LSI LOGIC

# LR3010/LR3010A Floating-Point Accelerator

Preliminary

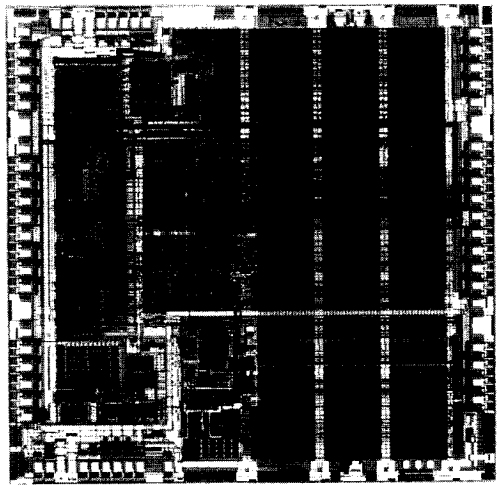
## Introduction

The LR3010 and LR3010A Floating-Point Accelerators (FPAs) from LSI Logic Corporation implement the floating-point coprocessor architecture of MIPS Microsystems in high-performance CMOS. Both FPAs are in full conformance with the MIPS specifications for floating-point units.

Current MIPS architecture has evolved from earlier RISC hardware and software development efforts at Stanford University. Developing the hardware and software together allowed system architects to make performance tradeoffs across the hardware/software boundary. Floating-point accelerator functions were implemented in hardware only if they measurably enhanced system performance without complicating the hardware design.

The LR3010 and LR3010A FPAs were designed to perform arithmetic operations on floating-point values. As an alternate execution unit for use with the LR3000 CPU series, the LR3010 monitors the same instruction stream as the CPU but only executes instructions that require its special architecture and register set. The LR3010 identifies FPA instructions by opcode, executes them and returns the results to memory or a CPU register.

As alternate execution units, the LR3010 series FPAs and LR3000 series CPUs are closely coupled in that they share the same data/instruction lines, have a similar pipelined architecture, execute optimized compiler code in parallel and share an instruction set that allows register-to-register transfers of data between units. Synchronization signals shared by the processor and coprocessors, such as $\overline{\text{CpSync}}$ and Clk2xPhi, make precise coordination between the alternate execution units possible.



**LR3010A Die**

Since the LR3010A is simply a faster version of the original LR3010, the term LR3010 will be used for both units in this datasheet.

## Features

- Implements the *IEEE Standard for Binary Floating-Point Arithmetic,* ANSI/IEEE Standard 754-1985
- Implements IEEE exception handling routines in hardware
- Provides double-precision floating-point operation
- Operates from 16.67 to 33.33 MHz
- Provides 64-bit wide internal data path for all floating-point operations
- Replaces NEC, Siemens, IDT and Performance floating-point units
- Package options include a ceramic, cavity-down, pin grid array (CPGA), a ceramic, cavity-down, leaded chip carrier (CLDCC) and a plastic quad flat pack (PQFP)
- Provides precise, efficient handling of pipeline stalls and exceptions
- Implements closely coupled, seamless main processor/coprocessor interface
- Supports integrated software including the RISC/os operating system (SVID-compliant version of UNIX) and high-performance optimizing compilers for C, Pascal, FORTRAN, Ada, COBOL and PL/1
- Supports complete development system in native hardware, software and applications development environment

## Block Diagram

Figure 1 shows a functional block diagram of the LR3010. Within the chip, separate paths exist for exponents and fractions. The fractions have separate operand paths, each 53 bits wide. Results of an operation are returned from a processing unit to the register unit where they are stored until the FPA drives them onto the data bus.

Separate modules exist for processing exponents and for addition, multiplication and division. The separate execution units allow simultaneous processing of several instructions at once, provided that the same resource, such as a mathematical processing unit (add, multiply, divide), is not required by more than one instruction at a time.

The left side of Figure 1 shows the signals that are shared by the FPA and CPU. One group of signals indicates when the main processor or a coprocessor is running, busy or in an exception processing mode. The second group consists of clock and timing signals that keep the processing cycles in both units synchronized. Signals and timing are discussed later in this datasheet.

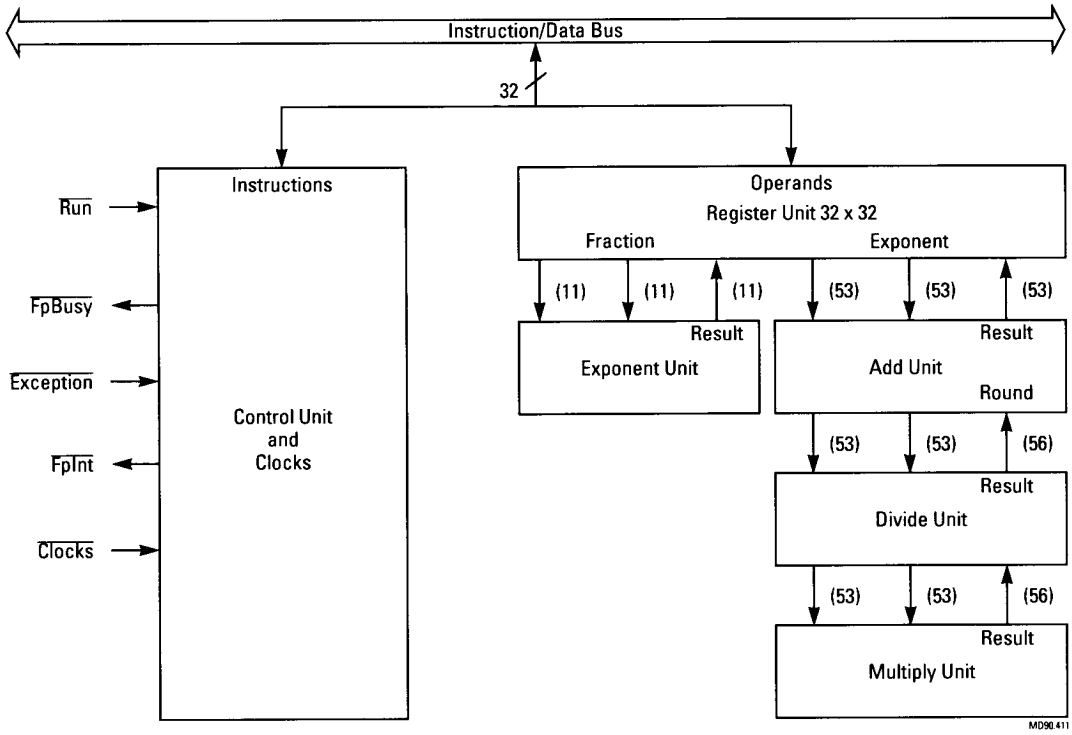



**Figure 1. FPA Functional Block Diagram**

## FPA Registers

The LR3010 FPA has 32 general-purpose registers designated FGR and two control registers designated FCR0 and FCR31.

**General-Purpose Registers** – Figure 2 shows the general-purpose and control registers in the FPA. The main processor sees the general purpose registers simply as 32 word-wide registers. The FPA sees them as building blocks for 64-bit, double-word registers suitable for double-precision representation.

As Figure 2 shows, all 64-bit floating-point registers (FPRs) consist of two floating-point general registers (FGR) and have even numbers; odd numbered registers are invalid, but are not checked by the hardware. General registers (FGRs) are numbered consecutively from 0 to 31.

The 64-bit wide FPA registers hold scalar floating-point values and permit overlapping and scheduling of floating-point operations. Each register holds one value of a single- or double-precision floating-point format. Two adjacent registers are used for double-precision operations. Single-precision floating-point operations (but not loads, stores and moves) leave the odd half of the result register undefined.

**FPA Registers**
(Continued)

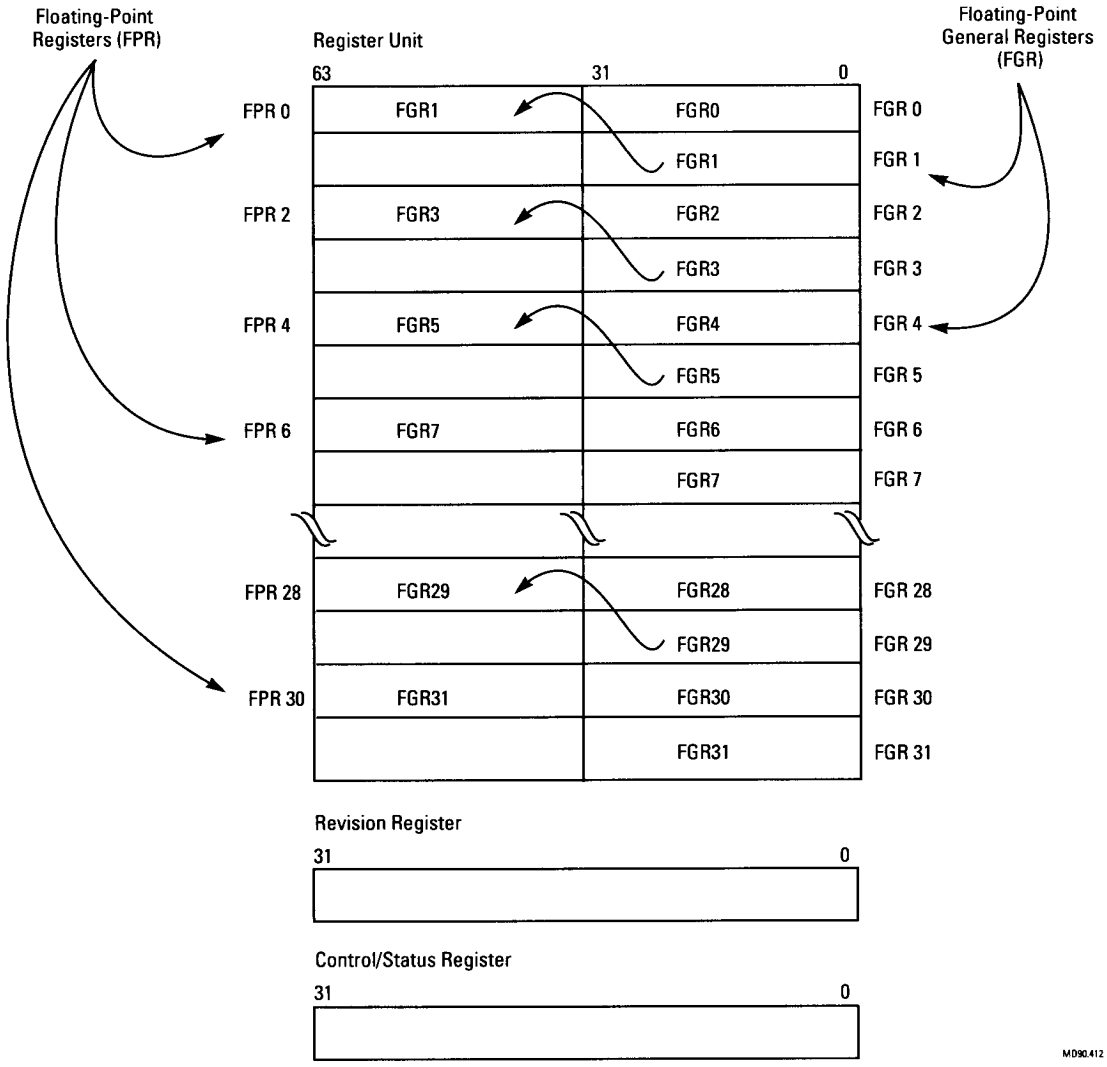



**Figure 2. Floating-Point Registers**

**Control Registers** – One coprocessor control register is dedicated to configuration and revision information. A second register is dedicated to status, diagnostics, exception handling, state saving and restoring, and control of rounding modes. Control Register 0 (FCR0) is the revision register. It is shown in Figure 3.

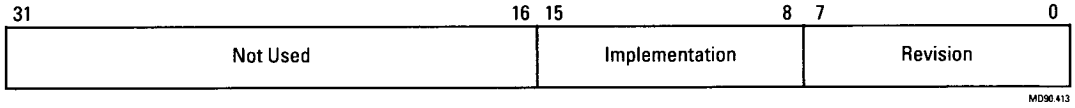



**Figure 3. Revision Register (FCR0)**

**FPA Registers**
(Continued)

The revision number is in the form *x.y* where *x* is held in bits [7:4] and *y* is held in bits [3:0].

The implementation field in the revision register is primarily for the internal use of each vendor. Software should not rely on these register fields for component identification.

Control Register 31 (FCR31) contains status and control information. It is accessed by instructions running in either kernel or user mode. It controls arithmetic rounding and the enabling of user level traps. It indicates exactly which exceptions occurred, if any, in the most recently executed instruction and which exception types have occurred since the FCR31 was last read.

Reading this register causes the FPA to complete all previous, uncompleted instructions in its pipeline. If an exception is taken as the pipeline is being emptied, the instruction that caused the exception is re-executed after the exception is serviced.

Figure 4 shows the register with internal fields.

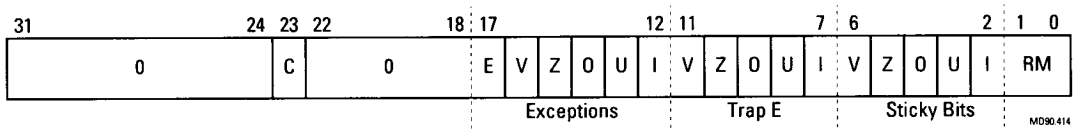



**Figure 4. Status/Control Register (FCR31)**

Each single letter mnemonic that identifies an exception in Figure 4 is defined in Table 1.

**Table 1. Exception Mnemonics**

| Exception Bit | Description |
|---|---|
| E | Unimplemented operation |
| V | Invalid operation |
| I | Inexact exception |
| Z | Divide-by-zero |
| O | Overflow exception |
| U | Underflow exception |

Control register FCR31 controls exception processing in the FPA. Three fields in the register control or track exception processing, and a fourth, rounding mode field, controls rounding and determines the default value that the FPA supplies when exceptions are not trapped. Exceptions include invalid operation, inexact exception, divide-by-zero, overflow and unimplemented operation exception, which is used when the hardware cannot execute the instruction or when underflow occurs.

The following paragraphs define each field in the FCR31control register.

**Rounding Mode (RM)** – The rounding modes include round to the nearest representable value, round toward zero, round toward $+\infty$ and round toward $-\infty$. The setting of the rounding mode bits modifies the exception processing defaults as shown in Table 5.

**Sticky Bits** – The LR3010 FPA sets the sticky bit for an exception regardless of whether traps are enabled. Unlike the exception field bits, sticky bits are not cleared as a result of executing floating-point instructions. They are only cleared by writing a new value into the FCR31 register.

**Trap Enabled (Trap E)** – When an exception occurs, the corresponding exception and sticky bits are both set. If the corresponding trap enable bit is set and the CPU is not already engaged in exception processing, it takes the trap and processes the exception.

**Exceptions** – The exception field is loaded (set or cleared) to reflect the exception status after each floating-point operation (excluding loads, stores and unformatted moves). The FPA flags an exception when an instruction is in the ALU stage of the FPA pipeline.

**Condition Bit (C)** – As shown in Figure 4, a floating-point compare instruction places the condition that it detected after a comparison into bit "C" of the FCR31 control register. The bit is set if the condition is true, and cleared if the condition is false. The bit is set or reset only after a comparison is made or a move control to coprocessor (CTC1) instruction is executed.

## Pipeline

The closely coupled interface between the CPU and FPA is possible in part because both processors have pipelined architecture. The FPA executes instructions through a 6-level pipeline. The CPU uses the write back or fifth level in the FPA pipeline to report its progress during exception processing and to maintain synchronization. Figure 5 illustrates the FPA pipestages. The shaded area shows the six processing stages that are executed simultaneously when the FPA is running.

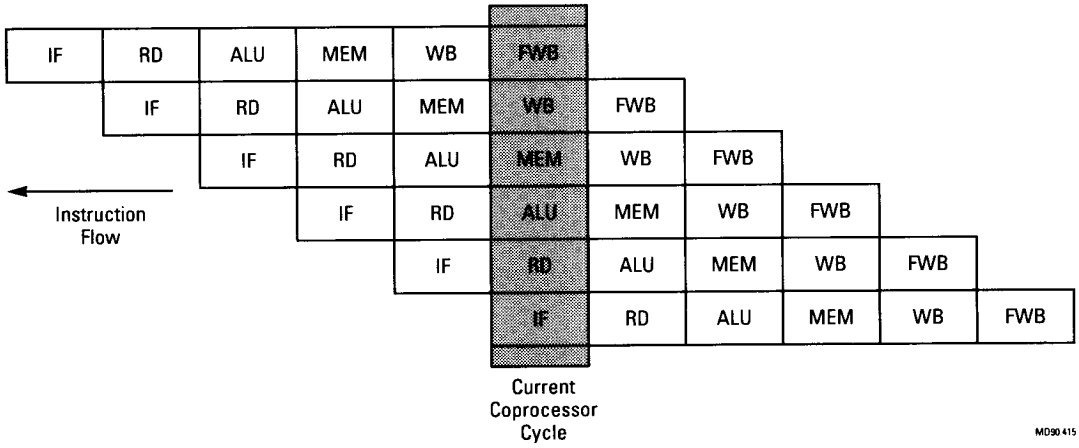



**Figure 5. FPA Pipeline**

The FPA pipeline levels and mnemonics shown in Figure 5 are defined in the following paragraphs.

**Instruction Fetch (IF)** – The LR3000 main processor generates all instruction addresses. The FPA simply decodes instructions in the instruction/data stream looking for those that require floating-point execution. FPA instructions are identified by their opcode, which occupies the first 6-bit field in the instruction format.

**Read Registers (RD)** – FPA instructions that are latched during the IF cycle are not actually decoded until the end of phase 1 of the RD cycle. During phase 2 of the RD cycle, the FPA fetches operands that the instruction requires from internal registers.

**Arithmetic Logic Unit (ALU)** – The instruction is executed during this pipestage. If the FPA detects an exception, it asserts $\overline{\text{FpInt}}$ during this cycle. If the instruction will require additional time to complete, the FPA initiates a stall by asserting FpBusy.

Stalls that are initiated by an exception in the ALU pipestage cause all instructions in the pipeline to stall. The instruction just entering the pipeline is discarded. After the exception is corrected, the discarded instruction will be reintroduced as the first instruction after the fixup cycle. To avoid frequent CPU stalls during lengthy ALU processing, the FPA overlaps instructions as long as there are no resource conflicts, data dependencies or exception conditions.

**Memory Access (MEM)** – Memory or the data cache is accessed during this cycle, if necessary. If the instruction is an FPA load or store, the FPA presents data to or captures data from the data bus during this pipestage. If, during the previous ALU cycle, the CPU trapped an interrupt for the instruction that is now in the FPA MEM cycle, then the CPU uses phase 2 of this cycle to notify the FPA that a trap was taken. The CPU asserts its $\overline{\text{Exception}}$ line during phase 2 of the MEM cycle to notify the FPA.

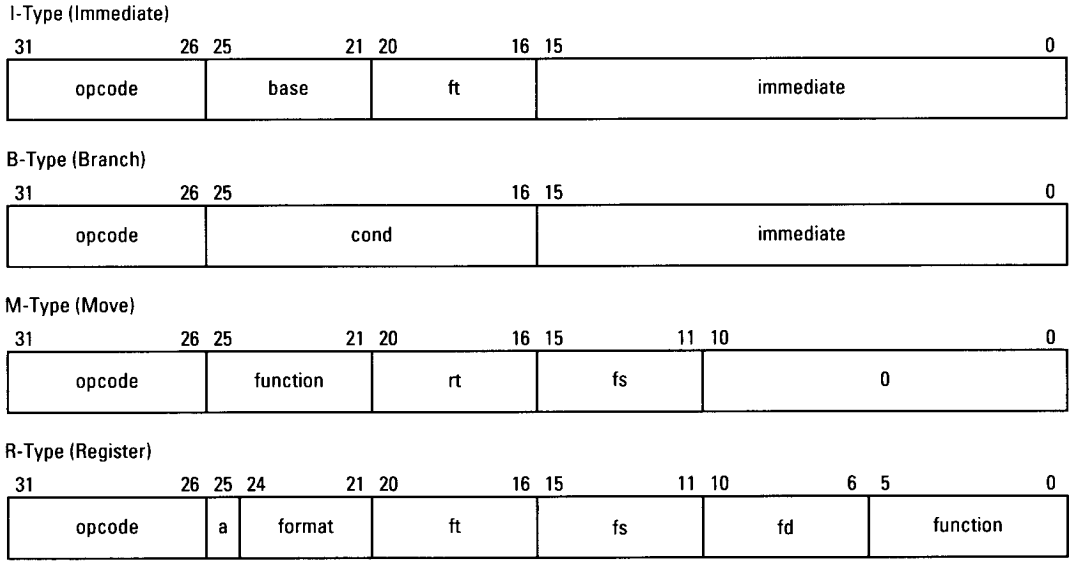
**Write Back (WB)** – The CPU informs the FPA of progress that it is making in exception processing during this pipestage. The FPA uses this stage only for that purpose.

**Floating-Point Write Back (FWB)** – The FPA writes processing results back to its registers during this pipestage.

## Instruction Set

FPA instructions are extensions of the basic MIPS CPU instruction set, so every FPA instruction consists of a single word (32 bits) aligned on a word boundary. Every instruction can be compiled to run efficiently with all other instructions in the CPU and FPA instruction set. Internally, the FPA is a more efficient engine for doing floating-point computations than the CPU, because it has separate execution units for mathematical operations, and it has extended floating-point registers that hold the operands for and results of double-precision floating-point computations. It also has hardware that converts between floating-point and fixed-point formats and compares values in floating-point representation. The optimizing compiler directs instructions that require any of these operations to the FPA.

**Instruction Format and Function** – There are only four FPA instruction formats. Each format is related to a particular instruction type with a specific function. The I-type (immediate) format is used for all loads and stores and is identical to the CPU load and store instruction except for the FPA opcode identifier. B-type (branch instruction) formats are variants of I-types in that an offset for a jump or branch address is contained in the instruction. A branch instruction is typically issued after an FPA comparison has set the condition bit in FPA control register FCR31. M-type instructions move data between CPU and FPA registers. R-type instructions specify both a source and destination register within the FPA where operands and results of a floating-point computation are stored. Instructions that require the R-type format include all arithmetical computations, conversions and comparisons. FPA instruction formats are illustrated in Figure 6.




**Instruction Format Legend**

| | |
|---|---|
| opcode | 6-bit operation code and coprocessor specifier |
| base | 5-bit CPU base register specifier |
| cond | 10-bit branch condition specifier |
| rt | 5-bit CPU source/destination general register specifier |
| ft | 5-bit FPA source/destination register specifier |
| fs | 5-bit FPA source register specifier |
| fd | 5-bit FPA destination register specifier |
| immediate | 16-bit address/branch displacement |
| function | 6-bit function code (MTC1, ADD, CVT.W) |
| format | 4-bit format specifier |
| 0 | undefined if non zero |
| a | 1-bit arithmetic operation identifier |

MD90.416

**Figure 6. FPA Instruction Set Formats**

**Instruction Set**
(Continued)

**FPA Instruction Fields** – The *opcode* and *function* instruction fields have constant 6- and 5-bit values. The *opcode* identifies the FPA as the selected execution unit for the next operation. The *function* field specifies the floating-point operation to perform. Several fields (rt, ft, fd, fs) in the instruction set specify registers that operands are to be taken from or that results are to be stored to. The *immediate* field contains a 16-bit address offset that is combined with the contents of a CPU register (identified as *base* in Figure 6) to complete the address for a load, store or branch instruction. The *format* field specifies a floating-point format for operands or results.

**Load and Store Instructions** –The operation and timing of FPA loads and stores are identical to those of the main processor. During FPA load and store operations, the CPU calculates memory addresses and controls the cache and memory interfaces. The FPA simply takes data from the bus during load operations and drives the bus during store operations. Data loaded into the FPA from caches or memory are not available to the instruction that immediately follows the load instruction, so there is a one instruction latency delay after a load operation.

The CPU reads the Data and Tag buses to check parity during a load operation. The FPA generates data parity during FPA store operations.

**Branch Instructions** – The BC1T instruction (branch if coprocessor 1 is true) takes a branch when a condition is true. The BC1F instruction (branch if coprocessor 1 is false) takes a branch when a condition is false. Branch instructions are actually executed by the CPU after the FPA asserts FpCond to indicate that the results of a comparison or branch condition are available in its control register (FCR31).

**FPA Move Instruction** – Four move instructions with the M-type format move data between the FPA and CPU. Two instructions move the contents of general registers from the CPU to the FPA or from the FPA to the CPU. The other two instructions move contents from the CPU general registers to the FPA control registers or from FPA control registers to CPU general registers.

**Floating-Point Computations** – The FPA performs arithmetic operations on values in floating-point representation. It also converts between fixed- and floating-point formats, and it compares the contents of two registers.

Four types of instruction are related to floating-point computation:

- Arithmetical operations include addition, subtraction, multiplication and division in single- or double-precision formats. These instructions are sometimes referred to as three operand register types.
- Floating-point operations include absolute value, move and negate operations. These instructions do not perform computations, but they do convert a numerical value to its absolute or negative value and move the results from a source to a destination register. These operations are sometimes referred to as two operand register types.
- Conversion operations convert operands or results in floating-point values to single or double floating-point values or to fixed-point values.
- Compare operations compare the values of two registers and post the result in the condition bit of the FCR31 control register. Table 2 shows the comparisons performed.

**Table 2. FPA Predicate Conditions**

| Mnemonic | Definition |
|---|---|
| F | False |
| T | True |
| UN | Unordered |
| OR | Ordered |
| EQ | Equal |
| NEQ | Not Equal |
| UEQ | Unordered or Equal |
| OLG | Ordered or Less Than or Greater Than |
| OLT | Ordered Less Than |
| OGE | Ordered Greater Than |
| OLE | Ordered Less Than or Equal |
| UGT | Unordered or Greater Than |
| ULE | Unordered or Less Than or Equal |
| OGT | Ordered Greater Than |
| SF | Signaling False |
| ST | Signaling True |
| NGLE | Not Greater Than or Less Than or Equal |
| GLE | Greater Than or Less Than or Equal |
| SEQ | Signaling Equal |
| SNE | Signaling Not Equal |
| NGL | Not Greater Than or Less Than |
| GL | Greater Than or Less Than |
| LT | Less Than |
| NLT | Not Less Than |
| NGE | Not Greater Than or Equal |
| GE | Greater Than or Equal |
| LE | Less Than or Equal |
| NLE | Not Less Than or Equal |
| NGT | Not Greater Than |
| GT | Greater Than |

**Instruction Set**
(Continued)

Table 2 extends the list of 26 predicates named in the IEEE Standard to include six predicates that test a comparison in the FPA arithmetical unit. Four mutually exclusive relations are possible: less than, greater than, equal and unordered. Invalid operations occur only when the comparisons include the less than and greater than characters, but not the unordered character in the *ad hoc* form of the predicate.

**Instruction Summary** – Table 3 lists all the instructions currently implemented in the LR3010 and LR3010A coprocessors. Any instructions that require computations not implemented in hardware cause the FPA to flag an unimplemented operation exception. The CPU then calls a routine from the floating-point emulation library to complete the instruction.

**Instruction Overlap** – When the FPA is running and not stalled, simultaneous execution is occurring on six separate instructions in its pipeline during each clock cycle. The pipeline allows concurrent operations as long as instructions do not require the same modular resource, such as the add, multiply or divide units. The consistent exceptions to concurrent operations are multiply and divide instructions, but even they allow some overlap. Figure 7 shows the execution cycles required for each FPA instruction. Notice that even in long instructions, there are only a few cycles when only one instruction is processed in the FPA.

**Table 3. Instruction Summary**

| Instruction | Description |
|---|---|
| **Load/Store/Move Instructions** | |
| LWC1 | Load Word to FPA |
| SWC1 | Store Word from FPA |
| MTC1 | Move Word to FPA |
| MFC1 | Move Word from FPA |
| CTC1 | Move Control Word to FPA |
| CFC1 | Move Control Word from FPA |
| **Computational Instructions** | |
| ADD.fmt | Floating-point Add |
| SUB.fmt | Floating-point Subtract |
| MUL.fmt | Floating-point Multiply |
| DIV.fmt | Floating-point Divide |
| ABS.fmt | Floating-point Absolute Value |
| MOV.fmt | Floating-point Move between Registers |
| NEG.fmt | Floating-point Negate |
| **Conversion Instructions** | |
| CVT.S.fmt | Floating-point Convert to Single-precision Floating-point |
| CVT.D.fmt | Floating-point Convert to Double-precision Floating-point |
| CVT.W.fmt | Floating-point Convert to Fixed-point |
| **Compare Instructions** | |
| C.cond.fmt | Floating-point Compare |
| **Branch Instructions** | |
| BC1T | Branch on Coprocessor 1 True |
| BC1F | Branch on Coprocessor 1 False |

**Instruction Set**
(Continued)

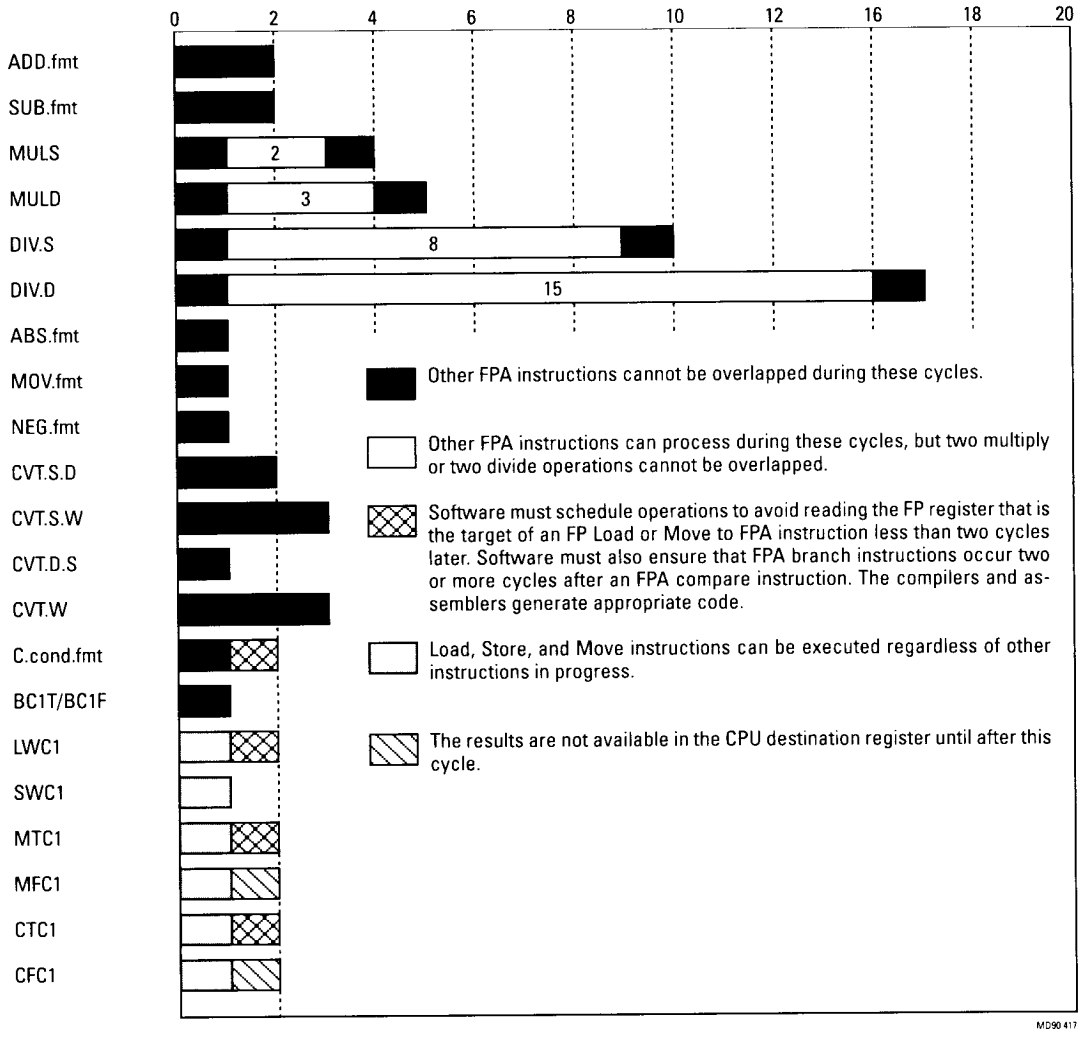



**Figure 7. Concurrent Instruction Processing**

## Exceptions

Five of the six of exceptions identified by the FPA are IEEE standard exceptions. The sixth, an unimplemented operation, is an IEEE optional exception. An unimplemented operation occurs when there is no hardware support for an instruction, and it must be processed in emulation using routines from the software floating-point library.

More exceptions are resolved in software by an unimplemented instruction exception when the exception traps are disabled than when they are enabled. This is shown in Table 4 by the relative number of "E" entries in columns 3 and 4. In the table, the letter "E" represents an unimplemented instruction exception.

Notice in Table 4 that some exception types can be signaled for the same cause and some types encompass several sources or causes. In all there are eight conditions that the IEEE includes within the five exception types.

**Exception Processing** – When an exception occurs, bits are set in both the exception and sticky bit fields of the FCR31 control register. (See the description of control register FCR31 above.) If the bit corresponding to the exception type is set in the trap enable field, a trap is taken. If the bit is not set in this field, the FPA begins default processing. The unimplemented operation exception (E), however, cannot be disabled; a trap is always taken.

When a floating-point exception trap is taken, the CE bit in the CPU Cause Register indicates that an external interrupt from the FPA is the cause of the trap. Precise exception handling is implemented, so the EPC register in the CPU contains the address of the instruction that caused the exception, and the operation and operands can be retrieved from memory after the fixup cycle.

For each IEEE standard exception, a sticky bit is set when the exception occurs, even if no corresponding exception trap is signaled. Unlike exception bits that are reset with each new instruction, the sticky bit can be reset only by writing a new value into the register. The sticky bits thus provide a record of the exceptions that have occurred, but do not indicate which instruction caused them. Sticky bits are saved or restored individually or as groups.

In the following paragraphs, each FPA-generated exception is discussed in detail.

**Invalid Operation Exception** – If one or both operands of an instruction are invalid, an invalid operation exception occurs. When the exception has a floating-point format and occurs without a trap, the result is a quiet NaN. If the destination has a fixed-point format, the result is indeterminate. The following are invalid operations:

- Addition or subtraction: magnitude subtraction of infinities, such as $(+\infty) + (-\infty)$ or $(-\infty) - (-\infty)$
- Multiplication: $0 \times \infty$ with any sign
- Division: 0/0 or $\infty/\infty$ with any sign
- Conversion of a floating-point number to a fixed-point format when an overflow, or operand value of infinity or NaN, precludes an accurate representation in that format
- Comparison of predicates involving "<" or ">" without "?" when the operands are "unordered"
- Any operation on a signaling NaN

Software may simulate the invalid operation exception for other operations that are invalid for the given source operands, such as IEEE specified functions implemented in software. Examples include: 1) remainder, *x* REM *y*, where *y* is zero or *x* is infinite, or 2) conversion of a floating-point number to a decimal format whose value causes an overflow or is infinity or NaN, or 3) transcendental functions, such as ln(-5).

**Table 4. Exception Causing Conditions**

| FPA Condition | IEEE Std | Trap Enable | Trap Disable | Description |
|---|---|---|---|---|
| Inexact result | I | I | I | Loss of accuracy |
| Exponent overflow | O I | O I | O I | Normalized exponent > Emax |
| Divide-by-zero | Z | Z | Z | Exponent = Emin - 1, mantissa = 0 |
| Overflow on convert | V | V | E | Source out of integer range |
| Signaling NaN source | V | V | E | A quiet NaN source generates a quiet NaN result |
| Invalid operation | V | V | E | For example, 0/0 |
| Exponent underflow | U | E | E | Normalized exponent < Emin |
| Denormalized source | — | E | E | Exponent = Emin -1, mantissa <> 0 |

**Exceptions**
(Continued)

**Divide-by-Zero Exception** – This exception occurs during a divide instruction if the divisor is zero and the dividend is a finite nonzero number. When no trap occurs, the result is a correctly signed infinity ($\infty$). If divide-by-zero traps are enabled, the result register is not modified, and the source registers are preserved.

Software may simulate this exception for other operations that produce a signed infinity, such as ln(0), csc(0) and $0^{-1}$.

**Overflow Exception** – The overflow exception is signaled when what would have been the magnitude of the rounded floating-point result, were the exponent range unbounded, is larger than the destination format's largest finite number. The result, when no trap occurs, is determined by the rounding mode and the sign of the intermediate result.

If overflow traps are enabled, the result register is not modified, and the source registers are preserved.

**Underflow Exception** – The FPA never generates an underflow exception and never sets the U bit in either the exception or sticky bit fields of the control/status register. If the FPA detects a condition that could be either an underflow or loss of accuracy, it generates an unimplemented operation exception. Size on underflow is detected after rounding. Loss of accuracy is detected as an inexact result.

**Inexact Exception** – The FPA signals an inexact exception when the rounded result of an operation is not exact or overflows without an overflow trap. If no other trap occurs, the rounded or overflowed result is delivered to the destination register. If inexact exception traps are enabled, the result register is not modified, and the source registers are preserved.

**Unimplemented Operating Exception** – If an operation is specified that the hardware cannot perform, an unimplemented operation exception occurs. This operation always causes a trap. Since the trap cannot be disabled, the instruction must be emulated in software. Normal instruction execution is then resumed.

This exception also occurs when the FPA attempts to execute an instruction with an operation code or format code that has been reserved for future architectural definitions.

This exception may also be signaled when unusual operands or results are detected, and no hardware is provided to handle the condition. Instances of this include, but are not limited to, denormalized operands or results, NaN operands and trapped overflow or underflow conditions. The use of this exception for such conditions is optional.

**Defaults** – When the FPA detects an exception but cannot obtain an exception processing stall from the CPU, the FPA supplies a default value and continues processing. Table 5 summarizes the default actions taken for each exception. Note that the default value supplied depends on the setting of the rounding mode bits.

**Table 5. Default for Exceptions**

| Exception | Rounding Mode | Default Action |
|---|---|---|
| Invalid (V) | — | Supply a quiet NaN. |
| Divide-by-Zero (Z) | — | Supply a properly signed ∞. |
| Overflow (O) | RN | Round overflow values to ∞ with the sign of the intermediate result. |
| | RZ | Round overflow values to the largest finite number in the format with the sign of the intermediate result. |
| | RP | Round negative overflow to the largest negative finite number in the format. Round positive overflows to +∞. |
| | RM | Round positive overflow to the largest finite number in the format. Round positive overflows to −∞. |
| Underflow (U) | — | Produce an unimplemented exception. |
| Inexact (I) | — | Supply a rounded result. |

## Signal Descriptions

The FPA does not communicate with any chip other than the CPU, so it has minimal interface signals. Timing between the FPA and the CPU is synchronized by shared clock signals. The interface signals and timing signals for the FPA are shown in Figure 8.

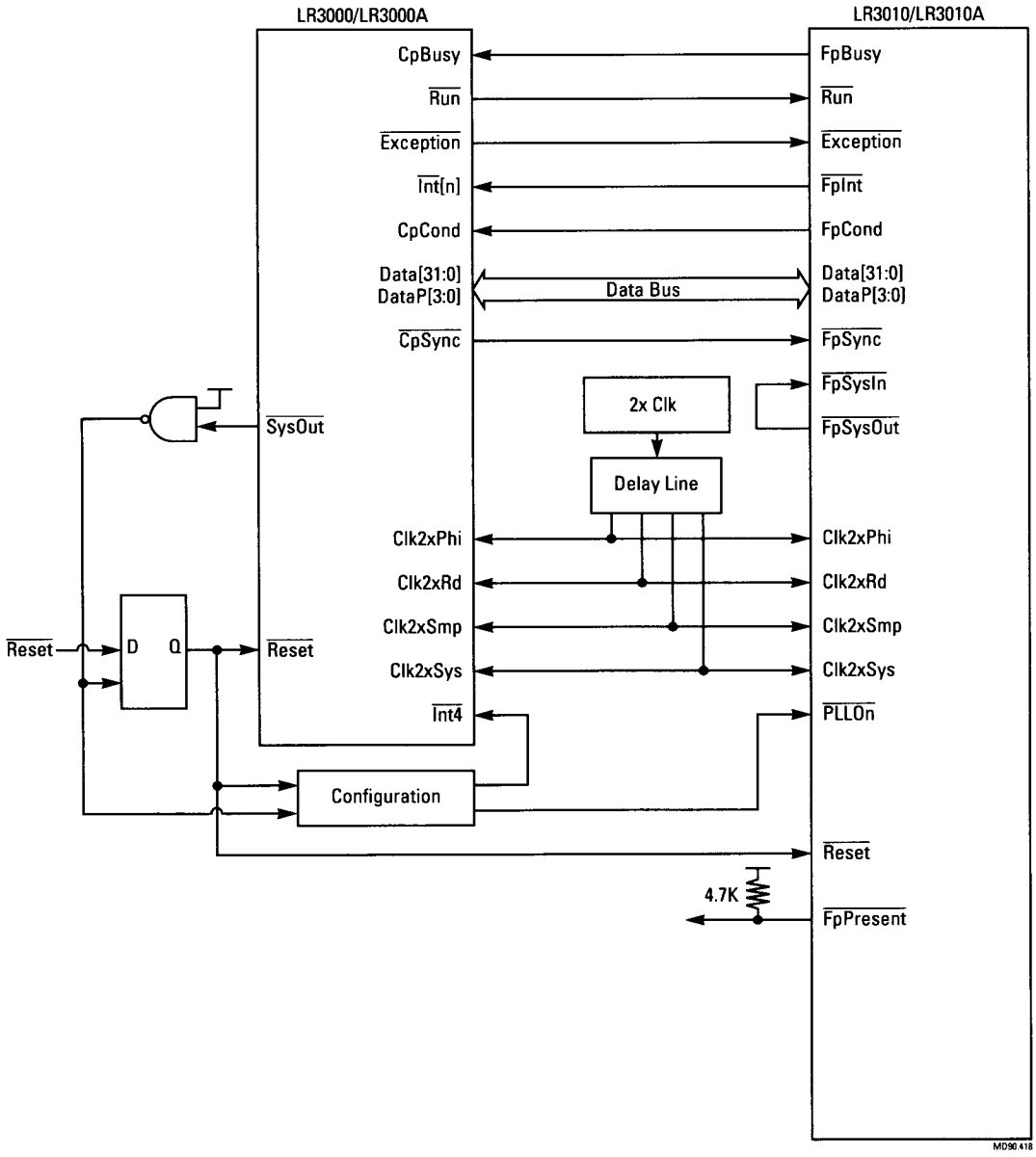



**Figure 8. FPA Signals**

**Signal Descriptions** (Continued)

All FPA signals are listed in alphabetical order in the following paragraphs. A line over a signal name, such as $\overline{\text{Run}}$, indicates the signal is active LOW.

**Clk2xPhi**
**2x Phase Clock** – Clk2xPhi is a double frequency clock that determines the relative position of internal phases 1 and 2. Other double frequency clocks are set relative to Clk2xPhi.

**Clk2xRd**
**2x Read Clock** – Clk2xRd is a double frequency clock that determines the disable point for the data drivers.

**Clk2xSys**
**2x System Clock** – Clk2xSys is a double frequency clock input used to generate $\overline{\text{FpSysOut}}$ in the FPA. See the $\overline{\text{FpSysOut}}$ signal description in this subsection for details on clock delays.

**Clk2xSmp**
**2x Sample Clock** – Clk2xSmp is a double frequency clock that determines the sample point for data coming into the FPA.

**Data[31:0]**
**Data Bus** – There are no address bus connections on the FPA coprocessor interface. The main processor generates all addresses to access instruction and data caches as well as memory. The FPA simply monitors the data bus to identify and then execute floating-point instructions. Instructions occupy the 32-bit, multiplexed instruction/data bus during the first phase of an instruction/data cycle. Data occupies the bus during the second.

**DataP[3:0]**
**Data Parity** – The FPA and CPU also share the data parity bus (DataP[3:0]). The main processor is normally responsible for parity, but during FPA store operations, the FPA generates even parity for each byte in the 32-bit data word. DataP0 contains parity for Data[7:0]; DataP1 contains parity for Data[15:8] and so on.

**$\overline{\text{Exception}}$**
**Exception** – The CPU $\overline{\text{Exception}}$ output signal is connected to the FPA $\overline{\text{Exception}}$ pin. The CPU asserts the $\overline{\text{Exception}}$ signal during run cycles to indicate it is processing an exception and during stall cycles to indicate additional information about the stall. The meaning of the signal depends on whether it is asserted during phase 1 or 2 of a run or stall cycle. There are four possible interpretations summarized in Table 6.

**FpBusy**
**Floating-point Busy** – Floating-point Busy (FpBusy) is asserted to initiate or terminate a coprocessor busy stall. The FPA asserts this signal during the ALU pipestage of an instruction processing cycle if it needs more time to resolve a data dependency, to continue an ALU operation past 1 clock cycle or to indicate it has detected an exception. After asserting FpBusy, the FPA disregards the data pair presented during the previous run cycle.

When the CPU detects FpBusy, it initiates a stall, which is terminated when the FPA deasserts FpBusy. The CPU then performs a fixup cycle, and the ignored data pair from the last run cycle is presented again. After the fixup cycle, normal run cycles resume.

**FpCond**
**Floating-point Condition** – The Floating-point Condition (FpCond) signal normally drives CpCond1 on the CPU. The FPA asserts this signal to indicate the result of the last comparison is available in its status register (FCR31). A branch may be taken as a result of the state of the compare instruction that activates this signal.

**Table 6. Exception Signal Encoding**

| Cycle | Phase | Mnemonic | Meaning (when asserted) |
|---|---|---|---|
| Run | 1 | $\overline{\text{Exc1W}}$ | An exception has occurred for the instruction currently in its *writeback* pipestage. |
| | 2 | $\overline{\text{IntGr2M}}$ | An interrupt is being granted for the instruction currently in the FPA *memory access* pipestage. |
| Stall | 1 | $\overline{\text{Fixup1}}$ | The current stall cycle is a *fixup* cycle and the contents of the data bus are valid. |
| | 2 | $\overline{\text{CpBusy2}}$ | The current stall cycle is a *Coprocessor Busy* stall. |

**Signal Descriptions**
(Continued)

**FpInt**
**Floating-point Interrupt** – The FPA signals exceptions by asserting FpInt, which is normally connected to a CPU interrupt pin Int1. The CPU samples its interrupt inputs during phase 2 of all run cycles and during the final fixup stage of a stall sequence.

FpInt is asserted during the ALU pipestage of the instruction that caused the exception.

To signal the FPA that it has been granted an interrupt and a processing stall, the CPU drives its Exception output LOW during the second phase of the memory access pipestage of the instruction that caused the exceptions. The FPA interprets its Exception pin being driven LOW at that time as an interrupt grant (*IntGr2M*) signal.

**FpPresent**
**Floating-point Present** – Floating-point Present (FpPresent) signals the existence of an FPA in the system. The signal, pulled up through 4.7 Kohms as shown in Figure 8, is typically used in a configuration register. When the FpPresent pin is driven HIGH, the FPA 3-states its outputs.

**FpSync**
**Floating-point Synchronization** – CpSync is generated by the main processor to synchronize data transfers between it and its coprocessors. The FPA receives CpSync at its FpSync input pin, compares its output clock (FpSysOut) with the CpSync signal, then dynamically adjusts a variable delay in its clock pulse to minimize timing skew between the two units. Also see the Phase Locked Loop On (PLLOn) signal definition.

**FpSysIn**
**Floating-point System In** – The FpSysIn signal is an input that monitors FpSysOut to determine its phase relation to the CPU SysOut signal. See FpSysOut for details.

**FpSysOut**
**Floating-point System Out** – In order to operate the processor system at maximum speed, skew between the processor and coprocessor synchronization signals must be minimized. The main processor provides a fixed *phase delay* in its input clock paths that is enabled to facilitate de-skewing of the processor and coprocessors. This phase delay is in addition to the delay that is introduced by clock buffering.

The FPA contains a variable delay in its input clock paths that is dynamically adjusted after comparing its output clock (FpSysOut) to the processor CpSync output. CpSync is nominally identical to SysOut and is provided specifically for processor-coprocessor synchronization. To help match the clocks, the FPA output clock FpSysOut is loaded to match the CpSync load.

The fixed phase delay is enabled by asserting the main processor Int4 input during reset. (Refer to Processor Initialization in the *LR3000 and LR3000A MIPS RISC Microprocessor Users Manual* for a discussion of the reset operation and the interrupt inputs.) When disabled, the Clk2xSys-to-SysOut delay assumes its nominal value.

**PLLOn**
**Phase Locked Loop On** – PLLOn enables the phase locked loop circuitry in the FPA that de-skews the high-speed timing between it and the CPU.The signal must be asserted during the reset cycle, and after reset it must remain asserted and held to ensure proper operation of the system. See the *LR3000 and LR3000A MIPS RISC Microprocessor Users Manual* from LSI Logic for reset procedures.

**Reset**
**Reset** – This synchronous initialization is driven LOW to reset the system to a known state. Reset is held LOW with the Phase Locked Loop On (PLLOn) signal to synchronize the CPU and FPA before processing begins.

**Run**
**Run** – The CPU drives the Run signal LOW to signal coprocessors that it is performing a run cycle. It drives Run HIGH when it is stalled. Processors and coprocessor are either running or stalled; there are no other operating states.

## LR3010 Specifications

The following tables show the electrical specifications and timing for the LR3010. Table 7 shows the LR3010 maximum ratings. Operation beyond the limits set forth in this table may impair the useful life of the device. Not more than one output should be shorted at a time. Duration of a short should not last more than 30 seconds.

### Table 7. Maximum Ratings

| Parameter | Symbol | Min | Max | Units |
|---|---|---|---|---|
| Supply Voltage | VCC | -0.5 | +7.0 | V |
| Input Voltage | VIN | -0.5[1] | +7.0 | V |
| Storage Temperature | TST | -65 | +150 | °C |

Note:
1. VIN Min = -3.0 V for pulse width less than 15 ns.

### Table 8. FPA Operating Range

| Range | Ambient Temperature | VCC |
|---|---|---|
| Commercial | 0°C to 70°C | 5 V ± 5% |
| Military | -55°C to 125°C | 5 V ± 5% |

### Table 9. DC Electrical Characteristics

| Parameter | Symbol | 16.67 MHz | | 20 MHz | | 25 MHz | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| Output High Voltage[1] | VOH | 3.5 | | 3.5 | | 3.5 | | V |
| Output Low Voltage[2] | VOL | | 0.4 | | 0.4 | | 0.4 | V |
| Input High Voltage | VIH[3] | 2.0 | VCC | 2.0 | VCC | 2.0 | VCC | V |
| Input Low Voltage | VIL[4] | -0.5 | 0.8 | -0.5 | 0.8 | -0.5 | 0.8 | V |
| Input High Voltage | VIHC[3,5] | 4.0 | VCC | 4.0 | VCC | 4.0 | VCC | V |
| Input Low Voltage | VILC[5] | -0.5 | 0.4 | -0.5 | 0.4 | -0.5 | 0.4 | V |
| Input High Voltage | VIHS[3,6] | 2.5 | VCC | 2.5 | VCC | 2.5 | VCC | V |
| Input Low Voltage | VILS[6] | -0.5 | 0.4 | -0.5 | 0.4 | -0.5 | 0.4 | V |
| Input Capacitance | CIn | | 10 | | 10 | | 10 | pF |
| Output Capacitance | COut | | 10 | | 10 | | 10 | pF |
| Load Capacitance | CLd[7] | | 50 | | 50 | | 50 | pF |
| Operating Current[8] | ICC | | 600 | | 650 | | 750 | mA |

Notes:
1. Test conditions for Output High Voltage: VCC = Min; IOH = -4 mA.
2. Test conditions for Output Low Voltage: VCC = Min; IOL = 4 mA.
3. All voltages designated VCC are at a nominal VCC + 0.5 V.
4. VIL Min = -3.0 V for pulse width less than 15 ns.
5. VIHC and VILC apply to $\overline{\text{Run}}$ and $\overline{\text{Exception}}$.
6. VIHS and VILS apply to Clk2xSys, Clk2xSmp, Clk2xRd, Clk2xPhi, $\overline{\text{FpSysIn}}$, $\overline{\text{FpSync}}$ and $\overline{\text{Reset}}$.
7. CLd is expressed as a per pin average. If the average capacitance across all outputs is above this level, the specification is violated. Operation above the CLd maximum may shorten the useful life of the FPA.
8. Test conditions for Operating Current: VCC = 5.5 V, 70°C.

### Table 10. Capacitive Load Deration

| Parameter | Symbol | 16.67 MHz | | 20 MHz | | 25 MHz | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| Load Deration | CLD | 0.5 | 2 | 0.5 | 1 | 0.5 | 1 | ns/25 pF |

## LR3010 Specifications (Continued)

### Table 11. Current Leakage Parameters

| Parameter | Symbol | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| Output Leakage Current (Low) | IOZL | VCC = 5.25 V<br>V = 0 V | -10 | +10 | µA |
| Output Leakage Current (High) | IOZH | VCC = 5.25 V<br>V = 5.25 V | -10 | +10 | µA |
| Input Low Current | IIL | VCC = 5.25 V<br>V = 0 V | -10 | +10 | µA |
| Input High Current | IIH | VCC = 5.25 V<br>V = 5.25 V | -10 | +10 | µA |

### Table 12. AC Clock Parameters

| Parameter[1] | Symbol | 16.67 MHz | | 20 MHz | | 25 MHz | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| Input Clock High[2] | $t_{CkHigh}$ | 12 | | 10 | | 8 | | ns |
| Input Clock Low[2] | $t_{CkLow}$ | 12 | | 10 | | 8 | | ns |
| Input Clock Period | $t_{CkP}$ | 30 | 1000 | 25 | 1000 | 20 | 1000 | ns |
| Clk2xSys to Clk2xSmp | | 0 | tCyc/4 | 0 | tCyc/4 | 0 | tCyc/4 | ns |
| Clk2xSmp to Clk2xRd | | 0 | tCyc/4 | 0 | tCyc/4 | 0 | tCyc/4 | ns |
| Clk2xSmp to Clk2xPhi | | 9 | tCyc/4 | 7 | tCyc/4 | 5 | tCyc/4 | ns |

Note:
1. The clock parameters apply to all four 2xClocks: Clk2xSys, Clk2xSmp, Clk2xRd and Clk2xPhi.
2. Clock transition time ≤ 5 ns for 16.67, 20 and 25 MHz machines.

### Table 13. AC Setup and Hold Timing

| Parameter | Symbol | Load (pF) | 16.67 MHz | | 20 MHz | | 25 MHz | | Units |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| Data Valid | $t_{DVal}$ | 25 | 2 | 3 | 1 | 3 | 1 | 3 | ns |
| Data Enable[1] | $t_{DEn}$ | | -1 | -2 | -0.5 | -1.5 | -0.5 | -1.5 | ns |
| Data Disable[1] | $t_{DDis}$ | | 0 | -1 | 0 | -1 | 0 | -0.5 | ns |
| Data Setup | $t_{DS}$ | | 9 | | 8 | | 7 | | ns |
| Data Hold | $t_{DH}$ | | -2.5 | | -2.5 | | -2.5 | | ns |
| FpBusy | $t_{FpBusy}$ | | 0 | 15 | 0 | 13 | 0 | 10 | ns |
| Fp Condition | $t_{FpCond}$ | | 0 | 35 | 0 | 30 | 0 | 25 | ns |
| Exception Set Up | $t_{ExS}$ | | 10 | | 9 | | 8 | | ns |
| Exception Hold | $t_{ExH}$ | | 0 | | 0 | | 0 | | ns |
| Fp Interrupt | $t_{FpInt}$ | | 0 | 40 | 0 | 30 | 0 | 25 | ns |
| Fp Move To | $t_{FpMove}$ | | 0 | 35 | 0 | 30 | 0 | 25 | ns |
| Run Setup | $t_{RunS}$ | | 10 | | 9 | | 8 | | ns |
| Run Hold | $t_{RunH}$ | | -2 | | -2 | | -2 | | ns |

Note:
1. Parameter guaranteed by design.

## LR3010A Specifications (Continued)

The following tables show the electrical specifications and timing for the LR3010A. All LR3010A specifications are preliminary and subject to change. Table 14 shows the LR3010A maximum ratings. Operation beyond the limits set forth in this table may impair the useful life of the device. Not more than one output should be shorted at a time. Duration of a short should not last more than 30 seconds.

### Table 14. Maximum Ratings

| Parameter | Symbol | Min | Max | Units |
|---|---|---|---|---|
| Supply Voltage | VCC | -0.5 | +7.0 | V |
| Input Voltage | VIN | -0.5[1] | +7.0 | V |
| Storage Temperature | TST | -65 | +150 | °C |

Note:
1. VIN Min = -3.0 V for pulse width less than 15 ns.

### Table 15. FPA Operating Range

| Range | Ambient Temperature | VCC |
|---|---|---|
| Commercial | 0°C to 70°C | 5 V ± 5% |
| Military | -55°C to 125°C | 5 V ± 5% |

### Table 16. DC Electrical Characteristics

| Parameter | Symbol | 16.67 MHz | | 25MHz | | 33.33 MHz | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| Output High Voltage[1] | VOH | 3.5 | | 3.5 | | 3.5 | | V |
| Output Low Voltage[2] | VOL | | 0.4 | | 0.4 | | 0.4 | V |
| Input High Voltage | VIH[3] | 2.0 | VCC | 2.0 | VCC | 2.0 | VCC | V |
| Input Low Voltage | VIL[4] | -0.5 | 0.8 | -0.5 | 0.8 | -0.5 | 0.8 | V |
| Input High Voltage | VIHC[3,5] | 4.0 | VCC | 4.0 | VCC | 4.0 | VCC | V |
| Input Low Voltage | VILC[5] | -0.5 | 0.4 | -0.5 | 0.4 | -0.5 | 0.4 | V |
| Input High Voltage | VIHS[3,6] | 3.0 | VCC | 3.0 | VCC | 3.0 | VCC | V |
| Input Low Voltage | VILS[6] | -0.5 | 0.4 | -0.5 | 0.4 | -0.5 | 0.4 | V |
| Input Capacitance | CIn | | 10 | | 10 | | 10 | pF |
| Output Capacitance | COut | | 10 | | 10 | | 10 | pF |
| Load Capacitance | CLd[7] | | 50 | | 50 | | 50 | pF |
| Operating Current[8] | ICC | | 500 | | 600 | | 700 | mA |

Notes:
1. Test conditions for Output High Voltage: VCC = Min; IOH = -4 mA.
2. Test conditions for Output Low Voltage: VCC = Min; IOL = 4 mA.
3. All voltages designated VCC are at a nominal VCC + 0.5 V.
4. VIL Min = -3.0 V for pulse width less than 15 ns.
5. VIHC and VILC apply to $\overline{\text{Run}}$ and $\overline{\text{Exception}}$.
6. VIHS and VILS apply to Clk2xSys, Clk2xSmp, Clk2xRd, Clk2xPhi, $\overline{\text{FpSysIn}}$, $\overline{\text{FpSync}}$ and $\overline{\text{Reset}}$.
7. CLd is expressed as a per pin average. If the average capacitance across all outputs is above this level, the specification is violated. Operation above the CLd maximum may shorten the useful life of the FPA.
8. Test conditions for Operating Current: VCC = 5.5 V, 70°C.

### Table 17. Capacitive Load Deration

| Parameter | Symbol | 16.67 MHz | | 25 MHz | | 33.33 MHz | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| Load Deration | CLD | 0.5 | 2 | 0.5 | 1 | 0.5 | 1 | ns/25 pF |

**LR3010A Specifications**
(Continued)

### Table 18. Current Leakage Parameters

| Parameter | Symbol | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| Output Leakage Current (Low) | IOZL | VCC = 5.25 V<br>V = 0 V | -10 | +10 | µA |
| Output Leakage Current (High) | IOZH | VCC = 5.25 V<br>V = 5.25 V | -10 | +10 | µA |
| Input Low Current | IIL | VCC = 5.25 V<br>V = 0 V | -10 | +10 | µA |
| Input High Current | IIH | VCC = 5.25 V<br>V = 5.25 V | -10 | +10 | µA |

### Table 19. AC Clock Parameters

| Parameter[1] | Symbol | 16.67 MHz | | 25 MHz | | 33.33 MHz | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | Min | Max | |
| Input Clock High[2] | $t_{CkHigh}$ | 12 | | 8 | | 6 | | ns |
| Input Clock Low[2] | $t_{CkLow}$ | 12 | | 8 | | 6 | | ns |
| Input Clock Period | $t_{CkP}$ | 30 | 1000 | 20 | 1000 | 15 | 1000 | ns |
| Clk2xSys to Clk2xSmp | | 0 | tCyc/4 | 0 | tCyc/4 | 0 | tCyc/4 | ns |
| Clk2xSmp to Clk2xRd | | 0 | tCyc/4 | 0 | tCyc/4 | 0 | tCyc/4 | ns |
| Clk2xSmp to Clk2xPhi | | 9 | tCyc/4 | 5 | tCyc/4 | 4.5 | tCyc/4 | ns |

Note:
1. The clock parameters apply to all four 2xClocks: Clk2xSys, Clk2xSmp, Clk2xRd and Clk2xPhi.
2. Clock transition time ≤ 5 ns for 16.67 and 25 MHz machines and ≤ 2.5 ns for 33.3 MHz machines.

### Table 20. AC Setup and Hold Timing

| Parameter | Symbol | Load (pF) | 16.67 MHz | | 25 MHz | | 33.33 MHz | | Units |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| Data Valid | $t_{DVal}$ | 25 | | 3 | | 3 | | 2.5 | ns |
| Data Enable[1] | $t_{DEn}$ | | | -2 | | -1.5 | | -1.5 | ns |
| Data Disable[1] | $t_{DDis}$ | | | -1 | 0 | -0.5 | 0 | -0.5 | ns |
| Data Setup | $t_{DS}$ | | 9 | | 7 | | 5 | | ns |
| Data Hold | $t_{DH}$ | | -2.5 | | -2.5 | | -2.5 | | ns |
| FpBusy | $t_{FpBusy}$ | | 0 | 15 | 0 | 10 | 0 | 7 | ns |
| Fp Condition | $t_{FpCond}$ | | 0 | 35 | 0 | 25 | 0 | 17 | ns |
| Exception Set Up | $t_{ExS}$ | | 10 | | 7 | | 4 | | ns |
| Exception Hold | $t_{ExH}$ | | 0 | | 0 | | 0 | | ns |
| Fp Interrupt | $t_{FpInt}$ | | | 40 | 0 | 25 | 0 | 19 | ns |
| Fp Move To | $t_{FpMove}$ | | 0 | 35 | 0 | 25 | 0 | 16 | ns |
| Run Setup | $t_{RunS}$ | | 17 | | 15 | | 12.5 | | ns |
| Run Hold | $t_{RunH}$ | | -2 | | -2 | | -2 | | ns |
| Stall Setup | $t_{StallS}$ | | 10 | | 9 | | 7 | | ns |
| Stall Hold | $t_{StallH}$ | | -2 | | -2 | | -2 | | ns |

Note:
1. Parameter guaranteed by design.

## Timing Waveforms

As a coprocessor, the FPA does not communicate with any chips in the system except the CPU. It monitors the data/instruction lines looking for FPA instructions and operands or places data on the bus under CPU control. As a result, the only timing diagrams relevant to the FPA show its interaction with the CPU.

The relation between various clock signals that were defined in the proceeding section is shown in Figure 13. The rest of the figures in this section illustrate critical FPA timing parameters.

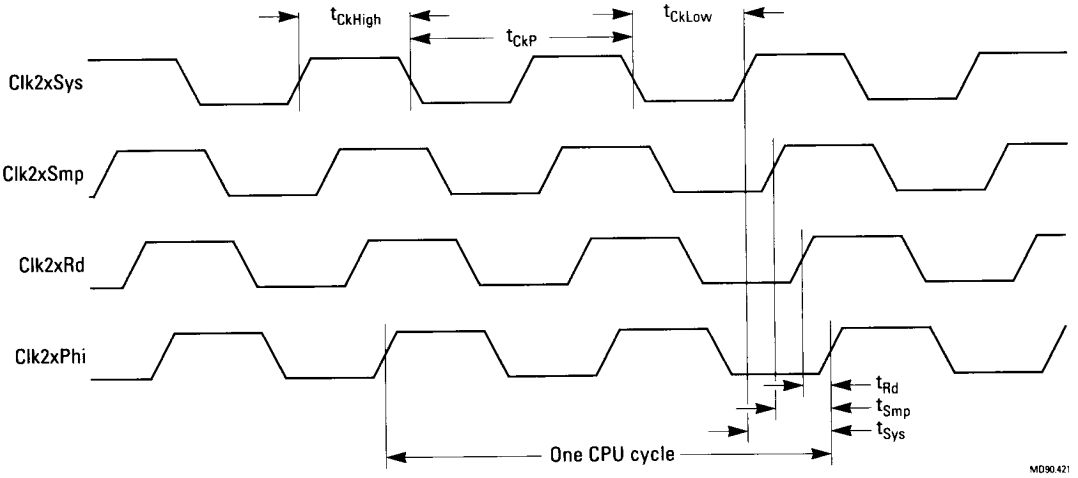



**Figure 9. Clock Timing**

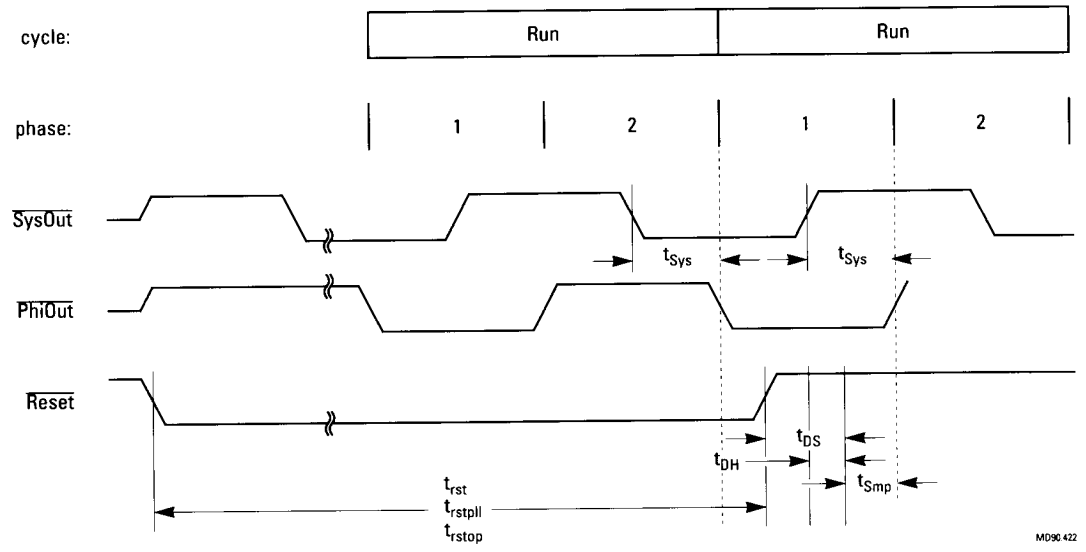



**Figure 10. Reset Timing**

**Timing Waveforms**
(Continued)

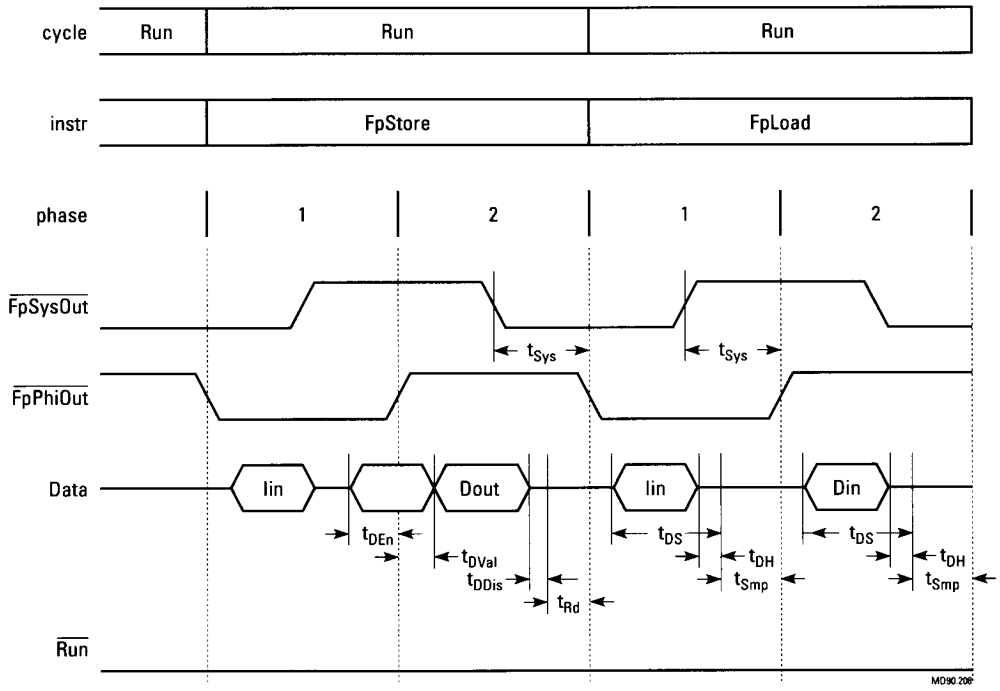



**Figure 11. Load Store Timing**

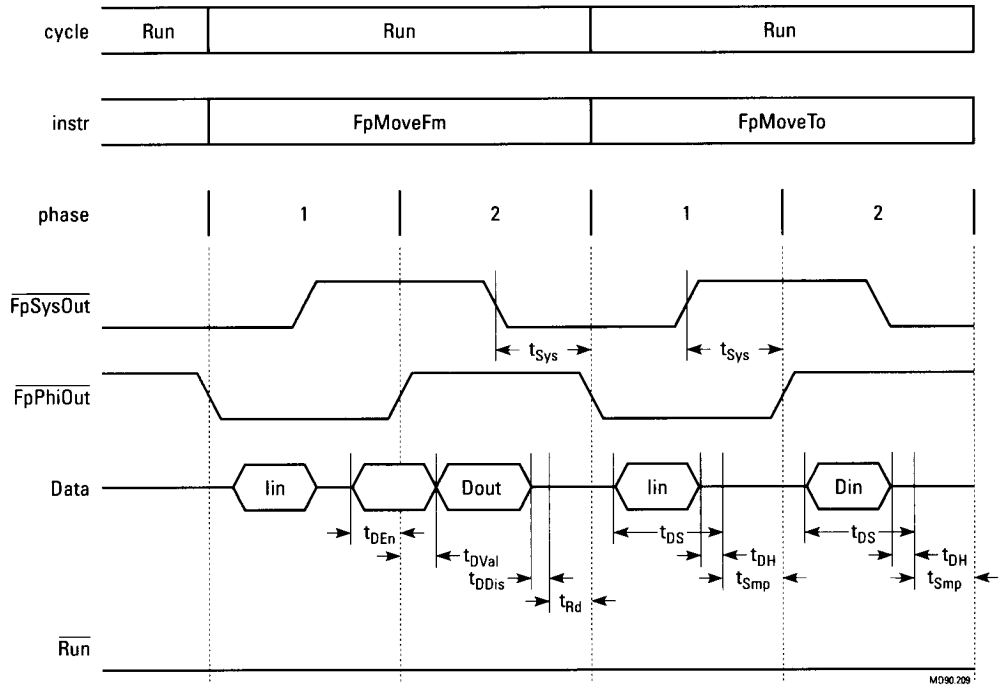



**Figure 12. FPA Transfer Timing**

**Timing Waveforms**
(Continued)

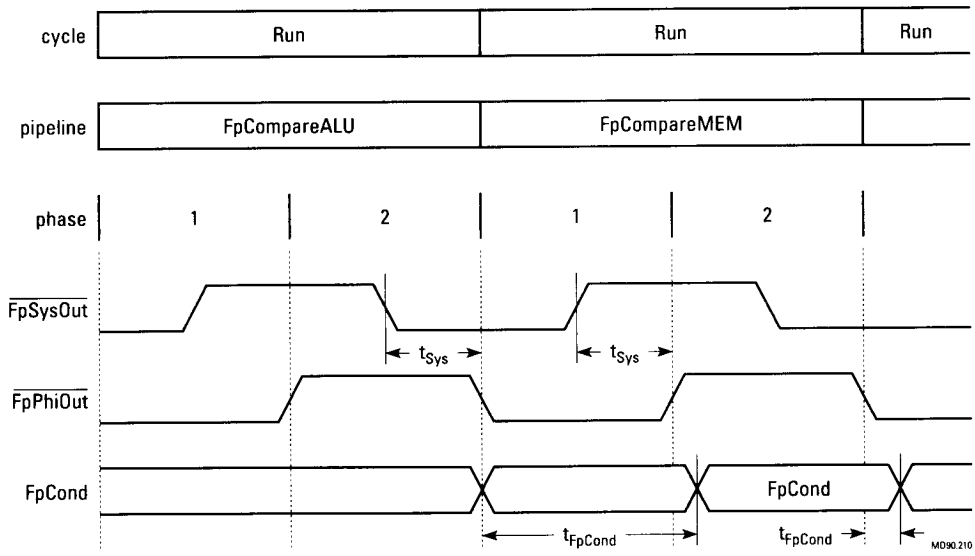



**Figure 13. FpCond Timing**

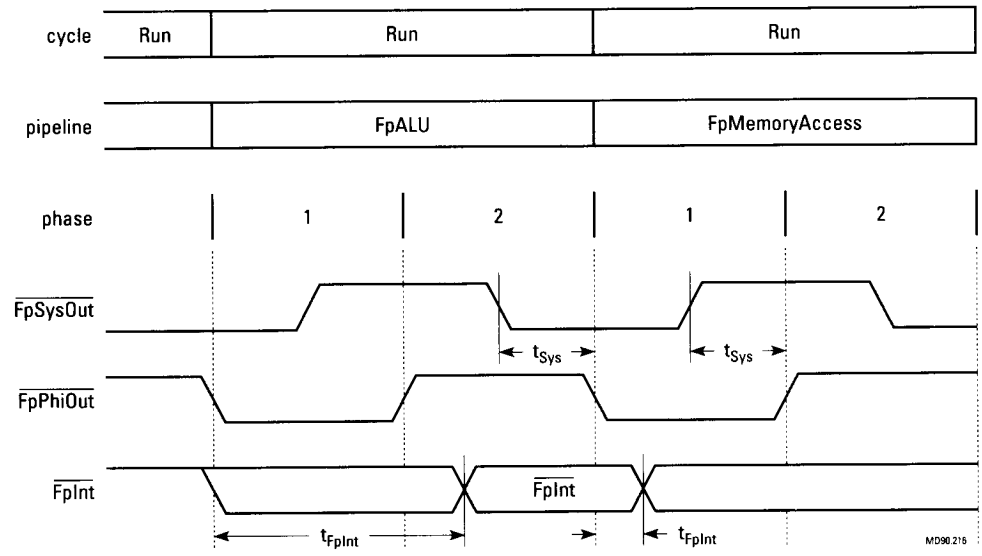



**Figure 14. Interrupt Timing**

**Timing Waveforms**
(Continued)

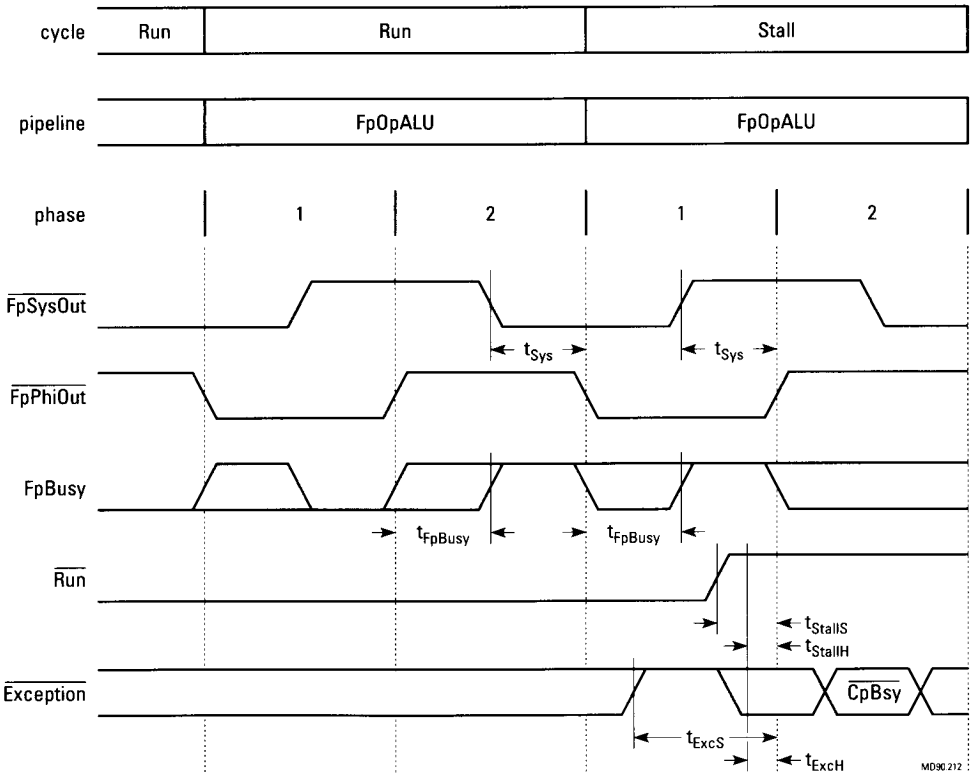



**Figure 15. FPA Busy Timing – Beginning of Stall**

**Timing Waveforms**
(Continued)

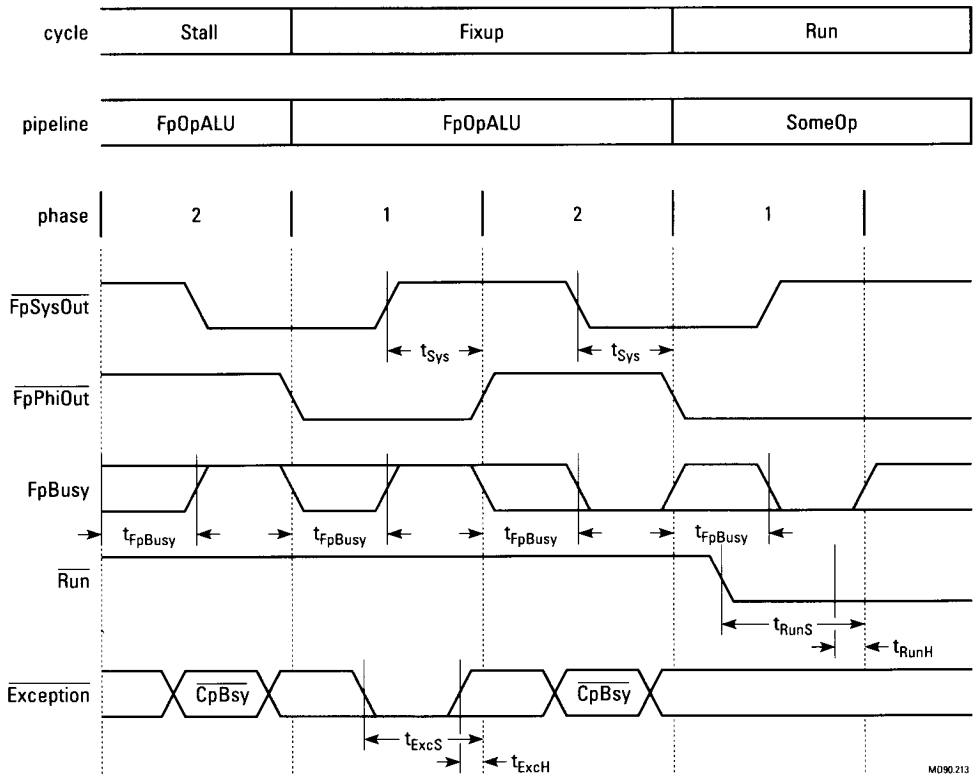



**Figure 16. FPA Busy Timing – End of Stall**

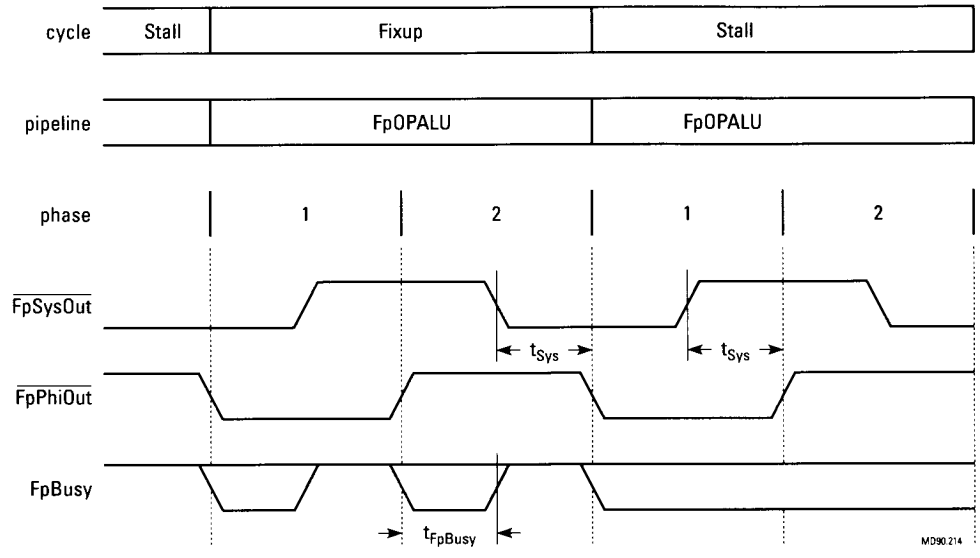



**Figure 17. FPA Busy Retry Timing**

**Timing Waveforms**
(Continued)

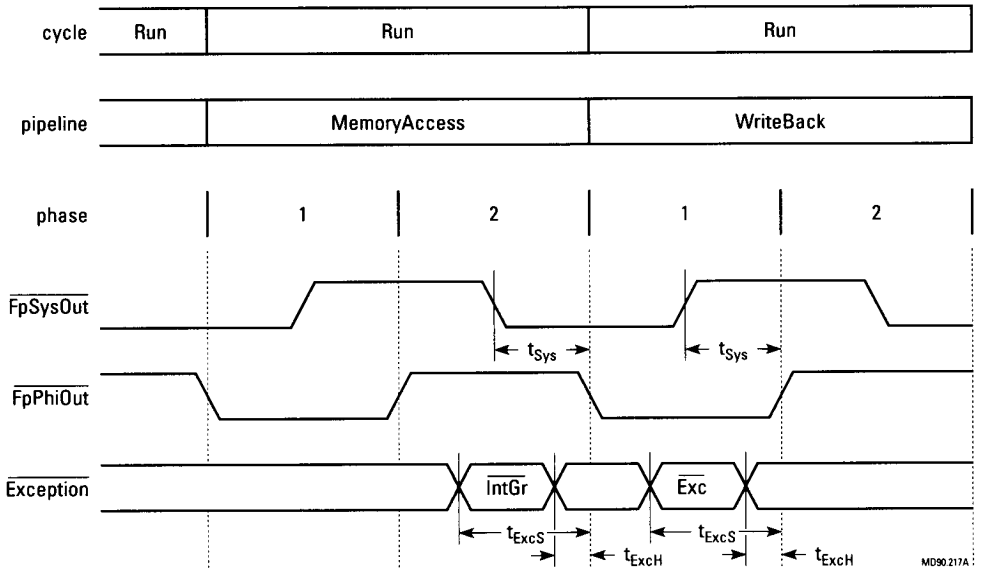



**Figure 18. FPA Exception Timing**

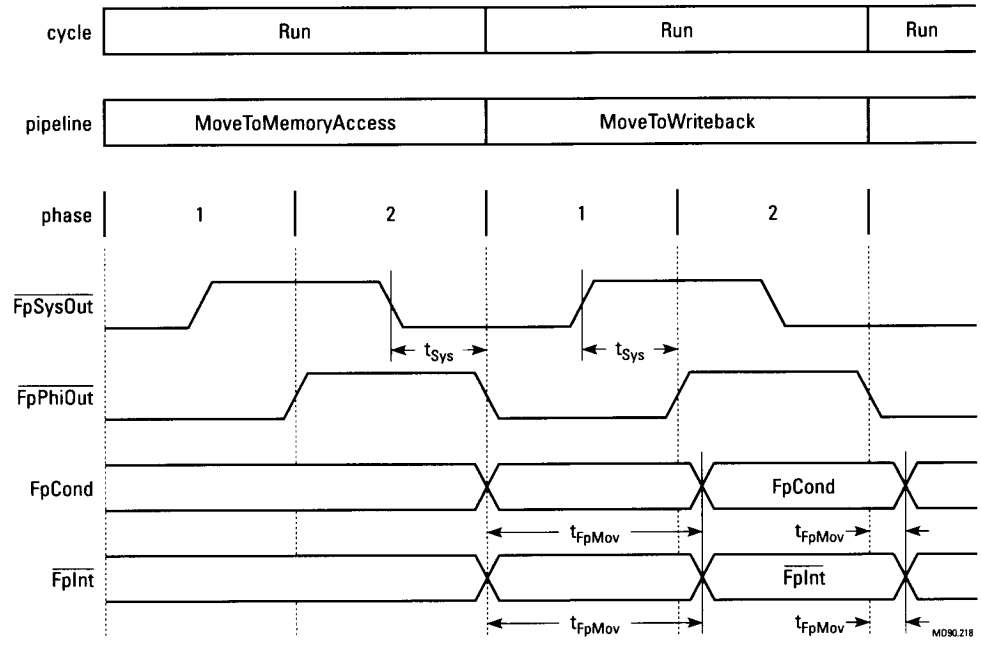



**Figure 19. Move to FPA Status Timing**

## Pinout, Package and Ordering Information

The LR3010 and LR3010A are available in three package options. The 84-pin cavity-down ceramic, pin-grid array (CPGA) is illustrated in Figure 20 with mechanical configuration shown in Figure 21. The 84-pin cavity-down ceramic leaded chip carrier (CLDCC) is shown in Figure 22 with mechanical configuration illustrated in Figure 23. The 160-pin plastic quad flat pack (PQFP) is shown in Figure 24 with mechanical configuration illustrated in Figure 25.

Table 21 shows the clock frequency, package types and order codes for the LR3010. Table 22 shows comparable information for the LR3010A.

**Table 21. LR3010 Ordering Information**

| Product Code | Clock (MHz) | Package | Type |
|---|---|---|---|
| LR3010HM-16 | 16.67 | 84-pin CPGA | Military |
| LR3010KM-16 | 16.67 | 84-pin CLDCC | Military |
| LR3010HM-20 | 20 | 84-pin CPGA | Military |
| LR3010KM-20 | 20 | 84-pin CLDCC | Military |
| LR3010HC-16 | 16.67 | 84-pin CPGA | Commercial |
| LR3010KC-16 | 16.67 | 84-pin CLDCC | Commercial |
| LR3010HC-20 | 20 | 84-pin CPGA | Commercial |
| LR3010KC-20 | 20 | 84-pin CLDCC | Commercial |
| LR3010HC-25 | 25 | 84-pin CPGA | Commercial |
| LR3010KC-25 | 25 | 84-pin CLDCC | Commercial |

**Table 22. LR3010A Ordering Information**

| Product Code | Clock (MHz) | Package | Type |
|---|---|---|---|
| LR3010AHC-16 | 16.67 | 84-pin CPGA | Commercial |
| LR3010AKC-16 | 16.67 | 84-pin CLDCC | Commercial |
| LR3010AHC-25 | 25 | 84-pin CPGA | Commercial |
| LR3010AKC-25 | 25 | 84-pin CLDCC | Commercial |
| LR3010AHC-33 | 33.33 | 84-pin CPGA | Commercial |
| LR3010AKC-33 | 33.33 | 84-pin CLDCC | Commercial |
| LR3010AQC-16 | 16.67 | 160-pin PQFP | Commercial |

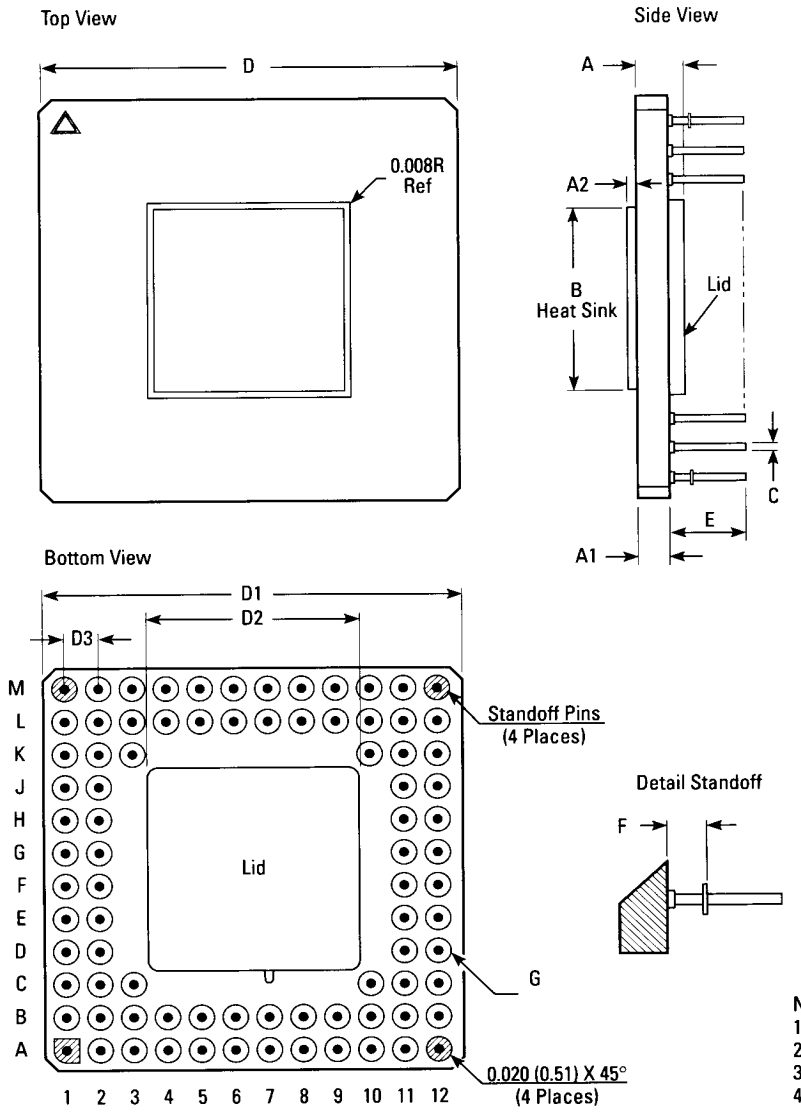
**Pinout, Package and Ordering Information** (Continued)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | GND | VCC | $\overline{\text{FpSysOut}}$ | GND | Clk2xSmp | GND | $\overline{\text{Reset}}$ | GND | $\overline{\text{FpSync}}$ | Data0 | VCC | GND |
| B | $\overline{\text{FpSysIn}}$ | Data31 | DataP3 | VCC | Clk2xSys | VCC | Clk2xPhi | VCC | $\overline{\text{PllOn}}$ | Data1 | Data3 | Data4 |
| C | GND | VCC | Clk2xRd | | | | | | | Data2 | VCC | GND |
| D | Data29 | Data30 | | | | | | | | | Data5 | Data6 |
| E | Data27 | Data28 | | | | | | | | | Data7 | DataP0 |
| F | GND | VCC | | | | | | | | | Data8 | Data9 |
| G | Data26 | DataP2 | | | | Top View | | | | | VCC | GND |
| H | Data24 | Data25 | | | | | | | | | Data11 | Data10 |
| J | Data23 | Data22 | | | | | | | | | Data13 | Data12 |
| K | GND | VCC | Data19 | | | | | | | Resvd0 | VCC | GND |
| L | Data21 | Data20 | Data18 | Data16 | VCC | FpBusy | $\overline{\text{Exception}}$ | VCC | Resvd2 | $\overline{\text{FpPresent}}$ | Data15 | Data14 |
| M | GND | VCC | Data17 | DataP1 | GND | FpCond | $\overline{\text{FpInt}}$ | GND | $\overline{\text{Run}}$ | Resvd1 | VCC | GND |

MD90 229

**Figure 20. 84-Pin Cavity-Down CPGA – Top View**

**Pinout, Package and Ordering Information**
(Continued)




| Dimension | | Inches (mm) |
|---|---|---|
| A | Min | 0.109 (2.77) |
| A1 | Min<br>Max | 0.066 (1.67)<br>0.084 (2.13) |
| A2 | Min<br>Max | 0.015 (0.38)<br>0.025 (0.64) |
| B[1] | Min<br>Max | 0.555 Sq (14.09)<br>0.565 Sq (14.35) |
| B[2] | Min<br>Max | 0.435 Sq (11.05)<br>0.455 Sq (11.55) |
| C | Min<br>Max | 0.016 (0.41)<br>0.020 (0.51) |
| D | Min<br>Max | 1.190 Sq (30.23)<br>1.215 Sq (30.88) |
| D1 | Min<br>Max | 1.09 Sq (27.81)<br>1.11 Sq (28.07) |
| D2 | Max | 0.620 Sq (15.75) |
| D3 | Min<br>Max | 0.095 (2.41)<br>0.105 (2.67) |
| E | Min<br>Max | 0.170 (4.32)<br>0.190 (4.82) |
| F | Min<br>Max | 0.040 (1.02)<br>0.060 (1.32) |
| G | Min<br>Max | 0.060 (1.42)<br>0.065 (1.78) |

Notes:
1. Dimensions apply to LR3010.
2. Dimensions apply to LR3010A.
3. Controlling dimension – inch (mm).
4. Drawing is not to scale. MD90.70C

**Figure 21. 84-Pin Cavity-Down CPGA – Mechanical Drawing**

**Pinout, Package and Ordering Information**
(Continued)

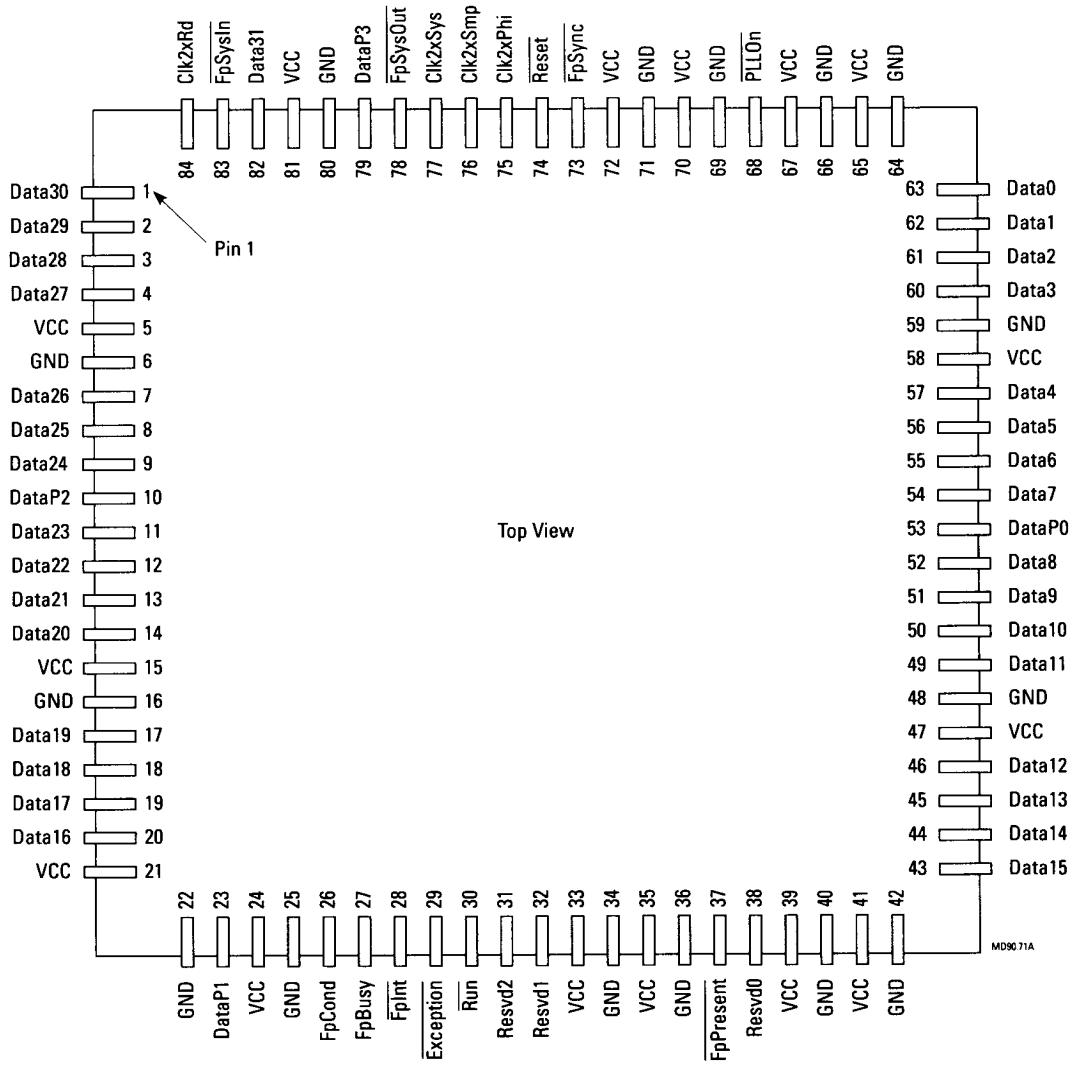
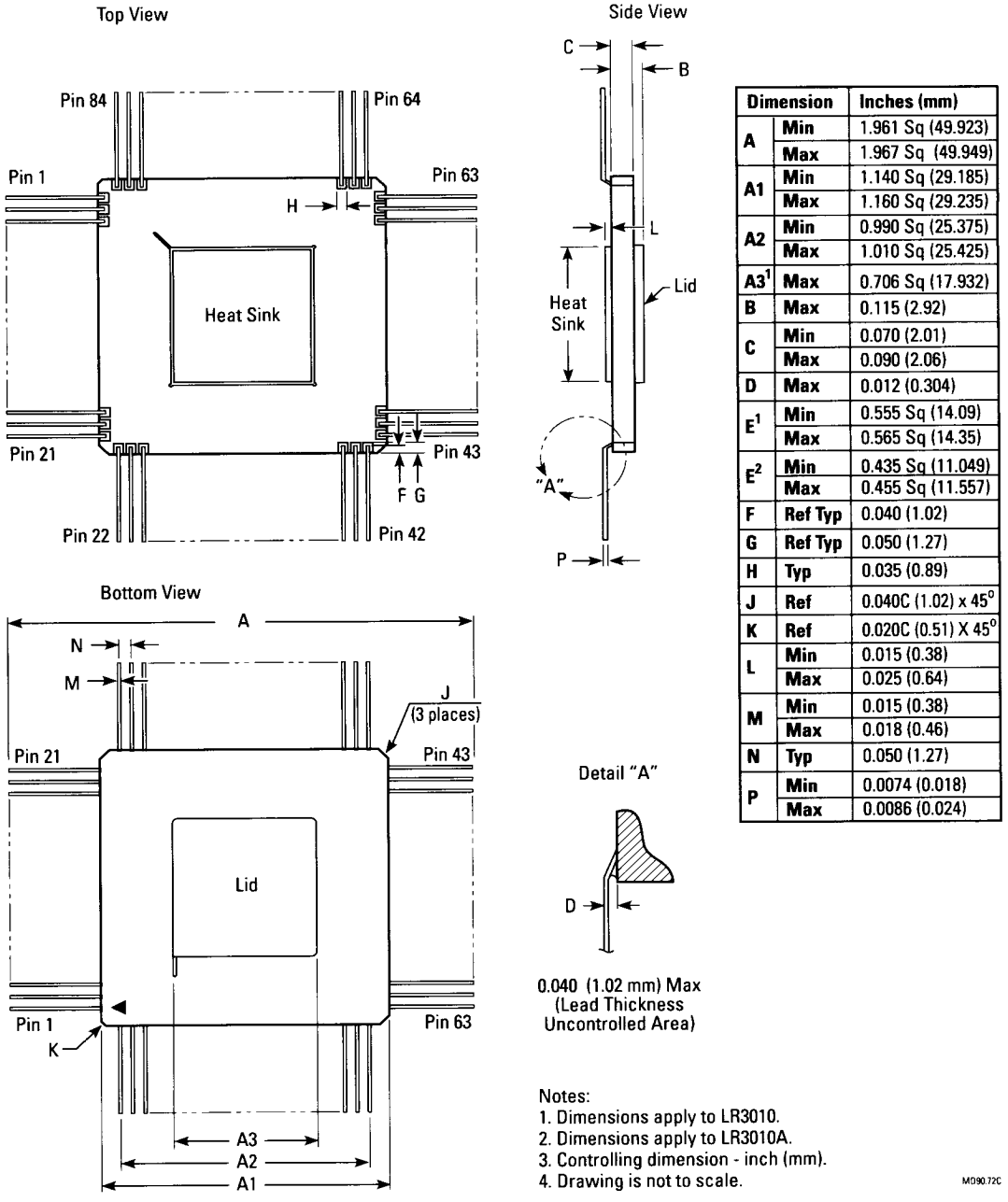



**Figure 22. 84-Pin Cavity-Down CLDCC – Top View**

**Pinout, Package and Ordering Information**
(Continued)



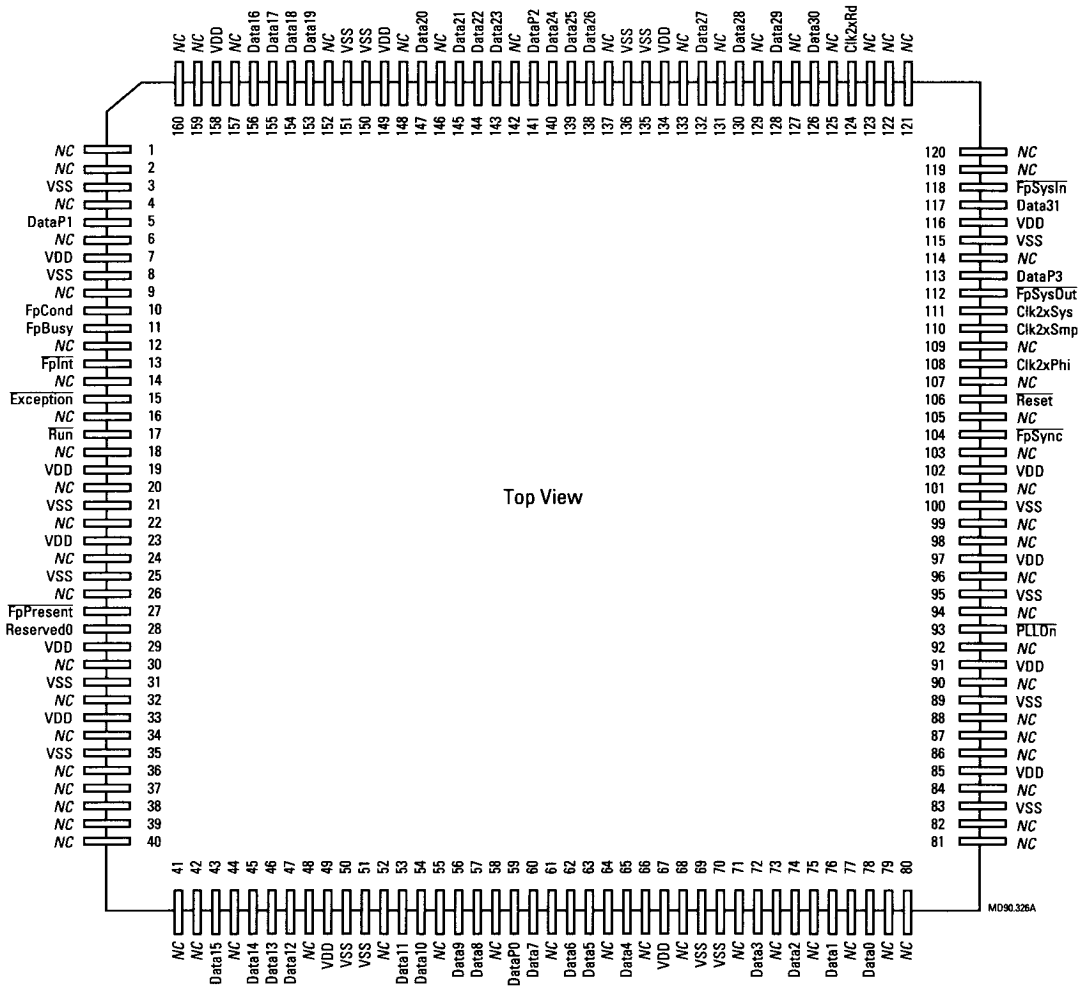

| Dimension | | Inches (mm) |
|---|---|---|
| A | Min | 1.961 Sq (49.923) |
| | Max | 1.967 Sq (49.949) |
| A1 | Min | 1.140 Sq (29.185) |
| | Max | 1.160 Sq (29.235) |
| A2 | Min | 0.990 Sq (25.375) |
| | Max | 1.010 Sq (25.425) |
| A3[1] | Max | 0.706 Sq (17.932) |
| B | Max | 0.115 (2.92) |
| C | Min | 0.070 (2.01) |
| | Max | 0.090 (2.06) |
| D | Max | 0.012 (0.304) |
| E[1] | Min | 0.555 Sq (14.09) |
| | Max | 0.565 Sq (14.35) |
| E[2] | Min | 0.435 Sq (11.049) |
| | Max | 0.455 Sq (11.557) |
| F | Ref Typ | 0.040 (1.02) |
| G | Ref Typ | 0.050 (1.27) |
| H | Typ | 0.035 (0.89) |
| J | Ref | 0.040C (1.02) x 45° |
| K | Ref | 0.020C (0.51) X 45° |
| L | Min | 0.015 (0.38) |
| | Max | 0.025 (0.64) |
| M | Min | 0.015 (0.38) |
| | Max | 0.018 (0.46) |
| N | Typ | 0.050 (1.27) |
| P | Min | 0.0074 (0.018) |
| | Max | 0.0086 (0.024) |

Notes:
1. Dimensions apply to LR3010.
2. Dimensions apply to LR3010A.
3. Controlling dimension - inch (mm).
4. Drawing is not to scale.

M090.72C

**Figure 23. 84-Pin Cavity-Down CLDCC – Mechanical Drawing**

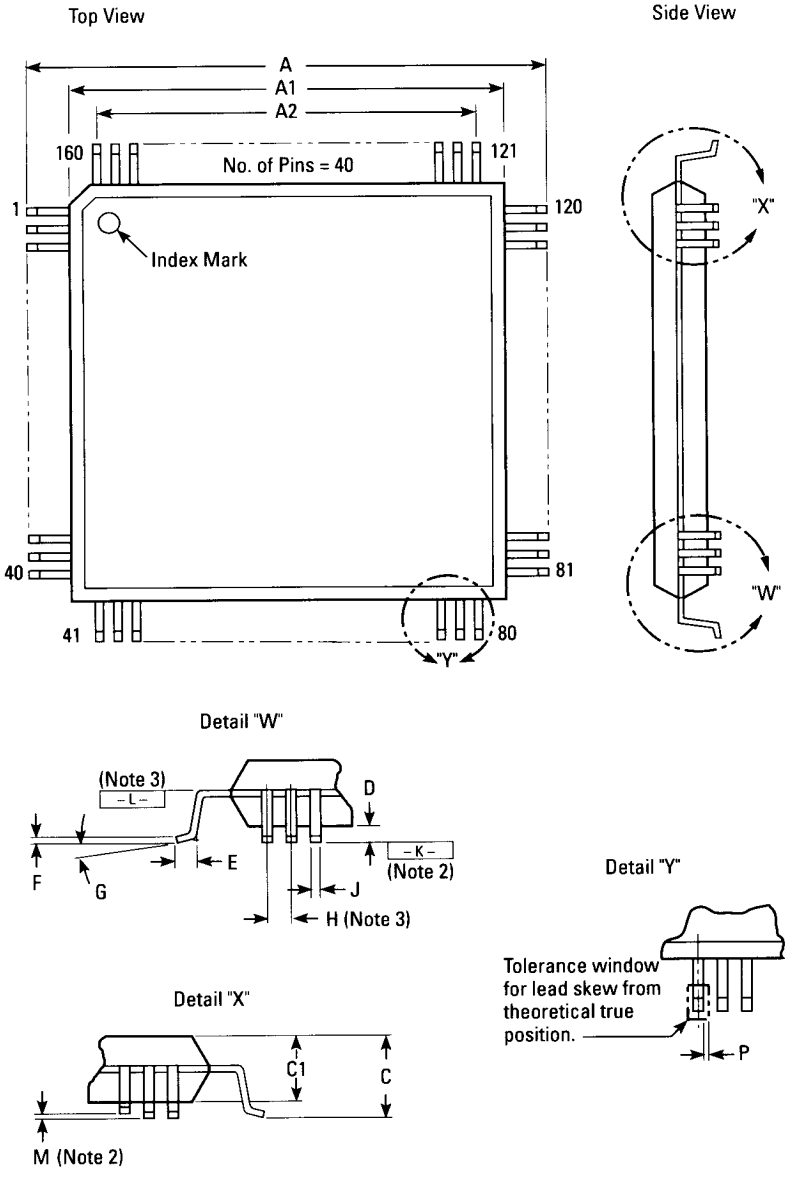
**Pinout, Package and Ordering Information**
(Continued)




Note:
1. *NC* pins are not connected.

**Figure 24. 160-Pin PQFP – Top View**

**Pinout, Package and Ordering Information**
(Continued)




| Dimension | | Inches (mm) |
|---|---|---|
| A | Min | 1.244 Sq (31.60) |
| | Max | 1.276 Sq (32.40) |
| A1 | Min | 1.098 Sq (27.90) |
| | Max | 1.106 Sq (28.10) |
| A2 | Ref | 0.998 Sq (25.35) |
| C | Max | 0.155 (3.94) |
| C1 | Max | 0.140 (3.55) |
| D | Max | 0.012 (0.30) |
| E | Min | 0.024 (0.061) |
| | Max | 0.039 (1.00) |
| F | Min | 0.004 (0.10) |
| | Max | 0.010 (0.25) |
| G | Min | 0 degrees |
| | Max | 10 degrees |
| H | Nom | 0.026 (0.65) |
| J | Min | 0.010 (0.25) |
| | Max | 0.014 (0.35) |
| M | Max | 0.004 (0.10) |
| P | Max | 0.002 (0.05) |

Notes:
1. Controlling dimension – inch, rounded to the nearest .001".
2. Coplanarity of all leads shall be within .004" (difference between the highest and lowest lead with seating plane –K– as reference).
3. Lead pitch determined at –L–.
4. Drawing is not to scale.

MD90 13C

**Figure 25. 160-Pin PQFP – Mechanical Drawing**

## Sales Offices and Design Resource Centers

**LSI Logic Corporation**
**Headquarters**
**Milpitas CA**
■ **408.433.8000**

**Alabama**
■ 205.883.3527

**Arizona**
602.951.4560

**California**
San Jose
■ 408.954.1561

Irvine
■ 714.553.5600

San Diego
619.689.7140

Encino
■ 818.379.2400

**Colorado**
303.756.8800

**Florida**
Altamonte Springs
407.339.2242

Boca Raton
■ 407.395.6200

**Illinois**
■ 708.773.0111

**Maryland**
Bethesda
■ 301.897.5800

Columbia
301.740.5664

**Massachusetts**
■ 617.890.0180 (Design Ctr)
617.890.0161 (Sales Ofc)

**Minnesota**
■ 612.921.8300

**New Jersey**
■ 201.549.4500

**New York**
Hopewell Junction
914.226.1620

Victor
716.223.8820

Camillus
315.468.1646

**North Carolina**
■ 919.872.8400

**Ohio**
**513.427.5476**

**Oregon**
503.645.9882

**Pennsylvania**
215.638.3010

**Texas**
Austin
512.892.7276

Dallas
■ 214.788.2966

**Washington**
■ 206.822.4384

**LSI Logic Corporation**
**of Canada Inc.**
**Headquarters**
Calgary
■ 403.262.9292

Edmonton
■ 403.450.4400

Ottawa
■ 613.592.1263

Montreal
■ 514.694.2417

Toronto
■ 416.620.7400

Vancouver
■ 604.433.5705

**France**
**LSI Logic S.A.**
■ 33.1.46206600

**Israel**
**LSI Logic Limited**
■ 972.3.5403741/4

**Italy**
**LSI Logic SPA**
■ 39.39.6056881

**Japan**
**LSI Logic K.K.**
**Headquarters**
Tokyo
■ 81.3.3589.2711

Tokyo
81.3.5275.1731

Tsukuba-Shi
■ 81.298.52.8371

Osaka
■ 81.6.947.5281

Yokohama
81.45.902.4111

**LSI Logic Corporation**
**of Korea Limited**
■ 82.2.561.2921

**Netherlands**
**LSI Logic/Arcobel**
■ 31.4120.30335

**Scotland**
**LSI Logic Limited**
■ 44.506.416767

**Sweden**
**LSI Logic Export AB**
46.8.703.4680

**Switzerland**
**LSI Logic/Sulzer**
■ 41.32.515441

**Taiwan**
**LSI Logic Corporation**
■ 886.2.755.3433

**United Kingdom**
**LSI Logic Limited**
**Headquarters**
Sidcup
■ 44.81.302.8282

Bracknell
■ 44.344.426544

**West Germany**
**LSI Logic GMBH**
**Headquarters**
Munich
■ 49.89.926903.0

Dusseldorf
■ 49.211.5961066

Stuttgart
■ 49.711.2262151

**AE Advanced Electronics**
Hannover
■ 49.511.3681756

**AE Advanced Electronics**
Munich
■ 49.89.93009855

■ Sales Offices with Design Resource Centers

This document is preliminary. As such, it contains data derived from functional simulations and performance estimates. LSI Logic has not verified the functional descriptions or electrical and mechanical specifications using production parts.



Printed in USA
031.7111.553800.026.LSI.6K.J





023722 ✓