# MIPS32® 24Kf™ Processor Core Datasheet June 30, 2005

The MIPS32® 24Kf™ core from MIPS Technologies is a high-performance, low-power, 32-bit MIPS RISC core designed for custom system-on-silicon applications. The core is designed for semiconductor manufacturing companies, ASIC developers, and system OEMs who want to rapidly integrate their own custom logic and peripherals with a high-performance RISC processor. Fully synthesizable and highly portable across processes, it can be easily integrated into full system-on-silicon designs, allowing developers to focus their attention on end-user products.

The 24Kf core implements the MIPS32 Release 2 Architecture in an 8-stage pipeline. It includes support for the MIPS16e™ application specific extension and the 32-bit privileged resource architecture. This standard architecture allows support by a wide range of industry standard tools and development systems.

To maintain high pipeline utilization, dynamic branch prediction is included in the form of a Branch History Table and a Return Prediction Stack. The Memory Management Unit (MMU) contains 4-entry instruction and 8-entry data Translation Lookaside Buffers (ITLB/DTLB) and a configurable 16/32/64 dual-entry joint TLB (JTLB) with variable page sizes. Alternatively, for applications not requiring address mapping or protection, the TLBs can be replaced with a simple Fixed Mapping mechanism.

The 24Kf core also features an IEEE 754 compliant Floating Point Unit (FPU). The FPU supports both single and double precision instructions.

The synthesizable 24Kf core includes a high performance Multiply/Divide Unit (MDU). The MDU is fully pipelined to support a single cycle repeat rate for 32x32 MAC instructions, which enables multiply-intensive algorithms to be performed efficiently. Further, in the 24Kf Pro™ Core, the optional CorExtend block can utilize the HI/LO registers in the MDU block. The CorExtend block allows specialized functions to be efficiently implemented.

Instruction and data level-one caches are configurable at 0, 16, 32, or 64 KB in size. Each cache is organized as 4-way set associative. Data cache misses are non-blocking and up to 4 may be outstanding. Two instruction cache misses can be outstanding. Both caches are virtually indexed and physically tagged to allow them to be accessed in the same cycle that the address is translated. To achieve high frequencies while using commercially available SRAM generators, the cache access is spread across two pipeline stages, leaving nearly an entire cycle for the SRAM access.

The Bus Interface Unit implements the Open Core Protocol (OCP) which has been developed to address the needs of SOC designers. This implementation features 64-bit read and write data buses to efficiently transfer data to and from the L1 caches. The BIU also supports a variety of core/bus clock ratios to give greater flexibility for system design implementations.

Optional interfaces are supported to external scratchpad or coprocessor blocks.

An Enhanced JTAG (EJTAG) version 3.10 compliant block allows for software debugging of the processor and includes a TAP controller as well as optional instruction and data virtual address/value breakpoints. Additionally, real-time tracing of instruction program counter, data address and data values can be supported.

Figure 1 shows a block diagram of the 24Kf core.

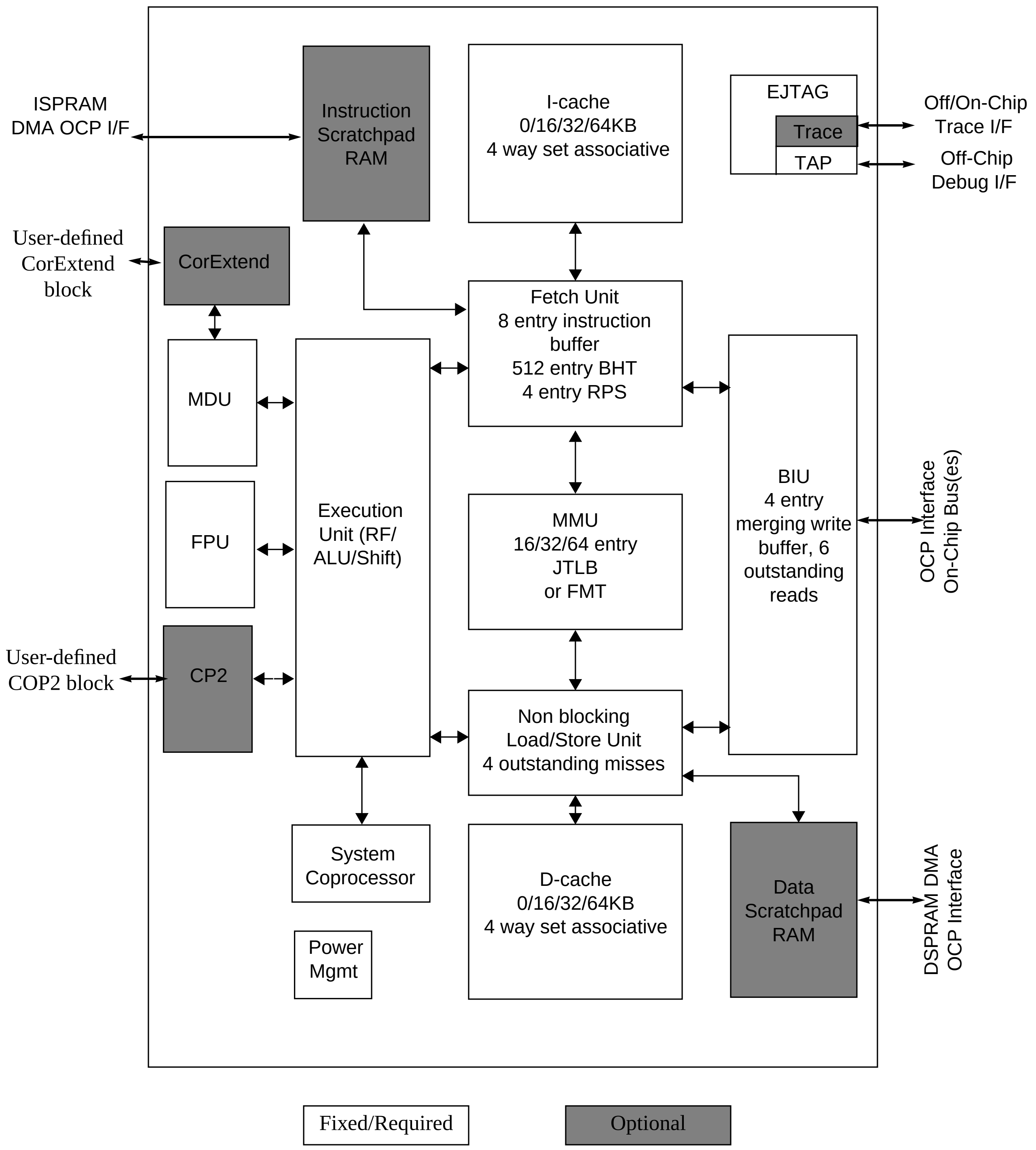


Figure 1 24Kf™ Core Block Diagram

## 24Kf™ Core Features

- 8-stage pipeline
- 32-bit address paths
- 64-bit data paths to caches and external interface
- MIPS32-Compatible Instruction Set
  - Multiply-Accumulate and Multiply-Subtract Instructions (MADD, MADDU, MSUB, MSUBU)
  - Targeted Multiply Instruction (MUL)
  - Zero/One Detect Instructions (CLZ, CLO)
  - Wait Instruction (WAIT)
  - Conditional Move Instructions (MOVZ, MOVN)
  - Prefetch Instruction (PREF)
- MIPS32 Enhanced Architecture (Release 2) Features
  - Vectored interrupts and support for external interrupt controller
  - Programmable exception vector base
  - Atomic interrupt enable/disable
  - GPR shadow registers (optionally, one or three additional shadows can be added to minimize latency for interrupt handlers)
  - Bit field manipulation instructions
- MIPS32 Privileged Resource Architecture
- Programmable Memory Management Unit
  - 16/32/64 dual-entry JTLB with variable page sizes
  - 4-entry ITLB
  - 8-entry DTLB
  - Optional simple Fixed Mapping Translation (FMT) mechanism
- MIPS16e™ Code Compression
  - 16 bit encodings of 32 bit instructions to improve code density
  - Special PC-relative instructions for efficient loading of addresses and constants
  - SAVE & RESTORE macro instructions for setting up and tearing down stack frames within subroutines
  - Improved support for handling 8 and 16 bit datatypes
- Programmable L1 Cache Sizes
  - Individually configurable instruction and data caches
  - Instruction and Data cache sizes of 0/16/32/64 KB
  - 4-Way Set Associative
  - Up to 4 outstanding load misses
  - Write-back and write-through support
  - 32-byte cache line size
  - Virtually indexed, physically tagged
  - Cache line locking support
  - Non-blocking prefetches
  - Optional parity support
- Bus Interface
  - OCP 2.1 compliant
  - OCP interface with 32-bit address and 64-bit data
  - OCP interface runs at core/bus clock ratios of 1, 1.5, 2, 2.5, 3, 3.5, 4 or 5 via a separate synchronous bus clock
  - Burst size of four 64-bit beats
  - 4 entry write buffer
  - “Simple” byte enable mode allows easier bridging to other bus standards
  - Extensions for front-side L2 cache
- Scratchpad RAM support
  - Independent Instruction and Data Scratchpad RAM
  - Independent of cache configuration
  - Independent 64 bit OCP interface for external DMA
  - External interface runs at the same core/bus clock ratio as that of BIU interface
  - Maximum size of 1MB each
  - Interface allows back-stalling the core
- Multiply/Divide Unit
  - Maximum issue rate of one 32x32 multiply per clock
  - 5 cycle multiply latency
  - Early-in iterative divide. Minimum 12 and maximum 38 clock latency (dividend (*rs*) sign extension-dependent)
- CorExtend™ User Defined Instruction Set Extensions (available in 24Kf Pro™ core)
  - Allows user to define and add instructions to the core at build time
  - Maintains full MIPS32 compatibility
  - Supported by industry standard development tools
  - Single or multi-cycle instructions
  - Separately licensed; a core with this feature is known as the 24Kf Pro™ core
  - Implemented in same block as MDU, allows HI and LO registers to be shared for MIPS32 and CorExtend multiply operations.
- Floating Point Unit (FPU)
  - IEEE-754 compliant Floating Point Unit
  - Compliant to MIPS 64b FPU standards
  - Supports single and double precision datatypes
  - Optionally run at 1:1 or 2:1 core/FPU clock ratio
- Coprocessor 2 interface
  - 64 bit interface to a user designed coprocessor
- Power Control
  - Minimum frequency: 0 MHz
  - Power-down mode (triggered by WAIT instruction)
  - Support for software-controlled clock divider
  - Support for extensive use of local gated clocks
- EJTAG Debug
  - Support for single stepping
  - Virtual instruction and data address/value breakpoints
  - TAP controller is chainable for multi-CPU debug
  - Cross-CPU breakpoint support

  - EJTAG version 3.10 compliant
- MIPS Trace
  - PC, data address and data value tracing w/ trace compression
  - Support for on-chip and off-chip trace memory
  - PDTrace version 4.1 compliant
- Testability
  - Full scan design achieves test coverage in excess of 99% (dependent on library and configuration options)
  - Optional memory BIST for internal SRAM arrays

## Architecture Overview

The 24Kf core contains a variety of blocks some of which are always present, while others are optional.

The required blocks are as follows:

- Fetch Unit
- Execution Unit
- MIPS16e recode
- System Control Coprocessor (CP0)
- Memory Management Unit (MMU)
- Cache Controllers
- Bus Interface Unit (BIU)
- Power Management
- Instruction Cache
- Floating Point Unit

Optional blocks include:

- CorExtend™ User Defined Instruction (UDI) support
- Enhanced JTAG (EJTAG) breakpoints
- MIPS Trace (PDTrace) support
- Instruction/Data cache
- Instruction/Data scratchpad
- COP2 interface

## Pipeline Flow

The 24Kf core implements an 8-stage pipeline. Three extra fetch stages are conditionally added when executing MIPS16e instructions. This pipeline allows the processor to achieve a high frequency while maintaining reasonable area and power numbers.

The 24Kf core pipeline consists of the following stages:

- IF - Instruction Fetch First
- IS - Instruction Fetch Second
- IR - Instruction Recode (MIPS16e only)
- IK - Instruction Kill (MIPS16e only)
- IT - Instruction Fetch Third (MIPS16e only)
- RF - Register File access
- AG - Address Generation
- EX - Execute
- MS - Memory Second
- ER - Exception Resolution
- WB - WriteBack

The 24Kf 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 diagram of the 24Kf core pipeline.

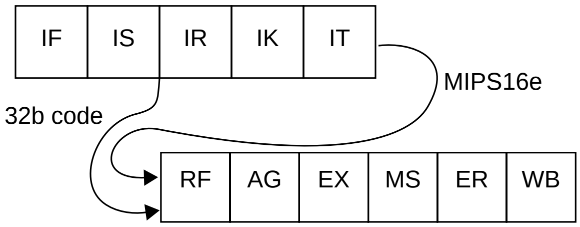


Figure 2   24Kf™ Core Pipeline

**IF Stage: Instruction Fetch First**

- I-cache tag/data arrays accessed
- Branch History Table accessed
- ITLB address translation performed
- Instruction watch and EJTAG break compares done

**IS - Instruction Fetch Second**

- Detect I-cache hit
- Way select
- MIPS32 Branch prediction

**IR - Instruction Recode**

- MIPS16e instruction recode

- MIPS16e branch prediction

**IK - Instruction Kill**

- MIPS16e instruction kill

**IT - Instruction Fetch Third**

- Instruction Buffer
- Branch target calculation

**RF - Register File Access**

- Register File access
- Instruction decoding/dispatch logic
- Bypass muxes

**AG - Address Generation**

- D-cache Address Generation
- Bypass muxes

**EX - Execute/Memory Access**

- Skewed ALU
- DTLB
- Start DCache access
- Branch Resolution
- Data watch and EJTAG break address compares

**MS - Memory Access Second**

- Complete DCache access
- DCache hit detection
- Way select mux
- Load align
- EJTAG break data value compare

**ER- Exception Resolution**

- Instruction completion
- Register file write setup
- Exception processing

**WB - Writeback**

- Register file writeback occurs on rising edge of this cycle

## 24Kf™ Core Logic Blocks

The 24Kf core consists of the following logic blocks, shown in Figure 1. These logic blocks are defined in the following subsections:

- Fetch Unit
- Execution Unit
- Floating Point Unit (FPU) / Coprocessor 1
- MIPS16e support
- System Control Coprocessor (CP0)
- Memory Management Unit (MMU)
- Cache Controller
- Bus Interface Unit (BIU)
- Power Management

### Fetch Unit

The 24Kf core fetch unit is responsible for fetching instructions and providing them to the rest of the pipeline, as well as handling control transfer instructions (branches, jumps, etc.). It calculates the address for each instruction fetch and contains an instruction buffer that decouples the fetching of instructions from their execution.

The fetch unit contains two structures for the dynamic prediction of control transfer instructions. A 512-entry Branch History Table (BHT) is used to predict the direction of branch instructions. It uses a bimodal algorithm with two bits of history information per entry. Also, a 4-entry Return Prediction Stack (RPS) is a simple structure to hold the return address from the most recent subroutine calls. The link address is pushed onto the stack whenever a JAL, JALR, or BGEZAL instruction is seen. Then that address is popped when a JR instruction occurs.

### Execution Unit

The 24Kf core execution unit implements a load/store architecture with single-cycle ALU operations (logical, shift, add, subtract) and an autonomous multiply/divide unit. The 24Kf core contains thirty-two 32-bit general-purpose registers used for integer operations and address calculation. Optionally, one or three additional register file shadow sets (each containing thirty-two registers) can be added to minimize context switching overhead during interrupt/exception processing. The register file consists of two read ports and one write port and is fully bypassed to minimize operation latency in the pipeline.

The execution unit includes:

- 32-bit adder used for calculating the data address
- Logic for verifying branch prediction
- Load aligner
- Bypass multiplexers used to avoid stalls when executing instructions streams where data producing instructions are followed closely by consumers of their results
- Leading Zero/One detect unit for implementing the CLZ and CLO instructions
- Arithmetic Logic Unit (ALU) for performing bitwise logical operations
- Shifter & Store Aligner

## Floating Point Unit (FPU) / Coprocessor 1

The 24Kf 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 FPU can be configured at build time to run at either the same or one-half the clock rate of the integer core. The FPU is not as deeply pipelined as the integer core so the maximum core frequency will only be attained with the FPU running at one-half the core frequency. The FPU is connected via an internal 64-bit coprocessor interface. Note that clock cycles related to floating point operations are listed in terms of FPU clocks, not integer core clocks.

The performance is optimized for single precision formats. Most instructions have one FPU cycle throughput and four FPU cycle latency. 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 core pipeline and does not stall when the integer pipeline stalls. This allows long-running FPU operations, such as divide 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 as “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.

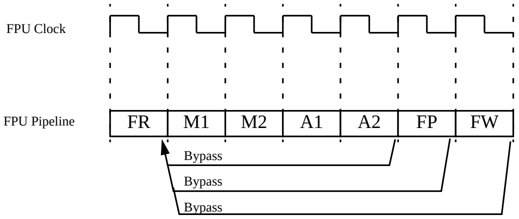


Figure 3 FPU Pipeline

### FPU Instruction Latencies and Repeat Rates

Table 1 contains the floating point instruction latencies and repeat rates for the 24Kf core. In this table ‘Latency’ refers to the number of FPU 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 per FPU cycle

Table 1 24Kf™ Core FPU Latency and Repeat Rate

| Opcode* | Latency (FPU cycles) | Repeat Rate (FPU cycles) |
|---|---|---|
| ABS.[S,D], NEG.[S,D], ADD.[S,D], SUB.[S,D], C.cond.[S,D], MUL.S | 4 | 1 |
| MADD.S, MSUB.S, NMADD.S, NMSUB.S, CABS.cond.[S,D] | 4 | 1 |
| CVT.D.S, CVT.PS.PW, CVT.[S,D].[W,L] | 4 | 1 |
| CVT.S.D, 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] | 4 | 1 |
| MOV.[S,D], MOVF.[S,D], MOVN.[S,D], MOVT.[S,D], MOVZ.[S,D] | 4 | 1 |
| MUL.D | 5 | 2 |
| MADD.D, MSUB.D, NMADD.D, NMSUB.D | 5 | 2 |
| RECIP.S | 13 | 10 |
| RECIP.D | 26 | 21 |
| RSQRT.S | 17 | 14 |
| RSQRT.D | 36 | 31 |
| DIV.S, SQRT.S | 17 | 14 |
| DIV.D, SQRT.D | 32 | 29 |
| MTC1, DMTC1, LWC1, LDC1, LDXC1, LUXC1, LWXC1 | 4 | 1 |
| MFC1, DMFC1, SWC1, SDC1, SDXC1, SUXC1, SWXC1 | 1 | 1 |
| * Format: S = Single, D = Double, W = Word, L = Longword | | |

**FPU Control Registers**

The FPU contains a number of control registers, 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. |

## MIPS16e™ Application Specific Extension

The 24Kf core includes support for the MIPS16e ASE. This ASE improves code density through the use of 16-bit encoding of many MIPS32 instructions plus some MIPS16e-specific instructions. PC relative loads allow quick access to constants. Save/Restore macro instructions provide for single instruction stack frame setup/teardown for efficient subroutine entry/exit.

## Multiply/Divide Unit (MDU)

The 24Kf core includes a multiply/divide unit (MDU) that contains a separate pipeline for integer multiply and divide operations. This pipeline operates in parallel with the integer unit pipeline and does not stall when the integer pipeline stalls. This allows any long-running MDU operations to be partially masked by system stalls and/or other integer unit instructions.

The MDU consists of a pipelined 32x32 multiplier, result/accumulation registers (HI and LO), a divide state machine, and the necessary multiplexers and control logic.

The MDU supports execution of one multiply or multiply accumulate operation every clock cycle.

Divide operations are implemented with a simple 1 bit per clock iterative algorithm. An early-in detection checks the sign extension of the dividend (*rs*) operand. If rs is 8 bits wide, 23 iterations are skipped. For a 16-bit-wide rs, 15 iterations are skipped, and for a 24-bit-wide rs, 7 iterations are skipped. Any attempt to issue a subsequent MDU instruction while a divide is still active causes a pipeline stall until the divide operation is completed.

Table 3 lists the latencies (number of cycles until a result is available) and repeat rates (peak issue rate of cycles until the operation can be reissued) for the 24Kf core multiply and divide instructions. The approximate latency and repeat rates are listed in terms of pipeline clocks. For a more detailed discussion of latencies and repeat rates, refer to Chapter 2 of the *MIPS32 24K Processor Core Family Software User's Manual*.

Table 3 24Kf™ Core Integer Multiply/Divide Unit Latencies and Repeat Rates

| Opcode | Operand Size (mul *rt*) (div *rs*) | Latency | Repeat Rate |
|---|---|---|---|
| MULT/MULTU,<br>MADD/MADDU,<br>MSUB/MSUBU | 32 bits | 5 | 1 |
| MUL | 32 bits | 5 | 1[1] |
| DIV/DIVU | 8 bits | 12/14 | 12/14 |
|  | 16 bits | 20/22 | 20/22 |
|  | 24 bits | 28/30 | 28/30 |
|  | 32 bits | 36/38 | 36/38 |

[1] If there is no data dependency, a MUL can be issued every cycle.

The MIPS architecture defines that the result 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.

In addition to the HI/LO targeted operations, the MIPS32 architecture also defines a multiply instruction, MUL, which places the least significant results in the primary register file instead of the HI/LO register pair.

Two other instructions, multiply-add (MADD) and multiply-subtract (MSUB), are used to perform the multiply-accumulate and multiply-subtract 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.

## 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 diagnostic capability, the operating modes (kernel, user, supervisor, and debug), and whether interrupts are enabled or disabled. Configuration information, such as cache size and associativity, presence of features like MIPS16e or floating point unit, is also available by accessing the CP0 registers, listed in Table 4.

Table 4 Coprocessor 0 Registers in Numerical Order

| Register Number | Register Name | Function |
|---|---|---|
| 0 | Index[3] | Index into the TLB array. |
| 1 | Random[3] | Randomly generated index into the TLB array. |
| 2 | EntryLo0[3] | Low-order portion of the TLB entry for even-numbered virtual pages. |
| 3 | EntryLo1[3] | Low-order portion of the TLB entry for odd-numbered virtual pages. |
| 4 | Context[1] | Pointer to page table entry in memory. |
| 5 | PageMask[3] | Control for variable page sizes in TLB entries. |
| 6 | Wired[3] | Controls the number of fixed ("wired") TLB entries. |
| 7 | HWREna | Enables access via the RDHWR instruction to selected hardware registers. |
| 8 | BadVAddr[1] | Reports the address for the most recent address-related exception. |
| 9 | Count[1] | Processor cycle count. |
| 10 | EntryHi[3] | High-order portion of the TLB entry. |
| 11 | Compare[1] | Timer interrupt control. |

Table 4 Coprocessor 0 Registers in Numerical Order

| Register Number | Register Name | Function |
|---|---|---|
| 12 | Status[1] | Processor status and control. |
| 12 | IntCtl[1] | Interrupt system status and control. |
| 12 | SRSCtl[1] | Shadow register set status and control. |
| 12 | SRSMap[1] | Provides mapping from vectored interrupt to a shadow set. |
| 13 | Cause[1] | Cause of last general exception. |
| 14 | EPC[1] | Program counter at last exception. |
| 15 | PRId | Processor identification and revision. |
| 15 | EBASE | Exception vector base register. |
| 16 | Config | Configuration register. |
| 16 | Config1 | Configuration register 1. |
| 16 | Config2 | Configuration register 2. |
| 16 | Config3 | Configuration register 3. |
| 16 | Config7 | Configuration register 7. |
| 17 | Reserved | Reserved in the 24Kf core. |
| 18 | WatchLo[1] | Low-order watchpoint address. |
| 19 | WatchHi[1] | High-order watchpoint address. |
| 20-22 | Reserved | Reserved in the 24Kf core. |
| 23 | Debug[2] | Debug control and exception status. |
| 23 | Trace Control[2] | PC/Data trace control register. |
| 23 | Trace Control2[2] | Additional PC/Data trace control. |
| 23 | User Trace Data[2] | User Trace control register. |
| 23 | TraceBPC[2] | Trace breakpoint control. |
| 24 | DEPC[2] | Program counter at last debug exception. |
| 25 | PerfCount | Performance counters and associated control. |
| 26 | ErrCtl | Used for software testing of cache arrays. |
| 27 | CacheErr | Cache parity error interface. |
| 28 | TagLo/ DataLo | Low-order portion of cache tag interface. |
| 29 | DataHi | Hi-order portion of cache tag interface. |
| 30 | ErrorEPC[1] | Program counter at last error. |
| 31 | DESAVE[2] | Debug handler scratchpad register. |

[1] Registers used in exception processing.

[2] Registers used during debug.

[3] Registers used in memory management.

Coprocessor 0 also contains the logic for identifying and managing exceptions. Exceptions can be caused by a variety of sources, including boundary cases in data, external events, or program errors. Table 5 shows the exception types in order of priority.

Table 5 24Kf™ Core Exception Types

| Exception | Description |
|---|---|
| Reset | Assertion of *SI_Reset* signal. |
| DSS | EJTAG Debug Single Step. |
| DINT | EJTAG Debug Interrupt. Caused by the assertion of the external *EJ_DINT* input, or by setting the EjtagBrk bit in the ECR register. |
| DDBLImpr/ DDBSImpr | Debug Data Break Load/Store Imprecise |
| NMI | Assertion of *SI_NMI* signal. |
| Interrupt | Assertion of unmasked hardware or software interrupt signal. |
| Deferred Watch | Deferred Watch (unmasked by K\|DM->!(K\|DM) transition). |
| DIB | EJTAG debug hardware instruction break matched. |
| WATCH | A reference to an address in one of the watch registers (fetch). |
| AdEL | Fetch address alignment error.<br>Fetch reference to protected address. |
| TLBL | Fetch TLB miss. |

Table 5 24Kf™ Core Exception Types (Continued)

| Exception | Description |
|---|---|
| TLBL | Fetch TLB hit to page with V=0. |
| I Cache Error | Instruction cache parity error |
| IBE | Instruction fetch bus error. |
| DBp | EJTAG Breakpoint (execution of SDBBP instruction). |
| Sys | Execution of SYSCALL instruction. |
| Bp | Execution of BREAK instruction. |
| CpU | Execution of a coprocessor instruction for a coprocessor that is not enabled. |
| CEU | Execution of a CorExtend instruction when CorExtend is not enabled. |
| RI | Execution of a Reserved Instruction. |
| FPE | Floating Point Exception |
| C2E | Coprocessor2 Exception |
| IS1 | Implementation specific Coprocessor2 exception |
| Ov | Execution of an arithmetic instruction that overflowed. |
| Tr | Execution of a trap (when trap condition is true). |
| Machine Check | TLB write that conflicts with an existing entry. |
| DDBL / DDBS | EJTAG Data Address Break (address only). |
| WATCH | A reference to an address in one of the watch registers (data). |
| AdEL | Load address alignment error.<br>Load reference to protected address. |
| AdES | Store address alignment error.<br>Store to protected address. |
| TLBL | Load TLB miss. |
| TLBL | Load TLB hit to page with V=0. |
| TLBS | Store TLB miss. |
| TLBS | Store TLB hit to page with V=0. |
| TLB Mod | Store to TLB page with D=0. |
| D Cache Error | Data cache parity error - imprecise |

Table 5 24Kf™ Core Exception Types (Continued)

| Exception | Description |
|---|---|
| DBE | Load or store bus error - imprecise |

### Interrupt Handling

The 24Kf core includes support for six hardware interrupt pins, two software interrupts, a timer interrupt, and a performance counter interrupt. These interrupts can be used in any of three interrupt modes, as defined by Release 2 of the MIPS32 Architecture:

- Interrupt compatibility mode, which acts identically to that in an implementation of Release 1 of the Architecture.
- Vectored Interrupt (VI) mode, which adds the ability to prioritize and vector interrupts to a handler dedicated to that interrupt, and to assign a GPR shadow set for use during interrupt processing. The presence of this mode is denoted by the VInt bit in the *Config3* register. This mode is architecturally optional; but it is always present on the 24Kf core, so the VInt bit will always read as a 1 for the 24Kf core.
- External Interrupt Controller (EIC) mode, which redefines the way in which interrupts are handled to provide full support for an external interrupt controller handling prioritization and vectoring of interrupts. This presence of this mode denoted by the VEIC bit in the *Config3* register. Again, this mode is architecturally optional. On the 24Kf core, the VEIC bit is set externally by the static input, *SI_EICPresent*, to allow system logic to indicate the presence of an external interrupt controller.

The reset state of the processor is to interrupt compatibility mode such that a processor supporting Release 2 of the Architecture, like the 24Kf core, is fully compatible with implementations of Release 1 of the Architecture.

VI or EIC interrupt modes can be combined with the optional shadow registers to specify which shadow set should be used upon entry to a particular vector. The shadow registers further improve interrupt latency by avoiding the need to save context when invoking an interrupt handler.

### GPR Shadow Registers

Release 2 of the MIPS32 Architecture optionally removes the need to save and restore GPRs on entry to high priority interrupts or exceptions, and to provide specified processor modes with the same capability. This is done by introducing multiple copies of the GPRs, called *shadow sets*, and allowing privileged software to associate a shadow set with entry to kernel mode via an interrupt vector or exception. The normal GPRs are logically considered shadow set zero.

The number of GPR shadow sets is a build-time option on the 24Kf core. Although Release 2 of the Architecture defines a maximum of 16 shadow sets, the core allows one (the normal GPRs), two, or four shadow sets. The highest number actually implemented is indicated by the $\text{SRSCtl}_{\text{HSS}}$ field. If this field is zero, only the normal GPRs are implemented.

Shadow sets are new copies of the GPRs that can be substituted for the normal GPRs on entry to kernel mode via an interrupt or exception. Once a shadow set is bound to a kernel mode entry condition, reference to GPRs work exactly as one would expect, but they are redirected to registers that are dedicated to that condition. Privileged software may need to reference all GPRs in the register file, even specific shadow registers that are not visible in the current mode. The RDPGPR and WRPGPR instructions are used for this purpose. The CSS field of the *SRSCtl* register provides the number of the current shadow register set, and the PSS field of the *SRSCtl* register provides the number of the previous shadow register set (that which was current before the last exception or interrupt occurred).

If the processor is operating in VI interrupt mode, binding of a vectored interrupt to a shadow set is done by writing to the *SRSMap* register. If the processor is operating in EIC interrupt mode, the binding of the interrupt to a specific shadow set is provided by the external interrupt controller, and is configured in an implementation-dependent way. Binding of an exception or non-vectored interrupt to a shadow set is done by writing to the ESS field of the *SRSCtl* register. When an exception or interrupt occurs, the value of $\text{SRSCtl}_{\text{CSS}}$ is copied to $\text{SRSCtl}_{\text{PSS}}$, and $\text{SRSCtl}_{\text{CSS}}$ is set to the value taken from the appropriate source. On an ERET, the value of $\text{SRSCtl}_{\text{PSS}}$ is copied back into $\text{SRSCtl}_{\text{CSS}}$ to restore the shadow set of the mode to which control returns.

## Modes of Operation

The 24Kf core supports four modes of operation: user mode, supervisor mode, kernel mode, and debug mode. User mode is most often used for application programs. Supervisor mode gives an intermediate privilege level with access to the ksseg address space. Supervisor mode is not supported with the fixed mapping MMU. Kernel mode is typically used for handling exceptions and operating system kernel functions, including CP0 management and I/O device accesses. An additional Debug mode is used during system bring-up and software development. Refer to "EJTAG Debug Support" on page 24 for more information on debug mode.

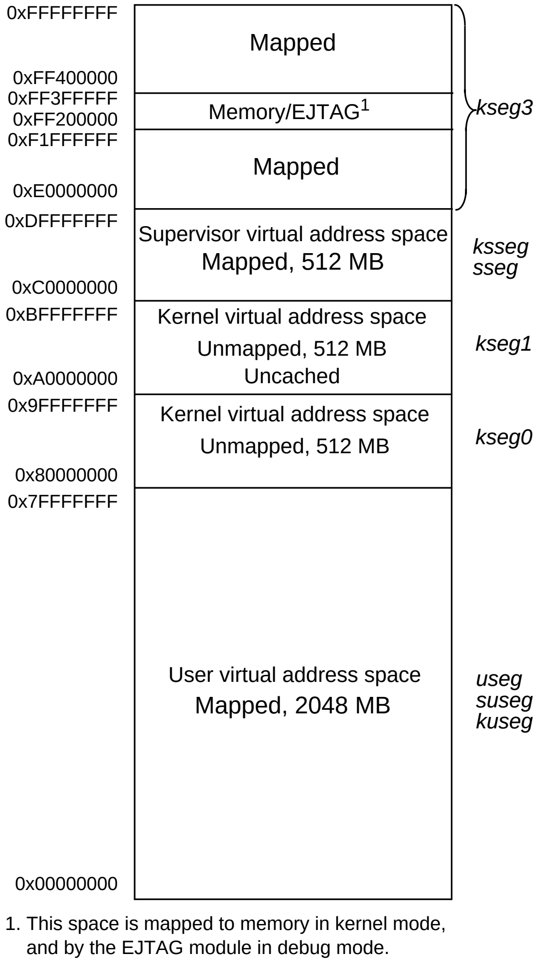


Figure 4 24Kf™ Core Virtual Address Map

## Memory Management Unit (MMU)

The 24Kf core contains a configurable Memory Management Unit (MMU) that is primarily responsible for converting virtual addresses to physical addresses and providing attribute information for different segments of memory.

Two types of MMUs are possible on the 24Kf core, selectable when the core is synthesized. Software can identify the type of MMU present by querying the MT field of the *Config* register.

1. Translation Lookaside Buffer (TLB) -style MMU. The basic TLB functionality is specified by the MIPS32 Privileged Resource Architecture (PRA). A TLB provides mapping and protection capability with per-page granularity. The 24Kf implementation allows a wide range of page sizes to be present simultaneously.
2. Fixed Mapping Translation (FMT) -style MMU. The FMT is much simpler and smaller than the TLB-style MMU, and is a good choice when the full protection and flexibility of the TLB is not needed.

## Translation Lookaside Buffer (TLB)

The TLB consists of three address translation buffers:

- 16/32/64 dual-entry fully associative Joint TLB (JTLB)
- 4-entry fully associative Instruction Micro TLB (ITLB)
- 8-entry fully associative Data Micro TLB (DTLB)

When an instruction or data address is calculated, the virtual address is compared to the contents of the appropriate micro TLB (uTLB). If the address is not found in the ITLB or DTLB, the JTLB is accessed. If the entry is found in the JTLB, that entry is then written into the uTLB. If the address is not found in the JTLB, a TLB exception is taken.

Figure 5 shows how the ITLB, DTLB, and JTLB are implemented in the 24Kf core.

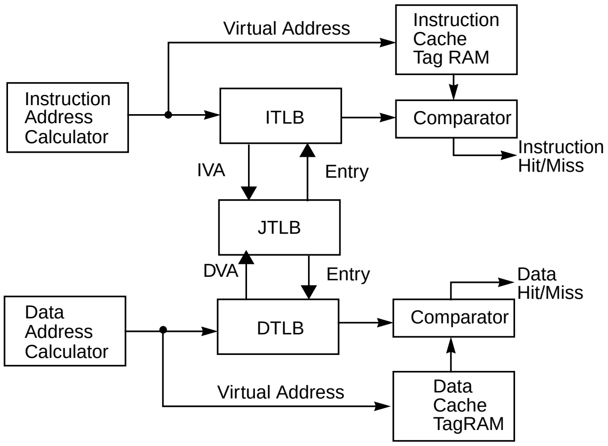


Figure 5 Address Translation During a Cache Access

### Joint TLB (JTLB)

The 24Kf core implements a fully associative JTLB containing 16, 32, or 64-dual-entries mapping up to 128 virtual pages to their corresponding physical addresses.

The purpose of the TLB is to translate virtual addresses and their corresponding ASIDs 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 as pairs of even and odd entries containing pages that range in size from 4 KB to 256 MB, in factors of four, into the 4 GB physical address space. The JTLB is organized in page pairs to minimize the overall size. Each *tag* entry corresponds to two 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 look-up.

### Instruction TLB (ITLB)

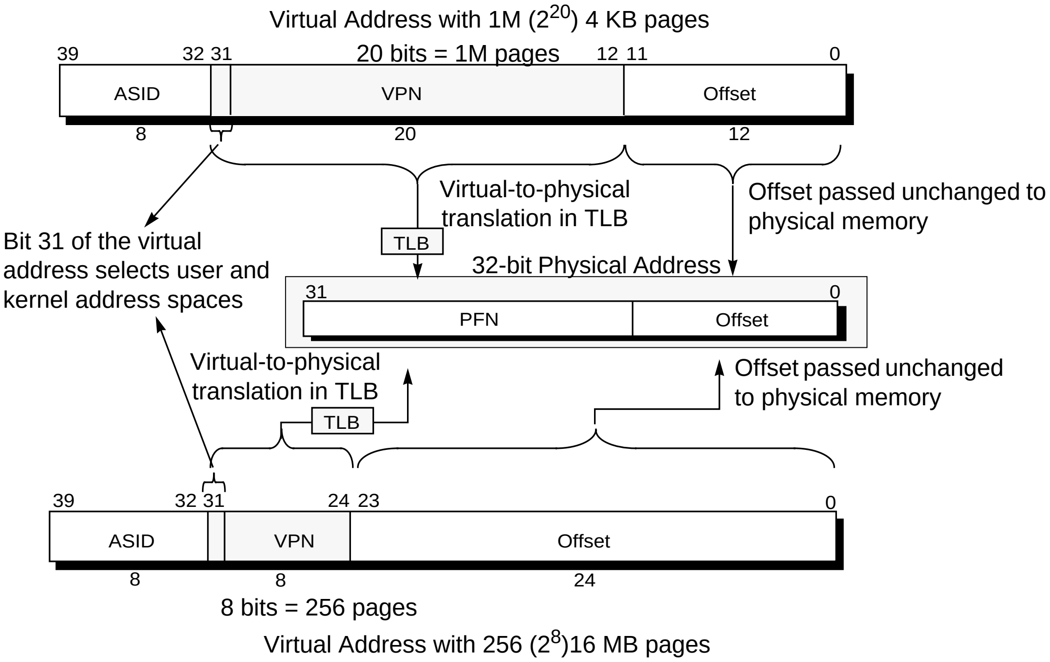
The ITLB is a small 4-entry, fully associative TLB dedicated to performing translations for the instruction stream. The ITLB only maps 4 KB or 1 MB pages/ subpages. For 4 KB or 1 MB pages, the entire page is mapped in the ITLB. If the main TLB page size is between 4 KB and 1 MB, only the current 4 KB subpage is mapped. Similarly, for page sizes larger than 1 MB, the current 1 MB subpage is mapped.

The ITLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing structure for the ITLB. If a fetch address cannot be translated by the ITLB, the JTLB is used to attempt to translate it in the following clock cycle, or when available. If successful, the translation information is copied into the ITLB for future use. There is a minimum two cycle ITLB miss penalty.

### Data TLB (DTLB)

The DTLB is a small 8-entry, fully associative TLB dedicated to performing translations for loads and stores. Similar to the ITLB, the DTLB only maps either 4 KB or 1 MB pages/subpages.

The DTLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing structure for the DTLB. If a load/store address cannot be translated by the DTLB, a lookup is done in the JTLB. If the JTLB translation is successful, the translation information is copied into the DTLB for future use. The DTLB miss penalty is also two cycles.



Virtual Address with 1M (2^20) 4 KB pages
20 bits = 1M pages
39
32 31
12 11
0
ASID
VPN
Offset
8
20
12
Virtual-to-physical translation in TLB
Offset passed unchanged to physical memory
Bit 31 of the virtual address selects user and kernel address spaces
TLB
32-bit Physical Address
31
0
PFN
Offset
Virtual-to-physical translation in TLB
TLB
Offset passed unchanged to physical memory
39
32 31
24 23
0
ASID
VPN
Offset
8
8
24
8 bits = 256 pages
Virtual Address with 256 (2^8)16 MB pages

Figure 6 32-bit Virtual Address Translation

### 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; there is a match 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 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 *miss* 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 for two different page sizes.

The top portion of Figure 6 shows a virtual address for a 4 KB page size. The width of the *Offset* in Figure 6 is defined by the page size. The remaining 20 bits of the address represent the virtual page number (*VPN*), and index the 1M-entry page table.

The bottom portion of Figure 6 shows the virtual address for a 16 MB page size. The remaining 8 bits of the address represent the VPN, and index the 256-entry page table.

In Figure 6, the virtual address is extended with an 8-bit address space identifier (ASID), which reduces the frequency of TLB flushes during context switches. This 8-bit ASID contains the number assigned to that process and is stored in the CP0 *EntryHi* register.

### Hits, Misses, and Multiple Matches

Each JTLB 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 joint TLB (JTLB). The granularity of JTLB mappings is defined in terms of TLB *pages*. The 24Kf core’s JTLB supports pages of different sizes ranging from 4 KB to 256 MB in factors of 4.

Table 6 shows the address bits used for even/odd bank selection depending on page size and the relationship between the legal values in the mask register and the selected page size.

Table 6 Mask and Page Size Values

| Pagemask[28:13] | Page Size | Even/Odd Bank Select Bit |
|---|---|---|
| 0000000000000000 | 4KB | VAddr[12] |
| 0000000000000011 | 16KB | VAddr[14] |
| 0000000000001111 | 64KB | VAddr[16] |
| 0000000000111111 | 256KB | VAddr[18] |
| 0000000011111111 | 1MB | VAddr[20] |
| 0000001111111111 | 4MB | VAddr[22] |
| 0000111111111111 | 16MB | VAddr[24] |
| 0011111111111111 | 64MB | VAddr[26] |
| 1111111111111111 | 256MB | VAddr[28] |

If no match occurs (TLB miss), an exception is taken and software refills the TLB from the page table resident in memory. Software can write over a selected TLB entry or use a hardware mechanism to write into a random entry.

The 24Kf 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 24Kf core takes a machine check exception, sets the TS bit in the CP0 *Status* register, and aborts the write operation.

Compared with previous cores from MIPS Technologies, the 24Kf core uses a more relaxed check for multiple matches in order to avoid machine check exceptions while flushing or initializing the TLB. On a write, all matching entries are disabled to prevent them from matching on future compares. A machine check is only signaled if the entry being written has its valid bit set, the matching entry in the TLB has its valid bit set, and the matching entry is not the entry being written.

### TLB Tag and Data Formats

Figure 7 shows the format of a TLB *tag* entry. The entry is divided into the follow fields:

- Global process indicator
- Address space identifier
- Virtual page number
- Compressed page mask

Setting the global process indicator (G bit) indicates that the entry is global to all processes and/or threads in the system. In this case, the 8-bit address space identifier (ASID) value is ignored since the entry is not relative to a specific thread or process.

The ASID field can help to reduce the frequency of TLB flushes on a context switches. The existence of the ASID allows multiple processes to exist in both the TLB and instruction caches. The current ASID value is stored in the *EntryHi* register and is compared to the ASID value of each entry. Figure 7 and Table 7 show the TLB tag entry format.

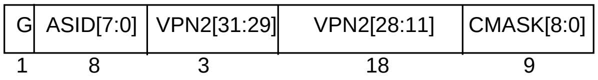


Figure 7 TLB Tag Entry Format

Table 7 TLB Tag Entry Fields

| Field Name | Description |
|---|---|
| G | Global Bit. When set, indicates that this entry is global to all processes and/or threads and thus disables inclusion of the ASID in the comparison. |
| ASID[7:0] | Address Space Identifier. Identifies with which process or thread this TLB entry is associated. |
| VPN2[31:29],<br>VPN2[28:13] | Virtual Page Number divided by 2. This field contains the upper bits of the virtual page number. Because it represents a pair of TLB pages, it is divided by 2. Bits 31:29 are always included in the TLB lookup comparison. Bits 28:13 are included depending on the page size. |
| CMASK[8:0] | Compressed page mask value. This field is a compressed version of the page mask. It defines the page size by masking the appropriate VPN2 bits from being involved comparison. It is also used to determine which address bit is used to make the even-odd page determination. |

Figure 8 and Table 8 show the TLB data array entry format.

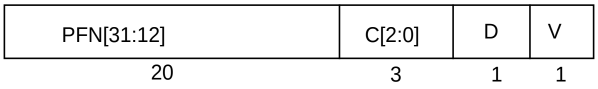


Figure 8 TLB Data Array Entry Format

Table 8 TLB Data Array Entry Fields

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>PFN[31:12]</td><td>Physical Frame Number. Defines the upper bits of the physical address.<br>For page sizes larger than the 4KB, only a subset of these bits is actually used.</td></tr>
<tr><td>C[2:0]</td><td>Cacheability. Contains an encoded value of the cacheability attributes and determines whether the page should be placed in the cache or not. The field is encoded as follows:
<table>
<tr><th>CS[2:0]</th><th>Coherency Attribute</th></tr>
<tr><td>0</td><td>Cacheable, noncoherent, write through, no write allocate.</td></tr>
<tr><td>1</td><td>Reserved</td></tr>
<tr><td>2</td><td>Uncached</td></tr>
<tr><td>3</td><td>Cacheable, noncoherent, write back, write allocate</td></tr>
<tr><td>4-6</td><td>Reserved</td></tr>
<tr><td>7</td><td>Uncached Accelerated</td></tr>
</table></td></tr>
<tr><td>D</td><td>“Dirty” or write-enable bit. Indicates that the page has been written and/or is writable. If this bit is set, stores to the page are permitted. If the bit is cleared, stores to the page cause a TLB Modified exception.</td></tr>
<tr><td>V</td><td>Valid bit. Indicates that the TLB entry, and thus the virtual page mapping, are valid. If this bit is set, accesses to the page are permitted. If the bit is cleared, accesses to the page cause a TLB Invalid exception.</td></tr>
</table>

### Page Sizes and Replacement Algorithm

To assist in controlling both the amount of mapped space and the replacement characteristics of various memory regions, the 24Kf core provides two mechanisms. First, the page size can be configured, on a per-entry basis, to map a page size from 4 KB to 256 MB (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 maps. For example, a typical frame buffer might 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 24Kf core provides a random replacement algorithm. However, the processor also provides a mechanism where a programmable number of mappings can be locked into the TLB via the CP0 *Wired* register, thus avoiding random replacement.

## Fixed Mapping Translation (FMT)

The 24Kf core optionally provides a Fixed Mapping Translation mechanism that is smaller and simpler than the full Translation Lookaside Buffer (TLB). Like a TLB, the FMT performs virtual-to-physical address translation and provides attributes for the different segments. Those segments that are unmapped in a TLB implementation (kseg0 and kseg1) are handled identically by the FMT.

Figure 9 shows how the FMT is implemented in the 24Kf core.

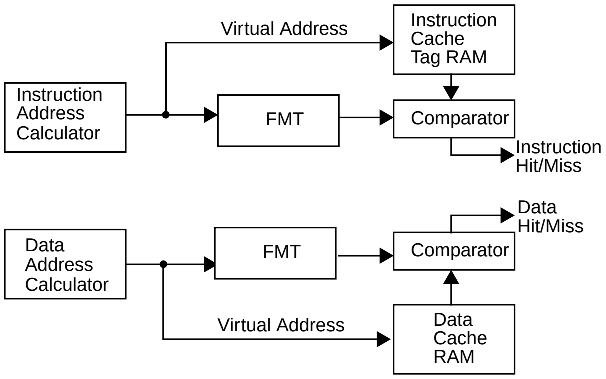


Figure 9 Address Translation During Access

In general, the FMT also determines the cacheability of each segment. These attributes are controlled via bits in the *Config* register. Table 9 shows the encoding for the K23 (bits 30:28), KU (bits 27:25), and K0 (bits 2:0) fields of the

Config register. Table 10 shows how the cacheability of the virtual address segments is controlled by these fields.

Table 9 Cache Coherency Attributes

| Config Register Fields K23, KU, and K0 | Cache Coherency Attribute |
|---|---|
| 0 | Cacheable, noncoherent, write-through, no write-allocate |
| 1 | Reserved |
| 2 | Uncached |
| 3 | Cacheable, noncoherent, write-back, write-allocate |
| 4-6 | Reserved |
| 7 | Uncached Accelerated |

In a 24Kf core with FMT, no translation exceptions can be taken, although address errors are still possible.

Table 10 Virtual Address Segments

| Segment | Virtual Address Range | Cacheability |
|---|---|---|
| useg<br>kuseg | 0x0000_0000-0x7FFF_FFFF | Controlled by the KU field (bits 27:25) of the Config register. See Table 9 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 9 for mapping. |
| kseg1 | 0xA000_0000-0xBFFF_FFFF | Always uncacheable. |
| kseg2 | 0xC000_0000-0xDFFF_FFFF | Controlled by the K23 field (bits 30:28) of the Config register. See Table 9 for mapping. |
| kseg3 | 0xE000_0000-0xFFFF_FFFF | Controlled by the K23 field (bits 30:28) of the Config register. See Table 9 for mapping. |

The FMT performs a simple translation to map from virtual addresses to physical addresses. This mapping is shown in Figure 10.

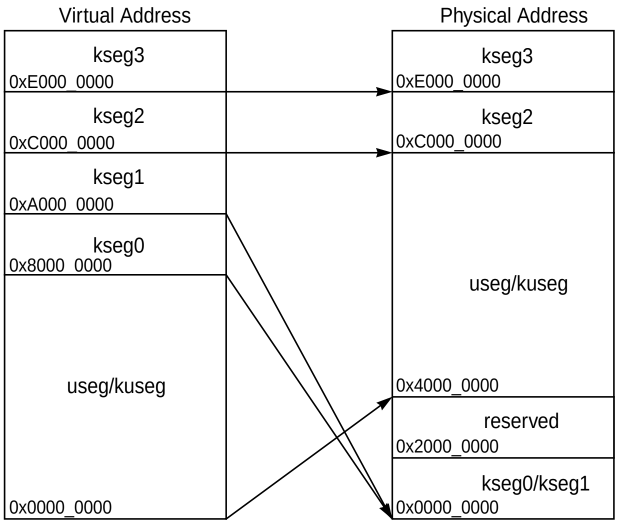


Figure 10 FMT Memory Map (ERL=0) in the 24Kf™ Core

When ERL=1, useg and kuseg become unmapped (virtual address is identical to the physical address) and uncached. This behavior is the same as if there was a TLB. This mapping is shown in Figure 11.

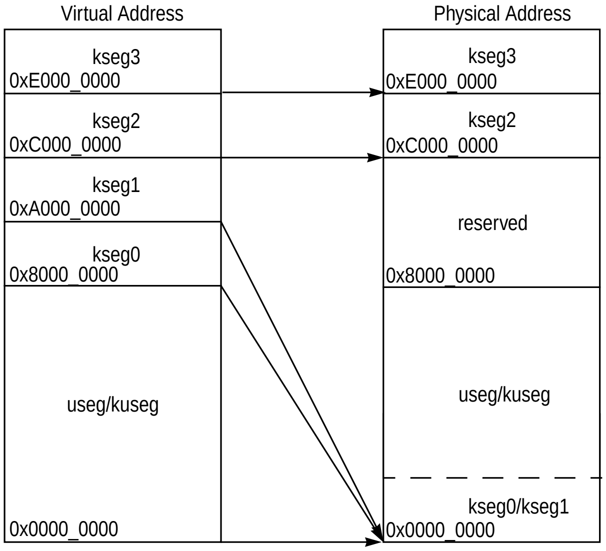


Figure 11 FMT Memory Map (ERL=1) in the 24Kf™ Core

## Instruction Cache

The instruction cache is an on-chip memory block of 16/32/64 KB, with 4-way associativity. Because 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. A tag entry holds 20 bits of physical address, a valid bit, a lock bit, and an optional parity bit per way. The instruction data entry holds two instructions (64 bits) per way, as well as 6 bits of pre-decode information to speed the decode of branch and jump instructions, and 9 optional parity bits (one per data byte plus one more for the pre-decode information). The LRU replacement bits (6b) are stored in a separate array.

The instruction cache block also contains and manages the instruction line fill buffer. Besides accumulating data to be written to the cache, instruction fetches that reference data in the line fill buffer are serviced either by a bypass of that data, or data coming from the external interface. The instruction cache control logic controls the bypass function.

The 24Kf 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.

## Data Cache

The data cache is an on-chip memory block of 0/16/32/64 KB, with 4-way associativity. Since the data cache is virtually indexed, the virtual-to-physical address translation occurs in parallel with the cache access. A tag entry holds 20 bits of physical address, a valid bit, a lock bit, and an optional parity bit per way. The data entry holds 64 bits of data per way, with optional parity per byte. There is an additional array holding dirty bits and LRU replacement algorithm bits (6b LRU, 4b dirty, and optionally 4b dirty parity).

Using 4KB pages in the TLB and 32 or 64KB cache sizes it is possible to get virtual aliasing. A single physical address can exist in multiple cache locations if it was accessed via different virtual addresses. For a 32KB data cache, there is an implementation option to eliminate virtual aliasing. If this option is not selected, or a 64KB cache is implemented, software must take care of any aliasing issues by using a page coloring scheme or some other mechanism.

In addition to instruction-cache locking, the 24Kf core also supports a data-cache locking mechanism identical to the instruction cache. Critical data segments are locked into the cache on a "per-line" basis. The locked contents can be updated on a store hit, but will not be selected for replacement on a cache 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.

## Cache Memory Configuration

The 24Kf core incorporates on-chip instruction and data caches that are usually implemented from readily available single-port synchronous SRAMs and accessed in two cycles: one cycle for the actual SRAM read and another cycle for the tag comparison, hit determination, and way selection. The instruction and data caches each have their own 64-bit data paths and can be accessed simultaneously. Table 11 lists the 24Kf core instruction and data cache attributes.

Table 11 24Kf™ Core Instruction and Data Cache Attributes

| Parameter | Instruction | Data |
|---|---|---|
| Size | 16, 32, or 64 KB* | 0, 16, 32, or 64 KB |
| Organization | 4 way set associative | 4 way set associative |
| Line Size | 32 Bytes* | 32 Bytes |
| Read Unit | 64 bits* | 64 bits |
| Write Policies | N/A | write-through without write allocate,<br>write-back with write allocate |
| Miss restart after transfer of | miss word | miss word |
| Cache Locking | per line | per line |

*Logical size of instruction cache. Cache physically contains some extra bits used for precoding the instruction type.

## Cache Protocols

The 24Kf 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 look-up misses, only main memory is written.
- **Write-back, write allocate:** Stores that miss in the cache will cause a cache refill. Store data, however, is only written to the cache. Caches lines that are written by stores will be marked as dirty. If a dirty line is selected for replacement, the cache line will be written back to main memory.
- **Uncached Accelerated:** Like uncached, data is never loaded into the cache. Store data can be gathered in a write buffer before being sent out on the bus as a bursted write. This is more efficient than sending out individual writes as occurs in regular uncached mode.

## Bus Interface (BIU)

The Bus Interface Unit (BIU) controls the external interface signals. The primary interface implements the Open Core Protocol (OCP). Additionally, the BIU includes a write buffer.

### OCP Interface

Table 12 shows the OCP Performance Report for the 24Kf core. This table lists characteristics about the core and the specific OCP functionality that is supported.

Table 12 OCP Performance Report

| *Core name* | 24Kf |
|---|---|
| Core Identity<br>Vendor Code<br>Core Code<br>Revision Code | TBD<br>TBD<br>0x93, visible in ProcessorID field of CP0 *PrID* register<br>Visible in Revision field of *PrID* register |
| Process dependent | No |
| Frequency range for this core | Synthesizable, so varies based on process, libraries, and implementation |
| Area | Synthesizable, so varies based on process, libraries, and implementation |
| Power Estimate | Synthesizable, so varies based on process, libraries, and implementation |
| Special reset requirements | No |
| Number of Interfaces | 1 OCP master, 2 OCP slave (DMA access for SPRAMs) |
| Interface Information:<br>• Name<br>• Type | <br>OCPMasterInterface<br>Master |
| **Master OCP Interface** | |
| a. Operations issued | RD, WR |
| b. Issue rate (per OCP cycle) | One per cycle, for all of the types listed above except for a non-standard RD (SYNC) which depends on ack latency. |
| Maximum number of operations outstanding | 6 read operations. All writes are posted, so the OCP fabric determines the maximum number of outstanding writes. |
| Burst support and effect on issue rates | Fixed burst length of four 64b beats with single request per burst. Burst sequences of WRAP or XOR supported. |

Table 12 OCP Performance Report

| | |
|---|---|
| High level flow control | None |
| Number of tags supported and use of those tags | Total of 8 tags: 6 tags for outstanding RD’s, 1 tag for WR & 1 tag for SYNC |
| Connection ID and use of connection information | None |
| Use of sideband signals | None |
| Implementation restrictions | 1. MReqInfo handled in a user defined way. 3 bits used to send cacheable attribute information or encode type of L2 CACHE instruction, 1 bit used to signify SYNC.<br>2. MAddrSpace is used (2 bits) to indicate L2/L3 access.<br>4. Core clock is synchronous but a multiple of the OCP clock. The ratios supported are 1:1, 1:1.5, 1:2, 1:2.5, 1:3, 1:3.5, 1:4 and 1:5. A helper pulse is required by the core to transfer data from/to the OCP interface without any hazards. |
| Interface Information: | |
| • Name | OCPSlaveInterface |
| • Type | Slave |
| **Slave OCP Interfaces (DMA interface to scratchpad)** | |
| a. Operations accepted | RD, WR |
| b. Issue rate (per OCP cycle) | One per cycle, for all of the types listed above except for a non-standard RD (SYNC) which is not supported. |
| Maximum number of operations outstanding | 2 outstanding operations which includes both RD & WR. |
| Burst support | Burst access is not supported |
| High level flow control | Back pressure from slave on data and command accept. Slave assumes no back pressure from the master. |
| Number of tags supported and use of those tags | Total of 8 tags. Any tag number can be used for read and write operation. |
| Connection ID and use of connection information | None |
| Use of sideband signals | None |
| Implementation restrictions | The slave interface operates at the same clock ratio as that of the master OCP interface. |

## Write Buffer

The BIU contains a merging write buffer. The purpose of this buffer is to store and combine write transactions before issuing them to the external interface. The write buffer is organized as four 32-byte buffers. Each buffer contains data from a single 32-byte aligned block of memory.

### Write Through

When using the write-through cache policy, the write buffer significantly reduces the number of write transactions on the external interface and reduces the amount of stalling in the core due to issuance of multiple writes in a short period of time.

### Write Back

The write buffer also holds eviction data for write-back lines. The load-store unit opportunistically pulls dirty data from the cache and sends it to the BIU. It is gathered in the write buffer and sent out as a bursted write.

**Uncached Accelerated**

For uncached accelerated references, the write buffer can gather multiple writes together and then perform a bursted write to increase the efficiency of the bus. Uncached accelerated gathering is supported for word and double word stores only.

Gathering of uncached accelerated stores will start on cache-line aligned addresses, i.e. 32 byte aligned addresses. Uncached accelerated word or double word stores that do not to meet the conditions required to start gathering will be treated like regular uncached stores.

Once an uncached accelerated store meets the requirements needed to start gathering, a gather buffer is reserved for this store. All subsequent uncached accelerated word or double word stores to the same 32B region will write sequentially into this buffer, independent of the word address associated with these latter stores. The uncached accelerated buffer is tagged with the address of the first store.

An uncached accelerated buffer is written to memory (flushed) if:

1. The last word in the 32-byte entry being gathered is written. (Implicit flush).
2. A PREF Nudge which matches the address associated with the gather buffer (Explicit flush).
3. A SYNC instruction is executed. (Explicit flush).
4. Bits <31:5> of the address of a Load instruction match the address associated with the gather buffer. (Implicit flush)
5. Bits <31:5> of the address of an uncached accelerated store do not match the address associated with the gather buffer. Uncached accelerated store to a different 32B region (Implicit flush)
6. An exception occurs. (Implicit flush)

When an uncached accelerated buffer is flushed, the address sent out on the system interface is the address associated with the gather buffer.

Caveats:

- Any uncached loads or stores to unrelated addresses that occur between uncached accelerated stores that are part of a gather sequence will go out of order. They will not enforce ordering.
- One constraint imposed on the gathering is that doubleword stores are only allowed to write to double word aligned locations in the buffer. For example if uncached accelerated gathering starts with a Store Word (SW/SWC1), it may not be followed by a Store Double (SDC1)
- Uncached accelerated stores of the following types are not intended to be used by software and may generate unpredictable results:
  - Byte, Half, or unaligned Stores
  - Store conditionals
- In order for software to be able to run functionally correct on implementations without uncached accelerated stores, software should always generate accesses starting on a cache-line aligned address, proceed to generate correctly incremented sequential addresses and observe the restrictions for uncached accelerated stores.

### Burst Order

The core is capable of generating burst transactions on the OCP interface. A burst transaction is used to transfer multiple related data items. Burst transactions on the 24Kf core always consist of a single request, followed by four beats of data transfer.

Burst read transactions initiated by the core always contain four 64b data transfers. In addition, the data requested is always a 32-byte-aligned block. Burst reads are always initiated for cacheable instruction or data reads which have missed in the primary instruction or data cache.

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 always starts with the critical word requested by the execution unit and proceeds in either an ascending or descending order wrapping at the end of an aligned block.

The burst order sequence may be sequential or sub-block. These are equivalent to WRAP and XOR as defined by the OCP protocol. The selection is determined by the static input pin, *SI_SBlock*.

Table 13 and Table 14 show the implied sequence of address bits 3 and 4 for the two possible burst orders. Since there is only a single request command for a burst sequence, note that only the starting address is actually transmitted by the core.

Table 13 Sequential Burst Order

| Starting Address OC_MAddr[4:3] | Address Progression of OC_MAddr[4:3] |
|---|---|
| 00 | 00, 01, 10, 11 |
| 01 | 01, 10, 11, 00 |
| 10 | 10, 11, 00, 01 |
| 11 | 11, 00, 01, 10 |

Table 14 Sub-block Burst Order

| Starting Address OC_MAddr[4:3] | Address Progression of OC_MAddr[4:3] |
|---|---|
| 00 | 00, 01, 10, 11 |
| 01 | 01, 00, 11, 10 |
| 10 | 10, 11, 00, 01 |
| 11 | 11, 10, 01, 00 |

Burst write transactions can also occur when a full 32-byte block is written to memory. This may occur in the case of a cache line eviction, or when a full line has been gathered in the write buffer. For writes, the burst sequence always starts with an initial address of 00 on *OC_MAddr[4:3]*, so the write burst sequence is actually the same for sequential or sub-block orders.

### SimpleBE Mode

To aid in attaching the 24Kf core to structures which cannot easily handle arbitrary byte enable patterns, there is a mode that generates only "simple" byte enables. Only byte enables representing naturally aligned byte, halfword, word, and doubleword transactions will be generated. Legal byte enable patterns are shown in Table 15.

Table 15 Valid SimpleBE Byte Enable Patterns

| OC_MByteEn[7:0] or OC_MDataByteEn[7:0] |
|---|
| 0000_0001 |
| 0000_0010 |
| 0000_0100 |
| 0000_1000 |
| 0001_0000 |
| 0010_0000 |
| 0100_0000 |
| 1000_0000 |
| 0000_0011 |
| 0000_1100 |
| 0011_0000 |
| 1100_0000 |
| 0000_1111 |
| 1111_0000 |
| 1111_1111 |

The only case where a read can generate "non-simple" byte enables is on an uncached tri-byte load (LWL/LWR). In SimpleBE mode, such reads will be converted into a word read on the external interface.

Writes with non-simple byte enable patterns can arise when a sequence of stores is processed by the merging write buffer, or from uncached tri-byte stores (SWL/SWR). In SimpleBE mode, these stores will be broken into multiple write transactions.

## Clocking

The core has 3 primary clock domains:

- Core domain - This is the main core clock domain, controlled by the *SI_ClkIn* clock input.
- OCP domain - This domain controls the OCP bus interface logic. This domain is synchronous to *SI_ClkIn*, but can be run at lower frequencies. Core to bus ratios of 1:1, 3:2, 2:1, 5:2, 3:1, 7:2, 4:1 and 5:1 are supported. The core does not contain an explicit OCP input clock; all flops are actually controlled by *SI_ClkIn*. To enable the core to determine the frequency and phase relationship between the core and OCP domains, a "helper" pulse, *SI_OCPSync*, is required in the *SI_ClkIn* domain. *SI_OCPSync* is used internally to control when to drive OCP outputs and when to sample OCP inputs. Figure 12 illustrates the required waveform for *SI_OCPSync* at the various clock ratios. All OCP outputs are registered. All OCP inputs except *OC_SCmdAccept* and *OC_SDataAccept* are also registered.

- TAP domain - This is a low speed clock domain for the EJTAG TAP controller, controlled by the *EJ_TCK* pin. It is asynchronous to *SI_ClkIn*.

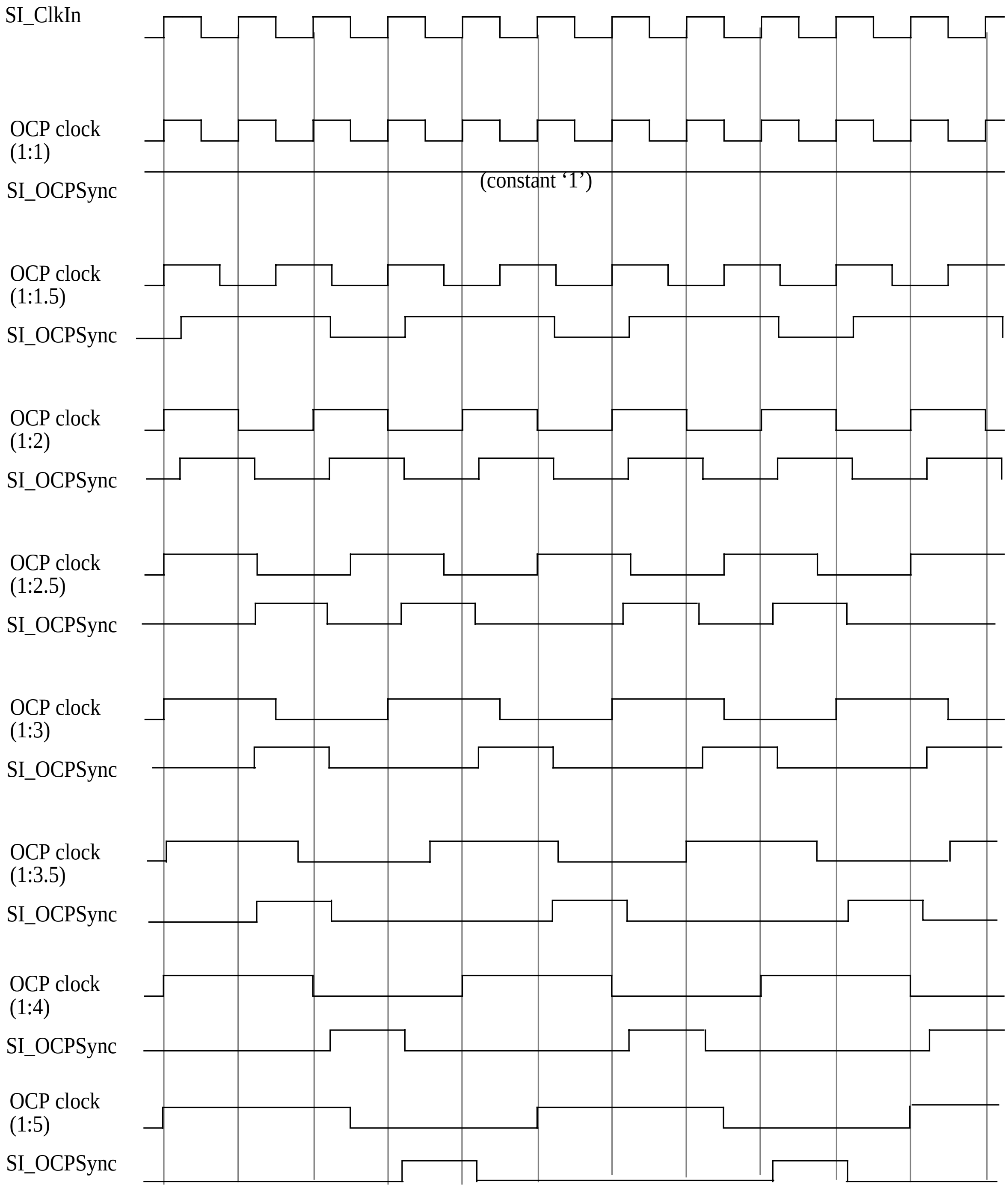


Figure 12 Required *SI_OCPSync* waveforms

## Hardware Reset

Unlike previous MIPS cores, a 24Kf core only has a single reset input. Historically, cold reset was used to reset a PLL. In synthesizable cores without a PLL, the two inputs were ORed together internally and then treated identically (except for a *Status* bit indicating which reset was seen). The 24Kf interface has removed the second reset type and only includes the *SI_Reset* pin.

The *SI_Reset* input is used to initialize critical hardware state. It can be asserted either synchronously or asynchronously to the core clock, *SI_ClkIn*, and will trigger a Reset exception. The reset signal is active high, and must be asserted for a minimum of 5 *SI_ClkIn* cycles. The falling edge triggers the Reset exception. The reset signal must be asserted at power-on or whenever hardware initialization of the core is desired.

In debug mode, EJTAG can request that a 'soft' reset be masked. This request is signalled via the EJ_SRstE pin. When this pin is deasserted, the system can choose to block some sources of soft reset. Hard resets, such as power-on reset or a reset switch should not be blocked by this signal.

## Power Management

The 24Kf core offers 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 slowing or halting the clocks, which reduces system power consumption during idle periods.

The 24Kf 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. The external agent then decides whether to place the device in a low power mode, such as reducing the system clock frequency.

Three additional bits, $\text{Status}_{\text{EXL}}$, $\text{Status}_{\text{ERL}}$, and $\text{Debug}_{\text{DM}}$ support the power management function by allowing the user to change the power state if an exception or error occurs while the 24Kf core is in a low power state. Depending on what type of exception is taken, one of these three bits will be asserted and reflected on the *SI_EXL*, *SI_ERL*, or *EJ_DebugM* outputs. The external agent can look at these signals and determine whether to leave the low power state to service the exception.

The following 4 power-down signals are part of the system interface and change state as the corresponding bits in the CP0 registers 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.
- The *EJ_DebugM* signal represents the state of the DM bit (30) in the CP0 *Debug* register.

### Instruction-Controlled Power Management

The second mechanism for invoking power-down mode is through execution of the WAIT instruction. When the WAIT instruction is executed, the internal clock is suspended; however, the internal timer and some of the input pins (*SI_Int[5:0], SI_NMI, and SI_Reset*) continue to run. Once the CPU is in instruction-controlled power management mode, any interrupt, NMI, or reset condition causes the CPU to exit this mode and resume normal operation.

The 24Kf core asserts the *SI_Sleep* signal, which is part of the system interface, whenever the WAIT instruction is executed. The assertion of *SI_Sleep* indicates that the clock has stopped and the 24Kf core is waiting for an interrupt.

### Local clock gating

A majority of the power consumed by the 24Kf core is often in the clock tree and clocking registers. The core has support for extensive use of local gated-clocks. Power-conscious implementors can use these gated clocks to significantly reduce power consumption within the core.

## CorExtend™ User Defined Instruction Extensions

The optional CorExtend User Defined Instruction (UDI) block enables the implementation of a small number of application-specific instructions that are tightly coupled to the core's integer execution unit.

The interface to the CorExtend block is similar to the Multiply-Divide Unit, allowing non-blocking, pipelined multi-cycle operations. A portion of the hooks into the MDU control logic and also allows the HI/LO accumulation registers to be used by the CorExtend block.

CorExtend instructions may operate on a general-purpose register, immediate data specified by the instruction word, or local state stored within the UDI block. The destination may be a general-purpose register, HI/LO, or local UDI state. The operation may complete in one cycle or multiple cycles, if desired.

### Coprocessor 2 interface

The 24Kf core can be configured to have an interface for an on-chip coprocessor. The interface allows the coprocessor to be tightly coupled to the processor core, allowing high performance solutions, like integrating a graphics accelerator or custom DSP.

The coprocessor interface is extensible and standardized on MIPS cores, allowing design reuse. The 24Kf core supports a subset of the full coprocessor interface standard: single issue, 64 bit in-order data transfers.

The coprocessor interface is designed to ease integration with customer IP. The interface allows high-performance communication between the core and coprocessor. There are no late or critical timing signals on the interface.

### Data Scratchpad RAM (DSPRAM)

The 24Kf core can be configured to include an optional Data scratchpad RAM independent of the data cache configuration. A separate OCP slave interface allows a DMA master to access the data scratchpad RAM.

To demonstrate use of the scratchpad capability, MIPS provides a default design that includes one contiguous 8KB RAM with cache like access. DSPRAM hit supersedes data cache hit. DSPRAM is indexed by virtual address. The hit information is based on the physical address in the base register. DSPRAM can be mapped to either cacheable or non-cacheable address space. A sophisticated arbitration scheme and instruction slip in the pipe prevents unnecessary stalls.

Only store instructions which are guaranteed to complete and hit in the DSPRAM, arbitrate for the RAM. The DMA access priority with respect to the core access is determined by the input pin *SI_DMA_Priority*. The DSPRAM interface supports multi-cycle access to the RAM array to accommodate slow devices or larger memory sizes. The interface allows addressing of DSPRAM sizes up to 1MB. The interface also supports 64-bit wide data access and provides a mechanism to back-stall the core pipeline.

### Instruction Scratchpad RAM (ISPRAM)

The 24Kf core can be configured to include an optional instruction scratchpad RAM independent of the instruction cache configuration. A separate OCP slave interface allows a DMA master to access the instruction scratchpad RAM.

To demonstrate use of the scratchpad capability, MIPS provides a default design that includes one contiguous 8KB RAM with cache like access. ISPRAM hit supersedes instruction cache hit. ISPRAM is indexed by virtual address. The hit information is based on the physical address in the base register. ISPRAM can be mapped to either cacheable or non-cacheable address space.

The DMA access priority with respect to the core access is determined by the input pin *SI_IDMA_Priority*. The ISPRAM interface supports multi-cycle access to the RAM array to accommodate slow devices or larger memory sizes. The interface allows addressing of ISPRAM sizes up to 1MB. The interface also supports 64-bit wide data access and provides a mechanism to back-stall the core pipeline.

### EJTAG Debug Support

The 24Kf core includes an Enhanced JTAG (EJTAG) block for use in the software debug of application and kernel code. In addition to standard user/supervisor/kernel modes of operation, the 24Kf core provides a Debug mode that is entered after a debug exception (derived from a hardware breakpoint, single-step exception, etc.) is taken and continues until a debug exception return (DERET) instruction is executed. During this time, the processor executes the debug exception handler routine.

Refer to the section called "External Interface Signals" on page 36 for a list of EJTAG interface signals.

The EJTAG interface operates through the Test Access Port (TAP), a serial communication port used for transferring test data in and out of the 24Kf core. In addition to the standard JTAG instructions, special instructions defined in the EJTAG specification define what registers are selected and how they are used.

#### Debug Registers

Three debug registers (*DEBUG*, *DEPC*, and *DESAVE*) have been added to the MIPS Coprocessor 0 (CP0) register set. The *DEBUG* register shows the cause of the debug exception and is used for setting up single-step operations. The *DEPC*, or Debug Exception Program Counter, 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 *DESAVE*, or Debug Exception Save, register enables the saving of general-purpose registers used during execution of the debug exception handler.

To exit debug mode, a Debug Exception Return (DERET) instruction is executed. When this instruction is executed, the system exits debug mode, allowing normal execution of application and system code to resume.

#### EJTAG Hardware Breakpoints

There are several types of simple hardware breakpoints defined in the EJTAG specification. These breakpoints stop the normal operation of the CPU and force the system into debug mode. There are two types of simple hardware breakpoints implemented in the 24Kf core: Instruction breakpoints and Data breakpoints.

During synthesis, the 24Kf core can be configured with or without hardware breakpoints. The following breakpoint options are supported:

- Zero or four instruction breakpoints
- Zero or two data breakpoints

Instruction breaks occur on instruction fetch operations, and the break is set on the virtual address. Instruction breaks can also be made on the ASID value used by the MMU. A mask can be applied to the virtual address to set breakpoints on a range of instructions.

Data breakpoints occur on load/store transactions. Breakpoints are set on virtual address and ASID values, similar to the Instruction breakpoint. Data breakpoints can be set on a load, a store, or both. Data breakpoints can also be set based on the value of the load/store operation. Finally, masks can be applied to both the virtual address and the load/store value.

#### MIPS Trace

The 24Kf core includes optional MIPS Trace support for real-time tracing of instruction addresses, data addresses and data values. The trace information is collected in an on-chip or off-chip memory, for post-capture processing by trace regeneration software.

On-chip trace memory may be configured in size from 0 to 8 MB; it is accessed through the existing EJTAG TAP interface and requires no additional chip pins. Off-chip trace memory is accessed through a special trace probe and can be configured to use 4, 8, or 16 data pins plus a clock.

## Testability

Testability for production testing of the core is supported through the use of internal scan and memory BIST.

### Internal Scan

Full mux-based scan for maximum test coverage is supported, with a configurable number of scan chains. ATPG test coverage can exceed 99%, depending on standard cell libraries and configuration options.

### Memory BIST

Memory BIST for the cache arrays, scratchpad memories and on-chip trace memory is optional, but can be implemented either through the use of integrated BIST features provided with the core, or inserted with an industry-standard memory BIST CAD tool.

#### Integrated Memory BIST

The core provides an integrated memory BIST solution for testing the internal cache SRAMs, scratchpad RAMs and on-chip trace RAM, using BIST controllers and logic tightly coupled to the cache subsystem. Several parameters associated with the integrated BIST controllers are configurable, including the algorithm (March C+ or IFA-13).

#### User-specified Memory BIST

Memory BIST can also be inserted with a CAD tool or other user-specified method. Wrapper modules and signal buses of configurable width are provided within the core to facilitate this approach.

## Build-Time Configuration Options

The 24Kf core allows a number of features to be customized based on the intended application. Table 16 summarizes the key configuration options that can be selected when the core is synthesized and implemented.

For a core that has already been built, software can determine the value of many of these options by querying an appropriate register field. Refer to the *MIPS32 24K Processor Core Family Software User's Manual* for a more complete description of these fields. The value of some options that do not have a functional effect on the core are not visible to software.

Table 16 Build-time Configuration Options

| Option | Choices | Software Visibility |
|---|---|---|
| Integer register file sets | 1, 2, or 4 | $\text{SRSCtl}_{\text{HSS}}$ |
| Integer register file implementation style | Flops or generator | N/A |
| Memory Management Type | TLB or FMT | $\text{Config}_{\text{MT}}$ |
| TLB Size | 16, 32, or 64 dual entries | $\text{Config1}_{\text{MMUSize}}$ |
| TLB data array implementation style | Flops or generator | N/A |
| Instruction hardware breakpoints | 0 or 4 | $\text{DCR}_{\text{IB}}$, $\text{IBS}_{\text{BCN}}$ |
| Data hardware breakpoints | 0 or 2 | $\text{DCR}_{\text{DB}}$, $\text{DBS}_{\text{BCN}}$ |
| MIPS Trace support | Present or not | $\text{Config3}_{\text{TL}}$ |
| MIPS Trace memory location | On-core, off-chip or both | $\text{TCBCONFIG}_{\text{OnT}}$,<br>$\text{TCBCONFIG}_{\text{OfT}}$ |
| MIPS Trace on-chip memory size | 256B - 8MB | $\text{TCBCONFIG}_{\text{SZ}}$ |
| MIPS Trace triggers | 0 - 8 | $\text{TCBCONFIG}_{\text{TRIG}}$ |
| CorExtend interface (Pro only) | Present or not | $\text{Config}_{\text{UDI}}$* |
| FPU clock ratio relative to integer core | 1:1 or 1:2 | $\text{Config7}_{\text{FPR}}$ |
| Coprocessor2 interface | Present or not | $\text{Config1}_{\text{C2}}$* |
| Instruction ScratchPad RAM interface | Present or not | $\text{Config}_{\text{ISP}}$* |
| Data ScratchPad RAM interface | Present or not | $\text{Config}_{\text{DSP}}$* |
| I-cache size | 16, 32, or 64 KB | $\text{Config1}_{\text{IS}}$ |
| D-cache size | 0, 16, 32, or 64 KB | $\text{Config1}_{\text{DL}}$, $\text{Config1}_{\text{DS}}$ |
| D-cache hardware aliasing support | Present or not (for 32KB only) | $\text{Config7}_{\text{AR}}$ |
| Cache parity | Present or not | $\text{ErrCtl}_{\text{PE}}$ |
| Memory BIST | Integrated (March C+ or March C+ plus IFA-13), custom, or none | N/A |
| Clock gating | Top-level, integer register file array, FPU register file array, TLB array, fine-grain, or none | N/A |
| * These bits indicate the presence of an external block. Bits will not be set if interface is present, but block is not. | | |

## Instruction Set

The 24Kf core instruction set complies with the MIPS32 instruction set architecture. Table 17 provides a summary of instructions implemented by the 24Kf core.

Table 17 24Kf™ Core Instruction Set

| Instruction | Description | Function |
|---|---|---|
| ABS.fmt | Floating Point Absolute Value<br>fmt = s,d | Fd = abs(Fs) |
| ADD | Integer Add | Rd = Rs + Rt |
| ADD.fmt | Floating Point Add<br>fmt = s,d | Fd = Fs + Ft |
| ADDI | Integer Add Immediate | Rt = Rs + Immed |
| ADDIU | Unsigned Integer Add Immediate | Rt = Rs $+_U$ Immed |
| ADDIUPC | Unsigned Integer Add Immediate to PC (MIPS16 only) | Rt = PC $+_u$ Immed |
| ADDU | Unsigned Integer Add | Rd = Rs $+_U$ Rt |
| AND | Logical AND | Rd = Rs & Rt |
| ANDI | Logical AND Immediate | Rt = Rs & ($0_{16}$ \|\| Immed) |
| ASMACRO | Application Specific Macro - allows macro sequences to be defined by implementor<br>(MIPS16 only) | Defined by implementor |
| B | Unconditional Branch<br>(Assembler idiom for: BEQ r0, r0, offset) | PC += (int)offset |
| BAL | Branch and Link<br>(Assembler idiom for: BGEZAL r0, offset) | GPR[31] = PC + 8<br>PC += (int)offset |
| BC1F | Branch On Floating Point False | if (cc[i] == 0) then<br>PC += (int)offset |
| BC1FL | Branch On Floating Point False Likely | if (cc[i] == 0)then<br>PC += (int)offset<br>else<br>Ignore Next Instruction |
| BC1T | Branch On Floating Point True | if(cc[i] == 1) then<br>PC += (int)offset |
| BC1TL | Branch On Floating Point True Likely | if (cc[i] == 1) then<br>PC += (int)offset<br>else<br>Ignore Next Instruction |
| BC2F | Branch On COP2 Condition False | if COP2Condition(cc) == 0<br>PC += (int)offset |
| BC2FL | Branch On COP2 Condition False Likely | if COP2Condition(cc) == 0<br>PC += (int)offset<br>else<br>Ignore Next Instruction |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| BC2T | Branch On COP2 Condition True | `if COP2Condition(cc) == 1`<br>`  PC += (int)offset` |
| BC2TL | Branch On COP2 Condition True Likely | `if COP2Condition(cc) == 1`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BEQ | Branch On Equal | `if Rs == Rt`<br>`  PC += (int)offset` |
| BEQL | Branch On Equal Likely | `if Rs == Rt`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BGEZ | Branch on Greater Than or Equal To Zero | `if !Rs[31]`<br>`  PC += (int)offset` |
| BGEZAL | Branch on Greater Than or Equal To Zero And Link | `GPR[31] = PC + 8`<br>`if !Rs[31]`<br>`  PC += (int)offset` |
| BGEZALL | Branch on Greater Than or Equal To Zero And Link Likely | `GPR[31] = PC + 8`<br>`if !Rs[31]`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BGEZL | Branch on Greater Than or Equal To Zero Likely | `if !Rs[31]`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BGTZ | Branch on Greater Than Zero | `if !Rs[31] && Rs != 0`<br>`  PC += (int)offset` |
| BGTZL | Branch on Greater Than Zero Likely | `if !Rs[31] && Rs != 0`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BLEZ | Branch on Less Than or Equal to Zero | `if Rs[31] \|\| Rs == 0`<br>`  PC += (int)offset` |
| BLEZL | Branch on Less Than or Equal to Zero Likely | `if Rs[31] \|\| Rs == 0`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BLTZ | Branch on Less Than Zero | `if Rs[31]`<br>`  PC += (int)offset` |
| BLTZAL | Branch on Less Than Zero And Link | `GPR[31] = PC + 8`<br>`if Rs[31]`<br>`  PC += (int)offset` |
| BLTZALL | Branch on Less Than Zero And Link Likely | `GPR[31] = PC + 8`<br>`if Rs[31]`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| BLTZL | Branch on Less Than Zero Likely | `if Rs[31]`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BNE | Branch on Not Equal | `if Rs != Rt`<br>`  PC += (int)offset` |
| BNEL | Branch on Not Equal Likely | `if Rs != Rt`<br>`  PC += (int)offset`<br>`else`<br>`  Ignore Next Instruction` |
| BREAK | Breakpoint | `Break Exception` |
| C.cond.fmt | Floating Point Compare<br>fmt = s,d | `cc[i] = Fs compare_cond Ft` |
| CACHE | Cache Operation | `See Software User’s Manual` |
| CEIL.L.fmt | Floating Point Ceiling to Long Fixed Point | `Fd = convert_and_round(Fs)` |
| CEIL.W.fmt | Floating Point Ceiling to Word Fixed Point | `Fd = convert_and_round(Fs)` |
| CFC1 | Move Control Word From Floating Point | `Rt = FP_Control[Fs]` |
| CFC2 | Move Control Word From Coprocessor 2 | `Rt = CCR[2, Rs]` |
| CLO | Count Leading Ones | `Rd = NumLeadingOnes(Rs)` |
| CLZ | Count Leading Zeroes | `Rd = NumLeadingZeroes(Rs)` |
| COP0 | Coprocessor 0 Operation | `See Software User’s Manual` |
| COP2 | Coprocessor 2 Operation | `See Coprocessor 2`<br>`Description` |
| CTC1 | Move Control Word To Floating Point | `FP_Control[Fs] = Rt` |
| CTC2 | Move Control Word To Coprocessor 2 | `CCR[2, n] = Rt` |
| CVT.D.fmt | Floating Point Convert to Double Floating Point<br>fmt = S,W,L | `Fd = convert_and_round(Fs)` |
| CVT.D.fmt | Floating Point Convert to Double Floating Point<br>fmt = S,W,L | `Fd = convert_and_round(Fs)` |
| CVT.L.fmt | Floating Point Convert to Long Fixed Point<br>fmt = S,D | `Fd = convert_and_round(Fs)` |
| CVT.S.fmt | Floating Point Convert to Single Floating Point<br>fmt = W,D,L | `Fd = convert_and_round(Fs)` |
| CVT.W.fmt | Floating Point Convert to Word Fixed Point<br>fmt = S,D | `Fd = convert_and_round(Fs)` |
| DERET | Return from Debug Exception | `PC = DEPC`<br>`Exit Debug Mode` |
| DI | Atomically Disable Interrupts | Rt = Status; $\text{Status}_{IE}$ = 0 |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| DIV | Divide | `LO = (int)Rs / (int)Rt`<br>`HI = (int)Rs % (int)Rt` |
| DIV.fmt | Floating Point Divide<br>fmt = S,D | `Fd = Fs/Ft` |
| DIVU | Unsigned Divide | `LO = (uns)Rs / (uns)Rt`<br>`HI = (uns)Rs % (uns)Rt` |
| EHB | Execution Hazard Barrier | Stop instruction execution until execution hazards are cleared |
| EI | Atomically Enable Interrupts | Rt = Status; $\text{Status}_{IE}$ = 1 |
| ERET | Return from Exception | `if SR[2]`<br>`  PC = ErrorEPC`<br>`else`<br>`  PC = EPC`<br>`  SR[1] = 0`<br>`SR[2] = 0`<br>`LL = 0` |
| EXT | Extract Bit Field | `Rt = ExtractField(Rs, pos, size)` |
| FLOOR.L.fmt | Floating Point Floor to Long Fixed Point<br>fmt = S,D | `Fd = convert_and_round(Fs)` |
| FLOOR.W.fmt | Floating Point Floor to Word Fixed Point<br>fmt = S,D | `Fd = convert_and_round(Fs)` |
| INS | Insert Bit Field | `Rt = InsertField(Rs, Rt, pos, size)` |
| J | Unconditional Jump | `PC = PC[31:28] \|\| offset<<2` |
| JAL | Jump and Link | `GPR[31] = PC + 8`<br>`PC = PC[31:28] \|\| offset<<2` |
| JALR | Jump and Link Register | `Rd = PC + 8`<br>`PC = Rs` |
| JALR.HB | Jump and Link Register with Hazard Barrier | Like JALR, but also clears execution and instruction hazards |
| JALRC | Jump and Link Register Compact - do not execute instruction in jump delay slot(MIPS16 only) | `Rd = PC + 2`<br>`PC = Rs` |
| JR | Jump Register | `PC = Rs` |
| JR.HB | Jump Register with Hazard Barrier | Like JR, but also clears execution and instruction hazards |
| JRC | Jump Register Compact - do not execute instruction in jump delay slot (MIPS16 only) | `PC = Rs` |
| LB | Load Byte | `Rt = (byte)Mem[Rs+offset]` |
| LBU | Unsigned Load Byte | `Rt = (ubyte))Mem[Rs+offset]` |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| LDC1 | Load Doubleword to Floating Point | `Ft = memory[base+offset]` |
| LDC2 | Load Doubleword to Coprocessor 2 | `CPR[2,Rt] = Mem[Rs+offset]` |
| LDXC1 | Load Doubleword Indexed to Floating Point | `Fd = memory[base+index]` |
| LH | Load Halfword | `Rt = (half)Mem[Rs+offset]` |
| LHU | Unsigned Load Halfword | `Rt = (uhalf)Mem[Rs+offset]` |
| LL | Load Linked Word | `Rt = Mem[Rs+offset]`<br>`LL = 1`<br>`LLAdr = Rs + offset` |
| LUI | Load Upper Immediate | `Rt = immediate << 16` |
| LUXC1 | Load Doubleword Indexed Unaligned to Floating Point | `Fd = memory[(base+index)psize-1..3` |
| LW | Load Word | `Rt = Mem[Rs+offset]` |
| LWC1 | Load Word to Floating Point | `Ft = memory[base+offset]` |
| LWC2 | Load Word To Coprocessor 2 | `CPR[2,Rt] = (word)Mem[Rs+offset]` |
| LWPC | Load Word, PC relative | `Rt = Mem[PC+offset]` |
| LWXC1 | Load Word Indexed to Floating Point | `Fd = memory[base+index]` |
| LWL | Load Word Left | `See Architecture Reference Manual` |
| LWR | Load Word Right | `See Architecture Reference Manual` |
| MADD | Multiply-Add | `HI \| LO += (int)Rs * (int)Rt` |
| MADD.fmt | Floating Point Multiply Add<br>fmt = S,D | `Fd = Fs * Ft + Fr` |
| MADDU | Multiply-Add Unsigned | `HI \| LO += (uns)Rs * (uns)Rt` |
| MFC0 | Move From Coprocessor 0 | `Rt = CPR[0, Rd, sel]` |
| MFC1 | Move From FPR | `Rt = Fs` |
| MFHC1 | Move From High Half of FPR | Rt = $Fs_{63..32}$ |
| MFC2 | Move From Coprocessor 2 | `Rt = CPR[2, Rd, sel]` |
| MFHC2 | Move From High Half of Coprocessor 2 | Rt = CPR[2, Rd, $sel]_{63..32}$ |
| MFHI | Move From HI | `Rd = HI` |
| MFLO | Move From LO | `Rd = LO` |
| MOV.fmt | Floating Point Move | `Fd = Fs` |
| MOVF | GPR Conditional Move on Floating Point False | `if (cc[i] == 0) then Rd = Rs` |
| MOVF.fmt | FPR Conditional Move on Floating Point False | `if (cc[i] == 0) then Fd = Fs` |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| MOVN | GPR Conditional Move on Not Zero | if Rt ≠ 0 then<br>  Rd = Rs |
| MOVN.fmt | FPR Conditional Move on Not Zero | if Rt ≠ 0 then<br>  Fd = Fs |
| MOVT | GPR Conditional Move on Floating Point True | if (cc[i] == 1) then Rd = Rs |
| MOVT.fmt | FPR Conditional Move on Floating Point True | if (cc[i] == 1) then Fd = Fs |
| MOVZ | GPR Conditional Move on Zero | if Rt = 0 then<br>  Rd = Rs |
| MOVZ.fmt | FPR Conditional Move on Zero | if (Rt == 0) then Fd = Fs |
| MSUB | Multiply-Subtract | HI \| LO -= (int)Rs * (int)Rt |
| MSUB.fmt | Floating Point Multiply Subtract<br>fmt = S,D | Fd = Fs * Ft - Fr |
| MSUBU | Multiply-Subtract Unsigned | HI \| LO -= (uns)Rs * (uns)Rt |
| MTC0 | Move To Coprocessor 0 | CPR[0, n, Sel] = Rt |
| MTC1 | Move To FPR | Fs = Rt |
| MTHC1 | Move To High Half of FPR | Fd = Rt \|\| Fs$_{31..0}$ |
| MTC2 | Move To Coprocessor 2 | CPR[2, n, sel] = Rt |
| MTHC2 | Move To High Half of Coprocessor 2 | CPR[2, Rd, sel] = Rt \|\|<br>CPR[2, Rd, sel]$_{31..0}$ |
| MTHI | Move To HI | HI = Rs |
| MTLO | Move To LO | LO = Rs |
| MUL | Multiply with register write | HI \| LO =Unpredictable<br>Rd = ((int)Rs * (int)Rt)$_{31..0}$ |
| MUL.fmt | Floating Point Multiply<br>fmt = S,D | Fd = Fs * Ft |
| MULT | Integer Multiply | HI \| LO = (int)Rs * (int)Rd |
| MULTU | Unsigned Multiply | HI \| LO = (uns)Rs * (uns)Rd |
| NEG.fmt | Floating Point Negate<br>fmt = S,D | Fd = neg(Fs) |
| NMADD.fmt | Floating Point Negative Multiply Add<br>fmt = S,D | Fd = neg(Fs * Ft + Fr) |
| NMSUB.fmt | Floating Point Negative Multiply Subtract<br>fmt = S,D | Fd = neg(Fs * Ft - Fr) |
| NOP | No Operation<br>(Assembler idiom for: SLL r0, r0, r0) | |
| NOR | Logical NOR | Rd = ~(Rs \| Rt) |
| OR | Logical OR | Rd = Rs \| Rt |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| ORI | Logical OR Immediate | Rt = Rs \| Immed |
| PREF | Prefetch | Load Specified Line into Cache |
| PREFX | Prefetch Indexed | Load Specified Line into Cache |
| RDHWR | Read Hardware Register | Allows unprivileged access to registers enabled by HWREna register |
| RDPGPR | Read GPR from Previous Shadow Set | Rt = SGPR[$\text{SRSCtl}_{PSS}$, Rd] |
| RECIP.fmt | Floating Point Reciprocal Approximation<br>fmt = S,D | Fd = recip(Fs) |
| RESTORE | Restore registers and deallocate stack frame (MIPS16 only) | See Architecture Reference Manual |
| ROTR | Rotate Word Right | Rd = $\text{Rt}_{sa-1..0}$ \|\| $\text{Rt}_{31..sa}$ |
| ROTRV | Rotate Word Right Variable | Rd = $\text{Rt}_{Rs-1..0}$ \|\| $\text{Rt}_{31..Rs}$ |
| ROUND.L.fmt | Floating Point Round to Long Fixed Point<br>fmt = S,D | Fd = convert_and_round(Fs) |
| ROUND.W.fmt | Floating Point Round to Word Fixed Point<br>fmt = S,D | Fd = convert_and_round(Fs) |
| RSQRT.fmt | Floating Point Reciprocal Square Root Approximation<br>fmt = S,D | Fd = rsqrt(Fs) |
| SAVE | Save registers and allocate stack frame (MIPS16 only) | See Architecture Reference Manual |
| SB | Store Byte | (byte)Mem[Rs+offset] = Rt |
| SC | Store Conditional Word | if LL = 1<br>mem[Rs+offset] = Rt<br>Rt = LL |
| SDBBP | Software Debug Break Point | Trap to SW Debug Handler |
| SDC1 | Store Doubleword from Floating Point | memory[base+offset] = Ft |
| SDC2 | Store Doubleword from Coprocessor 2 | Mem[Rs+offset] = CPR[2,Rt] |
| SDXC1 | Store Word Indexed from Floating Point | memory[base+index] = Fs |
| SEB | Sign Extend Byte | Rd = (byte)Rs |
| SEH | Sign Extend Half | Rd = (half)Rs |
| 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] |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| SLT | Set on Less Than | if (int)Rs < (int)Rt<br>  Rd = 1<br>else<br>  Rd = 0 |
| SLTI | Set on Less Than Immediate | if (int)Rs < (int)Immed<br>  Rt = 1<br>else<br>  Rt = 0 |
| SLTIU | Set on Less Than Immediate Unsigned | if (uns)Rs < (uns)Immed<br>  Rt = 1<br>else<br>  Rt = 0 |
| SLTU | Set on Less Than Unsigned | if (uns)Rs < (uns)Immed<br>  Rd = 1<br>else<br>  Rd = 0 |
| SQRT.fmt | Floating Point Square Root<br>fmt = S,D | Fd = sqrt(Fs) |
| SRA | Shift Right Arithmetic | Rd = (int)Rt >> sa |
| SRAV | Shift Right Arithmetic Variable | Rd = (int)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 | NOP |
| SUB | Integer Subtract | Rt = (int)Rs - (int)Rd |
| SUB.fmt | Floating Point Subtract<br>fmt = S,D | Fd = Fs - Ft |
| SUBU | Unsigned Subtract | Rt = (uns)Rs - (uns)Rd |
| SUXC1 | Store Doubleword Indexed Unaligned from Floating Point | memory[(base+index)psize-1..3] = Fs |
| SW | Store Word | Mem[Rs+offset] = Rt |
| SWC1 | Store Word From Floating Point | Mem[Rs+offset] = Fs |
| SWC2 | Store Word From Coprocessor 2 | Mem[Rs+offset] = CPR[2,Rt]$_{31..0}$ |
| SWL | Store Word Left | See Architecture Reference Manual |
| SWR | Store Word Right | See Architecture Reference Manual |
| SWXC1 | Store Word Indexed to Floating Point | memory[base+index] = Fs |
| SYNC | Synchronize | See Software User's Manual |
| SYNCI | Synchronize Caches to Make Instruction Writes Effective | Fore D-cache writeback and I-cache invalidate on specified address |

Table 17 24Kf™ Core Instruction Set (Continued)

| Instruction | Description | Function |
|---|---|---|
| SYSCALL | System Call | SystemCallException |
| TEQ | Trap if Equal | if Rs == Rt<br>TrapException |
| TEQI | Trap if Equal Immediate | if Rs == (int)Immed<br>TrapException |
| TGE | Trap if Greater Than or Equal | if (int)Rs >= (int)Rt<br>TrapException |
| TGEI | Trap if Greater Than or Equal Immediate | if (int)Rs >= (int)Immed<br>TrapException |
| TGEIU | Trap if Greater Than or Equal Immediate Unsigned | if (uns)Rs >= (uns)Immed<br>TrapException |
| TGEU | Trap if Greater Than or Equal Unsigned | if (uns)Rs >= (uns)Rt<br>TrapException |
| TLBWI | Write Indexed TLB Entry | See Software Users Manual |
| TLBWR | Write Random TLB Entry | See Software Users Manual |
| TLBP | Probe TLB for Matching Entry | See Software Users Manual |
| TLBR | Read Index for TLB Entry | See Software Users Manual |
| TLT | Trap if Less Than | if (int)Rs < (int)Rt<br>TrapException |
| TLTI | Trap if Less Than Immediate | if (int)Rs < (int)Immed<br>TrapException |
| TLTIU | Trap if Less Than Immediate Unsigned | if (uns)Rs < (uns)Immed<br>TrapException |
| TLTU | Trap if Less Than Unsigned | if (uns)Rs < (uns)Rt<br>TrapException |
| TNE | Trap if Not Equal | if Rs != Rt<br>TrapException |
| TNEI | Trap if Not Equal Immediate | if Rs != (int)Immed<br>TrapException |
| TRUNC.L.fmt | Floating Point Truncate to Long Fixed Point | Fd = convert_and_round(Fs) |
| TRUNC.W.fmt | Floating Point Truncate to Word Fixed Point | Fd = convert_and_round(Fs) |
| WAIT | Wait for Interrupts | Stall until interrupt occurs |
| WRPGPR | Write to GPR in Previous Shadow Set | SGPR[$\text{SRSCtl}_{\text{PSS}}$, Rd] = Rt |
| WSBH | Word Swap Bytes Within HalfWords | Rd = $\text{Rt}_{23..16}$ \|\| $\text{Rt}_{31..24}$ \|\|<br>$\text{Rt}_{7..0}$ \|\| $\text{Rt}_{15..8}$ |
| XOR | Exclusive OR | Rd = Rs ^ Rt |
| XORI | Exclusive OR Immediate | Rt = Rs ^ (uns)Immed |
| ZEB | Zero extend byte (MIPS16 only) | Rt = (ubyte) Rs |
| ZEH | Zero extend half (MIPS16 only) | Rt = (uhalf) Rs |

## External Interface Signals

This section describes the signal interface of the 24Kf microprocessor core.

The pin direction key for the signal descriptions is shown in Table 18 below.

The 24Kf core signals are listed in Table 19 below. Note that the signals are grouped by logical function, not by expected physical location. All signals, with the exception of *EJ_TRST_N*, are active-high signals. *EJ_DINT* and *SI_NMI* go through edge-detection logic so that only one exception is taken each time they are asserted.

Table 18 24Kf™ Core Signal Direction Key

| Dir | Description |
|---|---|
| I | Input to the 24Kf core sampled on the rising edge of the appropriate CLK signal. |
| O | Output of the 24Kf core, unless otherwise noted, driven at the rising edge of the appropriate CLK signal. |
| A | Asynchronous inputs that are synchronized by the core. |
| S | Static input to the 24Kf core. These signals are normally tied to either power or ground and should not change state while *SI_Reset* is deasserted. |
| SO | Static output from the 24Kf core. |

Table 19 24Kf™ Core Signal Descriptions

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td colspan="3"><b>System Interface</b></td></tr>
<tr><td colspan="3"><i>Clock Signals:</i></td></tr>
<tr><td><i>SI_ClkIn</i></td><td>I</td><td>Clock Input.</td></tr>
<tr><td><i>SI_OCPSync</i></td><td>I</td><td>Signal indicating phase and frequency relationships between <i>SI_ClkIn</i> and the OCP clock domain.The width of this pulse is related to an <i>SI_ClkIn</i> period.<br>Note that no direct OCP clock input is present on the core. Instead, all bus interface flops are clocked with the high-speed core clock, and the <i>SI_OCPSync</i> signal is used to indicate when inputs are sampled or outputs are enabled. The pattern for various OCP-to-core clock ratios is shown in the table below, assuming the pattern starts from the point where the rising edges of both clocks are aligned:
<table>
<tr><th>Clock Ratio</th><th>Sync Pattern</th></tr>
<tr><td>1:1</td><td><b>1</b>11...</td></tr>
<tr><td>1:1.5</td><td><b>110</b>110...</td></tr>
<tr><td>1:2</td><td><b>10</b>10...</td></tr>
<tr><td>1:2.5</td><td><b>01010</b>01010...</td></tr>
<tr><td>1:3</td><td><b>010</b>010...</td></tr>
<tr><td>1:3.5</td><td><b>0010010</b>0010010...</td></tr>
<tr><td>1:4</td><td><b>0010</b>0010...</td></tr>
<tr><td>1:5</td><td><b>00010</b>00010...</td></tr>
</table>
</td></tr>
<tr><td><i>SI_ClkOut</i></td><td>O</td><td>Reference Clock for external use. This clock signal provides a reference for deskewing any clock insertion delay created by the internal clock buffering in the core.</td></tr>
<tr><td colspan="3"><i>Reset Signals:</i></td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| *SI_NMI* | A | Non-Maskable Interrupt. An edge detect is used on this signal. When this signal is sampled asserted (high) one clock after being sampled deasserted, an NMI is posted to the core. |
| *SI_Reset* | A | Reset Signal. Causes a Reset Exception in the core. |
| *Power Management Signals:* | | |
| *SI_ERL* | O | This signal represents the state of the ERL bit (2) in the CP0 *Status* register and indicates the error level. The core asserts *SI_ERL* whenever a Reset, CacheError, or NMI exception is taken. |
| *SI_EXL* | O | This signal represents the state of the EXL bit (1) in the CP0 *Status* register and indicates the exception level. The core asserts *SI_EXL* whenever any exception other than a Reset, Cache Error, NMI, or Debug exception is taken. |
| *SI_RP* | O | This signal represents the state of the RP bit (27) in the CP0 *Status* register. Software can write this bit to indicate that a reduced power mode may be entered. |
| *SI_Sleep* | O | This signal is asserted by the core whenever the WAIT instruction is executed. The assertion of this signal indicates that the clock has stopped and that the core is waiting for an interrupt. |
| *Interrupt Signals:* | | |
| *SI_EICPresent* | S | Indicates whether an external interrupt controller is present. Value is visible to software in the $Config3_{VEIC}$ register field. |
| *SI_EISS[3:0]* | I | General purpose register shadow set number to be used when servicing an interrupt in EIC interrupt mode. |
| *SI_IAck* | O | Interrupt acknowledge indication for use in external interrupt controller mode. This signal is active for a single *SI_ClkIn* cycle when an interrupt is taken. When the processor initiates the interrupt exception, it loads the value of the *SI_Int[5:0]* pins into the $Cause_{RIPL}$ field (overlaid with $Cause_{IP7..IP2}$), and signals the external interrupt controller to notify it that the current interrupt request is being serviced. This allows the controller to advance to another pending higher-priority interrupt, if desired. |

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| *SI_Int[5:0]* | I/A | Active high Interrupt pins. These signals are driven by external logic and when asserted indicate an interrupt exception to the core. The interpretation of these signals depends on the interrupt mode in which the core is operating; the interrupt mode is selected by software.<br>The *SI_Int* signals go through synchronization logic and can be asserted asynchronously to *SI_ClkIn.* In External Interrupt Controller (EIC) mode, however, the interrupt pins are interpreted as an encoded value, so they must be asserted synchronously to *SI_ClkIn* to guarantee that all bits are received by the core in a particular cycle.<br>The interrupt pins are level sensitive and should remain asserted until the interrupt has been serviced.<br>In Release 1 Interrupt Compatibility mode:<br>• All 6 interrupt pins have the same priority as far as the hardware is concerned.<br>• Interrupts are non-vectored.<br>In Vectored Interrupt (VI) mode:<br>• The *SI_Int* pins are interpreted as individual hardware interrupt requests.<br>• Internally, the core prioritizes the hardware interrupts and chooses an interrupt vector.<br>In External Interrupt Controller (EIC) mode:<br>• An external block prioritizes its various interrupt requests and produces a vector number of the highest priority interrupt to be serviced.<br>• The vector number is driven on the *SI_Int* pins, and is treated as a 6-bit encoded value in the range of 0..63.<br>• When the core starts the interrupt exception, signaled by the assertion of *SI_IAck*, it loads the value of the *SI_Int[5:0]* pins into the $\mathit{Cause}_{\mathit{RIPL}}$ field (overlaid with $\mathit{Cause}_{\mathit{IP7..IP2}}$). The interrupt controller can then signal another interrupt. |
| *SI_IPL[5:0]* | O | Current interrupt priority level from the $\mathit{Cause}_{\mathit{IPL}}$ register field, provided for use by an external interrupt controller. This value is updated whenever *SI_IAck* is asserted. |
| *SI_IPPCI[2:0]* | S | Indicates the *SI_Int* hardware interrupt pin that the performance counter interrupt pin (*SI_PCInt*) is combined with external to the core. The value of this bus is visible to software in the $\mathit{IntCtl}_{\mathit{IPPCI}}$ register field. |
| *SI_IPTI[2:0]* | S | Indicates the *SI_Int* hardware interrupt pin that the timer interrupt pin (*SI_TimerInt*) is combined with external to the core. The value of this bus is visible to software in the $\mathit{IntCtl}_{\mathit{IPTI}}$ register field. |
| SI_PCInt | O | Performance Counter Interrupt. Asserted when bit 31 of any of the performance counters is set. This hardware pin represents the state of the $\mathit{Cause}_{\mathit{PC}}$ register field<br>For Release 1 Interrupt Compatibility mode or Vectored Interrupt mode:<br>In order for the core to take a performance counter interrupt, the *SI_PCInt* signal needs to be brought back into the core on one of the six *SI_Int* interrupt pins in a system-dependent manner. Traditionally, this has been accomplished by muxing *SI_PCInt* with *SI_Int[5]*. Exposing *SI_PCInt* as an output allows more flexibility for the system designer. Performance counter interrupts can be muxed or ORed into one of the interrupts, as desired in a particular system. The *SI_Int* hardware interrupt pin with which the *SI_PCInt* signal is merged is indicated via the *SI_IPPCI* static input pins.<br>For External Interrupt Controller (EIC) mode:<br>The *SI_PCInt* signal is provided to the external interrupt controller, which then prioritizes the performance counter interrupt with all other interrupt sources, as desired. The controller then encodes the desired interrupt value on the *SI_Int* pins. Since *SI_Int* is usually encoded, the *SI_IPPCI* pins are not meaningful in EIC mode. |

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>SI_SWInt[1:0]</i></td><td>O</td><td>Software interrupt request. These signals represent the value in the <i>IP[1:0]</i> field of the <i>Cause</i> register. They are provided for use by an external interrupt controller.</td></tr>
<tr><td><i>SI_TimerInt</i></td><td>O</td><td>Timer interrupt indication. This signal is asserted whenever the <i>Count</i> and <i>Compare</i> registers match and is deasserted when the <i>Compare</i> register is written. This hardware pin represents the value of the $Cause_{TI}$ register field.<br>Like SI_PCInt, this signal should be brought back into the core via one of the SI_Int pins. For compatibility or vectored interrupt mode, SI_IPTI should indicate which interrupt pin it has been merged with.</td></tr>
<tr><td colspan="3"><i>Configuration Inputs:</i></td></tr>
<tr><td><i>SI_SBlock</i></td><td>S</td><td>Controls the ordering of double-words within a bursted read request on the OCP interface. The value of this pin is visible in the BM field of the <i>Config0</i> register.
<table>
<tr><th><i>SI_SBlock</i></th><th>Burst Order</th></tr>
<tr><td>0</td><td>Sequential</td></tr>
<tr><td>1</td><td>Subblock</td></tr>
</table></td></tr>
<tr><td><i>SI_CPUNum[9:0]</i></td><td>S</td><td>Unique identifier to specify an individual core in a multi-processor system. The hardware value specified on these pins is available in the CPUNum field of the <i>EBase</i> register, so it can be used by software to distinguish a particular processor. In a single processor system, this value should be set to zero.</td></tr>
<tr><td><i>SI_Endian</i></td><td>S</td><td>Indicates the base endianness of the core. The value of this pin is visible in the BE field of the <i>Config0</i> register.
<table>
<tr><th><i>EB_Endian</i></th><th>Base Endian Mode</th></tr>
<tr><td>0</td><td>Little Endian</td></tr>
<tr><td>1</td><td>Big Endian</td></tr>
</table></td></tr>
<tr><td><i>SI_SimpleBE</i></td><td>S</td><td>The state of this signal can constrain the core to only generate certain byte enables on external interface transactions. This eases connection to some existing bus standards. The value of this pin is visible in the SB field of the <i>Config0</i> register.
<table>
<tr><th><i>SI_SimpleBE</i></th><th>Byte Enable Mode</th></tr>
<tr><td>0</td><td>All BEs allowed</td></tr>
<tr><td>1</td><td>Naturally aligned bytes, halfwords, words, and doublewords only</td></tr>
</table></td></tr>
<tr><td><i>SI_DMA_Priority</i></td><td>I</td><td>Force the DMA to have a higher priority</td></tr>
<tr><td><i>SI_IDMA_Priority</i></td><td>I</td><td>DMA request should be higher priority than core requests</td></tr>
<tr><td><i>SI_Ibs[3:0]</i></td><td>O</td><td>This signal reflects the state of the BS bits[3:0] in the Instruction Breakpoint Status (IBS) register when EJTAG hardware breakpoint for instruction is implemented.</td></tr>
<tr><td><i>SI_Dbs[1:0]</i></td><td>O</td><td>This signal reflects the state of the BS bits[1:0] in the Data Breakpoint Status (IBS) register when EJTAG hardware breakpoint for data is implemented.</td></tr>
<tr><td colspan="3"><b>L2 Interface</b>: Static inputs are needed to set up the CP0 <i>Config2</i> register if a Level 2 cache is present. Additional inputs are provided for performance counters related to an L2 cache.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>L2_LineSize[3:0]</i></td><td>S</td><td>Encoded line size of the external L2 cache. The value of these pins is visible in the SL field of the <i>Config2</i> register. Note that a value of 0 indicates that no L2 cache is present.
<table>
<tr><th>Encoding</th><th>L2 Line Size (bytes)</th></tr>
<tr><td>0</td><td>No L2 cache present</td></tr>
<tr><td>1</td><td>4</td></tr>
<tr><td>2</td><td>8</td></tr>
<tr><td>3</td><td>16</td></tr>
<tr><td>4</td><td>32</td></tr>
<tr><td>5</td><td>64</td></tr>
<tr><td>6</td><td>128</td></tr>
<tr><td>7</td><td>256</td></tr>
<tr><td>8-15</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td><i>L2_Sets[3:0]</i></td><td>S</td><td>Encoded number of L2 sets per way. The value of these pins is visible in the SS field of the <i>Config2</i> register.
<table>
<tr><th>Encoding</th><th>L2 Sets Per Way</th></tr>
<tr><td>0</td><td>64</td></tr>
<tr><td>1</td><td>128</td></tr>
<tr><td>2</td><td>256</td></tr>
<tr><td>3</td><td>512</td></tr>
<tr><td>4</td><td>1024</td></tr>
<tr><td>5</td><td>2048</td></tr>
<tr><td>6</td><td>4096</td></tr>
<tr><td>7</td><td>8192</td></tr>
<tr><td>8-15</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td><i>L2_Assoc[3:0]</i></td><td>S</td><td>Encoded associativity of the L2 cache. The value of these pins is visible in the SA field of the <i>Config2</i> register.
<table>
<tr><th>Encoding</th><th>L2 Associativity</th></tr>
<tr><td>0</td><td>Direct mapped</td></tr>
<tr><td>1</td><td>2</td></tr>
<tr><td>2</td><td>3</td></tr>
<tr><td>3</td><td>4</td></tr>
<tr><td>4</td><td>5</td></tr>
<tr><td>5</td><td>6</td></tr>
<tr><td>6</td><td>7</td></tr>
<tr><td>7</td><td>8</td></tr>
<tr><td>8-15</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td><i>L2_PCWB</i></td><td>I</td><td>Performance counter input. Indicates the number of L2 write backs (WB). One pulse (OCP clock width) per L2 WB event. The count is visible in CP0 <i>Performance Counter Register 0 Count.</i></td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>L2_PCAcc</i></td><td>I</td><td>Performance counter input. Indicates the number of L2 accesses. One pulse (OCP clock width) per L2 access event. The count is visible in CP0 <i>Performance Counter Register 1 Count.</i></td></tr>
<tr><td><i>L2_PCMiss</i></td><td>I</td><td>Performance counter input. Indicates the number of L2 misses. One pulse (OCP clock width) per L2 miss event. The count is visible in CP0 <i>Performance Counter Register 0 and 1 Count.</i></td></tr>
<tr><td><i>L2_PCMissCy</i></td><td>I</td><td>Performance counter input. Indicates the number of cycles the L2 is held due to misses. Note that this is not an <i>event</i> unlike <i>L2_PCWB</i>, <i>L2_PCAcc</i>, or <i>L2_PCMiss</i>. 1 pulse (OCP clock width) per L2 miss cycle. Also note that the count is in terms of OCP cycles and not <i>SI_ClkIn</i> clock cycles. This needs to be factored in while reading this counter.<br>Note: the count related to this signal is not currently visible in a CP0 <i>Performance Counter Register.</i><br>These 4 inputs can be redefined if no L2 is present or if desired. The count will in this case contain the number of OCP clock cycles this signal was high.</td></tr>
<tr><td colspan="3"><b>OCP Master System Interface</b>: These signals connect to the OCP Standard Master Interface.</td></tr>
<tr><td><i>OC_MCmd[2:0]</i></td><td>O</td><td>OCP command bus, indicates the type of transaction requested. Only some encoding are used and they are set in concert with the values on OC_MReqInfo and OC_MAddrSpace. The encoding used by the 24Kf core are shown in the following table:
<table>
<tr><th>Encoding</th><th>Command</th><th>Mnemonic</th><th>Description</th></tr>
<tr><td>0</td><td>Idle</td><td>IDLE</td><td>No transaction</td></tr>
<tr><td>1</td><td>Write</td><td>WR</td><td>Used for data write and L2 CACHE write or invalidate</td></tr>
<tr><td>2</td><td>Read</td><td>RD</td><td>Used for fetch or data read or L2 CACHE reads or SYNC.</td></tr>
<tr><td>3-7</td><td>Unused</td><td>-</td><td>Not used on 24Kf core</td></tr>
</table>
</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>OC_MReqInfo[3:0]</i></td><td>O</td><td>
<p>OCP command bus extension.</p>
<p><i>For transactions other than SYNC and CACHE, the OC_MReqInfo[2:0]</i> field encodes the cacheability attributes for a transaction; it uses the same encoding as the CCA field described in Table 9 on page 16.</p>
<p><i>OC_MReqInfo[3]</i> indicates that the transaction is due to a SYNC instruction; when this bit is high, the lower bits [2:0] indicate an uncached CCA type.</p>
<p>The encoding of the <i>OC_MReqInfo</i> field for all transactions other than CACHE is summarized in the following sub-table:</p>
<p><b>Encoding for all transactions other than CACHE</b></p>
<table>
<tr><th>Encoding</th><th>Command Information</th></tr>
<tr><td>0</td><td>cacheable, noncoherent, WT, NWA</td></tr>
<tr><td>1</td><td>reserved</td></tr>
<tr><td>2</td><td>uncached</td></tr>
<tr><td>3</td><td>cacheable, noncoherent, WB, WA</td></tr>
<tr><td>4-6</td><td>reserved</td></tr>
<tr><td>7</td><td>uncached accelerated</td></tr>
<tr><td>8-9</td><td>reserved</td></tr>
<tr><td>10</td><td>SYNC with uncached CCA</td></tr>
<tr><td>11-15</td><td>reserved</td></tr>
</table>
<p>If the transaction is a CACHE transaction to an off-core L2/L3 cache, then the lower 3 bits of <i>OC_MReqInfo</i> are identical to bits 20:18 of the CACHE opcode and indicate the type of operation (See the <i>MIPS32 24K Processor Core Family Software User's Manual</i> for details). This encoding is shown in the sub-table below. Note that a L2/L3 CACHE transaction is identified when one of the bits of <i>MAddrSpace[1:0]</i> are set to 1.</p>
<p>The encoding of the <i>OC_MReqInfo</i> field for the CACHE transaction is summarized in the following sub-table:</p>
<p><b>Encoding for CACHE transaction</b></p>
<table>
<tr><th>Encoding</th><th>Command Information</th></tr>
<tr><td>0</td><td>index Writeback Invalidate/ Index Invalidate</td></tr>
<tr><td>1</td><td>index Load Tag</td></tr>
<tr><td>2</td><td>index Store Tag</td></tr>
<tr><td>3</td><td>reserved</td></tr>
<tr><td>4</td><td>hit invalidate</td></tr>
<tr><td>5</td><td>hit writeback invalidate/ hit invalidate</td></tr>
<tr><td>6</td><td>hit writeback</td></tr>
<tr><td>7-15</td><td>reserved</td></tr>
</table>
</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>OC_MAddrSpace[1:0]</i></td><td>O</td><td>L2/L3 Address Space indicator. When the 24Kf core is issuing an L2 or an L3 CACHE operation, the corresponding bit (Bit [0] for L2, and Bit [1] for L3) is asserted. It indicates to the system that this OCP command is targeted to the address space of the L2 or L3 Cache.<br>The encoding of this field is summarized in the following table:
<table>
<tr><th>Encoding</th><th>Address Space</th></tr>
<tr><td>0</td><td>Normal address space</td></tr>
<tr><td>1</td><td>L2 address space</td></tr>
<tr><td>2</td><td>L3 address space</td></tr>
<tr><td>3</td><td>reserved</td></tr>
</table></td></tr>
<tr><td><i>OC_MAddr[31:3]</i></td><td>O</td><td>Physical doubleword address bus. Note that the least significant 3 address bits not included in this bus are decoded in the read (<i>OC_MByteEn</i>) or write (<i>OC_MDataByteEn</i>) byte enable fields. When <i>OC_MAddrSpace</i>[1:0] is not zero (to indicate a CACHE operation), then <i>OC_MAddr[31:5]</i> carries the cache line address (or the cache line index for Indexed CACHE ops).</td></tr>
<tr><td><i>OC_MBurstSeq[2:0]</i></td><td>O</td><td>Indicates type of burst sequence. The 24Kf core can only generate two possible values, determined by the <i>SI_SBlock</i> static input, as shown in the following table:
<table>
<tr><th>Encoding</th><th>Burst Sequence</th></tr>
<tr><td>2</td><td>Sequential: Critical dword first, with linear wrapping for subsequent beats.</td></tr>
<tr><td>4</td><td>Sub-block: Critical dword first, with increment/decrement for subsequent beats</td></tr>
<tr><td>0-1,3,5-7</td><td>Unused by 24Kf core</td></tr>
</table></td></tr>
<tr><td><i>OC_MTagID[2:0]</i></td><td>O</td><td>Transaction tag identifier. The encoding of this field is determined by the BIU buffer holding the outstanding transaction, as shown in the following table. Note: the 24Kf core assumes a non-reordering subset of the OCP Tag semantics. For more explanation, see "OCP Interface Transactions" on page 57.
<table>
<tr><th>Encoding</th><th>Tag Allocation</th></tr>
<tr><td>0</td><td>From Read buffer 0</td></tr>
<tr><td>1</td><td>From Read buffer 1</td></tr>
<tr><td>2</td><td>From Read buffer 2</td></tr>
<tr><td>3</td><td>From Read buffer 3</td></tr>
<tr><td>4</td><td>From Fetch buffer 0</td></tr>
<tr><td>5</td><td>From Fetch buffer 1</td></tr>
<tr><td>6</td><td>SYNC</td></tr>
<tr><td>7</td><td>WR, CACHE-RD, CACHE-WR</td></tr>
</table></td></tr>
<tr><td><i>OC_MBurstPrecise</i></td><td>SO</td><td>Indicates whether the burst length is precise. In the 24Kf core, burst lengths are always fixed at 4 beats, so this pin is statically set to 0x1.</td></tr>
<tr><td><i>OC_MBurstSingleReq</i></td><td>SO</td><td>Indicates whether there is a single request for all data transfers in a burst. In the 24Kf core, there is always a single command request so this pin is statically set to 0x1.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>*OC_MBurstLength[2:0]*</td><td>O</td><td>Number of 64b data transfers. Only two values are possible in the 24Kf core.
<table>
<tr><th>Encoding</th><th>Number of Transfers</th></tr>
<tr><td>1</td><td>1, single transfer</td></tr>
<tr><td>4</td><td>4-beat burst</td></tr>
<tr><td>others</td><td>Unused by 24Kf core</td></tr>
</table></td></tr>
<tr><td>*OC_MByteEn[7:0]*</td><td>O</td><td>Byte enables for reads. Includes data alignment, endianness and address. The correlation of each bit in the *OC_MByteEn* field to the returned read data bytes is shown in the following table:
<table>
<tr><th>*OC_MByteEn* signal</th><th>Requested byte to be returned on *OC_SData* bus</th></tr>
<tr><td>[0]</td><td>[7:0]</td></tr>
<tr><td>[1]</td><td>[15:8]</td></tr>
<tr><td>[2]</td><td>[23:16]</td></tr>
<tr><td>[3]</td><td>[31:24]</td></tr>
<tr><td>[4]</td><td>[39:32]</td></tr>
<tr><td>[5]</td><td>[47:40]</td></tr>
<tr><td>[6]</td><td>[55:48]</td></tr>
<tr><td>[7]</td><td>[63:56]</td></tr>
</table></td></tr>
<tr><td>*OC_MData[63:0]*</td><td>O</td><td>Write data bus from the 24Kf core.</td></tr>
<tr><td>*OC_MDataByteEn[7:0]*</td><td>O</td><td>Byte enables for writes. Includes data alignment, endianness and address. The correlation of each bit in the *OC_MDataByteEn* field to the write data bytes is shown in the following table. Note that the 24Kf core does not use *OC_MByteEn* for transferring byte enables during writes as some other OCP masters do.
<table>
<tr><th>*OC_MDataByteEn* signal</th><th>Valid write data byte on *OC_MData* bus</th></tr>
<tr><td>[0]</td><td>[7:0]</td></tr>
<tr><td>[1]</td><td>[15:8]</td></tr>
<tr><td>[2]</td><td>[23:16]</td></tr>
<tr><td>[3]</td><td>[31:24]</td></tr>
<tr><td>[4]</td><td>[39:32]</td></tr>
<tr><td>[5]</td><td>[47:40]</td></tr>
<tr><td>[6]</td><td>[55:48]</td></tr>
<tr><td>[7]</td><td>[63:56]</td></tr>
</table></td></tr>
<tr><td>*OC_MDataValid*</td><td>O</td><td>Valid write data on *OC_MData* bus.</td></tr>
<tr><td>*OC_MDataTagID[2:0]*</td><td>O</td><td>Write data tag identifier (for out of order returns). Per the encoding for *OC_MTagID*, the only valid value in the 24Kf core is 0x7.</td></tr>
<tr><td>*OC_MDataLast*</td><td>O</td><td>Last valid data in a write burst.</td></tr>
<tr><td>*OC_SData[63:0]*</td><td>I</td><td>Returned read data to core.</td></tr>
<tr><td>*OC_STagID[2:0]*</td><td>I</td><td>Return transaction tag ID. See *OC_MTagID* for encoding.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>*OC_SResp[1:0]*</td><td>I</td><td>Valid response from system controller. The encoding recognized by the 24Kf core are shown in the following table
<table>
<tr><th>Encoding</th><th>Command</th><th>Mnemonic</th><th>Description</th></tr>
<tr><td>0</td><td>No response</td><td>NULL</td><td>No response</td></tr>
<tr><td>1</td><td>Data valid / accept</td><td>DVA</td><td>Normal completion response</td></tr>
<tr><td>2</td><td>Reserved</td><td>-</td><td>Should not be used on 24Kf core</td></tr>
<tr><td>3</td><td>Response error</td><td>ERR</td><td>Signals bus error exception</td></tr>
</table>
</td></tr>
<tr><td>*OC_SRespLast*</td><td>I</td><td>Marks last data in read burst.</td></tr>
<tr><td>*OC_SCmdAccept*</td><td>I</td><td>System controller notifies the 24Kf core that the command is accepted.</td></tr>
<tr><td>*OC_SDataAccept*</td><td>I</td><td>System Controller notifies the 24Kf core that the write data is accepted.</td></tr>
<tr><td>*OC_MTagInOrder*</td><td>SO</td><td>Indicates if the request can be reordered. This pin is statically tied to 0x0.</td></tr>
<tr><td colspan="3">**CorExtend/MDU Interface**: On 24Kf Pro cores, there is an external interface to a combined CorExtend and MDU block. Refer to the *MIPS32™ 24K™ Pro Series™ CorExtend™ Instruction Integrator's Guide* for more details on these signals.</td></tr>
<tr><td colspan="3">**Coprocessor 2 Interface**</td></tr>
<tr><td>*CP2_gfclk*</td><td>O</td><td>Free running clock for coprocessor 2. Same as *SI_ClkIn*</td></tr>
<tr><td>*CP2_gclk*</td><td>O</td><td>Gated coprocessor 2 clock</td></tr>
<tr><td>*CP2_gscanenable*</td><td>O</td><td>Scanenable for coprocessor 2 module. This is same as *gscanenable*.</td></tr>
<tr><td colspan="3">**Dispatch:** These signals are used to transfer an instruction from the 24Kf core to the COP2 coprocessor.</td></tr>
<tr><td>*CP2_ir_0[31:0]*</td><td>O</td><td>Coprocessor Instruction Word.<br>*Valid in the cycle before CP2_as_0, CP2_ts_0 or CP2_fs_0.*</td></tr>
<tr><td>*CP2_irenable_0*</td><td>O</td><td>Enable Instruction Registering.<br>*When deasserted, no instruction strobes will be asserted in the following cycle.*</td></tr>
<tr><td>*CP2_as_0*</td><td>O</td><td>Coprocessor 2 Arithmetic Instruction Strobe.</td></tr>
<tr><td>*CP2_abusy_0*</td><td>I</td><td>Coprocessor 2 Arithmetic Busy.</td></tr>
<tr><td>*CP2_ts_0*</td><td>O</td><td>Coprocessor 2 To Strobe.</td></tr>
<tr><td>*CP2_tbusy_0*</td><td>I</td><td>To Coprocessor 2 Busy.</td></tr>
<tr><td>*CP2_fs_0*</td><td>O</td><td>Coprocessor 2 From Strobe.</td></tr>
<tr><td>*CP2_fbusy_0*</td><td>I</td><td>From Coprocessor 2 Busy.</td></tr>
<tr><td>*CP2_endian_0*</td><td>O</td><td>Big Endian Byte Ordering.<br>*Valid the cycle before CP2_as_0, CP2_fs_0 or CP2_ts_0.*</td></tr>
<tr><td>*CP2_inst32_0*</td><td>SO</td><td>MIPS32 Compatibility Mode - Instructions.<br>*Valid the cycle before CP2_as_0, CP2_fs_0 or CP2_ts_0.*<br>***Tied high in 24K.***</td></tr>
<tr><td>*CP2_kd_mode_0*</td><td>O</td><td>Kernel/Debug mode. When asserted, the processor is in kernel or debug mode.<br>*Valid the cycle before CP2_as_0, CP2_fs_0 or CP2_ts_0 is asserted.*</td></tr>
<tr><td colspan="3">**To COP Data transfer:** These signals are used when data is sent from the 24Kf core to the COP2 coprocessor, as part of completing a To Coprocessor instruction**.**</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| *CP2_tds_0* | O | Coprocessor To Data Strobe. |
| *CP2_torder_0[2:0]* | SO | Coprocessor To Order.<br>No out-of-order data to COP2.<br>***Forced to 000.*** |
| *CP2_tordlim_0[2:0]* | I | To Coprocessor Data Out-of-order Limit.<br>***Being ignored.*** |
| *CP2_tdata_0[63:0]* | O | To Coprocessor Data. |
| **From COP Data transfer:** These signals are used when data is sent to the 24Kf core from the COP2 coprocessor, as part of completing a From Coprocessor instruction. | | |
| *CP2_fds_0* | I | Coprocessor From Data Strobe. |
| *CP2_forder_0[2:0]* | I | Coprocessor From Order. No out-of-order support in 24K.<br>***Expected to be 000 in 24K.*** |
| *CP2_fordlim_0[2:0]* | SO | From Coprocessor Data Out-of-order Limit.<br>No out-of-order data to COP2.<br>***Forced to 000 in 24K.*** |
| *CP2_fdata_0[63:0]* | I | From Coprocessor Data. |
| **COP Condition Code Check:** These signals are used to report the result of a condition code check to the 24Kf core from the COP2 coprocessor. This is only used for BC2 instructions. | | |
| *CP2_cccs_0* | I | Coprocessor Condition Code Check Strobe. |
| *CP2_ccc_0* | I | Coprocessor Condition Code Check.<br>*When asserted, the branch should take the branch.*<br>*When deasserted, the branch should not take the branch.* |
| **Exceptions:** These signals are used by the COP2 coprocessor to report exception for each instruction. | | |
| *CP2_excs_0* | I | Coprocessor Exception Strobe. |
| *CP2_exc_0* | I | Coprocessor Exception. Valid when CP2_excs_0 is asserted. |
| *CP2_exccode_0[4:0]* | I | Coprocessor Exception Code. Valid when both CP2_excs_0 and CP2_exc_0 are asserted.<br>• 01010: RI (This will trigger RI in core pipeline.)<br>• 10000: Available for Coprocessor specific exception<br>• 10010: C2E exception.<br>• All others: Reserved |
| **Nullification:** These signals are used by the 24Kf core to signal nullification of each instruction to the COP2 coprocessor. | | |
| *CP2_nulls_0* | O | Coprocessor Null Strobe. |
| *CP2_null_0* | O | Nullify coprocessor instruction. |
| **Kill:** These signals are used by the 24Kf core to signal killing of each instruction to the COP2 coprocessor. | | |
| *CP2_kills_0* | O | Coprocessor Kill Strobe. |

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| *CP2_kill_0[1:0]* | O | Kill Coprocessor Instruction.<br>• 00 / 01: Instruction is not killed. OK for commit.<br>• 10: Instruction is killed not due to CP2_exc.<br>• 11: Instruction is killed due to CP2_exc. |
| **Miscellaneous COP2 signals:** | | |
| *CP2_reset* | O | Coprocessor Reset. Asserted when a reset is performed by the integer pipeline. |
| *CP2_present* | S | COP2 Present. |
| *CP2_idle* | I | Coprocessor Idle. Asserted when the coprocessor logic is idle.<br>*Enables the processor to go into sleep mode and shut down the clock.* |
| *CP2_perfcnt_event* | I | Implementation coprocessor performance counter event. |
| *CP2_tx32* | S | COP2 32-bit Transfers. When this signal is asserted, the integer unit must cause an RI exception for 64-bit COP2 TF instructions. |
| **Data scratchpad RAM (DSPRAM) Interface:**<br>This set of interface signals allows the data scratchpad RAM array to be accessed independent of the data cache.<br>Note: In order to achieve single cycle access, the ScratchPad interface is not fully registered, unlike most other core interfaces. This requires more careful timing considerations. | | |
| *SP_gfclk* | O | DSPRAM free running clock. This signal follow *SI_ClkIn* |
| *SP_gclk* | O | DSPRAM gated clock. This clock is shutdown when the processor is in sleep mode and top level clock gating is enabled. |
| *SP_greset_pre* | O | This reset signal should be registered within the DSPRAM module before use. The registered version of this signal follows the reset seen by rest of the core logic. |
| *SP_gscanenable* | O | Scanenable signal for DSPRAM module. This signal follows the gscanenable to the core. |
| *SP_parity_en* | O | DSPRAM parity enable |
| *SP_sleep_req_xx* | O | Asserted when entering sleep mode, the clock will be killed in the next cycle |
| *SP_wait_pd_xx* | O | A WAIT instruction pending in the pipeline |
| *SP_tag_rd_ag* | O | DSPRAM Tag Read Strobe. Asserted when a read on DSPRAM tag register (either base address or size register) is performed |
| *SP_tag_wr_ag* | O | DSPRAM Tag Write Strobe. Asserted when a write to DSPRAM tag register (either base address or size register) is performed |
| *SP_tag_sel_ag* | O | DSPRAM Tag read/write selection (0: base address register, 1: size register) |
| *SP_tag_wdata_ag[31:11]* | O | DSPRAM Tag write data. It is valid when SP_tag_wr_ag is asserted |
| *SP_data_addr_ag[19:2]* | O | Address of the SPRAM data access. This is valid during the cycle that SP_data_rd_ag or SP_data_wr_ag is asserted. |
| *SP_data_rd_ag* | O | DSPRAM data read strobe. |
| *SP_dma_rd_ag* | O | Indication that the read to DSPRAM is from a DMA request. |
| *SP_data_wr_ag* | O | DSPRAM data write strobe |
| *SP_data_wren_ag[7:0]* | O | DSPRAM Data Write mask. This is the byte enable for the write data to SPRAM. Only valid when *SP_data_wr_ag* is asserted |

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>SP_data_wdata_ag[63:0]</i></td><td>O</td><td>DSPRAM write data.</td></tr>
<tr><td><i>SP_data_wpar_ag[7:0]</i></td><td>O</td><td>Byte parity bits for write data bus (<i>SP_data_wdata_ag)</i></td></tr>
<tr><td><i>SP_req_id_ag[2:0]</i></td><td>O</td><td>Request identification from the core to the SPRAM. This is used to track loads from DSPRAM. It is valid the cycle <i>SP_data_rd_ag</i> is asserted</td></tr>
<tr><td><i>SP_core_reqpd_er</i></td><td>O</td><td>Asserted when a SPRAM instruction is pending. No DMA access will be issued to DSPRAM. This signal will be ignored when <i>SI_DMA_Priority</i> = 1</td></tr>
<tr><td><i>SP_mbsp_tosp_xx[n-1:0]</i></td><td>O</td><td>User defined variable width DSPRAM BIST sideband signal to DSPRAM.</td></tr>
<tr><td><i>SP_sp_tombsp_xx[n-1:0]</i></td><td>I</td><td>User defined variable width DSPRAM BIST sideband signal from DSPRAM.</td></tr>
<tr><td><i>SP_present</i></td><td>S</td><td>Presence of SPRAM</td></tr>
<tr><td><i>SP_parity_present</i></td><td>I</td><td>Indicates if parity logic is implemented in the DSPRAM module</td></tr>
<tr><td><i>SP_perfcnt_event</i></td><td>I</td><td>Implementation specific SPRAM performance counter event</td></tr>
<tr><td><i>SP_ram_busy</i></td><td>I</td><td>This is used for a non-pipelined multi-cycle SPRAM design. Asserted when the SPRAM will not be able the take any request in the next cycle. Any access to SPRAM will be stalled in the next cycle.</td></tr>
<tr><td><i>SP_busy_xx</i></td><td>I</td><td>Asserted when SPRAM is not idle. This will prevent the processor from entering the sleep mode and disable the clock</td></tr>
<tr><td><i>SP_tag_rdata_xx[31:11]</i></td><td>I</td><td>DSPRAM tag read data</td></tr>
<tr><td><i>SP_tag_msk_xx[31:12]</i></td><td>I</td><td>Address mask for different size SPRAM Tag comparison. When the mask bit is one, the address bit will participate in the tag comparison. When the mask is zero, the address bit will be excluded from tag comparison
<table>
<tr><th>SPRAM size</th><th>SP_tag_msk_xx[31:12]</th></tr>
<tr><td>4KB</td><td>1111_1111_1111_1111_1111</td></tr>
<tr><td>8KB</td><td>1111_1111_1111_1111_1110</td></tr>
<tr><td>16KB</td><td>1111_1111_1111_1111_1100</td></tr>
<tr><td>32KB</td><td>1111_1111_1111_1111_1000</td></tr>
<tr><td>64KB</td><td>1111_1111_1111_1111_0000</td></tr>
<tr><td>128KB</td><td>1111_1111_1111_1110_0000</td></tr>
<tr><td>256KB</td><td>1111_1111_1111_1100_0000</td></tr>
<tr><td>512KB</td><td>1111_1111_1111_1000_0000</td></tr>
<tr><td>1MB</td><td>1111_1111_1111_0000_0000</td></tr>
</table></td></tr>
<tr><td><i>SP_data_rdata_xx[63:0]</i></td><td>I</td><td>Data return form SPRAM read. It is valid the same cycle <i>SP_datavld_xx</i> is asserted. For single cycle access, read data should be returned the cycle after <i>SP_data_rd_ag</i> is asserted.</td></tr>
<tr><td><i>SP_data_rpar_xx[7:0]</i></td><td>I</td><td>Byte parity of SPRAM read data (<i>SP_data_rdata_xx)</i>. It is valid the same cycle <i>SP_datavld_xx</i> is asserted. For a SPRAM design where no parity is implemented, this signal is ignored.</td></tr>
<tr><td><i>SP_datavld_nxt_xx</i></td><td>I</td><td>Indicates that is a valid return data from SPRAM for a read.</td></tr>
<tr><td><i>SP_data_id_xx[2:0]</i></td><td>I</td><td>Instruction identification associated with the data returned. This is valid the same cycle SP_datavld_xx is asserted.</td></tr>
<tr><td><i>SP_dma_id_xx[2:0]</i></td><td>I</td><td>OCP TagID of a DSPRAM DMA access. It is valid the same cycle <i>SP_dma_wr_ag</i> or <i>SP_dma_rd_ag</i> is asserted.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>SP_dma_addr_xx[19:3]</i></td><td>I</td><td>Index of a DSPRAM DMA request. It is valid the same cycle <i>SP_dma_rd_ag</i> or <i>SP_dma_wr_ag</i> is asserted.</td></tr>
<tr><td><i>SP_dma_wdata_xx[63:0]</i></td><td>I</td><td>Write data for a DMA write request. It is valid the same cycle <i>SP_dma_wr_ag</i> is asserted.</td></tr>
<tr><td><i>SP_dma_wren_xx[7:0]</i></td><td>I</td><td>Byte write enable for a DMA write request.</td></tr>
<tr><td><i>SP_dma_rd_xx</i></td><td>I</td><td>DSPRAM DMA read strobe</td></tr>
<tr><td><i>SP_dma_wr_xx</i></td><td>I</td><td>DSPRAM DMA write strobe</td></tr>
<tr><td><i>SP_dma_stallreq_xx</i></td><td>I</td><td>Stall request from scratch pad to processor when there is a DMA access and <i>SI_DMA_Priority</i> = 1</td></tr>
<tr><td colspan="3"><b>DSPRAM External Interface (OCP Slave Interface)</b> This set of interface signals allows a DMA device to access the optional data scratchpad RAM.</td></tr>
<tr><td><i>OC_DMA_MCmd[2:0]</i></td><td>I</td><td>OCP command bus indicating the type of transaction requested. The following are the encoding and the transaction type supported by the slave.
<table>
<tr><th>Encoding</th><th>Command</th><th>Mnemonic</th><th>Description</th></tr>
<tr><td>0</td><td>Idle</td><td>IDLE</td><td>No transaction</td></tr>
<tr><td>1</td><td>Write</td><td>WR</td><td>Used for data write</td></tr>
<tr><td>2</td><td>Read</td><td>RD</td><td>Used for data read</td></tr>
<tr><td>3-7</td><td>Unused</td><td>-</td><td>Not used on 24Kf core</td></tr>
</table></td></tr>
<tr><td><i>OC_DMA_MTagID[2:0]</i></td><td>I</td><td>Transaction tag identifier.</td></tr>
<tr><td><i>OC_DMA_MAddr[31:3]</i></td><td>I</td><td>Physical doubleword address bus. This address should fall in the address range programmed in the DSPRAM module.</td></tr>
<tr><td><i>OC_DMA_MByteEn[7:0]</i></td><td>I</td><td>Byte enable for reads operation. In combination with the endianness this bus determines the byte addressed for the read operation. The correlation of each bit in the <i>OC_DMA_MByteEn</i> field to the returned read data bytes is shown in the following table:
<table>
<tr><th><i>OC_DMA_MByteEn</i> signal</th><th>Requested byte to be returned on <i>OC_DMA_SData</i> bus</th></tr>
<tr><td>[0]</td><td>[7:0]</td></tr>
<tr><td>[1]</td><td>[15:8]</td></tr>
<tr><td>[2]</td><td>[23:16]</td></tr>
<tr><td>[3]</td><td>[31:24]</td></tr>
<tr><td>[4]</td><td>[39:32]</td></tr>
<tr><td>[5]</td><td>[47:40]</td></tr>
<tr><td>[6]</td><td>[55:48]</td></tr>
<tr><td>[7]</td><td>[63:56]</td></tr>
</table></td></tr>
<tr><td><i>OC_DMA_MDataTagID[2:0]</i></td><td>I</td><td>Write data tag identifier</td></tr>
<tr><td><i>OC_DMA_MData[63:0]</i></td><td>I</td><td>Write data bus to the data scratchpad RAM</td></tr>
<tr><td><i>OC_DMA_MDataByteEn[7:0]</i></td><td>I</td><td>Byte lane selection for write data</td></tr>
<tr><td><i>OC_DMA_MDataValid</i></td><td>I</td><td>Write data (<i>OC_DMA_MData</i>) valid indication</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| *OC_DMA_SResp[1:0]* | O | Read/Write response (00=NULL, 01=VALID 11=ERR) |
| *OC_DMA_STagID[2:0]* | O | Return transaction tag ID. |
| *OC_DMA_SData[63:0]* | O | Return data read from the scratchpad RAM. |
| *OC_DMA_SCmdAccept* | O | Flow control for commands (0=BUSY, 1=READY)<br>• The core deasserts this signal when it is not ready to accept a new command. The master has to hold the new command till it is accepted by the core.<br>• The core asserts this signal when it has accepted a new command |
| *OC_DMA_SDataAccept* | O | Flow control for data (0=BUSY, 1=READY)<br>• The core deasserts this signal when it is not ready to accept a the data driven in *OC_DMA_SData*. The master has to hold the data till it is accepted by the core.<br>• The core asserts this signal when it has accepted the data driven in *OC_DMA_SData* |
| **Instruction scratchpad RAM (ISPRAM) Interface:**<br>This set of interface signals allows the instruction scratchpad RAM array to be accessed independent of the instruction cache.<br>Note: In order to achieve single cycle access, the ScratchPad interface is not fully registered, unlike most other core interfaces. This requires more careful timing considerations | | |
| *ISP_gclk* | O | Gated global clock |
| *ISP_gfclk* | O | Free-running global clock |
| *ISP_greset_pre* | O | Global reset. Must be registered prior to use |
| *ISP_gscanenable* | O | Global scanenable. Use to override local clock gating during scan. |
| *ISP_parity_en* | O | ISPRAM parity enable - used to control whether parity errors on DMA reads are reported or not. |
| *ISP_wait_pd_xx* | O | WAIT instruction has been executed and core is getting ready to go to sleep |
| *ISP_sleep_req_xx* | O | Entering sleep mode in the next cycle |
| *ISP_core_reqpd_xx* | O | There is a core access pending (stalling the DMA request) |
| *ISP_addr_ipf[19:3]* | O | Index for access. |
| *ISP_tag_sel_ipf* | O | Controls whether Size or base address register is written |
| *ISP_rd_ipf* | O | Read Strobe - both tag and data values are read |
| *ISP_dma_rd_ipf* | O | The read is a for a DMA access |
| *ISP_tag_wr_ipf* | O | Tag Write Strobe |
| *ISP_tag_wdata_ipf[31:11]* | O | Tag write data. |
| *ISP_data_wr_ipf* | O | Data Write Strobe |
| *ISP_data_wdata_ipf[69:0]* | O | Data write data (instructions + precode) |
| *ISP_data_wpar_ipf[8:0]* | O | Parity for write data |
| *ISP_present* | I | Presence of ISPRAM |
| *ISP_parity_present* | I | ISPRAM array has parity support |

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>ISP_perfcnt_event</i></td><td>I</td><td>Implementation specific ISPRAM performance counter event</td></tr>
<tr><td><i>ISP_ram_busy_</i></td><td>I</td><td>ISPRAM will not accept any request in the next cycle</td></tr>
<tr><td><i>ISP_busy_xx</i></td><td>I</td><td>ISPRAM is busy with request - for sleep mode.</td></tr>
<tr><td><i>ISP_tag_rdata_if[31:11]</i></td><td>I</td><td>Base address for SPRAM region.</td></tr>
<tr><td><i>ISP_tag_msk_if[19:12]</i></td><td>I</td><td>Mask for address comparison</td></tr>
<tr><td><i>ISP_data_rdata_is[69:0]</i></td><td>I</td><td>Read data from data port</td></tr>
<tr><td><i>ISP_data_rpar_is[8:0]</i></td><td>I</td><td>Parity for read data</td></tr>
<tr><td><i>ISP_datavld_nxt_if</i></td><td>I</td><td>Read data will be valid next cycle</td></tr>
<tr><td><i>ISP_dma_addr_xx[19:3]</i></td><td>I</td><td>Index of DMA access</td></tr>
<tr><td><i>ISP_dma_wdata_xx[63:0]</i></td><td>I</td><td>DMA write data</td></tr>
<tr><td><i>ISP_dma_rdreq_xx</i></td><td>I</td><td>DMA read request</td></tr>
<tr><td><i>ISP_dma_wrreq_xx</i></td><td>I</td><td>DMA write request</td></tr>
<tr><td><i>ISP_dma_stallreq_xx</i></td><td>I</td><td>DMA stall request to the core (when DMA has higher priority)</td></tr>
<tr><td colspan="3"><b>ISPRAM External Interface (OCP Slave Interface)</b> This set of interface signals allows a DMA device to access the optional instruction scratchpad RAM.</td></tr>
<tr><td><i>OC_IDMA_MCmd[2:0]</i></td><td>I</td><td>OCP command bus indicating the type of transaction requested. The following are the encoding and the transaction type supported by the slave.
<table>
<tr><th>Encoding</th><th>Command</th><th>Mnemonic</th><th>Description</th></tr>
<tr><td>0</td><td>Idle</td><td>IDLE</td><td>No transaction</td></tr>
<tr><td>1</td><td>Write</td><td>WR</td><td>Used for data write</td></tr>
<tr><td>2</td><td>Read</td><td>RD</td><td>Used for data read</td></tr>
<tr><td>3-7</td><td>Unused</td><td>-</td><td>Not used on 24Kf core</td></tr>
</table></td></tr>
<tr><td><i>OC_IDMA_MTagID[2:0]</i></td><td>I</td><td>Transaction tag identifier.</td></tr>
<tr><td><i>OC_IDMA_MAddr[31:3]</i></td><td>I</td><td>Physical doubleword address bus. This address should fall in the address range programmed in the ISPRAM module.</td></tr>
<tr><td><i>OC_IDMA_MDataValid</i></td><td>I</td><td>Write data (<i>OC_DMA_MData</i>) valid indication</td></tr>
<tr><td><i>OC_IDMA_MDataTagID[2:0]</i></td><td>I</td><td>Write data tag identifier</td></tr>
<tr><td><i>OC_IDMA_MData[63:0]</i></td><td>I</td><td>Write data bus to the instruction scratchpad RAM</td></tr>
<tr><td><i>OC_IDMA_SResp[1:0]</i></td><td>O</td><td>Read/Write response (00=NULL, 01=VALID 11=ERR)</td></tr>
<tr><td><i>OC_IDMA_STagID[2:0]</i></td><td>O</td><td>Return transaction tag ID.</td></tr>
<tr><td><i>OC_IDMA_SData[63:0]</i></td><td>O</td><td>Return data read from the scratchpad RAM.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| *OC_IDMA_SCmdAccept* | O | Flow control for commands (0=BUSY, 1=READY)<br>• The core deasserts this signal when it is not ready to accept a new command. The master has to hold the new command till it is accepted by the core.<br>• The core asserts this signal when it has accepted a new command |
| *OC_IDMA_SDataAccept* | O | Flow control for data (0=BUSY, 1=READY)<br>• The core deasserts this signal when it is not ready to accept a the data driven in *OC_IDMA_SData*. The master has to hold the data till it is accepted by the core.<br>• The core asserts this signal when it has accepted the data driven in *OC_IDMA_SData* |
| **EJTAG Interface** | | |
| TAP interface. These signals comprise the EJTAG Test Access Port. | | |
| *EJ_TRST_N* | I | Active-low Test Reset Input (TRST*) for the EJTAG TAP. At power-up, the assertion of *EJ_TRST_N* causes the TAP controller to be reset. |
| *EJ_TCK* | I | Test Clock Input (TCK) for the EJTAG TAP. |
| *EJ_TMS* | I | Test Mode Select Input (TMS) for the EJTAG TAP. |
| *EJ_TDI* | I | Test Data Input (TDI) for the EJTAG TAP. |
| *EJ_TDO* | O | Test Data Output (TDO) for the EJTAG TAP. |
| *EJ_TDOzstate* | O | Drive indication for the output of TDO for the EJTAG TAP at chip level:<br>1: The TDO output at chip level must be in Z-state<br>0: The TDO output at chip level must be driven to the value of *EJ_TDO*<br>IEEE Standard 1149.1-1990 defines TDO as a 3-stated signal. To avoid having a 3-state core output, the 24Kf core outputs this signal to drive an external 3-state buffer. |
| *Debug Interrupt:* | | |
| *EJ_DINTsup* | S | Value of DINTsup for the Implementation register. When high, this signal indicates that the EJTAG probe can use the DINT signal to interrupt the processor. |
| *EJ_DINT* | A | Debug exception request when this signal is asserted one clock period after being deasserted in the previous clock period. The request is cleared when debug mode is entered. Requests when in debug mode are ignored. |
| *Debug Mode Indication:* | | |
| *EJ_DebugM* | O | Asserted when the core is in Debug Mode. This can be used to bring the core out of a low power mode. In systems with multiple processor cores, this signal can be used to synchronize the cores when debugging. |
| *Device ID bits:* | | |
| These inputs provide an identifying number visible to the EJTAG probe.These inputs are always available for soft core customers. On hard cores, the core “hardener” can set these inputs to their own values. | | |

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><em>EJ_ManufID[10:0]</em></td><td>S</td><td>Value of the ManufID[10:0] field in the Device ID register. As per IEEE 1149.1-1990 section 11.2, the manufacturer identity code shall be a compressed form of JEDEC standard manufacturer’s identification code in the JEDEC Publications 106, which can be found at: <code>http://www.jedec.org/</code><br>ManufID[6:0] bits are derived from the last byte of the JEDEC code by discarding the parity bit. ManufID[10:7] bits provide a binary count of the number of bytes in the JEDEC code that contain the continuation character (0x7F). Where the number of continuations characters exceeds 15, these 4 bits contain the modulo-16 count of the number of continuation characters.</td></tr>
<tr><td><em>EJ_PartNumber[15:0]</em></td><td>S</td><td>Value of the PartNumber[15:0] field in the Device ID register.</td></tr>
<tr><td><em>EJ_Version[3:0]</em></td><td>S</td><td>Value of the Version[3:0] field in the Device ID register.</td></tr>
<tr><td colspan="3"><em>System Implementation Dependent Outputs:</em></td></tr>
<tr><td colspan="3">These signals come from EJTAG control registers. They have no effect on the core, but can be used to give EJTAG debugging software additional control over the system.</td></tr>
<tr><td><em>EJ_SRstE</em></td><td>O</td><td>Soft Reset Enable. EJTAG can deassert this signal if it wants to mask soft resets. If this signal is deasserted, none, some, or all soft reset sources are masked.</td></tr>
<tr><td><em>EJ_PerRst</em></td><td>O</td><td>Peripheral Reset. EJTAG can assert this signal to request the reset of some or all of the peripheral devices in the system.</td></tr>
<tr><td><em>EJ_PrRst</em></td><td>O</td><td>Processor Reset. EJTAG can assert this signal to request that the core be reset. This can be fed into the <em>SI_Reset</em> signal.</td></tr>
<tr><td colspan="3"><em>TCtrace Interface</em></td></tr>
<tr><td colspan="3">These signals enable an interface to optional off-chip trace memory. The TCtrace interface connects to the Probe Interface Block (PIB) which in turn connects to the physical off-chip trace pins.<br>Note that if on-chip trace memory is used, access occurs via the EJTAG TAP interface, and use of this interface is not required.</td></tr>
<tr><td><em>TC_ClockRatio[2:0]</em></td><td>O</td><td>Clock ratio. This is the clock ratio set by software in <em>TCBCONTROLB.CR</em>. The value will be within the boundaries defined by <em>TC_CRMax</em> and <em>TC_CRMin</em>. The table below shows the encoded values for clock ratio.
<table>
<tr><th>TC_ClockRatio</th><th>Clock Ratio</th></tr>
<tr><td>000</td><td>8:1 (Trace clock is eight times the core clock)</td></tr>
<tr><td>001</td><td>4:1 (Trace clock is four times the core clock)</td></tr>
<tr><td>010</td><td>2:1 (Trace clock is double the core clock)</td></tr>
<tr><td>011</td><td>1:1 (Trace clock is same as the core clock)</td></tr>
<tr><td>100</td><td>1:2 (Trace clock is one half the core clock)</td></tr>
<tr><td>101</td><td>1:4 (Trace clock is one fourth the core clock)</td></tr>
<tr><td>110</td><td>1:6 (Trace clock is one sixth the core clock)</td></tr>
<tr><td>111</td><td>1:8 (Trace clock is one eight the core clock)</td></tr>
</table></td></tr>
<tr><td><em>TC_CRMax[2:0]</em></td><td>S</td><td>Maximum clock ratio supported. This static input sets the CRMax field of the <em>TCBCONFIG</em> register. It defines the capabilities of the Probe Interface Block (PIB) module.This field determines the minimum value of <em>TC_ClockRatio</em>.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>*TC_CRMin[2:0]*</td><td>S</td><td>Minimum clock ratio supported. This input sets the CRMin field of the *TCBCONFIG* register. It defines the capabilities of the PIB module. This field determines the maximum value of *TC_ClockRatio*.</td></tr>
<tr><td>*TC_ProbeWidth[1:0]*</td><td>S</td><td>This static input will set the PW field of the *TCBCONFIG* register.<br>If this interface is not driving a PIB module, but some chip-level TCB-like module, then this field should be set to 2’b11 (reserved value for PW).
<table>
<tr><th>TC_ProbeWidth</th><th>Number physical data pin on PIB</th></tr>
<tr><td>00</td><td>4 bits</td></tr>
<tr><td>01</td><td>8 bits</td></tr>
<tr><td>10</td><td>16 bits</td></tr>
<tr><td>11</td><td>Not directly to PIB</td></tr>
</table>
</td></tr>
<tr><td>*TC_PibPresent*</td><td>S</td><td>Must be asserted when a PIB is attached to the TC Interface. When de-asserted (low) all the other inputs are disregarded.</td></tr>
<tr><td>*TC_TrEnable*</td><td>O</td><td>Trace Enable, when asserted the PIB must start running its output clock and can expect valid data on all other outputs.</td></tr>
<tr><td>*TC_Calibrate*</td><td>O</td><td>This signal is asserted when the Cal bit in the *TCBCONTROLB* register is set.<br>For a simple PIB which only serves one TCB, this pin can be ignored. For a multi-core capable PIB which also uses *TC_Valid* and *TC_Stall*, the PIB must start producing the calibration pattern when this signal is asserted.</td></tr>
<tr><td>*TC_DataBits[2:0]*</td><td>I</td><td>This input identifies the number of bits picked up by the probe interface module in each “cycle”.<br>If *TC_ClockRatio* indicates a clock-ratio higher than 1:2, then clock multiplication in the Probe logic is used. The “cycle” is equal to each core clock cycle.<br>If *TC_ClockRatio* indicates a clock-ratio lower than or equal to 1:2, then “cycle” is (clock-ratio * 2) of the core clock cycle. For example, with a clock ratio of 1:2, a “cycle” is equal to core clock cycle; with a clock ratio of 1:4, a “cycle” is equal to one half of core clock cycle.<br>This input controls the down-shifting amount and frequency of the trace word on *TC_Data[63:0]*. The bit width and the corresponding *TC_DataBits* value is shown in the table below.
<table>
<tr><th>*TC_DataBits[2:0]*</th><th>Probe uses following bits from *TC_Data* each cycle</th></tr>
<tr><td>000</td><td>*TC_Data[3:0]*</td></tr>
<tr><td>001</td><td>*TC_Data[7:0]*</td></tr>
<tr><td>010</td><td>*TC_Data[15:0]*</td></tr>
<tr><td>011</td><td>*TC_Data[31:0]*</td></tr>
<tr><td>100</td><td>*TC_Data[63:0]*</td></tr>
<tr><td>Others</td><td>Unused</td></tr>
</table>
This input might change as the value on *TC_ClockRatio[2:0]* changes.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>TC_Valid</i></td><td>O</td><td>Asserted when a valid new trace word is started on the <i>TC_Data[63:0]</i> signals.<br><i>TC_Valid</i> is only asserted when <i>TC_DataBits</i> is 100.</td></tr>
<tr><td><i>TC_Stall</i></td><td>I</td><td>When asserted, a new <i>TC_Valid</i> in the following cycle is stalled. <i>TC_Valid</i> is still asserted, but the <i>TC_Data</i> value and <i>TC_Valid</i> are held static, until the cycle after <i>TC_Stall</i> is sampled low.<br><i>TC_Stall</i> is only sampled in the cycle before a new <i>TC_Valid</i> cycle, and only when <i>TC_DataBits</i> is 100, indicating a full word of <i>TC_Data</i>.</td></tr>
<tr><td><i>TC_Data[63:0]</i></td><td>O</td><td>Trace word data. The value on this 64-bit interface is shifted down as indicated in <i>TC_DataBits[2:0]</i>. In the first cycle where a new trace word is valid on all the bits and <i>TC_DataBits[2:0]</i> is 100, <i>TC_Valid</i> is also asserted.<br>The Probe Interface Block (PIB) will only be connected to [(N-1):0] bits of this output bus. N is the number of bits picked up by the PIB in each core clock cycle. For clock ratios 1:2 and lower, N is equal to the number of physical trace pins (legal values of N are 4, 8, or 16). For higher clock ratios, N is larger than the number of physical trace pins.</td></tr>
<tr><td><i>TC_ProbeTrigIn</i></td><td>A</td><td>Rising edge trigger input. The source should be the Probe Trigger input. The input is considered asynchronous; i.e., it is double registered in the core.</td></tr>
<tr><td><i>TC_ProbeTrigOut</i></td><td>O</td><td>Single cycle (relative to the “cycle” defined the description of <i>TC_DataBits</i>) high strobe, trigger output. The target of this trigger is intended to be the external probe’s trigger output.</td></tr>
<tr><td><i>TC_ChipTrigIn</i></td><td>A</td><td>Rising edge trigger input. The source should be on-chip. The input is considered asynchronous; i.e., it is double registered in the core.</td></tr>
<tr><td><i>TC_ChipTrigOut</i></td><td>O</td><td>Single cycle (relative to core clock) high strobe, trigger output. The target of this trigger is intended to be an on-chip unit.</td></tr>
<tr><td colspan="3"><b>Memory BIST Interface</b></td></tr>
<tr><td colspan="3">These signals provide the interface to optional integrated or user-specified memory BIST capability for testing the SRAM arrays within the core.</td></tr>
<tr><td><i>MB_invoke</i></td><td>I</td><td>Enable signal for integrated BIST controllers.</td></tr>
<tr><td><i>MB_ic_algorithm[7:0]</i></td><td>S</td><td>Algorithm selection for I-Cache BIST controllers. For a core configured with IFA-13 BIST support for the I-Cache, bit0 is used to select the BIST algorithm:
<table>
<tr><th><i>MB_ic_algorithm[0]</i></th><th>I-Cache BIST Algorithm</th></tr>
<tr><td>0</td><td>March-C+</td></tr>
<tr><td>1</td><td>IFA-13</td></tr>
</table>
If the IFA-13 algorithm is selected, then <i>MB_ic_algorithm[5:1]</i> is used to determine the retention delay.</td></tr>
<tr><td><i>MB_dc_algorithm[7:0]</i></td><td>S</td><td>Algorithm selection for D-Cache BIST controllers. For a core configured with IFA-13 BIST support for the D-Cache, bit0 is used to select the BIST algorithm:
<table>
<tr><th><i>MB_dc_algorithm[0]</i></th><th>D-Cache BIST Algorithm</th></tr>
<tr><td>0</td><td>March-C+</td></tr>
<tr><td>1</td><td>IFA-13</td></tr>
</table>
If the IFA-13 algorithm is selected, then <i>MB_dc_algorithm[5:1]</i> is used to determine the retention delay.</td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td><i>MB_sp_algorithm[7:0]</i></td><td>S</td><td>Algorithm selection for data scratchpad (DSPRAM) BIST controllers. For a core configured with IFA-13 BIST support for the data scratchpad RAM, bit0 is used to select the BIST algorithm:
<table>
<tr><th><i>MB_sp_algorithm[0]</i></th><th>DSPRAM BIST Algorithm</th></tr>
<tr><td>0</td><td>March-C+</td></tr>
<tr><td>1</td><td>IFA-13</td></tr>
</table>
If the IFA-13 algorithm is selected, then <i>MB_sp_algorithm[5:1]</i> is used to determine the retention delay.</td></tr>
<tr><td><i>MB_isp_algorithm[7:0]</i></td><td>S</td><td>Algorithm selection for instruction scratchpad (ISPRAM) BIST controllers. For a core configured with IFA-13 BIST support for the data scratchpad RAM, bit0 is used to select the BIST algorithm:
<table>
<tr><th><i>MB_isp_algorithm[0]</i></th><th>DSPRAM BIST Algorithm</th></tr>
<tr><td>0</td><td>March-C+</td></tr>
<tr><td>1</td><td>IFA-13</td></tr>
</table>
If the IFA-13 algorithm is selected, then <i>MB_isp_algorithm[5:1]</i> is used to determine the retention delay.</td></tr>
<tr><td><i>MB_tr_algorithm[7:0]</i></td><td>S</td><td>Algorithm selection for trace memory BIST controllers. For a core configured with IFA-13 BIST support for the trace memory, bit0 is used to select the BIST algorithm:
<table>
<tr><th><i>MB_tr_algorithm[0]</i></th><th>Trace mem BIST Algorithm</th></tr>
<tr><td>0</td><td>March-C+</td></tr>
<tr><td>1</td><td>IFA-13</td></tr>
</table>
If the IFA-13 algorithm is selected, then <i>MB_tr_algorithm</i>[5:1] is used to determine the retention delay.</td></tr>
<tr><td><i>MB_done</i></td><td>O</td><td>Common completion indicator for all integrated BIST sequences.</td></tr>
<tr><td><i>MB_dd_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the data cache data array.</td></tr>
<tr><td><i>MB_dt_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the data cache tag array.</td></tr>
<tr><td><i>MB_dw_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the data cache way select array.</td></tr>
<tr><td><i>MB_id_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the instruction cache data array.</td></tr>
<tr><td><i>MB_it_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the instruction cache tag array.</td></tr>
<tr><td><i>MB_iw_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the instruction cache way select array.</td></tr>
<tr><td><i>MB_sp_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the data SPRAM array.</td></tr>
<tr><td><i>MB_isp_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the instruction SPRAM array.</td></tr>
<tr><td><i>MB_tr_fail</i></td><td>O</td><td>When high, indicates that the BIST test failed on the trace memory array.</td></tr>
<tr><td><i>MB_tombt[n-1:0]</i></td><td>I</td><td>Variable width input bus available for user-specified BIST applications.</td></tr>
<tr><td><i>MB_frommbt[n-1:0]</i></td><td>O</td><td>Variable width output bus available for user-specified BIST applications.</td></tr>
<tr><td colspan="3"><b>Scan Test Interface</b></td></tr>
</table>

Table 19 24Kf™ Core Signal Descriptions (Continued)

| Signal Name | Type | Description |
|---|---|---|
| These signals provide an interface for testing the core. The use and configuration of these pins are implementation-dependent. | | |
| *gscanenable* | I | This signal should be asserted while scanning vectors into or out of the core. The *gscanenable* signal must be deasserted during normal operation and during capture clocks in test mode. |
| *gscanmode* | I | This signal should be asserted during all scan testing both while scanning and during capture clocks. The *gscanmode* signal must be deasserted during normal operation. |
| *gscanramwr* | I | This signal controls the read and write strobes to the cache SRAM when *gscanmode* is asserted. |
| *gscanin[n-1:0]* | I | These signal(s) are the inputs to the scan chain(s). |
| *gscanout[n-1:0]* | O | These signal(s) are the outputs from the scan chain(s). |

## OCP Interface Transactions

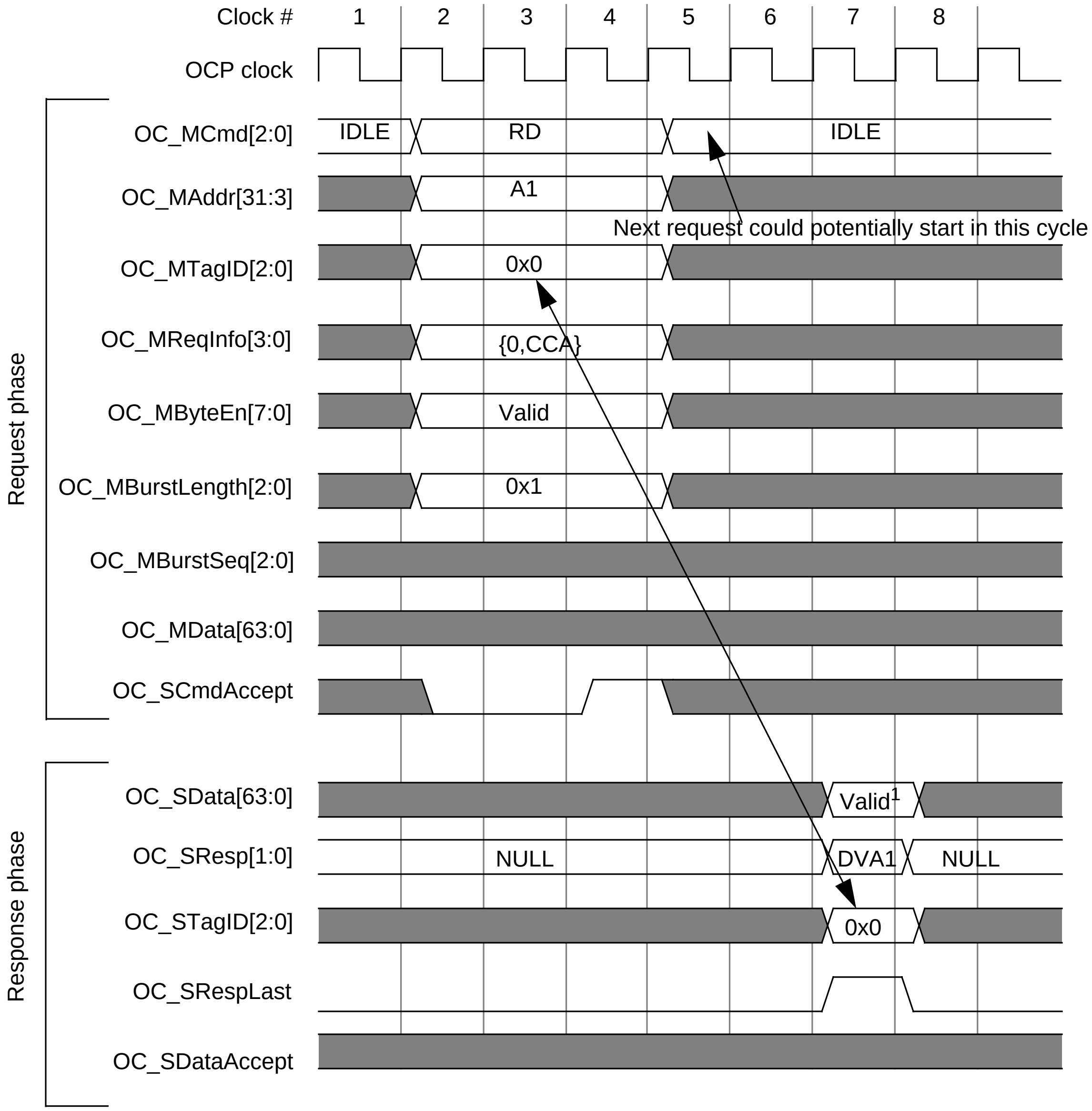
The following sections show timing diagrams for various OCP transactions.

The 24Kf core assumes that any agent or interconnect between it and the target (including the target) can re-order the transactions only if they take on the responsibility of resolving hazards/dependencies.

In a straightforward implementation of an OCP interconnect, the interconnect is expected not to re-order the transactions, but maintain the tagged interface all the way to the target. This essentially gives the transactions a tagging semantic so that Out Of Order (OOO) transaction return is supported. Note that the target is still responsible for checking hazards if it returns responses OOO.

### Single Read

Figure 13 shows a single read transaction, as would occur on an uncached fetch or load. The 24Kf core starts a request phase on clock 2 by switching the MCmd field from IDLE to RD. Simultaneously, it presents valid values on the address (*OC_MAddr*), tag (*OC_MTagID*), transaction info (*OC_MReqInfo*), byte enables (*OC_MByteEn*) and burst length (*OC_MBurstLength*). The slave is shown to flow control the master for one clock then accept the request by asserting *OC_SCmdAccept* in cycle 4, ending the request phase. The slave responds to this request in cycle 7 with a DVA on *OC_SResp*, valid data on *OC_SData*, the return tag ID on *OC_STagID* and last burst indication on *OC_SRespLast*. The request phase for a new transaction by the 24Kf core can potentially start in cycle 5.

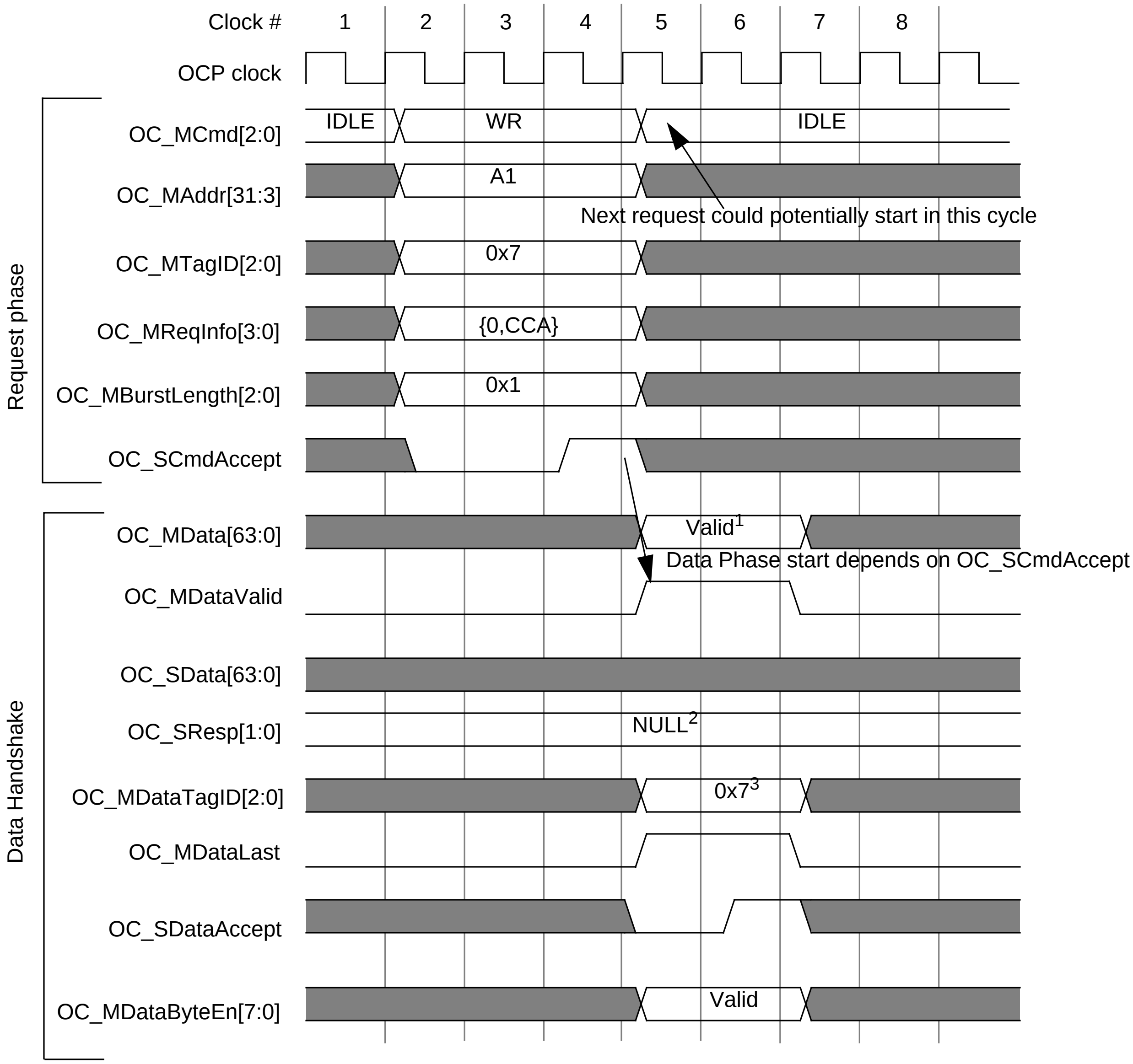


[1] Only the relevant bytes of OC_SData[63:0] carry valid data corresponding to the request signal OC_MByteEn[7:0] at the time of request.

Figure 13 Single OCP Read Transaction with flow control

## Single Write

Figure 14 shows a single write transaction which could be generated from an uncached store, or a write-through or uncached accelerated store that does not merge. The 24Kf core starts a request phase on clock 2 by switching the MCmd field from IDLE to WR. At the same time it presents valid values on the address (*OC_MAddr*), tag (*OC_MTagID*), transaction info (*OC_MReqInfo*), and burst length (*OC_MBurstLength*). The data part of the transaction starts when the 24Kf core asserts the *OC_MDataValid* in cycle 5 along with the data on *OC_MData and byte enables on OC_MDataByteEn*. The slave is shown to flow control the data phase (as is the request phase) by deasserting *OC_SDataAccept* for one cycle before accepting the transaction in cycle 6. A fixed value of 0x7 is used as a TagID for all writes.

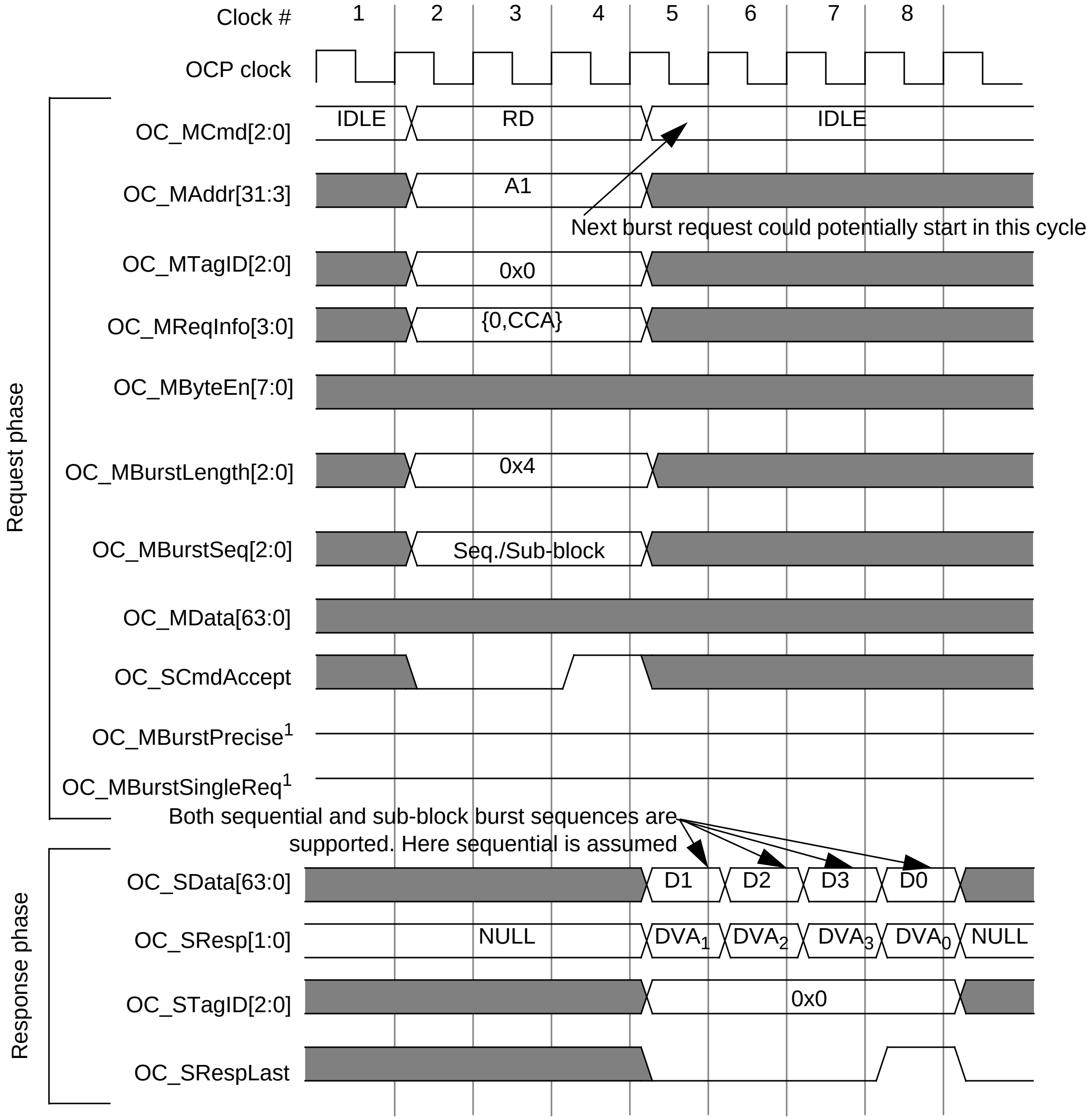


[1] Only the relevant bytes of OC_MData[63:0] carry valid data corresponding to the request signal OC_MDataByteEn[7:0] at the time of request.
[2] 24Kf core does not expect a response to a posted write. (Some OCP systems generate a response to posted writes in the same cycle as SCmdAccept).
[3] The core always forces the OC_MDataTagID[2:0] signal to be 0x7 for writes.

Figure 14 Single OCP Write Transaction with flow control

### Bursted Read

In Figure 15, a bursted read is shown. This is done on a cacheable load or fetch miss to refill the cache line. The core may be configured for either sequential or sub-block burst order. The transaction looks similar to the single read case except that a burst of four 64b data chunks are transferred in the burst order specified.

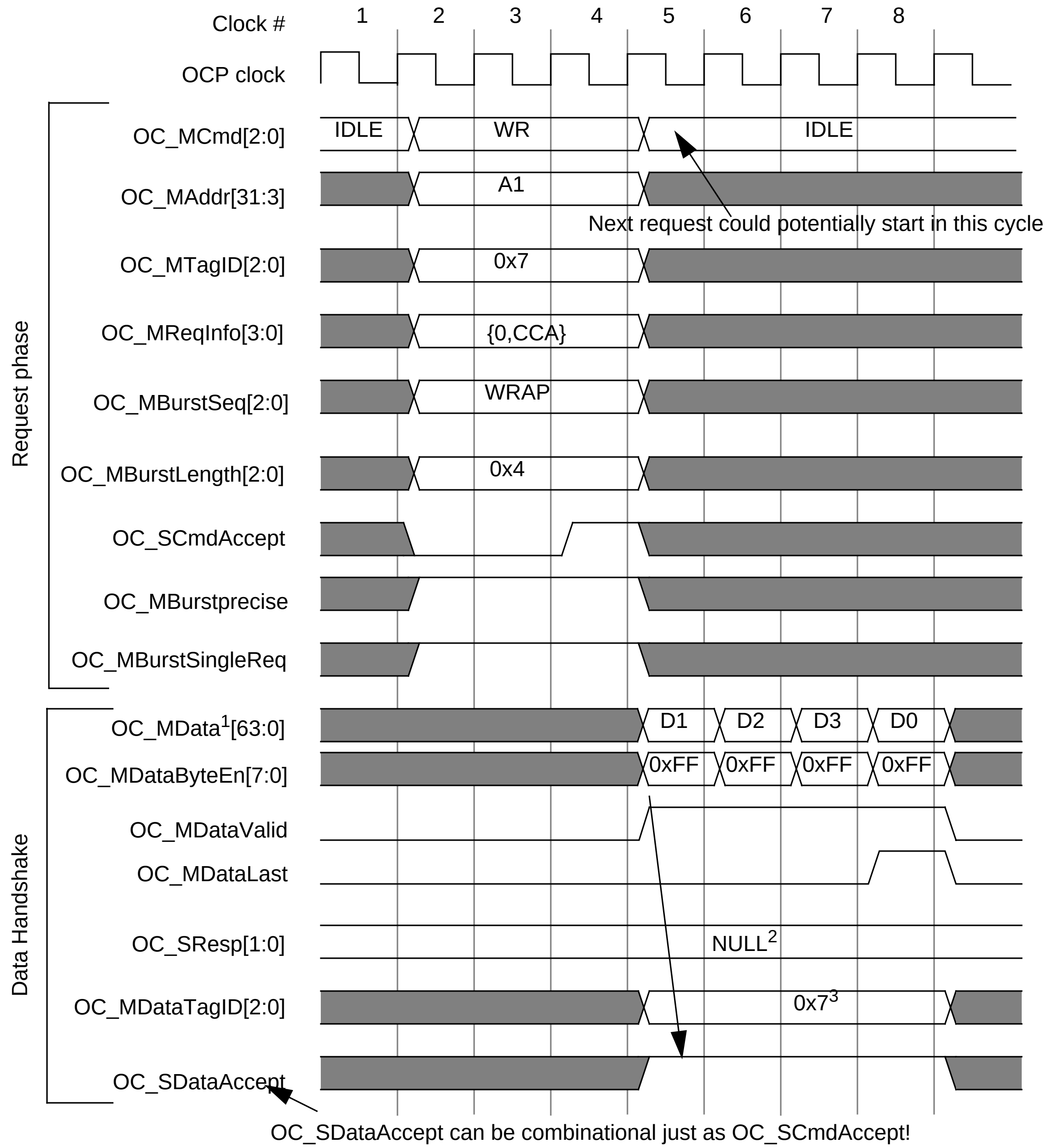


[1] OC_MBurstPrecise and OC_MBurstSingleReq are always forced to 0x1 by the 24Kf core since all bursts are precise and only have a single request cycle even if a burst of data is requested.

Figure 15 Burst OCP Read Transaction with flow control

### Bursted Write

Figure 16 depicts a bursted write transaction. This would typically be seen on a dirty cache line write-back, or when uncached accelerated or write-through stores gather an entire line. Bursted writes always begin at the lowest address of the line.The transaction looks similar to the single write case except that a burst of four 64b data chunks are transferred. Note that the flow control signal *OC_SDataAccept* can be asserted combinationally in cycle 5 as shown.



[1] The 24Kf core will only generate an aligned WRAP burst for burst writes, starting at burst address 0x0.
[2] Core does not expect a response to a posted burst write. (Some OCP systems generate a response to posted writes at the end of the burst, for example an ERR).
[3]Core always forces the OC_MDataTagID[2:0] signal to be 0x7 for writes.

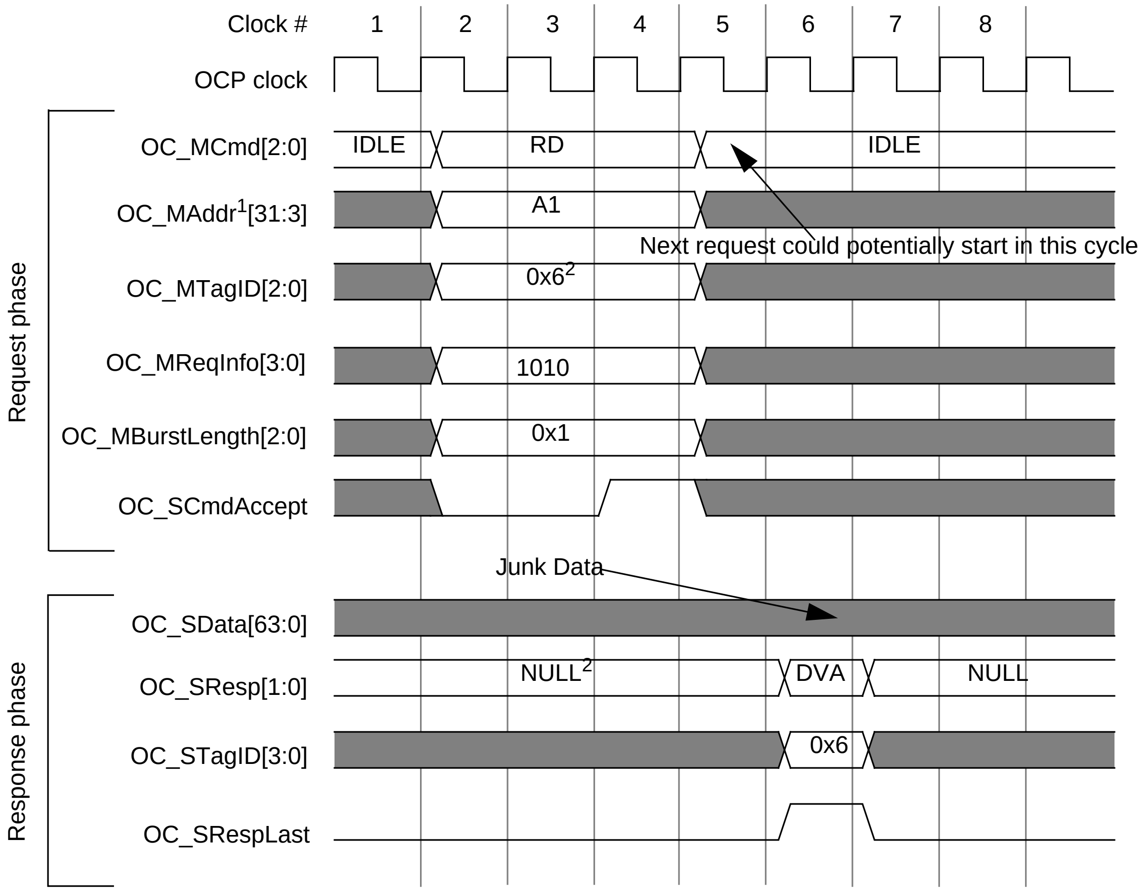
Figure 16 Burst OCP Write Transaction with flow control

### SYNC Transaction

Figure 17 shows a SYNC transaction. The 24Kf core will only generate this type of transaction for SYNC instructions. The transaction looks very similar to a single read (RD) transaction, along with the following properties:

1. *OC_MByteEn* is 0x0
2. *OC_MTagID* is 0x6
3. *OC_MAddr[31:8]* is 0x1fc000
4. *OC_MAddr[7:3]* carries the SYNC stype bits [10:6]
5. *OC_MReqInfo* is 0xA

The response is shown in clock 6 when the *OC_SResp* bus is asserted by the slave, indicating a DVA (DVA is a valid acknowledgment in OCP terminology).



[1] Since 24Kf cores uses RD for SYNC's, a conservative address (close to the MIPS boot vector) is sent out on OC_MAddr. This is a concatenation of {0x1fc000, sync_opcode_stype[4:0]}.
[2] Core always forces the OC_MDataTagID[2:0] signal to be 0x6 for SYNC's.

Figure 17 SYNC operation as a OCP RD Transaction

## Revision History

In the left hand page margins of this document you may find vertical change bars to note the location of significant changes to this document since its last release. Significant changes are defined as those which you should take note of as you use the MIPS IP. Changes to correct grammar, spelling errors or similar may or may not be noted with change bars. Change bars will be removed for changes which are more than one revision old.

Please note: Limitations on the authoring tools make it difficult to place change bars on changes to figures. Change bars on figure titles are used to denote a potential change in the figure itself. Certain parts of this document (Instruction set descriptions, EJTAG register definitions) are references to Architecture specifications, and the change bars within these sections indicate alterations since the previous version of the relevant Architecture document.

Table 20 Revision History

| Revision | Date | Description |
|---|---|---|
| 00.90 | July 17, 2003 | Initial version |
| 00.91 | July 31, 2003 | Updates based on early feedback |
| 00.92 | August 8, 2003 | Preliminary external release |
| 00.93 | September 12, 2003 | • Described several updates to the OCP interface (Thread model, SYNC behavior, MReqInfo, MAddrSpace)<br>• Added burst order section<br>• Added core-to-bus clocking relationship waveform and description |
| 00.94 | September 29, 2003 | • Removed I/O SError. Use interrupts instead for async bus errors<br>• EJ_DINT type should be A<br>• Added 4 L2 performance counter signals to I/O list<br>• Added sync pattern table to SI_OCPSync in external interface section. |
| 00.95 | December 3, 2003 | • Added Uncached Accelerated flush condition on store to a different 32B region<br>• Trademark updates |
| 01.00 | December 10, 2003 | • Updated template |
| 01.01 | January 27, 2004 | • Noted option of running FPU at same clock rate as integer core.<br>• Changed names of BIST-related interface signals; they now begin with *MB_*.<br>• Clarified that an OCP write data phase starts one cycle after the command phase is accepted. |
| 01.02 | February 11, 2004 | • Clarified number of possible hardware breakpoints.<br>• Fixed document template |
| 02.00 | March 5, 2004 | • Removed unused *SI_OCPClkIn* input pin.<br>• Renamed *gscan{in,out}_x* pins to *gscan{in,out}[n-1:0]*.<br>• Increased number of parity bits in I-cache data array, if parity is enabled.<br>• Updated MDU latencies |
| 02.01 | May 26, 2004 | • Added new static inputs to control selection of integrated memory BIST algorithm.<br>• Added Table 16 summarizing build-time configuration options. |

Table 20 Revision History

| Revision | Date | Description |
|---|---|---|
| 03.00 | September 15, 2004 | • Described SPRAM and COP2 interfaces.<br>• Added breakpoint status output pins.<br>• Updated OCP interface to OCP version 2.1, with use of TagID fields. |
| 03.01 | November 10, 2004 | • OCP Sync waveform clarified.<br>• Other minor improvements. |
| 03.02 | March 15, 2005 | • Described the MIPS Trace interface.<br>• Other minor updates |
| 03.03 | April 26, 2005 | • Described the ISPRAM interface.<br>• Other minor updates |
| 03.04 | June 30, 2005 | • Various enhancement updates |

MIPS Technologies reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error or omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Except as expressly provided in any written license agreement from MIPS Technologies or an authorized third party, the furnishing of this document does not give recipient any license to any intellectual property rights, including any patent rights, that cover the information in this document.

The information contained in this document shall not be exported, reexported, transferred, or released, directly or indirectly, in violation of the law of any country or international law, regulation, treaty, Executive Order, statute, amendments or supplements thereto. Should a conflict arise regarding the export, reexport, transfer, or release of the information contained in this document, the laws of the United States of America shall be the governing law.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government ("Government"), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or an authorized third party.

Template: DB1.13, Built with tags: 2B