FPA reset signal. This resets the pixels of all columns along the addressed row. FPA operation also requires detector bias voltages and grounding pins.

Focal Plane Array Signal Description

Focal plane array devices tested at the AEDC are organized as a matrix of photodetectors. Each photodetector cell collects photon energy and converts it into an electrical voltage which can be randomly read out of the device. Since there are fewer data channels than photodetectors on a FPA, each photodetector voltage must be clocked out of a data channel in a sequential manner. Figure 1.2 diagrams a typical FPA data channel (column) output. For a fixed

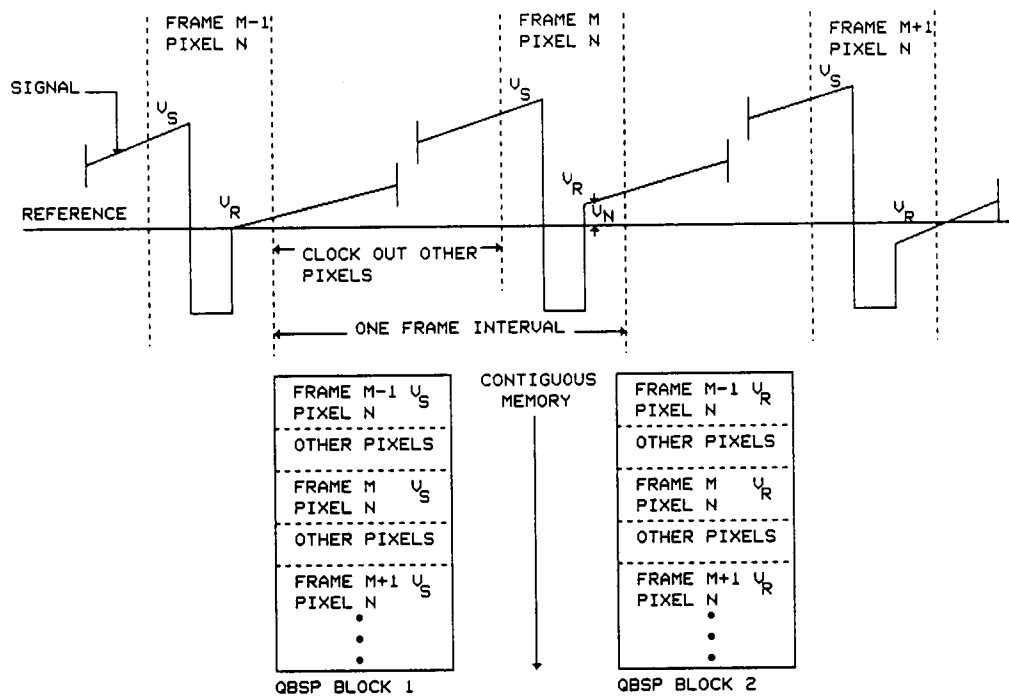


Figure 1.2. FPA Signal Description and System Storage.

row address, it is important to remember that many FPA data channels (columns) appear in parallel. As shown in Figure 1.2, the signal from each photodetector pixel can most easily be described by comparing it to the charging of a capacitor. When current is allowed to flow into a capacitor (analogous to photon energy irradiating an FPA detector), a voltage starts to build up across the capacitor (FPA detector output). It is during this period of time that the capacitor is said to be integrating current, or in the case of FPAs integrating photons. The number of photons which have irradiated an FPA pixel during a specified interval can be determined by resetting the pixel, reading the reset voltage, V_r , allowing the pixel to integrate for the specified time interval, and reading the final voltage, V_s . The number of photons which irradiated the FPA pixel can be calculated by taking the difference between V_s and V_r and dividing by the integration time. The process of resetting the FPA detector and digitizing the V_s and V_r values is accomplished by sequencing electronics which control the FPA and analog to digital (A/D) converters. As shown in Figure 1.2, all of the pixels are sequenced out and read during one integration interval, which is also referred to as a frame. This is accomplished by simultaneously reading

the pixels of a row, resetting all pixels on the row, advancing to the next row, and repeating this process until all rows are read and the frame is complete.

An important aspect of the FPA signal is detector reset noise. As illustrated in Figure 1.2, there is a nonuniformity in the reset voltage (V_r) read from the FPA after a row reset occurs. This level, depicted as V_n , can cause the computation of large noise levels depending on the correlation between the V_s and V_r values. As shown in Figure 1.2, if V_r in frame $M-1$ is subtracted from V_s in frame M , then it is said that V_s is correlated with V_r . Correlation between V_s and V_r eliminates detector reset noise since the level on V_r also appears on V_s , and will be cancelled when the difference between V_s and V_r is taken. If V_r in frame M is subtracted from V_s in frame M , then it is said that V_s is uncorrelated with V_r . Typically, data systems have used uncorrelated processing and assumed low reset noise since uncorrelated processing is much easier to implement using analog circuits. Some of the FPAs tested at the AEDC have appreciable levels of reset noise and require correlated processing. Other FPAs are designed with the intent of minimizing reset noise so that uncorrelated processing can be used. This requires

that the data system testing these devices have the capability to perform both correlated and uncorrelated processing.

Data Acquisition Requirements

The increasing size and speed of FPA devices place stringent requirements on the data systems' memory capacity and acquisition rates. The sampling of V_s and V_r implies that two data values be stored for each pixel in each frame. This requires 2560 storage elements per frame for an FPA with 20 columns and 64 rows. The capability to store 1000 frames assures a good time history of the FPA image in addition to providing enough samples for statistical computations. This gives a requirement for 2.56 million values to be stored. The size of each stored value should be 16 bits to ensure enough resolution in the A/D conversion electronics. This defines the total memory required to store 1000 frames of FPA data as 5.12 megabytes.

Typically, the integration time used when testing FPAs is 1ms. However, devices with integration time capabilities below 100us are not uncommon. Therefore, the data system to test these devices should have the capability to accept FPA data from a 20 column by 64 row device being operated at a 64us integration time.

This requires a 640 megabits per second acquisition rate by the data system.

Signal Processing Requirements

Both real time mission simulation testing and radiometric calibration place stringent requirements on the data system which will perform the FPA signal processing. A critical part of real time mission simulation testing is the ability of the data system to compute the intensity of each FPA pixel element in real time. Once the pixel intensities are computed, they are passed to imaging hardware which uses a color monitor to display the data. A pixel intensity is computed by simply averaging the value of the pixel over N frames giving

$$I_p = 1/N \sum_{i=1}^N [V_s(i) - V_r(i)] \quad (1.1)$$

When averaging several frames of data, the correlation (or lack thereof) between V_s and V_r becomes unimportant. This result is easily seen by writing Equation 1.1 in the form

$$I_p = \overline{V_s} - \overline{V_r} \quad (1.2)$$

where

$$\overline{Vs} = 1/N \sum_{i=1}^N Vs(i) \quad (1.3)$$

$$\overline{Vr} = 1/N \sum_{i=1}^N Vr(i) \quad (1.4)$$

Since the noise associated with Vr is can be considered as random and symetric about the mean, there is little difference between correlated and uncorrelated pixel intensities when averaged over many frames.

The number of operations required to compute the pixel intensities for an FPA can be found by considering the computational functions addition, subtraction, multiplication, and memory moves as single operations. Each pixel requires two moves to get Vs and Vr , a subtraction, and an addition for each frame. Two additional operations are required to perform a division and store the result for each pixel. The result is

$$MOP = (4N + 2)(NROW)(NCOL) / 10^6 \quad (1.5)$$

where MOP, N, NROW, and NCOL represent million of operations, number of frames, number of FPA rows, and number of FPA columns respectively. A 20 column by 64 row FPA requires .658 million operations to compute I_p for 128 frames. A typical integration time for real

time mission simulation would be 1ms. This implies 128ms to acquire 128 frames of data. These values indicate the need for a data system with greater than 5.2 million operations per second (MOPS) capability to have a continuous real time processing of pixel intensities.

A critical part of radiometric calibration is the ability to accurately compute the noise associated with each pixel on an FPA. When computing noise, it is important that the data system compute and compare both correlated and uncorrelated noise values. The root mean squared (RMS) pixel noise is computed using

$$\text{RMSp}^2 = \frac{1}{N} \sum_{i=1}^N \{ [V_s(i) - \overline{V_s}] - [V_r(i) - \overline{V_r}] \}^2 \quad (1.6)$$

The RMS noise on a single pixel element is represented by RMSp. The values of V_s and V_r are assumed to be correlated or uncorrelated depending on the type of processing desired. By using algebraic manipulations, Equation 1.6 can be written in the form

$$\begin{aligned} \text{RMSp}^2 &= \text{RMSVs}^2 + \text{RMSVr}^2 - \\ &2/N \sum_{i=1}^N [V_s(i) - \overline{V_s}][V_r(i) - \overline{V_r}] \end{aligned} \quad (1.7)$$

where

$$\text{RMSVs}^2 = 1/N \sum_{i=1}^N [\text{Vs}(i) - \overline{\text{Vs}}]^2 \quad (1.8)$$

$$\text{RMSVr}^2 = 1/N \sum_{i=1}^N [\text{Vr}(i) - \overline{\text{Vr}}]^2 \quad (1.9)$$

Equation 1.7 computes the RMSp value in a manner which enables comparison of three important noise components which could not be monitored by using Equation 1.6. The RMSVs and RMSVr values represent the RMS noise on the Vs and Vr voltage levels, respectively. The final term in Equation 1.7 is recognized as the cross-correlation for a given lag Tau between Vs and Vr. It should be noted that Equation 1.7 would be used for uncorrelated processing. Correlated processing can be accomplished by replacing Vr(i) in the cross-correlation term with Vr(i-1). By comparing these three noise components using correlated and uncorrelated signal processing, a much better appreciation can be obtained for the origin of the various noise sources in the system.

The MOP value necessary to compute the noise on each pixel of an FPA device using Equations 1.7 thru 1.9 is

$$\text{MOP} = (24N + 12)(\text{NROW})(\text{NCOL}) / 10^6 \quad (1.10)$$

This Equation is derived by taking Equation 1.5 and using it to compute the number of operations required in Equations 1.8 and 1.9. The result was then combined into Equation 1.7 to get the result in Equation 1.10. Using Equation 1.10, a 20 column by 64 row FPA requires 3.95 million operations to compute the noise on each FPA pixel over 128 frames. Although the computation of FPA noise does not require real time speeds, the processing should be done rapidly enough to ensure reasonable chamber occupancy times during testing. If the data system meets the 5.2 MOPS required for real time mission simulation, then the noise computations for all pixels in 128 frames will be completed in .76 seconds which is sufficient to minimize chamber occupancy.

Previous Processing Techniques

Some of the systems developed for FPA testing used high performance array processors to compute the pixel intensities and noise. The cost of this approach limits its use to only a few FPA channels per test run and thus makes it undesirable for use in mission simulation testing. The initial FPA data system developed at the AEDC focused on reduced cost and simultaneous acquisition of up to 20 FPA data channels.

As shown in Figure 1.3, the system acquired data via buffer memory cards which reside in the DEC host computer system. After the data had been acquired, the host QBUS processor reduced the data from each buffer and passed it on to the various display and storage routines. This approach met all of the initial testing requirements including the capability to perform correlated and uncorrelated noise processing. However, real time mission simulation requirements can not be accomplished with this system because of the load placed on the single DEC processor. Performance data demonstrates that a DEC 11/73 processor is only capable of .7 MOPS.

Thesis Description and Organization

This thesis will describe the design, development, and checkout of a digital signal processor which meets the current AEDC testing requirements for infrared FPA detectors using the state of the art in programmable logic devices, high density static memories, microprogrammed sequencers, and high speed integer processors.

The description of the signal processor is divided into several chapters. An overview of the basic architecture will be presented in chapter 2.

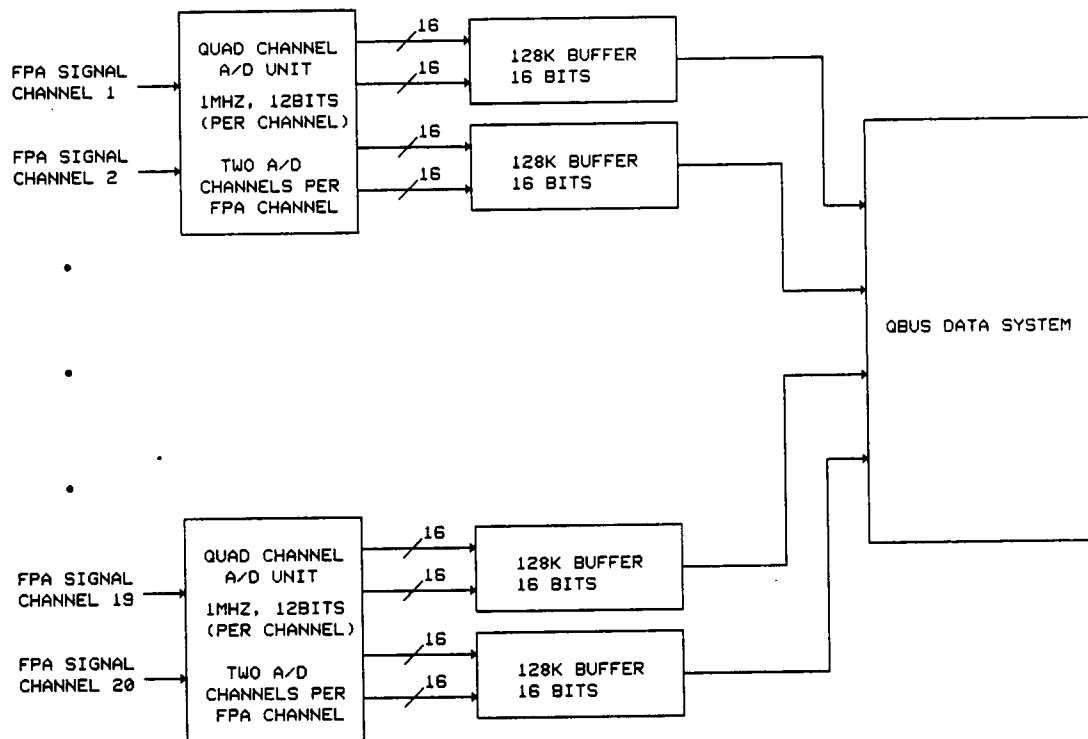


Figure 1.3. FPA Initial System Configuration.

A description of the system hardware, timing, and software is contained in chapters 3, 4, and 5, respectively. Chapter 6 describes the system checkout which includes speed performance data to verify real time system throughput, the development of an Equation to predict noise data from a known input, and implementation of an experiment which uses this Equation to verify correct noise computations. Chapter 7 gives a summary of the work done and draws certain conclusions.

CHAPTER 2

PROCESSOR ARCHITECTURAL OVERVIEW

Introduction

Since the FPA data channels operate in parallel, it was decided that the data acquisition and signal processing requirements outlined in chapter 1 would best be accomplished using a distributed processor approach. This would require the development of a single processor board which could acquire and process signals from only a few FPA data channels. By allowing several processor boards to operate in parallel, simultaneous acquisition and signal processing on all FPA data channels would be accomplished. A cost versus performance analysis showed that a signal processing board could be built which meet the requirements for two FPA data channels at a cost equal to 10% of previous array processor systems. This reduced cost enabled the build up of 10 boards providing a 20 channel capability which meets the AEDC testing requirements. As shown in Figure 2.1, the new approach retains the DEC QBUS interface [2] of the initial AEDC system, and distributes the processing over 10 boards operating in parallel.

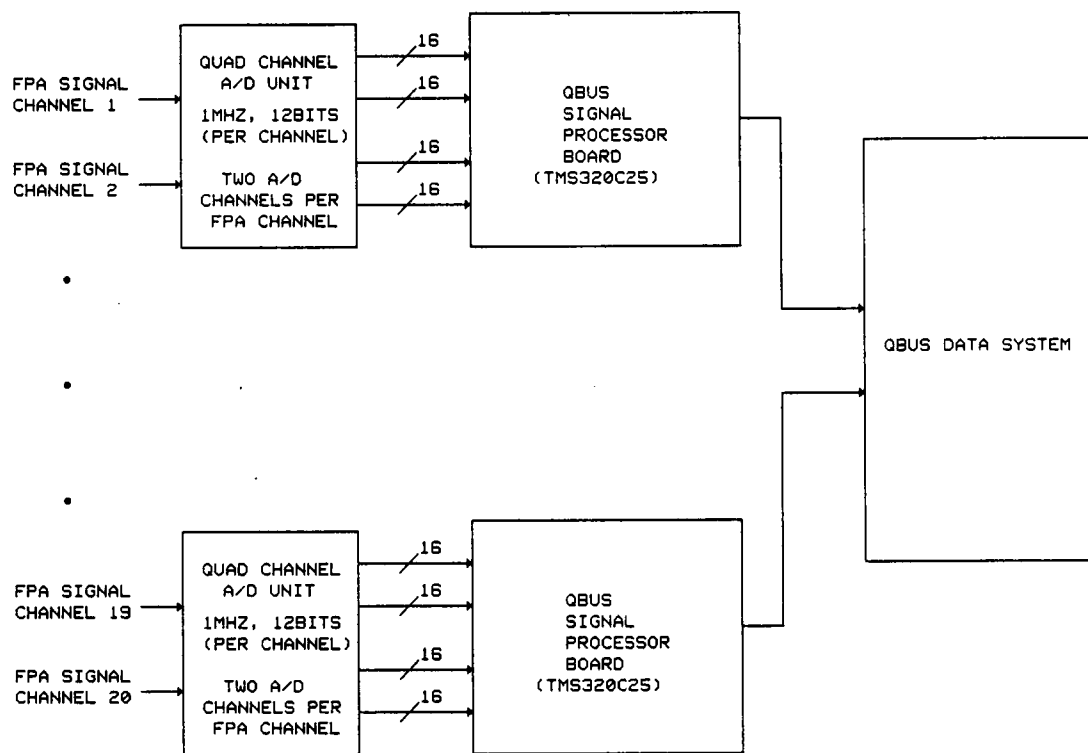


Figure 2.1. FPA Parallel Processing Upgrade.

This chapter describes the architecture of the QBUS signal processor (QBSP) which provides a low cost solution to maintaining the initial AEDC systems' memory storage capability, maximizing data throughput, and providing real time processing while maintaining computational fidelity. Figure 2.2 provides a block diagram of the QBSP which highlights the various system components and data paths.

Data Buffers

Real time signal processing implies that the QBSP will permit a continuous flow of data from the FPA sensor, and will process all of the data received. The QBSP architecture accomplishes this flow of data by allowing the acquisition of FPA data, the processing of FPA data, and the transfer of reduced FPA data into the host computer to occur simultaneously. As shown in Figure 2.2, the QBSP must interface between three data buses which include the FPA data bus, the signal processor data bus, and the host computer bus (QBUS). First-in-first-out (FIFO) memory devices are used to provide isolation between the three data buses. In addition to providing isolation, these devices have completely autonomous ports for input and output data which can be operated simultaneously. Figure 2.2

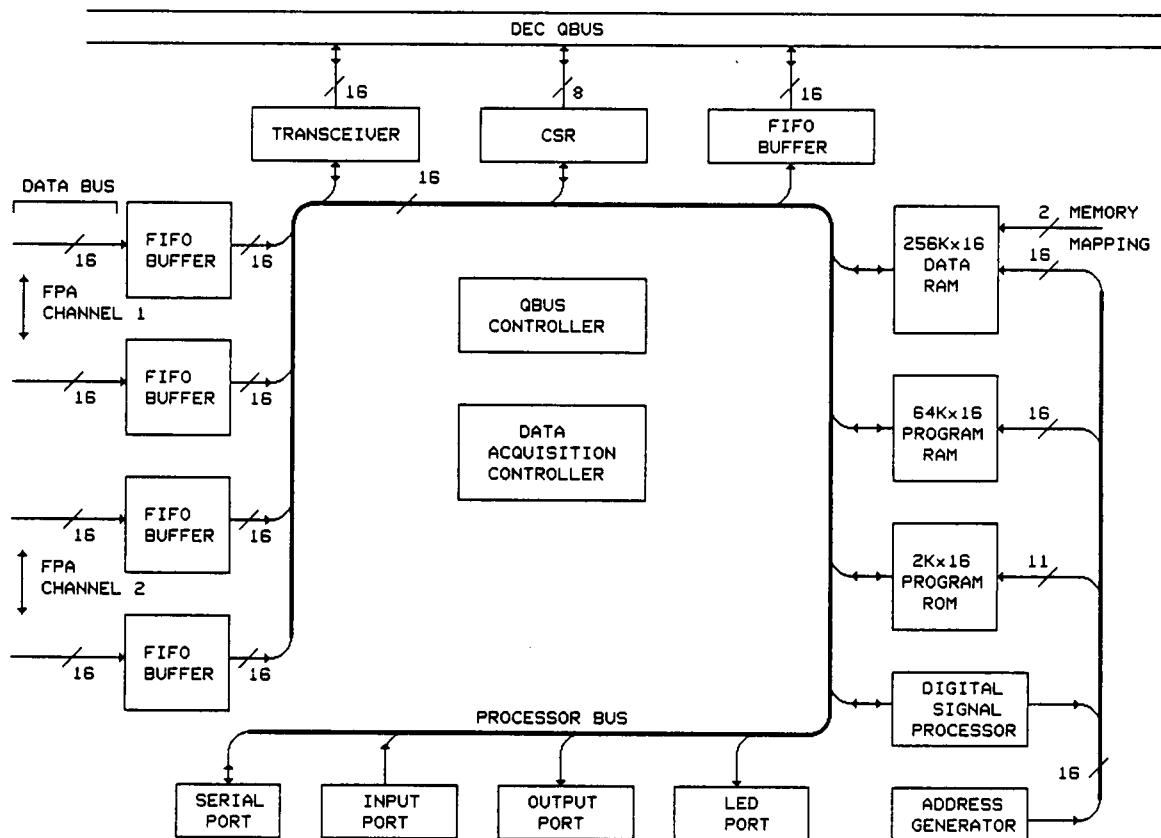


Figure 2.2. QBUS Signal Processor Architectural Overview.

illustrates how FIFO buffering separates the processor data bus from the FPA data bus and host computer bus. This type of architecture enables FPA data to be clocked into the FIFO ports, and reduced data to be clocked onto the host computer bus without disrupting activity on the signal processor bus. This approach is also useful in permitting high speed burst data acquisition since current technology has produced FIFO buffers with very high clocking rates and large memories.

Memory

Many of the high speed digital signal processors use a modified Harvard architecture [3]. The Harvard architecture separates program and data memory into completely different spaces so that program and data fetches can be overlapped. The modified Harvard architecture allows transfers between program and data memory, but maintains the parallelism obtained by overlapping program and data fetches. Figure 2.2 outlines the memory configuration of the QBSP demonstrating the separation between data and program memory.

Data is stored in 256K of random access memory. The 256K of data memory is divided into four blocks of

64K each. This was done to establish compatibility with much of the software developed for the initial data system. Two 64K blocks are used to store data from a single FPA channel. One stores values of V_s , and the other stores values of V_r . Figure 1.2 illustrates how the FPA data is stored in each 64K block of memory. There is a separate input channel and FIFO buffer for each 64K block of data RAM. All of the data memory, however, is connected to the processor bus. This requires that the data be multiplexed out of the FPA FIFO buffers and into the data RAM. Incorporated into the QBSP multiplexing electronics is the ability to halt the digital signal processor to permit a transfer from FIFO memory to data memory.

Data memory can be both loaded and unloaded by the host computer. This permits the transfer of raw data to the host computer while also permitting the host computer to load data into the QBSP for processing. This function is also particularly useful in system checkout. For example, system check can be accomplished by loading the QBSP with actual FPA data obtained from the initial buffer system, allowing the QBSP to reduce the data, and then comparing the results with results obtained by previous processing software.

A cost versus performance tradeoff can be made by allowing the data memory to operate with wait states. This can greatly reduce the cost of the board, without preventing it from meeting the requirements of chapter 1.

Programs reside in 2K of read only memory (ROM) or 64K of RAM. One of these two memories is mapped into the signal processors' memory space while it is running. There are three reasons why this approach was taken. First, there was a desire to permit the signal processing software to be downloaded from the host computer. This enables a large library of signal processing routines to be stored on the host computer disk, and quickly input to the QBSP when needed. Secondly, ROM programs enable the QBSP to operate in a stand alone configuration allowing the signal processor to start on power up. Thirdly, the ROM and RAM can be mapped so that when code is executed from ROM, the RAM program space becomes data memory space. Since the program RAM is four times smaller than data RAM, the cost for zero wait state operation of this memory is more reasonable. The availability of zero wait state data memory permits high speed control applications using the I/O ports since data can be more rapidly moved out of data RAM and into the control port.

As shown on Figure 2.2, a separate address generator is required when moving data from the QBUS or FIFO buffer ports into program or data RAM. The address bus is 16 bits wide to directly access each element in the 64K blocks. Since there are four data memory blocks, two memory mapping control signals are required to select a single block. These memory mapping signals can be controlled by three different devices on the QBSP. First, the QBUS controller must have access during data transfers between the QBSP and host computer. Secondly, the data acquisition controller must have access during data transfers from the FIFO buffers into data RAM. Thirdly, the signal processor must have access when it is executing code so that it can map the proper 64K block of data into its own data space.

When the signal processor is running, the address generator outputs are tristated. It is also important that the signal processor have the ability to tristate its address and data bus during transfers from the QBUS or FIFO buffers.

The address generator is comprised of four 16 bit counters. Each counter is selected by the memory mapping control signals. Only the QBUS controller and signal processor require access to program RAM. The

block one address is used by the address generator during host computer access of program RAM.

Input/output Ports

Various input/output (I/O) ports are required on the QBSP for control and diagnostic purposes. The control status register (CSR) enables host computer control of signal processor functions and data acquisition. It also allows the host computer to monitor the status of some important QBSP controller and signal processor outputs.

There are two parallel ports located on the processor bus that are used by the signal processor. One port contains 16 inputs and the other 16 outputs. These ports interface with programmable gain and offset electronics located in the FPA analog to digital conversion circuitry. An interrupt signal to the processor is located on the input port.

An additional output port, which is connected in parallel to eight light emitting diodes (LEDs), is also located on the processor bus.

The LEDs are used by the signal processors' application software. They have many functions including diagnostics and visual processor status.

The FIFO buffer between the host computer and processor bus is configured as an I/O port to the signal processor. A final I/O port is provided to enable memory mapping control of the data RAM by the signal processor. As discussed in the memory section, these lines can also be driven by the QBUS and data acquisition controllers.

Control

There are two controller devices used in the QBSP architecture. The first is the QBUS controller which is used to interface program RAM, data RAM, the host FIFO buffer, and control status register to the host computer. The second is the data acquisition controller which is used to multiplex the four FIFO ports into data RAM storage. It also functions as a host commands processor. The host computer can send up to 8 different command codes to the data acquisition controller. These commands instruct the controller to have the QBSP do various functions such as program execution, data acquisition, and host computer data transfers.

Figure 2.3 illustrates the master/slave relationship between the QBUS controller, data acquisition controller, signal processor, and host computer. The host computer is the master with the data acquisition

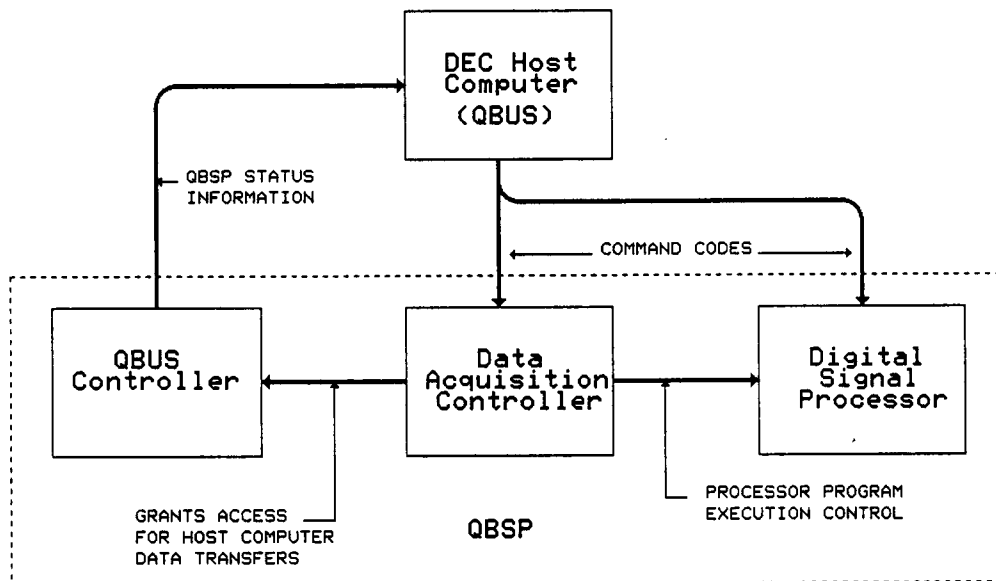


Figure 2.3. Host/QBSP Control Communication.

controller acting as slave. The QBUS controller only operates when given access by the data acquisition controller. However, some of the functions performed by the QBUS controller are completed independently of the data acquisition controller such as reading the host computer FIFO port and control status register. The signal processor only executes code when started by the data acquisition controller. Once the signal processor has started, it may receive commands directly from the host computer. A total of 32 command codes can be directly passed to the signal processor from the host computer.

Digital Signal Processor

There are several requirements involving multi-processing, computational speed, and accuracy which influence the type of digital signal processor (DSP) needed. Although there is only one signal processor on each QBSP board, the processor needs to have the same features that are incorporated into a multiprocessor design. The primary multiprocessor need is the capability to halt the processors' execution momentarily, and then allow it to continue from where it was halted. While the processor is halted, it should tristate its address and data bus to enable access into

program and data memory by other devices. Many digital signal processors currently available have the capability to execute instructions at 10 million instructions per second (MIPS). In addition, these types of processors have instruction sets that are optimized for signal processing tasks. This optimization enables several operations such as addition, multiplication, and data memory moves to be executed in a single instruction so that very high computational speeds are achieved. This type of signal processor also maintains very high computational fidelity by enabling 32 bit arithmetic operations. This number of bits is more than adequate to accomplish the FPA noise calculations. Finally, the signal processor must be low power since several QBSP boards will reside in a single QBUS backplane.

CHAPTER 3

PROCESSOR DESCRIPTION

Introduction

It is important to understand the particular components used to implement the QBSP design. This chapter describes the QBSP components and provides reasons why various components were chosen.

Data Buffers

Data buffering is accomplished using FIFO buffers. The FIFO memory modules used are IDT7M20x dual ported devices that range in depth from 512 to 16K bytes with access times from 35ns to 80ns [4]. Each of the different parts maintain pin compatibility since there are no external address lines required. By using integrated circuit sockets with the FIFOs, FIFO speed and depth can be changed as necessary to adapt to different applications which require fast data rates or larger storage requirements. Each of the FIFO buffers have separate data lines for input and output as well as separate read and write control lines to allow continuing data acquisition while the data acquisition controller is moving data from the FIFOs into data RAM

storage. The QBSP contains jumpers which provide 50 ohm terminations on each input of the FPA data bus. This is needed when receiving burst data at very high rates.

Memory

Programs for the DSP can be stored in program ROM or RAM. The ROM devices used are CY7C291 2Kx8 reprogrammable ROMs [5]. In the FPA application these devices are used to store onboard diagnostics. In applications where the QBSP might operate as a stand alone system, the need for a host computer to download program RAM can be eliminated by loading the ROMs with the processing software.

The device used for program RAM is an IDT7M624 (64Kx16) static RAM module [6]. This device is used to store the FPA application code for the DSP. The data acquisition controller automatically maps this memory to the DSP data space when executing code from the program ROM. This enables the DSP to execute with zero wait states while performing initialization diagnostics. With different program ROM contents, zero wait state operation could also be used to speed up the processing of data loaded into the QBSP by the host computer or to speed up data moves to the output ports.

The data memory consist of 16 HM62256 32Kx8 static RAMs [7]. Each integrated circuit (IC) is in a 68 pin dual in line package (DIP). Since the IC is 32Kx8, there are only 15 physical address lines. Figure 3.1 shows how the memory is decoded on the QBSP to provide a complete 256Kx16 module. Using the 32Kx8 DIP approach allowed more board space for other parts in the design. To keep the QBSP cost within budget, the access time of the data RAM was increased. This requires that one wait state be used whenever the DSP accesses data memory at clock frequencies above 27MHz.

The QBSP address generator is used to provide the correct address location during QBUS access into data or program RAM, and during data acquisition by the data acquisition controller. During DSP operation, the address generator is tristated to allow access to memory by the DSPs address lines. Figure 3.2 shows that the address generator consists of four 16 bit counters controlled by the data acquisition and QBUS controller. Each counter is implemented using an EP600 programmable logic device (PLD) [4]. By using four separate counters, asynchronous data acquisition can be accomplished. Each counter is mapped by its own internal logic to one of the four 64K banks of data memory. When the QBUS accesses program RAM, the address

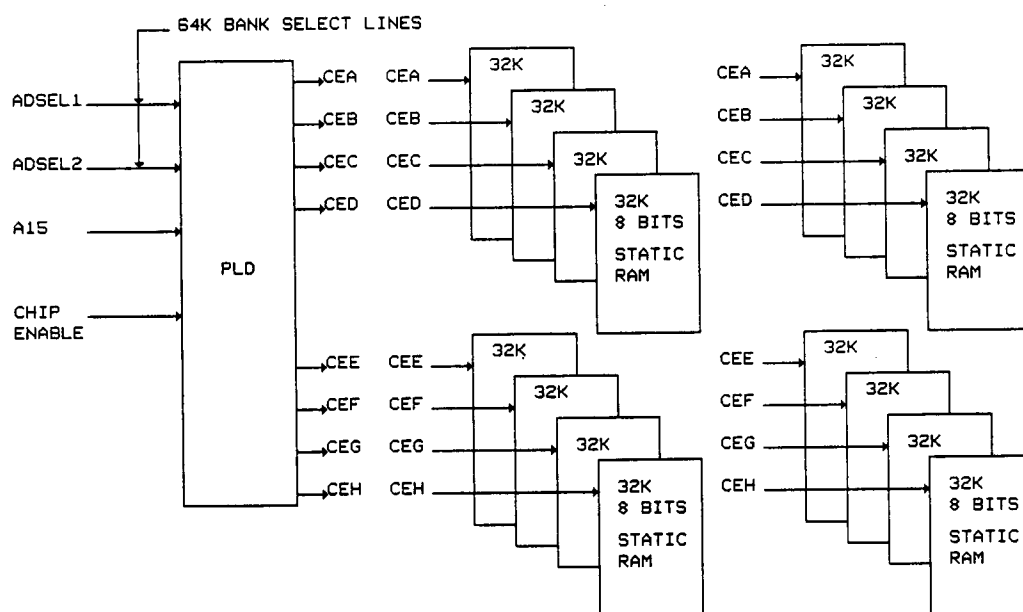


Figure 3.1. 256Kx16 Data Memory Decoding.

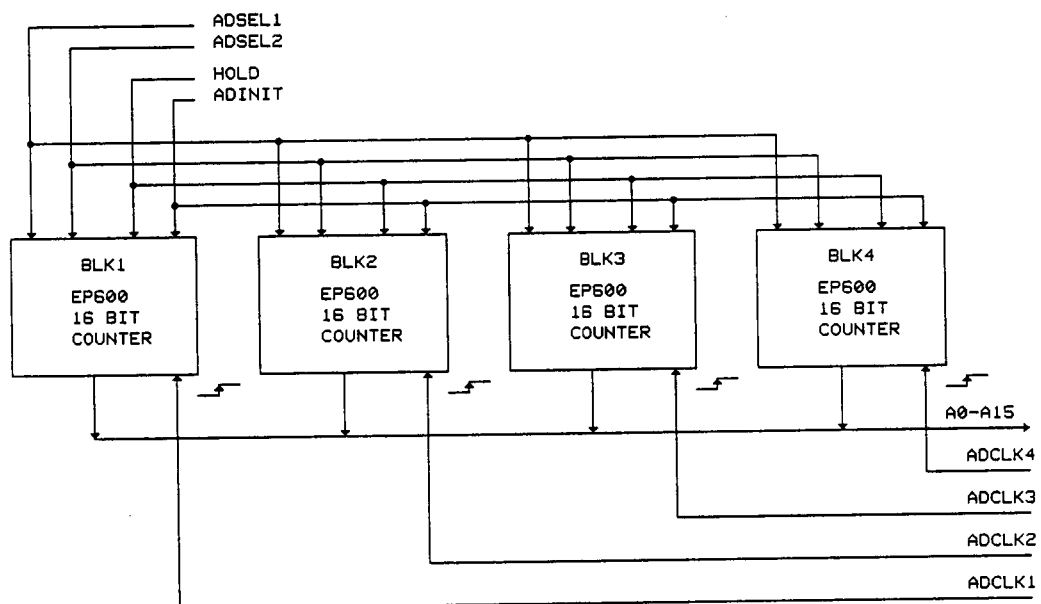


Figure 3.2. QBSP Address Generator.

range of bank one is always selected. Since the digital signal processor chosen for the QBSP reserves the lower 1K of data memory for such functions as interrupt vectors, the counters will always reset to 1024. The bank one counter is unique since it must also provide addresses for program RAM which starts at location zero. This is done by using the HOLD control line to instruct the counter to reset to 0 or 1024. In addition to controlling the bank zero reset, HOLD is used to tristate all of the counters during DSP program execution. The control line ADINIT is used to instruct each counter to reset on the next active clock edge. Since each counter has its own separate clock input, each counter can be independently reset during asynchronous operation. The ADSEL1 and ADSEL2 lines are used to insure that only one counter is active on the bus at a time and that these counters are normally controlled by the data acquisition controller unless it has granted access to the signal processor or QBUS controller.

Data Acquisition Controller

The primary control unit for the QBSP is the data acquisition controller which is implemented using the EPS448 reprogrammable microsequencer [4]. As diagramed

in Figure 3.3, the EPS448 acquires data from the four input data ports, controls interaction between the QBSP and DEC host computer, and enables DSP program execution from program memory RAM or ROM. Four of the eight inputs to the microsequencer are empty flags from the FPA FIFO input data buffers. The microsequencer uses these inputs to determine if data is available and then writes it into its proper data memory location. Since the microsequencer is reprogrammable, the algorithm used to acquire data from the four input ports can be optimized for specific applications. Figure 3.4 outlines the algorithm used to acquire FPA data.

Control of the DSP functions is accomplished by the host computer writing to the QBSP control port. A valid QBUS write latches the three least significant bits into the microsequencer inputs providing up to 8 unique command codes. Table 3.1 outlines how the eight codes are defined by the EPS448 microcode. The DLATCH signal of Figure 3.3 is low when the control code is valid. To minimize the number of microsequencer outputs, the address clocks and FIFO read/write controls are generated using demultiplexing logic that is implemented using PLDs. The PLDs also served to

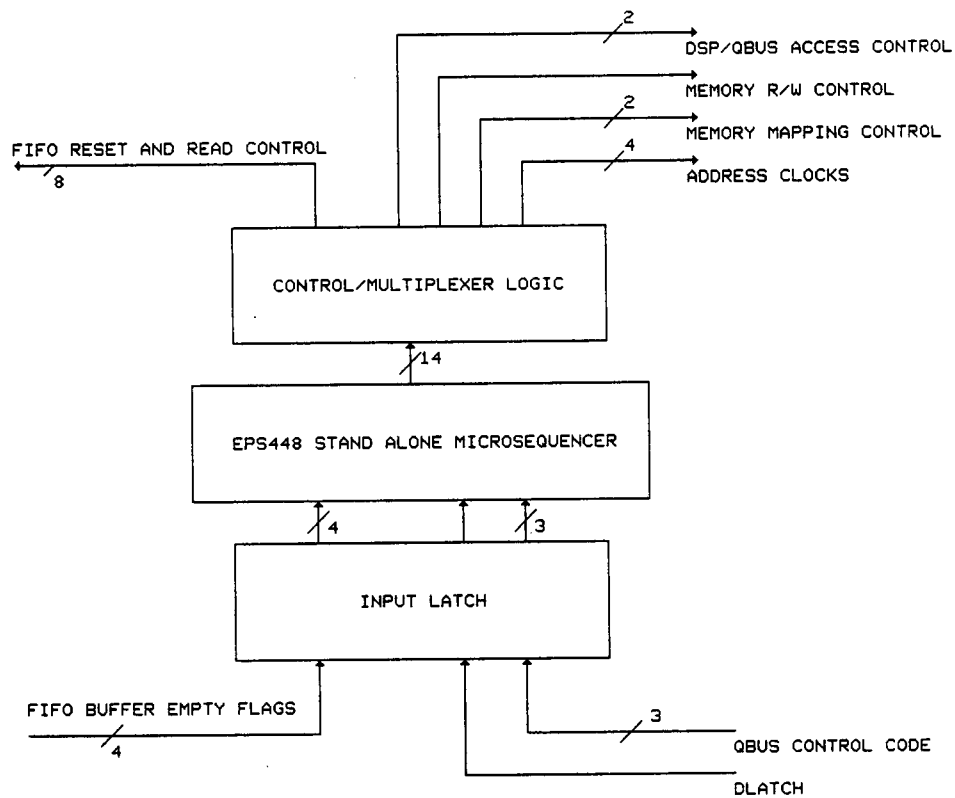


Figure 3.3. QBSP Data Acquisition Controller.

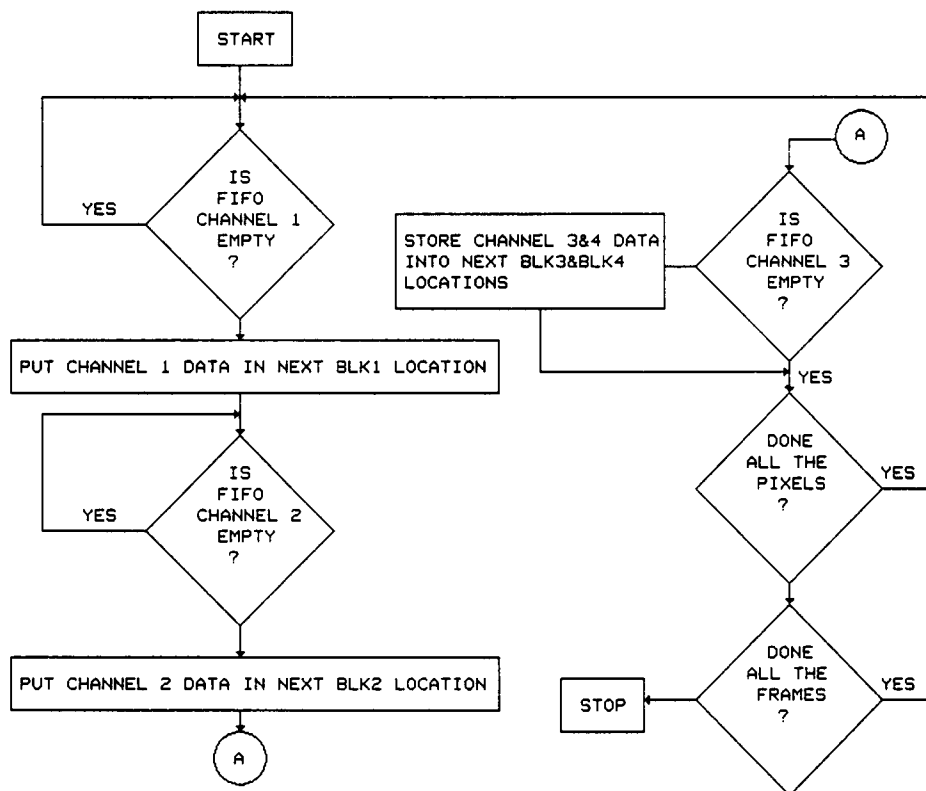


Figure 3.4. FPA Data Acquisition Algorithm.

Table 3.1. QBSP Microsequencer Command Codes.

Command code	Description
0	Microsequencer places the TMS320C25 off line and grants program and data memory access to the QBUS controller.
1	Microsequencer resets the FIFO data buffers then: if code 2 is sent, resets the counters to 1024, or if code 3 is sent, resets the BLK1 counter to 0.
2	Acquire 1 frame of FPA data.
3	Acquire 50 frames of FPA data.
4	Acquire 250 frames of FPA data.
5	Start TMS320C25 from program RAM.
6	Start TMS320C25 from program ROM.
7	Halt the TMS320C25 processor and grant QBUS access to program and data memory.

tristate the microsequencer control signals when access is given to the QBUS controller.

QBUS Controller

The host computer interfaces with the QBSP memory, CSR, and data buffer thru the QBUS controller which is diagramed in Figure 3.5. The QBUS controller has access to both the data buffer and CSR continually. Access to program memory and data memory must be granted by the data acquisition controller since it controls all data acquisition functions and DSP program execution. The QBUS controller is implemented using an EP900 PLD [4] minimizing space requirements and reducing interface problems such as propagation delay through multiple components.

The QBUS controller organizes a QBUS access to the QBSP into three phases. Phase one is a valid cycle determination which distinguishes a QBUS I/O access from a DMA or normal memory access, and then checks to see if the QBUS address is a valid QBSP location. When the necessary conditions have been met the VALCYL signal is asserted low and QCSYNC is asserted high to clock the three next to least significant address bits, BDAL1 thru BDAL3, into the QBUS controller. The VALCYL signal stays active until the QBUS transfer is

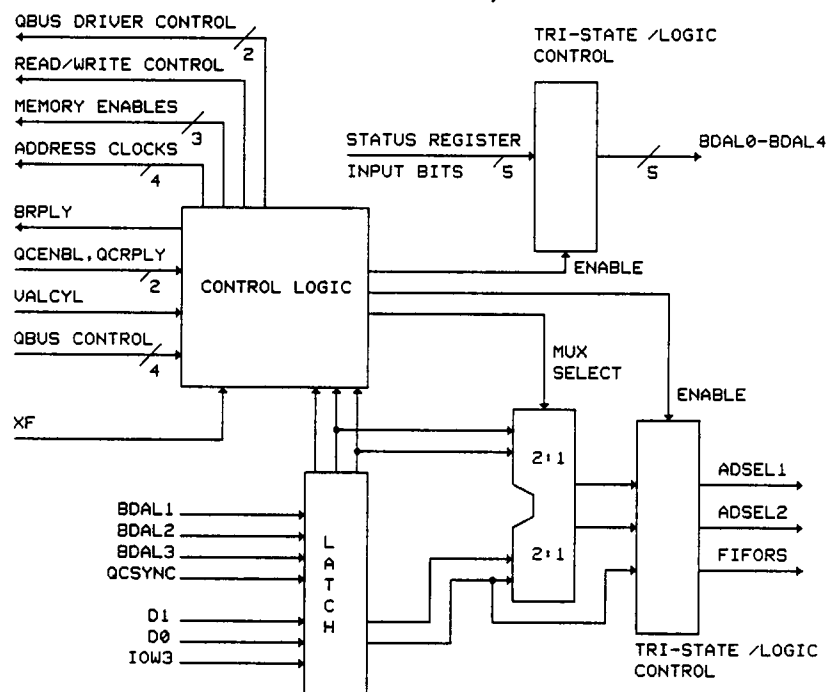


Figure 3.5. QBUS Interface Controller.

complete. The three address bits are used by the controller to determine which particular device on the QBSP is to be accessed by the host computer. Table 3.2 outlines which device is mapped to each of the eight possible locations. Notice that on the QBUS the least significant address bit (BDAL0) distinguishes memory bytes during byte transfers. Since the QBSP performs 16 bit word transfers, BDAL0 is not used and the QBSP address locations change in increments of two bytes.

During phase two the QBUS controller executes a write or read operation on the addressed device as instructed by the host computer control signals. The controller can only read or write to data and program memory when given access by the data acquisition controller. This requires that a certain amount of setup be done by the host computer before accessing data or program memory. The host computer must first request that the data acquisition controller give access to the QBUS controller. This is done by writing the proper control code to the CSR as documented in Table 3.1. The data acquisition controller will respond by setting QCENBL high and then grant access to the address clocks, R/W and memory enable lines from the QBUS controller.

Table 3.2. QBSP Address Definitions.

Address	Description
Base	Read/write to data memory bank 1.
Base + 2	Read/write to data memory bank 2.
Base + 4	Read/write to data memory bank 3.
Base + 6	Read/write to data memory bank 4.
Base + 8	Read/write to program memory.
Base + 10	Read the DSP/QBUS FIFO buffer.
Base + 12	Read the status bits from the CSR. Write control commands to the microsequencer and DSP.
Base + 14	Not used.

The third phase of operation is responding to the host computer at the completion of the read or write cycle. This is done with the QBUS controller BRPLY line. This line must not be driven except when the QBSP is completing a valid cycle. Therefore, the BRPLY line is held in the tristate condition until a valid cycle condition occurs. The internal logic of the QBUS controller asserts BRPLY low when a valid QBSP read or write is completed.

The QBUS controller also functions as an I/O port to the DSP. The ADSEL1 and ADSEL2 lines of Figure 3.5 are multiplexed so that they can be driven by the QBUS or DSP. The multiplexing function is controlled by the data acquisition controller. When the data acquisition controller starts the DSP execution of program code, ADSEL1 and ADSEL2 are driven by the DSP I/O port. When the data acquisition controller enables QBUS access, ADSEL1 and ADSEL2 are driven by the QBUS controller. If the data acquisition controller is acquiring data from the FIFO data buffers, the QBUS controller tristates ADSEL1 and ADSEL2 to enable data acquisition controller control. The latched data bits from the DSP I/O port are also used to reset the DSP/host FIFO buffer. The FIFO is reset by writing a zero into the DSP I/O port and setting high an external signal on the DSP to

enable the output. The FIFO reset signal from the QBUS controller is pulled high with a 10k ohm resistor. This allows normal FIFO memory operation while the output is not enabled (tristated). The tristating of this signal allows normal memory mapping usage of the DSP I/O port.

The QBSP is mapped into the host computer I/O space by eight DIP switches that are user defined. These DIP switches define the base address of the QBSP by comparing an eight bit word to the BDAL5 thru BDAL12 address/data lines on the QBUS. The device used for the comparison is a 74ALS520 [8] which has an enable signal tied to the I/O select (BBS7) line on the QBUS. The control signal (EQ) from the 74ALS520 is input to a pair of PLDs which generate the necessary VALCYL and QCSYNC signals for the QBUS controller. Figure 3.6 provides a block diagram of the address select logic and documents how the DIP switch settings are used to select a QBSP base address. Notice that BDAL4 is decoded so the QBSP responds to only one base address.

Input/output Ports

The DSP input port is used to read status information from the A/D conversion electronics. This port is buffered using PLDs which can be programmed to either invert or not invert the input signal as

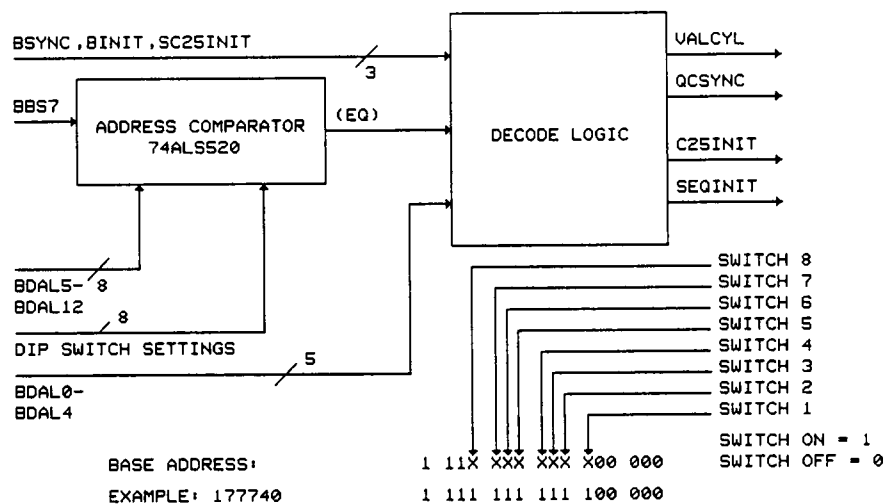


Figure 3.6. QBSP Base Address Selection.

required by the application. The input port connector also contains access to INT1 which is an external interrupt to the DSP. This is required when the processor must continue processing data until interrupted by an active low transition on INT1. When this transition is received, the DSP will finish its current instruction, save the processor state, and then vector to location four to begin executing code. The input port can be 50 ohm terminated by installing jumpers located on the board.

The DSP output port is used to control gain and offset in the A/D electronics. This port is latched by the processor and buffered to provide up to 50mA of current source/sink capability. This port is latched using PLD devices which can also be programmed to provide inverted outputs if required. This capability simplifies control software development since the programmer need not be concerned with the actual logic levels required for a particular device. The specific logic requirements can be programmed into the PLD by the engineer.

The QBUS control/status register can be read or written to by the host computer. A write to the CSR accesses the control port. This port contains two fields. The lower three bits, D0 thru D2, are read by

the data acquisition controller and the remaining bits, D3 thru D7, are read by the DSP. This provides the capability to tailor the QBSP to many different applications by simply writing new software to be run in the data acquisition controller and DSP. The control port can then be used to control software flow using D0 thru D7.

When the CSR is read, various status bits are passed to the host computer. These status bits contain information such as which controller device has access to the DSP data bus and whether or not the the QBUS FIFO buffer is empty. When the DSP is reading the control port, it is important to mask the unused bits since the QBSP design only drives the used bits during a control port access. Complete information of the status register is contained in Figure 3.7 and Table 3.3 describes its contents.

The LED diagnostics port is connected directly to the DSP and is used to provide a visual verification of the DSP operation. This port is mapped into the DSP I/O space and can be accessed only by the DSP. The FPA software uses this port to indicate which section of code is being executed by the DSP. When the DSP is executing diagnostics, this port is used to display error codes which inform the user of problems on the

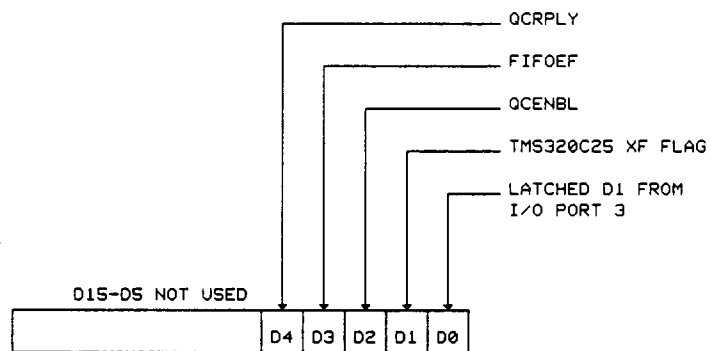


Figure 3.7. QBSP Control Status Register Contents.

Table 3.3. Control Status Register Description.

QCRPLY QCENBL		Description
0	0	Microsequencer has total memory access.
0	1	Microsequencer has bank select access and QBUS controller has read/write access.
1	0	Microsequencer has read/write access and QBUS controller has bank select access.
1	1	QBUS controller has total memory access.
XF FLAG		Description
0		QBUS FIFO will reset.
1		QBUS FIFO will not reset.
FIFOEF		Description
0		QBUS FIFO is empty.
1		QBUS FIFO is not empty.

QBSP board. The function of this port is always defined by the application DSP software.

Digital Signal Processor

The digital signal processor chosen for the QBSP is the Texas Instruments TMS320C25 shown in Figure 3.8 [3]. There are four reasons why the TMS320C25 digital signal processor is an acceptable choice for the QBSP design. First, the processor operates at 10 MIPS and uses a highly pipelined instruction set. Secondly, the processor is able to directly address 64K of external data memory and 64K of external program memory. Thirdly, the processor is designed to operate in a multiprocessor environment. This gives it the ability to tristate its address, data, and control lines. Fourthly, the processor uses complementary metal oxide semiconductor (CMOS) technology which provides low power consumption.

The TMS320C25 provides separate address spaces for data, program and I/O memory as outlined in Figure 3.9. A pin (MP/MC) on the TMS320C25 allows the program memory to be configured in one of two modes: a microprocessor mode and a microcomputer mode. The QBSP uses the microprocessor mode since this places all of the interrupt vector locations in external memory. It

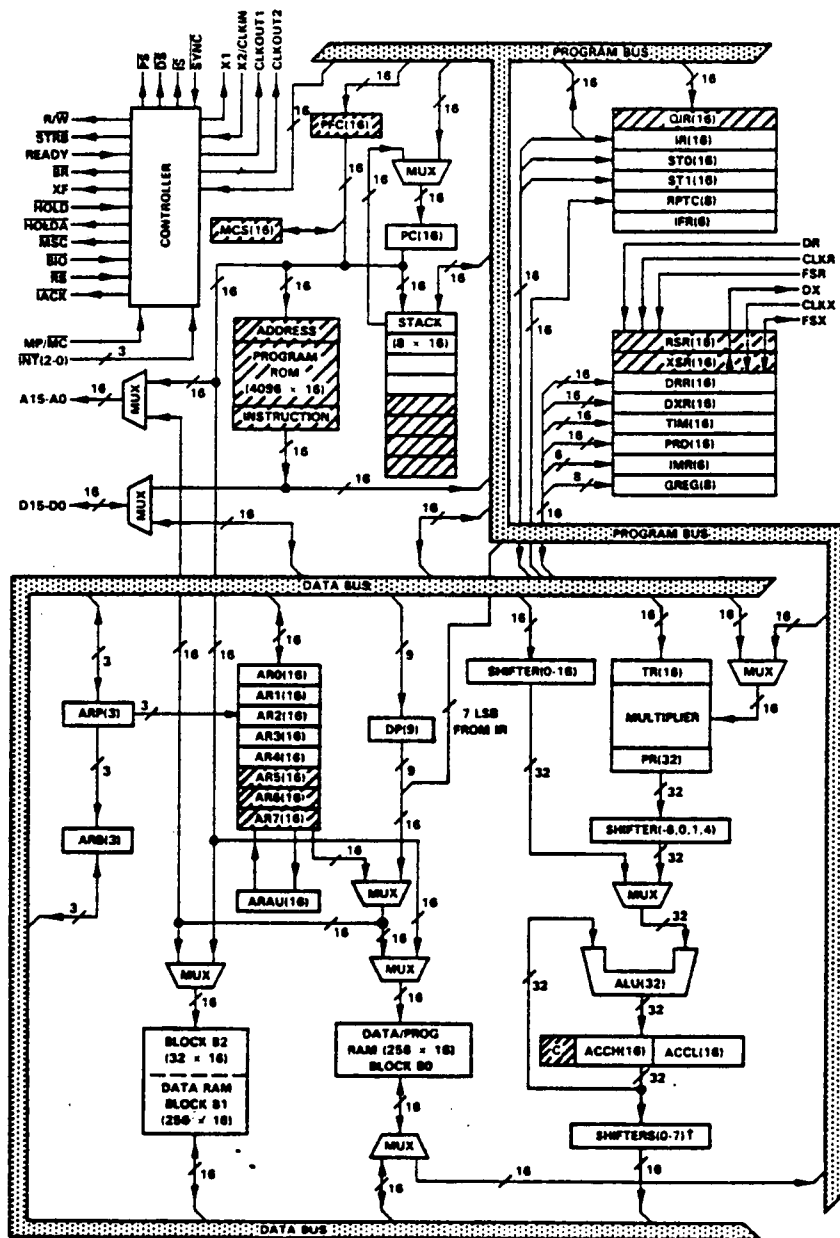
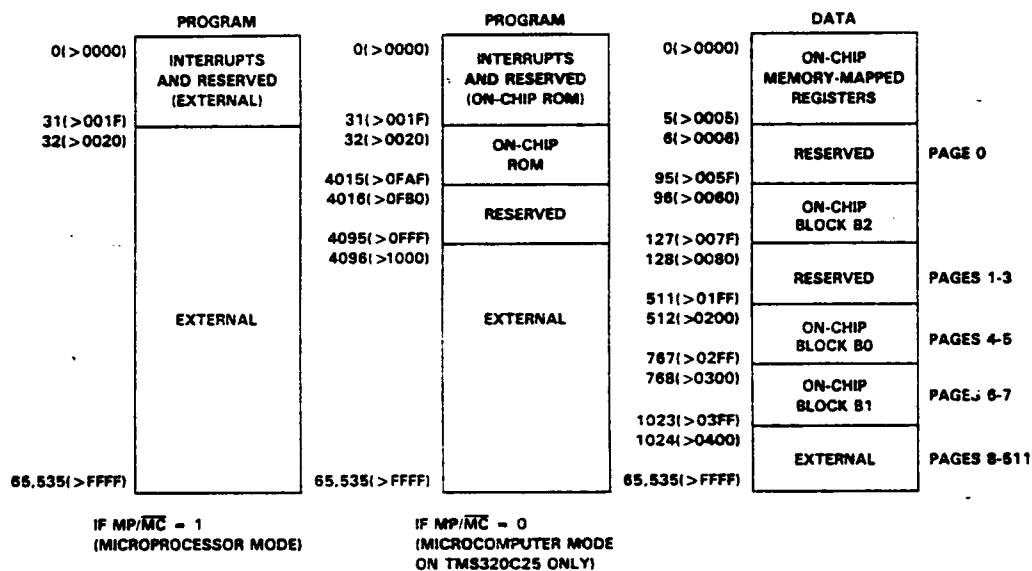
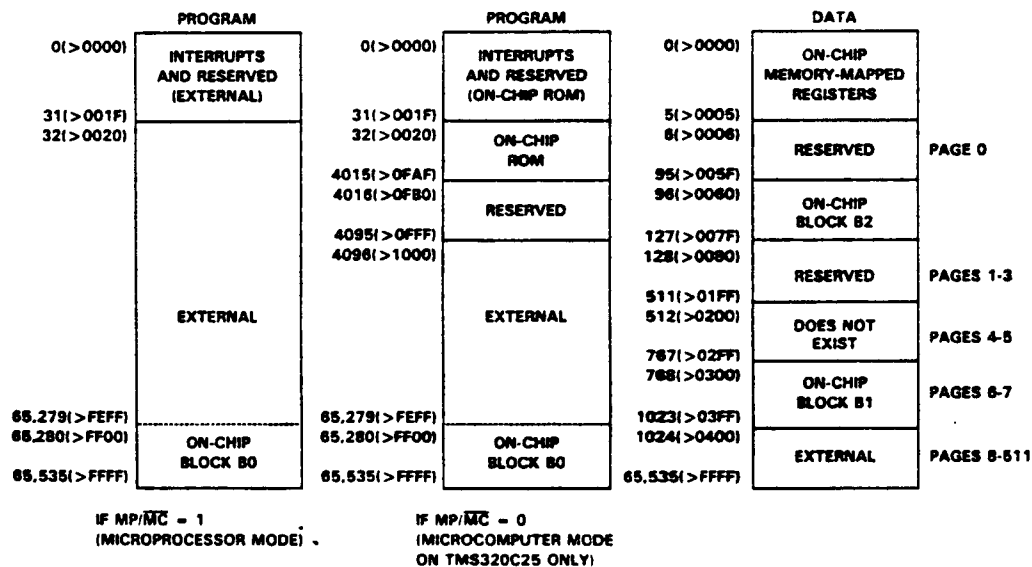


Figure 3.8. TMS320C25 Digital Signal Processor.



(a) MEMORY MAPS AFTER A CNFD INSTRUCTION



(b) MEMORY MAPS AFTER A CNFP INSTRUCTION

Figure 3.9. TMS320C25 Memory Configuration.

is the program memory in Figure 3.9 that can be mapped to ROM or RAM by the QBSP data acquisition controller. Internal DSP data memory consists of blocks B0, B1, and B2 which comprise 544 words of internal data RAM. The TMS320C25 has the unique capability to map block B0 into data RAM or program RAM by use of the CNFD and CNFP instructions [3]. Block B0 is located in pages 4 thru 5 as data memory, and locations 65,280 thru 65,535 as program memory. This capability allows short programs to be loaded into block B0 and executed at higher speeds than if they were resident in external RAM. The memory map of Figure 3.11 also indicates that external data RAM starts at location 1024. One of the four 64K blocks of the QBSP data memory is always mapped into this data space.

As shown in Figure 3.10, the TMS320C25 has a 32 bit arithmetic logic unit (ALU) and accumulator which can implement a wide range of logical and arithmetic instructions. Many of these instructions execute in a single clock cycle. The arithmetic instructions are primarily used in signal processing applications such as FPA data reduction. The logical instructions are needed in high speed bit manipulation required for control applications. Figure 3.10 illustrates the fact that the ALU always obtains one input from the

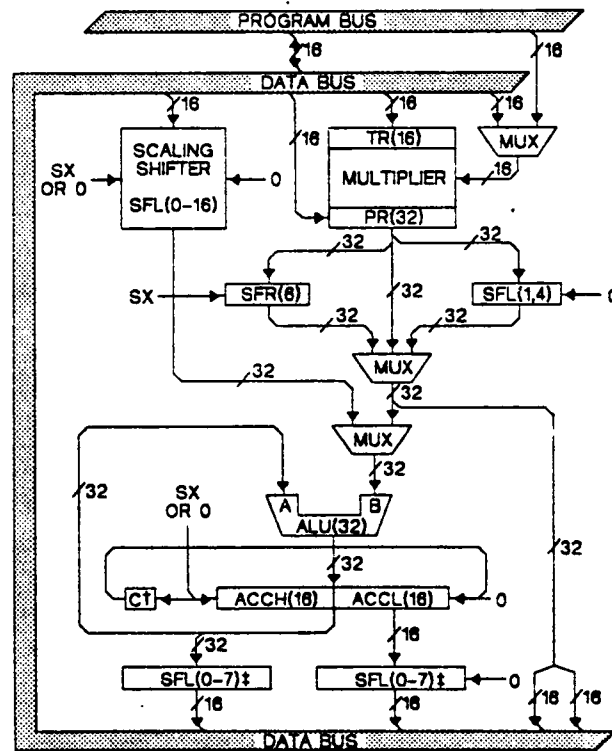


Figure 3.10. TMS320C25 Arithmetic Logic Unit.

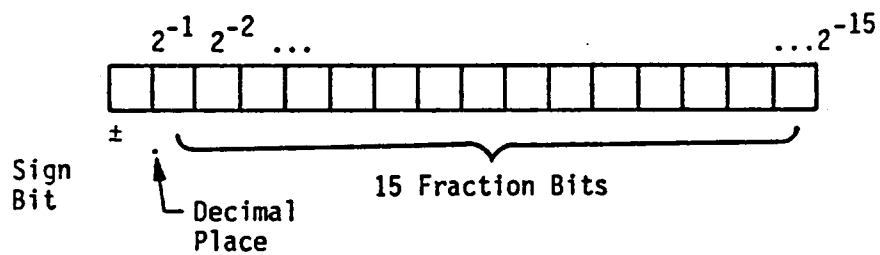
accumulator, while the other input may be provided from the product register (PR) or the input scaling shifter which has fetched data from RAM. After the the ALU has completed its instruction, the result is always passed back into the accumulator. The accumulator is split into two 16 bit segments so that it may be stored into data RAM. When storing data into RAM, the scaling shifter can do a left shift of 0 to 7 places. When loading the accumulator from data RAM, the data can be shifted left 0 to 15 places. This capability of the ALU is used by the FPA data reduction software to store all 32 bits of the noise data thus maintaining greater computational fidelity.

The 32 bit accumulator gives the TMS320C25 a large dynamic range. However, care must be taken to insure that integer overflow does not occur while processing data. The TMS320C25 has three solutions to overflow management. First, after any ALU instruction, the TMS320C25 can execute the branch on overflow instruction (BV). This allows the processor to enter an error handling routine to rescale the data as needed when the overflow condition occurs. Secondly, it is also possible to force the ALU multiplier to right shift its data before it is summed into the accumulator. This assures that no overflow is possible

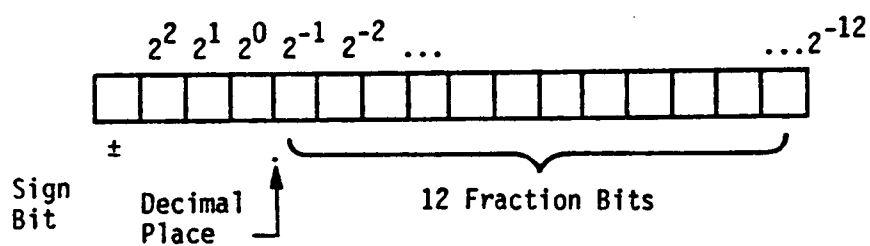
for a defined number of accumulations equal to 2^n where n is the number of bits shifted right. Thirdly, the TMS320C25 has an instruction which causes the accumulator to saturate when the processor reaches its most positive or negative values. The third method is implemented in the FPA data reduction software.

The TMS320C25 uses a fixed point arithmetic format. This format requires that care be taken when multiplying numbers in the ALU. Different fixed point formats are supported by the TMS320C25 because of its ability to left shift data as it is being moved from the accumulator into data RAM. This ability eliminates the extra sign bits generated by fixed point multiplication. Figure 3.11 shows the differences between Q15 and Q12 arithmetic formats. The FPA data is put in Q15 format before processing. This insures that no overflow will occur when multiplying data since the product of two fractions is also a fraction. The Q12 format is useful in applications such as digital filters which have coefficients that cannot be scaled to fractional values.

The BIO pin in Figure 3.8 is used to provide the DSP with status information about external devices. Although it can not be classified as an interrupt, there is a special purpose instruction (BIOZ) [3] which



a. Q15 format



b. Q12 format

Figure 3.11. Q15 and Q12 Arithmetic Formats.

can be used to poll the status of BIO. The QBSP design uses this capability to monitor the status of the QBUS FIFO buffer I/O port. If the buffer is full, BIO will go active low. Before placing data into the FIFO buffer, the software uses BIOZ to insure that the buffer is not full. If the buffer is full, the software can set up its own internal buffer or simply wait until the host computer acquires data from the QBUS FIFO buffer port.

Power Requirements

The QBSP obtains all of its power from the QBUS. The voltages applied to the QBSP must be between 4.5 and 5.5 volts with an RMS noise level below 100mV. The current required to operate the QBSP will not exceed 3.0 amps. Under normal usage the current averages 1.5 amps because only a portion of the I/O devices are active at a given time. The low power consumption of the QBSP enables several boards to reside in a single backplane for parallel operation.

CHAPTER 4

SYSTEM TIMING

Introduction

An important aspect of any hardware design is proper timing. This chapter details the timing considerations made while designing the QBSP. The use of wait states and how they relate to the DSP maximum clock rate is also discussed.

QBUS Timing

Figure 4.1 details the timing considerations for host computer access to data memory, program memory, FIFO buffer and the control status register. The QBUS address lines become active 75ns prior to BSYNC going active low and remain active 25ns after BSYNC is active. The QBSP logic decodes the address and if valid asserts VALCYL low and QCSYNC high within 20ns after BSYNC is active. This allows the QBUS controller to latch the valid address bits into its internal logic. A minimum of 100ns after BSYNC, the QBUS will assert BDIN low for a QBUS read request, or BDOUT low for a QBUS write request. Once a read or write request is made, the QBSP responds within 50ns with BRPLY low to

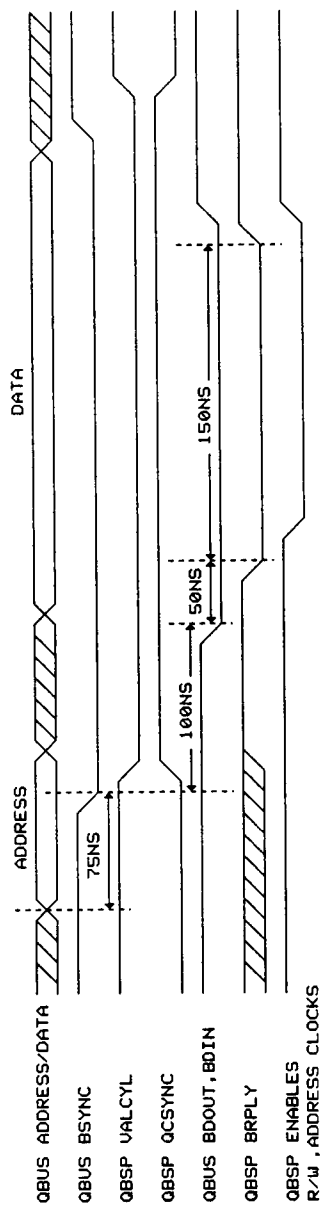


Figure 4.1. QBUS Interface Timing.

acknowledge the QBUS request. A minimum of 10ns after BRPLY low, all of the necessary QBSP control lines such as read/write, address clocks and chip enables are also active. After receiving BRPLY low, the QBUS asserts or accepts data on BDAL0 thru BDAL15 a minimum of 150ns prior to deasserting BDIN or BDOUT. When BDIN or BDOUT is deasserted, the QBSP responds by deasserting BRPLY within 50ns. The QBUS then allows 200ns for bus deskew and deasserts BSYNC. The BSYNC signal always remains active throughout a complete bus cycle.

Digital Signal Processor Timing

The TMS320C25 uses a four phase clock to enable its internal operation. Figure 4.2 shows how the four phases are generated from the external oscillator. The time interval between the Q1, Q2, Q3 and Q4 phases is referred to as Q and represents one quarter phase of a TMS320C25 cycle. Systems which use a 40MHz external oscillator have a Q value of 25ns.

Figure 4.3 details the timing considerations for a TMS320C25 access to program memory, data memory and I/O ports. The address lines and control signals from the TMS320C25 are asserted Q-15ns prior to STRB asserted low. The STRB signal is used to strobe bus data into or out of memory and I/O devices. The STRB

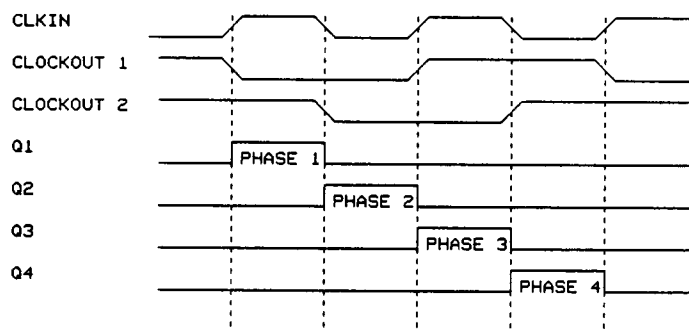


Figure 4.2. TMS320C25 Four Phase Clock.

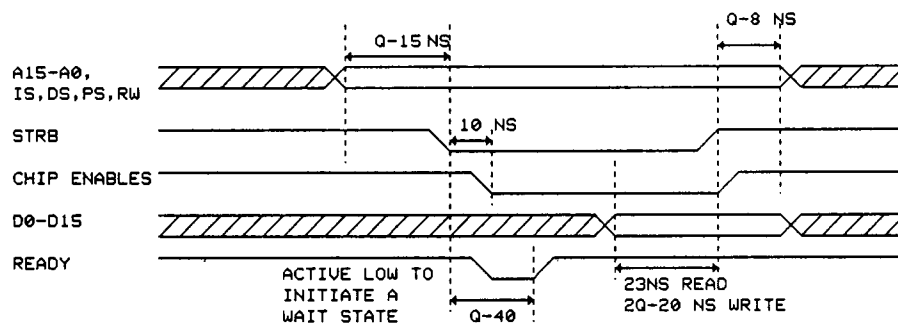


Figure 4.3. TMS320C25 Read/Write Timing.

signal stays asserted for 2Qns and is deasserted Q-8ns prior to address and data being deasserted. The data must be set up 23ns before STRB is deasserted for a read operation and 2Q-20ns for a write operation. As shown on Figure 4.3, the QBSP decoding logic inserts a 10ns delay after STRB is asserted before the memories and I/O ports are selected. The QBSP logic has the capability to insert a wait state using the READY input to the TMS320C25 [3]. At the beginning of Q1, the microcycle complete (MSC) signal is asserted low and remains low until the end of Q2. This enables the QBSP logic to assert READY low during Q2 if the device being accessed requires a longer cycle time. When the processor samples READY during Q2 and finds it asserted, it will extend the current access one microcycle. At the beginning of Q1 for the next microcycle, MSC will not be asserted and the device access will be completed at the end of that microcycle. Wait states are used to interface with devices that are slower than the TMS320C25 without reducing the processors' internal execution speed.

The maximum clock frequency can be obtained from individual device timing information recognizing that the slowest device in the system will be the limiting factor. The 256K data memory module is the slowest

device on the QBSP as discussed in chapter 3. The timing information in Figure 4.3 shows that the following specifications must be met:

$$TADsu < Q - 15ns \quad (4.1)$$

$$TDAsu < 2Q - 22 - Tpd \quad (4.2)$$

The address setup time ($TADsu$) for data memory is 0ns and data setup time, $TDAsu$, is 40ns [7]. The propagation delay through various PLDs (Tpd) is 10ns. These requirements place a restriction on Q of 36ns which limits the system to approximately 27MHz. By using a single wait state, Equation 2 would become

$$TDAsu < (2Q - 22 - Tpd) + 3Q \quad (4.3)$$

Using a single wait state enables the system clock to be set at 40MHz, thus maintaining maximum internal execution of clock cycles and providing maximum resolution to the time base generator.

Microsequencer Timing

The microsequencer inputs have a setup time of 18ns with a hold time of 0ns [4]. This enables a 10ns PLD to be clocked simultaneously with the microsequencer and still provide stable microsequencer

inputs for clock periods longer than 38ns. The QBSP has an onboard 20MHz clock to drive the microsequencer.

The startup sequence of the DSP is controlled by the microsequencer. As shown in Figure 3.6, the QBUS BINIT signal is used in the control logic to reset the microsequencer and DSP. Any time the QBUS issues a system reset by asserting the BINIT signal, the microsequencer is reset. This causes the microsequencer to reset the DSP and place it in hold mode, enable QBUS access to data memory, and then wait for a command code from the host computer. If a command code is sent to start the DSP executing code, the microsequencer will turn off QBUS access to data memory, release the DSP from hold mode, wait for three clock cycles to allow the address and data lines to stabilize, then release the DSP reset signal to begin program execution. Figure 4.4 documents a typical startup sequence of the QBSP entering the FPA data acquisition code.

CLOCKOUT 1

A0

A1

A2

D0

D1

D2

D3

HOLD

RESET

STRB

PS

DS

IS

R/W

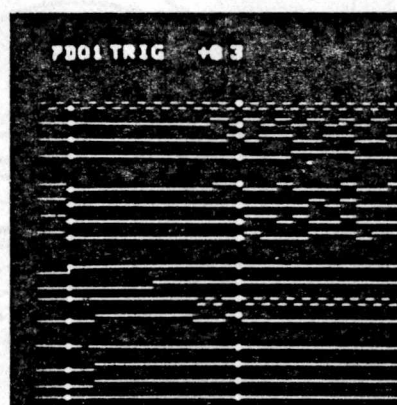


Figure 4.4. TMS320C25 Startup Sequence.

CHAPTER 5

SYSTEM SOFTWARE

Introduction

The software required for the FPA data reduction is described in this chapter. Much of the code developed for the QBSP involved host computer interface subroutines. These subroutines were written in a general format so they could be used more readily in future QBSP applications.

QBUS Interface Software

Several LSI11/73 assembly language routines were written to enable QBUS control of the QBSP. These routines can be called by FORTRAN and include routines to download QBSP program or data memory, start DSP execution from RAM or ROM, halt DSP program execution, initialize the QBSP address generators, interface with the CSR, and read data from the DSP/host computer FIFO port. A menu driven FORTRAN program was also developed to checkout each module and provide a general purpose, user friendly interface with the QBSP. Since the QBSP was designed using sequential address generators for data memory, the software developed to interface with

the 128K buffer cards of Figure 1.3 on page 15 is easily modified to interface with the QBSP.

FPA Application Software

The FPA data reduction software was developed using the TMS320C25 assembler/linker. The software developed is downloaded into program RAM of the QBSP and, when executed, receives all of its instructions from the control I/O port. Before the software is executed, the host computer must load data memory block one, starting at location 1024, with four variables. The variables include number of rows in the FPA, the number of frames to process, the number of right shifts for integer division, and the row 1 alignment offset. The data reduction software is divided into four modules which perform various signal processing functions on the FPA data memory contents. An FPMMAIN program provides interfacing with the control I/O port and determines which data reduction module is to be executed by the DSP. An FPMMEAN program computes the mean value of each FPA pixel over the number of frames specified by the variables loaded into data memory block one. An FPMRMS program computes the RMS noise on each pixel using both correlated processing and uncorrelated processing algorithms. A SHUFFL program

module is used to organize the FPA data if required and convert the data into a Q15 two's complement format.

CHAPTER 6

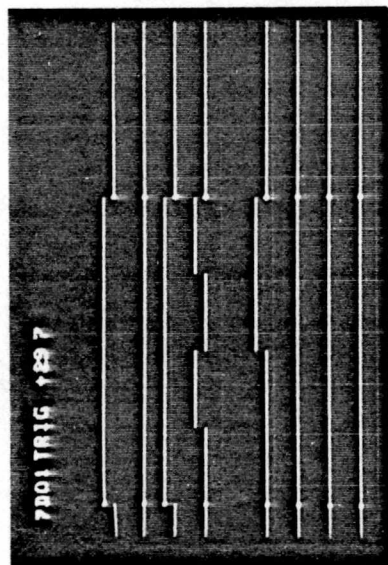
PROCESSOR CHECKOUT

Introduction

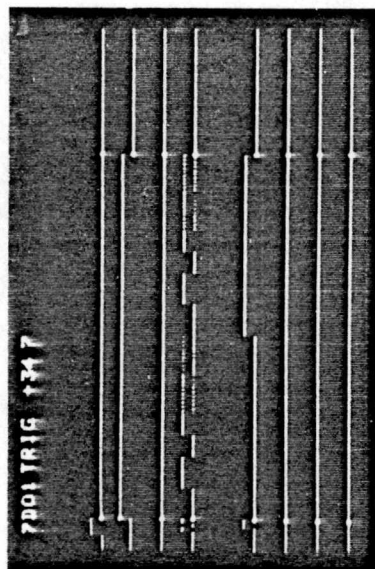
Critical requirements for the QBSP include real time data reduction and accurate computation of FPA noise parameters. This chapter will develop tests used to verify that both of these requirements are met with the QBSP design. Performance data from each test is also presented.

FPA Application Software Benchmarks

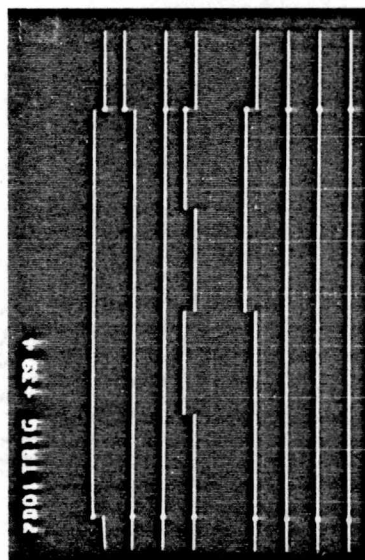
The speed performance of the FPA data reduction software was measured using a Tektronix logic analyzer. Each of the data reduction modules outputs a unique identification code to the LED I/O port as it is executed. This enables various sections of code to be easily benchmarked by setting the logic analyzer to trigger on the identification code for that section and then measuring the time for which that trigger is valid. Figure 6.1 documents the results for the three sections of code checked. The photographs taken show the identification code in the first three lines and the 256K memory mapping signals (ADSEL1,ADSEL2) in the



DATA CONVERSION 59.4MS



FPA PIXEL NOISE 173MS



FPA PIXEL INTENSITY 7.9MS

Figure 6.1. FPA Data Acquisition Code Benchmarks.

next two lines. The test showed that for 128 frames of data, a 64 row FPA required 59ms to convert the data into two's complement, 7.9ms to compute the pixel intensities and 173ms to compute the pixel uncorrelated and correlated noise. The FPA integration time was 1ms per frame. The test demonstrates that 128ms were spent acquiring data, but only 66.9ms were required to compute the pixel intensities. Recall from chapter 1 that the mean calculations required .689 MOP. Therefore, 66.9ms implies a computing power greater than 10 MOPS. The noise computations should be done rapidly to minimize the time required for testing. However, there is no real time requirement for these computations. The 173ms is sufficiently small to provide a reasonable chamber occupancy time.

It is important to note that when computing pixel intensities, most of the time is spent converting the data into a two's complement format. This time can be eliminated by modifying the A/D hardware to output two's complement data directly thus providing a system computational speed of 83 MOPS. This level of performance is more than enough to insure real time signal processing for even the faster FPAs.

FPA Noise Computations

As discussed in chapter 1, the FPA pixel noise is computed using the expression

$$\begin{aligned} \text{RMSp}^2 &= \text{RMSVs}^2 + \text{RMSVr}^2 - \\ &2/N \sum_{i=1}^N [\text{Vs}(i) - \overline{\text{Vs}}][\text{Vr}(i) - \overline{\text{Vr}}] \end{aligned} \quad (6.1)$$

where RMSp is the noise on a single pixel element. As shown in Figure 1.2 on page 5, the difference between uncorrelated and correlated processing depends on the frame from which Vr is taken. Equation 6.1 shows that the pixel noise can also be obtained from the RMS noise of Vs and Vr, and the cross-correlation at lag Tau between Vs and Vr. If the same noise input is applied to both Vs and Vr, then the cross-correlation becomes an autocorrelation function at a given lag Tau. The autocorrelation function for a signal can be obtained by taking the inverse Fourier transform of its power density spectrum (PDS) [9]. For identical noise applied to Vs and Vr in Equation 6.1, the autocorrelation function will depend on the amplitude and PDS of the input noise, and the time interval, Tau, between Vs and Vr samples. Figure 1.2 on page 5 shows that the value of Tau for uncorrelated processing is much less than for

correlated processing. Therefore, the theoretical QBSP output for noise inputs can be predicted from the bandwidth of the input noise, and the value of Tau for uncorrelated and correlated processing based on the FPA integration time.

The test setup outlined in Figure 6.2 was used to checkout the QBSP noise calculations. The PAR113 amplifier provided a noise source by connecting a 10K ohm resistor to its input and adjusting the gain to obtain a known RMS output voltage as measured by an RMS voltage meter. The 10K ohm resistor generated thermal noise which is a good wideband white noise source [9]. The PAR113 output was then amplified and input to channel 1 of the FPA data system on page 27. In addition to controlling the RMS noise level, the PAR113 amplifier can also control the noise bandwidth with a single pole lowpass output filter. An expression for the PDS was obtained from the known filter characteristics of the PAR113 amplifier and may be written as,

$$S(W) = \text{RMSn}^2 [(2)W_c / (W_c^2 + W^2)] \quad (6.2)$$

where $S(W)$ is the PDS of the PAR113 output, RMSn^2 is the total power in the noise, W_c is the 3db cutoff frequency of the PAR113 lowpass filter, and W is the

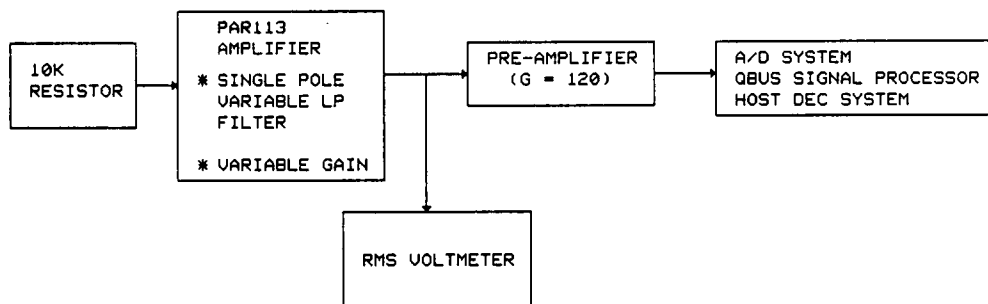


Figure 6.2. Noise Measurement Test Configuration.

angular frequency of each noise component. By taking the inverse fourier transform of the known PDS, an expression for the theoretical QBSP output can be written as,

$$\text{RMSp}^2 = (2)(\text{RMSn})^2 - \frac{(2)(\text{RMSn})^2}{\text{EXP}[(-Wc)|(\text{Tau})|]} \quad (6.3)$$

where Tau is the time lag for either correlated or uncorrelated processing. An experiment was completed which set the PAR113 output to various bandwidths and compared the QBSP reduced data with what was predicted by Equation (6.3). Figure 6.3 documents the result of this experiment and shows acceptable agreement between the theoretical and QBSP noise values. The slight disagreement at low bandwidths is attributed to poor stop band rejection by the PAR113 low pass filter.

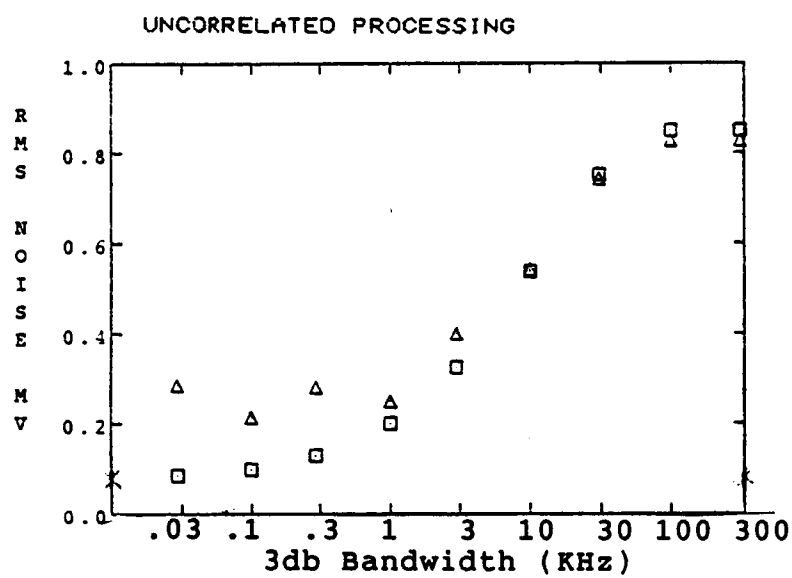
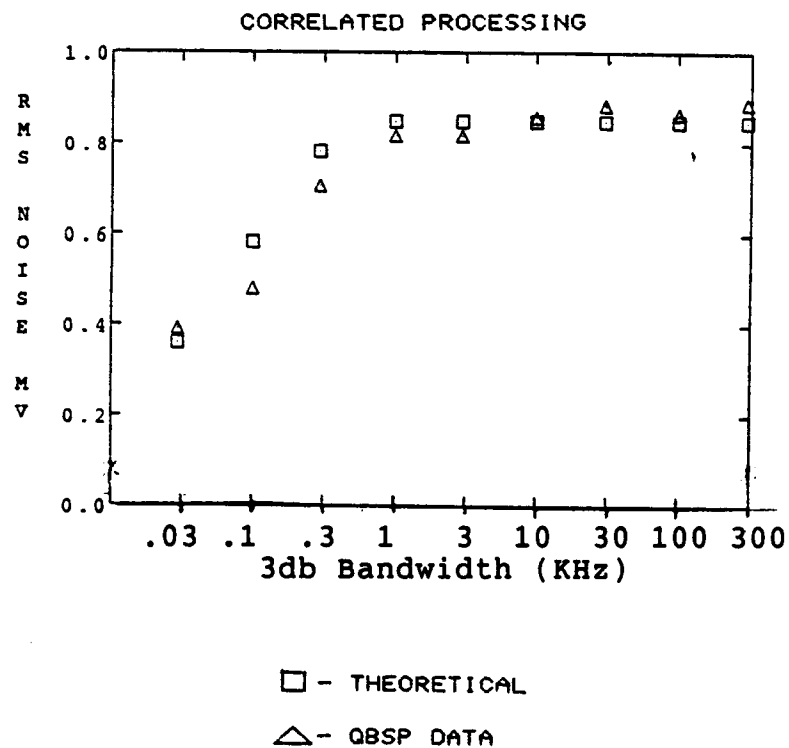


Figure 6.3. Theoretical .vs. QBSP Noise Data.

CHAPTER 7

SUMMARY AND CONCLUSION

A digital signal processor which meets the AEDCs FPA data acquisition and signal processing requirements was designed, fabricated, and tested. The signal processor will accept data from two FPA channels and compute the intensity of each pixel in real time. In addition, the signal processor will rapidly compute the noise associated with each pixel using uncorrelated and correlated signal processing. A test to verify correct noise processing was developed and used to successfully checkout the noise computations.

Parts were selected for the signal processor which enabled it to achieve its performance at less cost than other data systems. The signal processor contains programmable logic devices and programmable sequencers which provide design flexibility without the need to change the printed circuit board. These devices were used to provide all of the logic design on the signal processor and to interface the signal processor with a QBUS host computer system. The majority of the ICs used on the signal processor used low power technology. This enabled several processor boards to reside in a single QBUS backplane providing a

multiprocessor solution to FPA data acquisition and reduction.

Although the signal processor meets specific FPA requirements, it can also be used in other signal processing applications. The programmable architecture and QBUS interface make it well suited for many high speed data acquisition applications which require rapid signal processing. In addition, the parallel input and output ports can be used for applications needing high speed control. The serial port enables additional communication with external devices for control purposes.

The signal processor met all of its performance goals. Although no design changes are needed, there are two improvements which could be made. First, a direct memory interface between the signal processor and QBUS could be added. This would provide quicker transfer of data from the signal processor into the host computer memory. Secondly, the speed of the data memory could be increased to eliminate the need for a wait state when accessing data memory. Both of these changes would provide better throughput of data for future FPA devices which will operate faster than the current devices.

LIST OF REFERENCES

LIST OF REFERENCES

1. Dereniak, E.L., Crowe, D.G. (1984): "Optical Radiation Detectors.", John Wiley and Sons, New York.
2. Digital Equipment Corporation. "Microcomputers and Memories". Manard, Massachusettes, 1982, pp. 219-265. (Order Code EB-20912-20)
3. Texas Instruments Corporation. "Second-Generation TMS320 Users Guide". Houston, Texas, 1987. (Order Code SPRU014)
4. Altera Corporation. "Altera Data Book". Santa Clara, California, 1988.
5. Cypress Semiconductor Corporation. "CMOS Data Book". San Jose, California, 1988.
6. Integrated Device Technology Corporation. "High Performance CMOS Data Book". Santa Clara California, 1988.
7. Hitachi Corporation. "HM622526LP Series, HM622256LFP Series 32768-Word x 8 Bit High Speed CMOS Static RAM Data Sheet". San Jose, California, 1988.
8. Texas Instruments Corporation. "ALS/AS Logic Data Book". Houston, Texas, 1986. (Order Code SDAD001B)
9. Peebles, P. Z., Jr. (1980): "Probability, Random Variables And Random Signal Principles", McGraw-Hill, New York.
10. Data I/O Corporation. "ABEL 3.0". Redmond, Washington, 1988.

VITA

Robert Henry Fugerer was born in New York, New York. He received his early education in Gainesville, Florida and was graduated from Gainesville High School in 1977. He entered David Lipscomb College in the winter of 1978 then transferred to Vanderbilt University in 1980. A transfer of credit between the two schools allowed him to receive a Bachelor of Science degree in Engineering Science from David Lipscomb College, and a Bachelor of Science degree in Electrical Engineering from Vanderbilt University in the fall of 1983.

After graduation, he accepted a position with Calspan, Inc., AEDC group, in January, 1984, where he is currently employed as a research engineer in the technology section of the space systems branch. His work involves the development of digital hardware and related software with an emphasis on data acquisition and signal processing. The author entered The University of Tennessee Space Institute in the spring of 1984 and received his Master of Science degree in Electrical Engineering in December, 1988.

He is married to the former Susan L. LaVelle. They have two children, Benjamin and Jeremy.