5-1999

Development of a TIM-compliant TMS320C6x DSP module

Jeffrey Claude Patterson

To the Graduate Council:

I am submitting herewith a thesis written by Jeffrey Claude Patterson entitled "Development of a TIM-compliant TMS320C6x DSP module." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Bruce Bomar, Major Professor

We have read this thesis and recommend its acceptance:

Roy S. Joseph, L. Montgomery Smith

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Jeffrey Claude Patterson, entitled "Development of a TIM-Compliant TMS320C6x DSP Module." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

[illegible]

Dr. Bruce Bomar, Major Professor

We have read this thesis
and recommend its acceptance:

Roy D Joseph

L Montgomery Smith

Accepted for the Council:

[illegible]

Associate Vice Chancellor and
Dean of the Graduate School

Development of a TIM-Compliant TMS320C6x DSP Module

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

Jeffrey Claude Patterson
May 1999

ACKNOWLEDGEMENTS

I must thank my family for their undying love and support which has been a constant in my life. I must also pay tribute to my father Joe Patterson, and Bruce Sherwood. I will strive to approach their example.

I gratefully acknowledge the encouragement and suggestions I received from Dr. Bruce Bomar, which were invaluable to the success of this project. I would also like to thank Dr. Roy Joseph and Dr. L. Montgomery Smith for their contributions as instructors, and committee members for my thesis.

This research was supported in part by Sverdrup Technology, Inc., under contracts A-96B-08-98-09 and A-96B-08-99-06 with The University of Tennessee Space Institute (UTSI).

Essayons!

ABSTRACT

The development of a Texas Instruments Module (TIM) compliant Digital Signal Processor (DSP) module, using the Texas Instruments TMS320C6201 (C6201) DSP, is presented. Currently, DSP modules based on the Texas Instruments TMS320C4x (C4x) family of DSPs are widely used for message passing multiprocessing DSP applications such as real-time processing of data and image processing. The interconnection of the TIM-compliant C4x DSP modules is accomplished using motherboards based on standard bus types, such as VME or PCI, and communication ports (comm ports) built into the C4x DSP. The purpose of the work described in this thesis was to provide a TIM-compliant DSP module with the improved computational performance of the C6x family of DSPs, which would also be compatible with the existing VME or PCI bus motherboards.

One drawback to using the C6201 DSPs in this application is the lack of C4x type communication ports (comm ports) in these new DSPs. In order for the C6201 TIM to be compatible with the existing motherboards, it must provide C4x- compatible comm port functionality. An FPGA was used to convert the C6x host port into multiple C4x-compatible communication ports and to provide the potential for future co-processing hardware.

The major effort of this development was the designing, building and testing of the C6x module hardware and the C4x-compatible comm port interface implemented in the FPGA. The first phase of this design involved the hardware architecture; this consisted of the selection of components needed to fulfill the design constraints, and the design of the module printed circuit board (PCB). The major components of this DSP module consist of the C6201 DSP, the external memory devices, and an Altera EPF10K100A Field Programmable Gate Array (FPGA). The memory devices include 4 MB of SDRAM, 256kB of SBSRAM, and a 512kB Flash ROM for storing boot code. The second phase of this design dealt with the host port to comm port conversion hardware implemented in the FPGA. The C6x host port was used to exchange data and control information with the FPGA. This hardware was developed in the VHDL hardware description language and graphic design files using Altera MAX+PLUS II software.

The C6201 DSP module has been built and tested. The board successfully executed both read and write transfers with another motherboard using the C4x-compatible communication port interface. The data exchange was across a 2.5' ribbon cable at an average read transfer data rate of 7.18 Mbytes/S and an average write transfer data rate of 5.15 Mbytes/S.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
Introduction

The Texas Instruments TMS320C6201 (C6201) and the Texas Instruments TMS320C6701 (C6701) represent the next generation of high performance digital signal processors (DSPs). The C6201 fixed-point DSP is capable of executing 1600 million instructions per second (MIPS) in peak performance. The C6701 floating-point DSP is capable of 1 billion floating-point operations per second (GFLOP) in peak performance. Fortunately, these two new DSPs are pin compatible, so based on the application, the same printed circuit board design can be used for either the C6701 or the less expensive fixed-point C6201. The C6201 fixed-point DSP is used in applications such as data processing, digital broadcast equipment, ADSL modems, and networking systems. The C6701 floating-point DSP is used in high-precision applications, such as medical imaging, beam-forming, speech recognition, 3-D graphics, and scientific visualization.

Earlier DSPs such as the Texas Instruments TMS320C4x (C4x) family of DSPs are capable of 60 MFLOP peak performance. This is far less than the 1 GFLOP performance of a C6701 and will no longer suffice for the more computationally intensive applications now being pursued. However, an important strength of the C4x DSP family is the built-in, bi-directional communication ports (comm ports). These comm ports allow multiple C4x DSPs to be readily interconnected in a wide range of multiprocessor topologies. Unfortunately, the C6201/C6701 processors do not include these comm ports and it is this shortcoming that was addressed by the work presented in this thesis.

The Texas Instruments Module (TIM) standard was developed, which defined a module specification for C4x DSPs. These modules, commonly called TIMs, are daughter boards that can be interconnected by means of TIM standard compliant mother boards using the C4x comm ports. TIM standard mother boards are based on several common bus types such as VME, PCI, or ISA. The modular approach of the TIM standard allows flexibility and scalability. One may interconnect different types of TIM modules in many ways. Additional TIM modules can be interconnected by using multiple mother boards. In this way, the type, number, and interconnection topology of TIMs can be readily scaled based on the application.

This thesis describes the development of a TIM compliant C6x DSP module. To accomplish this, the lack of comm ports in the C6201/C6701 was addressed by including comm port functionality in a field programmable gate array (FPGA). Initially, one C4x compatible comm port was implemented for functional and performance testing. The comm port interface was designed such that it is readily scalable to four comm ports. The simulation and testing data for one comm port is described in detail. Expansion to four comm ports per C6x TIM is also discussed. The presentation describes the design of the TIM memory hierarchy and the layout, building, testing, and evaluation of the TIM. The C6x TIM developed in this thesis provides computational performance far superior to the C4x TIMs while retaining the flexible and cost-effective

TIM interconnection architecture. C6x TIM modules make use of motherboards already in use.

1.1 Module Specifications

The C6x DSP TIM was developed to satisfy the following requirements:

1) The same printed circuit board (PCB) will be used for assembly of either a C6201 fixed-point DSP module or a C6701 floating-point DSP module;
2) Conforms to the TIM-40 physical specifications for a "single module" including board size, component heights, and primary mother board connectors;
3) One C4x compatible comm port will be provided and the design will be readily scaleable to four comm ports;
4) The FPGA will be of sufficient capacity to provide the logic needed for both the comm port interface, and future co-processing functions.

1.2 Thesis Outline

Chapter 2 provides background information on the C6x DSPs, the Altera FLEX 10K FPGA used for implementing comm ports and future co-processing functions, the TIM standard, and C4x compatible data communications. Chapter 3 describes the design in more detail. This description first covers the components selected, the PCB design phases, and finally, a detailed description of the C4x compatible comm port interface implemented in the FPGA.

In Chapter 4, various module tests are described and the results are summarized. Chapter 5 contains a discussion of the thesis conclusions, and suggestions for future work with respect to the C6x DSP and the C6x TIM.

Chapter 2

Background

2.1 Texas Instruments TMS320C6x Digital Signal Processors

Texas Instruments TMS320C6201 DSPs feature eight functional units and 32 general-purpose, 32-bit registers [1]. This DSP family is a load/store architecture type. This means that all instructions operate on data in registers, rather than data in memory. The CPU can supply, at most, all eight functional units with a 32-bit instruction every clock cycle. This is accomplished by fetching 256- bit wide, very long instruction words (VLIW). At a clock rate of 200MHz, the TMS320C6201 (C6201) DSP is capable of a peak execution rate of 1600 million instructions per second (MIPS).

The CPU is organized into two sides; each including four functional units and a register file of 16, 32-bit registers [2]. The four types of functional units (.L, .S, .M, and .D) are arranged as shown in Figure 2-1. The functional units on one side of the CPU have full use of the registers on the same side. Each side also has a bus connected to the registers on the opposite side. This allows up to one read and one write per clock cycle for functional units accessing the register file on the opposite side of the CPU.

Two of the functional units, .D1 and .D2, conduct all transfers between the register files and the memory.

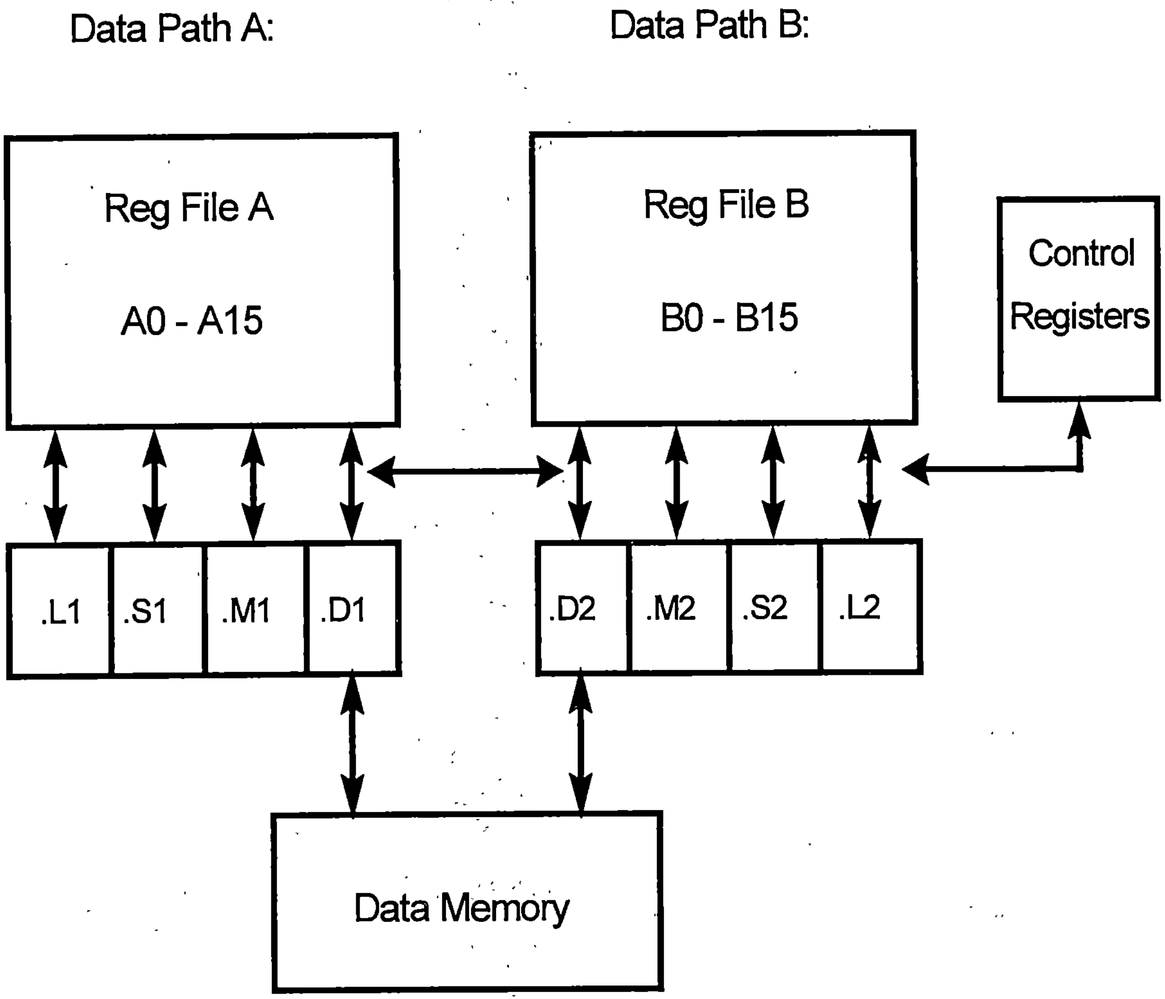


Figure 2-1. TMS320C6x - Arrangement of Functional Units [2]

Functional units .M1 and .M2 are solely multipliers. Arithmetic (add/subtract), logical, and branch instructions are executed by the .S and .L functional units.

TMS320C6701 (C6701) floating-point DSPs have the same overall architecture, and are pin-compatible with the C6201 fixed-point DSP. C6701 DSPs execute all of the C6201 fixed-point instructions, in addition to a superset of floating-point instructions [3]. These floating-point instructions are executed in the .L, .M, and .D functional units. C6701 DSPs operate at a maximum clock rate of 167 MHz. They can execute up to six floating-point operations per clock cycle and so are capable of a peak performance rate of 1 billion floating-point operations per second (1 GFLOPS) [3].

In order to understand how the C6201/C6701 DSPs (C6x) were incorporated into this design, an overview of the DSP peripherals is needed. The C6x DSPs have the following peripherals:

- Host port interface (HPI)
- External memory interface (EMIF)
- Multichannel buffered serial ports (McBSPs)
- Direct memory access (DMA) controller
- Interrupt selector
- 2 - 32-bit timers

All of the peripherals are controlled with various internal control registers [4]. These control registers can be accessed by the CPU, or through the HPI. Because they are central to the function of this design, the HPI and the EMIF will be described in detail in this section.

2.1.1 Host Port

The HPI is a 16-bit parallel port which allows an external host processor or controller (called the host) to access the DSP's memory by DMA transfer [4]. This access includes both the internal and external memory, as well as the DSP's memory-mapped peripherals. All exchanges across the 16-bit HPI data bus are two successive half words, where the word length is 32 bits. The host, but not the CPU, can access the host port address register (HPIA) and the host port data register (HPID). With these two registers, the HPI is connected to the CPU's memory space via DMA. A third register, the host port control register (HPIC), is accessible by both the host and the CPU. The host accesses these registers with various control signals, as described in Table 2-1. The nHINT signal is controlled by setting a certain bit in the HPIC register. This signal has no effect on data transfers. Use of the nHINT signal in this design is described later. The function of the HCNTL[1:0] bits is shown in Table 2-2. Host accesses start by setting the proper bits in the HPIC, followed by loading the HPIA. The address loaded into the HPIA determines the address at which the HPID access takes place. After that, reading data from, or writing data to the HPID can begin. Host reads or writes to the HPID can be conducted with an auto-incremented address in the HPIA. This allows rapid sequential accesses by the host.

All HPI bus accesses begin when the values of HCNTRL[1:0], HR/nW, and HHWIL are latched in upon the falling edge of nHCS.

Table 2-1. Host Port Interface Signals

Signal Name	Signal Type	Description
HD [15:0]	Input/Output	Data bus
HCNTL[1:0]	Input	Selects which internal register is accessed (HPIA, HPID, or HPIC)
HHWIL	Input	Indicates if the present half word is the first or second of a transfer
HR/nW	Input	Host Read/Write select
nHCS	Input	Data strobe
nHRDY	Output	Asynchronous ready status of the current access
nHINT	Output	Interrupt signal to the host device

Table 2-2. Functions of HCNTL[1-0] [4]

HCNTL[1]	HCNTL[0]	Description
0	0	Host reads or writes HPI control register (HPIC)
0	1	Host reads or writes HPI address register (HPIA)
1	0	Host reads or writes HPI data register (HPID), HPIA postincremented by a word address
1	1	Host reads or writes HPI data register (HPID), HPIA is not changed

HPI bus timing for a read access is shown in Figure 2-2. On the first half word access, HHWIL is latched in low on the falling edge of nHCS. During HPID accesses, following the first falling edge of nHCS, nHRDY goes high after a maximum delay of 12nS and remains high until the requested word has been fetched from DSP memory. nHRDY isn't affected by HPIC or HPIA accesses. For a read access, the data is transferred into the HPID from the DMA channel, and then latched in upon the falling edge of nHRDY and held valid until nHCS goes high. For a write access, the data placed on HD[15:0] by the host must be valid at the rising edge of nHCS. This process is repeated for the second half word access. During the second half word of a read access, it isn't necessary to wait on the nHRDY line to go low because the 32-bit word has already been fetched from DSP memory.

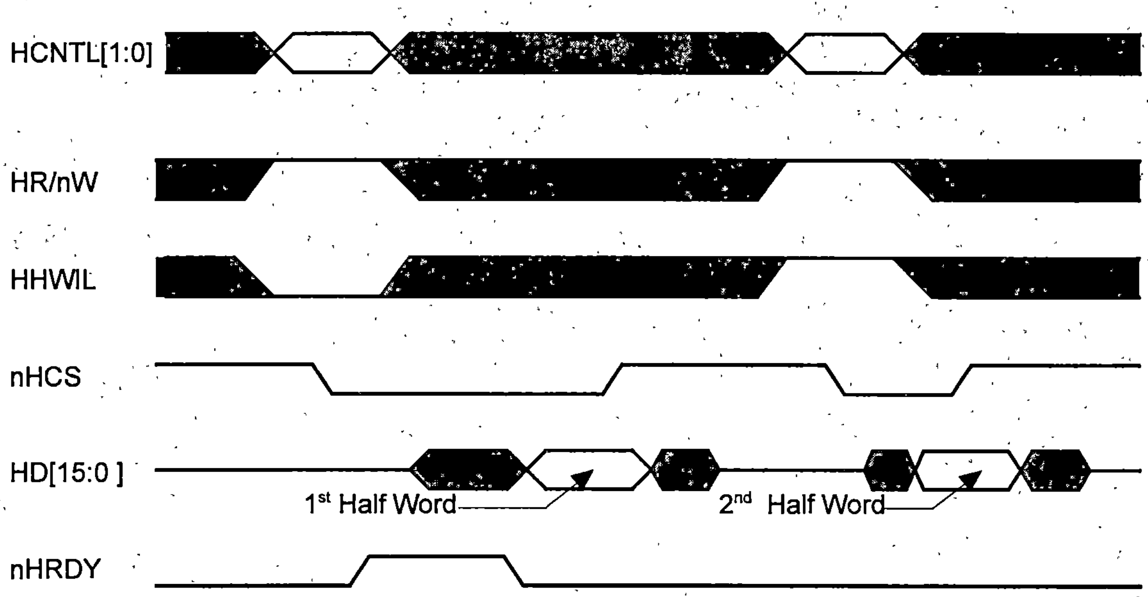


Figure 2-2. Host Port Interface Read Access Bus Timing

2.1.2 External Memory Interface

The external memory interface (EMIF) provides a direct interface to external devices such as synchronous burst SRAM (SBSRAM), synchronous DRAM (SDRAM), and asynchronous devices like ROMs, FIFOs, etc. Like the other peripherals, the EMIF is controlled by a set of memory-mapped registers [4]. These registers have to be loaded properly in order to use any external memory devices. There are three 16Mb memory blocks, and one 4MB memory block available in the C6x memory map for external memories. Each of these blocks are associated with one of four chip enable (CE) output pins.

One of the EMIF control registers, called the EMIF global control register, sets parameters for all the CE spaces. Some of these parameters include enabling the various clock lines, and determining the memory map mode for the DSP. There is also an EMIF CE space control register for each of the four CE spaces. The type of memory used at each respective CE is configured using these registers. Also, if an asynchronous memory type is selected, timing parameters are loaded into these registers. The other two EMIF registers are used to control SDRAM settings for any CE space where this type of memory is selected. The EMIF SDRAM control register is the first of these two registers. This is where the width of the memory device is selected, along with three common SDRAM timing parameters. The second of these two registers is the EMIF SDRAM timing register. This register stores the refresh period of the SDRAM.

2.2 Microprogrammed Controller

One method of controller design is the finite state machine approach. A finite state machine steps through states on successive clock edges in a pattern determined by the current state and the condition of the inputs. The next state is determined with combinational logic.

The microprogrammed approach to controller design is an alternative to the finite state machine approach. Microprogrammed design is based on the concept of considering the states of a controller as instructions (called microinstructions) that are executed on successive clock edges [5]. In this approach a set of microinstructions forms a microprogram, which is simply a set of bits in a memory array [6]. Microinstructions have bit fields that are the outputs of the controller, and other bit fields that are used to determine the next microinstruction (memory) address. A microprogrammed controller is organized as shown in Figure 2-3. The next address logic, called a sequencer, decodes the next address based on condition inputs and the next address bit fields in the microinstruction. Common microinstruction next address bit fields perform such functions as selecting which condition input to observe, the branch address, and the sequencer instruction. Sequencer instructions include operations such as: continue, branch on condition input equal to zero, branch on condition input equal to one, and unconditional branches.

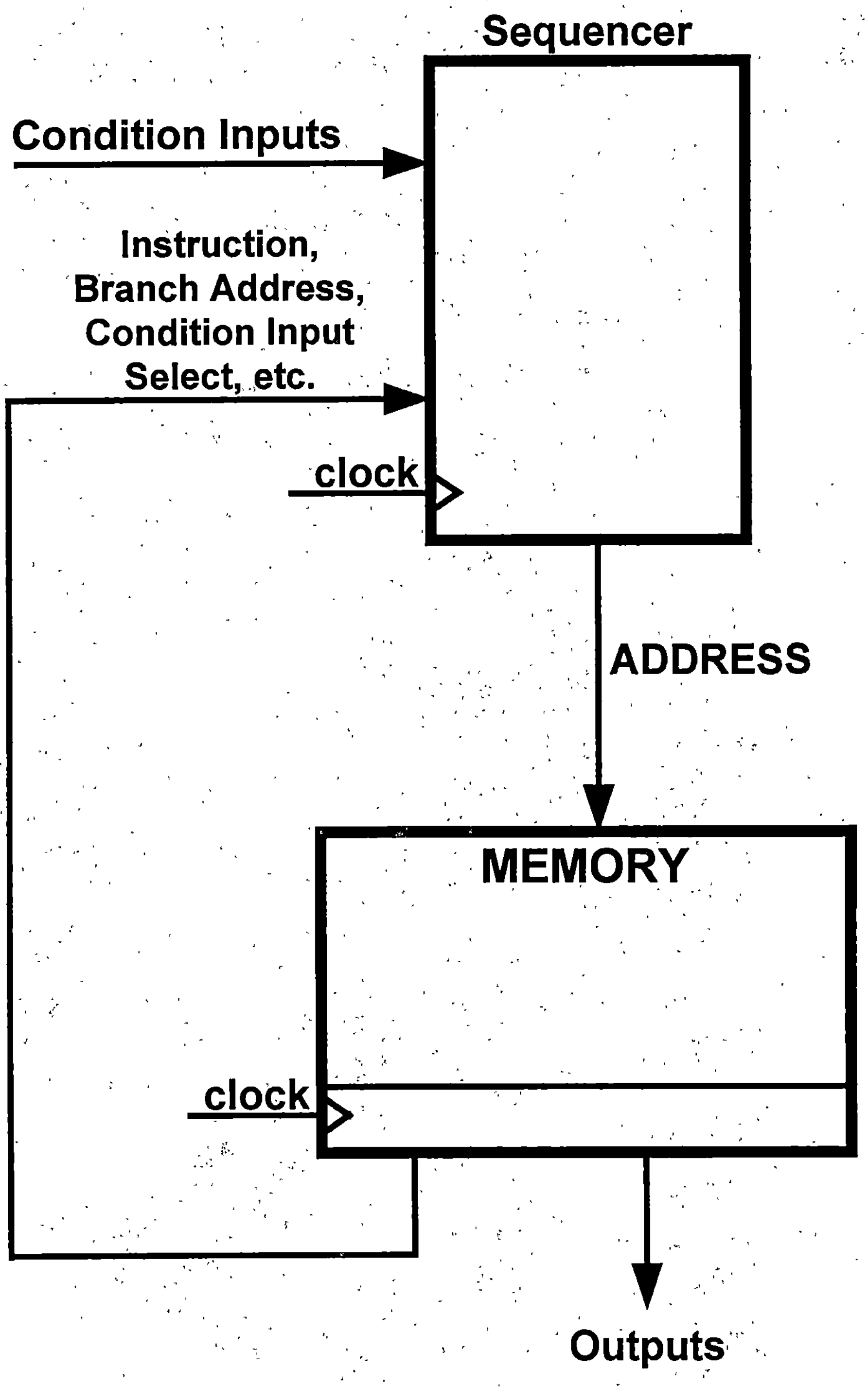


Figure 2-3. General Microprogrammed Controller

Notice that using the microprogrammed approach, memory is used where logic gates were used in the state machine approach. Therefore, control algorithm changes involve changing only the bit pattern in memory. This means that the timing characteristics of a given microprogrammed controller remain the same independent of the control algorithm (i.e. for any combination of microcode bits). This means that changes can be made to the microcode and the maximum clock rate of the controller will stay the same. However, in a state machine design, the output and next state logic delays vary with the combinational logic configuration. So changes in the control algorithm will affect the maximum clock rate. Also, unlike typical state machine designs, microprogrammed controller performance does not degrade with increasing numbers of states or output bits. In this design a microprogrammed controller was used as the host controller for the DSP HPI.

2.3 Altera FLEX10K Field Programmable Gate Array Family

The Altera FLEX10K FPGA family was used in this thesis. These reconfigurable devices are based on CMOS SRAM elements which are configured upon power-up [7]. Configuration can be conducted serially or with an 8-bit parallel data stream [8]. One common method of serial configuration is through the use of serial EPROMs. FLEX10K devices can also be configured serially or in parallel with a microprocessor using data stored in system RAM or ROM. Other methods of configuration are the Altera BitBlaster serial download cable or the ByteBlaster parallel port download cable connected to a PC serial port or parallel port.

Altera FLEX10K devices can also be programmed through their boundary-scan test (BST) circuitry with the BitBlaster serial download cable [9]. This is a dual use of the BST circuitry which is normally used for diagnostic purposes. The BST circuitry includes flip-flops at each input and output pin that can either force signals onto pins, or store data from the pin [10]. The stored data from the pins can be serially shifted out of the device and observed. The BST data may then be checked against expected values to ensure that the devices and their interconnections are working properly. The BST hardware in a FLEX10K device is compliant with the *IEEE Standard Test Access Port and Boundary-Scan Architecture* (IEEE Std1149.1-1990). The Joint Test Action Group (JTAG) developed the groundwork that led to this standard, so BST hardware is often referred to informally as JTAG hardware.

FLEX10K devices have both a logic array and an embedded memory array [7]. The Altera FLEX10K devices are available with densities of up to 250,000 usable gates. The logic array is used to implement general purpose logic functions. It is composed of logic array blocks (LABs) which contain about 96 usable logic gates [7]. Multiple LABs can be combined to implement larger logic functions. Each LAB consists primarily of 8 logic elements (LEs), and the LAB local interconnection network. An LE contains a four-input look-up table (LUT), a programmable flip-flop, connections to the LAB interconnect, connections to the chip-wide interconnect network, and other interconnects and control logic as shown in Figure 2-4. The flip-flop in each LE can be bypassed to implement purely combinational logic.

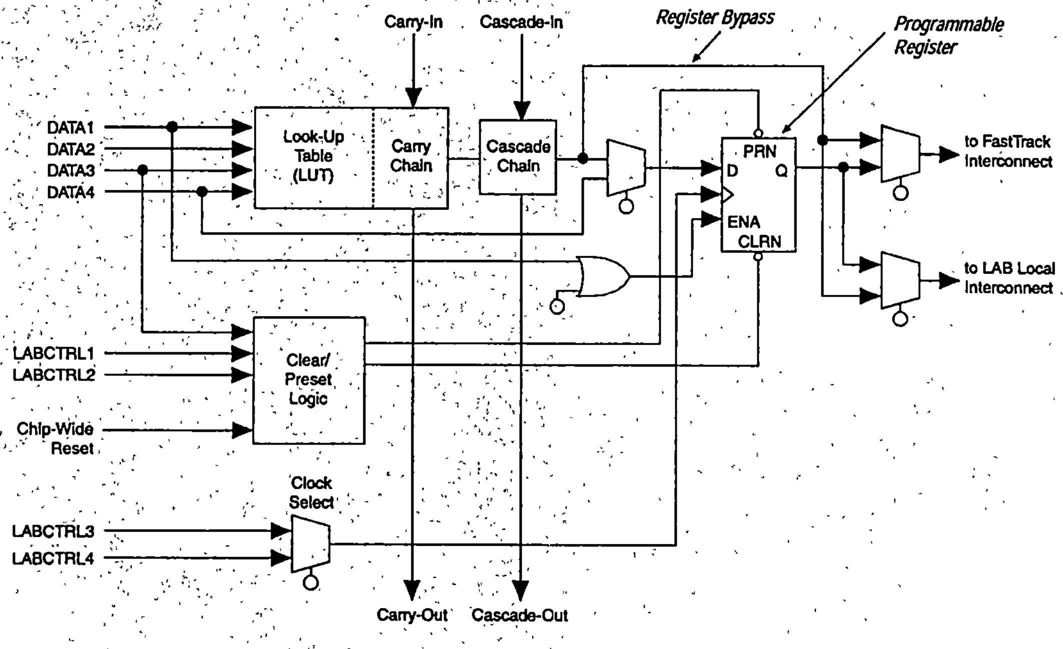


Figure 2-4. Altera FLEX10K Logic Element [7]

The embedded memory array is divided into embedded array blocks (EABs) of 2,048 bits each. There is one EAB for each row in the device. EABs can be used to implement RAM, ROM, or large LUT logic functions. EABs can also be combined to form larger memory blocks.

LABs and EABs are connected together with row and column interconnects as shown in Figure 2-5. The row and column interconnects are fast buses that run across the length and width of the device. At the end of each row and column interconnect, I/O elements connect the I/O pins to the row and column interconnect network.

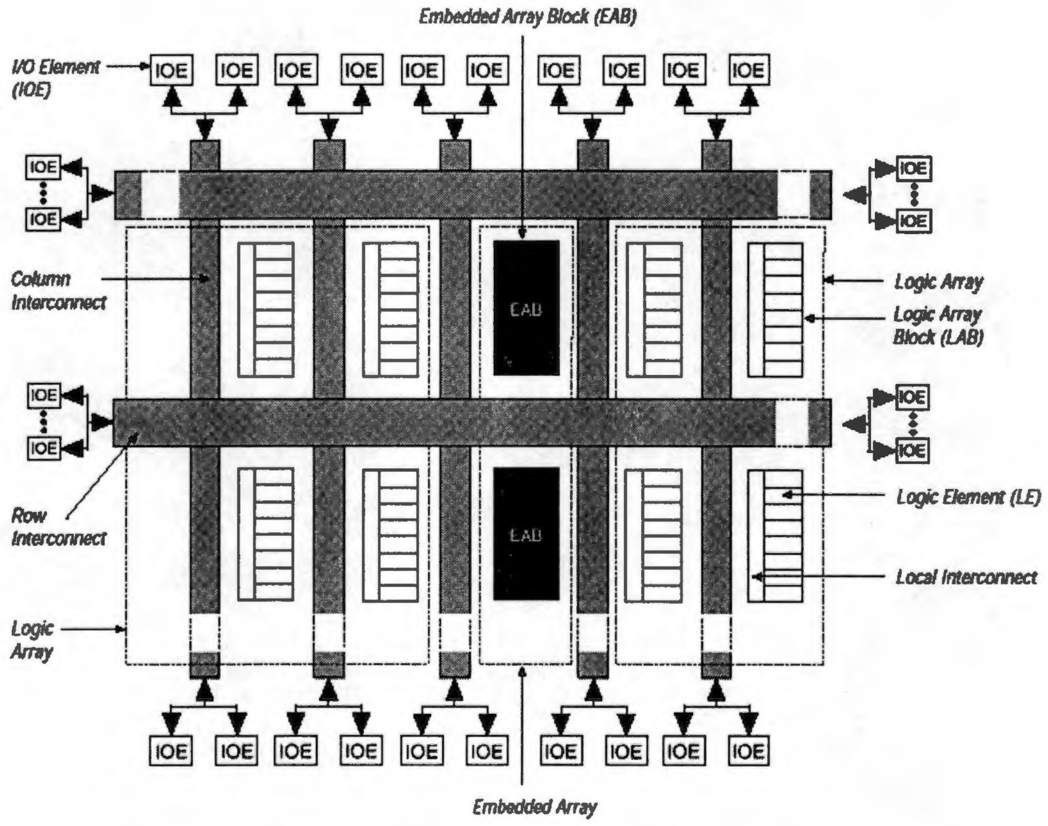


Figure 2-5. Altera FLEX10K Architecture [7]

2.4 Texas Instruments TMS320C4x - Compatible Data Communications

C4x DSPs have built-in comm ports which provide the means of communicating data between TIM daughter boards. These comm ports are bidirectional (one way at a time), byte wide, and support throughput rates up to 20 megabytes per second. A C4x comm port consists of eight data lines and four control lines. All twelve lines are bi-directional. Two of the lines are data handshake lines. The other two control lines are used to control the direction of the 8-bit data bus. A port sending data is said to "own the bus token" [11]. When the bus direction is changed, it is referred to as a "token transfer".

2.5 TIM - 40 Specification

The Texas Instruments Module (TIM) - 40 specification, often referred to as the TIM standard, is a module standard format originally based on the Texas Instruments TMS320C4x (C4x) DSP [12]. The purpose of the TIM standard was to provide a flexible and cost effective architecture for multiprocessor DSP systems. A TIM is a daughter board which meets certain physical and electrical specifications. This allows different modules to be arranged into a system which meets the requirements of a particular application. The TIM standard is an open specification. Though it was originally intended for the C4x processors, other processors or hardware can be used if the specifications for dimensions, pin-out, etc., are met.

The TIM standard physical specifications include: dimensions of the module, maximum component height on the top and bottom of the module, maximum component height on the motherboard under a module, etc. The physical specifications also include the type and placement of the connectors, and connector pin-out.

TIM daughter boards (TIMs) can be mounted on a wide range of mother board types. The TIM mother boards can be of any bus standard, as long as the module sites meet the TIM standard. Many types are available, for example: VME, PCI, PC/ISA, etc. A TIM is connected to a motherboard with two primary connectors. The connectors are Hirose brand, FX-4 series, 80-pin connectors that must meet the TIM standard regarding placement and pin-out. Each of the two primary connectors carry power (+5 V and ± 12 V DC), ground, control signals, and communication ports (comm ports). Shown in Figure 2-6, is a diagram of a typical TIM processor module.

The TIM standard provides a means for building multiprocessing systems that can be assembled to meet the needed performance standards for different applications in a cost effective way. The flexibility of this standard allows a system to be configured quickly with any number or type of modules and motherboards.

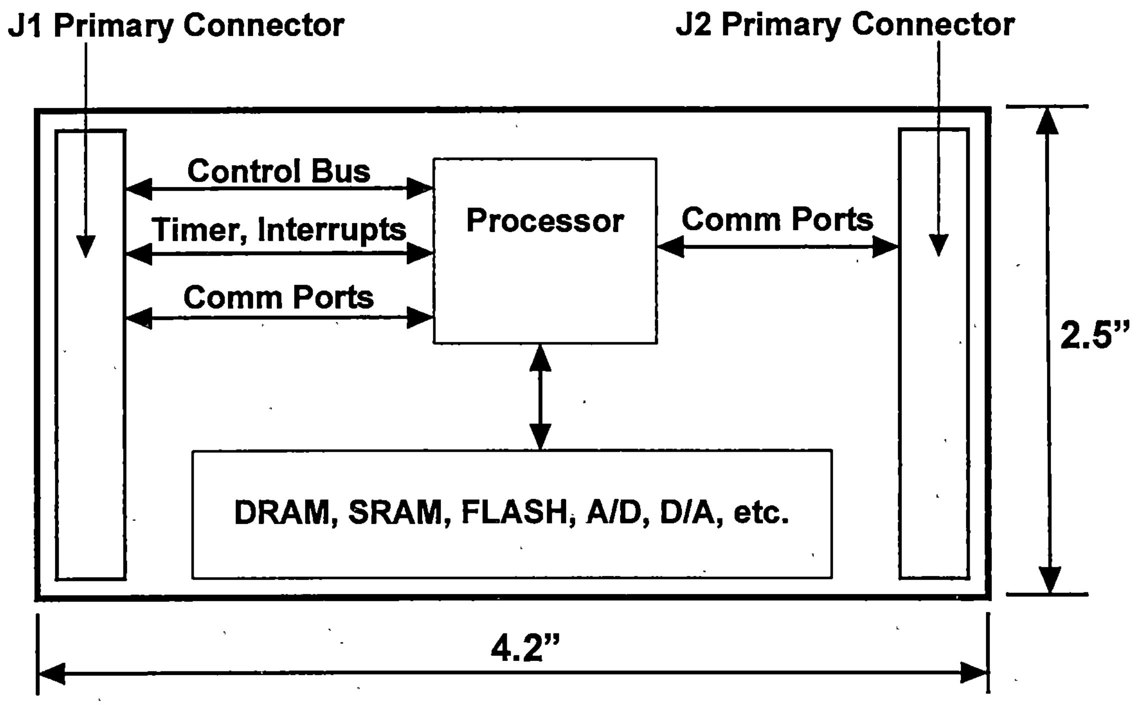


Figure 2-6. Diagram of a Typical TIM Processor Module

Chapter 3
Detailed Design Description

3.1 Major Component Selection

The first step in this design was selection of the major components. This design was built around the C6x DSP available in a 352-pin ball-grid array (BGA) package. Since the C6x DSP doesn't have the C4x-compatible comm ports needed for compatibility with the TIM standard motherboards, an interface was needed. This interface was implemented in a Field Programmable Gate Array (FPGA).

The FPGA was required to meet the following criterion:

- 3.3 Volt logic that can also tolerate 5 Volt inputs
- > 120 available I/O pins
- sufficient capacity of available logic to implement the C4x-compatible comm port interface and future coprocessing functions
- must fit on TIM module along with the other required components

In addition, the embedded memory architecture of the Altera FLEX10K devices is preferable for applications with look-up tables and microprogrammed controller implementations.

It was determined that the Altera FLEX 10KA device was the best choice for this design. This 3.3V part is available in three package types. One of those was much too large for this design. That left two packages that met the criteria, a 240-pin plastic quad flat pack (PQFP) and a 356-pin ball-grid array (BGA). The 356-pin BGA was selected for several reasons. First, the BGA package is not prone to lead damage like the PQFP. Second, the BGA has shorter leads than the PQFP, which means that the BGA has lower lead inductance as well. Lower lead inductance results in reducing ground bounce [13]. Third, the BGA has almost twice as many ground pins as the 240-pin PQFP, and this also reduces ground bounce by reducing the device's inductance to ground [13].

In order for this design to perform well, it was necessary to pay attention to the design of the processor's memory architecture. A typical hierarchy approach was used to determine what type of memories to select. This hierarchy consists of two elements: a fast but relatively small capacity static random access memory (SRAM) complemented by a slower but much larger dynamic random access memory (DRAM) [14]. Also, a nonvolatile memory was included to store the DSP's boot code and the configuration data for the FPGA.

A surface mounted, 512k X 8-bit, 3.3 V, flash programmable and erasable read only memory (PEROM) was selected as the nonvolatile memory. This device, commonly referred to as a flash memory, is in-system programmable at the 3.3V logic level provided by the DSP's EMIF. This device can be programmed, a 256 Byte sector at a time, by writing a specified control sequence to it, followed by the data to be loaded.

Next, a synchronous burst SRAM (SBSRAM) with the highest density and clock speed available was selected. This was a Micron 64k X 32 SBSRAM with a maximum clock rate of 100 MHz. The C6x EMIF supports SBSRAM operating at either half or full CPU clock speed [4]. Since the CPU is running at 160 MHz, the SBSRAM must operate at half the CPU clock rate.

SBSRAM has several performance advantages over conventional, asynchronous SRAM. Since the SBSRAM clocks all signals on the rising clock edge, the device can have data available on every clock cycle. Since the selected SBSRAM is pipelined, for the first access, there is a latency of two clock cycles after the address is clocked in [15]. For the next three data words, a word is available on every clock edge. This device has a clock to data time of only 5 nS, compared to a 15 nS access time for a high speed asynchronous SRAM [16]. 128k X 32 SBSRAM devices rated for a 166 MHz clock will be available by the second quarter of 1999. This design will accept these newer memories when they are available. This higher density device will also operate at the full CPU clock rate, thereby doubling its access rate.

For the module main memory, two 16M X 16-bit synchronous DRAMs (SDRAM), with a maximum clock speed of 100 MHz, were installed on each module. Again, the most dense and fastest clock speed devices commonly available were selected. The C6x EMIF also supports SDRAM and their high performance data burst functions. SDRAMs provide performance advantages over asynchronous DRAMs much like the comparison of SBSRAM with asynchronous SRAM. For example, a high performance early data out (EDO) DRAM may have an access time around 50 nS. So, if the EDO

DRAM was read in its page mode, it could provide the first data word in 50 nS, followed by words in the same page every 20 nS. The SDRAM device that was picked for this design has a read latency of three clock cycles for the first word accessed. After that, it can provide any data word in the same page every clock cycle. Therefore, if this SDRAM operates at 100 MHz, during a burst it provides the first data word in 30 nS, followed by a word in the same page every 10 nS.

Only a few components in this design operate at the 5 V supply voltage provided by the motherboard. All of the other devices operate at 3.3 V, with the exception of the C6201 DSP, which requires both 3.3 V and 2.5 V. To provide these two voltages, a fixed 3.3 V linear regulator, and an adjustable linear regulator were selected. The adjustable regulator is set at 2.5 V with two resistors.

After the major components were selected and it was determined that there was adequate space on the board for all of them, the architecture of the board was defined. The resulting C6x TIM architecture is shown in Figure 3-1.

3.2 Memory Map

External memories are assigned spaces in the memory map of the C6x DSP by connecting them to one of the four chip enable (CE) lines [4]. The 8-bit FLASH was

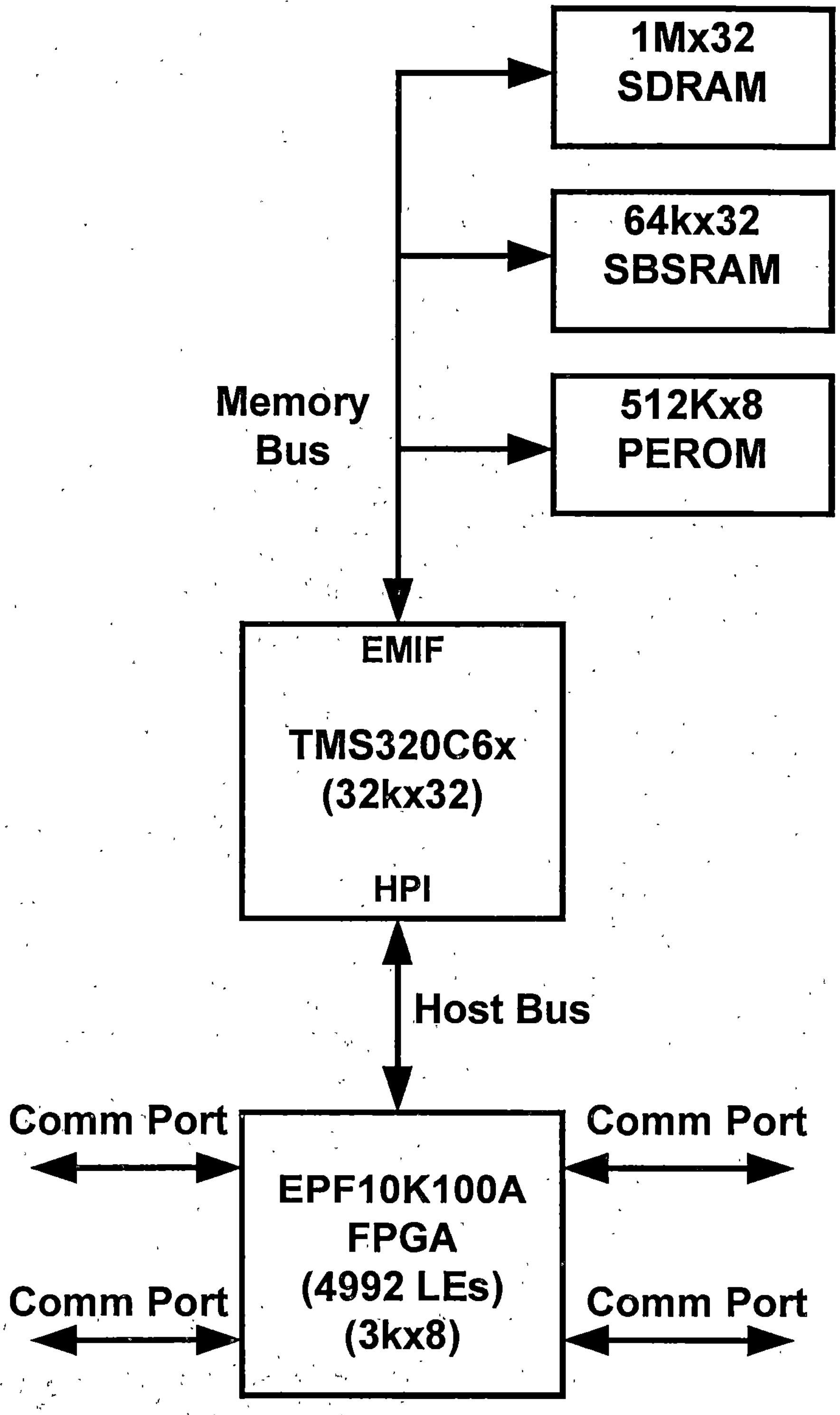


Figure 3-1. C6201/C6701 DSP Module

assigned CE1 because it is the only CE space that provides automatic packing and unpacking of 32-bit words. The SBSRAM was assigned to CE0 and the SDRAM was assigned to CE2. The resulting memory map is shown in Figure 3-2.

3.3 Printed Circuit Board Design

The C6x TIM printed circuit board (PCB) was designed using the HIWIRE II CAD software package. The design schematic and the PCB were drawn using this software. After the schematic was thoroughly checked, the schematic and the PCB layout were cross-checked using a HIWIRE II utility. Following that, the PCB drawings were sent to a contractor to fabricate the PCBs. When the bare boards arrived, they were sent, with all of the parts, to another contractor to assemble the parts onto the PCBs.

The completed PCB design was 10 layers thick. The layers consisted of five signal layers, three ground layers, a 3.3 V power layer, and a split 2.5 V/5 V layer. The PCB layer stack is shown in Figure 3-3. Due to the high density of this board, many signal traces had to run parallel to other traces on adjacent layers. In addition, many of these signals operate at high rates and short rise times (e.g. 80MHz clock lines to the SDRAM and SBSRAM). So, to control crosstalk between signal traces, the intervening ground layers were needed [13].

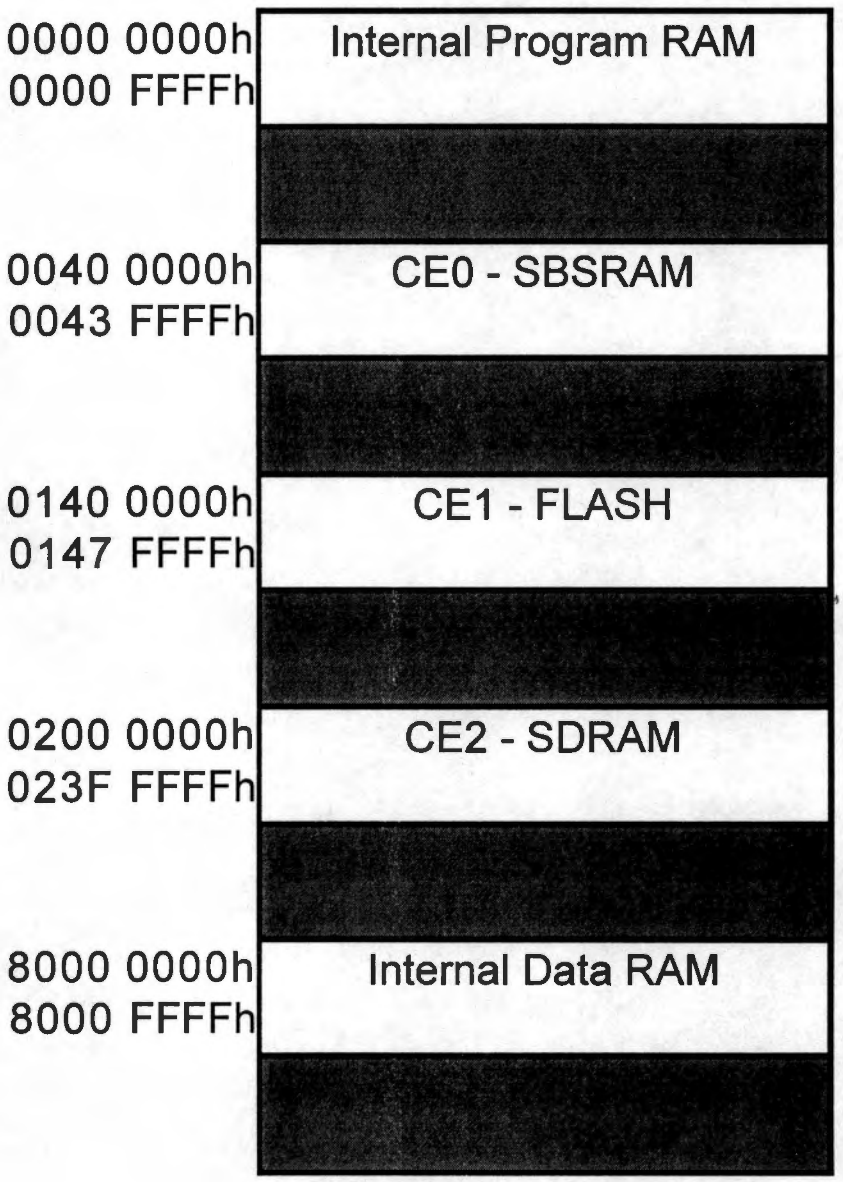


Figure 3-2. Memory Map Diagram

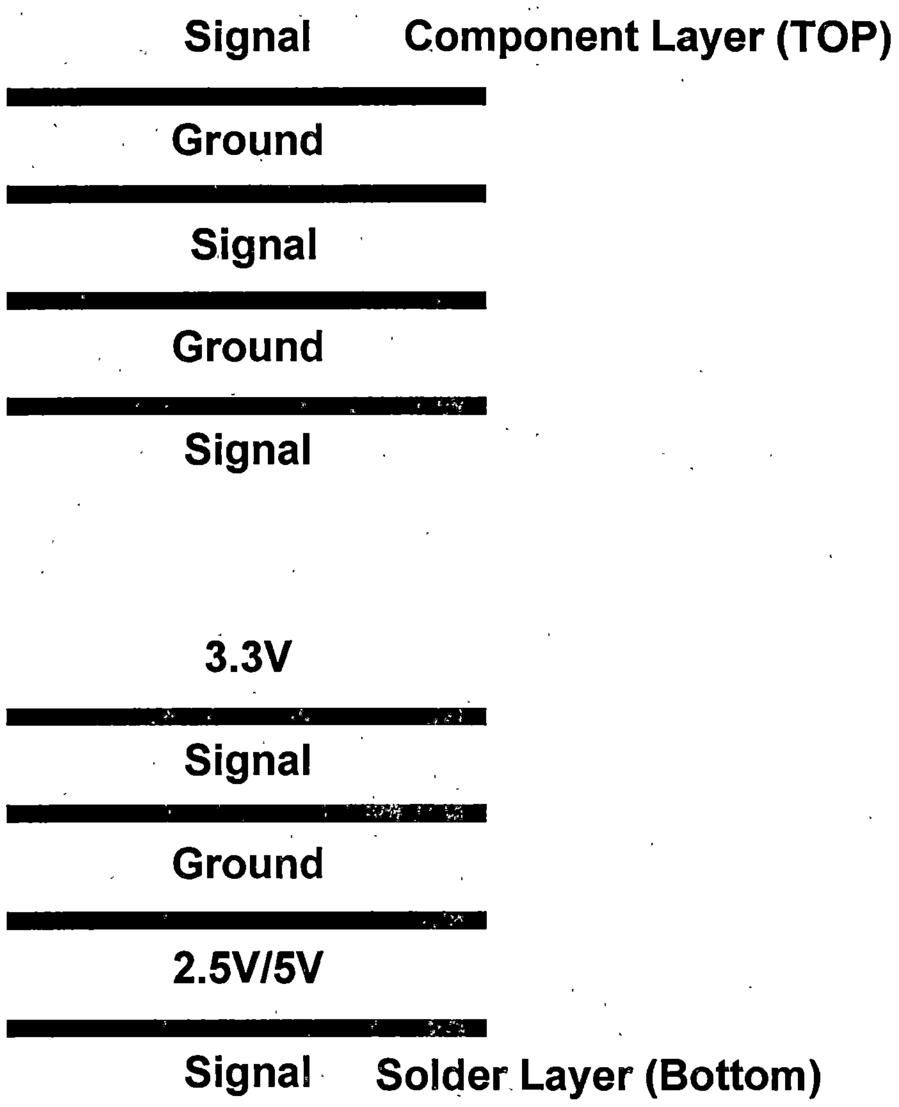


Figure 3-3. C6x DSP Module Printed Circuit Board Layer Stack

3.4 Configuration and Boot Modes

This section describes the configuration and booting procedures for the C6x TIM which provide the DSP with its initial code and the FPGA with its configuration data. Two DSP boot modes are supported in this design. These modes are the host boot mode and the flash boot mode. The boot mode is selected by placing the proper jumper before power-up. The location of the jumpers and other key features is shown in Figure 3-4.

During a host boot, the host device (the FPGA) can write to any location in the DSP's internal memory map [4]. Typically the various control registers would be configured and other code would be loaded into the DSP's internal memory at that time. While booting in this mode, the DSP is in a reset condition until a one is written into the DSPINT bit in the HPIC [4]. At that point, the DSP will exit from the reset condition, and begin executing code starting from address 0. This mode is used primarily during development work. This allows the FPGA to be configured directly and then it can load the DSP through the HPI. This initial FPGA configuration can be loaded via the FPGA JTAG interface.

Flash boot mode is the other method. In this mode, an automatic transfer of the first 16k, 32-bit words from the flash memory to the DSP internal program memory is conducted. This data is loaded into the DSP beginning at address 0. When this transfer is complete, the DSP exits from the reset state and begins executing code at address 0.

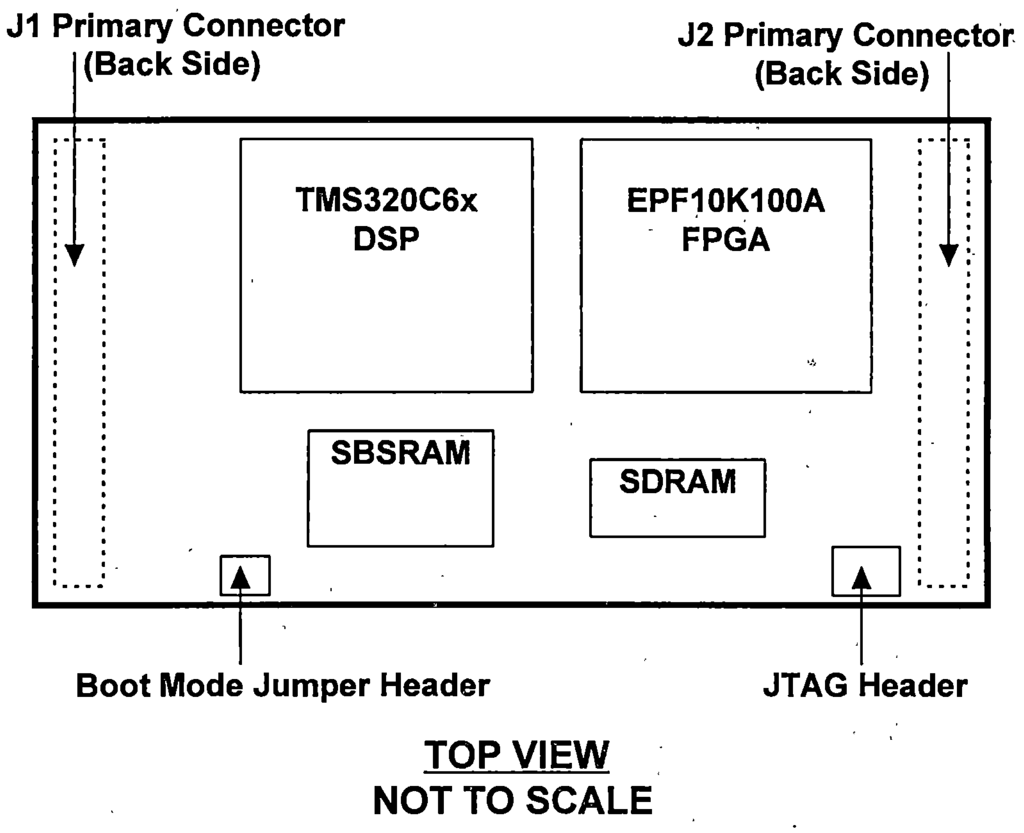


Figure 3-4. C6x DSP Module Layout

This code, in turn, reads data beyond the first 16k, 32-bit words in flash memory and uses that data to configure the FPGA. Once all development and testing of the module is complete, all of the DSP boot code and the FPGA configuration data can be programmed into the flash memory.

3.5 Texas Instruments TMS320C4x - Compatible Comm Port Interface

3.5.1 Overview

Since the C6x DSP has a single, 16-bit host port interface (HPI) instead of C4x-compatible comm ports, an interface was needed for compliance with the TIM specification. This interface is implemented in an Altera FLEX10K100A FPGA using VHDL and graphic design files. The interface architecture with four comm ports is shown in Figure 3-5.

The C6x TIM is designed to have sole control over its own address space on the module. Recall that any memory location in the DSP's memory map can be read or written via the HPI. Also note that the memory address to be accessed is first loaded into the HPIA by the host device (the FPGA) through the HPI. Therefore, a means of setting up read and write transfers with the HPI is needed so that the DSP can maintain control of its memory. To that end, the DSP first allocates a block of memory, either for a write transfer or a read transfer to one of the comm ports. This block of memory is

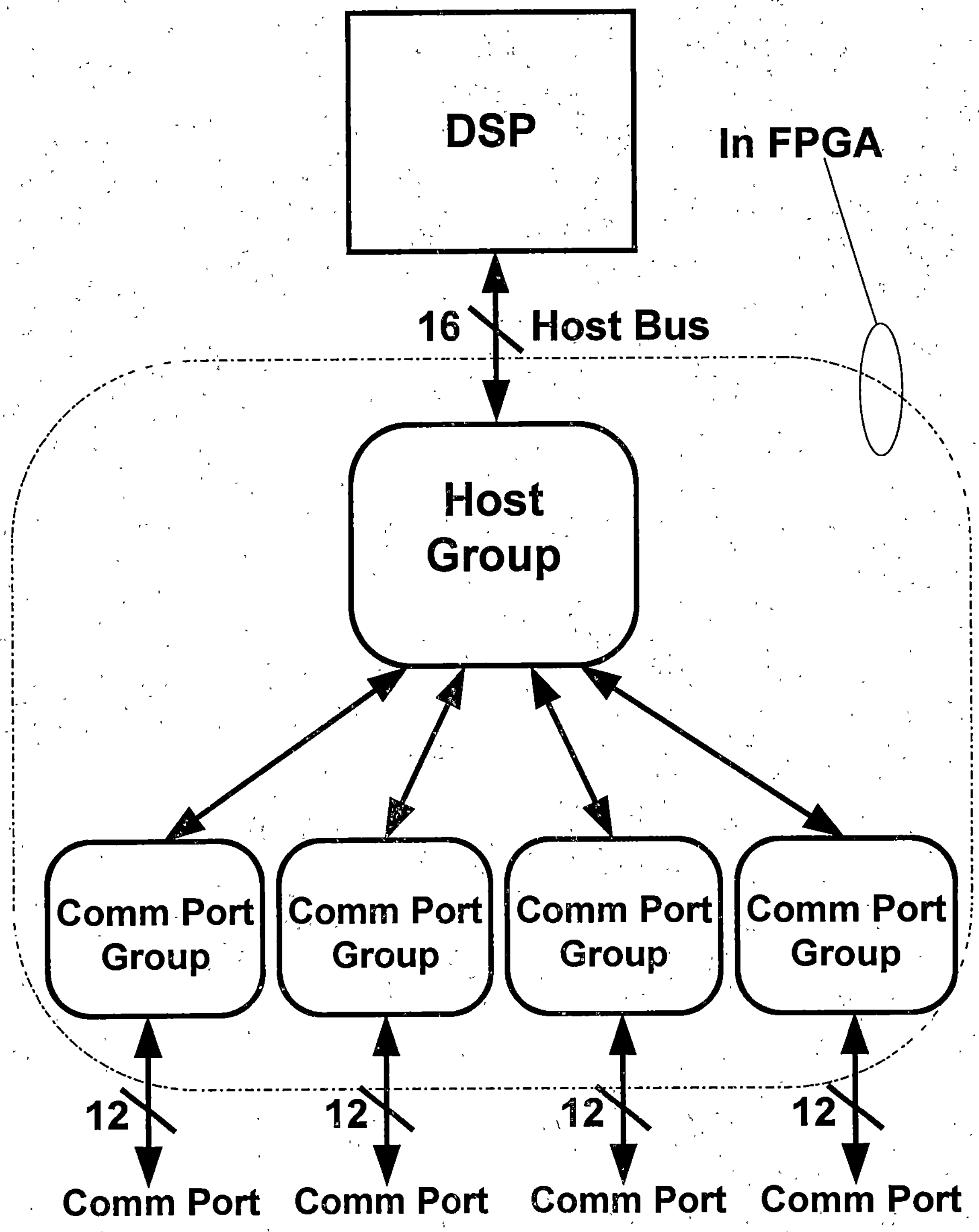


Figure 3-5. C4x Comm Port Interface With Four Comm Ports

defined by a starting address and its length in 32-bit words. Once the memory is allocated, the transfer is enabled.

In order to assign a block of memory to a comm port, the starting address and word count values are transferred to a set of counters in the FPGA. These counters are used to keep track of the status of the assigned memory blocks. Each of the four comm ports has a set of counters as shown in Figure 3-6. So, each comm port has a pair of counters used to set up input (write) transfers, and another pair of counters to set up output (read) transfers. The address counters keep up with the current address in the assigned memory block, and the word counters monitor the number of words that have not been accessed in the assigned memory block. After each word access through the HPI, the address counter is incremented by a word address (four), and the word counter is decremented by one. If a word counter has not decremented to zero, the comm port is still enabled for transfers. When a word counter decrements to zero, its respective done flag flip-flop is set, indicating that the pair of counters are no longer set up for transfers.

In order for a setup to occur, control signals are needed between the FPGA and the DSP. One of these signals is the nHINT line, which is used as a flag to the FPGA indicating that the DSP needs to set up one of the comm ports for a read or write transfer. After the nHINT line goes low, the host group begins the process of setting up one of the comm ports for a transfer. The first information that the host group needs is which comm port to set up, and if the setup is for a read or a write transfer. This information is retrieved from a reserved DSP memory location via the HPI. This word,

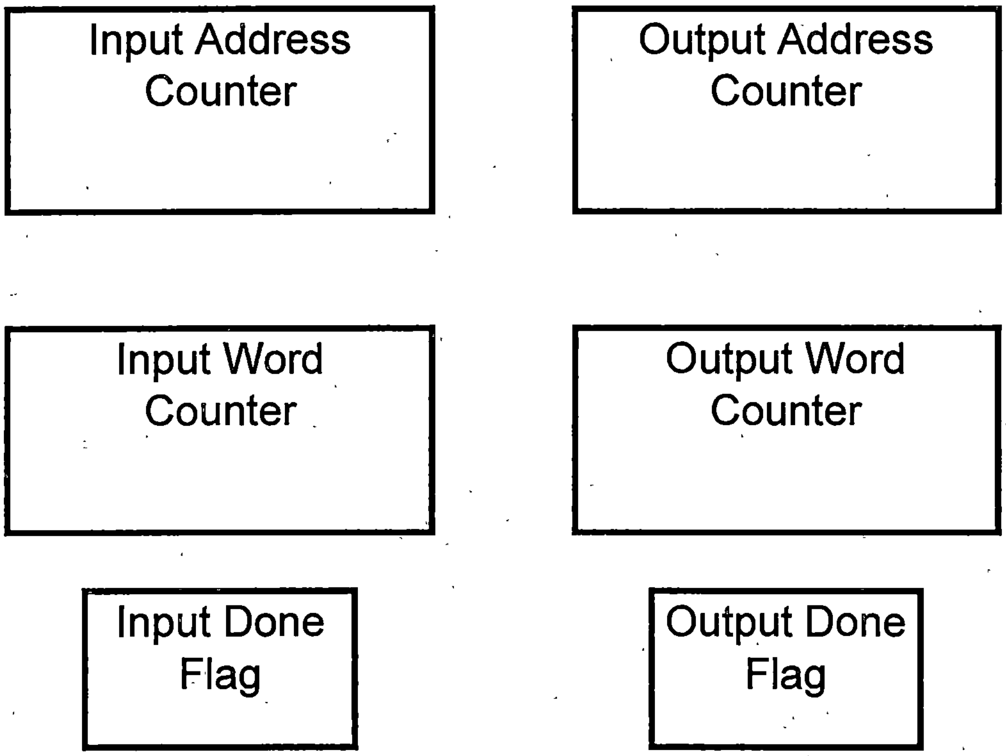


Figure 3-6. Comm Port Counter Diagram

along with the word counter value and the address counter value are located in three consecutive DSP memory locations, as shown in Figure 3-7. The other control signal between the DSP and the FPGA is the EXT_INT_4 line, which is one of the DSP's interrupt inputs. This signal is asserted by the host group to notify the DSP that the condition of one of the comm port done flags has changed (i.e. one of the flags went high, indicating that a pair of counters are no longer set up for a transfer). Before the EXT_INT_4 signal is asserted by the host group, all of the comm port done flag bits are written to a reserved memory location in the DSP, as shown in Figure 3-7. When the DSP receives the interrupt, it checks the status of the done flags at the memory location. At that point the DSP can determine if any of the comm ports need to be set up again.

For the remainder of this chapter, a C4x-compatible comm port interface with only one comm port is considered. The extension to multiple comm ports is straightforward and is discussed later. However, to simplify the checkout and debugging, only one comm port was actually implemented and tested for this thesis.

The comm port group handles the data communication over an 8-bit, bi-directional C4x comm port. This comm port group is a macro function previously designed by others. The comm port group has a 32-bit data input register, and a 32-bit data output register. To control data exchanges between the comm port group and the host group, there are four control lines(LOADACK, LOAD, DAV, and DATACK), as shown in Figure 3-8.

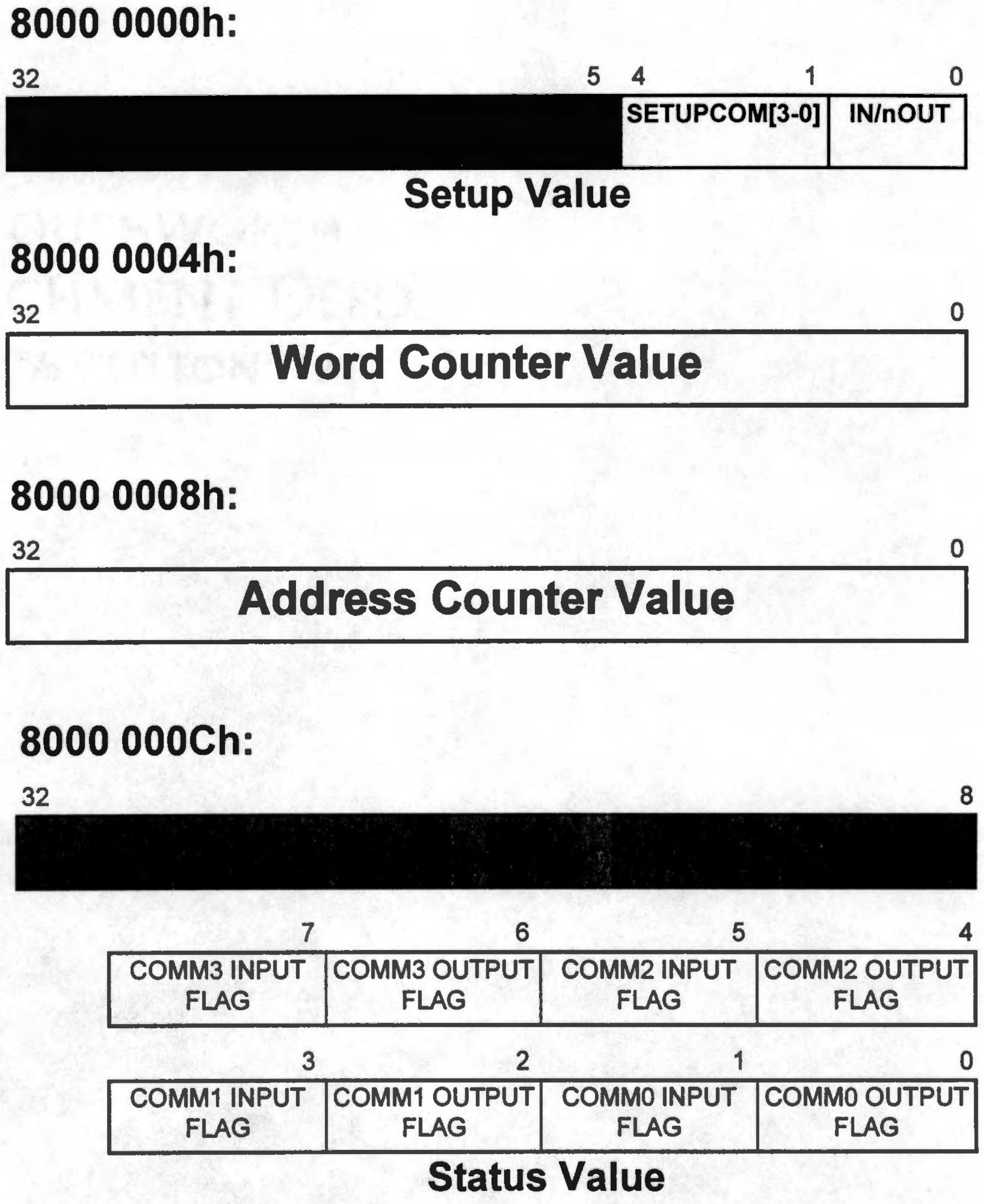


Figure 3-7. Control Word Memory Locations and Contents

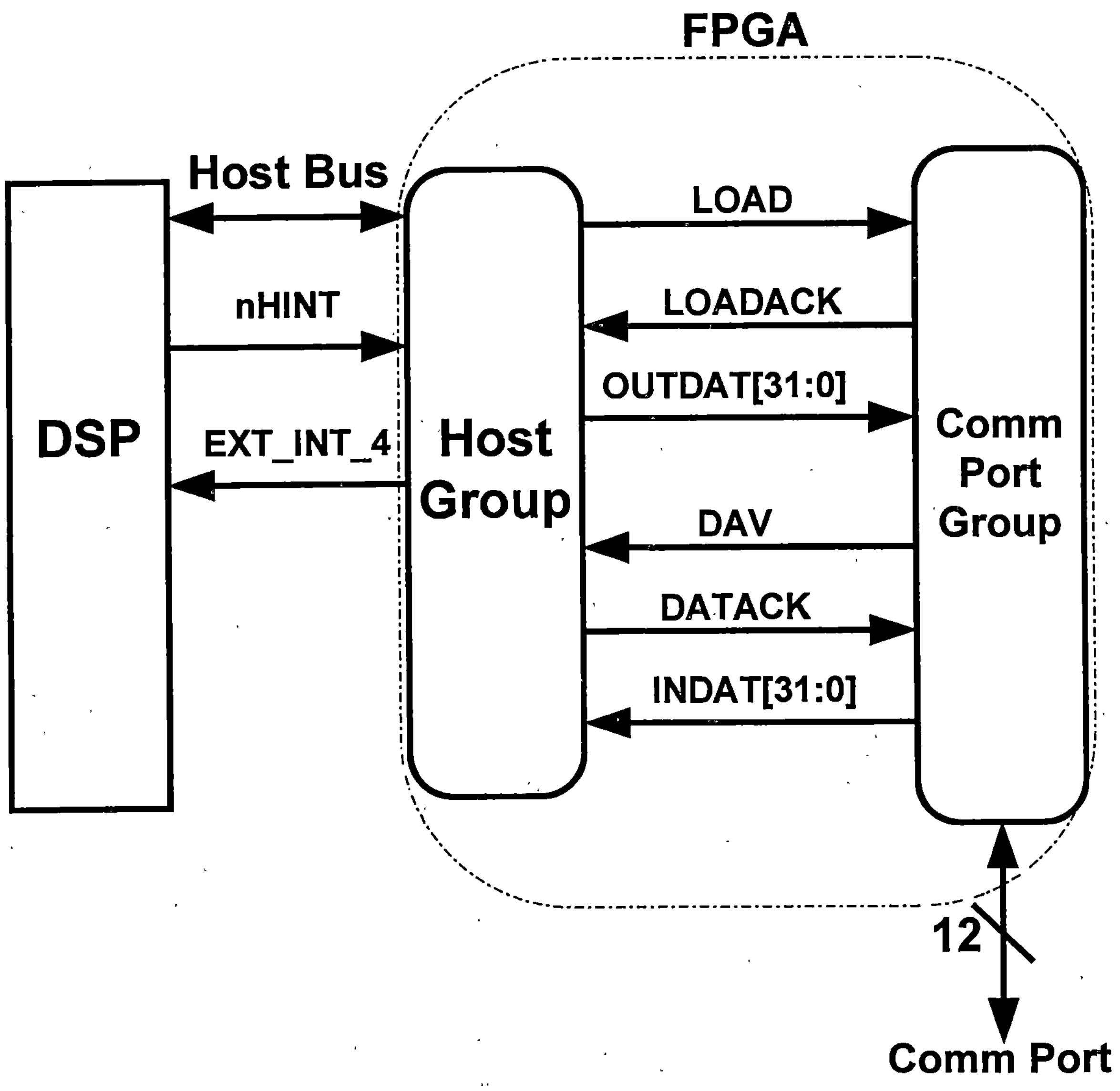


Figure 3-8. Control Signal Diagram

For output transfers (to the comm port group through OUTDAT[31:0]) the LOAD and LOADACK lines are used. When the LOADACK signal is high, the output register in the comm port group is empty, and data can be loaded. This means that a data word can be placed on the OUTDAT[31:0] lines by the host group and the LOAD line can be strobed to load the data word into the 32-bit output register.

For an input transfer, the comm port group loads the four bytes it has received via the comm port into its 32-bit input register. Once a valid data word is on the INDATA[31:0] lines, the DAV (data available) line will go high. When DAV is high, the host group can receive the data word. After the host group has latched in this data word, it will strobe the DATACK line. This allows the comm port group to proceed with loading another word into the data input register.

3.5.2 Input Transfers

Before an input transfer can start, the comm port must be set up. First, the DSP places the appropriate data in the control word memory locations, as shown in Figure 3-7. This data includes: which comm port to set up, that the transfer direction being set up is input, the word counter value, and the address counter value. After the control words are in place, the DSP asserts the nHINT line. The host group will retrieve these three control words through the HPI, followed by loading the input word counter and the input address counter. At that point, the comm port is set up for an input transfer. The input transfer starts when the comm port group receives data, places it on the INDAT[31:0] lines, and sets the DAV line high. When the host group detects the DAV line is high and the comm port group is set up for an input transfer, the host group will latch in the data word from INDAT[31:0]. Before the first word is written to the DSP, the current address in the input address counter is written into the HPIA register. Then, the first data word is written into the HPID register. If data keeps arriving at the comm port, and the comm port is still set up, the input transfers will continue, repeating this process. As long as no other HPI address is accessed, the HPI is used in its autoincrement mode so the address does not have to be loaded for every transfer. Once the input word counter has reached zero, the input transfer is stopped, and the controller will write the done flag bits to the DSP memory at address 8000 000Ch, as shown in Figure 3-7. After that, the controller will strobe the EXT_INT_4 pin, which alerts the DSP that the comm port setup status has changed.

3.5.3 Output Transfers

The comm port setup procedure for an output transfer is the same as the input transfer except for the transfer direction. Once the comm port is set up for an output transfer, and the LOADACK line is high (indicating that the comm port is ready to accept a 32-bit word) the output can start. If this is the first word of this output transfer, the address in the output address register is written to the HPIA register. Then the data can be read out of the HPID starting at that address.

Next, the data word is placed on OUTDAT[31:0] by the host group. Then the host group strobes the LOAD line high, loading the word into the 32-bit register in the comm port group. LOADACK will go low, then high again when the comm port is ready for another word. Transfers can continue as long as the data arrives and the comm port is set up. Once the output word counter reaches zero, the transfer stops, and the controller writes the done flag bits to the control word location in DSP memory at address 8000 000Ch. Then EXT_INT_4 is strobed high, and the DSP checks the done flag bits in memory.

3.5.4 Bi-directional Transfers

The previous two sections described only one-way data transfers. Realistically, a comm port could be set up for both input and output transfers at the same time. Since the host bus can only go in one direction at a time, it must be exchanged between input and output transfers. In that case, the controller must ensure fairness with the access to the host bus to prevent a deadlock.

To implement the fairness scheme, a loop procedure was used, called the Idle loop, and is shown in Figure 3-9. Note that both the read and write transfers are single word operations. That is to say, these transfers only read or write one word before branching back to the Idle loop. This is done to provide equal sharing of the host port between transfer directions. Also, notice that each type of operation re-enters the Idle loop in a different place. This is how the fairness scheme is implemented in the controller.

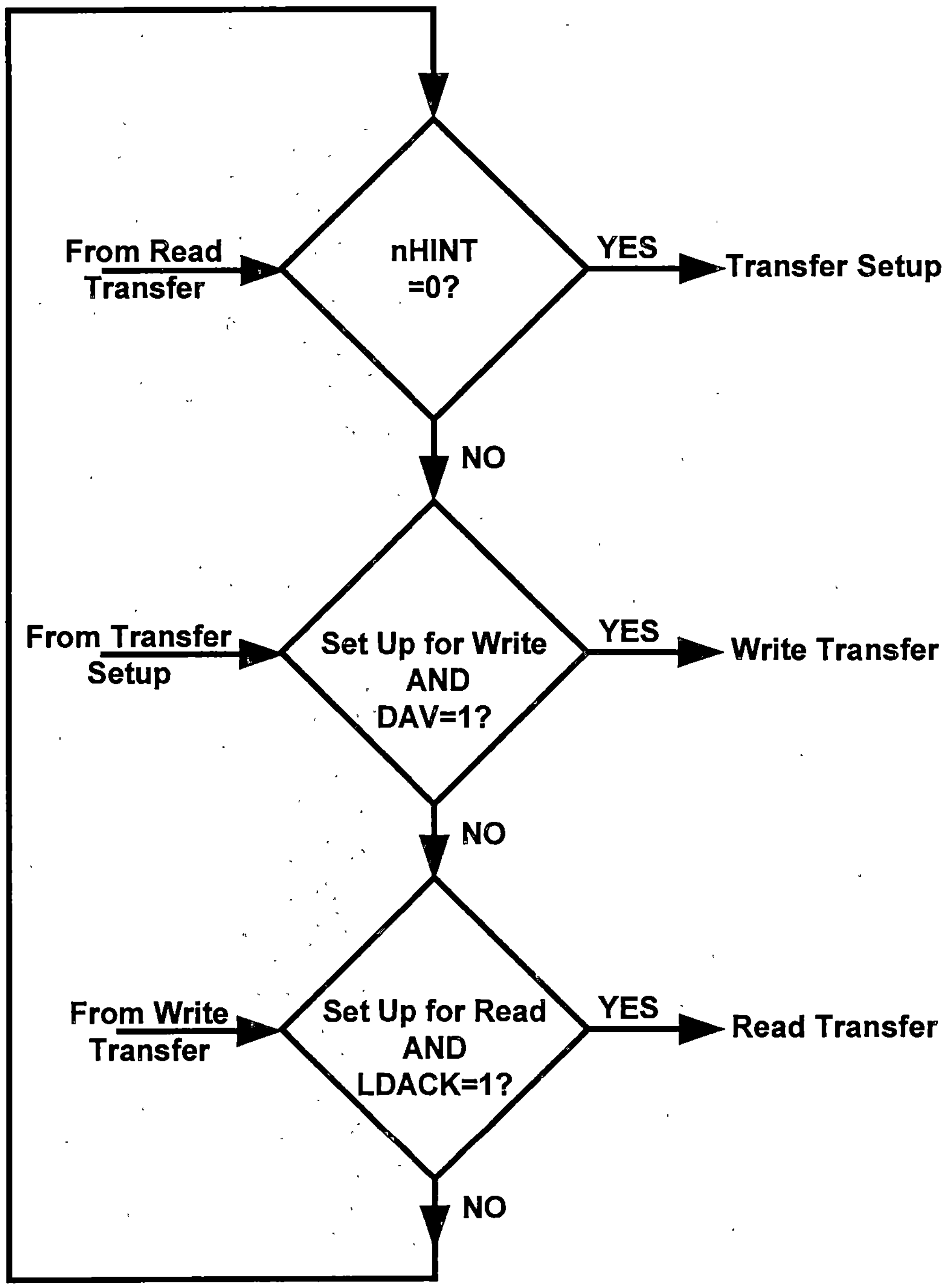


Figure 3-9. Idle Loop Flowchart

3.5.5 Implementation of the Host Group Hardware

The hardware implemented in the FPGA is organized into groups as shown in Figure 3-10. Notice that the "host group" considered in the previous sections has been split into two separate groups called the host port group, and the controller group. These groups are separated in this discussion to separate the data path hardware from the controller hardware.

The host port group consists of the hardware that forms the data path between the DSP and the comm port group. The host port group consists of two data paths, one for incoming data and one for data going out of the DSP. The host port group hardware is shown in Figure 3-11. The outgoing data path is the 32-bit register, called hiloreg. This register's 16-bit input is connected to the HPI data bus (HD[15-0]). Its 32-bit output is connected to the comm port group's 32-bit register, and the two word counters and the two address counters. This register loads the data on the HD[15-0] lines into its upper 16 bits when LDOUTHI is asserted. Likewise, it loads its lower 16 bits when LDOUTLO is asserted. This register essentially reassembles the 32-bit data word from two consecutive 16-bit reads of the HPI data bus.

The incoming data path, also shown in Figure 3-11, consists of a 32-bit register (called inreg), followed by three multiplexers. The 32-bit register latches in an incoming data word from the comm port group. The first multiplexer selects either the output of the register, or the output of one of the two address counters. The second

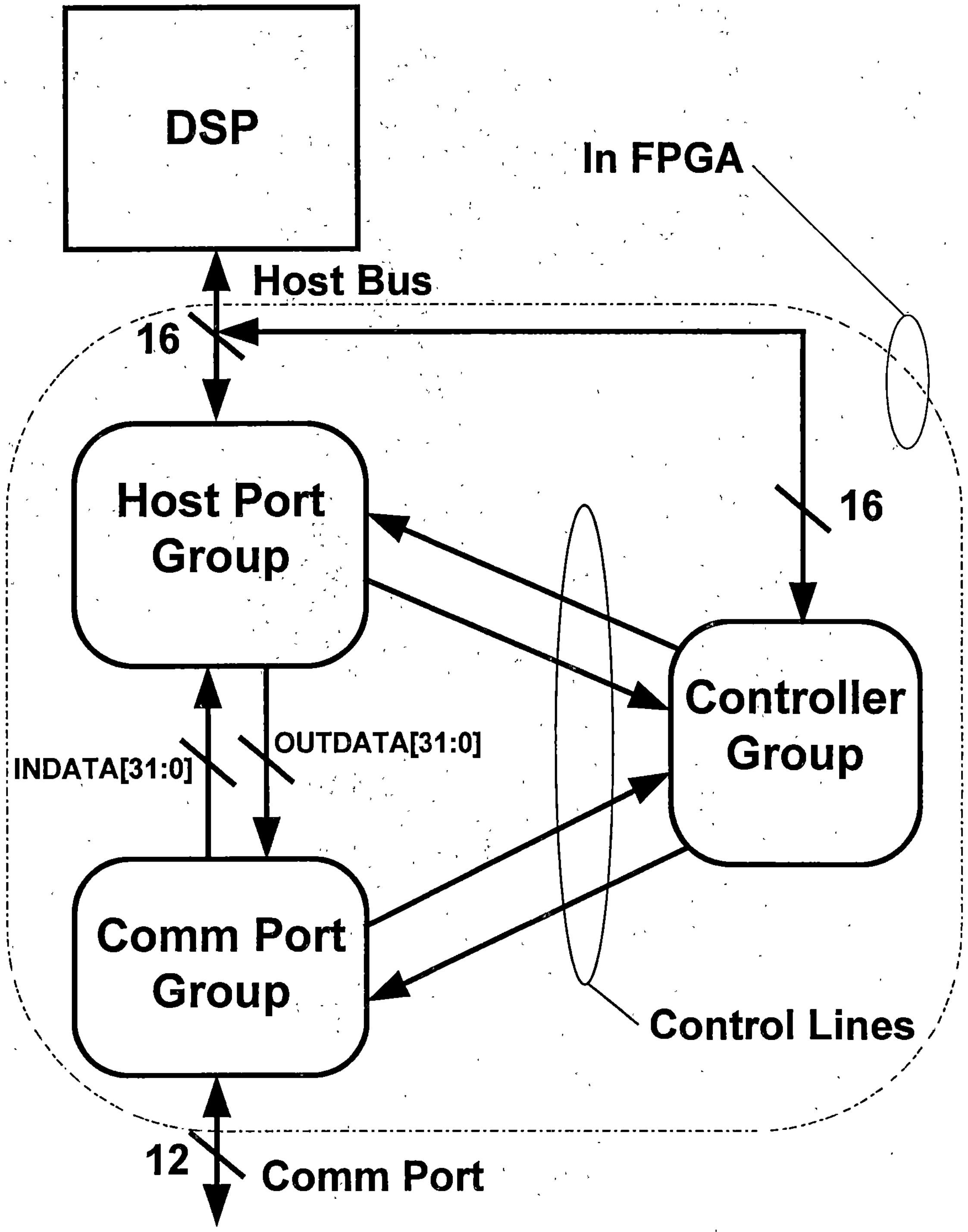


Figure 3-10. C4x-Compatible Comm Port Interface Organization

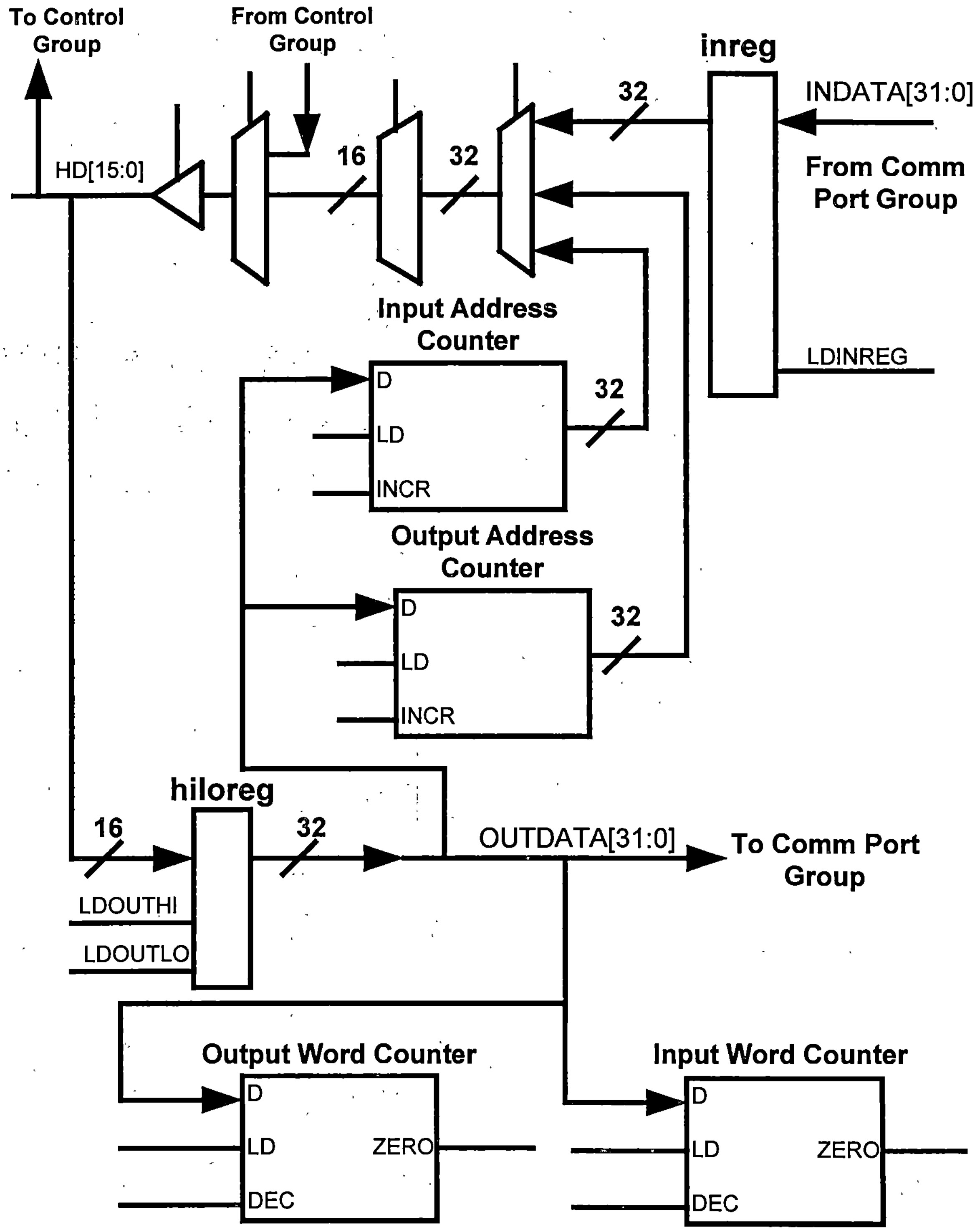


Figure 3-11. Host Port Group Block Diagram

multiplexer simply selects which half word is seen on the HPI data bus. The last multiplexer selects which group (either the host port group or the control group) is connected to the host data bus.

The central component of the hardware is the controller. The controller group consists of a microprogrammed controller, a 5-bit register, and a multiplexer. The controller group hardware is shown in Figure 3-12. The five bit register (called reg5) is used to store control bits retrieved from DSP memory at address 8000 0000h, as shown in Figure 3-7. The multiplexer selects between a 16-bit field from the microprogrammed controller, and the comm port status bits. The 16-bit field from the microprogrammed controller is used to generate the addresses for retrieving the three control words from DSP memory. It also generates the address for writing the comm port status bits into DSP memory at the reserved address.

The core hardware of the controller group is the microprogrammed controller, shown in Figure 3-13. In this design the memory array is 64 bits wide. 16 of these bits are for control of the next address hardware, while 48 bits are output bits for controlling the rest of the interface hardware and the DSP. The microprogram is stored in embedded array blocks in the FPGA. The memory array is implemented in an LPM_ROM in the FPGA. This is a macro function from the Altera MAX+PLUSII software that interconnects the embedded memory blocks in the FPGA to the desired configuration. The microinstructions have bit fields that determine the instruction to execute, condition input select, and the branch address. Four instructions are

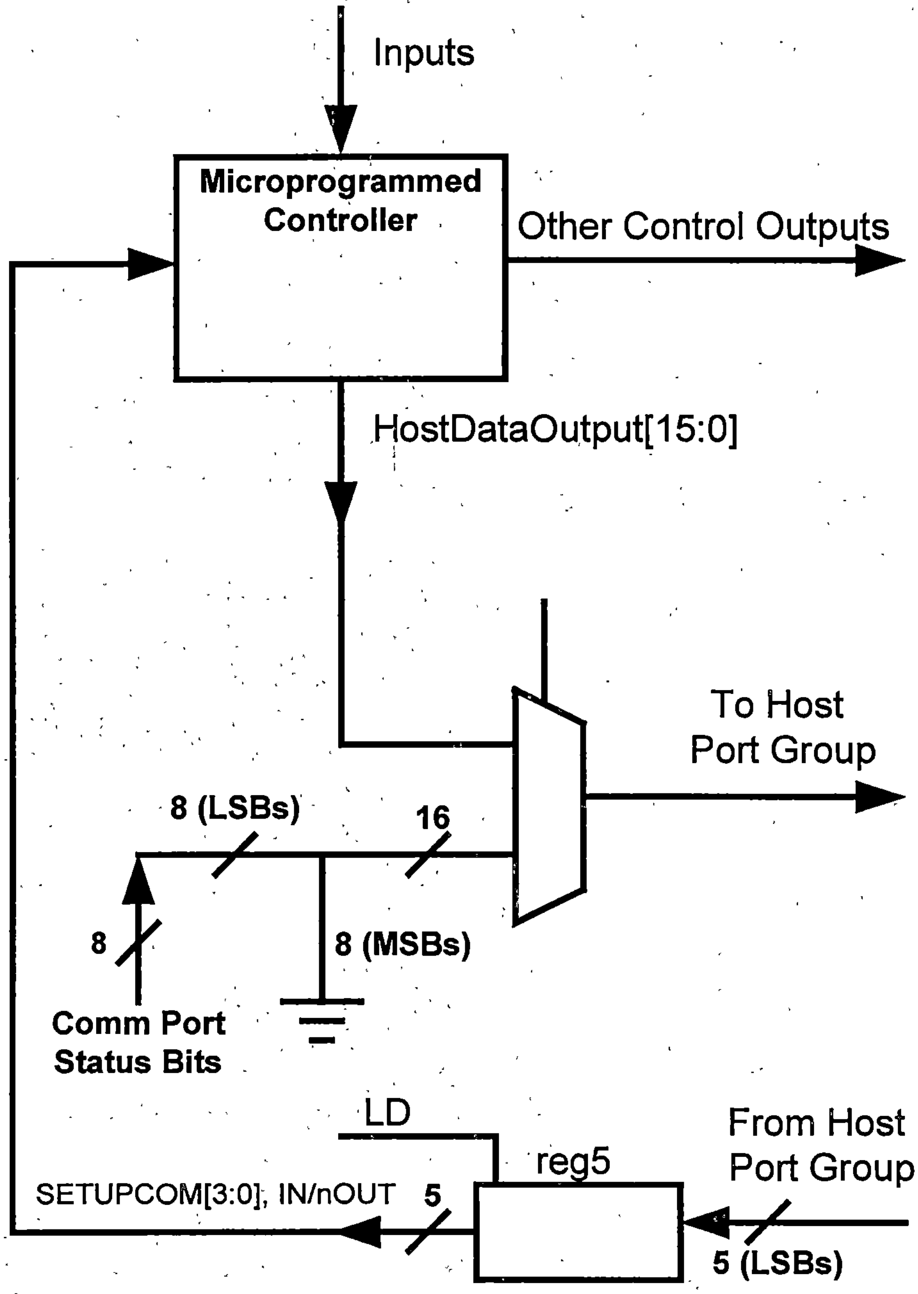


Figure 3-12. Controller Group Block Diagram

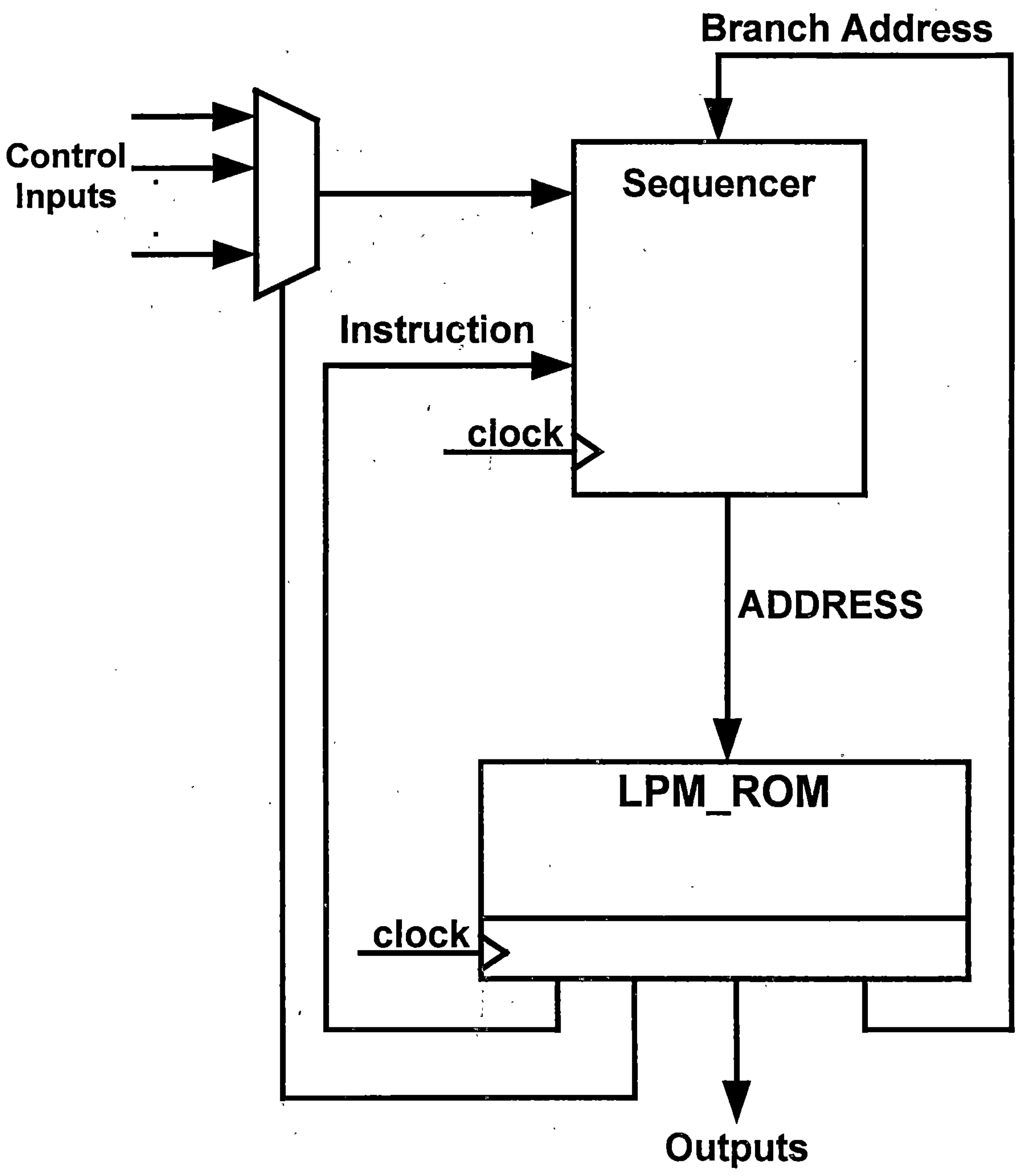


Figure 3-13. Microprogrammed Controller

available: continue, branch on condition code equal to zero, branch on condition code equal to one, and unconditional branch.

3.5.6 Controller Performance

Since the microprogrammed controller executes a microinstruction every clock cycle, the maximum data rate that the controller can attain is defined by how many microinstructions it takes to accomplish the transfer and by the clock period. Since the microprogrammed controller operates at the board clock rate of 40 MHz, each microinstruction takes 25 nS to execute.

For a read transfer, after the controller has strobed LOAD to load a word into the comm port group, it takes 8 clock cycles until the LOADACK signal is checked again. This means that if LOADACK is low for less than 200 nS after the LOAD line is strobed high, the controller will exit the Idle loop at the first opportunity and start the next read access. That is, it will branch back to the read loop in 6 clock cycles after entering the Idle loop. This results in the maximum read transfer rate. Assuming an auto-incrementing access, the fastest read transfer possible takes 19 clock cycles, which is the shortest path through the read loop and the Idle loop. This is 475 nS per 32-bit word or 119 nS per byte and corresponds to a maximum read rate of 8.42 MB/S.

For a write transfer, after the controller has strobed DATACK to latch the word into the host port group, it takes 10 clock cycles until the DAV signal is checked in the

Idle loop. This means that if DAV is low for less than 250 nS after the DATACK line has gone high, the controller will branch out of the Idle loop at the first opportunity, and conduct writes at its maximum rate. Assuming an auto-incrementing access, the fastest write transfer takes 18 clock cycles. This corresponds to a theoretical maximum write rate of 8.88 MB/S.

Chapter 4
Tests and Results

4.1 Test Methods

The checkout of this design began with tests to ensure that the major components on the board functioned. The first test was simply checking for short circuits on the board. This was done with a multimeter. No shorts were detected. The board was checked for shorts on several signal traces on each layer, and between the power layers and ground. This was done both before and after having components installed on the board.

The following tests were accomplished by configuring the FPGA through the JTAG header using an Altera BitBlaster download cable. A simple state machine controller was designed to load code into the DSP via its host port (HPI). Part of the memory in the FPGA was used to store a data file containing the DSP machine code, and any data that needed to be loaded for a test. To start this test the FPGA was programmed and when this was complete, the system implemented in the FPGA would start working. The DSP, which was in host boot mode, received data and code from the FPGA. When the FPGA was finished, it wrote to the host port control register to make the DSP exit reset. When the DSP exited reset, it immediately started executing code, beginning at address 0. The Texas Instruments C6201 Code Generation Tools package was used to develop the assembly language code and to provide the resulting machine code listings. During this development work, this was the method used to configure the FPGA and to load code and data into the DSP.

The purpose of the second test was to load code into the DSP which would simply start one of the DSP timers. This test was conducted using the method described above. In other words, when the DSP came out of reset, one of the timers would pulse at the frequency defined in the code. The timer0 output of the DSP was observed with an oscilloscope. This test yielded the proper timer output waveform, so it was successful. This test proved the following: the FPGA was being programmed successfully, the process used to communicate through the HPI worked, the host boot mode functioned as expected, and the DSP functioned at least in a limited manner.

The next tests involved the entire design, including the C4x-compatible comm port interface implemented in the FPGA. During these tests, a C6201 DSP module was installed on a TIM-compliant, VME motherboard. A PCI, TIM-compliant motherboard with a C44 DSP TIM was installed into a PC nearby. The two motherboards were connected with a 2.5' (76.2 cm) ribbon cable. This long ribbon cable produced the worst case delays that might be observed in a practical system. With this configuration, the design could be tested by conducting reads and writes between the C44 DSP TIM and the C6201 DSP TIM over a C4x-compatible comm port. An emulator was used to load and execute test code in the C44 DSP. This C44 test code was used to execute writes to the C6201 board and collect data from reads of the C6201 board. For the read and write tests, the microprogrammed controller was used to load machine code and data into the C6201 DSP, as well as controlling the comm port hardware.

The third test consisted of transferring data to the C44 from the C6201 (i.e. a read transfer). First, the C44 code was executed so that the C44 and PCI board would be ready to receive the data. Following that, the FPGA was programmed and the test commenced. The C6201 had a table of 16 32-bit words in memory to transfer. The code loaded into the C6201 would first start a setup procedure for a read transfer. After the comm port was set up for a read, and the bus was ready, the transfer would commence. An emulator was used to monitor the transferred words for review. Since the data was known, it was a simple matter to compare corresponding words to ensure that the system was working properly. Since the comm port being used on the C6201 board is a default input port, this test also proved that the interface would properly conduct a token transfer. The read transfer was also observed with a logic analyzer.

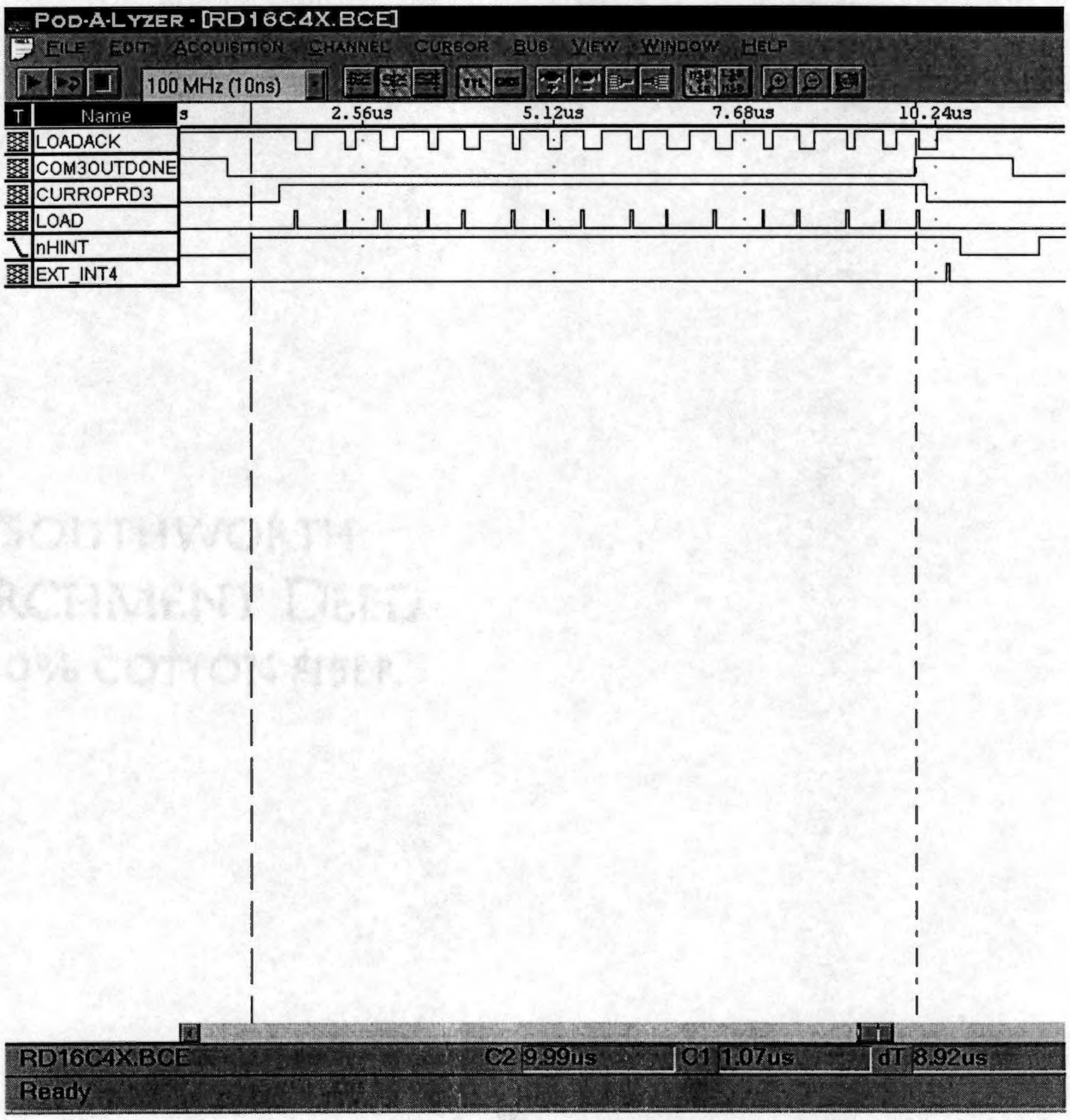
The fourth test was a data transfer into the C6201 board (i.e. a write transfer). In this test the FPGA was programmed first so that the C6201 would be ready for the data being written into it. Once the C6201 was running, it would set up the comm port for a write. Next, the C44 code was executed, which started the write transfer. The C44 would send a block of 16 32-bit words. For this test, the same 16 data words were already loaded in the C6201 in different memory locations than the presently written data. The C6201 code would start a timer at a given frequency if the two sets of data were equal. However, if the data written was not correct, the timer would run at a different frequency. This test was also observed with a logic analyzer.

4.2 Results

4.2.1 Read Transfer Results

The read transfer resulted in the correct data values appearing on the monitor. Therefore, the read transfer, along with a token transfer, was successfully executed. The average data rate was calculated using data from the logic analyzer output, which is shown in Figure 4-1. The time measured for a 16-word read transfer was 8.92 uS, which corresponds to an average data rate for a read transfers of 7.18 Mbytes/S. The theoretical maximum rate of the controller is 8.42 Mbytes/S.

The logic analyzer output shows that the read transfer progressed as expected. Each LOAD strobe represents a data word that was successfully read out of the C6201 TIM. The time the LOADACK line remained low (i.e. the comm port is not ready for another word) ranged from 100 nS to 290nS. Recall that if LOADACK is low for less than 200nS after the LOAD strobe, the controller operates at its maximum speed during an auto-incrementing read. Therefore, the difference in the measured data rate and the maximum rate was due to hardware outside of the C6201 TIM introducing delays between read transfers. When the LOADACK line was low for less than 200 nS, the time between the current and next LOAD strobes was approximately 480 nS. That corresponds to 19 clock cycles, which is the minimum possible number of clock cycles in which the controller can conduct a read transfer. When LOADACK was low for greater than 200 nS, the time between the two nearest LOAD strobes was 650 nS. One cycle



RD16C4X.BCE C2 9.99us C1 1.07us dT 8.92us
Ready

Figure 4-1. Read Transfer Test Logic Analyzer Output

through the Idle loop takes 7 clock cycles or 175 nS. This means that the LOADACK line was low until after it completed one cycle through the Idle loop for those slower reads. The variation in LOAD strobe intervals in discrete steps of approximately 175 nS was caused by the LOADACK line being detected high, either at the first or second Idle loop.

4.2.2 Write Transfer Results

The write transfer test also yielded successful transfers. The words were written into the C6201, and the test was observed using the logic analyzer. The logic analyzer output is shown in Figure 4-2. Each DATACK pulse represents a data word that was successfully written to the C6201 TIM. The write transfer time measured with the logic analyzer for 16, 32-bit words was 12.40 uS. This corresponds to an average data write rate of 5.15 Mbytes/S. The theoretical maximum write rate for the controller is 8.88 Mbytes/S.

The write transfer tests yielded a much lower data rate than the read transfers because the DAV low period was much longer than the average LDACK low period measured in the read transfer tests. This means that the data rate was lower because hardware other than the C6201 TIM introduced longer delays between write transfers than were introduced during the read transfers.

The write transfers resulted in very uniform high and low periods with respect to the DAV line. The DAV low period ranged from 490 nS to 500 nS. This was much

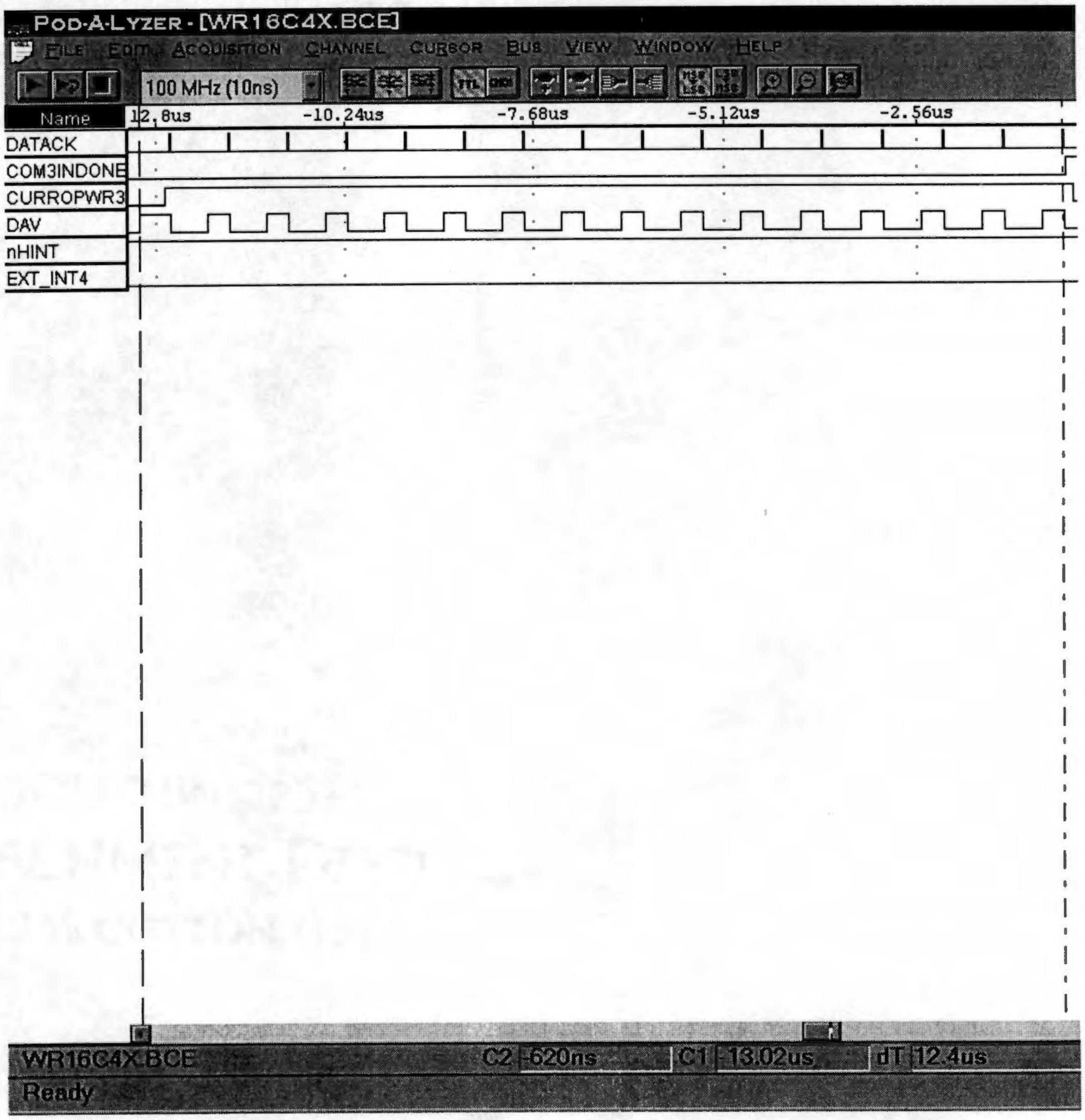


Figure 4-2. Write Transfer Test Logic Analyzer Output

higher than 250 nS, which is the maximum DAV low period such that the controller would operate at its maximum write transfer rate. The interval measured between DATACK strobes was 800 nS. This means that the controller conducted the write transfer, which requires 18 clock cycles (450 nS), and then the C6201 TIM waited in the Idle loop for 14 clock cycles (350 nS) for the other hardware to provide the next data word. Since one cycle through the Idle loop takes 7 clock cycles, the C6201 TIM had to wait for two Idle loops before the next word would arrive. If the external delays were a little less it is probable that only one idle loop would be executed. This would result in write transfer rates comparable to the measured read transfer rates.

Chapter 5
Conclusions, Expansion, and Future Work

5.1 Conclusions

The C6x DSP TIM developed in this thesis was built and tested. It can be used in a message passing, multiple processing system. The ability to conduct read and write data transfers through a C4x-compatible comm port was demonstrated.

The microprogrammed controller implemented in the FPGA was an effective means of controlling the comm port interface, which required nearly 100 instructions and 46 controller output bits. The embedded memory array in the Altera FLEX10K FPGA was of sufficient size to allow the microprogram to be loaded with more than 60% of the memory to spare. The access time of the memory array was sufficiently fast to meet the required clock rate of 40 MHz for the controller. The design with one comm port used only 11% of the logic resources in the FPGA. Additional comm ports will use the same controller, and so will require much less than 11% more per comm port. Therefore, ample logic resources and memory are free for use in future applications.

Given the complicated means needed for transferring data between the 16-bit C6x host port and a C4x-compatible comm port, the maximum data rates are relatively high. The maximum read transfer data rate is 8.42 Mbytes/S, and the maximum write transfer data rate is 8.88 Mbytes/S. Measured data transfer rates through a long (2.5') ribbon cable yielded an average rate of 7.18 Mbytes/S for read transfers and an average rate of 5.15 Mbytes/S for write transfers. The measured read transfer data rate of 7.18 Mbytes/S is good for such a long ribbon cable. If a shorter cable were used, the read data rate should reach the maximum rate of the controller. The lower write transfer data rate obtained in the tests were due to influences other than the C6201 TIM. Therefore, with some changes in the test configuration, higher data rates should be achieved.

5.2 Expansion to Four Comm Ports

The C4x - compatible comm port interface can easily be expanded to include four comm ports. The majority of the hardware needed is three additional comm port group modules, which can simply be copied. The control lines and data busses for the comm ports can be selected with various multiplexers. One multiplexer for example, would select which comm port's control signals would be observed by the controller. Another multiplexer would be needed to select which comm port to drive the host port data bus. The controller outputs would also need to be selected with a multiplexer so that the controller code would require very little modification. This approach will also use far less memory in the FPGA than having a set of controller output bits in the memory array for each comm port.

In the controller group, a two-bit counter could be used to keep track of the active comm port. This counter would be incremented on each pass through the Idle loop. The counter output would feed all of the multiplexers, thus selecting the active comm port. This would require a single bit to be added to the microinstruction, which would be an increment enable line for the two-bit counter. The Idle loop microinstruction sequence would only have to be modified to include enabling this counter once at the end of the loop.

Adding more comm ports will increase the number of possible network paths between processors, thereby increasing the potential bisection bandwidth of a system of TIMs. However, traversing the Idle loop several times between each access during a block transfer will degrade individual comm port data rates.

5.3 Future Work

This board is also capable of supporting the C6701 floating-point DSP. The C6701 requires a 1.8 V source for its processor core, instead of the 2.5 V source required by the C6201. Providing a 1.8 V source is accomplished by simply changing the values of four resistors on the board. Several TIMs will be built with the C6701 when these DSPs are available. Benchmark testing of the C6701 DSP will also be needed. Benchmark testing needs to be conducted on the C6201 TIM as well, with regard to both computational performance and the memory architecture.

The TIM developed in this thesis will accept a 128k X 32 SBSRAM capable of operating at clock rates of 166 MHz in place of the existing 64k X32 SBSRAM. This would allow the SBSRAM to be operated at the full CPU clock rate of 160 MHz, and it would have twice the capacity of the SBSRAM currently used on the C6x TIM. When these memory devices become available, it would be cost effective to install them on any new C6x TIMs to be built later.

After the final comm port configuration is determined, the DSP boot code and the FPGA configuration data should be loaded into the flash memory. This will allow effortless booting from the flash instead of loading code through the JTAG header. Also, a scheme for booting the DSP from the flash, then receiving additional code through one of the comm ports would be useful.

Methods of increasing the performance of the comm port interface should be explored. This will be especially important if four comm ports are used. One method would be increasing the controller clock rate. This would require that the microcoded controller used in this thesis be modified since its maximum clock rate is currently 44 MHz. The minimum signal pulse widths required for the HPI, both for high and low pulses, is 12.5 nS for the 160 MHz processor clock rate in this design. This means that controller clock speeds of up to 80 MHz will work with respect to the HPI timing requirements. However, a controller would need to be developed that can attain that clock rate.

REFERENCES

REFERENCES

[1] "TMX320C6201 Digital Signal Processor," Advance Information Data Sheet, Texas Instruments Corporation, Houston, TX, March 1998.

[2] "TMS320C62xx CPU and Instruction Set Reference Guide," Texas Instruments Corporation, July 1997.

[3] "TMX320C6701 Floating-Point Digital Signal Processor," Advance Information Data Sheet, Texas Instruments Corporation, Houston, TX, May 1998.

[4] "TMS320C6201/C6701 Peripherals Reference Guide," Texas Instruments Corporation, Houston, TX, March 1998.

[5] B. Wilkinson, Digital System Design. Englewood Cliffs, NJ: Prentice Hall, 1987.

[6] F.P. Prosser and D.E. Winkel, The Art of Digital Design. Englewood Cliffs, NJ: Prentice Hall, 1987.

[7] "FLEX 10K Embedded Programmable Logic Family," Data Sheet, v3, Altera Corporation, San Jose, CA, January 1998.

[8] "Configuring FLEX 10K Devices," Application Note 59, v1, Altera Corporation, San Jose, CA, December 1995.

[9] "BitBlaster Serial Download Cable," Data Sheet, v4.01, Altera Corporation, San Jose, CA, February 1998.

[10] "IEEE 1149.1 (JTAG) Boundary Scan Testing in Altera Devices," Application Note 39, v4.02, Altera Corporation, San Jose, CA, November 1998.

[11] "TMS320C4x User's Guide," Texas Instruments Corporation, Houston, TX, May 1991.

[12] "TIM-40 : TMX320C4x Module Specification," v1.00, Texas Instruments Corporation, Houston, TX, 1992.

[13] H. W. Johnson and M. Graham, High-Speed Digital Design : A Handbook of Black Magic. Upper Saddle River, NJ: Prentice Hall, 1993.

[14] D. A. Patterson and J. L. Hennessey, Computer Organization & Design : The Hardware / Software Interface. San Francisco, CA: Morgan Kaufmann Publishers, 1994.

[15] "MT58LC64K32/36D8 64K x 32/36 SYNCBURST SRAM," Data Sheet, Micron Technology, Inc., Nampa, ID, January 1997.

[16] "SyncBurst Features and Options," [Online Document], Micron Semiconductor Products, Inc., Nampa, ID, 10 December 1998, Available HTTP: http://www.micron.com/mti/msp/html/syncfeat.html

VITA

Jeffrey Patterson was born in Winchester, Tennessee on April 8, 1969. He attended elementary and secondary school in Franklin County, Tennessee. In 1993, he graduated from Tennessee Technological University with a Bachelor of Science degree in Electrical Engineering. While attending TTU, he was inducted into the Eta Kappa Nu Honor Society. Upon graduation at TTU, he was commissioned in United States Army Reserve. In October 1993, he graduated from the United States Army Engineer Officer's Basic Course.

Following graduation, he worked four years for the Army Corps of Engineers - Nashville District, as an Electrical Design Engineer. In 1997, he graduated from the Army Engineer Officer's Advanced Course. In the same year Mr. Patterson received a Graduate Research Assistantship at The University of Tennessee Space Institute. He received a Master of Science degree in Electrical Engineering in May 1999. He is currently employed with Raytheon Systems Company in Tucson, Arizona.