TSC692E
Floating Point Unit

User's Manual

for Embedded Real time 32–bit Computer

(ERC32)

for SPACE Applications

Table of Contents

1. Introduction 1
1.1. Scope 1

2. TSC692E Overview 1
2.1. SPARC RISC Standard Functions: 1
2.2. Fault Tolerant and Test MECHANISM Improvements: 1
2.3. Presentation of the ERC32 2
Concept 2
Functional Description 2

3. Standard TSC692E Functions 3
3.1. TSC692E Functional Description 4
3.2. Floating–Point/Integer Unit Interface 6
3.2.1. TSC692E RT Instruction Fetch and Execution 7
3.2.1.1. Instruction Fetch 8
3.2.1.2. Instruction Execution 9
3.2.1.2.1. Floating–Point Compare Execution 11
3.2.1.2.2. FPop Queuing 11
3.2.2. Instruction Pipeline Flush 11
3.2.2.1. Hold Signals 13
3.2.2.2. Interlocking with FHOLD 13
3.2.2.3. FNULL Signal 14
3.3. TSC692E Programming Model 14
3.3.1. TSC692E Registers 14
3.3.1.1. f Registers 14
3.3.1.2. FP Queue 15
3.3.1.3. Floating–Point Status Register (FSR) 16
3.3.2. TSC692E Floating–Point Instructions 18
3.3.3. TSC692E Internal Operation 20
3.3.3.1. Exception Handling 20
3.3.4. TSC692E IEEE–754 Compliance 22
3.3.4.1. IEEE Definitions 22
3.3.4.2. IEEE Floating–point Data Formats 23
3.3.4.2.1. Single–Precision Floating–Point 23
3.3.4.2.2. Double–Precision Floating–Point 24
3.3.5. NaN Format 24
3.3.6. TSC692E Exception Cases 25
3.4. TSC692E Signal Descriptions 26
3.4.1. Integer Unit Interface Signals 26
3.4.2. Coprocessor Interface Signals 27
3.4.3. System/Memory Interface Signals 27
3.4.4. TAP signals 29
3.4.5. Power and Clock Signals 29

4. Fault Tolerant and Test MECHANISM. 30
4.1. Fault Tolerant and Test Support Signals 30

4.1.1. Parity Checking 30
4.1.2. Master/Checker Mode 31
4.1.3. Test Access Port 31
4.1.4. Miscellaneous 31
4.2.Parity Checking 31
4.2.1. Introduction 31
4.2.2. Error handling scheme in TSC692E 31
4.2.3. Parity Checking on Control Pads for the FPU 32
4.2.3.1. Input control signals 32
4.2.3.1.1. Output control signals 32
4.2.4. Parity Checking on address bus 32
4.2.5. Parity Checking on data bus 32
4.2.6. Internal Parity Checking 32
4.2.7.Non RT 602 Mode 32
4.3. Master/checker Operation 34
4.3.1. Basic function 35
4.3.2. Master/Checker signals 35
4.4. IEEE Standard Test Access Port & Boundary-Scan Architecture 36
4.4.1. TAP signals 36
4.4.2. TAP Controller 36
4.4.3. The Instruction Register 36
4.4.3.1. Design and Construction of the instruction register 37
4.4.3.2. BYPASS Instruction 37
4.4.3.3. EXTEST Instruction 37
4.4.3.4. INTEST Instruction 37
4.4.3.5. SAMPLE/PRELOAD Instruction 37
4.4.4. The Device Identification Register 38
4.4.5. Internal Scan Path 38
4.5. Boundary scan test register 38
4.6. Parity on odd and even bits of the register file bits 38

5. Electrical and Mechanical Specifications. 39
5.1. TSC692E Maximum Ratings and DC Characteristics 39
5.1.1. TSC692E Maximum Rating 39
5.1.2. TSC692E Operating Range 39
5.1.3. TSC692E DC Characteristics Over the Operating Range 39
5.1.4. TSC692E AC Test Loads and Waveforms 40
5.2. TSC692E AC Characteristics 40
5.2.1. TSC692E AC Waveforms 43
5.3. TSC692E Package Descriptions 50
5.3.1. 160-Pin MQFP-L Package 50
5.3.2. 160-Pin MQFP-L Pin Assignment 51

List of Tables

Table 1. Load instruction execution 8
Table 2. Store instruction execution 8
Table 3. FPop execution 8
Table 4. FHOLD Resource/Operand Dependency Cases 14
Table 5. Floating–Point Status Register Summary 17
Table 6. Floating–Point Load and Store Instruction Cycle Count 19
Table 7. Floating–Point Operate (FPops) Instruction Cycle Count 20
Table 8. Untrapped FP result in same format as operand 24
Table 9. Untrapped FP result in different format 24
Table 10. FCC[1:0] Condition Codes 26
Table 11. priority within traps 32
Table 12. Instruction Register Encoding 37
Table 13. TSC692E Operating Range 39
Table 14. TSC692E DC Characteristics over the operating range 39
Table 15. TSC692E Capacitance Ratings [1] 39
Table 16. TSC692E Characteristics at 14/25 MHz 40

List of Figures

ERC32 Architecture 3
Figure 1. TSC692E Functional Block Diagram 5
Figure 2. TSC692E Block Diagram (without parity checking) 6
Figure 3. TSC692E Hardware Interface 7
Figure 4. Instruction Fetch (Cache Hit) 9
Figure 5. Floating–Point Instruction Dispatching 10
Figure 6. Floating–Point Compare (FCMP) Execution 10
Figure 7. Floating–Point Instruction Pipeline During A Trap 11
Figure 8. Effect of FLUSH on LDF Instruction 12
Figure 9. Effect of FLUSH on STF Instruction 12
Figure 10. Effect of FLUSH on FPop Instruction 12
Figure 11. Effect of FLUSH on FCMP Instruction 13
Figure 12. f Register Organization 15
Figure 13. f Register Addressing 15
Figure 14. Floating–Point Status Register 16
Figure 15. FPU Operation Modes 21
Figure 16. Floating–Point Exception Handshake 22
Figure 17. Single–Precision Floating–Point Format 23
Figure 18. Double–Precision Floating–Point Format 23
Figure 19. Parity Checking on Fractional Datapath 33
Figure 20. Parity Checking on Exponent Datapath 34
Figure 21. Master/Checker configuration 35
Figure 22. Instruction Register Cell 37
Figure 23. Boundary Scan Cell 38
Figure 24. TSC692E AC Test Loads and Waveforms 40
Figure 25. Floating–Point FHOLD Assertion 43
Figure 26. Clock and RESET Timing 43
Figure 27. Floating–Point Load Operation 44
Figure 28. Floating–Point Store Operation 44
Figure 29. Effect of FLUSH on Store Timing 45
Figure 30. Floating–Point Load Cache Miss 45
Figure 31. Floating–Point Store Cache Miss 46
Figure 32. Floating–Point Compare 46
Figure 33. Floating–Point Trap 47
Figure 34. TAP Signals 47
Figure 35. HWERROR Timing 48
Figure 36. PARITY Signals 48
Figure 37. MASTER/CHECKER Signals 49
Figure 38. HALT Signal 49

TSC692E Floating Point Unit

1. Introduction

1.1. Scope

This document presents a preliminary datasheet of the TSC692E RT Floating Point Unit device specification. It is organized in three chapters:

- Standard FPU (TSC692E) Functions (Chapter 3)
- Fault MECHANISM and Test MECHANISM (Chapter 4)
- Electrical and Mechanical Specification (Chapter 5)

Chapter 3 presents standard functions including some adaptations due to the introduction of fault tolerance MECHANISM. Without losing the full binary compatibility with the entire SPARC V7.0 application software base.

Chapter 4 and 5 deal with the new added functions introduced in the TSC692E to improve the reliability of space applications. These new functions do not impact the SPARC V7.0 compatibility.

2. TSC692E Overview

2.1. SPARC RISC Standard Functions:

- Full compatibility with Standard ANSI/IEEE 754-1985 for binary Floating Point Arithmetic
- 64-bit Internal Datapath
- Based on Floating-Point Unit from SUN
- Tightly coupled Integer-Unit interface

2.2. Fault Tolerant and Test MECHANISM Improvements:

- Parity checking on 98% of the total number of latches with hardware error traps
- Parity checking of address, data pads and IU/FPU control pads
- Master/Checker operation
- IEEE Standard Test Access Port & Boundary-Scan Architecture
- Possibility to disable the bus parity checking
- Manufactured using Atmel Wireless & Microcontrollers Space hardened 0.8 µm SCMOS-RT technology
- Part of the ERC32 high performance 32-bit computing core

To support applications requiring an extremely high level of reliability, the following improvements were introduced in the standard SPARC RISC FPU TSC692E:

- Several independent fault detection MECHANISMs to support the design of fault tolerant system like parity checking and master/checker operations.
- Support of sophisticated PC board level tests applicating the IEEE Standard Test Access Port and Boundary Scan Architecture.
- Hardening of the process by construction, using restricted full static CMOS design rules for all critical blocks of the circuit such as register file, ROMs, BUSSES etc...

- Hardened device processing using the 0.8 µm SCMOS-RT TECHNOLOGY.

Thanks to careful handling of the improvements, the introduced modifications have neither reduced the performance of the device nor changed the full binary compatibility with the entire SPARC V7.0 application software. Improvements in FPU design have decreased the power consumption.

2.3. Presentation of the ERC32

The TSC692E Floating Point Unit is, with the TSC691E Integer Unit and the TSC693E (Memory Controller), a part of the ERC32 Computing Core.

Concept

The objective of the ERC32 is to provide a high performance 32-bit computing core, with which computers for on-board embedded real-time applications can be built. The core will be characterized by low circuit complexity and power consumption. Extensive concurrent error detection and support for fault tolerance and reconfiguration will also be emphasized.

In addition to the main objective the ERC32 core will be possible to use for performance demanding research applications in deep space probes. The radiation tolerance and error masking are therefore important. For the real-time applications the system might be fail-operational rather than fail-safe. By including support for reconfiguration of the error-handling the different demands from the applications can be optimized for the best purpose in each case.

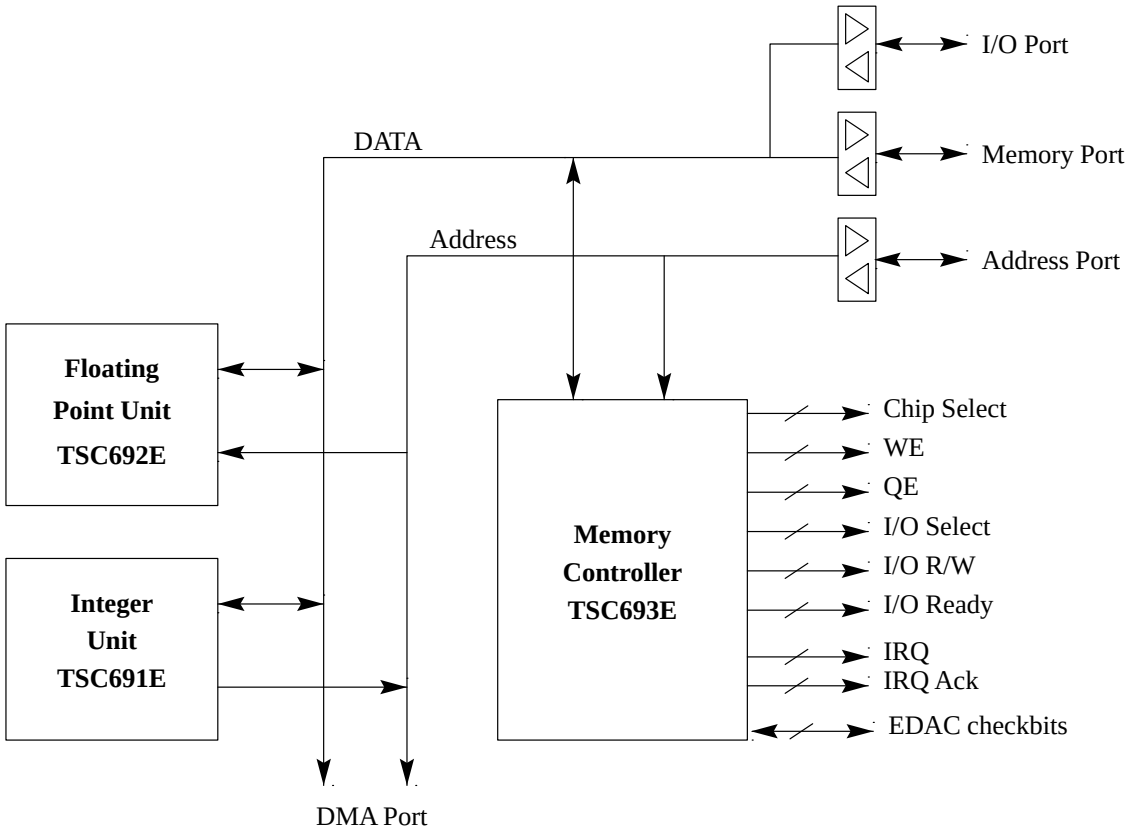
The ERC32 will be used as a building block only requiring memory and application specific peripherals to be added to form a complete on-board computer. All other system support functions will be provided by the core.

Functional Description

The ERC32 will incorporate the followings functions:

- Processor, which consists of one integer unit and one floating point unit. The processor includes concurrent error detection facilities.
- Memory controller (TSC693E), which is a unit consisting of all necessary support functions such as memory control and protection, EDAC, wait state generator, timers, interrupt handler, watch dog, UARTs and test and debug support. The unit also includes concurrent error detection facilities.
- Oscillator (optional).
- Buffers necessary to interface with memory and peripherals.

Next figure schematically shows the ERC32 architecture and external functions added to form a complete system.



ERC32 Architecture

3. Standard TSC692E Functions

The TSC692E Floating–Point Unit (FPU) is a high–performance, single–chip implementation of the SPARC reference floating–point unit. The TSC692E FPU is designed to provide execution of single and double–precision floating–point instructions concurrently with execution of integer instructions by the TSC691E Integer Unit (IU). The TSC692E is compliant to the ANSI/IEEE-754 (1985) floating–point standard.

The TSC692E provides a 64–bit Fractional/Exponent/Sign internal datapath for efficient execution of double–precision floating–point instructions. All implemented instructions are executed within hardware. For efficient data management, the TSC692E provides thirty–two 32–bit floating–point registers. These 32–bit registers can be concatenated for use as 64–bit registers for double–precision operations. The internal 64–bit architecture of the TSC692E allows high speed execution of both single– and double–precision operations.

The SPARC floating–point/integer unit interface supports concurrent execution of integer and floating–point instructions. The tightly coupled floating–point/integer unit interface requires the integer unit to provide all addressing and control signals for memory access. All instructions are fetched by the integer unit, and these instructions are simultaneously latched and decoded by both the TSC691E and TSC692E. Execution of a floating–point instruction is enabled by TSC691E, which signals the TSC692E to begin execution of the floating–point instruction when that instruction reaches the execute stage of the TSC691E instruction pipeline. In the case of a floating–point load or store instruction, the TSC691E executes the FP load or store in conjunction with the TSC692E by asserting address and control signals for memory access while the TSC692E loads or stores the data. All other floating–point instructions execute independently of the integer unit and in parallel with integer instruction execution.

The floating–point/integer unit interface provides hardware interlocking to ensure synchronization between the TSC691E and TSC692E. Hardware interlocking ensures software compatibility among SPARC systems with different levels of floating–point performance.

3.1. TSC692E Functional Description

Figure 1. illustrates the functional block diagram for the TSC692E. The fetch unit captures instructions and their addresses from the D[31:0] and A[31:0] busses. The decode unit contains logic to decode the floating–point instruction opcodes. The execution unit handles all instruction execution. The execution unit includes a floating–point queue (FP queue), which contains stored floating–point operate (FPop) instructions (see Section 3.3.2) under execution and their addresses. The execution unit controls the load unit, the store unit, and the datapath unit.

The load unit holds data that is fetched from memory via the data bus before it is written into the register file. The register file contains the 32 f registers. The exceptions/floating–point status register (FSR) unit keeps the status of completing FPops, as well as the operating mode of the TSC692E. The store unit holds data that is supplied to the data bus during a store operation. The dependency checking unit checks for conditions where the FPU must freeze the TSC691E integer unit pipeline so that an incoming instruction does not overflow the floating–point queue. The datapath unit contains arithmetic logic used by FPops to operate on the data in the register file and is comprised of a Fractional, Exponent and Sign units. Figure 2. gives a more detailed block diagram of the TSC692E.

The TSC692E provides three types of registers: *f* registers, FSR, and the FP queue. The *f* registers are the thirty–two floating–point operand registers, each 32–bits in size. Adjacent even–odd *f* register pairs (for instance, *f0* and *f*1 can be concatenated to support double–precision operands). The FSR is a 32–bit status and control register. It keeps track of rounding modes, floating–point trap types, queue status, condition codes, and various IEEE exception information. The floating–point queue contains the floating–point instruction currently under execution, along with its corresponding address. The floating–point queue provides an efficient method of handling floating–point exceptions. When an FPop instruction causes a floating–point exception, the queue contains the offending instruction/address pair along. The TSC691E integer unit acknowledges the floating–point exception, enters a floating–point trap routine, empties the queue, and corrects the exception case. After the exception case is corrected, unfinished floating–point instruction found in the floating–point queue is either executed or emulated in the trap handler before returning to normal execution.

The TSC692E depends upon the TSC691E to assert all addresses and control signals for memory access. Floating–point loads and stores are executed in conjunction with the TSC691E, which provides addresses and control signals while the TSC692E supplies or stores the data. Instruction fetch for integer and floating–point instructions is provided by the TSC691E. When the TSC691E integer unit asserts an address for an instruction fetch, it asserts the INST signal one clock later. The TSC692E floating–point unit uses INST to determine when a valid instruction is present on the D[31:0] bus. The instruction, which appears on the data bus on the next clock cycle, is latched and paired with its corresponding address. In any given cycle, one instruction/address pair is stored by the TSC692E, regardless of whether the instruction is an integer or floating–point instruction. This instruction/address pair may be selected for execution by the TSC691E upon asserting the FINS1 or FINS2 signal. The FINS1 or FINS2 signals enables a floating–point instruction to begin execution by the TSC692E.

Upon decoding a floating–point instruction, the TSC691E will assert the FINS1 or the FINS2 signal to enable the TSC692E to begin execution. The FINS1 or FINS2 signal is asserted during the decode stage of the floating–point instruction, and is recognized by the TSC692E at the beginning of the execute stage of the floating–point instruction. This ensures synchronization of the decode and execute stages of a floating–point instruction between instruction pipelines of the TSC691E and the TSC692E.

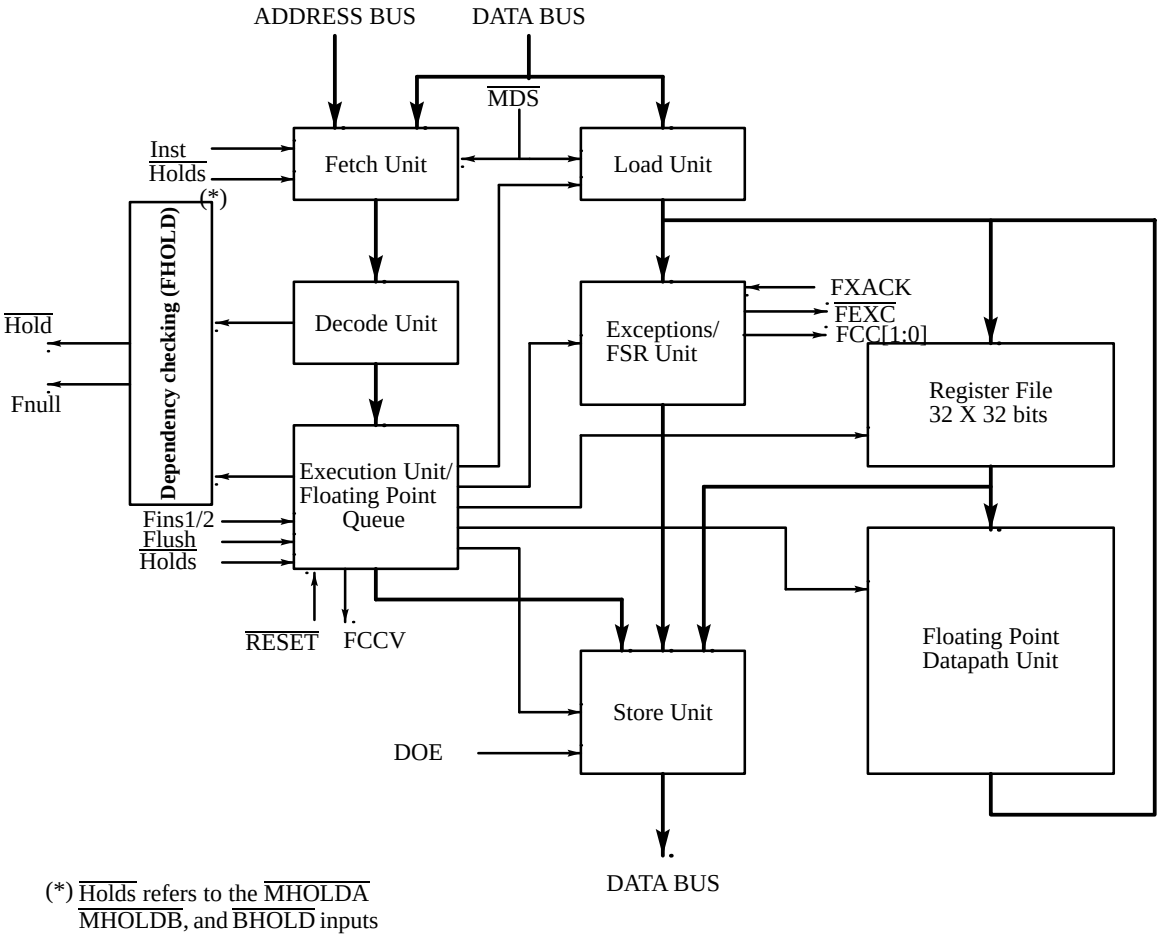


Figure 1. TSC692E Functional Block Diagram

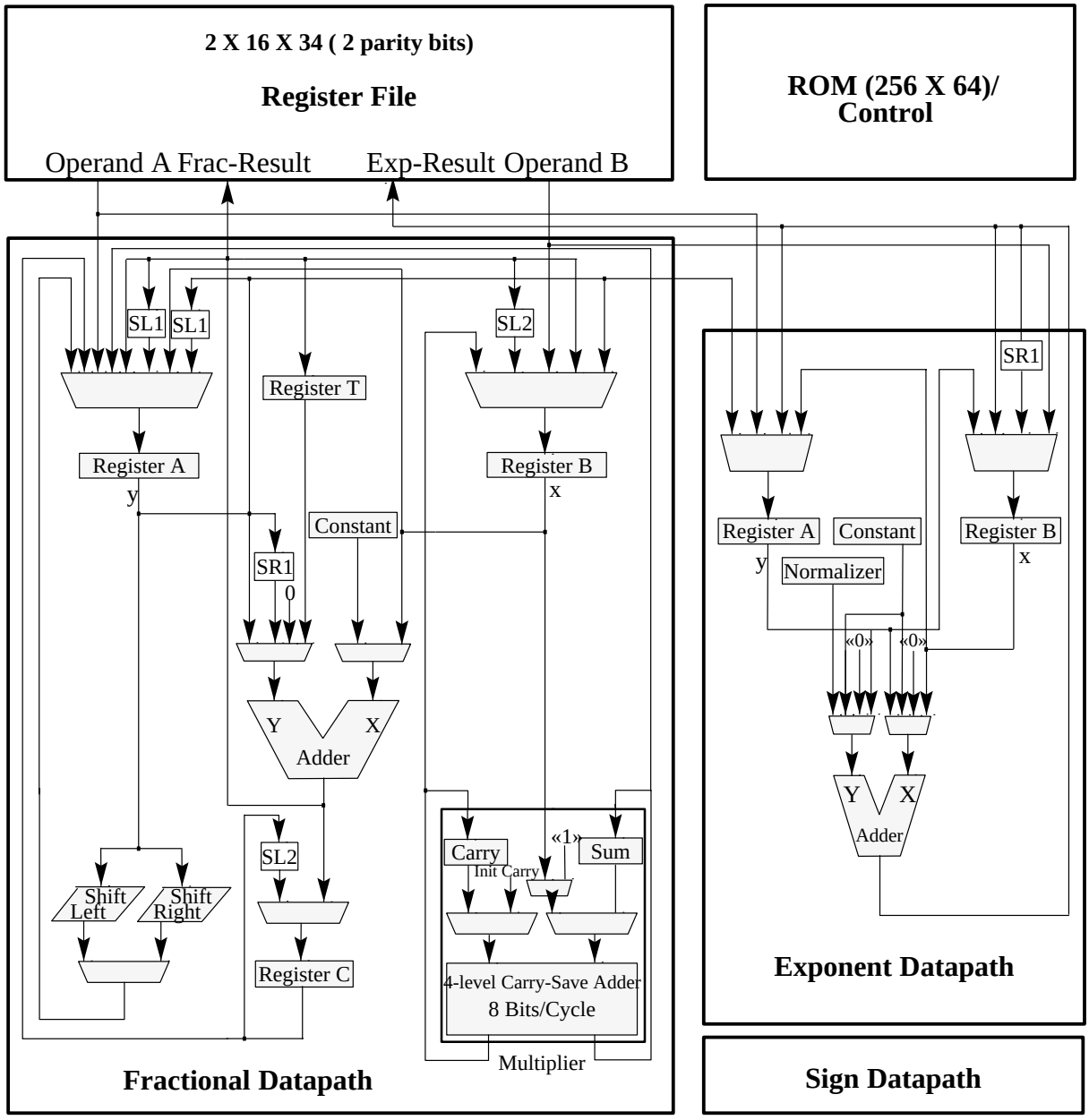


Figure 2. TSC692E Block Diagram (without parity checking)

3.2. Floating–Point/Integer Unit Interface

The TSC692E is designed to directly interface with the TSC691E without external glue logic. Figure 3. illustrates the signals required to interconnect the TSC691E and TSC692E. The control signals illustrated in Figure 3. are used to interface with the remainder of the CPU system components. The FNULL, $\overline{\text{RESET}}$, $\overline{\text{BHOLD}}$, $\overline{\text{MHOLDA}}$ or $\overline{\text{MHOLDB}}$,

$\overline{MDS}$, and $\overline{DOE}$ signals are used by the a Cache Controller(CC) and Memory Management Unit (MMU) for cache interface and virtual bus arbitration. The signal descriptions for the TSC692E signals are described in Section 3.4.

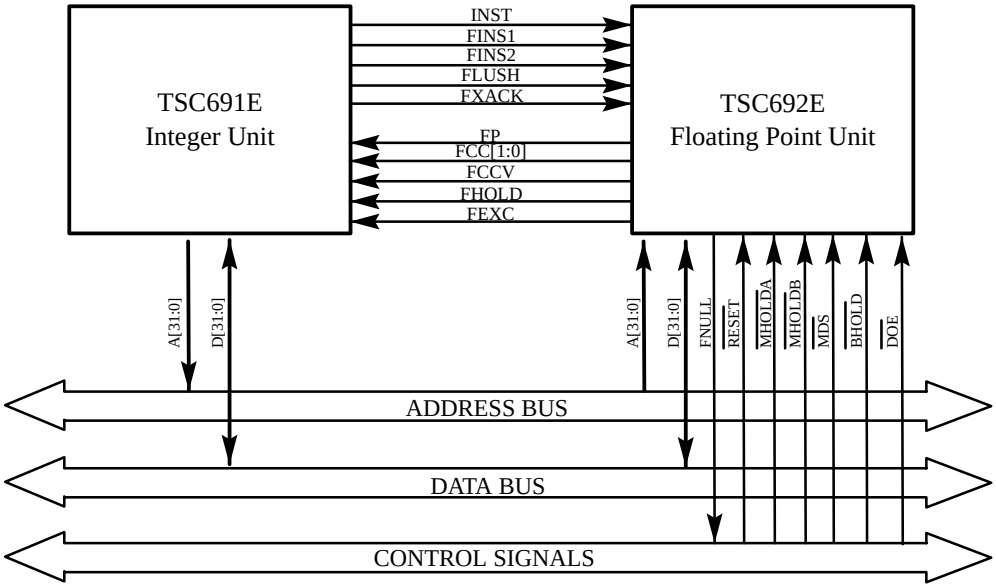


Figure 3. TSC692E Hardware Interface

3.2.1. TSC692E RT Instruction Fetch and Execution

The TSC692E uses a four–stage instruction pipeline consisting of fetch, decode, execute, and write stages (F, D, E, and W). The instruction pipelines for the TSC691E and the TSC692E are concurrent and synchronized; a floating–point instruction will be in the same stage in both processors. Multiple cycle instructions such as floating–point operate instructions (FPops) leave the pipeline after the W stage and enter the FP queue until completion.

Addresses for both integer unit and floating–point unit instructions are supplied by the TSC691E. The TSC692E FPU latches all instructions and the corresponding addresses from the D[31:0] and A[31:0] busses. The TSC692E uses the INST signal, supplied by the TSC691E, to identify an instruction fetch by the integer unit.

Decode of the latched instruction occurs on the next clock cycle, with both the IU and the FPU decoding the instruction simultaneously. During the decode stage of the floating–point instruction, the FPU checks for operand and resource dependencies. When the TSC691E integer unit decodes a FPop, it asserts the FINS1 or FINS2 signal. This occurs before the end of the decode stage, and is used by the TSC692E to initiate the execution of a floating–point instruction. If the TSC692E has detected an operand or resource dependency during the decode stage, the FPU will assert $\overline{FHOLD}$ as the instruction begins the execution stage. This freezes the integer unit's pipeline until the FPU can resolve the dependency.

If no resource or operand dependencies exist, the decoded floating–point instruction begins execution. Instructions entering execution are stored in the FP queue, where they are held until execution is completed. Note that if the FP

queue is full during an instruction's decode stage, the TSC692E asserts $\overline{\text{FHOLD}}$ as the instruction enters the execution stage in order to halt the TSC691E. $\overline{\text{FHOLD}}$ is released when space becomes available in the FP queue.

The following tables describe the execution phases of TSC692E instructions. Additional cycles beyond the F, D, E, and W stages are denoted as Wh (Write hold). Wh stages are equivalent to the additional cycles held by IOPs in the TSC691E.

Table 1. Load instruction execution

Cycle	Action
D stage	Decode instruction, check operand dependencies
E stage	$\overline{\text{FHOLD}}$ if necessary
W stage	Capture data from D[31:0] bus (LDF, LDFSR), capture MSW from D[31:0] bus (LDDF).
Wh1 stage	Write data into FP registers or FSR register (LDF, LDFSR), capture LSW from D[31:0] bus (LDDF)
Wh2 stage	Write data into register (LDDF)

Table 2. Store instruction execution

Cycle	Action
D stage	Decode instruction, check operand dependencies
E stage	$\overline{\text{FHOLD}}$ if necessary, read data from FSR register or FP queue
W stage (mid–cycle)	Drive data onto D[31:0] bus (STF, STFSR), drive MSW or FP queue address onto D[31:0] bus (STDF, STDFQ)
Wh1 stage (mid–cycle)	Stop driving D[31:0] bus (STF, STFSR), drive LSW or FP queue opcode onto D[31:0] bus (STDF, STDFQ)
Wh2 stage (mid–cycle)	Stop driving D[31:0] bus

Table 3. FPop execution

Cycle	Action
D stage	Decode FPop, check resource and operand dependencies
E stage	$\overline{\text{FHOLD}}$ if necessary, read operand(s) from register file
W stage	Read any additional operands from register file; start computing results
FP Queue	Compute, FPop in queue
FP Queue	Check exception status
FP Queue	Update FSR, write results or signal FP exception trap if necessary

3.2.1.1. Instruction Fetch

As the TSC691E fetches an instruction, the TSC692E captures it at the same time from the D[31:0] bus. The address corresponding to this instruction is captured from the A[31:0] in the previous cycle. The INST signal is used to determine when a valid instruction is present on the D[31:0] bus, and when a valid address has been fetched from the A[31:0] bus in the previous cycle. Figure 4. illustrates an example of an instruction fetch with a cache hit. The transactions on the address and data busses show two instruction fetches followed by a data fetch.

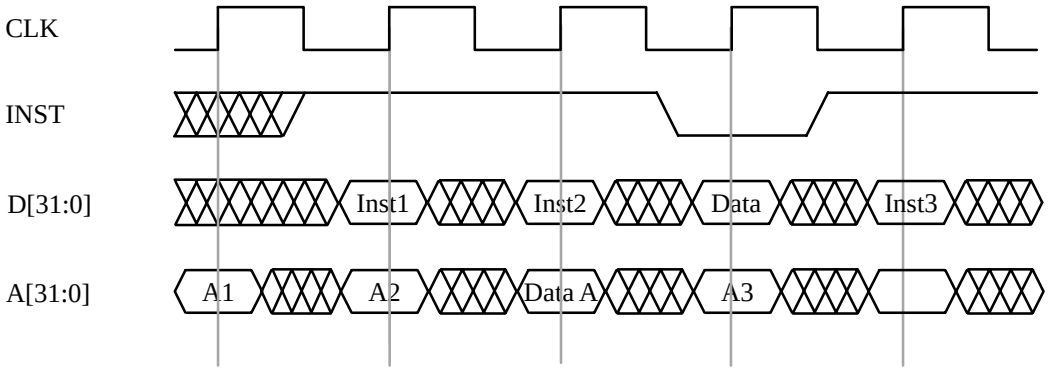


Figure 3. Instruction Fetch (Cache Hit)

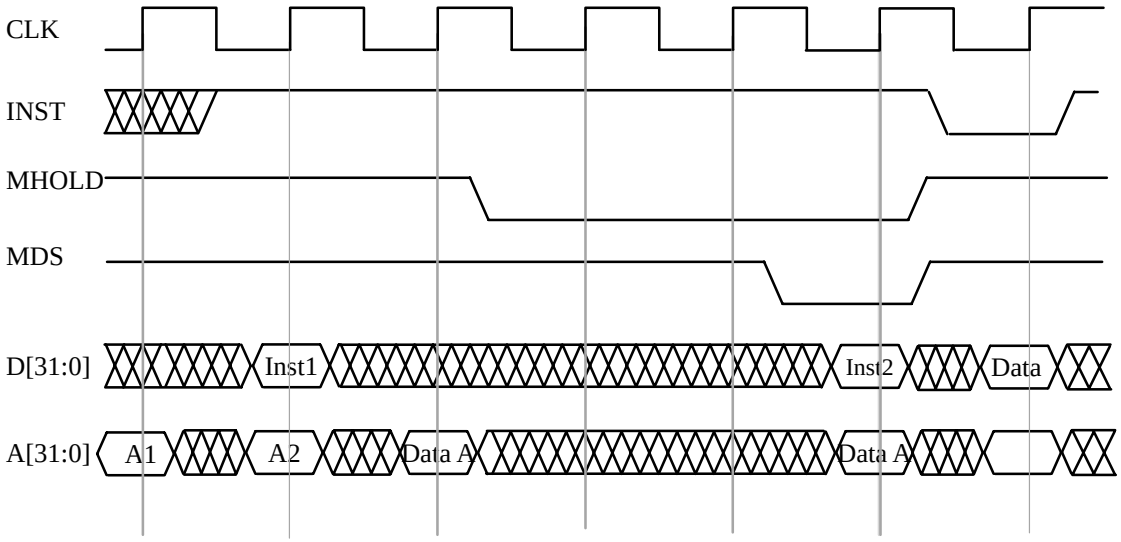


Figure 4. Instruction Fetch (Cache Miss on A2)

In the case of an instruction cache miss, a memory hold signal ($\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, or $\overline{\text{BHOLD}}$) is driven low by the cache system starting in the cycle following the instruction fetch. The instruction which was captured from the D[31:0] bus is invalid and is replaced when the system returns a valid instruction on the D[31:0] bus. The hold signal lasts for several cycles during which time the $\overline{\text{MDS}}$ signal is asserted by the cache system, notifying the TSC692E that the valid instruction is available on the D[31:0] bus. $\overline{\text{MDS}}$ is also used when there is a cache miss on data (via load instructions) so the instruction is reloaded only if INST was asserted in the previous non–hold cycle. The same sequence of transactions in Figure 4. are used in Figure 5. , except that the second instruction fetch (Inst 2) experiences a cache miss.

3.2.1.2. Instruction Execution

The FINS1 and FINS2 signals notify the TSC692E when to launch a floating–point instruction. When FINS1/FINS2 is received, the floating–point instruction is in the D stage of the TSC691E integer unit pipeline. The example in Figure 6. shows a situation where both FINS1 and FINS2 are used. A load instruction is followed by two FPops. The first FPop is fetched while the load instruction is executing. Because the load takes more than one cycle to execute, the starting of the FPop is deferred, and thus the instruction is held in the instruction buffer of the TSC692E. When the TSC691E reaches the D stage of the first FPop (Inst 2), it issues FINS2 to start the FPop. When the D stage of the second FPop (Inst 3) is reached, FINS1 is issued to start the second FPop.

FINS1 and FINS2 are never asserted in the same cycle. Both FINS1 and FINS2 are ignored in the following conditions:

1 - FLUSH is asserted.

2 - $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$,$\overline{\text{CHOLD}}$, or $\overline{\text{FHOLD}}$ is asserted.

3 - FCCV or CCCV is deasserted.

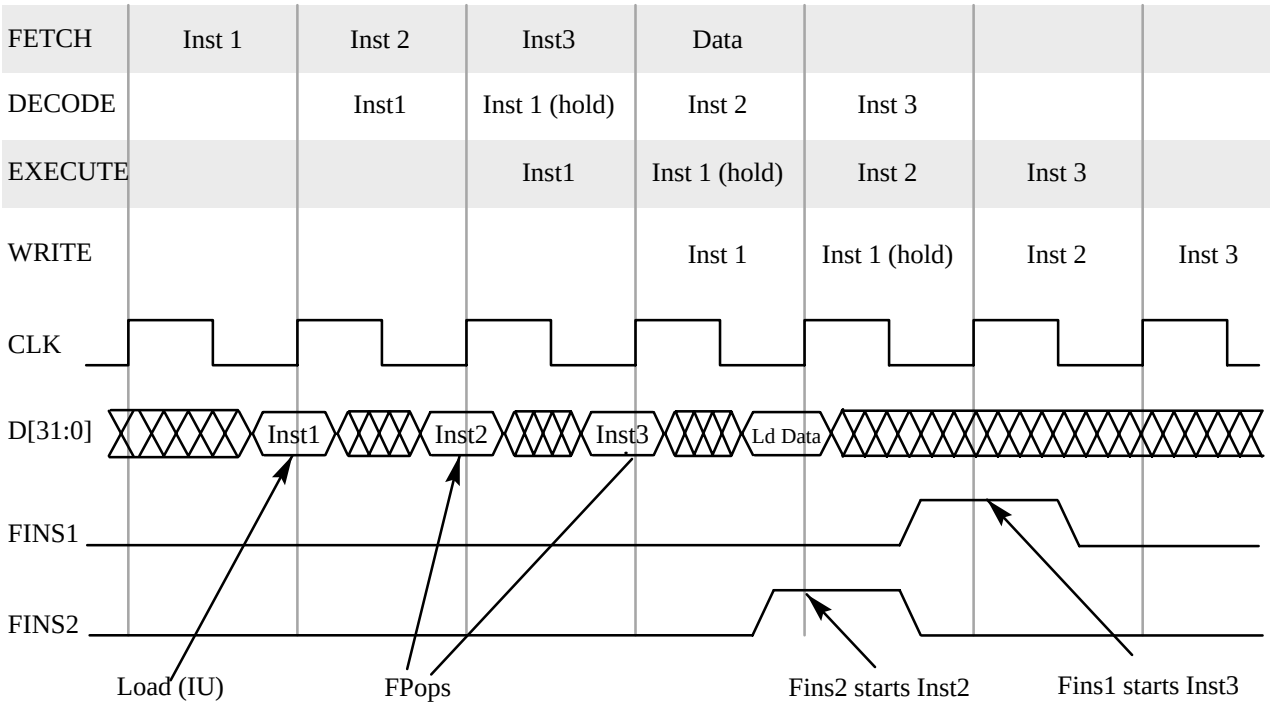


Figure 5. Floating–Point Instruction Dispatching

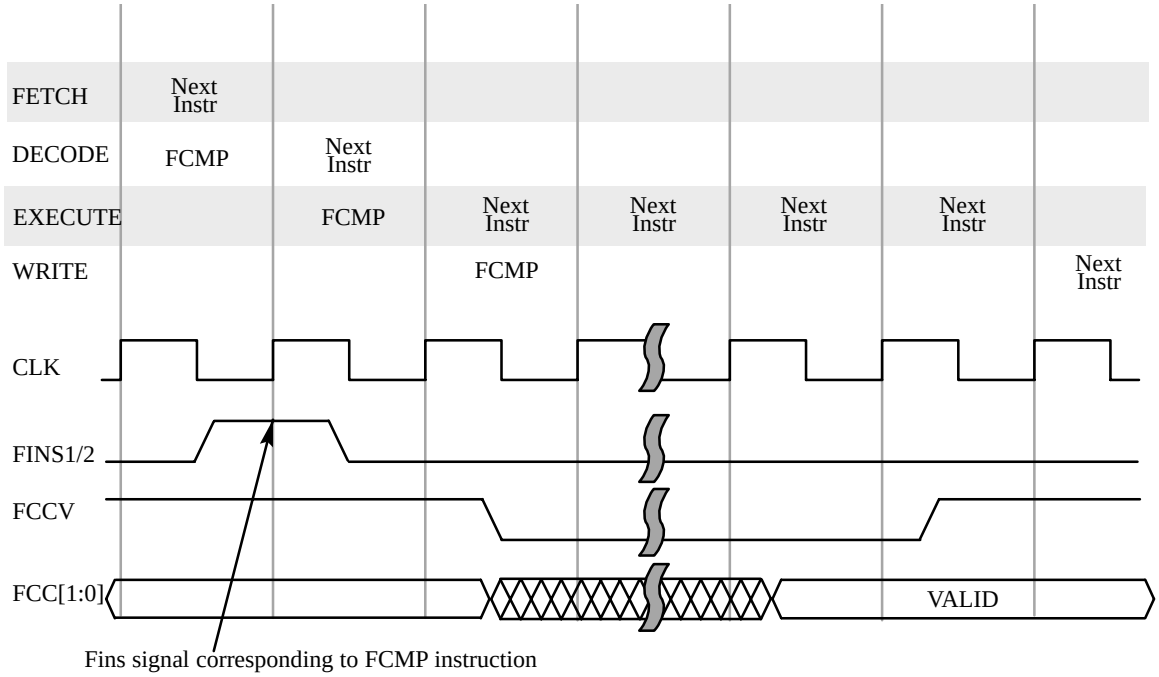


Figure 6. Floating–Point Compare (FCMP) Execution

3.2.1.2.1. Floating–Point Compare Execution

Floating–point compare instructions cause the instruction pipeline to be frozen by the use of FCCV, starting from the E stage of the instruction following the compare instruction until the FCC condition codes become valid. FCCV is deasserted, causing the TSC691E to $\overline{\text{HALT}}$ execution until FCCV is asserted. Figure 7. illustrates the timing of FCCV relative to the FCMP instruction and the FCC condition codes.

FCCV is deasserted in the W stage of the FCMP instruction. The instruction that immediately follows the FCMP is held in its E stage until FCCV is reasserted. FCC[1:0] is valid one cycle before FCCV is reasserted. For unimplemented compare instructions, the TSC692E freezes the instruction pipeline and causes an unimplemented FPop trap, which the TSC691E takes immediately.

3.2.1.2.2. FPop Queuing

When a FPop has passed the first cycle of the W stage and FLUSH has not been asserted, the FPop enters the FP queue. Note that the W stage of an FPop may be extended to more than one cycle if a hold condition exists. As an FPop completes execution successfully and results are written to the register file, it is removed from the FP queue.

3.2.2. Instruction Pipeline Flush

When a trap or interrupt occurs in the integer unit, normal program execution is $\overline{\text{HALT}}$ed and control is transferred to the trap handler. The instruction in the E stage of the pipeline and any instructions fetched after it are aborted and must be restarted after the trap handler is done (or emulated in the trap handler). Instructions that have not yet been transferred to the FP queue are aborted by the TSC692E when the trap occurs. The TSC691E asserts the FLUSH signal in the W stage of the instruction to be aborted (refer to Figure 8.). FPop which was issued before this instruction continues execution (and is in the queue) while instructions issued after it are aborted.

The following figures illustrate how each type of floating–point instruction is affected by the FLUSH signal. Figure 9. illustrates the effect of the FLUSH signal during a load floating–point instruction (LDF). A FLUSH signal asserted anytime on or before the last Wh stage of a load instruction causes the load to abort, leaving the contents of the floating–point register file unchanged.

Figure 10. illustrates the effect of FLUSH on a store floating–point instruction (STF). A FLUSH signal asserted on or before the last Wh stage of a store instruction causes the store to abort and the TSC692E to stop driving the D[31:0] bus by the middle of the next clock cycle.

Figure 11. illustrates the effect of FLUSH on a FPop instruction. A FLUSH signal asserted anytime on or before the W stage of a FPop instruction causes the FPop to abort, leaving the contents of the register file and the FSR unchanged by that instruction. FPop that has passed the W stage but is still executing (stored in the FP queue) is not affected.

Figure 12. illustrates the effect of FLUSH on a floating–point compare. FLUSH asserted in the W stage of a FCMP instruction causes the FCMP to abort, leaving the FSR unchanged by that instruction. FCCV is reasserted in the next clock cycle.

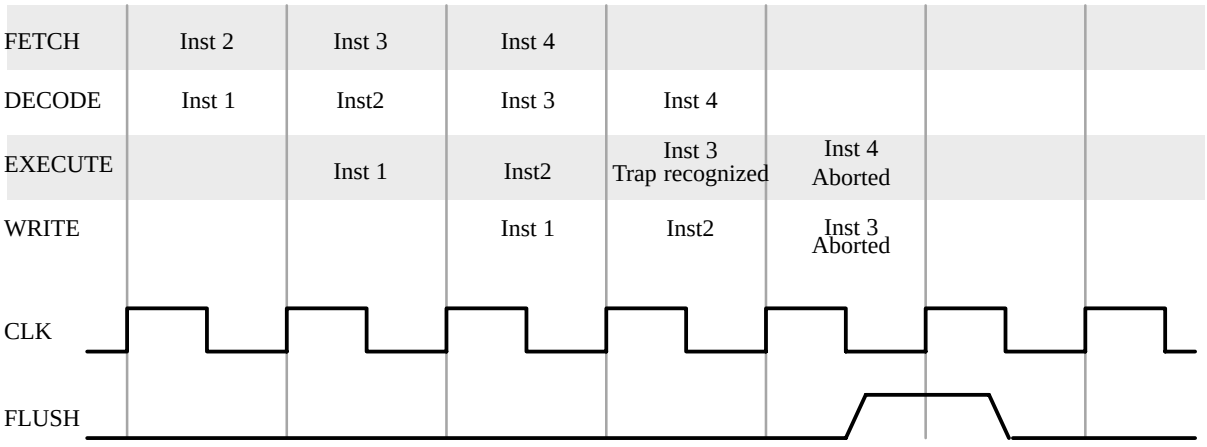


Figure 7. Floating–Point Instruction Pipeline During A Trap

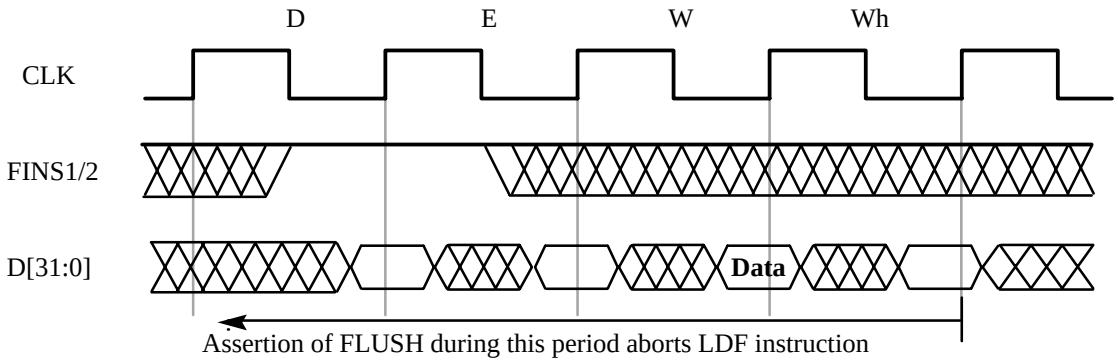


Figure 8. Effect of FLUSH on LDF Instruction

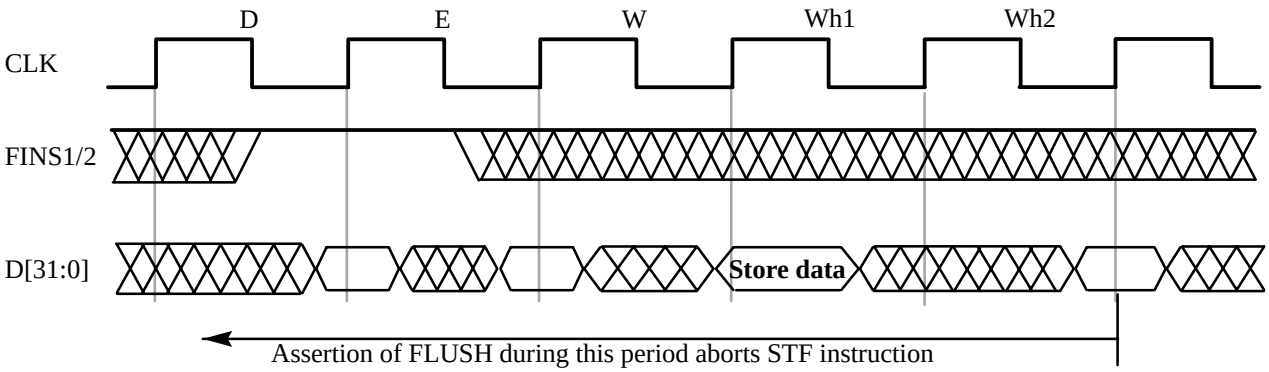


Figure 9. Effect of FLUSH on STF Instruction

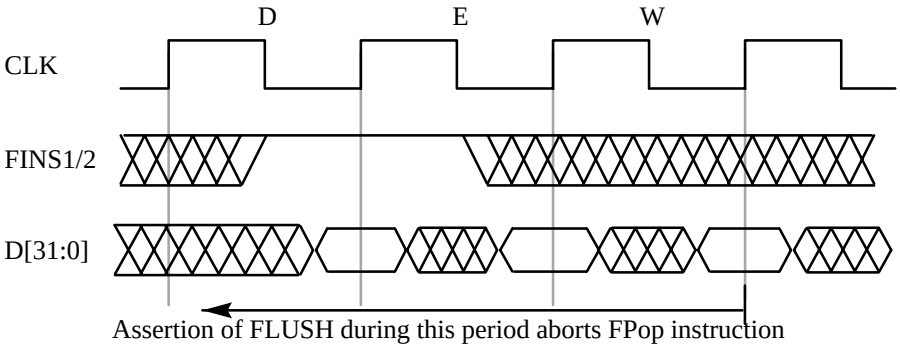


Figure 10. Effect of FLUSH on FPop Instruction

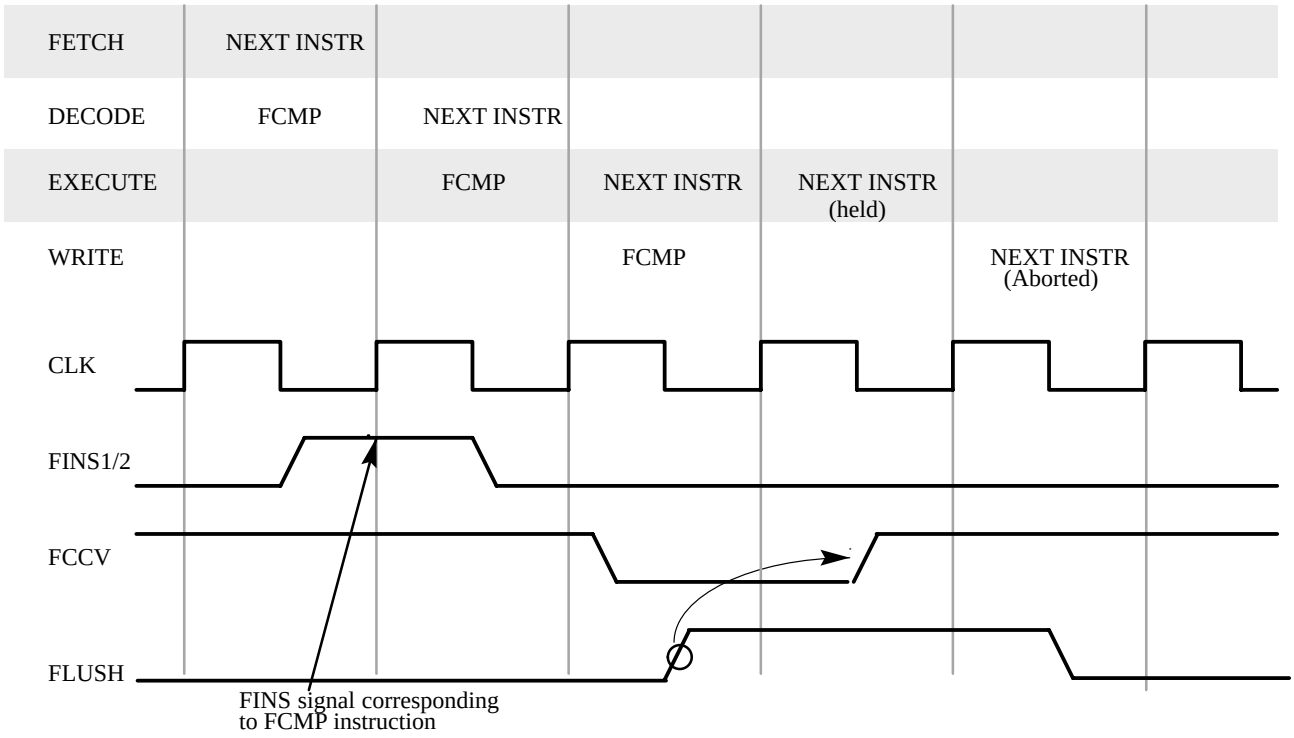


Figure 11. Effect of FLUSH on FCMP Instruction

3.2.2.1. Hold Signals

If $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$, or $\overline{\text{FHOLD}}$ is active, or FCCV or CCCV is inactive, the instruction pipelines of the TSC691E and TSC692E are frozen. $\overline{\text{FHOLD}}$ and FCCV are generated by the TSC692E, $\overline{\text{CHOLD}}$ and CCCV are generated by the coprocessor, and the others are generated by the system.

In the TSC692E, “freezing” or “holding” the instruction pipeline means that instructions that are still being tracked by the TSC691E are not allowed to continue executing. The instructions are allowed to continue execution when all of the hold signals are inactive and all of the condition code valid signals are active. Holds affect all load/store instructions, and only FPops which are in the F, D, E and W stages of the instruction pipeline. Hold signals do not affect the execution of a FPop in the FP queue.

3.2.2.2. Interlocking with $\overline{\text{FHOLD}}$

In some situations it is necessary to stop the TSC691E pipeline, either because a FP load/store instruction must be suspended due to an operand dependency, or because the TSC692E cannot accept any more instructions due to a resource dependency. $\overline{\text{FHOLD}}$ is used to freeze the instruction pipeline in these cases. Table 4. describes mandatory conditions under which $\overline{\text{FHOLD}}$ is asserted.

Operand dependencies listed in Table 4. apply to all FPops that are defined in the architecture. For example, suppose an unimplemented FPop is in the FP queue, waiting to cause an exception. If a store instruction is issued to the TSC692E to store the contents of the unimplemented FPop’s destination register, the store instruction must cause a $\overline{\text{FHOLD}}$ so that the wrong data is not stored. The unimplemented FPop eventually causes a trap that is taken by the TSC691E in the E stage of the store instruction.

The following simplification could be applied when handling all unimplemented FPops: when an unimplemented FPop has been issued to the TSC692E but has not yet caused a trap, assert $\overline{\text{FHOLD}}$ on the next floating–point instruction issued until $\overline{\text{FEXC}}$ is asserted. There is no loss in performance because any FPops entering the FP queue after the unimplemented FPop would be re–executed after the unimplemented FPop has been taken care of in the trap handler.

Table 4. $\overline{\text{FHOLD}}$ Resource/Operand Dependency Cases

Resource Dependencies:		
If the TSC692E will not have FP queue entry available to accommodate additional FPops, the TSC692E asserts $\overline{\text{FHOLD}}$ to stop the TSC691E from issuing any more instructions to the TSC692E.		
Operand Dependencies:		
LDF, LDDF	Load data from memory to f register	Load instructions must not overwrite the source or destination registers of any FPop that has not completed execution. In other words, the rd. field of the load instruction must not refer to the same *f* register as any valid rs1, rs2 or rd field of an outstanding FPop. The source registers of FPops (rs1, rs2) may not be altered because an FP exception trap would require that the source registers be unaltered for the trap handler.
STF, STDF	Store data from f register to memory	If a store instruction accesses an *f* register that is the destination register of an FPop that has not yet finished execution, the store instruction waits until all outstanding FPops with that register as a destination are complete.
LDFSR, STFSR	Load/store data between memory and floating–point status register	If any instructions are currently executing in the TSC692E when a LDFSR/STFSR instruction is issued by the TSC691E, the TSC692E holds until all instructions have completed execution and are no longer in the FP queue. If a LDFSR instruction is currently executing in the TSC692E when an FPop or STFSR is issued by the TSC691E, the TSC692E holds until LDFSR instruction has completed execution..
STDFQ	Store FP queue while qne=1 and in execution mode	If a STDFQ is issued by the TSC691E when the Floating-Point Queue is empty (qne=0) and the TSC692E is in execution mode, the TSC692E holds until STDFQ instruction has completed execution.
UNIMP	Unimplemented FP operation	If an unimplemented FPop has been issued to the TSC692E but has not yet caused a trap, the TSC692E holds on the next floating–point instruction issued by the TSC691E.

If the TSC692E goes into exception mode, $\overline{\text{FHOLD}}$ is deasserted. If there is a floating–point sequence error (see Section 3.3.3), $\overline{\text{FHOLD}}$ is asserted for two cycles. This is the only case where $\overline{\text{FHOLD}}$ is asserted in the exception mode.

If a floating–point trap condition occurs while $\overline{\text{FHOLD}}$ is asserted, $\overline{\text{FHOLD}}$ is deasserted at least one cycle after $\overline{\text{FEXC}}$ is asserted. Similarly, if FCCV is deasserted, it is reasserted at least one cycle after $\overline{\text{FEXC}}$ is asserted. For the $\overline{\text{FHOLD}}$ case, the TSC691E takes the FP trap on the FP instruction that triggered the $\overline{\text{FHOLD}}$.

3.2.2.3. FNULL Signal

FNULL is used to signal a pipeline delay of the TSC691E by the TSC692E. FNULL replaces FCCV and $\overline{\text{FHOLD}}$ for informing the system that the pipeline is being held. FNULL is asserted when either $\overline{\text{FHOLD}}$ is asserted or FCCV is deasserted. This signal is used as an input by the CC and MMU to monitor pipeline freezes initiated by the TSC692E.

3.3. TSC692E Programming Model

3.3.1. TSC692E Registers

The TSC692E has three types of user accessible registers: the *f* registers, the FP queue, and the Floating–point Status Register (FSR). The *f* registers are the TSC692E data registers. The FSR is the TSC692E status and operating mode register. The FP queue contains the TSC692E instruction that has started execution and is awaiting completion. The following section describes these registers in detail.

3.3.1.1. f Registers

The TSC692E provides 32 registers for floating–point operations, referred to as *f* registers. These registers are 32 bits in length, which can be concatenated to support 64–bit double words. Extended precision instructions are not supported in the TSC692E. Figure 13. illustrates the data organization for the *f* registers.

Integer and single precision data requires a single 32–bit *f* register. Double precision data requires 64 bits of storage and occupies an even–odd pair of adjacent *f* registers.

The TSC692E forces register addressing to match the data type specified by the floating–point instruction. This ensures data alignment in the *f* register file for double precision data. Figure 14. illustrates how the TSC692E uses the five register address bits in a floating–point instruction for the different types of data. Single data word transfers (integer,

single–precision floating–point) can be stored in any register. Consequently, all five bits of the register address specified in the floating–point instruction are valid. Double precision data must reside in an even–odd pair of adjacent registers. By ignoring the LSB of the register address for a FPop requiring a register pair, the TSC692E ensures data alignment.

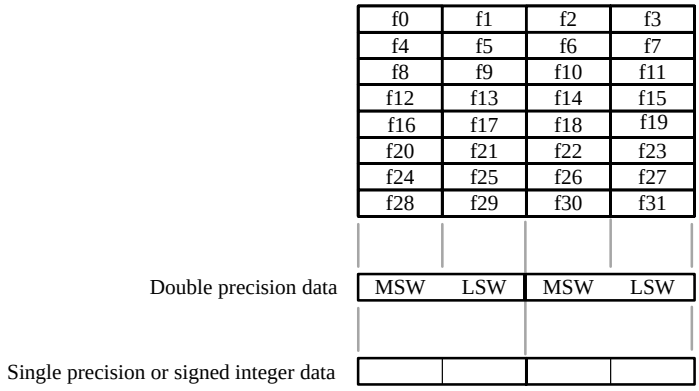


Figure 12. f Register Organization

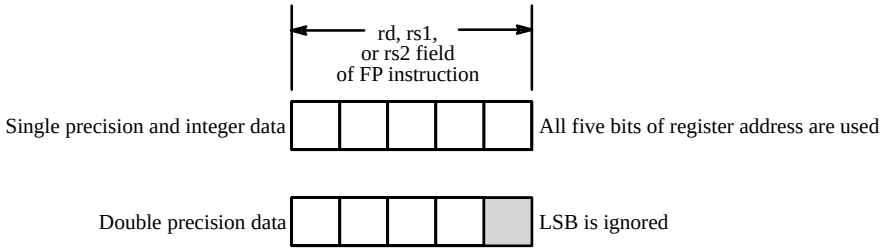


Figure 13. f Register Addressing

3.3.1.2. FP Queue

The TSC692E maintains a floating–point queue of the instruction that has started execution, but has yet to complete execution. The FP queue is used to accommodate the multiple clock nature of floating–point instructions and to support the handling of FP exceptions.

When the TSC692E encounters an exception case, it asserts $\overline{\text{FEXC}}$ and enters pending exception mode. The TSC692E remains in pending exception mode until the TSC691E encounters another floating–point instruction, at which time the TSC691E asserts the FXACK signal to force the TSC692E into exception mode. When the TSC692E enters the exception mode, floating–point execution halts until the FP queue is emptied. This allows the TSC691E to store the floating–point instructions under execution when the exception case occurred. Emptying the FP queue frees the TSC692E for use by the trap handler without losing the pre–exception state of the TSC692E.

The FP queue contains the 32–bit address and 32–bit FPop instruction of one instruction under execution. Floating–point load and store instructions and FP branch instructions are not queued. The entry of the FP queue is accessible by executing the store double floating–point queue (STDFQ) instruction. A load FP queue instruction does not exist, as the FP queue must be loaded by launching instructions.

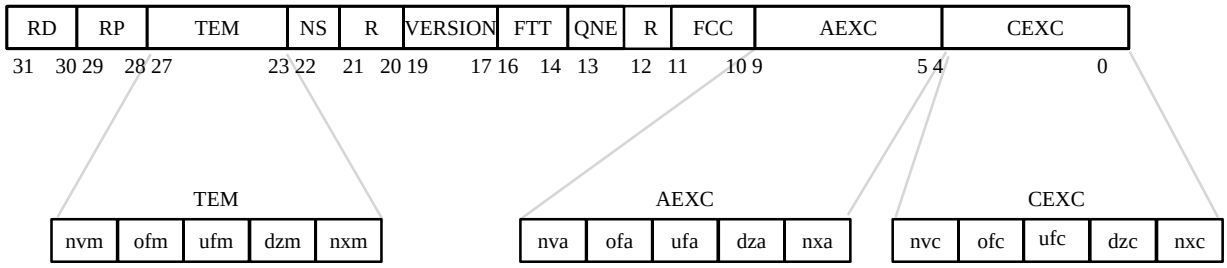


Figure 14. Floating–Point Status Register

3.3.1.3. Floating–Point Status Register (FSR)

The following paragraphs describe the bit fields of the floating–point status register (FSR). Refer to Table 5. (following page) for bit assignments for the FSR fields.

RD *FSR(31:30).* Rounding Direction: These two bits define the rounding direction used by the TSC692E during an FP arithmetic operation.

RP *FSR(29:28).* Unused - always set to 0.

TEM *FSR(27:23).* Trap Enable Mask: These five bits enable traps caused by FPops. These bits are ANDed (1= enable, 0= disable) with the bits of the CEXC (current exception field) to determine whether to force a floating–point exception to the TSC691E. All trap enable fields correspond to the similarly named bit in the CEXC field (see below). The TEM field only affects which bits in the CEXC field will cause the $\overline{\text{FEXC}}$ signal to be asserted.

NS *FSR(22).* Non–Standard floating point: This bit is always set to 0 (IEEE mode).

version *FSR(19:17).* The version number is used to identify the SPARC floating–point processor type. This field is set to 100 (4H) for the TSC692E, and is read–only.

FTT *FSR(16:14).* Floating–point Trap Type: This field identifies the floating point trap type of the current FP exception. This field can be read and written, and must be cleared by software.

QNE *FSR(13).* Queue Not Empty: This bit signals whether the FP queue is empty. (0= empty, 1= not empty)

FCC *FSR(11:10).* Floating–point Condition Codes: These two bits report the FP condition codes (see Table 5.).

AEXC *FSR(9:5).* Accumulated EXCeptions: This field reports the accumulated FP exceptions that are masked by the TEM field. All masked exception cases are ORed with the contents of the AEXC and accumulated as status. All accumulated fields have the same definition as the corresponding field for CEXC (see below). This field can be read and written, and must be cleared by software (see Table 5.).

CEXC *FSR(4:0).* Current EXCeptions: This field reports the current FP exceptions. This field is automatically cleared upon the execution of the next floating–point instruction. CEXC status is not lost upon assertion of a floating–point exception, because instructions following a valid exception are not executed by the TSC692E. The five CEXC bits are:

- *nvc = 1* indicates invalid operation exception. This is defined as an operation using an improper operand value. An example of this is 0/0.
- *ofc = 1* indicates overflow exception. The rounded result would be larger in magnitude than the largest normalized number in the specified format.
- *ufc = 1* indicates underflow exception. The rounded result is inexact, and would be smaller in magnitude than the smallest normalized number in the indicated format.
- *dzc = 1* indicates division–by–zero: X/0, where X is subnormal or normalized. Note that 0/0 does not set the dzc bit.
- *nxc = 1* indicates inexact exception. The rounded result differs from the infinitely precise correct result.

R *FSR 21,20 and 12.* Reserved - always set to 0.

Table 5. Floating–Point Status Register Summary

Field	Values	FSR bits	Description	Loadable by LDFSR
RD	0 - Round to nearest (tie-even)	31:30	Rounding Direction	yes
	1 - Round to 0			
	2 - Round to +			
	3 - Round to -			
RP	Unused Bits	29:28	Unused always set to 0	no
TEM	0 - Disable trap	27:23	Trap Enable Mask	yes
	1 - Enable trap			
	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	0 - Disable 1 - Enable	22	Non–standard Floating–point: 0 = IEEE mode; multiplier and ALU generate denormalized operand exceptions and produce unrounded normalized values on underflow exceptions. 1 = FAST mode; multiplier and ALU flush denormalized operands to zero and round underflow results to zero.	no
version	0 - 7	19:17	FPU version number	no
FTT	0 - None	16:14	Floating–point trap type	no
	1 - IEEE Exception			
	2 - Unfinished FPop			
	3 - Unimplemented FPop			
	4 - Sequence Error			
	5 - Data Bus Error			
	6 - Restartable Error			
	7 - Non-Restartable Error			
QNE	0 - queue empty	13	Queue Not Empty	no
FCC	0 - =	11:10	Floating–point Condition Codes	yes
	1 - <			
	2 - >			
	3 - Unordered			
AEXC		9:5	Accrued Exception Bits	yes
	NVA	9	accrued invalid exception	
	OFA	8	accrued overflow exception	
	UFA	7	accrued underflow exception	

Field	Values	FSR bits	Description	Loadable by LDFSR
	DXA	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	
r	Always set to 0	21, 20, 12	reserved bits	no

3.3.2. TSC692E Floating–Point Instructions

SPARC floating–point instructions are separated into three groups: floating–point load/store, floating–point branch (FBfcc), and floating–point operate instructions (FPops). Floating–point load/store instructions are used to transfer data to and from the data registers (*f* registers). FP load/store instructions also allow the TSC691E integer unit to read and write the floating–point status register (FSR) and to read the entry of the floating–point queue. Floating–point load and store instructions are executed by both the TSC691E and the TSC692E; the TSC691E supplying all address and control signals for memory access and the TSC692E loading or storing the data.

Floating–point branch (FBfcc) instructions (and coprocessor branch instructions (CBccc)) are executed by the TSC691E, since the TSC691E is responsible for generating address and control signals for memory access. Conditional FBfcc branches are based upon the FCC[1:0] signals supplied by the TSC692E. FCC[1:0] is set by executing a FCMP instruction, which belongs to the FPop group of instructions. Floating–point branch instructions will cause the TSC691E to recognize a pending floating–point exception in the same manner as other floating–point instructions (see Section 3.3.3).

FPops include all other floating–point instructions executed by the TSC692E. Floating–point operate instructions (FPops) include basic numeric operations (add, subtract, multiply, and divide), conversions between data types, register to register moves, and floating–point number comparison. FPops operate only on data in the floating–point registers.

The SPARC architecture supports four data types: 32–bit signed integer, single–precision FP, double–precision FP, and extended–precision FP. Extended precision instructions are defined in the SPARC architecture, but are not supported in the TSC692E. The TSC692E supports execution of extended precision floating–point instructions by asserting an unimplemented instruction trap. This allows the TSC691E to trap to a software emulation of extended precision floating–point.

Seven load/store instructions are executed by the TSC692E. The following describes the TSC692E load/store instructions:

LDF and LDDF transfer data from memory to *f* registers 32 and 64 bits at a time, respectively.

STF and STDF transfer data from the *f* registers to memory in data widths of 32 and 64 bits.

LDFSR and STFSR allow the FSR to be read and written to.

STDFQ is a privileged instruction which allows the FP queue to be read.

All FPops operate only on data located in the *f* registers. The FPops are divided into four groups: basic arithmetic operations, compares, format conversions, and register–to–register moves. Move operations do not cause exceptions. The converts, moves and the square root instruction use only a single source operand. FP compare instructions modify only the FCC[1:0] signals. FPops are dispatched in one cycle in the TSC691E, and require multiple cycles to execute in the TSC692E.

Table 6. and Table 7. illustrate the TSC692E instructions and their execution cycle count. For further information on the SPARC floating–point instructions, please refer to Chapter 6, SPARC Instruction Set.

Table 6. Floating–Point Load and Store Instruction Cycle Count

Mnemonic	Operation	Cycles
LDF	load floating–point	2
LDDF	load double floating–point	3
LDFSR	load FSR	2
STF	store floating–point	3
STDF	store floating–point double	4
STFSR	store FSR	3
STDFQ	store double FP queue	4

Table 7. Floating–Point Operate (FPops) Instruction Cycle Count

Mnemonic	Operation	Cycles[a] Min.	Max.	Typ.
FABSs	absolute value	2	2	2
FADDs	add single	4	4	17
FADDd	add double	4	4	17
FCMPs	compare single	4	4	15
FCMPd	compare double	4	4	15
FCMPEs	compare single and exception if unordered	4	4	15
FCMPEd	compare double and exception if unordered	4	4	15
FDIVs	divide single	6	20	38
FDIVd	divide double	6	35	56
FMOVs	move	2	2	2
FMULs	multiply single	5	5	25
FMULd	multiply double	7	9	32
FNEGs	negate	2	2	2
FSQRTs	square root single	6	37	51
FSQRTd	square root double	6	65	80
FSUBs	subtract single	2	4	17
FSUBd	subtract double	4	4	17
FdTOi	convert double to integer	7	7	14
FdTOs	convert double to single	3	3	16
FiTOs	convert integer to single	5	6	13
FiTOd	convert integer to double	4	6	13
FsTOi	convert single to integer	6	6	13
FsTOd	convert single to double	2	2	14

[a]. These cycle counts assume that the operands are available in the register file. A load-use interlock may add up to 2 cycles to the typical cycle count.
[b]. Max. Cycles is for NaN and Denormalized subresults.

3.3.3. TSC692E Internal Operation

The TSC692E operates in one of three modes: execution mode, pending exception mode, and exception mode (see Figure 16. *).* After reset, the TSC692E enters execution mode, which is the normal mode of operation. When the TSC692E encounters a floating–point exception condition, the TSC692E asserts $\overline{\text{FEXC}}$ and enters the pending exception mode. All FPop instructions under execution at this point are suspended. The TSC691E asserts FXACK and enters the floating–point trap when the next floating point instruction is encountered. Upon receiving FXACK, the TSC692E FPU enters exception mode. The TSC692E returns to execution mode as soon as the trap handler empties the FP queue using STore Double Floating–point Queue instructions (STDFQ).

3.3.3.1. Exception Handling

Upon encountering an exception condition, the TSC692E asserts $\overline{\text{FEXC}}$ to notify the TSC691E that a floating–point exception has occurred and enters the pending exception mode. The TSC691E enters the trap handler on the next

floating–point instruction it encounters in the instruction stream, asserting FXACK to signal to the TSC692E that the trap is being taken. At this point, the TSC692E enters exception mode and the FP queue contains instruction and address of the FP operation which caused the FP exception (see Figure 16.).

Upon receiving FXACK from the TSC691E, the mode of the TSC692E changes from pending exception to exception mode. An FP exception can only be caused while the FPU is moving between these two modes or by executing STDFQ when FP queue is empty (qne field in FSR equal to 0). All FPops in the TSC692E stop executing during pending exception and exception modes. While in exception mode, the TSC692E will execute only store floating–point instructions until the FP queue is emptied. All floating–point store instructions are allowed while in this operating mode (particularly STDFQ and STFSR) and they cannot cause an exception trap. Any load or FPop issued to the TSC692E while in this mode causes a sequence error and returns the TSC692E to exception pending mode and sets the ftt sequence error field in FSR. The instruction that caused the sequence error is not entered into the FP queue. Once the queue is emptied by STDFQ instruction, the TSC692E returns to execution mode.

If a STDFQ instruction is executed when the FP queue is empty (qne field in FSR equal to 0, FPU in execute mode), the FPU generates an immediate trap and sets the ftt field in FSR to sequence error, but the FPU remains in the execute mode.

Figure 17. illustrates the handshake of signals between the TSC691E and the TSC692E during a floating–point exception. The qne (queue not empty) bit of the FSR is shown in Figure 17. to illustrate the dependency of clearing the FP queue to return to execution mode.

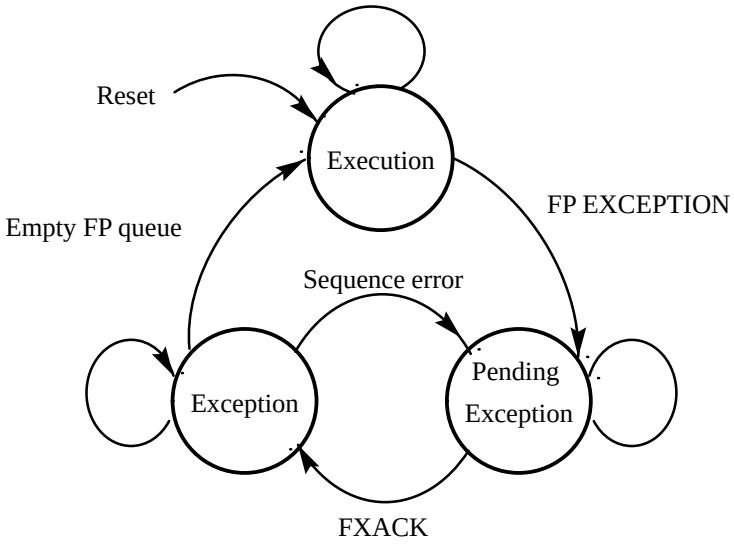


Figure 15. FPU Operation Modes

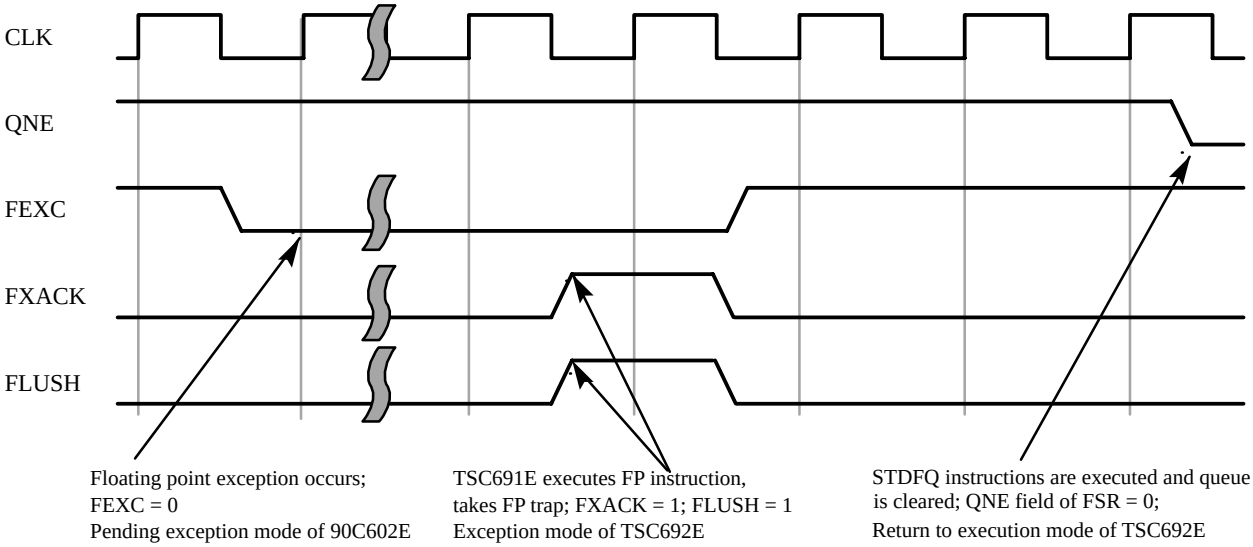


Figure 16. Floating–Point Exception Handshake

3.3.4. TSC692E IEEE–754 Compliance

The TSC692E meets the requirements of the IEEE Std. 754–1985 for floating–point arithmetic. Accuracy of the results of its operations are within ±½ LSB, as specified by the IEEE standard. The following sections describe the IEEE format as implemented on the TSC692E.

3.3.4.1. IEEE Definitions

The following terms are used extensively in describing the IEEE–754 floating–point data formats. This section is directly quoted from the *IEEE Standard for Binary Floating–Point Arithmetic.*

biased exponent

The sum of the exponent and a constant (bias) chosen to make the biased exponent's range nonnegative. (Note in the remainder of this section, the term "exponent" refers to a biased exponent.)

binary floating–point number

A bit string characterized by three components: a sign, a signed exponent and a significand. Its numerical value, if any, is the signed product of its significand and two raised to the power of its exponent.

Denormalized

Denormalized numbers are those numbers whose magnitude is smaller than the smallest magnitude representable in the format. They have a zero exponent and a denormalized non–zero fraction. Denormalized fraction means that the hidden bit is zero.

The TSC692E can directly operate on denormalized operands. The TSC692E never assert an unfinished FPop exception when an operation results in a denormalized number.

denormalized number

(DNRM) A non–zero floating–point number whose exponent has a reserved value, usually the format's minimum, and whose explicit or implicit leading significand bit is zero. (Denormalized numbers are also referred to as subnormal in this text.)

fraction

The field of the significand that lies to the right of its implied binary point.

NaN

Not a number, a symbolic entry encoded in floating–point format. They are used to signal invalid operations and as a way of passing status information through a series of calculations. NaNs arise in one of two ways: they can be generated by the TSC692E upon an invalid operation or they may be supplied by the user as an input

operand. NaN is further subdivided into two categories: quiet and signaling. Signaling NaNs signal the invalid operation exception whenever they appear as operands. Quiet NaNs propagate through almost every arithmetic operation without signaling exceptions.

Normalized

Most calculations are performed on normalized numbers. For single–precision, they have a biased exponent range of 1 to 255, which results in a true exponent range of -126 to +127. The normalized number type implies a normalized significand (hidden bit is 1).

significand

The component of a binary floating–point number that consists of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.

true exponent

The component of a binary floating–point number that normally signifies the integer power to which 2 is raised in determining the value of the represented number.

Zero

The IEEE zero has all fields except the sign field equal to zero. The sign bit determines the sign of zero (i.e., the IEEE format defines a +0 and a -0).

3.3.4.2. IEEE Floating–point Data Formats

The TSC692E directly supports single– and double–precision floating–point data formats. Extended–precision instructions (non-implemented) encountered by the TSC692E cause an unimplemented instruction trap to be asserted by the TSC692E. This allows software to emulate extended–precision instructions through the use of a trap handler. Single– and double-precision formats are described in this section.

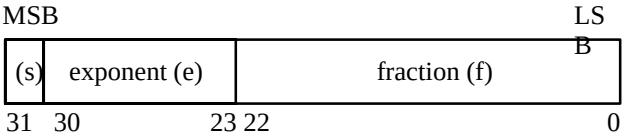


Figure 17. Single–Precision Floating–Point Format

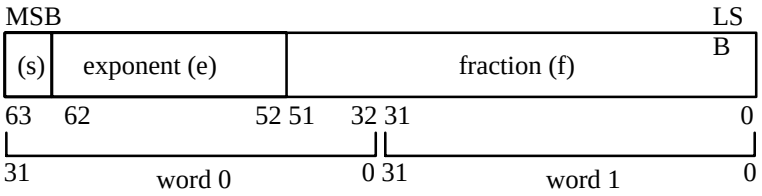


Figure 18. Double–Precision Floating–Point Format

3.3.4.2.1. Single–Precision Floating–Point

Single–precision floating–point data are 32–bits wide and consist of three fields: a single sign bit (s), an eight–bit biased exponent (e), and a 23–bit fraction (f). Figure 18. illustrates the single–precision floating–point format.

The IEEE standard defines single–precision floating–point numbers according to the following conventions:

(+0, -0)	If e = 0 and f = 0, then the value V = $(-1)^{s}$ * (0) Note that two representations of zero exist, one positive and one negative
DNRM (denormalized)	If e = 0 and f ≠ 0, then the value V = DNRM
Normalized	If 0 < e < 255, then value V = $(-1)^{s}$ * (2^{e-127}) * (1.f) Note that 1.f is the significand. The one to the left of the binary point is the so–called “hidden bit.” This bit is not

stored as part of the floating–point word; it is implied. For a number to be normalized, it must have this one to the left of the binary point.

(+∞, –∞) If e = 255 and f = 0, then value V = $(-1)^s * \infty$

NaN (not a number) If e = 255 and f ≠ 0, then value V = NaN.

The value is a quiet NaN if the first bit of the fraction is 1, and a signaling NaN if the first bit of the fraction is 0 (at least one bit must be non–zero).

3.3.4.2.2. Double–Precision Floating–Point

Double–precision floating–point data are 64–bits wide and consist of three fields: a single sign bit (s), an eleven–bit biased exponent (e), and a 52–bit fraction (f). Figure 19. illustrates the double–precision floating–point format.

The IEEE standard defines double–precision floating–point numbers according to the following conventions:

(+0, -0) If e = 0 and f = 0, then value V = $(-1)^s * (0)$

DNRM If e = 0 and f ≠ 0, then value V = DNRM

Normalized If 0 < e < 2047, then value V = $(-1)^s * (2^{e-1023}) * (1.f)$

(+∞, –∞) If e = 2047 and f = 0, then value V = $(-1)^s * \infty$

NaN If e = 2047 and f ≠ 0, then value V = NaN.

The value is a quiet NaN if the first bit of the fraction is 1, and a signaling NaN if the first bit of the fraction is 0 (at least one bit must be non–zero).

3.3.5. NaN Format

The TSC692E uses different NaN format. Table 8. and Table 9. give returned values for untrapped floating-point result.

Table 8. Untrapped FP result in same format as operand

RS2, RS1	number	QNaN2	SNaN2
none	IEEE 754	QNaN2	ME_NaN
number	IEEE 754	QNaN2	ME_NaN
QNAN1	QNaN1	QNaN1	ME_NaN
SNAN1	ME_NaN	ME_NaN	ME_NaN

- QNaN results have their sign bit equal to 0.
- ME_NaN is 0x 7fff 0000 (Single Precision)
- ME_NaN is 0x 7fff e000 0000 0000 (Double Precision)

Table 9. Untrapped FP result in different format

RS2 Operation	+QNaN	-QNaN	+SNaN	-SNaN
fstoi	+imax	-imax	+imax	-imax
fstod	(QNaN2)	(QNaN2)	ME_NaN	ME_NaN
fdtos	ME_NaN	ME_NaN	ME_NaN	ME_NaN
fdtoi	+imax	-imax	+imax	-imax

- -imax is 0x 8000 0000
- +imax is 0x 7fff ffff
- (QNaN2) is a copy of the mantissa bits of the operand, with the extra low
- order bits zeroed, and the sign bit zeroed

3.3.6. TSC692E Exception Cases

The following section describes the TSC692E exception cases, including exceptions specified by the IEEE–754 standard.

Unfinished FPop. In IEEE-754 standard, this exception case can occur when operations on normalized floating–point numbers either encounter a denormalized operand or produce a denormalized result. This exception case is asserted upon executing any FPop encountering a NaN as one of the operands. The TSC692E never asserts this exception since all implemented instructions are executed within hardware.

Unimplemented FPop. This exception is asserted by the TSC692E upon encountering a defined SPARC FPop instruction that is not supported by the TSC692E. This includes all operations using extended–precision format operands. The trap handler is expected to emulate the unimplemented instruction.

Sequence Error. This exception is asserted by the TSC692E when a floating–point instruction (other than FP store) is attempted after the TSC692E has entered either pending exception or exception mode. The TSC692E suspends all instruction execution with the exception of FP stores until the FP exception has been acknowledged and the FP queue has been cleared.

IEEE Exceptions. This class of exceptions is defined as part of the IEEE–754 Standard. The five exceptions defined as IEEE Exceptions are reported in the CEXC and AEXC fields of the FSR. These exceptions are: invalid, overflow, underflow, division–by–zero, and inexact. The only exceptions that can coincide are inexact with overflow and inexact with underflow. The following paragraphs discuss these exception cases.

Invalid Operation. The invalid operation exception is signaled if an operand is invalid for the operation to be performed. The result, when the exception occurs without a trap, shall be a quiet NaN provided the destination has a floating–point format. The invalid operations are

1 - Any operation on a signaling NaN

2 - Addition or subtraction: Magnitude subtraction of infinities such as (+) + (-)

3 - Multiplication: 0 x ∞

4 - Division: 0/0 or ∞ / ∞

5 - Square root if the operand is less than zero

6 - Conversion of a binary floating–point number to an integer or decimal format when overflow, infinity, or NaN precludes a faithful representation in that format and this cannot otherwise be signaled

7 - Floating–point compare operations: when one or more of the operands are NaN

Division–by–zero.

If the divisor is zero and the dividend is a finite nonzero number, then the division by zero exception shall be signaled. The result, when no trap occurs, shall be a correctly signed 1.

Overflow.

The overflow exception shall be signaled whenever the destination format's largest finite number is exceeded in magnitude by what would have been the rounded floating–point result were the exponent range unbounded. The result, when no trap occurs, shall be determined by the rounding mode and the sign of the intermediate result as follows:

1- Round to nearest carries all overflows to 1 with the sign of the intermediate result

2 - Round toward 0 carries all overflows to the format's largest finite number with the sign of the intermediate result.

3 - Round toward - carries positive overflows to the format's largest positive finite number, and carries negative overflows to $-\infty$.

4 - Round toward + carries negative overflows to the format's most negative finite number, and carries positive overflows to $+\infty$.

Underflow.

The TSC692E asserts an underflow exception when the rounded result is inexact and would be smaller in magnitude than the smallest normalized number in the specified format.

Inexact.

The inexact exception is generated whenever there is a loss of accuracy (or significance) in the result. The TSC692E computes results to higher precision than the number of fraction bits in the format. If any of the fraction bits to the right of the LSB was one prior to rounding, the inexact exception is signaled.

3.4. TSC692E Signal Descriptions

The following sections describe the external signals of the TSC692E. Active low signals are marked with an overbar, active high signals are not.

3.4.1. Integer Unit Interface Signals

FP *active-low output* **(Floating–point Present):**

This signal indicates to the TSC691E that a FPU is present in the system. In the absence of a FPU, this signal is pulled up to VCC by a resistor. This is a static signal; it always asserts a low output. The TSC691E generates a floating–point disable trap if FP is not asserted during the execution of a floating–point instruction. FP is three-state output controlled by $\overline{\text{TOE}}$ signal.

FCC[1:0] *output* **(Floating–point Condition Codes):**

The FCC[1:0] bits indicate the current condition code of the FPU, and are valid only if FCCV is asserted. FBfcc instructions use the value of these bits during the execute cycle if they are valid. If the FCC[1:0] bits are not valid, then FCCV is released, which $\overline{\text{HALT}}$s the TSC691E until the FCC bits become valid. FCC[1:0] is three-state output controlled by $\overline{\text{TOE}}$ signal.

Table 10. FCC[1:0] Condition Codes

FCC1	FCC0	Condition
0	0	equal
0	1	Op1 < Op2
1	0	Op1 > Op2
1	1	Unordered

FCCV *output* **(Floating–point Condition Codes Valid):**

The TSC692E asserts the FCCV signal when the FCC[1:0] represent a valid condition. The FCCV signal is deasserted when a pending floating–point compare instruction exists in the floating–point queue. FCCV is reasserted when the compare instruction is completed and FCC bits are valid. FCCV is three-state output controlled by $\overline{\text{TOE}}$ signal.

$\overline{\text{FHOLD}}$ *output* **(Floating–point HOLD):**

The $\overline{\text{FHOLD}}$ signal is asserted by the TSC692E if it cannot continue execution due to a resource or operand dependency. The TSC692E checks for all dependencies in the decode stage, and if necessary, asserts $\overline{\text{FHOLD}}$ in the next cycle.

The $\overline{\text{FHOLD}}$ signal is used by the TSC691E to freeze its pipeline in the same cycle. The TSC692E must eventually de–assert $\overline{\text{FHOLD}}$ to release the TSC691E pipeline. $\overline{\text{FHOLD}}$ is three-state output controlled by $\overline{\text{TOE}}$ signal.

$\overline{\text{FEXC}}$ *output* **(Floating–point EXCeption):**

The $\overline{\text{FEXC}}$ is asserted if a floating–point exception has occurred. It remains asserted until the TSC691E acknowledges that it has taken a trap by asserting FXACK.

Floating–point exceptions are taken only during the execution of a floating–point instruction. The TSC692E releases $\overline{\text{FEXC}}$ when it receives FXACK. $\overline{\text{FEXC}}$ is three-state output controlled by $\overline{\text{TOE}}$ signal.

$\overline{\text{FEXC}}$ is also asserted when an error occurs (in that case $\overline{\text{HWERROR}}$ will be also asserted).

FIPAR *output* **(Floating–point Unit to Integer Unit Control Parity):**

This signal contains the odd parity over the FCC[1:0], FCCV, $\overline{\text{FEXC}}$ and $\overline{\text{FHOLD}}$ bits. The parity bit is generated by the FPU and will be checked by the IU. FIPAR is three-state output controlled by $\overline{\text{TOE}}$ signal.

FXACK *input* **(Floating–point eXception ACKnowledge):**

The FXACK signal is asserted by the TSC691E to acknowledge to the TSC692E that the current FP trap is taken.

INST *input* **(INSTruction fetch):**

The INST signal is asserted by the TSC691E whenever a new instruction is being fetched. It is used by the TSC692E to latch the instruction on the D[31:0] bus into the FPU instruction buffer.

The TSC692E has two instruction buffers (D1 and D2) to save the last two fetched instructions (see Figure 3.). When INST is asserted, the new instruction enters the D1 buffer and the old instruction is pushed into the D2 buffer.

FINS1 *input* **(Floating–point INStruction in buffer 1):**

The FINS1 signal is asserted by the TSC691E during the decode stage of a FPU instruction if the instruction is stored in the D1 buffer of the TSC692E. The TSC692E uses this signal to launch the instruction in the D1 buffer into its execute stage instruction register.

FINS2 *input* **(Floating–point INStruction in buffer 2):**

The FINS2 signal is asserted by the TSC691E during the decode stage of a FPU instruction if the instruction is stored in the D2 buffer of the TSC692E. The TSC692E uses this signal to launch the instruction in the D2 buffer into its execute stage instruction register.

FLUSH *input* **(Floating–point instruction fLUSH):**

The FLUSH signal is asserted by the TSC691E to signal to the TSC692E to flush the instructions in its instruction registers. This may happen when a trap is taken by the TSC691E. The TSC691E will restart the flushed instructions after returning from the trap.

FLUSH has no effect on instructions in the floating–point queue. In addition to freezing the FPU pipeline, the TSC692E uses FLUSH to shut off the D bus drivers during store operations. To ensure correct operation of the TSC692E, FLUSH must not change state more than once during a clock cycle.

IFPAR *input* **(Integer Unit to Floating–point Unit Control Parity):**

This signal contains the odd parity over the FINS1, FINS2, FLUSH, FXACK and INST bits. The parity bit is generated by the IU and will be checked by the FPU.

3.4.2. Coprocessor Interface Signals

$\overline{\text{CHOLD}}$ *input* **(Coprocessor HOLD):**

The $\overline{\text{CHOLD}}$ signal is asserted by the coprocessor if it cannot continue execution. The coprocessor must check all dependencies in the decode stage of the instruction and assert the $\overline{\text{CHOLD}}$ signal, if necessary, in the next cycle. The coprocessor must eventually de–assert this signal to unfreeze the TSC691E and TSC692E pipelines. The $\overline{\text{CHOLD}}$ signal is latched with a transparent latch in the TSC692E before it is used.

CCCV *input* **(Coprocessor Condition Codes Valid):**

The coprocessor asserts the CCCV signal when the CCC[1:0] represent a valid condition. The CCCV signal is deasserted when a pending coprocessor compare instruction exists in the coprocessor queue. CCCV is reasserted when the compare instruction is completed and the CCC[1:0] bits are valid. The TSC692E will enter a wait state if CCCV is deasserted. The CCCV signal is latched with a transparent latch in the TSC692E before it is used.

3.4.3. System/Memory Interface Signals

A[31:0] *input* **(Address bus [31:0]):**

The address bus for the TSC692E is an input–only bus. The TSC691E supplies all addresses for instruction and data fetches for the TSC692E. The TSC692E captures addresses of floating–point instructions from the A[31:0] bus into the DDA register. When INST is asserted by the TSC691E, the contents of the DDA is transferred to the DA1 register.

APAR *input* **(Address Bus Parity):**

This signal is used by the FPU to check the odd parity over the 32-bit address.

D[31:0] *input/output* **(Data bus [31:0]):**

The D[31:0] bus is driven by the FPU only during the execution of floating–point store instructions. The store data is sent out unlatched and must be latched externally before it is used. Once latched, store data is valid during the second data cycle of a store single access and on the second and third data cycle of a store double access. The data alignment for load and store instructions is done inside the FPU. A double word is aligned on an eight–byte boundary. A single word is aligned on a four–byte boundary. When output, D[31:0] is three-state controlled by $\overline{\text{DOE}}$ signal.

DPAR *bidirectional* **(Data Bus Parity):**

This signal contains the odd parity over the 32-bit bidirectional data bus. In case of store data operations the parity bit is generated and launched in parallel by the FPU. In case of load data operations the parity is checked by the FPU. When output, DPAR is three-state controlled by $\overline{\text{DOE}}$ signal

$\overline{\text{DOE}}$ *input* **(Data Output Enable):**

The $\overline{\text{DOE}}$ signal is connected directly to the data output drivers and DPAR driver and must be asserted during normal operation. Deassertion of this signal three–states all output drivers on the data bus and DPAR signal. This signal should be deasserted only when the bus is granted to another bus master, i.e, when either $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$, $\overline{\text{MHOLDA}}$, or $\overline{\text{MHOLDB}}$ is asserted.

$\overline{\text{TOE}}$ *input* **(Test Output Enable):**

The $\overline{\text{TOE}}$ signal allows to disable all output control signals (except TAP signals): FNULL, FP, FCC[1:0], FCCV, $\overline{\text{FEXC}}$, $\overline{\text{FHOLD}}$, FIPAR, $\overline{\text{HWERROR}}$ and $\overline{\text{MCERR}}$.

$\overline{\text{MHOLDA}}$, **$\overline{\text{MHOLDB}}$** *input* **(Memory HOLD):**

Asserting $\overline{\text{MHOLDA}}$ or $\overline{\text{MHOLDB}}$ freezes the TSC692E pipeline. Either $\overline{\text{MHOLDA}}$ or $\overline{\text{MHOLDB}}$ is used to freeze the FPU (and the IU) pipelines during a cache miss (for systems with cache) or when slow memory is accessed.

$\overline{\text{BHOLD}}$ *input* **(Bus HOLD):**

This signal is asserted by the system's I/O controller when an external bus master requests the data bus. Assertion of this signal will freeze the FPU pipeline. External logic should guarantee that after de–assertion of $\overline{\text{BHOLD}}$, the state of all inputs to the chip is the same as before $\overline{\text{BHOLD}}$ was asserted.

$\overline{\text{MDS}}$ *input* **(Memory Data Strobe):**

The $\overline{\text{MDS}}$ signal is used to load data into the FPU when the internal FPU pipeline is frozen by assertion of $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$.

FNULL *output* **(FPu NULLify cycle):**

This signal signals to the memory system when the TSC692E is holding the instruction pipeline of the system. This hold would occur when $\overline{\text{FHOLD}}$ is asserted or FCCV is deasserted. This signal is used by the memory system in the same fashion as the integer unit's INULL signal. The system needs this signal because the IU's INULL does not take into account holds requested by the FPU. FNULL is three-state output controlled by $\overline{\text{TOE}}$ signal.

$\overline{\text{RESET}}$ *input* **(RESET):**

Asserting the $\overline{\text{RESET}}$ signal resets the pipeline and sets all registers (except f registers) and the writable fields of the FSR to zero. The $\overline{\text{RESET}}$ signal must remain asserted for a minimum of nine cycles. It is protected by a glitch removal filter and pulses which are so short that they are detected only during one clock period are not influencing the FPU. $\overline{\text{RESET}}$ signal is also protected with two-rail coding and an error detected will lead to a trap and indicate Internal Parity Error.

$\overline{\text{HWERROR}}$ *output* **(ERROR State):**

This signal is asserted whenever an error occurs in the FPU. The FPU will enter the exception pending mode and will assert $\overline{\text{FEXC}}$. $\overline{\text{HWERROR}}$ is deasserted when FTT field in FSR is changed.

$\overline{\text{CMODE}}$ *input* **(Master/checker operation):**

Assertion of this signal sets the FPU to act as a checker only in a master/checker configuration. All output signal except $\overline{\text{HWERROR}}$, $\overline{\text{MCERR}}$ and TAP signals will be high «Z». $\overline{\text{CMODE}}$ is a static signal and will not change when running.The $\overline{\text{CMODE}}$ signal may only be changed when the $\overline{\text{RESET}}$ and/or the $\overline{\text{HALT}}$ signal is asserted.

$\overline{\text{MCERR}}$ *output* **(Comparison Error):**

This signal is asserted in checker mode when a comparison error occurs on the internal signals vis-a-vis the output signals of the master FPU. In single mode, this signal is asserted when a stuck-at fault is detected between pin and output buffer. It is deasserted when the error disappears.

$\overline{\text{602MODE}}$ *input* **(Normal $\overline{\text{602MODE}}$ Operation):**

Forcing this input low disables the parity checking of all input signals. This means the TSC692E will operate with standard input signals.Nevertherless, internal parity check remains active and parity on the data and address bus is generated internally. $\overline{\text{602MODE}}$ is a static signal and will not change when running. The $\overline{\text{602MODE}}$ signal may only be changed when the $\overline{\text{RESET}}$ and/or the $\overline{\text{HALT}}$ signal is asserted.

$\overline{\text{HALT}}$ *input* **($\overline{\text{HALT}}$ Mode):**

When asserted this input will freeze the FPU pipeline and the clock. All information placed in the registers of the FPU remains unchanged. By deasserting $\overline{\text{HALT}}$, execution of the FPU will resume.

Data valid on output buffers before $\overline{\text{HALT}}$ was asserted are restored after deassertion of $\overline{\text{HALT}}$.

When the FPU is in $\overline{\text{HALT}}$ mode, the TAP is still operating.

3.4.4. TAP signals

TCLK *input* **(JTAG Test Clock)**

JTAG test clock input.

TMS *input* **(JTAG Test Mode Select)**

The TMS signal is interpreted by the TAP controller to control test operations. Received signals are sampled at the rising edge of the TCLK signal.

TDI *input* **(JTAG Test Data Input)**

Serial input data applied to this port is fed either into the instruction register or into a test data register, depending on the sequence previously applied to the TMS signal. Received input data is sampled at the rising edge of the TCLK signal.

$\overline{\text{TRST}}$ *input* **(JTAG Test $\overline{\text{RESET}}$)**

The TAP's test logic is $\overline{\text{RESET}}$ when a logical 0 is applied to this port.

TDO *output* **(JTAG Test Data Output)**

Depending on the sequence previously applied to the TMS input, the content of either the instruction register or the data register are serially shifted out toward the TDO output. Data out of TDO are clocked at the falling edge of the TCLK signal. TDO should be in the inactive state except when scanning is in progress (use of three-state driver).

3.4.5. Power and Clock Signals

CLK *input* **(CLocK):**

The CLK signal is used for clocking the FPU's pipeline registers. It is high during the first half of the processor cycle and low during the second half. The rising edge of CLK defines the beginning of each pipeline stage in the FPU.

VCCO, VCCI, VCCT *input* **(Power):**

These pins provide +5V power to various sections of the processor. Power is supplied on three different busses to provide clean, stable power to each section: output drivers, main internal circuitry, and the input circuits.

VCCO pins supply the output driver;

VCCI pins supply main internal circuitry;

VCCT pins supply the input circuit.

VSSO, VSSI, VSST *input* **(Ground):**

These pins provide ground return for the power signals. Ground is supplied on three different busses to match the power signals to each section:

VSSO pins for the output driver;

VSSI pins for the main internal circuitry;

VSST pins for the input circuit bus.

4. Fault Tolerant and Test MECHANISM.

FAULT TOLERANT MECHANISM:

- Parity checking on 100% of the total number of latches with hardware error traps
- Parity checking of address, data pads and control pads
- Master/Checker operation
- Parity on odd and even bits of the register file for a better detection of SEU
- Fabricated using Atmel Wireless & Microcontrollers Space hardened 0.8 µm SCMOS technology

TEST MECHANISM:

- IEEE Standard Test Access Port & Boundary-Scan Architecture
- Internal Scan Path to test the internal parity error detection during off-line test
- Possibility to $\overline{\text{HALT}}$ the FPU by an external signal

4.1. Fault Tolerant and Test Support Signals

Some signals have been added for fault tolerant and test MECHANISM improvement. Those new signals can be classified as follow:

4.1.1. Parity Checking

Address Parity Checking:

APAR - Address Bus Parity (input)

Data Parity Checking:

DPAR - Data Bus Parity (bidirectional)

FPU control signal Parity Checking:

IFPAR - IU to FPU Control Parity (input)

FIPAR - FPU to IU Control Parity (output)

Parity Checking Error Output

$\overline{\text{HWERROR}}$ - Error State (output)

Note that all parity bits are defined as odd parity over the concerned busses. The odd parity definition is:

the number of ones in a word, including the parity bit, is always odd (eg 00000000 --> P = 1, 00000001 --> P = 0)

4.1.2. Master/Checker Mode

$\overline{\text{CMODE}}$ - Checker Mode (input)

$\overline{\text{MCERR}}$ - Comparison Error (output)

4.1.3. Test Access Port

TCLK - Test Clock (input)

$\overline{\text{TRST}}$ - Test $\overline{\text{RESET}}$ (input)

TMS - Test Mode Select (input)

TDI - Test Data Input (input)

TDO - Test Data Output (Output)

4.1.4. Miscellaneous

$\overline{\text{602MODE}}$ - Normal $\overline{\text{602MODE}}$ Operation (input)

$\overline{\text{HALT}}$ - $\overline{\text{HALT}}$ (input)

4.2.Parity Checking

4.2.1. Introduction

In the TSC692E, 98% of the FPU registers are parity bit protected. Address and data busses, control signals to and from IU are also parity protected. Checking of registers and busses is performed only if the register or the bus is used by the current instruction. With this approach, unused registers/busses will not cause an error and downtime of the system will be limited.

The parity checking is disabled during and after $\overline{\text{RESET}}$ until the latches used are set. During Initialization sequence, all internal registers (7 registers, FSR) must be written in order to initialize the clock bits. All internal registers (except f registers) and all writable fields of FSR are set to zero when asserting $\overline{\text{RESET}}$.

4.2.2. Error handling scheme in TSC692E

Since the FPU only performs calculations, the solution for handling all errors detected by the internal concurrent error detection in the FPU is to handle them as exceptions and enter the cause in the FTT field in the FSR. The FTT field is three bit and is coded to get eight trap types. The solution is to define the trap types used for internal error to be trap types number 5 to 7.

The FPU can signal parity errors externally by using signal $\overline{\text{HWERROR}}$. The FPU will enter the exception pending mode and $\overline{\text{FEXC}}$ will be asserted in the same cycle as $\overline{\text{HWERROR}}$. The address and the failing FP operate instruction are stored in the queue. Analysis of error type is possible by software assistance by reading FSR. Three error types are defined:

Data Bus Error (FTT field of FSR equal to 5)

This type of error concerns parity errors on the data bus.

Restartable Error (FTT field of FSR equal to 6)

This type of error concerns parity errors in the FPU that were detected before changing the FPU state and could be removed by restarting the instruction (IU to FPU control bus, ...).

Non-Restartable Error (FTT field of FSR equal to 7)

This type of error concerns parity errors that were detected after the state of the FPU was changed and could not be removed by restarting the instruction (FSR, Register File,...).

$\overline{\text{HWERROR}}$ will be asserted low in case of any of above errors, and stay asserted until the next FPop encountered in the instruction stream (after a STDFQ instruction) modifies the FTT field of FSR.

When multiple hardware traps occur at the same cycle, the highest priority trap is taken, and lower priority traps are ignored. The priority applied on the hardware traps of the FPU are defined as follow:

Table 11. priority within traps

FSR.tt	Type of trap	priority
1,2,3,4	IEEE,unfinished,Unimp,Seq.Err traps	4
5	Data Bus Error trap	2
6	Restartable Error trap	3
7	Non restartable trap	1

Note: Priority 1 is for highest priority.

4.2.3. Parity Checking on Control Pads for the FPU

The control signals between the IU and the FPU are protected by a parity bit.

4.2.3.1. Input control signals

There is a five bit input control bus: FINS1, FINS2, FLUSH, FXACK and INST.

The parity input pad for these five signals is IFPAR (IU to FPU PARity).

This parity bit is generated by the IU.

4.2.3.1.1. Output control signals

There is a five bit output control bus: FCC<1:0>, FCCV, $\overline{\text{FEXC}}$ and $\overline{\text{FHOLD}}$.

The parity output pad for these signals is FIPAR (FPU to IU PARity).

This parity bit is generated by the FPU and checked by the IU.

FIPAR is three-state output controlled by $\overline{\text{TOE}}$ signal.

4.2.4. Parity Checking on address bus

The 32 bit address bus contains a parity bit calculated by the IU and sent out on the APAR pad. The parity bit is checked by the FPU for all FPop instructions and it will generate a Non Restartable trap only in the case of a STDFQ instruction.

4.2.5. Parity Checking on data bus

The DPAR bidirectional signal contains the odd parity over the 32-bit data bus.

When the FPU receives a data (LOAD) or an instruction, the parity bit is checked by the FPU.

In case of a STORE data instruction, the parity bit is generated and launched in parallel by the FPU.

DPAR is three-state output controlled by $\overline{\text{DOE}}$ signal.

4.2.6. Internal Parity Checking

All internal registers are parity protected. The FPU includes parity generation and checking on all internal datapaths (see Figure 20. , page 45)(see Figure 21. , page 46).

4.2.7.Non RT 602 Mode

To be able to use normal IU (i.e. TSC691E), parity on the data bus has to be generated internally and parity checking on the control bus must be turned off. Nevertheless, internal parity check remains active.

This feature is controlled by asserting the $\overline{\text{602MODE}}$ input signal. $\overline{\text{602MODE}}$ is a static signal and will not change when running.

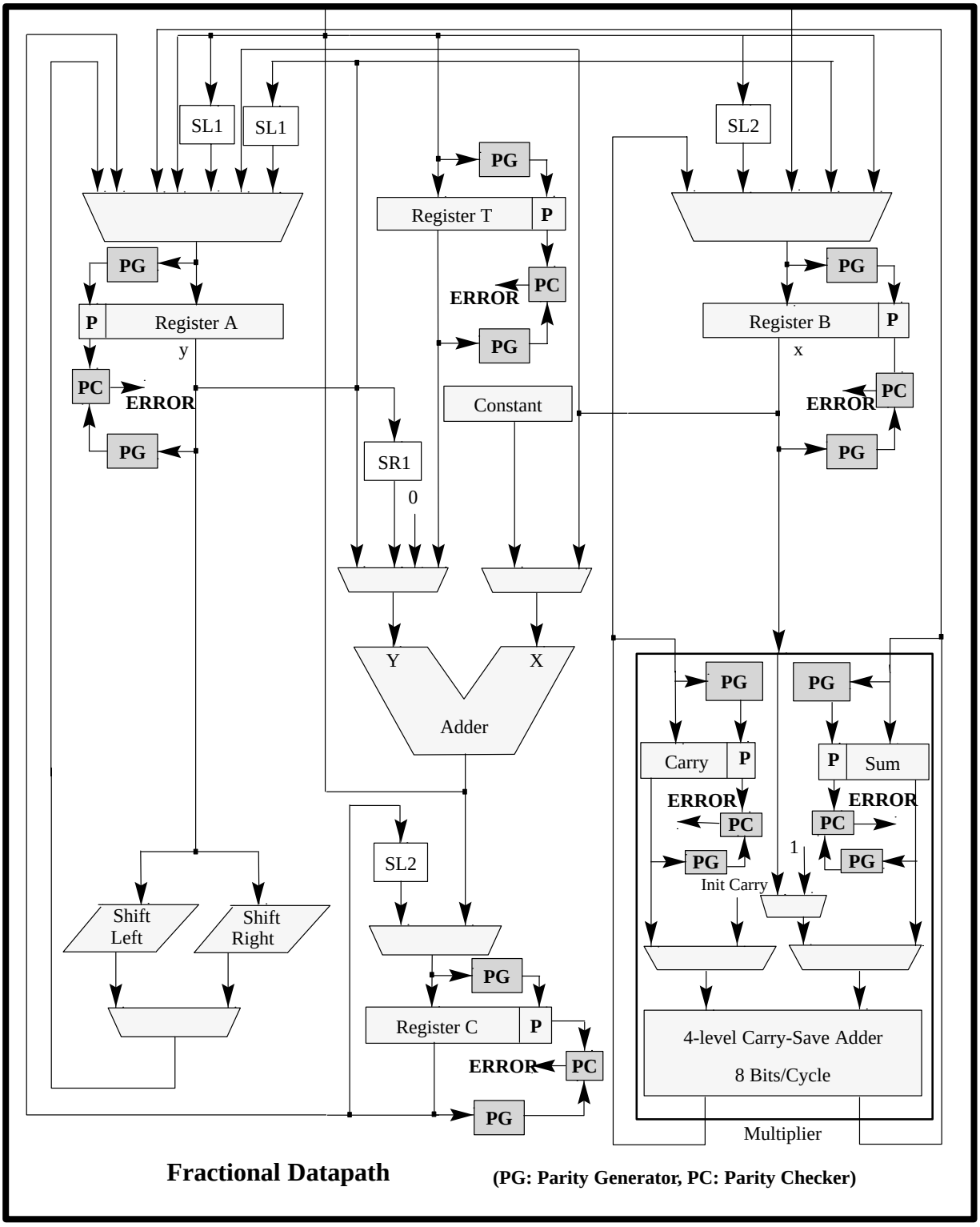


Figure 19. Parity Checking on Fractional Datapath

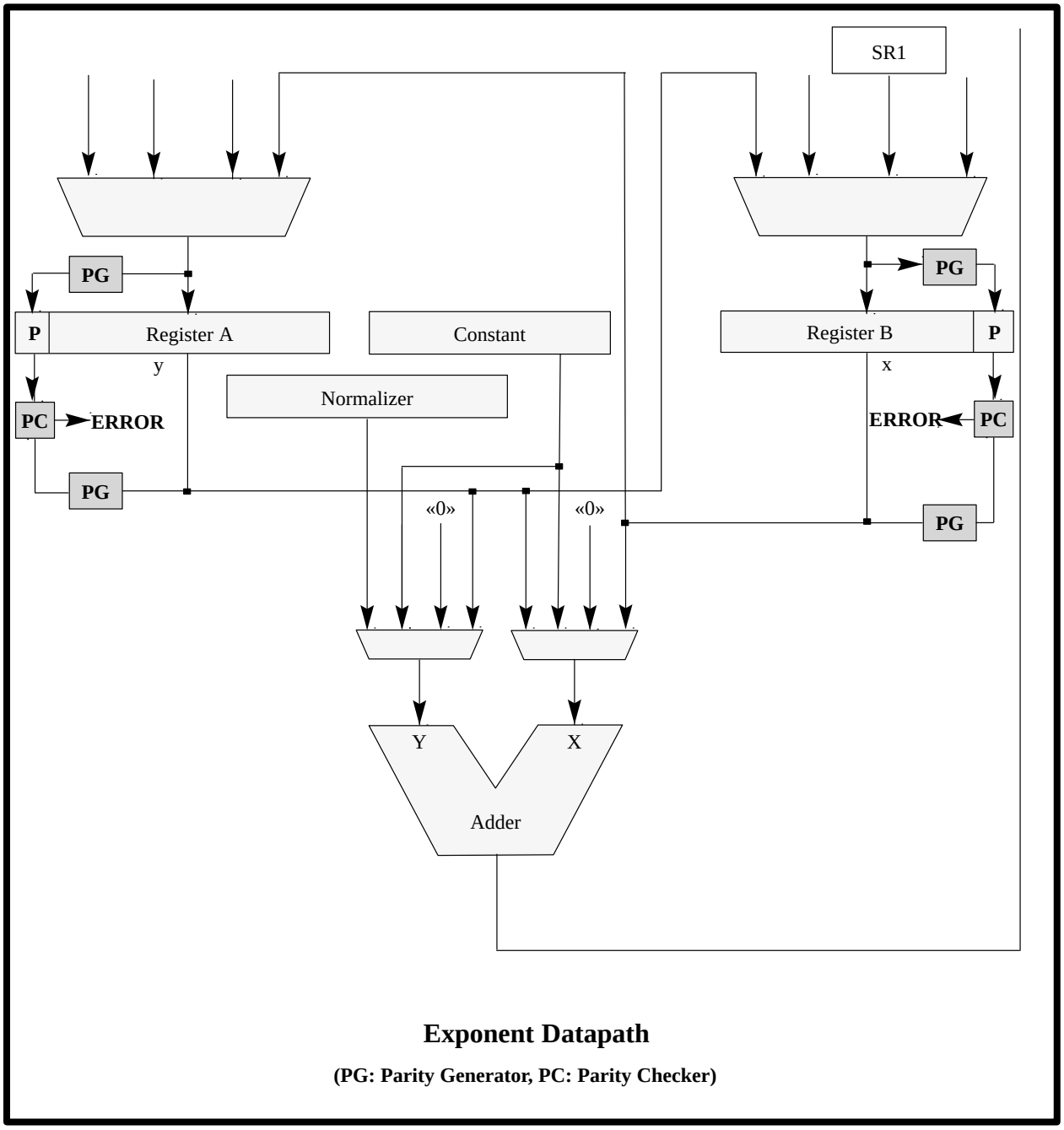


Figure 20. Parity Checking on Exponent Datapath

4.3. Master/checker Operation

The **TSC692E** includes comparator circuits at the outputs to support fault detection. Applications requiring a high level of reliability can use this Master/Checker operation to introduce fault behavior on system level. By duplication of units and without the use of external comparators 100% of the internal errors are detected, especially those errors that are not detected by the internal concurrent error detection MECHANISM.

4.3.1. Basic function

By programming of the $\overline{\text{CMODE}}$ signal, the **TSC692E** can be configured either as master or checker. The master and at least one checker circuit are working in parallel and execute the same program. While the master is forcing the data bus, the checker is in a read and compare mode. This means the output buffers are disabled and the external busses are compared by the checker with its internal results. If a mismatch occurs on any output, then the $\overline{\text{MCERR}}$ signal is asserted. In this case, the system hardware and/or software can take appropriate action.

If the master FPU signals an internal error before a comparison error is indicated, it is possible to stop execution of the two FPUs by asserting the $\overline{\text{HALT}}$ signal, disable the master FPU, change the slave FPU to master FPU and continue execution. $\overline{\text{CMODE}}$ signal can be changed when $\overline{\text{RESET}}$ signal is asserted or when the FPU is in $\overline{\text{HALT}}$ mode.

An external/internal mismatch can occur for two reasons.

First, a short or other electrical failure can force the output signal to a fixed voltage. For example, a bus signal can be shorted to ground. When the circuit drives a high voltage on the bus, the external signal will be pulled low and a mismatch will occur.

The second way is that an external/internal mismatch can occur in the master/checker mode. Figure 22 shows a basic master/checker configuration using two **TSC692E** devices.

Using the master/checker solution there is a possibility that the system can continue with the correct remaining unit or with both after restoration of state of the faulty unit. If an internal error is indicated in the checker, it could be ignored. The TSC693E requires error signals from both the master and the checker. In case of corruption the system behavior is defined by the TSC693E.

The FPU shall also use the Master/Checker function in single mode for detection of stuck-at-one and stuck-at-zero faults on the checked buffers (asserting then $\overline{\text{MCERR}}$ for Master FPU).

On a master processor, the three-state control signals ($\overline{\text{DOE}}$ and $\overline{\text{TOE}}$) disable the checker mode of the three-stated buffers.

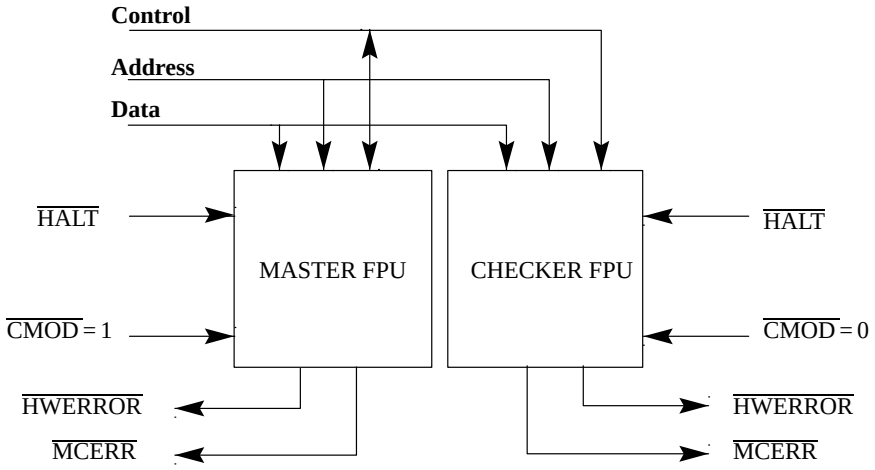


Figure 21. Master/Checker configuration

4.3.2. Master/Checker signals

$\overline{\text{CMODE}}$ *input* **(Master/checker operation):**

Assertion of this signal sets the FPU to act as a checker only in a master/checker configuration. All output signal except $\overline{\text{HWERROR}}$, $\overline{\text{MCERR}}$ and TAP signals will be high «Z». $\overline{\text{CMODE}}$ is a static signal and will not change when running.The $\overline{\text{CMODE}}$ signal may only be changed when the $\overline{\text{RESET}}$ and/or the $\overline{\text{HALT}}$ signal is asserted.

$\overline{\text{MCERR}}$ *output* **Comparison Error:**

This signal is asserted in checker mode when a comparison error occurs on the internal output signals (except $\overline{\text{HWERROR}}$ and TAP signals) vis-a-vis the output signals of the master FPU. It is deasserted when the error disappears.

4.4. IEEE Standard Test Access Port & Boundary-Scan Architecture

The FPU includes a Test Access Port (TAP) interface (**IEEE standard 1149.1**). This interface is used for debugging and test purposes.

This interface provides standardized approaches to:

testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board or other substrate.

support of testing the integrated circuit itself.

observing or modifying activity during the component's normal operation.

4.4.1. TAP signals

The Test Access Port includes the following five connections: TCLK, TMS, $\overline{\text{TRST}}$, TDI and TDO. Dedicated TAP connections are required to allow access to the full range of mandatory features of this standard.

TCLK (*input*)

The Test Clock Input provides the clock for the test logic defined by this standard.The IEEE standards requires that TCLK can be stopped at 0 indefinitely without causing any change to the state of the test logic.

TMS (*input*)

The signal received by TMS is decoded by the TAP controller to control test operation. TMS is sampled on the rising edge of TCLK and has to change on the falling edge of TCLK

TDI (*input*)

Serial test instructions and data are received by the test logic by TDI. TDI is sampled on the rising edge of TCLK and has to change on the falling edge of TCLK.

$\overline{\text{TRST}}$ (*input*)

The $\overline{\text{TRST}}$ input provides for asynchronous initialization of the TAP controller.

TDO (*output*)

TDO is the serial output for test instructions and data from the test logic defined in the standard.

4.4.2. TAP Controller

The TAP controller is a synchronous finite state machine that responds to changes at the TMS and TCLK signal of the TAP and controls the sequence of operations of the circuit defined by the IEEE standard.

4.4.3. The Instruction Register

The Instruction Register allows an instruction to be shifted into the design. The instruction is used to select the test to be performed or the test data register to be accessed or both. A number of mandatory and optional instructions are defined by the standard. The instructions SAMPLE/PRELOAD, BYPASS, INTEST and EXTEST are implemented on this chip.

The private instruction TESTPAR will be implemented to access the internal scan path registers. These registers are not publicly accessible and will be used to test the internal parity logic.

IR encoding is shown in Table 12.

Table 12. Instruction Register Encoding

IR value	Instruction	Registers
000001	SAMPLE/PRELOAD	Boundary Scan Registers
111111	BYPASS	Bypass Register
000011	INTEST	Boundary Scan Registers
000000	EXTEST	Boundary Scan Registers
100000	IDCODE	Identification Register
proprietary	TESTPAR	Internal Scan Registers

4.4.3.1. Design and Construction of the instruction register

The instruction register is a shift-based design having an optional parallel input. These parallel inputs permit capture of design-specific information in the Capture-IR state. Figure 23 illustrates an example implementation of an Instruction Register Cell.

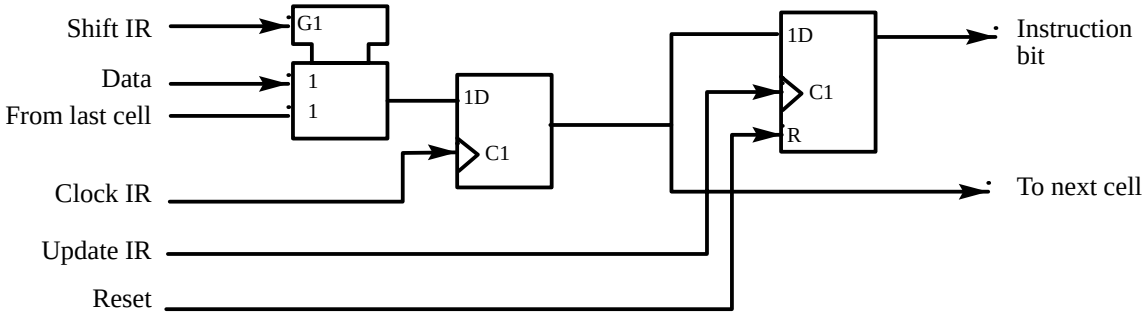


Figure 22. Instruction Register Cell

4.4.3.2. BYPASS Instruction

The BYPASS register contains a single shift register stage, used to speed-up shifting at the board level, through components which are not activated.

4.4.3.3. EXTEST Instruction

EXTEST instruction shall connect the BOUNDARY SCAN register between TDI and TDO. It is used to test connections between components on the board level. All output signals can be disabled by using the EXTEST instruction (except TAP).

4.4.3.4. INTEST Instruction

INTEST instruction allows testing of the on-chip system logic while the component is assembled on the board, with each test pattern and response being shifted through the boundary-scan register.

4.4.3.5. SAMPLE/PRELOAD Instruction

SAMPLE instruction allows normal operation of the system logic with the ability to sample signals entering and leaving the component without affecting circuit operation.

PRELOAD allows a value to be preloaded on the latched outputs of the boundary scan register. This instruction does not modify the system behavior.

4.4.4. The Device Identification Register

The Device Identification Register is implemented on this chip. It contains the TSC692E's assigned component identifier, 0x0B6410B1. It is selected by the IDCODE instruction.

4.4.5. Internal Scan Path

An Internal Scan Path is implemented to provide the off-line test of the internal parity error detection. This Internal Scan Path is controlled by the TAP and forces some nodes in the generation circuit of the parity bits. This will then result in a value with the wrong parity. When this value is read again, an error will be detected if the error detection works correctly. This chain will have one bit for each parity generator.

4.5. Boundary scan test register

The Boundary-scan technique involves the inclusion of a shift register stage (contained in a Boundary-scan cell) adjacent to each component pin so that signals at component boundaries can be controlled and observed using scan testing principles.

Figure 24. illustrates an example implementation for a Boundary-scan cell that could be used for an input or output connection to an integrated circuit. Dependent on the control signals applied to the multiplexers, data can either be loaded into the scan register from the Signal-in port (e.g., the input pin), or driven from the register through the Signal-out port of the cell (e.g., into the core of the component design). The second flip-flop (controlled by clock B) is provided to ensure that the signals driven out of the cell in the latter case are held while new data is shifted into the cell using clock A.

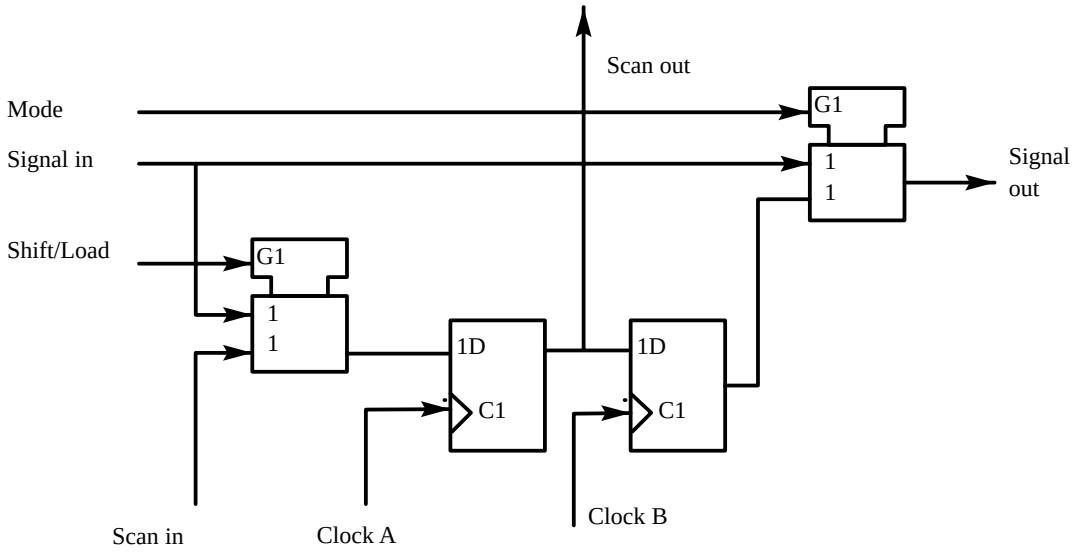


Figure 23. Boundary Scan Cell

4.6. Parity on odd and even bits of the register file bits

It is known that the impact from an SEU may flip adjacent bits in a register file. Those multiple bit errors might be impossible to detect with one parity bit error. Though these cases with multiple bit errors due to SEU are probably more rare than one bit errors, they cannot be neglected, especially not in the register file, which corresponds to about 50% of the entire amount of registers in the FPU.

One solution to this problem is to generate two parity bits for a 32-bits word, one for even bits and one for odd bits. This is done in the register file and will remove all multiple bit errors due to SEU.

5. Electrical and Mechanical Specifications.

5.1. TSC692E Maximum Ratings and DC Characteristics

5.1.1. TSC692E Maximum Rating

Storage Temperature . . . . . -65 ° C to +150 ° C

Ambient Temperature with Power Applied . . . . . -55 ° C to +125 ° C

Supply Voltage[1] . . . . . -0.5 V to +7.0 V

Input Voltage . . . . . -0.5 V to +7.0 V

5.1.2. TSC692E Operating Range

Table 13. TSC692E Operating Range

Range	Ambient Temperature[a]	Vcc
Military	-55° C to +125° C	5V +/- 10%

[a]. Ambient temperature is defined as the ‘instant on’ case temperature.

5.1.3. TSC692E DC Characteristics Over the Operating Range

Table 14. TSC692E DC Characteristics over the operating range

Parameters	Description	Test Conditions	Min.	Max.	Units
V_{OH}	Output HIGH Voltage	V_{CC} = Min., I_{OH} = -2.0 mA	2.4		V
V_{OL}	Output LOW Voltage	V_{CC} = Min., I_{OL} = 4.0 mA		0.5	V
V_{IH}	Input HIGH Voltage		2.1	V_{CC}	V
V_{IL}	Input LOW Voltage		-0.5	0.8	V
I_{IZ}	Input Leakage Current	V_{CC} = Max., $V_{SS} \le V_{IN} \le V_{CC}$	-10	10	µA
I_{OZ}	Output Leakage Current	V_{CC} = Max., $V_{SS} \le V_{out} \le V_{CC}$	-15	15	µA
I_{SC}	Output Short Circuit Current	V_{CC} = Max., V_{out} = 0V	-30	-350	mA
I_{CCOP}	TSC692E Supply Current	V_{CC} = Max, f = 14 MHz		180	mA
I_{CCSB}	Standby Current	Vcc = Max, f=0 MHz		3	mA

Table 15. TSC692E Capacitance Ratings [1]

Parameters	Description	Max. (pF)
C_{IN}	Input Capacitance	10
C_{OUT}	Output Capacitance	12
C_{IO}	Input/Output Bus Capacitance	15

[1]. Tested initially and after any design or process changes that may affect these parameters.
Test conditions are: V_{CC} = 5.0 V, T_A = 25, C.f = 1 MHz

5.1.4. TSC692E AC Test Loads and Waveforms

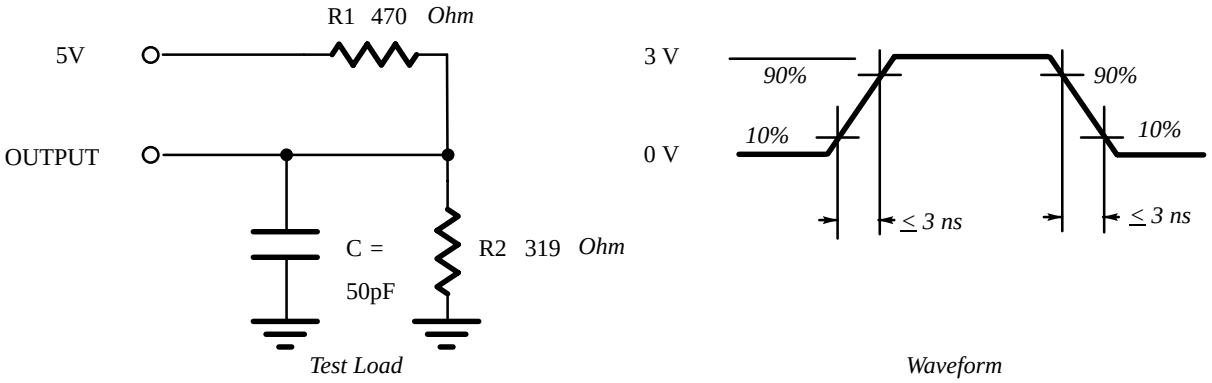


Figure 24. TSC692E AC Test Loads and Waveforms

5.2. TSC692E AC Characteristics

Table 16. TSC692E Characteristics at 14/25 MHz

Parameter		Description	Ref. Edge	spec. 14 MHz	
				Min	Max
1	t_{CY}	Clock Cycle[a]		71	
2	t_{CHL}	Clock High and Low		33	
3	t_{AS}	A[31:0] Setup	CLK+	7	
4	t_{AH}	A[31:0] Hold	CLK+	6	
5	t_{DIS}	D[31:0] Input Setup	CLK+	7	
6	t_{DIH}	D[31:0] Input Hold	CLK+	6	
7	t_{DOD}	D[31:0] Output Delay	CLK-		35
8	t_{DOH}	D[31:0] Data Valid	CLK-	4	
9	t_{DOFFL}	D[31:0] Output Turn–off	FLUSH+		56
10	t_{DOHFL}	D[31:0] Output Valid	FLUSH+	0	
11	t_{DOFOE}	D[31:0] Output Turn–off[b]	$\overline{DOE}$+		27
12	t_{DONOE}	D[31:0] Output Turn–on	$\overline{DOE}$-		27
13	t_{DOHOE}	D[31:0] Output Valid	$\overline{DOE}$-	0	
14	t_{FIS}	FINS1/2 Setup	CLK+	40	
15	t_{FIH}	FINS1/2 Hold	CLK+	2.5	
16	t_{INS}	INST Setup	CLK+	29	
17	t_{INH}	INST Hold	CLK+	2	
18	t_{FXS}	FXACK Setup	CLK+	29	
19	t_{FXH}	FXACK Hold	CLK+	2	

Parameter		Description	Ref. Edge	spec. 14 MHz	
				Min	Max
20	t_{FLS}	FLUSH Setup	CLK+	38	
21	t_{FLH}	FLUSH Hold	CLK+	2	
22	t_{RES}	$\overline{\text{RESET}}$ Setup	CLK+	27	
23	t_{REH}	$\overline{\text{RESET}}$ Hold	CLK+	3	
24	t_{Atmel}	$\overline{\text{MHOLDA}}$ Setup [c]	CLK-	4	
25	t_{MHH}	$\overline{\text{MHOLDA}}$ Hold	CLK-	9	
26	t_{MDS}	$\overline{\text{MDS}}$ Setup	CLK-	4	
27	t_{MDH}	$\overline{\text{MDS}}$ Hold	CLK-	9	
28	t_{FHD}	$\overline{\text{FHOLD}}$ Delay	CLK-		40
29	t_{FHH}	$\overline{\text{FHOLD}}$ Valid	CLK-	5	
30	t_{FHDFI}	$\overline{\text{FHOLD}}$ Delay	FINS1/2+		29
31	t_{FHDFL}	$\overline{\text{FHOLD}}$ Delay	FLUSH+		50
32	t_{FHDMH}	$\overline{\text{FHOLD}}$ Delay	$\overline{\text{MHOLD}}$-		65
33	t_{FCCVD}	FCCV Delay	CLK-		40
34	t_{FCCVH}	FCCV Valid	CLK-	5	
35	$t_{FCCVDFL}$	FCCV Delay	FLUSH+		50
36	$t_{FCCVDMH}$	FCCV Delay	$\overline{\text{MHOLD}}$-		65
37	t_{FCCD}	FCC[1:0] Delay	CLK+		47
38	t_{FCCH}	FCC[1:0] Valid	CLK+	5	
39	t_{FED}	$\overline{\text{FEXC}}$ Delay	CLK+		47
40	t_{FEH}	$\overline{\text{FEXC}}$ Valid	CLK+	5	
41	t_{FND}	FNULL Delay	CLK+		36
42	t_{FNH}	FNULL Valid	CLK+	3	
43	t_{TCY}	TCLK Clock Cycle		100	1000
44	t_{TMS}	TMS Setup	TCLK+	20	
45	t_{TMH}	TMS Hold	TCLK+	25	
46	t_{TDIS}	TDI Setup	TCLK+	20	
47	t_{TDIH}	TDI Hold	TCLK+	25	
48	t_{TRS}	$\overline{\text{TRST}}$ Setup	TCLK+	20	
49	t_{TRH}	$\overline{\text{TRST}}$ Hold	TCLK+	25	
50	t_{TDOD}	TDO Delay	TCLK-		45
51	t_{TDOH}	TDO Valid	TCLK-	5	
52	t_{APS}	APAR Setup	CLK+	6	

Parameter		Description	Ref. Edge	spec. 14 MHz	
				Min	Max
53	t_{APH}	APAR Hold	CLK+	6	
54	t_{DPIS}	DPAR Input Setup	CLK+	6	
55	t_{DPIH}	DPAR Input Hold	CLK+	4	
56	t_{DPOD}	DPAR Output Delay	CLK-		45
57	t_{DPOH}	DPAR Output Valid	CLK-	4	
58	t_{IFS}	IFPAR Setup	CLK+	16	
59	t_{IFH}	IFPAR Hold	CLK+	3	
60	t_{FIPD}	FIPAR Delay[d]	CLK+		50
61	t_{FIPH}	FIPAR Valid	CLK+	5	
62	t_{MCD}	$\overline{\text{MCERR}}$ Delay	CLK+		45
63	t_{MCH}	$\overline{\text{MCERR}}$ Valid	CLK+	5	
64	t_{602S},t_{CMS}	$\overline{\text{602MODE}}$/$\overline{\text{CMODE}}$ Setup[e]	CLK+	18	
65	t_{HAS}	$\overline{\text{HALT}}$ Setup	CLK-	13	
66	t_{HAH}	$\overline{\text{HALT}}$ Hold	CLK-	4	
67	t_{ERD}	$\overline{\text{HWERROR}}$ Delay	CLK+		45
68	t_{ERH}	$\overline{\text{HWERROR}}$ Valid	CLK+	5	

[a]. Parameter t_{crf} (Clock rise and fall) is set to 0.8 V/ns (min).
[b]. Idem for $\overline{\text{TOE}}$ and output signals: FCCV, FCC[1:0], $\overline{\text{FEXC}}$, $\overline{\text{FHOLD}}$, FNULL, FIPAR and $\overline{\text{FP}}$ (param. 11, 12 and 13)
[c]. This specification applies also to $\overline{\text{MHOLDB}}$, $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$ and CCCV signals
[d]. FIPAR evaluated with $\overline{\text{FHOLD}}$ and FCCV sampled on CLK- and latched on CLK+.Needs same logic in IU to calculate correct PARITY (param. 60 and 61)
[e]. $\overline{\text{602MODE}}$/$\overline{\text{CMODE}}$ shall be changed to be related to positive clock edge during $\overline{\text{RESET}}$ active or $\overline{\text{HALT}}$ active

5.2.1. TSC692E AC Waveforms

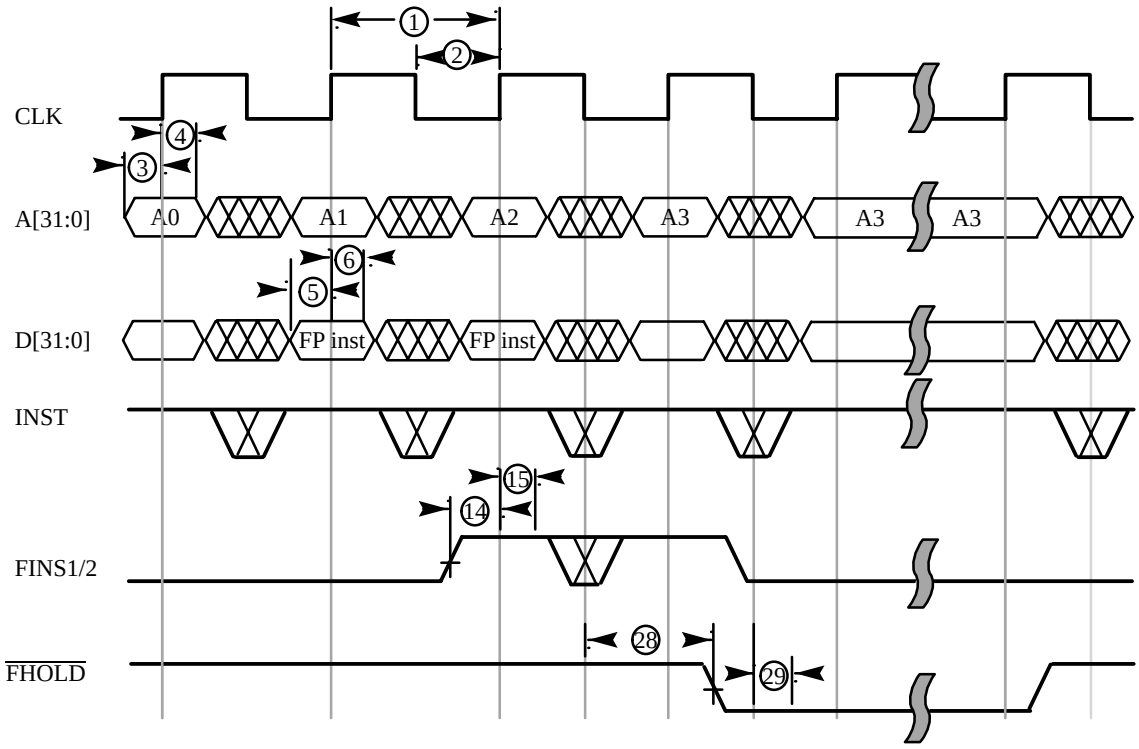


Figure 25. Floating–Point $\overline{\text{FHOLD}}$ Assertion

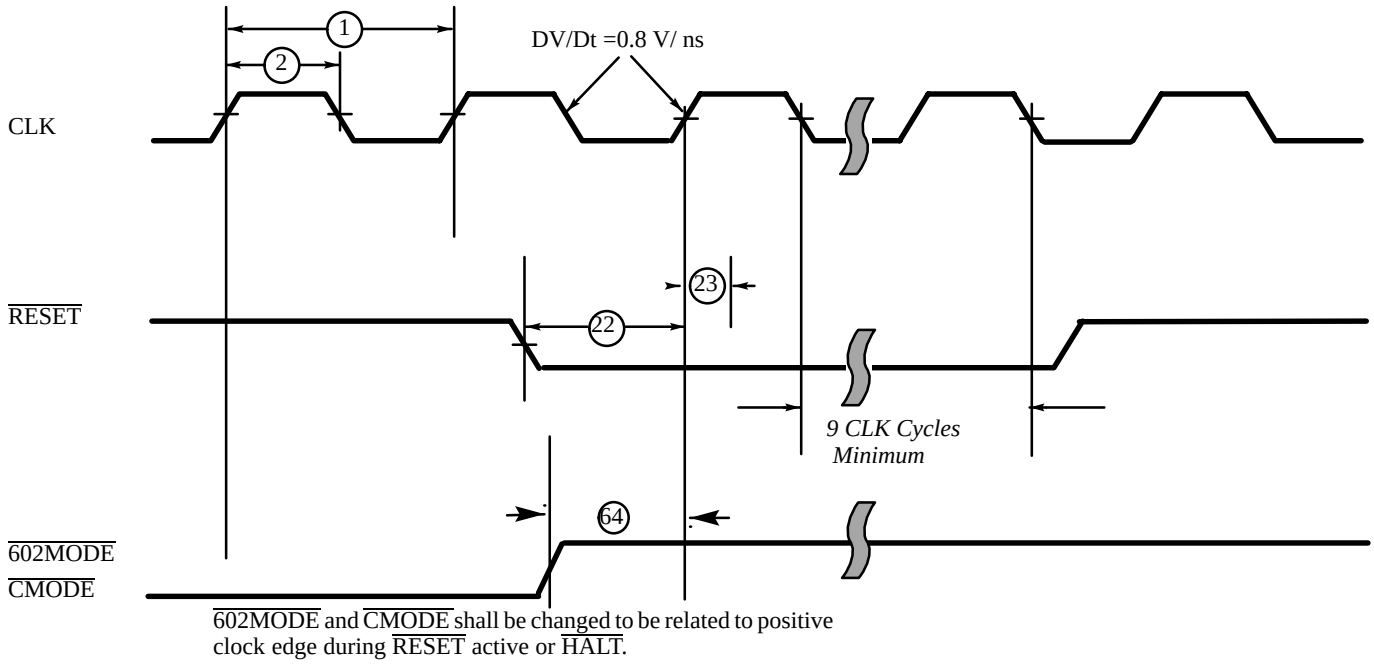


Figure 26. Clock and $\overline{\text{RESET}}$ Timing

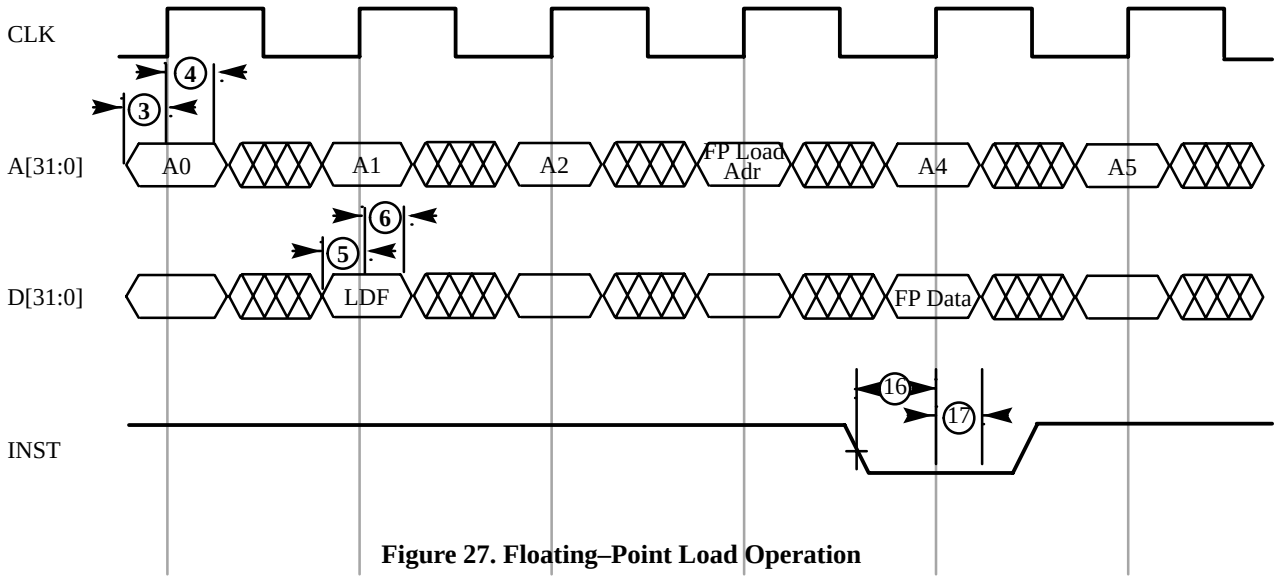


Figure 27. Floating–Point Load Operation

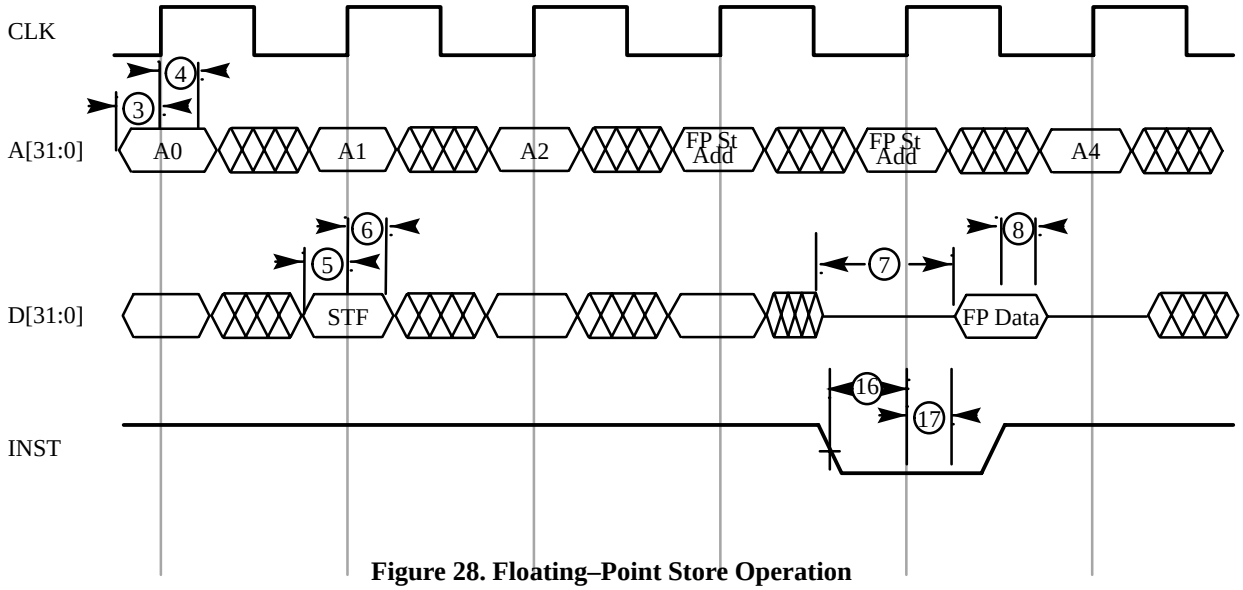


Figure 28. Floating–Point Store Operation

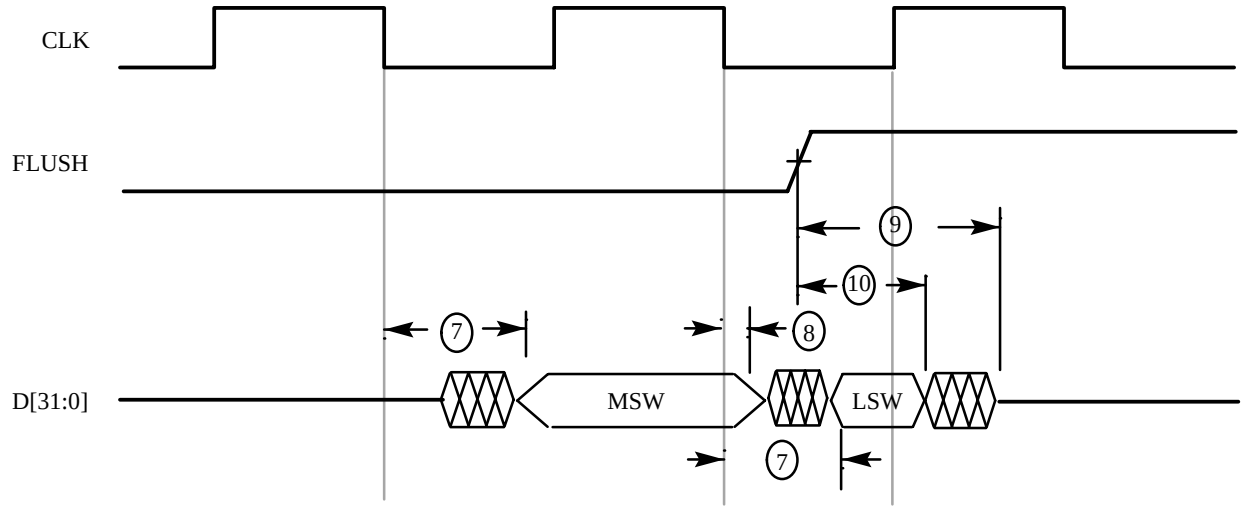


Figure 29. Effect of FLUSH on Store Timing

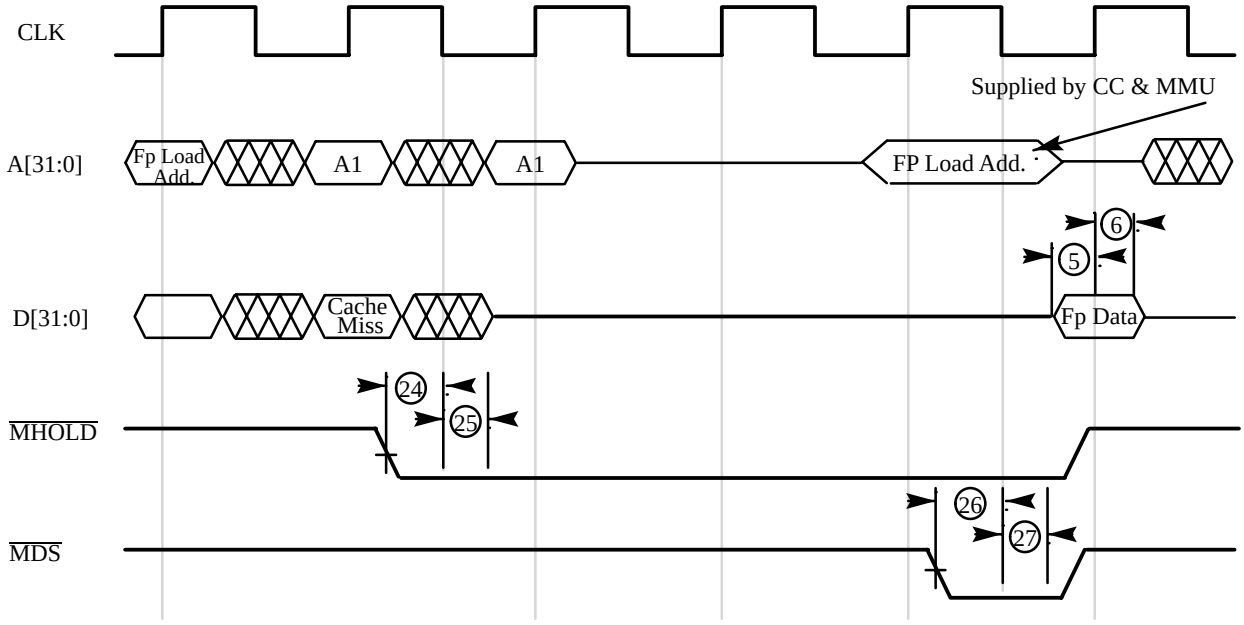


Figure 30. Floating–Point Load Cache Miss

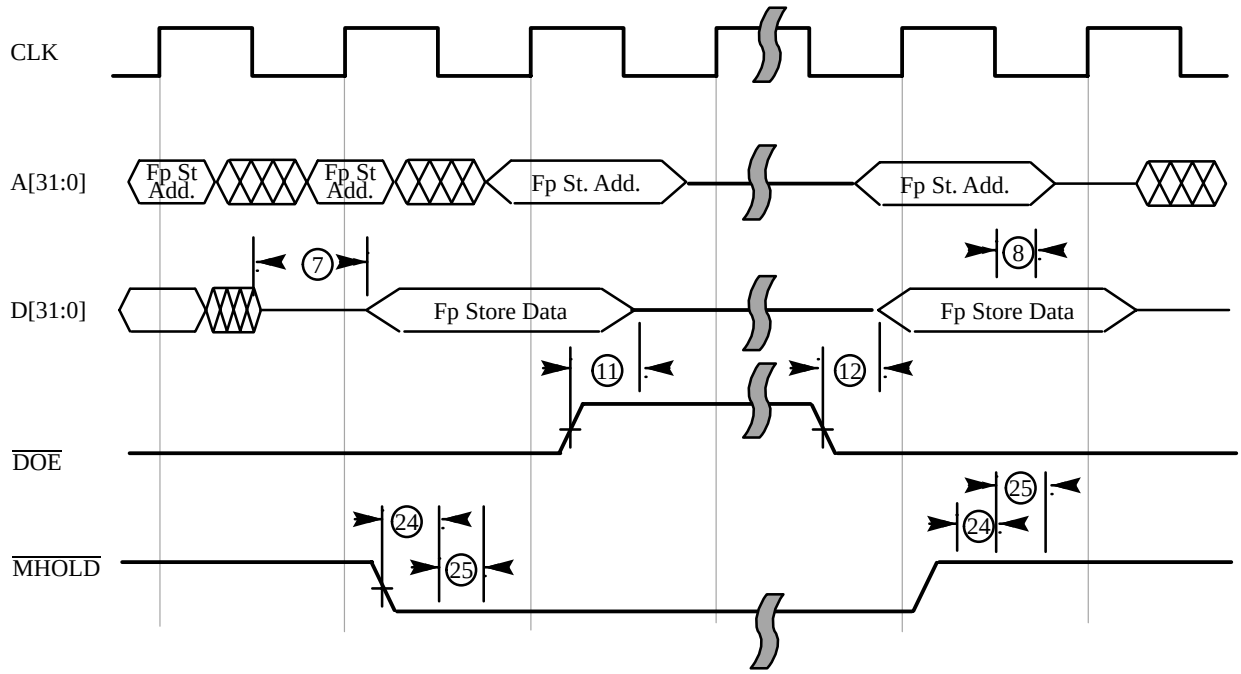


Figure 31. Floating–Point Store Cache Miss

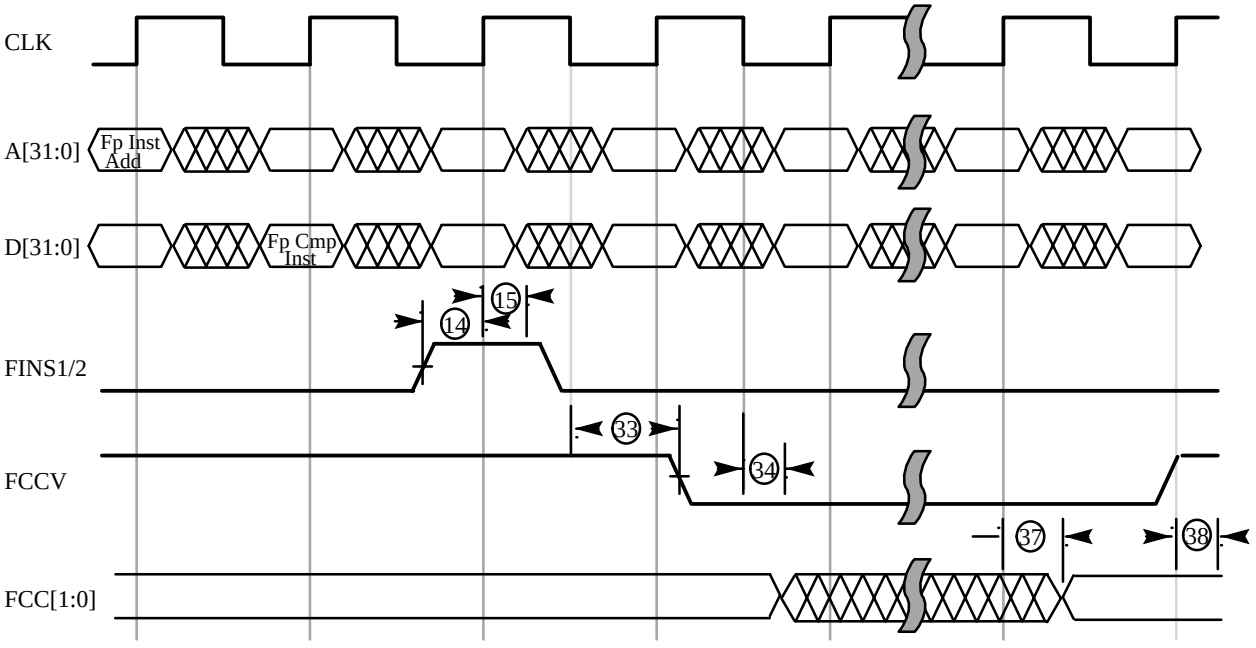


Figure 32. Floating–Point Compare

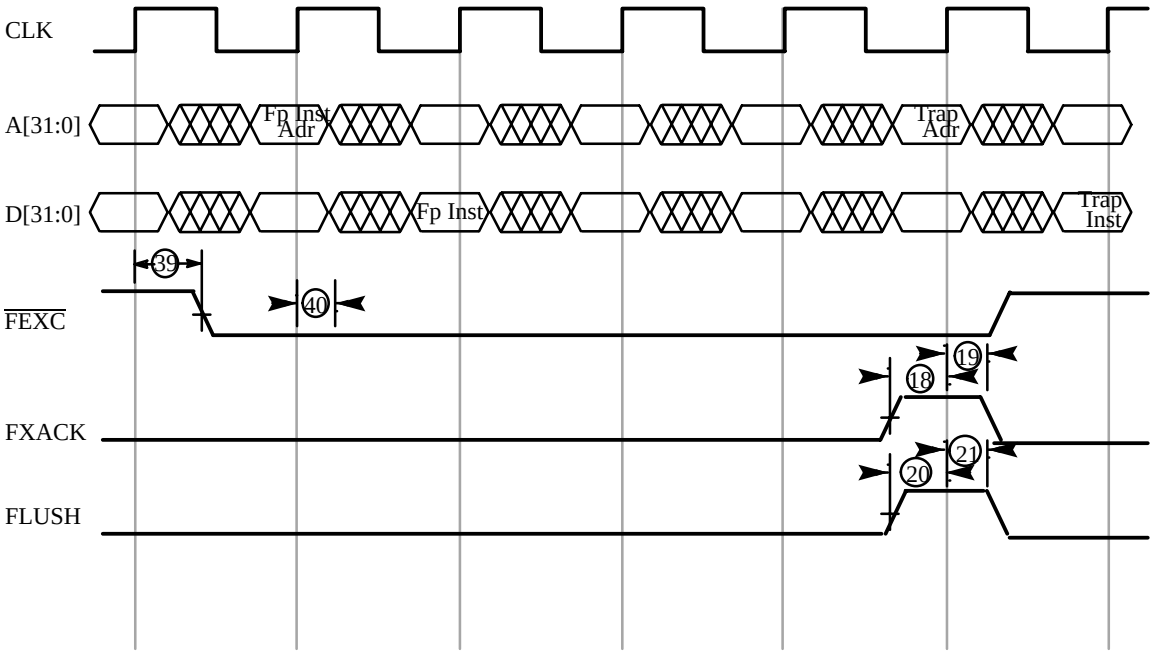


Figure 33. Floating–Point Trap

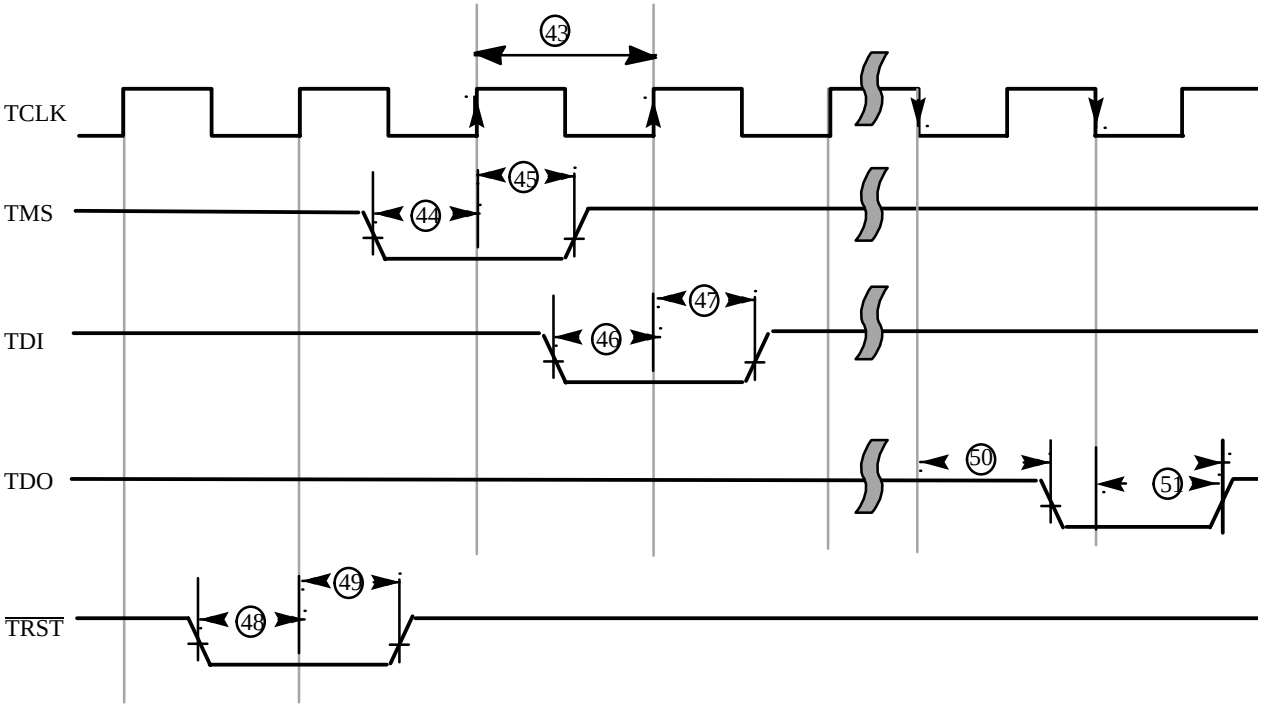


Figure 34. TAP Signals

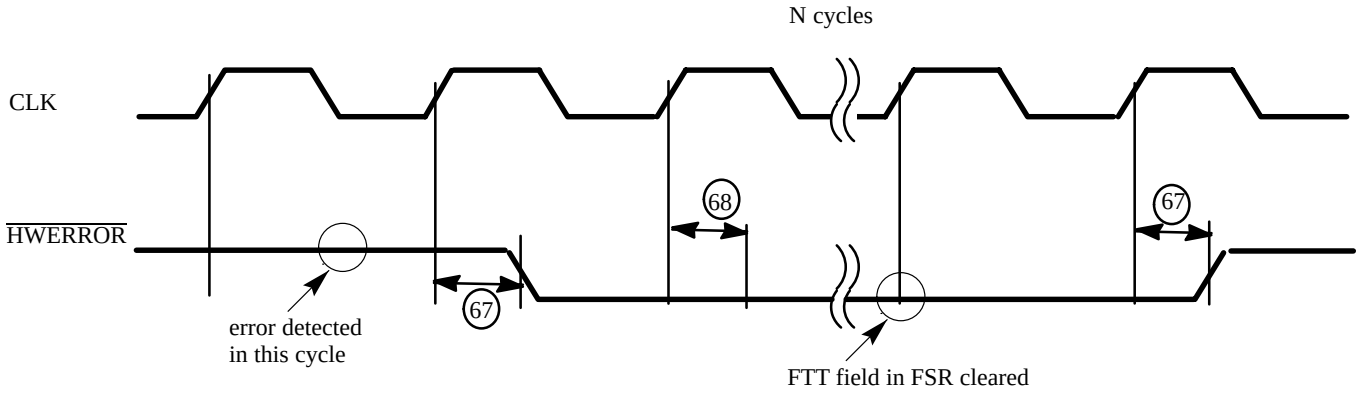


Figure 35. $\overline{\text{HWERROR}}$ Timing

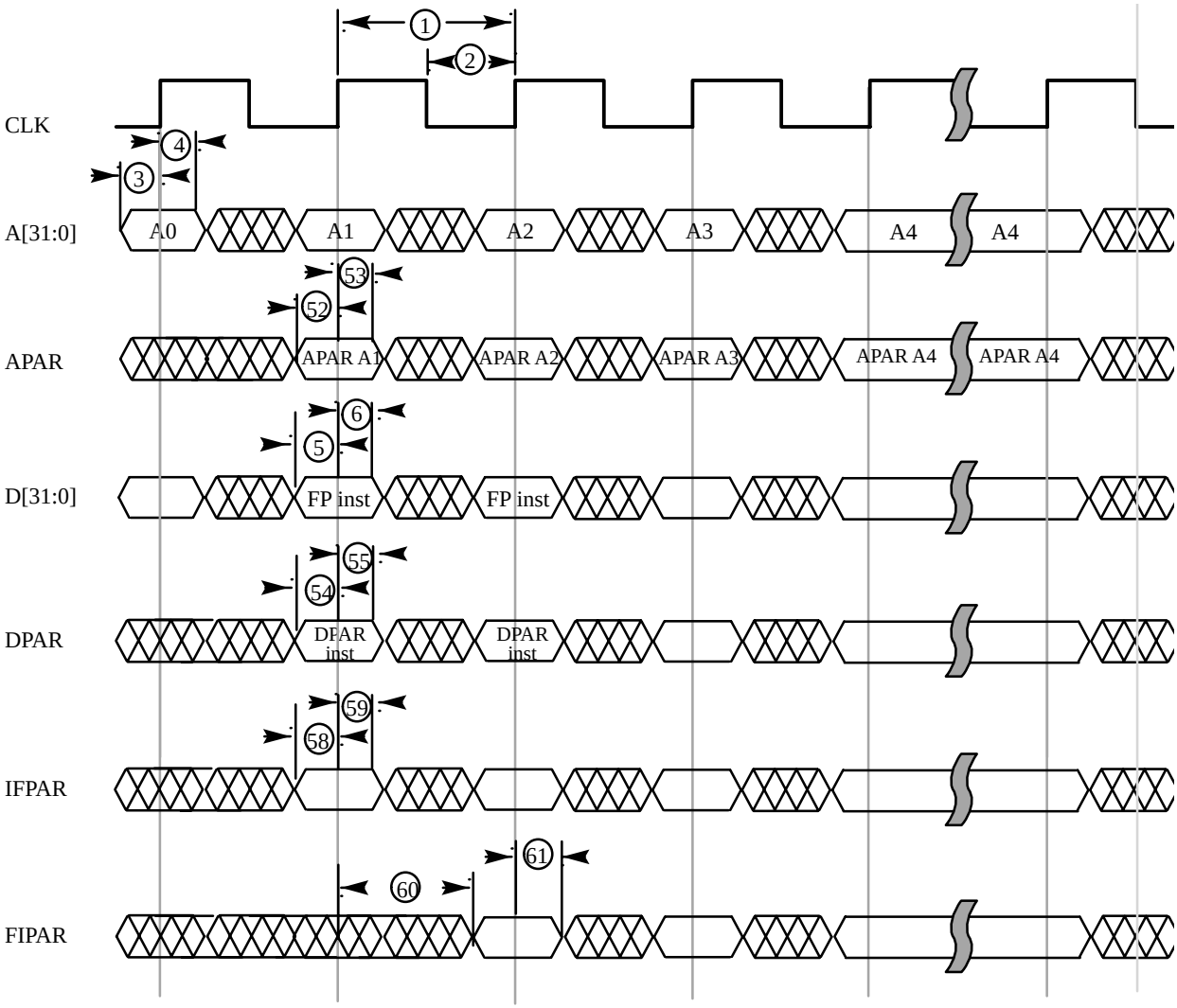


Figure 36. PARITY Signals

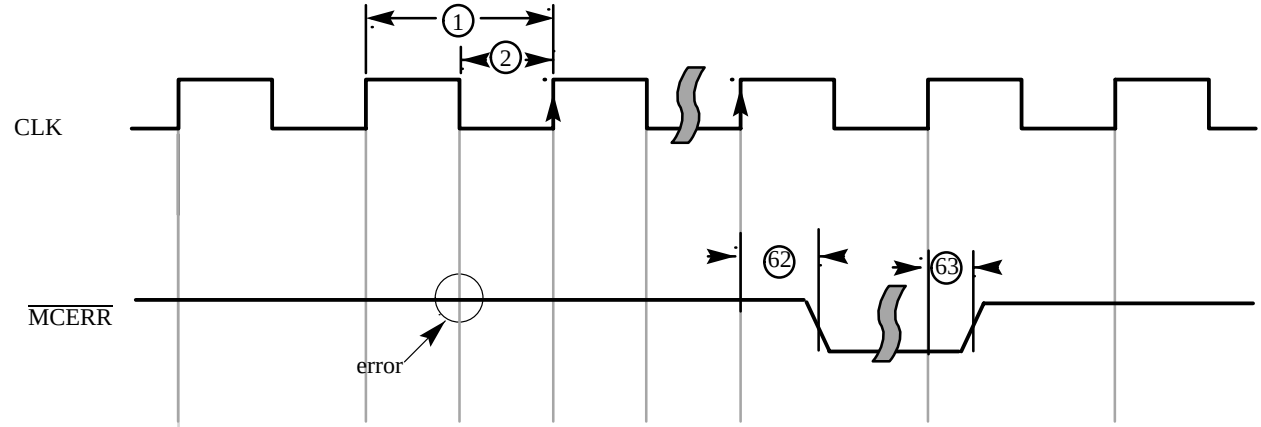
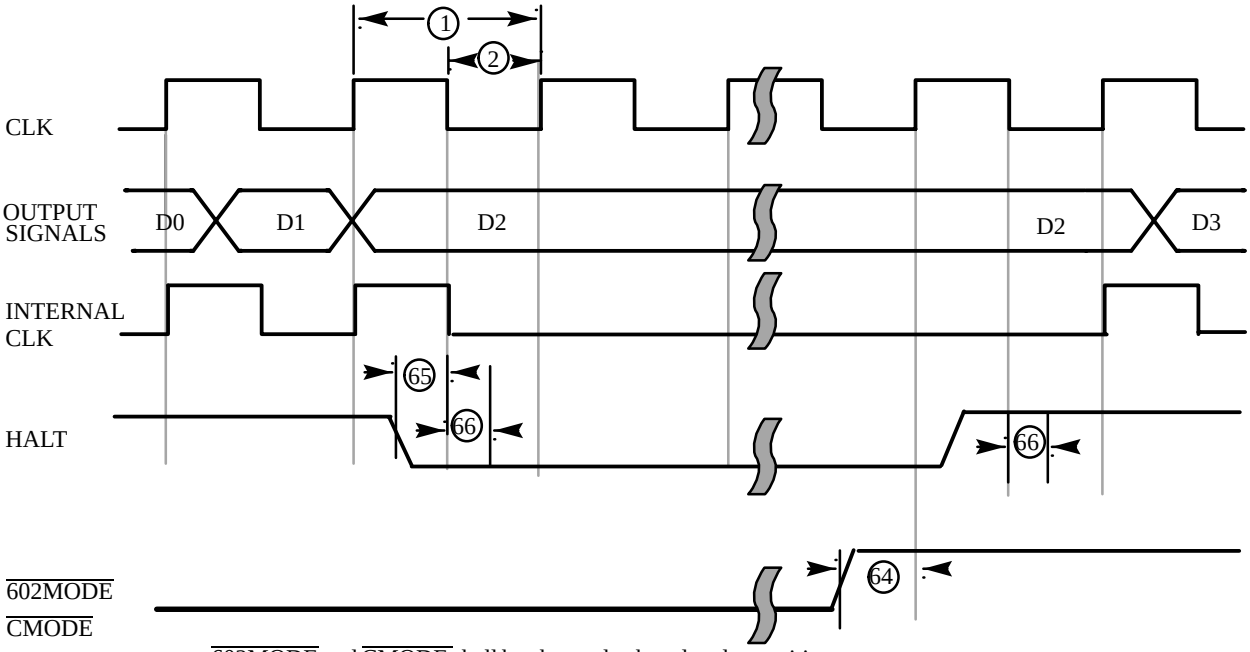


Figure 37. MASTER/CHECKER Signals

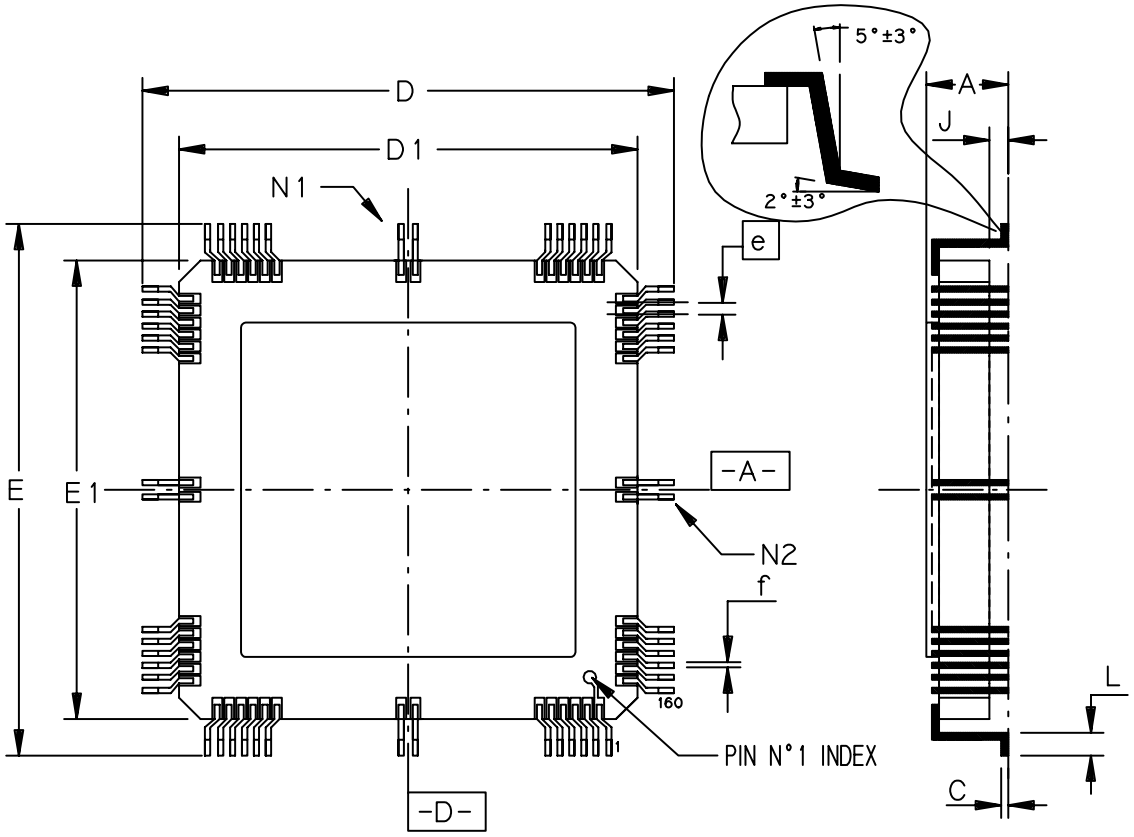


602MODE and CMODE shall be changed to be related to positive clock edge during RESET active or HALT.

Figure 38. HALT Signal

5.3. TSC692E Package Descriptions

5.3.1. 160-Pin MQFP-L Package



	MM		INCH	
	Min	Max	Min	Max
A	2.44	3.60	.096	.142
C	0.15 TYP		.006 TYP	
D	31.93	32.67	1.257	1.286
D1	26.93	27.47	1.060	1.082
E	31.93	32.67	1.257	1.286
E1	26.93	27.47	1.060	1.082
e	0.65 BSC		.0256 BSC	
f	0.30 REF		.012 REF	
J	0.50	1.00	.020	.040
L	1.21	1.41	.047	.056
N1	40		40	
N2	40		40	

5.3.2. 160-Pin MQFP-L Pin Assignment

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	A31	41	CLK	81	TDO	121	VSSI
2	A30	42	VSSI	82	TDI	122	VCCI
3	A29	43	VCCI	83	VCCI	123	VSSO
4	A28	44	VSSO	84	TMS	124	D15
5	A27	45	VCCO	85	$\overline{\text{TRST}}$	125	VCCO
6	VCCI	46	VCCT	86	VSSI	126	D14
7	A26	47	INST	87	TCLK	127	D13
8	A25	48	$\overline{\text{FEXC}}$	88	FINS2	128	VCCT
9	A24	49	$\overline{\text{FP}}$	89	FINS1	129	VCCI
10	A23	50	VSSI	90	VSSI	130	D12
11	A22	51	$\overline{\text{TOE}}$	91	VSSO	131	D11
12	VSSI	52	$\overline{\text{MDS}}$	92	D31	132	VSSO
13	A21	53	$\overline{\text{MHOLDA}}$	93	VCCI	133	VSSI
14	A20	54	VCCI	94	D30	134	VCCO
15	A19	55	$\overline{\text{MHOLDB}}$	95	VCCO	135	D10
16	A18	56	$\overline{\text{BHOLD}}$	96	D29	136	D9
17	A17	57	VSSI	97	D28	137	VSSO
18	A16	58	VSSO	98	VSSO	138	D8
19	VSSI	59	FNULL	99	VSSI	139	VCCI
20	A15	60	$\overline{\text{FHOLD}}$	100	D27	140	D7
21	A14	61	VCCO	101	D26	141	VCCO
22	A13	62	$\overline{\text{CHOLD}}$	102	D25	142	D6
23	A12	63	$\overline{\text{RESET}}$	103	VCCO	143	D5
24	VCCI	64	VSSI	104	D24	144	VSSO
25	A11	65	VCCI	105	VCCI	145	D4
26	A10	66	FCCV	106	VSSO	146	VSSI
27	A9	67	FIPAR	107	D23	147	D3
28	A8	68	CCCV	108	D22	148	D2
29	VSSI	69	VSSO	109	VSSI	149	D1
30	A7	70	VSST	110	VSST	150	VCCI
31	A6	71	FCC0	111	D21	151	VSST

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
32	VSST	72	FCC1	112	D20	152	VCCO
33	A5	73	VCCI	113	VCCO	153	D0
34	A4	74	FXACK	114	VSSO	154	DPAR
35	A3	75	$\overline{\text{HWERROR}}$	115	D19	155	$\overline{\text{MCERR}}$
36	A2	76	VSSI	116	VSSI	156	$\overline{\text{CMODE}}$
37	VCCI	77	FLUSH	117	D18	157	VSSO
38	A1	78	VCCO	118	D17	158	VSSI
39	A0	79	IFPAR	119	D16	159	$\overline{\text{HALT}}$
40	APAR	80	VSSO	120	DOE	160	$\overline{\text{602MODE}}$