Air Force Institute of Technology

AFIT Scholar

12-1997

FPGA Processor Implementation for the Forward Kinematics of the UMDH

Steven M. Parmley

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Computer Engineering Commons

Recommended Citation

Parmley, Steven M., "FPGA Processor Implementation for the Forward Kinematics of the UMDH" (1997). *Theses and Dissertations*. 5737.
https://scholar.afit.edu/etd/5737

FPGA PROCESSOR IMPLEMENTATION FOR
THE FORWARD KINEMATICS OF THE UMDH

THESIS

Steven M. Parmley

AFIT/GE/ENG/97D-21

DTIC QUALITY INSPECTED 4

19980210 054

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/97D-21

FPGA PROCESSOR IMPLEMENTATION FOR
THE FORWARD KINEMATICS OF THE UMDH

THESIS

Steven M. Parmley

AFIT/GE/ENG/97D-21

FPGA PROCESSOR IMPLEMENTATION FOR

THE FORWARD KINEMATICS OF THE UMDH

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the Requirements for the Degree of

Masters of Science in Electrical Engineering

Steven M. Parmley, B.S.
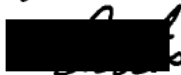
December 1997

FPGA PROCESSOR IMPLEMENTATION FOR

THE FORWARD KINEMATICS OF THE UMDH

Steven M. Parmley, B.S.

Approved:

Major Don Gelosh, Ph.D. Chairman	25 Nov 97 date
Dr. Curtis Spenny	2 Dec. 97 date
Dr. Kuldip Rattan	2 Dec. 97 date
Major Dean Schneider, Ph.D.	3 Dec 97 date
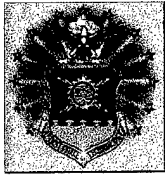
Acknowledgments

I would like to take a moment to thank my thesis advisor, Maj Don Gelosh, and my committee members, Dr. Curtis Spenny, Maj Dean Schneider, and Dr. Kuldip Rattan for their guidance and support during this thesis effort.

A special thanks goes out to my co-workers from Wright Laboratories Avionics Directorate. Capt. Tony Kadrovach was instrumental in helping me narrow the thesis topic. Mrs. Kerry Kill helped me get a grasp with the Exemplar and XACT tools. An extra thanks goes to a truly motivated engineer, Mr. Gary Fecher. Gary's help with the Xilinx hardware proved invaluable.

The Dayton Area Graduate Studies Institute (DAGSI) deserves a cheer for giving me the opportunity to attend AFIT. On the same note, Mrs. Mary Jane McCormick always managed to handle the paperwork with a smile.

Finally, I would like to thank my family for their support and encouragement over the past two years.

<div style="text-align: right;">Steven M. Parmley</div>

Table of Contents

Page

<u>List of Figures</u>

List of Tables

AFIT/GE/ENG/97D-21

Abstract

The focus of this research was on the implementation of a forward kinematic algorithm for the Utah MIT Dexterous Hand (UMDH). Specifically, the algorithm was synthesized from mathematical models onto a Field Programmable Gate Array (FPGA) processor. This approach is different from the classical, general-purpose microprocessor design where all robotic controller functions including forward kinematics are executed serially from a compiled programming language such as C. The compiled code and subsequent real-time operating system must be stored on some form of nonvolatile memory, typically magnetic media such as a fixed or hard disk drive, along with other computer hardware components to allow the user to load and execute the software. With a future goal of moving the controllers to a portable platform like a dexterous prosthetic hand for amputee patients, the application of such a hardware implementation is impossible.

Instead, this research explores a different implementation based on a modular approach of dedicated hardware controllers. The controller for the forward kinematics of the UMDH is used as a test case. The resulting FPGA processor replaces a robotic system's burden of repetitive and discrete software system calls with a stand-alone hardware interface that appears more like a single hardware function call. The robotic system is free to tackle other tasks while the FPGA processor is busy computing the results of the algorithm.

The forward kinematic algorithm for the UMDH was chosen as test case due to its familiarity among the academic community. Although considerable time was spent deriving the equations, the specifics of the UMDH algorithm itself was not the focus of this thesis. Rather, the focus was on the implementation of such an extensive and complex algorithm onto an FPGA processor. Forward kinematic algorithms from other portable robotic devices such as planetary rovers, flight line bomb loaders, or teleoperation systems could have been implemented just as well.

This thesis is divided into three parts. First, the UMDH is examined and the forward kinematic equations for it are developed. This stage will be different for every robotic system, but the process will remain the same. Second, the resulting equations are evaluated for maximum and minimum numeric ranges and amounts of desired precision. This information is used in the third part, where mathematical, memory storage, and controller functional units are developed. Specifically, VHDL models are created, simulated, synthesized, and placed into an FPGA processor.

1. Introduction

1.1   Background

Although robotic devices have been in existence for many years, they were hindered due to the high computational demands until the digital computer revolution came about. Today, highly sophisticated control algorithms are written in software, usually with a real time operating system such as Chimera(Khosla), VX-Works(Wind), or Condor(Narasimhan) and executing on a VME based processor or similar dedicated hardware platform. Each part of the algorithm may be executing concurrently with other parts and may be highly repetitive in nature.

One particular part that is highly repetitive is the calculation of the forward kinematics of the device. The forward kinematics allow the angles of the device to be transformed to the spatial position and orientation of the end of the device. Even a small motion at the base of the device may cause considerable motion farther out on the tip of the device, so the transform must be calculated repetively in order to keep track of the device in Cartesian coordinates.

1.2   Problem Statement

The forward kinematics of the Utah MIT Dexterous Hand (UMDH) (Sarcos) will be developed and implemented on a Xilinx Field Programmable Gate Array (FPGA) (Xilinx). The result is a Forward Kinematic Processor for the UMDH that will autonomously calculate the results while the surrounding system performs more task specific operations.

1.3   Assumptions

Although the process used to calculate the forward kinematics is the same for most common robotic devices, there could exist a device or devices which would not easily map to the algorithms discussed. On example is a parallel linkage device like a bomb loader. It is assumed that the developed algorithm is for the UMDH specifically and that all UMDHs are mechanically identical.

1.4   Approach

The design of the Forward Kinematic Processor starts with the development of the forward kinematic algorithm specifically for the UMDH. This algorithm is evaluated for arithmetic and transcendental properties and arranged such that a minimum amount of hardware time is required. The required arithmetic and transcendental operations lead to the development of functional units to process the numeric data. The functional units are then integrated into one complete processing unit, and synthesized from VHDL code to logic blocks on a Xilinx FPGA.

1.5   Overview

The remaining chapters of this document describe the development and implementation of the Forward Kinematic Processor. Chapter 2 reviews the mathematical foundation of general forward kinematics and applies it to the specific nature of the UMDH. Chapter 3 looks at the results of Chapter 2, particularly the equations for position and orientation, and evaluates them for magnitude constraints, required precision, and operational occurrences. Chapter 4 describes the development of a VHDL model that simulates the digital hardware implementation of an application specific microprocessor that can compute the equations from Chapter 2. Chapter 5

deals with synthesizing the model directly to an Xilinx FPGA. Chapter 6 evaluates the results and

Chapter 7 discusses recommendations and possible future work.
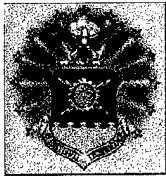
2. Literature Review and Background

2.1 Review

As mentioned in Chapter 1, a typical robotics research environment consists of a real time operating system supported by a relatively large hardware platform. The use of such a system allows researchers to quickly change various parameters of the control structure for robotic devices. Although dedicated hardware may show an increase in performance for a particular application, to build and maintain it is sometimes too much overhead for researchers whose primary focus is robotics, not hardware design (Narasimhan).

The concept of a dexterous prosthetic hand requires a contoller that moves with the device. Obviously, a generalized hardware platform would be much too large to be portable. Such area requirements may necessitate a custom hardware implementation (Narasimhan). With the hopes of a stand-alone dexterous prosthetic hand and the advent and popularity of the FPGA, it is now possible to merge the two technologies and create a truly portable solution. As the controller algorithms in the research laboratory are upgraded, they can be downloaded into the existing hardware of the hand using the reconfigurable properties of the FPGA (Xilinx).

2.2 Introduction

This chapter discusses a method to represent the mechanical attributes of a particular manipulator. This representation is then used to determine the transformation from the relative angles of each link to the 3-dimensional coordinate locations and orientations of the tip of the end link. The process, known as forward kinematics, is then applied to the unique nature of the Utah

MIT Dexterous Hand (UMDH). Specifically, the thumb mechanism of the UMDH is evaluated

and the resulting control equations will form the basis for FPGA implementation in the remaining

chapters.

2.3 Review of Forward Kinematic Computations and the Denavit-Hartenberg Notation (Craig)

In order to represent the mechanical attributes of any general purpose manipulator, a

convention is formulated that will relate the various physical parts that make up the manipulator.

It is composed of rigid links connected by joints to allow for relative motion of the neighboring

links. Most manipulators have joints that are either revolute or prismatic as shown in Figure 2.1.

Revolute joints are typical hinge style joints and the unit of measurement is the joint angle

between the two halves of the joint. Prismatic joints are designed such that one half can slide

back and forth in relation to the fixed half. The measuring unit is the joint offset between the two

halves. Other possible joint configurations include cylindrical, planar, screw, and spherical

(Craig:69).

Link 0 is considered to be the immobile base of the manipulator. Link 1 is the first moving

part, followed by link 2, and so on out to the end link n. The axes of the joints which connect the

links are measured relative to the previous axis. Each joint axis defines a vector in which the next

link in the chain will rotate about. However, the link and its previous joint are given the same

index. This vector is based on the coordinate frame of the previous joint. There are two

quantities to measure the difference between the two axes as shown in Figure 2.2. First, the link

length a_{i-1} is the distance of the line that is mutually perpendicular to both axes. Second, the link
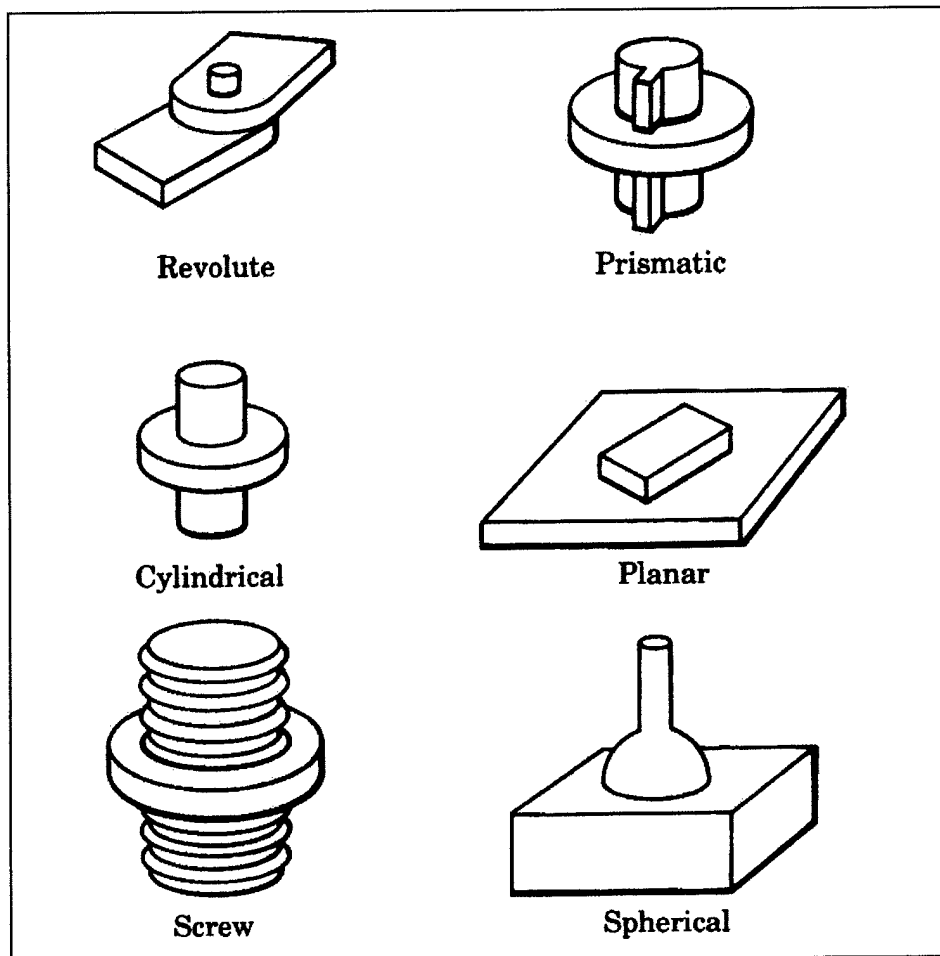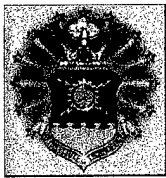
Figure 2.1.   The Six Possible Joints

twist α_{i-1} is the angle between the i-1 axis and a parallel projection of the axis i onto the origin

point of the perpendicular line found earlier.

For links that have a common joint between them, there are two quantities that can be

measured. First, the link offset d_i is the distance between the connection points of the two links

along the axis of the common joint. If this value is zero, then that implies a door like hinge. If the

value is non-zero, then that implies a sort of scissors-like hinge where the two links use the same
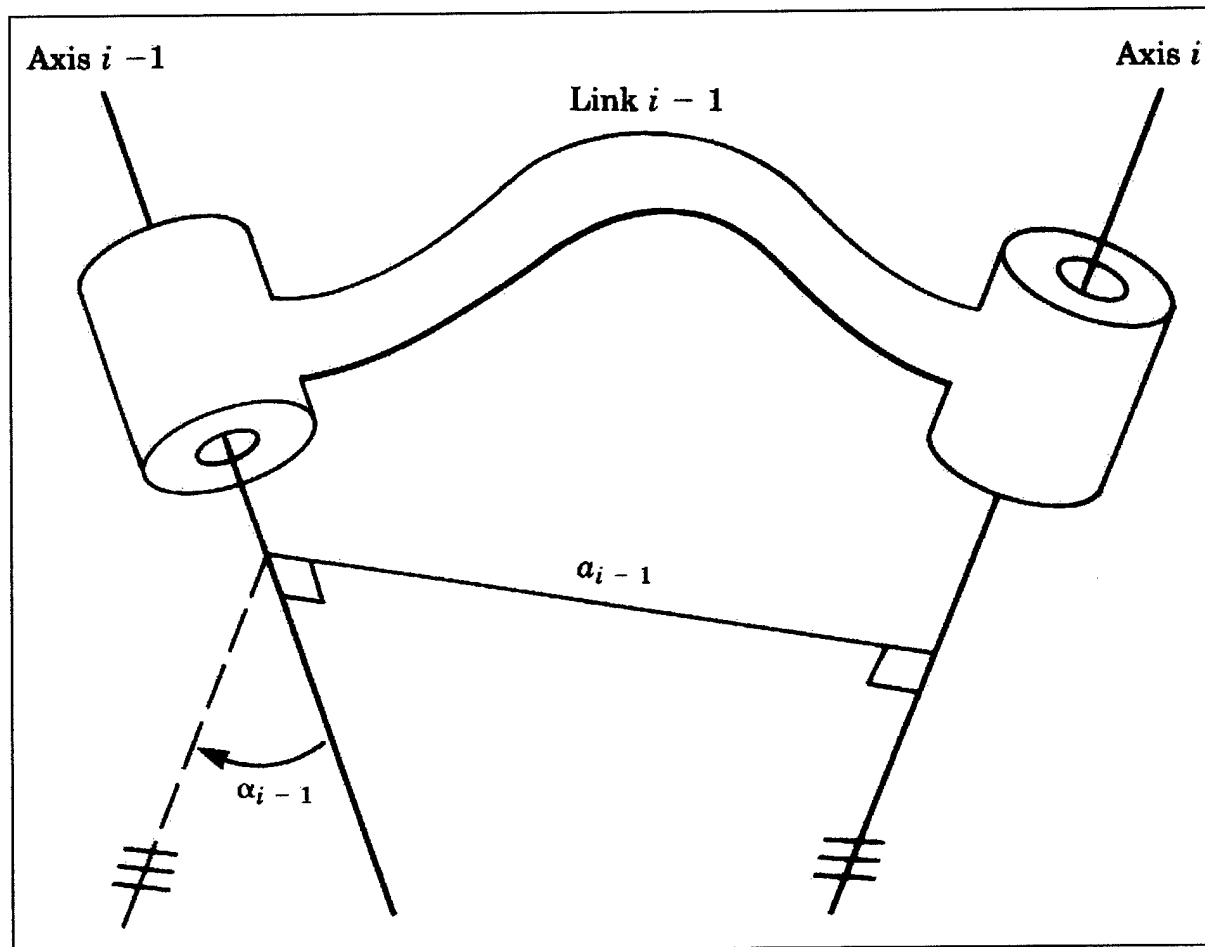
Figure 2.2.   **Link Length and Link Twist**

value is non-zero, then that implies a sort of scissors-like hinge where the two links use the same

joint but are slightly offset from each other. Secondly, the joint angle θ_i is the rotational

difference between the two links about their common joint. These two quantities are shown in

Figure 2.3. If the joint is revolute, then the link offset is fixed and the joint angle will be allowed

to vary. Similarly, if the joint is prismatic, then the joint angle is fixed and the link offset is

allowed to vary. For the first and last links, the fixed quantity will be set to zero (Craig:73).
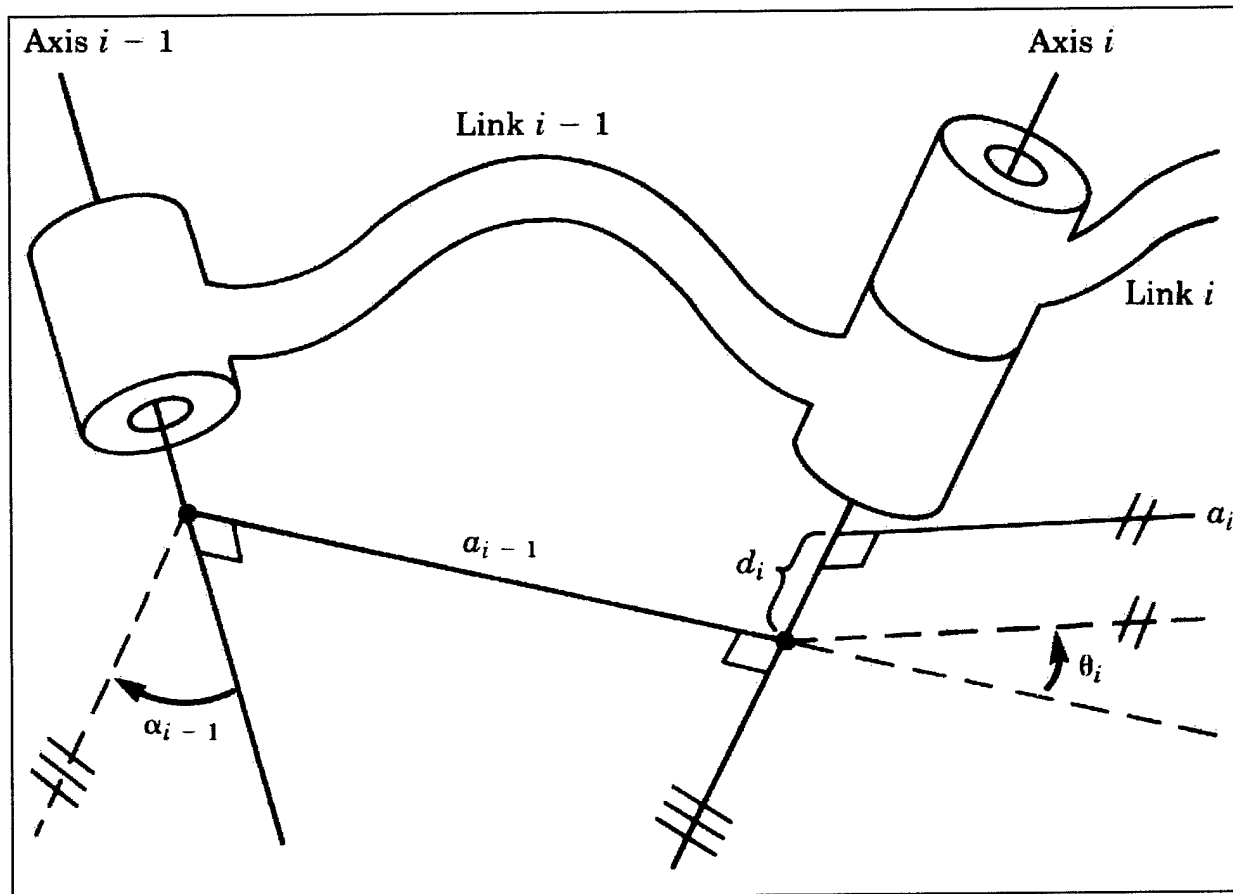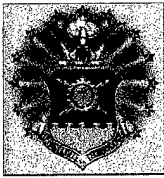
Figure 2.3.   Link Offset and Joint Angle

These four quantities, link length a_{i-1}, link twist α_{i-1}, link offset d_i, and joint angle θ_i, allow for the unique description of any common manipulator. Together, they form a convention known as the Denavit-Hartenberg notation (Craig:74). The four quantities are then regularly placed into a DH table containing the information for all degrees of freedom of the manipulator (Craig:68-82; Rattan:37-44).

The next step is to relate the frames of links i and i-1. To do this, three intermediate frames are created to allow the transformation form one link to the next. Figure 2.4 shows the addition of these three frames, denoted R, Q, and P (Craig:83).
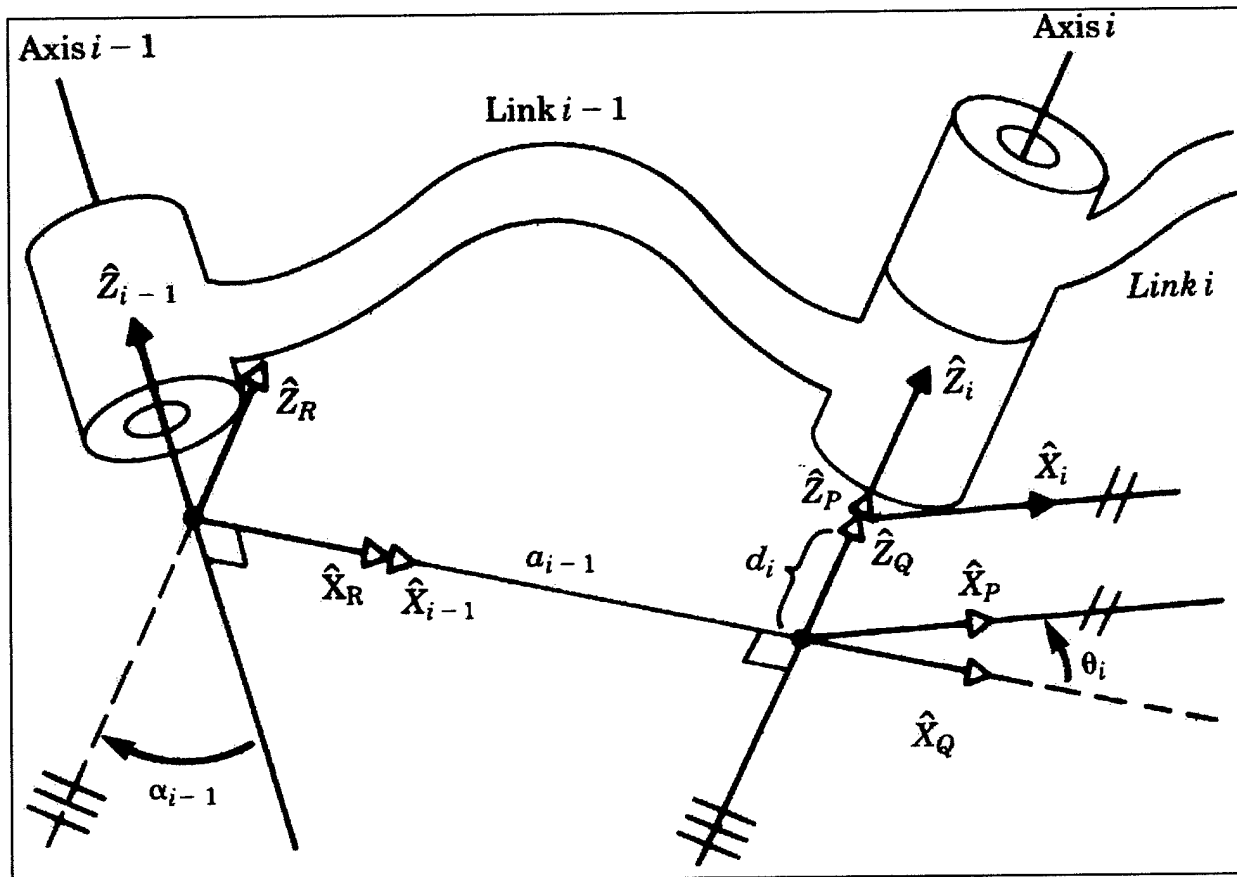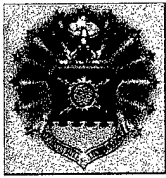
Figure 2.4   Intermediate Frames

First, the R frame is placed at the same origin as the i-1 frame but rotated about the x-axis by the link twist α_{i-1} amount. The Q frame is then placed in the same orientation as P but it is shifted along the x-axis by the link length a_{i-1} amount towards the next link. The R frame is then placed at the same origin as Q but rotated by the z-axis by the joint angle θ_i amount. Finally, the frame of link i has the same orientation as R but it is shifted along the z-axis by the link offset d_i amount towards the next link (Craig:83-84;Rattan:45-52).

Because moving from i-1 to R is a rotation, its rotational matrix is given by Equation 2.1.

The transformation from R to Q is given by the positional scaling vector of Equation 2.2.

Together, Equations 2.1 and 2.2 form the transformation matrix shown in Equation 2.3.

$$\text{Rotation about x-axis} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha_{i-1}) & -\sin(a_{i-1}) \\ 0 & \sin(a'_{i-1}) & \cos(\alpha_{i-1}) \end{bmatrix} \quad \text{Equation 2.1}$$

$$\text{Scaling along x-asis} = \begin{bmatrix} a_{i-1} \\ 0 \\ 0 \end{bmatrix} \quad \text{Equation 2.2}$$

$$\text{Transform (i-1 to Q)} = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & \cos(\alpha_{i-1}) & -\sin(a_{i-1}) & 0 \\ 0 & \sin(a'_{i-1}) & \cos(\alpha_{i-1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Equation 2.3}$$

Similarly, moving from Q to P is a rotation. Its rotational matrix is given by equation 2.4.

The transformation from P to i is given by the positional scaling vector of Equation 2.5.

Together, Equations 2.4 and 2.5 form the transformation matrix shown in Equation 2.6.

Rotation about z-axis $=
\begin{bmatrix}
\cos(\theta_i) & -\sin(\theta_i) & 0 \\
\sin(\theta_i) & \cos(\theta_i) & 0 \\
0 & 0 & 1
\end{bmatrix}$ **Equation 2.4**

Scaling along z-axis $=
\begin{bmatrix}
0 \\
0 \\
d_i
\end{bmatrix}$ **Equation 2.5**

Transform (Q to i) $=
\begin{bmatrix}
\cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\
\sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\
0 & 0 & 1 & d_i \\
0 & 0 & 0 & 1
\end{bmatrix}$ **Equation 2.6**

The complete transformation is the matrix multiplication of Equations 2.3 and 2.6. This is the transformation from the i-1 to the i link and is shown in Equation 2.7.

Transform (i-1 to i) $=
\begin{bmatrix}
\cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\
\sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\
\sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\
0 & 0 & 0 & 1
\end{bmatrix}$

Equation 2.7

To find the nth frame, simply multiply the transforms of each intermediate frame together as in Equation 2.8a. Equation 2.8b shows the final transformation matrix from 0 to n. The result is a 4 by 4 matrix that represents the orientation of frame n with respect to frame 0 and the
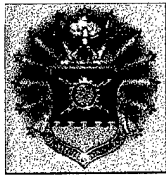
location of the last link with respect to frame 0. The first column represents the normal vector N, the second column represents the sliding vector S, the third column represents the approach vector A, and the fourth column represents the position vector P. Due to the nature of the zeros and ones in Equations 2.3 and 2.6, the fourth row will always be [0 0 0 1] (Craig:84-85; Rattan:53, 55).

$$\substack{0 \\ n}T = (\substack{0 \\ 1}T)(\substack{1 \\ 2}T)(\substack{2 \\ 3}T)\ldots(\substack{n-1 \\ n}T)$$

Equation 2.8a

$$\substack{0 \\ 4}T = \begin{bmatrix} N_x & S_x & A_x & P_x \\ N_y & S_y & A_y & P_y \\ N_z & S_z & A_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 2.8b

If there is an extension from the last joint, such as a tool or a finger tip of length L in the case of the UMDH, the orientation is the same as the joint itself, but the position is shifted by the amount L along the normal vector n of the joint. Equations 2.9, 2.10, and 2.11 show the modification to the position vector from the last joint to get the new position vector of the end of the extension (Solanki and Rattan:72).

$$P_x^{'} = P_x + N_x L$$

Equation 2.9

$$P_y^{'} = P_y + N_y L$$

Equation 2.10
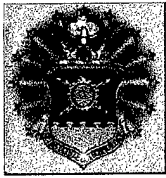
$$P_z^{\,\prime} = P_z + N_z L$$

<div align="right">**Equation 2.11**</div>

2.4   UMDH Forward Kinematic Computations

The UMDH shown in figure 2.5 is composed of three fingers and a thumb. The three

fingers are kinematically identical with the exception of their offsets at the knuckle locations. The

thumb is slightly different from the fingers and it is located between the first and second fingers on

the palm of the hand.
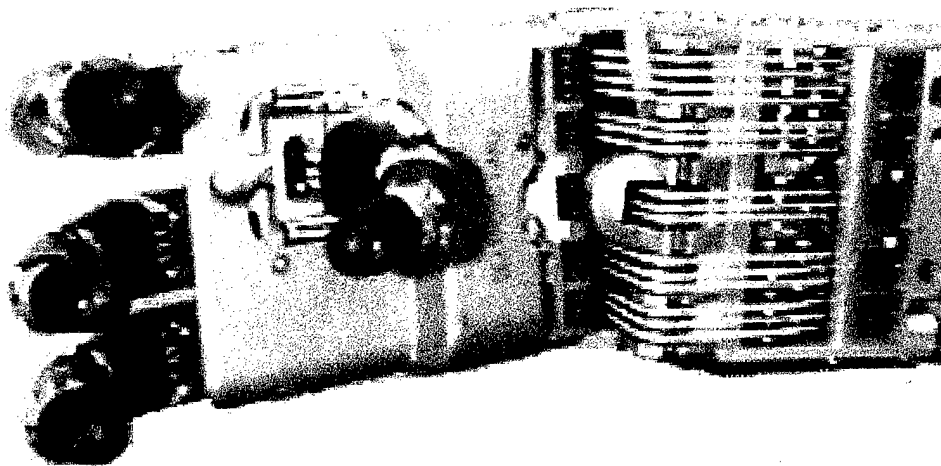
Figure 2.5.   Utah MIT Dextrous Hand

Figures 2.6 and 2.7 show the top and side view of the UMDH respectively (Solanki and

Rattan:67-68). Notice how the 0th frame is located back towards the wrist. It is defined at this

location because it is the intersection of the joint axis for both the thumb and the middle finger.

This could have been chosen at a different location but would result in more complicated

transformation matricies (Solanki and Rattan:66).
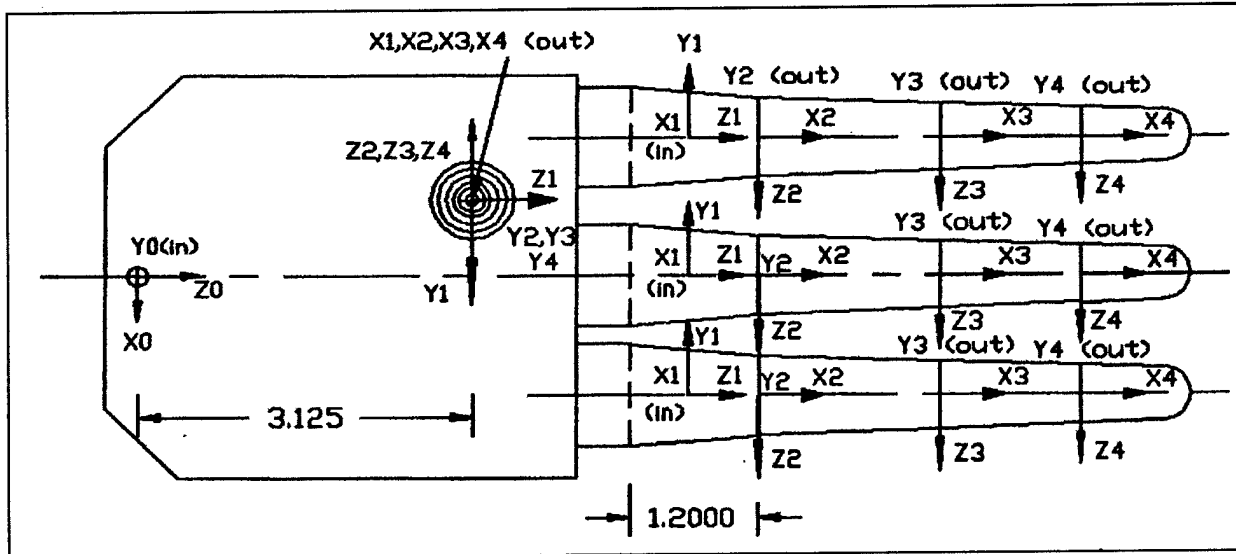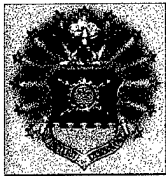
Figure 2.6.   Top View of UMDH (thumb extends out of page)
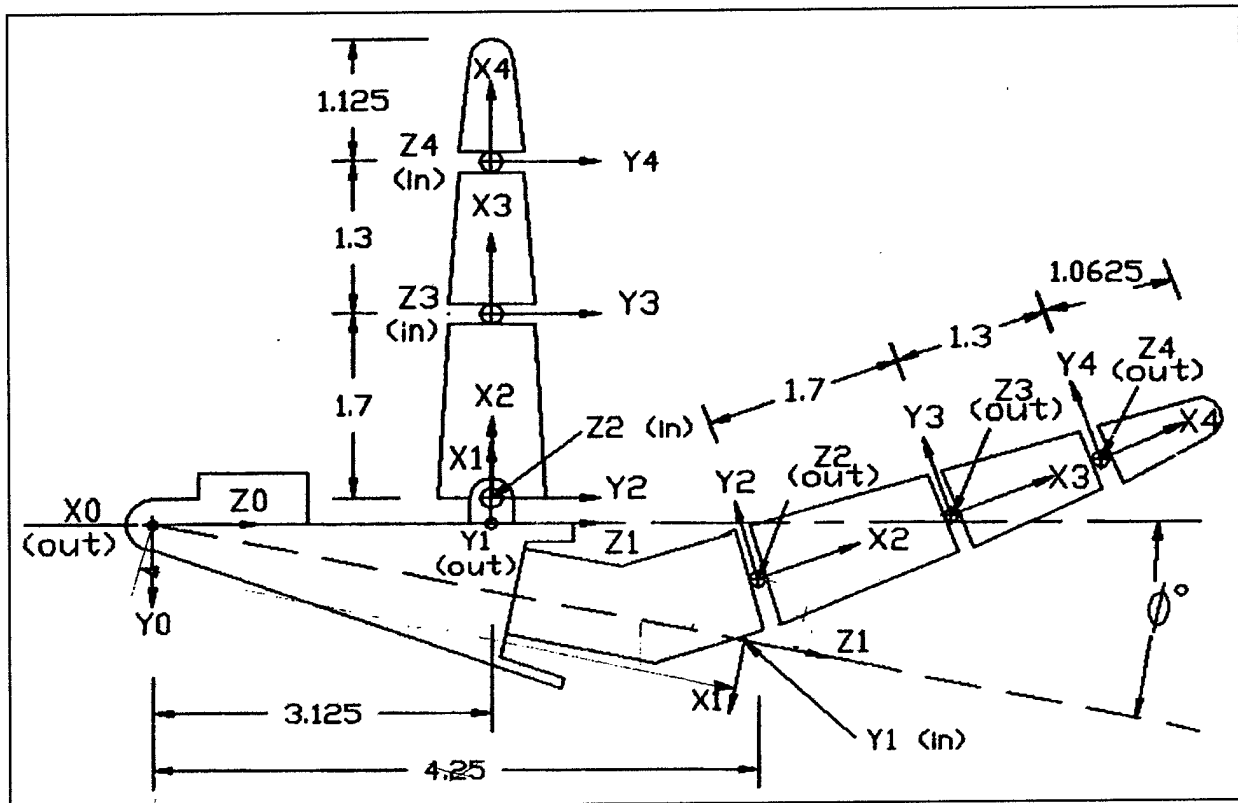
Figure 2.7.   Side View of UMDH
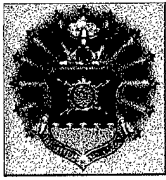
Because the three fingers and the thumb are almost kinematically identical, only one will be further explored. The thumb mechanism alone represents a serial chain manipulator with four degrees of freedom resulting from the four revolute joints. The DH table for the thumb of the UMDH in this configuration is shown in Table 2.1 (Solanki and Rattan:69). Using these values and Equation 2.7, each link relationship can be calculated. Replacing the i and i-1 variables with the fixed quantities from the DH table results in much simplified versions of the transformation matrices. Equations 2.12 through 2.15 shows each intermediate matrix (Solanki and Rattan:70).

Table 2.1.   DH table for Thumb of UMDH

i	link twist	link length	link offset	joint angle
1	$\alpha_0 = 0°$	$a_0 = -0.75"$	$d_1 = 3.125"$	θ_1
2	$\alpha_1 = 90°$	$a_1 = 0.375"$	$d_2 = 0"$	θ_2
3	$\alpha_2 = 0°$	$a_2 = 1.700"$	$d_3 = 0"$	θ_3
4	$\alpha_3 = 0°$	$a_3 = 1.300"$	$d_4 = 0"$	θ_4

$$
{}^0_1T = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & a_0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Equation 2.12

$$
{}_2^1T = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_1 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Equation 2.13

$$
{}_3^2T = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & a_2 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Equation 2.14

$$
{}_4^3T = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & a_3 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Equation 2.15

These four transformation matrices are concatenated into one using Equation 2.8. The result, after consecutive matrix multiplications, is shown in Equation 2.16 (Solanki and Rattan:71).

$$
{}_4^0T = \begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4) & -\cos(\theta_1)\sin(\theta_2+\theta_3+\theta_4) & \sin(\theta_1) & a_0+\cos(\theta_1)(a_1+a_2\cos(\theta_2)+a_3\cos(\theta_2+\theta_3)) \\ \sin(\theta_1)\cos(\theta_2+\theta_3+\theta_4) & -\sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4) & -\cos(\theta_1) & \sin(\theta_1)(a_1+a_2\cos(\theta_2)+a_3\cos(\theta_2+\theta_3)) \\ \sin(\theta_2+\theta_3+\theta_4) & \cos(\theta_2+\theta_3+\theta_4) & 0 & a_2\sin(\theta_2)+a_3\sin(\theta_2+\theta_3)+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Equation 2.16
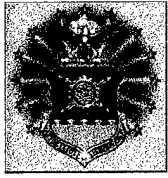
The elements within the matrix of Equation 2.16 are one to one equivalent to Equation 2.8b. The resulting twelve equations out of sixteen (four equations are a constant 0 or 1) can now be used as the basis for the remaining chapters.

2.5   Conclusions

This chapter investigated a mathematical method for the calculation of the forward kinematic equations of the thumb mechanism of the Utah MIT Dexterous Hand. The resulting Equation 2.16 = 2.8b represents the locations and orientation of the last joint of the UMDH. It does not directly give the location of the tip of the thumb. It will require the application of Equations 2.9 through 2.11 to derive such information from 2.16. The L term can be fixed as the length of the last link, or 1.3 inches if the desired answer is for the tip of the thumb. Other L values can be used to represent tools attached to the tip. Such tools might be force or temperature sensors. The remaining chapters will deal with the Equation 2.16 since this represents the base configurations of all UMDHs.

3. Algorithm Analysis and Profiling

3.1 Introduction

Before a physical computational architecture can be defined for implementation, the twelve equations derived in Chapter 2 need to be evaluated in the context of the desired performance of the UMDH. Only those hardware components that are absolutely necessary will be implemented. It is proposed that the desired forward kinematic processor deals only with mathematical operations and does not work with concepts such as character strings, addressing modes, or conditional branches typically found in a general purpose microprocessor. Therefore, this chapter deals with the trade-offs involved in finding an optimum hardware representation for both high performance and low hardware overhead.

3.2 Numeric Magnitude

The first metric that is evaluated is the notion of numeric magnitude. We need to know the highest valued (positive or negative) number that is ever used within any stage in the calculation of the equation. This defines the amount of hardware needed to hold such a number.

To determine such a number, the algorithm was written in the C language as a procedure call and is listed in Appendix A. The procedure is called by the main routine for many different UMDH configurations. Each of the four joints of the UMDH are controlled by nested FOR loops which cause the angles to sweep through each joint's given range shown in Table 3.1 (Solanki and Rattan:69). The results of the equations for each particular configuration were written to a data

file. The data file was then imported into the Matlab environment and searched for the maximum

and minimum values as listed in Appendix A. The values of the angles, including intermediate

steps where up to three angles are added together, show that they never exceed the range +360 to

-360 degrees. Intermediate additions, subtractions, and multiplications never exceed -2.3864 to

+3.3750. The final results of the NSAP matrix never exceed -2.3864 to +5.6271.

Table 3.1.   Kinematic Range of UMDH

Joint Angle	Range of motion in degrees
θ_1	-45 to 135
θ_2	-15 to 60
θ_3	6.5 to 90
θ_4	0 to 90

The implementation of the integer portions of such numbers can be accomplished directly

with just four bits of hardware (three bits represent the integers 0 to 7 and one bit for the sign).

However, since the values obtained are just a sample of the results from entire range of the

UMDH, and not an exhaustive test. This represents the minimum hardware size required. Also,

the future expansion to another type of manipulator may require more than just four bits.

Therefore, at least four bits will be held for now for hardware implementation..

3.3 Numeric Precision

The second metric used is the numeric precision required by the system. The UMDH was designed with metal joints that are controlled remotely via a set of tendons running around plastic pulleys. The coulomb friction of the joints and pulleys causes a motion deadband every time a joint stops. The electronic control system of the UMDH attempts to track the desired position of each joint, but it is limited by these mechanical properties. Consequently, simply turning up the gains of the UMDH controller would not suffice because that causes the joints to become unstable and to begin oscillating.

Therefore, in an attempt to avoid decreasing performance beyond that of the current system and to avoid possible truncation problems at intermediate stages in the equations, the number of decimal bits required is set to eight. This allows for a resolution of 0.003906250 per least significant bit since the last bit is the placeholder for 2^{-8}. If the value is representative of an angle, then it is clear that 0.003906250 degrees is much higher a precision than the UMDH could ever track. If the value represents a Cartesian coordinate of the end of the finger, then the same applies to 0.003906250 inches. Although the UMDH was modeled as an ideal body of rigid links, all devices will inherently flex to some extent.

3.4 Mathematical Operator Usage

The 12 equations are examined for occurrences of additions/subtractions, multiplications, or cosines/sines. A brute force approach by simply counting the number of operations found in Equation 2.16 results in 22 additions, three subtractions, 12 multiplications, 11 cosines, and nine sines. However, many of the terms in the 12 equations appear in more than one location.

Therefore, the number of operations can be reduced by sharing these terms. Both the cosine and sine of three angles are used three separate times. Similarly, the entire last half of P_x and P_y are identical. If the order of calculation for the 12 equations takes advantage of the common terms then the number of operations can be reduced to seven additions, three subtractions, 10 multiplications, four cosines, and four sines. This is a 68.2% decrease in additions, 16.6% decrease in multiplication, 63.6% decrease in cosines, and 55.5% decrease in sines. The subtractions remain unchanged because of the negative signs on P_y, S_x and S_y.

3.5 Conclusions

This chapter evaluated the equations from Chapter 2 to determine the best representation of the numbers. We determined that the absolute largest number only required four bits but that more bits for higher numbers may be required in future implementations. To keep the precision of each number, eight bits are required for a minimum of 1/256th difference between each number.

Therefore, the implementation of the numbers in hardware are done with a total of eight bits for the integer portion and eight bits for the decimal portion. Together, the 16 bits form the basis for a fixed point number with the binary point in the center between the set of eight bits. This results in a maximum number of +127.99609375 and a minimum number of -128.00000000.

Finally, we determined that the 12 equations can be calculated in just 28 operations if common terms are reused. This is a decrease of 50.9% from the original 57 operations.

4. VHDL Model

4.1 Introduction

This chapter discusses the first step in the implementation of the forward kinematic processor. The step is the development of behavioral VHDL models for each of the required mathematical operations found in Equation 2.16 as well as temporary register-based memory and other structures used to route the data within the processor. Finally, a structural VHDL model for the entire processor is developed. Each model is developed and simulated using the Synopsys Analyzer and Simulator (Synopsys) before synthesis in Chapter 5.

4.2 Functional Units

In all models, the 16-bit fixed-point representation of all numeric data will be implemented as a bit vector of size 15 down to 0. The binary point is implied to be at the center, between bits 8 and 9.

4.2.1 Cosine/Sine Unit.

The first functional unit developed was the cosine and sine unit. Both transcendental functions are designed into one model as shown in Figure 4.1. The unit calculated the cosine or sine by means of an external lookup table. An address is generated and sent to a ROM chip that returns the result back to the cosine/sine unit. Since the specifications of the external ROM chip were not known at the start of the design, the model incorporated the ability to set the delay before the unit latches the results from the ROM. These wait states allow the possibility of the use of slower ROM devices.
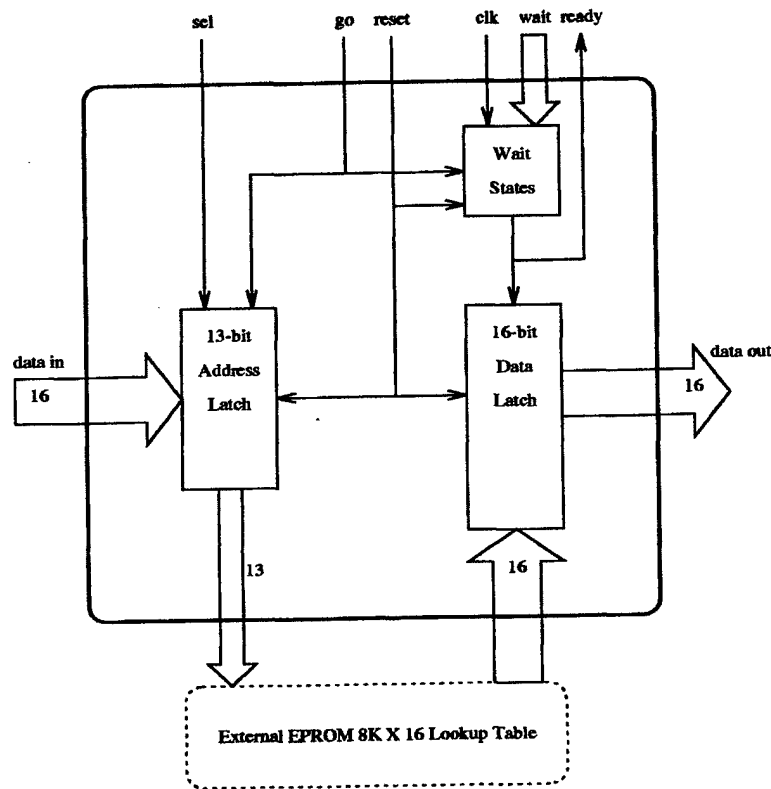
Figure 4.1. Cosine/Sine Unit Block Diagram

For example, if the system clock of the forward kinematic processor has a clock period of

40 ns (25 MHz) and the ROM device has an access time of only 150 ns, then the number of wait

states would be set to three. Three wait states causes three extra 40 ns clock cycles in addition to

the current cycle, for a total of 4 cycles or 160 ns. This prevents the cosine/sine unit from reading

in incorrect data early.

The state machine is shown in Figure 4.2. A reset signal during any state will force the

system to state 0. In state 0, the ready output signal is not asserted, the number of wait states are

calculated, the temporary counter is set to zero and look-up table address is formed and sent to

the external ROM. To form the address, the unit takes as input a 16-bit vector and strips off the

lower 11 bits, representative of three bits of integer and eight bits of decimal. Also, the highest

bit, representing the sign, is also pulled out. Finally, an input signal called sel, that determines

cosine or sine, is also taken and these 13 bits form the address into the ROM lookup table

containing the results of both cosine and sine.
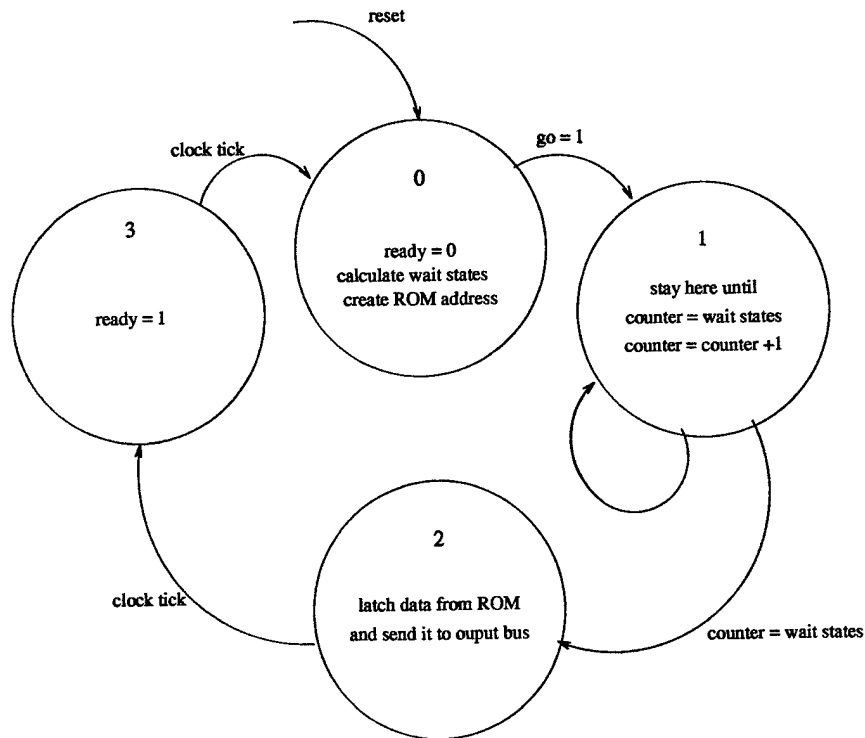
Figure 4.2. Cosine/Sine Unit State Machine

The unit will stay in state 0 until the go input signal is asserted. Once in state 1, it will stay

there, incrementing the counter until it matches the precalculated number of wait states. It will

then move to state 2 where the results from the ROM look-up table are latched into the output

bus. The unit then transitions to state 3 at the next rising edge of the clock and the ready output

signal is asserted. The next transition on the rising edge of the clock is back to state 0, where it

waits for the next cycle.

The behavioral VHDL model for the cosine/sine model is listed in Appendix B.1.1. The VHDL testbench code and results for it are listed in Appendix B.1.2. The testbench sends the unit through the eight possible wait states with a simulated external ROM. These results are shown in Appendix B.1.3.

4.2.2 Adder/Subtractor Unit.

The adder and subtractor are contained within one functional unit. The subtractor is implemented using the adder model and inverting the secondary input before applying it to the adder. In both cases, two 16-bit numbers are input into the unit and one 16-bit number is output as shown in Figure 4.3. There are no provisions for overflow or underflow conditions because of the nature of the operands. At no time should there occur an overflow or underflow condition.
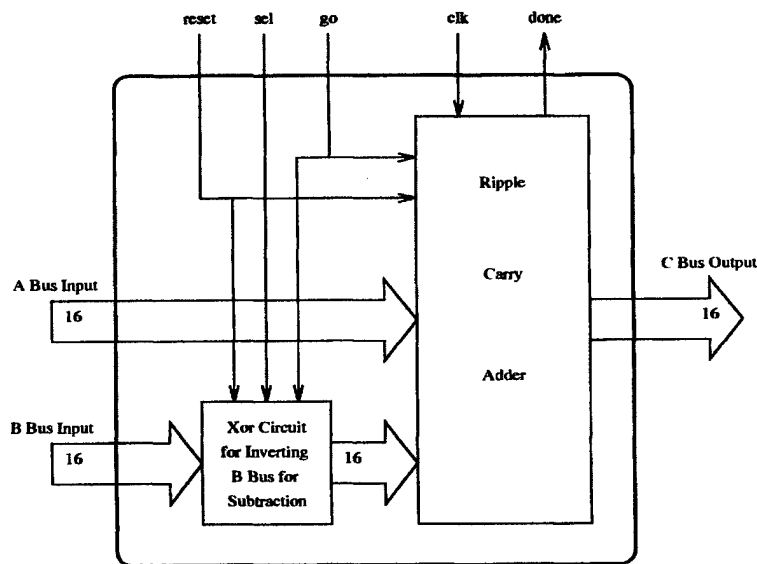
Figure 4.3. Adder/Subtractor Unit Block Diagram
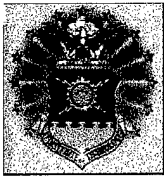
The unit starts at idle in state 0 shown in Figure 4.4. When the go input signal is asserted,

the unit starts by calculating the sum and carry terms of Equation 4.1 and 4.2 for the least

significant bits, where A and B are inputs bits and C is the carry in from the previous bit.

(Weste and Eshraghian:517). Each clock tick causes the unit to progress to the next state and

calculate the next bit. After sixteen clock ticks, all sums have been calculated and the result is

sent to the output bus. A done output signal is asserted indicating completion and the state

machine returns to state 0 in preparation for another addition or subtraction.
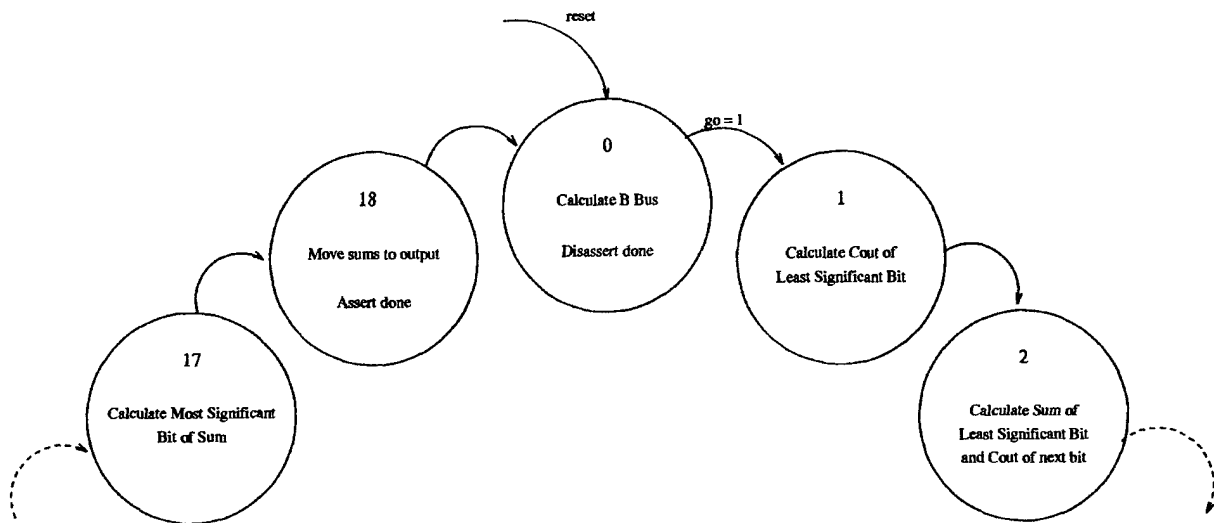
Figure 4.4. Adder/Subtractor Unit State Machine

Carry = AB + C(A+B)   **Equation 4.1**

Sum = ABC + (A+B+C)Carry   **Equation 4.2**
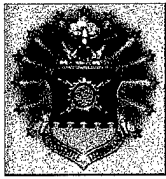
Typically, an adder/subtractor would not be implemented as a state machine requiring at least 16 clock ticks. However, since the target platform is an FPGA, and the timing of the synthesized design will not be known until Chapter 5, it is impossible to determine how long it will take to allow all the sum and carry terms to ripple their results to the final result. Therefore, the unit indicates to the surrounding system when it has completed the final state by asserting the done signal. If at any time the reset signal is asserted, the unit is forced back to state 0.

The behavioral VHDL model for the adder/subtractor model is listed in Appendix B.2.1. The VHDL testbench code for it is listed in Appendix B.2.2. The testbench sends the unit through 30 different additions and 30 different subtractions. These results are shown in Appendix B.2.3.

4.2.3 Multiplier Unit.

The multiplier unit has the same data interface as the adder/subtractor unit. Figure 4.5 shows the two 16-bit inputs and one 16-bit result. Once again there are no provisions for overflow or underflow. Typically two 16-bit numbers multiplied together would result in a 32-bit result, but in this specific implementation, the numbers should never exceed 16-bits, a constraint of the 16-bit architecture.

The multiplier actually uses a modified copy of the adder/subtractor inside its design. The adder/subtractor is extended to 32-bits to handle the accumulation of the partial products. The multiplier follows the same basic data flow as the adder/subtractor except that it requires many more states to calculate the result. Figure 4.6 shows the state machine for the multiplier unit.
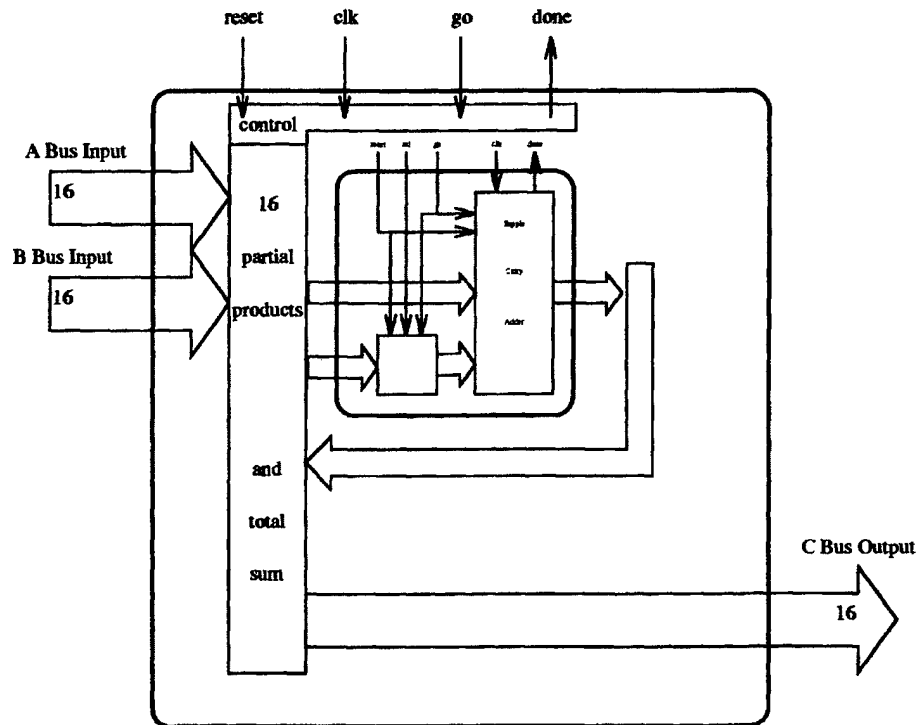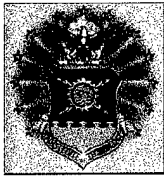
Figure 4.5. Multiplier Unit Block Diagram
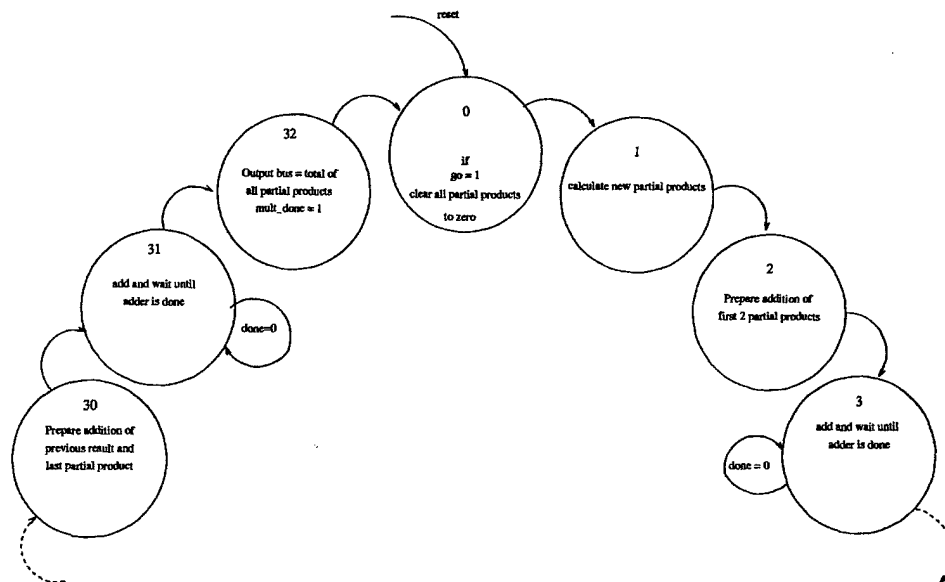
Figure 4.6. Multiplier Unit State Machine

It stays idle in state 0 until the go input signal is asserted. Each of 16 partial products are calculated and then repetitively added up to form the final result. Similar to the adder/subtractor unit, when the final state is reached, an output signal ready is asserted to indicate to the surrounding system that multiplication is complete. If at any time the reset signal is asserted, the unit is forced back to state 0.

The behavioral VHDL model for the multiplier model is listed in Appendix B.3.1. The VHDL testbench code for it is listed in Appendix B.3.2. The testbench sends the unit through the same 30 inputs as the adder/subtractor but multiplies rather than adds or subtracts. These results are shown in Appendix B.3.3.

4.2.4 Register File Unit.

The register file unit is used to store the starting angles of the UMDH, certain constants from the DH table, temporary and intermediate calculations, and the 12 equation results. It is designed to hold the 16-bit numbers in any of 32 different locations, except for the first two locations. The first location is hard wired to always hold a zero value and the second location holds a hard wired one value. This was designed early on because of the expected need to increment by one or to allow for moves from one location to another through the adder/subtractor unit with one of the inputs being zero.

It is designed with one 16-bit input bus called the C bus and two 16-bit output buses called the A and B bus as shown in Figure 4.7. The data of the C bus is written to any of the remaining 30 locations by use of the C bus address and a latch signal. Data can be read from any

of the 32 locations to both A and B bus by using the A and B address. If the reset signal is

asserted, the 30 locations are forced to zero.

C address   C latch   reset   clk A address B address

16 bit wide

32 word long

Register file

C Bus Input

16

d
e
c
o
d
e
r

d
e
c
o
d
e
r

d
e
c
o
d
e
r

A Bus Output

16

B Bus Output

16
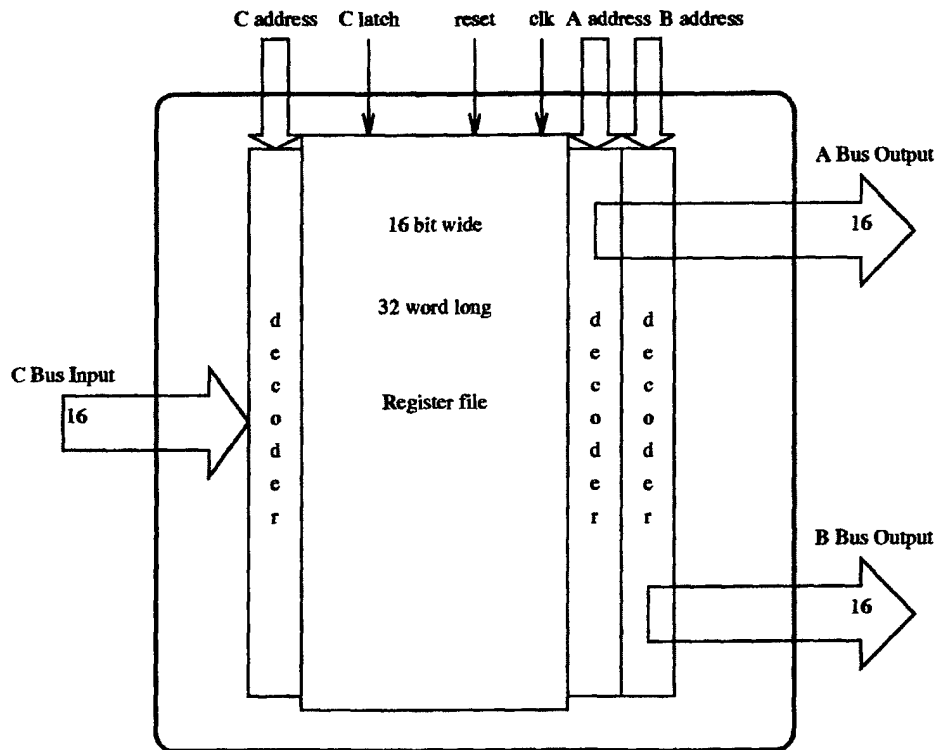
Figure 4.7. Register File Unit Block Diagram

The behavioral VHDL model for the register file model is listed in Appendix B.4.1. The

VHDL testbench code for it is listed in Appendix B.4.2. The testbench has three parts. In the

first part, a reset is asserted and the zero register and one register are verified as well as that the

remaining 30 were forced to zero. In the second part, all 32 registers are given test values. In the

third part, all 32 registers are evaluated again showing that all but the two hard wired registers

accepted the values. These results are shown in Appendix B.4.3.

4.2.5 Latches and Multiplexors.

The latches and multiplexors are required in the design as glue logic between the other

functional units. To start, there is a 16-bit latch as shown in Figure 4.8. When its latch signal is

asserted, the input bus is transferred to the output and held at that value until the next time this

latch is asserted. This design requires two latches as described in the next section. The

behavioral model for the latch is found in Appendix B.5.1 and its testbench is located in B.5.2.

The results of the testbench are found in Appendix B.5.3.
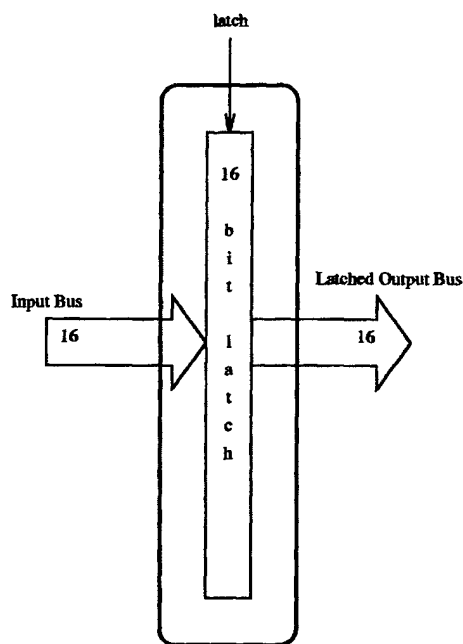
Figure 4.8. Latch Unit Block Diagram

Also required is a multiplexor as shown in Figure 4.9. It directs one of four inputs to a

single output. The multiplexor is 16 bits wide for all inputs and outputs and is controlled by two

input signals determining the one of four paths.
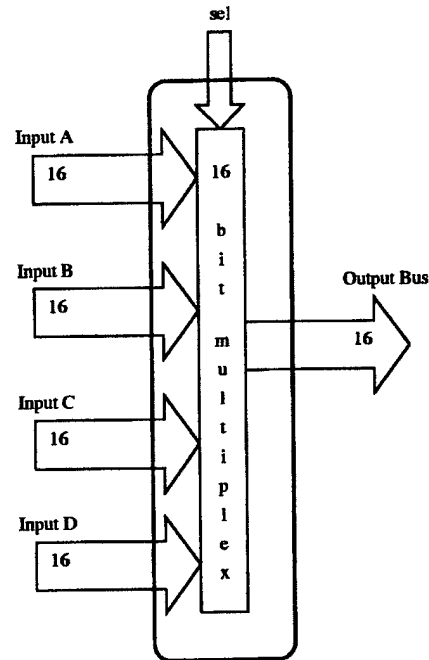
Figure 4.9. Multiplexor Unit Block Diagram

The behavioral VHDL model for the multiplexor is found in Appendix B.6.1 and its

testbench is located in B.6.2. The results of the testbench are found in Appendix B.6.3.

4.2.6 FKP Core.

The functional units designed so far are brought together to form the core of the Forward

Kinematic Processor (FKP). This core encapsulates the functional units such that they appear like

a single large functional unit. Two latches and one multiplexor are used to glue the other

functional units together so that data can travel from unit to unit in a productive manner. Figure

4.10 shows the connections of the units inside the core. There is one 16-bit data input bus which

is routed to the input data latch. From there, the data is passed though the multiplexor and back

around to the register file for storage. Once data is loaded into the registers, they can be sent to

the cosine, sine, addition, subtraction, or multiplication units and rolled back around to the

register file via the multiplexor again. When the desired computations are complete, the data in a

register is sent to the output latch and then to the output bus. To control the dataflow, all of the

control signals from each of the functional units are passed as control signals for the core unit.

This model does not handle the actual control of the core, but rather gives one concise shell for

everything inside it.
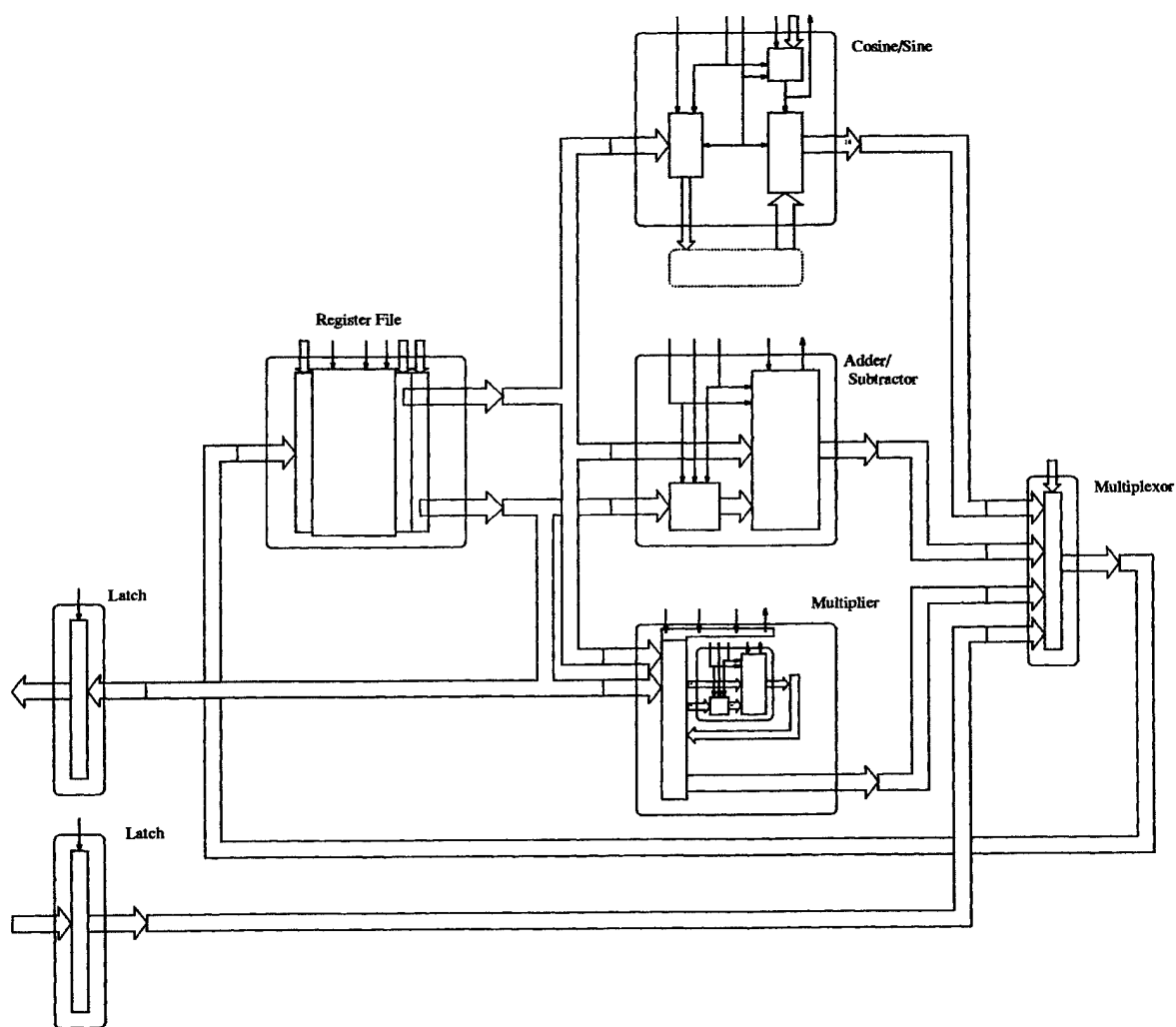
Figure 4.10. FKP Core Block Diagram

The structural VHDL model of the FKP core is shown in Appendix B.7.1 with the testbench in B.7.2. The testbench performs the actions described above on some data. It was designed to prove functionality of the core since each subunit has already been verified. The results are shown in Appendix B.7.3.

4.2.7 Microcode Store.

This section defines the instruction set of the processor. Because this is an application specific design, the instruction set contains only commands for moving data in and out, and performing one of the arithmetic or transcendental operations. Table 4.1 shows all possible instructions utilized within the processor. The microcode for each instruction is derived from the testbench of the FKP core. Since the FKP core does not supply autonomous control over the functional units, each simulated instruction was hard coded in sequence. The microcode store has taken each simulated instruction and formed each into a procedure (opcode) call with its parameters (operands) being the passed into the procedure. All procedures are contained in a package model that can be called by the control unit of the next section.

The behavioral VHDL package model of the instructions are shown in Appendix B.8.1 with the testbench in Appendix B.8.2 performing the same operations as the FKP core testbench. The results in Appendix B.8.3 show that the replacement of the autonomous microcode performs identically to the hard coded testbench of the FKP core.

Table 4.1. FKP Instruction Set

Instruction	Description
move_in (R, data)	Latch input bus, pass data through multiplexor to register R
move_out (data, R)	Move data out of register R, through output latch to output bus
add (R1, R2, R3)	Send data from two registers (R2 and R3) to two inputs of adder/subtractor unit, add, send result back to register R1
sub (R1, R2, R3)	Send data from two registers (R2 and R3) to two inputs of adder/subtractor unit, subtract, send result back to register R1
mult (R1, R2, R3)	Send data from two registers (R2 and R3) to two inputs of multiplier unit, multiply, send result back to register R1
cos (R1, R2)	Send data from register R2 to input of cosine/sine unit, perform cosine, send result back to register R1
sin (R1, R2)	Send data from register R2 to input of cosine/sine unit, perform sine, send result back to register R1

4.2.8 Control Unit.

The control unit can now utilize the microcode store package to make the FKP core perform the various instructions without the burden of worrying about dataflow on every single clock tick. The control unit allows interface with the outside world via an six bit control port and a seven bit command port as shown in Table 4.2 and 4.3 respectively. The control unit is a shell for the microcode store and the FKP core as shown in Figure 4.11.

Table 4.2. Control Port

Bit #	5	4	3	2	1	0
Name	Clock	Reset	Strobe	Ready	DataGetValid	DataGetAck
IN/OUT	IN	IN	IN	OUT	OUT	IN

Table 4.3. Command Port

Description bit #	CMD1 6	CMD0 5	A4 4	A3 3	A2 2	A1 1	A0 0
Set Register	0	0	A4	A3	A2	A1	A0
Get Register	0	1	A4	A3	A2	A1	A0
Run	1	0	X	X	X	X	X

External EPROM Lookup Table

Data IN   16   FKP CORE   Data OUT   16

8   8

Command Port   Control Port
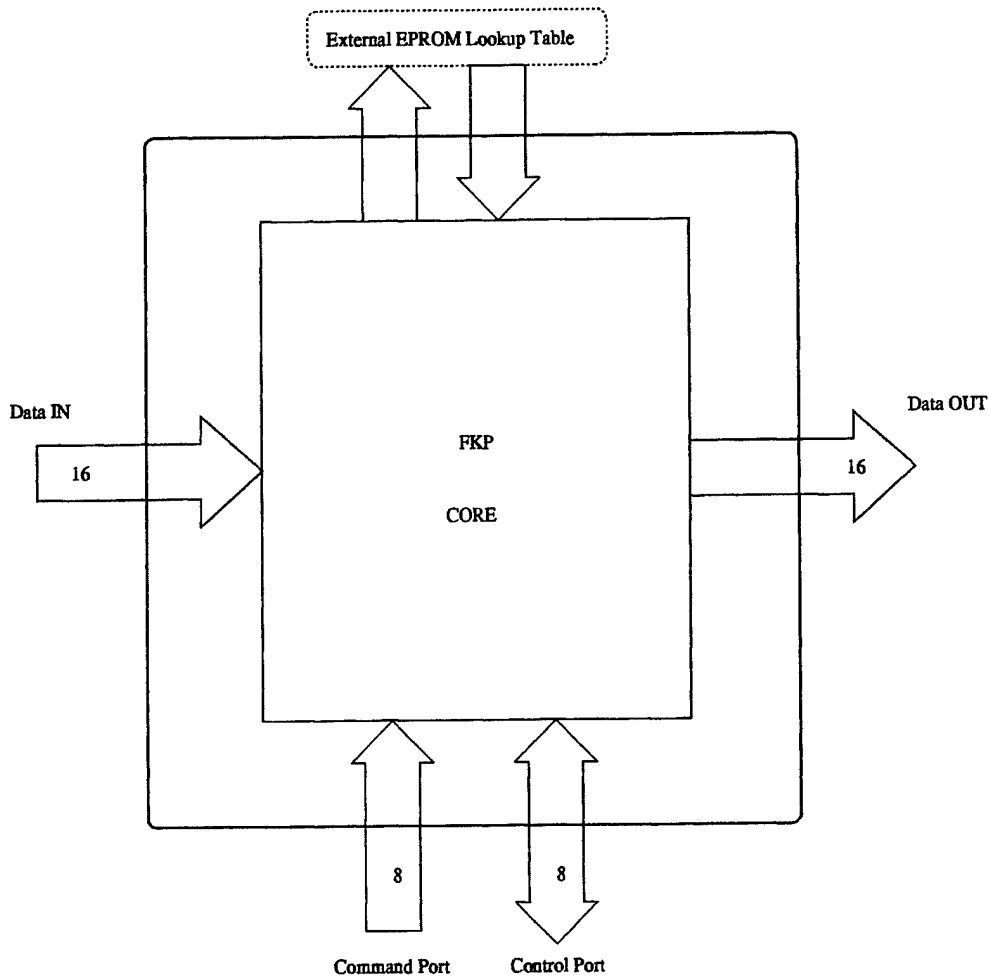
Figure 4.11. FKP System Block Diagram

The clock input is the overall system clock for the processor. The reset is the overall system reset for the processor. The remaining bits of the control port are utilized in conjunction with the command port. After system reset, the ready output signal is asserted, indicating that the processor is available to perform one of the three functions: set register, get register, or run. The user sets the CMD0 and CMD1 bits to correspond to the desired function and asserts the strobe input signal. The processor will deassert the ready signal, evaluate the command port and take the appropriate action. When the function is complete, the ready signal is reasserted.

If the function is a set register, then the 16-bit input data bus is latched in and routed to the register designated by bits A4-A0 of the command port. If the function is a get function, then the register designated by bits A4-A0 are sent through the output latch and to the 16-bit data output bus. Finally, if the function is run, then the A4-A0 bits are ignored and the predetermined sequence of instructions is executed.

The sequence is arranged to take advantage of any common terms found in the 12 equations of Chapter 2. Chapter 3 evaluated the equations and determined that there would be seven additions, three subtractions, 10 multiplication's, four cosines, and four sines. This would require a total of 28 instructions. However, this did not count for the data moves into and out of the processor using the set and get functions. Table 4.4a shows the operations involved with moving in the angles and possibly some constants into the registers. The register locations that hold this constant data is fixed due to the fact that the run function will expect the correct data in these locations. The first time theses data values are loaded, both constants (a's) and angles (b's)

are required. But from then on, only the new set of angles are needed because the constants do

not change and are not written over unless due to power loss or system reset.

Table 4.4a.   Operations Involved with the Set Function

Step #	Register #	Instruction and Description
1a	2	*move_in (2, a0)* = move link length 0 into register 2
2a	3	*move_in (3, a1)* = move link length 1 into register 3
3a	4	*move_in (4, a2)* = move link length 2 into register 4
4a	5	*move_in (5, a3)* = move link length 3 into register 5
5a	6	*move_in (6, d1)* = move link offset 1 into register 6
1b	7	*move_in (7, θ1)* = move theta 1 into register 7
2b	8	*move_in (8, θ2)* = move theta 2 into register 8
3b	9	*move_in (9, θ3)* = move theta 3 into register 9
4b	10	*move_in (10, θ4)* = move theta 4 into register 10

With the constants and angles loaded, the run function can be initiated. Table 4.4b shows

the internal steps involved with calculating the results of the twelve equations. There is one extra

add of step 18 due to the internal move of the zero in the zero register to register 28.

Figure 4.4b. Internal Operations During Run Function

Step #	Register #	Instruction and Description
2	11	*cos(11, 7)* = cos(θ1)
3	12	*sin(12, 7)* = sin(θ2)
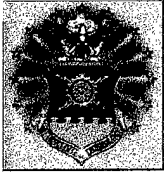
4	13	$cos(13, 8) = \cos(\theta2)$
5	14	$add(14,8, 9) = \theta2+\theta3$
8		$add(14, 14, 10) = \theta2+\theta3+\theta4$
6	15	$sin(15, 14) = \sin(\theta2+\theta3)$
7	16	$cos(16, 14) = \cos(\theta2+\theta3)$
19	17	$mult(17, 4, 13) = a2 \cos(\theta2)$
21		$add(17, 17, 18) = a2 \cos(\theta2) + a3 \cos(\theta2+\theta3)$
22		$add(17, 17, 3) = a1 + a2 \cos(\theta2) + a3 \cos(\theta2+\theta3)$
20	18	$mult(18, 5, 16) = a3 \cos(\theta2+\theta3)$
23		$mult(18, 17, 11) = \cos(\theta1)(a1 + a2 \cos(\theta2) + a3 \cos(\theta2+\theta3))$
26	19	$mult(19, 4, 12) = a2 \sin(\theta2)$
11	20	$mult(20, 11, 25) = \cos(\theta1)\cos(\theta2+\theta3+\theta4)$
12	21	$mult(21, 26, 25) = \sin(\theta1)\cos(\theta2+\theta3+\theta4)$
9	22	$sin(22, 14) = \sin(\theta2+\theta3+\theta4)$
13	23	$mult(23, 11, 22) = \cos(\theta1)\sin(\theta2+\theta3+\theta4)$
14		$sub(23, 0, 23) = -(\cos(\theta1)\sin(\theta2+\theta3+\theta4))$
15	24	$mult(24, 26, 22) = \sin(\theta1)\sin(\theta2+\theta3+\theta4)$
16		$sub(24, 0, 24) = -(\sin(\theta1)\sin(\theta2+\theta3+\theta4))$
10	25	$cos(25, 14) = \cos(\theta2+\theta3+\theta4)$
1	26	$sin(26, 7) = \sin(\theta1)$

17	27	$sub(27, 0, 11) = -\cos(\theta1)$
18	28	$add(28, 0, 0) = 0$
24	29	$add(29, 18, 2) = a0 + \cos(\theta1)(a1 + a2 \cos(\theta2) + a3 \cos(\theta2+\theta3))$
25	30	$mult(30, 17, 26) = \sin(\theta1)(a1 + a2 \cos(\theta2) + a3 \cos(\theta2+\theta3))$
27	31	$mult(31, 5, 15) = a3 \sin(\theta2+\theta3)$
28		$add(31, 31, 19) = a2 \sin(\theta2) + a3 \sin(\theta2+\theta3)$
29		$add(31, 31, 6) = a2 \sin(\theta2) + a3 \sin(\theta2+\theta3) + d1$

The get functions can now be used to retrieve the last 12 registers for the results of the 12 equations. Each value is moved out one at a time and in any order the user desires.

The structural VHDL model of the Forward Kinematic Processor is shown in Appendix B.9.1.

4.3 Conclusions

This chapter developed the models of each of the required functional units. Each model was tested as a stand-alone design before integration into the Forward Kinematic Processor. Once the initial five constants are loaded in, the processor takes four instructions to load the angles, 29 instructions to calculate the results, and 12 instructions to get them out, for a total of 45 instructions. The processor was then tested from the top most level of the design model. With the simulation of the processor complete, the next step in the implementation is synthesis to an FPGA. This is described in Chapter 5.

5. VHDL To FPGA Synthesis

5.1   Introduction

The goal of this chapter is to move the FKP design modeled in the hardware description language straight to an FPGA implementation. The models were behavioral descriptions of the functional units with a top level structural description of the entire processor. At this level of abstraction, there is no implied physical architecture. We have not even worked with a gate level representation of the design. The synthesis into an FPGA induces an explicit physical architecture based on the target device; in this case the Xilinx 4020E.

5.2   VHDL Source Restrictions

VHDL was originally designed as a simulation and modeling language. The concept of synthesis directly from the model was not included in the design of the language. Therefore, some of the constructs found in VHDL are not synthesizable. The most obvious limitation is the use of specific time delays. For example, the statement "wait for 10ns" or "A <= B after 5ns" has no meaning to a synthesis tool because there is no on-chip clock to direct when the action is to take place. Also, constructs such as access types, records, recursive subprograms, and multidimensional arrays are non-synthesizable (Raines; Ailes:21).

Most of these restrictions were known when beginning the development of the models from Chapter 4, but some unexpected and potentially detrimental constraints appeared as the design moved on. First was the use of more than one signal inside of process sensitivity list. Typically, many signals can be listed in the sensitivity list of the process, indicating execution of

the process if any of the listed signals changes state. The synthesis tools could only handle one signal in the list. A process that is dependent on both the clock and the reset signal would cause errors during synthesis. To work around this problem, most all sensitivity lists became empty forcing continuous execution, with the clock events being listed as a separate wait statement within the process body. The second problem pertains to the need to assert a signal for one clock period and then deassert it on the next clock period. Such an event infers a clock wait between the two transitions, but only one wait statement is allowed on each pass through the process body. The result is a streamlined hardware description such as "A<=B; wait until clock tick; A<=not(B); wait until clock tick" being unrolled to an explicit state machine where the execution through the process body takes a different path for each state. Each state then contains a unique command for "A<=B" or "A<=not(B)" and there is only one wait statement for all paths.

5.3   Design Flow

There are four major tools used to perform the synthesis step. The Synopsys VHDL analyzer is used to compile the VHDL code. This includes compilation of the testbenches for each functional unit. The functional units are then simulated with the Synopsys VHDL simulator. These two tools together, both executing on a UNIX platform, form the primary development tools of the models (Synopsys). Because both the Analyzer and Simulator do not aim towards synthesis, the restrictions from section 5.2 are ignored and pushed aside for later tools. The other three major tools are Synopsys Design Analyzer and Exemplar Leonardo for synthesis, and Xilinx XACTstep for mapping.

5.3.1 Synopsys Design Analyzer

The Synopsys Design Analyzer started out as the primary UNIX synthesis tool. Within

the Design Analyzer is a feature called the FPGA Compiler. It accepts VHDL as input and

attempts to produce a hybrid Synopsys/Xilinx netlist. The drawback to using this tool is its

turnaround time. Typically, a small model such as the cosine/sine unit will take upwards of two

hours to generate the netlist (Synopsys).

5.3.2 Exemplar Leonardo

The PC/Windows 95 based Exemplar Leonardo application turned out to be quicker than

Synopsys and much easier to learn and use. The following sequence describes the path used to

generate a correctly targeted netlist (Exemplar). First, the program is loaded and the startup

screen is shown in Figure 5.1.


```
File   IO   Optimize   Report   Hierarchy   Tools   Options                          Help

xmplr      Flow Guide...    Toolbar...    Schematic Viewer...    Design Browser...    Quit...

Leonardo - V4.0.3
Copyright 1990-1996 Exemplar Logic, Inc.  All rights reserved.


   *** Welcome to Interactive Leonardo Version V4.0.3 ***

News :
  * Enter "help" to get an overview of all commands
  * Enter <command> -help  to get usage of each command

Session history will be logged to file 'exemplar.his'
LEONARDO{1}:
                                                          Cancel
```

Figure 5.1. Exemplar Logic Leonardo Startup Screen

The first action taken is to click on the Flow Guide button. The Flow Guide shown in

Figure 5.2 appears. Because we wish to customize certain aspects of the design, the Customize

Flow Guide button is clicked. Another window appears that allows us to inform the tool that the

design consists of multiple VHDL files because many of the functional units depend on a package

or header file. We also select the option of packing the configurable logic blocks (CLB) of a

Xilinx FPGA, decomposition of Look Up Tables (LUT), and reporting of area used as shown in

Figure 5.3. The result is a variation of Figure 5.2 with the extra steps added into the design Flow

Guide of Figure 5.4.
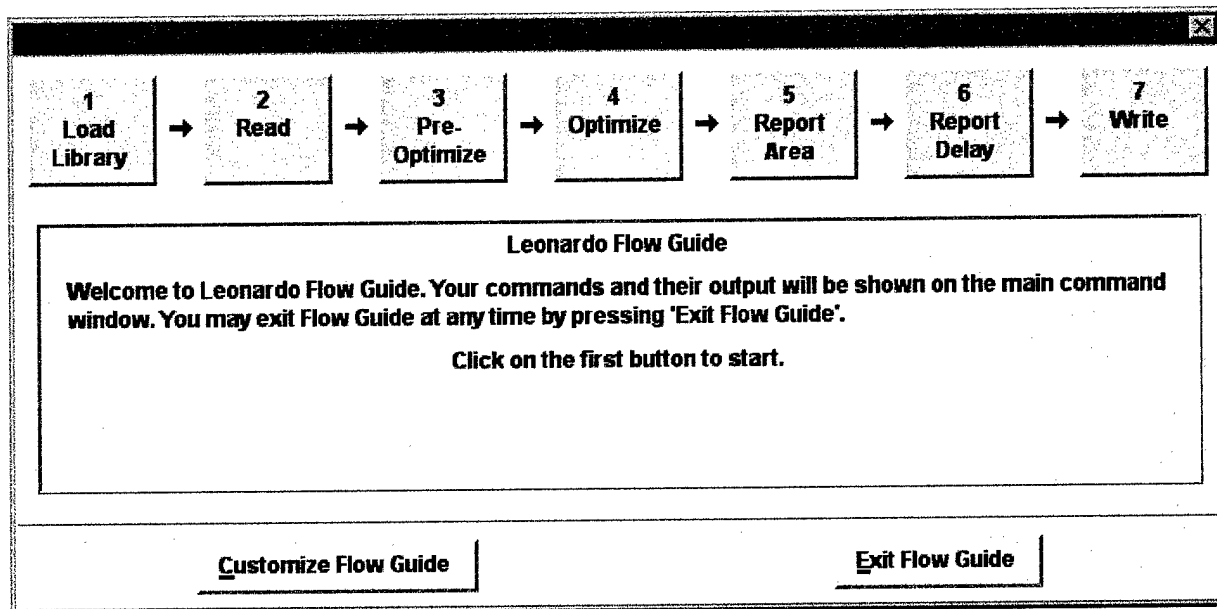
Figure 5.2. Leonardo Flow Guide

The first button, Load Library, is selected and we choose the 4000E family as shown in

Figure 5.5. The second button is used repetitively to read in and analyze the VHDL files. A

window appears that allows the filename to be input as shown in Figure 5.6. As each file is being

read in, any warning messages are displayed regarding synthesis problems.

Check all boxes that apply to your design:

Input Flow:
- ☑ Multiple VHDL or Verilog Input Files
- ☐ Altera EDIF input file
- ☐ Design with instantiated modgen cell

Optimize Flow:
- ☑ Technology specific module generation
- ☑ Extract counters, decoders and rams
- ☐ Specify constraints for optimization/timing optimization
- ☐ Timing Optimization
- ☑ Pack CLBs (Xilinx)

Reporting Flow:
- ☑ Report Area
- ☐ Report Delay

Output Flow:
- ☑ Decompose LUTs (FLEX, ORCA, Xilinx 3k/4k/5k)
- ☐ Load balancing for Actel, QuickLogic and ASICs
- ☐ Generate timespec for Xilinx
- ☐ Altera EDIF output file

Run Flow Guide   Cancel

Figure 5.3. Customize Flow Guide

Once all the VHDL files are loaded in, the design is elaborated based on the top level

entity description. Figure 5.7 shows the Elaborate window. Clicking the elaborate button

automatically determines what the top level is and considers its port declaration as the I/O of the

design. Next, the Pre-Optimize step is accomplished, shown in Figure 5.8, followed by the

selection of the Modgen Library in Figure 5.9, and the resolution of the Modgens shown in Figure

5.10.
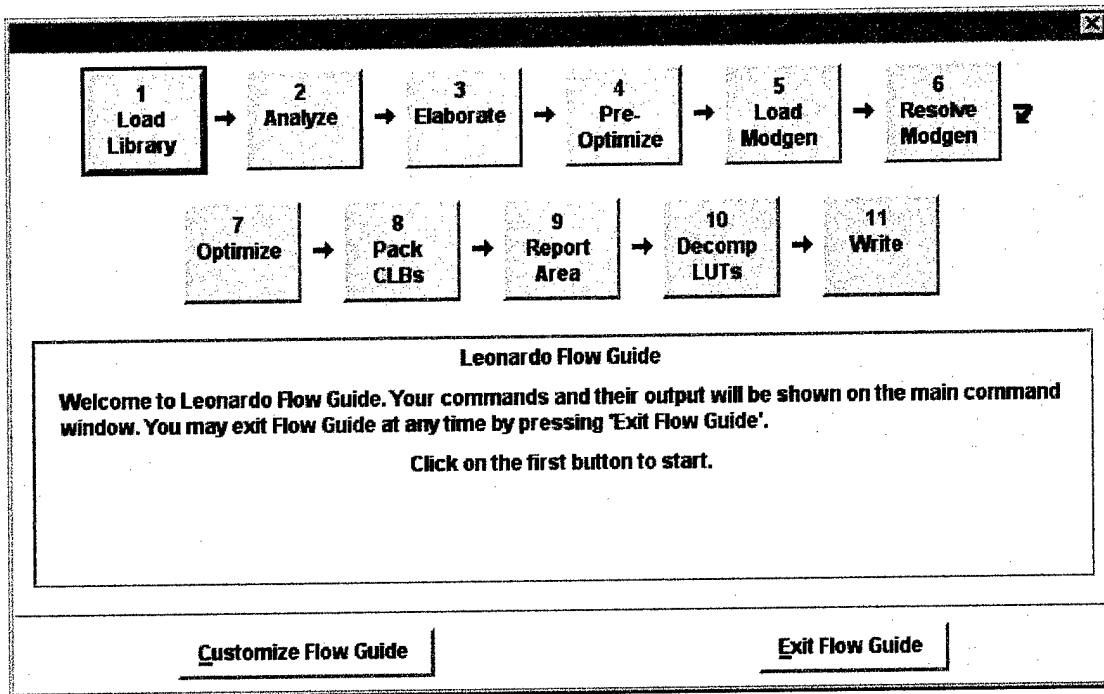
Figure 5.4. Customized Flow Guide
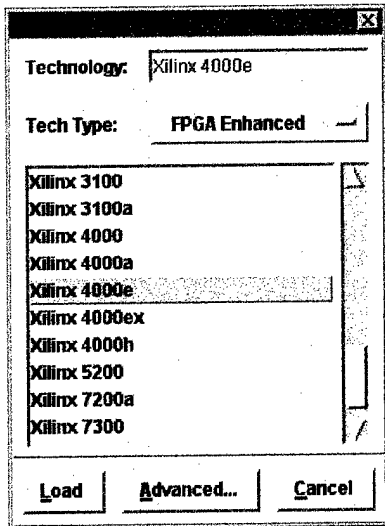
Figure 5.5. Load Library

Figure 5.6. Analyze

Root Entity:

Architecture:

Work Library: work

Parameters:

Generics:

Elaborate   Cancel

Figure 5.7. Elaborate

Design: .work.reg_file_16_e.behavior

Switches:
■ Share common logic

■ Remove unused (dangling) logic

■ Extract counters, decoders and rams

⌐ Perform operation on only a single level of hierarchy

Pre-Optimize   Cancel

Figure 5.8. Pre Optimize

Modgen Library: Xilinx 4K

Lucent ORCA-2A
Lucent ORCA-2C
Xilinx 3K
Xilinx 3K XBLOX
Xilinx 4K
Xilinx 4K XBLOX
Xilinx 5K
Xilinx 7K
General ASIC Technologies
General FPGA Technologies

Load   Cancel

Figure 5.9. Load Modgen Library

Design: .work.reg_file_16_e.behavior

Switches:
⌐ Preserve hierarchy

⌐ Default Resolving

⌐ Perform resolving only at the top level of hierarchy

Resolve   Advanced...   Cancel
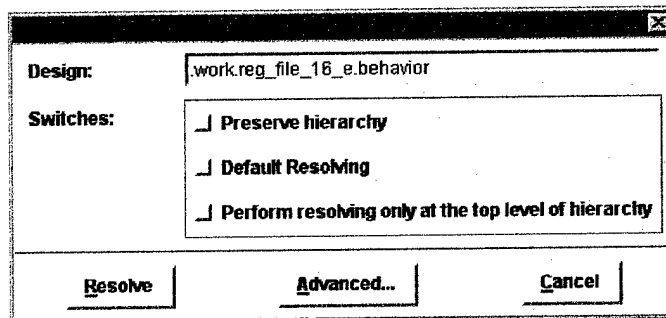
Figure 5.10. Resolve Modgen

The heart of this design flow is the Optimize step, where we can choose what type of optimization to do. The exhaustive selection will require multiple hours to complete. On the other hand, a quick optimization may only require five to 10 minutes. Because we are primarily concerned with area and not with speed, the area optimization box is checked as shown in Figure 5.11. The results of the optimization are shown in Figure 5.12, but the numbers are not entirely accurate. The critical path is listed as 29ns. However, the design has not yet been placed and routed on the chip. We will see later in Chapter 6 that the critical path is closer to 100ns.
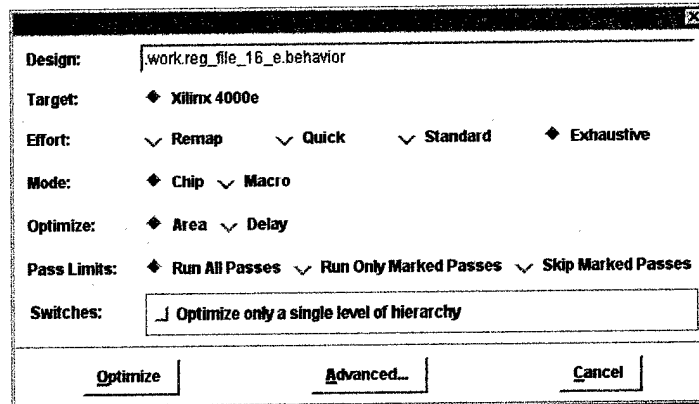
Figure 5.11. Optimize

```
-- Start optimization for design .work.reg_file_16_e.behavior

      Pass        Area      Delay        DFFs  PIs    POs  --CPU--
                  (FGs)     (ns)                            min:sec
      1           809        29          256    31     32    00:54

                            Resource Use Estimate

      Technology:      xi4e
      Area:            809 Function Generators
      Critical Path:   29 ns
      DFFs:            256 (in CLBs or IOBs)
      IOFFs:           32 (in IOBs)
      HM CLBs:         0
      Input Pins:      31
      Output Pins:     32
```

Figure 5.12. Results of Optimization

The optimized design is then packed into the CLBs by using the window shown in Figure

5.13, followed by decomposing the LUTs within the CLBs shown in Figure 5.14.
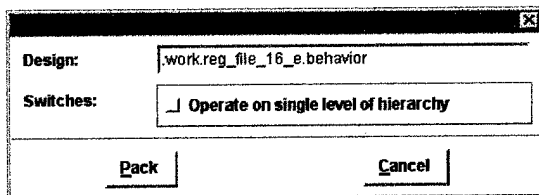
Figure 5.13. Pack CLBs
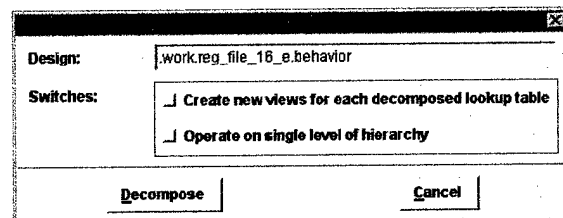
Figure 5.14. Decompose LUTs

The final step is the writing of the Xilinx Netlist Format (XNF) file to disk as shown in

Figure 5.15.
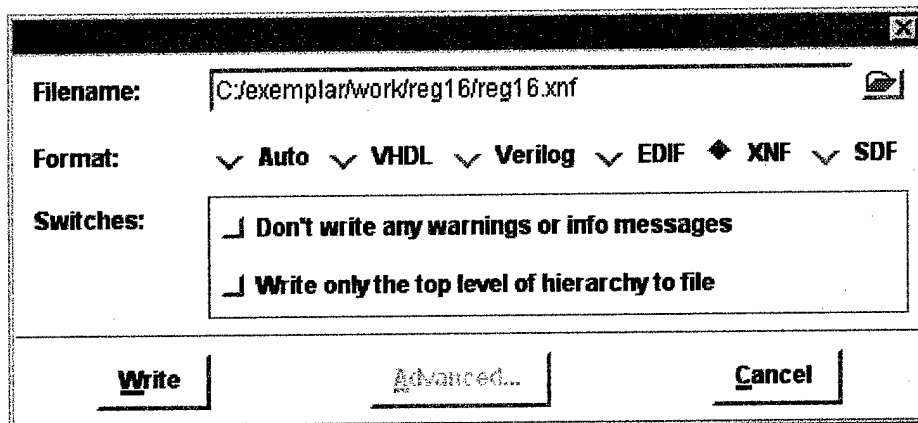
Figure 5.15. Write XNF

5.3.3 Xilinx XACTstep M1

The Xilinx XACTstep program picks up where the Exemplar tools stop. It inputs the

XNF file and sets up a project manager screen that keeps track of the version and revision of the

design as shown in Figure 5.16. Once loaded in as a project the design is implemented as shown
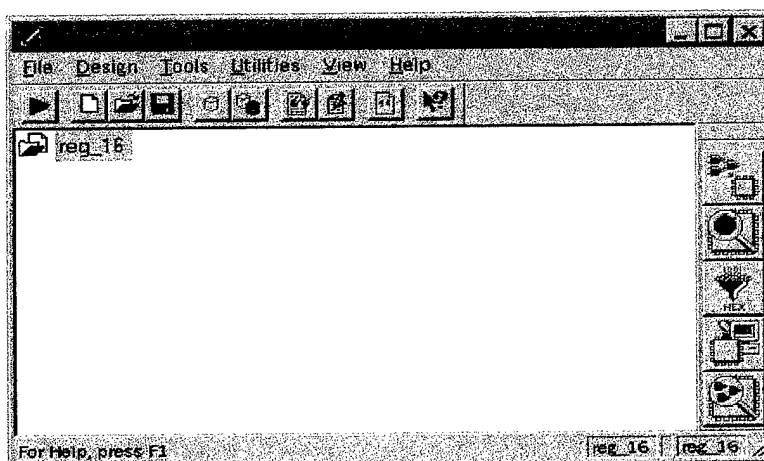
Figure 5.16. XACTstep Design Manager

in Figure 5.17. The target device is chosen, along with the current version and revision number.

Additional options shown in Figure 5.18 allow a constraint file to be added to the design. In this

case, a UCF file is used to lock certain I/O names to actual pins on the FPGA. Also, the

configuration template can be edited from this screen. Figure 5.19 shows the configuration

options screen. Both the inputs and the outputs are set to CMOS thresholds and the DONE, M0,

M1, and M2 mode pins are set to have an internal pull-up resistor.
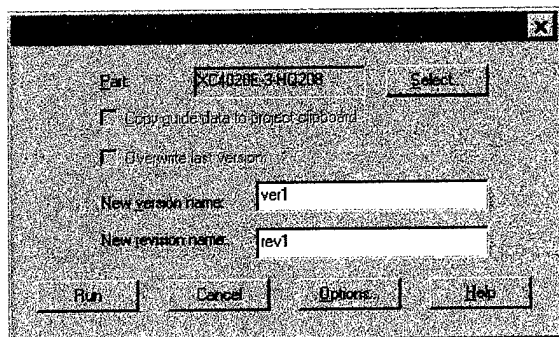
Figure 5.17. Implementation Window   **Figure 5.18. Implementation Options**
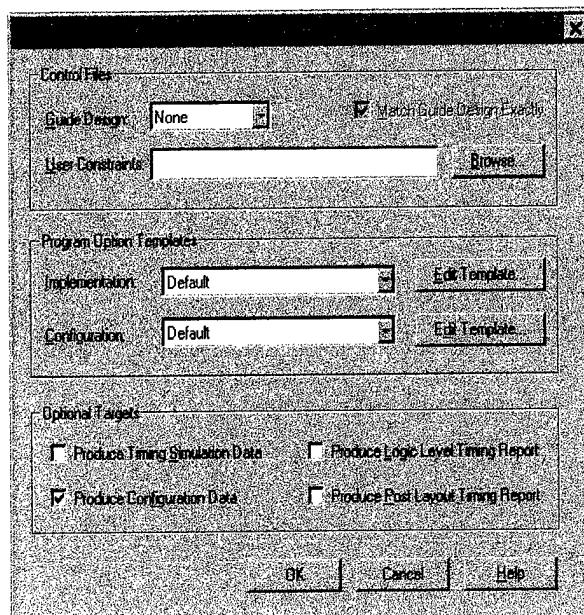
The Flow engine is now invoked and the process of translating, mapping, placing and

routing, and configuring is performed. Figure 5.20 shows the Flow Engine and the results of a

synthesized design. The result is a BIT file that is ready for download into the FPGA.
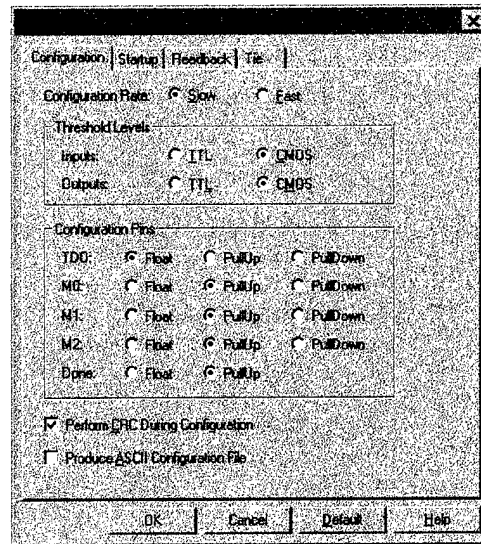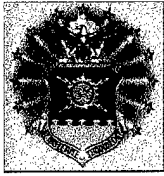
Figure 5.19. Configuration Options
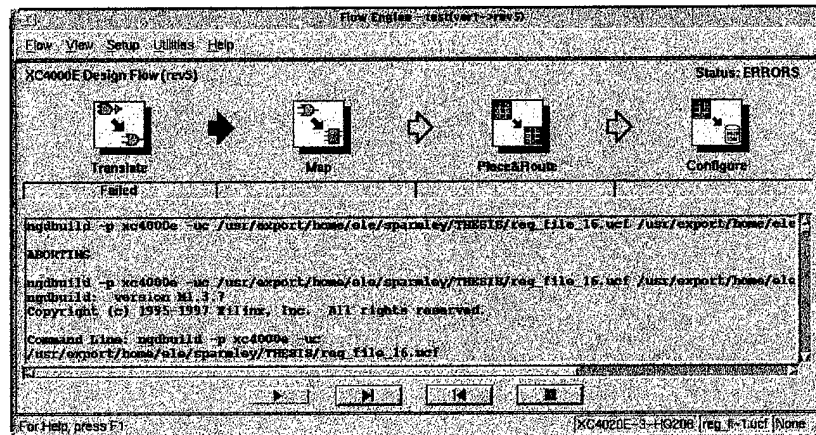
Figure 5.20. Flow Engine

5.4   Bitstream file to FPGA

The BIT file is downloaded to the FPGA using the Hardware Debugger utility of the

XACTstep program. An X-Checker cable is used between the FPGA and the host computer's

serial port. The Hardware Debugger then sends the proper headers, frames of data, and trailers

down the X-Checker cable and into the FPGA.

5.5   Conclusions

This chapter discussed the procedures for synthesizing VHDL models to FPGA

implementations. The process works, however the FKP processor cannot fit entirely on the target

4020E FPGA. If the target FPGA was much larger in capacity than the 4020E, then in theory, the

entire design could be placed into one device. Instead, half of the register file unit is pushed

through Exemplar Leonardo and Xilinx XACTstep and programmed into the 4020E that is

available in the laboratory. Figure (5.21) shows the CLB and routing layout for the register file in

the 4020E. This design used 40% of the total available CLBs, 27% of the total available IOBs,

and 12% of the total CLKIOBs of the 4020E. A text log of the XACTstep process from XNF

format to BIT format is listed in Appendix C.
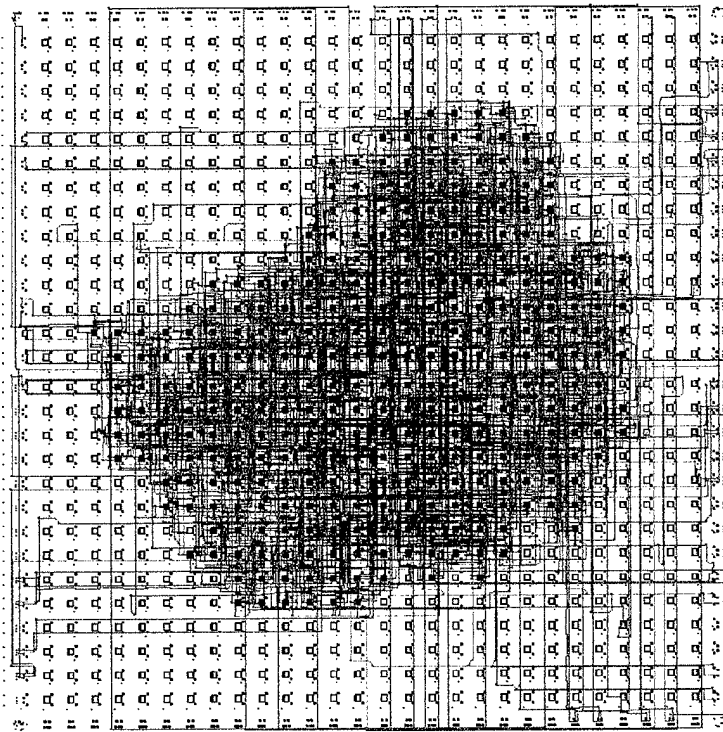
Figure 5.21. 4020E CLB and Routing for the Half Register File Unit

6. FPGA Verification

6.1   Introduction

This chapter investigates the physical implementation of one of the functional unit models

into a Xilinx 4020E FPGA. The Logic Master XL100 by Integrated Measurements Systems

(Integrated) will serve as the testbed for the programmed device. Because the 4020E package is

not directly compatible with the IMS, a custom adapter is developed.

6.2   IMS Logic Master XL100 tester

The IMS Logic Master XL100, shown in Figure 6.1, can support up to 100MHz data and

clock rates with up to 224 I/O channels. To test the 4020E FPGA, one XL PGA Auto Socket

Card is used to form the interface to the IMS.
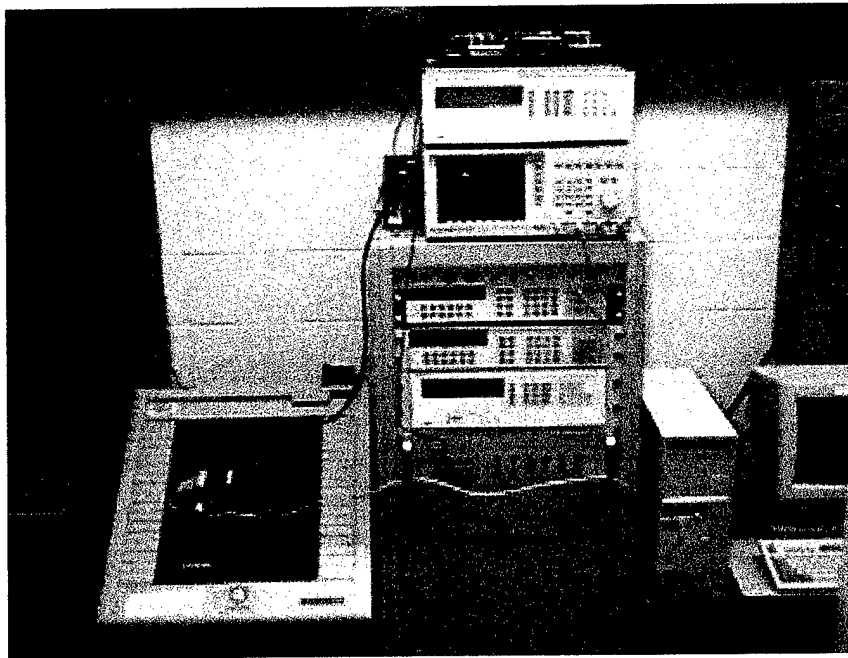
Figure 6.1. The IMS Logic Master XL100

6.3   HQ208 Chip Carrier and Daughter Board

The Xilinx 4020E FPGA is contained in a Heat-sinked Quad Flat Pack (HQFP) 208 pin

package (Xilinx:10-35). Because the device does not have pins that can be easily inserted into a

test circuit board, an adapter from Ironwood Electronics (see Appendix D) is used to mount the

FPGA to the test board. The adapter is wire-wrapped to a set of connectors which match up with

connectors installed on the IMS socket card. Figure 6.2 shows the completed test unit. Also

shown in Figure 6.2 is the Xilinx X-Checker cable for downloading the serial bit stream from the

host PC to the FPGA.
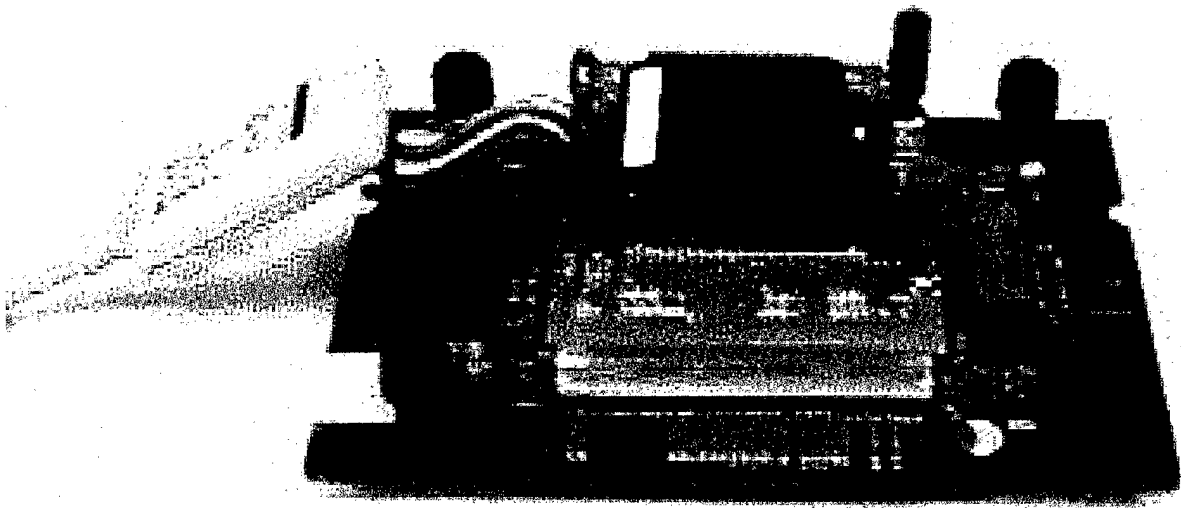
Figure 6.2. Completed Test Unit

There are 16 ground connections and seven +5 Volt connections to the adapter. The

power supply is external to the IMS to allow the FPGA to be programmed and hold its

configuration when the IMS is not cycling a test. When the IMS finishes a test and sits idle, it

removes all power to the device under test. This would erase the configuration every time the

IMS stopped a test cycle because the configuration is stored in internal latches (Xilinx 13-39). By keeping power supplied to the FPGA, even while idle, the configuration is retained. One possible solution to the loss of configuration is to program a PROM device instead of the FPGA directly. The PROM can then hold the configuration information even when the power is removed, and transfer the data into the FPGA every time the system powers up.

Also connected to the adapter are control pins for the FPGA. The TCK pin is pulled up to Vcc to prevent the device from entering into a boundary scan EXTEST during the download process(Xilinx:13-30). The M0, M1, and M2 pins are also pulled up to Vcc to force the device into Serial Slave mode. This mode is the simplest to implement. The Init, Done, Rst, and Prog pins are all pulled up to Vcc. Combined those with the Din and Cclk from the X-Checker and we have the setup shown in Figure (6.3) (Xilinx:5-18).

The remaining connections represent either input or output of the FPGA. The Ironwood Electronics data sheet in Appendix C shows the 4020E pin name and number associated with the adapter pin numbers and corresponding IMS connections.

There is a switch wired to the Prog pin to allow a forced reset of the FPGA. This causes the configuration to be erased and the device will prepare for a new download. The small green LED indicates power to the FPGA from the external supply. The red LED indicates that the IMS has output 5 Volts on the J13 channel.
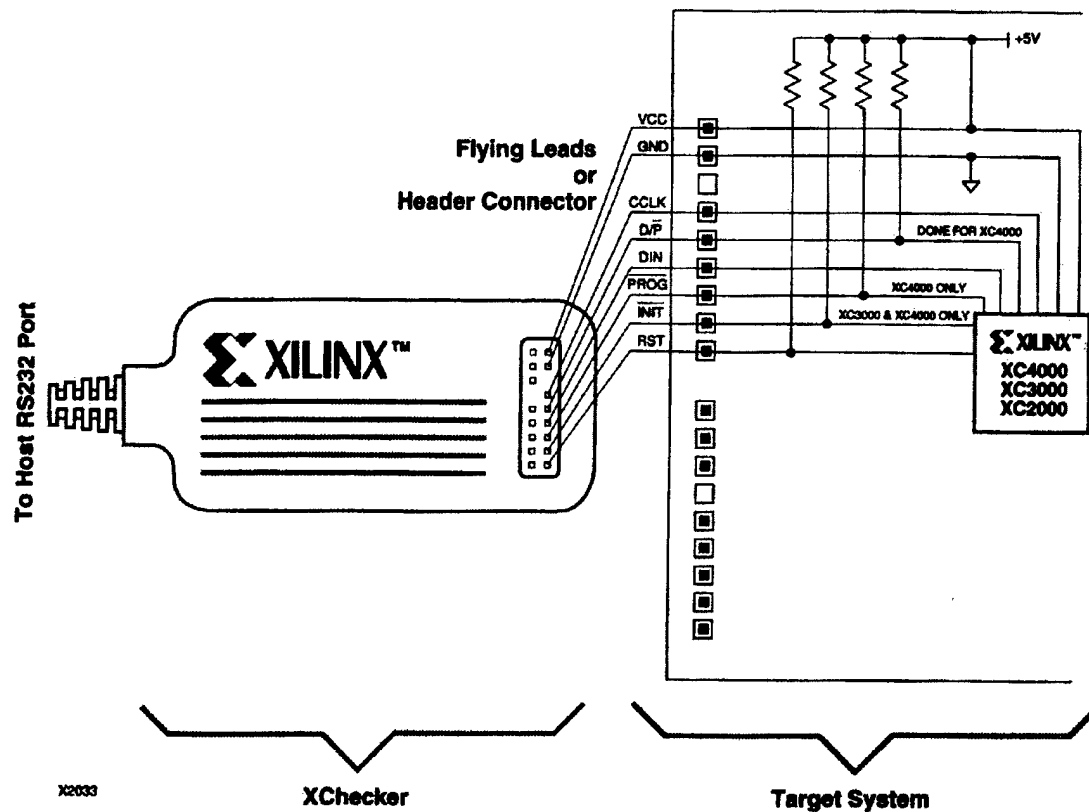
Figure 6.3. Slave Serial Download

6.4   Functional Unit Testing

The first design that was successfully tested was a combination AND/OR gate utilizing

four I/O pins and one CLB out of a total of 784. The AND/OR gate was modeled in VHDL and

pushed all the way through to implementation. Fastest speed rating on the gates was 11 ns, or

90.9MHz.

The second design was the half register file unit from Chapter 5. The only difference in

the process the second time was the addition of a UCF constraint file to force the I/O pins to

predetermined locations. Even if the model changes and causes a resynthesis of the design, the surrounding environment of the FPGA does not have to change.

The IMS tester allowed for a functionality and speed test of the FPGA. For the functional test, the register file is reset and all 16 registers are output to the A and B bus in opposite orders. Figure 6.4 shows the waveforms and indicates that all registers except number 1 is cleared to a zero. If we recall from Chapter 4, the number 1 register always holds a numeric 1.0, and the number 0 register always holds a numeric 0.0.

After the registers are cleared, all 16 registers are written to with a different bit. Once again the two output buses A and B are given the values of each register in opposite order. The waveform shows that both the A and B bus can retrieve the stored information from all registers, with the exception of registers 0 and 1.

The speed test is performed by decreasing the IMS clock period until the above functionality test fails. At 48.5 ns, the test fails. Because the cycle of the register file is two cycles of the IMS, the actual failure time is a 97 ns clock period, or 10.3MHz.
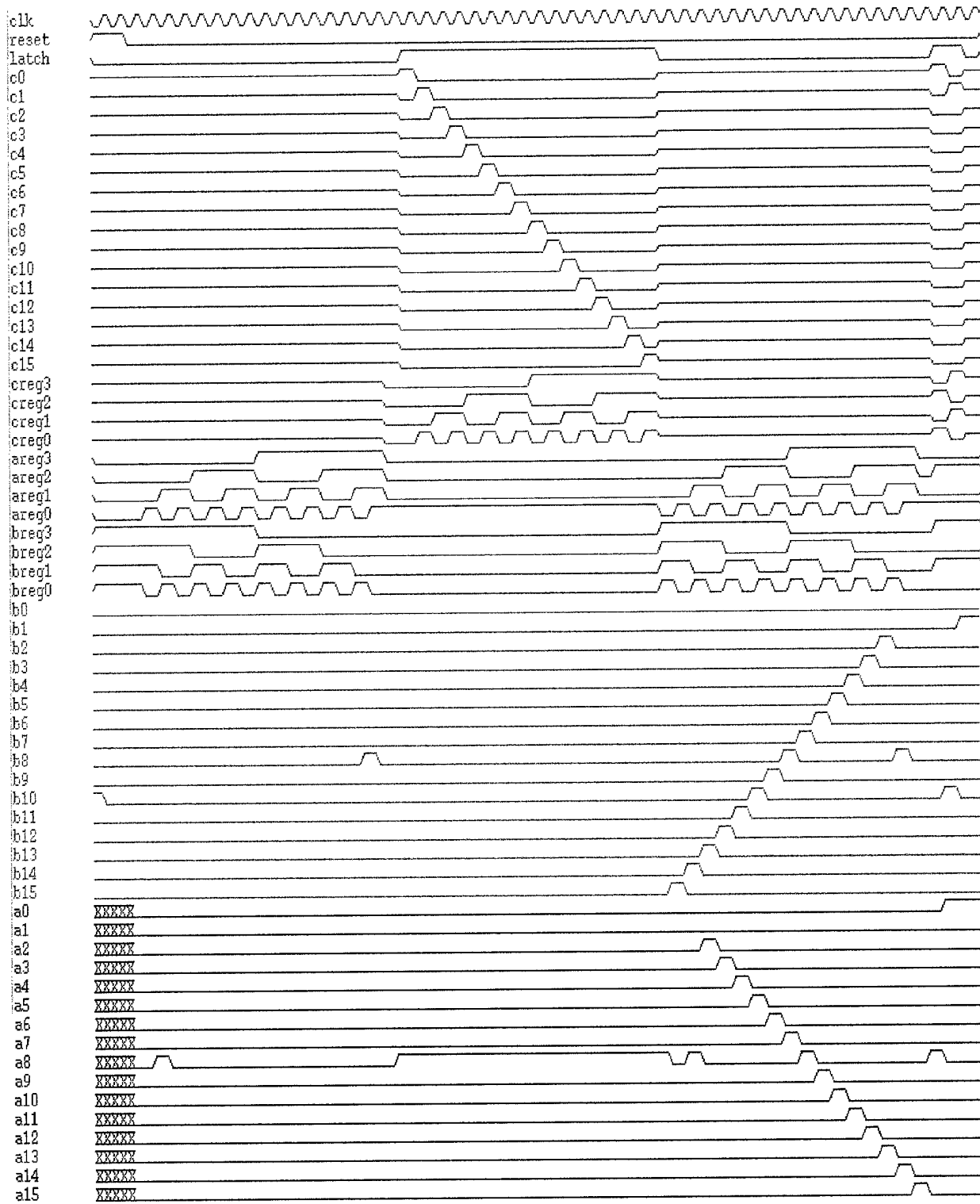
Figure 6.4. IMS Waveform Results of Register File

6.5   Conclusions

This chapter showed the physical implementation and electrical verification of only the half sized register file that was synthesized in Chapter 5. A Xilinx 4020E FPGA was configured from the host PC using a custom adapter board and electrically tested by using the IMS test station. The entire FKP model could not be implemented because the size of the design. It would require multiple 4020E FPGAs or possibly one FPGA from a higher density device, both of which were not available at the time of implementation. However, the success of the half sized register file indicates that the entire FKP model could have also been implemented successfully, assuming the model is correct and a multi-device partitioner program is available.

7. Conclusions and Recommendations for Future Work

7.1   Conclusions

The objective of this research was to implement the forward kinematic algorithm for the Utah MIT Dexterous Hand (UMDH) by creating VHDL models and directly synthesizing them into an FPGA. The forward kinematics of the UMDH were developed and analyzed and the resulting algorithm shows that 12 separate equations each containing multiple mathematical operations are needed. If common expressions are shared between equations, a total of 28 operations are required. These shared terms are stored in the register file unit and are sent to either a cosine/sine unit, an adder/subtractor unit, or a multiplier unit as the algorithm proceeds. The input (angles) and output (transformation matrix) are transferred through dedicated I/O buses. The design results in a semi-autonomous Forward Kinematic Processor (FKP) that can calculate the forward kinematics every time the surrounding system issues a run command. The surrounding system does not deal with the intricacies of the algorithm and can tackle other system tasks while the FKP is busy.

It was planned that the entire algorithm would fit into a single FPGA. However, without the availability of high density FPGAs in the laboratory, only a small portion of the design was able to become realized in hardware. The register file unit was chosen as the sub-model to implement because it contains combinational logic similar to all the other units plus memory storage. After a few iterations with the floorplanning tools, the register file itself proved to be larger than one 4020E FPGA. The register file unit was reduced to half its size and resynthesized.

The new design successfully fit using 40% of the configurable logic blocks of the 4020E. The design was programmed into a 4020E FPGA and tested using an IMS Logic Master XL. Electrical verification shows an upper bound on the clock frequency to be 10.3 MHz, above which the registers begin to hold incorrect data.

7.2 Lessons Learned

It can be concluded that small designs can accurately map into the FPGA and with short turn-around times. The Xilinx 4020E does not have the capacity that was initially expected and proved to be too small for the entire FKP design. The FKP core model and everything underneath is completely synthesizable. This required some restrictions on the coding style to avoid multiple signals in sensitivity lists, multiple wait statements in a process, and any reference to a specific delay of time.

7.3   Recommendations

The first issue to be addressed is the optimization of the VHDL code for synthesis. Some VHDL compilers support the use of in-line macro declarations for instantiation of complete structures such as fast adders already designed into the device. The use of such structures can not only speed up the design, but also take up less FPGA area. Secondly, this research focused solely on Xilinx devices. Using other vendors products such as Altera's MAX Plus II software and their Flex10K series of FPGAs may produce better or worse results. Third, portions of the FKP itself could be redesigned. The multiplier unit uses a 32-bit adder as on of its components. The adder/subtractor unit is 16 bits by itself. The two units could be merged into an ALU, thus eliminating the 16-bit adder and allowing all additions and subtractions to pass through the 32-bit

component of the ALU. The increased overhead to choose either multiplication or addition/subtraction should be minimal compared to the area saved by removing the 16-bit adder/subtractor unit. Fourth, investigation into partitioning tools for Xilinx devices may allow the design to be spread across multiple FPGAs. Last, the microstore and contoller units are not entirly synthesizable. Both need to be modified to adhere to the synthesis restrictions.

7.4 Ideas for Future Work

The architecture of the design could be modified to resemble more of a macropipeline structure. The core could be divided into three parts. The first part would calculate the angles needed. The second part would calculate the sines and cosines. The third part would perform the multiplications, additions and subtractions. The result would be a higher throughput system but with a two stage delay to get the answers. On the other hand, the two data buses, one input and one output, could be merged into a single I/O bus.

The design was based on the idea of the functional units each being a separate state machine and synchronously handshaking with the control unit. This allowed all timing propagation delays within the CLBs, IOBs and routing to be ignored. The result is a design that may waste time during a stage that is simple because the stage that requires the longest time restricts the rest of the design from going any faster. A possible better approach would be a more combinational, less state machine design. This would require knowledge of the delays of the circuit as it is placed into the FPGA.

Different algorithms such as the inverse kinematics of the UMDH or a gross/fine motion controller could be investigated using the same concepts and procedures developed here.

The investigation into PROM development for truly portable systems should be addressed. The PROM device can serially download the configuration of the FPGA every time the system powers up. This property of the FPGA also allows dynamic reconfiguration of parts of the design, allowing the controller of the FKP to swap in and out functional units as needed.

Bibliography

Ailes, John W. Automatic Digital Hardware Synthesis Using VHDL. MS Thesis, Naval Post
Graduate School, Monterey Ca, September 1990. (AD A246 976)

Cohen, Ben. VHDL Coding Styles and Methodologies: An In-Depth Tutorial. Norwell Ma:
Kluwer Academic Publishers, 1995.

Craig, John J. Introduction to Robotics: Mechanics and Control. Reading Ma: Addison-Wesley,
1989.

Exemplar Logic. HDL Synthesis Guide: Release 4.0. Alameda Ca: 1996

----. Leonardo User's Guide: Release 4.0. Alameda Ca: 1996

Integrated Measurement Systems. Logic Master Series: Product Training Manual XL.
Beaverton Or: no date.

----. Verification Solutions: A Guide to Design Verification and Test. Beaverton Or: 1988.

Khosla, P. K. and D.B Stewart. Program Documentation, Chimera 3.1: The Real Time
Operating System for Reconfigurable Sensor Based Control Systems. Carnegie Mellon
University, The Robotics Institute, Department of Electrical and Computer Engineering:
1993.

Narasimhan, Sundar, David M. Siegel, and John M. Hollerbach. A Standard Architecture for
Controlling Robots. Massachusetts Institute of Technology, Artificial Intelligence
Laboratory. July 1988 (AD A195 929)

Raines, Rick Capt. USAF. Class Notes, CSCE 487, Intro to Digital System Design. School of
Engineering, Air Force Institute of Technology, Wright Patterson AFB OH, Summer
Quarter 1996.

Rattan, Kuldip. Class Notes, Ceg 456, Introduction to Robotics. Wright State University,
Dayton OH, Spring Quarter 1995.

Sarcos Incorporated. Hand Electronics Documentation Package. Salt Lake City Utah: March
1987.

Solanki, Ranvir Singh and Kuldip S. Rattan. A Kinematic Study of the Merlin 6500 Robot and
the UTAH/MIT Dexterous Hand and a Simulation on their Combined Behavior.
AAMRL-TR-88-059. Wright Patterson AFB, OH: Harry G. Armstrong Aerospace
Medical Research Laboratory, September 1988 (AD-A203 907).

Synopsys. Iview On-line Documentation: Analyzer and Simulator V3.3b. Mountan View Ca:
    1997

Synopsys. Iview On-line Documentation: FPGA Compiler. Mountan View Ca: 1994

Weste, Neil H. E. and Kamran Eshraghian. Principles of CMOS VLSI Design: A System
    Perspective. Reading Ma: Addison-Wesley, 1993.

Wind River Systems. VX-Works. Alamed Ca.

XACTstep. Version M1, IBM, CDROM. Computer Software. Xilinx, San Jose Ca: 1997.

Xilinx. The Programmable Logic Data Book. San Jose Ca: 1996.

----. XACT Hardware & Peripherals Guide. San Jose Ca: April, 1994.

----. XACT User Guide. San Jose Ca: April, 1994.

----. Xilinx Alliance Series: Quick Start Guide vM1.3. San Jose Ca: 1997.

Appendix A: Code for behavioral Algorithm

A.1   C code

"umdh.h" c code header file

```
/* ****************************************************************   */
/*                                                                    */
/*    umdh.h                                                          */
/*                                                                    */
/*       Steve Parmley                                                */
/* _____                          */
/*                                                                    */
/*    Defines kinematic parameters of umdh thumb manipulators.        */
/*                                                                    */
/* ****************************************************************   */


#define UMDH_A0   (-0.75)
#define UMDH_A1   (0.375)
#define UMDH_A2   (1.7)
#define UMDH_A3   (1.3)
#define UMDH_D1   (3.125)
#define UMDH_D2   (0.0)
#define UMDH_D3   (0.0)
#define UMDH_D4   (0.0)
```
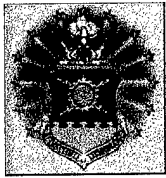
"range.c" c code

```
/* ************************************************************ */
/*                                                             */
/*  range.c                                                    */
/*                                                             */
/*  Steve Parmley  -  UMHD forward kinematic function          */
/* _____                */
/*                                                             */
/*                                                             */
/*  Compute forward kinematics given current joint positions   */
/*  and writes all temp values to disk                         */
/*  Compile with  gcc range.c -lm                              */
/* ************************************************************ */


/* ************************************************************ */
/*         include files                                       */
/* ************************************************************ */

#include <math.h>
#include "umdh.h"
#include <stdio.h>
#include <stdlib.h>


/* ************************************************************ */
/*        umdhFwdKin              Compute forward kinematics.   */
/* ************************************************************ */

void  umdhFwdKin(float *jtang, float *noap, FILE *rangeptr)
{
  float      a0,a1,a2,a3, d1,d2,d3,d4;
  float      c1, c2, c3, c4;
  float s1, s2, s3, s4;
  float c23,s23,c234,s234;


  a0 = UMDH_A0;
  a1 = UMDH_A1;
  a2 = UMDH_A2;
  a3 = UMDH_A3;
  d1 = UMDH_D1;
  d2 = UMDH_D2;
  d3 = UMDH_D3;
  d4 = UMDH_D4;

  s1 = sin(jtang[0]);    c1 = cos(jtang[0]);
  s2 = sin(jtang[1]);    c2 = cos(jtang[1]);
  s3 = sin(jtang[2]);    c3 = cos(jtang[2]);
  s4 = sin(jtang[3]);    c4 = cos(jtang[3]);
  s23 = s2*c3 + c2*s3;   c23 = c2*c3 - s2*s3;
  s234 = sin(jtang[1]+jtang[2]+jtang[3]);
  c234 = cos(jtang[1]+jtang[2]+jtang[3]);

  fprintf(rangeptr,"%f\n%f\n%f\n%f\n%f\n%f\n%f\n%f\n",s1,s2,s3,s4,c1,c2,c3,c4);
  fprintf(rangeptr,"%f\n%f\n%f\n%f\n%f\n",s2*c3,c2*s3,s23,c23);


  /* n vector         */

  noap[0] = c1*c234;
  noap[1] = s1*c234;
  noap[2] = s234;


  /* o vector         */
```

```c
noap[3] = -c1*s234;
noap[4] = -s1*s234;
noap[5] = c234;


/* a vector        */

noap[6] = s1;
noap[7] = -c1;
noap[8] = 0.0;


/* p vector        */

noap[9] = a0 + c1*(a1 + a2*c2 + a3*c23);
noap[10] = s1*(a1 + a2*c2 + a3*c23);
noap[11] = a2*s2 + a3*s23 + d1;


fprintf(rangeptr,"%f\n%f\n%f\n%f\n%f\n",a3*c23,
                                        a2*c2,
                                        a1+a2*c2+a3*c23,
                                        c1*(a1+a2*c2+a3*c23),
                                        s1*(a1+a2*c2+a3*c23));
return;
}


main ()
{
FILE *fp;
FILE *rangeptr;

float  jtang[6];
float  noap[12];
float  step = 3.1415 /8.0;

fp = fopen("fwdkin.dat","w");
rangeptr = fopen("range.dat","w");

for (jtang[0]=-3.1415/4.0;jtang[0] < 3.1415 / 4.0*3.0; jtang[0]=jtang[0]+step)
  for (jtang[1]=0.0;jtang[1] < 3.1415 / 3.0; jtang[1]=jtang[1]+step)
    for (jtang[2]=0.0;jtang[2] < 3.1415 / 2.0; jtang[2]=jtang[2]+step)
      for (jtang[3]=0.0;jtang[3] < 3.1415 / 2.0; jtang[3]=jtang[3]+step)
          {
          umdhFwdKin(jtang,noap,rangeptr);
              fprintf(fp,"%f\t%f\t%f\t%f\n",jtang[0],jtang[1],jtang[2],jtang[3]);
              fprintf(fp,"%f\t%f\t%f\t%f\n" ,noap[0],noap[3],noap[6],noap[9]);
              fprintf(fp,"%f\t%f\t%f\t%f\n" ,noap[1],noap[4],noap[7],noap[10]);
              fprintf(fp,"%f\t%f\t%f\t%f\n\n",noap[2],noap[5],noap[8],noap[11]);
          }

fclose(fp);
fclose(rangeptr);

}
```

A.2   Matlab code

"fk.m" Matlab code

```
% Steve Parmley                           %
% Matlab code that loads data generated by C code %
% Plots positions of last joint and arc of fingertip     %


clear;
close all;
load fwdkin.dat;
for i=1:599,
  nx(i)=fwdkin(i*4+2,1);
  ny(i)=fwdkin(i*4+3,1);
  nz(i)=fwdkin(i*4+4,1);
  ox(i)=fwdkin(i*4+2,2);
  oy(i)=fwdkin(i*4+3,2);
  oz(i)=fwdkin(i*4+4,2);
  ax(i)=fwdkin(i*4+2,3);
  ay(i)=fwdkin(i*4+3,3);
  az(i)=fwdkin(i*4+4,3);
  px(i)=fwdkin(i*4+2,4);
  py(i)=fwdkin(i*4+3,4);
  pz(i)=fwdkin(i*4+4,4);

  ppx(i) = px(i) + nx(i) * 1.125;
  ppy(i) = py(i) + ny(i) * 1.125;
  ppz(i) = pz(i) + nz(i) * 1.125;
end;

for i=1:24,
  px1(i) = px(i);
  py1(i) = py(i);
  pz1(i) = pz(i);
  ppx1(i) = ppx(i);
  ppy1(i) = ppy(i);
  ppz1(i) = ppz(i);

  px2(i) = px(i+24);
  py2(i) = py(i+24);
  pz2(i) = pz(i+24);
  ppx2(i) = ppx(i+24);
  ppy2(i) = ppy(i+24);
  ppz2(i) = ppz(i+24);

  px3(i) = px(i+49);
  py3(i) = py(i+49);
  pz3(i) = pz(i+49);
  ppx3(i) = ppx(i+49);
  ppy3(i) = ppy(i+49);
  ppz3(i) = ppz(i+49);

  px4(i) = px(i+74);
  py4(i) = py(i+74);
  pz4(i) = pz(i+74);
  ppx4(i) = ppx(i+74);
  ppy4(i) = ppy(i+74);
  ppz4(i) = ppz(i+74);

  px5(i) = px(i+149);
  py5(i) = py(i+149);
  pz5(i) = pz(i+149);
  ppx5(i) = ppx(i+149);
  ppy5(i) = ppy(i+149);
  ppz5(i) = ppz(i+149);
```

```
px6(i) = px(i+224);
py6(i) = py(i+224);
pz6(i) = pz(i+224);
ppx6(i) = ppx(i+224);
ppy6(i) = ppy(i+224);
ppz6(i) = ppz(i+224);


px7(i) = px(i+299);
py7(i) = py(i+299);
pz7(i) = pz(i+299);
ppx7(i) = ppx(i+299);
ppy7(i) = ppy(i+299);
ppz7(i) = ppz(i+299);

end;



plot3(ppx1,ppy1,ppz1,'-',px1,py1,pz1,'+',ppx2,ppy2,ppz2,'--',px2,py2,pz2,'o',ppx3,ppy3,ppz3,'-.',px3,py3,pz3,'x');
grid;
view(-45,10);
axis([-3 3 -6 0 1 7]);
title ('UMDH Thumb Motion (joint 0 fixed)');
h=legend('Fingertip Positions (Joint 2 Location A)','Joint 3 Positions (Joint 2 Location A)','Fingertip Positions (Joint 2 Location B)','Joint 3
Positions (Joint 2 Location B)','Fingertip Positions (Joint 2 Location C)','Joint 3 Positions (Joint 2 Location C)');
axes(h);
```

"range.m" Matlab code

```
% Steve Parmley                          %
% Matlab code that loads data generated by C code %
% Plots positions of last joint and arc of fingertip      %

load fwdkin.dat;
max(fwdkin)
min(fwdkin)

load range.dat;
max(range)
min(range)
```
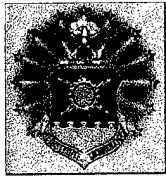
Appendix B: VHDL Functional Unit Models
and Simulation Testbenches

B.1 Cosine/Sine Unit

B.1.1   Cosine/Sine Model

```
-- Project:                      Thesis
-- Filename:                     cos_sin.vhd
-- Other files required:
-- Date:                         Sept 19 97
-- Entity/Architecture Name:     cos_sin_e/behavior
-- Developer:                    Steve Parmley


library IEEE;
  use IEEE.std_logic_1164.all;

entity cos_sin_e is
  port (cos_sin_reset        :       in      std_ulogic;
        cos_sin_clk          :       in      std_ulogic;
        cos_sin_A_bus        :       in      std_ulogic_vector(15 downto 0);
        cos_sin_go           :       in      std_ulogic;
        cos_sin_sel          :       in      std_ulogic;
        cos_sin_wait         :       in      std_ulogic_vector(2 downto 0);
        cos_sin_ready        :       out     std_ulogic;
        cos_sin_C_bus        :       out     std_ulogic_vector(15 downto 0);
     -- the following describes the connection to the rom
        cos_sin_rom_addr     :       out     std_ulogic_vector(12 downto 0);
        cos_sin_rom_data     :       in      std_ulogic_vector(15 downto 0));
end cos_sin_e;

architecture behavior of cos_sin_e is
begin

 lookup: process
variable state : integer;
variable wait_count, wait_counter : integer;

-- create sinks for four bits not used of A_bus
variable temp1,temp2,temp3,temp4 : std_ulogic;

  begin

        if cos_sin_reset = '1' then
                state := 0;
        end if;

        wait until (cos_sin_clk'event and cos_sin_clk='1');

        if state = 0 then
                -- turn off all signals
                cos_sin_ready <= '0';

                -- calculate how many waits
                wait_count := 0;
                wait_counter := 0;
                if cos_sin_wait(0) = '1' then
```
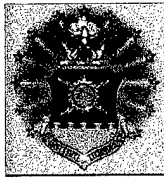
```
                        wait_count := wait_count + 1;
            end if;
            if cos_sin_wait(1) = '1' then
                        wait_count := wait_count + 2;
            end if;
            if cos_sin_wait(2) = '1' then
                        wait_count := wait_count + 4;
            end if;
            -- copy over lower 8 decimal bits and 3 LSBs of integer
            cos_sin_rom_addr(10 downto 0) <= cos_sin_A_bus(10 downto 0);
            -- copy in sign bit
            cos_sin_rom_addr(11) <= cos_sin_A_bus(15);
            -- copy in selector bit for cos or sin function
            cos_sin_rom_addr(12) <= cos_sin_sel;


            -- sink the 4 unused bits
            temp1 := cos_sin_A_bus(11);
            temp2 := cos_sin_A_bus(12);
            temp3 := cos_sin_A_bus(13);
            temp4 := cos_sin_A_bus(14);

            -- wait for go signal
            if cos_sin_go = '1' then
                        state := 1;
            end if;
end if;
if state = 1 then
            -- induce rom wait states for slower external devices
            if wait_count = wait_counter then
                        state := 2;
            else
                        wait_counter := wait_counter + 1;
            end if;
end if;
if state = 2 then
            -- latch data
            cos_sin_C_bus <= cos_sin_rom_data;
            -- indicate to control that the information is latched
            cos_sin_ready <= '1';
            -- wait one cycle and
            state := 3;
elsif state = 3 then
            -- ready signal
            cos_sin_ready <= '0';
            -- start over
            state := 0;
end if;

end process lookup;
end behavior;
```

B.1.2   Cosine/Sine Testbench

```
-- Project:                        Thesis
-- Filename:                       cos_sin-bench.vhd
-- Other files required:
-- Date:                           sept 19 97
-- Entity/Architecture Name:       cos_sin_tb/test
-- Developer:                      Steve Parmley


library IEEE;
  use IEEE.std_logic_1164.all;




entity cos_sin_tb is
end cos_sin_tb;

architecture test of cos_sin_tb is

component cos_sin_e
  port (cos_sin_reset         :       in      std_ulogic;
        cos_sin_clk           :       in      std_ulogic;
        cos_sin_A_bus         :       in      std_ulogic_vector(15 downto 0);
        cos_sin_go            :       in      std_ulogic;
        cos_sin_sel           :       in      std_ulogic;
        cos_sin_wait          :       in      std_ulogic_vector(2 downto 0);
        cos_sin_ready         :       out     std_ulogic;
        cos_sin_C_bus         :       out     std_ulogic_vector(15 downto 0);
        -- the following describes the connection to the rom
        cos_sin_rom_addr  :           out     std_ulogic_vector(12 downto 0);
        cos_sin_rom_data  :           in      std_ulogic_vector(15 downto 0));
end component;

signal sys_reset, sys_clk, go, sel, ready : std_ulogic := '0';
signal waits           : std_ulogic_vector(2 downto 0) := "000";
signal angle_in : std_ulogic_vector(15 downto 0);
signal result          : std_ulogic_vector(15 downto 0);
signal rom_address : std_ulogic_vector(12 downto 0);
signal rom_result  : std_ulogic_vector(15 downto 0);

begin
U1 : cos_sin_e
  PORT MAP   (sys_reset,
                    sys_clk,
                    angle_in,
                    go,
                    sel,
                    waits,
                    ready,
                    result,
                    rom_address,
                    rom_result);

clock : process
begin
        sys_clk <= not(sys_clk);
        wait for 10 ps;
end process clock;

rst : process
begin
        sys_reset <= '1';
```

```vhdl
                wait for 5 ps;
                sys_reset <= '0';
                wait for 15000 ps;
end process rst;

exercise : process
  variable wait_count : integer := 0;
  begin
                -- do it again with more waits
                case wait_count is
                        when 0 => waits <= "000";
                        when 1 => waits <= "001";
                        when 2 => waits <= "010";
                        when 3 => waits <= "011";
                        when 4 => waits <= "100";
                        when 5 => waits <= "101";
                        when 6 => waits <= "110";
                        when 7 => waits <= "111";
                        when others =>    wait until sys_clk'event and sys_clk='1';
                                          wait until sys_clk'event and sys_clk='1';
                                          wait until sys_clk'event and sys_clk='1';
                                          wait until sys_clk'event and sys_clk='1';
                                          wait until sys_clk'event and sys_clk='1';
                                          wait until sys_clk'event and sys_clk='1';
                                          ASSERT false
                                REPORT "DONE"
                                SEVERITY failure;
                        end case;

        wait_count := wait_count + 1;

        wait until sys_clk'event and sys_clk='0';
        -- processor is setting up input bus
        angle_in(15 downto 1) <= "000100100011010";
        angle_in(0) <= waits(0);
        -- set selection to sin or cos
        sel <= waits(0);

        -- wait for a while
        wait until sys_clk'event and sys_clk='1';
        -- and initiate function
        go <= '1';

        -- wait for function to report ready
        wait until ready = '1' and ready'event;

        wait until sys_clk'event and sys_clk='1';

        -- turn off go signal
        go <= '0';



  end process exercise;

rom : process
  begin
        wait until rom_address'event;

        -- make up some rom data (inverse of the address for now)
        rom_result(12 downto 0) <= not(rom_address(12 downto 0));

        -- fill in the rest
        rom_result(15 downto 13) <= "111";
end process rom;
```

end test;

```
CONFIGURATION cos_sin_c OF cos_sin_tb IS
  FOR test
    FOR ALL: cos_sin_e
      USE ENTITY WORK.cos_sin_e(behavior);
    END FOR;
  END FOR;
END cos_sin_c;
```

B.1.3 Cosine/Sine Results
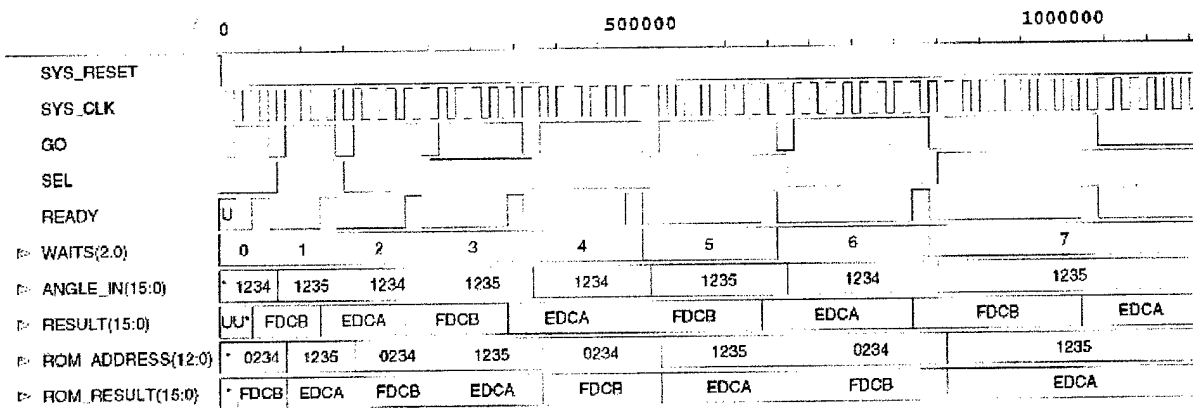
B.2 Adder/Subtractor Unit

B.2.1 Adder/Subtractor Model

```
-- Project:                        Thesis
-- Filename:                       adder.vhd
-- Other files required:
-- Date:                           sept 30 97
-- Entity/Architecture Name:       adder_e/behavior
-- Developer:                      Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
  use IEEE.std_logic_1164.all;

entity adder_e is
  port (adder_reset          :    in      std_ulogic;
            adder_clk         :    in      std_ulogic;
            adder_A_bus       :    in      std_ulogic_vector(15 downto 0);
            adder_B_bus       :    in      std_ulogic_vector(15 downto 0);
            adder_go          :    in      std_ulogic;
            adder_sel         :    in      std_ulogic;
            adder_done        :    out     std_ulogic;
            adder_C_bus       :    out     std_ulogic_vector(15 downto 0));
end adder_e;

architecture behavior of adder_e is
Signal state : integer;
Signal Bxor : std_ulogic_vector(15 downto 0);
Signal Cout : std_ulogic_vector(15 downto 0);
Signal SUM : std_ulogic_vector(15 downto 0);

begin

 addsub : process

  begin

  wait until adder_clk'event and adder_clk='1';

  if adder_reset = '1' then
          state <= 0;
  end if;

  if adder_go = '1' then

   if state = 0 then
           Bxor(0) <= adder_B_bus(0) xor adder_sel;
           Bxor(1) <= adder_B_bus(1) xor adder_sel;
           Bxor(2) <= adder_B_bus(2) xor adder_sel;
           Bxor(3) <= adder_B_bus(3) xor adder_sel;
           Bxor(4) <= adder_B_bus(4) xor adder_sel;
           Bxor(5) <= adder_B_bus(5) xor adder_sel;
           Bxor(6) <= adder_B_bus(6) xor adder_sel;
           Bxor(7) <= adder_B_bus(7) xor adder_sel;
           Bxor(8) <= adder_B_bus(8) xor adder_sel;
           Bxor(9) <= adder_B_bus(9) xor adder_sel;
           Bxor(10) <= adder_B_bus(10) xor adder_sel;
```

```
            Bxor(11) <= adder_B_bus(11) xor adder_sel;
            Bxor(12) <= adder_B_bus(12) xor adder_sel;
            Bxor(13) <= adder_B_bus(13) xor adder_sel;
            Bxor(14) <= adder_B_bus(14) xor adder_sel;
            Bxor(15) <= adder_B_bus(15) xor adder_sel;
            state <= 1;
    elsif state = 1 then
            Cout(0)    <= ((adder_A_bus(0) and Bxor(0)) or (adder_sel and (adder_A_bus(0) or Bxor(0)) ));
            state <= 2;
    elsif state = 2 then
            SUM(0)   <= ((adder_A_bus(0) and Bxor(0) and adder_sel) or ((adder_A_bus(0) or Bxor(0) or adder_sel) and (not Cout(0))));
            Cout(1)   <= ((adder_A_bus(1) and Bxor(1)) or (Cout(0) and (adder_A_bus(1) or Bxor(1)) ));
            state <= 3;
    elsif state = 3 then
            SUM(1)   <= ((adder_A_bus(1) and Bxor(1) and Cout(0)) or ((adder_A_bus(1) or Bxor(1) or Cout(0)) and (not Cout(1))));
            Cout(2)   <= ((adder_A_bus(2) and Bxor(2)) or (Cout(1) and (adder_A_bus(2) or Bxor(2)) ));
            state <= 4;
    elsif state = 4 then
            SUM(2)   <= ((adder_A_bus(2) and Bxor(2) and Cout(1)) or ((adder_A_bus(2) or Bxor(2) or Cout(1)) and (not Cout(2))));
            Cout(3)   <= ((adder_A_bus(3) and Bxor(3)) or (Cout(2) and (adder_A_bus(3) or Bxor(3)) ));
            state <= 5;
    elsif state = 5 then
            SUM(3)   <= ((adder_A_bus(3) and Bxor(3) and Cout(2)) or ((adder_A_bus(3) or Bxor(3) or Cout(2)) and (not Cout(3))));
            Cout(4)   <= ((adder_A_bus(4) and Bxor(4)) or (Cout(3) and (adder_A_bus(4) or Bxor(4)) ));
            state <= 6;
    elsif state = 6 then
            SUM(4)   <= ((adder_A_bus(4) and Bxor(4) and Cout(3)) or ((adder_A_bus(4) or Bxor(4) or Cout(3)) and (not Cout(4))));
            Cout(5)   <= ((adder_A_bus(5) and Bxor(5)) or (Cout(4) and (adder_A_bus(5) or Bxor(5)) ));
            state <= 7;
    elsif state = 7 then
            SUM(5)   <= ((adder_A_bus(5) and Bxor(5) and Cout(4)) or ((adder_A_bus(5) or Bxor(5) or Cout(4)) and (not Cout(5))));
            Cout(6)   <= ((adder_A_bus(6) and Bxor(6)) or (Cout(5) and (adder_A_bus(6) or Bxor(6)) ));
            state <= 8;
    elsif state = 8 then
            SUM(6)   <= ((adder_A_bus(6) and Bxor(6) and Cout(5)) or ((adder_A_bus(6) or Bxor(6) or Cout(5)) and (not Cout(6))));
            Cout(7)   <= ((adder_A_bus(7) and Bxor(7)) or (Cout(6) and (adder_A_bus(7) or Bxor(7)) ));
            state <= 9;
    elsif state = 9 then
            SUM(7)   <= ((adder_A_bus(7) and Bxor(7) and Cout(6)) or ((adder_A_bus(7) or Bxor(7) or Cout(6)) and (not Cout(7))));
            Cout(8)   <= ((adder_A_bus(8) and Bxor(8)) or (Cout(7) and (adder_A_bus(8) or Bxor(8)) ));
            state <= 10;
    elsif state = 10 then
            SUM(8)   <= ((adder_A_bus(8) and Bxor(8) and Cout(7)) or ((adder_A_bus(8) or Bxor(8) or Cout(7)) and (not Cout(8))));
            Cout(9)   <= ((adder_A_bus(9) and Bxor(9)) or (Cout(8) and (adder_A_bus(9) or Bxor(9)) ));
            state <= 11;
    elsif state = 11 then
            SUM(9)   <= ((adder_A_bus(9) and Bxor(9) and Cout(8)) or ((adder_A_bus(9) or Bxor(9) or Cout(8)) and (not Cout(9))));
            Cout(10)<= ((adder_A_bus(10) and Bxor(10)) or (Cout(9) and (adder_A_bus(10) or Bxor(10)) ));
            state <= 12;
    elsif state = 12 then
            SUM(10)  <= ((adder_A_bus(10) and Bxor(10) and Cout(9)) or ((adder_A_bus(10) or Bxor(10) or Cout(9)) and (not
Cout(10))));
            Cout(11)<= ((adder_A_bus(11) and Bxor(11)) or (Cout(10) and (adder_A_bus(11) or Bxor(11)) ));
            state <= 13;
    elsif state = 13 then
            SUM(11)  <= ((adder_A_bus(11) and Bxor(11) and Cout(10)) or ((adder_A_bus(11) or Bxor(11) or Cout(10)) and (not
Cout(11))));
            Cout(12)<= ((adder_A_bus(12) and Bxor(12)) or (Cout(11) and (adder_A_bus(12) or Bxor(12)) ));
            state <= 14;
    elsif state = 14 then
            SUM(12)  <= ((adder_A_bus(12) and Bxor(12) and Cout(11)) or ((adder_A_bus(12) or Bxor(12) or Cout(11)) and (not
Cout(12))));
            Cout(13)<= ((adder_A_bus(13) and Bxor(13)) or (Cout(12) and (adder_A_bus(13) or Bxor(13)) ));
            state <= 15;
    elsif state = 15 then
```

```
          SUM(13) <= ((adder_A_bus(13) and Bxor(13) and Cout(12)) or ((adder_A_bus(13) or Bxor(13) or Cout(12)) and (not
Cout(13))));
          Cout(14)<= ((adder_A_bus(14) and Bxor(14)) or (Cout(13) and (adder_A_bus(14) or Bxor(14)) ));
          state <= 16;
    elsif state = 16 then
          SUM(14) <= ((adder_A_bus(14) and Bxor(14) and Cout(13)) or ((adder_A_bus(14) or Bxor(14) or Cout(13)) and (not
Cout(14))));
          Cout(15)<= ((adder_A_bus(15) and Bxor(15)) or (Cout(14) and (adder_A_bus(15) or Bxor(15)) ));
          state <= 17;
    elsif state = 17 then
          SUM(15) <= ((adder_A_bus(15) and Bxor(15) and Cout(14)) or ((adder_A_bus(15) or Bxor(15) or Cout(14)) and (not
Cout(15))));
          state <= 18;
    elsif state = 18 then
          adder_C_bus <= SUM;
          adder_done <= '1';
    end if;
    else
          adder_done <= '0';
          state <= 0;

    end if;


  end process addsub;
end behavior;
```

B.2.2   Adder/Subtractor Testbench

```
-- Project:              Thesis
-- Filename:             adder-bench.vhd
-- Other files required:
-- Date:                 sept 30 97
-- Entity/Architecture Name:   adder_tb/test
-- Developer:            Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
   use IEEE.std_logic_1164.all;



entity adder_tb is
end adder_tb;

architecture test of adder_tb is

constant   Atest00    : std_ulogic_vector(15 downto 0) := "0000000000000000";
constant   Atest01 : std_ulogic_vector(15 downto 0) := "0000000000000001";
constant   Atest02    : std_ulogic_vector(15 downto 0) := "0000000000000010";
constant   Atest03 : std_ulogic_vector(15 downto 0) := "0000000000000011";
constant   Atest04    : std_ulogic_vector(15 downto 0) := "0101010101010101";
constant   Atest05 : std_ulogic_vector(15 downto 0) := "1010101010101010";
constant   Atest06    : std_ulogic_vector(15 downto 0) := "1111111111111110";
constant   Atest07 : std_ulogic_vector(15 downto 0) := "1111111101111111";
constant   Atest08    : std_ulogic_vector(15 downto 0) := "0111111111111111";
constant   Atest09 : std_ulogic_vector(15 downto 0) := "1111111111111111";
```

```vhdl
constant  Btest00    : std_ulogic_vector(15 downto 0) := "0000000000000000"; -- +/- 0
constant  Btest01 : std_ulogic_vector(15 downto 0) := "0000000000000001"; -- +/- 1
constant  Btest02    : std_ulogic_vector(15 downto 0) := "0000000000000010"; -- +/- 2
constant  Btest03 : std_ulogic_vector(15 downto 0) := "0000000100000000"; -- +/- 256
constant  Btest04    : std_ulogic_vector(15 downto 0) := "1000000000000000"; -- +/- 32K
constant  Btest05 : std_ulogic_vector(15 downto 0) := "1111111111111111"; -- +/- 65534


constant  add : std_ulogic := '0';
constant  sub : std_ulogic := '1';
component adder_e
   port (adder_reset            :       in        std_ulogic;
           adder_clk            :       in        std_ulogic;
           adder_A_bus          :       in        std_ulogic_vector(15 downto 0);
           adder_B_bus          :       in        std_ulogic_vector(15 downto 0);
           adder_go             :       in        std_ulogic;
           adder_sel            :       in        std_ulogic;
           adder_done           :       out       std_ulogic;
           adder_C_bus          :       out       std_ulogic_vector(15 downto 0));
end component;

signal sys_clk,sys_reset, go, sel, done : std_ulogic := '0';
signal A,B, result     : std_ulogic_vector(15 downto 0);


begin
U1 : adder_e
   PORT MAP    (sys_reset,
                  sys_clk,
                  A,
                  B,
                  go,
                  sel,
                  done,
                  result);


clock : process
begin
          sys_clk <= not(sys_clk);
          wait for 10 ps;
end process clock;


exercise : process
  variable inputA, inputB : std_ulogic_vector(15 downto 0);
  begin

  sys_reset <= '0';

  For i in 0 to 1 loop
  -- add or sub

          CASE i IS
                  WHEN 0 => sel <= add;
                  WHEN 1 => sel <= sub;
          END CASE;

          for j in 0 to 9 loop
            for l in 0 to 5 loop
          -- pick a test
                  CASE j IS
                            WHEN 0 => inputA := Atest00;
                            WHEN 1 => inputA := Atest01;
                            WHEN 2 => inputA := Atest02;
                            WHEN 3 => inputA := Atest03;
```

```
                    WHEN 4 => inputA := Atest04;
                    WHEN 5 => inputA := Atest05;
                    WHEN 6 => inputA := Atest06;
                    WHEN 7 => inputA := Atest07;
                    WHEN 8 => inputA := Atest08;
                    WHEN 9 => inputA := Atest09;
            END CASE;
            CASE I IS
                    WHEN 0 => inputB := Btest00;
                    WHEN 1 => inputB := Btest01;
                    WHEN 2 => inputB := Btest02;
                    WHEN 3 => inputB := Btest03;
                    WHEN 4 => inputB := Btest04;
                    WHEN 5 => inputB := Btest05;
            END CASE;



            go <= '0';

            wait until done = '0';


            FOR k IN 0 TO 15 loop
                    A(k) <= inputA(k);
                    B(k) <= inputB(k);
            end loop;

            wait until sys_clk'event and sys_clk='0';

            go <= '1';

            wait until done ='1';

            go <= '0';
        end loop;
      end loop;
  end loop;


            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';



    ASSERT false
            REPORT "DONE"
            SEVERITY failure;
  end process exercise;
end test;
```

```
CONFIGURATION adder_c OF adder_tb IS
  FOR test
    FOR ALL: adder_e
      USE ENTITY WORK.adder_e(behavior);
    END FOR;
  END FOR;
END adder_c;
```

B.2.3 Adder/Subtractor Results

A(15:0)	0000							0001			
B(15:0)	0000	0001	0002	0100	8000	FFFF	0000	0001	0002	0100	
RESULT(15:0)	UUUU	0000	0001	0002	0100	8000	FFFF	0001	0002	0003	0

A(15:0)	0001				0002						
B(15:0)	8000	FFFF	0000	0001	0002	0100	8000	FFFF	0000	0001	0
RESULT(15:0)	0101	8001	0000	0002	0003	0004	0102	8002	0001	0003	0(

A(15:0)	0003				5555						
B(15:0)	0002	0100	8000	FFFF	0000	0001	0002	0100	8000	FFFF	0(
RESULT(15:0)	0004	0005	0103	8003	0002	5555	5556	5557	5655	D555	55

	13000000	14000000	15000000	1600000

SYS_CLK										
GO										
SEL										
DONE										
A(15:0)			AAAA					FFFE		
B(15:0)	0000	0001	0002	0100	8000	FFFF	0000	0001	0002	0100	80
RESULT(15:0)	5554	AAAA	AAAB	AAAC	ABAA	2AAA	AAA9	FFFE	FFFF	0000	00F

	17000000	18000000	19000000	20000000

SYS_CLK										
GO										
SEL										
DONE										
A(15:0)		FFFE				FF7F				
B(15:0)	8000	FFFF	0000	0001	0002	0100	8000	FFFF	0000	0001	000
RESULT(15:0)	00FE	7FFE	FFFD	FF7F	FF80	FF81	007F	7F7F	FF7E	7FFF	800

	21000000	22000000	23000000	24000000

SYS_CLK										
GO										
SEL										
DONE										
A(15:0)		7FFF				FFFF				
B(15:0)	0002	0100	8000	FFFF	0000	0001	0002	0100	8000	FFFF	000
RESULT(15:0)	8000	8001	80FF	FFFF	7FFE	FFFF	0000	0001	00FF	7FFF	FFFF

	25000000	26000000	27000000	28000000

SYS_CLK										
GO										
SEL										
DONE										
A(15:0)			0000					0001		
B(15:0)	0000	0001	0002	0100	8000	FFFF	0000	0001	0002	0100	800
RESULT(15:0)	FF	0000	FFFF	FFFE	FF00	8000	0001		0000	FFFF	FF01

29000000   30000000   31000000   32000000

SYS_CLK
GO
SEL
DONE

A(15:0)	0001	0002								
B(15:0)	8000	FFFF	0000	0001	0002	0100	8000	FFFF	0000	0001	0002
RESULT(15:0)	FF*	8001	0002		0001	0000	FF02	8002	0003	0002

33000000   34000000   35000000   36000000

SYS_CLK
GO
SEL
DONE

A(15:0)	0003		5555							
B(15:0)	00*	0100	8000	FFFF	0000	0001	0002	0100	8000	FFFF	0000
RESULT(15:0)	00*	0001	FF03	8003	0004	5555	5554	5553	5455	D555	5556

37000000   38000000   39000000   40000000

SYS_CLK
GO
SEL
DONE

A(15:0)	AAAA						FFFE			
B(15:0)	00*	0001	0002	0100	8000	FFFF	0000	0001	0002	0100	8000
RESULT(15:0)	5*	AAAA	AAA9	AAA8	A9AA	2AAA	AAAB	FFFE	FFFD	FFFC	FEFE

41000000   42000000   43000000   44000000

SYS_CLK
GO
SEL
DONE

A(15:0)	FFFE	FF7F							7F
B(15:0)	8*	FFFF	0000	0001	0002	0100	8000	FFFF	0000	0001	0002
RESULT(15:0)	F*	7FFE	FFFF	FF7F	FF7E	FF7D	FE7F	7F7F	FF80	7FFF	7FFE

	45000000			46000000			47000000		48000000

SYS CLK

GO

SEL

DONE

▷ **A(15:0)** : 7FFF / FFFF

▷ **B(15:0)** : 0· | 0100 | 8000 | FFFF | 0000 | 0001 | 0002 | 0100 | 8000 | FFFF

▷ **RESULT(15:0)** : ·| 7FFD | 7EFF | FFFF | 8000 | FFFF | FFFE | FFFD | FEFF | 7FFF | 0000

B.3 Multipler Unit

B.3.1 Multiplier Model

```
-- Project:                    Thesis
-- Filename:                   adder32.vhd
-- Other files required:
-- Date:                       sept 30 97
-- Entity/Architecture Name:   adder32_e/behavior
-- Developer:                  Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
  use IEEE.std_logic_1164.all;

entity adder32_e is
  port (adder_reset         :     in      std_ulogic;
              adder_clk      :     in      std_ulogic;
              adder_A_bus    :     in      std_ulogic_vector(31 downto 0);
              adder_B_bus    :     in      std_ulogic_vector(31 downto 0);
              adder_go       :     in      std_ulogic;
              adder_sel      :     in      std_ulogic;
              adder_done     :     out     std_ulogic;
              adder_C_bus    :     out     std_ulogic_vector(31 downto 0));
end adder32_e;

architecture behavior of adder32_e is
Signal state : integer;
Signal Bxor : std_ulogic_vector(31 downto 0);
Signal Cout : std_ulogic_vector(31 downto 0);
Signal SUM : std_ulogic_vector(31 downto 0);

begin

 addsub : process

  begin

  wait until adder_clk'event and adder_clk='1';

  if adder_reset = '1' then
          state <= 0;
  end if;

  if adder_go = '1' then

   if state = 0 then
           Bxor(0) <= adder_B_bus(0) xor adder_sel;
           Bxor(1) <= adder_B_bus(1) xor adder_sel;
           Bxor(2) <= adder_B_bus(2) xor adder_sel;
           Bxor(3) <= adder_B_bus(3) xor adder_sel;
           Bxor(4) <= adder_B_bus(4) xor adder_sel;
           Bxor(5) <= adder_B_bus(5) xor adder_sel;
           Bxor(6) <= adder_B_bus(6) xor adder_sel;
           Bxor(7) <= adder_B_bus(7) xor adder_sel;
           Bxor(8) <= adder_B_bus(8) xor adder_sel;
           Bxor(9) <= adder_B_bus(9) xor adder_sel;
           Bxor(10) <= adder_B_bus(10) xor adder_sel;
```

```
            Bxor(11) <= adder_B_bus(11) xor adder_sel;
            Bxor(12) <= adder_B_bus(12) xor adder_sel;
            Bxor(13) <= adder_B_bus(13) xor adder_sel;
            Bxor(14) <= adder_B_bus(14) xor adder_sel;
            Bxor(15) <= adder_B_bus(15) xor adder_sel;
            Bxor(16) <= adder_B_bus(16) xor adder_sel;
            Bxor(17) <= adder_B_bus(17) xor adder_sel;
            Bxor(18) <= adder_B_bus(18) xor adder_sel;
            Bxor(19) <= adder_B_bus(19) xor adder_sel;
            Bxor(20) <= adder_B_bus(20) xor adder_sel;
            Bxor(21) <= adder_B_bus(21) xor adder_sel;
            Bxor(22) <= adder_B_bus(22) xor adder_sel;
            Bxor(23) <= adder_B_bus(23) xor adder_sel;
            Bxor(24) <= adder_B_bus(24) xor adder_sel;
            Bxor(25) <= adder_B_bus(25) xor adder_sel;
            Bxor(26) <= adder_B_bus(26) xor adder_sel;
            Bxor(27) <= adder_B_bus(27) xor adder_sel;
            Bxor(28) <= adder_B_bus(28) xor adder_sel;
            Bxor(29) <= adder_B_bus(29) xor adder_sel;
            Bxor(30) <= adder_B_bus(30) xor adder_sel;
            Bxor(31) <= adder_B_bus(31) xor adder_sel;
            state <= 1;
    elsif state = 1 then
            Cout(0)    <= ((adder_A_bus(0) and Bxor(0)) or (adder_sel and (adder_A_bus(0) or Bxor(0)) ));
            state <= state + 1;
    elsif state = 2 then
            SUM(state-2)       <= ((adder_A_bus(state-2) and Bxor(state-2) and adder_sel) or ((adder_A_bus(state-2) or Bxor(state-2)
or adder_sel) and (not Cout(state-2))));
            Cout(state-1)      <= ((adder_A_bus(state-1) and Bxor(state-1)) or (Cout(state-2) and (adder_A_bus(state-1) or Bxor(state-
1)) ));
            state <= state + 1;
    elsif state >= 3 and state <= 32 then
            SUM(state-2)       <= ((adder_A_bus(state-2) and Bxor(state-2) and Cout(state-3)) or ((adder_A_bus(state-2) or Bxor(state-
2) or Cout(state-3)) and (not Cout(state-2))));
            Cout(state-1)      <= ((adder_A_bus(state-1) and Bxor(state-1)) or (Cout(state-2) and (adder_A_bus(state-1) or Bxor(state-
1)) ));
            state <= state + 1;
    elsif state = 33 then
            SUM(31) <= ((adder_A_bus(31) and Bxor(31) and Cout(30)) or ((adder_A_bus(31) or Bxor(31) or Cout(30)) and (not
Cout(31))));
            state <= state +1;
    elsif state = 34 then
            adder_C_bus <= SUM;
            adder_done <= '1';
    end if;
  else
            adder_done <= '0';
            state <= 0;

  end if;


 end process addsub;
end behavior;
```

```vhdl
-- History:
-- Last Analyzed On:
_____

library IEEE;
  use IEEE.std_logic_1164.all;

entity mult_e is
  port (mult_reset              :       in      std_ulogic;
              mult_clk          :       in      std_ulogic;
              mult_A_bus        :       in      std_ulogic_vector(15 downto 0);
              mult_B_bus        :       in      std_ulogic_vector(15 downto 0);
              mult_go           :       in      std_ulogic;
              mult_done         :       out     std_ulogic;
              mult_C_bus        :       out     std_ulogic_vector(15 downto 0));
end mult_e;

architecture behavior of mult_e is

Signal state, state2 : integer;
Signal    result00,result01,result02,result03,result04,result05,result06,result07,
          result08,result09,result10,result11,result12,result13,result14,result15
          : std_ulogic_vector(31 downto 0);

signal sys_clk, sys_reset, go, sel, done : std_ulogic := '0';
signal A,B, result      : std_ulogic_vector(31 downto 0);

component adder32_e
  port (adder_reset             :       in      std_ulogic;
              adder_clk         :       in      std_ulogic;
              adder_A_bus       :       in      std_ulogic_vector(31 downto 0);
              adder_B_bus       :       in      std_ulogic_vector(31 downto 0);
              adder_go          :       in      std_ulogic;
              adder_sel         :       in      std_ulogic;
              adder_done        :       out     std_ulogic;
              adder_C_bus       :       out     std_ulogic_vector(31 downto 0));
end component;

begin
U1 : adder32_e
  PORT MAP   (sys_reset,
                    sys_clk,
                    A,
                    B,
                    go,
                    sel,
                    done,
                    result);

  sys_clk <= mult_clk;
  sel <= '0';
  sys_reset <= mult_reset;

  addsub : process

    begin

    wait until mult_clk'event and mult_clk='1';

    if mult_reset = '1' then
            state <= 0;
    end if;

    if mult_go = '1' then
```

```
if state = 0 then
                    result00 <= "00000000000000000000000000000000";
                    result01 <= "00000000000000000000000000000000";
                    result02 <= "00000000000000000000000000000000";
                    result03 <= "00000000000000000000000000000000";
                    result04 <= "00000000000000000000000000000000";
                    result05 <= "00000000000000000000000000000000";
                    result06 <= "00000000000000000000000000000000";
                    result07 <= "00000000000000000000000000000000";
                    result08 <= "00000000000000000000000000000000";
                    result09 <= "00000000000000000000000000000000";
                    result10 <= "00000000000000000000000000000000";
                    result11 <= "00000000000000000000000000000000";
                    result12 <= "00000000000000000000000000000000";
                    result13 <= "00000000000000000000000000000000";
                    result14 <= "00000000000000000000000000000000";
                    result15 <= "00000000000000000000000000000000";

         state <= 1;
elsif state = 1 then
         for i in 0 to 15 loop
             if mult_B_bus(i) = '1' then
                    case i is
                       when 0 => result00(15 downto 0) <= mult_A_bus ;
                       when 1 => result01(16 downto 1) <= mult_A_bus ;
                       when 2 => result02(17 downto 2) <= mult_A_bus ;
                       when 3 => result03(18 downto 3) <= mult_A_bus ;
                       when 4 => result04(19 downto 4) <= mult_A_bus ;
                       when 5 => result05(20 downto 5) <= mult_A_bus ;
                       when 6 => result06(21 downto 6) <= mult_A_bus ;
                       when 7 => result07(22 downto 7) <= mult_A_bus ;
                       when 8 => result08(23 downto 8) <= mult_A_bus ;
                       when 9 => result09(24 downto 9) <= mult_A_bus ;
                       when 10 => result10(25 downto 10) <= mult_A_bus ;
                       when 11 => result11(26 downto 11) <= mult_A_bus ;
                       when 12 => result12(27 downto 12) <= mult_A_bus ;
                       when 13 => result13(28 downto 13) <= mult_A_bus ;
                       when 14 => result14(29 downto 14) <= mult_A_bus ;
                       when 15 => result15(30 downto 15) <= mult_A_bus ;
                       when others =>
                    end case;
             end if;
         end loop;
         state <= 2;

elsif state = 2 then
         go <= '0';
         if done = '0' then
                    A <= result00;
                    B <= result01;
                    state <= 3;
         end if;
elsif state = 3 then
         go <= '1';
         if done = '1' then
                    state <= 4;
                    state2 <= 0;
         end if;


elsif state >= 4 and state <= 15 then
         if state2 = 0 then
                    go <= '0';
                    if done = '0' then
                              A <= result;
```

```
case state is
    when 4 => B <= result02;
    when 5 => B <= result03;
    when 6 => B <= result04;
    when 7 => B <= result05;
    when 8 => B <= result06;
    when 9 => B <= result07;
    when 10 => B <= result08;
    when 11 => B <= result09;
    when 12 => B <= result10;
    when 13 => B <= result11;
    when 14 => B <= result12;
    when 15 => B <= result13;
    when 16 => B <= result14;
    when 17 => B <= result15;
    when others =>
    end case;
    state2 <= 1;

            end if;
    elsif state2 = 1 then
            go <= '1';
            if done = '1' then
                    state2 <= 0;
                    state <= state +1;
            end if;
    end if;

elsif state = 18 then
    mult_C_bus <= result(23 downto 8);
    mult_done <= '1';
end if;
else
    mult_done <= '0';
    state <= 0;
end if;


end process addsub;
end behavior;
```


B.3.2   Multiplier Testbench

```
-- Project:                     Thesis
-- Filename:                    adder32-bench.vhd
-- Other files required:
-- Date:                        sept 30 97
-- Entity/Architecture Name:    adder32_tb/test
-- Developer:                   Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
    use IEEE.std_logic_1164.all;



entity adder32_tb is
end adder32_tb;
```

```vhdl
architecture test of adder32_tb is

constant  Atest00   : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000000";
constant  Atest01 : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000001";
constant  Atest02   : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000010";
constant  Atest03 : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000011";
constant  Atest04   : std_ulogic_vector(31 downto 0) := "01010101010101010101010101010101";
constant  Atest05 : std_ulogic_vector(31 downto 0) := "10101010101010101010101010101010";
constant  Atest06   : std_ulogic_vector(31 downto 0) := "11111111111111111111111111111110";
constant  Atest07 : std_ulogic_vector(31 downto 0) := "11111110111111111111111101111111";
constant  Atest08   : std_ulogic_vector(31 downto 0) := "01111111111111111111111111111111";
constant  Atest09 : std_ulogic_vector(31 downto 0) := "11111111111111111111111111111111";


constant  Btest00   : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000000";
constant  Btest01 : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000001";
constant  Btest02   : std_ulogic_vector(31 downto 0) := "00000000000000000000000000000010";
constant  Btest03 : std_ulogic_vector(31 downto 0) := "00000000000000000000000100000000";
constant  Btest04   : std_ulogic_vector(31 downto 0) := "10000000000000000000000000000000";
constant  Btest05 : std_ulogic_vector(31 downto 0) := "11111111111111111111111111111111";

constant   add : std_ulogic := '0';
constant   sub : std_ulogic := '1';
component adder32_e
   port (adder_reset             :          in        std_ulogic;
                adder_clk          :          in        std_ulogic;
                adder_A_bus        :          in        std_ulogic_vector(31 downto 0);
                adder_B_bus        :          in        std_ulogic_vector(31 downto 0);
                adder_go           :          in        std_ulogic;
                adder_sel          :          in        std_ulogic;
                adder_done         :          out       std_ulogic;
                adder_C_bus        :          out       std_ulogic_vector(31 downto 0));
end component;

signal sys_clk,sys_reset, go, sel, done : std_ulogic := '0';
signal A,B, result      : std_ulogic_vector(31 downto 0);


begin
U1 : adder32_e
   PORT MAP    (sys_reset,
                        sys_clk,
                        A,
                        B,
                        go,
                        sel,
                        done,
                        result);

clock : process
begin
        sys_clk <= not(sys_clk);
        wait for 10 ps;
end process clock;


exercise : process
 variable inputA, inputB : std_ulogic_vector(31 downto 0);
 begin

 sys_reset <= '0';

 For i in 0 to 1 loop
 -- add or sub

        CASE i IS
```

```
                    WHEN 0 => sel <= add;
                    WHEN 1 => sel <= sub;
        END CASE;

    for j in 0 to 9 loop
        for l in 0 to 5 loop
    -- pick a test
            CASE j IS
                        WHEN 0 => inputA := Atest00;
                        WHEN 1 => inputA := Atest01;
                        WHEN 2 => inputA := Atest02;
                        WHEN 3 => inputA := Atest03;
                        WHEN 4 => inputA := Atest04;
                        WHEN 5 => inputA := Atest05;
                        WHEN 6 => inputA := Atest06;
                        WHEN 7 => inputA := Atest07;
                        WHEN 8 => inputA := Atest08;
                        WHEN 9 => inputA := Atest09;
            END CASE;
            CASE l IS
                        WHEN 0 => inputB := Btest00;
                        WHEN 1 => inputB := Btest01;
                        WHEN 2 => inputB := Btest02;
                        WHEN 3 => inputB := Btest03;
                        WHEN 4 => inputB := Btest04;
                        WHEN 5 => inputB := Btest05;
            END CASE;


            go <= '0';

            wait until done = '0';


            FOR k IN 0 TO 31 loop
                    A(k) <= inputA(k);
                    B(k) <= inputB(k);
            end loop;

            wait until sys_clk'event and sys_clk='0';

            go <= '1';

            wait until done ='1';

            go <= '0';
        end loop;
        end loop;
end loop;


                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
```

```
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';
            wait until sys_clk'event and sys_clk='0';


  ASSERT false
        REPORT "DONE"
        SEVERITY failure;
  end process exercise;
end test;

CONFIGURATION adder32_c OF adder32_tb IS
  FOR test
    FOR ALL: adder32_e
      USE ENTITY WORK.adder32_e(behavior);
    END FOR;
  END FOR;
END adder32_c;
```

```
-- Project:                      Thesis
-- Filename:                     mult-bench.vhd
-- Other files required:
-- Date:                         oct 10 97
-- Entity/Architecture Name:     mult_tb/test
-- Developer:                    Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:
```

```
library IEEE;
  use IEEE.std_logic_1164.all;



entity mult_tb is
end mult_tb;

architecture test of mult_tb is

constant  Atest00   : std_ulogic_vector(15 downto 0) := "0000000000000000";
constant  Atest01 : std_ulogic_vector(15 downto 0) := "0000000000000001";
constant  Atest02   : std_ulogic_vector(15 downto 0) := "0000000000000010";
constant  Atest03 : std_ulogic_vector(15 downto 0) := "0000000000000011";
constant  Atest04   : std_ulogic_vector(15 downto 0) := "0101010101010101";
constant  Atest05 : std_ulogic_vector(15 downto 0) := "1010101010101010";
constant  Atest06   : std_ulogic_vector(15 downto 0) := "1111111111111110";
constant  Atest07 : std_ulogic_vector(15 downto 0) := "1111111101111111";
constant  Atest08   : std_ulogic_vector(15 downto 0) := "0111111111111111";
constant  Atest09 : std_ulogic_vector(15 downto 0) := "1111111111111111";


constant  Btest00   : std_ulogic_vector(15 downto 0) := "0000000000000000"; -- +/- 0
constant  Btest01 : std_ulogic_vector(15 downto 0) := "0000000000000001"; -- +/- 1
constant  Btest02   : std_ulogic_vector(15 downto 0) := "0000000000000010"; -- +/- 2
constant  Btest03 : std_ulogic_vector(15 downto 0) := "0000000100000000"; -- +/- 256
constant  Btest04   : std_ulogic_vector(15 downto 0) := "1000000000000000"; -- +/- 32K
constant  Btest05 : std_ulogic_vector(15 downto 0) := "1111111111111111"; -- +/- 65534

constant  add : std_ulogic := '0';
```

```
constant  sub : std_ulogic := '1';
component mult_e
  port (mult_reset              :       in      std_ulogic;
             mult_clk           :       in      std_ulogic;
             mult_A_bus         :       in      std_ulogic_vector(15 downto 0);
             mult_B_bus         :       in      std_ulogic_vector(15 downto 0);
             mult_go            :       in      std_ulogic;
             mult_done          :       out     std_ulogic;
             mult_C_bus         :       out     std_ulogic_vector(15 downto 0));
end component;

signal sys_clk,sys_reset, go, done : std_ulogic := '0';
signal A,B, result      : std_ulogic_vector(15 downto 0);


begin
U1 : mult_e
  PORT MAP   (sys_reset,
                  sys_clk,
                  A,
                  B,
                  go,
                  done,
                  result);


clock : process
begin
          sys_clk <= not(sys_clk);
          wait for 10 ps;
end process clock;



exercise : process
variable   inputA, inputB         : std_ulogic_vector(15 downto 0);
 begin

          sys_reset <= '0';

          for j in 0 to 9 loop
             for I in 0 to 5 loop
          -- pick a test
                    CASE j IS
                            WHEN 0 => inputA := Atest00;
                            WHEN 1 => inputA := Atest01;
                            WHEN 2 => inputA := Atest02;
                            WHEN 3 => inputA := Atest03;
                            WHEN 4 => inputA := Atest04;
                            WHEN 5 => inputA := Atest05;
                            WHEN 6 => inputA := Atest06;
                            WHEN 7 => inputA := Atest07;
                            WHEN 8 => inputA := Atest08;
                            WHEN 9 => inputA := Atest09;
                    END CASE;
                    CASE I IS
                            WHEN 0 => inputB := Btest00;
                            WHEN 1 => inputB := Btest01;
                            WHEN 2 => inputB := Btest02;
                            WHEN 3 => inputB := Btest03;
                            WHEN 4 => inputB := Btest04;
                            WHEN 5 => inputB := Btest05;
                    END CASE;


                    go <= '0';
```

```vhdl
                    wait until done = '0';


                    FOR k IN 0 TO 15 loop
                            A(k) <= inputA(k);
                            B(k) <= inputB(k);
                    end loop;

                    wait until sys_clk'event and sys_clk='0';

                    go <= '1';

                    wait until done ='1';

                    go <= '0';
            end loop;
        end loop;


                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';
                    wait until sys_clk'event and sys_clk='0';


    ASSERT false
            REPORT "DONE"
            SEVERITY failure;
  end process exercise;
end test;

CONFIGURATION mult_c OF mult_tb IS
  FOR test
    FOR ALL: mult_e
      USE ENTITY WORK.mult_e(behavior);
    END FOR;
  END FOR;
END mult_c;
```

B.3.3 Multiplier Results

(Adder32)

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
A(31:0)	00000000					0*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	0*
RESULT(31:0)	UUUUUUUU	00000000	00000001	00000002	00000100	80000000	F*

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
A(31:0)	00000001					0*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	0*
RESULT(31:0)	FFFFFFFF	00000001	00000002	00000003	00000101	80000001	00*

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
A(31:0)	00000002					00*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	00*
RESULT(31:0)	00000000	00000002	00000003	00000004	00000102	80000002	00*

Panel 1 (14000000 — 16000000)

SYS_CLK
SYS_RESET
GO
SEL
DONE

A(31:0)				00000003			55*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	00*
RESULT(31:0)	00000001	00000003	00000004	00000005	00000103	80000003	00*

Panel 2 (18000000 — 20000000 — 22)

SYS_CLK
SYS_RESET
GO
SEL
DONE

A(31:0)				55555555			AA*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	00*
RESULT(31:0)	00000002	55555555	55555556	55555557	55555655	D5555555	55*

Panel 3 (22000000 — 24000000 — 26000C)

SYS_CLK
SYS_RESET
GO
SEL
DONE

A(31:0)				AAAAAAAA			FF*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	00*
RESULT(31:0)	55555554	AAAAAAAA	AAAAAAAB	AAAAAAAC	AAAAABAA	2AAAAAAA	AA*

28000000   30000000

SYS_CLK
SYS_RESET
GO
SEL
DONE

Signal						
A(31:0)	FFFFFFFE					FF*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	000*
RESULT(31:0)	AAAAAAA9	FFFFFFFE	FFFFFFFF	00000000	000000FE	7FFFFFFE	FFF*

32000000   34000000

SYS_CLK
SYS_RESET
GO
SEL
DONE

Signal						
A(31:0)	FF7FFF7F					7FF*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	000*
RESULT(31:0)	FFFFFFFD	FF7FFF7F	FF7FFF80	FF7FFF81	FF80007F	7F7FFF7F	FF7*

36000000   38000000

SYS_CLK
SYS_RESET
GO
SEL
DONE

Signal						
A(31:0)	7FFFFFFF					FFF*
B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	000*
RESULT(31:0)	FF7FFF7E	7FFFFFFF	80000000	80000001	800000FF	FFFFFFFF	7FF*

40000000   **42000000**

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
► A(31:0)			FFFFFFFF			000*
► B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	000*
► RESULT(31:0)	7FFFFFFE	FFFFFFFF	00000000	00000001	000000FF	7FFFFFFF	FFF*

44000000   **46000000**   **48**

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
► A(31:0)			00000000			0000*
► B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	0000*
► RESULT(31:0)	FFFFFF*	00000000	FFFFFFFF	FFFFFFFE	FFFFFF00	80000000	0000*

48000000   **50000000**   **520000**

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
► A(31:0)			00000001			0000*
► B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	0000*
► RESULT(31:0)	00000001	00000000	FFFFFFFF	FFFFFF01	80000001	0000*

54000000 ... **56000000**

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
► A(31:0)	00000002					0000*
► B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	0000*
► RESULT(31:0)	00000002		00000001	00000000	FFFFFF02	80000002	00000*

58000000 ... **60000000**

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
► A(31:0)	00000003					5555*
► B(31:0)	00000000	00000001	00000002	00000100	80000000	FFFFFFFF	0000*
► RESULT(31:0)	00000003		00000002	00000001	FFFFFF03	80000003	00000*

62000000 ... **64000000**

SYS_CLK						
SYS_RESET						
GO						
SEL						
DONE						
► A(31:0)	55555555					AAAA*
► B(31:0)	000000*	00000001	00000002	00000100	80000000	FFFFFFFF	00000*
► RESULT(31:0)	000000*	55555555	55555554	55555553	55555455	D5555555	55555*

Panel 1 (timeline: 66000000, 68000000)

Signal							
SYS_CLK							
SYS_RESET							
GO							
SEL							
DONE							
A(31:0)	AAAAAAAA						FFFFF*
B(31:0)	000000*	00000001	00000002	00000100	80000000	FFFFFFFF	00000*
RESULT(31:0)	555555*	AAAAAAAA	AAAAAAA9	AAAAAAA8	AAAAA9AA	2AAAAAAA	AAAA*

Panel 2 (timeline: 70000000, 72000000, 74)

Signal							
SYS_CLK							
SYS_RESET							
GO							
SEL							
DONE							
A(31:0)	FFFFFFFE						FF7FF*
B(31:0)	000000*	00000001	00000002	00000100	80000000	FFFFFFFF	00000*
RESULT(31:0)	AAAAA*	FFFFFFFE	FFFFFFFD	FFFFFFFC	FFFFFEFE	7FFFFFFE	FFFFF*

Panel 3 (timeline: 74000000, 76000000, 780000)

Signal							
SYS_CLK							
SYS_RESET							
GO							
SEL							
DONE							
A(31:0)	FF7FFF7F						7FFFF*
B(31:0)	000000*	00000001	00000002	00000100	80000000	FFFFFFFF	000000*
RESULT(31:0)	FFFFF*	FF7FFF7F	FF7FFF7E	FF7FFF7D	FF7FFE7F	7F7FFF7F	FF7FF*

	80000000					82000000	
SYS_CLK							
SYS_RESET							
GO							
SEL							
DONE							
A(31:0)			7FFFFFFF				FFFFF*
B(31:0)	000000*	00000001	00000002	00000100	80000000	FFFFFFFF	000000*
RESULT(31:0)	FF7FF*	7FFFFFFF	7FFFFFFE	7FFFFFFD	7FFFFEFF	FFFFFFFF	800000*

	84000000					86000000	
SYS_CLK							
SYS_RESET							
GO							
SEL							
DONE							
A(31:0)			FFFFFFFF				
B(31:0)	00000*	00000001	00000002	00000100	80000000	FFFFFFFF	
RESULT(31:0)	80000*	FFFFFFFF	FFFFFFFE	FFFFFFFD	FFFFFEFF	7FFFFFFF	0000*

B.4 Register Unit

B.4.1 Register Model

```
— Project:                    Thesis
— Filename:                   reg_file_pkg.vhd
— Other files required:
— Date:                       sept 23 97
— Entity/Architecture Name:   na
— Developer:                  Steve Parmley
```

```vhdl
library IEEE;
  use IEEE.std_logic_1164.all;

package reg_file_pkg is

        subtype addr is integer range 31 downto 0;

end reg_file_pkg;
```

```
— Project:                    Thesis
— Filename:                   reg_file.vhd
— Other files required:       reg_file_pkg.vhd
— Date:                       sept 23 97
— Entity/Architecture Name:   reg_file_e/behavior
— Developer:                  Steve Parmley
```

```vhdl
library IEEE;
  use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;

entity reg_file_e is
  port (reg_file_reset            :    in     std_ulogic;
        reg_file_clk              :    in     std_ulogic;
        reg_file_C_bus            :    in     std_ulogic_vector(15 downto 0);
        reg_file_C_reg_latch      :    in     std_ulogic;
        reg_file_C_reg_addr       :    in     addr;
        reg_file_A_bus            :    out    std_ulogic_vector(15 downto 0);
        reg_file_A_reg_addr       :    in     addr;
        reg_file_B_bus            :    out    std_ulogic_vector(15 downto 0);
        reg_file_B_reg_addr       :    in     addr);
end reg_file_e;

architecture behavior of reg_file_e is
begin

  registers: process
subtype reg is std_ulogic_vector(15 downto 0);
type bank is array(31 downto 0) of reg;
variable regs : bank;
  begin


        if reg_file_reset = '1' then
                for index in 31 downto 2 loop
                        regs(index) := "0000000000000000";
                end loop;
```

```
                        -- force reg 0 and 1 to zero and one values
                        regs(0) := "0000000000000000";
                        regs(1) := "0000000100000000";
            end if;

            wait until (reg_file_clk'event and reg_file_clk='1');

            -- take care of write function first
            if reg_file_C_reg_latch = '1' then
                        if (reg_file_C_reg_addr = 0) or (reg_file_C_reg_addr = 1) then
                                    -- can not write to the zero and 1 registers
                        else
                                    regs(reg_file_C_reg_addr) := reg_file_C_bus;
                        end if;
            end if;

            -- now do A bus
                        reg_file_A_bus <= regs(reg_file_A_reg_addr);

            -- now do B bus
                        reg_file_B_bus <= regs(reg_file_B_reg_addr);

  end process registers;
end behavior;
```

B.4.2   Register Testbench

```
-- Project:                        Thesis
-- Filename:                       reg_file-bench.vhd
-- Other files required:           reg_file_pkg.vhd, reg_file.vhd
-- Date:                           sept 23 97
-- Entity/Architecture Name:       reg_file_tb/test
-- Developer:                      Steve Parmley


library IEEE;
  use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;

entity reg_file_tb is
end reg_file_tb;

architecture test of reg_file_tb is

component reg_file_e
  port (reg_file_reset              :       in      std_ulogic;
              reg_file_clk           :       in      std_ulogic;
              reg_file_C_bus         :       in      std_ulogic_vector(15 downto 0);
              reg_file_C_reg_latch   :       in      std_ulogic;
              reg_file_C_reg_addr    :       in      addr;
              reg_file_A_bus         :       out     std_ulogic_vector(15 downto 0);
              reg_file_A_reg_addr    :       in      addr;
              reg_file_B_bus         :       out     std_ulogic_vector(15 downto 0);
              reg_file_B_reg_addr    :       in      addr);
end component;

signal sys_reset, sys_clk : std_ulogic := '0';
signal bus_C, bus_A, bus_B : std_ulogic_vector(15 downto 0);
signal reg_addr_A, reg_addr_B, reg_addr_C : addr;
signal reg_latch_C : std_ulogic;
```

```
begin
U1 : reg_file_e
   PORT MAP   (sys_reset,
                    sys_clk,
                    bus_C,
                    reg_latch_C,
                    reg_addr_C,
                    bus_A,
                    reg_addr_A,
                    bus_B,
                    reg_addr_B);

clock : process
begin
         sys_clk <= not(sys_clk);
         wait for 10 ps;
end process clock;

rst : process
begin
         sys_reset <= '1';
         wait for 5 ps;
         sys_reset <= '0';
         wait for 15000 ps;
end process rst;

exercise : process
 begin

         reg_latch_C <= '0';
         bus_C <= "ZZZZZZZZZZZZZZZZ";
         reg_addr_A <= 15;
         reg_addr_B <= 15;
         reg_addr_C <= 0;
         wait until sys_clk'event and sys_clk ='0';

         -- verify that all regs are clear (except for zero regs 0 and 1)
         for i in 31 downto 0 loop
                  reg_addr_A <= i;
                  -- get B in reverse order to show dual bus works
                  reg_addr_B <= 31-i;
                  wait until sys_clk'event and sys_clk ='0';
         end loop;

         reg_addr_A <= 15;
         reg_addr_B <= 15;
         reg_addr_C <= 15;

         wait until sys_clk'event and sys_clk ='0';
         wait until sys_clk'event and sys_clk ='0';
         wait until sys_clk'event and sys_clk ='0';

         -- write some info to the regs
         reg_addr_C <= 0;
         bus_C <= "0100000000000001";
         wait until sys_clk'event and sys_clk ='0';
         reg_latch_C <= '1';
         wait until sys_clk'event and sys_clk ='0';
         reg_latch_C <= '0';

         reg_addr_C <= 1;
         bus_C <= "0100000000000010";
         wait until sys_clk'event and sys_clk ='0';
         reg_latch_C <= '1';
```

```
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 2;
bus_C <= "0100000000000011";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 3;
bus_C <= "0100000000000100";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 4;
bus_C <= "0100000000000101";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 5;
bus_C <= "0100000000000110";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 6;
bus_C <= "0100000000000111";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 7;
bus_C <= "0100000000001000";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 8;
bus_C <= "0100000000001001";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 9;
bus_C <= "0100000000001010";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 10;
bus_C <= "0100000000001011";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';
```

```
reg_addr_C <= 11;
bus_C <= "0100000000001100";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 12;
bus_C <= "0100000000001101";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 13;
bus_C <= "0100000000001110";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 14;
bus_C <= "0100000000001111";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 15;
bus_C <= "0100000000010000";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 16;
bus_C <= "1000000000000001";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 17;
bus_C <= "1000000000000010";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 18;
bus_C <= "1000000000000011";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 19;
bus_C <= "1000000000000100";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 20;
```
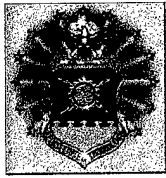
```
bus_C <= "1000000000000101";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 21;
bus_C <= "1000000000000110";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 22;
bus_C <= "1000000000000111";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 23;
bus_C <= "1000000000001000";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 24;
bus_C <= "1000000000001001";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 25;
bus_C <= "1000000000001010";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 26;
bus_C <= "1000000000001011";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 27;
bus_C <= "1000000000001100";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 28;
bus_C <= "1000000000001101";
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '1';
wait until sys_clk'event and sys_clk ='0';
reg_latch_C <= '0';

reg_addr_C <= 29;
bus_C <= "1000000000001110";
wait until sys_clk'event and sys_clk ='0';
```

```
                    reg_latch_C <= '1';
                    wait until sys_clk'event and sys_clk ='0';
                    reg_latch_C <= '0';

                    reg_addr_C <= 30;
                    bus_C <= "1000000000001111";
                    wait until sys_clk'event and sys_clk ='0';
                    reg_latch_C <= '1';
                    wait until sys_clk'event and sys_clk ='0';
                    reg_latch_C <= '0';

                    reg_addr_C <= 31;
                    bus_C <= "1000000000010000";
                    wait until sys_clk'event and sys_clk ='0';
                    reg_latch_C <= '1';
                    wait until sys_clk'event and sys_clk ='0';
                    reg_latch_C <= '0';


                    bus_C <= "ZZZZZZZZZZZZZZZZ";
                    reg_addr_C <= 15;
                    wait until sys_clk'event and sys_clk ='0';
                    wait until sys_clk'event and sys_clk ='0';
                    wait until sys_clk'event and sys_clk ='0';

                    -- verify that all regs are correct (except for zero regs 5 and 6)
                    for i in 31 downto 0 loop
                            reg_addr_A <= i;
                            reg_addr_B <= 31-i;
                            wait until sys_clk'event and sys_clk ='0';
                    end loop;

                    wait until sys_clk'event and sys_clk ='1';

                    ASSERT false
                            REPORT "DONE"
                            SEVERITY failure;

end process exercise;
end test;

CONFIGURATION reg_file_c OF reg_file_tb IS
  FOR test
    FOR ALL: reg_file_e
      USE ENTITY WORK.reg_file_e(behavior);
    END FOR;
  END FOR;
END reg_file_c;
```

B.4.3 Register Results

	0	100000	200000	300000
SYS_RESET				
SYS_CLK				
BUS_C(15:0)			ZZZZ	
BUS_A(15:0)	UU*		0000	
BUS_B(15:0)	UU* 0000 0100		0000	
REG_ADDR_A	15 31 30 29	28 27 26 25	24 23 22 21 20 19	18 17 16 15
REG_ADDR_B	15 0 1 2	3 4 5 6	7 8 9 10 11 12	13 14 15 16
REG_ADDR_C			0	
REG_LATCH_C				

	400000	500000	600000
SYS_RESET			
SYS_CLK			
BUS_C(15:0)		ZZZZ	
BUS_A(15:0)	0000		0100
BUS_B(15:0)		0000	
REG_ADDR_A	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
REG_ADDR_B	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	
REG_ADDR_C		0	15
REG_LATCH_C			

	700000	800000	900000	1000000
SYS_RESET				
SYS_CLK				
BUS_C(15:0)	ZZZZ 4001 4002	4003 4004	4005 4006	4007 4008
BUS_A(15:0)			0000	
BUS_B(15:0)			0000	
REG_ADDR_A			15	
REG_ADDR_B			15	
REG_ADDR_C	15 0 1	2 3	4 5	6 7
REG_LATCH_C				

2100000   2200000   2300000

	2100000															
SYS_RESET																
SYS_CLK																
BUS_C(15:0)										ZZZZ						
BUS_A(15:0)	4010	8010	800F	800E	800D	800C	800B	800A	8009	8008	8007	8006	8005	8004	8003	8002 80
BUS_B(15:0)	4010	0000	0100	4003	4004	4005	4006	4007	4008	4009	400A	400B	400C	400D	400E	400F 40
REG_ADDR_A	15	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17 16
REG_ADDR_B	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14 15
REG_ADDR_C										15						
REG_LATCH_C																

2400000   2500000   2600000   27000

	2400000															
SYS_RESET																
SYS_CLK																
BUS_C(15:0)								ZZZZ								
BUS_A(15:0)	8001	4010	400F	400E	400D	400C	400B	400A	4009	4008	4007	4006	4005	4004	4003	0100	0000
BUS_B(15:0)	4010	8001	8002	8003	8004	8005	8006	8007	8008	8009	800A	800B	800C	800D	800E	800F	8010
REG_ADDR_A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_ADDR_B	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
REG_ADDR_C								15								
REG_LATCH_C																

B.5   Latch

B.5.1   Latch Model

```
-- Project:                          Thesis
-- Filename:                         latch.vhd
-- Other files required:
-- Date:                             Oct 17 97
-- Entity/A_busrchitecture Name:     latch_e/behavior
-- Developer:                        Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
   use IEEE.std_logic_1164.all;

entity latch_e is
  port (latch_en            :        in      std_ulogic;
            latch_A_bus      :        in      std_ulogic_vector(15 downto 0);
            latch_O_bus      :        out     std_ulogic_vector(15 downto 0));
end latch_e;

architecture behavior of latch_e is
begin

latch : process (latch_en, latch_A_bus)
 begin
  if latch_en = '1' then
            latch_O_bus <= latch_A_bus;
   end if;
 end process latch;
end behavior;
```

B.5.2   Latch Testbench

```
-- Project:                    Thesis
-- Filename:                   mux4_1-bench.vhd
-- Other files required:
-- Date:                       Oct 17 97
-- Entity/Architecture Name:   mux4_1_tb/test
-- Developer:                  Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
   use IEEE.std_logic_1164.all;



entity latch_tb is
end latch_tb;
```

```vhdl
architecture test of latch_tb is

constant  Atest0     : std_ulogic_vector(15 downto 0) := "0000000000000000";
constant  Atest1  : std_ulogic_vector(15 downto 0) := "0101010101010101";
constant  Atest2     : std_ulogic_vector(15 downto 0) := "1111111111111111";
constant  Atest3  : std_ulogic_vector(15 downto 0) := "1010101010101010";



component latch_e
  port (latch_en              :        in        std_ulogic;
           latch_A_bus        :        in        std_ulogic_vector(15 downto 0);
           latch_O_bus        :        out       std_ulogic_vector(15 downto 0));
end component;

signal en            : std_ulogic := '0';
signal A,O                         : std_ulogic_vector(15 downto 0);

begin
U1 : latch_e
  PORT MAP    (en,
                 A,
                 O);


exercise : process
  begin
           wait for 5 ps;

           For j in 0 to 3 loop

                   CASE j is
                           WHEN 0 => A <= Atest0;
                           WHEN 1 => A <= Atest1;
                           WHEN 2 => A <= Atest2;
                           WHEN 3 => A <= Atest3;
                   end CASE;

                   wait for 5 ps;

                   en <= '1';

                   wait for 5 ps;

                   en <= '0';

                   wait for 20 ps;

           end loop;


      ASSERT false
                   REPORT "DONE"
                   SEVERITY failure;
  end process exercise;
end test;

CONFIGURATION latch_c OF latch_tb IS
  FOR test
    FOR ALL: latch_e
      USE ENTITY WORK.latch_e(behavior);
    END FOR;
  END FOR;
END latch_c;
```
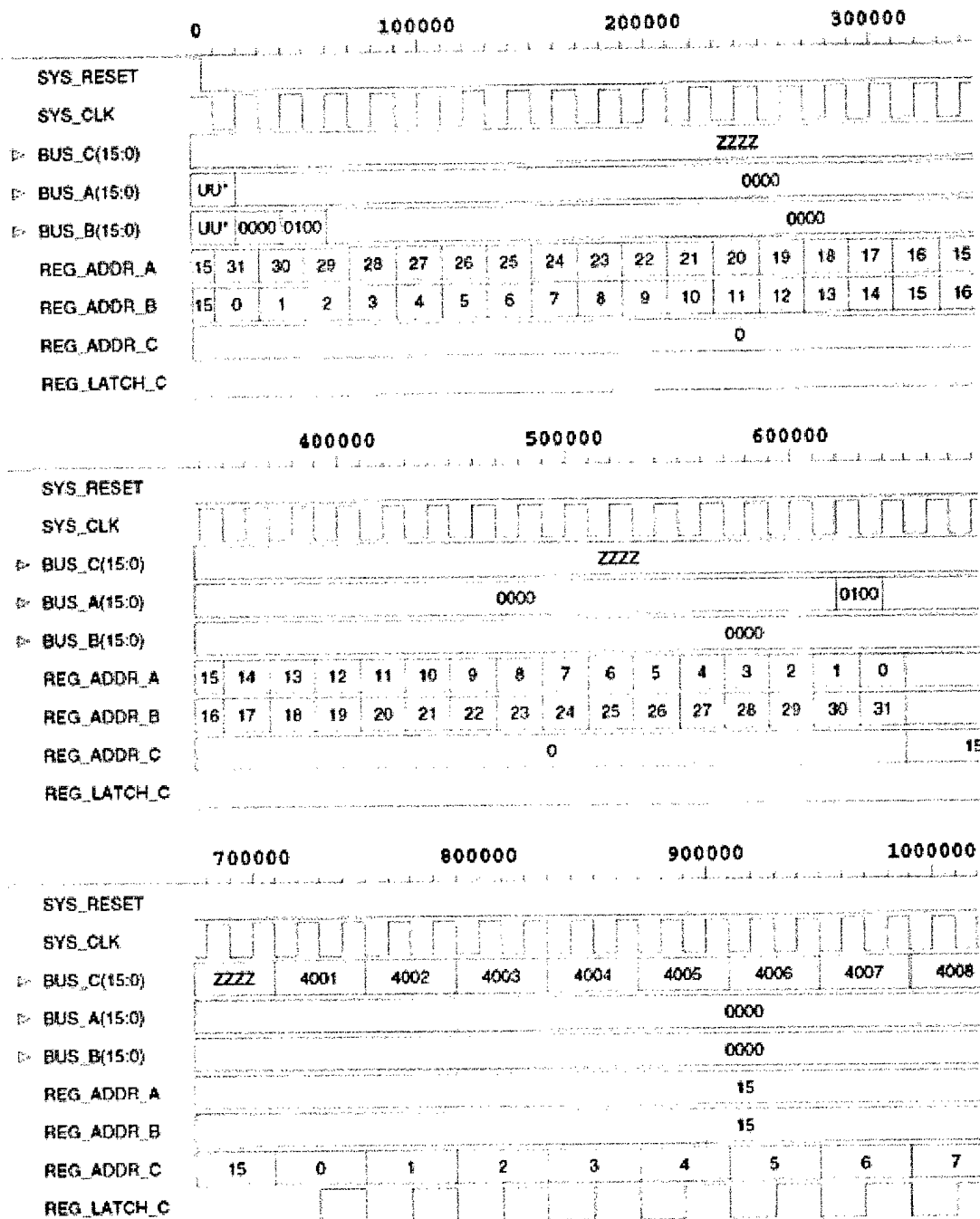
B.5.3 Latch Results

	0		50000		100000	
EN						
▷ A(15:0)	UU*	0000	5555	FFFF		AAAA
▷ O(15:0)	UUUU	0000	5555	FFFF		AAAA

B.6 Multiplexor

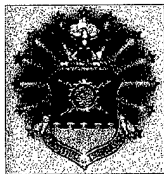## B.6.1   Multiplexor Model

```
-- Project:                          Thesis
-- Filename:                         mux4_1.vhd
-- Other files required:
-- Date:                             Oct 17 97
-- Entity/A_busrchitecture Name:     mux4_1_e/behavior
-- Developer:                        Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
  use IEEE.std_logic_1164.all;

entity mux4_1_e is
  port (mux_clk              :     in      std_ulogic;
           mux_sel           :     in      std_ulogic_vector(1 downto 0);
           mux_A_bus         :     in      std_ulogic_vector(15 downto 0);
           mux_B_bus         :     in      std_ulogic_vector(15 downto 0);
           mux_C_bus         :     in      std_ulogic_vector(15 downto 0);
           mux_D_bus         :     in      std_ulogic_vector(15 downto 0);
           mux_O_bus         :     out     std_ulogic_vector(15 downto 0));
end mux4_1_e;

architecture behavior of mux4_1_e is
begin

mux : process
 begin
   wait until mux_clk'event and mux_clk='1';
   case mux_sel is
           when "00" => mux_O_bus <= mux_A_bus;
           when "01" => mux_O_bus <= mux_B_bus;
           when "10" => mux_O_bus <= mux_C_bus;
           when "11" => mux_O_bus <= mux_D_bus;
           when others => mux_O_bus <= mux_A_bus;
   end case;
 end process mux;
end behavior;
```

B.6.2   Multiplexor Testbench

```
-- Project:                    Thesis
-- Filename:                   mux4_1-bench.vhd
-- Other files required:
-- Date:                       Oct 17 97
-- Entity/Architecture Name:   mux4_1_tb/test
-- Developer:                  Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:
```

```vhdl
library IEEE;
  use IEEE.std_logic_1164.all;



entity mux4_1_tb is
end mux4_1_tb;

architecture test of mux4_1_tb is

constant  Atest0    : std_ulogic_vector(15 downto 0) := "0000000000000000";
constant  Btest0  : std_ulogic_vector(15 downto 0) := "0101010101010101";
constant  Ctest0    : std_ulogic_vector(15 downto 0) := "1111111111111111";
constant  Dtest0  : std_ulogic_vector(15 downto 0) := "1010101010101010";
constant  Atest1    : std_ulogic_vector(15 downto 0) := "0000111100001111";
constant  Btest1  : std_ulogic_vector(15 downto 0) := "1111000011110000";
constant  Ctest1    : std_ulogic_vector(15 downto 0) := "1100110011001100";
constant  Dtest1  : std_ulogic_vector(15 downto 0) := "0011001100110011";


constant  A_sel : std_ulogic_vector := "00";
constant  B_sel : std_ulogic_vector := "01";
constant  C_sel : std_ulogic_vector := "10";
constant  D_sel : std_ulogic_vector := "11";

component mux4_1_e
  port (mux_clk              :         in        std_ulogic;
          mux_sel            :         in        std_ulogic_vector(1 downto 0);
          mux_A_bus          :         in        std_ulogic_vector(15 downto 0);
          mux_B_bus          :         in        std_ulogic_vector(15 downto 0);
          mux_C_bus          :         in        std_ulogic_vector(15 downto 0);
          mux_D_bus          :         in        std_ulogic_vector(15 downto 0);
          mux_O_bus          :         out       std_ulogic_vector(15 downto 0));
end component;

signal sel          : std_ulogic_vector(1 downto 0) := "11";
signal A,B,C,D,O    : std_ulogic_vector(15 downto 0);
signal sys_clk                  : std_ulogic := '0';

begin
U1 : mux4_1_e
  PORT MAP    (sys_clk,
                  sel,
                  A,
                  B,
                  C,
                  D,
                  O);

clock : process
begin
        sys_clk <= not(sys_clk);
        wait for 10 ps;
end process clock;


exercise : process
  begin
        wait for 20 ps;

        For j in 0 to 1 loop

                CASE j is
                        WHEN 0 => A <= Atest0;
                                  B <= Btest0;
                                  C <= Ctest0;
```
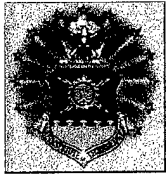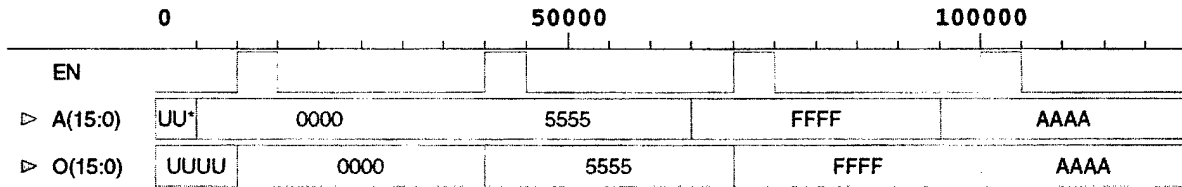
```
                              D <= Dtest0;
               WHEN 1 => A <= Atest1;
                          B <= Btest1;
                          C <= Ctest1;
                          D <= Dtest1;
          end CASE;


          For i in 0 to 3 loop
                 CASE i IS
                            WHEN 0 => sel <= A_sel;
                            WHEN 1 => sel <= B_sel;
                            WHEN 2 => sel <= C_sel;
                            WHEN 3 => sel <= D_sel;
                 END CASE;

                 wait until sys_clk'event and sys_clk = '1';
          end loop;
       end loop;


     ASSERT false
                 REPORT "DONE"
                 SEVERITY failure;
 end process exercise;
end test;

CONFIGURATION mux4_1_c OF mux4_1_tb IS
  FOR test
    FOR ALL: mux4_1_e
      USE ENTITY WORK.mux4_1_e(behavior);
    END FOR;
  END FOR;
END mux4_1_c;
```

B.6.3   Multiplexor Results

	0				50000				100000		
▷ SEL(1:0)	3	0	1	2	3	0	1	2	3		
▷ A(15:0)	UU*			0000					0F0F		
▷ B(15:0)	UU*			5555					F0F0		
▷ C(15:0)	UU*			FFFF					CCCC		
▷ D(15:0)	UU*			AAAA					3333		
▷ O(15:0)	UU*	0000	5555	FFFF	AAAA	0F0F	F0F0	CCCC	3333		

B.7 FKP Core

B.7.1 FKP Core Model

```
-- Project:                          Thesis
-- Filename:                         fkp_core_core.vhd
-- Other files required:             all FKP files
-- Date:                             Oct 17 97
-- Entity/Architecture Name:         fkp_core_e/behavior
-- Developer:                        Steve Parmley
-- Function:
-- Limitations:
-- History:
-- Last Analyzed On:


library IEEE;
    use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;


entity fkp_core_e is
    port (fkp_core_clk                 :        in        std_ulogic;
            fkp_core_reset              :        in        std_ulogic;
            fkp_core_data_in            :        in        std_ulogic_vector(15 downto 0);
            fkp_core_data_out           :        out       std_ulogic_vector(15 downto 0);
            fkp_core_data_in_latch      :        in        std_ulogic;
            fkp_core_data_out_latch     :        in        std_ulogic;
            fkp_core_c_reg_latch        :        in        std_ulogic;
            fkp_core_c_reg_addr         :        in        addr;
            fkp_core_a_reg_addr         :        in        addr;
            fkp_core_b_reg_addr         :        in        addr;
            fkp_core_cos_sin_ready      :        out       std_ulogic;
            fkp_core_cos_sin_go         :        in        std_ulogic;
            fkp_core_cos_sin_sel        :        in        std_ulogic;
            fkp_core_cos_sin_wait       :        in        std_ulogic_vector(2 downto 0);
            fkp_core_rom_addr           :        out       std_ulogic_vector(12 downto 0);
            fkp_core_rom_data           :        in        std_ulogic_vector(15 downto 0);
            fkp_core_adder_go           :        in        std_ulogic;
            fkp_core_adder_sel          :        in        std_ulogic;
            fkp_core_adder_done         :        out       std_ulogic;
            fkp_core_mult_go            :        in        std_ulogic;
            fkp_core_mult_done          :        out       std_ulogic;
            fkp_core_mux_sel            :        in        std_ulogic_vector(1 downto 0));
end fkp_core_e;

architecture structural of fkp_core_e is

-- SIGNALS
signal    cos_sin_to_mux, adder_to_mux, mult_to_mux, data_in_to_mux : std_ulogic_vector(15 downto 0);
signal  mux_to_regs, A_bus, B_bus        : std_ulogic_vector(15 downto 0);


-- COMPONENTS
component adder_e
    port (adder_reset          :        in        std_ulogic;
            adder_clk           :        in        std_ulogic;
            adder_A_bus         :        in        std_ulogic_vector(15 downto 0);
            adder_B_bus         :        in        std_ulogic_vector(15 downto 0);
            adder_go            :        in        std_ulogic;
            adder_sel           :        in        std_ulogic;
```

```
                adder_done        :        out      std_ulogic;
                adder_C_bus       :        out      std_ulogic_vector(15 downto 0));
end component;

component mult_e
  port (mult_reset    :        in       std_ulogic;
        mult_clk   :            in       std_ulogic;
        mult_A_bus      :        in       std_ulogic_vector(15 downto 0);
        mult_B_bus      :        in       std_ulogic_vector(15 downto 0);
        mult_go         :        in       std_ulogic;
        mult_done       :        out      std_ulogic;
        mult_C_bus      :        out      std_ulogic_vector(15 downto 0));
end component;

component cos_sin_e
  port (cos_sin_reset :        in       std_ulogic;
        cos_sin_clk     :        in       std_ulogic;
        cos_sin_A_bus   :        in       std_ulogic_vector(15 downto 0);
        cos_sin_go      :        in       std_ulogic;
        cos_sin_sel     :        in       std_ulogic;
        cos_sin_wait    :        in       std_ulogic_vector(2 downto 0);
        cos_sin_ready   :        out      std_ulogic;
        cos_sin_C_bus   :        out      std_ulogic_vector(15 downto 0);
        -- the following describes the connection to the rom
        cos_sin_rom_addr: out         std_ulogic_vector(12 downto 0);
        cos_sin_rom_data: in          std_ulogic_vector(15 downto 0));
end component;

component reg_file_e
  port (reg_file_reset              :        in       std_ulogic;
        reg_file_clk                :        in       std_ulogic;
        reg_file_C_bus              :        in       std_ulogic_vector(15 downto 0);
        reg_file_C_reg_latch        :        in       std_ulogic;
        reg_file_C_reg_addr         :        in       addr;
        reg_file_A_bus              :        out      std_ulogic_vector(15 downto 0);
        reg_file_A_reg_addr         :        in       addr;
        reg_file_B_bus              :        out      std_ulogic_vector(15 downto 0);
        reg_file_B_reg_addr         :        in       addr);
end component;

component latch_e
  port (latch_en     :        in       std_ulogic;
        latch_A_bus     :        in       std_ulogic_vector(15 downto 0);
        latch_O_bus     :        out      std_ulogic_vector(15 downto 0));
end component;

component mux4_1_e
  port (mux_clk          :        in       std_ulogic;
        mux_sel          :        in       std_ulogic_vector(1 downto 0);
        mux_A_bus        :        in       std_ulogic_vector(15 downto 0);
        mux_B_bus        :        in       std_ulogic_vector(15 downto 0);
        mux_C_bus        :        in       std_ulogic_vector(15 downto 0);
        mux_D_bus        :        in       std_ulogic_vector(15 downto 0);
        mux_O_bus        :        out      std_ulogic_vector(15 downto 0));
end component;


begin
U_adder_1 : adder_e
   PORT MAP  (fkp_core_reset,
                fkp_core_clk,
                A_bus,
                B_bus,
                fkp_core_adder_go,
                fkp_core_adder_sel,
```
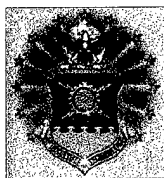
```
                        fkp_core_adder_done,
                        adder_to_mux);

U_mult_1 : mult_e
  PORT MAP   (fkp_core_reset,
                        fkp_core_clk,
                        A_bus,
                        B_bus,
                        fkp_core_mult_go,
                        fkp_core_mult_done,
                        mult_to_mux);

U_cos_sin_1 : cos_sin_e
  PORT MAP   (fkp_core_reset,
                        fkp_core_clk,
                        A_bus,
                        fkp_core_cos_sin_go,
                        fkp_core_cos_sin_sel,
                        fkp_core_cos_sin_wait,
                        fkp_core_cos_sin_ready,
                        cos_sin_to_mux,
                        fkp_core_rom_addr,
                        fkp_core_rom_data);


U_reg_file_1 : reg_file_e
  PORT MAP   (fkp_core_reset,
                        fkp_core_clk,
                        mux_to_regs,
                        fkp_core_c_reg_latch,
                        fkp_core_c_reg_addr,
                        A_bus,
                        fkp_core_a_reg_addr,
                        B_bus,
                        fkp_core_b_reg_addr);

U_mux4_1_1 : mux4_1_e
  PORT MAP   (fkp_core_clk,
                        fkp_core_mux_sel,
                        cos_sin_to_mux,
                        adder_to_mux,
                        mult_to_mux,
                        data_in_to_mux,
                        mux_to_regs);

U_latch_in : latch_e
  PORT MAP   (fkp_core_data_in_latch,
                        fkp_core_data_in,
                        data_in_to_mux);

U_latch_out : latch_e
  PORT MAP   (fkp_core_data_out_latch,
                        B_bus,
                        fkp_core_data_out);

end structural;
```

B.7.2   FKP Core Testbench

```
-- Project:                    Thesis
-- Filename:                   fkp_core-bench.vhd
-- Other files required:       fkp_core.vhd
-- Date:                       Oct 20 97
-- Entity/Architecture Name:   fkp_core_tb/test
-- Developer:                  Steve Parmley


library IEEE;
  use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;


entity fkp_core_tb is
end fkp_core_tb;

architecture test of fkp_core_tb is

component fkp_core_e
  port (fkp_core_clk               :       in        std_ulogic;
        fkp_core_reset             :       in        std_ulogic;
        fkp_core_data_in           :       in        std_ulogic_vector(15 downto 0);
        fkp_core_data_out          :       out       std_ulogic_vector(15 downto 0);
        fkp_core_data_in_latch     :       in        std_ulogic;
        fkp_core_data_out_latch    :       in        std_ulogic;
        fkp_core_c_reg_latch       :       in        std_ulogic;
        fkp_core_c_reg_addr        :       in        addr;
        fkp_core_a_reg_addr        :       in        addr;
        fkp_core_b_reg_addr        :       in        addr;
        fkp_core_cos_sin_ready     :       out       std_ulogic;
        fkp_core_cos_sin_go        :       in        std_ulogic;
        fkp_core_cos_sin_sel       :       in        std_ulogic;
        fkp_core_cos_sin_wait      :       in        std_ulogic_vector(2 downto 0);
        fkp_core_rom_addr          :       out       std_ulogic_vector(12 downto 0);
        fkp_core_rom_data          :       in        std_ulogic_vector(15 downto 0);
        fkp_core_adder_go          :       in        std_ulogic;
        fkp_core_adder_sel         :       in        std_ulogic;
        fkp_core_adder_done        :       out       std_ulogic;
        fkp_core_mult_go           :       in        std_ulogic;
        fkp_core_mult_done         :       out       std_ulogic;
        fkp_core_mux_sel           :       in        std_ulogic_vector(1 downto 0));
end component;

signal sys_reset, sys_clk : std_ulogic := '0';
signal a_reg_addr, b_reg_addr, c_reg_addr : addr;
signal data_in, data_out : std_ulogic_vector(15 downto 0);
signal data_in_latch, data_out_latch, c_reg_latch, cos_sin_ready : std_ulogic;
signal cos_sin_go, cos_sin_sel, adder_go, adder_sel, adder_done : std_ulogic;
signal mult_go, mult_done                                           : std_ulogic;
signal cos_sin_wait : std_ulogic_vector(2 downto 0);
signal rom_addr : std_ulogic_vector(12 downto 0);
signal rom_data : std_ulogic_vector(15 downto 0);
signal mux_sel  : std_ulogic_vector(1 downto 0);

type opcode is (illegal, movein, moveout, move, cosine, sine, addition, subtraction, multiplication);
signal instruction : opcode;

begin
U1 : fkp_core_e
  PORT MAP   (sys_clk,
```
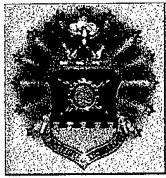
```vhdl
                        sys_reset,
                        data_in,
                        data_out,
                        data_in_latch,
                        data_out_latch,
                        c_reg_latch,
                        c_reg_addr,
                        a_reg_addr,
                        b_reg_addr,
                        cos_sin_ready,
                        cos_sin_go,
                        cos_sin_sel,
                        cos_sin_wait,
                        rom_addr,
                        rom_data,
                        adder_go,
                        adder_sel,
                        adder_done,
                        mult_go,
                        mult_done,
                        mux_sel);

clock : process
begin
           sys_clk <= not(sys_clk);
           wait for 10 ps;
end process clock;

rst : process
begin
           sys_reset <= '1';
           wait for 40 ps;

           sys_reset <= '0';
           wait for 50000 ps;
end process rst;

exercise : process
 begin

-- quick test
           instruction <= illegal;
           data_in_latch <= '0';
           data_out_latch <= '0';
           c_reg_latch <= '0';
           cos_sin_go <= '0';
           cos_sin_wait <= "111";
           adder_go <= '0';
           mult_go <= '0';
           a_reg_addr <= 15;
           b_reg_addr <= 15;
           c_reg_addr <= 15;
           mux_sel <= "00";
           wait for 60 ps;
           wait until sys_clk'event and sys_clk='1';

   -- MOVE IN
           instruction <= movein;
           data_in <= "0000000000000101";
           wait until sys_clk'event and sys_clk='1';

           mux_sel <= "11";
           c_reg_addr <= 2;
           data_in_latch <= '1';
           wait until sys_clk'event and sys_clk='1';
```
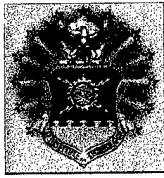
```
        data_in_latch <= '0';
        c_reg_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        c_reg_latch <= '0';
-- END MOVE IN

-- MOVE OUT
        instruction <= moveout;
        b_reg_addr <= 2;
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '0';
-- END MOVE OUT

-- MOVE IN
        instruction <= movein;
        data_in <= "0000000001001011";
        wait until sys_clk'event and sys_clk='1';

        mux_sel <= "11";
        c_reg_addr <= 3;
        data_in_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        data_in_latch <= '0';
        c_reg_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        c_reg_latch <= '0';
-- END MOVE IN

-- MOVE OUT
        instruction <= moveout;
        b_reg_addr <= 3;
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '0';
-- END MOVE OUT


-- ADD
        instruction <= addition;
        a_reg_addr <= 2;
        b_reg_addr <= 3;
        c_reg_addr <= 10;
        adder_sel <= '0';
        mux_sel    <= "01";
        wait until sys_clk'event and sys_clk='1';

        adder_go <= '1';
        wait until adder_done = '1';

        adder_go <= '0';
        c_reg_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        c_reg_latch <= '0';
```
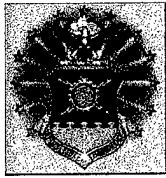
```
-- END ADD


-- MOVE OUT
        instruction <= moveout;
        b_reg_addr <= 10;
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '0';
-- END MOVE OUT

-- MOVE
        instruction <= move;
        a_reg_addr <= 0;
        b_reg_addr <= 10;
        c_reg_addr <= 11;
        adder_sel <= '0';
        mux_sel    <= "01";
        wait until sys_clk'event and sys_clk='1';

        adder_go <= '1';
        wait until adder_done = '1';

        adder_go <= '0';
        c_reg_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        c_reg_latch <= '0';
-- END MOVE


for i in 0 to 3 loop

-- SUB
        instruction <= subtraction;
        a_reg_addr <= 11;
        b_reg_addr <= 1;
        c_reg_addr <= 11;
        adder_sel <= '1';
        mux_sel    <= "01";
        wait until sys_clk'event and sys_clk='1';

        adder_go <= '1';
        wait until adder_done = '1';

        adder_go <= '0';
        c_reg_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        c_reg_latch <= '0';
-- END ADD


-- MOVE OUT
        instruction <= moveout;
        b_reg_addr <= 11;
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '0';
```

```
-- END MOVE OUT


end loop;

  -- Multiply
          instruction <= multiplication;
          a_reg_addr <= 2;
          b_reg_addr <= 3;
          c_reg_addr <= 31;
          mux_sel      <= "10";
          wait until sys_clk'event and sys_clk='1';


          mult_go <= '1';
          wait until mult_done = '1';


          mult_go <= '0';
          c_reg_latch <= '1';
          wait until sys_clk'event and sys_clk='1';


          c_reg_latch <= '0';
  -- END ADD


for i in 0 to 31 loop
  -- MOVE OUT
          instruction <= moveout;
          b_reg_addr <= i;
          wait until sys_clk'event and sys_clk='1';


          data_out_latch <= '1';
          wait until sys_clk'event and sys_clk='1';


          data_out_latch <= '0';
  -- END MOVE OUT
end loop;


  -- COSINE
          instruction <= cosine;
          cos_sin_sel <= '0';
          a_reg_addr <= 2;
          mux_sel <= "00";
          c_reg_addr <= 15;
          wait until sys_clk'event and sys_clk='1';


          cos_sin_go <= '1';
          wait until cos_sin_ready='1';


          cos_sin_go <= '0';
          c_reg_latch <= '1';
          wait until sys_clk'event and sys_clk='1';


          c_reg_latch <= '0';

       --

  -- MOVE OUT
          instruction <= moveout;
          b_reg_addr <= 15;
          wait until sys_clk'event and sys_clk='1';


          data_out_latch <= '1';
          wait until sys_clk'event and sys_clk='1';


          data_out_latch <= '0';
```
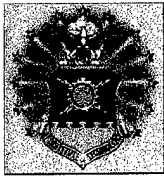
```
-- END MOVE OUT

-- SINE
        instruction <= sine;
        cos_sin_sel <= '1';
        a_reg_addr <= 3;
        mux_sel <= "00";
        c_reg_addr <= 16;
        wait until sys_clk'event and sys_clk='1';

        cos_sin_go <= '1';
        wait until cos_sin_ready='1';

        cos_sin_go <= '0';
        c_reg_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        c_reg_latch <= '0';
--

-- MOVE OUT
        instruction <= moveout;
        b_reg_addr <= 16;
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '1';
        wait until sys_clk'event and sys_clk='1';

        data_out_latch <= '0';
-- END MOVE OUT

        wait until sys_clk'event and sys_clk='1';
        wait until sys_clk'event and sys_clk='1';
        wait until sys_clk'event and sys_clk='1';

        ASSERT false
                REPORT "DONE"
                SEVERITY failure;

end process exercise;

rom : process
  begin
        wait until rom_addr'event;

        -- make up some rom data (inverse of the address for now)
        rom_data(12 downto 0) <= not(rom_addr(12 downto 0));

        -- fill in the rest
        rom_data(15 downto 13) <= "000";
  end process rom;

end test;

CONFIGURATION fkp_core_c OF fkp_core_tb IS
  FOR test
    FOR ALL: fkp_core_e
      USE ENTITY WORK.fkp_core_e(structural);
    END FOR;
  END FOR;
END fkp_core_c;
```
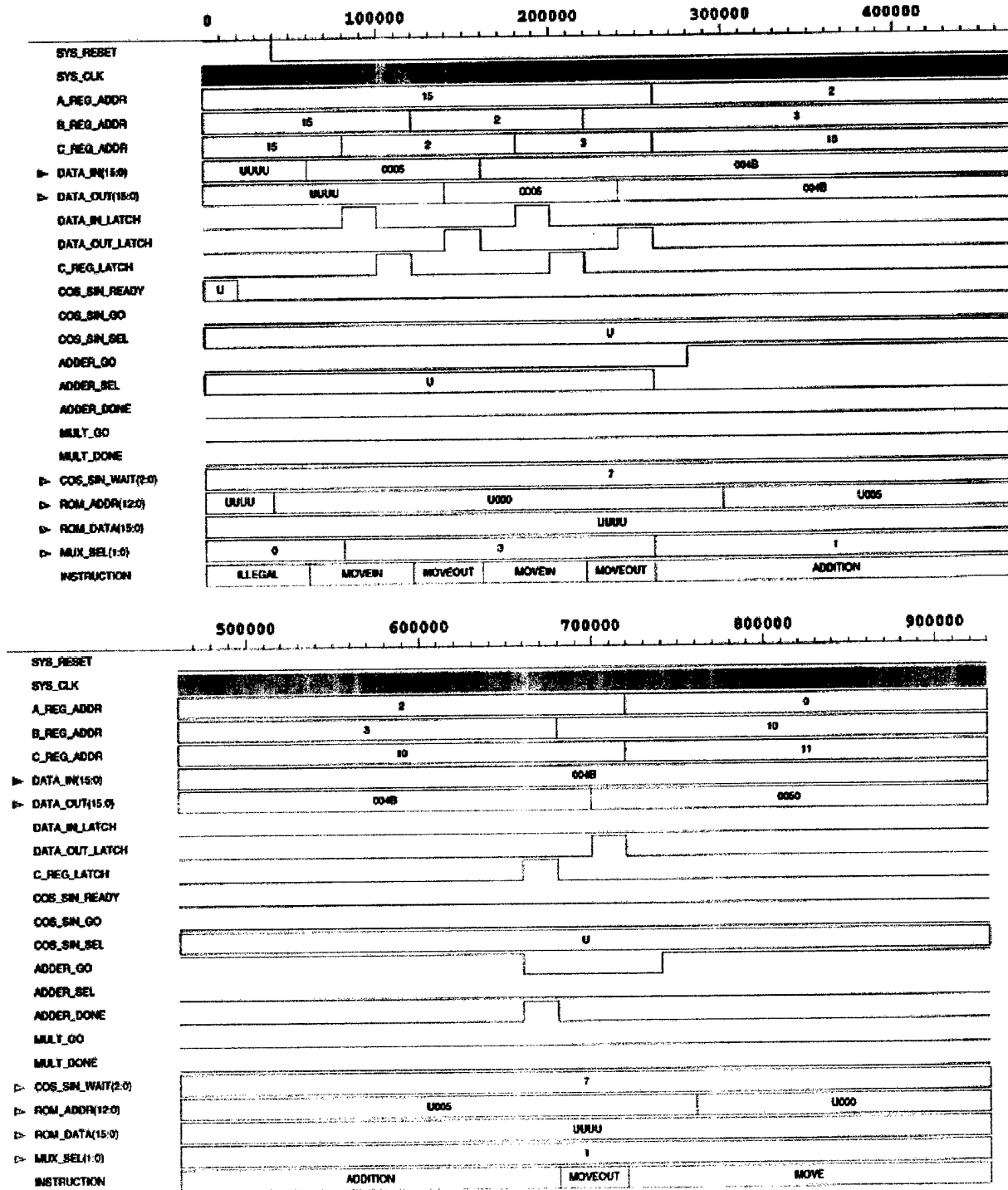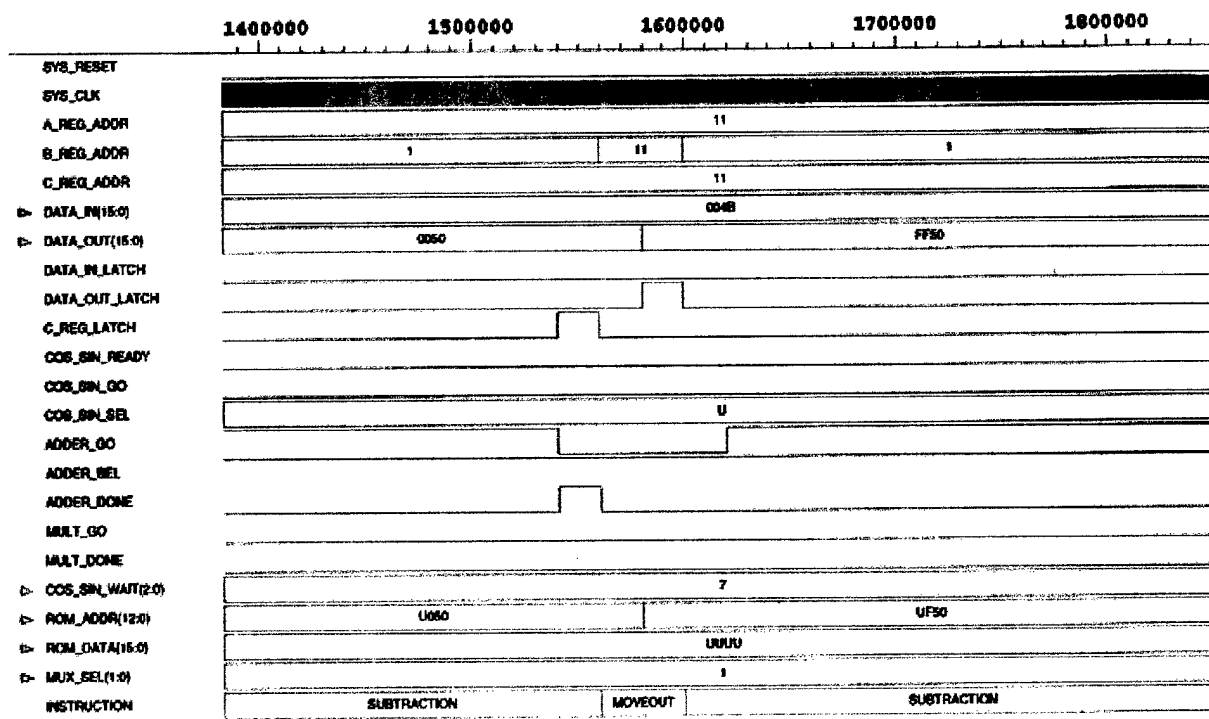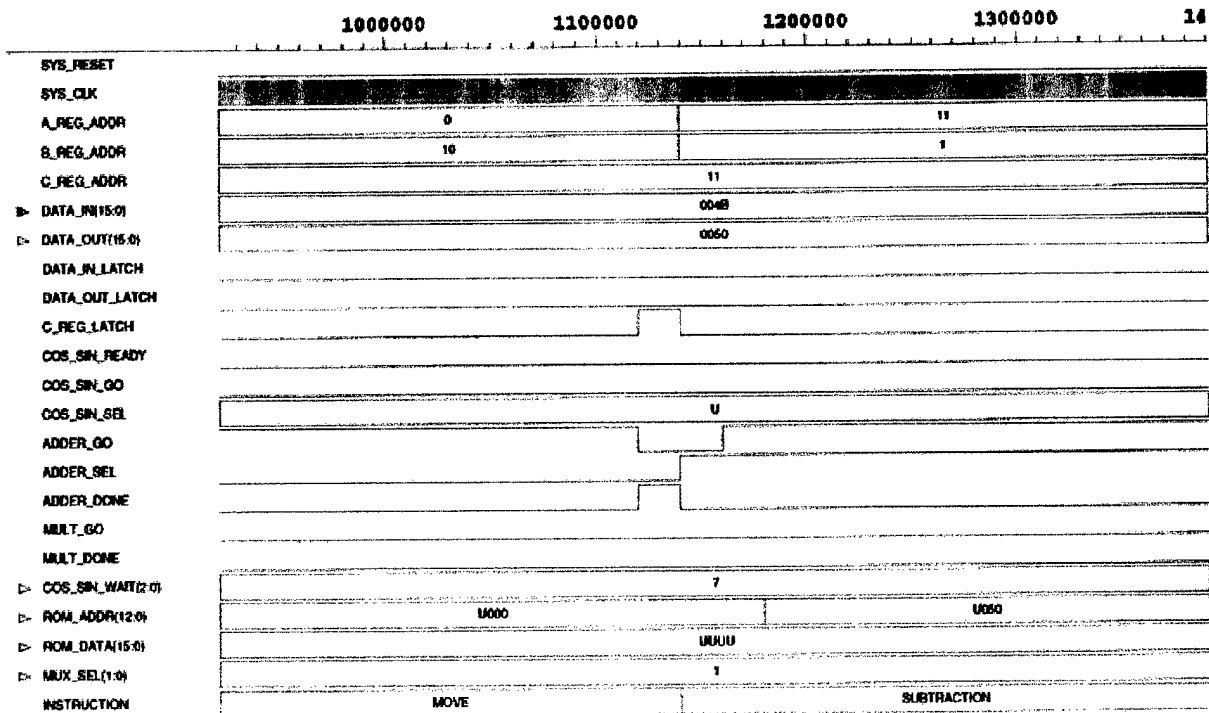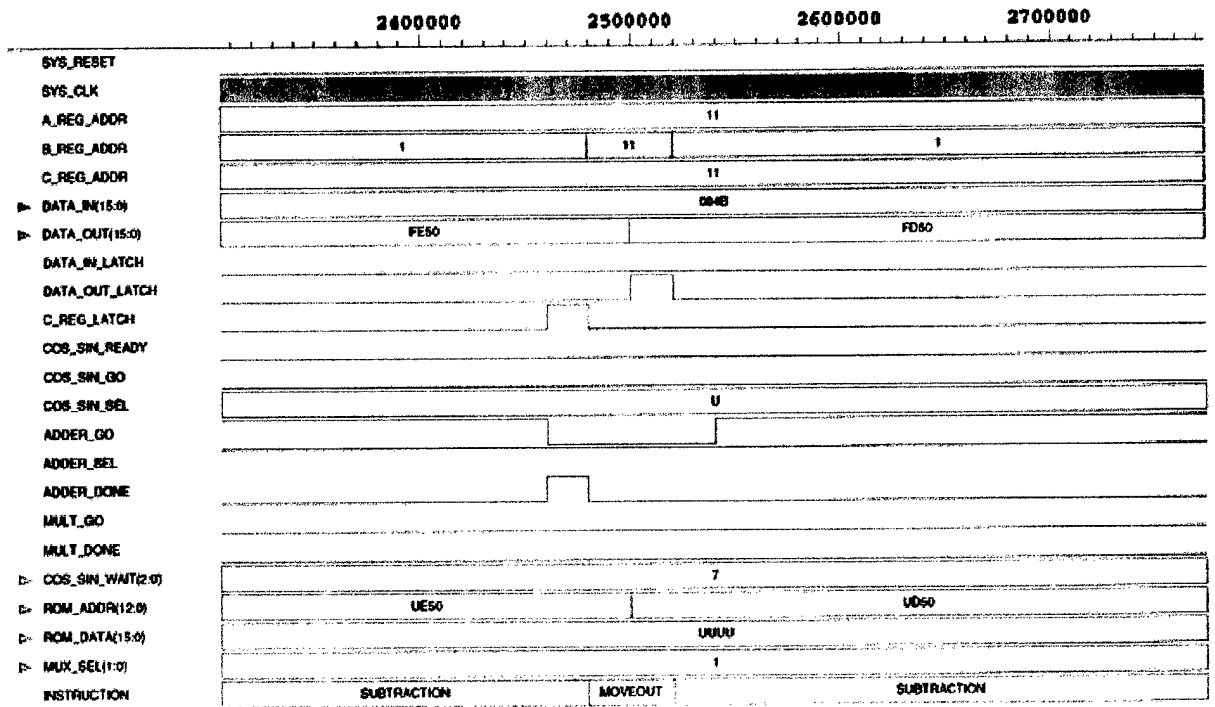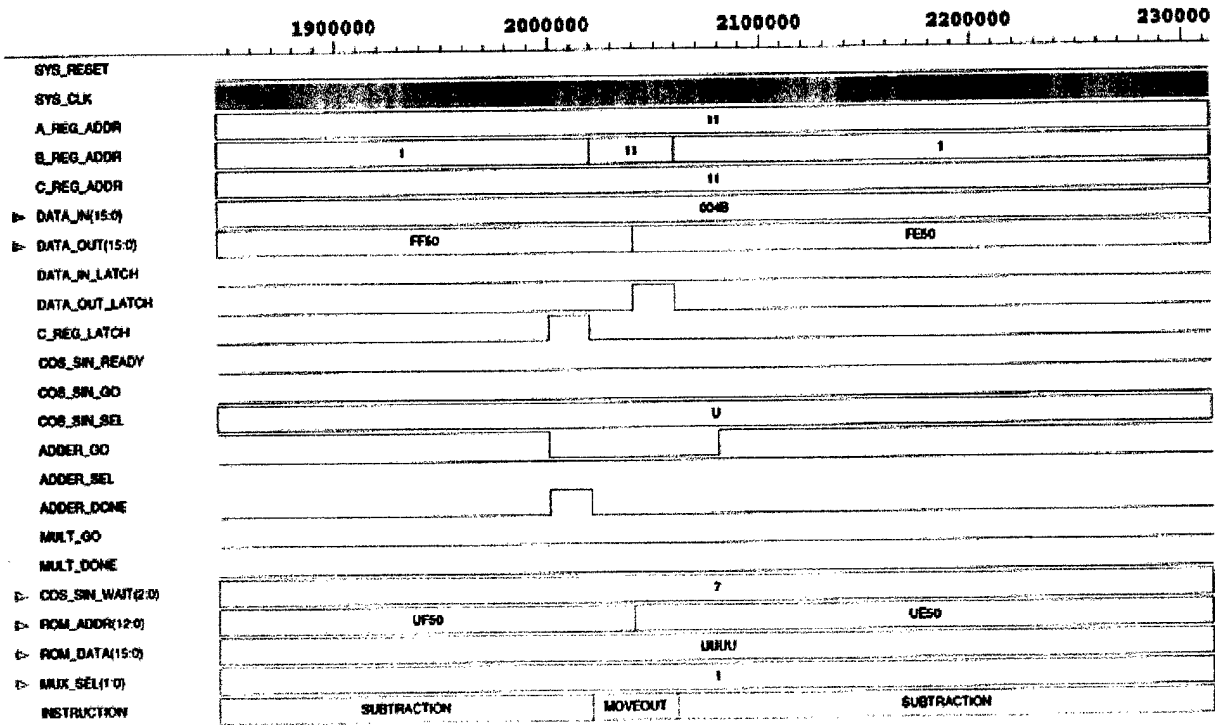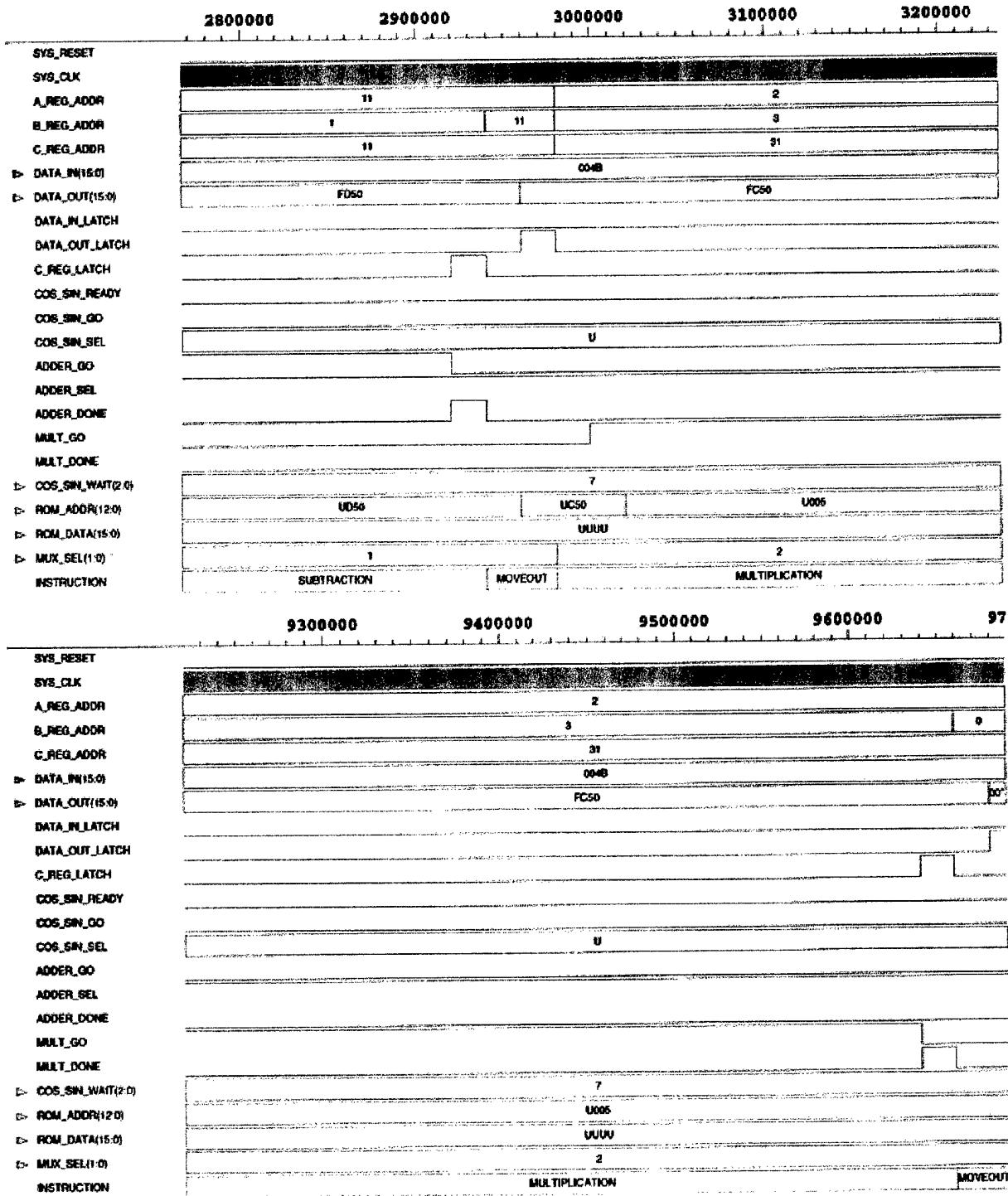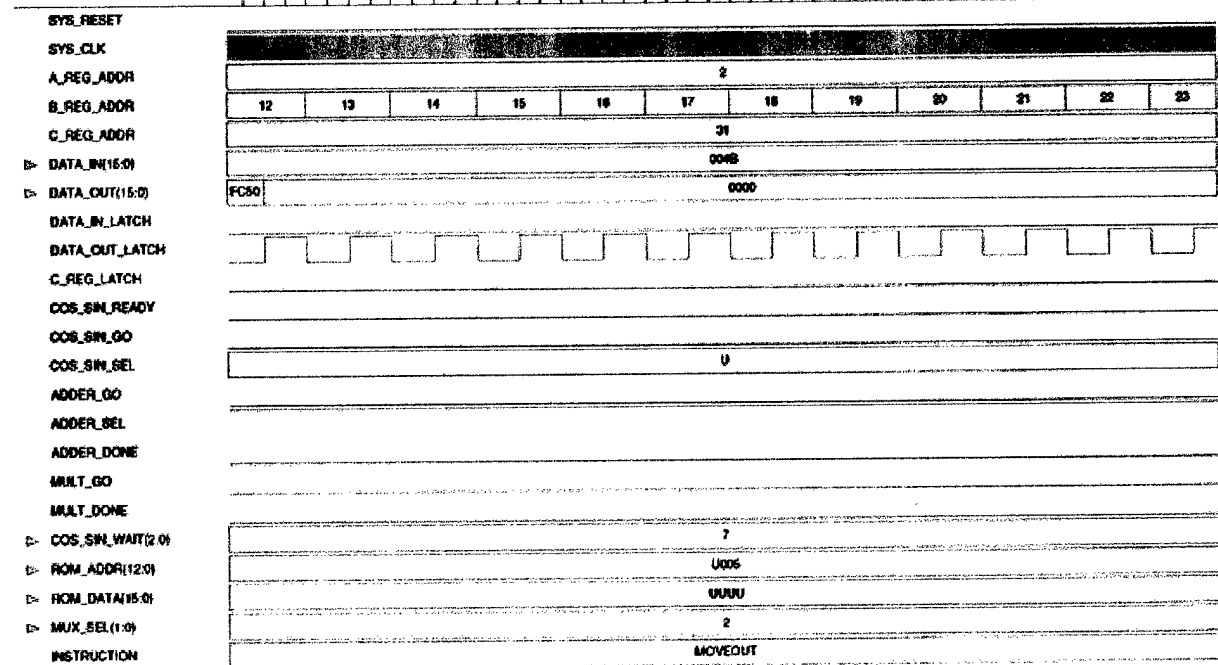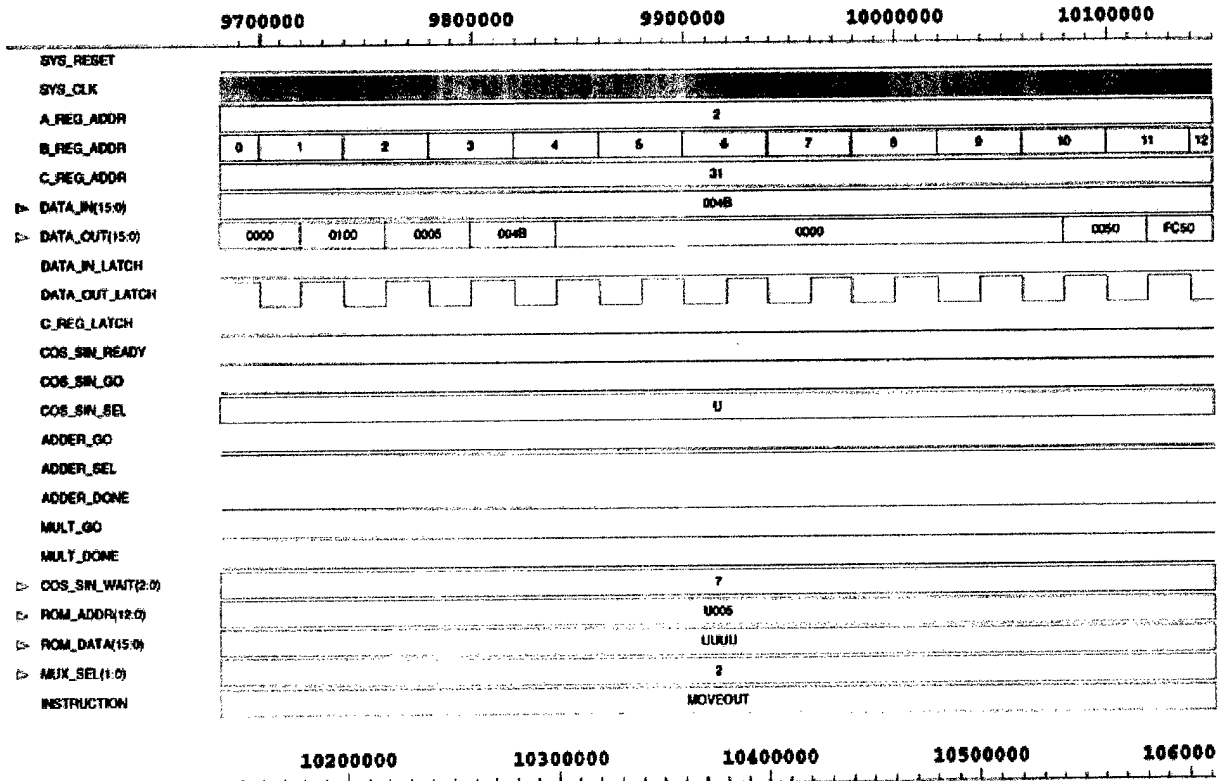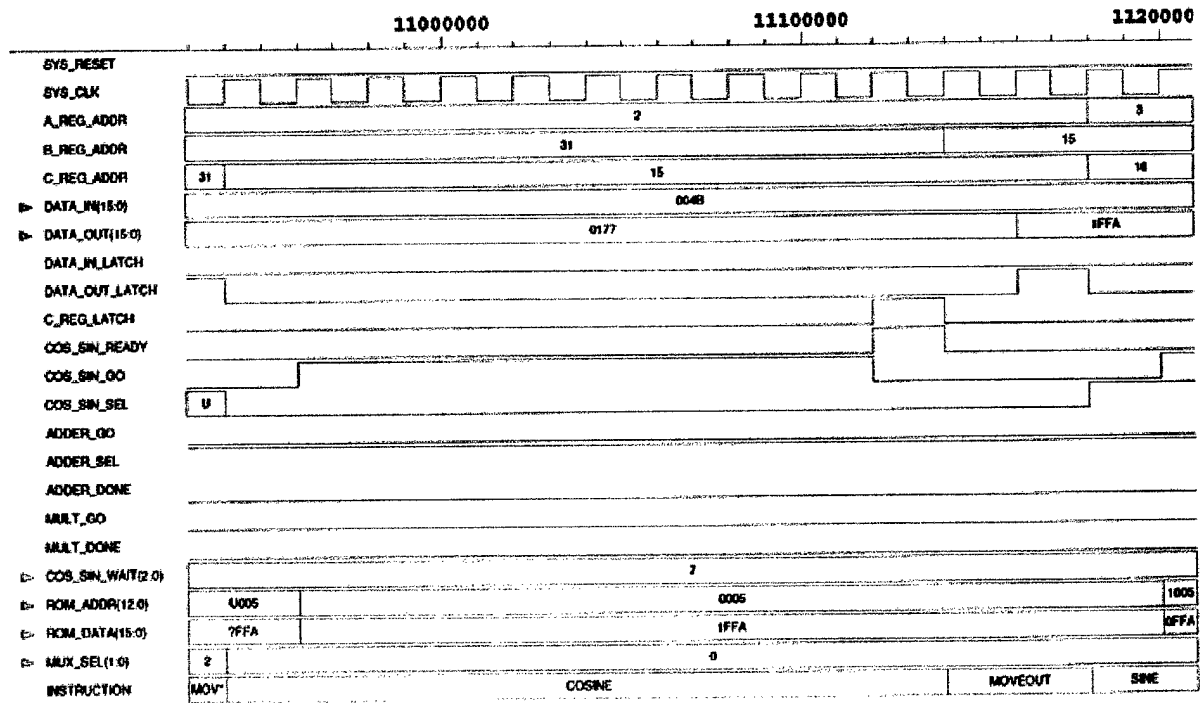
B.7.3 FKP Core Results
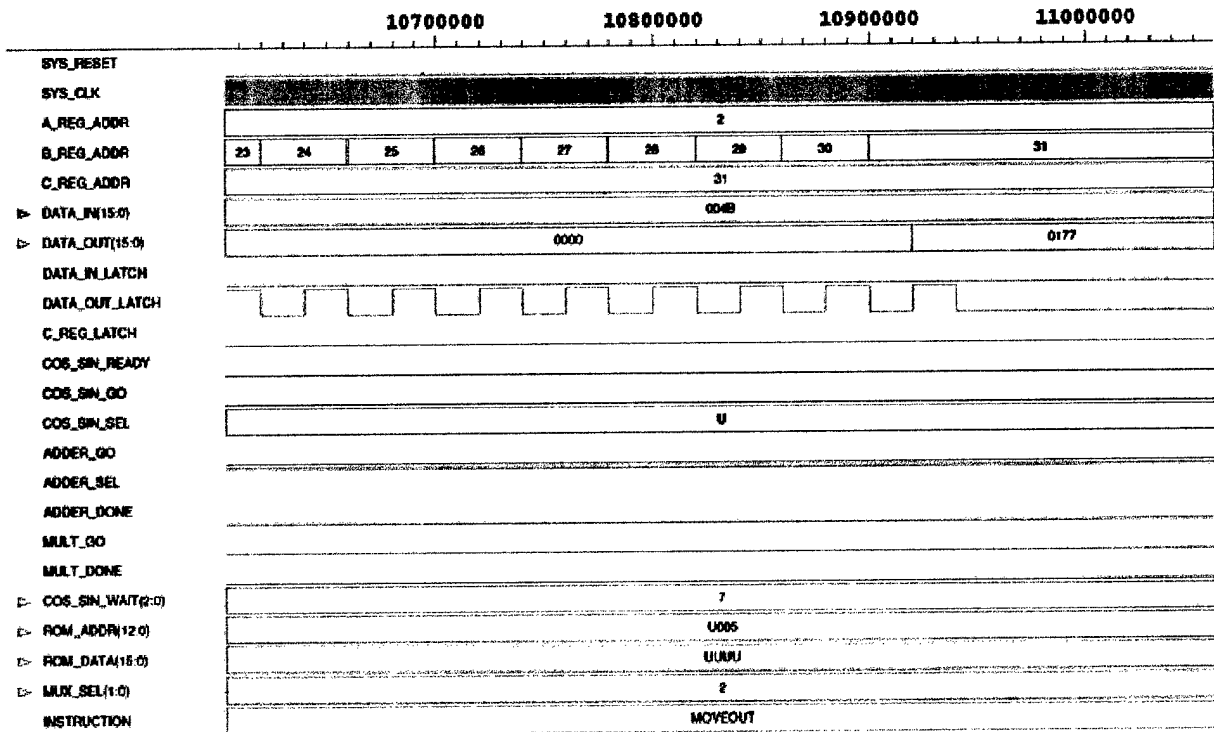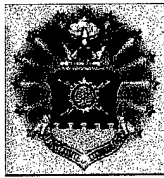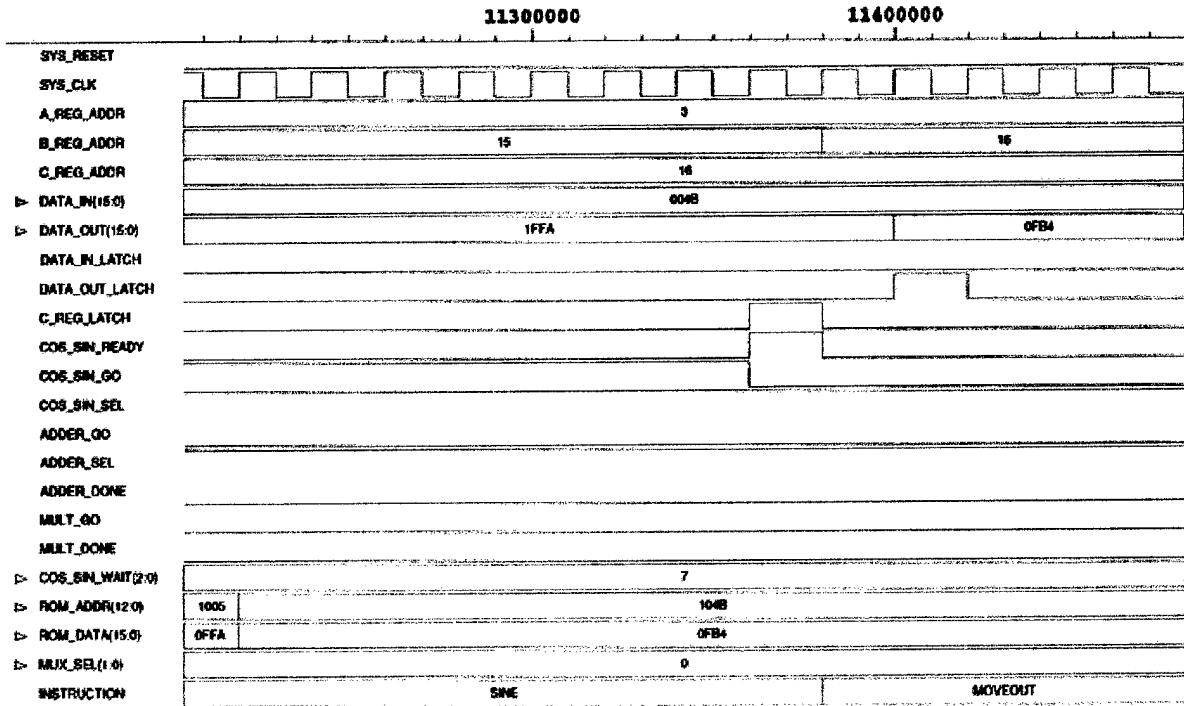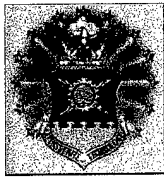
B.5 Microstrore

B.5.1   Microstrore Model

```
-- Project:                    Thesis
-- Filename:                   microstore_head.vhd
-- Other files required:
-- Date:                       Oct 31 97
-- Entity/Architecture Name:   n/a
-- Developer:                  Steve Parmley
```

```vhdl
library IEEE;
  use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;


Package  MICROSTORE is


procedure move_in   (SIGNAL reg: in addr;
                              SIGNAL sys_clk: in std_ulogic;
                              SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                              SIGNAL c_reg_addr: out addr;
                              SIGNAL data_in_latch: out std_ulogic;
                              SIGNAL c_reg_latch: out std_ulogic);

procedure move_out  (SIGNAL reg: in addr;
                              SIGNAL sys_clk: in std_ulogic;
                              SIGNAL b_reg_addr: out addr;
                              SIGNAL data_out_latch: out std_ulogic);

procedure add       (SIGNAL reg1, reg2, reg3: in addr;
                              SIGNAL sys_clk: in std_ulogic;
                              SIGNAL adder_done: in std_ulogic;
                              SIGNAL a_reg_addr, b_reg_addr, c_reg_addr: out addr;
                              SIGNAL adder_sel: out std_ulogic;
                              SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                              SIGNAL adder_go: out std_ulogic;
                              SIGNAL c_reg_latch: out std_ulogic);

procedure sub       (SIGNAL reg1, reg2, reg3: in addr;
                              SIGNAL sys_clk: in std_ulogic;
                              SIGNAL adder_done: in std_ulogic;
                              SIGNAL a_reg_addr, b_reg_addr, c_reg_addr: out addr;
                              SIGNAL adder_sel: out std_ulogic;
                              SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                              SIGNAL adder_go: out std_ulogic;
                              SIGNAL c_reg_latch: out std_ulogic);

procedure mult      (SIGNAL reg1, reg2, reg3: in addr;
                              SIGNAL sys_clk: in std_ulogic;
                              SIGNAL mult_done: in std_ulogic;
                              SIGNAL a_reg_addr, b_reg_addr, c_reg_addr: out addr;
                              SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                              SIGNAL mult_go: out std_ulogic;
                              SIGNAL c_reg_latch: out std_ulogic);

procedure cos       (SIGNAL reg1, reg2:in addr;
                              SIGNAL sys_clk: in std_ulogic;
                              SIGNAL cos_sin_ready: in std_ulogic;
```

```
                    SIGNAL cos_sin_sel: out std_ulogic;
                    SIGNAL a_reg_addr, c_reg_addr: out addr;
                    SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                    SIGNAL cos_sin_go: out std_ulogic;
                    SIGNAL c_reg_latch: out std_ulogic);

procedure sin     (SIGNAL reg1, reg2:in addr;
                    SIGNAL sys_clk: in std_ulogic;
                    SIGNAL cos_sin_ready: in std_ulogic;
                    SIGNAL cos_sin_sel: out std_ulogic;
                    SIGNAL a_reg_addr, c_reg_addr: out addr;
                    SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                    SIGNAL cos_sin_go: out std_ulogic;
                    SIGNAL c_reg_latch: out std_ulogic);


end MICROSTORE;




_____

-- Project:                   Thesis
-- Filename:                  microstore.vhd
-- Other files required:
-- Date:                      Oct 31 97
-- Entity/Architecture Name:  n/a
-- Developer:                 Steve Parmley
_____


library IEEE;
   use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;


Package body MICROSTORE is

-- MOVE_IN (reg)    assume that data is present on input of latch
 procedure move_in   (SIGNAL reg: in addr;
                        SIGNAL sys_clk: in std_ulogic;
                        SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                        SIGNAL c_reg_addr: out addr;
                        SIGNAL data_in_latch: out std_ulogic;
                        SIGNAL c_reg_latch: out std_ulogic) is

  begin
            -- set mux to allow data in latch to reg
            mux_sel <= "11";

            -- set up register to write to
            c_reg_addr <= reg;

            -- latch the data already present on the input of the latch
            data_in_latch <= '1';

            wait until sys_clk'event and sys_clk='1';

            -- hold latched value
            data_in_latch <= '0';

            -- and copy it into register file
            c_reg_latch <= '1';

            wait until sys_clk'event and sys_clk='1';

            -- hold it in register file
```
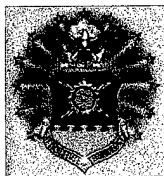
```vhdl
            c_reg_latch <= '0';
    end move_in;


-- MOVE_OUT (reg)
 procedure move_out   (SIGNAL reg: in addr;
                            SIGNAL sys_clk: in std_ulogic;
                            SIGNAL b_reg_addr: out addr;
                            SIGNAL data_out_latch: out std_ulogic) is

    begin
            -- set up register to write to
            b_reg_addr <= reg;

            wait until sys_clk'event and sys_clk='1';

            -- latch the data from the register file to the output
            data_out_latch <= '1';

            wait until sys_clk'event and sys_clk='1';

            -- hold it on the output
            data_out_latch <= '0';
    end move_out;

-- ADD (reg1, reg2, reg3)
  procedure add        (SIGNAL reg1, reg2, reg3: in addr;
                            SIGNAL sys_clk: in std_ulogic;
                            SIGNAL adder_done: in std_ulogic;
                            SIGNAL a_reg_addr, b_reg_addr, c_reg_addr: out addr;
                            SIGNAL adder_sel: out std_ulogic;
                            SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                            SIGNAL adder_go: out std_ulogic;
                            SIGNAL c_reg_latch: out std_ulogic) is

    begin
            -- set up two terms from reg file
            a_reg_addr <= reg2;
            b_reg_addr <= reg3;

            -- set up new register to hold result
            c_reg_addr <= reg1;

            -- set adder/subtractor to add
            adder_sel  <= '0';

            -- set mux to allow add result to go to register
            mux_sel      <= "01";

            wait until sys_clk'event and sys_clk='1';

            -- initiate adder unit
            adder_go <= '1';

            wait until adder_done = '1';
            wait until sys_clk'event and sys_clk='1';

            -- release adder unit
            adder_go <= '0';

            -- latch result into regiter
            c_reg_latch <= '1';

            wait until sys_clk'event and sys_clk='1';

            -- hold result in register
            c_reg_latch <= '0';
```

```
        end add;


-- SUB (reg1, reg2, reg3)
    procedure sub      (SIGNAL reg1, reg2, reg3: in addr;
                                SIGNAL sys_clk: in std_ulogic;
                                SIGNAL adder_done: in std_ulogic;
                                SIGNAL a_reg_addr, b_reg_addr, c_reg_addr: out addr;
                                SIGNAL adder_sel: out std_ulogic;
                                SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                                SIGNAL adder_go: out std_ulogic;
                                SIGNAL c_reg_latch: out std_ulogic) is
    begin
            -- set up two terms from reg file
            a_reg_addr <= reg2;
            b_reg_addr <= reg3;

            -- set up new register to hold result
            c_reg_addr <= reg1;

            -- set adder/subtractor to sub
            adder_sel <= '1';

            -- set mux to allow add result to go to register
            mux_sel    <= "01";

            wait until sys_clk'event and sys_clk='1';

            -- initiate adder unit
            adder_go <= '1';

            wait until adder_done = '1';
            wait until sys_clk'event and sys_clk='1';

            -- release adder unit
            adder_go <= '0';

            -- latch result into regiter
            c_reg_latch <= '1';

            wait until sys_clk'event and sys_clk='1';

            -- hold result in register
            c_reg_latch <= '0';
    end sub;


-- MULTIPLY (reg1, reg2, reg3)
    procedure mult     (SIGNAL reg1, reg2, reg3: in addr;
                                SIGNAL sys_clk: in std_ulogic;
                                SIGNAL mult_done: in std_ulogic;
                                SIGNAL a_reg_addr, b_reg_addr, c_reg_addr: out addr;
                                SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                                SIGNAL mult_go: out std_ulogic;
                                SIGNAL c_reg_latch: out std_ulogic) is
    begin
            -- set up two terms from reg file
            a_reg_addr <= reg2;
            b_reg_addr <= reg3;

            -- set up new register to hold result
            c_reg_addr <= reg1;

            -- set mux to allow mult resutl to go to register
            mux_sel    <= "10";
```

```
                wait until sys_clk'event and sys_clk='1';

                -- initiate multiplier unit
                mult_go <= '1';

                wait until mult_done = '1';
                wait until sys_clk'event and sys_clk='1';

                -- release mult unit
                mult_go <= '0';

                -- latch results into register
                c_reg_latch <= '1';

                wait until sys_clk'event and sys_clk='1';

                -- hold results in register
                c_reg_latch <= '0';
        end mult;

-- COS (reg1, reg2)
    procedure cos       (SIGNAL reg1, reg2:in addr;
                            SIGNAL sys_clk: in std_ulogic;
                            SIGNAL cos_sin_ready: in std_ulogic;
                            SIGNAL cos_sin_sel: out std_ulogic;
                            SIGNAL a_reg_addr, c_reg_addr: out addr;
                            SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                            SIGNAL cos_sin_go: out std_ulogic;
                            SIGNAL c_reg_latch: out std_ulogic) is

    begin
                -- set unit to do cosine
                cos_sin_sel <= '0';

                -- set input to A register
                a_reg_addr <= reg2;

                -- set up mux to allow cos/sin unit to go to registers
                mux_sel <= "00";

                -- set up new register to put result
                c_reg_addr <= reg1;

                wait until sys_clk'event and sys_clk='1';

                -- initiate unit
                cos_sin_go <= '1';

                wait until cos_sin_ready='1';
                wait until sys_clk'event and sys_clk='1';

                -- release unit
                cos_sin_go <= '0';

                -- latch result into register
                c_reg_latch <= '1';

                wait until sys_clk'event and sys_clk='1';

                -- hold result in register
                c_reg_latch <= '0';
        end cos;

-- SIN (reg1, reg2)
    procedure sin       (SIGNAL reg1, reg2:in addr;
```

```
                    SIGNAL sys_clk: in std_ulogic;
                    SIGNAL cos_sin_ready: in std_ulogic;
                    SIGNAL cos_sin_sel: out std_ulogic;
                    SIGNAL a_reg_addr, c_reg_addr: out addr;
                    SIGNAL mux_sel: out std_ulogic_vector(1 downto 0);
                    SIGNAL cos_sin_go: out std_ulogic;
                    SIGNAL c_reg_latch: out std_ulogic) is
  begin
            -- set unit to do sine
            cos_sin_sel <= '1';

            -- set input to A register
            a_reg_addr <= reg2;

            -- set up mux to allow cos/sin unit to go to registers
            mux_sel <= "00";

            -- set up new register to put result
            c_reg_addr <= reg1;

            wait until sys_clk'event and sys_clk='1';

            -- initiate unit
            cos_sin_go <= '1';

            wait until cos_sin_ready='1';
            wait until sys_clk'event and sys_clk='1';

            -- release unit
            cos_sin_go <= '0';

            -- latch result into register
            c_reg_latch <= '1';

            wait until sys_clk'event and sys_clk='1';

            -- hold result in register
            c_reg_latch <= '0';
    end sin;

end MICROSTORE;
```

B.5.2   Microstrore Testbench

```
-- Project:                  Thesis
-- Filename:                 microstore-bench.vhd
-- Other files required:     microstore.vhd
-- Date:                     Oct 31 97
-- Entity/Architecture Name: microstore_tb/test
-- Developer:                Steve Parmley
```

```
library IEEE;
  use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;
use WORK.microstore.all;

entity microstore_tb is
end microstore_tb;

architecture test of microstore_tb is
```

```vhdl
component fkp_core_e
    port (fkp_core_clk                :      in       std_ulogic;
          fkp_core_reset              :      in       std_ulogic;
          fkp_core_data_in            :      in       std_ulogic_vector(15 downto 0);
          fkp_core_data_out           :      out      std_ulogic_vector(15 downto 0);
          fkp_core_data_in_latch      :      in       std_ulogic;
          fkp_core_data_out_latch     :      in       std_ulogic;
          fkp_core_c_reg_latch        :      in       std_ulogic;
          fkp_core_c_reg_addr         :      in       addr;
          fkp_core_a_reg_addr         :      in       addr;
          fkp_core_b_reg_addr         :      in       addr;
          fkp_core_cos_sin_ready      :      out      std_ulogic;
          fkp_core_cos_sin_go         :      in       std_ulogic;
          fkp_core_cos_sin_sel        :      in       std_ulogic;
          fkp_core_cos_sin_wait       :      in       std_ulogic_vector(2 downto 0);
          fkp_core_rom_addr           :      out      std_ulogic_vector(12 downto 0);
          fkp_core_rom_data           :      in       std_ulogic_vector(15 downto 0);
          fkp_core_adder_go           :      in       std_ulogic;
          fkp_core_adder_sel          :      in       std_ulogic;
          fkp_core_adder_done         :      out      std_ulogic;
          fkp_core_mult_go            :      in       std_ulogic;
          fkp_core_mult_done          :      out      std_ulogic;
          fkp_core_mux_sel            :      in       std_ulogic_vector(1 downto 0));
end component;

signal sys_reset, sys_clk : std_ulogic := '0';
signal a_reg_addr, b_reg_addr, c_reg_addr : addr;
signal data_in, data_out : std_ulogic_vector(15 downto 0);
signal data_in_latch, data_out_latch, c_reg_latch, cos_sin_ready : std_ulogic;
signal cos_sin_go, cos_sin_sel, adder_go, adder_sel, adder_done : std_ulogic;
signal mult_go, mult_done                                          : std_ulogic;
signal cos_sin_wait : std_ulogic_vector(2 downto 0);
signal rom_addr : std_ulogic_vector(12 downto 0);
signal rom_data : std_ulogic_vector(15 downto 0);
signal mux_sel : std_ulogic_vector(1 downto 0);

type opcode is (illegal, movein, moveout, move, cosine, sine, addition, subtraction, multiplication);
signal instruction : opcode;
signal reg1, reg2, reg3 : addr;

begin
U1 : fkp_core_e
    PORT MAP  (sys_clk,
               sys_reset,
               data_in,
               data_out,
               data_in_latch,
               data_out_latch,
               c_reg_latch,
               c_reg_addr,
               a_reg_addr,
               b_reg_addr,
               cos_sin_ready,
               cos_sin_go,
               cos_sin_sel,
               cos_sin_wait,
               rom_addr,
               rom_data,
               adder_go,
               adder_sel,
               adder_done,
               mult_go,
               mult_done,
               mux_sel);
```

```
clock : process
begin
        sys_clk <= not(sys_clk);
        wait for 10 ps;
end process clock;

rst : process
begin
        sys_reset <= '1';
        wait for 40 ps;

        sys_reset <= '0';
        wait for 50000 ps;
end process rst;

exercise : process
 begin

        instruction <= illegal;
        data_in_latch <= '0';
        data_out_latch <= '0';
        c_reg_latch <= '0';
        cos_sin_go <= '0';
        cos_sin_wait <= "111";
        adder_go <= '0';
        mult_go <= '0';
        a_reg_addr <= 15;
        b_reg_addr <= 15;
        c_reg_addr <= 15;
        mux_sel <= "00";
        wait for 60 ps;
        wait until sys_clk'event and sys_clk='1';

-- MOVE IN
        instruction <= movein;
        data_in <= "0000000000000101";
        reg1 <= 2;
        wait until sys_clk'event and sys_clk='1';
        move_in(reg1,sys_clk,mux_sel,c_reg_addr,data_in_latch,c_reg_latch);
-- END MOVE IN


-- MOVE OUT
        instruction <= moveout;
        reg1 <= 2;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
-- END MOVE OUT

-- MOVE IN
        instruction <= movein;
        data_in <= "0000000001001011";
        reg1 <= 3;
        wait until sys_clk'event and sys_clk='1';
        move_in(reg1,sys_clk,mux_sel,c_reg_addr,data_in_latch,c_reg_latch);
-- END MOVE IN

-- MOVE OUT
        instruction <= moveout;
        reg1 <= 3;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
-- END MOVE OUT
```

```
-- ADD
        instruction <= addition;
        reg1 <= 10;
        reg2 <= 2;
        reg3 <= 3;
        wait until sys_clk'event and sys_clk='1';
        add(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
                  adder_sel,mux_sel,adder_go,c_reg_latch);
-- END ADD


-- MOVE OUT
        instruction <= moveout;
        reg1 <= 10;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
-- END MOVE OUT

-- MOVE
        instruction <= move;
        reg1 <= 11;
        reg2 <= 0;
        reg3 <= 10;
        wait until sys_clk'event and sys_clk='1';
        add(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
                  adder_sel,mux_sel,adder_go,c_reg_latch);
-- END MOVE


for i in 0 to 3 loop

  -- SUB
        instruction <= subtraction;
        reg1 <= 11;
        reg2 <= 11;
        reg3 <= 1;
        wait until sys_clk'event and sys_clk='1';
        sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
                  adder_sel,mux_sel,adder_go,c_reg_latch);
  -- END ADD


  -- MOVE OUT
        instruction <= moveout;
        reg1 <= 11;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
  -- END MOVE OUT


end loop;

  -- Multiply
        instruction <= multiplication;
        reg1 <= 31;
        reg2 <= 2;
        reg3 <= 3;
        wait until sys_clk'event and sys_clk='1';
        mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
                  mux_sel, mult_go, c_reg_latch);

  -- END ADD
```

```
for i in 0 to 31 loop
  -- MOVE OUT
        instruction <= moveout;
        reg1 <= i;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
  -- END MOVE OUT
end loop;


  -- COSINE
        instruction <= cosine;
        reg2 <= 2;
        reg1 <= 15;
        wait until sys_clk'event and sys_clk='1';
        cos(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
                c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);

  --

  -- MOVE OUT
        instruction <= moveout;
        reg1 <= 15;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
  -- END MOVE OUT

  -- SINE
        instruction <= sine;
        reg2 <= 3;
        reg1 <= 16;
        wait until sys_clk'event and sys_clk='1';
        sin(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
                c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);

  --

  -- MOVE OUT
        instruction <= moveout;
        reg1 <= 16;
        wait until sys_clk'event and sys_clk='1';
        move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
  -- END MOVE OUT

        wait until sys_clk'event and sys_clk='1';
        wait until sys_clk'event and sys_clk='1';
        wait until sys_clk'event and sys_clk='1';

        ASSERT false
                REPORT "DONE"
                SEVERITY failure;

end process exercise;

rom : process
  begin
        wait until rom_addr'event;

        -- make up some rom data (inverse of the address for now)
        rom_data(12 downto 0) <= not(rom_addr(12 downto 0));

        -- fill in the rest
        rom_data(15 downto 13) <= "000";
  end process rom;

end test;
```
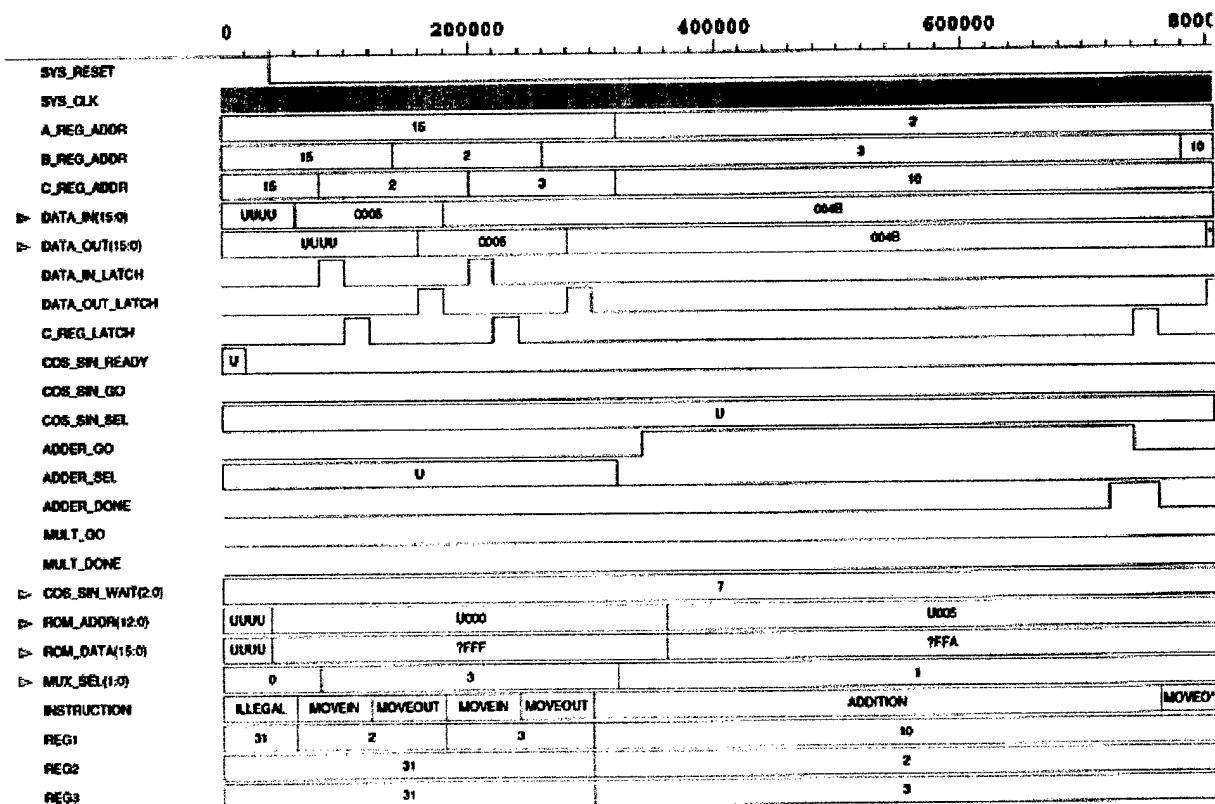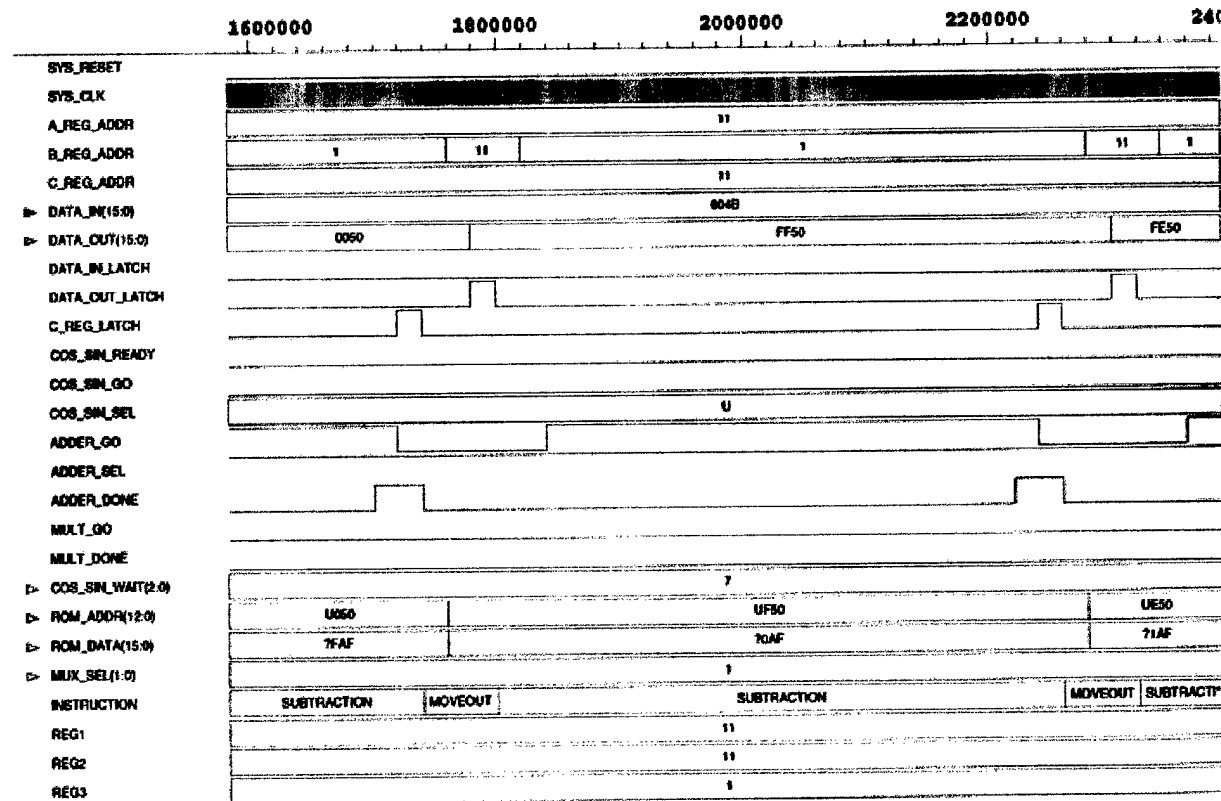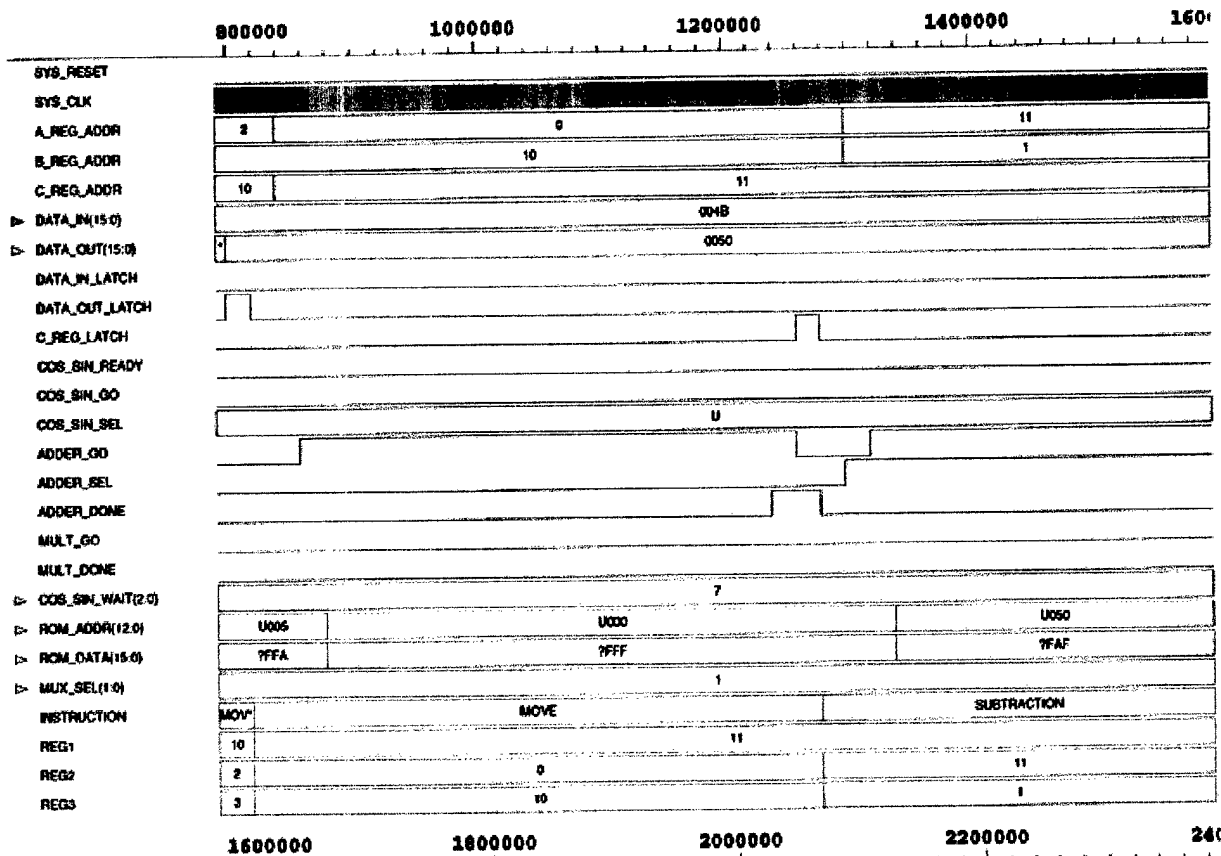
```
CONFIGURATION microstore_c OF microstore_tb IS
  FOR test
    FOR ALL: fkp_core_e
      USE ENTITY WORK.fkp_core_e(structural);
    END FOR;
  END FOR;
END microstore_c;
```
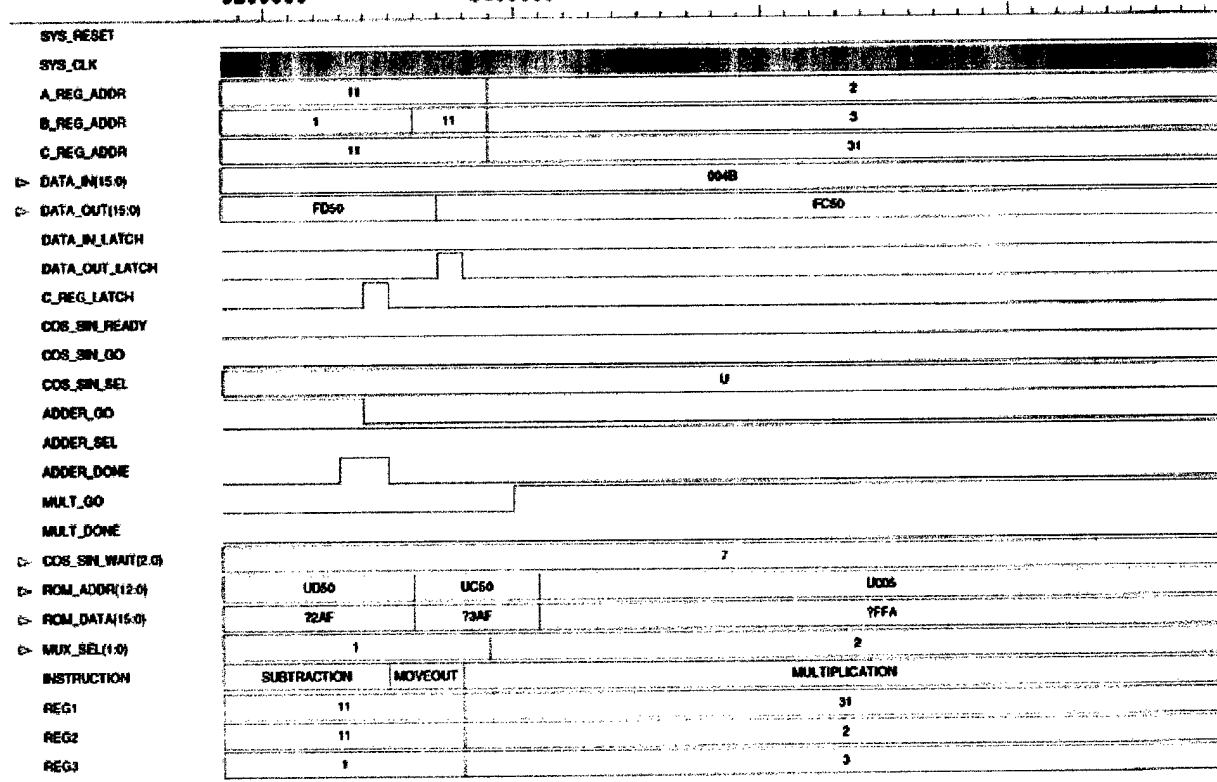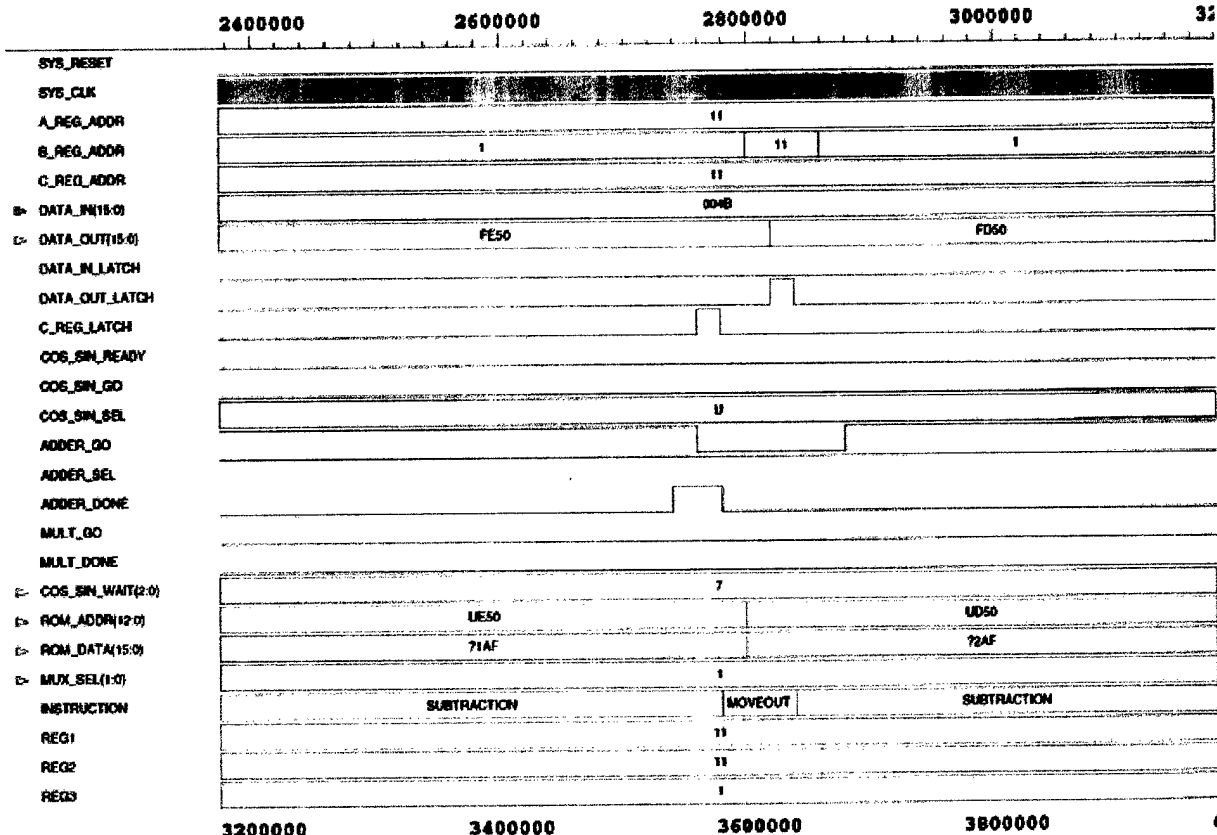
B.5.3   Microstrore Results

11200000   11400000   11600000   11800000

Signal	Value
SYS_RESET	
SYS_CLK	
A_REG_ADDR	2
B_REG_ADDR	16 17 18 19 20 21 22 23 24 25 26 27 28 29
C_REG_ADDR	31
DATA_IN(15:0)	004B
DATA_OUT(15:0)	0000
DATA_IN_LATCH	
DATA_OUT_LATCH	
C_REG_LATCH	
COS_SIN_READY	
COS_SIN_GO	
COS_SIN_SEL	U
ADDER_GO	
ADDER_SEL	
ADDER_DONE	
MULT_GO	
MULT_DONE	
COS_SIN_WAIT(2:0)	7
ROM_ADDR(12:0)	U005
ROM_DATA(15:0)	7FFA
MUX_SEL(1:0)	2
INSTRUCTION	MOVEOUT
REG1	16 17 18 19 20 21 22 23 24 25 26 27 28 29
REG2	2
REG3	3

12000000   12200000   12400000   12600000

Signal	Value
SYS_RESET	
SYS_CLK	
A_REG_ADDR	2 3
B_REG_ADDR	29 30 31 15 16
C_REG_ADDR	31 15 16
DATA_IN(15:0)	004B
DATA_OUT(15:0)	0000 0177 1FFA 0FB4
DATA_IN_LATCH	
DATA_OUT_LATCH	
C_REG_LATCH	
COS_SIN_READY	
COS_SIN_GO	
COS_SIN_SEL	U
ADDER_GO	
ADDER_SEL	
ADDER_DONE	
MULT_GO	
MULT_DONE	
COS_SIN_WAIT(2:0)	7
ROM_ADDR(12:0)	U005 0005 10* 104B
ROM_DATA(15:0)	7FFA 1FFA 0F* 0FB4
MUX_SEL(1:0)	2 0
INSTRUCTION	MOVEOUT COSINE MOVEOUT SINE MOVEOUT
REG1	30 31 15 16
REG2	2 3
REG3	3

B.9 Control

B.5.9   Control Model

```
— Project:                              Thesis
— Filename:                             fkp.vhd
— Other files required:                 all FKP files
— Date:                                 Oct 17 97
— Entity/Architecture Name:             fkp_e/behavior
— Developer:                            Steve Parmley
— Function:
— Limitations:
— History:
— Last Analyzed On:


library IEEE;
  use IEEE.std_logic_1164.all;

use WORK.reg_file_pkg.all;
use WORK.microstore.all;

entity fkp_e is
  port (fkp_cntprt7_clock              :       in       std_ulogic;
          fkp_cntprt6_reset            :       in       std_ulogic;
          fkp_cntprt5_strobe           :       in       std_ulogic;
          fkp_cntprt4_ready            :       out      std_ulogic;
          fkp_cntprt3_dgv              :       out      std_ulogic;
          fkp_cntprt2_dga              :       in       std_ulogic;
—         fkp_cntprt1_dsv              :       in       std_ulogic;
—         fkp_cntprt0_dsa              :       out      std_ulogic;

          fkp_cmdprt6_cmd1             :       in       std_ulogic;
          fkp_cmdprt5_cmd0             :       in       std_ulogic;
          fkp_cmdprt4_a4               :       in       std_ulogic;
          fkp_cmdprt3_a3               :       in       std_ulogic;
          fkp_cmdprt2_a2               :       in       std_ulogic;
          fkp_cmdprt1_a1               :       in       std_ulogic;
          fkp_cmdprt0_a0               :       in       std_ulogic;

          fkp_data_in                  :       in       std_ulogic_vector(15 downto 0);
          fkp_data_out                 :       out      std_ulogic_vector(15 downto 0);

          fkp_rom_addr                 :       out      std_ulogic_vector(12 downto 0);
          fkp_rom_data                 :       in       std_ulogic_vector(15 downto 0));
end fkp_e;

architecture structural of fkp_e is

-- SIGNALS
signal sys_reset, sys_clk : std_ulogic := '0';
signal a_reg_addr, b_reg_addr, c_reg_addr : addr;
signal data_in, data_out : std_ulogic_vector(15 downto 0);
signal data_in_latch, data_out_latch, c_reg_latch, cos_sin_ready : std_ulogic;
signal cos_sin_go, cos_sin_sel, adder_go, adder_sel, adder_done : std_ulogic;
signal mult_go, mult_done                                       : std_ulogic;
signal cos_sin_wait : std_ulogic_vector(2 downto 0);
signal rom_addr : std_ulogic_vector(12 downto 0);
signal rom_data : std_ulogic_vector(15 downto 0);
signal mux_sel : std_ulogic_vector(1 downto 0);

type opcode is (illegal, movein, moveout, move, cosine, sine, addition, subtraction, multiplication);
```

```vhdl
signal instruction : opcode;
signal reg1, reg2, reg3 : addr;

signal state : integer;


-- COMPONENTS
component fkp_core_e
    port (fkp_core_clk              :       in      std_ulogic;
            fkp_core_reset          :       in      std_ulogic;
            fkp_core_data_in        :       in      std_ulogic_vector(15 downto 0);
            fkp_core_data_out       :       out     std_ulogic_vector(15 downto 0);
            fkp_core_data_in_latch  :       in      std_ulogic;
            fkp_core_data_out_latch :       in      std_ulogic;
            fkp_core_c_reg_latch    :       in      std_ulogic;
            fkp_core_c_reg_addr     :       in      addr;
            fkp_core_a_reg_addr     :       in      addr;
            fkp_core_b_reg_addr     :       in      addr;
            fkp_core_cos_sin_ready  :       out     std_ulogic;
            fkp_core_cos_sin_go     :       in      std_ulogic;
            fkp_core_cos_sin_sel    :       in      std_ulogic;
            fkp_core_cos_sin_wait   :       in      std_ulogic_vector(2 downto 0);
            fkp_core_rom_addr       :       out     std_ulogic_vector(12 downto 0);
            fkp_core_rom_data       :       in      std_ulogic_vector(15 downto 0);
            fkp_core_adder_go       :       in      std_ulogic;
            fkp_core_adder_sel      :       in      std_ulogic;
            fkp_core_adder_done     :       out     std_ulogic;
            fkp_core_mult_go        :       in      std_ulogic;
            fkp_core_mult_done      :       out     std_ulogic;
            fkp_core_mux_sel        :       in      std_ulogic_vector(1 downto 0));
end component;


begin
U1 : fkp_core_e
    PORT MAP    (sys_clk,
                sys_reset,
                data_in,
                data_out,
                data_in_latch,
                data_out_latch,
                c_reg_latch,
                c_reg_addr,
                a_reg_addr,
                b_reg_addr,
                cos_sin_ready,
                cos_sin_go,
                cos_sin_sel,
                cos_sin_wait,
                rom_addr,
                rom_data,
                adder_go,
                adder_sel,
                adder_done,
                mult_go,
                mult_done,
                mux_sel);


controller : process
variable r1 : integer;
    begin
            sys_clk <= fkp_cntprt7_clock;

            -- system wide reset ?
```

```
if fkp_cntprt6_reset = '1' then
        sys_reset <= '1';
        wait until sys_clk'event and sys_clk='1';
        sys_reset <= '1';
        state <= 0;
        fkp_cntprt4_ready <= '1';
        fkp_cntprt3_dgv <= '0';
end if;

-- ready to accept command
if state = 0 then
        -- either set, get, or run
        if fkp_cntprt5_strobe = '1' then
                -- set
                if fkp_cmdprt6_cmd1='0' and fkp_cmdprt5_cmd0='0' then
                        -- set not ready flag
                        fkp_cntprt4_ready <= '0';

                        -- set the register designated by the a4-a0 bits to
                        -- the data from the input data bus

                        -- MOVE IN
                        instruction <= movein;
                        data_in <= fkp_data_in;

                        -- transform bits to integer
                        r1 := 0;
                        if fkp_cmdprt4_a4 = '1' then
                                r1 := r1 + 16;
                        end if;
                        if fkp_cmdprt3_a3 = '1' then
                                r1 := r1 + 8;
                        end if;
                        if fkp_cmdprt2_a2 = '1' then
                                r1 := r1 + 4;
                        end if;
                        if fkp_cmdprt1_a1 = '1' then
                                r1 := r1 + 2;
                        end if;
                        if fkp_cmdprt0_a0 = '1' then
                                r1 := r1 + 1;
                        end if;

                        -- set target register
                        reg1 <= r1;

                        wait until sys_clk'event and sys_clk='1';

                        move_in(reg1,sys_clk,mux_sel,c_reg_addr,data_in_latch,c_reg_latch);
                        -- END MOVE IN

                        wait until fkp_cntprt5_strobe = '0';

                        -- set ready flag
                        fkp_cntprt4_ready <= '1';

                -- get
                elsif fkp_cmdprt6_cmd1='0' and fkp_cmdprt5_cmd0='1' then
                        -- set not ready flag
                        fkp_cntprt4_ready <= '0';

                        -- get the register designated by the a4-a0 bits to
                        -- the data from the input data bus

                        -- MOVE OUT
```

```
instruction <= moveout;

-- transform bits to integer
r1 := 0;
if fkp_cmdprt4_a4 = '1' then
        r1 := r1 + 16;
end if;
if fkp_cmdprt3_a3 = '1' then
        r1 := r1 + 8;
end if;
if fkp_cmdprt2_a2 = '1' then
        r1 := r1 + 4;
end if;
if fkp_cmdprt1_a1 = '1' then
        r1 := r1 + 2;
end if;
if fkp_cmdprt0_a0 = '1' then
        r1 := r1 + 1;
end if;

-- set target register
reg1 <= r1;

wait until sys_clk'event and sys_clk='1';
move_out(reg1,sys_clk,b_reg_addr,data_out_latch);
-- END MOVE OUT

fkp_data_out <= data_out;

-- let user know data is valid
fkp_cntprt3_dgv <= '1';

wait until fkp_cntprt2_dga ='1';
-- user has data

-- release dgv
fkp_cntprt3_dgv <= '0';

wait until fkp_cntprt5_strobe = '0';

-- set ready flag
fkp_cntprt4_ready <= '1';


-- run
elsif fkp_cmdprt6_cmd1='1' and fkp_cmdprt5_cmd0='0' then
        -- set not ready flag
        fkp_cntprt4_ready <= '0';

        -- ASSUME that the 5 constansts are located in r2,r3,r4,r5,r6
        -- ASSUME that the 4 angles are located in r7,r8,r9,r10
        -- this was accomplished using the set function

        -- See table 4.4b of Thesis for order of operations

        -- **** STEP 1 ****
        -- desc:  reg 26 = sin of theta 1
        instruction <= sine;
        reg1 <= 26;
        reg2 <= 7;
        wait until sys_clk'event and sys_clk='1';
        sin(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
                c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);

        -- **** STEP 2 ****
```

```
-- desc:  reg 11 = cos of theta 1
instruction <= cosine;
reg1 <= 11;
reg2 <= 7;
wait until sys_clk'event and sys_clk='1';
cos(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 3 ****
-- desc:  reg 12 = sin of theta 2
instruction <= sine;
reg1 <= 12;
reg2 <= 8;
wait until sys_clk'event and sys_clk='1';
sin(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 4 ****
-- desc:  reg 13 = cos of theta 2
instruction <= cosine;
reg1 <= 13;
reg2 <= 8;
wait until sys_clk'event and sys_clk='1';
cos(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 5 ****
-- desc:  reg 14 = theta 2 + theta 3
instruction <= addition;
reg1 <= 14;
reg2 <= 8;
reg3 <= 9;
wait until sys_clk'event and sys_clk='1';
add(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
          adder_sel,mux_sel,adder_go,c_reg_latch);


-- **** STEP 6 ****
-- desc:  reg 15 = sin of theta 2+3
instruction <= sine;
reg1 <= 15;
reg2 <= 14;
wait until sys_clk'event and sys_clk='1';
sin(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 7 ****
-- desc:  reg 16 = cos of theta 2+3
instruction <= cosine;
reg1 <= 16;
reg2 <= 14;
wait until sys_clk'event and sys_clk='1';
cos(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 8 ****
-- desc:  reg 14 = theta 2 + theta 3 + theta 4
instruction <= addition;
reg1 <= 14;
reg2 <= 14;
reg3 <= 10;
wait until sys_clk'event and sys_clk='1';
add(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
          adder_sel,mux_sel,adder_go,c_reg_latch);


-- **** STEP 9 ****
```

```
-- desc:  reg 22 = sin of theta 2+3+4
instruction <= sine;
reg1 <= 22;
reg2 <= 14;
wait until sys_clk'event and sys_clk='1';
sin(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 10 ****
-- desc:  reg 25 = cos of theta 2+3+4
instruction <= cosine;
reg1 <= 25;
reg2 <= 14;
wait until sys_clk'event and sys_clk='1';
cos(reg1,reg2, sys_clk,cos_sin_ready,cos_sin_sel,a_reg_addr,
          c_reg_addr, mux_sel,cos_sin_go,c_reg_latch);


-- **** STEP 11 ****
-- desc:  reg 20 = cos (th1) * cos (th2+th3+th4)
instruction <= multiplication;
reg1 <= 20;
reg2 <= 11;
reg3 <= 25;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
          mux_sel, mult_go, c_reg_latch);


-- **** STEP 12 ****
-- desc:  reg 21 = sin (th1) * cos (th2+th3+th4)
instruction <= multiplication;
reg1 <= 21;
reg2 <= 26;
reg3 <= 25;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
          mux_sel, mult_go, c_reg_latch);


-- **** STEP 13 ****
-- desc:  reg 23 = cos (th1) * sin (th2+th3+th4)
instruction <= multiplication;
reg1 <= 23;
reg2 <= 11;
reg3 <= 22;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
          mux_sel, mult_go, c_reg_latch);


-- **** STEP 14 ****
-- desc:  reg 23 = -(cos(th1) * sin(th2+th3+th4)
instruction <= subtraction;
reg1 <= 23;
reg2 <= 0;
reg3 <= 23;
wait until sys_clk'event and sys_clk='1';
sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
          adder_sel,mux_sel,adder_go,c_reg_latch);


-- **** STEP 15 ****
-- desc:  reg 24 = sin (th1) * sin (th2+th3+th4)
instruction <= multiplication;
reg1 <= 24;
reg2 <= 26;
reg3 <= 22;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
```

```
                    mux_sel, mult_go, c_reg_latch);

    -- **** STEP 16 ****
    -- desc:  reg 24 = -(sin(th1) * sin(th2+th3+th4)
    instruction <= subtraction;
    reg1 <= 24;
    reg2 <= 0;
    reg3 <= 24;
    wait until sys_clk'event and sys_clk='1';
    sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
            adder_sel,mux_sel,adder_go,c_reg_latch);

    -- **** STEP 17 ****
    -- desc:  reg 27 = -(cos(th1))
    instruction <= subtraction;
    reg1 <= 27;
    reg2 <= 0;
    reg3 <= 11;
    wait until sys_clk'event and sys_clk='1';
    sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
            adder_sel,mux_sel,adder_go,c_reg_latch);

    -- **** STEP 18 ****
    -- desc:  reg 28 = 0
    instruction <= addition;
    reg1 <= 28;
    reg2 <= 0;
    reg3 <= 0;
    wait until sys_clk'event and sys_clk='1';
    sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
            adder_sel,mux_sel,adder_go,c_reg_latch);

    -- **** STEP 19 ****
    -- desc:  reg 17 = a2 * cos (th2)
    instruction <= multiplication;
    reg1 <= 17;
    reg2 <= 4;
    reg3 <= 13;
    wait until sys_clk'event and sys_clk='1';
    mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
            mux_sel, mult_go, c_reg_latch);

    -- **** STEP 20 ****
    -- desc:  reg 18 = a3 * cos (th2+th3)
    instruction <= multiplication;
    reg1 <= 18;
    reg2 <= 5;
    reg3 <= 16;
    wait until sys_clk'event and sys_clk='1';
    mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
            mux_sel, mult_go, c_reg_latch);

    -- **** STEP 21 ****
    -- desc:  reg 17 = a2*cos(th2) + a3*cos(th2+th3)
    instruction <= addition;
    reg1 <= 17;
    reg2 <= 17;
    reg3 <= 18;
    wait until sys_clk'event and sys_clk='1';
    sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
            adder_sel,mux_sel,adder_go,c_reg_latch);

    -- **** STEP 22 ****
    -- desc:  reg 17 = a1 + a2*cos(th2) + a3*cos(th2+th3)
    instruction <= addition;
```

```
reg1 <= 17;
reg2 <= 17;
reg3 <= 3;
wait until sys_clk'event and sys_clk='1';
sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
        adder_sel,mux_sel,adder_go,c_reg_latch);


-- **** STEP 23 ****
-- desc:  reg 18 = cos(th1) * (a1 + a2*cos(th2) + a3 * cos (th2+th3))
instruction <= multiplication;
reg1 <= 18;
reg2 <= 17;
reg3 <= 11;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
        mux_sel, mult_go, c_reg_latch);


-- **** STEP 24 ****
-- desc:  reg 29 = a0 + cos(th1)*(a1 + a2*cos(th2) + a3*cos(th2+th3)
instruction <= addition;
reg1 <= 29;
reg2 <= 18;
reg3 <= 2;
wait until sys_clk'event and sys_clk='1';
sub(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
        adder_sel,mux_sel,adder_go,c_reg_latch);


-- **** STEP 25 ****
-- desc:  reg 30 = sin(th1) * (a1 + a2*cos(th2) + a3 * cos (th2+th3))
instruction <= multiplication;
reg1 <= 30;
reg2 <= 17;
reg3 <= 26;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
        mux_sel, mult_go, c_reg_latch);


-- **** STEP 26 ****
-- desc:  reg 19 = a2*sin(th2)
instruction <= multiplication;
reg1 <= 19;
reg2 <= 4;
reg3 <= 12;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
        mux_sel, mult_go, c_reg_latch);


-- **** STEP 27 ****
-- desc:  reg 31 =  a3 * sin (th2+th3))
instruction <= multiplication;
reg1 <= 31;
reg2 <= 5;
reg3 <= 15;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
        mux_sel, mult_go, c_reg_latch);


-- **** STEP 28 ****
-- desc:  reg 31 = a2 * sin(th2) + a3 * sin (th2+th3))
instruction <= addition;
reg1 <= 31;
reg2 <= 31;
reg3 <= 19;
wait until sys_clk'event and sys_clk='1';
mult(reg1,reg2,reg3,sys_clk,mult_done,a_reg_addr,b_reg_addr,c_reg_addr,
```
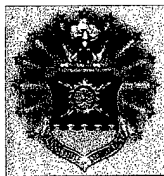
```
                              mux_sel, mult_go, c_reg_latch);

                    -- **** STEP 29 ****
                    -- desc:  reg 31 = a2 * sin(th2) + a3 * sin (th2+th3)) + d1
                    instruction <= addition;
                    reg1 <= 31;
                    reg2 <= 31;
                    reg3 <= 6;
                    wait until sys_clk'event and sys_clk='1';
                    add(reg1,reg2,reg3,sys_clk,adder_done,a_reg_addr,b_reg_addr,c_reg_addr,
                    adder_sel,mux_sel,adder_go,c_reg_latch);


                    wait until fkp_cntprt5_strobe = '0';

                    -- set ready flag
                    fkp_cntprt4_ready <= '1';


              end if;
          end if;
      end if;

  end process controller;
end structural;
```

Appendix C: XACTstep Synthesis Log File for Register File

ngdbuild -p xc4000e C:\exemplar\work\reg16\reg16.xnf xc4000e.ngd
ngdbuild: version M1.3.7
Copyright (c) 1995-1997 Xilinx, Inc. All rights reserved.

Command Line: ngdbuild -p xc4000e C:\exemplar\work\reg16\reg16.xnf xc4000e.ngd

Launcher: Using rule XNF_RULE
Launcher: reg16.ngo being compiled because it does not exist
Launcher: Running xnf2ngd from C:\exemplar\work\reg16\xproj\ver1\
Launcher: Executing xnf2ngd -p xc4000e -u "C:\exemplar\work\reg16\reg16.xnf"
"reg16.ngo"
xnf2ngd: version M1.3.7
Copyright (c) 1995-1997 Xilinx, Inc. All rights reserved.
   using XNF gate model
   reading XNF file "C:/exemplar/work/reg16/reg16.xnf" ...
   Writing NGO file "reg16.ngo" ...
Launcher: "xnf2ngd" exited with an exit code of 0.

Reading NGO file "C:/exemplar/work/reg16/xproj/ver1/reg16.ngo" ...
Reading component libraries for design expansion...

Running Timing Specification DRC...
Timing Specification DRC complete with no errors or warnings.

Running Logical Design DRC...
Logical Design DRC complete with no errors or warnings.

NGDBUILD Design Results Summary:
   2148 total blocks expanded.
Writing NGD file "xc4000e.ngd" ...

Writing NGDBUILD log file "xc4000e.bld"...

NGDBUILD Done.

map -p xc4020e-3-hq208 -o map.ncd ../xc4000e.ngd reg16.pcf
map: version M1.3.7
Copyright (c) 1995-1997 Xilinx, Inc. All rights reserved.
Reading NGD file "../xc4000e.ngd"...
Using target part "4020ehq208-3".
MAP xc4000e directives:
   Partname="xc4020e-3-hq208".
   No Guide File specified.
   No Guide Mode specified.
   Covermode="area".
   Coverlutsize=4.
   Coverfgsize=4.
   Perform logic replication.
   Pack CLBs to 97%.
Processing logical timing constraints...
Running general design DRC...
Verifying F/HMAP validity based on pre-trimmed logic...
Removing unused logic...
Processing global clock buffers...
WARNING:baste:24 - All of the external outputs in this design are using
   slew-rate-limited output drivers. The delay on speed critical outputs can be
   dramatically reduced by designating them as fast outputs in the original
   design. Please see your vendor interface documentation for specific
   information on how to do this within your design-entry tool.
Optimizing...
Removed Logic Summary:

Design Summary:
    Number of warnings: 1
    Number of errors: 0
    Number of CLBs: 315 out of 784
        Flops/latches: 224
        4 input LUTs: 621
        3 input LUTs: 183
    Number of bonded IOBs: 63 out of 160
    Number of clock IOBs: 1 out of   8
        IO flops/latches: 32
    Number of primary CLKs: 1 out of   4
Writing design file "map.ncd"...

par -w -l 4 -d 0 map.ncd reg16.ncd reg16.pcf
PAR: Xilinx Place And Route M1.3.7.
Copyright (c) 1995-1997 Xilinx, Inc. All rights reserved.

Constraints file: reg16.pcf

Placement level-cost: 4-1

Loading device database for application par from file "map.ncd".
    "reg16" is an NCD, version 2.27, device xc4020e, package hq208, speed -3
Loading device for application par from file '4020e.nph' in environment
d:/xilinx.
Device speed data version: x1_0.79 PRELIMINARY.

Device utilization summary:

    IO   63/224   28% used
                    63/160   39% bonded
    LOGIC   315/784   40% used
    SPECIAL   1/3023   0% used

    CLKIOB   1/8   12% used
    IOB   62/224   27% used

    CLB   315/784   40% used

    PRI-CLK   1/4   25% used

Starting initial Placement phase. REAL time: 13 secs
Finished initial Placement phase. REAL time: 14 secs

Starting Constructive Placer. REAL time: 15 secs .
Placer score = 1081980
Placer score = 977380
Placer score = 886140
Placer score = 853480
Placer score = 783540
Placer score = 705220
Placer score = 634260
Placer score = 577740
Placer score = 486240
Placer score = 439200
Placer score = 375240
Placer score = 332160
Placer score = 298500
Placer score = 284400
Placer score = 271260
Placer score = 260940

Placer score = 255660
Placer score = 252840
Placer score = 248700
Placer score = 246900
Placer score = 245640
Placer score = 244680
Placer score = 244320
Placer score = 242160
Placer score = 241920
Placer score = 241140
Placer score = 240240
Placer score = 239220
Placer score = 238920
Placer score = 238560
Placer score = 237900
Finished Constructive Placer. REAL time: 11 mins 30 secs

Dumping design to file "reg16.ncd".

Starting Optimizing Placer. REAL time: 11 mins 31 secs
Optimizing ......
Swapped 30 comps.
Xilinx Placer [1]   235080   REAL time: 12 mins 40 secs
Optimizing .....
Swapped 5 comps.
Xilinx Placer [2]   234840   REAL time: 13 mins 45 secs
Finished Optimizing Placer. REAL time: 13 mins 45 secs

Dumping design to file "reg16.ncd".

Total REAL time to Placer completion: 13 mins 47 secs
Total CPU time to Placer completion: 13 mins 47 secs

0 connection(s) routed; 2231 unrouted.
Starting router resource preassignment
Completed router resource preassignment. Real time: 13 mins 49 secs
Starting iterative routing.
End of iteration 1
2231 successful; 0 unrouted; (0) real time: 14 mins
Constraints are met.
Power and ground nets completely routed.
Dumping design to file "reg16.ncd".
Starting cleanup
End of cleanup iteration 1
2231 successful; 0 unrouted; (0) real time: 15 mins 17 secs
Dumping design to file "reg16.ncd".
Total CPU time 15 mins 18 secs
Total REAL time: 15 mins 18 secs
Completely routed.
End of route. 2231 routed (100.00%); 0 unrouted.
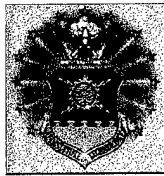No errors found.

Total REAL time to Router completion: 15 mins 20 secs
Total CPU time to Router completion: 15 mins 20 secs

Generating PAR statistics.
Timing Score: 0

Dumping design to file "reg16.ncd".

All signals are completely routed.

Total REAL time to PAR completion: 15 mins 28 secs
Total CPU time to PAR completion: 15 mins 28 secs

PAR done.

bitgen reg16.ncd -l -w -f bitgen.ut

Loading device database for application Bitgen from file "reg16.ncd".
  "reg16" is an NCD, version 2.27, device xc4020e, package hq208, speed -3
Loading device for application Bitgen from file '4020e.nph' in environment
d:/xilinx.

BITGEN: Xilinx Bitstream Generator M1.3.7
Copyright (c) 1995-1997 Xilinx, Inc. All rights reserved.

Running DRC.
DRC detected 0 errors and 0 warnings.
Saving ll file in "reg16.ll".
Creating bit map...
Saving bit stream in "reg16.bit".

xcpy reg16.bit C:\exemplar\work\reg16\reg16.bit

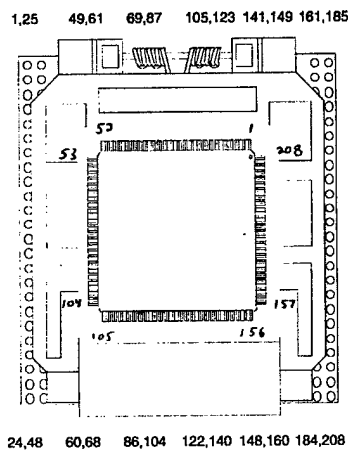xcpy reg16.ll C:\exemplar\work\reg16\reg16.ll

Appendix D: Ironwood Electronics Adapter to IMS and FPGA Pinouts
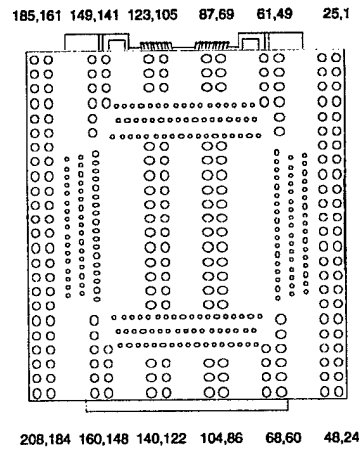
Ironwood Electronics, Inc.

PO Box 21151 • St Paul, MN 55121 • (612) 452-8100 • Fax (612) 452-8400

PA-QFE208SB2-C-Z-02(W) Map
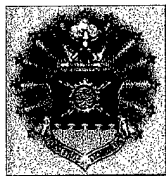**Rev B**

Top View   Bottom View

QFE 4020	Base Ims		QFE 4020	Base Ims		QFE 4020	Base Ims		QFE 4020	Base Ims
— 1	153		19	108 A_{12}		GND37	74		Vcc 55	4
GND2	144		20	107 B_{13}		38	71 A_1		M₂ 56	26
— 3	151		21	130 C_{13}		39	73 A_2		57	30
Clk 4	154		22	110 D_{13}		40	72 A_3		HDC 58	29
5	143		23	106 E_{13}		41	69 A_4		59	1
6	150		24	111 F_{13}		42	70 A_5		60	31
7	152		GND25	112		43	62 A_6		61	6
8	149		Vcc 26	105		44	49 A_7		LDC 62	25
TCk 9	141		27	87 G_{13}		45	61 A_8		63	32
10	142 A_{14}		28	94 H_{13}		46	52		64	7
11	123 B_{14}		29	88		47	63		65	27
12	124 C_{14}		30	93		M₁ 48	50		66	33
13	126 D_{14}		31	92		GND49	54		GND67	8
GND14	125		32	89		M0 50	64		68	3
15	127		33	76		— 51	51		69	34
16	128		34	90		— 52	53		70	9
17	129		35	75		— 53	2		71	28
18	109		36	91		— 54	5		72	35

— No Connect

Ironwood Electronics, Inc.

PO Box 21151 • St Paul, MN 55121 • (612) 452-8100 • Fax (612) 452-8400

PA-QFE208SA1-C-Z-01, C1788 Map
Rev B

QFE (4020)	Base (ImS)	QFE (4020)	Base (ImS)	QFE (4020)	Base (ImS)	QFE (4020)	Base (ImS)
73	10	— 107	58	141	135	175	181 B_{1S}
74	95	P.. 108	55	GND142	138	176	174 C_{1S}
75	36	109	66 J_1	143	136 N_5	177	199 D_{1S}
76	11	110	59 J_2	144	137 N_8	178	114 E_{1S}
$I...$77	77	111	57 J_3	145	140 N_6	179	173 F_{1S}
V_{cc}78	12	112	60 J_4	146	139 N_7	180	198 G_{1S}
GND79	13	113	68 K_1	147	147 N_1	181	132 H_{1S}
80	78	114	67 K_2	148	160 N_3	GND182	197
81	37	115	86 K_3	149	148 N_2	V_{cc}183	196
82	14	116	85 K_4	150	157 N_4	184	131
83	96	117	83	O.N151	146	185	172
84	38	118	84	152	159	186	195
85	15	GND119	82	CCLK153	155	187	113
86	45	120	81	U_{cc}154	145	188	171
87	39	121	80	— 155	158	189	194
88	16	122	100 m_1	— 156	156	190	164
89	22	123	101 m_2	— 157	207	191	170
GND90	40	124	102 m_3	— 158	204	192	193
91	17 B_1	125	79 m_4	159	205	193	187
92	46 B_2	126	99 m_5	GND160	183	GND194	169
93	41 B_3	127	103 m_6	161	179	195	192
94	18 B_4	128	98 m_7	162	180	196	163
95	48 B_5	129	97 m_8	163	208 P_1	197	168
96	42 B_6	V_{cc}130	104	164	178 P_2	198	191
97	19 B_7	GND131	122	165	203 P_3	199	161
98	24 B_8	132	115	166	184 P_4	200	167
99	43	RST133	121	167	177 P_5	201	190
100	44	134	116	168	202 P_6	202	185
GND101	47	135	117	169	182 P_7	203	166
— 102	20	136	120	170	176 P_8	204	165
Do..103	21	137	133	GND171	201	V_{cc}205	162
— 104	23	138	119	172	206	— 206	189
— 105	56	139	134	173	175	— 207	188
V_{cc}106	65	140	118	174	200 A_{1S}	— 208	186

2

Vita

Mr. Steven M. Parmley was born on ████████ in ████████.

He graduated from Miamisburg Sr. High in 1989 and started undergraduate studies at Wright State University. He earned a Bachelor of Science in Computer Engineering in December 1995. He earned a full scholarship from the Dayton Area Graduate Studies Institute and is a candidate for a Masters of Science in Electrical Engineering from the Air Force Institute of Technology, Wright Patterson Air Force Base.

His relevant experience started at Energy Innovations, Inc. with supervisory control and data acquisition systems. He then worked for the Southwestern Ohio Council for Higher Education performing real-time system development in the Robotics and Automation Applications Group of the Air Force Institute of Technology followed by a similar task for the American Society for Engineering Education. In May 1997, he became a DoD employee for the Hardware/Software Division of the Avionics Directorate of Wright Laboratories, Wright Patterson Air Force Base.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

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

4. TITLE AND SUBTITLE

FPGA Processor Implementation for the Forward Kinematics of the UMDH

5. FUNDING NUMBERS

6. AUTHOR(S)

Steven M. Parmley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
2750 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/97D-21

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFTA
Bldg 620 Suite 32
2241 Avionics Circle
WPAFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The focus of this research was on the implementation of a forward kinematic algorithm for the Utah MIT Dexterous Hand (UMDH). Specifically, the algorithm was synthesized from mathematical models onto a Field Programmable Gate Array (FPGA) processor. This approach is different from the classical, general-purpose microprocessor design where all robotic controller functions including forward kinematics are executed serially from a compiled programming language such as C. The compiled code and subsequent real-time operating system must be stored on some form of nonvolatile memory, typically magnetic media such as a fixed or hard disk drive, along with other computer hardware components to allow the user to load and execute the software. With a future goal of moving the controllers to a portable platform like a dexterous prosthetic hand for amputee patients, the application of such a hardware implementation is impossible.

Instead, this research explores a different implementation based on a modular approach of dedicated hardware controllers. The controller for the forward kinematics of the UMDH is used as a test case. The resulting FPGA processor replaces a robotic system's burden of repetitive and discrete software system calls with a stand-alone hardware interface that appears more like a single hardware function call. The robotic system is free to tackle other tasks while the FPGA processor is busy computing the results of the algorithm.

14. SUBJECT TERMS

UMDH, Utah MIT Dextrerous Hand, FPGA, Field Programmable Gate Array, Forward Kinematics, Synthesis, Xilinx

15. NUMBER OF PAGES

178

16. PRICE CODE

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