JUL 27 1992

LSI LOGIC

LR33050 MIPS IFX Processor Preliminary

Introduction

The LR33050 MIPS IFX (Integer and Floating-Point Accelerator) Processor is part of LSI Logic's LR33000 MIPS Self-Embedding™ processor family. The LR33050 is a 32-bit Reduced Instruction Set Computer (RISC) processor that has been optimized for use in high-performance embedded applications. The LR33050 combines all the features of the LR33000 with a Floating-Point (FP) core. It is compatible with the MIPS-1 Instruction Set Architecture (ISA). The FP core implements the IEEE Standard for binary floating-point arithmetic. Because the LR33050 combines MIPS-compatible CPU and FP cores and a high level of integration, it is well suited for embedded applications and can execute up to 35 MIPS at 40 MHz.

The LR33050 maintains the same pinout as the LR33000 to provide an easy upgrade path, and includes several features that greatly increase performance, reduce system component count and ease the overall system design task.

The MIPS-1 architecture incorporates traditional RISC elements, such as instruction pipelining, a register-to-register instruction set and a large, general-purpose register file. Separate on-chip data and instruction caches allow the LR33050 to access data and fetch an instruction in a single processor cycle. The LR33050's greater efficiency allows it to provide higher levels of performance than other RISC machines, even those that run at higher clock rates.

In addition to its cache memories, the LR33050's on-chip support features include a DRAM controller, three timer/counters, enhanced debug support and a simple, high-performance I/O interface – all features that are commonly required in embedded applications.

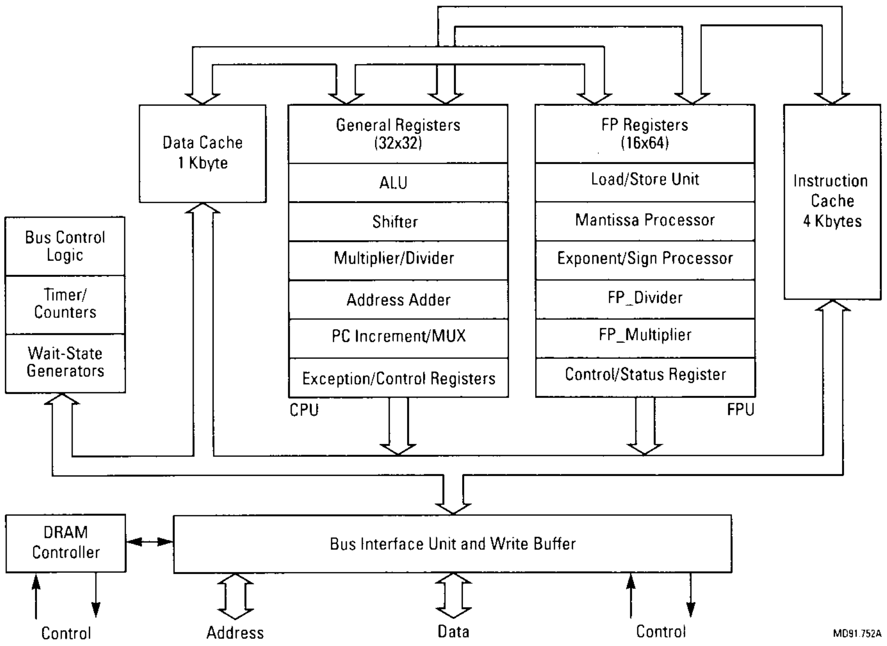


Figure 1. LR33050 Processor Block Diagram

LSI LOGIC

LR33050 MIPS IFX Processor
Preliminary

Features

- RISC architecture
  - MIPS-1 instruction set
  - Single-cycle instruction execution
  - R2000, R3000/A and LR33000 binary compatibility for user software
- Implements the IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985
- Supports single- and double-precision data formats
- On-chip floating-point unit
  - Complete R3010/A instruction set compatibility
  - Single- and double-precision add, subtract, multiply, divide, negate, absolute value
  - Conversion to/from all supported formats
  - Comparison instruction
- Provides 64-bit wide internal data path for all floating-point operations
- High on-chip integration
  - 4-Kbyte instruction cache
  - 1-Kbyte data cache
  - Three timer/counters
  - DRAM controller
  - Four-word-deep write buffer
- Enhanced debug capability
  - Hardware breakpoint registers
  - Instruction trace capabilities
- Simple I/O interface
  - Direct interface to industry-standard DRAMs
  - DRAM burst-mode writes with configurable page size
  - 8- and 32-bit wide boot PROM support
  - Direct interface to other memories and peripherals
  - Direct memory access (DMA) support
  - Programmable wait-state generators
- Single 1X clock input
- High-performance implementation
  - Five-stage integer and floating-point pipelines
  - Precise, efficient handling of pipeline stalls and exceptions
  - Both caches accessible in a single CPU cycle
- Strong integrated software support includes high-performance optimizing compilers for C, Pascal, FORTRAN, Ada, COBOL and PL/1
- Complete development tool support available from the FasTrack33K program
- Low-power static CMOS design
- Available in 25-, 33- and 40-MHz versions
- Available in two packages
  - 160-pin metal quad flat package (MQUAD)
  - 155-pin ceramic pin grid array (CPGA)

Block Diagram

The LR33050 provides several features that especially suit it for embedded applications. Figure 1 is a block diagram of the processor; descriptions of the blocks follow.

CPU

The CPU implements the high-performance MIPS architecture. The CPU performs Integer Unit (IU) functions.

Instruction Set Compatibility – The LR33050 maintains full binary compatibility with the R2000, R3000/A and LR33000 microprocessors.

Efficient Pipelining – The CPU's five-stage pipeline design assists in obtaining an execution rate approaching one instruction per cycle. Pipeline stalls and exceptional events are handled precisely and efficiently.

Full 32-Bit Operation – The CPU contains thirty-two 32-bit general-purpose data registers. All instructions and addresses are 32 bits wide to provide a 4-Gbyte address space.

Enhanced Debug Features – The CPU incorporates program trace logic and hardware breakpoint from the program counter and data address registers to ease software debugging.

FPU

The FPU is fully compatible with the ANSI/IEEE Standard 754-1985 for binary floating-point arithmetic.

Instruction Set Compatibility – The LR33050 maintains full binary compatibility with the R2010 and R3010/A microprocessors.

Full 64-Bit Operation – The FPU contains sixteen, 64-bit registers that can each be used to hold a single-precision or double-precision value. The FPU also includes a 32-bit control/status register that provides access to all IEEE-Standard exception-handling capabilities.

Load/Store Instruction Set – The FPU uses a load/store-oriented instruction set, with single-cycle loads and stores. Floating-point opera-

Block Diagram (Continued)

tions are started in a single cycle and their execution is overlapped with other fixed-point or floating-point operations.

Seamless Integration to the CPU Pipeline – The FPU uses a five-stage pipeline identical to that of the CPU and runs in lock step with the CPU. Instruction execution, data transfers and exceptions are handled synchronously with the CPU. Because each unit receives and executes instructions in parallel, some floating-point instructions can execute at the same single-cycle per instruction rate as fixed-point instructions.

Deep Pipeline of Computational Unit – The computational unit in the FPU is pipelined to allow a high throughput of operations. Up to six floating-point instructions can be executing concurrently in the computational unit of the FPU, as shown in Figure 13 on page 14.

On-Chip Cache Memory – The LR33050 contains a 4-Kbyte instruction cache and a 1-Kbyte data cache. The CPU can access both caches in a single clock cycle – which allows the CPU to execute one instruction per cycle when executing from cache memory. Both caches are direct-mapped and support bus snooping to maintain cache coherency. The data cache is write-through.

On-Chip DRAM Controller – The LR33050 has an integral DRAM Controller that supports most popular DRAMs. The DRAM Controller features include complete support for page-mode DRAMs, $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh and DMA by separate I/O devices. The DRAM Controller also has burst-write capabilities with a programmable page size.

Bus Interface Unit (BIU) – The LR33050's BIU provides a simple memory interface that easily connects to I/O devices, supports both 8- and 32-bit PROMs and includes programmable wait-state generators. The BIU supports optional parity generation and parity checking. Parity checking may be suspended on a per-memory-access basis and the LR33050 provides a dedicated output to indicate parity errors to external logic. The BIU incorporates a four-word-deep write buffer allowing consecutive stores to be queued without stalling the CPU.

On-Chip Timer/Counters – The LR33050 includes three timer/counters: two 24-bit general-purpose timer and a 12-bit Refresh Timer, which supports the on-chip DRAM Controller and can also support an off-chip DRAM Controller.

MIPS CPU Core

The following subsections describe the essential features of the MIPS CPU Core, including its registers, instruction set, system control processor (CP0) and operating modes.

CPU Registers

The LR33050 CPU provides 32 general-purpose, 32-bit registers, a 32-bit Program Counter and two 32-bit registers (HI and LO). The HI and LO registers hold the results of integer multiply and divide operations. The CPU registers are shown in Figure 2. Notice that the figure does not contain a Program Status Word (PSW) register; the functions traditionally provided by a PSW register are instead provided by the Status and Cause Registers incorporated within the system control coprocessor (CP0).

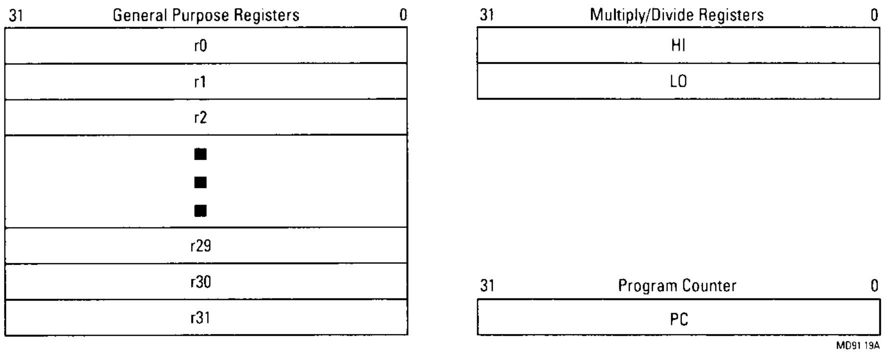


Figure 2. CPU Registers

MIPS CPU Core
(Continued)

Instruction Set Overview

All CPU instructions are 32 bits long. As shown in Figure 3, instructions have three basic formats: I-type (immediate), J-type (jump) and R-type (register). Instruction decoding is simplified by this approach. More complicated (and less frequently used) operations and addressing modes can be synthesized by the compiler using sequences of simple RISC-type instructions. The assembler recognizes some CISC-type instructions for programming ease, but translates them into sequences of simple instructions.

The CPU instruction set is divided into the following groups:

- **Load/Store** instructions are the only instructions that move data between memory and general registers. These instructions are I-type, because the only addressing mode supported is base register plus 16-bit, signed immediate offset.
- **Computational** instructions perform arithmetic, logical and shift operations on values in registers. These instructions are both R-type (both operands and the result are registers) and I-type (one operand is a 16-bit immediate).
- **Jump** and **Branch** instructions change the control flow of a program. Jumps are always to a paged, absolute address formed by combining a 26-bit target with four bits of the program counter (J-type format, for subroutine calls), or 32-bit register byte addresses (R-type, for returns and dispatches). Branches have 16-bit offsets relative to the program counter (I-type). Jump and Link instructions save a return address in register r31.
- **Coprocessor 0** instructions perform operations on the system control coprocessor (CP0) registers to manipulate the exception-handling facilities of the processor.
- **Coprocessor 1** instructions perform the floating-point operations.
- **Coprocessor 2 and 3** instructions perform operations on internal coprocessors that are not yet implemented by the LR33050. See the section entitled "Compatibility with the R2000, R3000 and LR33000 Microprocessors" on page 19 for details.
- **Special** instructions perform a variety of tasks including movement of data between special and general registers, system calls and breakpoint. These instructions are always R-type.

Table 1 summarizes the instruction set of the LR33050 processor. For a detailed description of the instructions, refer to the *LR33050 MIPS IFX Processor User's Manual.*

System Control Coprocessor (CP0)

The LR33050's system control coprocessor (CP0) supports exception-handling functions of the LR33050. Figure 4 summarizes the register set.

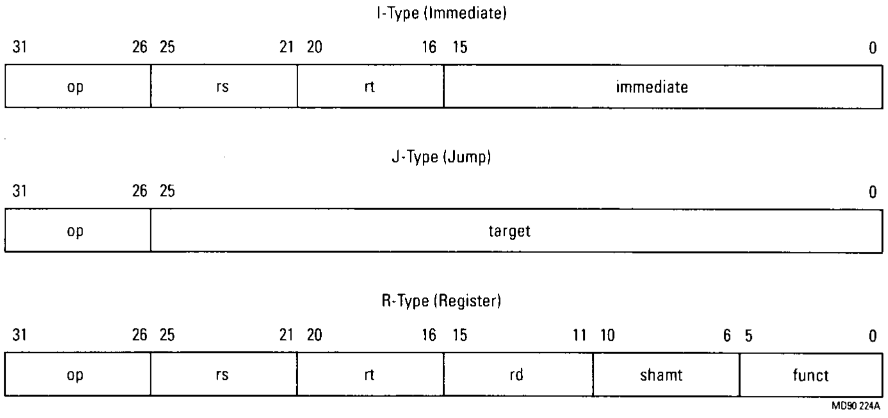


Figure 3. CPU Instruction Formats

MIPS CPU Core
(Continued)

Table 1. LR33050 CPU Instruction Summary

Op	Description
Load/Store Instructions	
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
SB	Store Byte
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right
Arithmetic Instructions: ALU Immediate	
ADDI	Add Immediate
ADDIU	Add Immediate Unsigned
SLTI	Set on Less Than Immediate
SLTIU	Set on Less Than Immediate Unsigned
ANDI	AND Immediate
ORI	OR Immediate
XORI	Exclusive OR Immediate
LUI	Load Upper Immediate
Arithmetic Instructions: 3-Operand, Register-Type	
ADD	Add
ADDU	Add Unsigned
SUB	Subtract
SUBU	Subtract Unsigned
SLT	Set on Less Than
SLTU	Set on Less Than Unsigned
AND	AND
OR	OR
XOR	Exclusive OR
NOR	NOR
Shift Instructions	
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
SLLV	Shift Left Logical Variable
SRLV	Shift Right Logical Variable
SRAV	Shift Right Arithmetic Variable
Multiply/Divide Instructions	
MULT	Multiply
MULTU	Multiply Unsigned
DIV	Divide
DIVU	Divide Unsigned
MFHI	Move From HI
MTHI	Move To HI
MFLO	Move From LO
MTLO	Move To LO
Jump and Branch Instructions	
J	Jump
JAL	Jump And Link
JR	Jump Register
JALR	Jump And Link Register
BEQ	Branch on Equal
BNE	Branch on Not Equal
BLEZ	Branch on Less than or Equal to Zero
BGTZ	Branch on Greater Than Zero
BLTZ	Branch on Less Than Zero
BGEZ	Branch on Greater than or Equal to Zero
BLTZAL	Branch on Less Than Zero And Link
BGEZAL	Branch on Greater than or Equal to Zero And Link
Special Instructions	
SYSCALL	System Call
BREAK	Breakpoint
Coprocessor Instructions	
BCzT	Branch on Coprocessor z True
BCzF	Branch on Coprocessor z False
System Control Coprocessor (CP0) Instructions	
MTC0	Move To CP0
MFC0	Move From CP0
RFE	Restore From Exception

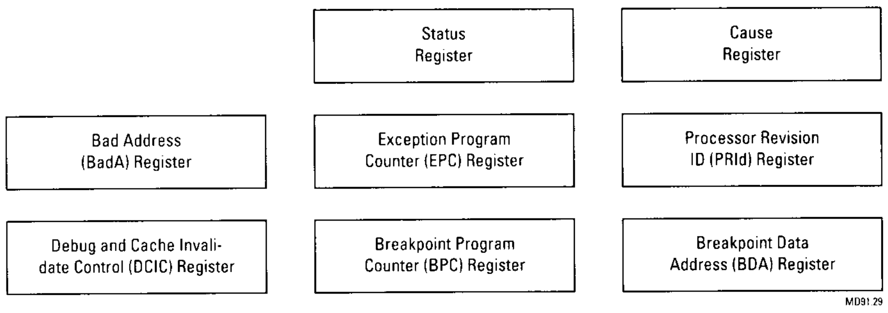

Status Register
Cause Register
Bad Address (BadA) Register
Exception Program Counter (EPC) Register
Processor Revision ID (PRId) Register
Debug and Cache Invalidate Control (DCIC) Register
Breakpoint Program Counter (BPC) Register
Breakpoint Data Address (BDA) Register
MD91.29

Figure 4. The CP0 Exception-Handling Registers

MIPS CPU Core
(Continued)

Operating Modes

To facilitate the separation of user and supervisory software, the LR33050 has two operating modes: *user* and *kernel*. Normally, the processor operates in the user mode until an exception is detected, which forces it into the kernel mode. It remains in the kernel mode until a Restore From Exception (RFE) instruction is executed. Figure 5 shows the address space for the two operating modes.

User Mode – In user mode, a single, uniform address space (kuseg) of 2 Gbytes is available.

Kernel Mode – Four separate segments are defined in kernel mode:

- kuseg – References to this 2-Gbyte segment are treated just like user mode references, thus streamlining kernel access to user data.
- kseg0 – References to this 512-Mbyte segment use cache memory and are hard-mapped to the first 512 Mbytes of memory.
- kseg1 – References to this 512-Mbyte segment do not use the cache and are hard-mapped to the same 512-Mbyte segment of memory space as kseg0.
- kseg2 – References to this 1-Gbyte segment are not mapped.

The cacheability of data and instructions in kuseg, kseg0 and kseg2 is controlled on a per access basis by the $\overline{\text{CACHD}}$ signal.

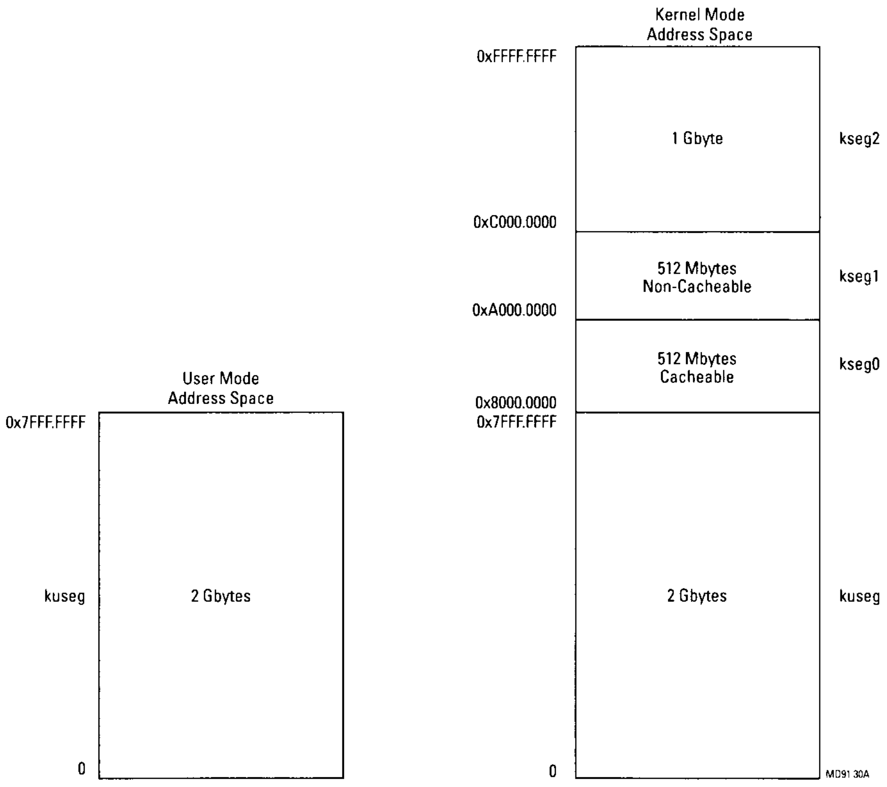


Figure 5. LR33050 Address Space

MIPS FPU Core

The following subsections describe the essential features of the MIPS FPU core, including its programming model, data formats, instruction set, floating-point exceptions and procedures for saving or restoring the register state.

Programming Model
This section describes the three types of FPU registers: Floating-Point General Registers, Floating-Point Registers and Floating-Point Control Registers.

Floating-Point General Registers (FGR) – The 32 FGRs are all 32-bit, directly addressable, physical registers. The FGRs are numbered consecutively from 0 to 31.

Floating-Point Registers (FPR) – The FPRs are logical registers that store data values during floating-point operations. Each of the 16 FPRs is 64 bits wide and can hold floating-point values in single- or double-precision format. As shown in Figure 6, an FPR is created by concatenating two adjacent FGRs. Each FPR is even-numbered. Odd-numbered FPRs are invalid, but are not checked by the hardware. Single-precision floating-point operations – except loads, stores or moves – leave the odd half of the result register undefined.

Floating-Point Control Registers – This subsection defines two floating-point control registers (FCR0 and FCR31) in the Coprocessor 1 control register space. FCR0, which is shown in Figure 7, is dedicated to implementation and revision information. FCR31, which is shown in Figure 8, is dedicated to rounding-mode control, exception handling and state saving.

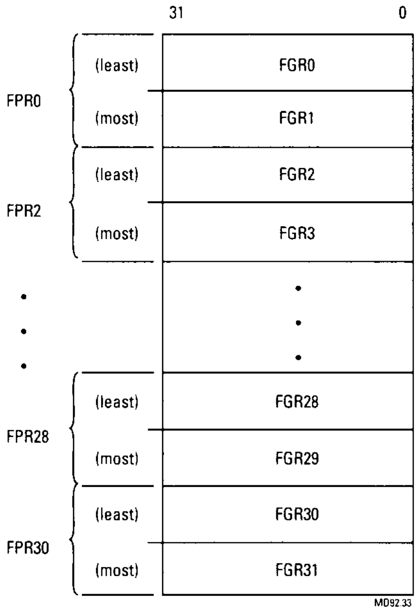


Figure 6. Floating-Point Registers

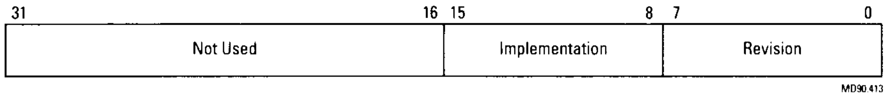


Figure 7. Implementation/Revision Register (FCR0)

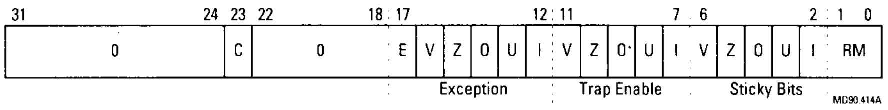


Figure 8. Control/Status Register (FCR31)

MIPS FPU Core
(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/Status Register 31 (FCR31) contains control and status information. It is accessed by instructions running in either kernel or user mode and enables user level traps. FCR31 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. Three fields in the register control or track exception processing. A fourth field, rounding mode, controls arithmetic rounding and determines the default value that the FPU supplies when exceptions are not trapped. Exceptions include invalid operation, inexact exception, divide-by-zero, overflow and unimplemented operation exception.

Reading FCR31 with a CFC1 instruction causes the FPU to complete all previous, uncompleted instructions in its pipeline. If an exception is taken as the pipeline is being emptied, the CFC1 instruction can be executed again after the exception is processed.

Figure 8 shows FCR31 with internal fields. Each single letter mnemonic that identifies an exception in Figure 8 is defined in Table 2.

Table 2. Exception Mnemonics

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

The following paragraphs define the FCR31 fields:

Rounding Mode (RM) – The rounding modes include round to the nearest representable value, round toward zero, round toward +∞ and round toward -∞. The setting of the rounding-mode bits modifies the exception processing defaults as shown in Table 7 on page 15.

Sticky Bits – The FPU sets the sticky bit for an exception when traps are disabled. Unlike the exception-field bits, sticky bits are not cleared as a result of executing floating-point instructions. They are cleared only by writing a new value into the FCR31 register with a CTC1 instruction.

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

Exception – 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 FPU flags an exception when an instruction is in the ALU stage.

Condition Bit (C) – As shown in Figure 8, 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.

Data Formats
The MIPS-1 architecture supports fixed-point format and 32-bit single-precision and 64-bit double-precision IEEE-Standard floating-point formats. Floating-point representations in the MIPS-1 architecture are composed of three fields:

1. 1-bit sign: $s = s$
2. Biased exponent: $e = E+\text{bias}$
3. Fraction $f = 1.b_1b_2...b_{p-1}$ where
   1 = hidden integer bit,
   p = 24 for single-precision format,
   p = 53 for double-precision format

MIPS FPU Core
(Continued)

The unbiased exponent E can be any integer between two values E_{Min} and E_{Max} inclusive, and can also be two other reserved values: 1) E_{Min}-1 to encode ±0 and denormalized numbers, and 2) E_{Max}+1 to encode ±∞ and NaNs (Not-A-Number). Each representable non-zero value has only one encoding for single- and double-precision formats.

For single- and double-precision formats, the values of E and f determine the value of a number, v, as shown below:

- If $E = E_{Max}+1$ and $f \neq 0$, then v is NaN, regardless of s.
- If $E = E_{Max}+1$ and $f = 0$, then $v = (-1)^s \infty$.
- If $E_{Min} \leq E \leq E_{Max}$, then $v = (-1)^s 2^E (1.f)$.
- If $E = E_{Min}-1$ and $f \neq 0$, then $v = (-1)^s 2^{E_{Min}} (0.f)$.
- If $E = E_{Min}-1$ and $f = 0$, then $v = (-1)^s 0$.

The 32-bit format has a 24-bit signed magnitude fraction field and an 8-bit exponent, as shown in Figure 9.

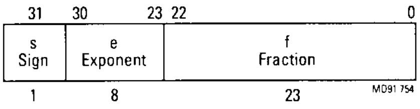


Figure 9. Single-Precision, Floating-Point Format

The 64-bit format has a 53-bit signed magnitude fraction field and an 11-bit exponent, as shown in Figure 10.

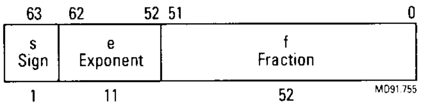


Figure 10. Double-Precision, Floating-Point Format

Instruction Set
FPU instructions are extensions of the basic MIPS CPU instruction set, so every FPU 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 FPU instruction set. The coprocessor unit 1 (COP1) operation instructions are used to implement the floating-point operations supported by the MIPS-1 architecture.

Internally, the FPU is a more efficient engine than the CPU for doing floating-point computations, because it has separate execution units for mathematical operations, and it has extended floating-point registers that hold the operands 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 FPU.

Instruction Format and Function – There are only four FPU 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 FPU 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 FPU comparison has set the condition bit in FPU control register FCR31. M-type instructions move data between CPU and FPU registers. R-type instructions specify both a source and destination register within the FPU where operands and results of a floating-point computation are stored. Instructions that require the R-type format include all

MIPS FPU Core
(Continued)

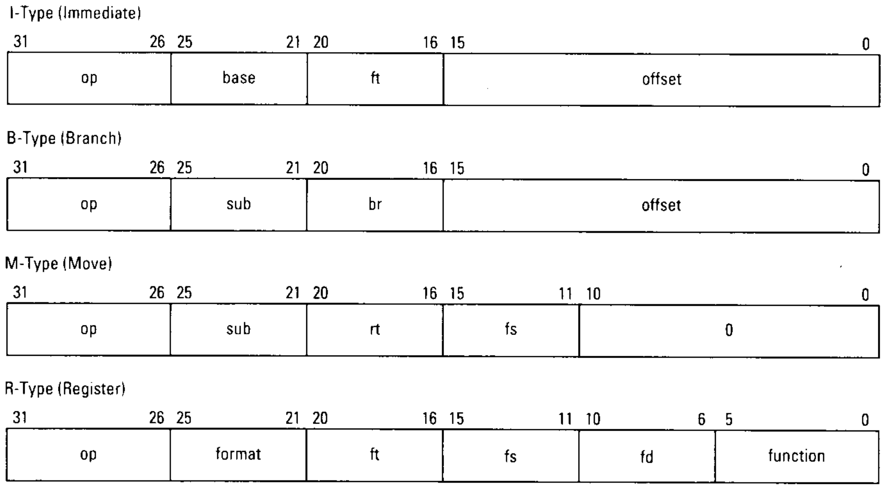
arithmetical computations, conversions and comparisons. FPU instruction formats are illustrated in Figure 11.

FPU Instruction Fields – The *opcode* identifies the FPU as the selected execution unit for the next operation. The *function* field specifies the floating-point operation that will be performed. 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 an CPU register (identified as *base* in Figure 11) 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 FPU loads and stores are identical to those of the main processor. During FPU load and store operations, the CPU calculates memory addresses and controls the cache and memory interfaces. The FPU simply takes data from the bus during load operations and drives the bus during store operations. Data loaded into the FPU 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 LR33050 reads the Data and Parity buses to check parity during a load operation. The FPU generates data parity during FPU 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



Instruction Format Legend

op	is a six-bit operation code
base	is a five-bit CPU base register specifier
ft	is a five-bit FPU source or destination (for loads) register specifier
offset	is a 16-bit immediate or branch displacement
sub	is a five-bit sub-operation code
br	is a five-bit branch code
rt	is a five-bit CPU source or destination general register specifier
fs	is a five-bit FPU source register specifier
0	is undefined if non-zero
fd	is a five-bit FPU destination register specifier
format	is a five-bit data format specifier
function	is a six-bit function code

MD91 756A

Figure 11. FPU Instruction Formats

MIPS FPU Core
(Continued)

results of a comparison or branch condition are available in its control register (FCR31).

Move Instructions – Four move instructions with the M-type format move data between the FPU and CPU. Two instructions, MTC1 and MFC1, move the contents of general registers from the CPU to the FPU or from the FPU to the CPU. The other two instructions, CTC1 and CFC1, move contents from the CPU general registers to the FPU control registers or from FPU control registers to CPU general registers.

Floating-Point Computations – The FPU 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 instructions are related to floating-point computations:

- 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-precision 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 3 shows the comparisons performed.

Table 3 extends the list of 26 predicates named in the IEEE Standard to include six predicates that test a comparison in the FPU 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 descriptive form of the predicate.

Table 3. FPU Predicate Conditions

Mnemonic	Condition	Description
F	False	false
UN	Unordered	?
EQ	Equal	=
UEQ	Unordered or Equal	?=
OLT	Ordered Less Than	NOT(?>=)
ULT	Unordered or Less Than	?<
OLE	Ordered Less Than or Equal	NOT(?>)
ULE	Unordered or Less Than or Equal	?<=
SF	Signaling False	
NGLE	Not Greater Than or Less Than or Equal	NOT(<=>)
SEQ	Signaling Equal	
NGL	Not Greater Than or Less Than	NOT(<>)
LT	Less Than	<
NGE	Not Greater Than or Equal	NOT(>=)
LE	Less Than or Equal	<=
NGT	Not Greater Than	NOT(>)
T	True	true
OR	Ordered	NOT(?)
NEQ	Not Equal	NOT(=)
OLG	Ordered or Less Than or Greater Than	NOT(?=)
UGE	Unordered or Greater Than or Equal	?>=
OGE	Ordered Greater Than or Equal	NOT(?<)
UGT	Unordered or Greater Than	?>
OGT	Ordered Greater Than	NOT(?<=)
ST	Signaling True	
GLE	Greater Than or Less Than or Equal	<=>
SNE	Signaling Not Equal	
GL	Greater Than or Less Than	<>
NLT	Not Less Than	NOT(<)
GE	Greater Than or Equal	>=
NLE	Not Less Than or Equal	NOT(<=)
GT	Greater Than	>

MIPS FPU Core
(Continued)

FPU Instruction Summary – Table 4 lists all the FPU instructions currently implemented in the LR33050. For a detailed description of the instructions, refer to the *LR33050 MIPS IFX Processor User's Manual.* Any instructions that require computations not implemented in hardware cause the FPU to flag an unimplemented operation exception. The CPU then calls a routine from the floating-point emulation library to complete the instruction. The time to execute an FPU instruction ranges from one cycle to 34 cycles. Figure 12 on the next page shows the execution cycles (ALU) required for each FPU instruction.

Table 4. LR33050 FPU Instruction Summary

Instruction	Description
Load/Store/Move Instructions	
LWC1	Load Word to FPU
SWC1	Store Word from FPU
MTC1	Move Word to FPU
MFC1	Move Word from FPU
CTC1	Move Control Word to FPU
CFC1	Move Control Word from FPU
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 Overlap – When the FPU is running and not stalled, simultaneous execution can occur on as many as six separate instructions during each clock cycle. The FPU allows concurrent operations as long as instructions do not require the same compute resource, such as the multiply or divide units. Figure 13 on page 14 illustrates the overlapping of several FPU instructions. Although processing of a single instruction consists of five pipestages, the FPU does not require that the instruction actually be completed in five cycles to avoid stalling the pipeline. If an instruction does not require the resources being used by a preceding instruction, and it has no data dependencies on any uncompleted instructions or exceptional conditions, then execution continues. Table 5 lists the latency and throughput cycle times for single- and double-precision operations. Latency is the number of cycles needed to generate results.

Table 5. Instruction Latency and Throughput Values

	Latency		Throughput[1]	
Operation	**Single**	**Double**	**Single**	**Double**
ADD	4	4	1	1
SUB	4	4	1	1
CVT	4	4	1	1
MUL	6	7	3	4
DIV	19	34	16	31
ABS	1	1	1	1
MOV	1	1	1	1
NEG	1	1	1	1
C.cond	3	3	1	1

Note(s):
1. Assumes that operands do not have any data dependencies.

Floating-Point Exceptions

Five of the six of exceptions identified by the FPU are IEEE-standard exceptions. The sixth, an unimplemented operation exception, allows software to handle the exceptional conditions that are not handled in hardware.

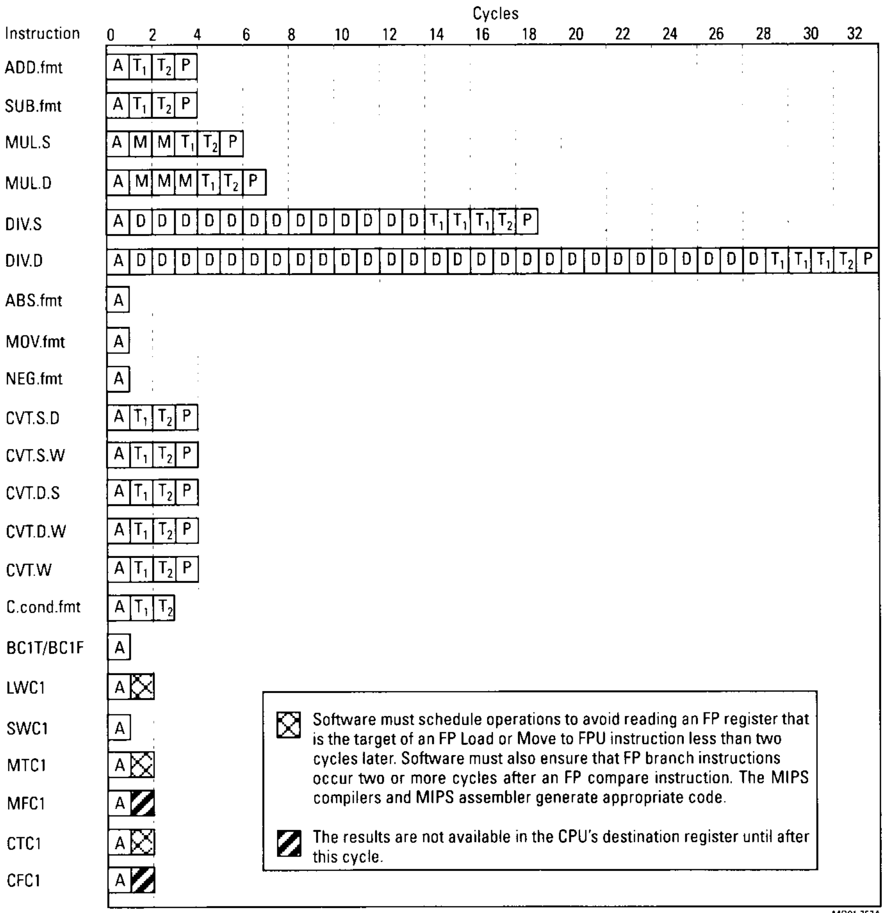
Table 6 on the next page shows the eight conditions detected by the FPU that can cause exceptions. It also lists the IEEE-standard requirements and the FPU behavior for these exception-causing conditions. Some exception types can be signaled for the same cause, and some types encompass several sources or causes.

MIPS FPU Core
(Continued)

Table 6. Exception-Causing Conditions

FPU Condition	IEEE Standard[1]	Trap Enable[1]	Trap Disable[1]	Description
Inexact result	I	I	I	Loss of accuracy
Exponent overflow	O I	O I	O I	Normalized exponent > E_{max}
Divide-by-zero	Z	Z	Z	Exponent = E_{min} - 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 < E_{min}
Denormalized source	—	E	E	Exponent = E_{min} -1, mantissa <> 0

Note(s):
1. Refer to Table 2 on page 8 for a description of the exception mnemonics.



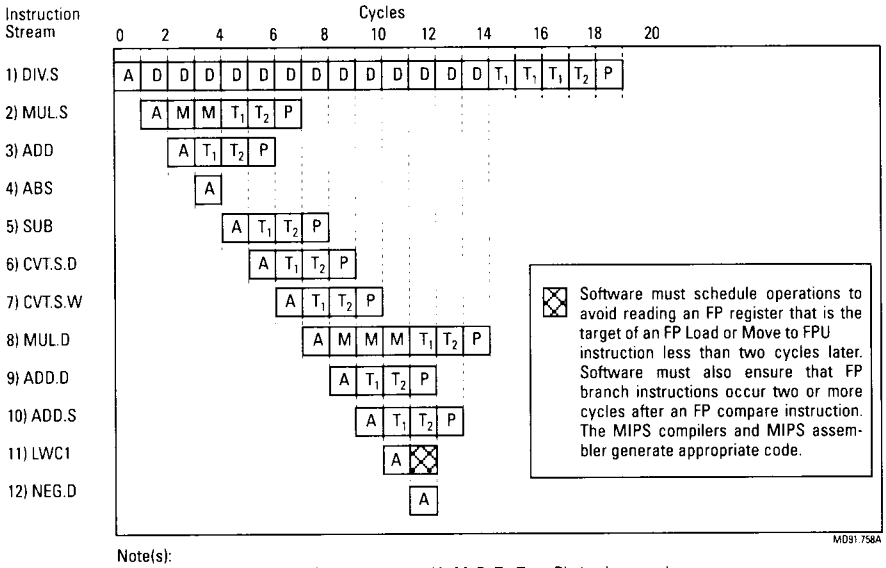
☒ Software must schedule operations to avoid reading an FP register that is the target of an FP Load or Move to FPU instruction less than two cycles later. Software must also ensure that FP branch instructions occur two or more cycles after an FP compare instruction. The MIPS compilers and MIPS assembler generate appropriate code.

▨ The results are not available in the CPU's destination register until after this cycle.

Note(s):
1. A, M, D, T_1, T_2 and P indicate different resources within the FPU.

Figure 12. FPU Instruction Execution Times

MIPS FPU Core
(Continued)



▨ Software must schedule operations to avoid reading an FP register that is the target of an FP Load or Move to FPU instruction less than two cycles later. Software must also ensure that FP branch instructions occur two or more cycles after an FP compare instruction. The MIPS compilers and MIPS assembler generate appropriate code.

Note(s):
1. FPU instructions cannot share a resource (A, M, D, T_1, T_2 or P) simultaneously.

Figure 13. Overlapping FPU Instructions

Exception Processing – When an exception occurs, bits may be set in both the exception and sticky bit fields of the FCR31 control register. (See the description of control register FCR31 on page 8.) 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 FPU begins default processing. The unimplemented operation exception (E), however, cannot be disabled; a trap is always taken.

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.

If traps are disabled, a sticky bit is set for each IEEE-standard exception when the exception occurs. Unlike exception bits that are reset with each new instruction, the sticky bit is 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 a group.

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

Invalid Operation Exception – If one or both of the operands of an instruction are invalid, an invalid operation exception occurs. 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 ∞/∞ 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 arithmetic operation on a signaling NaN. If at least one operand of the instruction is a signaling NaN, the invalid operation exception is signaled. Note that ABS and NEG are arithmetic operations, but MOV (move) is not.

MIPS FPU Core
(Continued)

The invalid operation exception may be simulated by software for other operations that are invalid for the source operands, such as IEEE specified functions implemented in software. Examples include:

- remainder, *x* REM *y*, where *y* is zero or *x* is infinite
- conversion of a floating-point number to a decimal format whose value causes an overflow or is infinity or NaN
- transcendental functions, such as ln(-5)

If invalid operation traps are enabled, the original operand values are not modified. If traps are disabled, the FPU always signals an unimplemented exception because it does not create the NaN that should be returned, according to the IEEE Standard, under these circumstances.

Divide-by-Zero Exception – This exception occurs during a divide instruction if the divisor is zero and the dividend is a finite non-zero number.

If divide-by-zero traps are enabled, the result register and source registers are undisturbed. When no trap occurs, the result is a correctly signed infinity (∞).

Overflow Exception – If the magnitude of the rounded floating-point result – were the exponent range unbounded – is larger than the largest finite number of the destination format, the overflow exception occurs.

If overflow traps are enabled, the result register and source registers are undisturbed. As shown in Table 7, when no trap occurs, the result is determined by the rounding mode and the sign of the intermediate result.

Underflow Exception – The FPU never generates an underflow exception or sets the U bit in either the exception or sticky bit fields of the Control/Status Register. Instead, the FPU generates an unimplemented operation exception if it detects a condition that could be either an underflow or loss of accuracy.

Inexact Exception – The FPU signals an inexact exception when the rounded result of an operation is not exact or if the result overflows.

Before execution of a floating-point operation begins, the FPU attempts to determine whether the operation may possibly cause an exception. If the FPU determines that the operation may result in an exception, it uses the coprocessor stall mechanism to execute the instruction.

Because inexact results cannot be predetermined, the FPU uses the stall mechanism to execute all floating-point operations if inexact exception traps are enabled. Note that inexact exception traps should be used only when necessary because the coprocessor stall mechanism adversely affects performance.

If inexact exception traps are enabled, the result register and source registers are undisturbed. If no other trap occurs, the rounded or overflowed result is sent to the destination register.

Unimplemented Operation Exception – This exception occurs when the FPU attempts to execute an instruction with an operation code or format code that has been reserved for future use. Note that this trap cannot be disabled. The original operand values are undisturbed.

Table 7. Default for Exceptions

Exception	Rounding Mode	Default Action (No exception trap signaled)
Invalid (V)	—	Generate an unimplemented exception.
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 negative overflows to −∞.
Underflow (U)	—	Generate an unimplemented exception.
Inexact (I)	—	Supply a rounded result.

MIPS FPU Core
(Continued)

The unimplemented operation exception may also be signaled when one of the following conditions occurs: denormalized operand, denormalized results, NaN operand, invalid operation with trap disabled, or underflow.

When an unimplemented operation exception occurs, the operation is then emulated in software. If an IEEE exception occurs while the operation is being emulated in software, these exceptions must, in turn, be simulated.

Defaults – When a floating-point operation generates an exceptional result, but no exception trap is signaled, the FPU takes a default action to supply a substitute value for the original exceptional result. Table 7 summarizes the default actions taken for each exception. Note that the default action depends on the exception. For an overflow exception, the default action also depends on the current rounding-mode bits.

Saving and Restoring State
All FGRs (Floating-Point General Registers) can be saved/restored to/from memory with 32 store/load instructions. The contents of the Control/Status Register are usually saved first and restored last. The CTC1 (Move Control To Coprocessor 1) and CFC1 (Move Control From Coprocessor 1) instructions are used to save the Control/Status Register contents.

Reading the Control/Status Register while the coprocessor is still executing floating-point instructions will cause those instructions to complete before the register contents are moved to the main processor. If one of these instructions results in an exception, the exception is saved in the exception field of the Control/Status Register.

Because the exception field of the Control/Status Register contains the result of only one instruction, the FPU examines source operands first and attempts to determine whether an operation can possibly cause an exception before the operation is actually executed. If the FPU determines that the operation may cause an exception, it executes the operation with a coprocessor stall mechanism – which ensures that only a single, possibly exceptional instruction is executed at a time.

Note that if a zero value is written to the exception field of the Control/Status Register, all bits in the field are cleared. After the state of the Control/Status Register is restored and the bits in the exception field have been cleared, the coprocessor can restart normal processing.

Pipeline Architecture

The LR33050 contains separate pipelines for the CPU and FPU. The pipelines consist of the following five primary stages:

1. **IF** – Fetch the instruction from the instruction cache (I-Cache).
2. **RD** – Read any required operands from CPU or FPU registers while decoding the instruction.
3. **ALU** – Perform the required arithmetic or logical operation on instruction operands.
4. **MEM** – Access memory from the data cache (D-Cache).
5. **WB** – Write results back to the CPU or FPU register file.

Each of these stages usually requires approximately one CPU cycle as shown in Figure 14; parts of some operations overlap into another cycle while other operations require only one half cycle to complete.

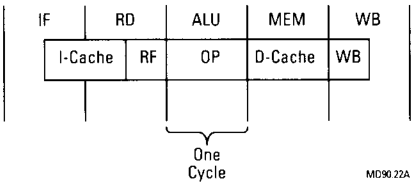


Figure 14. Instruction Execution Sequence

Pipeline Architecture (Continued)

The CPU and FPU each use their own five-stage pipeline to achieve an instruction execution rate approaching one instruction per CPU cycle. Thus execution of five instructions at a time can be overlapped as shown in Figure 15.

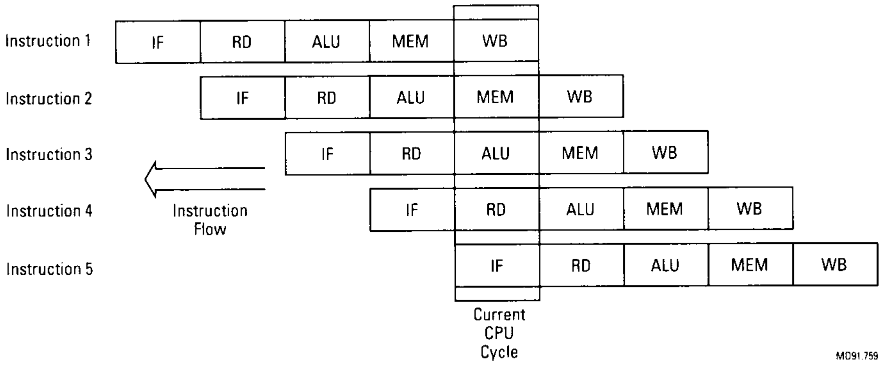


Figure 15. LR33050 Instruction Pipeline

On-Chip Cache Memories

The LR33050 contains a 4-Kbyte instruction cache and a 1-Kbyte data cache. Both caches are directly mapped to the LR33050's address space. Cache memories hold instructions and data that are repetitively accessed by the CPU (for example, within a program loop) and thus reduce the number of references that must be made to the slower main memory.

To provide an additional performance advantage, the LR33050 has separate data and instruction buses to allow the processor to access both caches during a single clock cycle when executing from cache memory. Figure 1 on page 1 shows the buses. The dual buses allow the processor to obtain data and instructions at the CPU cycle rate, even during load and store operations.

The LR33050 has several features that ensure coherency between its caches and main memory. The LR33050 uses bus snooping when another system device is performing DMA. During bus snooping, the LR33050 monitors the address bus. When the LR33050 detects an address that matches the address for a valid entry in the cache, it invalidates that entry to ensure that the data therein does not become stale. To ensure that the main memory is consistent with the data cache, the LR33050 uses a write-through technique. Whenever the LR33050 executes a write and there is a data cache hit, that data is also written through to main memory. In addition to these automatic cache coherency features, software can invalidate cache entries by executing store instructions with the cache invalidate enable bits in the Debug and Cache Invalidate Control Register (DCIC) set to one.

The LR33050 supports block refill for both caches. Refill block sizes can be set to two, four, eight, or 16 words, and the refill sizes can be set independently for each cache.

On-Chip DRAM Controller

The LR33050 has an integral DRAM Controller that supports most popular DRAMs. The DRAM Controller is mapped to a fixed address space from 0 to 0x0FFF.FFFF, which provides 256 Mbytes of address space for DRAM arrays. For accesses within that address space, the DRAM Controller generates $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, $\overline{\text{OE}}$, and the other control signals necessary to read and write DRAMs. The timing of both the $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ signals is programmable, and the Controller provides a data ready signal to the processor at the appropriate time in the DRAM access cycle. For small DRAM arrays, the only external logic required is an address multiplexer. Larger arrays require buffers for the control and data signals. Figure 16 on the next page shows a simple DRAM system.

On-Chip DRAM Controller (Continued)

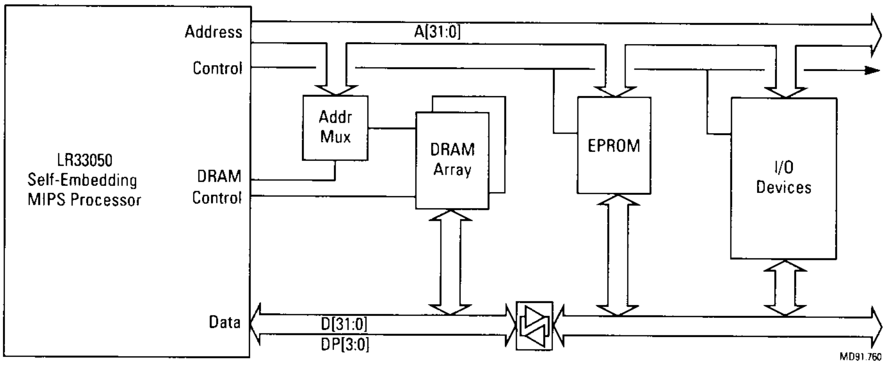


Figure 16. LR33050 Subsystem Interfaces

When the DRAM Controller is disabled, the DRAM address space can be used to access other types of memory or standard I/O devices.

To provide higher system performance, the DRAM Controller utilizes the page mode provided by some DRAMs to support block refills of cache. To provide a further performance boost, the DRAM Controller (with the addition of PAL-implemented control logic) supports interleaved memories.

The DRAM Controller can also be configured to do burst writes when consecutive stores are done to addresses within a page. The page size can be configured to range between 16 words and 2K words. Burst-mode operation and page size are set up in the Configuration Register. Refer to the *LR33050 MIPS IFX Processor User's Manual* for more information about the Configuration Register.

The DRAM Controller uses the $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ technique to support DRAM refresh. The LR33050 provides a 12-bit Refresh Timer/Counter to signal refresh intervals. The Refresh Timer/Counter can also be used to support external DRAM controllers, and the LR33050's DRAM Controller can be disabled by software.

To provide designers with maximum flexibility, the DRAM Controller supports DMA by external devices and can directly support dual-port DRAMs for those systems that require them.

Bus Interface Unit (BIU)

The LR33050's BIU provides a simple memory interface that connects to I/O devices, supports both 8- and 32-bit PROMs, and includes a programmable wait-state generator. In addition, the BIU includes a four-word-deep write buffer and parity logic.

As shown in Figure 16, the LR33050's memory interface consists of a 32-bit address bus, a 32-bit data bus, a 4-bit parity bus, and the control signals necessary to manage transfers at the interface. The interface is directly compatible with industry standard I/O devices and memory devices, such as UARTs, DMA controllers and SRAMs.

I/O Device Support

To simplify the connection of I/O devices, the LR33050 provides an I/O select signal that it asserts for reads or writes in the address range 0x1E00.0000 to 0x1EFF.FFFF. The I/O select signal simplifies the address decoders required for I/O devices that are mapped to this range. One of the LR33050's two wait-state generators further simplifies the connection of I/O devices within this address range. When enabled, the wait-state generator stalls the processor for a programmable number of cycles (from 0 to 31). During the last wait state, the wait-state generator asserts the data ready signal, terminating the memory transaction. To accommodate I/O devices of various speeds, software can reprogram the wait-state generator before each access, or faster devices can terminate an access cycle early by asserting the data ready signal before the programmed number of wait states has expired. The LR33050 also supports byte-wide devices by performing byte gathering during reads if an external device asserts the LR33050's $\overline{\text{BWIDE}}$ input.

Bus Interface Unit (BIU) (Continued)

PROM Support

To simplify the connection of PROMs, the LR33050 provides a PROM select signal that it asserts for reads or writes in the address range 0x1F00.0000 to 0x1FFF.FFFF. The LR33050 supports byte-wide devices by performing byte gathering when the LR33050's $\overline{\text{BWIDE}}$ input is asserted. One of the LR33050's two wait-state generators further simplifies the connection of PROMs within this address range. When enabled, the wait-state generator stalls the processor for a programmable number of cycles (from 0 to 31). During the last wait state, the wait-state generator asserts the data ready signal, terminating the memory transaction. To accommodate PROMs of various speeds, software can reprogram the wait-state generator before each access, or faster devices can terminate an access cycle early by asserting the data ready signal before the programmed number of wait states has expired.

The LR33050's bootstrap vector falls within the PROM select signal's address range (0x1F00.0000 to 0x1FFF.FFFF), and the LR33050 maps its exception vectors into the PROM select address range when the Bootstrap Exception Vector (BEV) bit in the Status Register is set to one. The BEV bit is automatically set to one on reset and power up.

Four-Word-Deep Write Buffer

To eliminate processor stalls during write accesses, the LR33050 includes a four-word-deep write buffer. The write buffer captures data (and the associated address) output by the processor and ensures that the data are passed on to main memory. The write buffer also checks for in-page writes. If consecutive in-page writes are found in the buffer, the DRAM Controller performs burst-mode writes. If the write buffer is full during a write access, the BIU stalls the processor until the buffer is no longer full. The write buffer can be configured as four-word deep or one-word deep through the 4D bit in the Configuration Register. Refer to the *LR33050 MIPS IFX Processor User's Manual* for more information about the Configuration Register.

Parity Logic

The BIU supports optional parity generation and checking. Checking may be enabled on a per-memory-access basis. The LR33050 provides the PERR output to indicate parity errors to external logic.

On-Chip Timer/ Counters

The LR33050 has two 24-bit timer/counters, Timer 1 and Timer 2, that software can use to time events. These two decrementing timers are clocked by the system clock, and they may interrupt the processor when they reach zero. When the timers reach zero, the timer hardware reloads the counter with the initial value and continues decrementing.

Timer 1 is wire-ORed to Interrupt 0 and Timer 2 is wire-ORed to Interrupt 1. When the timers decrement to zero, they assert the interrupt signals until software resets the interrupt bits in the appropriate Timer Control Register. To allow software to time external events, Timer 2 has an enable input, Timer-2-Enable, and a timeout signal, Timer-2-Timeout, which toggles each time the counter decrements to zero.

Each timer is initialized and controlled by two memory-mapped registers: an Initial Count Register and a Control Register.

Compatibility with the R2000, R3000 and LR33000 Microprocessors

The LR33050 is pin-compatible and electrically compatible with the LR33000, which allows a direct upgrade to current LR33000-based systems that include an FPU coprocessor. The LR33050 is fully compatible to ANSI/IEEE Standard 754-1985 floating-point arithmetic and provides instruction set compatibility with the R2000, R3000 and LR33000 microprocessors. Therefore software developed for the R2000, R3000 and LR33000 can be easily ported to systems based on the LR33050, significantly reducing software development time and resource requirements.

However, the LR33050 does not support external coprocessors and does not implement the R2000/R3000 memory management unit (MMU). In addition, the internal cache memories are managed differently and two hardware breakpoint registers have been added to facilitate software debugging. These differences are further described below.

Coprocessor Support

The LR33050 does not support external coprocessors. To prevent unpredictable results should a coprocessor instruction be decoded,

Compatibility with the R2000, R3000, and LR33000 Microprocessors (Continued)

all external coprocessors should be disabled by setting the appropriate coprocessor usability bits in CP0's Status Register to zero. However, note that bit 1 must remain enabled for the FPU coprocessor. When the external coprocessors are disabled in this way, the detection of a coprocessor instruction causes a Coprocessor Unusable Exception. Table 8 lists the unsupported coprocessor instructions for Coprocessor 0, 2 and 3.

Table 8. Unimplemented Instructions

Opcode	Description
Coprocessor Instructions	
LWCz	Load Word from Coprocessor
SWCz	Store Word to Coprocessor
MTCz	Move To Coprocessor[1]
MFCz	Move From Coprocessor[1]
CTCz	Move Control To Coprocessor
CFCz	Move Control From Coprocessor
COPz	Coprocessor Operation
System Control Coprocessor (CP0) Instructions	
TLBR	Read indexed TLB entry
TLBWI	Write Indexed TLB entry
TLBWR	Write Random TLB entry
TLBP	Probe TLB for matching entry

Note(s):
1. Implemented for CP0 only.

The LR33050 provides four coprocessor condition signals that software can test using the Branch on Coprocessor z True and Branch on Coprocessor z False (BCzT and BCzF) instructions. This facility allows software to poll hardware that is connected to the condition signals. The corresponding coprocessor usability bit in CP0's Status Register must be set to use the test instruction.

Memory Management Unit Support

The LR33050 does not implement the R2000/R3000 memory management unit, which includes the Translation Lookaside Buffer (TLB). Consequently, the LR33050 does not implement any of the instructions designed to maintain and use the TLB. If the processor detects a TLB instruction, it causes a Reserved Instruction Exception. Table 8 lists the unimplemented TLB instructions. In addition, the LR33050 does not implement the TLB registers in CP0: EntryHi, EntryLo, Index, Random and Context. References to those registers cause a Reserved Instruction Exception.

The System Control Processor: CP0

The TLB registers mentioned above have been removed; however, the CP0 includes additional debug and control registers, which are summarized in Figure 4 on page 5.

Debug Registers – The LR33050 includes program counter and data address breakpoint registers to simplify software debugging. These registers have been incorporated into CP0, and software accesses them using Move From Coprocessor and Move To Coprocessor (MFC0 and MTC0) instructions, as in the LR33000.

Debug and Cache Invalidate Control Register – This register contains the enable and status bits for the LR33050's debug mechanism and the control bits for the Cache Controller's invalidate mechanism. (The LR33050 does not implement the R2000/R3000 cache control bits, which were located in the Status Register.)

Signal Descriptions

This section describes the signals that comprise the LR33050's bit-level interface to other devices. This section is divided into five sections:

- Memory Interface Signals
- Bus Arbitration Signals
- Peripheral Interface Signals
- Status Signals
- Miscellaneous Signals

Memory Interface Signals

This subsection describes the LR33050's memory interface, which consists of a 32-bit Address Bus, a 32-bit Data Bus, and the control signals required to regulate transactions with memory and I/O subsystems. The memory interface also contains the signals required to control DRAM arrays directly.

A[31:0]

Address Bus [31:0] – When it is bus master, the LR33050 uses A[31:0] to transmit instruction and data addresses to the memory and peripherals. When another device owns the bus, the LR33050 snoops addresses generated by the bus master (if $\overline{\text{CACHD}}$ is asserted).

$\overline{\text{AS}}$

Address Strobe – $\overline{\text{AS}}$ indicates that a valid address is present on the Address Bus. During memory transactions as bus master, the LR33050 asserts $\overline{\text{AS}}$ one half cycle after it asserts $\overline{\text{MXS}}$.

$\overline{\text{BERR}}$

Bus Error – External logic asserts this signal to indicate that an exceptional condition has

Signal Definitions (Continued)

occurred. When $\overline{\text{BERR}}$ is asserted, the LR33050 terminates the current memory transaction and takes a Bus Error Exception.

BFREQ
Block Fetch Request – The LR33050 asserts this signal to indicate that it wants to perform a block-fetch transaction. The LR33050 asserts BFREQ whenever a cache miss occurs.

$\overline{\text{BFTCH}}$
Block Fetch – In response to the LR33050's assertion of BFREQ, external memory control logic asserts this signal to indicate that the LR33050 can expect a block transfer from memory.

When the LR33050's DRAM Controller is used to manage memory, the DRAM Controller acknowledges the BFREQ signal internally and automatically. Unless there is an external device that is capable of block-fetch transactions, $\overline{\text{BFTCH}}$ should be tied to VDD when using the internal DRAM Controller.

$\overline{\text{BWIDE}}$
Byte-Wide Port – External logic asserts $\overline{\text{BWIDE}}$ to indicate that the memory interface is eight bits wide. When $\overline{\text{BWIDE}}$ is asserted, the LR33050 executes four memory cycles with sequential byte addresses, beginning with the effective address. (The byte-wide feature supports partial-word accesses, but always performs four memory cycles.) If the effective address is not modulo four, the LR33050 wraps around to get all of the bytes in the word in which the effective address falls.

For byte-wide transactions, the LR33050 always takes data from bits D[7:0].

$\overline{\text{CACHD}}$
Cacheable Data – External logic asserts this signal to indicate that the instruction or data fetched during the current cycle may be stored in the LR33050's on-chip cache. If $\overline{\text{CACHD}}$ is not asserted, the LR33050 does not cache the instruction or data fetched during the current transaction.

D[31:0]
Data Bus [31:0] – The LR33050 uses D[31:0] to transfer data and instructions to and from memory and peripherals. When another device owns the bus, it may drive or sample these signals.

$\overline{\text{DCAS}}$
DRAM Column Address Strobe – During a DRAM memory transaction, the LR33050's DRAM Controller asserts this signal to indicate that the address presented to the DRAM array is the column address.

DMXS
DRAM Mux Select – The LR33050's DRAM controller outputs DMXS for the Data Select input of the multiplexers required to multiplex the row and column address from the Address Bus onto the DRAM address bus. When HIGH, DMXS indicates that the row address should be selected. When LOW, DMXS indicates that the column address should be selected.

$\overline{\text{DOE}}$
DRAM Output Enable – The LR33050's DRAM Controller asserts this signal to enable DRAM data outputs.

DP[3:0]
Data Parity [3:0] – DP[3:0] allow the LR33050 to check and transmit parity bits for the four data bus bytes. The LR33050 uses even parity.

$\overline{\text{DRAS}}$
DRAM Row Address Strobe – During a DRAM memory transaction, the LR33050's DRAM Controller asserts $\overline{\text{DRAS}}$ to indicate that the address presented to the DRAM array is the row address.

$\overline{\text{DRDY}}$
Data Ready – External logic asserts this signal to indicate that it can accept or is providing data. The assertion of $\overline{\text{DRDY}}$ terminates the memory transaction. Because the LR33050 must wait for external logic to assert $\overline{\text{DRDY}}$ before it can complete a memory transaction, slow devices can use $\overline{\text{DRDY}}$ to stall the processor. However, when the automatic wait state generator is enabled, the LR33050 asserts an internal version of $\overline{\text{DRDY}}$ to end transactions.

$\overline{\text{EPSEL}}$
EPROM Select – The LR33050 asserts this signal to indicate that it is accessing the $\overline{\text{EPSEL}}$ address space (0x1F00.0000 to 0x1FFF.FFFF). $\overline{\text{EPSEL}}$ can be used as a chip select. If automatic wait state generation is enabled, the LR33050 generates an internal $\overline{\text{DRDY}}$ signal a programmable number of cycles after the transaction begins. The external PROM logic may override

Signal Definitions (Continued)

the internal wait-state generator by asserting the $\overline{\text{DRDY}}$ signal before the programmed number of wait states have passed.

$\overline{\text{IOSEL}}$
I/O Select – The LR33050 asserts this signal to indicate that it is accessing the $\overline{\text{IOSEL}}$ address space (0x1E00.0000 to 0x1EFF.FFFF). $\overline{\text{IOSEL}}$ can be used as a chip select. If automatic wait-state generation is enabled, the LR33050 generates its own $\overline{\text{DRDY}}$ signal a programmable number of cycles after the transaction begins. The external I/O logic may override the internal wait-state generator and use the $\overline{\text{DRDY}}$ signal to control the number of wait states.

$\overline{\text{MXS}}$
Memory Transaction Start – The LR33050 asserts $\overline{\text{MXS}}$ for one clock cycle at the beginning of a memory transaction to indicate the start of the transaction.

$\overline{\text{PEN}}$
Parity Enable – When asserted by external logic during a read transaction, the LR33050 performs byte-wise checking for even parity on the Data and Data Parity Buses. When deasserted, the LR33050 does not check parity. External logic that is connected to the LR33050's byte-wide port should never assert $\overline{\text{PEN}}$.

PERR
Parity Error – The LR33050 asserts PERR to indicate that it has detected a parity error during the just-completed memory transaction. PERR can be tied to an interrupt input to cause a parity error exception.

$\overline{\text{RD}}$
Read Strobe – When bus master, the LR33050 asserts $\overline{\text{RD}}$ to indicate that the current transaction is a read transaction and that memory may drive data onto D[31:0]. When deasserted, only the LR33050 may drive D[31:0].

When external logic is bus master, it should drive $\overline{\text{RD}}$ as described in the previous paragraph.

$\overline{\text{RT}}$
Read Transaction – When bus master, the LR33050 asserts $\overline{\text{RT}}$ to indicate that the current memory cycle is a read transaction. The LR33050 deasserts $\overline{\text{RT}}$ during write transactions.

When the LR33050 is not bus master, the LR33050 monitors this signal to determine whether it should perform cache snooping.

When external logic is bus master, it should drive $\overline{\text{RT}}$ as described in the previous paragraph.

$\overline{\text{WR}}$[3:0]
Byte Write Strobes [3:0] – The LR33050 asserts these signals to indicate that there is valid data on the corresponding byte of the Data Bus. Some of the 1-Mbit DRAMs and most of the 4-Mbit DRAMs specify that their $\overline{\text{WE}}$ inputs must be high during $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh. For these devices, it is necessary to generate the $\overline{\text{WE}}$ inputs external to the LR33050. Figure 17 shows one example of how the $\overline{\text{WR}}$[3:0] and $\overline{\text{RTO}}$ signals from the LR33050 may be used to generate $\overline{\text{WE}}$[3:0] externally.

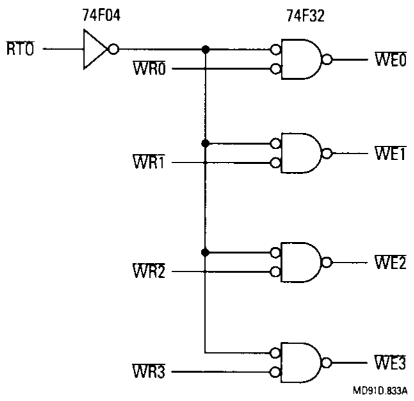


Figure 17. External $\overline{\text{WE}}$ Inputs

Bus Arbitration Signals
This subsection describes the LR33050's bus arbitration signals. The LR33050's memory interface is designed to be used as a general-purpose bus, and these signals allow intelligent peripherals to request bus mastership to perform DMA operations or access the LR33050's on-chip cache memories.

$\overline{\text{BREQ}}$
Bus Request – External bus masters assert this signal to request control of the Address and Data Buses and the relevant control signals.

Signal Definitions (Continued)

BGNT
Bus Grant – The LR33050 asserts this signal to grant control of the memory buses to the requesting device. When BGNT is asserted, the LR33050 3-states the following signals:

D[31:0]	DP[3:0]
A[31:0]	$\overline{\text{MXS}}$
$\overline{\text{AS}}$	$\overline{\text{RT}}$
$\overline{\text{RD}}$	$\overline{\text{DT}}$
$\overline{\text{WR}}$[3:0]	BFREQ

When BGNT is asserted, the requesting bus master can control the 3-state signals.

$\overline{\text{DMAR}}$
DMA Request – External logic asserts this signal to request use of the LR33050's DRAM Controller. The $\overline{\text{DMAR}}$ and $\overline{\text{DMAC}}$ signals are not related to the $\overline{\text{BREQ}}$ and BGNT signals. External $\overline{\text{DRDY}}$ cannot be used for DRAM space with this type of DMA.

$\overline{\text{DMAC}}$
DMA Cycle – The LR33050 asserts this signal to acknowledge the $\overline{\text{DMAR}}$ and to indicate that the DRAM Controller has started the access for the requesting device. During such DMA accesses, the DRAM Controller manages the $\overline{\text{DRAS}}$, $\overline{\text{DCAS}}$ and DMXS signals. The DMA device must drive the RAM's write strobes, address bus and data bus; a configuration that requires multiport RAMs.

During DMA operations by other devices, the LR33050 continues to control and use the A[31:0], D[31:0] and DP[3:0]. If both external logic and the LR33050 need to use the DRAM Controller, they take turns, switching control at the end of each memory transaction.

D$TST
Data Cache Access – When this signal is asserted, external logic can read and write the LR33050's data cache. In data cache access mode the external device controls A[31:0], D[31:0], $\overline{\text{AS}}$ and $\overline{\text{RT}}$ signals.

I$TST
Instruction Cache Access – When this signal is asserted, external logic can read and write the LR33050's instruction cache. In instruction cache access mode the external device controls the A[31:0], D[31:0], $\overline{\text{AS}}$ and $\overline{\text{RT}}$ signals.

Peripheral Interface Signals
The LR33050 contains three timer/counters. Two of the three timers can be controlled via the peripheral interface signals described in this subsection.

CPC[3:0]
Coprocessor Condition [3:0] – The four Coprocessor Condition signals, CPC[3:0], carry condition inputs that the LR33050 can separately test using coprocessor branch instructions. Note that the corresponding coprocessor usable bit (Cu[3:0]) in the Status Register must be set in order to test a condition input.

Although the CPC inputs are tested using coprocessor branch instructions, the LR33050 hardware makes no assumptions about the actual source of these signals. Therefore the use of these condition inputs is application-dependent. Note that CPC1 is ORed internally with the condition bit from the on-chip FPU. CPC1 must be held inactive when the FPU condition bit is being tested, and the FPU condition bit must be cleared when the state of CPC1 is being tested.

INT[5:0]
Interrupt [5:0] – External logic asserts these signals to cause the LR33050 to take an interrupt exception. Note that interrupts are not sampled in any other type of stall cycles. An instruction sees the interrupt only during the ALU pipestage. After the assertion of an interrupt and the occurrence of an interrupt exception, the interrupts continue to be sampled to provide a level-sensitive indication of the active interrupt or interrupts. The interrupts are not latched within the processor when an interrupt exception occurs.

The interrupt inputs can be individually disabled or masked by setting the appropriate bit in the LR33050's Status Register. Note that INT3 is ORed internally with the interrupt signal from the on-chip FPU.

$\overline{\text{RTO}}$
Refresh Timeout – The LR33050 asserts this signal to indicate that the Refresh Timer has counted down from its preset value to zero. The LR33050 continues to assert $\overline{\text{RTO}}$ until external logic asserts RTACK, but the Counter does not stop running: it loads the initial value and con-

Signal Definitions
(Continued)

tinues to count. When the DRAM Controller is enabled, this signal indicates that a refresh cycle is in progress. When the DRAM Controller is disabled, external DRAM control logic can use the assertion of $\overline{\text{RTO}}$ to start a refresh cycle.

RTACK
Refresh Timer Acknowledge – External logic asserts RTACK to indicate that it has completed the refresh operation initiated by the LR33050's assertion of $\overline{\text{RTO}}$. The assertion of RTACK causes the LR33050 to deassert $\overline{\text{RTO}}$. External logic must assert RTACK for two cycles to guarantee that $\overline{\text{RTO}}$ resets.

When the DRAM Controller is enabled, RTACK must be tied low. If DRAM refresh is not used, setting the Refresh Timer value to zero causes a constant timeout instead of disabling the Timer.

T2EN
Timer 2 Enable – External logic asserts this signal to enable the operation of Timer 2. Timer 2 counts as long as T2EN is asserted.

T2TO
Timer 2 Timeout – The LR33050 toggles T2TO to indicate that the LR33050's Timer 2 has counted down from its preset value to zero. T2TO is set LOW on cold and warm reset.

Status Signals
The LR33050 generates four signals that indicate its internal state. Those signals are described in this subsection.

BRTKN
Branch Taken – When asserted, this signal indicates that the condition has been met for a branch instruction in the LR33050's pipeline. The LR33050 signals branches during the ALU stage of the pipeline, which is when the control transfer occurs.

$\overline{\text{DT}}$
Data Transaction – The LR33050 asserts this signal to indicate that the current bus transaction is a data read or write transaction. The LR33050 deasserts this signal to indicate that the current bus transaction is an instruction fetch.

$\overline{\text{STALL}}$
Stall – The LR33050 asserts this signal to indicate that it is in a stall state.

TDONE
Test Mode Transaction Done – The LR33050 asserts this signal to indicate that a cache access mode read or write operation is complete.

Miscellaneous Signals
The LR33050 has six miscellaneous inputs, which are described in this subsection.

BENDN
Big Endian – This signal controls the byte ordering of the LR33050. When BENDN is tied HIGH, byte 0 is the most-significant byte (big endian). When BENDN is tied LOW, byte 0 is the least-significant byte (little endian).

$\overline{\text{FRCM}}$
Force Cache Miss – When external logic asserts $\overline{\text{FRCM}}$, the LR33050 detects a cache miss on the current instruction fetch. This signal is intended for test and in-circuit emulation purposes.

$\overline{\text{HIGHZ}}$
High Impedance – When external logic asserts $\overline{\text{HIGHZ}}$, the LR33050 3-states all of its outputs. SYSCLK need not be active for $\overline{\text{HIGHZ}}$ to 3-state the LR33050 outputs. In addition, $\overline{\text{HIGHZ}}$ selects between warm and cold processor starts after reset. When $\overline{\text{HIGHZ}}$ and $\overline{\text{RESET}}$ are asserted together, the LR33050 executes a cold start after $\overline{\text{RESET}}$ is deasserted. When $\overline{\text{RESET}}$ is asserted alone, the LR33050 executes a warm start. The DRAM Controller is *not* re-initialized on a warm start.

$\overline{\text{RESET}}$
Reset – A LOW-to-HIGH transition of $\overline{\text{RESET}}$ initializes the processor and initiates a non-maskable RESET exception. Driving $\overline{\text{RESET}}$ LOW does not halt the processor or cause any other action. Only a LOW-to-HIGH transition of $\overline{\text{RESET}}$ initializes the processor.

SYSCLK
System Clock – This signal is the clock input to the processor. It determines the instruction cycle time of the processor.

THIT
Test Hit – The LR33050 asserts THIT during a read of its instruction or data caches to indicate that the cache tags matched the address of the read. This signal is intended for test purposes.

LR33050
MIPS IFX
Processor
Preliminary

LSI LOGIC

Specifications

This section provides the electrical specifications for the LR33050. Table 9 lists the absolute maximum rating for the LR33050. Operation beyond the limits set forth in this table may impair the useful life of the device.

Table 9. Absolute Maximum Ratings

Symbol	Parameter	Limits[1]	Unit
VDD	DC Supply	-0.3 to +7	V
VIN	Input Voltage	-0.3 to VDD +0.3	V
IIN	DC Input Current	±10	mA
TSTG	Storage Temperature Range (Plastic)	-40 to +125	°C

Note(s):
1. Referenced to VSS.

Table 10 lists the LR33050's recommended operating conditions.

Table 10. Recommended Operating Conditions

Symbol	Parameter	Limits	Unit
VDD	DC Supply	+4.75 to +5.25	V
TA	Ambient Temperature	0 to +70	°C

Table 11 lists the input and output capacitances for the LR33050.

Table 11. Capacitance

Symbol	Parameter[1]	Min	Max	Unit
CIN	Input Capacitance		5	pF
COUT	Output Capacitance		10	pF
CIO	I/O Bus Capacitance		15	pF

Note(s):
1. Measurement conditions are VIN = 5.0 V, TA = 25°C, and clock frequency = 1 MHz.

Table 12 lists the DC electrical specifications for the LR33050. Table 13 lists the AC electrical specifications for the LR33050, and Figures 18 through 20 are timing waveforms which illustrate the AC timing values. In the AC specifications, all timing is referenced to 1.5 V, and all output timing assumes 55 pF of capacitive load.

Note: Not more than one output should be shorted at a time; duration of the short should not exceed 30 seconds.

Table 12. DC Characteristics

Symbol	Parameter	Condition[1]	Min	Typ	Max	Unit
VIL	Voltage Input LOW		–	–	0.8	V
VIH	Voltage Input HIGH		2.0	–	–	V
VOH	Voltage Output HIGH	IOH = -4.0 mA IOH = -8.0 mA	2.4 2.4	4.5 4.5	– –	V V
VOL	Voltage Output LOW	IOL = 4.0 mA IOL = 8.0 mA	– –	0.2 0.2	0.4 0.4	V V
IIL	Current Input Leakage	VDD = Max, VIN = VDD or VSS	-10	±1	10	µA
IOZ	Current 3-State Output Leakage	VDD = Max, VOUT = VSS or VDD	-10	±1	10	µA
IIPU	Current Input Pull-Up	VIN = VSS or 3.5 V	-35	-115	-350	µA
IOZU	Current 3-State Output w/ Pull-up	VIN = VSS or 3.5 V	-2	–	-175	µA
IOSP4	Current P-Channel Output Short Circuit (4 mA Output Buffers)[2]	VDD = Max, VOUT = VSS	-25	-70	-140	mA
IOSP8	Current P-Channel Output Short Circuit (8 mA Output Buffers)[2]	VDD = Max, VOUT = VSS	-50	-140	-280	mA
IOSN4	Current N-Channel Output Short Circuit (4 mA Output Buffers)[2]	VDD = Max, VOUT = VDD	30	75	140	mA
IOSN8	Current N-Channel Output Short Circuit (8 mA Output Buffers)[2]	VDD = Max, VOUT = VDD	60	150	280	mA
IDD	Quiescent Supply Current	VIN = VDD or VSS	–	–	2	mA
ICC	Dynamic Supply Current	VDD = Max, f = 25 MHz VDD = Max, f = 33 MHz VDD = Max, f = 40 MHz	– – –	400 460 500	700 800 900	mA mA mA

Note(s):
1. Specified at VDD equals 5 V ± 5% at ambient temperature over the specified range.
2. Note more than one output may be shorted at a time for a maximum duration of one second.

Specifications
(Continued)

Table 13. AC Timing Values

#	Parameter	Description	40 MHz		33 MHz		25 MHz		
			Min	Max	Min	Max	Min	Max	Unit
13	t_{SHAV}	SYSCLK+ to Address Valid	–	17	–	20	–	26	ns
14	t_{SHBRV}	SYSCLK+ to BFREQ Valid[1]	–	15	–	16	–	20	ns
15	t_{SHBRI}	SYSCLK+ to BFREQ Invalid	–	15	–	16	–	20	ns
16	t_{SHML}	SYSCLK+ to $\overline{MXS}$ Low	–	15	–	16	–	20	ns
17	t_{SHRTL}	SYSCLK+ to $\overline{RT}$ Low	–	15	–	16	–	20	ns
18	t_{SHRTH}	SYSCLK+ to $\overline{RT}$ High	–	14	–	15	–	18	ns
19	t_{SHDTV}	SYSCLK+ to $\overline{DT}$ Valid	–	15	–	16	–	20	ns
20	t_{SHDTI}	SYSCLK+ to $\overline{DT}$ Invalid	–	15	–	16	–	20	ns
21	t_{SLAL}	SYSCLK- to $\overline{AS}$, $\overline{RD}$ Low	–	17	–	20	–	23	ns
22	t_{SLSL}	SYSCLK- to $\overline{IOSEL}$ or $\overline{EPSEL}$ Low	–	14	–	15	–	18	ns
23	t_{SLSH}	SYSCLK- to $\overline{IOSEL}$ or $\overline{EPSEL}$ High	–	14	–	15	–	18	ns
24	t_{AVASL}	Address Valid to $\overline{AS}$, $\overline{RD}$, $\overline{IOSEL}$, $\overline{EPSEL}$ Low	5	–	5	–	5	–	ns
25	t_{CSSL}	Control[2] Setup to SYSCLK-	4	–	5	–	6	–	ns
26	t_{CHSL}	Control[3,4] Hold from SYSCLK-	3	–	4	–	5	–	ns
26a	t_{BEHSL}	$\overline{BERR}$ Hold from SYSCLK-	5	–	6	–	7	–	ns
27	t_{DSSH}	Data and Parity Setup before SYSCLK+	0	–	0	–	0	–	ns
28	t_{DHSH}	Data and Parity Hold from SYSCLK+	6	–	7	–	9	–	ns
29	t_{SHASH}	SYSCLK+ to $\overline{AS}$, $\overline{RD}$ High	–	12	–	13	–	15	ns
30	t_{AHSH}	Address Hold from $\overline{AS}$, $\overline{RD}$, $\overline{IOSEL}$, $\overline{EPSEL}$ High	0	–	0	–	0	–	ns
31	t_{SHWRV}	SYSCLK+ to $\overline{WR}$[3:0] Valid	–	15	–	16	–	20	ns
32	t_{SHDV}	SYSCLK+ to Data and Parity Valid	–	19	–	21	–	26	ns
33	t_{SHWRH}	SYSCLK+ to $\overline{WR}$[3:0] High	–	13	–	14	–	16	ns
34	t_{WRHDI}	$\overline{WR}$[3:0] High to Data and Parity Invalid	10	–	10	–	10	–	ns
51	t_{SLRL}	SYSCLK- to $\overline{DRAS}$ Low	–	13	–	14	–	17	ns
52	t_{SLRH}	SYSCLK- to $\overline{DRAS}$ High	–	13	–	14	–	17	ns
53	t_{ASRL}	Address Setup before $\overline{DRAS}$ Low	5	–	5	–	8	–	ns
54	t_{SLMSL}	SYSCLK- to DMXS Low	–	13	–	14	–	16	ns
55	t_{SLMSH}	SYSCLK- to DMXS High	–	13	–	14	–	16	ns
56	t_{SLCAL}	SYSCLK- to $\overline{DCAS}$ Low	–	13	–	14	–	17	ns
57	t_{SLOEL}	SYSCLK- to $\overline{DOE}$ Low	–	13	–	14	–	17	ns
58	t_{SHCAH}	SYSCLK+ to $\overline{DCAS}$ High	–	13	–	14	–	17	ns
59	t_{SHOEH}	SYSCLK+ to $\overline{DOE}$ High	–	13	–	14	–	17	ns

Note(s):
1. Block-fetch transactions must have one wait state in the first word of a block fetch. A block-fetch transaction is requested on any read access to cacheable memory after a cache miss.
2. Includes $\overline{DRDY}$, $\overline{BERR}$, $\overline{CACHD}$, $\overline{BWIDE}$, $\overline{PEN}$ and $\overline{BFTCH}$.
3. Includes $\overline{DRDY}$, $\overline{CACHD}$, $\overline{BWIDE}$, $\overline{PEN}$ and $\overline{BFTCH}$.
4. Once sampled at SYSCLK-, $\overline{BWIDE}$ is ignored until four bytes have been fetched or a bus error has been detected. $\overline{PEN}$ is ignored if $\overline{BWIDE}$ is sampled at SYSCLK-.

LSI LOGIC

Specifications
(Continued)

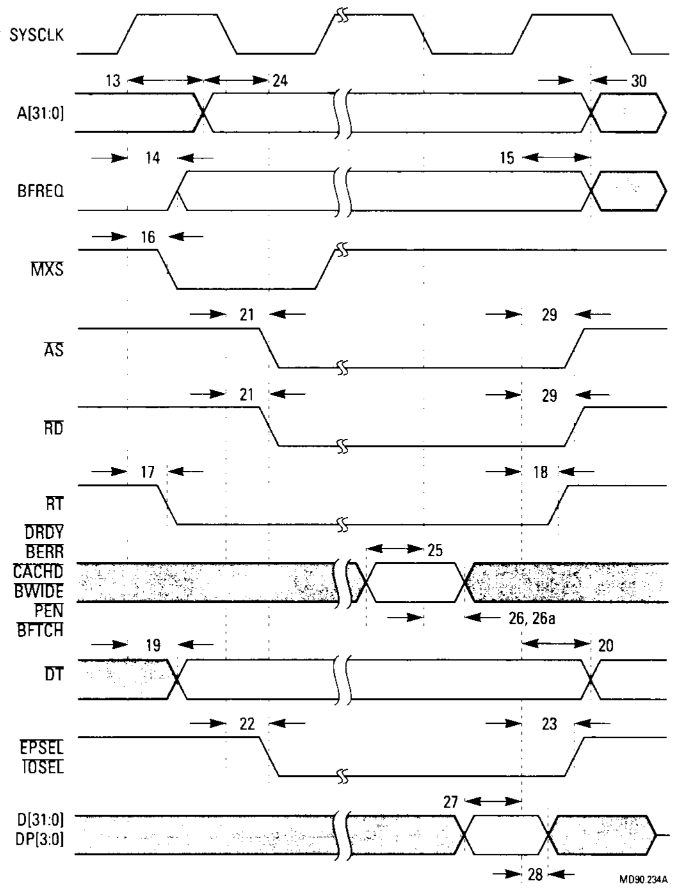


Figure 18. Read Transaction Timing

Specifications
(Continued)

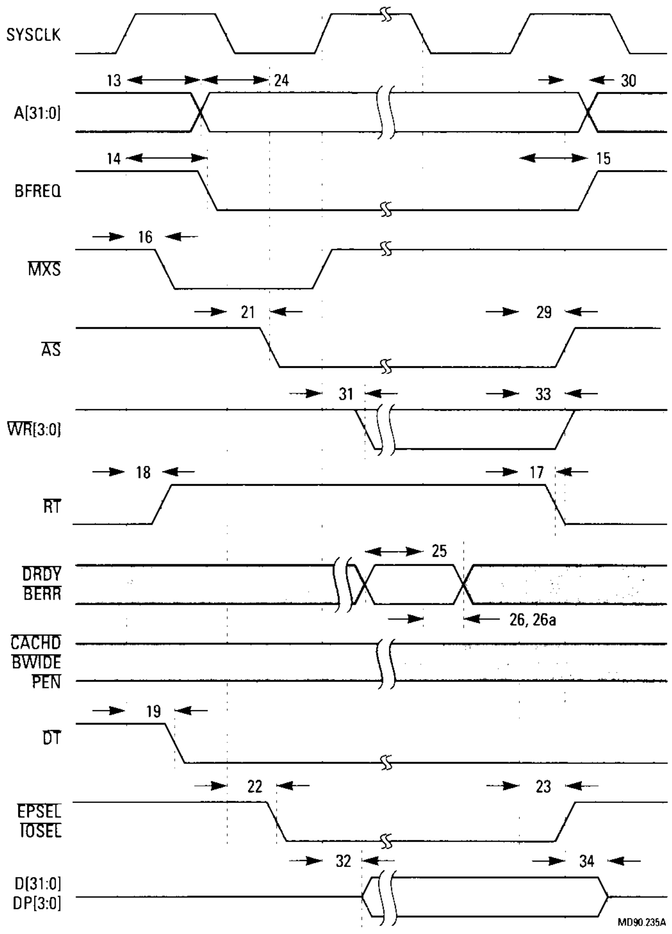


Figure 19. Write Transaction Timing

LSI LOGIC

Specifications
(Continued)

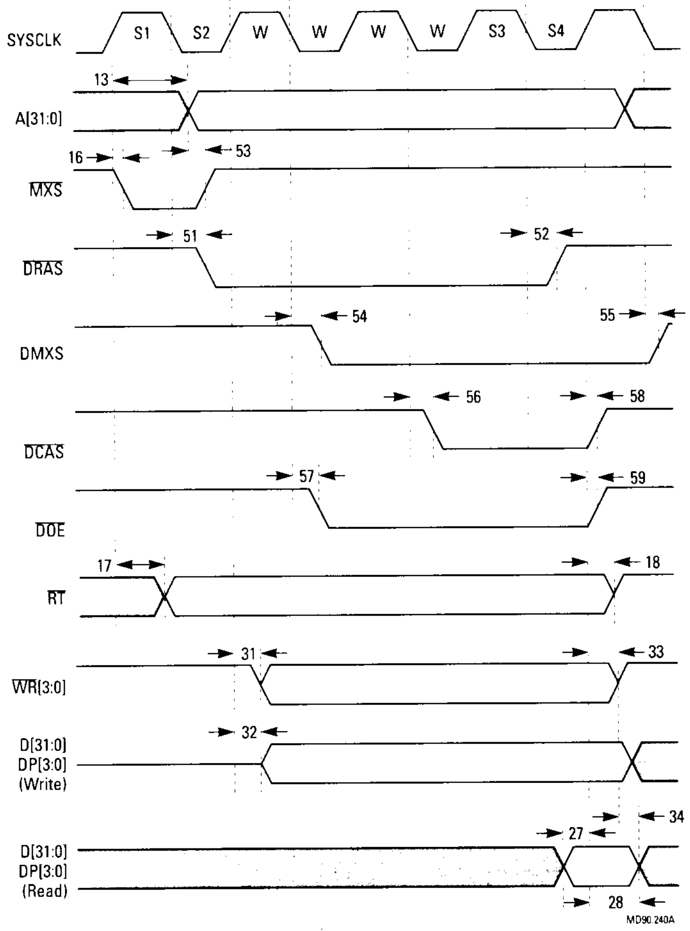


Figure 20. DRAM Access Timing

LSI LOGIC

Package and Ordering Information

The LR33050 is available in two different packages: a 155-pin ceramic pin grid array (CPGA) and a 160-pin metal quad flat package (MQUAD). Designers will choose the package that meets the cost and performance requirements of their application.

Table 14 lists and describes the two packages by order number.

This section contains two types of information for each package type: a pinout and a mechanical drawing. For the 155-pin CPGA, Figure 21 and Figure 22 contain the two types of information. For the 160-pin MQUAD, Figure 23 and Figure 24 contain the two types of information.

Table 14. LR33050 Ordering Information

Order Number	Clock Frequency (MHz)	Package Type	Operating Range
LR33050MC-25	25	160-pin MQUAD	Commercial
LR33050MC-33	33	160-pin MQUAD	Commercial
LR33050HC-25	25	155-pin CPGA	Commercial
LR33050HC-33	33	155-pin CPGA	Commercial
LR33050HC-40	40	155-pin CPGA	Commercial

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A		VSS	D7	D10	VSS	D14	D17	VSS	VDD	D22	D25	D28	D31	$\overline{T2TO}$	VDD	VSS
B	VDD	D6	*	D9	D12	VDD	D16	D19	D20	D23	D26	D29	$\overline{CACHD}$	VSS	*	VDD
C	D4	D5	*	D8	D11	D13	D15	D18	D21	D24	D27	D30	$\overline{PEN}$	*	PERR	$\overline{RTO}$
D	D1	D2	D3											DP0	DP1	DP2
E	*	$\overline{DMAR}$	D0											DP3	BFREQ	$\overline{DOE}$
F	CPC2	CPC3	TDONE											$\overline{RD}$	$\overline{RT}$	$\overline{BREQ}$
G	BENDN	CPC0	CPC1											BGNT	$\overline{MXS}$	$\overline{DRDY}$
H	VSS	D$TST	I$TST											$\overline{AS}$	$\overline{BFTCH}$	VSS
J	VDD	SYSCLK	*											$\overline{DCAS}$	$\overline{DRAS}$	VDD
K	$\overline{STALL}$	$\overline{DT}$	RTACK											$\overline{WR0}$	$\overline{BERR}$	DMXS
L	T2EN	$\overline{EPSEL}$	$\overline{IOSEL}$											$\overline{WR3}$	$\overline{WR2}$	$\overline{WR1}$
M	$\overline{DMAC}$	$\overline{HIGHZ}$	$\overline{RESET}$											A0	*	$\overline{BWIDE}$
N	INT5	INT4	INT2											A3	A2	A1
P	INT3	$\overline{FRCM}$	*	INT1	A29	A26	A23	A21	BRTKN	A16	A13	VDD	A10	A6	A5	A4
R	VSS	*	*	A31	A28	A25	A22	A20	A19	A17	A14	THIT	A11	A8	*	VSS
T	VDD	VSS	INT0	A30	A27	A24	VSS	VDD	VSS	A18	A15	A12	VSS	A9	A7	VDD

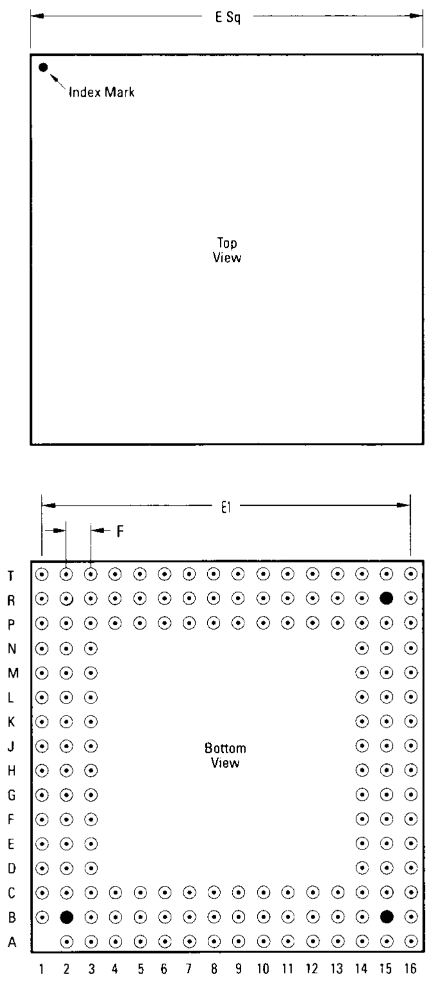
Top View

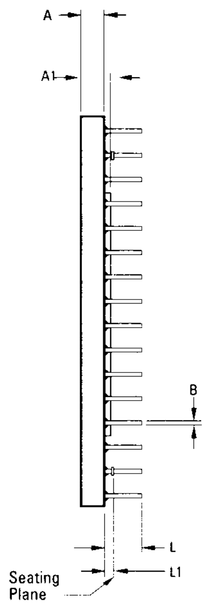
MD91.35A

Note(s):
1. Pins indicated with an asterisk (*) are reserved – do not connect.

Figure 21. 155-Pin CPGA Pinout

Package and Ordering Information
(Continued)



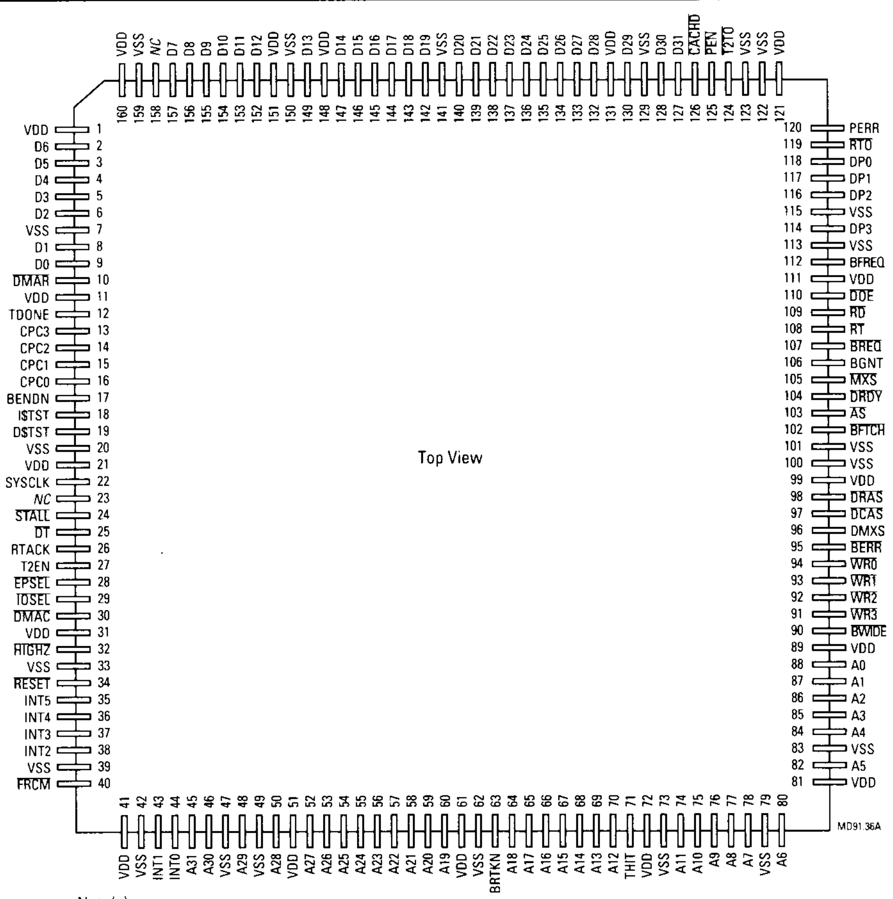


Dimen-sions		Inches (mm)	Dimen-sions		Inches (mm)
A	Min	0.091 (2.311)	E1	Ref	1.500 (38.10)
	Max	0.111 (2.819)	F	Typ	0.100 (2.540)
A1	Max	0.136 (3.454)	L	Min	0.185 (4.699)
B	Min	0.016 (0.406)		Max	0.195 (4.953)
	Max	0.020 (0.508)	L1	Min	0.065 (1.651)
E	Min	1.643 (41.73)		Max	0.075 (1.905)
	Max	1.677 (42.60)			

MD90.251

Figure 22. 155-Pin CPGA Mechanical Drawing

Package and Ordering Information
(Continued)

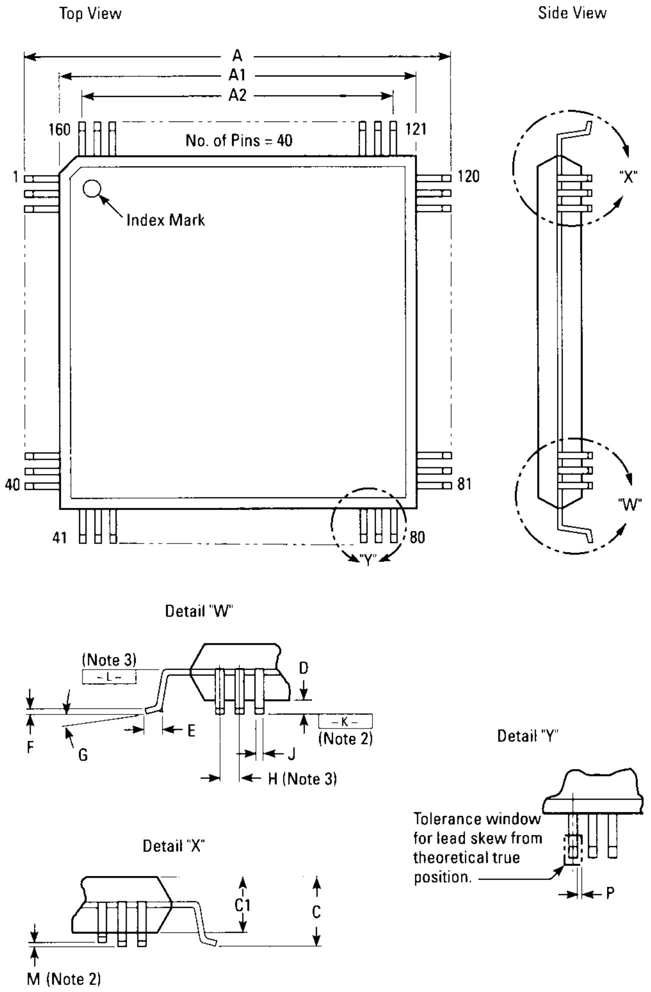


Note(s):
1. Pins indicated with *NC* are not connected – do not connect.

Figure 23. 160-Pin MQUAD Pinout

LR33050 MIPS IFX Processor
Preliminary

Package and Ordering Information
(Continued)



Dimension		Millimeters (Inches)
A	Min	31.00 Sq (1.220)
	Max	31.40 Sq (1.236)
A1	Min	27.56 Sq (1.085)
	Max	27.72 Sq (1.091)
A2	Ref	25.35 Sq (0.998)
C	Max	3.86 (0.152)
C1	Max	3.43 (0.135)
D	Max	0.51 (0.020)
E	Min	0.61 (0.024)
	Max	1.00 (0.039)
F	Min	0.10 (0.004)
	Max	0.25 (0.010)
G	Min	0 degrees
	Max	10 degrees
H	Nom	0.65 (0.026)
J	Min	0.25 (0.010)
	Max	0.35 (0.014)
M	Max	0.01 (0.004)
P	Max	0.05 (0.002)

Notes:
1. Controlling dimension – mm.
2. Coplanarity of all leads shall be within .010 mm (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.

MD91 505

Figure 24. 160-Pin MQUAD Mechanical Drawing

LSI LOGIC

Sales Offices and Design Resource Centers

LSI Logic Corporation Headquarters
Milpitas CA
■ **408.433.8000**
FAX: 408.434.6457

Alabama
■ 205.883.3527
FAX: 205.883.3529

Arizona
602.951.4560
FAX: 602.951.4580

California
San Jose
■ 408.954.1561
FAX: 408.954.1565

Irvine
■ 714.553.5600
FAX: 714.474.8101

San Diego
619.689.7140
FAX: 619.689.7145

Encino
■ 818.379.2400
FAX: 818.783.5548

Colorado
303.399.1112
FAX: 303.399.6558

Florida
Melbourne
407.728.9481
FAX: 407.728.9587

Boca Raton
■ 407.395.6200
FAX: 407.394.2865

Illinois
■ 708.995.1600
FAX: 708.995.1622

Maryland
Bethesda
■ 301.897.5800
FAX: 301.897.8389

Columbia
301.740.5664
FAX: 301.740.5603

Massachusetts
■ 617.890.0180
FAX: 617.890.6158

Minnesota
■ 612.921.8300
FAX: 612.921.8399

New Jersey
■ 908.549.4500
FAX: 908.549.4802

New York
Hopewell Junction
914.226.1620
FAX: 914.226.1351

Victor
716.223.8820
FAX: 716.223.8822

Camillus
315.468.1646
FAX: 315.488.2947

North Carolina
■ 919.783.8833
FAX: 919.783.8909

Ohio
513.438.2821
FAX: 513.438.2317

Oregon
503.645.9882
FAX: 503.645.6612

Pennsylvania
215.830.1404
FAX: 215.657.4920

Texas
Austin
512.329.1044
FAX: 512.327.1274

Dallas
■ 214.788.2966
FAX: 214.233.9234

Washington
■ 206.822.4384
FAX: 206.827.2884

LSI Logic Corporation of Canada Inc. Headquarters
Calgary
■ 403.262.9292
FAX: 403.262.9494

Burnaby
■ 604.294.8444
FAX: 604.294.8443

Edmonton
■ 403.450.4400
FAX: 403.450.4411

Kanata
■ 613.592.1263
FAX: 613.592.3253

Montreal
■ 514.694.2417
FAX: 514.694.2699

Toronto
■ 416.620.7400
FAX: 416.620.5005

Denmark
EV Johanssen Electronic AS
■ 45.31.839022
FAX: 45.31.839222

Finland
Oy Fintronic AB
358.0.6926022
FAX: 358.0.6821251

France
LSI Logic S.A.
■ 33.146.206600
FAX: 33.146.203138

Microel S.A.
33.1.69070824
FAX: 33.1.69071723

Germany
LSI Logic GmbH European Headquarters
Munich
49.89.99313100
FAX: 49.89.936806

LSI Logic GmbH Headquarters
Munich
■ 49.89.9269030
FAX: 49.89.917096

Dusseldorf
■ 49.211.5961066
FAX: 49.211.592130

Stuttgart
■ 49.711.139690
FAX: 49.711.8661428

AE Advanced Electronics
Bad Camberg
■ 49.6434.5041
FAX: 49.6434.4277

Advanced Logic GbmH
Magdeburg
37.91.48112
FAX: 37.91.48112

AE Advanced Electronics
Munich
■ 49.89.9935580
FAX: 49.89.99355899

AE Advanced Electronics
Stuttgart
49.711.881118
FAX: 49.711.8661359

Israel
LSI Logic Limited
■ 972.3.5403741
FAX: 972.3.5403747

Italy
LSI Logic SPA
■ 39.39.6056881
FAX: 39.39.6057867

Japan
LSI Logic K.K. Headquarters
Tokyo
■ 81.33.589.2711
FAX: 81.33.589.2740

Tokyo
81.33.5275.1731
FAX: 81.33.5275.1739

Tsukuba-Shi
■ 81.298.52.8371
FAX: 81.298.52.8376

Osaka
■ 81.6.947.5281
FAX: 81.6.947.5287

Yokohama
■ 81.45.902.4111
FAX: 81.45.902.4533

LSI Logic Corporation of Korea Limited
■ 82.2.561.2921
FAX: 82.2.554.9327

Netherlands
LSI Logic/Arcobel
■ 31.4120.30335
FAX: 31.4120.30635

Norway
Art Chip AS
47.2.720740
FAX: 47.2.656960

Scotland
LSI Logic Limited
■ 44.506.416767
FAX: 44.506.414836

Spain
LSI Logic S.A.
34.1.5705600
FAX: 34.1.5702807

Sweden
LSI Logic Limited
■ 46.8.703.4680
FAX: 46.8.7506647

Switzerland
LSI Logic Sulzer AG
■ 41.32.536363
FAX: 41.32.536367

Taiwan
LSI Logic Corporation
■ 886.2.755.3433
FAX: 886.2.755.5176

United Kingdom
LSI Logic Limited
Bracknell
■ 44.344.426544
FAX: 44.344.481039

Manhattan Skyline Limited
Maidenhead
44.628.75851
FAX: 44.628.782812

■ 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 on Recycled Paper
Printed in USA
012.7111.5538.003.6K.TP.WCP

032890 K