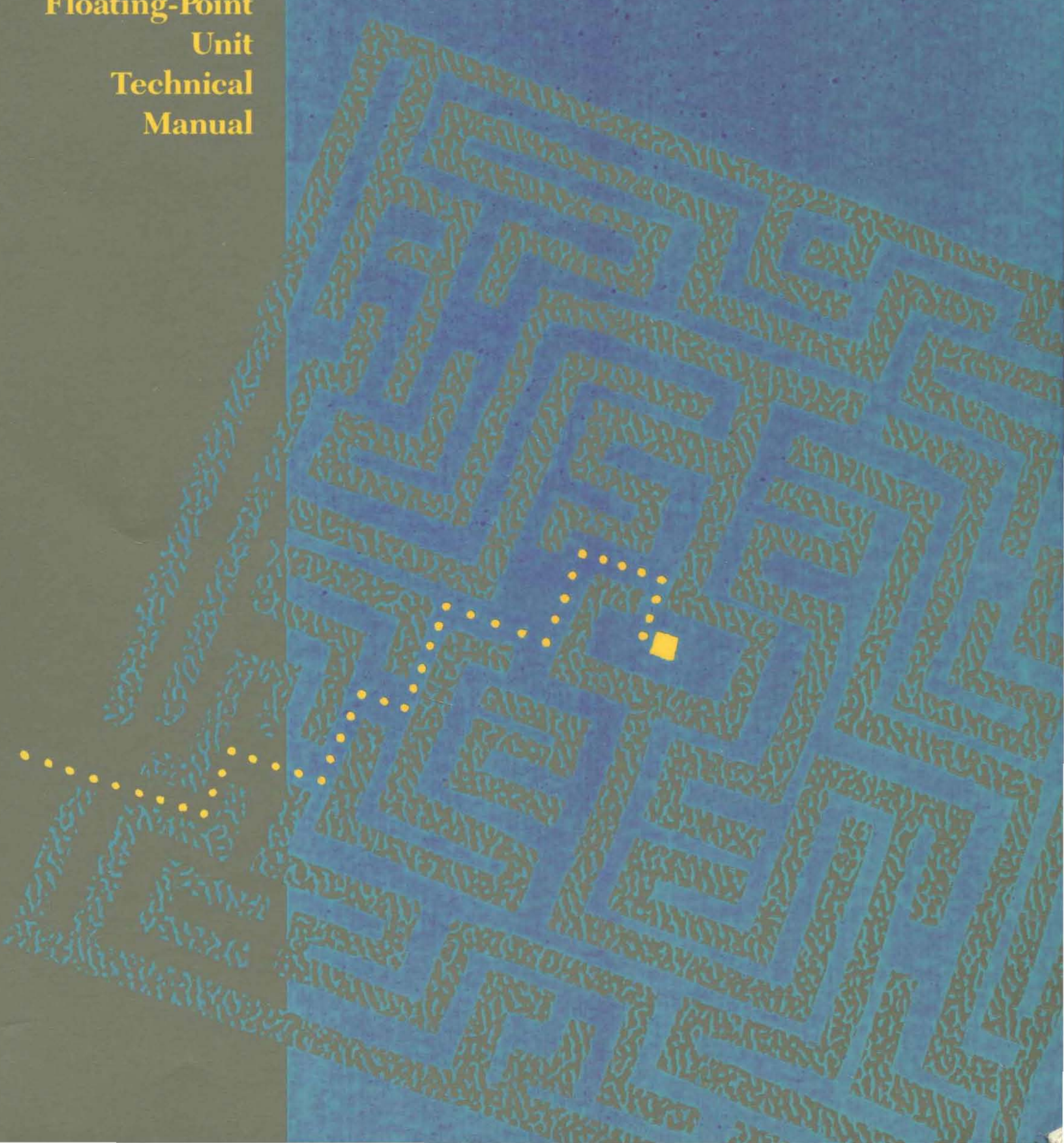
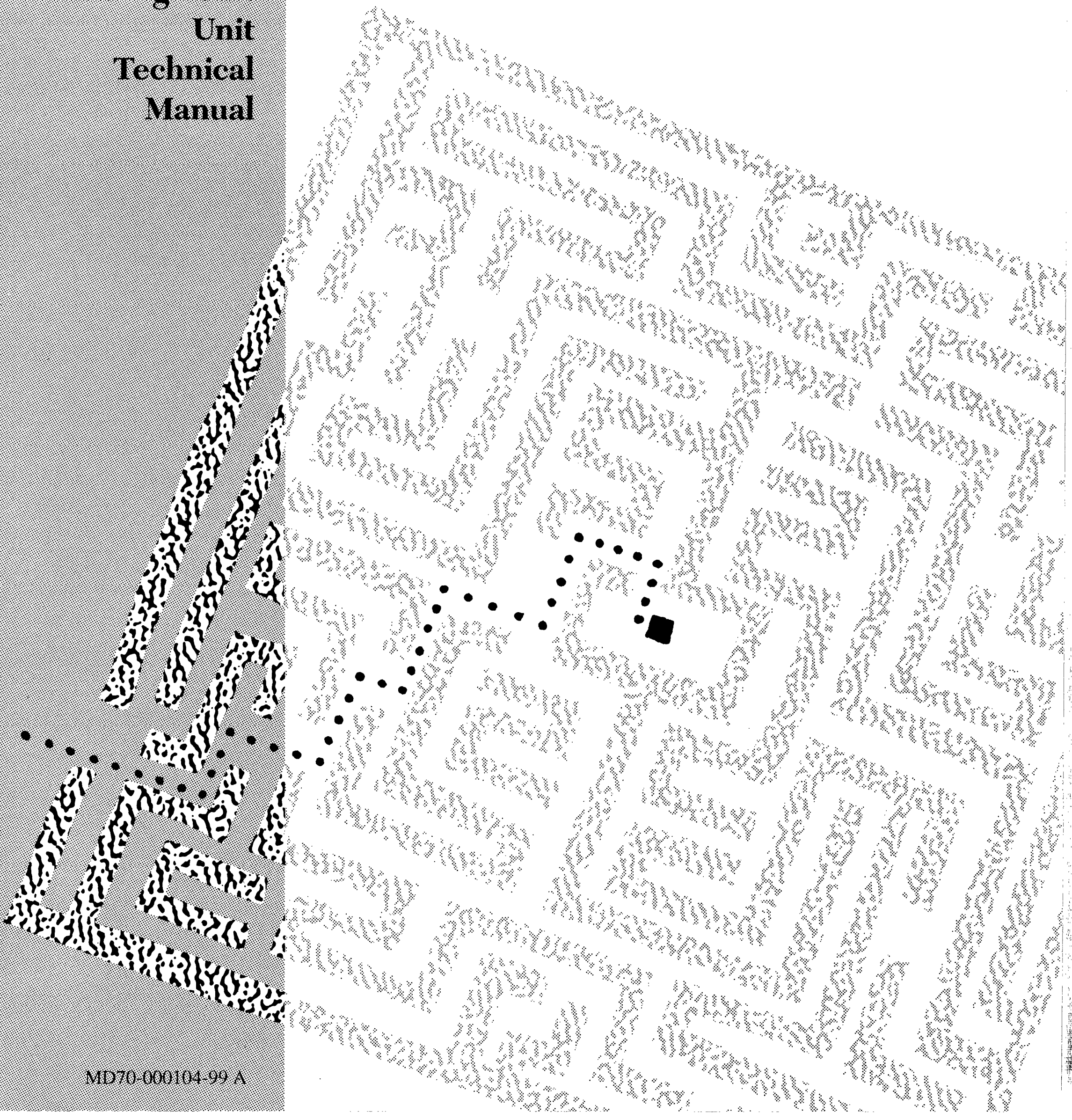
LSI LOGIC

L64814 Floating-Point Unit Technical Manual



LSI LOGIC

L64814 Floating-Point Unit Technical Manual



MD70-000104-99 A

Publications are stocked at the address given below. Requests should be addressed to:

LSI Logic Corporation
1551 McCarthy Boulevard
Milpitas, CA 95035
Fax (408) 433-6802

Preface

This Technical Manual describes the L64814 Floating Point Unit (FPU) from LSI Logic Corporation. The FPU is the single-chip floating-point processor in the high-performance LSI Logic SPARC (Scalable Processor Architecture) microprocessor family. The FPU provides fast floating-point arithmetic operations, compares, and format conversions, as well as efficient data loads, stores, and moves.

Audience

The L64814 FPU Technical Manual provides the information required by system-level programmers and hardware designers to design software and hardware systems that use the FPU. Note that the manual assumes a familiarity with computer architectures, in particular SPARC, and with hardware design and implementation. This manual also assumes that the designer has access to information on other components of the SPARC CPU core.

For the system-level programmer, this document describes how the FPU implements the SPARC floating-point processor functionality and instruction set, as outlined in the *SPARC Architecture Manual.* The functional description is at the bit-level.

For the hardware designer, this document provides the electrical, logical and physical data necessary to integrate the FPU into a particular implementation of a SPARC microprocessor-based design.

Organization

This manual consists of these chapters:

- **1. Introduction:** This chapter provides an overview of the FPU's functionality and its features. It also describes the FPU's conformance with the SPARC computer architecture.
- **2. FPU Architecture and Operation:** This chapter presents the basic architecture and operation of the FPU.
- **3. Internal Operation and Organization:** This chapter discusses how the FPU executes floating-point instruction. It provides a detailed description of the FPU internal architecture, and illustrates the FPU timing.
- **4. External Interface:** This chapter describes the FPU in a system context. It discusses the IU, memory, and coprocessor interface configurations and describes the interface signals and their associated protocols. The chapter also presents the FPU's electrical requirements and AC timing, and it provides pinout and packaging information.

Related Publications

For additional information on various related topics, refer to the following documents:

L64811 SPARC Integer Unit (IU) Technical Manual, MD70-000102-99
LSI Logic Corporation, 1551 McCarthy Boulevard, Milpitas, CA 95035
Fax (408) 433-6802

L64815 Memory Management, Cache Control, and Cache Tags Unit (MCT) Technical Manual, MD70-000101-99
LSI Logic Corporation, 1551 McCarthy Boulevard, Milpitas, CA 95035
Fax (408) 433-6802

L64853 SBus DMA Controller Technical Manual, MD70-000109-99
LSI Logic Corporation, 1551 McCarthy Boulevard, Milpitas, CA 95035
Fax (408) 433-6802

SPARC Architecture Manual, MD70-000111-99
LSI Logic Corporation, 1551 McCarthy Boulevard, Milpitas, CA 95035
Fax (408) 433-6802

Contents

Tables

Chapter 1: Introduction

This Technical Manual describes the L64814 Floating-Point Unit (FPU) from LSI Logic Corporation. The FPU is a high-performance, CMOS implementation of the SPARC (Scalable Processor ARChitecture) floating-point unit. SPARC is the 32-bit RISC (Reduced Instruction Set Computer) system architecture from Sun Microsystems.

The LSI Logic L64800 Family of devices which implement and support SPARC-system development includes the L64811 Integer Unit (IU), the L64815 Memory Management, Cache Control, and Cache Tags Unit (MCT), and the L64853 DMA Controller, in addition to the L64814 FPU. The FPU combines a floating-point controller with a high-throughput floating-point processor to provide a single-chip floating-point processor solution for SPARC-based systems.

This manual includes these four chapters:

- Introduction
- FPU Architecture and Operation
- Internal Operation and Organization
- External Interface

This chapter, Introduction, presents an overview of the FPU. The chapter has this organization.

1.1 FPU Features lists the most important features of the FPU.

1.2 SPARC Architecture Conformance summarizes how the FPU conforms with the SPARC floating-point processor architecture as outlined in the *SPARC Architecture manual.*

1.1 FPU Features

The L64814 provides the following features.

- High-performance operation

  Provides double-precision Linpack floating-point operation at up to:

L64814-25 MHz	3.8 MFlops
L64814-33 MHz	5.0 MFlops
L64814-40 MHz	6.0 MFlops

- Low-cost solution

  Integrates a floating-point controller and floating-point processor on a single chip for cost-efficient system implementation.

- Wide range of operating frequencies

  25, 33, and 40 MHz versions.

- Implements IEEE exception handling directly in hardware.
- 64-bit wide internal datapath for all floating-point operations provides highly efficient double-precision performance.
- Connects directly to the L64811 Integer Unit (IU) and L64815 Memory Management, Cache Control, and Cache Tags Unit (MCT).
- Pin-compatible with the WEITEK Abacus 3171 and Texas Instruments TMS390C602 Floating Point Units.
- Available in the advanced 143-pin, plastic or ceramic cavity-up pin grid array packages.

1.2 SPARC Architecture Conformance

This section discusses how the L64814 implementation conforms with the SPARC floating-point unit architecture as outlined in the *SPARC Architecture Manual.* The discussion has this organization:

1.2.1 Instruction Set

The FPU implements the ANSI/IEEE 754-1985 standard for floating-point arithmetic. It operates concurrently with the IU to execute single- and double-precision floating-point operations, as well as register-to-register move instructions, floating-point loads and stores, and floating-point queue and state register instructions. Supported floating-point operations (FPops) are: add, subtract, multiply, divide, square root, compare, and convert. All instructions not currently implemented in the L64814 hardware generate an instruction trap, so that the instructions can be emulated in software. Note that the FPU handles all IEEE exceptions in hardware, except for denormals in the floating-point multiplier unit.

The FPU provides hardware support for integer, single-precision, and double-precision operations. Because it does not provide direct hardware-support for extended-precision operations, the FPU traps extended-precision instructions and the operating system emulates them in software. Refer to Chapter 3 for more specific information on the FPU instruction set and internal operation.

Instruction traps can occur due to unfinished floating-point operate instructions, unimplemented instructions, IEEE exceptions, or sequence errors. In any case, the state of the FPU must remain unaltered, except for the Floating-Point Status Register (FSR) fields which describe the exception. This freeze allows the trap handler to examine both the FSR and the source registers for the operation, so it can properly emulate the instruction.

1.2.2 Modes of Operation

The FPU operates in one of three modes:

- execution
- pending exception
- exception

Following a reset, the FPU enters **execution mode**; execution mode is the normal operating mode for the FPU. When the FPU signals the IU to take a floating-point exception trap, then the FPU enters **pending exception mode**. The IU takes the trap when it decodes the next floating-point instruction; at this time, the FPU enters **exception mode**.

In exception mode, the trap handler empties the floating-point queue of instructions and instruction addresses. As soon as the queue is empty, the FPU returns to execution mode. Refer to Chapter 3 for additional information on FPU operating modes and exception trap handling.

1.2.3 Floating-Point Register File

The FPU contains a floating-point register file, also referred to as the f-registers, which holds the operands for all floating-point operations. Floating-point load instructions read data from memory to the f-registers; floating-point store instructions write data from the f-registers to memory.

The register file contains 32, 32-bit registers, configured as eight rows of four registers each. Figure 1.1 shows the register file organization. Figure 1.1 also illustrates the data representation in the register file. Because integer and single-precision data require 32 bits, any f-register can store an integer or single-precision operand. Double-precision data is 64 bits wide, so one adjacent even-odd pair of f-registers, either the left or right half of a row, can store one double-precision operand. Note that the even (left-most) f-register of the pair stores the most-significant word (MSW) of the operand, while the odd (right-most) f-register of the pair stores the least-significant word (LSW).

Register File Organization:

f-registers				
	f0	f1	f2	f3
	f4	f5	f6	f7
	f8	f9	f10	f11
	f12	f13	f14	f15
	f16	f17	f18	f19
	f20	f21	f22	f23
	f24	f25	f26	f27
	f28	f29	f30	f31

Data Representation in the f-registers:

single-precision or integer data	W	W	W	W
double-precision data	MSW	LSW	MSW	LSW

Figure 1.1 Floating-Point Register File

Figure 1.2 illustrates how the floating-point instructions address the f-registers. Integer and single-precision data require a full five-bit address to access one of the 32 registers. Because double-precision data can reside in any of 16 register-pairs, accessing double-precision data requires only a four-bit address; the least-significant bit (LSB) is ignored.

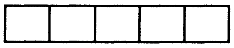
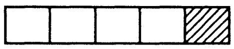
Data Type	Address (rd, rs1, or rs2 field) in Instruction
single-precision or integer data	 five-bit register file address
double-precision data	 four-bit register file address; LSB is ignored

Figure 1.2 Floating-Point Status Register Addressing

1.2.4 Floating-Point Status Register (FSR)

The FSR is a 32-bit register which holds FPop status information as well as attributes which the FPU uses during operation. When the system resets the FPU, only the version field and the reserved bits in the FSR retain their values, because they are tied high or low; all other fields are undefined.

Figure 1.3 shows the FSR with all fields labeled. The fields are summarized in Table 1.1, and explained further following the table.

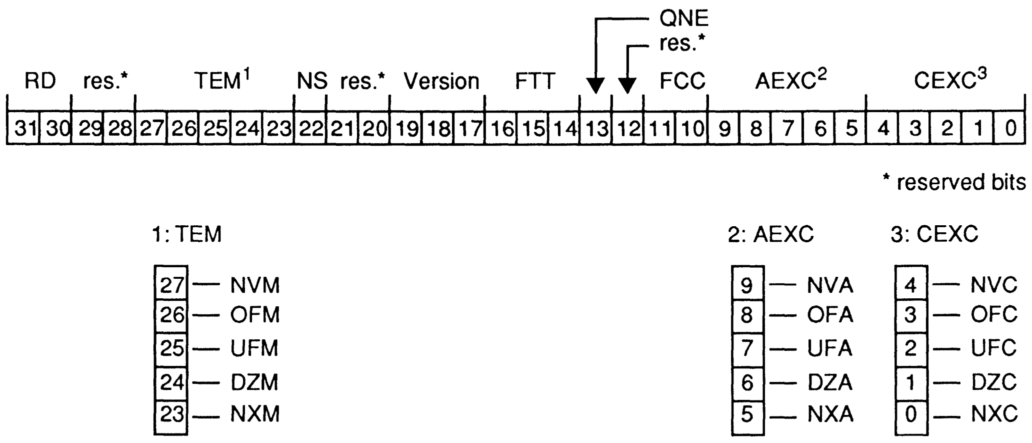


Figure 1.3 Floating-Point Status Register (FSR)

Field	FSR Bits	Description	Value(s)	Loadable[1]
RD	31:30	Rounding Direction	00 - Round to nearest (tie is even) 01 - Round to 0 10 - Round to +∞ 11 - Round to -∞	Yes
reserved	29:28		00	No
TEM	27:23	Trap Enable Mask	0 - Disable individual trap 1 - Enable individual trap	Yes
NVM	27	Invalid Operation Trap Mask		
OFM	26	Overflow Trap Mask		
UFM	25	Underflow Trap Mask		
DZM	24	Divide-by-zero Trap Mask		
NXM	23	Inexact Trap Mask		
NS	22	Nonstandard Floating-Point	0 - Disable nonstandard mode 1 - Enable nonstandard mode	Yes
reserved	21:20		00	No
Version	19:17	Version Number for FPU	001	No
FTT	16:14	Floating-Point Trap Type	000 - None 001 - IEEE Exception 010 - Unfinished FPop 011 - Unimplemented FPop 100 - Sequence Error	No
QNE	13	Queue Not Empty	0 - Queue empty 1 - Queue not empty	No
reserved	12		0	No
FCC	11:10	Floating-Point Condition Code	00 - = 01 - < 10 - > 11 - ? (unordered)	Yes
AEXC	9:5	Accrued Exception Bits		Yes
NVA	9	Accrued Invalid Exception		
OFA	8	Accrued Overflow Exception		
UFA	7	Accrued Underflow Exception		
DZA	6	Accrued Divide-by-zero Exception		
NXA	5	Accrued Inexact Exception		
CEXC	4:0	Current Exception Bits		Yes
NVC	4	Current Invalid Exception		
OFC	3	Current Overflow Exception		
UFC	2	Current Underflow Exception		
DZC	1	Current Divide-by-zero Exception		
NXC	0	Current Inexact Exception		
Note 1. The entries in this column indicate whether the LDFSR instruction can load this field.				

Table 1.1 Floating-Point Status Register (FSR) Summary

RD: The RD field state determines the rounding direction for operations in the FPU. Software may load this field via the LDFSR (Load Floating-Point Status Register) command.

TEM, AEXC, and CEXC: The configuration of these fields determines how the FPU handles traps. Software may load all three fields via the LDFSR command. The TEM field specifies which traps are enabled. When an exception occurs, the CEXC field status changes to reflect the current exception. Each bit of the TEM field is logically ANDed with its corresponding bit in the CEXC field, and if the resulting value is nonzero, then an IEEE exception trap occurs. If an exception occurs but is masked by the state of TEM, then it is logically ORed into the corresponding AEXC files. Note that results of FPops whose exceptions are masked are written into the register file, whereas any FPop that causes a trap does not write its results to the register file.

NS: The NS bit, when set via LDFSR, affects any subsequent floating-point multiply, divide, or square root operations. The impact is basically that if any input operand is subnormal, then the operand becomes zero, and if any result is subnormal, then the result is set to zero. These conditions are referred to as *abrupt underflow*. For example, on a multiply instruction, if one or both of the inputs is subnormal, then the result is set to zero and no exception occurs. If, however, the input operands are not subnormal but the result is, then the result is set to zero and an underflow exception occurs.

With a divide instruction, if the dividend is subnormal and the divisor is neither subnormal nor zero, then the result is zero and no exception occurs. If the dividend is neither subnormal nor zero and the divisor is subnormal, then the result goes to infinity and a divide-by-zero exception occurs. If both the divisor and dividend are either subnormal or zero, then the result is a NaN (Not a Number) and an invalid exception occurs. Finally, if only the result is subnormal, then it becomes zero and an underflow exception occurs.

For a square root instruction, if the operand is subnormal, then the result is zero and no exception occurs.

In typical usage, if NS is set, then AEXC and CEXC are ignored. Note that in situations where many underflows could occur, the programmer may not care about following the IEEE floating-point arithmetic standard mentioned earlier. In this case, he or she can achieve a significant improvement in program execution speed by setting NS; the amount of improvement depends on the density of instructions which would cause exceptions in the program.

Version: The version field contains a number, assigned by Sun Microsystems, which denotes the version on the FPU. This number is 001 for the current FPU.

FTT: The FTT field is updated at the completion of every FPop. If the FPop completes in a normal fashion, then the FPU writes a zero at the end of the FPop. If, however, a trap occurs, then the FPU writes the correct trap type into the FTT field.

QNE: The trap handler reads the QNE bit to determine when the handler has stored the entire floating-point queue and the queue is empty. During execution mode in the FPU, the QNE bit always returns zero to an STFSR (store floating-point status register) instruction, because all FPops complete execution prior to execution of an STFSR.

FCC: The FCC field holds the floating-point condition code bits which compare operations use to determine the results of comparisons.

Chapter 2: FPU Architecture and Operation

This chapter provides an overview of the basic architecture and operation of the FPU. The chapter has the following organization.

> **2.1 Functional Overview** presents a functional block diagram and block-level description for the FPU.
>
> **2.2 Basic Instruction Execution** discusses the instruction cycles which comprise floating-point load/store instructions and floating-point operate instructions (FPops).

Note that Chapter 3 expands on the topics introduced in Chapter 2, to provide a more detailed description of the FPU, its instruction set, and its internal operation.

2.1 Functional Overview

This section introduces the functional blocks which comprise the L64814 FPU and briefly describes their basic operation.

During normal instruction execution, the FPU accesses the Data and Address buses for instructions and instruction addresses, and then decodes the instructions. When a floating-point instruction occurs, the FPU performs the specified operation. Figure 2.1 shows a simplified block diagram of the FPU.

In the diagram, the **Fetch Unit** captures each instruction and its address from the Data and Address buses, respectively. The **Decode Unit** decodes the instruction opcodes and makes them available to the Execution Unit.

The **Execution Unit and Floating-Point Queue** handles floating-point instruction execution. When the L64811 Integer Unit (IU) decodes a valid floating-point operate (FPop) or floating-point load/store instruction, it signals the FPU. The FPU latches the instruction and address from the Decode Unit and starts execution. The Execution Unit includes the two-deep floating-point queue (FQ), which holds the instructions and addresses for the currently executing instructions.

The **Dependency Checker** determines whether the instruction depends on the results or the resources required by other floating-point instructions ahead of it in the queue. If a dependency exists, then the Dependency Checker freezes the instruction pipeline until the dependency is cleared.

The **Load Unit** holds data fetched from memory until the FPU writes it to the **Register File**. The Register File, also referred to as the f-registers, consists of 32, 32-bit registers. These registers store data (operands) for FPops and floating-point load/store instructions.

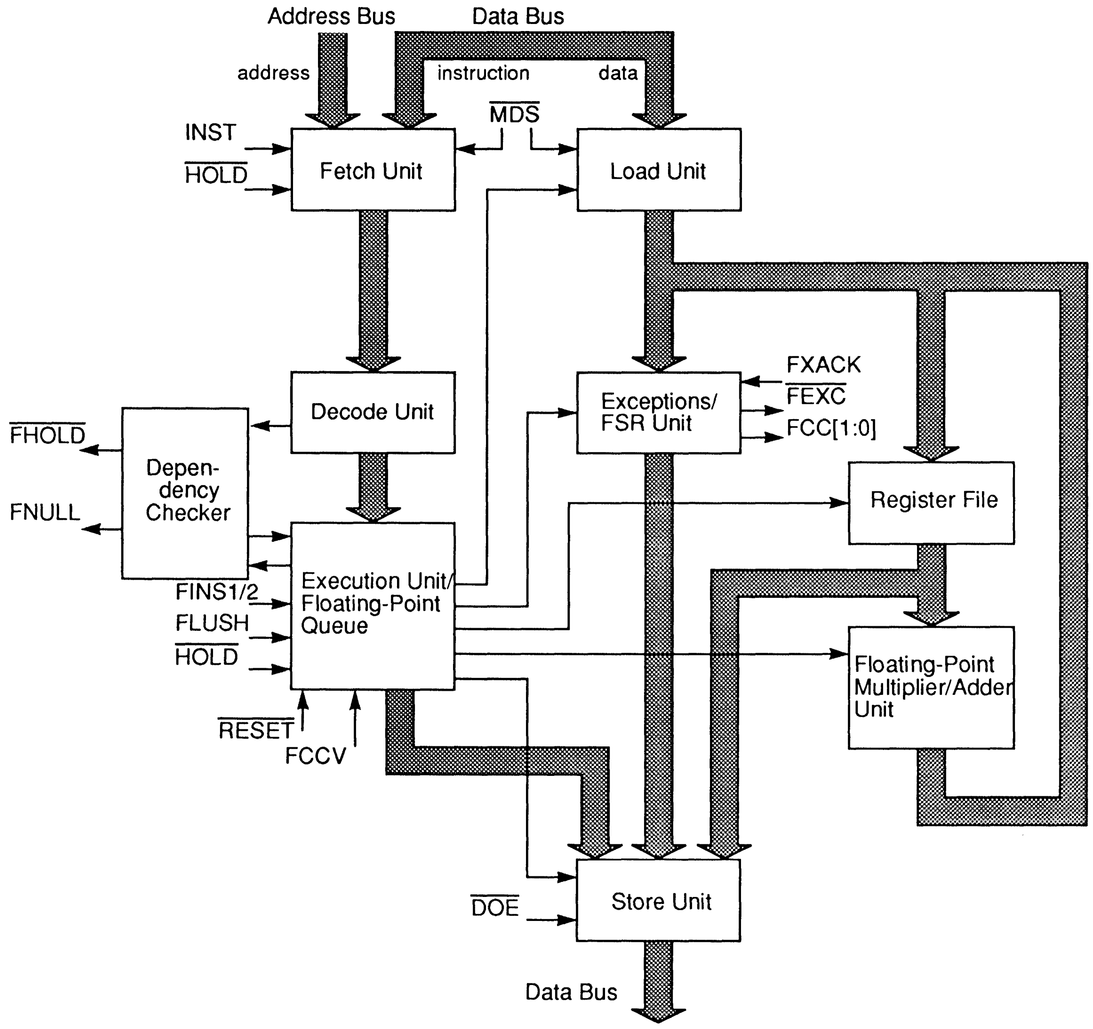


Figure 2.1 L64814 FPU Functional Block Diagram

The **Floating-Point Multiplier/Adder Unit** contains the 32-bit adder and 32-bit multiplier which FPops use to operate on data in the Register File. Because the FPU includes a separate multiplier and adder, it supports parallel execution of FPops.

The **Exceptions/FSR Unit** maintains the status of FPops completing execution, as well as that of the operating mode of the FPU. The Floating-Point Status Register (FSR) is a 32-bit register whose fields store the status and operating mode information.

The **Store Unit** holds data which the FPU drives onto the Data bus during execution of a floating-point store instruction.

The next section provides more information on the execution of floating-point load/store instructions and FPops.

2.2 Basic Instruction Execution

This section discusses the instruction cycles which execute the two basic types of floating-point instructions: floating-point load/store instructions and FPops.

Basic instruction execution in the FPU consists of four stages:

- F: Fetch
- D: Decode
- E: Execute
- W: Write

Note that the F or Fetch stage precedes the Decode stage, but not in a predictable fashion. For example, load instructions cause the FPU to perform the Fetch one or more cycles ahead of the Decode. For this reason, the Fetch does not always occur in the cycle immediately preceding the Decode; the only certainty is that the Fetch does precede the Decode. Also, as explained below, some instructions require extra W stages to complete execution. These extra W stages are referred to as W-help or Wh stages. For more detailed information on instruction execution, refer to Chapter 3.

This section has the following organization:

2.2.1 Floating-Point Load and Store Instructions

2.2.2 Floating-Point Operate Instructions (FPops)

2.2.1 Floating-Point Load and Store Instructions

Floating-point load and store instructions handle three types of transactions:

- transfers to and from memory
- transfers to and from the floating-point status register (FSR)
- transfers from the floating-point queue

More specifically, in the first type of transaction floating-point load and store instructions can either access data in memory and **load** the data into the FPU or **store** the data from the FPU into memory. In the second type of transaction, they can also load the FSR or read and store the FSR value; recall from Section 2.1 that the FSR maintains status and operating-mode information for the FPU. In the third transaction type, which occurs during exception handling, floating-point store instructions clear the floating-point queue (FQ) of all instructions and addresses; the exception (trap) handler performs the instructions in software, along with the instruction which caused the trap. Load and store instructions are discussed below.

Load Instructions

The floating-point load instructions are:

- LDF: load single-precision or integer data
- LDDF: load double-precision data
- LDFSR: load the FSR

The FPU executes these instructions as shown in Table 2.1 below. Where the instructions differ in the action performed during a particular cycle, the appropriate action for each instruction is called out explicitly.

Instruction Cycle	Action
D-Stage	Decode instruction and check operand and resource dependencies.
E-Stage	Hold execution of this instruction if a dependency exists; otherwise, start execution.
W-Stage	LDF or LDFSR: capture data from the Data bus. LDDF: capture most-significant word (MSW) of data from the Data bus.
Wh1-stage	LDF or LFFSR: write data into the Register File (LDF) or FSR (LSFSR). LDDF: capture least-significant word (LSW) of data from the Data bus.
Wh2-stage	LDDF: write data into the Register File.

Table 2.1 Execution of Load Instructions

Note that during load instruction execution, the D- and E-stages of an FPop or store instruction may overlap the W- and Wh-stages of a previous load instruction, even if the FPop or store has an operand dependency on the load instruction. Also, the FPU waits until the load instruction reaches the end of the last Wh-stage (Wh1 for an LDF or LDFSR, and Wh2 for an LDDF) to actually write the data into the register file, so that if a FLUSH occurs the load does not change the state of the register file. This approach requires a technique called **operand forwarding** to make data available for use in an early stage of execution of the following FPop. Operand forwarding works this way.

Two situations can occur which require operand forwarding for efficient operation; when a floating-point load to an f-register in the Register File is followed immediately either by an FPop which uses the contents of that f-register or by a floating-point store of the f-register. In either of these cases, the FPU has only one and one-half cycles from the time when data is captured in the FPU until that same data must be either available in the FPU datapath (for an FPop) or driven onto the Data bus (for a store). Operand forwarding utilizes special logic which forwards data from the Data bus directly to the execution path of the FPop or the store, without going through the Register File. The write to the Register File occurs in parallel with the execution of the FPop or store instruction.

If the IU takes a trap during the floating-point load instruction, then the FPU aborts the load and does not write the load data into the Register File. The FPU flushes the load instruction out of the floating-point queue (FQ) and with it flushes the FPop or store which was to use the data; the trap handler performs both the load and the succeeding instruction, to ensure that the instruction does not use invalid data.

Store Instructions

The floating-point store instructions are:

- STF: store single-precision or integer data
- STDF: store double-precision data
- STFSR: store the FSR
- STDFQ: store the floating-point queue (FQ)

The FPU executes these instructions as shown in Table 2.2 below. Again, where the instruction actions differ in a particular cycle, the appropriate action for each instruction is called out explicitly.

Instruction Cycle	**Action**
D-Stage	Decode instruction and check operand and resource dependencies.
E-Stage	Hold execution of this instruction if a dependency exists. Otherwise, read data from the Register File, FSR, or FQ.
W-Stage	STF or STFSR: drive data onto the Data bus.
(mid-cycle)	STDF: drive most-significant word (MSW) of data onto the Data bus. STDFQ:drive the FQ instruction address onto the Data bus.
Wh1-stage (mid-cycle)	STF or STFSR: write data into the Register File (LDF) or FSR (LSFSR). STDF: capture least-significant word (LSW) of data from the Data bus. STDFQ:drive the FQ instruction onto the Data bus.
Wh2-stage (mid-cycle)	STDF or STDFQ: stop driving the Data bus.

Table 2.2 Execution of Store Instructions

Note that the D- and E-stages of a store instruction may overlap with the W- and Wh- stages of other load or store instructions.

For more detailed information on load and store instruction execution, including sample timing diagrams, refer to Chapter 3.

2.2.2 Floating-Point Operate Instructions (FPops)

The FPU can perform a wide variety of integer, single-precision, and double-precision FPops including: add, subtract, multiply, divide, square root, compare, and convert. Note that the IU executes floating-point branch instructions, and that these instructions are fundamentally transparent to the FPU.

The FPU executes FPops as shown in Table 2.3.

Instruction Cycle	Action
D-Stage	Decode FPop and check operand and resource dependencies.
E-Stage	Hold execution of this instruction if a dependency exists. Otherwise, read operand from the Register File.
W-Stage	Read any additional operand from the Register File; start computing results.
Queue	Compute; FPop is in FQ.
• • •	• • •
Queue	Check exception status.
Queue	Update FSR; write results, or signal a floating-point exception trap if necessary.

Table 2.3 Execution of Floating-Point Operate Instructions

Note that the number of cycles that an FPop takes to read operands depends on the type of FPop. Table 3.2 in Chapter 3 summarizes the instruction cycle counts for floating-point instructions.

Chapter 3: Internal Operation and Organization

This chapter discusses in detail how the FPU executes floating-point instructions. First it presents the SPARC floating-point instruction set. Then it describes the FPU architecture at the register level, and provides timing diagrams which illustrate the instruction execution cycles. The chapter has this organization.

3.1 Instruction Set introduces the FPU instruction set and provides performance information.

3.2 Internal Organization presents a more detailed version of the FPU Architecture.

3.3 Instruction Execution discusses the Fetch, Decode, and Execute stages introduced in Chapter 2.

3.4 Exception Handling explains how the FPU handles operand dependencies, unimplemented instructions, and other conditions which cause exceptions.

3.5 Halting Instruction Execution describes the signals and conditions which stop instruction execution in the FPU and in the IU.

3.1 Instruction Set

The *SPARC Architecture Manual* specifies a complete set of instructions for a SPARC-compatible floating-point processor unit. The processor may either perform all of these instructions in hardware or may perform some in hardware and emulate the rest in software.

The manual specifies two main types of instructions: floating-point load/store instructions and floating-point operate instructions (FPops). Note that, as mentioned in the previous chapter, the FPU does not perform floating-point branch instructions; instead, the IU executes these instructions, and the execution is transparent to the FPU.

Table 3.1 lists the FPU instruction set. The list includes load instructions, store instructions, and four classes of FPops.

The FPU performs three load and four store instructions. LDF and LDDF transfer data from memory to the FPU Register File 32 or 64 bits at a time, respectively. STF and STDF transfer data from the Register File to memory, also 32 (STF) or 64 (STDF) bits at a time. LDFSR and STFSR write to and read from the floating-point status register (FSR). STDFQ is a privileged instruction which reads the entries from the floating-point queue (FQ).

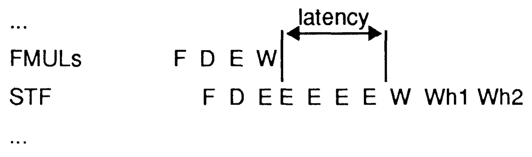
The four classes of FPops are: basic arithmetic operations, compares, format conversions, and register-to-register moves. Note that these move operations do not cause exceptions; exceptions are discussed later, in Section 3.4. The convert, move, and square root instructions use only one source operand, and the compare instructions do not produce a result.

Mnemonic	Instruction
LDF	Load floating-point operand
LDDF	Load double-precision floating-point operand
LDFSR	Load floating-point status register (FSR)
STF	Store floating point operand
STDF	Store double-precision floating-point operand
STFSR	Store floating-point status register (FSR)
STDFQ	Store double-precision floating-point queue (FQ)
FiTO(s,d,x[1])	Convert integer to (single, double, extended[1])-precision floating point
F(s,d,x[1])TOi	Convert (single, double, extended[1])-precision floating-point to integer
FsTO(d,x[1])	Convert single-precision floating-point to (double, extended[1])-precision floating-point
FdTO(s,x[1])	Convert double-precision floating-point to (single, extended[1])-precision floating-point
FxTO(s,d)[1]	Convert extended-precision floating-point to (single, double)-precision floating-point[1]
FMOVs	Move a word from one f-register to another
FNEGs	Negate the operand (invert the sign bit)
FABSs	Take the absolute value (clear the sign bit)
FSQRT(s,d,x[1])	Calculate the (single, double, extended[1])-precision square root
FADD(s,d,x[1])	Add the (single, double, extended[1])-precision operands
FSUB(s,d,x[1])	Subtract the (single, double, extended[1])-precision operands
FMUL(s,d,x[1])	Multiply the (single, double, extended[1])-precision operands
FDIV(s,d,x[1])	Divide the (single, double, extended[1])-precision operands
FCMP(s,d,x[1])	Compare the (single, double, extended[1])-precision operands
FCMPE(s,d,x[1])	Compare the (single, double, extended[1])-precision operands and cause an exception if unordered (that is, if at least one is a NaN, i.e. Not a Number)
Note 1. Trapped instruction	

Table 3.1 FPU Instruction Set

As discussed in the *SPARC Architecture Manual* and mentioned above, an FPU which executes the floating-point instruction set may implement a subset of the instructions in hardware. The FPU then **traps** the unimplemented instructions, and the system software actually performs these trapped instructions. The trap handler emulates the unimplemented instructions by reading the appropriate operands from the Register File (via store floating-point instructions), emulating the FPop in integer-only calculations, and writing the result back to the Register File (via load floating-point instructions). The trap handler also updates the FSR. The L64814 implements in hardware all SPARC floating-point instructions which operate on integer, single-precision, and double-precision data. It traps all extended-precision floating-point instructions for execution in software.

Understanding instruction latency is crucial for optimal compiler design. The latency for an instruction is defined here as the number of cycles from the end of the instruction's W-stage until the result of the instruction is available to be stored. In other words, if in the instruction stream the instruction of interest is directly followed by a store of its result, then the instruction latency is the number of extra E-cycles which the store instruction experiences until the result is available. For example, a multiply followed by a store of the multiply's result looks like this:



In this situation, the four cycles of the latency period are wasted. A far more efficient way to handle instruction latency is for the compiler to insert four other non-dependent instructions between the multiply and its store. Because of the parallel paths through the multiplier and adder in the FPU, one of these instructions can even be an FPop which uses the adder-portion of the datapath, like this:

```
...
FMULs
non-floating-point instruction
FADDs
non-floating-point instruction
non-floating-point instruction
STF            (stores result of FMULs above)
...
```

Table 3.2 shows the instruction latency for each floating-point instruction which the FPU implements in hardware. Note that the table has two parts, to differentiate between instructions which use the floating-point multiplier and those which use the floating-point adder. This differentiation is useful because, as mentioned above, the compiler can successfully interleave these two classes of instructions.

a) Instructions which use the FPU Multiplier:

Instruction	Latency
FMULs.d	4
FDIVs,d	33
FSQRTs.d	45

b) Instructions which use the FPU Adder:

Instruction	Latency
FADDs.d	4
FSUBs,d*	4
FiTOs,d	4
FsTOi,d	4
FdTOi,s	4
FMOVs	2
FNEGs	2
FABSs	2
FCMPs,d	3
FCMPEs,d	3

Table 3.2 FPU Instruction Latencies

The following partial listing illustrates the standard Linpack benchmark loop used to estimate FPU performance.

```
        ldd             [%i1], %f6
        fmuld           %f30, %f6, %f6
        ldd             [%i0], %f10
        ldd             [%i1+8], %f14
        fadd            %f10,%f6, %f10
        fmuld           %f30, %f14, %f14
        ldd             [%i0+8], %f18
        std             %f10, [%i0]
        fadd            %f18, %f14, %f18
        ldd             [%i1+16], %f22
        fmuld           %f30, %f22, %f22
        ldd             [%i0+16], %f26
        std             %f18, [%i0+8]
        faddd           %f26, %f22, %f26
        ldd             [%i1+24], %f0
        fmuld           %f30, %f0, %f0
        ldd             [%i0+24], %f4
        std             %f26, [%i0+16]
        faddd           %f4, %f0, %f4
        inc             -4, %i5
        add             %i5, -3, %l1
        tst             %l1
        inc             32, %i1
        std             %f4, [%i0+24]
        bge             LOOP
        inc             32, %i0
```

Notice that the loop includes eight FPops, stores, loads, and a few other instructions. To determine the performance predicted by this loop, assume the following cycle counts:

- store instructions: 4 cycles
- load instructions: 3 cycles
- other instructions: 1 cycle

With regard to these *other* (that is, not load or store) instructions, both because FPops go into the floating-point queue and because instructions dependent upon the results of loads and stores are spaced far enough apart in the loop, these instructions take effectively one cycle to execute.

Based on these cycle counts, the floating-point instructions constitute eight cycles out of the loop total of 54 cycles. To determine FPU performance, use this equation:

Performance = 8/54 x clock frequency(MHz)

Table 3.3 summarizes the Linpack performance for the FPU.

Linpack Performance		
Condition	**33 MHz**	**40 MHz**
No Cache Miss	4.98 Mflops	6.04 Mflops
25% Degradation due to: Overhead Cache Misses System Effects	3.76 Mflops	4.53 Mflops

Table 3.3 Linpack Performance Summary

Although the above Linpack benchmark is widely used to estimate floating-point performance, the inner loop of the benchmark is limited by the performance of load and store instructions in this L64814 implementation. An alternate performance measure is peak MFlops. The following sequence of FPops, repeated indefinitely, results in a sustained performance of two FPops every five cycles:

```
fadd %f0,%f2,%f4
fmuld %f6,%f8,%f10
```

Using the performance equation from above results in:

Performance = 2/5 x clock frequency (MHz).

Table 3.4 summarizes the peak performance for the FPU.

Peak Performance	
33 MHz	16.0 Mflops
40 MHz	13.2 Mflops

Table 3.4 Peak Performance Summary

3.2 Internal Organization

This section describes the L64814 FPU architecture in more detail. It introduces the instruction pipeline which fetches, decodes, and controls execution of floating-point instructions. It also presents the datapath through the floating-point multiplier and adder. Figure 3.1 shows a detailed block diagram of the FPU with the instruction pipeline and datapath called out.

This section has the following organization.

3.2.1 Instruction Pipeline

3.2.2 Datapath

3.2.1 Instruction Pipeline describes the left side of Figure 3.1; **3.2.2 Datapath** describes the right side.

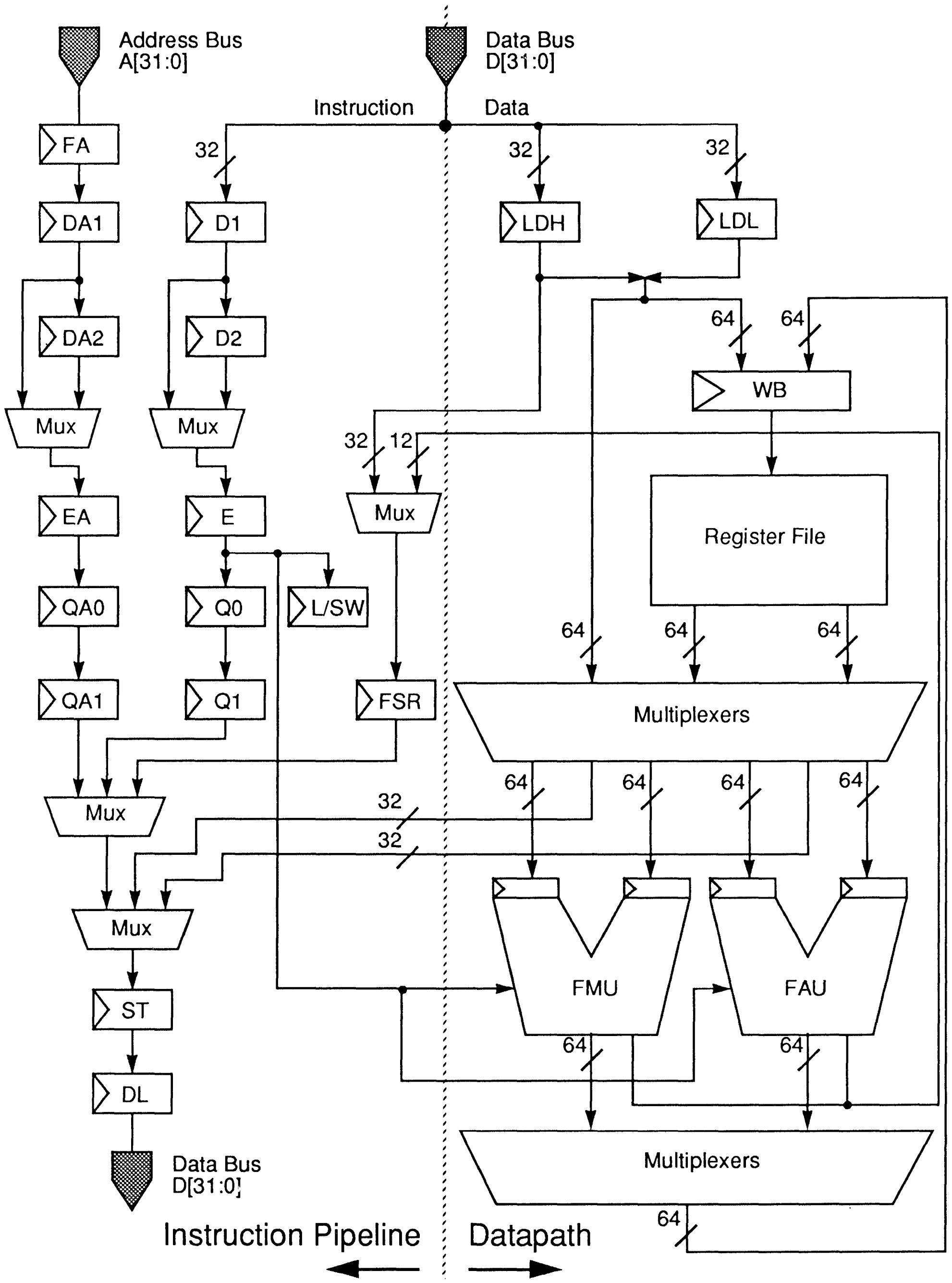


Figure 3.1 L64814 FPU Register-Level Block Diagram

3.2.1 Instruction Pipeline

The instruction pipeline controls floating-point instruction execution in the FPU. It accesses and decodes all instructions from the Data bus and all addresses from the Address bus. When a floating-point instruction occurs, the pipeline prepares it for execution and holds it during execution. It directs operands and manages the resources in the datapath.

Figure 3.2 isolates the instruction pipeline from Figure 3.1. The register naming conventions indicate which registers are involved in various stages of instruction execution. For example, registers D1 and DA1 hold one instruction and instruction address pair, respectively, for decoding. Registers D2 and DA2 hold a second instruction and instruction address pair for decoding. Similarly, E and EA hold an instruction and address for execution, and the registers with the Q prefix hold instructions and addresses in the floating-point queue. The L/SW (load/store word) register is a parallel path to the FQ; it is used for load and store instruction execution. The stages are discussed in more detail in Section 3.3.

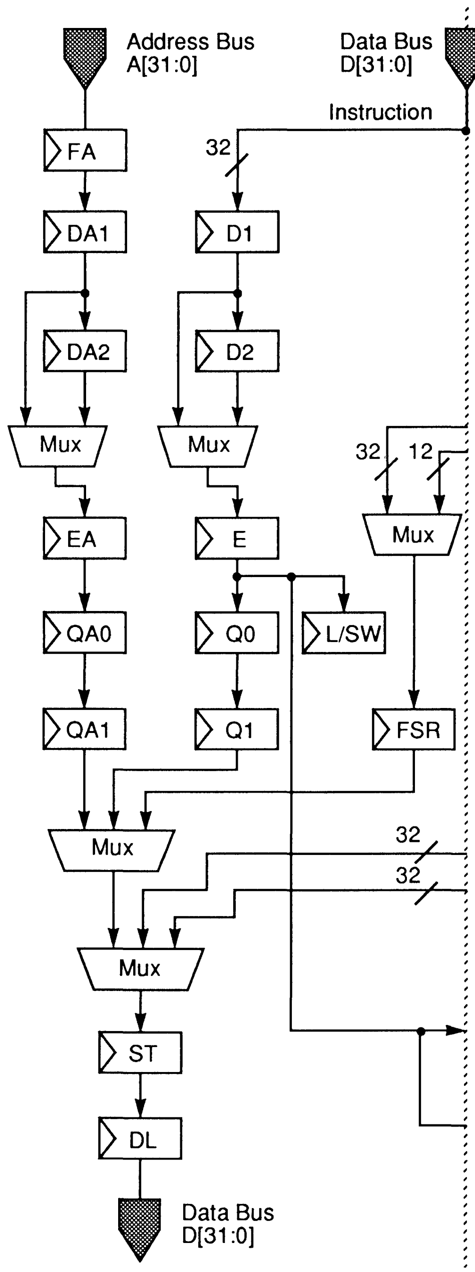


Figure 3.2 FPU Instruction Pipeline

3.2.2 Datapath

The FPU datapath includes not only the adder and multiplier, but also the Register File and a variety of multiplexers and registers. Figure 3.3 shows a more detailed block diagram of the FPU datapath. In the figure, the LD-prefix registers are used during load instructions to latch data from memory; for double-precision data, LDL latches the least-significant word (LSW), while LDH latches the most-significant word (MSW). Also, the Register File is shown partitioned into even and odd halves. Recall from Chapter 2 that double-precision data is stored in an even-odd pair of f-registers, with the MSW in the even register and the LSW in the odd register.

Figure 3.3 clearly illustrates that the FPU supports true double-precision calculations with a 64-bit-wide datapath. It also shows the adder (FAU) and multiplier (FMU) as distinct units which can operate in parallel for more efficient instruction execution. Note that the FPU can write data from the Data bus into the Register File and provide it to the multiplier or adder in the same clock cycle; Section 2.2 discusses this capability, called operand forwarding, in more detail.

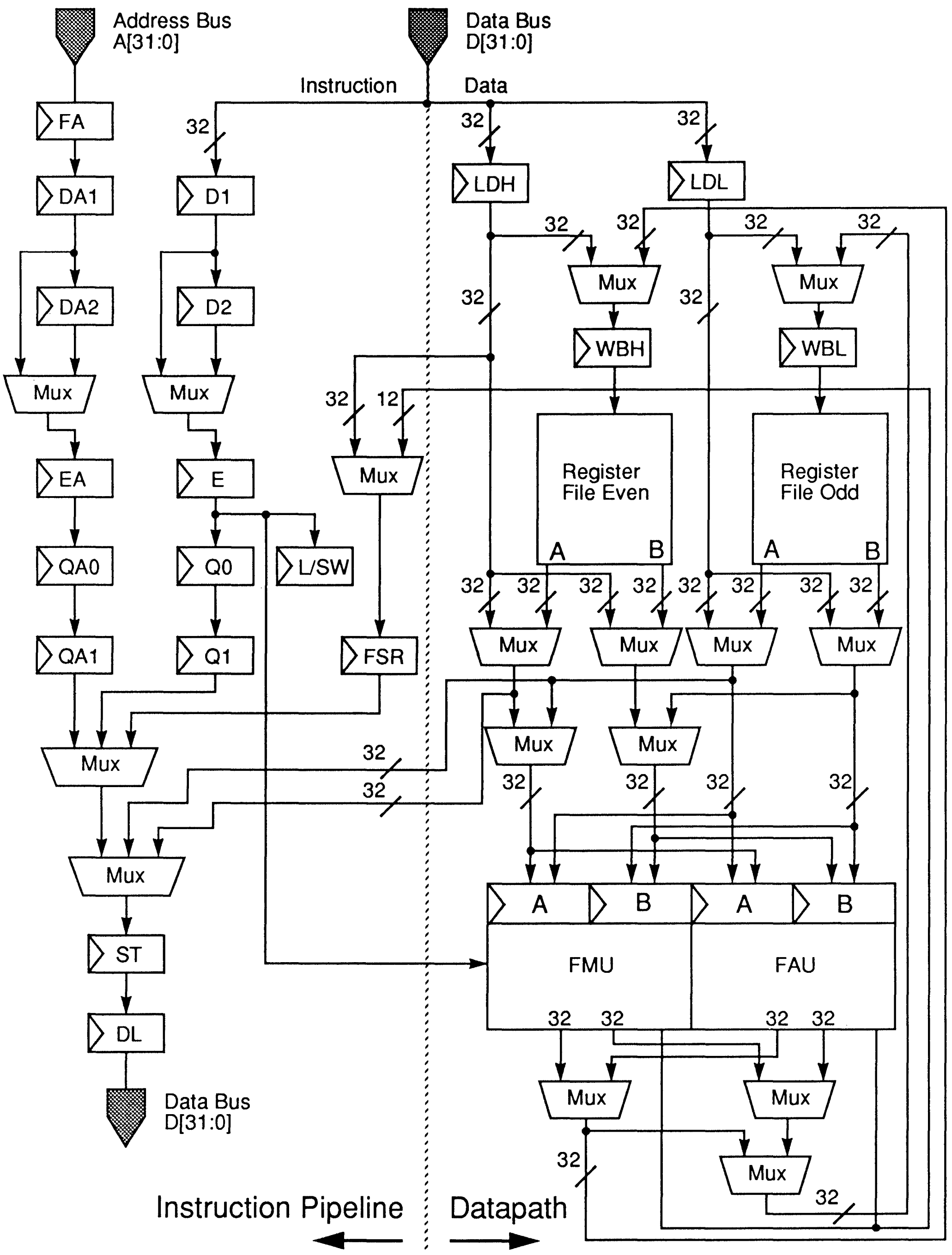


Figure 3.3 FPU Datapath Detailed Diagram

3.3 Instruction Execution

This section discusses in detail the various stages of floating-point instruction execution. It also describes the signals which control instruction execution, in particular those from the Integer Unit (IU).

This section has the following organization.

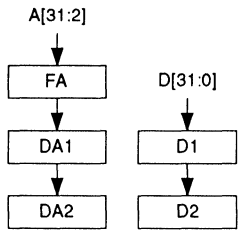
3.3.1 Fetch

When the IU fetches an instruction, the FPU captures it from the Data bus at the same time. The FPU has already captured the address corresponding to the instruction during the previous cycle. Specifically, the IU asserts the INST signal when a valid instruction is present on the Data bus and a valid address was fetched from the Address bus on the previous cycle; the FPU uses INST and the clock to determine when to capture the next instruction.

In any given cycle, the FPU saves the two most recent instruction/address pairs in the D and DA registers. The IU can select either of the two instructions for execution.

Figure 3.4a illustrates the fetch and decode stages of instruction execution in the FPU pipeline. When the IU decodes a valid floating-point instruction, it asserts either FINS1 or FINS2 to signal the FPU to start executing the instruction in D1 or D2, respectively. Figure 3.4b shows the timing for an instruction fetch, I2, which experiences a cache hit. The timing diagram illustrates the flow of data and addresses through each register. The actual transactions pictured on the Data and Address buses show the instruction fetches for I1 and I2, where I1 is not a floating-point instruction but I2 is and, as mentioned above, experiences a cache hit. Note that for this example, all signals which may hold or *freeze* the pipeline are inactive.

a) Register Configuration



b) Timing

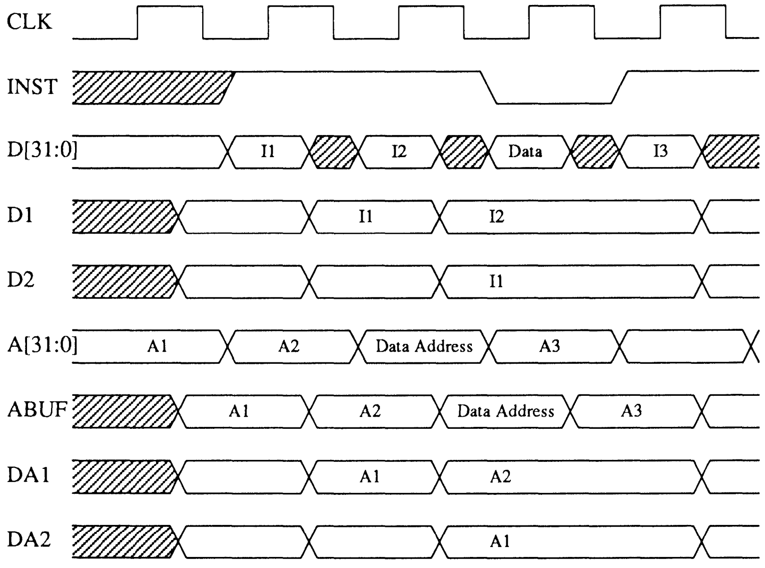


Figure 3.4 Instruction Fetch Timing (Cache Hit)

In the figure, INST stays high as long as valid instructions are available on the Data bus. INST goes low when data is available on the bus, to prevent the FPU from overwriting the instructions in D1 and D2 and their addresses in DA1 and DA2. Note, however, that because instruction I1 is not a floating-point instruction, both that instruction and its address are eventually written over in registers D2 and DA2.

When a cache miss occurs, the system drives low one of the memory hold signals, $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, or $\overline{\text{BHOLD}}$, in the cycle following the instruction fetch. The instruction, which the FPU captured from the Data bus, is invalid and the FPU replaces it when the system returns the valid instruction on the Data bus.

The memory hold signal remains active for several cycles, during which the system asserts $\overline{\text{MDS}}$ to notify the FPU that the valid instruction is available on the Data bus. On load instructions which experience a data cache miss, the system also asserts $\overline{\text{MDS}}$ to ensure that the instruction is reloaded only if an instruction cache miss occurred (and, therefore, INST was asserted) in the last non-hold cycle.

Figure 3.5 shows the same sequence of Data and Address bus transactions as Figure 3.4 except that the floating-point instruction I2 experiences an instruction cache miss. The $\overline{\text{HOLD}}$ signal in the figure represents one of the possible memory hold signals mentioned above; it goes low after the instruction cache miss. When $\overline{\text{MDS}}$ goes low, a valid instruction is available on the Data bus. Note that the data address which appears on the Address bus in the cycle following A2 reappears on that bus when the valid I2 becomes available.

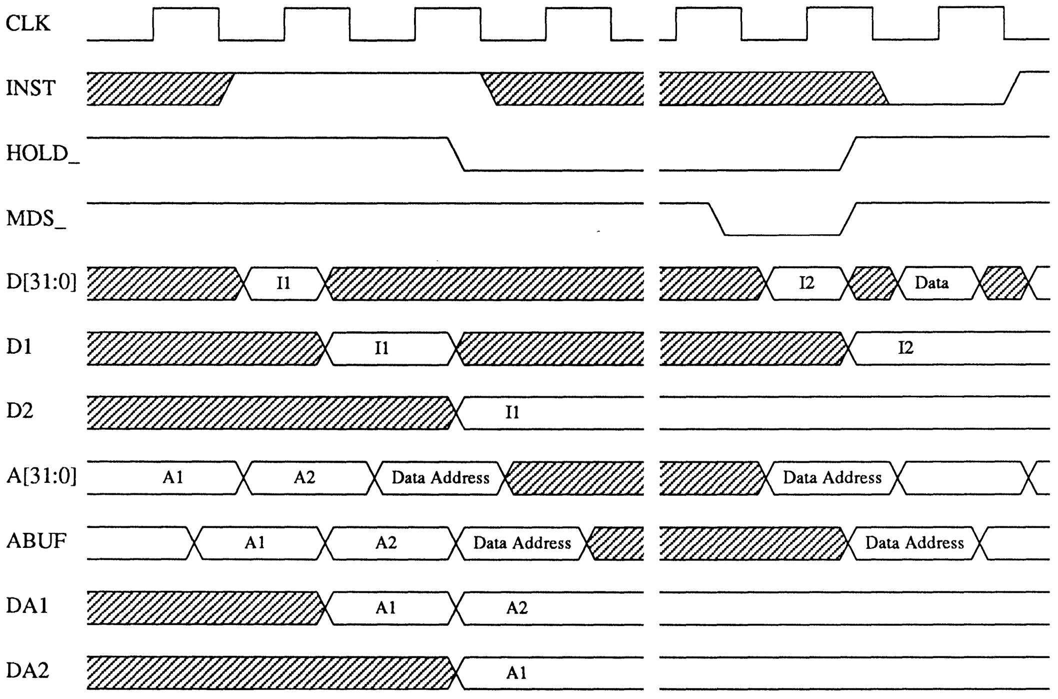


Figure 3.5 Instruction Fetch (Cache Miss on I2)

3.3.2 Decode

As mentioned previously, the FPU latches all valid instructions off the Data bus, not just floating-point instructions. The FPU then decodes part of the instruction to check for dependencies and to determine the instruction type. The FPU performs the remainder of the instruction decoding during the execute stage.

The instruction decoding which occurs prior to the execution stage makes the execution in the pipeline more efficient. Note that any non-floating-point instructions are over-written in the decode registers by succeeding instructions.

The FPU decodes instructions as specified in the *SPARC Architecture Manual.* Any FPop which is unimplemented, for example an extended-precision operation, and any opcode which is undefined are decoded as unimplemented. The IU handles all other illegal instructions, such as those with illegal opcodes which look like floating-point load or store instructions.

3.3.3 Execute

This section discusses floating-point instruction execution for these types of instructions:

FPops

Floating-Point Load and Store Instructions

Floating-Point Compare Instructions

FPops

The signals FINS1 and FINS2 from the IU notify the FPU when to start executing a floating-point instruction; at this time, the selected instruction is in one of the decode registers, either D1 or D2. Figure 3.6 illustrates an example where both D1 and D2 contain valid FPops, so the IU asserts both FINS1 and FINS2. A load instruction (I1) immediately precedes the two FPops, so both are fetched while the load instruction executes. Because the load instruction requires more than one cycle to execute, however, the IU defers starting execution on the FPops, as indicated by the help stages for the decode, execute, and write of I1. During these help stages, the FPU holds the FPops in the D registers.

When the first FPop (I2) enters its D-stage, the IU asserts FINS2 to start execution. Similarly, when the second FPop (I3), held in the register D1, enters the D-stage, the IU asserts FINS1 to start the FPU executing I3.

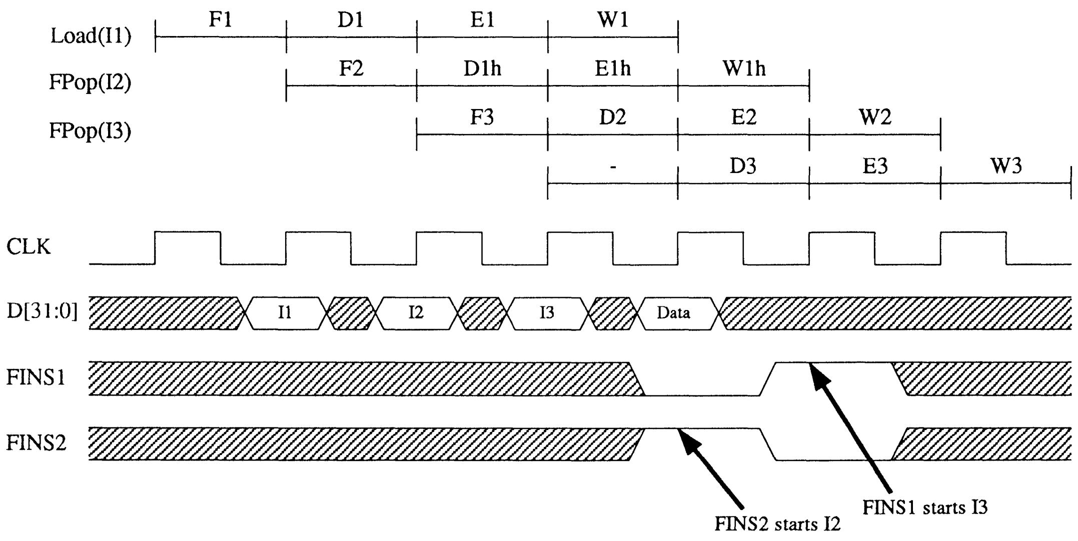


Figure 3.6 Dispatching Floating-Point Instructions

If an FPop passes the first cycle in its W-stage and the IU has not asserted FLUSH, then the instruction enters the floating-point queue (FQ). After an FPop enters the FQ, it executes until completion, even if a FLUSH occurs or a memory hold or other condition freezes the FPU pipeline, unless a trap occurs. Note that the W-stage of an FPop may extend to more than one cycle if a hold condition exists. When an FPop completes execution successfully and writes results to the Register File, then the FPop is removed from the FQ. The Q1 and QA1 registers always contain the instruction/address pair of the oldest FPop which the FPU is still executing.

Note that the IU never asserts FINS1 and FINS2 in the same cycle. Also, the FPU ignores FINS1 and FINS2 during any of these conditions:

- FLUSH is asserted
- $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$, or $\overline{\text{FHOLD}}$ is asserted
- FCCV or CCCV is asserted

Floating-Point Load and Store Instructions

Floating-point load and store instruction execution timing varies depending on whether or not the instructions and data are in the cache. Another factor which affects timing is whether the load or store data is integer or single-precision, or is double-precision.

The FPU convention is that double-precision load and store instructions present the most-significant word (MSW) first, followed by the least-significant word (LSW). This convention corresponds to the even-numbered f-register's data preceding the odd-numbered register's data, for an even-odd f-register pair which stores double-precision data.

Figure 3.7 and Figure 3.8 illustrate two examples of double-precision load instruction execution. Figure 3.7 shows a cache hit on both the MSW and LSW, while Figure 3.8 shows a cache miss on both words. Note that for both double-precision load and store instructions, cache misses may occur on either or both halves of the double-precision data. Also, single-precision and integer loads and stores obey the same cache hit and miss timing as the first word of the double-precision load or store.

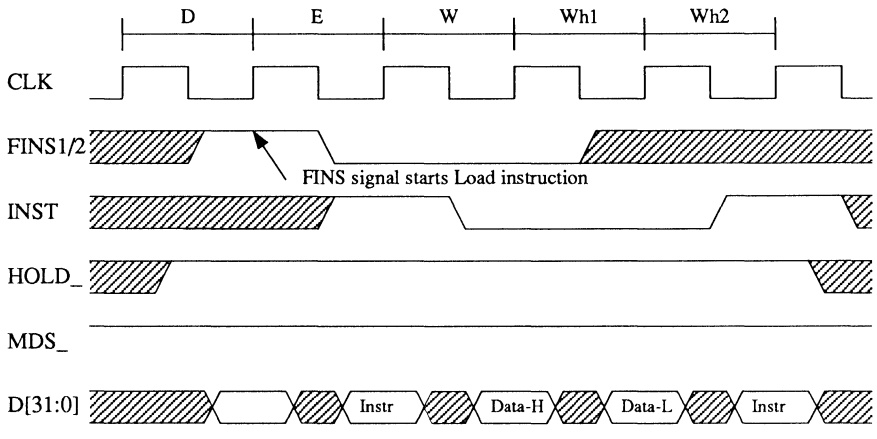


Figure 3.7 Double-Precision Load (Cache Hit on Both Words)

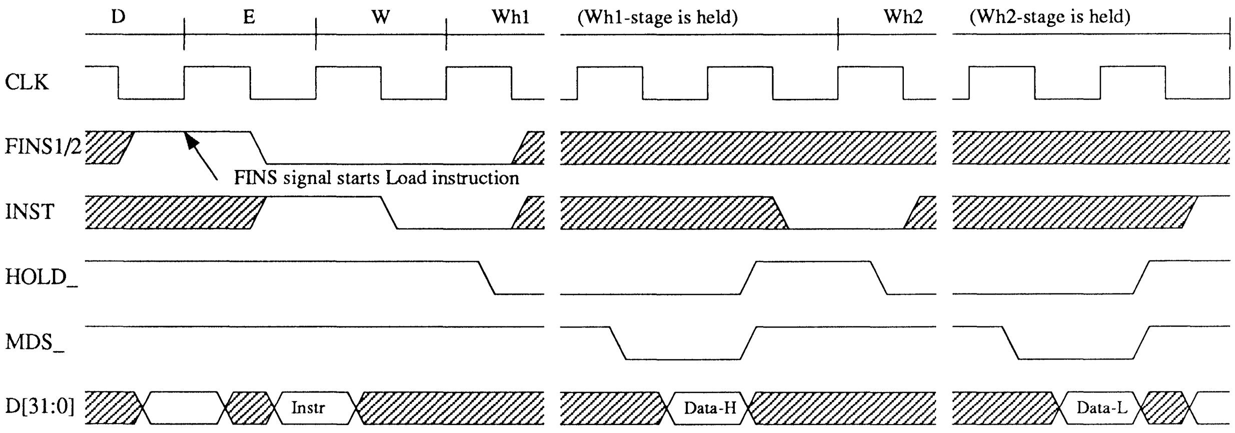


Figure 3.8 Double-Precision Load (Cache Miss on Both Words)

Store instructions drive data onto the Data bus referenced to the falling edge of the clock signal. The FPU drives the Data bus starting from the middle of the W-stage (that is, at the falling edge of the clock) in the store instruction. If the IU asserts FLUSH, then the FPU stops driving the Data bus by the middle of the next cycle.

The next three figures illustrate three examples of double-precision store instruction execution. Figure 3.9 shows a cache hit on both words, Figure 3.10 a cache miss on the instruction fetch preceding the store, and Figure 3.11 a cache miss on the MSW. Note in Figure 3.10 that D[31:0] represents data which the FPU puts out onto the Data bus.

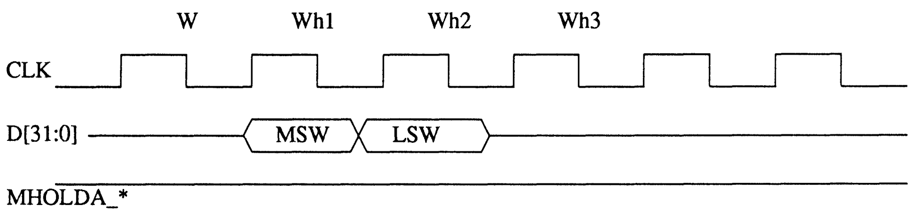


Figure 3.9 Double-Precision Store (Cache Hit on Both Words)

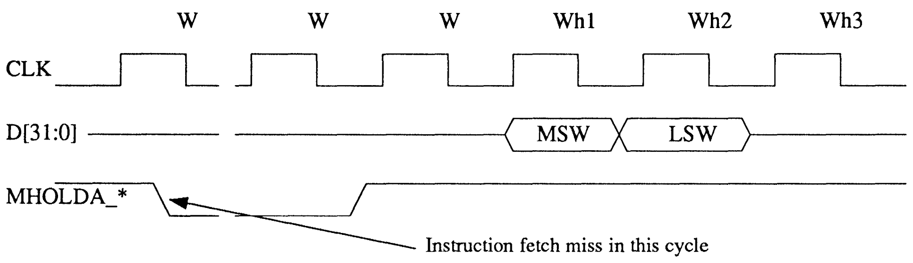


Figure 3.10 Double-Precision Store (Cache Miss on Instruction Fetch Preceding Store)

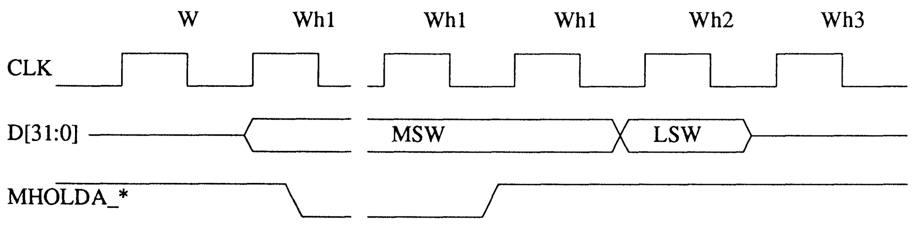


Figure 3.11 Double-Precision Store (Cache Hit on MSW)

Floating-point load and store instructions do not follow the same path as FPops through the floating-point queue. As mentioned previously, load and store instructions do not enter the FQ. Instead, to allow the FPU to perform load/store instructions and FPops in parallel, load and store instructions move from the E-registers directly into the L/SW (load/store word) register, a path parallel to the FQ, to complete execution. Refer to Figure 3.1 to see the FQ and L/SW register paths.

Floating-Point Compare Instructions

When a floating-point compare instruction occurs, the FPU deasserts the FCCV (floating-point condition code valid) signal to freeze the instruction pipeline, starting in the E-stage of the instruction following the compare instruction. This instruction holds in its E-stage, FCCV remains deasserted, and the instruction pipeline stays frozen until the floating-point condition codes (FCC[1:0]) become valid. One cycle later, the FPU reasserts FCCV.

All floating-point compare instructions cause this behavior, whether implemented or unimplemented. For unimplemented compare instructions, the FPU freezes the instruction pipeline and causes an unimplemented FPop trap which the IU immediately takes. For more information about trap handling, refer to Section 3.3.

Figure 3.12 illustrates the FCCV timing relative to the floating-point compare instruction (FCMP in the figure) and the condition codes, FCC[1:0].

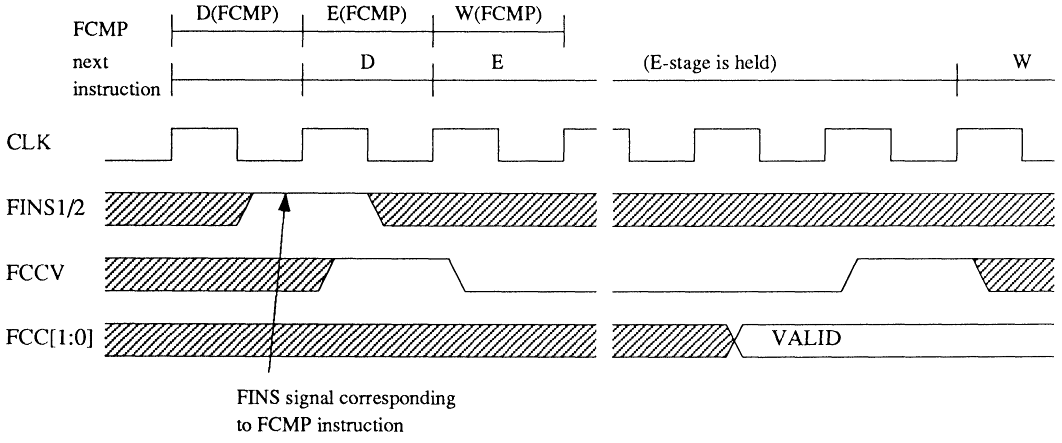


Figure 3.12 Floating-Point Compare Instruction Timing

3.4 Exception Handling

This section discusses how the FPU handles exceptions, also referred to as traps. It describes the FPU modes of operation and what causes the FPU to change operating modes. It also provides details on flushing the floating-point queue during trap handling.

The section has this organization:

3.4.1 FPU Modes of Operation

The FPU has three possible modes of operation:

- execution
- pending exception
- exception

Figure 3.13 shows the FPU operating modes and the transitions between modes.

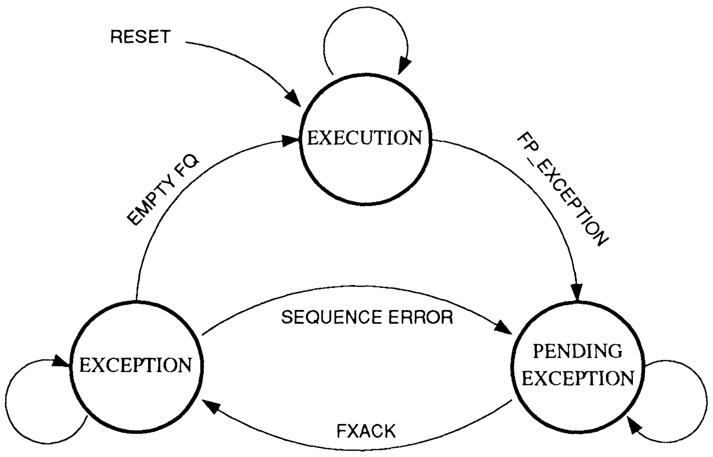


Figure 3.13 FPU Modes of Operation

During normal instruction execution, the FPU operates in **execution mode**. When a floating-point exception occurs, the FPU asserts $\overline{\text{FEXC}}$ to notify the IU about the exception and to direct the IU to take the trap. The FPU then enters **pending exception mode**.

When the IU encounters the next floating-point instruction in the instruction stream, it takes the trap and asserts FXACK to notify the FPU. The FPU then enters **exception mode**.

When the IU takes a trap, it halts normal program execution and transfers control to the trap handler. The FPU aborts the instruction in the E-stage of the floating-point pipeline and any instructions after it; the IU restarts these instructions after the trap handler is done. Any FPops which entered the pipeline prior to this instruction and which are in the floating-point queue complete execution.

Figure 3.14 shows the instruction pipeline during a trap. Note that the IU asserts the FLUSH signal in the W-stage of the aborted instruction.

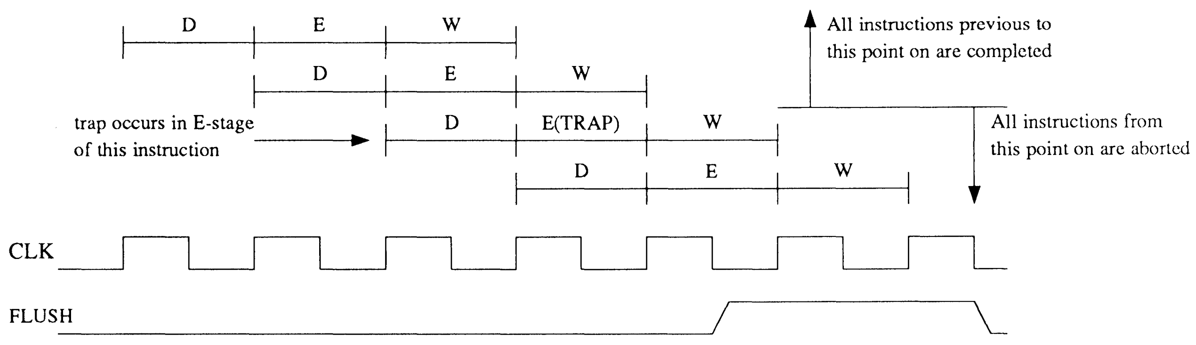

D
E
W
D
E
W
trap occurs in E-stage of this instruction
D
E(TRAP)
W
D
E
W
All instructions previous to this point on are completed
All instructions from this point on are aborted
CLK
FLUSH

Figure 3.14 FPU Instruction Pipeline during a Trap

In exception mode, the FPU performs store instructions from the trap handler to empty or **flush** the FQ; flushing is discussed later in this section. The trap handler then performs the appropriate actions to handle the particular type of trap. For example, if an unimplemented instruction causes the trap, then the trap handler emulates the instructions from the queue as well as the unimplemented instruction.

Note that when the FPU is in exception mode, if the IU issues a floating-point load instruction or FPop, a sequence error occurs. The FPU can only perform store floating-point queue (STDFQ) instructions until it empties the queue and the trap handler is done.

Figure 3.15 summarizes the handshake which the IU and FPU perform when a floating-point exception occurs.

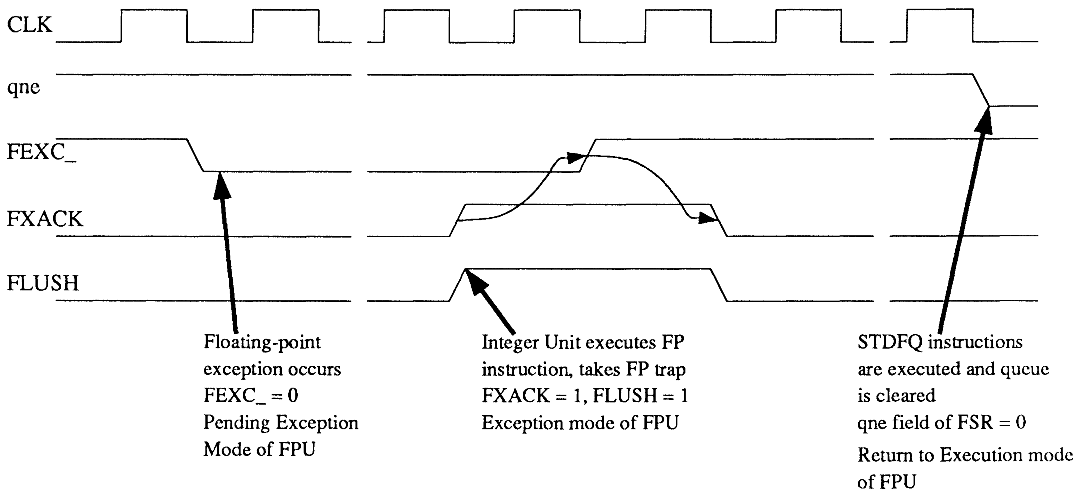

CLK
qne
FEXC_
FXACK
FLUSH
Floating-point exception occurs FEXC_ = 0 Pending Exception Mode of FPU
Integer Unit executes FP instruction, takes FP trap FXACK = 1, FLUSH = 1 Exception mode of FPU
STDFQ instructions are executed and queue is cleared qne field of FSR = 0 Return to Execution mode of FPU

Figure 3.15 FPU/IU Exception Handshaking Sequence

3.4.2 Flushing Floating-Point Instructions

This discussion explains and illustrates the effect which a flush has on these types of floating-point instructions:

Floating-Point Load Instructions

Floating-Point Store Instructions

FPops

Floating-Point Compare (FCMP) Instructions

Floating-Point Load Instructions

If the IU asserts the FLUSH signal any time before or during the last Wh-stage of a load instruction, then the load aborts and leaves the contents of the Register File unchanged. Figure 3.16 shows the effect of FLUSH on a floating-point load (LDF) instruction.

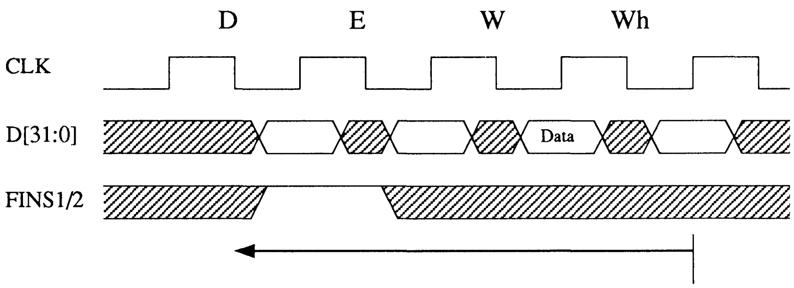


Figure 3.16 Effect of FLUSH on a Floating-Point Load Instruction

Floating-Point Store Instructions

If the IU asserts FLUSH any time before or during the last Wh-stage of a floating-point store (STF) instruction, then the store aborts and the FPU stops driving the Data bus by the middle of the next clock cycle. Figure 3.17 illustrates the effect of FLUSH on a floating-point store instruction.

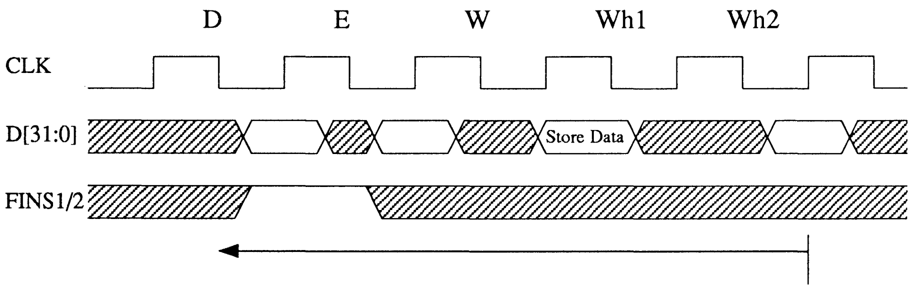


Figure 3.17 Effect of FLUSH on a Floating-Point Store Instruction

FPops

If the IU asserts FLUSH any time before or during the W-stage of an FPop, then the FPop aborts, leaving the contents of the Register File and FSR unchanged. Figure 3.18 shows the effect of FLUSH on FPop instruction execution.

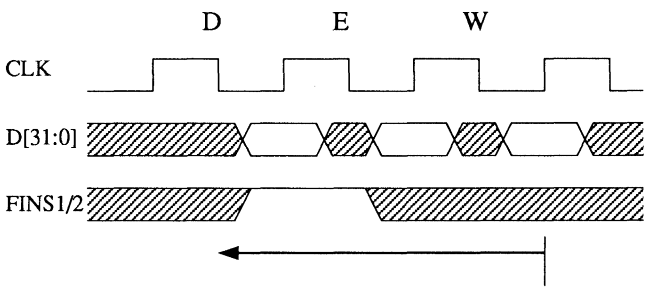


Figure 3.18 Effect of FLUSH on an FPop Instruction

Floating-Point Compare (FCMP) Instructions

If the IU asserts FLUSH before or during the W-stage of an FCMP instruction, then the FCMP aborts and leaves the FSR unchanged. The FPU reasserts FCCV on the next clock cycle. Figure 3.19 illustrates the effect of FLUSH on FCMP instruction execution and on FCCV.

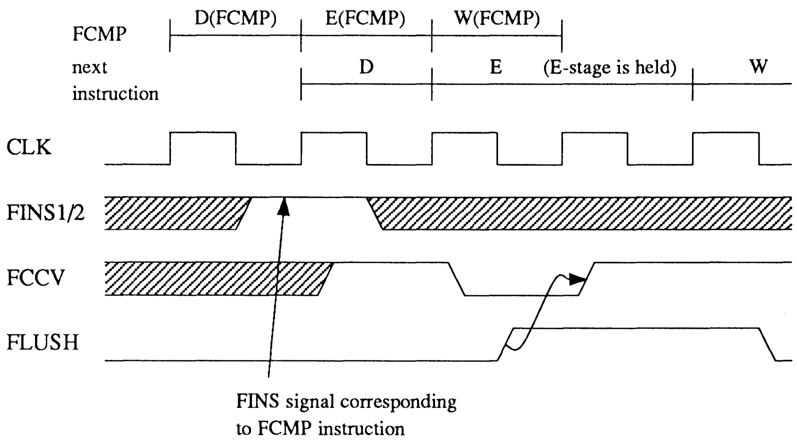


Figure 3.19 Effect of FLUSH on a Floating-Point Compare Instruction and FCCV

3.5 Halting Instruction Execution

This section discusses the signals and conditions which can halt instruction execution in both the FPU and the IU. The section is organized this way.

3.5.1 Freezing the FPU Pipeline describes the various memory hold and other signal associated with halting floating-point instruction execution.

3.5.2 Interlocking the IU Pipeline explains the situations which cause the FPU to halt IU instruction execution.

3.5.1 Freezing the FPU Pipeline

These signals can all freeze the FPU instruction pipeline and halt instruction execution:

- $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$ from the memory subsystem
- $\overline{\text{CHOLD}}$ or CCCV from a coprocessor
- $\overline{\text{FHOLD}}$ or FCCV from the FPU itself

A pipeline freeze stops execution of all load/store instructions, and of all FPops which are in the F-, D-, or E-stages of the pipeline. These instructions may continue execution when all of the hold signals ($\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$, and $\overline{\text{FHOLD}}$) are inactive and all the condition code valid signals (CCCV, FCCV) are active. Note that a pipeline freeze does not affect FPops already in the queue; they continue execution.

The system needs to know when the FPU is freezing the instruction pipeline, so that it stops issuing instructions to the IU. Instead of looking at both $\overline{\text{FHOLD}}$ and FCCV, the two FPU signals which may hold the floating-point pipeline, the system simply examines the FNULL signal.

The FPU asserts FNULL whenever it freezes the pipeline; specifically, the FPU asserts FNULL whenever it asserts $\overline{\text{FHOLD}}$ or deasserts FCCV. Figure 3.20 shows the FNULL timing relative to $\overline{\text{FHOLD}}$ and FCCV. Note in the figure that although $\overline{\text{FHOLD}}$ and FCCV are referenced to the falling edge of the clock, FNULL is referenced to the rising edge of the clock.

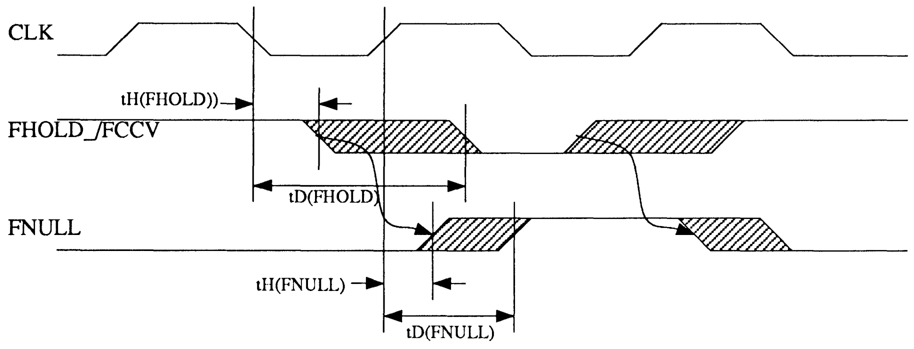


Figure 3.20 FNULL Timing

3.5.2 Interlocking the IU Pipeline

In two types of situations, the FPU must halt the IU instruction pipeline: first, when the FPU must hold a load/store instruction due to an operand dependency, and second, when the FPU cannot accept any more instructions due to a resource dependency. In either case, the FPU asserts $\overline{\text{FHOLD}}$ to freeze the IU instruction pipeline.

Table 3.5 shows the operand dependency conditions under which the FPU must assert $\overline{\text{FHOLD}}$.

Instruction	Action	Operand Dependency
LDF, LDDF	Load memory to Register File	Load instruction may not overwrite any source or destination register of any FPop which has not completed execution. Specifically, the rd (destination register) field in any load instruction cannot refer to the same f-register as any valid rs1 or rs2 (source register) or rd (destination register) field in any outstanding FPop. The source registers must remain unaltered in case a floating-point exception occurs where the trap handler would require the original source register values.
STF, STDF	Store data from the f-register to memory	A store instruction may not access an f-register that is the destination register of an FPop which has not yet finished execution. In this case, the store instruction must hold until all outstanding FPops with that register as a destination complete execution.
LDFSR, STFSR	Load/Store data between memory and the FSR	The FPU cannot perform an LDFSR or STFSR while the FPU is executing any other instruction, because that instruction may need to utilize or change some value in the FSR. Therefore, if the FPU is executing any instruction when the IU issues a LDFSR or STFSR, the FPU holds until all instructions in the queue complete execution and the queue is empty.

Table 3.5 Operand Dependencies which Halt the IU Instruction Pipeline

The operand dependencies of Table 3.5 apply to all FPops which are defined in the *SPARC Architecture Manual*, including those which are unimplemented in the FPU. For example, suppose that an unimplemented FPop is in the FQ, waiting to cause an exception. If the next instruction is a floating-point store instruction which needs to store the contents of the unimplemented FPop's destination register, then the store must cause an $\overline{\text{FHOLD}}$ so that it does not store the incorrect data. The unimplemented FPop eventually causes a trap, which the IU takes during the E-stage of the store instruction.

With regard to resource dependencies, when the FQ is full the FPU cannot accommodate any additional FPops. The FPU asserts $\overline{\text{FHOLD}}$ when the FQ is full and the IU tries to signal an additional FPop, to stop the IU from issuing any more instructions to the FPU. Specifically, when the FQ is full the FPU asserts $\overline{\text{FHOLD}}$ if the IU asserts either FINS1 or FINS2 and the incoming instruction is an FPop.

If the FPU goes into exception mode, it deasserts $\overline{\text{FHOLD}}$. In exception mode, the only case where the FPU asserts FHOLD is if a sequence error occurs. In this case, the FPU asserts $\overline{\text{FHOLD}}$ for one cycle.

If a floating-point exception occurs while the FPU is asserting $\overline{\text{FHOLD}}$, then the FPU deasserts $\overline{\text{FHOLD}}$ at least one cycle after it asserts $\overline{\text{FEXC}}$. In this case only, the IU takes the floating-point trap on the instruction which triggered the $\overline{\text{FHOLD}}$. Note that if asserting $\overline{\text{FEXC}}$ did not remove the hold, then a deadlock would occur.

Figure 3.21 shows the interaction of $\overline{\text{FEXC}}$ with $\overline{\text{FHOLD}}$ and FCCV. Note that $\overline{\text{FHOLD}}$ is referenced to the falling edge of the clock.

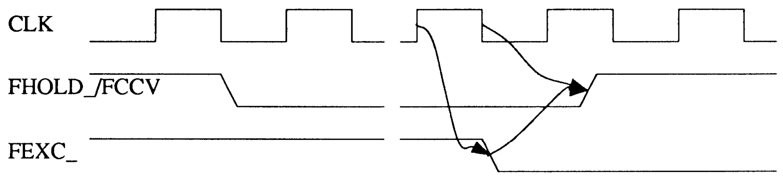


Figure 3.21 Effect of $\overline{\text{FEXC}}$ on $\overline{\text{FHOLD}}$ and FCCV

Chapter 4: External Interface

This chapter discusses the L64814 FPU in a system context. It shows the interface configurations and describes the interface signals. The chapter has the following organization.

> **4.1 Interface Overview** illustrates how the FPU interfaces with the other members of the SPARC family.
>
> **4.2 Pin Summary** classifies the interface signals between the FPU and the IU, memory subsystem, and coprocessor.
>
> **4.3 Pin Description** provides a complete pin list and description.
>
> **4.4 AC Timing** illustrates the AC timing characteristics for the FPU.
>
> **4.5 Electrical Requirements** lists the electrical specifications for the FPU.
>
> **4.6 Packaging** explains the package options for the FPU.

4.1 Interface Overview

Figure 4.1 shows how to connect the FPU with the L64811 IU and L64815 MCT in a system configuration. These interfaces are very simple because the devices can directly interconnect. For more specific information on the interconnect signals, refer to the next two sections in this chapter.

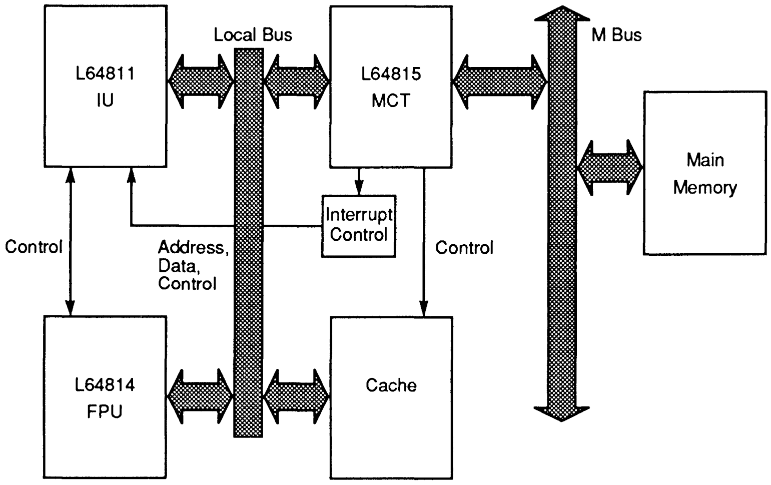


Figure 4.1 FPU-IU-MCT Interconnect Diagram

4.2 Pin Summary

This section lists and classifies the FPU signals according to pin function and pin type.

The FPU has 83 signal pins, in these I/O categories:

- 44 input signals
- 7 output signals
- 32 bidirectional signals

These signals can be further classified according to use:

- 13 signals between the FPU and the IU/coprocessor
- 70 signals between the FPU and the system/memory

Of these 83 signals, a maximum of 39 may be driving at the same time.

Table 4.1 summarizes the pin function and pin type for the IU and coprocessor interface signals; Table 4.2 does the same for the system/memory interface.

Pin Name	Pin Description/Function	Pin Type
FCCV	Floating-point condition code valid	Output/2-state
FCC[1:0]	Floating-point condition code bits	Output/2-state
$\overline{\text{FEXC}}$	Floating-point exception	Output/2-state
$\overline{\text{FHOLD}}$	Floating-point hold	Output/2-state
$\overline{\text{FP}}$	Floating-point unit present	Output/2-state
FINS1	Floating-point instruction fetched	Input/2-state
FINS2	Floating-point instruction fetched	Input/2-state
INST	Instruction fetch cycle	Input/2-state
FLUSH	Flush the FPU instruction pipeline	Input/2-state
FXACK	Floating-point exception acknowledge	Input/2-state
CCCV	Coprocessor condition valid	Input/2-state
$\overline{\text{CHOLD}}$	Coprocessor hold	Input/2-state

Table 4.1 IU/Coprocessor Interface Signals

Pin Name	Pin Description/Function	Pin Type
D[31:0]	Data bus	Bidir/3-state
A[31:2]	Address bus	Input/2-state
$\overline{\text{MHOLDA}}$	Hold signal from memory	Input/2-state
$\overline{\text{MHOLDB}}$	Hold signal from memory	Input/2-state
$\overline{\text{BHOLD}}$	Hold signal from memory	Input/2-state
$\overline{\text{MDS}}$	Memory data strobe	Input/2-state
$\overline{\text{DOE}}$	Disable Data bus output drivers	Input/2-state
FNULL	Signal FPU hold of instruction pipeline	Output/2-state
$\overline{\text{TOE}}$	Test output enable	Input/2-state
$\overline{\text{RESET}}$	System reset	Input/2-state
CLOCK	FPU clock	Input/2-state

Table 4.2 System/Memory Interface Signals

4.3 Pin Description

This section lists and describes the FPU signals.

A[31:2] **Address Bus[31:2]**

A[31:0] comprise the Address bus common to the FPU, IU, and memory subsystem. From this bus, the FPU latches the instruction address for each instruction fetched. Because instructions are stored on 32-bit boundaries, it is unnecessary to fetch the two lowest-order bits of the address, A[1:0].

CCCV **Coprocessor Condition Codes Valid**

The coprocessor uses this signal to notify the IU and FPU when the coprocessor condition codes are valid. When CCCV is deasserted, the instruction pipeline freezes until the instruction that generates the coprocessor condition codes completes execution and the codes become valid.

$\overline{\text{CHOLD}}$ **Coprocessor Hold**

When the coprocessor detects a condition where it cannot accept any more instructions, it asserts this signal to freeze the instruction pipeline. This signal is similar to the $\overline{\text{FHOLD}}$ signal generated by the FPU.

CLK	**Clock**	This signal provides the system clock to the FPU.
$\overline{\text{DOE}}$	**Data Bus Driver Output Enable**	Deasserting this signal turns off the FPU Data bus drivers. The system deasserts this signal when another bus master needs to use the Data bus.
D[31:0]	**Instruction/Data Bus [31:0]**	D[31:0] comprise the instruction/data bus common to the FPU, IU, and memory subsystem. The FPU fetches all instructions from this bus. In addition, on floating-point load/store instructions, the FPU receives/sends data from/to memory on this bus.
FCCV	**Floating-Point Condition Codes Valid**	The FPU asserts this signal when the floating-point condition codes, FCC[1:0], become valid. When the FPU deasserts this signal, the instruction pipeline freezes.
FCC[1:0]	**Floating-Point Condition Codes [1:0]**	These signals are the FPU condition code; they are valid only when FCCV is asserted. During the execution of a Branch on floating-point condition code (Bfcc) instruction, the IU uses these bits to make branching decisions. These signals are the same as the FCC field of the FSR.
$\overline{\text{FEXC}}$	**Floating-Point Exception**	The FPU asserts this signal to notify the IU that a floating-point exception has occurred and that the IU should take the trap. The signal remains asserted until the IU acknowledges that it has taken the trap by asserting FXACK.
$\overline{\text{FHOLD}}$	**Floating-Point Hold**	The FPU asserts this signal when it cannot accept any more floating-point instructions due to resource or data dependencies. The FPU deasserts the signal when the dependency is removed.
FINS1	**Floating-Point Instruction Select 1**	The IU asserts this signal during the decode stage of a floating-point instruction to notify the FPU that it should start executing the last instruction fetched.
FINS2	**Floating-Point Instruction Select 2**	The IU asserts this signal during the decode stage of a floating-point instruction to notify the FPU that it should start executing the second-to-last instruction fetched.

FLUSH **FPU Instruction Pipeline Flush**

The IU asserts this signal to notify the FPU to abort the floating-point instructions that are still in the pipeline and have not yet entered the queue. The IU typically asserts this signal when it takes a trap, and it restarts the aborted instructions after the trap handler completes execution. Instructions which are already in the queue complete execution.

FNULL **Floating-Point Null**

The FPU asserts this signal to notify the memory subsystem that the FPU is freezing the instruction pipeline. It asserts FNULL whenever it asserts $\overline{\text{FHOLD}}$ or deasserts FCCV. The memory system uses FNULL in the same fashion as the IU's NULL_CYC signal; it needs the additional signal because NULL_CYC does not take into account FPU holds.

$\overline{\text{FP}}$ **Floating-Point Unit**

This signal tells the IU that a floating-point unit is present in the system. The signal typically has a pullup resistor holding it high at the IU input; when the FPU is plugged into the board, the FPU pulls the signal low.

FXACK **Floating-Point Exception Acknowledge**

The IU asserts FXACK to signal the FPU that is has taken the requested floating-point exception trap. In response, the FPU deasserts $\overline{\text{FEXC}}$.

INST **Instruction**

The IU asserts this signal when it is fetching a new instruction; it signals the FPU to copy the instruction being fetched.

$\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$

Memory Hold

The memory subsystem asserts these signals to freeze the instruction pipeline.

$\overline{\text{MDS}}$ **Memory Data Strobe**

The memory subsystem asserts this signal to strobe an instruction or data into the FPU during a cache miss situation. One or more memory hold signals are active at the same time.

$\overline{\text{RESET}}$ **System Reset**

The system asserts this pin to reset the FPU.

$\overline{\text{TOE}}$ **Test Output Enable**

For chip and board test, tying this pin HIGH three-states all of the FPU output drivers, including the D bus.

4.4 AC Timing

This section provides the AC timing characteristics for the FPU. It includes timing tables for the IU/ coprocessor interface signals, the system/memory interface and miscellaneous signals, and the clock input specification. Following the timing tables are timing diagrams which illustrate the timing parameters in the tables.

This section uses the following notation.

- Tdo: propagation delay time of an output referenced to a given clock edge
- Tho: hold time of an output referenced to the given clock edge
- Tsi: setup time of an input referenced to the given clock edge
- Thi: hold time of an input referenced to the given clock edge
- Toff: turn-off time for a 3-state output driver after the rising edge of $\overline{\text{DOE}}$
- Ton: turn-on time for a 3-state output driver after the falling edge of $\overline{\text{DOE}}$
- CLK+: rising edge of the clock signal CLK
- CLK-: falling edge of the clock signal CLK

Please also note the following information:

- All times are in ns.
- Output loading is assumed to be 50 pF for signals driving to the system and 25pF for signals driving to the IU.
- Minimum output loading (for minimum time calculations) is assumed to be 15 pF.
- Clock references are made with respect to the 1.5 V level of the clock.

Pin Name	Parameter	Parameter Number	Min/Max	25 MHz	33 MHz	40 MHz	Units
Clock Period	Tcyl	1	Min	40	30	25	ns
Clock High Time	Tclh	2	Min	18	13	11	ns
Clock Low Time	Tcll	3	Min	18	13	11	ns
Clock Rise Time	Tcrt	-	Max	1	1	1	V/ns
Clock Fall Time	Tcft	-	Max	1	1	1	V/ns

Table 4.3 AC Timing: Clock Input Specification

Pin Name	Parameter	Parameter Number	Reference Edge	Min/Max	25 MHz (ns)	33 MHz (ns)	40 MHz (ns)
$\overline{\text{FHOLD}}$, FCCV	Tdo	4	CLK-	Max	30	24	20
	Tho	5	CLK-	Min	6	6	4
FCC[1:0], FEXC	Tdo	6	CLK+	Max	27	21	18
	Tho	7	CLK+	Min	5	5	2
FINS1, FINS2	Tsi	8	CLK+	Min	10	10	9
	Thi	9	CLK+	Min	3.5	3.5	3
FXACK, FLUSH, INST	Tsi	10	CLK+	Min	17	14	11
	Thi	11	CLK+	Min	3	3	3
CCCV, $\overline{\text{CHOLD}}$	Tsi	12	CLK-	Min	7	5	4
	Thi	13	CLK-	Min	7	5	4

Table 4.4 AC Timing: IU and Coprocessor Signals

Pin Name	Parameter	Parameter Number	Reference Edge	Min/Max	25 MHz (ns)	33 MHz (ns)	40 MHz (ns)
FNULL	Tdo	14	CLK+	Max	20	15	13
	Tho	15	CLK+	Min	4	4	3
D[[31:0] out	Tdo	16	CLK-	Max	20	15	13
	Tho	17	CLK-	Min	4	4	3
D[[31:0] in	Tsi	18	CLK+	Min	3	3	2
	Thi	19	CLK+	Min	5	5	4
A[31:2]	Tsi	20	CLK+	Min	5	3	3
	Thi	21	CLK+	Min	7	7	7
$\overline{\text{MHOLDA}}$,	Tsi	22	CLK-	Min	7	5	4
$\overline{\text{MHOLDB}}$,	Thi	23	CLK-	Min	7	5	4
$\overline{\text{BHOLD}}$							
$\overline{\text{MDS}}$	Tsi	24	CLK-	Min	5	5	4
	Thi	25	CLK-	Min	7	5	4
$\overline{\text{RESET}}$	Tsi	26	CLK-	Min	15	10	8
	Thi	27	CLK-	Min	3	3	2
$\overline{\text{DOE}}$	Ton	28	CLK+	Min	20	17	15
	Toff	29	CLK+	Min	20	17	15

Table 4.5 AC Timing: System/Memory Interface Signals

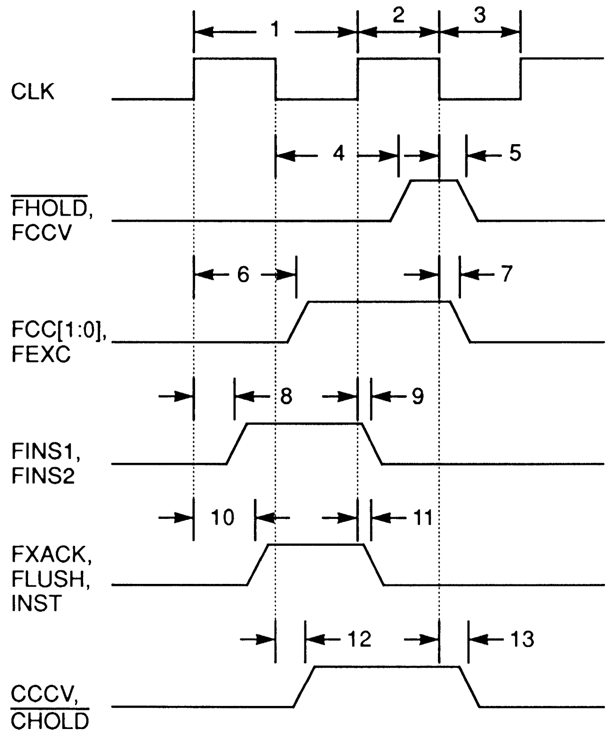


Figure 4.2 AC Timing Parameters: IU and Coprocessor Signals

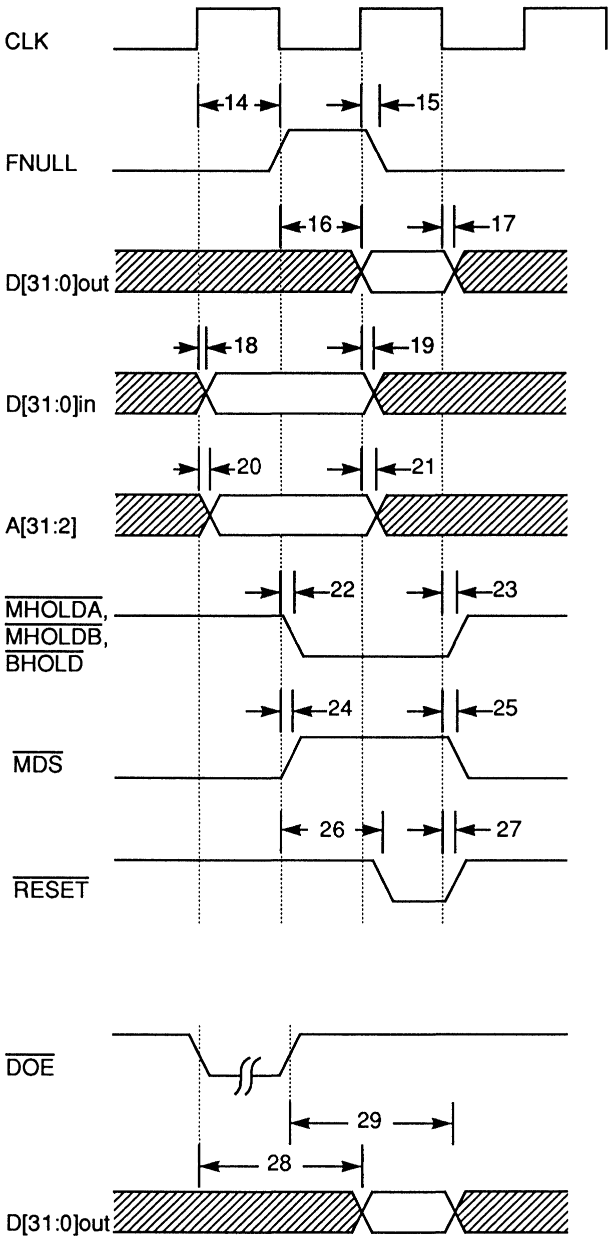


Figure 4.3   AC Timing: System/Memory Interface Signals

4.5 Electrical Requirements

This section includes the following specifications for the L64814:

- Absolute Maximum Ratings (Table 4.6)
- Recommended Operating Conditions (Table 4.7)
- DC Characteristics (Table 4.8)
- Capacitance (Table 4.9)

Symbol	Parameter	Limits[1]	Unit
V_{DD}	DC Supply	-0.3 to +7	V
V_{IN}	Input Voltage	-0.3 to V_{DD} +0.3	V
I_{IN}	DC Input Current	±10	mA
T_{STG}	Storage Temperature Range (Plastic)	-40 to +125	°C
T_{STG}	Storage Temperature Range (Ceramic)	-65 to +150	°C
Note 1. Referenced to V_{SS}			

Table 4.6 Absolute Maximum Ratings

Symbol	Parameter	Limits	Unit
V_{DD}	DC Supply	+4.75 to +5.25	V
T_A	Ambient Temperature	-0 to +70	°C

Table 4.7 Recommended Operating Conditions

Symbol	Parameter	Condition[1]	Min	Typ	Max	Units
V_{IL}	Voltage Input LOW				0.8	V
V_{IH}	Voltage Input HIGH		2.0			V
V_{OH}	Voltage Output HIGH	I_{OH} = -8.0 mA	2.4	4.5		V
V_{OL}	Voltage Output LOW	I_{OL} = 8.0 mA		0.2	0.4	V
I_{IH}	Current Input HIGH	$V_{IN} = V_{DD}$			10	µA
I_{IL}	Current Input LOW	$V_{IN} = V_{SS}$			-10	µA
I_{OH}	Current Output HIGH	V_{OH} = 2.4 V	-4.0			mA
I_{OL}	Current Output LOW	V_{OL} = 0.4 V	4.0			mA
I_{OZ}	Current 3-State Output Leakage	$V_{OH} = V_{DD}$ or V_{SS}	-10	±1	10	µA
I_{OS}	Current Output Short Circuit	V_{DD} = Max, output shorted to V_{DD}	15	50	130	mA
		V_{DD} = Max, output shorted to V_{SS}	-5	-25	-150	mA
I_{DD}	Quiescent Supply Current	$V_{IN} = V_{DD}$ or V_{SS}			10	mA
I_{CC}	Dynamic Supply Current	f = 10 MHz at 5.25 V f = 33 MHz at 5.25 V f = 40 MHz at 5.25 V			90 300 380	mA mA mA
Note 1. Specified at V_{DD} equals 5V ±5%; ambient temperature over the specified range.						

Table 4.8 DC Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
C_{IN}	Input Capacitance	V_{IN} = 5.0 V, T_A = 25° C, f = 1 MHz			10	pF
C_{OUT}	Output Capacitance	V_{IN} = 5.0 V, T_A = 25° C, f = 1 MHz			12	pF

Table 4.9 Capacitance

4.6 Packaging

The L64814 FPU is available in a 143-pin, cavity-up ceramic or plastic pin grid array package.This section provides information including the packaging options, pin diagram, pin list, and package outline drawing for these packages.

Order Number	Description
L64814CG-25	25 MHz, 143-pin CPGA (Commercial Range)
L64814NC-25	25 MHz, 143-pin PPGA (Commercial Range)
L64814CG-33	33 MHz, 143-pin CPGA (Commercial Range)
L64814NC-33	33 MHz, 143-pin PPGA (Commercial Range)
L64814CG-40	40 MHz, 143-pin CPGA (Commercial Range)
L64814NC-40	40 MHz, 143-pin PPGA (Commercial Range)

Table 4.10 Packaging Options

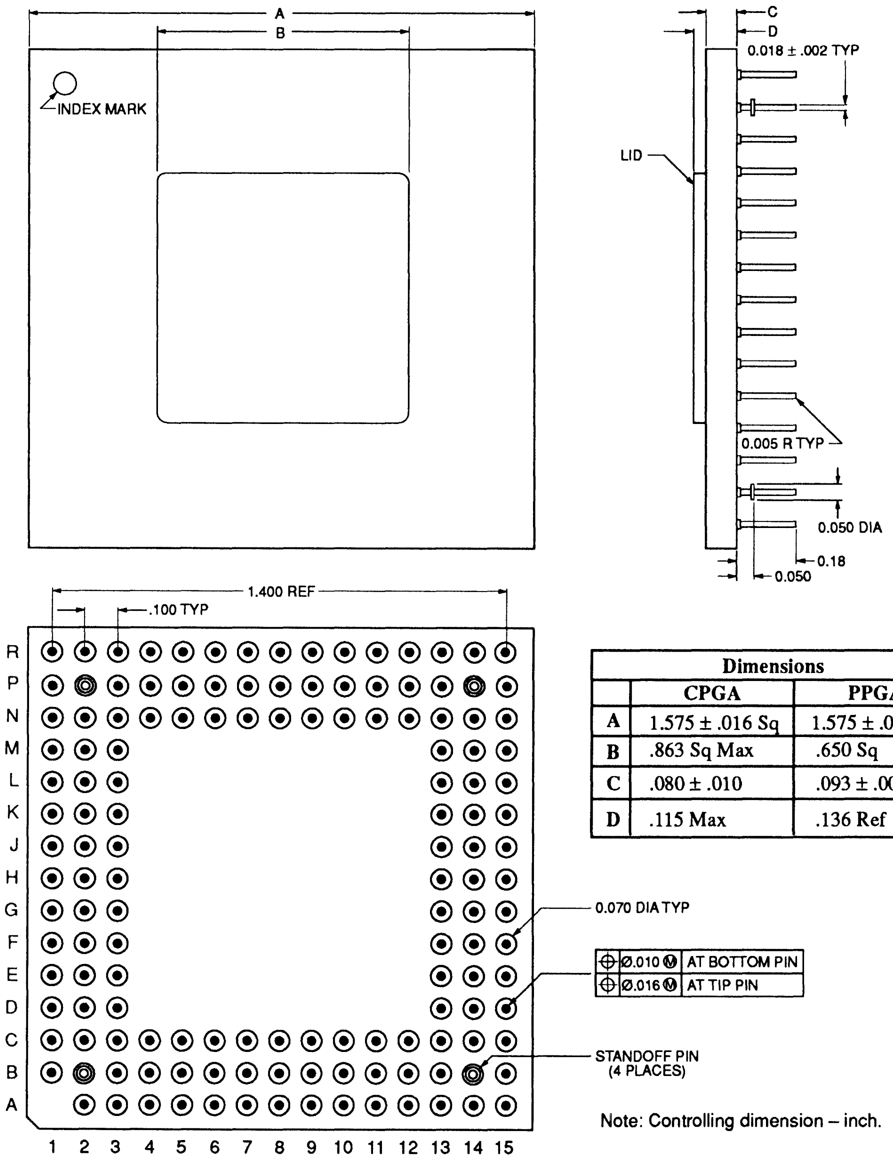
Signal Name	Pin Number	Signal Name	Pin Number	Signal Name	Pin Number	Signal Name	Pin Number
A00	H1	D00	H3	FINS2	N15	VCC	N2
A01	H2	D01	J1	FLUSH	L13	VCC	N14
A02	L1	D02	K1	FNULL	G15	VCC	P2
A03	M1	D03	L2	$\overline{\text{FP}}$	R15	VCC	P5
A04	P1	D04	N1	FXACK	E15	VCC	P7
A05	R1	D05	M3	INST	M15	VCC	P10
A06	P4	D06	R3	$\overline{\text{MDS}}$	K14	VCC	P11
A07	R4	D07	R5	$\overline{\text{MHOLDA}}$	J14	VCC	P12
A08	P6	D08	N6	MHOLDB	K15	VCC	P14
A09	R6	D09	R7	$\overline{\text{RESET}}$	F13	VCC	P15
A10	R8	D10	N8	$\overline{\text{TOE}}$	N9	VCC	R2
A11	P8	D11	R9	*missing pin*	A1	GND	A15
A12	R10	D12	P9	*reserved*	C7	GND	C3
A13	R11	D13	R12	*reserved*	J3	GND	C4
A14	R13	D14	N11	VCC	B2	GND	C6
A15	R14	D15	P13	VCC	B3	GND	C9
A16	F1	D16	G1	VCC	B6	GND	C10
A17	G2	D17	F2	VCC	B8	GND	C11
A18	E1	D18	F3	VCC	B13	GND	C12
A19	E2	D19	D1	VCC	B14	GND	C13
A20	E3	D20	C1	VCC	B15	GND	D3
A21	C2	D21	B1	VCC	C5	GND	D13
A22	A3	D22	A2	VCC	C8	GND	D14
A23	B4	D23	B5	VCC	C14	GND	F14
A24	A5	D24	A4	VCC	D2	GND	G3
A25	A6	D25	B7	VCC	J13	GND	G14
A26	A8	D26	A7	VCC	K2	GND	H13
A27	A9	D27	B9	VCC	K13	GND	K3
A28	A10	D28	B10	VCC	L14	GND	L3
A29	A11	D29	B11	VCC	L15	GND	M13
A30	A12	D30	B12	VCC	M2	GND	N3
A31	A13	D31	A14			GND	N4
$\overline{\text{BHOLD}}$	J15	FCCV	C15			GND	N5
CCCV	E13	FCC0	E14			GND	N7
$\overline{\text{CHOLD}}$	H14	FCC1	D15			GND	N10
CLK	G13	$\overline{\text{FEXC}}$	F15			GND	N12
$\overline{\text{DOE}}$	J2	$\overline{\text{FHOLD}}$	H15			GND	N13
		FINS1	M14			GND	P3

Figure 4.4 143-Pin Cavity-Up Pin Grid Array Pin List

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	*missing pin*	D22	A22	D24	A24	A25	D26	A26	A27	A28	A29	A30	A31	D31	GND
B	D21	VCC	VCC	A23	D23	VCC	D25	VCC	D27	D28	D29	D30	VCC	VCC	VCC
C	D20	A21	GND	GND	VCC	GND	*	VCC	GND	GND	GND	GND	GND	VCC	FCCV
D	D19	VCC	GND										GND	GND	FCC1
E	A18	A19	A20										CCCV	FCC0	FXACK
F	A16	D17	D18										$\overline{\text{RESET}}$	GND	$\overline{\text{FEXC}}$
G	D16	A17	GND										CLK	GND	FNULL
H	A00	A01	D00										GND	$\overline{\text{CHOLD}}$	$\overline{\text{FHOLD}}$
J	D01	$\overline{\text{DOE}}$	*										VCC	$\overline{\text{MHOLDA}}$	$\overline{\text{BHOLD}}$
K	D02	VCC	GND										VCC	$\overline{\text{MDS}}$	$\overline{\text{MHOLDB}}$
L	A02	D03	GND										FLUSH	VCC	VCC
M	A03	VCC	D05										GND	FINS1	INST
N	D04	VCC	GND	GND	GND	D08	GND	D10	$\overline{\text{TOE}}$	GND	D14	GND	GND	VCC	FINS2
P	A04	VCC	GND	A06	VCC	A08	VCC	A11	D12	VCC	VCC	VCC	D15	VCC	VCC
R	A05	VCC	D06	A07	D07	A09	D09	A10	D11	A12	A13	D13	A14	A15	$\overline{\text{FP}}$

* *Reserved Pins*

Figure 4.5 143-Pin Cavity-Up Pin Grid Array Pin Diagram – Top View



Dimensions		
	CPGA	PPGA
A	1.575 ± .016 Sq	1.575 ± .005 Sq
B	.863 Sq Max	.650 Sq
C	.080 ± .010	.093 ± .003
D	.115 Max	.136 Ref

Note: Controlling dimension – inch.

Figure 4.6 143-Pin Cavity-Up Pin Grid Array Package Outline

Sales Offices and Design Resource Centers

LSI Logic Corporation Headquarters
■ **1551 McCarthy Blvd**
Milpitas CA 95035
Tel: 408.433.8000
Telex: 171092
FAX: 408.434.6457

Alabama
■ 600 Boulevard South
Suite 104P
Huntsville, AL 35802
Tel: 205.883.3527
FAX: 205.883.3513

Arizona
8283 N Hayden Rd
Suite 270
Scottsdale AZ 85258
Tel: 602.951.4560
FAX: 602.951.4580

California
■ 2540 N. 1st Street
Suite 201
San Jose CA 95131
Tel: 408.954.1561
FAX: 408.954.1565

■ Two Park Plaza
Suite 1000
Irvine CA 92714
Tel: 714.553.5600
FAX: 714.474.8101

5625 Ruffin Rd
Suite 200
San Diego CA 92123
Tel: 619.541.7092
FAX: 619.541.2758

■ 15281 Ventura Blvd
Suite 275
Encino CA 91436
Tel: 818.379.2400
FAX: 818.783.5548

Colorado
3801 E Florida Ave
Suite 400
Denver CO 80210
Tel: 303.756.8800
FAX: 303.759.3486

Florida
201 Park Place
Suite 300
Altamonte Springs FL 32701
Tel: 407.339.2242
FAX: 407.831.3919

■ 1900 Glades Rd
Suite 201
Boca Raton FL 33431
Tel: 407.395.6200
FAX: 407.394.2865

Georgia
6525 The Corners Parkway
Suite 400
Norcross GA 30092
Tel: 404.448.4898
FAX: 404.449.9236

Illinois
■ One Pierce Place
Suite 400 East
Itasca IL 60143
Tel: 708.773.0111
FAX: 708.773.4631

Maryland
■ 6903 Rockledge Dr
Suite 230
Bethesda MD 20817
Tel: 301.897.5800
FAX: 301.897.8389

10480 Little Patuxent Pkwy
Suite 500
Columbia MD 21044
Tel: 301.740.5664
FAX: 301.740.2048

Massachusetts
■ 1601 Trapelo Rd
Waltham MA 02154
Tel: 617.890.0180 (Design Ctr)
Tel: 617.890.0161 (Sales Ofc)
FAX: 617.890.6158

Michigan
455 E Eisenhower Parkway
Suite 108
Ann Arbor MI 48108
Tel: 313.930.6975
Telex: 706456
FAX: 313.930.6978

Minnesota
■ 8300 Norman Center Drive
Suite 730
Minneapolis MN 55437
Tel: 612.921.8300
FAX: 612.921.8399

New Jersey
■ 379 Thornall St
Edison NJ 08837
Tel: 201.549.4500
FAX: 201.549.4802

New York
1065 Route 82
Hopewell Junction
New York NY 12533
Tel: 914.226.1620
FAX: 914.226.1315

North Carolina
■ 4601 Six Forks Rd
Suite 528, Phase 2
Raleigh NC 27609
Tel: 919.872.8400
FAX: 919.783.8909

Oregon
15455 NW Greenbrier Pkwy
Suite 210A
Beaverton OR 97006
Tel: 503.645.9882
FAX: 503.645.6612

Pennsylvania
Three Neshaminy Interplex
Suite 301
Trevose PA 19047
Tel: 215.638.3010
FAX: 215.245.4705

Texas
6034 W Courtyard
Suite 305
Austin TX 78730
Tel: 512.338.2140
FAX: 512.343.2612

■ 5080 Spectrum Drive
Suite 1010 West
Dallas TX 75248
Tel: 214.788.2966
FAX: 214.233.9234

Washington
■ 3015 112th Avenue NE
Suite 205
Bellevue WA 98004
Tel: 206.822.4384
FAX: 206.827.2884

Austria
LSI Logic/Ing. Ernst Steiner
■ Hummelgasse 14
A-1130 Wien
Tel: 43.222.827474.0
TWX: 135026 es a
FAX: 43.222.8285617

LSI Logic Corporation of Canada, Inc.
Headquarters
■ Petro-Canada Centre
#3410 150-6th Avenue SW
Calgary AB T2P 3Y7
Tel: 403.262.9292
FAX: 403.262.9494

■ 150 Karl Clark Road
Edmonton AB T6N 1E2
Tel: 403.450.4400
FAX: 403.450.4411

■ #220 4259 Canada Way
Burnaby BC V5G 1H1
Tel: 604.433.5705
FAX: 604.433.8443

■ #400 260 Hearst Way
Kanata ON K2L 3H1
Tel: 613.592.1263
Telex: 053.3849
FAX: 613.592.3253

■ #1110 401 The West Mall
Etobicoke ON M9C5J5
Tel: 416.620.7400
FAX: 416.620.5005

■ #600 755 St Jean Boulevard
Pointe Claire PQ H9S 5M9
Tel: 514.694.2417
FAX: 514.694.2699

France
LSI Logic S.A.
■ Tour Chenonceaux
204 Rond-point du Pont de Sevres
92516 Boulogne Billancourt
Tel: 33.1.46212525
Telex: 631475
FAX: 33.1.46203138

Israel
LSI Logic Limited
■ 40 Sokolov St
Ramat Hasharon 47235
Tel: 972.3.5403741/4
Telex: 371662
FAX: 972.3.5403747

Italy
LSI Logic SPA
■ Centro Direzionale Colleoni
Palazzo Orione, Ing. 1
20041 Agrate Brianza (MI)
Tel: 39.39.6056881
FAX: 39.39.653564

Japan
LSI Logic K. K.
■ Kokusai-Shin Akasaka
West Wing 13th Floor
6-1-20 Akasaka, Minato-Ku
Tokyo 108
Tel: 81.3.589.2711
FAX: 81.3.589.2740

■ 2-10-1 Kashuga
Tsukuba-Shi, Ibaraki 305
Tel: 81.298.52.8371
FAX: 81.298.52.8376

■ Twin 21 MID Tower 31st Floor
2-1-61 Shiromi, Higashi-Ku
Osaka 540
Tel: 81.6.947.5281
FAX: 81.6.947.5287

LSI Logic Corporation of Korea Limited
■ 7th Floor 1304-3
Namseoul Building
Seocho-Dong
Seocho-Ku
Tel: 82.2.561.2921
FAX: 82.2.554.9327

Netherlands
LSI Logic/Arcobel B.V.
■ Griekenweg 25
Postbox 344
NL-5340 AH Oss
Tel: 31.4120.30335
TWX: 37489
FAX: 31.4120.30635

Scotland
LSI Logic Limited
■ Lomond House
Beveridge Square
Livingstone EH54 6QS
Tel: 44.506.416767
FAX: 44.506.414836

Sweden
LSI Logic Export AB
TorShamnsgatan 39
S-16440 Kista
Tel: 46.8.703.4680
FAX: 46.8.7506647

Switzerland
LSI Logic/Sulzer Gmbh
■ Mittelstr. 24
CH-2560 Nidau/Biel
Tel: 41.32.515441
FAX: 41.32.516507

Taiwan
LSI Logic Corporation
■ 678 Tun Hua S. Rd
3rd Floor-2
Taipei, Taiwan R.O.C.
Tel: 02.755.3433
FAX: 02.755.5176

United Kingdom
LSI Logic Limited
■ Grenville Place
The Ring
Bracknell
Berkshire RG121BP
Tel: 44.344.426544
Telex: 848679
FAX: 44.344.481.039

West Germany
LSI Logic GmbH
Headquarters
■ Arabella Strasse 33
8000 Munich 81
Tel: 49.89.926903.0
FAX: 49.89.917096

■ Niederkasseler Lohweg 8
4000 Dusseldorf 11
Tel: 49.211.5961066
TWX: 8587248
FAX: 49.211.592130

■ Buechsenstrasse 15
7000 Stuttgart 1
Tel: 49.711.2262151
TWX: 723813
FAX: 49.711.2261124

AE Advanced Electronics
■ Stefan-George-Ring 19
8000 Munich 81
Tel: 49.89/93009855
FAX: 49.89/93009866

AE Advanced Electronics
■ Theatrestr. 14
3000 Hannover 1
Tel: 49.511/3681756
FAX: 49.511/3681759

■ Sales Offices with Design Resource Centers

Printed in USA
129.8302.6537.5K.IM.W

LSI LOGIC

LSI Logic
Corporation
1551 McCarthy Blvd
Milpitas CA 95035
408.433.8000

Order No. L64814T Rev A
Part No. MD70-000104-99 A