A Single Chip Low Power Implementation of an Asynchronous FFT Algorithm for Space Applications

Bruce W. Hunt

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/97D-08

A SINGLE CHIP LOW POWER IMPLEMENTATION OF AN ASYNCHRONOUS FFT ALGORITHM FOR SPACE APPLICATIONS

THESIS

Bruce William Hunt
Second Lieutenant, USAF

AFIT/GCS/ENG/97D-08

AFIT/GCS/ENG/97D-08

A SINGLE CHIP LOW POWER IMPLEMENTATION OF AN ASYNCHRONOUS FFT ALGORITHM FOR SPACE APPLICATIONS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Bruce William Hunt, B.S.E.E.

Second Lieutenant, USAF

December 1997

AFIT/GCS/ENG/97D-08

A SINGLE CHIP LOW POWER IMPLEMENTATION OF AN ASYNCHRONOUS FFT ALGORITHM FOR SPACE APPLICATIONS

Bruce William Hunt, B.S.E.E.

Second Lieutenant, USAF

Approved:

[illegible]	14 Nov 97
Maj Don S. Gelosh Thesis Advisor	Date
[illegible]	14 NOV 97
Lt Col David M. Gallagher Committee Member	Date
[illegible]	28-Nov-97
Dr. Kenneth S. Stevens Committee Member	Date
Bruce W. Suter	14 Nov 97
Dr. Bruce W. Suter Committee Member	Date
[illegible]	13 Nov 97
Maj Charles P. Brothers Committee Member	Date

Acknowledgements

Out of the depths of knowledge darkness I have risen to know and understand many new things making the completion of this thesis possible. For this, I have many people to thank. First of all, my advisor, Major Don Gelosh for providing the VLSI instruction necessary and his strong support despite numerous administrative responsibilities. I must also thank my two remote committee members, Major Charles Brothers who, doubling as my sponsor, traveled to attended many of my design reviews and was always quick to answer my numerous questions about radiation and space. Also I am grateful to Dr. Kenneth Stevens for the hours I spend on the phone with him, even into the early morning hours on many occasions to get the asynchronous synthesis tools to function properly. Dr. Stevens is also worthy of thanks for his many reviews of my papers and the raw instruction in the area of asynchronous systems. Dr. Suter was also helpful to me in the area of signal processing, especially in understanding the unique algorithm at the root of this project. Finally, I must thank Lt Col David Gallagher for helping me out at the start of this project and occasionally making me step back to realize what I really learned.

This acknowledgements section would not be complete without thanking the other "slaves" that I have spent my time with here at AFIT. This especially goes to Mr. Steve Parmley and 1 Lt Geogre Roelke for all the lunch and dinner breaks including deep conversation about anything not related to VLSI. Additionally, I must mention each of those I have worked with here in the lab: Capt Joe Bouchard, Capt Glen Kading, Capt John Ortiz, Capt Randy Whitman, 2 Lt George Harrison, 2 Lt Paul Kladitis, and Greg Richardson, our illustrious administrator.

The most important event during my AFIT stint has to be the marriage to my wife, Amber. Despite our geographical separation, I always feel her near to my heart as she has been a huge source of encouragement and support to complete a good thesis despite constant frustration. I think she is the only person that will be happier than I when this experience is over.

Above all, I must thank my God who, through his Holy Book, has told me "I can do all things through Christ which strengtheneth me" (Phillipians 4:13). A special thanks must go here to Capt Randy Whitman who also provided me with some inspiration as a fellow Christian engaging in many discussions during our late evenings in the VLSI lab.

Bruce William Hunt

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCS/ENG/97D-08

Abstract

A fully asynchronous fixed point FFT processor is introduced for low power space applications. The architecture is based on an algorithm developed by Suter and Stevens specifically for a low power implementation. The novelty of this architecture lies in its high localization of components and pipelining with no need to share a global memory. High throughput is attained using large numbers of small, local components working in parallel. A derivation of the algorithm from the discrete Fourier transform is presented followed by a discussion of circuit design parameters; specifically, those relevant to space applications. The generic architecture is explained with a survey of the 16-point FFT architecture specific to this project. An implementation, which included a test chip fabricated through MOSIS, is described. Finally, simulation results based on layout extractions are presented and an outline for future work is given.

A SINGLE CHIP LOW POWER IMPLEMENTATION

OF AN ASYNCHRONOUS FFT ALGORITHM FOR SPACE APPLICATIONS

I. Introduction

1.1 Introduction

This document presents a multipurposed research effort. In May 1997, Air Force Institute of Technology faculty members Bruce W. Suter and Kenneth S. Stevens submitted a patent proposal for a low power architecture for a Fast Fourier Transform (FFT) processor [20]. The first goal of this research was to validate the claims made in the patent proposal by implementing the architecture in complementary metal oxide semiconductor (CMOS) technology. Because this is a low power architecture, it became attractive for space applications so a low power implementation that functions in a radiation-hostile environment was also desired. This chapter introduces some important background information including the problem statement and research methodology. The final section provides an overview to the rest of the document.

1.2 Problem Statement

Asynchronous technology is just starting to make a firm impact in the very large scale integration (VLSI) design community. Although asynchronous design has a lot of potential, researchers admit that not many working commercial-scale designs have been fabricated to demonstrate its advantages. The literature review found few single chip FFT processor implemented in an asynchronous design. Also, current FFT applications for space do not use many low power design techniques.

The purpose of this research is to determine suitability of the Suter/Stevens algorithm for a CMOS implementation. Speed and power metrics will be collected and compared to existing DSP applications to aid in this determination. Both speed and power are important for validation of the algorithm and architecture. This research also hopes to demonstrate that the speed and low power benefits extend to space-bound circuits. In a space bound system, speed and power are traded off to reach goals that will meet system requirements. Lower power architectures will hopefully lead to faster designs.

1.3 Methodology

To implement this algorithm in silicon, many design steps are necessary. First of all, high level decisions will be made regarding CMOS technology, data type, logic type (i.e., static, pre-charge, pass-transistor, hybrid, etc.), point size, etc. Once these are narrowed down, there are many futher design decisions to be made at lower levels when each component of the architecture is being built. It is best to use a high level simulation language, such as VHDL, to test potential implementations before a lot of time is invested. Decisions during the design process will change as there is a convergence toward the "optimal" design. When enough designs are complete (i.e., laid out in MAGIC), extractions and tests need to be run to determine the likelihood of functionality and performance.

1.3.1 Constraints and Assumptions. First of all, at least the minimum iteration of the Suter/Stevens architecture shall be implemented to prove the concept. Secondly, the layouts shall be performed using a radiation tolerant cell library provided by the sponsoring organization. Furthermore, additional low-level radiation minded design techniques are to be used including, but not limited to, low fanout margin, static storage, and static logic. Finally, one or more fabrications shall be made using the MOSIS HP 0.8 μm fabrication facility.

1.4 Overview

This document is organized into seven chapters. This first chapter provides an introduction to the research. Chapter II gives a summary of the currently available VLSI Digital Signal Processing circuits for a wide range of applications, specifically those for asynchronous designs radiation hardened designs. The conclusion of Chapter II displays a chart showing the current DSP application efficiencies with the potential efficiency estimation for the Suter/Stevens FFT architecture.

Chapter III provides a brief background on the three major design areas encompassed by this project. Asynchronous design is defined, compared with synchronous design and several asynchronous design methodologies are discussed. The second section covers radiation effects on micro-electronics including ionizing radiation, single event effects, neutron radiation, and dose-rate effects. For each type, the causes, parameter changes, and mitigating techniques are mentioned. The radiation section concludes with a description of one of the standard cells used in the circuit layouts for this project. The final background section covers the derivation of the Suter/Stevens FFT algorithm from the discrete Fourier transform (DFT) and the base-case FFT computational unit, the FFT-4. The FFT-4 comes from a radix-2 decimation-in-time FFT algorithm.

Chapter IV discusses the high level issues surrounding the radiation tolerant design, the choice of data type, and the parallelism. Then, the generic Suter/Stevens architecture is presented followed by the architecture specific to this research project. Design choices and solutions for each component in the architecture are discussed. In some cases more than one design is covered to demonstrate the flexibility of the architecture.

Chapter V contains the low level implementation of each piece of hardware for an FFT-16. Circuit designs, block diagrams, and state transition diagrams are included to demonstrate how the puzzle pieces interact to complete a working system.

Chapter VI presents the results of the test chip fabricated during the design process. Changes made to the design as a result of the test chip are included showing why the signal transition modifications were necessary. Simulation results of extracted layouts are presented for the FFT-16 at three different Vdd levels. These simulation results are then compared against some of the processors from Chapter II to demonstrate the effectiveness of the Suter/Stevens algorithm and architecture.

Chapter VII wraps up the thesis with major conclusions based on the results from Chapter VI. Some lessons learned during the course of the project are also mentioned with some guidelines for future work on this project.

II. Literature Review

2.1 Introduction

The range and complexity of available digital signal processing (DSP) and fast Fourier transform (FFT) processors is quite broad. A sampling of all different processors is given here because it is likely this project's architecture can compete very well in terms of speed, power, and energy efficiency through a broad spectrum of applications. The chips discussed here include processors created in academia and industry. A table giving general comparison of several FFT processors is provided followed by brief sections describing several processor characteristics. The chip to follow from this project is the Fully Asynchronous Suter/Stevens Transform (FASST). This chapter concludes with a graphical comparison of the processors presented in Table 2.1.

The Figure Of Merit (FOM) used here is the same as in Chapter VI where it will be discussed at length. It should be noted that a *lower* FOM demonstrates better energy efficiency.

2.2 Synchronous Academic FFT Processors

There are several synchronous FFT processors produced at universities across the country. Two published examples include COBRA from the University of Colorado [18], and SPIFFEE from Stanford University [1].

The COBRA chip can perform up to a 64-point FFT on a single chip but can execute a 1024-point FFT in a 16 by 16 chip array with a programmable control unit. COBRA is not designed for low power but Table 2.1 indicates it is very fast in the FFT-1024 computation.

The Spiffee chip, on the other hand, is designed for low power and high speed. Currently there are three versions of the Spiffee processor. Spiffee1 has been fabricated and functions over a Vdd range from 3.3 V to 1.25 V. It is capable of functioning at 1.1 V and 1.0 V when a -0.5 Volt n-well bias is used. Spiffee Low V_t and Spiffee ULP have not yet been fabricated, but their numbers are shown in Table 2.1 to demonstrate their state-of-the-art efficiency. Both of these chips are designed to run at a 0.4 V an Vdd with

Table 2.1 Tabular Comparison of FFT Processor Performance

Processor Name	Dataword Format	Supply Voltage (Volts)	1024 pt exec. time (μs)	Power (mW)	Efficiency $\left(\frac{\text{nJ}}{\text{Unit-Transform}}\right)$
Texas Inst. C40	Float	5.0	1298	4500	5704
L64280 LSI	Float	5.0	26	20000	507
Texas Mem. TM-66	Float	5.0	65	7000	444
Dassault Electronique	Block Float	5.0	128	12000	1500
Array Micro DaSP/PaC/Ras	Block Float	5.0	131	9750	1247
Plessey PDSP16510A	Block Float	5.0	98	3000	287
DSP Arch. DSP-24	Block Float	3.3	21	3500	71.8
COBRA Colorado State	Fixed Point	5.0	9.5	7700	71.4
FASST AFIT	Fixed Point	5.0 3.3 2.2	192 256 425	2580 598 182	483 149 75
Spiffeel Stanford	Fixed Point	3.3 2.5 1.5 1.25 1.1 1.0	30 42 125 252 330 547	845 339 42 15 9.5 5.2	24.7 13.9 5.1 3.7 3.1 2.8
Spiffee Low V_t Stanford	Fixed Point	0.4	93	<9.7	0.880 (projected)
Spiffee ULP Stanford	Fixed Point	0.4	61	8	0.476 (projected)

an estimated power consumption of less than 10mW. Spiffee Low V_t and Spiffee ULP use a separately tuned substrate network to get the threshold voltages of the transistor very near zero. This enables exceptionally low Vdd and high energy efficiency.

Since both of these processors operate on fixed point data, their commercial application is limited. COBRA is additionally limited because it requires a large number of chips to perform the FFT-1024. Spiffee has the advantage that it is a single chip processor. However, its low Vdd and differentially tuned V_t would make wide scale commercial use almost unlikely. Neither of these processors can match the commercial DSP processors in terms of capability.

2.3 Asynchronous Academic DSP/FFT Processors

Previous works have already brought DSP and asnychronous signaling together. However, numbers for throughput, computation time, and power consumption were not given. This makes it difficult to compare them to the other FFT or programmable DSP processors. Like this Suter/Stevens FFT project, the goal for past works has been to develop an architecture well suited to the asynchronous signaling. Each has a different approach that is nothing like that implemented in this project. Despite these differences, they are discussed here to provide some background data.

The oldest work comes from the University of California at Berkeley where a fully asynchronous digital signal processor was built [10]. This has many of the same functional units of the Suter/Stevens architecture but these are arranged in a single, general purpose pipeline instead of local pipelines. The functional units are all self-timed as well, demonstrating the heirarchical composition of an asynchronous system. It was fabricated at 2.0 μm N-well technology and employs dynamic CMOS logic.

The next two designs are both from Flinders University in Australia. The first work highlighted an architecture design for DSP custom applications [5]. It is interesting that the main feature of this architecture is an asynchronous interconnection network that can communicate between multiple functional units. This is exactly the opposite goal of the Suter/Stevens architecture where shared, global devices are to be minimized or eliminated altogether. Timing, rather than power, was the main issue the researchers were trying to solve so the architectural choice may not be completely unacceptable. Another advantage of the design presented is flexibility because the functional units can be modified for specific applications.

The last design from Flinders University is a reconfigurable multi-chip FFT [16]. This is a very limited example because it operates on only 3-bit input data-words and 5-bit output data-words. Despite this, the developers took special care to ensure the algorithm they chose could work well with asynchronous signaling. A slight variation of a radix-2 decimation-in-time algorithm was chosen with an FFT-4 selected for the base case. It is a fully pipelined system employing a dynamic CMOS logic scheme. As in the previous Flinders example, the use of wide feedback busses is not avoided. Again we see a disregard for a low power architecture.

2.4 Commercial DSP Processors

There are a variety of commercial DSP processors available in all dataword formats. Typically the commercially available products are application specific processors which can do many operations related to DSP rather than just FFTs. These additional functions may include digital filtering, matrix multiplication, or multidimensional convolution. This increased capability generally makes it difficult for commercial processors to compete with the energy efficiency of academic processors designed for that purpose.

2.5 DSP Processors for Space

Because this is such a limited area in the DSP realm, it was difficult to attain numbers for comparison to other available systems. One current method for DSP applications is the use of the RAD6000 microprocessor. It is a rad-hard clone (software equivalent) of the IBM R/S6000. Because the RAD6000 is a general purpose machine, there are obvious losses in power and speed to perform DSP applications.

One method of comparison that can be used for the FFTs in space is the Texas Instruments C40 architecture. Texas Instruments is currently developing a radiation hardened version of the C40 architecture for space. The normal C40 architecture is the first entry in Table 2.1. Note that it has the worst energy efficiency characteristics of all the chips compared here. This definitely leaves the door wide open for better designs for space.

2.6 Summary

Figure 2.1 gives a graphical representation of the present technology along with probable areas of application for the Suter/Stevens FFT architecture. In this figure, the Y axis displays the computation time where the increasing axis represents less time. The X axis displays power consumption where the increasing axis represents lower power. The best performance is located in the upper right corner where computation time is the fastest and power consumption is the lowest. The worst performance is located in the lower left corner where computation time is the slowest and power consumption is the highest.

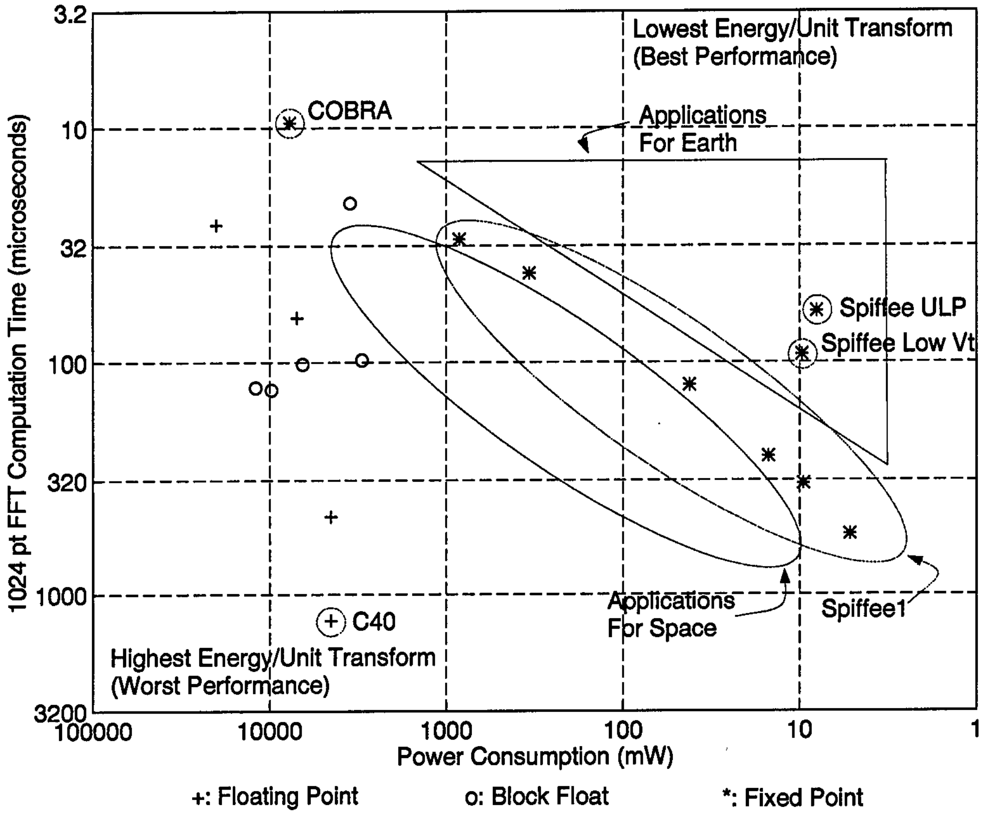


Figure 2.1 Comparison of Existing FFT Processors

Since this is a low power, high performance ***architecture*** and not application specific for low energy design or a specific data type, we should be able to see the benefits of the Suter/Stevens architecture map to fixed point, block float, and floating point applications for both Earth and space.

III. Background

This project combines design principles and techniques from several areas of VLSI design. They include asynchronous timing, radiation tolerance, and digital signal processing (DSP). Each portion is discussed at a high level of detail.

3.1 Asynchronous Design

Synchronous design relies on regular clock pulses that trigger flip-flop circuits to progress through the finite states of a sequential circuit. Conversely, asynchronous design relies on externally and internally generated signals, or handshaking, to progress through the finite states of a sequential circuit. Essentially, asynchronous design is a "self-timed" event based protocol scheme rather than a time based protocol scheme. This section discusses some advantages and disadvantages of each protocol with an overview of the asynchronous methodologies and implementations used in this project.

3.1.1 Synchronous Versus Asynchronous Design. With any design, there are decisions and tradeoffs that need to be made. Many times the decision is clear based on the advantages and disadvantages of each choice. For a long time now, most large scale manufacturers have chosen a synchronous timing protocol over an asynchronous timing protocol for good reason. Clocking a circuit is a good way to eliminate race conditions and other circuit hazards since clock cycles can easily be extended to wait for combinational logic to settle properly. Because early designers favored the ease of synchronous circuitry, that technology matured more quickly than asynchronous circuitry leading to a refined library of CAD tools. Recently, the clock has become the problem for other design issues like power consumption.

The following sections discuss some advantages and disadvantages of the asynchronous timing protocol choice.

3.1.1.1 Asynchronous Design Advantages. Asynchronous design removes the global clock from the circuit resulting in many advantages relating to power and area[1]. First of all, the area consumed by large clock drivers and routing is freed up for other uses. Because CMOS circuits consume most of their power when switching occurs, asynchronous circuits should consume less power because they switch only when useful work is done. A global clock will always be switching, even when the circuit is not productive (i.e., waiting for a memory read).

Asynchronous circuits, when properly designed, can run on average case delay or data dependent delay. The speed of the system is not bound by the worst case clock cycle length so pieces of hardware outside of the critical path do not have to be optimized for speed. They could be optimized for low leakage or low power for other reasons. Additionally, components that lie in the critical path of operation that are slow for certain data dependencies will not dramatically impede the overall circuit performance as long as the "slow data" occurs infrequently.

In a synchronous circuit, the setup and hold times of state machines and logic blocks dictate the clock frequency affecting the overall system speed. As a result, it is important to keep delays through different paths and segments of the circuit fairly constant, even if the different paths receive varied amounts of usage. Conversely, asynchronous circuitry does not require optimization of slower portions of hardware because the next state of the machine waits for a handshake signal after any amount of time. Therefore, rarely used portions of the circuit can be left unoptimized without significantly decreasing the overall performance.

The adaptation to parameter changes is a definite advantage that makes asynchronous systems attractive for this project. As later sections in this chapter show, there are parameter changes that take place in a circuit when it operates in a radiation environment. The asynchronous signaling can be tailored to adapt to slower performance, reduced drive, and other changes associated with radiation effects.

[1]The area advantage is sometimes cancelled by the increased control circuitry required for an asynchronous system.

Asynchronous systems are inherently modular making it easy to combine them into larger systems. As long as the data and control interfaces match, integration of asynchronous systems is easy. There are other asynchronous design advantages but the ones mentioned here are the most applicable to this project.

3.1.1.2 Asynchronous Design Disadvantages.

The difficulty of developing an asynchronous circuit in an ad hoc fashion is the first major disadvantage. To construct synchronous systems, all the designer must do is surround blocks of combinational logic with latches and registers. Concerns for timing are resolved by clocking slow enough for the longest delay in each stage. In asynchronous technology, the designer must pay more attention to the dynamic state of the circuit to ensure there are no race conditions or signal glitches that will put the circuit into an unknown, incorrect, or unresponsive state.

The second major roadblock to asynchronous design is the lack of a well developed set of tools. Since clocked circuitry has been the mainstay of VLSI design for the last two decades, the CAD tools for synchronous design are well developed and very common. Asynchronous design has only recently gained attention in the VLSI domain so the current quantity and completeness of CAD tools for asynchronous design are limited at this point. Obviously, if asynchronous design proliferates, a well developed tool set will follow.

3.1.2 Asynchronous Design Methodology.

Just as there are many design decisions that can be made with synchronous design (i.e., single phase clock, multi-phase clock), there are also many ways to asynchronously implement a sequential circuit. In general, all asynchronous design methodologies replace a global clock with control and/or data handshakes. This means that each progression through the states of the sequential circuit is controlled by a handshake between a sender and a receiver rather than rising or falling clock edges. Despite similarities in handshaking, delay models used in asynchronous systems vary among the different methodologies.

Before we get into detail on each of the design methodologies, a few definitions and acronyms will be covered. the first is an asynchronous finite state machine or AFSM. This is simply an asynchronous implementation of the Mealy state machine model [14] without clocking. The event based method used to advance the states is discussed Section 3.1.2.3. Completion detection refers to the ability of a self-timed system to indicate to a controller that its operation has completed. Sections 3.1.2.1 through 3.1.2.5 outline the methodologies used in this project.

3.1.2.1 Fundamental Mode Bounded-Delay. The simplest and most intuitive way of implementing a circuit asynchronously is the fundamental mode bounded delay. A bounded delay refers to assumptions made from simulations that the delay through a group of logic gates or wires is bounded to some finite period of time. This delay can then be modeled to correctly execute control signaling. Examples of synchronous circuits and their bounded-delay asynchronous counterparts are featured in Figure 3.1. The top two examples show how standard combinational logic in a pipeline segmenat can be converted. In the clocked example, it is assumed the length of the clock cycle is long enough for all the values to become valid before the clock pulses and the values are latched into the next register. In the asynchronous implementation, a delay element greater than or equal to the longest delay of the combinational logic is put in parallel with the component to delay the input request signal to the next register to ensure that the outputs will be valid by the time they are latched. In the bottom two circuits, the latches that control the present/next state bits are replaced again with delay elements that delay the next state signals for a period greater than or equal to the combinational logic. In the general scheme of asynchronous logic, however, this is a poor implementation because the worst case data dependence must be modeled rather than an average case data dependence.

There are some cases where it is difficult to implement completion detection into a functional unit or combinational logic block so some bounded delay assumptions must be made and built into the circuit to ensure correct performance. In general, though, there are better ways to model delays in an asynchronous circuit.

3.1.2.2 Non-fundamental Mode Bounded-Delay. Still using bounded delay assumptions (apart from delay elements) Lee Hollaar developed an application to the bounded-delay model that extended it beyond the fundamental mode using a "one-hot" architecture [9]. Simply viewed, each state is represented by an RS flip-flop. Figure 3.2

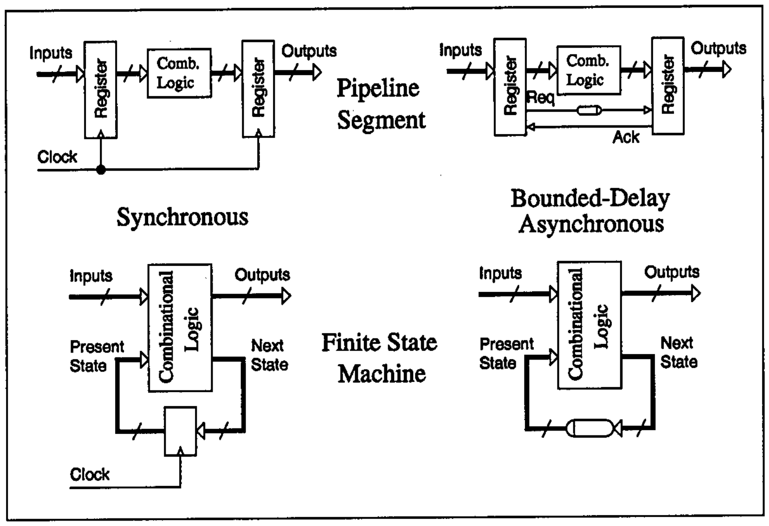


Figure 3.1 Synchronous and Bounded-Delay Asynchronous Examples

contains a simple example of a one-hot sequential state machine segment (left) with the gate structure on the right.

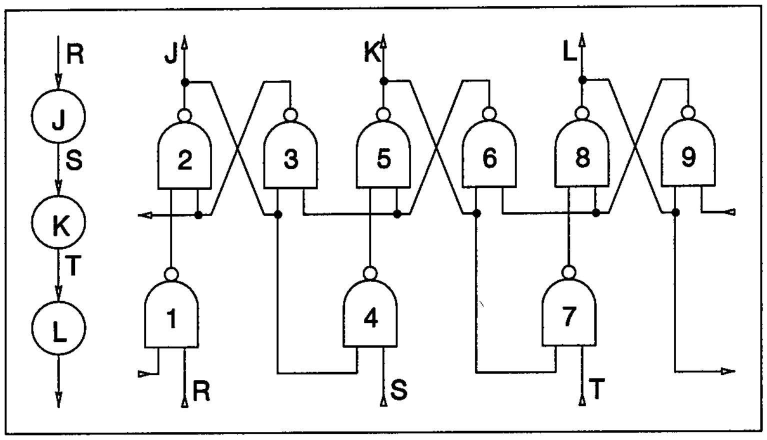
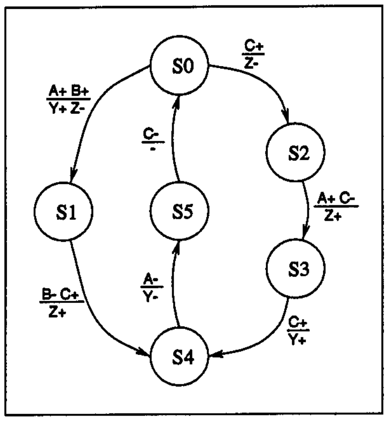


Figure 3.2 One-Hot State Machine Example

Upon entering a state by an external input (R, S, or T), the current state flip-flop is set. The SET line in the RS flip-flop is also the RESET line to the previous state's RS flip-flop. After a period of gate delays, the state will settle making only one of the state bits high (thus the "one-hot" name). This one-hot row assignment is still limited because only one input is allowed to change at a time and proper settling time must be allowed. The concept of the one-hot row assingment was very attractive for the regular flow of the FFT computation but instead of simply using the RS implementation for each state, burst-mode AFSMs (discussed in the next section) are used for each "state."

3.1.2.3 Burst-Mode. The burst-mode design methodolgy is used only for asynchronous control circuitry. Instead of allowing only one input to change at a time, many inputs are allowed to change in any order. When all have changed according to the specification, an output burst is released upon the transition to the next state [3]. Figure 3.3 shows a simple burst mode AFSM with the transition table on the left and the transition diagram on the right.

S_n	S_{n+1}	Input Burst	Output Burst
0	1	A+ B+	Y+ Z-
0	2	C+	Z-
1	4	B- C+	Z+
2	3	A+ C-	Z+
3	4	C+	Y+
4	5	A-	Y-
5	0	C-	-



State Transition Table State Transition Diagram

Figure 3.3 Example Burst-Mode AFSM

The input bursts follow some simple bounded delay rules. First of all, the inputs may arrive in any order and at any time. Second, the state machine must be given adequate time to react and settle from an input burst before receiving another. Third, no input burst may be a subset of another in any given state. This last rule is important so the order of signal arrival will not affect the correctness of the next state. The tools for burst-mode AFSM specification tools are some of the better ones developed for asynchronous circuit design so far. The burst mode designs originated with Coates, Stevens, and Davis [3] who developed the Most Excellent Asynchronous Tool (MEAT) for a project at Hewlett Packard. Others have followed including the 3D tool [22] used extensively for the numerous AFSMs used in this project.

3.1.2.4 Delay Insensitive. The delay insensitive design methodology considers the delays through wires and gates in the circuit to be indefinite. This is different from the bounded-delay models because they assume that signals become valid after a certain settling period.

Because we are concerned with wire delays as well as gate delays, each data line must have built in completion detection. This is done using dual rail data where two lines are used to represent each bit. Table 3.1 shows how each bit is used to transfer data in a return-to-zero (RZ) dual rail data protocol and Figure 3.4 shows how the dual-rail data is generated and interpreted.

Table 3.1 Dual-Rail Data Line Representation

D_0	D_1	Condition
0	0	Data Invalid
0	1	Logic High
1	0	Logic Low
1	1	Error

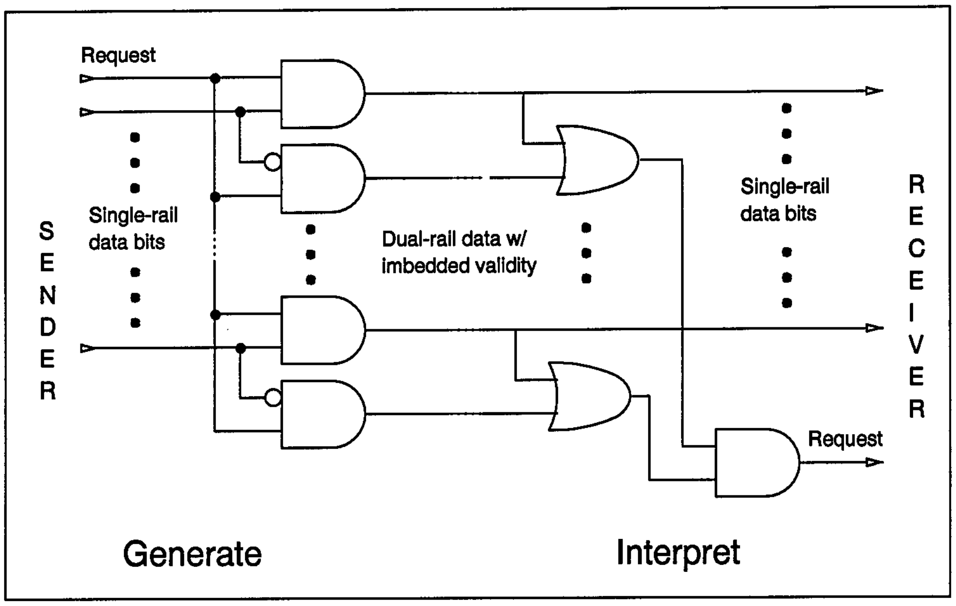


Figure 3.4 Dual-Rail Data Generation and Intpretation

A non-return-to-zero (NRZ) dual rail scheme may also be used which would simply detect a change on either line to indicate a one or zero. Though this scheme is faster in terms of transfer because there are fewer transitions, the time savings is easily lost in control logic complexity and area [7].

Requiring two lines for each data bit results in a huge area cost, especially in a 16- or 32-bit architecture. There is additional area overhead because many times, a single request signal must be derived from several dual-rail data lines. The gates necessary for this consume additional area. Because of these area concerns, the actual occurence of true delay-insensitive circuits is very limited.

3.1.2.5 Speed Independent. The speed independent timing model is similar to the delay insensitive design except delays in wires are ignored. Essentially, we make some assumptions about delays as in the bounded-delay models. This project primarily uses a speed-independent, four-phase data-bundling timing model. The handshaking and data transfer will be discussed in the next couple of paragraphs.

Data bundling handles valid and invalid data differently than the dual rail data. Figure 3.5 shows the request-acknowledge handshaking between a sender and a receiver. Note the sender's output data becomes valid causing REQ to go high (1). Once the receiver gets the input data, the ACK signal is raised (2). When the sender senses the assertion of the ACK signal, it lowers REQ (3) and now processes the next data set, waiting for ACK to lower once again. When the receiver lowers ACK (4) the sender knows it may again begin the handshake protocol when the new data is ready.

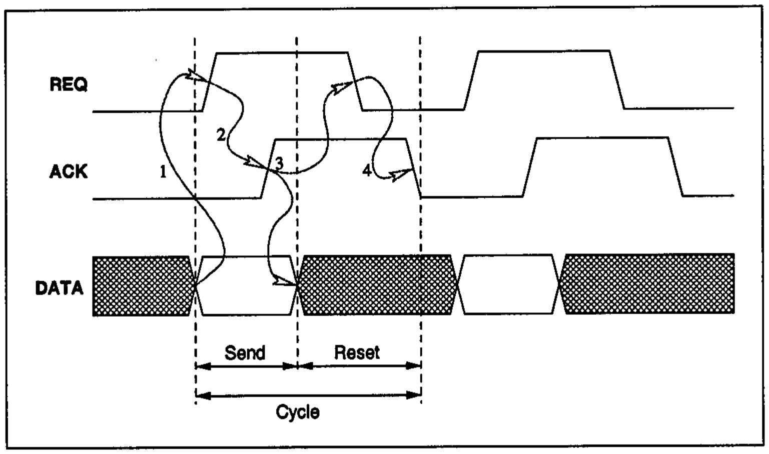


Figure 3.5 Bundled Data

This is known as four-phase data bundling because for each data transfer segment, the REQ-ACK handshake goes through four phases (10, 11, 01, 00). A two-phase handshaking method also exists for the data bundling but, like the dual-rail data, it is a bit more complex than the four phase handshaking because the hardware must be able to detect high-low transitions and low-high transitions.

3.2 Radiation Effects on Micro-electronics and Other Issues

This section covers some of the prevailing radiation effects on electronics. The first two sections, 3.2.1 and 3.2.2, discuss radiation events that occur mainly in space. The last two sections, 3.2.3 and 3.2.4, discuss the radiation effects caused mainly by nuclear weapons. Each part contains the source, effects, parameter changes, and countermeasures used for each form of harmful radiation. Following this, the cell from the library to be used in this project is examined to demonstrate its radiation tolerant[2] features.

3.2.1 Ionizing Radiation. Gamma rays, x-rays, electrons, protons, and heavy ions are all causes of ionizing radiation which primarily affects the oxide layers of a CMOS circuit. Upon irradiation, electron-hole pairs are generated and evenly distributed throughout the SiO_2 layer. Many of these pairs recombine within 100μsec, but some free electrons are swept out by the electric field in the gate insulator. The trapped holes that remain in the insulator cause a negative shift in the MOSFET threshold voltage. Over time, the holes slowly migrate toward the most negative potential within the SiO_2. If this most negative potential is the channel (N-FETs), the holes will tend to migrate toward the insulator-channel interface *decreasing* V_t from its pre-rad or initial value. If this most negative potential is the gate(P-FETs), then the holes migrate toward the insulator-gate interface *decreasing* V_t from the initial value. After a period of time, the holes are annealed out of the SiO_2, allowing V_t to shift back toward its initial value [11,15].

The key parameter changes caused by ionizing radiation is the threshold voltage shift. For P-FETS, V_t is shifted negatively at all dose levels because the trapped holes in

[2]A radiation *tolerant* circuit can withstand ~100 krad(Si) and still maintain proper function. A *rad* is a unit of absorbed dose equal to 0.01 Joule absorbed per kilogram of any material.

the oxide and the interface states work together. For N-FETS, V_t is shifted negatively at low dose levels and initially at high dose levels [11]. At high total dose levels, the N-FET V_t may eventually shift positively as interface states begin to dominate the trapped holes.

A thin gate oxide can mitigate the effects of ionizing radiation by allowing fewer electron-hole pairs to be formed and reducing the hole lifetime in the oxide. A reentrant form of the N-FET prevents the field oxide from interacting with the gate of the transistor, thus ensuring the trapped charges in the field oxide will not turn the N-FET on. Figure 3.6 shows the standard N-FET. There are two edges of the channel exposed to the bulk substrate where trapped charges in the field oxide can interfere with the threshold voltage. The reentrant N-FET (also in Figure 3.6) has no edges exposed to the bulk substrate, preventing field oxide trapped charge interference.

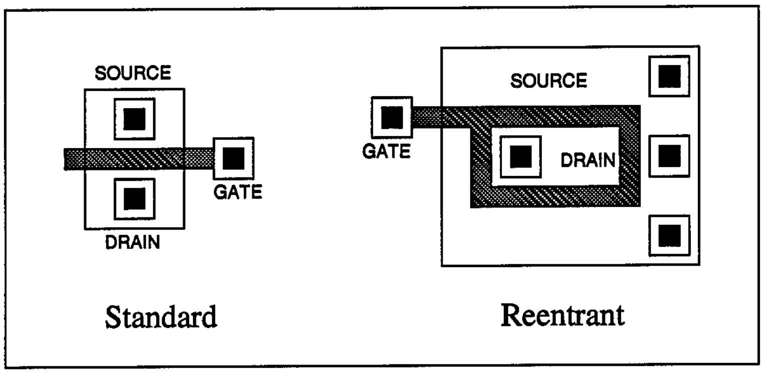


Figure 3.6 N-FET Designs

3.2.2 Single Event Effects (SEE). Heavy ions, protons, neutrons, and gamma rays are all known to cause SEE. In space, heavy ions and protons are the most prevalent causes. SEE can be divided into two classes of errors.

The more severe hard errors for CMOS devices include gate rupture and latch-up. Gate rupture occurs when a single ion passes through the gate oxide layer. Gate rupture can only happen when there is a high electric field in the oxide. A gate rupture causes heating of the dielectric and possible thermal runaway. Latch-up is the product of parasitic bipolar-junction transistors forming between opposite diffusion regions, the substrate, and one of the wells. When a trigger voltage is reached (around -0.7 volts), the node will "latch" at a holding voltage and remain there drawing large amounts of current, regardless of the inputs or outputs of the node.

To mitigate the hard errors, a variety of steps must be taken. For instance, the best way to counter the latch-up condition is to fabricate the circuit on an insulator to remove the parasitic pnpn configuration where latch-up transistors could form. If a silicon-on-insulator (SOI) or silicon on saphire (SOS) solution is not feasible due to cost, gate count, etc., oppositely diffused material around the n+ and p+ regions (called guard rings) also helps reduce the possibility of a latch-up condition.

The less severe soft errors, or single event upset (SEU), include register or memory bit upsets, and transient logic upsets. Each of these topics can be quite in-depth so this section will just touch on some of the ways in which SEU can be attenuated. First of all, fully static logic is the easiest way to reduce the impact of SEU, especially the soft errors. Any kind of dynamic or pre-charge logic could fall as easy prey to a transient value since the evaluation period allows circuit nodes to "float." To harden against memory or register bit upsets, fully static storage is frequently used.

These are the main SEE issues relating to this project. Sources such as [15] give a much more complete exploration of single event effects and mitigation techniques.

3.2.3 Neutron Radiation. Neutron radiation is caused primarily by particles released from a nuclear fission or fusion reaction. The physical main effect of neutron radiation in MOSFETs is that the damage to the silicon crystal structure. The neutrons are a heavy subatomic particle releasing much of their energy upon impact. The defects that these neutrons create become traps for the majority carriers.

Channel resistance for both N-FETS and P-FETS increases along with the polysilicon trace resistance. The increased resistance is a result of the removal of majority carriers leaving a larger percentage of minority carriers. This will lower the gate voltage causing the threshold voltage, V_t, to move closer to zero. This will make the transistor easier to turn on (more leakage current) but with reduced drive capability because there are fewer majority carriers and more leakage current due to the dislocation damage at the SiO_2 interface.

The drive capability of the transistors is further hindered by the increased resistance of the polysilicon traces.

As an example, transistors that are normally off will allow additional leakage current to flow turning them slightly on. The transistors that are on have to deal with the additional channel resistance creating a voltage divider turning them slightly off. To counter these changes, a smaller fanout margin and larger transistor widths should be used to pass signals better when a transistor is on. The lowered V_t and increased leakage current may require larger Vdd and GND rails to supply the necessary current.

3.2.4 Dose-Rate Effects. As the name implies, the dose-rate effects apply to a circuit that dwells in a short-lived hostile environment. The main effects of dose-rate radiation are photocurrent generations in the pn junctions, trapped-hole generation followed by trapped-hole annealing, latch-up, and upset. The photocurrents are generated by the creation of electron-hole pairs in a pn junction (this pn junction can be between the source, drain, or bulk material). The trapped-hole generation and annealing in the oxide layers is very similar to that of the ionizing radiation (see Section 3.2.1 above).

A circuit designed to resist the effects of dose-rate radiation should include additional power supply current to sink the photo currents, and latch-up prevention measures. The additional power supply current could easily be built in by making sure the voltage supply can generate more current than necessary in a pre-rad condition.

3.2.5 The Radiation Tolerant Cell Library. For this project, a cell library containing many of the radiation countermeasures mentioned in Section 3.2.3 through Section 3.2.2 was used. This radiation tolerant cell library was developed jointly by The Air Force Research Laboratory and Mission Research Corporation [12]. The library contains only fully static logic. For multiplexors, registers, flip-flops, and latches, this is unique because most applications of each involve pass-transistor logic. For both the N-FETS and P-FETS, large widths are used to ensure low channel resistance and large drive capability. Reentrant N-FET transistors are used to prevent the field oxide that may easily ionize from affecting the transistor V_t. To prevent latch-up, n-type ohmic rings surround p+ diffusion regions and p-type ohmic rings surround n+ diffusion regions further turning off the potential parasitic pnpn transistors. Wide supply and ground rails coupled with frequent substrate and diffusion contacts ensure good current supply and even potential across the bulk substrate, n-wells, or the large diffusion regions.

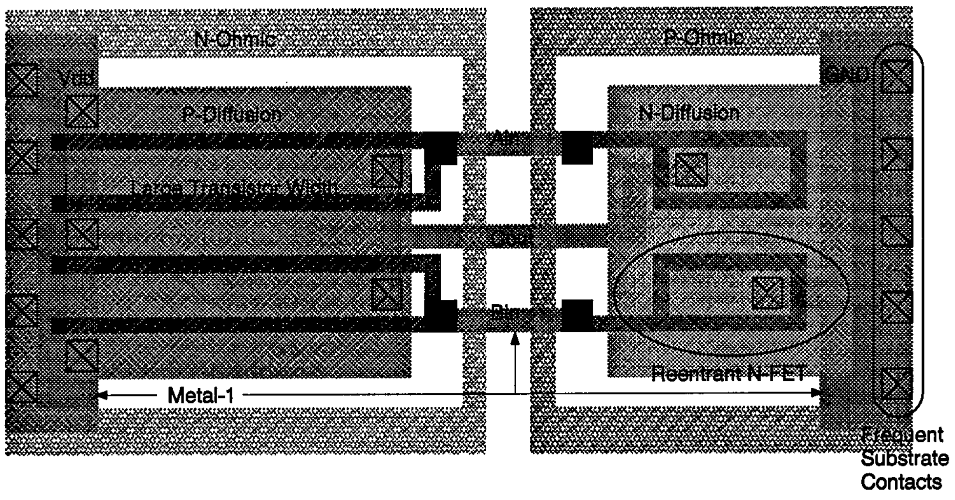


Figure 3.7 AFRL/MRC 2 Input NAND gate

3.2.6 Summary of Radiation Effects. Some important walk-away knowledge from this radiation effects section indicates there is no perfect solution for putting electronics into space. Designers can use preventive measures at all different cost levels. Because power, area, and speed are all important issues, these designers attempt to get the most protection with the least amount of development and fabrication costs while maintaining operation within other constraints. Inevitably, trade-offs among radiation hardened[3] or tolerant characteristics, development time, and other VLSI design issues will always be driving factors in the production of space electronics.

3.3 Digital Signal Processing (DSP)

DSP is a major application of VLSI circuits. The Fourier transform is a very important DSP function due to its wide range of applications throughout engineering and

[3]A radiation ***hardened*** circuit can withstand ~1Mrad(Si) and still maintain proper function. This is generally achieved through a hardened fabrication process.

physical sciences. This section covers the equations and algorithms behind the DSP architecture used in this project.

3.3.1 The Discrete Fourier Transform (DFT). The most common form of the Fourier transform is the DFT commonly represented by Equation 3.1

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nm}{N}} \tag{3.1}$$

where $x(n)$ consists of N samples of a finite sequence and $X(m)$ consists of N frequency components of $x(n)$'s Fourier transform. Note that there are N complex multiplies ($4N$ real multiplies) per frequency evaluation. The number of multiplies needed to calculate the DFT for N frequencies is $N \times (4N) = 4N^2$ which is $O(N^2)$ operations. Equation 3.1 can be rewritten using the substitution $W_N = e^{-j\frac{2\pi}{N}}$ giving

$$X(m) = \sum_{n=0}^{N-1} x(n) W_N^{nm} \tag{3.2}$$

There are two properties of W_N that enable the calculation of the DFT in less than $O(N^2)$ operations. The relationships

$$W_N^{m(N-n)} = W_N^{-mn} = (W_N^{mn})^*$$

and

$$W_N^{(m+N)n} = W_N^{mn} = W_N^{m(N+n)}$$

demonstrate the complex conjugate[4] symmetry of W_N and that W_N is periodic in both n and m respectively. Clever algorithms use these two properties to "divide" the original DFT problem into smaller and smaller problems until a base case is reached. Usually this base case is small enough so it can be easily "conquered" with a butterfly or some other simple arithmetic operation. These divide and conquer algorithms reduce the running time of a DFT computation from $O(N^2)$ to $O(Nlog_2N)$ making it a ***Fast*** Fourier Transform (FFT).

[4] $(A + jB)^* = (A - jB)$

FFT research was largely pioneered by J. W. Cooley and J. W. Tukey in a paper they published in 1965 [4] but the FFT was probably first proposed by the mathematician Gauss in 1805 [8]. Since 1965, many researchers have accomplished a great deal in developing new algorithms for various purposes including prime or near prime length, incomplete sequences, sequences not of mod 2 length, etc [13].

3.3.2 The Suter/Stevens FFT Algorithm. A wavelet approach was applied by Suter and Stevens [21] to create a hardware realization that parallelizes and localizes the computations of an FFT in such a way as to reduce the overall power consumption. We can represent N from Equation 3.1 as N_1N_2. Then, by the division theorem for integers, let $m = m_2N_1 + m_1$ and $n = n_1N_2 + n_2$ where $m_1, n_1 = 0, 1, \cdots, N_1 - 1$ and $m_2, n_2 = 0, 1, \cdots, N_2 - 1$. When given a sequence, $x(n)$, its polyphase components [19] are defined as $x_k(Mn + k)$ where $k = 0, 1, \ldots, M - 1$. Now, the original N point FFT problem can be divided into equivalence classes where $X_{m_1}(m_2) = X(m_2N_1 + m_1)$ and $x_{n_2}(n_1) = x(n_1N_2 + n_2)$. Using this polyphase notation of the FFT enables the DFT from Equation 3.1 to be written as

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) e^{-j\frac{2\pi(m_2N_1+m_1)(n_1N_2+n_2)}{N}} \tag{3.3}$$

With the exponential numerator multiplied out, the result becomes

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) e^{-j\frac{2\pi m_2N_1n_1N_2}{N}} e^{-j\frac{2\pi m_2N_1n_2}{N}} e^{-j\frac{2\pi m_1n_1N_2}{N}} e^{-j\frac{2\pi m_1n_2}{N}} \tag{3.4}$$

where the first exponential is equal to unity. Further simplification leads to

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \left[e^{-j\frac{2\pi m_1n_2}{N}} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) e^{-j\frac{2\pi m_1n_1}{N_1}} \right] e^{-j\frac{2\pi m_2n_2}{N_2}} \tag{3.5}$$

or

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \left[W_N^{m_1n_2} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) W_{N_1}^{m_1n_1} \right] W_{N_2}^{m_2n_2} \tag{3.6}$$

in W_N notation. The motivation behind manipulating the FFT into the form of Equation 3.6, and how this maps to a low power implementation will be more clear following a discussion of the generic architecture in Section 4.2.

This algorithm can be implemented hierarchically, allowing the N_1 and N_2 blocks to be a smaller instantiation of the algorithm. For instance, if $N = 1024$ where $N_1 = 16$, and $N_2 = 64$, the N_1 and N_2 blocks can each be decomposed into $N_{11} = N_{12} = 4$ and $N_{21} = N_{22} = 8$. The Derivation of the FFT-4 in Section 3.3.3 demonstrates that four-point FFTs can be implemented without multiplication, so a hierarchical decomposition which maps the leaf nodes to four-point FFTs is the most efficient realization in terms of area and power. Working in the opposite direction, it is easy to see that FFT point sizes that are an *even* power of the base FFT point size work best.

3.3.3 The Base Case FFT. Using a divide and conquer algorithm like the one above, eventually a base case is reached that cannot be divided further. Presented here is the derivation of the FFT-4 from the DFT as it is implemented in this project. This is a standard decimation-in-time algorithm with a minor modification to eliminate the need for a complex-valued multiply.

Using the relations $X(m) = Re\{X(m)\}+jIm\{X(m)\}$, $x(n) = Re\{x(n)\}+jIm\{x(n)\}$, and $j \times (Re\{x\} + jIm\{x\}) = -Im\{x\} + jRe\{x\}$, the complex DFT is written as

$$\mathbf{X}(m) = \mathbf{W_4}\mathbf{x}(n) \tag{3.7}$$

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \tag{3.8}$$

in matrix form where $\mathbf{X}(m)$ represents the output sequence, $\mathbf{W_4}$ is the DFT matrix, and $\mathbf{x}(n)$ is the input sequence. Performing the matrix multiplication, Equation 3.8 can be written as

$$X(0) = x(0)W_4^0 + x(1)W_4^0 + x(2)W_4^0 + x(3)W_4^0 \tag{3.9}$$

$$
\begin{aligned}
X(1) &= x(0)W_4^0 + x(1)W_4^1 + x(2)W_4^2 + x(3)W_4^3 && (3.10)\\
X(2) &= x(0)W_4^0 + x(1)W_4^2 + x(2)W_4^4 + x(3)W_4^6 && (3.11)\\
X(3) &= x(0)W_4^0 + x(1)W_4^3 + x(2)W_4^6 + x(3)W_4^9 && (3.12)
\end{aligned}
$$

From Section 3.3.1 we recognize by symmetry that

$$
\begin{aligned}
W_4^0 &= -W_4^2\\
W_4^1 &= -W_4^3
\end{aligned}
$$

and by periodicity that

$$
\begin{aligned}
W_4^0 &= W_4^4\\
W_4^1 &= W_4^9\\
W_4^2 &= W_4^6 = -W_4^0
\end{aligned}
$$

Expressing all W_4^n in Equations 3.9 through 3.12 with W_4^0 and W_4^1 where $W_4^0 = 1$ and $W_4^1 = -j$ renders

$$
\begin{aligned}
X(0) &= x(0) + x(1) + x(2) + x(3) && (3.13)\\
X(1) &= x(0) - jx(1) - x(2) + jx(3) && (3.14)\\
X(2) &= x(0) - x(1) + x(2) - x(3) && (3.15)\\
X(3) &= x(0) + jx(1) - x(2) - jx(3) && (3.16)
\end{aligned}
$$

If we let $a = x(0) + x(2)$, $b = x(1) + x(3)$, $c = x(0) - x(2)$, and $d = x(1) - x(3)$, where a, b, c, and d are complex variables, we can factor the j's and substitute the intermediate values into Equations 3.13 through 3.16 giving

$$
\begin{aligned}
X(0) &= a + b && (3.17)\\
X(1) &= c - jd && (3.18)\\
X(2) &= a - b && (3.19)
\end{aligned}
$$

$$X(3) \quad = \quad c + jd \tag{3.20}$$

This simplified computation is easily visualized in the signal flow graph of Figure 3.8.

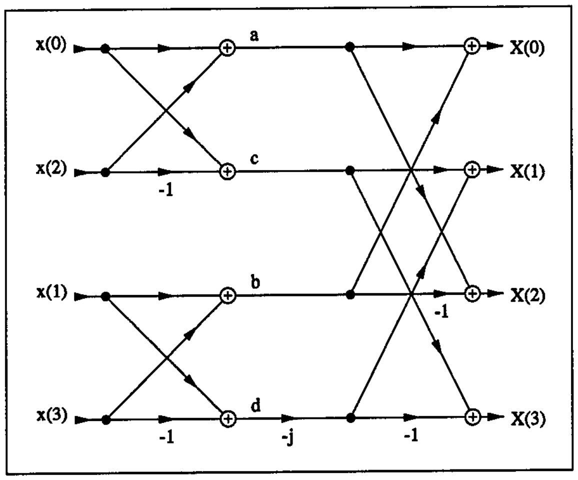


Figure 3.8 Four Point FFT Signal Flow Graph

Ordinarily, the $-j$ multiplier on the d path in the signal flow graph would require a complex-valued multiply. In the realities of the computation, however, the real and imaginary components of the intermediate values a, b, c, and d are kept separate so Equations 3.13 through 3.16 become

$$X(0) \quad = \quad (a_R + ja_I)(1) + (b_R + jb_I)(1) \tag{3.21}$$
$$X(1) \quad = \quad (c_R + jc_I)(1) + (d_R + jd_I)(-j) \tag{3.22}$$
$$X(2) \quad = \quad (a_R + ja_I)(1) - (b_R + jb_I)(1) \tag{3.23}$$
$$X(3) \quad = \quad (c_R + jc_I)(1) - (d_R + jd_I)(-j) \tag{3.24}$$

Distributing the $-j$ term and grouping the real and imaginary components together renders

$$X(0) \quad = \quad (a_R + b_R) + j(a_I + b_I) \tag{3.25}$$
$$X(1) \quad = \quad (c_R + d_I) + j(c_I - d_R) \tag{3.26}$$
$$X(2) \quad = \quad (a_R - b_R) + j(a_I - b_I) \tag{3.27}$$
$$X(3) \quad = \quad (c_R - d_I) + j(c_I + d_R) \tag{3.28}$$

The signal flow graph in Figure 3.9 represents the separated real and complex computations of Equations 3.25 through 3.28.

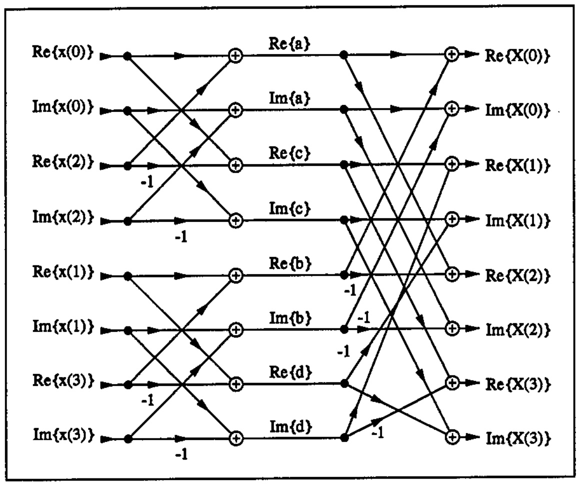


Figure 3.9 Four Point Complex FFT Signal Flow Graph

Using this rearrangement of values, only 16 addition or subtraction operations are required with no multiplication making this ideal for the base case.

3.4 Summary

As you can see, this project has married three deep topics, all of which could not be exhausted by years of research. It is recommended that the references cited in this chapter are explored for more detailed information on asynchronous design, radiation effects on micro-electronics, and the fast Fourier transform.

IV. Design

The purpose of this research is to demonstrate the feasibility of implementing an FFT algorithm in a micro-electronic circuit using VLSI. This chapter begins by discussing many of the high level design issues that surround this project. Following these issues, the generic Suter/Stevens architecture is discussed along with the design decisions regarding each of the major components.

4.1 High Level Design Issues

This section covers many of the higher level design issues that affect other lower level design issues in the actual implementation.

4.1.1 Designing Low Power Space Applications.

Solar panels and nuclear generators are the only way a satellite can acquire energy. Therefore both peak and standby power must be kept to a minimum.

A common method of reducing power consumption in integrated circuits is to lower the supply voltage, yielding a quadratic improvement in power at a linear cost in performance when the transitors are operating in the saturated region. However, scaling the voltage of a CMOS circuit allows it to become more susceptible to SEU because the noise margin between a logic high and a logic low is reduced. SEU possibilities are further acerbated due to the threshold shifts that occur under radiation. Therefore, voltage scaling must be used judiciously and in general has more restrictions than in Earth-bound electronics.

The power and complexity required to implement many CMOS functions can be reduced using circuit techniques such as dynamic, pre-charge, and pass-gate logic. Unfortunately these techniques are also to be avoided since the single event effects can prey very easily on these structures. Design is largely limited to static logic gates.

Fortunately standard CMOS processes can be used while still achieving radiation tolerance. The AFRL/MRC cell library discussed in Section 3.2.5 is specifically designed for the MOSIS Hewlett Packard 0.8μm fabrication. An unfortunate side effect of this convenience is that the radiation requirements result in devices much larger than would be used otherwise, consuming more power. For example, the minimum size inverter width[5] in the AFRL/MRC cell library is 50λ for the N-FET and 90λ for the P-FET! Typically the minimum sized transistor for both N and P-FETs is 3λ to 5λ. With the additional rad tolerant characteristics, the total size of the inverter cell is $42\lambda \times 119\lambda$.

Architecture becomes the primary means of reducing power in space applications, due to constraints to voltage scaling, circuit structure, and device size. This FFT architecture implemented using asynchronous circuits significantly reduces the power compared to other space worthy designs. The most significant contributions to low power in this architecture are twofold. First, the algorithm has been designed to maximize locality, point-to-point data pipelining, and hierarchy. The only shared structures in the design is the decimator inputs, expander outputs, and pipelined crossbar switch (all discussed in Section 4.2). Second, the operating frequency is greatly reduced by decimation allowing devices and drivers to be undersized. This can significantly reduce the capacitance of transmitting data signals. The asynchronous implementation technology allows the common frequency changes to be supported at minimal energy dissipation.

4.1.2 Data Type.

The data type used in this FFT implementation is 32-bit fixed point with 16 bits used to represent the real portion of each sample and 16 bits to represent the imaginary portion of each sample. Figure 4.1 shows how each of the 16-bit values are broken up.

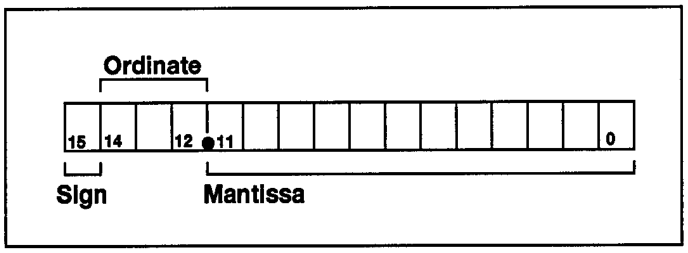


Figure 4.1 16-Bit Fixed Point Representation

[5]λ notation is used because CMOS is scaleable. λ is equal to $\frac{1}{2}$ the smallest feature size of a given CMOS fabrication technology. In this case, $\lambda = 0.4\mu$m because 0.8μm technology is used.

The sign bit is the most significant bit, the ordinate ($\geq |1|$) occupies the next three bits and the mantissa ($< |1|$) occupies the remaining twelve bits. The signed two's complement 16-bit fixed point data works just like signed two's complement integers so the range of data representation is $1000.0000\,0000\,0000_2$ ($-8.000\,000\,000\,000_{10}$) to $0111.1111\,1111\,1111_2$ ($7.999\,755\,859\,375_{10}$).

Although we need to worry about overflow with addition and subtraction, we must also be concerned with the error introduced by the complex-valued multiplier using fixed point representation. Keep in mind that the multiply operation in this application can only return a magnitude less than or equal to the input. Referring to Equation 3.6, the constant multipliers matrix is characterized by $W_N^{m_1 n_2}$. For this project (as we will see in Section 4.3) $N = 16$, and $m_1, n_2 = 0, 1, 2, 3$ rendering,

$$W_{16}^{m_1 n_2} = \begin{bmatrix} W_{16}^0 & W_{16}^0 & W_{16}^0 & W_{16}^0 \\ W_{16}^0 & W_{16}^1 & W_{16}^2 & W_{16}^3 \\ W_{16}^0 & W_{16}^2 & W_{16}^4 & W_{16}^6 \\ W_{16}^0 & W_{16}^3 & W_{16}^6 & W_{16}^9 \end{bmatrix} \tag{4.1}$$

Using $W_N = e^{-j\frac{2\pi}{N}}$ we can determine the approximate decimal value of each matrix element resulting in

$$W_{16}^{m_1 n_2} = \begin{bmatrix} 1.0000 & 1.0000 & 1.0000 & 1.0000 \\ 1.0000 & 0.9239 - j0.3827 & 0.7071 - j0.7071 & 0.3827 - j0.9239 \\ 1.0000 & 0.7071 - j0.7071 & 0.0000 - j1.0000 & -0.7071 - j0.7071 \\ 1.0000 & 0.3827 - j0.9239 & -0.7071 - j0.7071 & -0.9239 + j0.3827 \end{bmatrix} \tag{4.2}$$

Even using decimal representation, error would still be introduced because some of the matrix elements are irrational. However, the introduced error is worse for the binary representation used in this project. Assuming we have a binary number and a decimal number, each with the same number of digits, the binary number will only be able to represent $log_2(n)$ of the magnitude of a decimal number. Table 4.1 shows how each binary representation stacks up to decimal equivalents.

Table 4.1 Constant Representation

Decimal Value (15 places)	Binary Representation	Decimal Conversion	Resultant error (%)
0.38268343236509	0000.0110 0001 1111	0.382568359375	-0.03007
0.70710678118655	0000.1011 0101 0000	0.70703125	-0.01974
0.92387953251129	0000.1110 1100 1000	0.923828125	-0.00556

Table 4.1 shows the error introduced by the 16-bit fixed point representation is minimal using a 12-bit mantissa. Without normalization after each sub-FFT computation, the ordinate magnitude increases linearly with each FFT point size increment. Assuming, for example, that the input data can fit within a 3-bit ordinate ($-8 \leq n \leq 7.999\ldots$), the output ordinate can have a maximum size of that listed in the right column of Table 4.2 for each point size given. The last row (an FFT-1024) indicates only two bits can be used for the mantissa. Two mantissa bits can only be accurate to 0.25_{10}!

Table 4.2 Ordinate Requirements for Fixed Point Data

FFT Point Size	Maximum Output Ordinate
4	5 bits
16	7 bits
64	9 bits
256	11 bits
1024	13 bits

Fixed point representation has limitations for larger FFT point sizes unless the input magnitude is limited to $< |1|$. Otherwise, a floating point or a hybrid block floating point data type (normalization after each sub-FFT computation) should be used to ensure maximal accuracy.

4.1.3 Parallelism Versus Control Path. The tradeoff between the amount of architectural parallelism and the size of the control path became an important high level design consideration during the course of this project. A higher degree of parallelism leads to faster computation, higher throughput, and less control circuitry at the expense of more die area. A lower degree of parallelism (more component reuse) reduces die area but increases computation time, reduces throughput, and increases the amount and complexity of control circuitry. Since a working circuit that fits on a reasonably sized chip is desired, it is important to find the "middle ground" to get a circuit working with only a fair amount of work and still fit onto a chip.

4.2 Generic Architecture

There are six major components required to implement the FFT of Equation 3.6. The block diagram of Figure 4.2 shows how each of the components fit into the computation. First of all, the data is decimated in time into N_2 sequences of length N_1. Then, the FFT of each sequence is computed (the interior summation), followed by the multiplication of the constants ($W_N^{m_1 n_2}$). The complex-valued multiply execution time is critical for the overall N- point FFT performance since $N_1 \times (N_2 - 1)$ operations are required. After the complex multiply, the partial transformed data is interleaved, as required by the FFT, through a pipelined crossbar switch. This pipes a data stream to the N_2-point FFT blocks. Finally the data, which is decimated in frequency, goes to an expander to correctly sequence each fully transformed element in the output sequence as $X(m)$ and regenerate the input frequency.

4.3 Specific Architecture for This Project

The focus of this design is to demonstrate the functionality of the algorithm with the minimum iteration. Therefore $N = 16$ was chosen with $N_1 = N_2 = 4$. This allowed the basic architecture to reuse the N_1 and N_2 blocks, drastically saving on area with only a minimum of control overhead.

Control operates in data-flow pipelines, using data bundling and four-phase handshaking protocol between each stage. Control of every major component is implemented using one-hot encoded state machines [9]. There are 16 unique burst-mode asynchronous finite state machines (AFSMs) used in the control structures. Some of the AFSMs are very simple like the ones used to control register locking which have 3 states with 2 inputs and 2 outputs. Others are more complex like the multiplier controller which has 9 states with 8 inputs and 6 outputs. Sections 4.3.1 to 4.3.5 discuss the design choices made for each part of the architecture.

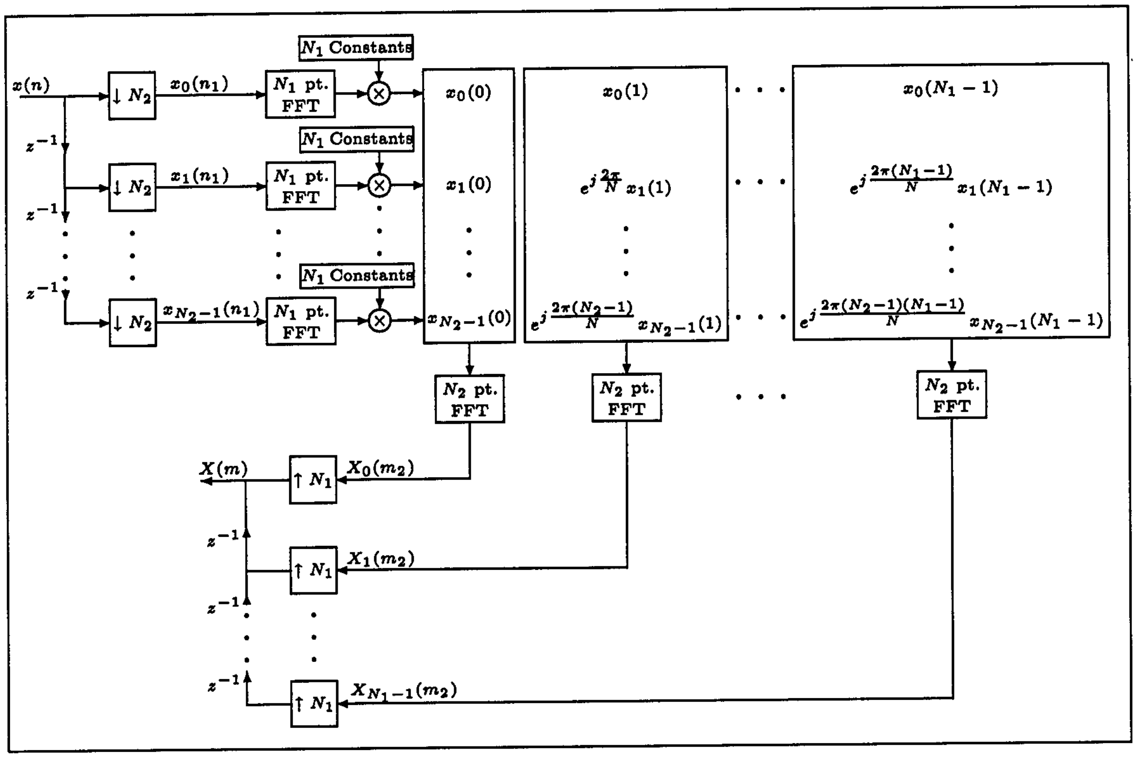


Figure 4.2 Generic Block Diagram of the Suter/Stevens FFT Algorithm

4.3.1 Decimator. The purpose of the decimator is to break down the N input values into N_2, N_1 value sets. In this case, where $N = 16$ and $N_1 = N_2 = 4$, the first sequence includes x_0, x_4, x_8, and x_{12}, the second sequence includes x_1, x_5, x_9, and x_{13}, and so on.

Designs Considered. In the synchronous domain, decimation is a bit more complex than in the asynchronous domain. Instead of having to insert 0s between every nth sample, we simply take only n samples and ignore the rest of the sequence. For this project, three different ways to asynchronously decimate a sequence were examined.

The first is to use a brute-force state machine that will have one request line in and N_2 request lines out. As a request comes in, the next request out line is raised until the cycle repeats. It was found after quick attempts to implement this specification that the MEAT [3] and 3D [22] tools produced unwieldy Boolean equations that were to large to implement efficiently in CMOS.

Another approach is to use a global one-hot sequence. In a global one-hot sequence, the request in is routed to the N_2 one-hot machines and each one raises its local request signal at the appropriate time (more will follow on this in Section 5.2.1). The global one-hot approach has the advantage that only one machine delay lies between the global request and the local request.

The last approach pondered for this design is a decimator tree. Instead of specifying a decimate-by-N_2, we specified a decimate-by-n where $n^h = N_2$ and h is a positive integer. Then we formed a tree of the n blocks to compose the full size decimator. There are two drawbacks of the decimator tree. First of all, $2^{N_2} - 1$ decimator units are required. For long input sequences, this is a genuine area concern. Secondly, as the height of the decimator tree grows, the delay from a request at the root will experience significant latency ($\lceil log_h N \rceil$) as it propagates to the proper leaf node.

There are at least two types of decimator trees that can be used. One would contain all the request and acknowledge handshaking where the farthest leaf of the tree has to acknowledge before it trickles back up the tree so the root decimator can acknowledge back to the sender. The other implementation, which is potentially more efficient, uses an OR structure to fan-in all of the leaf acknowledge signals and the decimator tree is only concerned about the request signals being raised in the proper order.

Chosen Design. The design used in the VHDL FFT-16 simulation is a decimator tree with external acknowledgment. Since the input sample handshake can only happen one value at a time, the acknowledgments are all mutually exclusive so there is no hazard potential by ORing all N_2 acknowledge signals together.

4.3.2 FFT-4. Intuitively, the FFT-4 element takes the 4 point FFT of each decimated sequence. There are four of these blocks for the 16-point FFT assuming they are used for both the N_1 and N_2 as in this project.

Designs Considered. The FFT-4 seems to be the architectural component with the most design flexibility. The initial design direction was to minimize the number of registers and ALUs to save on area and because the FFT-4 was not the bottle neck of the data path. Schemes using eight 16-bit registers with two or four ALUs were examined initially. Using the burst mode AFSM synthesis tools, it was soon clear the asynchronous control units performing the necessary operations for each of these schemes were complex and difficult to implement in silicon. Attempts were made to divide up the control as much as possible but then integrating the divided state machines became a difficult task. Before too long, it was discovered this approach was counterproductive and a new design was pursued.

Chosen Design. For simplicity and ease of implementation, a modified data-flow design was used for FFT-4 requiring no register reuse. There are eight 16-bit input registers, eight 16-bit intermediate registers, and one ALU. The FFT-4 control is divided into three sections. The input stage is responsible for reading in the four 32-bit input words, placing them in the eight 16-bit input registers. The intermediate stage is responsible for the eight intermediate value computations. Each involves reading data from two input registers, performing an ALU operation, and writing the result to an intermediate register. The output stage follows the completion of the intermediate stage by producing the four output values, one component at a time. These output computations require reading from two intermediate registers, performing an ALU operation and forwarding the result to the next stage of the FFT-16 computation.

The ALU used in this project is based on the static dual rail asynchronous adder from work done at the University of Manchester [6]. This is a ripple carry adder that has a carry bit for both carry out 0 and carry out 1. This is an average case, data dependent, ALU with completion detection. This ALU achieves average case performance because each slice of the adder can identify a carry generate situation (both inputs are a logic high) and a carry kill situation (both inputs are a logic low). In the case of a carry generate, the slice can immediately raise the carry out 1 bit regardless of the carry in. Similarly, the carry kill case raises the carry out 0 bit regardless of the carry in. If the inputs do not indicate either one of these cases, the slice must wait for the appropriate carry in signal to be raised before determining the carry out. When a certain slice raises one of its carry out signals, it raises a completion signal that is fed into an AND tree with all the other stages. This ALU is used in both the FFT-4 and the complex-valued multiplier.

4.3.3 Complex-Valued Multiplier. The complex-valued multiplier is responsible for multiplying two 32-bit complex values together to produce a third 32-bit value. Normally this multiplication would produce a 64-bit complex value but in order to implement the hardware properly, the 32 least significant bits (LSBs) of the mantissa are truncated.

The Booth Multiplication Algorithm. The average case delay characteristics of the Booth multiplication algorithm make it a prime choice for asynchronous design. This benefit, coupled with the significant amount of prior work in this area, led to minimal consideration of other designs.

The average case speed advantage for the Booth multiplier lies in the assumption that shifting takes less time than adding or subtracting. By determining whether or not an ALU operation is required, and performing one only when necessary will save overall computation time. The standard Booth (radix-2) algorithm works by examining the LSB of the current multiplier, X_n, as well as the previous (pre-shifted) multiplier LSB, X_{n-1} (The first two bits examined are X_0 and 0). Table 4.3 displays the instruction for each combination of decoded bits.

Table 4.3 Radix-2 Booth Encoding

X_n	X_{n-1}	Operation
0	0	Shift product right
0	1	Add multiplicand, then shift
1	0	Subtract multiplicand, then shift
1	1	Shift product right

When employing the standard Booth algorithm, the number of shifts required to complete a multiply operation is equal to the length of the multiplier. The worst case timing for this radix-2 algorithm will occur when the multiplier is a string of alternating ones and zeros. In our incessant desire to make the fast–faster, an improvement was made to the standard Booth algorithm where three bits are examined at once instead of just two [2]. When operating on three bits at a time, the number of shifts required is halved. However, it is more likely an ALU operation will be required. Table 4.4 shows the operations required for each bit combination. Again, the first set of bits examined is X_1, X_0, and 0.

Table 4.4 Radix-4 Booth Encoding

X_n	X_{n-1}	X_{n-2}	Operation
0	0	0	Shift product right
0	0	1	Add 1 $\times$ multiplicand, then shift
0	1	0	Add 1 $\times$ multiplicand, then shift
0	1	1	Add 2 $\times$ multiplicand, then shift
1	0	0	Subtract 2 $\times$ multiplicand, then shift
1	0	1	Subtract 1 $\times$ multiplicand, then shift
1	1	0	Subtract 1 $\times$ multiplicand, then shift
1	1	1	Shift product right

Previous Work. As a special study at the Air Force Institute of Technology, Sam SanGregory undertook the asynchronous multiplier design. He pursued a low power design using pass-transistor, dynamic, and pre-charge logic, all of which should be avoided for a radiation tolerant design. Nonetheless, many of his designs were translated into radiation tolerant implementations and used with only slight modifications in some cases.

Chosen Design. From Section 3.3.1 we know that a complex-valued multiply is actually four simple multiplies with one addition and one subtraction,

$$\begin{aligned} X \times Y &= (X_r + jX_i) \times (Y_r + jY_i) && (4.3)\\ &= X_r \times Y_r + j \times (X_r \times Y_i + X_i \times Y_r) + j^2 \times (X_i \times Y_i) && (4.4)\\ &= X_r \times Y_r - X_i \times Y_i + j(X_r \times Y_i + X_i \times Y_r) && (4.5)\end{aligned}$$

where the real and imaginary components are kept as separate values.

We also know from Section 4.2 that the multiply is in the critical path. To make the multiplier fast and small, a two-cycle dual multiply scheme was implemented. One way of looking at Equation 4.5 is to see that X_r is multiplied by two values and X_i is multiplied by the same two values. The chosen design of the FFT-4 element from Section 4.3.2 makes only the real or the imaginary word of each component available at once. From this, the chosen "multiply^2" design decodes the Booth instructions from the FFT-4 data-word and operates on both multiplicand components. Following the two cycles, the final subtraction and addition operations are performed to complete the complex-valued multiply.

Time Saving Modifications. In the work performed by SanGregory [17], some ingenious time saving modifications were pursued. These time saving short-cuts were based on the assumption that, for DSP applications, many multiplicands would be equal to zero, one, or negative one. When one of these values appear, the multiply could finish quickly and save power. Because we are working with an asynchronous design, any speed improvements in the critical path are helpful to improve the overall performance of the chip.

Table 4.5 Extra Storage Decode

S_1	S_0	Operation
0	0	Multiply X by 0
0	1	Multiply X by 1
1	0	Multiply X by -1
1	1	Multiply X by Y

Since the Multiplicands are stored in some kind of memory, it would not be difficult to store the additional S_1 and S_0 bits along with the constant multiplicand.

The way the complex-valued multiplier is designed for this project (receives only the real or imaginary data-word at a time) will not allow for these modifications to work. However, the FFT-4 design proposed in Section 7.2.1 would allow the shortcuts to work in conjunction with the "multiplier^2" design. Because the real and imaginary data-words from the FFT-4 would appear simultaneously, the multiply short cut or the Booth instructions could be decoded from the multiplicand in memory propagating the product as fast and energy efficiently as possible.

4.3.4 Pipelined Crossbar Switch. The purpose of the pipelined crossbar switch is to interleave the partially transformed values after the multiply operation and before the second FFT. FFT-4_0 will operate on the first output of each multiplier, FFT-4_1 will operate on the second output of each multiplier and so on.

The crossbar switch is complex in terms of inputs and outputs, however, its design is not that difficult. As you can see from the petri-net diagram of Figure 4.3, it consists of eight token loops, each with four token stops. There are four horizontal loops which are advanced by each multiplier handshake. The four vertical loops are in a ripple chain and are advanced by the acknowledgment from the reused FFT-4 elements. There are four 32 bit registers, one for each of the multiplier outputs. Since the arrival of values may be out of order, a multiplier may have to wait to forward its output. To prevent idle time and increase parallelism, one value from each multiplier is latched. If the second value arrives before the first is routed, it will not be latched and the multiplier will have to wait.

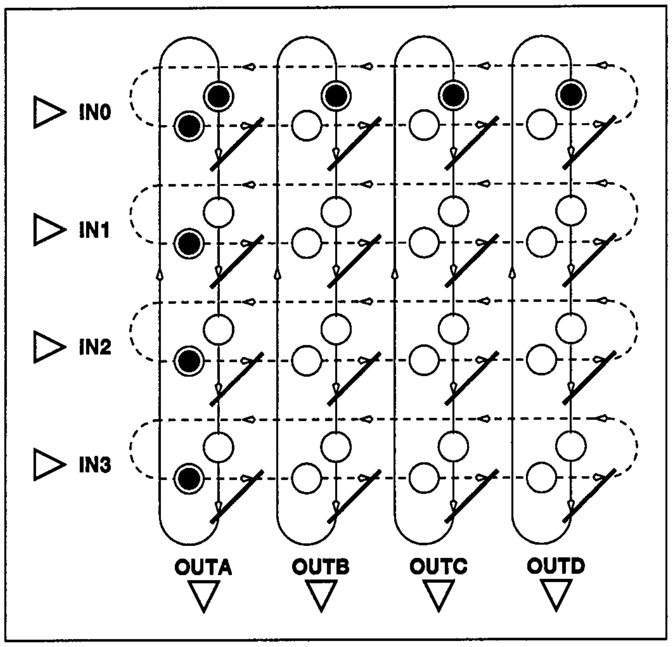


Figure 4.3 Pipelined Crossbar Switch Petri-net

When both tokens are present at a given stop, a tri-state device is enabled and the proper value is put onto one of the four output busses to the FFT-4 elements. When this tri-state device is enabled a handshake is also commenced with the specified FFT-4 element. When the Acknowledge returns from the FFT-4 element, the horizontal loop token advances to the next stop. This process will go on until all 16 data elements proceed through the crossbar.

4.3.5 Expander. The final operation in the algorithm is to compose the final output sequence following the N_2 point FFT. In hardware, the expander receives X_0, X_4, X_8, and X_{12} from the FFT-4_0 block, X_1, X_5, X_9, and X_{13} from the FFT-4_1 block, and so on. In sequencing these outputs, the expander will closely match the input sampling frequency.

Previous Work. Originally, the same concept used to implement the decimator was to be used in implementing the expander. Specifically, the design was to have an expander tree put together to complete the N_2 expansions to assemble the correct output. A burst-mode specification was developed for this that proved correct as far as the control path was concerned but implementing the data path became too complex to implement this design.

Chosen Design. Based on the success of the ripple loop of the crossbar, the design for the expander followed suit. In this instance, however, there is only one loop of four ripple one-hots. The loop is not allowed to progress unless the proper order of requests is maintained. It is possible for the components to become valid before their turn but they are ignored until the loop token appears. Figure 4.4 shows how this ordering works after some inputs and outputs have been transfered. Note how the token is waiting

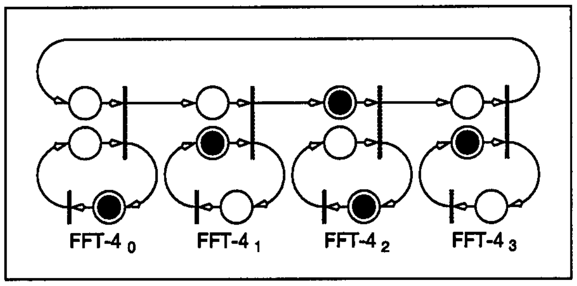


Figure 4.4 Expander Petri-net

for an output from FFT-4_2 but FFT-4_3 already has its output ready. The FFT-4_3 output will have to wait until FFT-4_2 produces its output and the token is advanced to the next stop.

4.4 Summary

This chapter has attempted to highlight all work that has been accomplished to date on this project discussing each at a high level. Chapter V will cover detailed implementation of each piece of hardware.

V. Implementation

This chapter gives an in-depth look at how each part of the FFT-16 system is implemented. Included in each section are block diagrams of all different levels, state transition diagrams, and actual gate level implementations when necessary. Each sub-system is discussed in the order encountered in the calculation of the 16 point FFT.

5.1 Decimator

Unlike the theory behind the decimator, its implementation is very simple. Essentially, the decimator is a router that diverts the input handshake into the proper FFT-4 block (0-3). Because the data lines are common to all four FFT-4 blocks they are routed to all, but the proper values are latched only when one of the blocks receives a handshake. Figure 5.1 shows the decimate-by-two state machine and Figure 5.2 shows how the decimator is implemented as a tree of three 2:1 two cycle decimators.

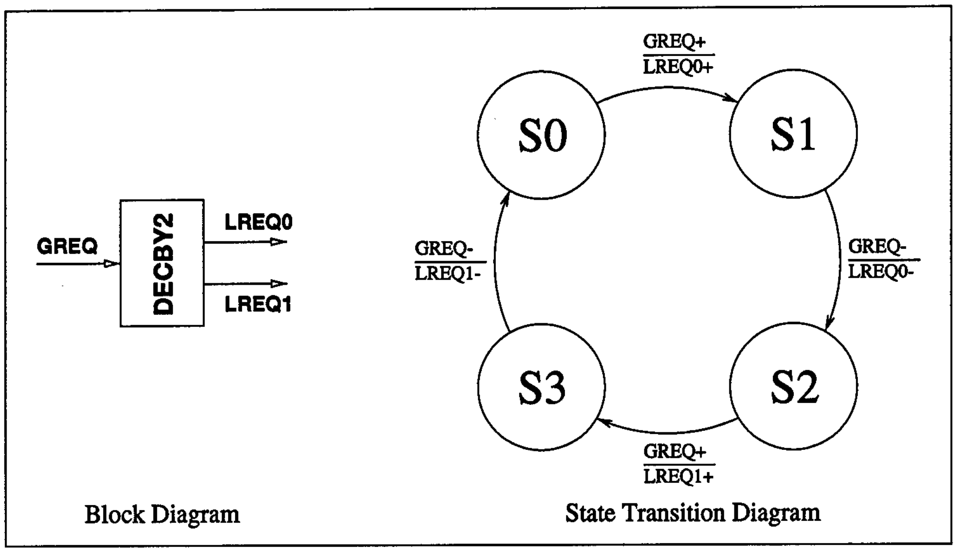


Figure 5.1 Decimate-By-Two State Machine

With this design, acknowledgment is handled externally. This will be further explained in Section 5.2.1. Using this machine, the decimate by four is created by linking the even and odd pairs together at the leaf node. This is shown in Figure 5.2.

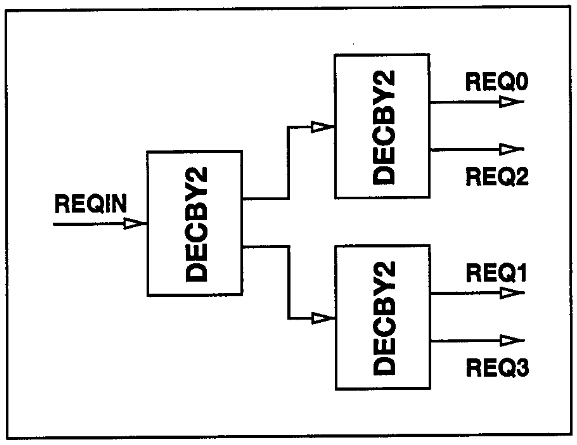


Figure 5.2 Decimate-By-Four Block Diagram

The only real challenge of the decimator design is to make sure that each of the leaf "REQUEST" signals have enough drive to get to each of the FFT-4 blocks. This will be handled in the layout.

5.2 FFT-4

To abide by the purpose of choosing an N that is a perfect square, the FFT-4 block executes both the N_1 and N_2 point FFT computations. Each of the four FFT-4 blocks consist of four major parts as shown in Figure 5.3.

The first operation reads the four 32-bit input values into the input register bank with a series of REQIN/ACKOUT handshakes. When x_0, x_1, and x_2 are read in, the first of the intermediate operations may begin by reading two values out of the eight registers in the input stage, performing addition or subtraction in the Arithmetic Logic Unit and storing the sum or difference in the intermediate register bank. When the eight intermediate calculations are complete, the output control stage executes the final eight operations to complete the FFT calculation. Each one of these operations begins by reading two values out of the intermediate register bank, performing arithmetic on them and concludes with releasing one 16 bit value at a time to the next stage. Sections 5.2.1 through 5.2.4 discuss each of the four major components at length.

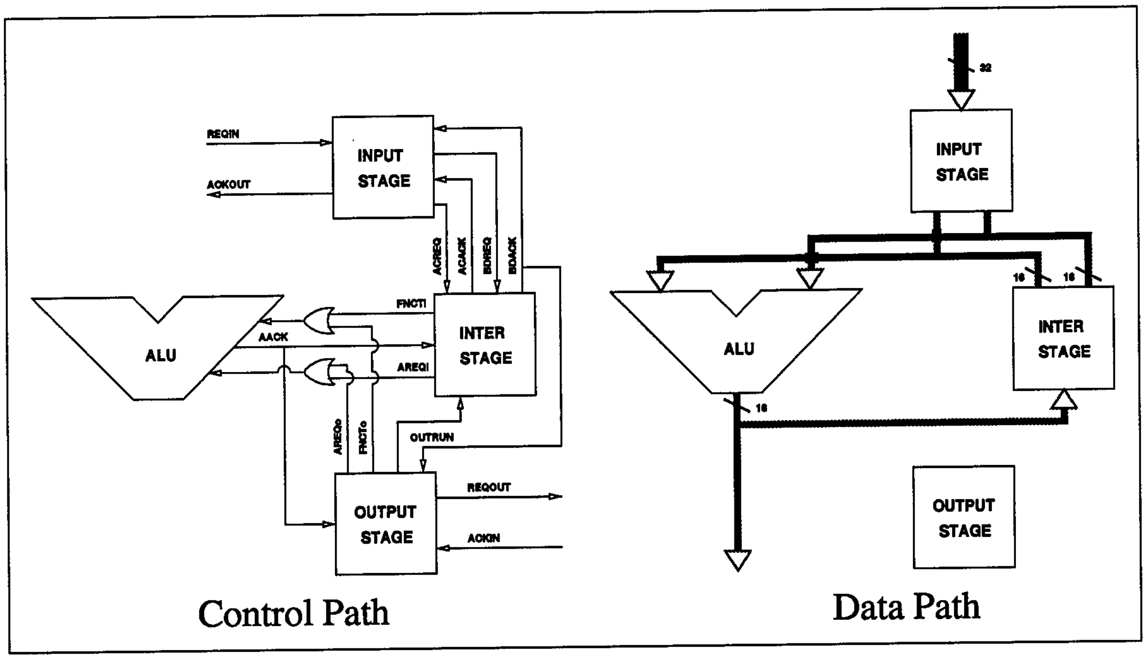


Figure 5.3 FFT-4 Block Diagram

5.2.1 Input Control/Registers. The overall function of the input portion is to receive four 32-bit complex values, each with a REQ/ACK handshake with the previous stage and store each in a dedicated static register. The input stage also handshakes with the intermediate computational block when the necessary input values become available to create the intermediate values *a*, *b*, *c*, and *d*.

Global One-Hot Encoded AFSM. Each new input value has its own register so the control unit needs to write enable the dedicated register at the appropriate time. This is accomplished using a global one-hot encoded state machine. Figure 5.4 shows the block diagram and state transition diagram for this machine. Each global one-hot will assert its LREQ or "Local Request" signal when both GREQ, or "Global Request" and GO are high. This implementation becomes useful when many are "chained" together as in Figure 5.5. When a repeated GREQ signal is given, the waveforms of Figure 5.6 result.

Input Control. The function of the input control is to keep track of the number of input handshakes so the intermediate control block can be signaled when

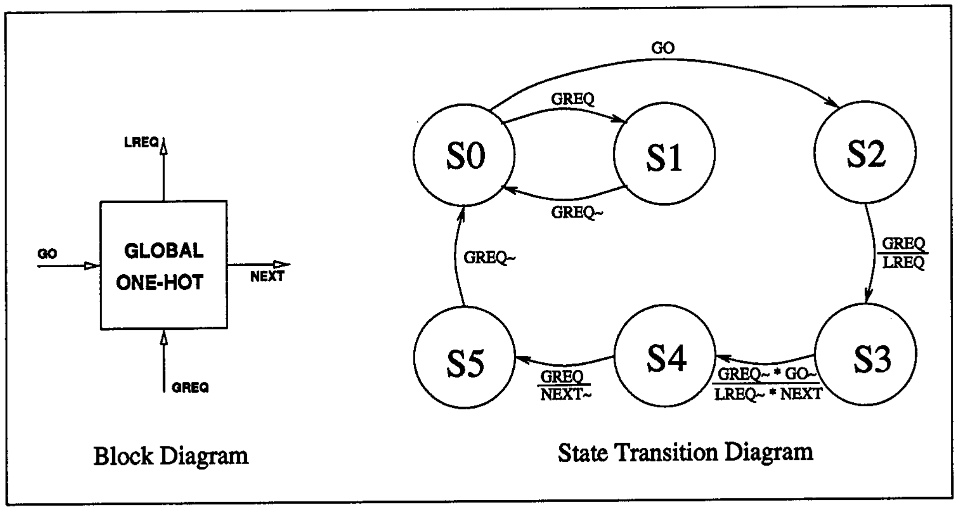


Figure 5.4 Global One-Hot State Machine

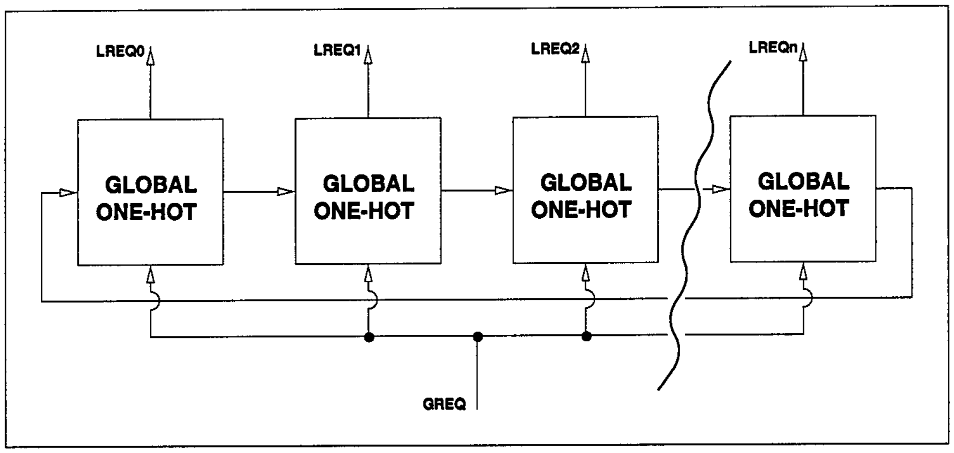


Figure 5.5 Loop of Global One-Hot Machines

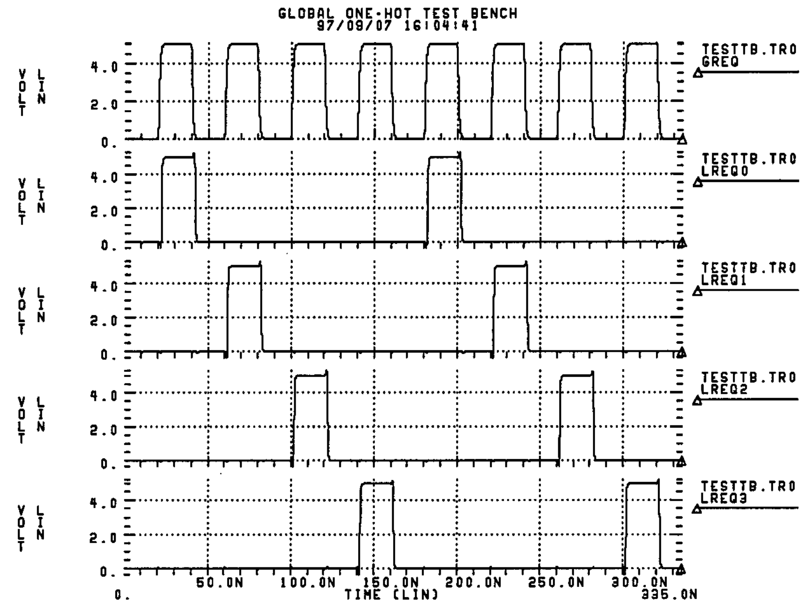


Figure 5.6 Four Global One-Hot Waveforms

enough values are present to begin intermediate arithmetic. Figure 5.7 contains the state transition diagram which is a single loop of states that handshakes with the intermediate stage when x_0 and x_2 are ready, and when x_1 and x_3 are ready. The actual complexity of this state machine is fairly large because there are very few distinguing variables between the states.

Register Cell. Since the main function of the input portion of the FFT-4 element is to store data, the most important component is the register cell. Figure 5.8 shows the typical cell used throughout the entire FFT-16 design in some variation or another.

This register cell is based on a clocked register design contained in the AFRL/MRC library. All clocking components have been removed leaving this simple ASFM. The double invert on the WEN, or write-enable, signal is necessary so upon a high-to-low WEN transition, the non-data inputs on each of the AND portions of the AND-OR-INVERT

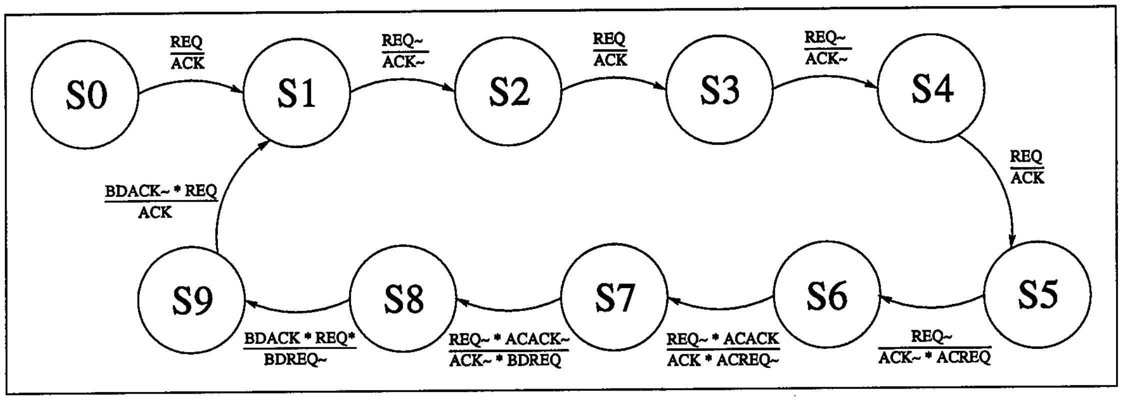


Figure 5.7 Input Control State Machine

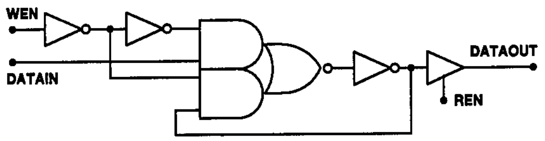


Figure 5.8 Standard Register Cell

gate are high for a period of one inverter gate delay, removing the hazard on the gate. If only a single inverter was used for the $\overline{\text{WEN}}$ input, upon the same high-to-low transition, there would be a logic low on the non-data input of each AND portion for the period of an inverter gate delay causing the hold value of of the register cell to be zero regarless of its intended value.

Putting it all Together. Now that all of the components comprising the input stage of the FFT-4 have been described, we can take a look at how it all works together. Figure 5.9 shows how the IREQ signal from the input control state machine becomes the global request to the four one-hot machines. The local request from each machine becomes the write enable to each of the registers. One important thing to point out is the 4-way fan-in (16 way fan-in internal to the register) for the write acknowledge only comes from the imaginary component register in each pair saving area and time. The correctness of the delay with only one register fanin for the ACKOUT signal was verified using Spice[6].

[6]Avanti Corp. Hspice version 95.1.

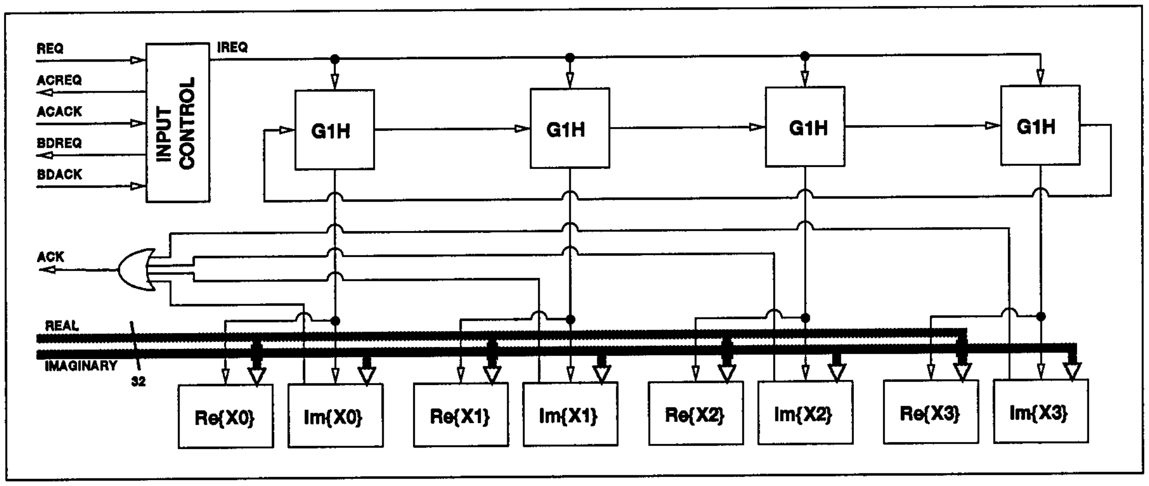


Figure 5.9 Input Block Diagram

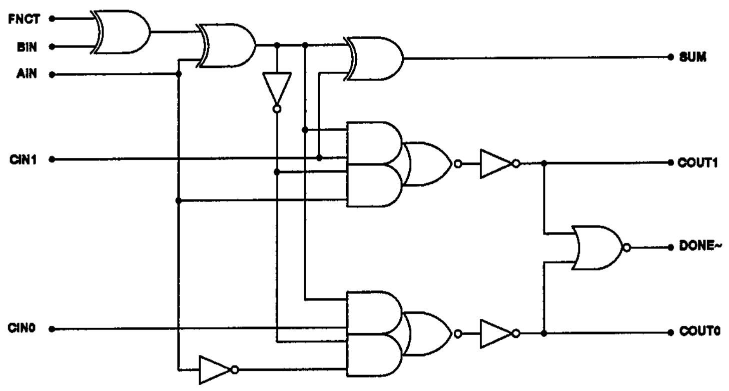


Figure 5.10 Nominal ALU Stage

5.2.2 Arithmetic Logic Unit. Two basic circuits make up the dual-rail carry-ripple ALU. Figure 5.10 represents the 16 nominal stages of the ALU. Each slice uses an XOR gate on the B input to select inversion if the Function Bit is asserted. Then the SUM is computed using the B or $\overline{B}$ XORed with A followed by an XOR with CIN1. The computation of the carry bits can happen in one of two ways. If the A and B inputs indicate a carry-kill (00) or a carry-generate (11) condition, the COUT0 or COUT1 bit can be raised accordingly. This allows the carry chain to be shorter than the worst case most of the time allowing for faster operation. If the input data does not determine the proper carry, the stage must wait for one of the carry-in bits to become valid before the appropriate

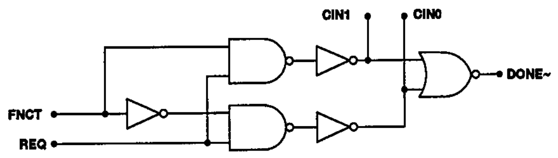


Figure 5.11 ALU Setup Stage

carry bit is raised. The $\overline{\text{DONE}}$ output from each stage is fed into a NOR-NAND tree [7] to compose the AACK signal.

To initiate the ALU operation, the AREQ signal is sent to the special "stage-1" of Figure 5.11. Here the carry chain begins by raising the COUT0 or COUT1 depending on the FNCT input. If there are no carries from this setup stage, the DONE signal becomes the completion signal for the ALU (AACK).

5.2.3 Intermediate Control/Registers. The intermediate operations are the most complex in the FFT-4 because they involve reading a 16 bit value from the input register file, performing an ALU operation (either addition or subtraction) and then writing the sum or difference to one of the eight intermediate registers. This operation cycle repeats eight times during the intermediate phase so it is apparent that generating the control signals for the ALU and the register banks is quite complex. Much of this complex signaling, though, is handled using a new approach of the one-hot architecture.

Ripple One-Hot Encoded AFSM with Context. Unlike the global one-hot sequence, the ripple one-hot sequence works with external signaling only at the beginning and end of an indefinite length sequence. Figure 5.12 contains the block diagram and state transition diagram of the ripple one-hot state machine and Figure 5.13 shows how a sequence of these ripple one-hots is linked together. It is important to note that this implementation of the ripple one-hot sequence uses two enabling signals for each block, the REQ signal and the CONTEXT signal. The CONTEXT signal is raised earlier, to allow setup time for the ALU inputs whereas when the REQ signal is raised, the PREQ or process request signal is immediately raised and routed to an OR tree to enable the ALU.

[7]Normally this would be a NAND-NOR tree to achieve the 17-way AND function but the tree starts with $\overline{\text{DONE}}$ instead of DONE.

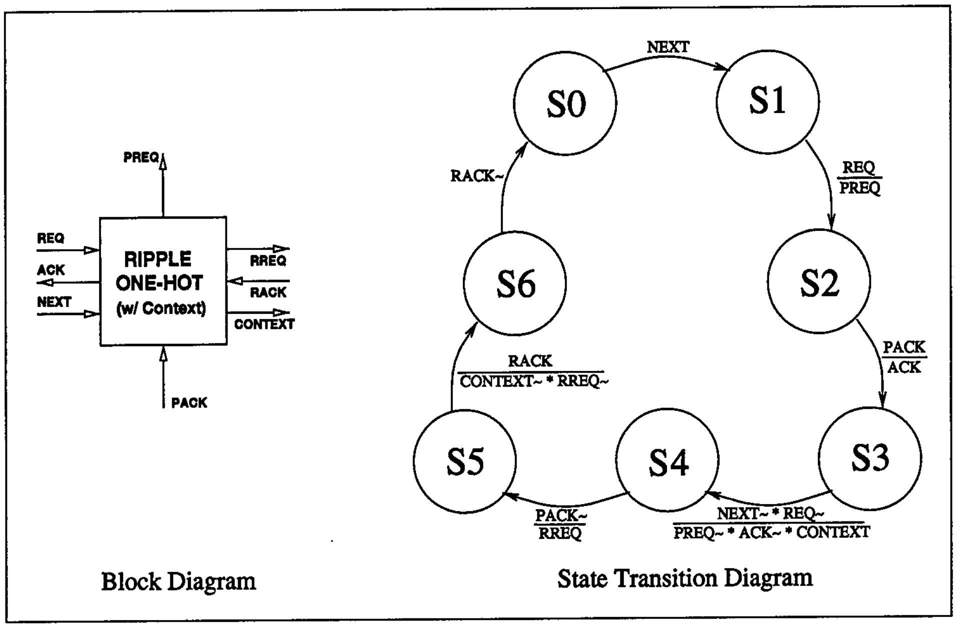


Figure 5.12 Ripple One-Hot State Machine With Context

Intermediate State Machine. The intermediate state machine exchanges handshakes with the input state machine for the computation of the intermediate values. The intermediate state machine must also know that all output values have been completed before the next cycle begins via the OUTRUN signal.

To start the ripple one-hot sequence, the intermediate state machine of Figure 5.14 sends the FREQ signal and receives the FACK and LREQ signals. Since it is performing

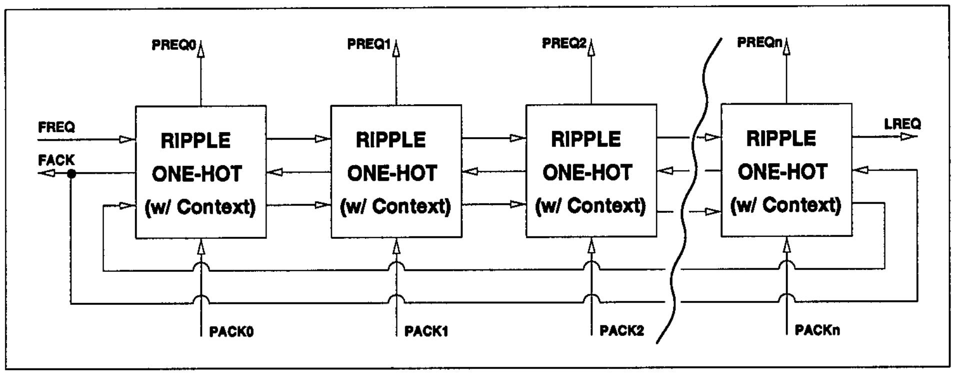


Figure 5.13 Loop of Ripple One-Hot Machines

the one-hot ripple cycle for both the calculation of a and c as well as b and d, it must do it twice.

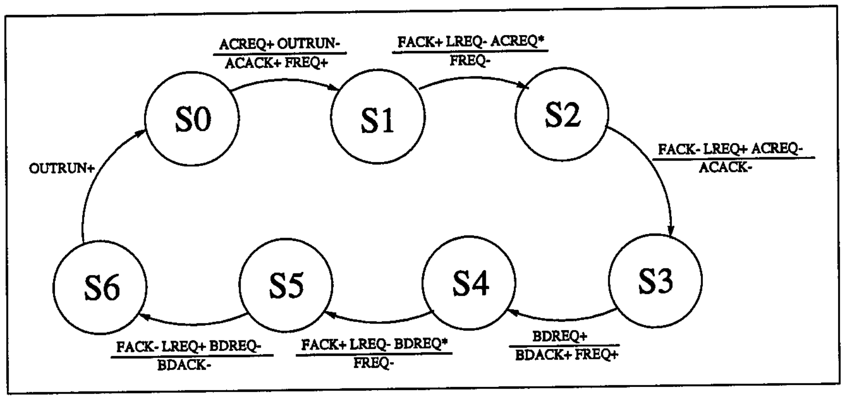


Figure 5.14 Intermediate State Machine

Putting it Together. When connected in the full architecture of the intermediate control, Figure 5.15 results. It is important to point out some key information about the intermediate stage block diagram. All blocks labeled "DEC" represent 1:2 decoders with the BDACK signal acting as the selector. When BDACK is low, the top or left signal is selected and when BDACK is high, the bottom or right signal is selected. These decoders enable reuse of the four ripple one-hot machines thereby reducing the size of the control path.

The AND-OR-INVERT structure on the input register read enable signals is there to make sure the proper register is enabled even after the context signal is lowered.

Usage of the AND gate on the intermediate register write enable is present to make sure the write is enabled only long enough for the valid data to get written. Once the PREQ signal goes low, some the outputs of the ALU will change quickly. If the register is write enabled too long, the incorrect value will be locked in. The OR-AND structure on the PACK or process acknowledge input of each block is necessary to ensure the process resets through the ALU and not when PREQ is lowered. If the OR gate were not used and the intermediate write enable was simply PREQ AND AACK, the setup time for the ALU would be reduced and delay elements could be required.

The delay element on the FREQ signal is required since the OR-AND gate is on the first context signal. This is used because the context signal looping around from the last one-hot to the first is high when the intermediate stage is idle. If this high context signal propagates to the register enabling lines, there will be more than one driven value on each data path. This delay element is also present in the output state machine but only an AND gate is required in the first context signal since only one pass through the one-hot sequence completes the operation.

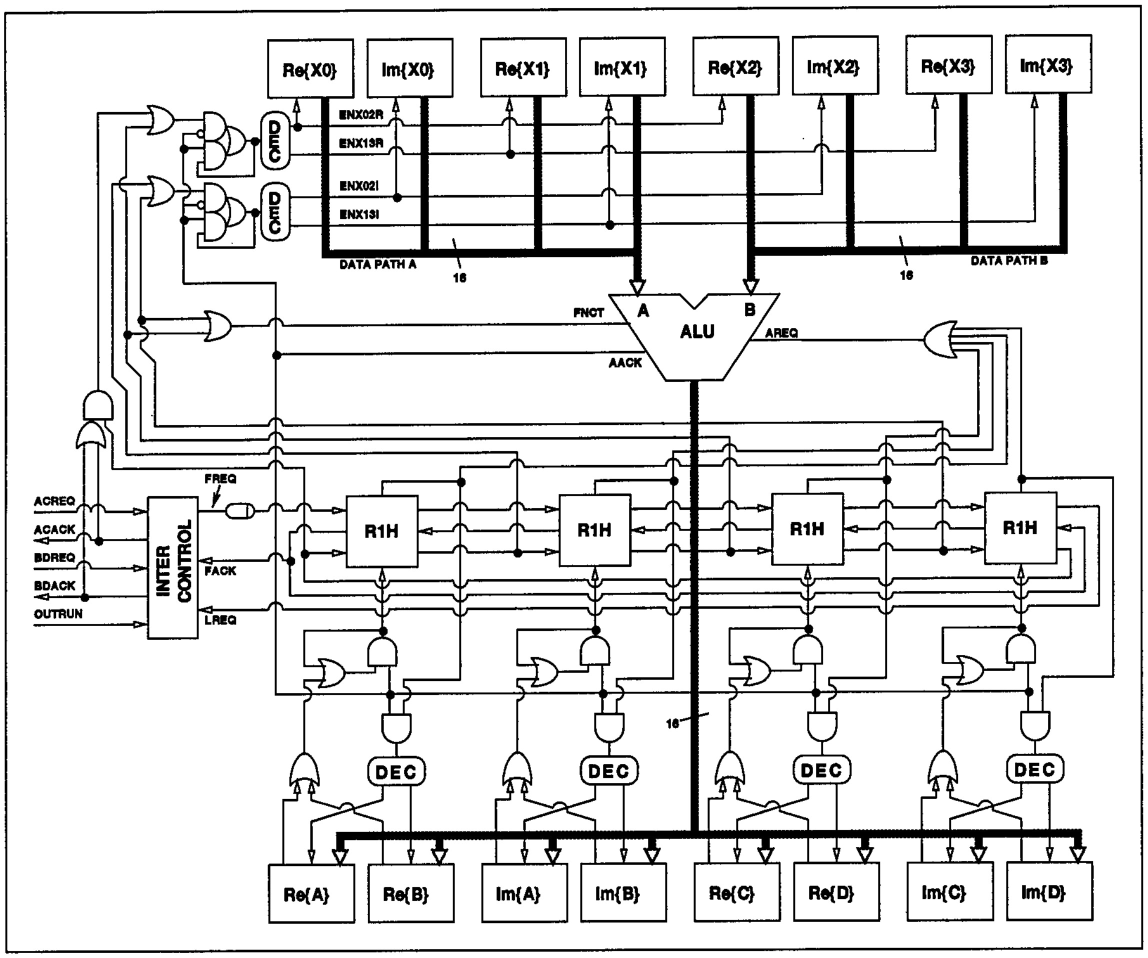


Figure 5.15 Intermediate Block Diagram

Table 5.1 contains the step-by-step operation of the intermediate stage. The regularity of the ALU computations allows for the reuse of the ripple one-hot sequence for the computation of *b* & *d* following the computation of *a* & *c*.

Table 5.1 Intermediate Stage Function

Operation	Datapath A	Datapath B	ALU	Write
1	Re$\{x_0\}$	Re$\{x_2\}$	Add	Re$\{a\}$
2	Im$\{x_0\}$	Im$\{x_2\}$	Add	Im$\{a\}$
3	Re$\{x_0\}$	Re$\{x_2\}$	Subtract	Re$\{c\}$
4	Im$\{x_0\}$	Im$\{x_2\}$	Subtract	Im$\{c\}$
5	Re$\{x_1\}$	Re$\{x_3\}$	Add	Re$\{b\}$
6	Im$\{x_1\}$	Im$\{x_3\}$	Add	Im$\{b\}$
7	Re$\{x_1\}$	Re$\{x_3\}$	Subtract	Re$\{d\}$
8	Im$\{x_1\}$	Im$\{x_3\}$	Subtract	Im$\{d\}$

5.2.4 Output Control. Table 5.2 shows that the first output value (X_0) requires the computation of the intermediate value b indicating the output machine must wait until all the intermediate calculations have been completed before it begins. Figure 5.16 shows the S0 $\rightarrow$ S1 transition and the S1 $\rightarrow$ S2 transition are dependent on BDACK rising and lowering respectively before the output stage may begin operation.

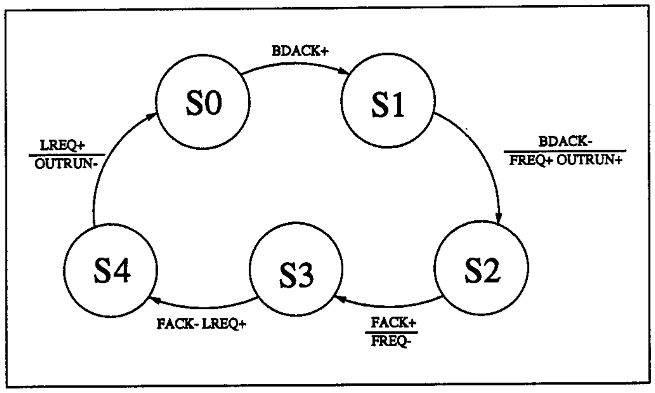


Figure 5.16 Output Finite State Machine

The output block diagram, pictured in Figure 5.17, shows many of the same features of the intermediate stage with a few notable exceptions. First of all, there are eight ripple one-hot machines in the sequence instead of just four. This is required because the ALU operations and operands do not fall into a repeatable pattern as in the intermediate operations. Secondly, the OR-AND structure holding the PACK signal high to each one-hot is reliant on the ACKIN from the previous stage rather than the AACK signal from the ALU. The output stage and the intermediate stage do share the same ALU so even

Table 5.2 Output Stage Function

Operation	Datapath A	Datapath B	ALU	Write
1	Re$\{a\}$	Re$\{b\}$	Add	Re$\{X_0\}$
2	Im$\{a\}$	Im$\{b\}$	Add	Im$\{X_0\}$
3	Re$\{c\}$	Im$\{d\}$	Add	Re$\{X_1\}$
4	Im$\{c\}$	Re$\{d\}$	Subtract	Im$\{X_1\}$
5	Re$\{a\}$	Im$\{b\}$	Subtract	Re$\{X_2\}$
6	Im$\{a\}$	Re$\{b\}$	Subtract	Im$\{X_2\}$
7	Re$\{c\}$	Im$\{d\}$	Subtract	Re$\{X_3\}$
8	Im$\{c\}$	Re$\{d\}$	Add	Im$\{X_3\}$

though it is not reflected in the two block diagrams of Figures 5.15 and 5.17, the control path shown in Figure 5.3 is correct.

5.3 Complex Multiplier

Refering to the previous chapter on this subject, a complex multiply of two values actually requires four real multiplies followed by one addition and one subtraction. This implementation performs two real multiplies in parallel followed by the ALUs dedicated to addition and subtraction. Section 5.3.1 describes the implementation of the real multiply followed by the data path and control path for the completion of the real multiply.

5.3.1 Real Multiply. Figure 5.18 contains the control and half of the data path of the radix-4 16-bit real multiplier. Starting with the REQIN signal, the first block encountered is the loader state machine. The loader puts the multiplier in the X register at the appropriate time. A series of state machine loops are performed by the multiplier state machine which controls the ALU math and the shifting in the product register. When the correct number of shifts have been performed, the real multiplier control unit will handshake with the rest of the complex multiplier to compute and store $Re\{Z\}$ and $Im\{Z\}$ in the crossbar memory element. Notice the dotted line indicating the hardware portion that is shared between the parallel real multiplies.

To understand the two data paths of the real multiplier (hereafter refered to as the multiplier[2]), refer to Figure 5.19. Although all the control in the multiply unit is shared, only the control for the ALUs is illustrated to simplify the figure. Because the Booth

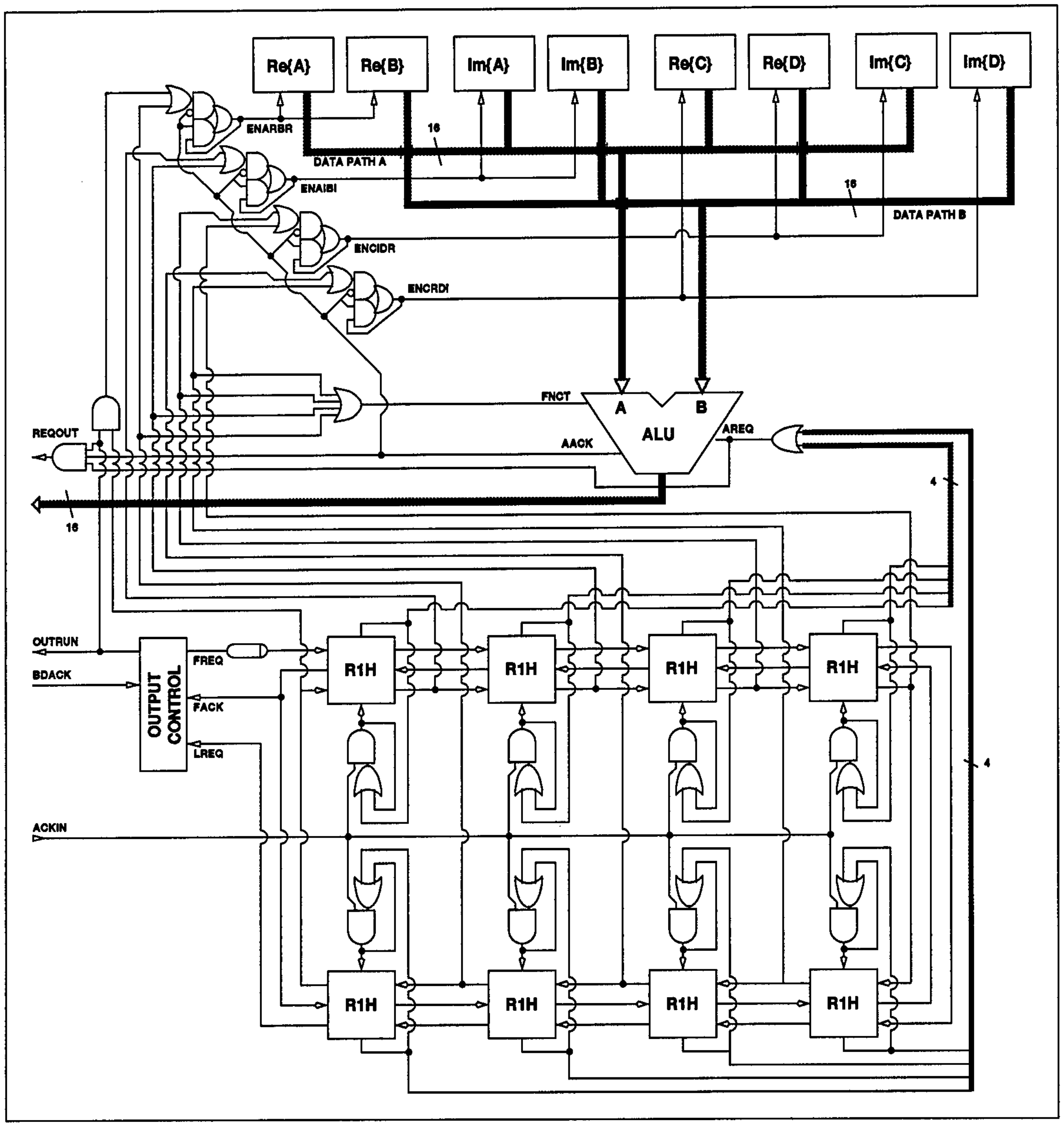


Figure 5.17 Output Block Diagram

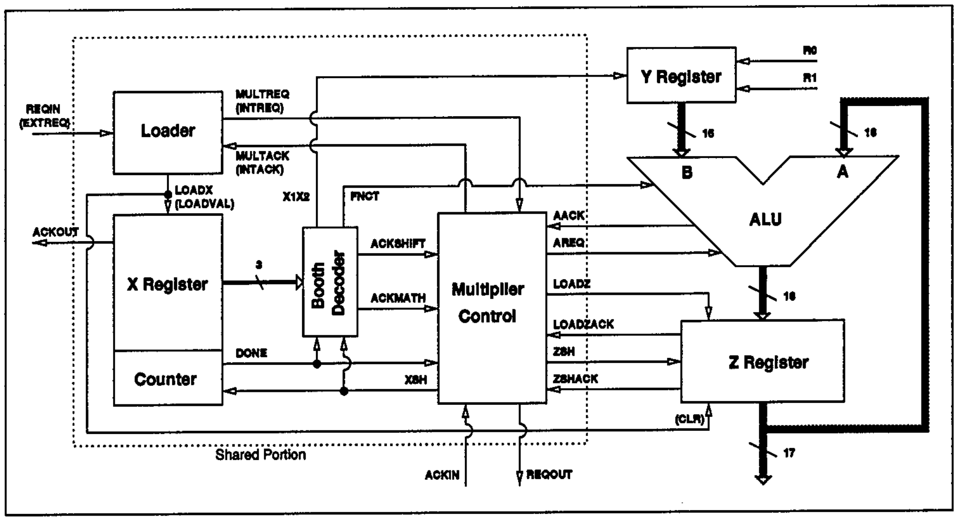


Figure 5.18 Real Multiply Partial Block Diagram

decoded ALU instruction will be the same for both multiplies, the AREQ signal is routed to both ALUs. A C-element is required to synchronize the AACK signals from each ALU because the ALU operating time is data dependent and the real and imaginary constant additions will likely complete at different times. Each data path has its own ALU, shift register, and constant storage.

Loader. Continuing with the elements of the multiplier^2, we start with the loader in Figure 5.20. It is responsible for making sure the multiplier^2 is not functioning (MULTACK is low) before starting the next multiply operation. Figure 5.18 shows how the LOADX signal also clears the Z Register. For this reason, the loader must wait until the MULTACK signal is low before loading a new value because the final output adder and subtractor of the complex multiplier rely on the static storage of each Z Register for proper completion.

Multiplier Register. The next item is the X register or multiplier register. The X register holds the 16-bit multiplier with an additional zero in storage as X_{-1}. Since this is a radix four Booth algorithm, there are eight shifts required to cycle through one real multiply operation. The number of shifts is controled by the counter containing nine global one-hot machines in series. The first eight shifts enable the proper

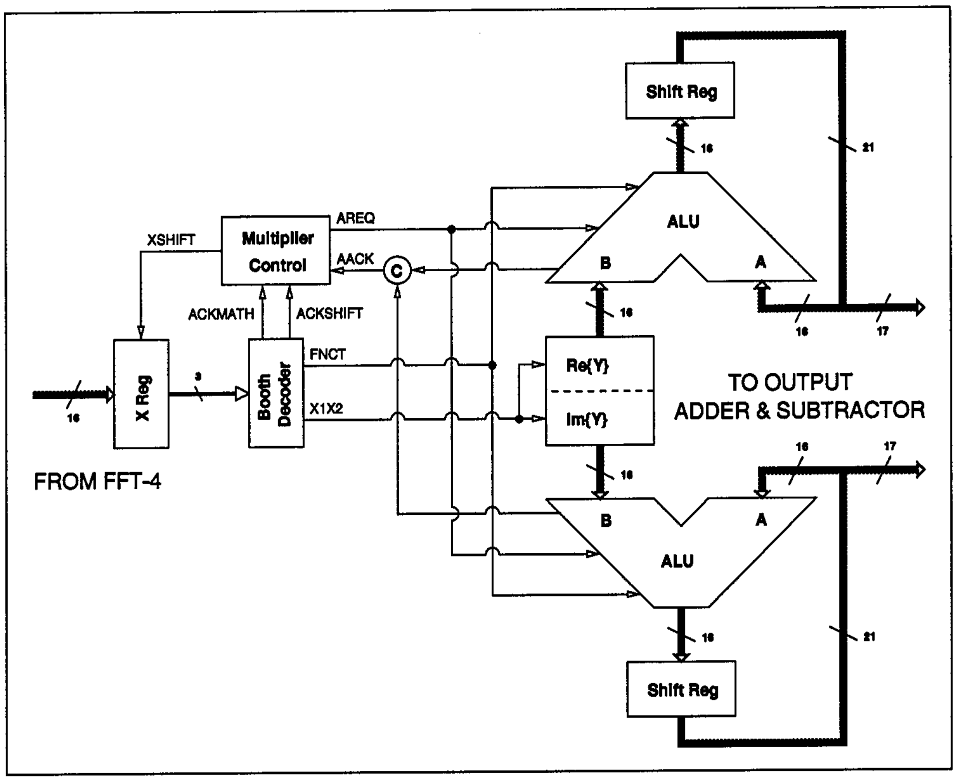


Figure 5.19 Multiplier[2] Block Diagram

three bit set from multiplier while the ninth shift raises the DONE signal to notify the multiplier control that instructions have all been decoded.

The multiplier register is static and the bits do not actually shift from register to register. When the XSH signal comes into the counter, each local request turns on tri-state buffers for three register bits at a time. The first shift enables b_{-1}, b_0, and b_1. The second shift enables b_1, b_2, and b_3 and so on with the eigth local request enabling b_{13}, b_{14}, and b_{15}.

Booth Decoder. Once the three bit instruction is sent to the Booth decoder shown in Figure 5.21, it will determine whether or not an ALU operation is required. If so, the ACKMATH signal will be raised. If not, the ACKSHIFT signal will be raised. Regardless of which acknowledge signal is used, the FNCT[8] bit determining

[8]0: ALU addition; 1: ALU subtraction.

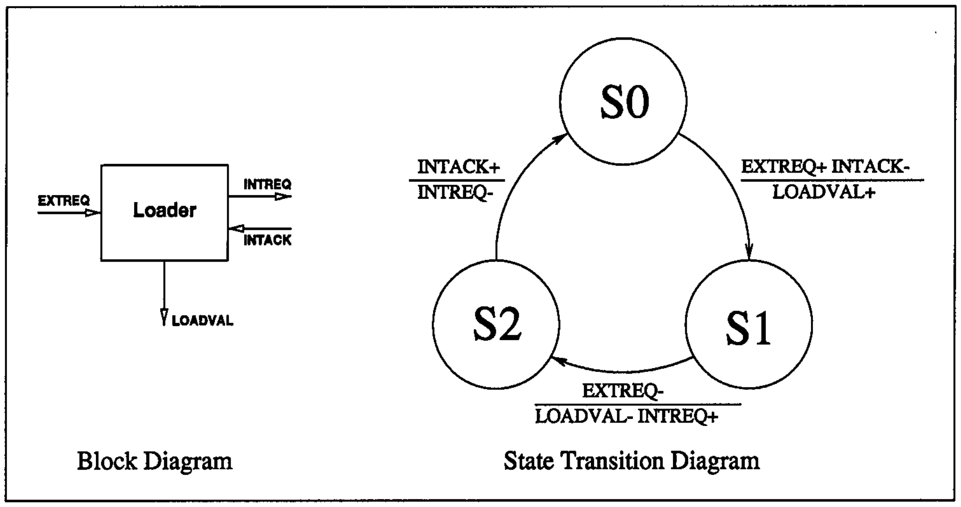


Figure 5.20 Loader State Machine

addition or subtraction and the X1X2[9] is selected. This can be done because FNCT and X1X2 became don't care bits when the ALU is not required for that operation. An added benefit is that it makes the decoding logic much less complex.

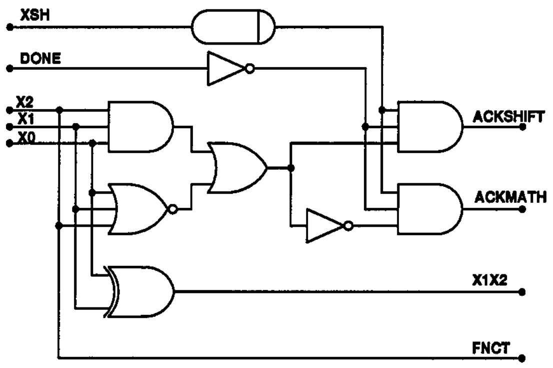


Figure 5.21 Radix-4 Booth Decoder

The DONE signal is inverted and input to the two output AND gates to make sure that when DONE is raised, the Booth decoder will not respond with an ACKMATH or ACKSHIFT which could alter the data in the product registers. The XSH signal is also injected after a delay element to prevent the X2, X1, and X0 line values from decoding until the new values become valid. Without the proper delay element used here, the Booth

[9]0: Select multiplicand × 1; 1: Select multiplicand × 2.

decoder has the potential to raise the ACKSHIFT signal and the ACKMATH signal during the same XSH cycle.

After building this design and understanding more about bounded-delay models, probably a better approach to the three Booth instruction bits would be to use a dual-rail delay-insensitive scheme similar to the carry chain in the ALU. This would clearly be a safer implementation because there would be no dependence on delay elements.

Multiplier Control. Now we get to the heart of the multiplier^2 control, the multiplier AFSM (multAFSM) of Figure 5.22. Refering once again to Figure 5.18, you can see the relative input and output complexity of the multAFSM. The state transition diagram, though, shows the operation really is not that complicated with three possible loops that the multAFSM can take during operation. Starting in state 0, the multiplier control receives the MULTREQ signal, responds by raising the MULTACK, and waits for MULTREQ to lower. Once this happens, the first of nine XSH signals is raised. Then, depending on the Booth instruction, the control unit can follow the ACKSHIFT loop or the ACKMATH loop. There is no shifting in the ACKMATH loop because the outputs of the ALU are in a hardwire shift-by-two formation reducing computation time and control complexity within the real multiply. After the X vector has been fully decoded by the first eight shifts, the ninth XSH pulse will raise the DONE signal and finish the first loop of the AFSM by handshaking with the complex portion of the multiplier^2.

Multiplicand Register. The Y register (multiplicand register) is pretty basic for the FFT-16 multiplier. Because only four constants are required for the FFT-16, local storage of the constants does not use much die area. Using this locality of storage concept, the design of Figure 5.23 constitutes the cell used for each bit. In this cell, there is a 4:1 multiplexor with two select bits, R_0, and R_1, implemented using two 3-3 AND-OR-INVERT gates joined by a two input NAND gate. Also, since there is an option between the current bit or the previous bit, an additional 2-2 AND-OR-INVERT gate is used as a multiplexor with the X1X2 signal selecting. When X1X2 is low, the current bit is selected for output and when X1X2 is high, the previous bit is selected.

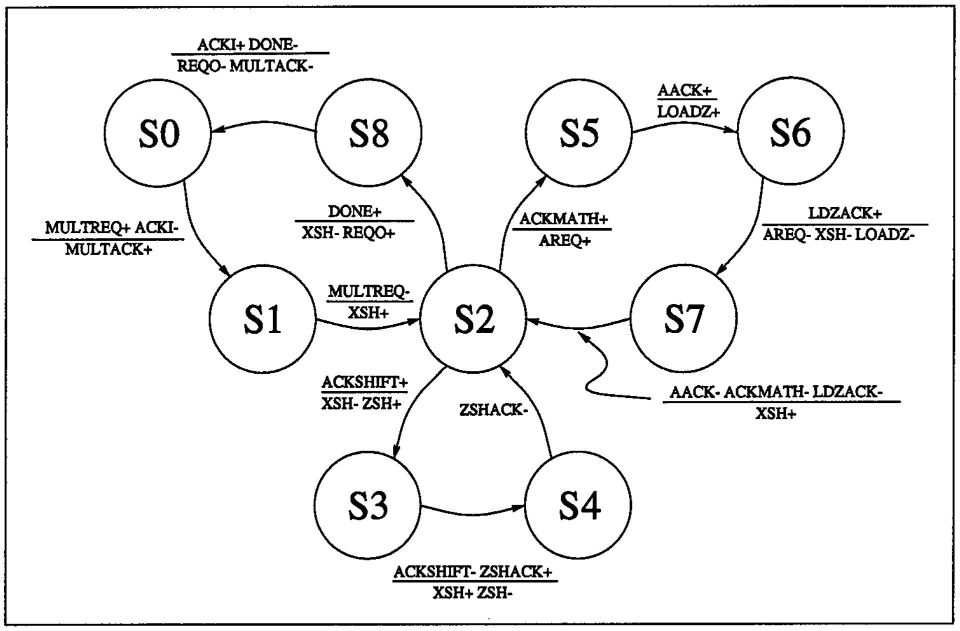


Figure 5.22 Multiplier State Machine

Arithmetic Logic Unit. The ALU here in the real multiply is very similar to that of the FFT-4 element. The only difference is that it is being fed by two dedicated registers so there is less setup time required to begin an operation. Some delay is still required because the X1X2 and FNCT bits may change around the same time the ACKMATH signal is raised which still requires a delay element on the AREQ signal. As was stated earlier, there is no need to shift following an addition or subtraction because the sum/difference lines of the adder are already hardwire shifted by two.

Product Register. The product register or shift register is one of the most unique items in the real multiplier. It is implemented as a hardwire shift by two element using a two phase pulse. The two phase pulse is generated using cross-coupled two input NOR gates with an RC network delay element. Figure 5.24 shows the implementation.

The unique part stems from the usage of a static shift cell implemented with the 3-2 AND-OR-INVERT gate of Figure 5.25 where the FB signal must turn on and off at precisely the right time so the hold value of the register is not lost. This is accomplished by

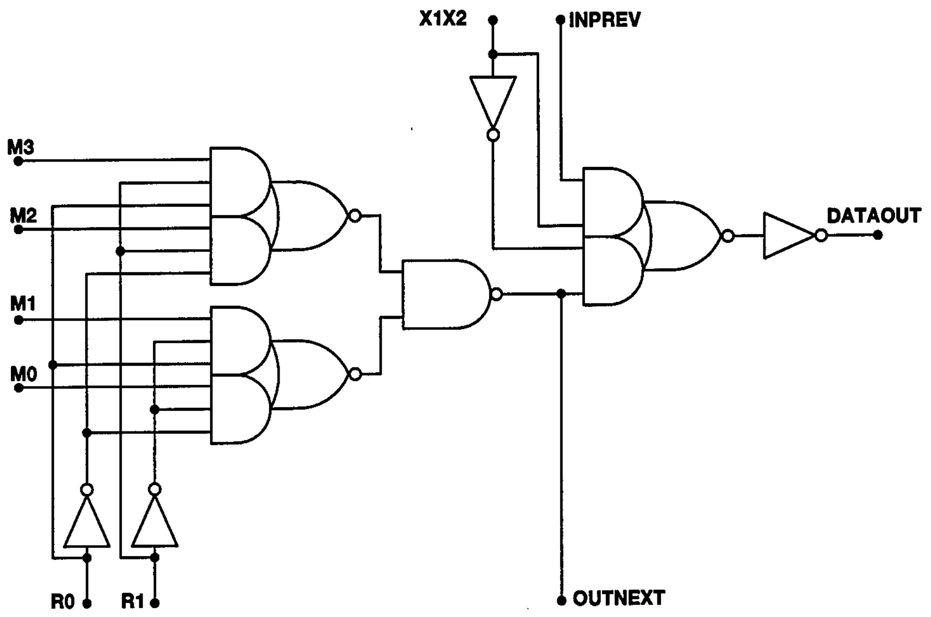


Figure 5.23 Multiplicand Regsiter Cell

an INVERT-NAND structure fanning out to two 1X2 inverting drivers for the 21 segments of the shift register.

There are a few advantages in using this implementation instead of the four tristate inverters or tristate buffers. The first of these is the size. This static implementation uses about 40% less area than the dynamic shift register implementation. There is no need to distribute $\overline{\text{PHI1}}$ or a CLR signal to all the registers since the $\overline{\text{CLR}}$ input into the FB NAND GATE can clear out the AOI222 and there is only the positive input on PHI1 (the SHIFTIN enable) because it is not driving a tristate device. The PHI2 and $\overline{\text{PHI2}}$ are both required because a tristate device is necessary to isolate the SHIFTOUT from the hold value of the register to prevent a shifting value from propagating beyond its designated cell.

Because this shift register uses static storage, the "hold" node can be distributed to its outputs without tri-state buffering. This eliminated the need for a read enabling signal to be distributed to each stage but it also led to a problem with the latching for the input from the ALU and the output to the ALU. The only other disadvantage is the power consumption. Because this is a static element, the nodal voltage is constantly charged with intermediate results.

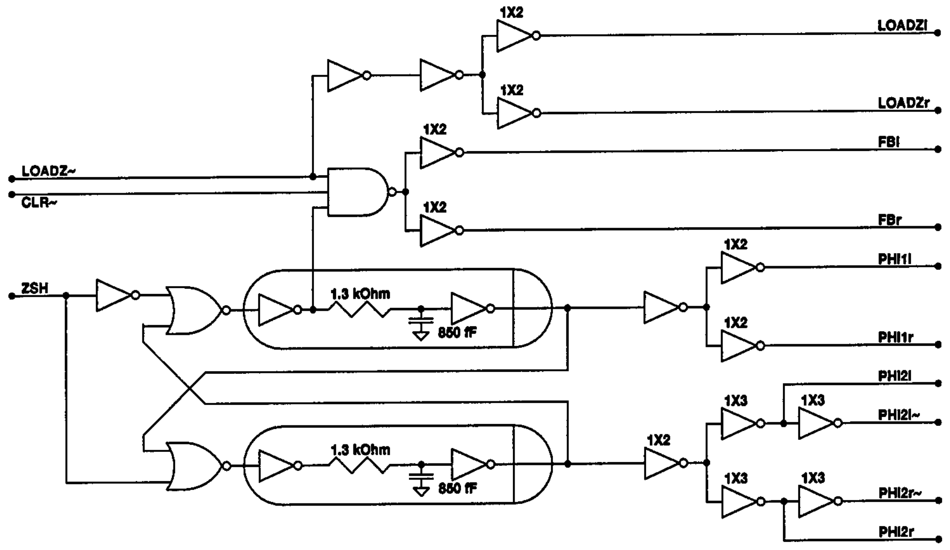


Figure 5.24 Two Phase Pulse Generator

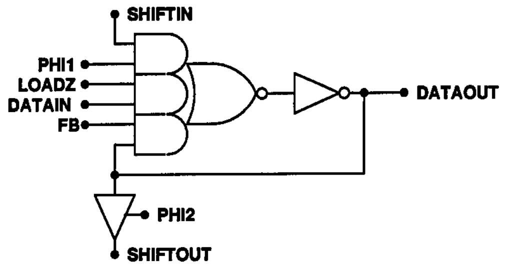


Figure 5.25 Product Register Bit

5.3.2 Complex Multiply Completion. Following the two pairs of simple multiplies, there is still more to do. Table 5.3 shows the step by step operation of the complex multiply operation and how each step corresponds to the data path depicted in Figure 5.26.

As shown in Figure 5.26, each output of the multiplier[2] block connects to one A and B input of an adder and subtracter. Each ALU-A input contains a latch to store the integer product of the first two multiplications as described in step 2 of Table 5.3. The second multiplication result can be passed directly to the adder/subtracter and used with the latched value to produce the complex-valued result. Figure 5.26(a) shows the circuit after the arrival of Re$\{X\}$. The first two partial products have been computed and latched in Figure 5.26(b). Since the FFT-4 will likely produce its outputs faster than the multiplier

Table 5.3 Complex Multiplication Operation Steps

Step	Operation	Corresponds to ...
1	Receive Re$\{X\}$ from FFT-4	Figure 5.26 (a)
2	Multiply Re$\{X\}$ by Re$\{Y\}$ and Im$\{Y\}$ creating the two partial products $XrYr$ and $j * XrYi$	Figure 5.26 (b)
3	Receive Im$\{X\}$ from FFT-4	Figure 5.26 (b)
4	Multiply Im$\{X\}$ by Re$\{Y\}$ and Im$\{Y\}$ creating the two partial products $j * XiYr$ and $XiYi$	Figure 5.26 (c)
5	Subtract $XiYi$ from $XrYr$ to produce Re$\{Z\}$ and add $j * XrYi$ and $j * XiYr$ to produce Im$\{Z\}$	Figure 5.26 (d)

can use them, Im$\{X\}$ will probably arrive early. Despite this, Im$\{X\}$ will not be used until after it is latched in step 3 of Table 5.3. The second multiplication pair has completed in Figure 5.26(c) and is held statically on the data lines. The final step of Table 5.3 occurs when all four integer multiplication products are present, and the final complex products can be computed and latched into the crossbar switch, as shown in Figure 5.26(d).

The control path for this computation is shown in Figure 5.27. Note how the REQOUT signal from the real multiply is fed into the decimate-by-two AFSM. On the first REQOUT signal, the Dec-By-2 write-enables the register elements and "requests" the ALUs on the second REQOUT. The ACKIN signal can come from either the ALU-A latches or the crossbar element. Because the two signals are mutually exclusive, they too can simply be ORed together to return to the multiplier[2]. The last item in the complex control path, found in Figure 5.28, is the multiplicand counter AFSM. Since there are four constant values used in a 16 point FFT calculation, each time the AREQ signal is raised (in the context of Figure 5.27 only), the register constant is incremented by one.

5.4 Pipelined Crossbar Switch

The pipelined crossbar switch is very big in terms of I/O since all the partially transformed data will pass through it during operation. The one-hot encoded sequences work very well in the crossbar switch. The petri-net scheme (Figure 4.3) of the crossbar shows eight loops for the 4-by-4 crossbar switch. The four horizontal loops use the global one-hot in a similar manner to the input stage of the FFT-4. The four vertical loops use yet another implementation of the ripple one-hot encoded AFSM. This usage is the same

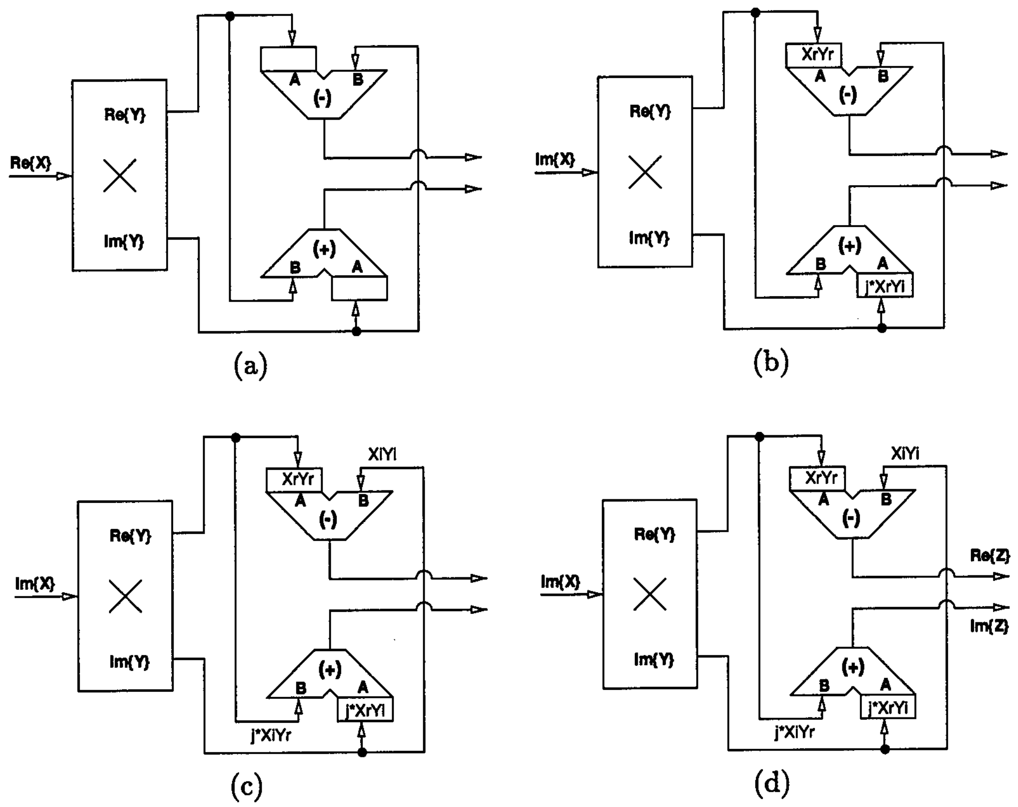


Figure 5.26 Complex Multiplier Data Path Operation

as the one in the intermediate and output stages of the FFT-4 without the context setup. The simpler state transition diagram and block diagram is given in Figure 5.29. The four horizontal loops use the same global one-hot configuration used in the input stage of the FFT-4.

We will first examine the horizontal loops of the crossbar switch, pictured in Figure 5.30, since they are the first encountered by the multiplier outputs. As the last paragraph mentioned, the one-hot AFSMs used here are similar to the input stage of the FFT-4. Note how the latching of a 32-bit value occurs largely the same way as the multiplier receives a 16-bit value. The only difference lies in the addition of the inverted feedback from the loader INTREQ line back to an AND gate with the REQIN signal from the multiplier. The loader with feedback to the multiplier request signal is present as a pipeline latch so the multiplier can produce outputs in parallel with the previous results being forwarded across the crossbar switch to the FFT-4 elements. This latching is essential to the con-

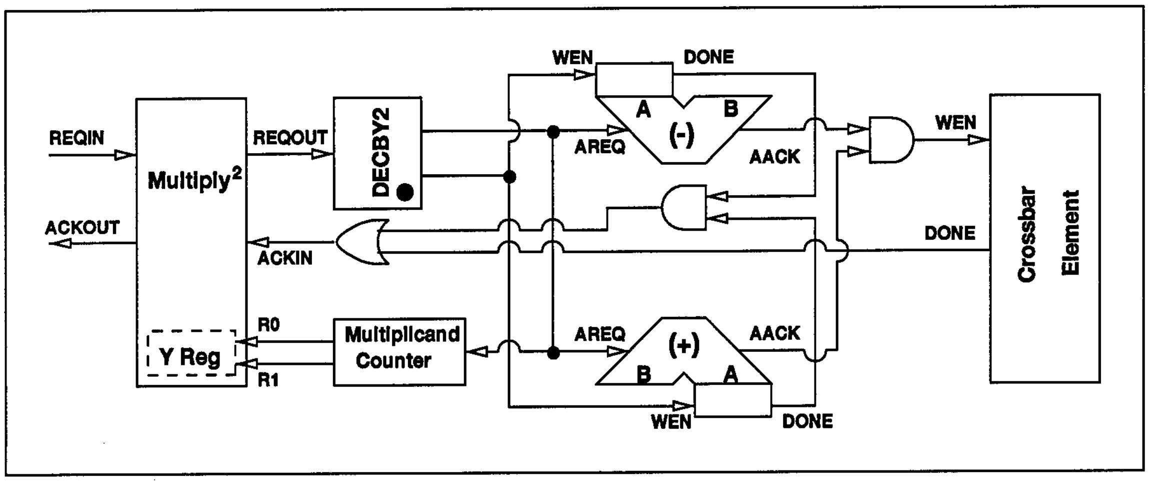


Figure 5.27 Complex Multiplier Control Path

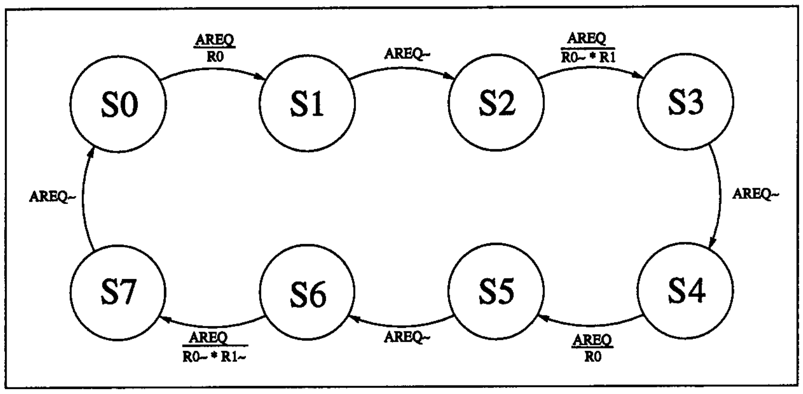


Figure 5.28 Multiplicand Counter State Machine

currency in the overall FFT computation. After one value latches through the HOLD state machine (to be discussed in Section 5.5), the INTREQ signal may be raised but the crossbar row may not immediately raise the INTACK signal if it is waiting for a token. Without the inverted INTREQ feedback, the next REQIN rise will start the loading cycle before the previous value gets used. This is an unwanted condition that would ruin the FFT-16 computation.

Each register has four read ports, one to drive each column in the crossbar, thus more evenly dividing the tri-state buffering and wired-ORs. Note how the LREQ and PREQ signals must be high to read enable the proper read port. This is essentially saying that both the horizontal and vertical tokens are necessary to move data through the switch. Here we can see how the horizontal loops advance. When the vertical ACKIN

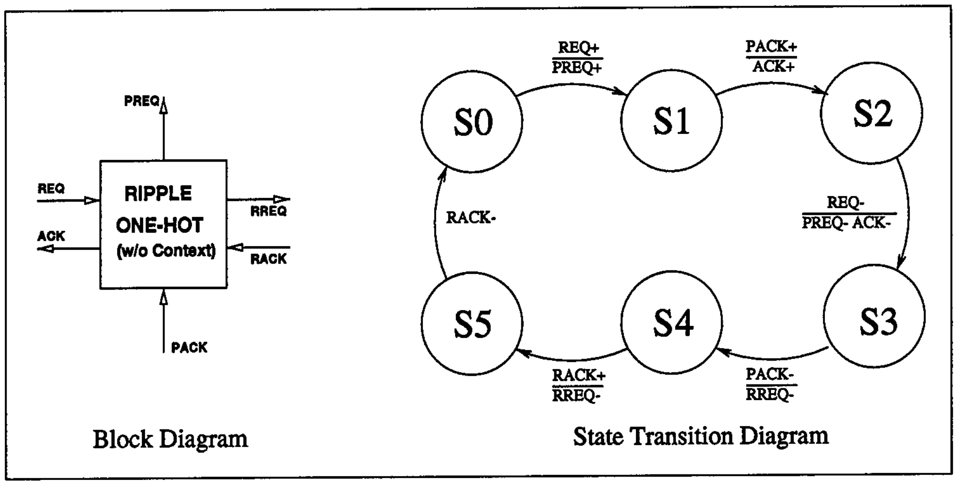


Figure 5.29 Ripple One-Hot AFSM Without Context

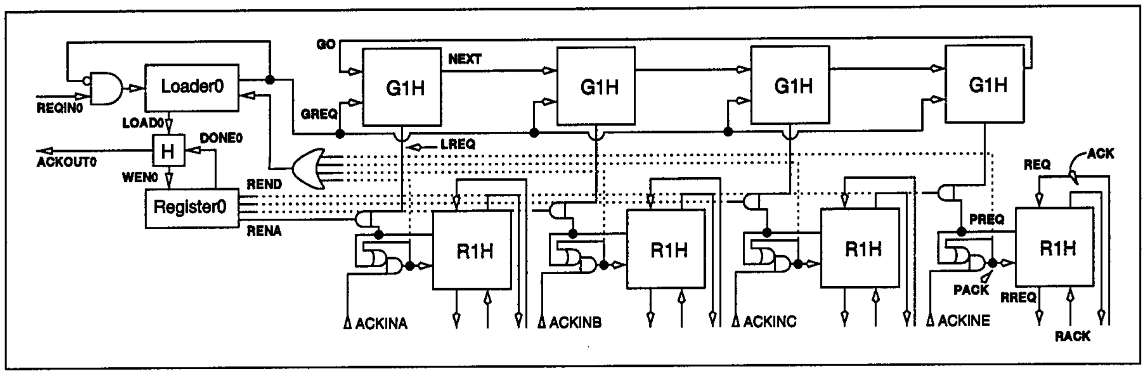


Figure 5.30 Horizontal Crossbar Switch Loop

signal arrives, it will assert the appropriate PACK and INTACK signals (governed by the OR-AND structure). Since the acknowledgement from the FFT-4 blocks will be mutually exclusive in any row, all of the ACKIN signals (routed through the OR-AND structure) can be ORed together to form the INTACK signal returning to the loader AFSM.

The vertical loops are somewhat less complicated because less function is necessary. The first PREQ in each loop is high upon a system reset. This effectively means that all the tokens are waiting in the top of the loop. Figure 5.31 contains the simplified vertical crossbar switch loop. Upon receiving an INTREQ from one of the horizontal loops, the PREQ and LREQ signals are ANDed together and routed down to a four-input OR gate. This constitutes the REQOUT signal to the FFT-4. When the FFT-4 responds with an ACKIN, it is routed to back to each of the ripple one-hot machines. Only the active one-hot will be acknowledged using the OR-AND gates allowing the loop to proceed to the next token stop.

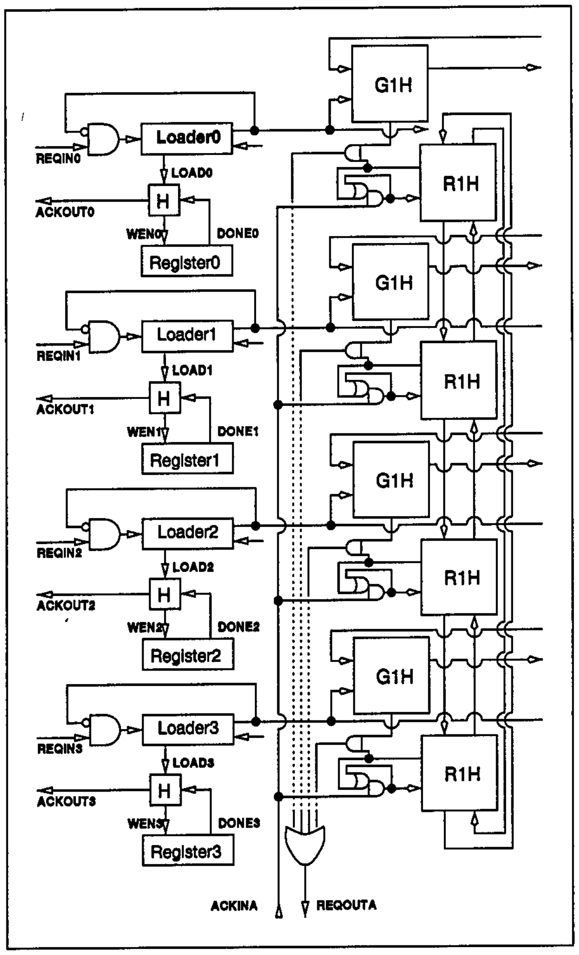


Figure 5.31 Vertical Crossbar Switch Loop

5.5 Expander

The expander is the last major block in the computation of the FFT-16. The main functional portion of the expander consists of the last implementation of the one-hot encoded sequence. This loop of four blocks in the middle of Figure 5.32 is the same ripple sequence as used in the vertical loops of the crossbar switch.

Figure 5.32 shows two new state machines that are used to latch the incoming data from each FFT-4 element. For simplicity, only one of the four stages will be examined.

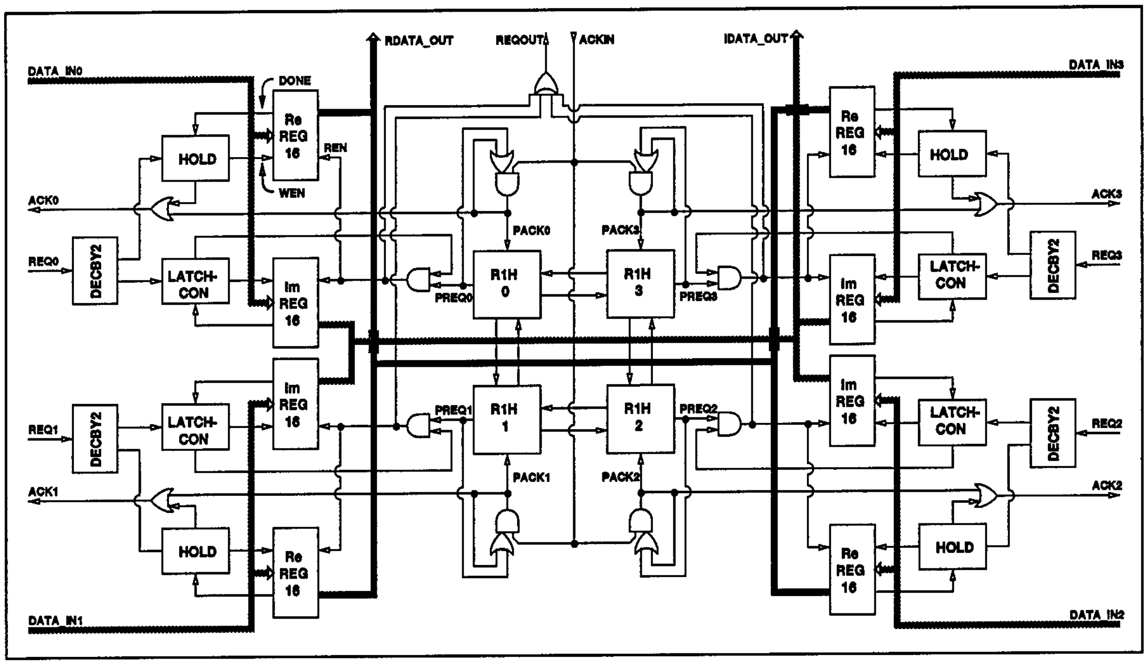


Figure 5.32 Expand-By-Four Block Diagram

A REQ0 comes in from FFT-4_0. The decimator then requests the hold state machine (Figure 5.33) which will raise WEN going to the real register, wait for DONE to assert, lower the WEN signal, wait for DONE to deassert, and finally raise the ACKOUT signal. This order of operations ensures that the register will be write-enabled only long enough to get a value locked in so there will be no incorrect data latched due to changing bus values.

Similarly, the latchcon state machine ensures the imaginary value is locked before proceeding to relinquish the token to the one-hot chain. The latchcon state machine receives the request from the decimator and raises WEN for the imaginary register. It will then wait for DONE to be raised at which time latchcon will lower WEN. When DONE lowers, indicating the value is locked, REQOUT will be raised, effectively passing the token to the one-hot chain.

To advance the one-hot chain, both the REQOUT from the latchcon state machine and the corresponding PREQ signal must be high. This combination will read enable both the real and imaginary registers and raise the REQOUT signal for the whole FFT-16.

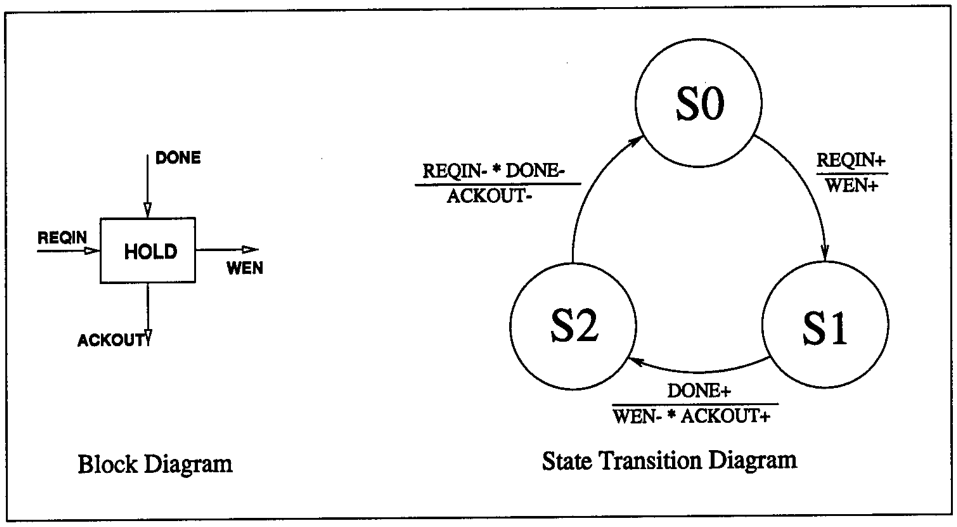


Figure 5.33 Hold State Machine

When the ACKIN is recieved, the token advances to the next stage and the current stage resets.

5.6 Additional FFT-16 Elements

This section includes a description of each of the elements required to reuse the FFT-4 elements and eliminate the need for a Multiplier$_0$ (defined in 5.6.4). The motivation behind the design and use of these components is that the area used by the FFT-4 is quite large compared to everything else in the FFT-16. By designing and implementing the following components, the four FFT-4 blocks can be reused adding slightly to the timing and control overhead but saving tremendously on area. Given the dataflow nature of the FFT-4, input data can be received while it is still outputing values to the multiplier so there is no stalling in overall FFT-16 pipeline. This feature is something that should be kept in mind when designing the new FFT-4 element mentioned in Section 7.2.1.

Figure 5.35 demonstrates how each component in this section fits around the FFT-4 to enable its reuse.

5.6.1 Input Multiplexor. The input multiplexor must allow the proper inputs to enter the FFT-4 element. For the first part of operation, the inputs come from outside

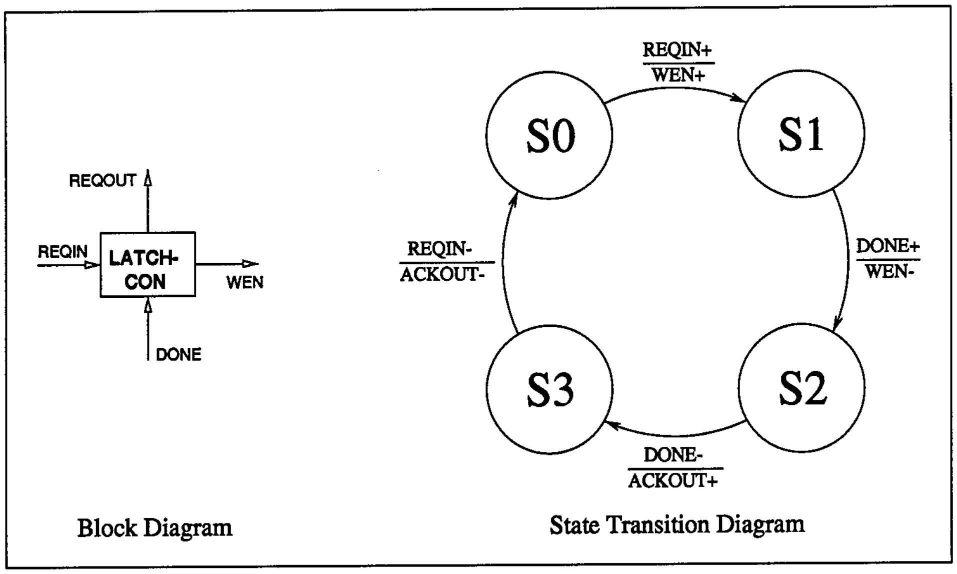


Figure 5.34 Latchcon State Machine

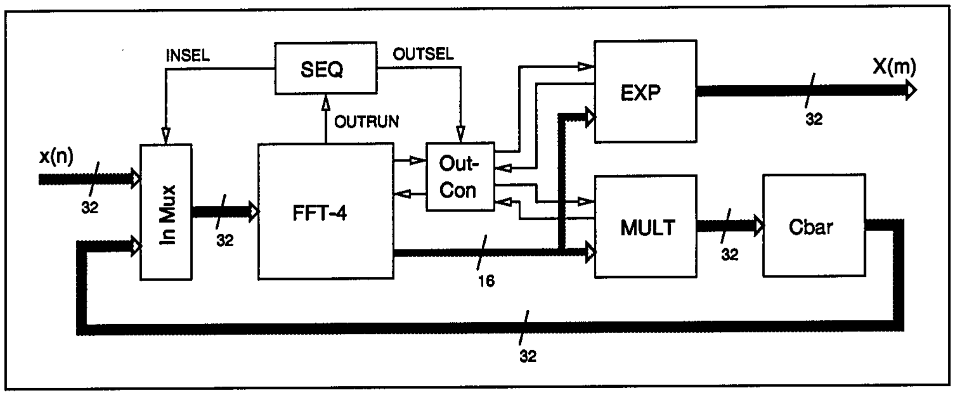


Figure 5.35 Additional FFT-4 Reuse Components

the FFT-16. For the second part of operation, the inputs come from the crossbar switch. Since the asynchronous signaling allows values to propagate through the Multiplier$_0$ and the crossbar switch so quickly, it is necessary that the multiplexor transition rapidly after all of one set of inputs has been received. This is accomplished using the sequencer discussed in Section 5.6.2. The input multiplexor also properly routes the input handshaking signals of the FFT-4 block.

5.6.2 Sequencer. The sequencer is responsible for selecting the proper input and output blocks of the FFT-4 element. It keeps track of the FFT-4's operation stage by monitoring the OUTRUN signal. The block diagram and state transition diagram are in Figure 5.36. The INSEL output is the selector for the input multiplexor (Section 5.6.1)

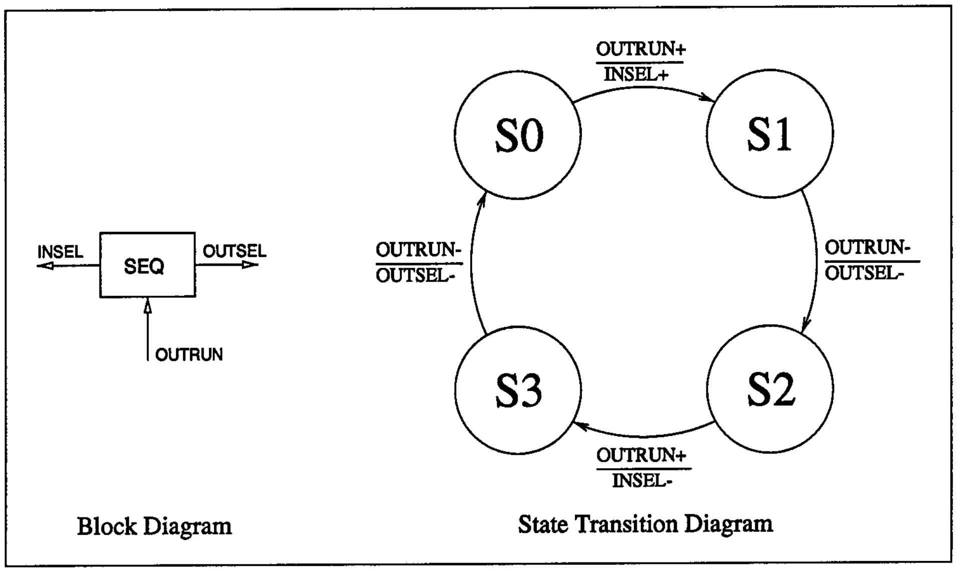


Figure 5.36 Sequencer State Machine

and OUTSEL is the selector for the output control unit (Section 5.6.3). Initially, the INSEL bit selects the FFT-16 input to the FFT-4 block and the OUTSEL bit selects the complex-valued multiplier as the output. As soon as OUTRUN is raised, the input is switched from the FFT-16 input to the crossbar switch output. When OUTSEL lowers, indicating the first set of outputs is done, the Output selector switches to the expander from the multiplier. The last half of the sequencer cycle brings the INSEL and OUTSEL bits back to the original settings.

5.6.3 Output Control Selector. The output controller is simply a grouping of a 1:2 decoder and an OR gate. As a request signal propagates from the FFT-4 block, the output control element will route it to either the multiplier or the expander, depending on the condition of the OUTSEL bit. Since the acknowledgements from the multiplier and the expander will be mutually exclusive, they are simply ORed together to go back into FFT-4. Figure 5.37 shows a simple block diagram of the output control unit.

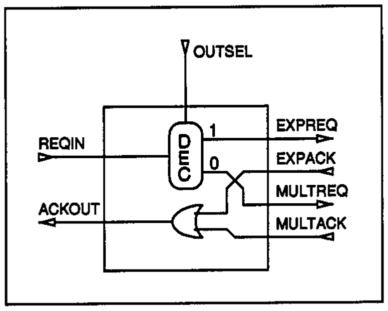


Figure 5.37 Output Control Selector

5.6.4 16:32 Latch. Refering to Section 4.1.2, Equation 4.2 it is apparent that all the values used in the first multiplier (Multiplier_0) are $1 + \jmath 0$. Instead of performing 4 multiplies by 1 (pass a value through), a 16:32 latch is used to compose 32-bit data-words from the separate real and imaginary words. Data is received 16 bits at a time. The bits are first latched into the "real" register followed by the handshake completion with the FFT-4. Then, the second request from the FFT-4 is forwarded to the crossbar switch along with the full 32 bits of data, just like the complex-valued multiplier.

5.7 Summary

This chapter has discussed each component necessary to compute the FFT-16 for this project. Chapter 4 discussed the motivation for each one of these designs. In some cases the implementations discussed here were found to be inferior to other designs. These new designs are proposed in Chapter 7 for potential future work.

VI. Results

6.1 Introduction

The original goal of this project was to implement the Suter/Stevens FFT algorithm in silicon using asynchronous *and* low power design techniques to demonstrate that the benefits of both design methodologies could be combined for extreme energy efficiency. When this project gained sponsorship from the Space Vehicles Directorate of the Air Force Research Laboratory, demonstrating the asynchronous architecture could reduce power consumption became the main goal because standard low power design techniques do not generally coincide with a radiation tolerant design.

6.2 FFT-4 Test Chip

A test chip using the design from Chapter IV and most of the implementation of Chapter V was fabricated using the 0.8μm HP foundry with MOSIS design rules. The data-words were only six bits for the real portion and six bits for the imaginary portion.

6.2.1 Design of the Test Chip. The test chip contained an FFT-4 implementation using the four major components discussed in Sections 5.2.1 to 5.2.4. There were slight differences in several parts of the design. A global one-hot sequence that contained internal acknowledgement was used instead of the external acknowledge chain used in the current design. There were some changes with the intermediate stage as well. Instead of having a single state machine, an "ac" AFSM was used in conjunction with a "bd" AFSM. These two state machines were synchrononized with a 1:2 decoder on the FREQ and LACK (last acknowledge) signal and an OR gate on the FACK and LREQ signals. Both and LREQ and LACK were used because there was difficulty in setting up the complete ripple loop CONTEXT and request signals.

Complete working simulations were not attained prior to submittal. However, IRSIM[10] simulations indicated the chip could perform the four input reads and the first ALU operation.

[10]Berkeley IRSIM v. 9.2.

6.2.2 Test Chip Results.

The results of the test chip were very negative. Initial tests gave no response to the input. When an ACKIN signal was asserted, all of the outputs would follow the ACKIN value. Later determinations indicated this was caused by a trace incorrectly placed across GND metal. A laser cut was made across this wire which should have enabled two output computations to execute correctly, but the results were still negative.

6.2.2.1 Reasons for Failure.

There are several potential reasons for why the test chip failed. Some of the results map to known failures, but not all.

Full VHDL with a data path was not completed. Because the entire control path worked properly in behavioral VHDL, it was assumed the data path would work properly as well. The term "data dependence" should have more fervently indicated the need to run complete VHDL simulations prior to lower level design work. Sections 6.2.3.1 and 6.2.3.2 contain discussions on circuit hazards that were detected when the data path was added to the behavioral VHDL.

Line loading was a new concept that did not appear until the four major components of the test chip were connected. As an example, consider a metal trace length L and width W. We know that the time constant (τ) for charging or discharging a node is RC. The trace resistance is $\propto L \div W$ and the trace capacitance is $\propto L \times W$ so

$$\begin{aligned}\tau = RC &\propto \frac{L}{W} \times (L \times W)\\ \tau &\propto L^2\end{aligned}$$

The time constant of the trace increased according to length squared! When the subcomponents of the FFT-4 were assembled, the bounded-delay assumptions that were made before component integration were suddenly invalid.

Although there is no evidence to substantiate this, another possibility is the test chip was not reseting properly. Since the simulation results indicated that part of the chip was functional yet this was not observed raised suspicion that reseting may be the problem.

6.2.3 Design Modifications Following Test Chip Fabrication. Initially, it was thought that all the added complexity of having two intermediate state machines and the internal acknowledgement of the global one-hot sequence along with several other issues caused the incompleteness of valid simulations.

Based on trace loading revelations, much more care was taken to ensure proper drive capability on all lengthy traces in the circuit. The architecture is good for keeping signals local, but the size of the radiation tolerant cells overwhelms the concept of locality in some instances.

The potential reseting problem has not been addressed because this was not discovered until after the test chip returned from fabrication. This will be discussed in Chapter VII.

6.2.3.1 Register Read Enabling in the FFT-4. In Figure 6.1, we see the problem with register read enabling. The incorrect design of the left block diagram corresponds to the solid wave lines. The correct block diagram on the right corresponds to the dashed lines and arrows. With the incorrect design, the ALU input values ***will*** change before the ALU has the opportunity to fully reset. The addition of the AND-OR structure in the top right shifts the transitions of the read enable signals. Now, the current read enable signal remains high to ensure that the ALU input values do not change until the AACK signal has lowered indicating a full reset. Then, the next read enabling sign is not raised until the ALU fully resets ensuring there is no conflict on the data lines.

Executing behavioral VHDL, including the data path, highlighted this error because the AACK signal would go to an 'X' condition even though the correct sum had already been issued (exactly what happened in the test chip IRSIM results). It is interesting to note that the data should have no effect on the AACK signal because of the "stage -1". As soon as AREQ lowers, both carry signals go to zero, raising the $\overline{\text{DONE}}$ signal for that stage which deasserts the AACK signal.

6.2.3.2 Intermediate Register Locking in the FFT-4. Though present, the problem with register locking was not discovered in the test chip simulations because the

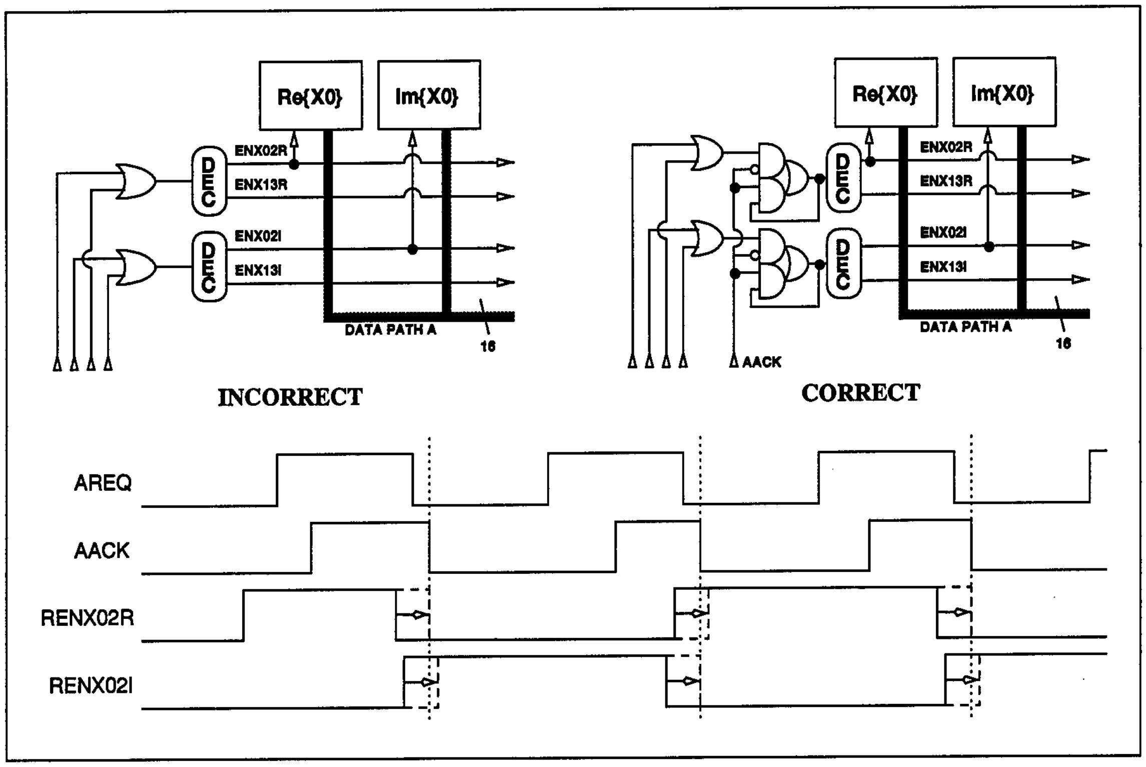


Figure 6.1 Register Read Enabling

original test benches were incomplete. The behavioral VHDL again was the key to discovering that the wrong value was being latched into the intermediate registers. Figure 6.2 contains the incorrect and correct designs along with the transition shift waveform. The key with reseting the PACK signal through the register was so that the ALU completed its reset operation before PACK was lowered. The problem was that when the ALU resets after AREQ is lowered, the outputs begin to change in most cases. The correct design in the upper right still allows the PACK signal to reset through the ALU, but not through the register.

6.2.4 Test Chip Summary. Following the discovery of these design flaws from the behavioral VHDL, the simulation results from the test chip were reexamined, and the errors observed there were similar to the VHDL simulation results before the modifications were made. The conclusion is that, even if the reseting worked properly and the layout error had not been made, the test chip still would not have functioned properly.

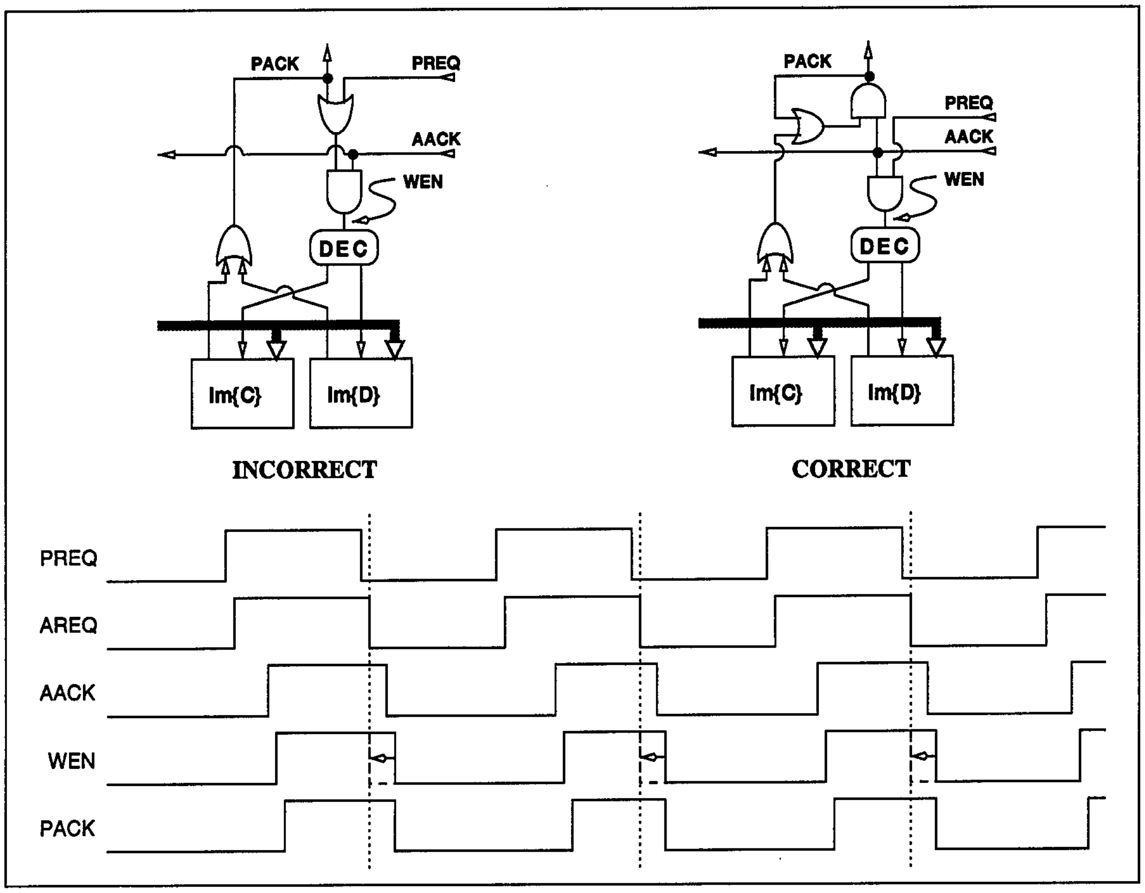


Figure 6.2 Intermediate Register Locking

6.3 16-bit FFT-16 Simulation Results

Some power and timing results based on SPICE simulations on extracted layouts were attained. The AFRL/MRC cell library is designed to be maximally radiation tolerant when Vdd is 5.0 V. However, 2.2V is customary for many of today's low power designs due to the benefits of voltage scaling. The preliminary numbers use a middle-ground Vdd of 3.3 V. VHDL simulations at this voltage have been projected to the system timing chart of Figure 6.3. The results with Vdd of 5.0V and 2.2V have been included along with the baseline of 3.3 V to examine the AFRL/MRC operating range as well as to permit closer comparisons to other low-power FFT chips.

Note how the processing delay of all the major components overlap. The actual amount of overlap (pipelining) will vary depending on the data but this figure gives a good

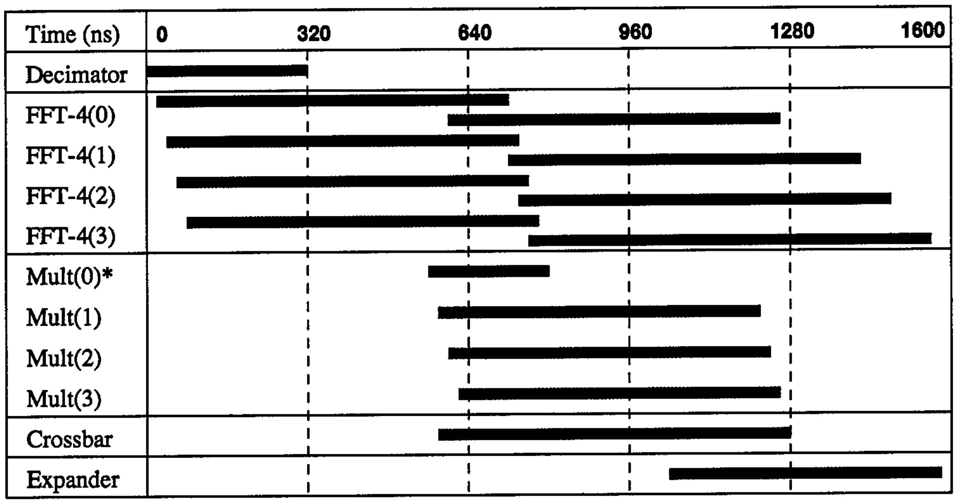


Figure 6.3 FFT-16 System Timing

timing estimate. The asterisk by the Mult(0) line in Figure 6.3 indicates this is not actually a multiplier. The constants for the 0^{th} sequence are all equal to $1 + \jmath 0$ so no multiplication is required. In place of a multiplier, the 16:32 latch discussed in Section 5.6.4 is used so a full 32-bit complex value is sent to the crossbar switch. Based on empirical SPICE data, system and component timing can be extrapolated to Vdd = 5.0 V and Vdd = 2.2 V. Table 6.1 shows the timing comparison between the three Vdd levels.

Table 6.1 Processing Delay by Element (ns)

Vdd	Decimator	FFT-4	Multiplier	Crossbar	Expander	FFT-16
5.0 V	240	480	480	510	390	1200
3.3 V	320	640	640	680	520	1600
2.2 V	531	1062	1062	1129	863	2656

The actual SPICE power numbers for each component running at the frequencies in Table 6.1 can project power consumption for the FFT-16. The component and system power consumption numbers are given in Table 6.2.

Table 6.2 Power Consumption by Element (mW)

Vdd	Decimator	FFT-4	Multiplier	Crossbar	Expander	FFT-16
5.0 V	5.8	182	350	114	74	1076
3.3 V	1.1	45	86	25	18	264
2.2 V	0.6	11	22	9	5	67

In order to compare these results to prior work, we need to extend the timing and power figures to larger point sizes. We know that the time to compute an FFT increases according to $Nlog_2N$. The power and energy computations are a little trickier because the number of components increases along with the size of the decimator, crossbar switch, and expander. Table 6.3 gives the extrapolated computation times and Table 6.4 give the extrapolated energy numbers.

Table 6.3 Extrapolation of Timing Measurements (μs)

Vdd	FFT-16	FFT-32	FFT-128	FFT-256	FFT-1024
5.0 V	1.2	3.0	16.8	38.4	192.0
3.3 V	1.6	4.0	22.4	51.2	256.0
2.2 V	2.7	6.6	37.2	85.0	425.0

Table 6.4 Extrapolation of Energy Consumption (μJ)

Vdd	FFT-16	FFT-32	FFT-128	FFT-256	FFT-1024
5.0 V	1.29	3.29	26.2	68.0	494.9
3.3 V	0.422	1.07	8.4	21.6	153.25
2.2 V	0.178	0.455	3.7	9.9	77.5

Since these numbers are very rough estimates, direct comparisons against current FFT chips are not that conclusive. These comparisons are still drawn to show that, despite using a power hungry cell library and disregarding many known power reduction techniques, similar power efficiency numbers can be achieved with architecture and asynchronous design techniques.

Table 6.5 Energy Efficiency Comparison

Chip	Vdd (V)	Power (mW)	Time (μs)	Energy/unit-transform (nJ)
FASST	5.0	2578	192	483
Plessey PDSP16510A	5.0	3000	98	287
FASST	3.3	599	256	149
Spiffeel	3.3	845	30	24.7
FASST	2.2	182	425	75
Spiffeel	2.5	339	42	13.9

Table 6.5 gives a rough comparison among FASST, the SPIFFEE project at Stanford University, and a commercial FFT processor (The Plessey PDSP 16510A). It is fair to point

out that the Plessey DSP chip uses a block-float data format instead of fixed point which accounts for some of the additional energy required. The figure-of-merit used here, that of energy consumed per unit transform, compares the energy efficiency of the architecture in generating a result and is independent of the frequency of execution. We must also point out that the sample frequency of this asynchronous design is considerable faster than that of any of the comparison processors. The numbers for this project will remain fairly constant for larger point sizes due to the hierarchical nature of our FFT algorithm. However, as the point-size grows, additional hierarchical layers are required which will result in increased power consumption.

The 2.2 V Vdd FASST entry in Table 6.5 probably could not be used in space because of the single event effects discussed in Section 3.2.2. It is presented here to show how the efficiency FOM scales between the different Vdd levels.

6.4 Summary

Despite the multiple test chip failures for only six-bit data, the simulation results for the 16-bit designs and the extrapolations to larger FFT point sizes show a lot of promise. It seems all of the known errors regarding the test chip have been corrected and future designs should work. The recommendations for continuation of this project outline how this project should be continued.

VII. Conclusions and Future Work Possibilities

7.1 Project Conclusions

It was not certain at the beginning of this design that the power reduction originally intended could still be attained using the radiation tolerant standard cells. Even if the simulation results from Chapter VI are off by 10-15%, this project has still demonstrated that an asynchronous architecture alone can limit power consumption to a point competitive with Earth-bound low power devices.

This point alone is reason enough to continue the work in this area. Refering once again to Figure 2.1, it is conceivable, even probable, that the success of this low power architecture will map to high speed designs using low power techniques or even a more complex data type although the size of these circuits would likely become a concern.

7.2 Potential Future Designs

In the course of design, many times the success of an implementation is not determined at the higher levels of design. For instance, it was not determined that the early attempts of the FFT-4 design which reused registers were too complex to use until they were mostly laid out. This became the case with the FFT-4 implementation discussed in Chapters IV and V. It was thought that the division of the control and the data flow would work well but the control complexity became overwhelming in addition to the unwanted addition of data busses for the ALU. These design frustrations lead to a new way to implement the FFT-4.

7.2.1 New FFT-4 Element. Toward the end of this project, a new idea for the FFT-4 element was proposed but little actual work has been done to demonstrate its effectiveness or advantages. When the FFT-4 element that was actually used was finally laid out, it was very easy to see the complexity of control and register files was greater than what was desired. Also, the use of data busses should be avoided because of the constant charging and discharging. The new design would use 16 adder and subtracter units because each one would have dedicated, known inputs. The control would be minimal because the parallelism would be a maximum. The area would not be too much greater than the current design because of the large number of registers and the complexity of the control logic.

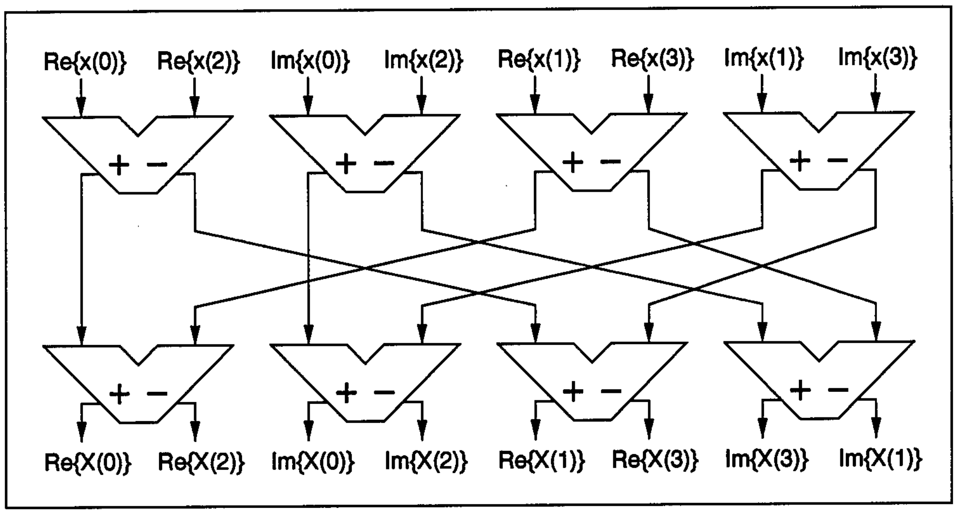


Figure 7.1 Proposed FFT-4 Datapath

The data path of this new implementation could look something like Figure 7.1 where the addition and subtraction operations are separate in each block. There are obviously many ways to actually implement this. One way could reduce the number of ALUs to eight, using each one for one addition and one subtraction. Using static latches in a similar manner to the complex addition and subtraction in the multiplier.

7.2.2 FFT-8 Implementation. For FFT point sizes of 128 points or 1024 points, either an FFT-2 or an FFT-8 implementation will be required. Dr. Suter developed a potential FFT-8 implementation using two complex multiplies up front followed by computations similar to the current FFT-4 design (complex control signaling). From the previous section, it should be understood that these designs implemented with complex control units lead to a lower probability of sucess and are more difficult to design and implement. It is recommended that the FFT-2 implementation of Figure 7.2 be used as the four N_2-point FFTs with the two N_1-point FFTs to be an FFT-4 design.

Additionally, this FFT-8 would require a decimate-by-two, one complex-valued multiplier, a four by two crossbar switch, and an expand-by-four. Each of these designs could

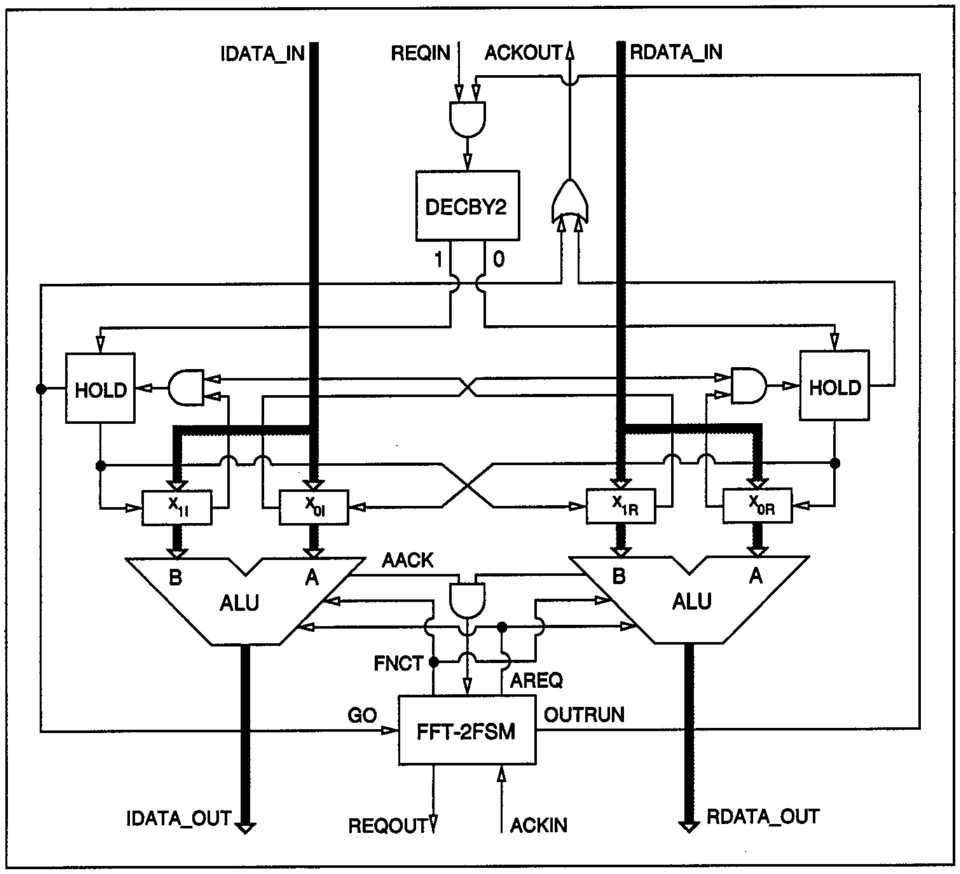


Figure 7.2 FFT-2 block diagram

easily be borrowed from the existing FFT-16 design and modified for the FFT-8. The FFT-2fsm state machine specification is contained in Section A.8 of Appendix A.

7.3 Recommendations for Continuation

This project has a lot of potential for future improvements. When work continues on this project, it is recommended that further work on this project use the FFT-4 presented in Section 7.2.1. This should be built to ensure the full complex data-word becomes available to the multiplier at the same time so the modifications in Section 4.3.3 can be used in conjunction with the 2 cycle dual multiply. Making the entire data word available will also reduce the complexity of the expander making it more area conscience.

Other design changes should be considered for the shift register and booth decoder. Currently, bounded delay models have been built and verified with SPICE. However, there are better ways to implement each design.

Before this project gets too ambitious, it is recommended that a working FFT-16 design be the highest goal of a masters' thesis. The background information on this project, especially the asynchronous logic portion, is very involved and much time should be spent to understand it before moving into the design phase of a thesis.

Appendix A. 3D ASFM Specifications and Boolean Equations

This appendix contains all the burst-mode ASFM specifications used in the FFT-16 design in addition to some extras for future consideration.

A.1 Global One-Hot Specifications

A.1.1 Nominal Stage.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;        Date:  22 July 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  abconehot.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  This is the nominal stage in a global one-hot sequence with
;;;               external acknowledgement.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  ysh     0
input  go      0

output enab    0
output next    0

;;;  Current   Next     Input Burst      | Output Burst
;;;   State    State                     |

        0       1       ysh+             |
        1       0       ysh-             |
        0       2       go+              |
        2       3       ysh+             | enab+
        3       4       ysh- go-         | next+
        4       5       ysh+             | next- enab-
        5       0       ysh-             |

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;  Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; enab =
;   ysh go +
;   ysh' enab +
;   enab $Y00'
;
```

```
; next =
;   ysh' go' enab
;
; $Y00 =
;   next +
;   ysh $Y00 +
;   ysh' go' enab
;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
```

A.1.2 Final Stage.

```
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;;;        Date:  22 July 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  donehot.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  This is the last stage in a global one-hot sequence with
;;;               external acknowledgement.
;;;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------

input  ysh     0
input  go      0

output enab    0
output next    1

;;;  Current   Next    Input Burst      | Output Burst
;;;   State    State                    |

        0       1      ysh+             | next-
        1       2      ysh-             |
        2       4      go+              |
        2       3      ysh+             |
        3       2      ysh-             |
        4       5      ysh+             | enab+
        5       0      ysh- go-         | enab- next+

;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;;;
;;;  Equations:
;;;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;
; enab =
;   ysh go +
;   ysh enab +
;   go enab
;
; next =
;   ysh' go' $Y00'
;
; $Y00 =
;   go' $Y00 +
;   ysh go' enab'
;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
```

A.2 Ripple One-Hot Specifications With Context

A.2.1 Nominal Stage.

```
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;         Date:  17 June 1997
;;;      Version:  1
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  ronehot.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  The function of this component is to function peacefully
;;;               with its rippling neighbors to accomplish real work.
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------

input req 0
input nnext 0
input pack 0
input rack 1

output ack 0
output  context 0
output preq 0
output rreq 0

;;;  Current   Next     Input Burst     | Output Burst
;;;   State    State                    |

       0        1       rack- nnext+    |
       1        2       req+            | preq+
       2        3       pack+           | ack+
       3        4       nnext- req-     | preq- ack- context+
       4        5       pack-           | rreq+
       5        0       rack+           | context- rreq-

;;;------------------------------------------------------------------------
```

```
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; ack =
;   req pack +
;   nnext pack
;
; context =
;   rack' context +
;   req' nnext' pack
;
; preq =
;   req +
;   nnext pack
;
; rreq =
;   pack' rack' context
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.2.2 Final Stage.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;        Date:  15 July 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  dronehot.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The is the last one-hot in a ripple sequence of one-hots
;;;               with context.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input req 0
input nnext 0
input pack 0
input rack 0

output ack 0
output  context 1
output preq 0
output rreq 1

;;;  Current  Next    Input Burst     | Output Burst
;;;   State   State                   |

        0       1       rack+         | context- rreq-
        1       2       rack-           |
        2       3       nnext+          |
        3       4       req+            | preq+
        4       5       pack+           | ack+
        5       6       nnext- req-     | preq- ack- context+
        6       0       pack-           | rreq+

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; ack =
;   req pack +
;   nnext pack
;
; context =
;   rack' context +
;   req' nnext' pack
;
; preq =
;   req +
;   nnext pack
;
; rreq =
;   pack' rack' context
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.3 Ripple One-Hot Specifications Without Context

A.3.1 Nominal Stage.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;         Date:  4 October 1997
;;;      Version:  1
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  onehot3.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  This is the nominal one-hot component in a ripple seqence
;;;               with no context
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input req 0
input pack 0
input rack 0

output ack 0
output preq 0
output rreq 0

;;;  Current   Next     Input Burst     | Output Burst
;;;   State    State                    |

       0        1       req+            | preq+
       1        2       pack+           | ack+
       2        3       req-            | preq- ack-
       3        4       pack-           | rreq+
       4        5       rack+           | rreq-
       5        0       rack-           |

;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;
;;;  Equations:
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;
; ack =
;   req pack
;
; preq =
;   req
;
; rreq =
;   pack' rack' $Y00
;
; $Y00 =
;   req' pack +
;   rack' $Y00
;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
```

A.3.2 Final Stage.

```
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;         Date:  4 October 1997
;;;      Version:  1
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  onehot4.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  This is the final one-hot component in a ripple seqence
;;;               with no context.
;;;
;;;------------------------------------------------------------------------
```

```
;;;---------------------------------------------------------------------

input req 0
input pack 0
input rack 0

output ack 0
output preq 0
output rreq 1

;;; Current   Next     Input Burst     | Output Burst
;;;  State    State                    |

       0       1       rack+           | rreq-
       1       2       rack-           |
       2       3       req+            | preq+
       3       4       pack+           | ack+
       4       5       req-            | preq- ack-
       5       0       pack-           | rreq+

;;;---------------------------------------------------------------------
;;;---------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;---------------------------------------------------------------------
;;;---------------------------------------------------------------------
;
; ack =
;   req pack
;
; preq =
;   req
;
; rreq =
;   req' pack' rack' $Y00'
;
; $Y00 =
;   rack +
;   req' $Y00
;
;;;---------------------------------------------------------------------
;;;---------------------------------------------------------------------
```

A.4 FFT-4

These three AFSM specifications are unique to the FFT-4 element.

A.4.1 Input Stage Control.

```
;;;---------------------------------------------------------------------
;;;---------------------------------------------------------------------
;;;        Date:  17 June 1997
;;;     Version:  3
;;;
;;;      Author:  Bruce William Hunt
;;;
```

```
;;;   Filename:  inputfsm.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;   Function:   The function of this state machine is to govern the
;;;               operation of the input global one-hot sequence that fills
;;;               the input registers and handshakes with the interfsm when
;;;               enough values are present to begin the intermediate
;;;               computations.
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------

input  req    0
input  acack  0
input  bdack  0

output ack    0
output acreq  0
output bdreq  0

;;;  Current   Next     Input Burst     | Output Burst
;;;   State    State                    |

        0        1      req+            | ack+
        1        2      req-            | ack-
        2        3      req+            | ack+
        3        4      req-            | ack-
        4        5      req+            | ack+
        5        6      req-            | ack- acreq+
        6        7      req+ acack+     | ack+ acreq-
        7        8      req- acack-     | ack- bdreq+
        8        9      bdack+ req*     | bdreq-
        9        1      bdack- req+     | ack+

;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;
;;;  Equations:
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;
; ack =
;   req acack +
;   acack ack +
;   req bdack' acreq' bdreq'
;
; acreq =
;   acack' acreq +
;   req' bdack' $Y00 $Y04 $Y05'
;
; bdreq =
;   bdack' bdreq +
;   req' acack' bdack' $Y05
;
; $Y00 =
;   req bdack' bdreq' +
;   bdack' $Y00
;
; $Y01 =
;   bdack +
;   req' $Y01 +
;   req $Y02 +
;   $Y01 $Y02
;
; $Y02 =
;   bdack' $Y02 +
;   req' bdack' $Y00
;
; $Y03 =
;   bdack' $Y03 +
;   req' $Y01 $Y04'
;
; $Y04 =
;   bdack +
;   req $Y03 +
;   req' $Y04 +
;   $Y02 $Y04
;
; $Y05 =
;   req acack +
;   bdack' $Y05
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.4.2 Intermediate Stage Control.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;        Date:  4 August 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  interfsm.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The function of this state machine is to control the
;;;               computation of the intermediate FFT-4 values a, b, c, and d.
;;;
;;;       Notes:  f denotes the first one-hot component and l denotes the
;;;               last one-hot component.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  acreq    0
input  bdreq    0
input  fack     0
input  lreq     1
input  outrun   1
```

```
output acack    0
output bdack    0
output freq     0

;;;  Current    Next     Input Burst           | Output Burst
;;;   State     State                          |

        0         1      acreq+ outrun-        | acack+ freq+
        1         2      lreq- fack+ acreq*    | freq-
        2         3      lreq+ fack- acreq-    | acack-
        3         4      bdreq+                | bdack+ freq+
        4         5      lreq- fack+ bdreq*    | freq-
        5         6      lreq+ fack- bdreq-    | bdack-
        6         0      outrun+               |

;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;
; acack =
;   acreq outrun' +
;   fack acack +
;   lreq' acack +
;   acack freq
;
; bdack =
;   bdreq +
;   fack acack' +
;   lreq' acack' +
;   bdack freq
;
; freq =
;   fack' freq +
;   lreq freq +
;   bdreq fack' $Y00' +
;   acreq fack' outrun' $Y01
;
; $Y00 =
;   bdreq $Y00 +
;   fack $Y00 +
;   lreq' $Y00 +
;   fack lreq' acack'
;
; $Y01 =
;   outrun +
;   fack' $Y01 +
;   lreq $Y01 +
;   bdreq fack' $Y00'
;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
```

A.4.3 Output Stage Control.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;         Date:  15 July 1997
;;;      Version:  2
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  outputfsm.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  The function of this state machine is to control the
;;;               computation of the 8 output values and the handshaking
;;;               with the next stage of the FFT-16 (the multiplier or the
;;;               expander).
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  bdack  0
input  ack    0
input lreq    1

output outrun 0
output req    0

;;;  Current   Next     Input Burst     | Output Burst
;;;   State    State                    |

        0        1       bdack+         |
        1        2       bdack-         | req+ outrun+
        2        3       ack+           | req-
        3        4       ack- lreq-     |
        4        0       lreq+          | outrun-

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; outrun =
;   ack +
;   lreq' +
;   $Y00 +
;   bdack' $Y01
;
; req =
;   bdack' ack' $Y01
;
; $Y00 =
;   ack +
;   lreq $Y00
;
```

```
; $Y01 =
;   bdack +
;   ack' $Y01
;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
```

A.5 Multiplier

These two AFSM specifications are unique to the multiplier.

A.5.1 Multiplier Control.

```
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;        Date:  24 July 1997
;;;     Version:  2.03
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  multfsm.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The function of this state machine is to control the real
;;;               integer multiply process.  The multfsm interfaces with the
;;;               loader, x register, booth decoder, ALU, shift register, and
;;;               the complex control unit (a decby2).
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------

input  multreq  0
input  ackshift 0
input  ackmath  0
input  aack     0
input  ldxack   0
input  xshack   0
input  done     0
input  acki     1

output multack  0
output ysh      0
output xsh      0
output areq     0
output loadx    0
output reqo     0

;;;  Current   Next     Input Burst          | Output Burst
;;;   State    State                         |

        0        1      multreq+ acki-       | multack+
        1        2      multreq-             | ysh+
        2        3      ackshift+            | ysh- xsh+
        3        4      ackshift- xshack+    | ysh+ xsh-
        4       2       zshack-                 |
        2       5       ackmath+                | areq+
        5       6       aack+                   | loadz+
        6       7       ldzack+                 | areq- ysh- loadz-
        7       2       aack- ackmath- ldzack-  | ysh+
        2       8       done+                   | ysh- reqo+
        8       0       acki+  done-            | reqo- multack-

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
;
; multack =
;   done +
;   multreq acki' +
;   acki' multack
;
;  ysh =
;    ackshift' zshack +
;    ackshift' ldzack' done' ysh +
;    multreq' ackshift' ackmath' aack' ldzack' done' acki' multack zsh' reqo'
;
; zsh =
;   ackshift +
;   zshack' zsh
;
; areq =
;   ackmath ldzack' ysh
;
; loadz =
;   aack ldzack' ysh
;
; reqo =
;   done +
;   ackin' reqo
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.5.2 Multiplicand Counter.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;         Date:  12 August 1997
;;;      Version:  1
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  ycount.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  The function of this state machine is to increment the
;;;             constant used from the multiplicand register during an
;;;             FFT-16 operation.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  areq    0

output r0      0
output r1      0

;;;  Current   Next     Input Burst     | Output Burst
;;;   State    State                    |

        0       1       areq+           | r0+
        1       2       areq-           |
        2       3       areq+           | r0- r1+
        3       4       areq-           |
        4       5       areq+           | r0+
        5       6       areq-           |
        6       7       areq+           | r0- r1-
        7       0       areq-           |

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; r0 =
;   areq' r0 +
;   areq $Y00' $Y01' +
;   r0 $Y00' $Y01'
;
; r1 =
;   areq' r1 +
;   r1 $Y00' +
;   areq $Y01
;
; $Y00 =
;  areq $Y00 +
;   r0 $Y00 +
;   areq' r0 r1
;
; $Y01 =
;   areq $Y01 +
;   r0 $Y01 +
;   areq' r0 r1'
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.6 Smaller, General AFSMs

These state machines are used in a variety of applications throughout the FFT architecture. The expand-by-two is not actually used but is included since it is mentioned in Section 4.3.5.

A.6.1 Decimate-By-Two.

```
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;        Date:  12 August 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  decby2.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The function of this state machine is to decimate an input
;;;               handshake sequence by two.
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------

input  reqin   0

output req0    0
output req1    0

;;;  Current    Next     Input Burst       | Output Burst
;;;   State     State                      |

        0        1       reqin+            | req0+
        1        2       reqin-            | req0-
        2        3       reqin+            | req1+
        3        0       reqin-            | req1-

;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;
; req0 =
;   reqin $Y00'
;
; req1 =
;   reqin $Y00
;
; $Y00 =
;   reqin $Y00 +
;   reqin' $Y01 +
;   $Y00 $Y01
;
; $Y01 =
;   req0 +
;   reqin $Y00' +
;   reqin' $Y01
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.6.2 Loader.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;        Date:  13 September 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  loader.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The function of this state machine is to perform stage
;;;               latching of input data to allow pipelined-like execution.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  extreq  0
input  intack  1

output loadval 0
output intreq  0

;;;  Current   Next     Input Burst      | Output Burst
;;;   State    State                     |

        0       1       extreq+ multack- | loadval+
        1       2       extreq-          | loadval- intreq+
        2       0       intack+          | intreq-

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; loadval =
;   extreq intack'
;
; intreq =
;   extreq' intack' $Y00
;
; $Y00 =
;   extreq intack' +
;   intack' $Y00
;
;;;-----------------------------------------------------------------------
```

```
;;;------------------------------------------------------------------------
```

A.6.3 Hold.

```
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;         Date:  6 October 1997
;;;      Version:  1
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  hold.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  The function of this state machine is to write enable a
;;;               16 bit register only long enough for the new value to be
;;;               written to prevent the incorrect value from being locked.
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------

input  wenin  0
input  done   0

output wenout 0
output ack    0

;;;  Current   Next     Input Burst      | Output Burst
;;;   State    State                     |

       0        1       wenin+           | wenout+
       1        2       done+            | wenout- ack+
       2        0       wenin- done-     | ack-

;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;
; wenout =
;   wenin done' ack'
;
; ack =
;   done +
;   wenin ack
;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
```

A.6.4 Latch Controller.

```
;;;------------------------------------------------------------------------
```

```
;;;-----------------------------------------------------------------------
;;;        Date:  3 October 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  latchcon.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The function of this state machine is to ensure that a
;;;               a value has been fully latched into a latch or register
;;;               before it will be read by a later operation in a sequence.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  latchin  0
output latchout 0

output loadt  0
input  loadtack  0

;;;  Current   Next     Input Burst      | Output Burst
;;;   State    State                     |

       0        1       latchin+         | loadt+
       1        2       loadtack+        | loadt-
       2        3       loadtack-        | latchout+
       3        0       latchin-         | latchout-

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; latchout =
;   latchin loadtack' $Y00
;
; loadt =
;   latchin loadtack' $Y00'
;
; $Y00 =
;   loadtack +
;   latchin $Y00
;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
```

A.6.5 Expand-By-Two.

```
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;        Date:  5 October 1997
;;;      Version:  1
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  expby2.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  The function of this state machine is to upsample an input
;;;               sequence by two.  The state machine has to take special
;;;               care to order the data properly since the asynchronous
;;;               architecture can allow the 1st element to request before
;;;               the 0th.
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------

input  reqin0  0
input  reqin1  0
input  ackin   0

output ackout0 0
output ackout1 0
output reqout  0

;;;  Current   Next     Input Burst            | Output Burst
;;;   State    State                           |

       0       1       reqin0+ reqin1*         | ackout0+ reqout+
       1       2       reqin0- ackin+ reqin1*  | ackout0- reqout-
       2       3       reqin1+ ackin- reqin0*  | ackout1+ reqout+
       3       4       reqin1- ackin+ reqin0*  | ackout1- reqout-
       4       1       reqin0+ ackin- reqin1*  | ackout0+ reqout+

;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;-----------------------------------------------------------------------
;;;-----------------------------------------------------------------------
;
; ackout0 =
;   reqin0 ackout0 +
;   ackin' ackout0 +
;   reqin0 ackin' $Y01'
;
; ackout1 =
;   reqin1 ackout1 +
;   ackin' ackout1 +
;   reqin1 ackin' $Y01
;
; reqout =
;   reqin0 ackout0 +
;   reqin1 ackout1 +
;   ackin' reqout +
;   reqin1 ackin' $Y01 +
;   reqin0 ackin' $Y01'
;
; $Y00 =
;   ackout1 +
;   reqin0' $Y00 +
;   ackin $Y00 +
;   reqin1 ackin' $Y01
;
; $Y01 =
;   reqin1 $Y01 +
;   ackin' $Y01 +
;   $Y00' $Y01 +
;   reqin0' ackin $Y00'
;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
```

A.7 Sequencer Control

This is the sequencer control to reuse the FFT-4 element in the computation of the FFT-16. Keep in mind this is unique to the current design of the FFT-4 and will not work if modifications are made.

```
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;;;        Date:  6 October 1997
;;;     Version:  1
;;;
;;;      Author:  Bruce William Hunt
;;;
;;;    Filename:  sequencer.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;    Function:  The function of this state machine is to select the proper
;;;               input to and output from the reused FFT-4 blocks.
;;;
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------

input  outrun 0

output insel  0
output outsel 0

;;;  Current   Next     Input Burst      | Output Burst
;;;   State    State                     |

        0        1      outrun+          | insel+
        1        2      outrun-          | outsel+
        2        3      outrun+          | insel-
        3        0      outrun-          | outsel-

;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
```

```
;;;
;;;   Equations:
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;
; insel =
;   outrun' insel +
;   outrun outsel' +
;   insel outsel'
;
; outsel =
;   outrun' insel +
;   outrun outsel +
;   insel outsel
;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
```

A.8 FFT-2 Controller

```
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;         Date:  19 November 1997
;;;      Version:  2
;;;
;;;       Author:  Bruce William Hunt
;;;
;;;     Filename:  fft2fsm.nounc
;;;
;;; Description:  This is the behavioral specification of an asynchronous
;;;               finite state machine designed to be synthesized using 3D.
;;;
;;;     Function:  The function of this state machine is to control the
;;;               computation of the fft-2.  It assumes full latching in
;;;               the expand-by-four for the completion of the FFT-8
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------

input  go     0
input  aack   1
input  ackin  1

output areq   0
output fnct   0
output reqout 0

;;;  Current   Next     Input Burst       | Output Burst
;;;    State   State                      |

        0       1       go+ ackin- aack-  | areq+ outrun+
        1       2       go* aack+         | reqout+
        2       3       go- ackin+        | reqout- areq- fnct+
        3       4       ackin- aack-      | areq+
        4       5       aack+             | reqout+
        5       0       ackin+          | reqout- areq- fnct- outrun-

;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;;;
;;;   Equations:
;;;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
;
; areq =
;   go +
;   ackin' areq +
;   ackin' fnct
;
; fnct =
;   aack' fnct +
;   ackin' fnct +
;   go' ackin $Y00'
;
; reqout =
;   go aack reqout +
;   aack ackin' areq
;
; $Y00 =
;   aack $Y00 +
;   ackin $Y00 +
;   aack ackin' fnct
;
;;;------------------------------------------------------------------------
;;;------------------------------------------------------------------------
```

Appendix B. Project Files

3D	**/projects/fasst/3D**
MEAT	**/projects/fasst/meat**
VHDL	**/projects/fasst/vhdl**
MAGIC	**/projects/fasst/layout**
IRSIM	**/projects/fasst/irsim**
HSPICE	**/projects/fasst/hspice**

Bibliography

1. Baas, Bevan M. "An Energy-Efficient Single-Chip FFT Processor." *Symposium on VLSI Circuits*. 164–165. June 1996.
2. Bewick, Gary and Michael J. Flynn. *Binary Multiplication Using Partially Redundant Multiples*. Technical Report CSL-TR-92-528, Stanford University, June 1992.
3. Coates, William S., et al. "Automatic Synthesis of Fast Compact Self-Timed Control Circuits." *IFIP Working Conference on Design Methodologies*. 193–208. April 1993.
4. Cooley, James W. and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, *19*:297–301 (April 1965).
5. Fan, Xingcha and Neil Bergmann. "Architecture Design of a Fully Asynchronous VLSI Chip for DSP Custom Applications." *International Symposium on Circuits and Systems5*. 2112–2115. 1992.
6. Farnsworth, Craig, et al. "Utilising dynamic logic for low power consumption in asyncrhonous circuits." *International Symposium on Advanced Research in Asynchronous Circuits and Systems*. 186–194. November 1994.
7. Hauck, Scott. "Asynchronous Design Methodologies: An Overview," *Proceedings of the IEEE*, *83*(1):69–93 (January 1995).
8. Heideman, Michael T., et al. "Gauss and the History of the Fast Fourier Transform," *IEEE Acoustics, Speech, and Signal Processing*, *1*:14–21 (October 1984).
9. Hollaar, Lee A. "Direct Implementation of Asynchronous Control units," *IEEE Transactions on Computers*, *C-31*(12):1133–41 (December 1982).
10. Jacobs, Gordon M. and Robert W. Brodersen. "A Fully Asynchronous Digital Signal Processor Using Self-Timed Circuits," *IEEE Journal of Solid-State Circuits*, *25*(6):1526–1536 (December 1990).
11. Jr., Charles. P. Brothers. *Rapid and Accurate Timing Simulation of Radiation-Hardened Digital Microelectronics Using VHDL*. PhD dissertation, Air Force Institute of Technology (AU), March 1994.
12. Jr., Charles P. Brothers, et al. "Radiation Hardeneing Techiniques for Commercially Produced Microelectronics for Space Guidance and Control Applications." *20th Annual American Astronautical Society Guidance and Control Conference*. February 1997.
13. Loan, Charles Van. *Computational Frameworks for the Fast Fourier Transform*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.
14. Mano, M. Morris. *Digital Design*. Englewood Cliffs, NJ: Prentice Hall, 1991.
15. Messenger, George C. and Milton S. Ash. *The Effects of Radiation on Electronic Systems*. New York, NY: Van Nostrand Reinhold, 1992.

16. Morton, Shannon V., et al. "An Event Controlled Reconfigurable Multi-chip FFT." *International Symposium on Advanced Research in Asynchronous Circuits and Sustems.* 144–153. November 1994.

17. SanGregory, Sam L. "A 16-bit Asynchronous Multiplier." An AFIT CSCE-699 Project, June 1996.

18. Sunada, Glen, et al. "COBRA: An 1.2 Million Transistor Expandable Column FFT Chip." *ICCD*. 546–550. 1994.

19. Suter, Bruce W. *Multirate and Wavelet Signal Processing*. San Diego, CA: Academic Press, 1997.

20. Suter, Bruce W. and Kenneth S. Stevens, "Low Energy Consumption, High Performance Fast Fourier Transform." U. S. Patent Number: 08/863,239, May 1997.

21. Suter, Bruce W. and Kenneth S. Stevens. "Low Power, High Performance FFT Design." *IMACS Workd Congress on Scientific Computation, Modeling, and Applied Mathematics*, edited by A. Sydow. 99–104. June 1997.

22. Yun, Kenneth Y. *Synthesis of Asynchronous Controllers for Heterogeneous Systems.* PhD dissertation, Stanford University, August 1994.

Vita

Second Lieutenant Bruce W. Hunt was born on [redacted] in [redacted]. He graduated from high school in Stillman Valley, Illinois in 1992. He then attended Northern Illinois University in DeKalb, Illinois for a period of one year before receiving the wonderful opportunity to attend college in the city of Chicago. From the Illinois Institute of Technology, he received a B.S.E.E. with honors in 1996 and a commission through the Air Force Reserve Officers' Training Corps, Detachment 195 the same year. Upon graduation, he attended the Air Force Institute of Technology to pursue a master of science degree in computer systems.

Permanent address: [redacted]

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE A SINGLE CHIP LOW POWER IMPLEMENTATION OF AN ASYN-CRONOUS FFT ALGORITHM FOR SPACE APPLICATIONS	5. FUNDING NUMBERS
6. AUTHOR(S) Bruce W. Hunt, Second Lieutenant, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/97D-1
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Major Charles P. Brothers Space Vehicles Directorate, Air Force Research Laboratory 3550 Aberdeen Ave SE, Bldg 887 Kirtland AFB, NM 87117-5776	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A fully asynchronous fixed point FFT processor is introduced for low power space applications. The architecture is based on an algorithm developed by Suter and Stevens specifically for a low power implementation. The novelty of this architecture lies in its high localization of components and pipelining with no need to share a global memory. High throughput is attained using large numbers of small, local components working in parallel. A derivation of the algorithm from the discrete Fourier transform is presented followed by a discussion of circuit design parameters specifically, those relevant to space applications. The generic architecture is explained with a survey of the 16 point FFT architecture specific to this project. An implementation, which included a test chip fabricated through MOSIS, is described. Finally, simulation results based on layout extractions are presented and an outline for future work is given.

14. SUBJECT TERMS VLSI, Asynchronous, Rad Tolerant, FFT	15. NUMBER OF PAGES 126
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL