The MIPS64™ 5Kf™ processor core from MIPS Technologies is a synthesizable, highly-integrated 64-bit MIPS® RISC microprocessor core designed for high-performance, low-power, low-cost embedded applications. To semiconductor manufacturing companies and system OEMs who are building complex System-On-Chip ASIC devices, the 5Kf core offers the long-awaited benefits of an easy-to-integrate, synthesizable core that provides 64-bit address and data paths along with the 64-bit computing power of an R5000®-class processor. The 5Kf core is portable across processes, is highly configurable, and is easily integrated into standard design flows, thereby reducing time to market and allowing designers to focus their attention on end-user products. The 5Kf core is ideally positioned to support new products for emerging segments of the digital consumer, network, and office automation markets where floating point performance is required. The power-management features of the 5Kf core make it ideally suited for use in battery-powered applications.

The 5Kf core implements the MIPS64 Architecture. It contains special multiply-accumulate, conditional move, prefetch, wait, leading zero/one detect instructions, and the 64-bit privileged resource architecture. The 5Kf core also features a high performances IEEE 754 compliant Floating Point Unit (FPU). The FPU supports both single and double precision instructions. It includes the multiply add instruction, which can issue every cycle, whereby both a multiply and an add single precision operation can be performed in every cycle. The 5Kf core can dual issue a floating point arithmetic instruction with a floating point load/store or integer instruction, whereby two instructions can be executed every cycle in floating point applications. A coprocessor interface is also provided, which allows designers a way to easily extend their architectures by addition of custom functionality, such as network, or graphics coprocessors.

The memory management unit contains a configurable 16, 32, or 48 dual-entry Joint TLB (JTLB) with variable page sizes, a 4-entry Instruction micro TLB (ITLB), and a 4-entry Data micro TLB (DTLB). Using a TLB with the 5Kf core is optional. The alternative is to use a far simpler Fixed Mapping Translation (FMT) scheme.

Optional instruction and data caches are fully configurable from 0 - 64 KBytes in size, with a maximum size of 16 KBytes/ way in a 4-way set associative implementation. In addition, each cache can be organized as direct-mapped, 2-way, 3-way, or 4-way set associative. The 5Kf core supports an instruction scheduling mechanism that eliminates pipeline stalls on cache misses, and a load scheduling slot is also supported.

To ease software debugging, the EJTAG debug solution in the 5Kf core includes instruction software breakpoints, a single-step feature, and a dedicated Debug Mode. Optional hardware breakpoints include 4 instruction and 2 data breakpoints. An optional Test Access Port (TAP) forms the interface to an external debug host and provides a dedicated communication channel for debugging of an embedded system.

Figure 1 shows a block diagram of the 5Kf core. The core is divided into *required* and *optional* blocks as shown.

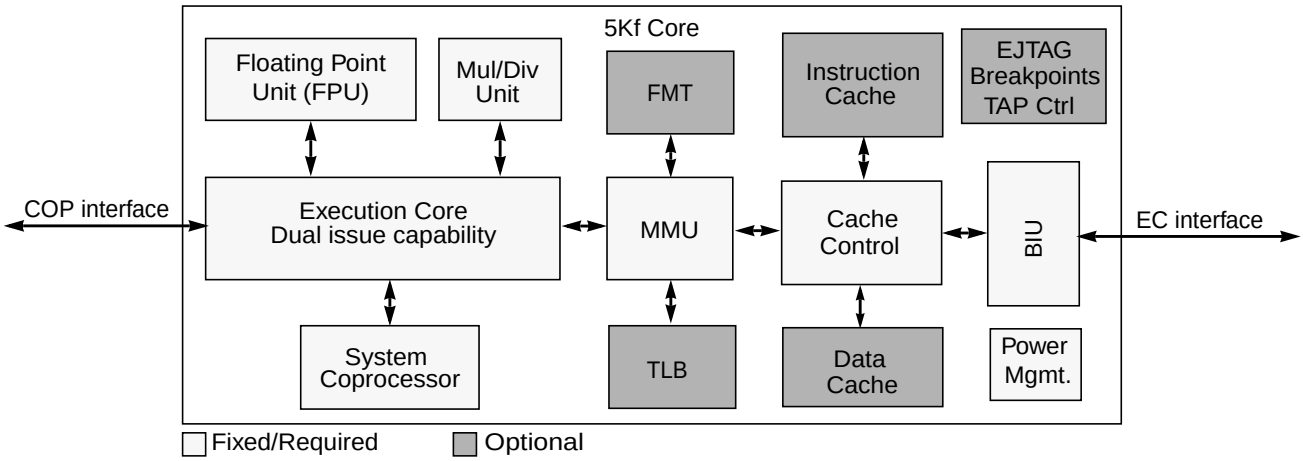

Figure 1 5Kf Core Block Diagram

Features

- 64-bit Data and Address Path
  (42-bit virtual and 36-bit physical address space)
- MIPS64 Compatible Instruction Set
  - Based on MIPS V™ Instruction Set Architecture
  - Multiply-Accumulate and Multiply-Subtract Instructions (MADD, MADDU, MSUB, MSUBU)
  - Targeted Multiply Instruction (MUL)
  - Zero/One Detect Instructions (CLZ, CLO, DLCO, DCLZ)
  - Wait Instruction (WAIT) for low power control
  - Conditional Move Instructions (MOVZ, MOVN)
  - Prefetch Instructions (PREF, PREFX)
- Dual-issue Floating Point Unit / Coprocessor 1
  - Fully pipelined IEEE 754 compliant floating point unit with both single and double precision instructions
  - Includes multiply add instruction
  - Maximum issue rate of one multiply add single (MADD.S) instruction every clock
  - Maximum issue rate of one multiply add double (MADD.D) instruction every other clock
  - FPU executes independently of integer pipeline
  - Fast flush-to-zero mode to optimize performance
- Dual-issue superscalar micro-architecture capable of executing:
  - 1 integer and 1 arithmetic floating point instruction
  - 1 floating point arithmetic and 1 floating point load/store instruction
- General Purpose Coprocessor Interface
  - Supports all MIPS V instructions
  - Supports COP2 coprocessors
  - Utilizes high-performance features of the integer unit
  - Dual-issue capability as for floating point instructions
- Multiply/Divide Unit
  - Maximum issue rate of one 32x16 multiply per clock
  - Maximum issue rate of one 32x32 multiply every other clock
  - Maximum issue rate of one 64x64 multiply every 9 clocks
  - 37 clock latency on 32/32 divides
  - 69 clock latency on 64/64 divide
  - Early-in feature for divides allows results sooner for smaller dividend values
- MIPS64 privileged resource architecture
  - Count/Compare registers for real-time timer interrupts
  - Instruction and Data watch registers for software breakpoints
  - Separate interrupt exception vector
  - Supervisor Mode operation
  - Performance Monitoring logic for analyzing application speed
- Memory Management Unit
  - 16, 32, or 48 dual-entry JTLB with variable page sizes or a simple Fixed Mapping Translation (FMT) mechanism (optional)
  - 4-entry instruction micro TLB
  - 4-entry data micro TLB
  - Support for 8-bit ASID
  - Support for 4 KB - 16 MB page sizes
- Programmable Cache Sizes
  - Individually configurable instruction and data caches
  - Sizes from 0 - 16 KBytes/way (64 KBytes maximum)
  - Direct Mapped, 2-, 3-, or 4-Way Set Associative
  - Non-blocking loads
  - 32-byte cache line size, doubleword sectored
  - Virtually indexed, physically tagged
  - Support for locking cache lines
  - Non-blocking prefetches
  - Optional parity protection
- Simple Bus Interface Unit (BIU)
  - All I/Os fully registered
  - Separate, unidirectional 36-bit address and 64-bit data buses
  - 32-byte write buffer (4 doublewords)
  - 1-line (32-byte) eviction buffer
- Power Control
  - Minimum frequency: 0 MHz
  - Power-down mode (triggered by WAIT instruction)
  - Support for software controlled clock divider
  - Sleep mode: During this mode the clocks are shut off. Sleep mode is entered automatically from power-down mode after all bus activity stops.
- EJTAG Debug Support
  - Software Debug Breakpoint Instruction (SDBBP)
  - Single-step feature
  - Debug Mode
  - Optional hardware breakpoints (4 instruction and 2 data breakpoints)
  - Optional Test Access Port (TAP) interface to debug host, including fast data download/upload feature
- Testability for Production Test:
  - Muxed-FF fullscan design with configurable number of scan chains. ATPG test coverage can exceed 99% (library and configuration dependent).
  - Optional memory BIST, either through integrated memory test (March C+ or IFA-13 algorithm) or by use of industry standard memory BIST CAD tools.

Architectural Overview

The 5Kf core contains both required and optional blocks. Optional blocks can be added to the 5Kf core based on the needs of the implementation. The required blocks are as follows:

- Execution Unit
- Floating Point Unit (FPU)
- Multiply/Divide Unit (MDU)
- System Control Coprocessor (CP0)
- Memory Management Unit (MMU)
- Translation Lookaside Buffer (TLB) or Fixed Mapping Translation (FMT)
- Cache Controllers
- Bus Interface Unit (BIU)
- Basic EJTAG debug features
- Power Management

Optional blocks include:

- Instruction Cache
- Data Cache
- EJTAG Debug Test Access Port (TAP)
- EJTAG Hardware Breakpoints
- Memory BIST module

The section entitled "5Kf Core Required Logic Blocks" on page 4 discusses the required blocks. The section entitled "5Kf Core Optional Logic Blocks" on page 16 discusses the optional blocks.

Pipeline Flow

The 5Kf core implements a high-performance 6-stage pipeline:

- Instruction fetch (I stage)
- Dispatch (D stage)
- Register read (R stage)
- Execution (E stage)
- Memory access (M stage)
- Writeback (W stage)

The 5Kf core implements a bypass mechanism that allows the result of an operation to be forwarded directly to the instruction that needs it without having to write the result to the register and then read it back.

Figure 2 shows a timing diagram of the 5Kf core pipeline.

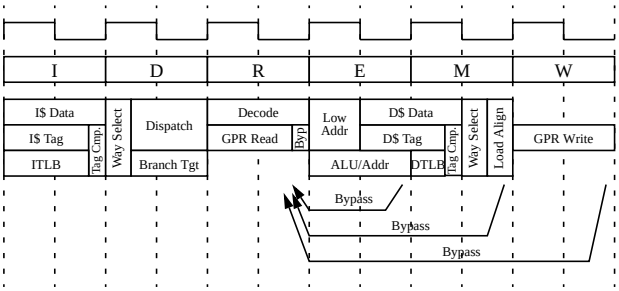

Figure 2 5Kf Core Pipeline

I Stage: Instruction Fetch

During the Instruction Fetch stage:

- The Translation Lookaside Buffer (TLB) or the Fixed Mapping Translation (FMT) performs the virtual-to-physical address translation for instruction fetch addresses.
- An instruction is fetched from instruction cache.

D Stage: Dispatch

During the Dispatch stage:

- Branch decode and prediction is performed.
- An instruction is dispatched to the coprocessor/integer unit.

R Stage: Register Read

During the Register Read stage:

- The General Purpose Register (GPR) file is read.
- The instruction is decoded.

E Stage: Execution

During the Execution stage:

- The Arithmetic Logic Unit (ALU) computes the arithmetic or logical operation for register-to-register instructions.
- The ALU determines whether the branch condition is true.

- All multiply and divide operations begin.
- The ALU calculates the full virtual address for load and store instructions.
- The cache look-up starts for loads and stores.

M Stage: Memory Access

During the memory access stage:

- The Data Translation Lookaside Buffer (DTLB) or the Fixed Mapping Translation (FMT) performs the virtual-to-physical address translation for data load/store addresses.
- The data cache lookup completes.
- Load data is aligned.

W Stage: Writeback

During the writeback stage:

- For register-to-register or load instructions, the instruction result is written back to the register file.

Modes of Operation

The 5Kf core supports four modes of operation: User Mode, Supervisor Mode, Kernel Mode, and Debug Mode. User Mode is most often used for applications programs.

Kernel and Supervisor Modes are typically used for handling exceptions and operating system functions, including CP0 management and I/O device accesses. Debug Mode is used for EJTAG software debugging and is similar to Kernel Mode, but also allows programming of debug resources and has special handling of exceptions and other debug related issues.

The processor enters Kernel Mode both at reset and when an exception is taken. While in Kernel Mode, software has access to the entire address space as well as all CP0 registers. User Mode accesses are limited to a subset of the virtual address space and can be inhibited from accessing CP0 functions.

5Kf Core Required Logic Blocks

The 5Kf core consists of the following required logic blocks as shown in Figure 1. These logic blocks are defined in the following subsections:

- Execution Unit
- Floating Point Unit (FPU) / Coprocessor 1
- Multiply/Divide Unit (MDU)
- System Control Coprocessor (CP0)
- Cache Controllers
- Memory Management Unit (MMU)
- Translation Lookaside Buffer (TLB) or Fixed Mapping Translation (FMT)
- Bus Interface Control (BIU)
- Basic EJTAG debug features
- Power Management

Execution Unit

The 5Kf core execution unit implements a load/store architecture with single-cycle ALU operations (logical, shift, add, subtract). The 5Kf core contains thirty-two 64-bit general-purpose registers used for integer operations and address calculation. The register file consists of two read ports and two write ports and is fully bypassed to minimize operation latency in the pipeline.

The execution unit includes:

- 64-bit adder used for calculating arithmetic results and the data addresses
- Program counter the next instruction address
- Logic for branch determination and branch target address calculation
- Load and store aligner
- Bypass multiplexers used to avoid stalls when executing instructions streams where data producing instructions are followed closely by consumers of their results.
- Instruction buffer that eliminates penalties to the pipeline when branches are predicted correctly, and reduces the penalty to one pipeline bubble when a branch is mispredicted.
- Zero/One detect unit for implementing the CLZ, DCLZ, CLO, and DCLO instructions.
- Logic unit for performing bitwise logical operations

Floating Point Unit (FPU) / Coprocessor 1

The 5Kf core Floating Point Unit (FPU) implements the MIPS64 ISA (Instruction Set Architecture) for floating-point computation. The implementation supports the ANSI/IEEE Standard 754 (IEEE Standard for Binary Floating-Point Arithmetic) for single and double precision data formats. The FPU contains thirty-two 64-bit floating-point registers used for floating point operations.

The performance is optimized for single precision formats. Most instructions have a 1 cycle throughput and 4 cycle latency. The FPU can dual issue arithmetic and load/store instructions, whereby arithmetic operations can operate continuously, while data is provided and retrieved.

The FPU implements the MIPS64 multiply-add (MADD) and multiply-sub (MSUB) instructions with intermediate rounding after the multiply function. The result is guaranteed to be the same as executing a MUL and an ADD instruction separately, but the instruction latency, instruction fetch, dispatch bandwidth, and the total number of register accesses are improved.

IEEE denormalized input operands and results are supported by hardware for some instructions. IEEE denormalized results are not supported by hardware in general, but a fast flush-to-zero mode is provided to optimize performance. The fast flush-to-zero mode is enabled through the FCCR register, and use of this mode is recommended for best performance when denormalized results are generated.

The FPU has a separate pipeline for floating point instruction execution. This pipeline operates in parallel with the integer unit (IU) pipeline and does not stall when the IU pipeline stalls. This allows long-running FPU operations, such as divides or square root, to be partially masked by system stalls and/or other integer unit instructions. Arithmetic instructions are always dispatched and completed in order, but loads and stores can complete out of order. The exception model is ‘precise’ at all times. The FPU is also denoted coprocessor 1.

FPU Pipeline

The FPU implements a high-performance 7-stage pipeline:

- Decode, register read and unpack (FR stage)
- Multiply tree - double pumped for double (M1 stage)
- Multiply complete (M2 stage)
- Addition first step (A1 stage)
- Addition second and final step (A2 stage)
- Packing to IEEE format (FP stage)
- Register writeback (FW stage)

The FPU implements a bypass mechanism that allows the result of an operation to be forwarded directly to the instruction that needs it without having to write the result to the FPU register and then read it back.

Figure 3 shows the FPU pipeline with dispatch from the integer pipeline.

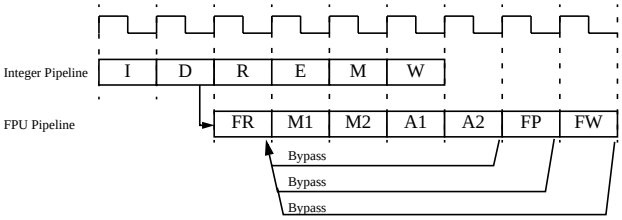

Figure 3 FPU Pipeline

FPU Instruction Latencies and Repeat Rates

Table 1 contains the floating point instruction latencies and repeat rates for the common cases. In this table ‘Latency’ refers to the number of cycles necessary for the first instruction to produce the result needed by the second instruction. The ‘Repeat Rate’ refers to the maximum rate at which an instruction can be executed.

Table 1 5Kf Core FPU Latency and Repeat Rate

Opcode*	Latency (cycles)	Repeat Rate (cycles)
ABS.[S,D], NEG.[S,D], ADD.[S,D], SUB.[S,D], MUL.S, MADD.S, MSUB.S, NMADD.S, NMSUB.S	4	1
MUL.D, MADD.D, MSUB.D, NMADD.D, NMSUB.D	5	2
RECIP.S	13	10
RECIP.D	25	21
RSQRT.S	17	14
RSQRT.D	35	31
DIV.S, SQRT.S	17	14
DIV.D, SQRT.D	32	29
C.cond.[S,D] to FPU inst. / other inst.	1 / 2	1
CVT.D.S, CVT.[S,D].[W,L]	4	1
* Format: S = Single, D = Double, W = Word, L = Longword		

Table 1 5Kf Core FPU Latency and Repeat Rate

Opcode*	Latency (cycles)	Repeat Rate (cycles)
CVT.S.D	6	1
CVT.[W,L].[S,D], CEIL.[W,L].[S,D], FLOOR.[W,L].[S,D], ROUND.[W,L].[S,D], TRUNC.[W,L].[S,D]	5	1
MOV.[S,D], MOVF.[S,D], MOVN.[S,D], MOVT.[S,D], MOVZ.[S,D]	4	1
LWC1, LDC1, LDXC1, LUXC1, LWXC1	3	1
MTC1, DMTC1, MFC1, DMFC1	2	1
* Format: S = Single, D = Double, W = Word, L = Longword		

FPU Control Registers

The FPU contains a number of control register, and these are listed in Table 2

Table 2 Coprocessor 1 Registers in Numerical Order

Register Number	Register Name	Function
0	FIR	Floating Point Implementation Register. Identifies the capabilities of the floating point unit.
25	FCCR	Floating Point Condition Codes Register. Alternate way of reading the FP condition codes in the FCSR.
26	FEXR	Floating Point Exceptions Register. Alternate way of reading the exception condition codes in the FCSR.
28	FENR	Floating Point Enables Register. Alternate way of reading the Enables field in the FCSR.
31	FCSR	Floating Point Control and Status Register.

Multiply/Divide Unit (MDU)

The 5Kf core contains a Multiply/Divide Unit (MDU) with a separate pipeline for multiply and divide operations. This pipeline operates in parallel with the Integer Unit (IU) pipeline and does not stall when the IU pipeline stalls. This allows long-running MDU operations, such as divides, to be partially masked by system stalls and/or other integer unit instructions.

The MDU consists of a 32x16 booth recoded multiplier, result/accumulation registers (HI and LO), a divide state machine, and all necessary multiplexers and control logic. The first number shown (‘32’ of 32x16) represents the *rs* operand. The second number (‘16’ of 32x16) represents the *rt* operand. The 5Kf core only checks the value of the latter (*rt)* operand to determine how many times the operation must pass through the multiplier. The 16x16 and 32x16 operations pass through the multiplier once, allowing for a multiply operation every clock. A 32x32 operation passes through the multiplier twice, allowing for a multiply operation every other clock. A 64x64 operation passes through the multiplier nine times, allowing for a multiply operation every nine clocks.

Appropriate interlocks are implemented to stall the issue of back-to-back 32x32 and 64x64 multiply operations. Multiply operand size is automatically determined by logic built into the MDU.

Divide operations are implemented with a simple 1 bit per clock iterative algorithm. A 32-bit divide requires 37 clock cycles to complete, while a 64-bit divide requires 69 clock cycles. Any attempt to issue a subsequent MDU instruction while a divide is still active causes an IU pipeline stall until the divide operation is completed.

However, the divider has an early-in feature which detects the size of the dividend in 8-bit increments. When a smaller dividend is detected, the algorithm reduces the number of iterations accordingly.

Table 3 lists the latencies (number of cycles until a result is available) for the 5Kf core multiply and divide instructions.

Table 3 5Kf Core Integer Multiply/Divide Unit Latencies

Opcode	Operand Size	Latency (cycles)
MULT/MULTU, MADD/MADDU, MSUB/MSUBU, DMULT/DMULTU	16 bit	1
	32 bit	2
	64 bit	9
MUL	16 bit	2
	32 bit	3

Table 3 5Kf Core Integer Multiply/Divide Unit Latencies

Opcode	Operand Size	Latency (cycles)
DIV/DIVU, DDIV/DDIVU	8 bit	11
	16 bit	19
	24 bit	27
	32 bit	35
DDIV/DDIVU	40 bit	43
	48 bit	51
	56 bit	59
	64 bit	67

The MIPS architecture defines that the results of a multiply or divide operation be placed in the HI and LO registers. Using the move-from-HI (MFHI) and move-from-LO (MFLO) instructions, these values can be transferred to the general purpose register file.

The 5Kf core implements an additional multiply instruction, MUL, which specifies that multiply results be placed in the general purpose register file instead of the HI/LO register pair. This instruction avoids the explicit MFLO instruction, normally required in order to use the results of multiply operations.

Two other instructions, multiply-add (MADD) and multiply-subtract (MSUB), are used to perform multiply-accumulate operations. The MADD instruction multiplies two numbers and then adds the product to the current contents of the HI and LO registers. Similarly, the MSUB instruction multiplies two operands and then subtracts the product from the HI and LO registers. The MADD and MSUB operations are commonly used in DSP algorithms.

The DMULT/DMULTU and DDIV/DDIVU instructions are used to support 64-bit operands.

Exception Logic

The Exception block contains the logic for identifying and managing exceptions. Exceptions can be caused by a variety of sources, including boundary cases in data, TLB misses, external events, or program errors.

Table 4 5Kf Core Exception Types

Exception	Description
Reset	Assertion of SI_ColdReset signal.
Soft Reset	Assertion of SI_Reset signal.
DSS	Debug Single Step.
DINT	Debug Interrupt.
DDBLImpr	Debug Data Break on Load Imprecise.
NMI	Assertion of EB_NMI signal.
Cache Error - Data Access	A cache error occurred on a load or store data reference (imprecise).
Machine Check	TLB write that conflicts with an existing entry.
DBE	Load or store bus error.
Interrupt	Assertion of unmasked HW or SW interrupt signal.
Deferred Watch	Deferred Watch.
DIB	Debug Instruction Hardware Break.
Watch - Instruction Fetch	A watch address match was detected on an instruction fetch.
AdEL	Instruction fetch address alignment error. Instruction fetch reference to protected address.
TLB Refill - Instruction Fetch	Instruction Fetch TLB miss.
TLB Invalid - Instruction Fetch	The valid bit was zero in the TLB entry matching the address referenced by a load or store instruction.
Cache Error - Instruction Fetch	A cache error occurred on an instruction fetch.
IBE	Instruction fetch bus error.
SDBBP	Software Debug Breakpoint. Execution of the SDBBP instruction.

Table 4 5Kf Core Exception Types (Continued)

Exception	Description
Execution Exceptions	CpU, MDMX: Execution of a coprocessor instruction for a coprocessor that is not enabled.
	RI: Execution of a Reserved Instruction.
	Execution of a 64-bit instruction causes a reserved instruction exception if executed in User Mode when PX and UX are both 0.
	Bp: Execution of BREAK instruction.
	SC: Execution of SYSCALL instruction.
	Ov: Execution of an arithmetic instruction that overflowed.
	Tr: Execution of a trap (when trap condition is true).
	FPE: Floating Point Exception
	C2E: COP2 Exception
DDBL / DDBS	Debug Data Break on Load (address only). Debug Data Break on Store (address only or address + data value).
Watch - Data Access	A reference to an address in one of the watch registers (data).
AdEL	Load Address Alignment Error. Load reference to protected address.
AdES	Store Address Alignment Error. Store to a protected address.
TLB Refill - Data Access	TLB miss occurred on a data access.
TLB Invalid - Data Access	The valid bit was zero in the TLB entry matching the address referenced by a load or store instruction.
TLB Modified - Data Access	The dirty bit was zero in the TLB entry matching the address referenced by a store instruction.
Cache Error - instruction cache	Cache error detected in the instruction cache by the CACHE instruction.

System Control Coprocessor (CP0)

In the MIPS architecture, CP0 is responsible for the virtual-to-physical address translation and cache protocols, the exception control system, the processor's diagnostics capability, and the operating modes (Kernel, Supervisor, User, and Debug). Configuration information such as cache size and set associativity is available by accessing the CP0 registers.

Table 5 Coprocessor 0 Registers

Register Number	Register Name	Function
0	Index[1]	Index into the TLB array.
1	Random[1]	Randomly generated index into the TLB array.
2	EntryLo0[1]	Low-order portion of the TLB entry for even-numbered virtual pages.
3	EntryLo1[1]	Low-order portion of the TLB entry for odd-numbered virtual pages.
4	Context[1]	Pointer to page table entry in memory.
5	PageMask[1]	Control for variable page size in TLB entries.
6	Wired[1]	Controls the number of fixed ("wired") TLB entries.
7	Reserved	Reserved.
8	BadVAddr	Reports the address for the most recent address-related exception.
9	Count	Processor cycle count.
10	EntryHi[1]	High-order portion of the TLB entry.
11	Compare	Timer interrupt control.
12	Status	Processor status and control.
13	Cause	Cause of last general exception.
14	EPC	Program counter at last exception.
15	PRId	Processor identification and revision.
16	Config/ Config1	Config register (Select = 0). Config register 1 (Select = 1).
17	Reserved	Reserved.
18	WatchLo	Low-order watchpoint address.
19	WatchHi	High-order watchpoint address.

Table 5 Coprocessor 0 Registers (Continued)

Register Number	Register Name	Function
20	XContext[1]	Extended Addressing Page Table Context.
21 - 22	Reserved	Reserved.
23	Debug	Debug control and exception status.
24	DEPC	Program counter at last debug exception.
25	PerfCount	Performance counter interface.
26	ErrCtl	Parity/ECC error control and status.
27	CacheErr	Cache parity error control and status.
28	TagLo/ DataLo	Low-order portion of cache tag interface (Select = 0). Low-order portion of cache data interface (Select = 1).
29	TagHi/ DataHi	High-order portion of cache tag interface (Select = 0). High-order portion of cache data interface (Select = 1).
30	ErrorEPC	Program counter at last error.
31	DESAVE	Debug handler scratch pad register.

1. Registers used only with a TLB-based MMU.

Cache Controllers

The 5Kf core instruction and data cache controllers support caches of various sizes, organizations, and set-associativity. For example, the data cache can be 8 KBytes in size and 2-way set associative, while the instruction cache can be 16 KBytes in size and 4-way set associative. Each cache can be accessed in a single processor cycle. In addition, each cache has its own 64-bit data path. Both caches can be accessed in the same pipeline clock cycle. Table 6 shows the cache options in the 5Kf.

The 5Kf supports the following cache protocols.

- Uncached (write around)
- Cacheable, noncoherent, write through, no write allocate
- Cacheable, noncoherent, write through, write allocate
- Cacheable, noncoherent, write-back (write allocate)
- Uncached accelerated

Refer to "5Kf Core Optional Logic Blocks" on page 16 for more information on instruction and data cache organization.

Table 6 5Kf Processor Cache Options

Cache Size (KBytes)	Associativity	Way Size (KBytes)	Number of Sets
0	NA	0	0
4	Direct Mapped	4	128
8	2-way	4	128
	Direct Mapped	8	256
12	3-way	4	128
16	4-way	4	128
	2-way	8	256
	Direct Mapped	16	512
24	3-way	8	256
32	4-way	8	256
	2-way	16	512
48	3-way	16	512
64	4-way	16	512

Memory Management Unit (MMU)

The 5Kf core contains a fully functional MMU that translates virtual addresses to physical addresses.

With support for 64-bit operations and address calculation, the MIPS64 architecture implicitly defines and provides support for a 64-bit virtual address space, sub-divided into four segments selected by bits 63:62 of the virtual address. To provide compatibility for 32-bit programs and MIPS32™ processors, a 2^{32}-byte Compatibility address space is defined, separated into two non-contiguous ranges in which the upper 32 bits of the 64-bit address are the sign extension of bit 31. The Compatibility address space is similarly sub-divided into segments selected by bits 31:29 of the virtual address.

Figure 4 shows the layout of the address spaces, including the Compatibility address space and the segmentation of each address space.

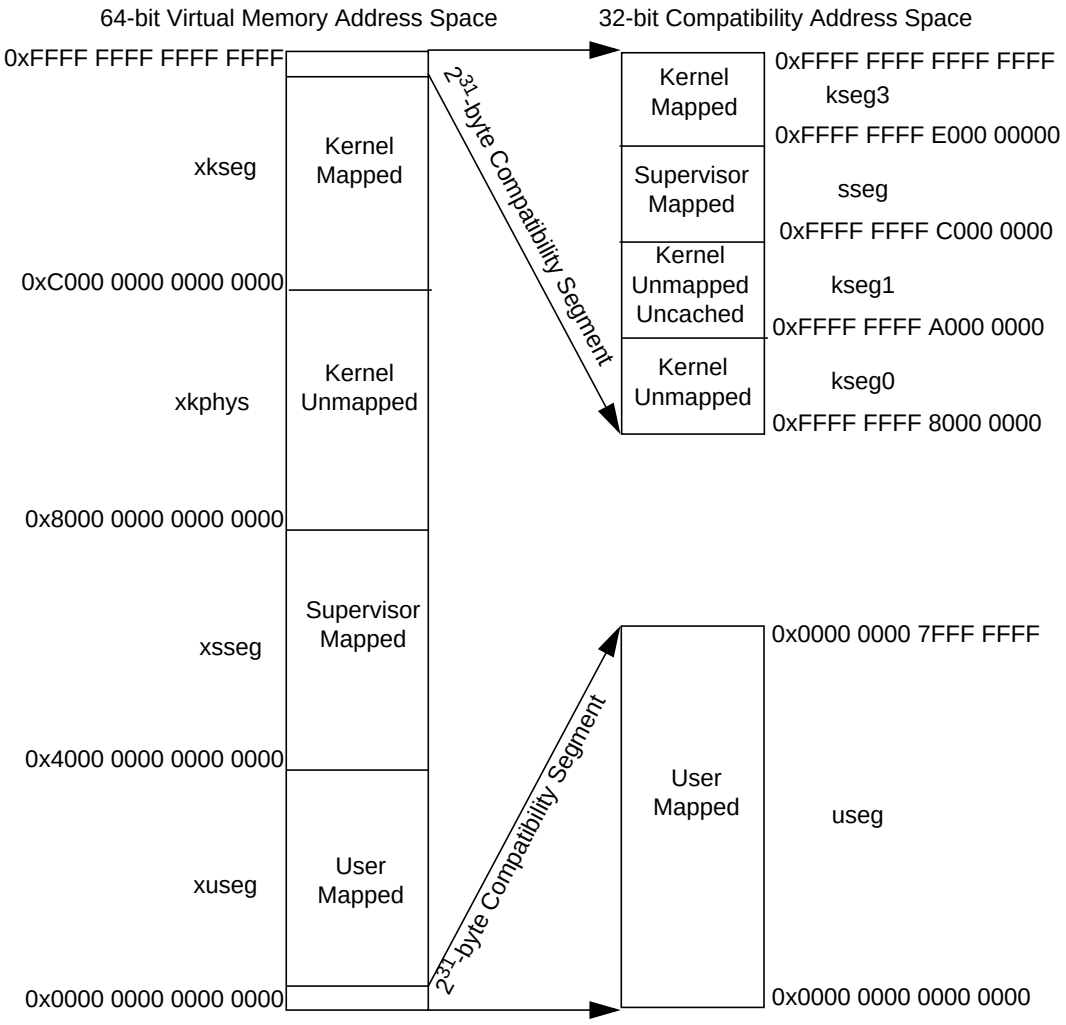

Figure 4 Virtual Address Spaces

Table 7 Virtual Memory Address Spaces

$VA_{63..62}$	Segment Name(s)	Maximum Address Range	Reference Legal from Mode(s)	Actual Segment Size
11_2	kseg3	0xFFFF FFFF FFFF FFFF through 0xFFFF FFFF E000 0000	Kernel, Debug	2^{29} bytes
	sseg ksseg	0xFFFF FFFF DFFF FFFF through 0xFFFF FFFF C000 0000	Supervisor, Kernel, Debug	2^{29} bytes
	kseg1	0xFFFF FFFF BFFF FFFF through 0xFFFF FFFF A000 0000	Kernel, Debug	2^{29} bytes
	kseg0	0xFFFF FFFF 9FFF FFFF through 0xFFFF FFFF 8000 0000	Kernel, Debug	2^{29} bytes
	xkseg	0xFFFF FFFF 7FFF FFFF through 0xC000 0000 0000 0000	Kernel, Debug	$(2^{40} - 2^{31})$ bytes
10_2	xkphys	0xBFFF FFFF FFFF FFFF through 0x8000 0000 0000 0000	Kernel, Debug	eight 2^{36} byte regions
01_2	xsseg xksseg	0x7FFF FFFF FFFF FFFF through 0x4000 0000 0000 0000	Supervisor, Kernel, Debug	2^{40} bytes
00_2	xuseg xsuseg xkuseg	0x3FFF FFFF FFFF FFFF through 0x0000 0000 8000 0000	User, Supervisor, Kernel, Debug	2^{40} bytes
	useg suseg kuseg	0x0000 0000 7FFF FFFF through 0x0000 0000 0000 0000	User, Supervisor, Kernel, Debug	2^{31} bytes

Each Segment of an Address Space is classified as “Mapped” or “Unmapped”. A “Mapped” address is one that is translated through the TLB or other memory management translation unit. An “Unmapped” address is one which is not translated through the TLB and which provides a window into the lowest portion of the physical address space, starting at physical address zero, and with a size corresponding to the size of the unmapped Segment.

Additionally, the kseg1 Segment is classified as “Uncached”. References to this Segment bypass all levels of the cache hierarchy and allow direct access to memory without any interference from the caches.

Table 7 lists some of the same information in tabular form as shown in Figure 4.

Translation Lookaside Buffers (TLB)

This and the following sections assumes a 5Kf core with the TLB option. Later sections deal with a 5Kf core with the FMT option.

The MMU consists of three translation lookaside buffers;

- 16, 32, or 48 dual-entry fully associative Joint TLB (JTLB)
- 4-entry fully associative Instruction TLB (ITLB)
- 4-entry fully associative Data TLB (DTLB)

When an instruction address is calculated, the virtual address is compared to the contents of the 4-entry ITLB. If the address is not found in the ITLB, the JTLB is accessed.

If the entry is found in the JTLB, that entry is then written into the ITLB. If the entry is not found in the JTLB, a TLB refill exception is taken.

When a load/store address is calculated, the virtual address is compared to the contents of the 4-entry DTLB. If the address is not found in the DTLB, the JTLB is accessed. If the entry is found in the JTLB, that entry is then written into the DTLB. If the entry is not found in the JTLB, a TLB refill exception is taken.

Figure 5 shows how the DTLB, ITLB, and JTLB are implemented in the 5Kf core.

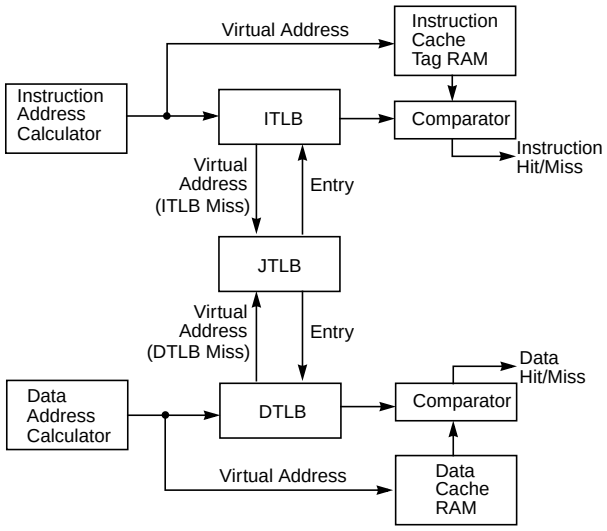

Figure 5 Address Translation During a Cache Access

Joint TLB

The 5Kf core implements a 16, 32, or 48 dual-entry, fully associative JTLB that maps 32, 64, or 96 virtual pages to their corresponding physical addresses. The JTLB is organized in pairs of even and odd entries containing pages that range in size from 4-KBytes to 16-MBytes. The purpose of the TLB is to translate virtual addresses and their corresponding ASID into a physical memory address. The translation is performed by comparing the upper bits of the virtual address (along with the ASID) against each of the entries in the *tag* portion of the joint TLB structure.

The JTLB is organized in page pairs to minimize the overall size. Each *tag* entry corresponds to 2-data entries, an even page entry and an odd page entry. The highest order virtual address bit not participating in the tag comparison is used to determine which of the data entries is used. Since page size can vary on a page-pair basis, the determination of which address bits participate in the comparison and which bit is used to make the even-odd determination is decided dynamically during the TLB lookup. The JTLB may be considered backup storage for the ITLB and DTLB.

Instruction TLB

The ITLB is a 4-entry, fully associative TLB dedicated to performing translations for the instruction stream. The ITLB only maps 4-KByte pages/sub-pages.

The ITLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing store for the ITLB. If an instruction fetch address cannot be translated by the ITLB, an ITLB miss is issued, and the JTLB is used to attempt to translate it in the following clock cycle. If successful, the translation information is copied into the ITLB for future use. A second ITLB lookup is performed, this time hitting in the ITLB. This ITLB miss sequence has a penalty of two extra clock cycles.

Data TLB

The DTLB is a small 4-entry, fully associative TLB dedicated to performing translations for the data stream. The DTLB only maps 4-KByte pages/sub-pages.

The DTLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing store for the DTLB. If a load/store address cannot be translated by the DTLB, a DTLB miss is issued, and the JTLB is used to attempt to translate it in the following clock cycle. If successful, the translation information is copied into the DTLB for future use. As for the ITLB, this DTLB miss sequence has a penalty of two extra clock cycles.

If there are simultaneous ITLB and DTLB misses, the DTLB gets first priority when accessing the JTLB, giving a total of three latency cycles.

Virtual to Physical Address Translation

Converting a virtual address to a physical address begins by comparing the virtual address from the processor with the virtual addresses in the TLB. A match occurs when the virtual page number (VPN) of the address is the same as the VPN field of the entry, and either:

- The Global (*G*) bit of both the even and odd pages of the TLB entry is set, or
- The ASID field of the virtual address is the same as the ASID field of the TLB entry.

This match is referred to as a TLB *hit*. If there is no match, a TLB *refill* exception is taken by the processor and

software is allowed to refill the TLB from a page table of virtual/physical addresses in memory.

Figure 6 shows a flow diagram of the address translation process. The 5Kf processor uses a 64-bit virtual address with 40-bit virtual segments. Physical addresses are 36 bits wide. The top portion of Figure 6 shows a virtual address for a 4-KByte page size. The width of the *Offset* in Figure 6 is defined by the page size. The remaining upper bits of the address represent the virtual page number (VPN).

The bottom portion of Figure 6 shows the virtual address for a 16-Mbyte page size. The remaining upper bits of the address represent the VPN.

In this figure, the virtual address is supplemented by a unique 8-bit address space identifier (ASID), which eliminates TLB flushing during a context switch. The ASID contains the number assigned to that process and is stored in the CP0 *EntryHi* register.

Hits, Misses, and Multiple Matches

Each TLB entry contains a tag portion and a data portion. If a match is found, the upper bits of the virtual address are replaced with the page frame number (PFN) stored in the corresponding entry in the data array of the TLB. If no match occurs (TLB miss), an exception is taken and software refills the TLB from the page table resident in memory.

The 5Kf core implements a TLB write compare mechanism to ensure that multiple TLB matches do not occur. On the TLB write operation, the write value is compared with all other entries in the TLB. If a match occurs, the 5Kf core takes a machine check exception, sets the TS bit in the CP0 *Status* register, and aborts the write operation.

Page Sizes and Replacement Algorithm

To assist in controlling both the amount of mapped space and the replacement characteristics of various memory regions, the 5Kf core provides two mechanisms. First, the page size can be configured, on a per entry basis, to map a page size of 4 KBytes to 16 Mbytes (in multiples of 4).

The CP0 *PageMask* register is loaded with the mapping page size, which is then entered into the TLB when a new entry is written. Thus, operating systems can provide special purpose mappings. For example, a typical frame buffer can be memory mapped with only one TLB entry.

The second mechanism controls the replacement algorithm when a TLB miss occurs. To select a TLB entry to be written with a new mapping, the 5Kf core provides a random replacement algorithm. However, the processor also provides a mechanism whereby a programmable number of mappings can be locked into the TLB via the *Wired* register, thus avoiding random replacement.

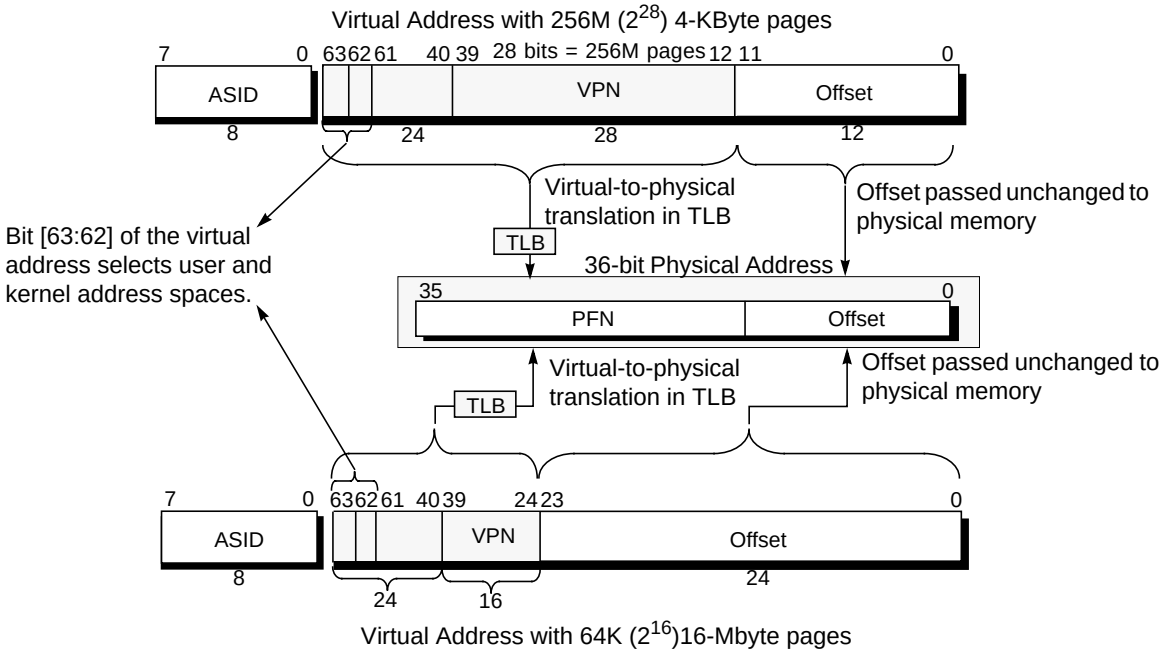

Figure 6 Virtual Address Translation

Fixed Mapping Translation (FMT)

The 5Kf core provides a simple fixed mapping translation (FMT) mechanism that is smaller than the TLB in the MIPS64 5Kf core and more easily synthesized. Like the TLB, the FMT performs virtual-to-physical address translation and provides attributes for the different segments. Those segments that are unmapped in the 5Kf core's TLB implementation (kseg0 and kseg1) are translated identically by the FMT and TLB.

With the FMT, only 32-bit addresses can be translated.

Figure 7 shows how the FMT is implemented in the 5Kf core.

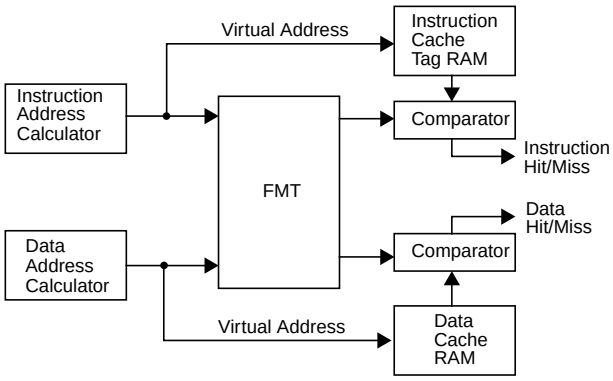

Figure 7 Address Translation During a Cache Access

The FMT also determines the cacheability of each segment. These attributes are controlled via bits in the Config register. Table 8 shows the encoding for the K23 (bits 30:28), KU (bits 27:25), and K0 (bits 2:0) fields of the Config register.

Table 8 Cache Coherency Attributes

Config Register Fields K23, KU, and K0	Cache Coherency Attribute
0	Cacheable, noncoherent, write through, no write allocate
1	Cacheable, noncoherent, write through, write allocate
2	Uncached (write around)
3, 4, 5, 6	Cacheable, noncoherent, write back (write allocate)
7	Uncached accelerated

In the 5Kf core, no translation exceptions can be taken, although address errors are still possible.

Table 9 Cacheability of Segments with FMT[a]

Segment	Virtual Address Range	Cacheability
useg/ kuseg	0x0000_0000- 0x7FFF_FFFF	Controlled by the KU field (bits 27:25) of the Config register. See Table 8 for mapping. This segment is always uncached when ERL = 1.
kseg0	0x8000_0000- 0x9FFF_FFFF	Controlled by the K0 field (bits 2:0) of the Config register. See Table 8 for mapping.
kseg1	0xA000_0000- 0xBFFF_FFFF	Always uncacheable
sseg	0xC000_0000- 0xDFFF_FFFF	Controlled by the K23 field (bits 30:28) of the Config register. See Table 8 for mapping.
kseg3	0xE000_0000- 0xFFFF_FFFF	Controlled by the K23 field (bits 30:28) of the Config register. See Table 8 for mapping.

a.Only 32-bit addresses.

The FMT performs a simple translation to map from virtual addresses to physical addresses. This mapping is shown in Figure 8.

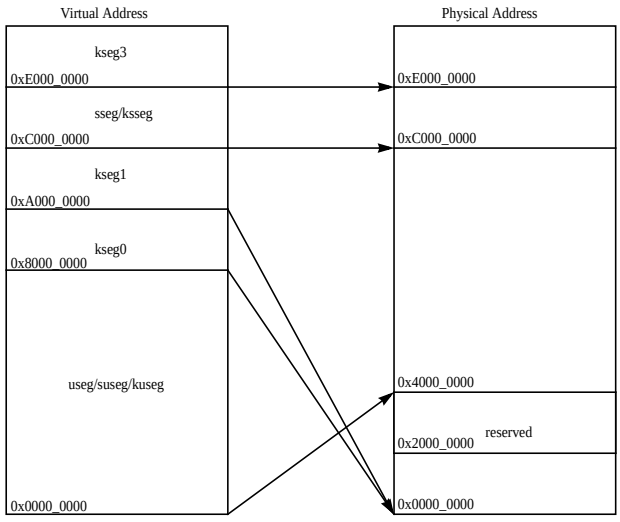

Figure 8 FMT Memory Map (ERL=0) in the 5Kf Core

When ERL = 1, useg and kuseg become unmapped and uncached. This behavior is the same as if there was a TLB. This mapping is shown in Figure 9.

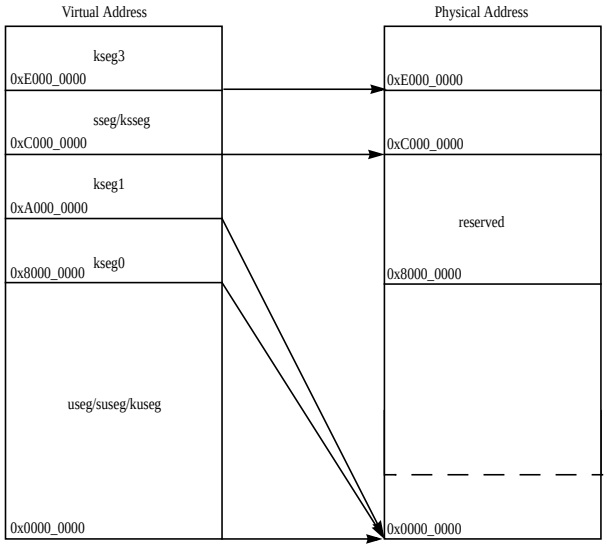

Figure 9 FMT Memory Map (ERL=1) in the 5Kf Core

Bus Interface (BIU)

The Bus Interface Unit (BIU) controls the external interface signals and contains three buffers for managing the flow of read and write data onto the external bus.

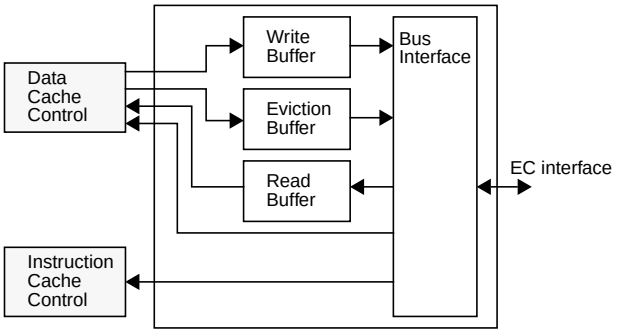

Figure 10 Bus Interface Unit Buffers

The write buffer is used during store transactions to the BIU and consists of a 32-byte write buffer and 1-line eviction buffer. The 32-byte buffer is used to buffer (and combine for Uncached Accelerated) write-through and uncached store transactions before issuing them at the external interface. When accessing with a write-through cache policy, the write buffer significantly reduces the amount of stalling in the core caused by the issuance of multiple writes in a short period of time. The 1-line eviction buffer is used on cache-line write backs.

The 32-byte read buffer is essentially a 4-doubleword deep FIFO. This buffer is required in order to allow cache line refills from the BIU to start immediately, even if the data cache controller must complete a line eviction before being able to receive data. If the bus latency is short and data is returned to the 5Kf core before the eviction is complete, the incoming data is buffered until the eviction is complete. If the cache controller is ready to accept the data as it is returned from the BIU, the FIFO is bypassed and the data is forwarded directly from the BIU to the data cache controller.

Uncached Accelerated Stores

For uncached accelerated stores, the write buffer:

- Attempts to merge consecutive word stores into a single doubleword store.
- Attempts to gather four doublewords into a burst transaction.

Note that the first doubleword of a burst must have address bits 4:0 equal to zero.

Power Management

The 5Kf processor cores offer a number of power management features, including low-power design, active power management and power-down modes of operation. The core is a static design that supports a WAIT instruction designed to signal the rest of the system that execution and clocking should be halted, thereby reducing system power consumption during idle periods.

The 5Kf core provides two mechanisms for system-level low-power support:

- Register-controlled power management
- Instruction-controlled power management

Register Controlled Power Management

The RP bit in the CP0 Status register provides a software mechanism for placing the system into a low power state. The state of the RP bit is available externally via the SI_RP signal. Two additional bits, EXL and ERL, support the power management function by allowing the user to change the power state if an exception or error occurs while the 5Kf core is in a low power state. The EXL bit is available externally via the SI_EXL signal. The ERL bit is available externally via the SI_ERL signal.

These 3 power down signals are part of the system interface and change state as the corresponding bits in the CP0 *Status* register are set or cleared.

- The SI_RP signal represents the state of the RP bit (27) in the CP0 Status register.
- The SI_EXL signal represents the state of the EXL bit (1) in the CP0 Status register.
- The SI_ERL signal represents the state of the ERL bit (2) in the CP0 Status register.

Instruction Controlled Power Management

The second mechanism for invoking power down mode is through execution of the WAIT instruction. If the bus is idle at the time the WAIT instruction reaches the M stage of the pipeline, the internal clocks are suspended and the pipeline is frozen. However, the internal timer and some of the input pins (SI_Int[5:0], SI_NMI, SI_Reset, SI_ColdReset, and EJ_DINT) continue to run. If the bus is not idle at the time the WAIT instruction reaches the M stage of the pipeline stalls until the bus becomes idle, at which time the clocks are stopped.

Execution of the WAIT instruction causes the 5Kf core to assert the SI_Sleep signal, thereby indicating to external agents that the device is in low-power mode.

Once the CPU is in instruction controlled power management mode, any enabled interrupt, NMI or debug interrupt (through EJ_DINT) causes the CPU to exit this mode. The device re-enters instruction controlled power management mode once the next WAIT instruction is executed.

Coprocessor Interface

This interface allows a single coprocessor to be connected to the 5Kf processor core. This interface has the following features:

- It is easy to understand. By keeping the interface as simple as possible, designers will be able to concentrate on the coprocessor functionality, not its interface.
- Minimal interface logic is required. This reduces area and power overhead.
- Performance is not compromised. This interface is compatible with all high-performance features of the 5Kf microprocessor core.

All MIPS64 compliant coprocessor instructions are supported. This includes COP1, COP2, MDMX, and COP1X instructions.

This interface is pipeline independent. That is to say, the pipeline microarchitecture of the coprocessor need not match that of the 5Kf integer unit. This allows for great flexibility in the type and construction of the coprocessor logic.

To fully execute all coprocessor instructions, several data transfers must happen. The Coprocessor interface implements simple transfers for each of these required items:

- **Instruction Dispatch**. Starts coprocessor instructions
- **To COP Data**. Transfers data to the coprocessor
- **From COP Data**. Transfers data from the coprocessor
- **Coprocessor Condition Code Check**. Transfers coprocessor condition check result to the integer unit
- **GPR Data**. Transfers additional data from the integer unit general-purpose register file to the coprocessor
- **Coprocessor Exceptions**. Notifies the integer unit if any coprocessor exceptions happened for an instruction
- **Instruction Nullification**. Notifies coprocessor if instructions are nullified or not (due to the delay slot instruction of a branch likely not taken)
- **Instruction Killing**. Notifies coprocessor when instructions can commit state or not.

5Kf Core Optional Logic Blocks

The 5Kf core contains the following optional logic blocks, as shown in the block diagram in Figure 1.

- Instruction Cache
- Data Cache
- EJTAG Debug Support

Instruction Cache

The instruction cache is an optional on-chip memory block of up to 64 KBytes (16 KBytes/way in a 4-way set associative implementation). The instruction cache consists of three on-chip RAMs:

- Instruction Tag RAM (I-Tag RAM)
- Instruction Data RAM (I-Data RAM)
- Instruction Way Selection RAM (I-WS RAM)

The I-Tag RAM contains 24 bits of physical address, 1 valid bit, an optional parity bit, and a lock bit. The I-WS RAM contains a 6-bit status for the way selection algorithm. The I-Data RAM contains the data from main memory as well as 8 optional parity bits.

As the instruction cache is virtually indexed, the virtual-to-physical address translation occurs in parallel with the cache access rather than having to wait for the physical address translation.

The instruction cache block also contains and manages the instruction line fill buffer. Instruction fetches that reference instructions being refilled are streamed whenever possible, or returned as a hit after the refill has completed.

The 5Kf core supports instruction cache-locking. Cache locking allows critical code or data segments to be locked into the cache on a “per-line” basis, enabling the system programmer to maximize the efficiency of the system cache. The cache locking function is always available on all instruction cache entries. Entries can then be marked as locked or unlocked on a per entry basis using the CACHE instruction.

During an instruction cache lookup, the virtual address of the instruction fetch is made available prior to the I-stage and is used to index the I-Data and I-Tag RAMs. These RAMs are read at the beginning of the I-stage to determine if the required instruction resides in the cache. The physical address from the MMU is compared with up to 4 tags from the I-Tag RAM, depending on the associativity of the cache. The I-WS RAM is updated when a fetch returns as a hit.

Data Cache

The data cache is an optional on-chip memory block of up to 64 KBytes (16 KBytes/way in a 4-way set associative implementation). The data cache consists of three on-chip RAMs:

- Data Tag RAM (D-Tag RAM)
- Data Data RAM (D-Data RAM)
- Data Way Selection RAM (D-WS)

The D-Tag RAM contains 24 bits of physical address, 1 valid bit, an optional parity bit, and a lock bit. The D-WS RAM contains (in a 4-way set associative configuration) a 6-bit status for the way selection algorithm, 4 dirty bits and optional 4 parity bits (one dirty bit and one parity bit per way). The D-Data RAM contains the data from main memory as well as 8 optional parity bits.

The 5Kf core also supports a data cache locking mechanism identical to the instruction cache. Critical data segments can be locked into the cache on a “per-line” basis. The locked contents can be updated on a store hit, but cannot be selected for replacement on a load or store miss.

The cache locking function is always available on all data cache entries. Entries can then be marked as locked or unlocked on a per entry basis using the CACHE instruction.

During a data cache lookup, the virtual address of the data load is made available in the E-stage and is used to index the D-Data and D-Tag RAMs. These RAMs are read in the E-stage and M-stage to determine if the required data resides in the cache. The physical address from the MMU is compared with up to 4 tags from the D-Tag RAM, depending on the associativity in the cache. The D-WS RAM is updated when a load returns as a hit, or on a store to the cache.

Cache Memory Configuration

The 5Kf core incorporates on-chip instruction and data caches that can each be accessed in a single processor cycle. Each cache has its own 64-bit data path and can be accessed in the same pipeline clock cycle. Table 10 lists the 5Kf core instruction and data cache attributes:

Table 10 5Kf Core Instruction and Data Cache Attributes

Parameter	Instruction	Data
Size	0 - 64 KBytes	0 - 64 KBytes
Organization	1 - 4 way set associative	1 - 4 way set associative
Line Size	32 bytes	32 bytes
Read Unit	64 bits	64 bits
Write Policy	na	1) Uncached 2) Write-through, no write-allocate 3) Write-through with write-allocate 4) Write-back, write-allocate 5) Uncached accelerated
Cache Locking	per line	per line

Cache Protocols

The 5Kf core supports the following cache protocols:

- **Uncached:** Addresses in a memory area indicated as uncached are not read from the cache. Stores to such addresses are written directly to main memory, without changing cache contents.
- **Write-through, No Write Allocate:** Loads and instruction fetches first search the cache, reading main memory only if the desired data does not reside in the cache. On data store operations, the cache is first searched to see if the target address is cache resident. If it is resident, the cache contents are updated, and main memory is also written. If the cache lookup misses, only main memory is written.
- **Write-through, Write Allocate:** Loads and instruction fetches first search the cache, reading main memory only if the desired data does not reside in the cache. On data store operations the cache is searched to determine if the target address is cache-resident. If it is resident, the cache contents are updated and the data is written to main memory. If the cache lookup misses, the line is refilled into the cache. The data is written to the cache and to main memory.
- **Write-back, Write Allocate:** Loads and instruction fetches first search the cache, reading main memory only if the desired data does not reside in the cache. On data store operations the cache is searched to determine if the target address is cache-resident. If it is resident, the cache contents are updated and the line is marked as dirty (store operation). If the cache lookup misses, the line is refilled into the cache, marked as dirty, and the cache contents are updated with the store data. If a line to be replaced is marked as dirty, it is evicted from the cache before the new line is read into the cache. The line can only be marked as dirty on a store to a Write-back Write Allocate line.
- **Uncached Accelerated:** Same as uncached except that the processor will attempt to merge consecutive stores into a burst write transaction on the bus, thus optimizing bus utilization.

EJTAG Debug Support

The 5Kf core provides EJTAG debug support for use in development of application code. The EJTAG debug support introduces a Debug Mode of operation, which is similar to Kernel Mode in some aspects, but also allows for programming of the debug resources and has special handling characteristics for managing exceptions and other debug related issues. Debug Mode is entered after a debug exception is taken. A debug exception can be caused by several sources.

- The Software Debug Breakpoint (SDBBP) instruction which is used as an instruction breakpoint.
- The single-step feature after the execution of one instruction (two instructions for jump, branch, and delay slot) in Non-Debug Mode.
- A debug interrupt requested by assertion of the EJ_DINT signal or through the Test Access Port (TAP).
- Hardware breakpoints, either on instruction or data access.

Three debug registers (DEBUG, DEPC, and DESAVE) are included in the MIPS Coprocessor 0 (CP0) register set. The DEBUG register shows the cause of the debug exception and is used for the setting up of single step operations. The Debug Exception Program Counter (DEPC) register holds the address on which the debug exception was taken. This is used to resume program execution after the debug operation finishes. Finally, the Debug Exception Save (DESAVE) register enables the saving of all general purpose registers used during execution of the debug exception handler.

To exit Debug Mode, a Debug Exception Return (DERET) instruction is executed. Execution is resumed in the mode in which the debug exception occurred. This allows for nonintrusive execution of the debug handler.

Hardware Breakpoints

Hardware breakpoints are provided as an optional feature. Four instruction breakpoints and two data breakpoints are supported. The hardware breakpoints compare all instruction fetches and data accesses in Non-Debug Mode with the programmed breakpoints. A debug exception is taken when a hardware breakpoint matches, whereby the normal application is suspended and Debug Mode is entered.

Instruction hardware breakpoints are set on the instruction virtual address and can also compare the ASID value used by the MMU. A bit mask can apply to the virtual address to set a breakpoints on a range of instructions.

Data hardware breakpoints are set on the virtual address and ASID value, similar to the instruction breakpoint. These breakpoints can match explicit load/store accesses. Data breakpoints can also be set based on the data value of

the load/store operation. Finally, masks can be applied to both the virtual address and the load/store data value.

Test Access Port Interface

The 5Kf core provides optional Test Access Port (TAP) logic that forms a dedicated interface to the debug host. The TAP allows the debug host to provide the debug handler through the EJTAG debug memory area. Therefore, no integration of the debug handler in system memory is necessary. The debug host can also force the core into Debug Mode through the TAP by generating a debug interrupt request.

5Kf Core Reset

The 5Kf core has two types of reset input signals: SI_Reset and SI_ColdReset.

The SI_ColdReset signal must be asserted on either a power-on reset or a cold reset. In a typical application, a power-on reset occurs when the machine is first turned on. A cold reset (also called a hard reset) typically occurs when the machine is already on and the system is rebooted. However, a cold reset has the same overall effect as a power-on reset in that it completely initializes the internal state machines of the 5Kf core without saving any state information.

The SI_Reset and SI_ColdReset signals determine the type of reset operation as shown in Table 11.

Table 11 5Kf Reset Types

SI_Reset	SI_ColdReset	Action
0	0	Normal Operation, no reset.
1	0	Warm or Soft reset.
x	1	Cold or Hard reset.

The assertion of the SI_Reset signal causes a warm reset. A warm reset restarts the 5Kf core, but preserves some of the processors internal state. The assertion of SI_Reset causes a soft reset exception within the 5Kf core.

In addition to the normal hard and soft resets, the 5Kf core supports EJTAG boot, where the core performs the initialization of a reset exception and takes a Debug Interrupt exception. This allows debug software to initialize the processor debug resources before the target software starts to run without adding debug code to the target software.

No instruction fetches are performed from the reset exception vector for EJTAG boot. The first instruction fetch is from the debug exception vector.

Testability for Production Test

The design supports testability for production test through internal scan and memory BIST.

Internal Scan

Muxed flip-flop fullscan design is supported for maximum coverage, with a configurable number of scan chains. ATPG test coverage can exceed 99% (library and configuration dependent).

Memory BIST

Memory BIST is optional, but can be implemented either through use of integrated memory BIST provided with the 5Kf core, or inserted with an industry standard memory BIST CAD tool.

Integrated Memory BIST

The 5Kf core provides an integrated memory BIST solution for test of cache RAMs using a memory BIST controller integrated in the cache system of the 5Kf core. The inclusion of the integrated memory BIST controller is optional and several parameters including algorithm (March C+ or IFA-13) is configurable.

Memory BIST Inserted by CAD Tool

Memory BIST can also be inserted by an industry standard memory BIST CAD tool. Wrapper modules and signal busses of configurable width are provided in the 5Kf core for easy adaptation to the provided BIST controller.

5Kf Core Instructions

The 5Kf core instruction set complies with the MIPS64 instruction set architecture. The instructions are divided into base instructions and floating point instructions.

5Kf Core Base Instructions

Table 12 provides a summary of the base instructions implemented by the 5Kf core.

Table 12 5Kf Core Base Instruction Set

Instruction	Description	Function
ADD	Add	`Rd = Rs + Rt`
ADDI	Add Immediate	`Rt = Rs + Immed`
ADDIU	Unsigned Add Immediate	`Rt = (uns)Rs + Immed`
ADDU	Unsigned Add	`Rd = (uns)Rs + Rt`
AND	Logical AND	`Rd = Rs & Rt`
ANDI	Logical AND Immediate	`Rt = Rs & (0`$_{48}$` || Immed)`
BC2F	Branch On Coprocessor 2 False	`if COP2_condition == 0` `PC += offset`

BC2FL	Branch On Coprocessor 2 False Likely	`if COP2_condition == 0`
		`PC += offset`
		`else`
		`  Ignore Next Instruction`

BC2T	Branch On Coprocessor 2 True	`if COP2_condition == 1` ` PC += offset`

BC2TL	Branch On Coprocessor 2 True Likely	`if COP2_condition == 1`
		` PC += offset`
		`else`
		`  Ignore Next Instruction`

BEQ	Branch On Equal	`if Rs == Rt` ` PC += offset`

BEQL	Branch On Equal Likely	`if Rs == Rt`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BGEZ	Branch on Greater Than or Equal To Zero	`if !Rs[63]` `  PC += offset`

BGEZAL	Branch on Greater Than or Equal To Zero And Link	`if !Rs[63]`
		`  GPR[31] = PC + 8`
		`   PC += offset`

BGEZALL	Branch on Greater Than or Equal To Zero And Link Likely	`if !Rs[63]`
		`  GPR[31] = PC + 8`
		`   PC += offset`
		`else`
		`  Ignore Next Instruction`

Table 12 5Kf Core Base Instruction Set (Continued)

Instruction	Description	Function

BGEZL	Branch on Greater Than or Equal To Zero Likely	`if !Rs[63]`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BGTZ	Branch on Greater Than Zero	`if !Rs[63] && Rs != 0` `  PC += offset`

BGTZL	Branch on Greater Than Zero Likely	`if !Rs[63] && Rs != 0`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BLEZ	Branch on Less Than or Equal to Zero	`if Rs[63] || Rs == 0` `  PC += offset`

BLEZL	Branch on Less Than or Equal to Zero Likely	`if Rs[63] || Rs == 0`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BLTZ	Branch on Less Than Zero	`if Rs[63]` `  PC += offset`

BLTZAL	Branch on Less Than Zero And Link	`if Rs[63]`
		`  GPR[31] = PC + 8`
		`  PC += offset`

BLTZALL	Branch on Less Than Zero And Link Likely	`if Rs[63]`
		`  GPR[31] = PC + 8`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BLTZL	Branch on Less Than Zero Likely	`if Rs[63]`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BNE	Branch on Not Equal	`if Rs != Rt` `  PC += offset`

BNEL	Branch on Not Equal Likely	`if Rs != Rt`
		`  PC += offset`
		`else`
		`  Ignore Next Instruction`

BREAK	Breakpoint	Breakpoint Exception
CACHE	Cache Operation	See MIPS64 5K Processor Core Family Software User's Manual
CFC2	Control From Coprocessor 2	`Rt = CCR[2, Rd]`
CLO	Count Leading Ones	`Rd = NumLeadingOnes(Rs[31:0])`
CLZ	Count Leading Zeroes	`Rd = NumLeadingZeroes(Rs[31:0])`
CTC2	Control To Coprocessor 2	`CCR[2, Rd] = Rt`
DADD	Doubleword Add	`Rd = Rs + Rt`
DADDI	Doubleword Add Immediate	`Rt = Rs + Immed`

Table 12 5Kf Core Base Instruction Set (Continued)

Instruction	Description	Function
DADDIU	Unsigned Doubleword Add Immediate	`Rt = Rs + Immed`
DADDU	Unsigned Doubleword Add	`Rd = Rs + Rt`
DCLO	Doubleword Count Leading Ones	`Rd = NumLeadingOnes(Rs)`
DCLZ	Doubleword Count Leading Zeros	`Rd = NumLeadingZeroes(Rs)`
DDIV	Doubleword Divide	`LO = Rs / Rt` `HI = Rs % Rt`
DDIVU	Unsigned Doubleword Divide	`LO = (uns)Rs / Rt` `HI = (uns)Rs % Rt`
DERET	Debug Exception Return	`PC = DEPC` `Exit Debug Mode`
DIV	Divide	`LO = Rs / Rt` `HI = Rs % Rt`
DIVU	Unsigned Divide	`LO = (uns)Rs / Rt` `HI = (uns)Rs % Rt`
DMFC0	Doubleword Move From Coprocessor 0	`Rt = CPR[0, Rd, sel]`
DMFC2	Doubleword Move From Coprocessor 2	`Rt = CPR[2, Rd]`
DMTC0	Doubleword Move To Coprocessor 0	`CPR[0, Rd, sel] = Rt`
DMTC2	Doubleword Move To Coprocessor 2	`CPR[2, Rd] = Rt`
DMULT	Doubleword Multiply	`HI|LO = Rs * Rd`
DMULTU	Unsigned Doubleword Multiply	`HI|LO = (uns)Rs * Rd`
DSLL	Doubleword Shift Left Logical	`Rd = Rt << sa`
DSLLV	Doubleword Shift Left Logical Variable	`Rd = Rt << Rs[4:0]`
DSLL32	Doubleword Shift Left Logical Plus 32	`Rd = Rt << sa+32`
DSRA	Doubleword Shift Right Arithmetic	`Rd = Rt >> sa`
DSRAV	Doubleword Shift Right Arithmetic Variable	`Rd = Rt >> Rs[4:0]`
DSRA32	Doubleword Shift Right Arithmetic Plus 32	`Rd = Rt >> sa+32`
DSRL	Doubleword Shift Right Logical	`Rd = (uns)Rt >> sa`
DSRLV	Doubleword Shift Right Logical Variable	`Rd = (uns)Rt >> Rs[4:0]`
DSRL32	Doubleword Shift Right Logical Plus 32	`Rd = (uns)Rt >> sa+32`
DSUB	Doubleword Subtract	`Rd = Rs - Rt`
DSUBU	Unsigned Doubleword Subtract	`Rd = (uns)Rs - Rt`

Table 12 5Kf Core Base Instruction Set (Continued)

Instruction	Description	Function

ERET	Return from Exception	`if SR[2]`
		`  PC = ErrorEPC`
		`else`
		`  PC = EPC`
		`SR[1] = 0`
		`SR[2] = 0`
		`LL = 0`

J	Unconditional Jump	`PC = PC[63:28] || offset<<2`
JAL	Jump and Link	`GPR[31] = PC + 8` `PC = PC[63:28] || offset<<2`
JALR	Jump and Link Register	`Rd = PC + 8` `PC = Rs`
JR	Jump Register	`PC = Rs`
LB	Load Byte	`Rt = (byte)Mem[Rs+offset]`
LBU	Unsigned Load Byte	`Rt = (ubyte)Mem[Rs+offset]`
LD	Load Doubleword	`Rt = Mem[Rs+offset]`
LDC2	Load Doubleword to Coprocessor 2	`CPR[2,Rt] = Mem[Rs+offset]`
LDL	Load Doubleword Left	See MIPS64 5K Processor Core Family Software User's Manual
LDR	Load Doubleword Right	See MIPS64 5K Processor Core Family Software User's Manual
LH	Load Halfword	`Rt = (half)Mem[Rs+offset]`
LHU	Unsigned Load Halfword	`Rt = (uhalf)Mem[Rs+offset]`

LL	Load Linked Word	`Rt = (word)Mem[Rs+offset]`
		`LL = 1`
		`LLAdr = Rs + offset`

LLD	Load Linked Doubleword	`Rt = Mem[Rs+offset]`
		`LL = 1`
		`LLAdr = Rs + offset`

LUI	Load Upper Immediate	`Rt = immediate << 16`
LW	Load Word	`Rt = (word)Mem[Rs+offset]`
LWC2	Load Word to Coprocessor 2	`CPR[2,Rt] = (word)Mem[Rs+offset]`
LWL	Load Word Left	See MIPS64 5K Processor Core Family Software User's Manual
LWR	Load Word Right	See MIPS64 5K Processor Core Family Software User's Manual
LWU	Load Word Unsigned	`Rt = (uword)Mem[Rs+offset]`
MADD	Multiply-Add	`HI|LO += Rs * Rt`
MADDU	Multiply-Add Unsigned	`HI|LO += (uns)Rs * Rt`

Table 12 5Kf Core Base Instruction Set (Continued)

Instruction	Description	Function
MFC0	Move From Coprocessor 0	`Rt = CPR[0, Rd, sel]`
MFC2	Move From Coprocessor 2	`Rt = CPR[2, Rd]`
MFHI	Move From HI	`Rd = HI`
MFLO	Move From LO	`Rd = LO`
MOVF	Move Conditional on Floating Point False	`if cc[i] == 0 then` `  GPR[rd] = GPR[rs]`
MOVN	Move Conditional on Not Zero	`if Rt != 0 then` `  Rd = Rs`
MOVT	Move Conditional on Floating Point True	`if cc[i] == 1 then` `  GPR[rd] = GPR[rs]`
MOVZ	Move Conditional on Zero	`if Rt == 0 then` `  Rd = Rs`
MSUB	Multiply-Subtract	`HI|LO -= Rs * Rt`
MSUBU	Multiply-Subtract Unsigned	`HI|LO -= (uns)Rs * Rt`
MTC0	Move To Coprocessor 0	`CPR[0, Rd, sel] = Rt`
MTC2	Move To Coprocessor 2	`CPR[2, Rd] = Rt`
MTHI	Move To HI	`HI = Rs`
MTLO	Move To LO	`LO = Rs`
MUL	Multiply with register write	`HI|LO = Unpredictable` `Rd = Rs * Rd`
MULT	Integer Multiply	`HI|LO = Rs * Rd`
MULTU	Unsigned Multiply	`HI|LO = (uns)Rs * Rd`
NOR	Logical NOR	`Rd = ~(Rs | Rt)`
OR	Logical OR	`Rd = Rs | Rt`
ORI	Logical OR Immediate	`Rt = Rs | Immed`
PREF	Prefetch	Prefetch data from memory
PREFX	Prefetch Indexed	Prefetch data from memory using (GPR+GPR) addressing
SB	Store Byte	`(byte)Mem[Rs+offset] = Rt`

SC	Store Conditional Word	`if LL == 1`
		`  (word)Mem[Rs+offset] = Rt`
		`Rt = LL`

SCD	Store Condition Doubleword	`if LL == 1`
		`  Mem[Rs+offset] = Rt`
		`Rt = LL`

SD	Store Doubleword	`Mem[Rs+offset] = Rt`
SDBBP	Software Debug Breakpoint	Debug breakpoint exception

Table 12 5Kf Core Base Instruction Set (Continued)

Instruction	Description	Function
SDC2	Store Doubleword from Coprocessor 2	`Mem[Rs+offset] = CPR[2,Rt]`
SDL	Store Doubleword Left	See MIPS64 5K Processor Core Family Software User's Manual
SDR	Store Doubleword Right	See MIPS64 5K Processor Core Family Software User's Manual
SH	Store Half	`(half)Mem[Rs+offset] = Rt`
SLL	Shift Left Logical	`Rd = Rt << sa`
SLLV	Shift Left Logical Variable	`Rd = Rt << Rs[4:0]`

SLT	Set on Less Than	`if Rs < Rt`
		`  Rd = 1`
		`else`
		`  Rd = 0`

SLTI	Set on Less Than Immediate	`if Rs < Immed`
		`  Rt = 1`
		`else`
		`  Rt = 0`

SLTIU	Unsigned Set on Less Than Immediate	`if (uns)Rs < Immed`
		`  Rt = 1`
		`else`
		`  Rt = 0`

SLTU	Unsigned Set on Less Than	`if (uns)Rs < Rt`
		`  Rd = 1`
		`else`
		`  Rd = 0`

SRA	Shift Right Arithmetic	`Rd = Rt >> sa`
SRAV	Shift Right Arithmetic Variable	`Rd = Rt >> Rs[4:0]`
SRL	Shift Right Logical	`Rd = (uns)Rt >> sa`
SRLV	Shift Right Logical Variable	`Rd = (uns)Rt >> Rs[4:0]`
SSNOP	Superscalar Inhibit No Operation	See MIPS64 5K Processor Core Family Software User's Manual
SUB	Subtract	`Rd = Rs - Rt`
SUBU	Unsigned Subtract	`Rd = (uns)Rs - Rt`
SW	Store Word	`(word)Mem[Rs+offset] = Rt`
SWC2	Store Word from Coprocessor 2	`(word)Mem[Rs+offset] = CPR[2,Rt]`
SWL	Store Word Left	See MIPS64 5K Processor Core Family Software User's Manual
SWR	Store Word Right	See MIPS64 5K Processor Core Family Software User's Manual
SYNC	Synchronize Memory	See MIPS64 5K Processor Core Family Software User's Manual

Table 12 5Kf Core Base Instruction Set (Continued)

Instruction	Description	Function
SYSCALL	System Call	`SystemCallException`
TEQ	Trap if Equal	`if Rs == Rt` `  TrapException`
TEQI	Trap if Equal Immediate	`if Rs == Immed` `  TrapException`
TGE	Trap if Greater Than or Equal	`if Rs >= Rt` `  TrapException`
TGEI	Trap if Greater Than or Equal Immediate	`if Rs >= Immed` `  TrapException`
TGEIU	Unsigned Trap if Greater Than or Equal Immediate	`if (uns)Rs >= Immed` `  TrapException`
TGEU	Unsigned Trap if Greater Than or Equal	`if (uns)Rs >= Rt` `  TrapException`
TLBWI	Write Indexed TLB Entry	See MIPS64 5K Processor Core Family Software User's Manual
TLBWR	Write Random TLB Entry	See MIPS64 5K Processor Core Family Software User's Manual
TLBP	Probe TLB for Matching Entry	See MIPS64 5K Processor Core Family Software User's Manual
TLBR	Read Indexed TLB Entry	See MIPS64 5K Processor Core Family Software User's Manual
TLT	Trap if Less Than	`if Rs < Rt` `  TrapException`
TLTI	Trap if Less Than Immediate	`if Rs < Immed` `  TrapException`
TLTIU	Unsigned Trap if Less Than Immediate	`if (uns)Rs < Immed` `  TrapException`
TLTU	Unsigned Trap if Less Than	`if (uns)Rs < Rt` `  TrapException`
TNE	Trap if Not Equal	`if Rs != Rt` `  TrapException`
TNEI	Trap if Not Equal Immediate	`if Rs != Immed` `  TrapException`
WAIT	Wait for Interrupts	Stall until interrupt occurs
XOR	Exclusive OR	`Rd = Rs ^ Rt`
XORI	Exclusive OR Immediate	`Rt = Rs ^ (uns)Immed`

5Kf Core Floating Point Instructions

Table 13 provides a summary of the floating point instructions implemented by the 5Kf core.

Table 13 5Kf Floating Point Instruction Set

Instruction	Format*	Description	Function
ABS.fmt	S, D	Floating Point Absolute Value	Fd = abs(Fs)
ADD.fmt	S, D	Floating Point Add	Fd = Fs + Ft
BC1F		Branch on Floating Point False	if (cc[i] == 0) then PC += (int)offset

BC1FL		Branch on Floating Point False Likely	if (cc[i] == 0) then
			PC += (int)offset
			else
			NullifyCurrentInstruction()

BC1T		Branch on Floating Point True	if (cc[i] == 1) then PC += (int)offset

BC1TL		Branch on Floating Point True Likely	if (cc[i] == 1) then
			PC += (int)offset
			else
			NullifyCurrentInstruction()

C.cond.fmt	S, D	Floating Point Compare	cc[i] = Fs compare_cond Ft
CEIL.L.fmt	S, D	Floating Point Ceiling to Long Fixed Point	Fd = convert_and_round(Fs)
CEIL.W.fmt	S, D	Floating Point Ceiling to Word Fixed Poin	Fd = convert_and_round(Fs)
CFC1		Copy Word from Floating Point Control Register	Rt = FP_Control[Fs]
CTC1		Copy Word to Floating Point Control Register	FP_Control[Fs] = Rt
CVT.D.fmt	S, W, L	Floating Point Convert to Double Floating Point	Fd = convert_and_round(Fs)
CVT.L.fmt	S, D	Floating Point Convert to Long Fixed Point	Fd = convert_and_round(Fs)
CVT.S.fmt	W, D, L	Floating Point Convert to Single Floating Point	Fd = convert_and_round(Fs)
CVT.W.fmt	S, D	Floating Point Convert to Word Fixed Point	Fd = convert_and_round(Fs)
DIV.fmt	S, D	Floating Point Divide	Fd = Fs / Ft
DMFC1		Move Doubleword from FPR	Rt = Fs
DMTC1		Move Doubleword to FPR	Fs = Rt
FLOOR.L.fmt	S, D	Floating Point Floor to Long Fixed Point	Fd = convert_and_round(Fs)
FLOOR.W.fmt	S, D	Floating Point Floor to Word Fixed Point	Fd = convert_and_round(Fs)
LDC1		Load Doubleword to Floating Point	Ft = memory[base+offset]
LDXC1		Load Doubleword Indexed to Floating Point	Fd = memory[base+index]
LUXC1		Load Doubleword Indexed Unaligned to Floating Point	Fd = memory[(base+index)$_{psize-1..3}$]
LWC1		Load Word to Floating Point	Ft = memory[base+offset]
LWXC1		Load Word Indexed to Floating Point	Fd = memory[base+index]
MADD.fmt	S, D	Floating Point Multiply Add	Fd = Fs * Ft + Fr
MFC1		Move Word from FPR	Rt = Fs
MOV.fmt	S, D	Floating Point Move	Fd = Fs
MOVF.fmt	S, D	Floating Point Conditional Move on Floating Point False	if (cc[i] == 0) then Fd = Fs

* Instruction Format Type: S = Single, D = Double, W = Word, L = Longword

Table 13 5Kf Floating Point Instruction Set (Continued)

Instruction	Format*	Description	Function
MOVN.fmt	S, D	Floating Point Conditional Move on Non-Zero	if (Rt != 0) then Fd = Fs
MOVT.fmt	S, D	Floating Point Conditional Move on Floating Point True	if (cc[i] = 1) then Fd = Fs
MOVZ.fmt	S, D	Floating Point Conditional Move on Zero	if (Rt == 0) then Fd = Fs
MSUB.fmt	S, D	Floating Point Multiply Subtract	Fd = Fs * Ft - Fr
MTC1		Move Word to FPR	Fs = Rt
MUL.fmt	S, D	Floating Point Multiply	Fd = Fs * Ft
NEG.fmt	S, D	Floating Point Negate	Fd = neg(Fs)
NMADD.fmt	S, D	Floating Point Negative Multiply Add	Fd = neg(Fs * Ft + Fr)
NMSUB.fmt	S, D	Floating Point Negative Multiply Subtract	Fd = neg(Fs * Ft - Fr)
RECIP.fmt	S, D	Floating Point Reciprocal Approximation	Fd = recip(Fs)
ROUND.L.fmt	S, D	Floating Point Round to Long Fixed Point	Fd = convert_and_round(Fs)
ROUND.W.fmt	S, D	Floating Point Round to Word Fixed Point	Fd = convert_and_round(Fs)
RSQRT.fmt	S, D	Floating Point Reciprocal Square Root Approximation	Fd = rsqrt(Fs)
SDC1		Store Doubleword to Floating Point	memory[base+offset] = Ft
SDXC1		Store Doubleword Indexed to Floating Point	memory[base+index] = Fs
SQRT.fmt	S, D	Floating Point Square Root	Fd = sqrt(Fs)
SUB.fmt	S, D	Floating Point Subtract	Fd = Fs - Ft
SUXC1		Store Doubleword Indexed Unaligned to Floating Point	memory[$(base+index)_{psize-1..3}$] = Fs
SWC1		Store Word to Floating Point	memory[base+offset] = Ft
SWXC1		Store Word Indexed to Floating Point	memory[base+index] = Fs
TRUNC.L.fmt	S, D	Floating Point Truncate to Long Fixed Point	Fd = convert_and_round(Fs)
TRUNC.W.fmt	S, D	Floating Point Truncate to Word Fixed Point	Fd = convert_and_round(Fs)

* Instruction Format Type: S = Single, D = Double, W = Word, L = Longword

5Kf Core Signal Descriptions

This section describes the signal interface of the 5Kf core. The pin direction key for the signal descriptions is shown in Table 14 below.

Table 14 5Kf Core Signal Direction Key

Dir	Description
I	Input to the 5Kf core, unless otherwise noted, sampled on the rising edge of the appropriate clock signal.
O	Output of the 5Kf core, unless otherwise noted, driven at the rising edge of the appropriate clock signal.
A	Asynchronous input that is synchronized by the core.
S	Static input to the 5Kf core. These signals are normally tied to either power or ground and should not change state while SI_ColdReset is deasserted.

The 5Kf core signals are listed by function in Table 15.

Table 15 5Kf Signal Descriptions

Signal Name	Type	Description
System Interface		
SI_ClkIn	I	Clock input. All inputs and outputs, except a few of the EJTAG signals, are sampled and/or asserted relative to the rising edge of this signal.
SI_ClkOut	O	Reference clock for the external bus interface. This clock signal is intended to provide a reference for de-skewing any clock insertion delay created by the internal clock buffering in the 5Kf core.
SI_ColdReset	A	Hard reset signal. This signal must be asserted during either a power-on reset or a cold reset. The assertion of SI_ColdReset completely initializes the internal state machines of the 5Kf core without saving any state information. To get predictable results during a reset operation, the power supply must be stable and the SI_ClkIn input clock to the 5Kf core running before SI_ColdReset is deasserted. When SI_ColdReset is deasserted, a reset exception is taken by the 5Kf core.
SI_Endian	S	Indicates the base endianess of the 5Kf core. SI_Endian: 0 = Little Endian; 1 = Big Endian

Table 15 5Kf Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>SI_SimpleBE[1:0]</td><td>S</td><td>The state of these signals can constrain the core to only generate certain byte enables on EC™ interface transactions. This eases connection to some existing bus standards.
<table>
<tr><th>SI_SimpleBE[1:0]</th><th>Byte Enable Mode</th></tr>
<tr><td>0</td><td>Byte enable patterns that match the patterns generated by load and store instuctions.</td></tr>
<tr><td>1</td><td>Naturally aligned bytes, half-words, words and dwords only</td></tr>
<tr><td>2</td><td>Reserved</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>SI_ERL</td><td>O</td><td>This signal represents the state of the ERL bit in the CP0 Status register and indicates the error level. The 5Kf core asserts SI_ERL whenever any exception other than a Reset, Soft Reset, NMI, or Cache Error exception is taken.</td></tr>
<tr><td>SI_EXL</td><td>O</td><td>This signal represents the state of the EXL bit in the CP0 Status register and indicates the exception level. SI_EXL is asserted when an exception is taken and this exception is not a Reset, Soft Reset, NMI, Cache Error, or debug exception.</td></tr>
<tr><td>SI_Int[5:0]</td><td>I</td><td>When asserted, these signals indicate the corresponding interrupt request to the 5Kf core.</td></tr>
<tr><td>SI_NMI</td><td>I</td><td>When sampled asserted, this signal causes the 5Kf core to take an NMI exception. After the NMI exception is taken, SI_NMI must be deasserted before it can cause another NMI exception.</td></tr>
<tr><td>SI_PRIdOpt[7:0]</td><td>I</td><td>This signals is used as the upper 8 bits of the CP0 PrID register.</td></tr>
<tr><td>SI_Reset</td><td>A</td><td>Warm reset signal. This signal must be asserted for a warm reset. When asserted, a soft reset exception is asserted to the 5Kf core. A warm reset operation restarts the 5Kf core and initializes almost all the CP0 state initialized by hard reset.</td></tr>
<tr><td>SI_RP</td><td>O</td><td>This signal represents the state of the RP bit in the CP0 Status register.</td></tr>
<tr><td>SI_Sleep</td><td>O</td><td>This signal is asserted by the 5Kf core whenever the WAIT instruction is executed. The assertion of this signal indicates that the clock has stopped and that the 5Kf core is in power-down mode.</td></tr>
<tr><td>SI_TimerInt</td><td>O</td><td>This signal is asserted when the Count and Compare registers first match and is deasserted when the compare register is written.</td></tr>
<tr><td colspan="3">EC™ Interface</td></tr>
<tr><td>EB_A[35:3]</td><td>O</td><td>Address bus. Only valid when EB_AValid is asserted.</td></tr>
<tr><td>EB_ARdy</td><td>I</td><td>Assertion of this signal indicates whether the external logic is ready for a new address. The 5K core does not complete the address phase until the clock cycle after EB_ARdy is sampled asserted.</td></tr>
</table>

Table 15 5Kf Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>EB_AValid</td><td>O</td><td>Assertion of this signal indicates that the values on the address bus and access type lines are valid (signifying an address phase is ongoing). EB_AValid is always valid and cannot be deasserted between address phases within a burst.</td></tr>
<tr><td>EB_BE[7:0]</td><td>O</td><td>Indicates which bytes of the EB_RData or EB_WData buses are involved in the data phase corresponding to the current address phase. If an EB_BE signal is asserted, the associated byte is being read or written. EB_BE lines are only valid while EB_AValid is asserted.
During bursts all lines must be asserted.
The tables below lists the values that EB_BE can take in default mode and in SimpleBE mode.
<table>
<tr><th colspan="4">Byte enables supported, SI_SimpleBE[1:0]=0</th></tr>
<tr><td>00000001</td><td>00000010</td><td>00000100</td><td>00001000</td></tr>
<tr><td>00010000</td><td>00100000</td><td>01000000</td><td>10000000</td></tr>
<tr><td>11000000</td><td>00110000</td><td>00001100</td><td>00000011</td></tr>
<tr><td>11100000</td><td>01110000</td><td>00001110</td><td>00000111</td></tr>
<tr><td>11110000</td><td>00001111</td><td>11111000</td><td>00011111</td></tr>
<tr><td>11111100</td><td>00111111</td><td>11111110</td><td>01111111</td></tr>
<tr><td>11111111</td><td></td><td></td><td></td></tr>
</table>
<table>
<tr><th colspan="4">Byte enables supported, SI_SimpleBE[1:0]=1</th></tr>
<tr><td>00000001</td><td>00000010</td><td>00000100</td><td>00001000</td></tr>
<tr><td>00010000</td><td>00100000</td><td>01000000</td><td>10000000</td></tr>
<tr><td>00000011</td><td>00001100</td><td>00110000</td><td>11000000</td></tr>
<tr><td>00001111</td><td>11110000</td><td>11111111</td><td></td></tr>
</table>
<table>
<tr><th>EB_BE Signal</th><th>Read Data Bits Sampled</th><th>Write Data Bits Driven Valid</th></tr>
<tr><td>EB_BE[0]</td><td>EB_RData[7:0]</td><td>EB_WData[7:0]</td></tr>
<tr><td>EB_BE[1]</td><td>EB_RData[15:8]</td><td>EB_WData[15:8]</td></tr>
<tr><td>EB_BE[2]</td><td>EB_RData[23:16]</td><td>EB_WData[23:16]</td></tr>
<tr><td>EB_BE[3]</td><td>EB_RData[31:24]</td><td>EB_WData[31:24]</td></tr>
<tr><td>EB_BE[4]</td><td>EB_RData[39:32]</td><td>EB_WData[39:32]</td></tr>
<tr><td>EB_BE[5]</td><td>EB_RData[47:40]</td><td>EB_WData[47:40]</td></tr>
<tr><td>EB_BE[6]</td><td>EB_RData[55:48]</td><td>EB_WData[55:48]</td></tr>
<tr><td>EB_BE[7]</td><td>EB_RData[63:56]</td><td>EB_WData[63:56]</td></tr>
</table></td></tr>
<tr><td>EB_BFirst</td><td>O</td><td>Assertion of this signal indicates the address phase is the first address phase of a burst. EB_BFirst is always valid.</td></tr>
<tr><td>EB_BLast</td><td>O</td><td>Assertion of this signal indicates the address phase is the last address phase of a burst. Note that the time for assertion of EB_BLast is determined by use of EB_Burst, EB_BFirst, and EB_BLen. EB_BLast is always valid.</td></tr>
</table>

Table 15 5Kf Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>EB_BLen[1:0]</td><td>O</td><td>EB_BLen[1:0] indicate the length (number of address/data phases) of the burst. This signal is an implementation-specific static output.
<table>
<tr><th>EB_BLength[1:0]</th><th>Burst Length</th></tr>
<tr><td>0</td><td>reserved</td></tr>
<tr><td>1</td><td>4</td></tr>
<tr><td>2</td><td>reserved</td></tr>
<tr><td>3</td><td>reserved</td></tr>
</table></td></tr>
<tr><td>EB_Burst</td><td>O</td><td>Assertion of this signal indicates that the current address phase is for a cache fill or a write burst. EB_Burst is always valid.</td></tr>
<tr><td>EB_BusClkActive</td><td>I</td><td>Must be driven HIGH.</td></tr>
<tr><td>EB_EWBE</td><td>I</td><td>Indicates that all external write buffers are empty. The external write buffers must deassert EB_EWBE in the cycle following the assertion of the corresponding EB_WDRdy and keep EB_EWBE deasserted until the external write buffers are empty.</td></tr>
<tr><td>EB_Instr</td><td>O</td><td>Assertion of this signal indicates that the address is for an instruction fetch as opposed to a data read. EB_Instr is only valid when EB_AValid is asserted.</td></tr>
<tr><td>EB_RBErr</td><td>I</td><td>Bus error indicator for read transactions. EB_RBErr is always valid. Only assert it in the same cycle that the corresponding EB_RdVal is asserted.</td></tr>
<tr><td>EB_RData[63:0]</td><td>I</td><td>Read data bus. Valid at the end of a read data phase (on the rising clock edge where EB_RdVal is sampled asserted).</td></tr>
<tr><td>EB_RdVal</td><td>I</td><td>Assertion of this signal indicates that the external logic is driving read data on EB_RData (it ends a read data phase). EB_RdVal must always be valid. EB_RdVal must never be asserted until after the corresponding EB_ARdy is sampled asserted.</td></tr>
<tr><td>EB_SBlock</td><td>SI</td><td>When this signal is asserted, sub-block ordering is used for addressing bursts. When this signal is deasserted, sequential addressing is used.</td></tr>
<tr><td>EB_WBErr</td><td>I</td><td>Bus error indicator for write transactions. EB_WBErr is always valid. Only assert it in the cycle following an asserted sample of the corresponding EB_WDRdy.</td></tr>
<tr><td>EB_WData[63:0]</td><td>O</td><td>Write data bus. Kept unchanged and stable during a write data phase until the write data phase ends (the positive clock edge following an asserted sample of EB_WDRdy).</td></tr>
<tr><td>EB_WDRdy</td><td>I</td><td>Assertion of this signal indicates that the external logic is ready to process a write; it ends a write data phase and the EB_WData can change after the positive clock edge that follows the positive clock edge where EB_WDRdy is sampled asserted. EB_WDRdy is not sampled until the rising edge where the corresponding EB_ARdy is sampled asserted.</td></tr>
<tr><td>EB_Write</td><td>O</td><td>Assertion of this signal indicates that the address phase is for a write. Deassertion of this signal indicates that the address phase is for a read. This signal is only valid when EB_AValid is asserted.</td></tr>
</table>

Table 15 5Kf Signal Descriptions (Continued)

Signal Name	Type	Description
EB_WWBE	O	Assertion of this signal indicates that the 5K core is waiting for external write buffers to empty. EB_WWBE can be asserted when EB_EWBE is asserted, but if EB_EWBE is deasserted and EB_WWBE is asserted, EB_EWBE must be asserted eventually.
Coprocessor Interface: Instruction Dispatch		
CP2_as_0	O	Coprocessor 2 Arithmetic Instruction Strobe Asserted in the cycle after an arithmetic coprocessor 2 instruction is available on CP2_ir_0. If CP2_abusy_0 was asserted in the previous cycle, this signal will not be asserted.
CP2_abusy_0	I	Coprocessor 2 Arithmetic Busy When asserted, a coprocessor 2 arithmetic instruction will not be dispatched. CP2_as_0 will not be asserted in the cycle after this signal is asserted.
CP2_ts_0	O	Coprocessor 2 To Strobe Asserted in the cycle after a To COP2 Op instruction is available on CP2_ir_0. If CP2_tbusy_0 was asserted in the previous cycle, this signal will not be asserted.
CP2_tbusy_0	I	To Coprocessor 2 Busy When asserted, a To COP2 Op will not be dispatched. CP2_ts_0 will not be asserted in the cycle after this signal is asserted.
CP2_fs_0	O	Coprocessor 2 From Strobe Asserted in the cycle after a From COP2 Op instruction is available on CP2_ir_0. If CP2_fbusy_0 was asserted in the previous cycle, this signal will not be asserted.
CP2_fbusy_0	I	From Coprocessor 2 Busy When asserted, a From COP2 Op will not be dispatched. CP2_fs_0 will not be asserted in the cycle after this signal is asserted.
CP2_ir_0[31:0]	O	Coprocessor Instruction Word Valid in the cycle before CP2_as_0, CP2_ts_0, or CP2_fs_0 is asserted.
CP2_irenable_0	O	Enable Instruction Registering When deasserted, no instruction strobes will be asserted in the following cycle. When asserted, there may be an instruction strobe asserted in the following cycle. Instruction strobes include CP2_as_0, CP2_ts_0, and CP2_fs_0.
CP2_order_0[2:0]	O	Coprocessor Dispatch Order Since the 5Kf core is a single-issue machine, the value of this signal will always be 3’b0. Valid when CP2_as_0, CP2_ts_0, or CP2_fs_0 is asserted.
CP2_inst32_0	O	MIPS32 Compatibility Mode - Instructions When asserted, the dispatched instruction is restricted to the MIPS32 subset of instructions. Please refer to the MIPS64 ISA specification for a complete description of MIPS32 compatibility mode. Valid the cycle before CP2_as_0, CP2_fs_0, or CP2_ts_0 is asserted.

Table 15 5Kf Signal Descriptions (Continued)

Signal Name	Type	Description
CP2_endian_0	O	Big-Endian Byte Ordering When asserted, the processor is using big-endian byte ordering for the dispatched instruction. When deasserted, the processor is using little-endian byte ordering. Valid the cycle before CP2_as_0, CP2_fs_0, or CP2_ts_0 is asserted.
CP2_as_1	O	Coprocessor 2 Arithmetic Instruction Strobe Asserted in the cycle after an arithmetic coprocessor 2 instruction is available on CP2_ir_1. If CP2_abusy_1 was asserted in the previous cycle, this signal will not be asserted.
CP2_abusy_1	I	Coprocessor 2 Arithmetic Busy When asserted, a coprocessor 2 arithmetic instruction will not be dispatched. CP2_as_1 will not be asserted in the cycle after this signal is asserted.
CP2_ir_1[31:0]	O	Coprocessor Instruction Word Valid in the cycle before CP2_as_1 is asserted.
CP2_irenable_1	O	Enable Instruction Registering When deasserted, no instruction strobes will be asserted in the following cycle. When asserted, there may be an instruction strobe asserted in the following cycle. Instruction strobe is CP2_as_1.
CP2_order_1[2:0]	O	Coprocessor Dispatch Order Since the 5Kf core is a single-issue machine, the value of this signal will always be 3’b0. Valid when CP2_as_1 is asserted.
CP2_adisable_1	S	Inhibit Arithmetic Dispatch When asserted, arithmetic instructions are dispatched using Issue Group 0. When deasserted, arithmetic instructions are dispatched using Issue Group 1.
CP2_inst32_1	O	MIPS32 Compatibility Mode - Instructions When asserted, the dispatched instruction is restricted to the MIPS32 subset of instructions. Please refer to the MIPS64 architecture specification for a complete description of MIPS32 compatibility mode. Valid the cycle before CP2_as_1 is asserted.
CP2_endian_1	O	Big-Endian Byte Ordering When asserted, the processor is using big-endian byte ordering for the dispatched instruction. When deasserted, the processor is using little-endian byte ordering. Valid the cycle before CP2_as_1 is asserted.
Coprocessor Interface: To Coprocessor Data (For all To COP Ops)		
CP2_tds_0	O	Coprocessor To Data Strobe Asserted when To COP Op data is available on CP2_tdata_0.

Table 15 5Kf Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>CP2_torder_0[2:0]</td><td>O</td><td>Coprocessor To Order
Specifies which outstanding To COP Op the data is for. The 5Kf core will never drive this signal to a value greater than 3’b1. Valid only when CP2_tds_0 is asserted.
<table>
<tr><th>CP2_torder_0</th><th>Order</th></tr>
<tr><td>3’b000</td><td>Oldest outstanding
To COP Op data transfer</td></tr>
<tr><td>3’b001</td><td>2nd oldest
To COP Op data transfer</td></tr>
<tr><td>3’b010</td><td>Reserved</td></tr>
<tr><td>3’b011</td><td>Reserved</td></tr>
<tr><td>3’b100</td><td>Reserved</td></tr>
<tr><td>3’b101</td><td>Reserved</td></tr>
<tr><td>3’b110</td><td>Reserved</td></tr>
<tr><td>3’b111</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>CP2_tordlim_0[2:0]</td><td>S</td><td>To Coprocessor Data Out-of-Order Limit
This signal forces the integer processor core to limit how much it can reorder To COP Data. The value on this signal corresponds to the maximum allowed value to be used on CP2_torder_0[2:0].</td></tr>
<tr><td>CP2_tdata_0[63:0]</td><td>O</td><td>To Coprocessor Data
Data to be transferred to the coprocessor. For single-word transfers, data is valid on CP2_tdata_0[31:0]. Valid when CP2_tds_0 is asserted.</td></tr>
<tr><td colspan="3">Coprocessor Interface: From Coprocessor Data (For all From COP Ops)</td></tr>
<tr><td>CP2_fds_0</td><td>I</td><td>Coprocessor From Data Strobe
Asserted when From COP Op data is available on CP2_fdata_0.</td></tr>
<tr><td>CP2_forder_0[2:0]</td><td>O</td><td>Coprocessor From Order
Specifies which outstanding From COP Op the data is for. The 5Kf core does not support values greater than 3’b1. Valid only when CP2_fds_0 is asserted.
<table>
<tr><th>CP2_forder_0</th><th>Order</th></tr>
<tr><td>3’b000</td><td>Oldest outstanding
From COP Op data transfer</td></tr>
<tr><td>3’b001</td><td>2nd oldest
From COP Op data transfer</td></tr>
<tr><td>3’b010</td><td>Reserved</td></tr>
<tr><td>3’b011</td><td>Reserved</td></tr>
<tr><td>3’b100</td><td>Reserved</td></tr>
<tr><td>3’b101</td><td>Reserved</td></tr>
<tr><td>3’b110</td><td>Reserved</td></tr>
<tr><td>3’b111</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Table 15 5Kf Signal Descriptions (Continued)

Signal Name	Type	Description
CP2_fordlim_0[2:0]	O	From Coprocessor Data Out-of-Order Limit This signal forces the coprocessor to limit how much it can reorder From COP Data. The value on this signal corresponds to the maximum allowed value to be used on CP2_forder_0[2:0]. The 5Kf core drives this signal to 3'b001 as a static output.
CP2_fdata_0[63:0]	I	From Coprocessor Data Data to be transferred from coprocessor. For single-word transfers, data is valid on CP2_fdata_0[31:0]. Valid when CP2_fds_0 is asserted.
Coprocessor Interface: Coprocessor Condition Code Check (Only for BC1, MOVCI, BC2 Ops)		
CP2_cccs_0	I	Coprocessor Condition Code Check Strobe Asserted when condition code check results are available on CP2_ccc_0.
CP2_ccc_0	I	Coprocessor Condition Code Check Valid when CP2_cccs_0 is asserted. When asserted, the instruction checking the condition code should proceed with its execution. (i.e. branch or move data) When deasserted, the instruction should not execute its conditional operation. (i.e. do not branch and do not move data)
CP2_cccs_1	I	Coprocessor Condition Code Check Strobe Asserted when condition code check results are available on CP2_ccc_1.
CP2_ccc_1	I	Coprocessor Condition Code Check Valid when CP2_cccs_1 is asserted. When asserted, the instruction checking the condition code should proceed with its execution. (i.e. branch or move data) When deasserted, the instruction should not execute its conditional operation. (i.e. do not branch and do not move data)
Coprocessor Interface: Coprocessor Exceptions		
CP2_excs_0	I	Coprocessor Exception Strobe Asserted when coprocessor exception signalling is available on CP2_exc_0.
CP2_exc_0	I	Coprocessor Exception When deasserted, the coprocessor is not causing an exception. When asserted, signifies that the coprocessor is causing an exception. The type of exception is encoded on the signal CP2_exccode_0[4:0]. Valid when CP2_excs_0 is asserted.

Table 15 5Kf Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>CP2_exccode_0[4:0]</td><td>I</td><td>Coprocessor Exception Code
Valid when CP2_excs_0 is asserted and CP2_exc_0 is asserted.
<table>
<tr><th>CP2_exccode_0</th><th>Exception</th></tr>
<tr><td>5’b01010</td><td>Reserved Instruction Exception</td></tr>
<tr><td>5’b01111</td><td>Floating Point Exception</td></tr>
<tr><td>5’b10000</td><td>Available for implementation-specific use</td></tr>
<tr><td>5’b10001</td><td>Available for implementation-specific use</td></tr>
<tr><td>5’b10010</td><td>COP2 Exception</td></tr>
<tr><td>other values</td><td>Reserved
If other values are signalled, the operation of the integer processor core is UNPREDICTABLE.</td></tr>
</table></td></tr>
<tr><td>CP2_excs_1</td><td>I</td><td>Coprocessor Exception Strobe
Asserted when coprocessor exception signalling is available on CP2_exc_1.</td></tr>
<tr><td>CP2_exc_1</td><td>I</td><td>Coprocessor Exception
When deasserted, the coprocessor is not causing an exception. When asserted, signifies that the coprocessor is causing an exception. The type of exception is encoded on the signal CP2_exccode_1[4:0]. Valid when CP2_excs_1 is asserted.</td></tr>
<tr><td>CP2_exccode_1[4:0]</td><td>I</td><td>Coprocessor Exception Code
Valid when CP2_excs_1 is asserted and CP2_exc_1 is asserted.
<table>
<tr><th>CP2_exccode_1</th><th>Exception</th></tr>
<tr><td>5’b01010</td><td>Reserved Instruction Exception</td></tr>
<tr><td>5’b01111</td><td>Floating Point Exception</td></tr>
<tr><td>5’b10000</td><td>Available for implementation-specific use</td></tr>
<tr><td>5’b10001</td><td>Available for implementation-specific use</td></tr>
<tr><td>5’b10010</td><td>COP2 Exception</td></tr>
<tr><td>other values</td><td>Reserved
If other values are signalled, the operation of the integer processor core is UNPREDICTABLE.</td></tr>
</table></td></tr>
<tr><td colspan="3">Coprocessor Interface: Instruction Nullification</td></tr>
<tr><td>CP2_nulls_0</td><td>O</td><td>Coprocessor Null Strobe
Asserted when a nullification signal is available on CP2_null_0.</td></tr>
<tr><td>CP2_null_0</td><td>O</td><td>Nullify Arithmetic or To/From Coprocessor Instruction
When deasserted, the integer processor core is signalling that the instruction is not nullified. When asserted, the integer processor core is signalling that the instruction is nullified. Valid when CP2_nulls_0 is asserted.</td></tr>
<tr><td>CP2_nulls_1</td><td>O</td><td>Coprocessor Null Strobe
Asserted when a nullification signal is available on CP2_null_1.</td></tr>
</table>

Table 15 5Kf Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>CP2_null_1</td><td>O</td><td>Nullify Arithmetic Coprocessor Instruction
When deasserted, the integer processor core is signalling that the instruction is not nullified. When asserted, the integer processor core is signalling that the instruction is nullified. Valid when CP2_nulls_1 is asserted.</td></tr>
<tr><td colspan="3">Coprocessor Interface: Instruction Killing</td></tr>
<tr><td>CP2_kills_0</td><td>O</td><td>Coprocessor Kill Strobe
Asserted when kill signalling is available on CP2_kill_0.</td></tr>
<tr><td>CP2_kill_0[1:0]</td><td>O</td><td>Kill Coprocessor Instruction
Valid when CP2_kills_0 is asserted.
<table>
<tr><th>CP2_kill_0[1:0]</th><th>Type of Kill</th></tr>
<tr><td>00</td><td rowspan="2">Instruction is not killed and can commit its results</td></tr>
<tr><td>01</td></tr>
<tr><td>10</td><td>Instruction is killed.
(not due to CP2_exc_0)</td></tr>
<tr><td>11</td><td>Instruction is killed
(due to CP2_exc_0)</td></tr>
</table></td></tr>
<tr><td>CP2_kills_1</td><td>O</td><td>Coprocessor Kill Strobe
Asserted when kill signalling is available on CP2_kill_1.</td></tr>
<tr><td>CP2_kill_1[1:0]</td><td>O</td><td>Kill Coprocessor Instruction
Valid when CP2_kills_1 is asserted.
<table>
<tr><th>CP2_kill_1[1:0]</th><th>Type of Kill</th></tr>
<tr><td>00</td><td rowspan="2">Instruction is not killed and can commit its results</td></tr>
<tr><td>01</td></tr>
<tr><td>10</td><td>Instruction is killed.
(not due to CP2_exc_1)</td></tr>
<tr><td>11</td><td>Instruction is killed
(due to CP2_exc_1)</td></tr>
</table></td></tr>
<tr><td colspan="3">Coprocessor Interface: Miscellaneous</td></tr>
<tr><td>CP2_reset</td><td>O</td><td>Coprocessor Reset
Asserted when a hard or soft reset is performed by the integer processor core. At a minimum, this signal will be asserted for 1 cycle.</td></tr>
<tr><td>CP2_present</td><td>S</td><td>COP2 Present
Must be asserted when COP2 hardware is connected to the Coprocessor Interface.</td></tr>
<tr><td>CP2_idle</td><td>I</td><td>Coprocessor Idle
Asserted when the coprocessor logic is idle. Enables the integer processor core to go into sleep mode and shut down the internal integer processor core clock. Valid only if CP2_present is asserted.</td></tr>
</table>

Table 15 5Kf Signal Descriptions (Continued)

Signal Name	Type	Description
CP2_tx32	I	Coprocessor 32-bit Transfers When asserted, the integer unit will signal an RI exception for 64bit COP2 TF instructions. This is a static input and must always be valid.
EJTAG Interface		
EJ_DebugM	O	This signal is asserted by the 5Kf core whenever it is in Debug Mode.
EJ_DINT	I	Debug exception request when this signal is asserted in a CPU clock period after being deasserted in the previous CPU clock period. The request is cleared when Debug Mode is entered. Requests when in Debug Mode are ignored.
EJ_DINTsup	S	Indicates if debug interrupts requested from the probe through the assertion of EJ_DINT are supported.
EJ_ManufID[10:0]	S	Value of the ManufID[10:0] field in the EJTAG TAP Device ID register.
EJ_PartNumber[15:0]	S	Value of the PartNumber[15:0] field in the EJTAG TAP Device ID register.
EJ_PerRst	O	Implementation-dependent peripheral reset. Has no effect on the core.
EJ_PrRst	O	Implementation-dependent processor reset. Has no effect on the core.
EJ_SRstE	O	Implementation-dependent soft reset enable. Has no effect on the core.
EJ_TCK	I	Test Clock Input for the EJTAG TAP.
EJ_TDI	I	Test Data Input for the EJTAG TAP.
EJ_TDO	O	Test Data Output for the EJTAG TAP.
EJ_TDOzstate	O	Output drive indication for the chip pin outputting the EJ_TDO signal. When asserted, the chip pin outputting EJ_TDO must be 3-stated.
EJ_TMS	I	Test Mode Select Input for the EJTAG TAP.
EJ_TRST_N	I	Test Reset Input for the EJTAG TAP. The EJ_TRST_N must be asserted at power-up of the 5Kf core to reset the test access port.
EJ_Version[3:0]	S	Value of the Version[3:0] field in the EJTAG TAP Device ID register.
Performance Monitoring Interface		
PM_DCacheHit	O	This signal is asserted whenever there is a data cache hit.
PM_DCacheMiss	O	This signal is asserted whenever there is a data cache miss.
PM_DTLBHit	O	This signal is asserted whenever there is a data TLB hit.
PM_DTLBMiss	O	This signal is asserted whenever there is a data TLB miss.
PM_ICacheHit	O	This signal is asserted whenever there is an instruction cache hit.
PM_ICacheMiss	O	This signal is asserted whenever there is an instruction cache miss.
PM_InstnComplete	O	This signal is asserted each time an instruction completes in the pipeline.
PM_ITLBHit	O	This signal is asserted whenever there is an instruction TLB hit.
PM_ITLBMiss	O	This signal is asserted whenever there is an instruction TLB miss.

Table 15 5Kf Signal Descriptions (Continued)

Signal Name	Type	Description
PM_JTLBHit	O	This signal is asserted whenever there is a joint TLB hit.
PM_JTLBMiss	O	This signal is asserted whenever there is a joint TLB miss.
Production Test Interface for Internal Scan and Memory Built-In Self Test (BIST)		
BistIn[]	I	Configurable width signal bus for user implemented BIST of internal RAMs.
BistOut[]	O	Configurable width signal bus for user implemented BIST of internal RAMs.
MemBistInvoke	I	Invoke signal for integrated memory BIST of internal cache RAMs.
MemBistDone	O	Done signal for integrated memory BIST of internal cache RAMs.
MemBistFail	O	Fail indication signal for integrated memory BIST of internal cache RAMs.
ScanEnable	I	This signal should be asserted while scanning vectors into or out of the core. The ScanEnable signal must be deasserted during normal operation and during capture clocks in test mode.
ScanIn[]	I	Configurable width signal bus used for scan chain input.
ScanMode	I	This signal should be asserted during all scan testing, both while scanning and during capture clocks. The ScanMode signal must be deasserted during normal operation.
ScanOut[]	O	Configurable width signal bus used for scan chain output.

5Kf EC Interface Transactions

The 5Kf core implements unidirectional data buses: EB_RData[63:0] for read operations and EB_WData[63:0] for write operations. It can cause a maximum number of 16 outstanding bus transactions. The following sections describe four basic bus transactions: single read, single write, burst read, and burst write.

Single Read

Figure 11 shows the basic timing relationships of signals during a single read transaction. During a single read cycle, the 5Kf core drives address onto EB_A[35:3] and byte enable information onto EB_BE[7:0]. The EB_ARdy input signal is driven by external logic and controls the generation of addresses on the bus.

The address is driven whenever it becomes available, regardless of the state of EB_ARdy. However, the 5Kf core always continues to drive the address until the clock after EB_ARdy is sampled asserted. For example, at the rising edge of the clock 2 in Figure 11, the EB_ARdy signal is sampled low, indicating that external logic is not ready to accept the new address.

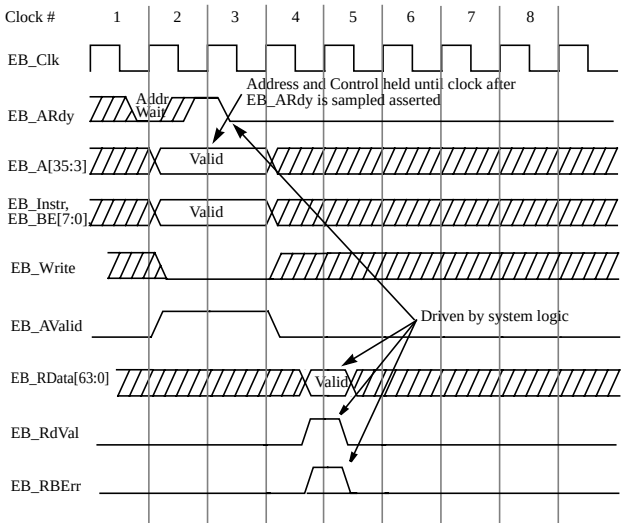

Figure 11 Single Read Transaction Timing Diagram

However, the 5Kf core still drives EB_A[35:3] in this clock as shown. At the rising edge of the clock 3 the 5Kf core samples EB_ARdy asserted and continues to drive the address until the rising edge of clock 4.

The EB_Instr signal is only asserted during a single read cycle if there is an instruction fetch from non-cacheable memory space. The EB_AValid signal is asserted in each clock that EB_A[35:3], EB_BE[7:0]. EB_Instr, and EB_Write are valid on the bus. The 5Kf core drives the EB_Write signal low to indicate a read transaction.

The EB_RData[63:0] and EB_RdVal signals are first sampled at the rising edge of clock 4, one clock after EB_ARdy is sampled asserted. Data is sampled on every clock thereafter until EB_RdVal is sampled asserted.

If a bus error occurs during the data transaction, external logic asserts the EB_RBErr signal in the same clock as EB_RdVal.

Single Write

Figure 12 shows a typical write transaction. The 5Kf core drives address and control information onto the EB_A[35:3] and EB_BE[7:0] signals at the rising edge of clock 2. As in the single read cycle, these signals are valid until the clock edge after the EB_ARdy signal is sampled asserted. The 5Kf core asserts the EB_Write signal to indicate that a valid write cycle is on the bus, and EB_AValid to indicate that a valid address is on the bus.

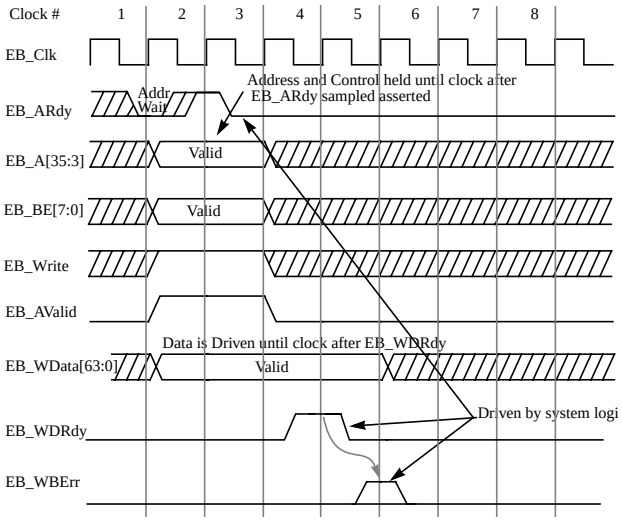

Figure 12 Single Write Transaction Timing Diagram

The 5Kf core drives write data onto EB_WData[63:0] in the same clock as the address and continues to drive data until the clock edge after the EB_WDRdy signal is sampled asserted. If a subsequent write transaction is started while the 5Kf core is waiting for EB_WDRdy to be asserted by external logic, the corresponding data will not be driven until the clock after EB_WDRdy is sampled asserted. If a bus error occurs during a write operation, external logic asserts the EB_WBErr signal one clock after asserting EB_WDRdy.

Burst Read

The 5Kf core is capable of generating burst transactions on the bus. A burst transaction is used to transfer multiple data items in one transaction.

Figure 13 shows an example of a burst read transaction. Burst read transactions initiated by the 5Kf core always contain four data transfers in sequence. In addition, the data requested is always a 32 byte-aligned block.

The order of words within this 32-byte block varies depending on which of the words in the block is being requested by the execution unit and the ordering protocol selected. The burst starts with the word requested by the execution unit and proceeds in a predetermined address order as shown in Table 16.

Table 16 Address Ordering Protocols

Starting Address EB_A[4:3]	Sequential Addressing EB_A[4:3] (EB_SBlock = 0)	Subblock Addressing EB_A[4:3] (EB_SBlock = 1)
00	00, 01, 10, 11	00, 01, 10, 11
01	01, 10, 11, 00	01, 00, 11, 10
10	10, 11, 00, 01	10, 11, 00, 01
11	11, 00, 01, 10	11, 10, 01, 00

The 5Kf core drives address and control information onto the EB_A[35:3] and EB_BE[7:0] signals at the rising edge of clock 2. As in the single read cycle, these signals are valid until the clock edge after the EB_ARdy signal is sampled asserted. The 5Kf core continues to drive EB_AValid as long as a valid address is on the bus.

The EB_Instr signal is asserted if the cycle is an instruction fetch. The EB_Burst signal is asserted throughout the cycle to indicate that a burst transaction is in progress. The 5Kf core asserts the EB_BFirst signal in the same clock as address 1 is driven to indicate the start of a burst cycle. The EB_Last signal is asserted along with the last address of the burst.

The processor samples EB_RData[63:0] on the next rising edge after EB_ARdy is sampled asserted, which in this example is the rising edge of clock 3. However, since EB_RDVal is deasserted in clock 3, data is sampled again at the rising edge of clock 4. External logic asserts EB_RdVal to indicate that valid data is on the bus. The 5Kf core latches data internally whenever EB_RdVal is sampled asserted.

Note that at the rising edge of clocks 3 and 6 in Figure 13, the EB_RdVal signal is sampled deasserted, causing a wait state before Data 1 and between Data 2 and Data 3. External logic asserts the EB_RBErr signal in the same clock as data if a bus error occurs during that data transfer.

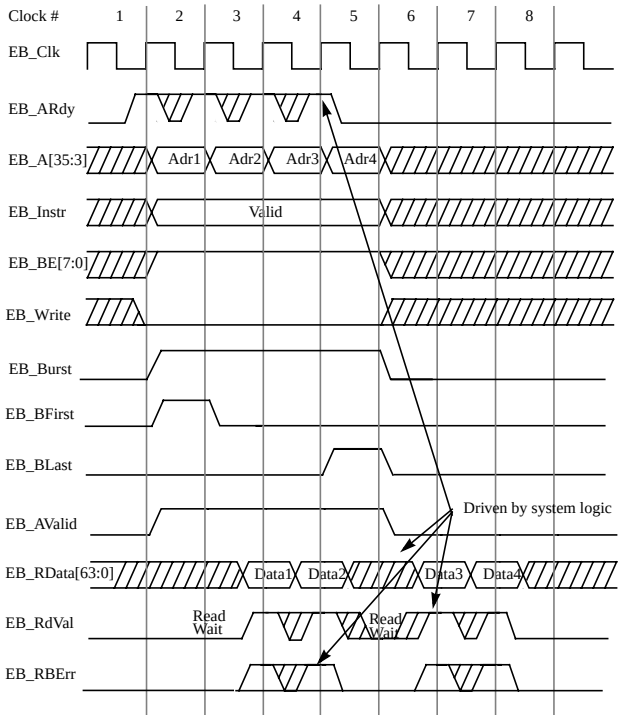

Figure 13 Burst Read Transaction Timing Diagram

Burst Write

Burst write transactions are used to empty one of the write buffers. A burst transaction is only performed if the write buffer contains 32 bytes of data associated with the same aligned memory block, otherwise individual write transactions are performed. Figure 14 shows a timing diagram of a burst write transaction. Unlike the read burst, a write burst always begins with EB_A[4:3] equal to 00b.

The 5Kf core drives address and control information onto the EB_A[35:3] and EB_BE[7:0] signals at the rising edge of clock 2. As in the single read cycle, these signals are valid until the clock edge after the EB_ARdy signal is sampled asserted. The 5Kf core continues to drive EB_AValid as long as a valid address and control signals are on the bus.

The 5Kf core asserts the EB_Write, EB_Burst, and EB_AValid signals during the time the address is driven. EB_Write indicates that a write operation is in progress. The assertion of EB_Burst indicates that the current operation is a burst.

The 5Kf core asserts the EB_BFirst signal in the same clock as address 1 is driven to indicate the start of a burst cycle. In the clock that the last address is driven, the 5Kf core asserts EB_BLast to indicate the end of the burst transaction.

In Figure 14, the first doubleword of data (Data1) is driven in clocks 2 and 3. The EB_WDRdy signal is sampled deasserted at the rising edge of clock 2, causing the processor to continue to drive data in clock 3. When EB_WDRdy is sampled asserted at the rising edge of clock 3, the 5Kf core responds by driving the second doubleword (Data2) in clock 4.

External logic drives the EB_WBErr signal one clock after the corresponding assertion of EB_WDRdy if a bus error has occurred as shown by the arrows in Figure 14.

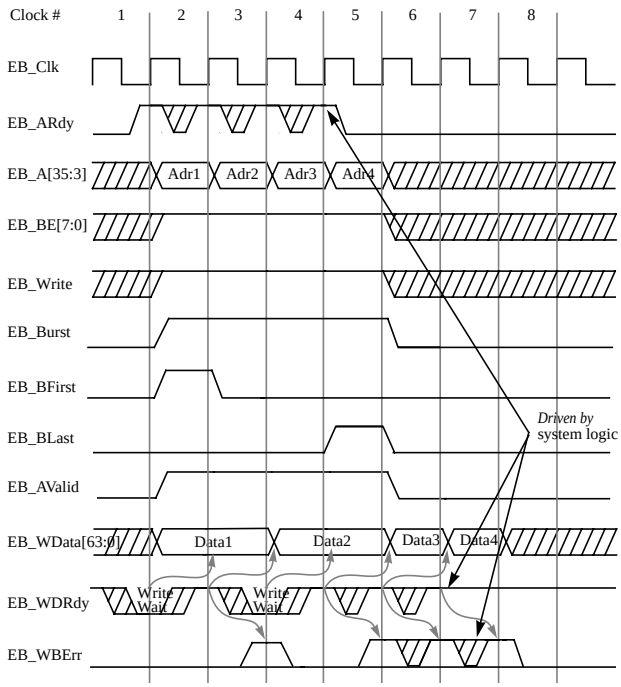

Figure 14 Burst Write Transaction Timing Diagram

Document Number: MD00109
01.03-2B