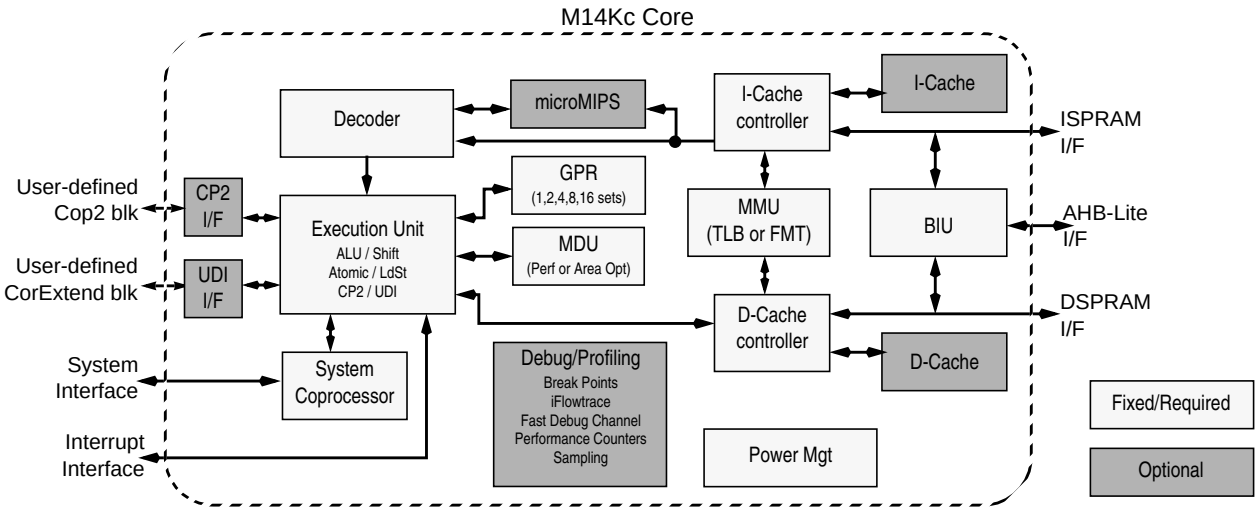
MIPS32® M14Kc ™ Processor Core Datasheet December 17, 2010

The MIPS32® M14Kc™ core from MIPS® Technologies is a member of the MIPS32 M14K™ processor core family. It is a high performance, small-silicon-area, 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. It is fully synthesizable and highly portable across processes, and can be easily integrated into full system-on-silicon designs, allowing developers to focus their attention on end-user products. The M14Kc core is ideally positioned to support new products for emerging segments of the digital consumer, network, systems, and information-management markets, enabling new tailored solutions for embedded applications.

The M14Kc implements the MIPS32 Release-2 Architecture in a 5-stage pipeline. It includes support for the microMIPS™ ISA, an Instruction Set Architecture with optimized MIPS32 16-bit and 32-bit instructions, that provides a significant reduction in code size with a performance equivalent to MIPS32. The M14Kc is a successor to the 4KE®, designed from the same microarchitecture, but with additional features and enhancements including the Microcontroller Application-Specific Extension (MCU™ ASE), enhanced interrupt handling, lower interrupt latency, built-in native AMBA®-3 AHB-Lite Bus Interface Unit (BIU), and additional power saving, debug, and profiling features.

Figure 1 shows a block diagram of the M14Kc core. The core is divided into *required* and *optional* (shaded) blocks.

Figure 1 MIPS 32® M14Kc™ Core Block Diagram



The M14Kc retains the following features and functions from the 4KE processor core:

- Support for MIPS32 Release 2 Architecture with MIPS32 instruction decoder.
- Support for multiple shadow register sets.
- The Memory Management Unit (MMU), selectable between a Translation Lookaside Buffer (TLB) or a simple Fixed Mapping Translation (FMT) mechanism.
- Multiply/Divide Unit (MDU), configurable between performance/area optimizations. The high-performance optimization supports a single-cycle 32x16-bit MAC instruction or two-cycle 32x32-bit instructions. Instruction and data caches are fully configurable from 0 to 64 Kbytes in size. In addition, each cache can be organized as direct-mapped or 2-way, 3-way, or 4-way set associative. Load and fetch cache misses only block until the critical word becomes available. The pipeline resumes execution while the remaining words are being written to the cache. Both caches are virtually indexed and physically tagged to allow them to be accessed in the same clock in which the address is translated.

An optional Enhanced JTAG (EJTAG version 4.52) block allows for single-stepping of the processor as well as instruction and data virtual address/value breakpoints. iFlowtrace™ version 2.0 is also supported to add real-time instruction program counter and special events trace capability for debug. Additionally, Fast Debug Channel, Performance Counters, and PC/Data sampling functions are added to enrich debug and profiling features on the M14Kc core.

Features

- 5-stage pipeline
- 32-bit Address and Data Paths
- 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)
- 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 (one, three, seven, or fifteen additional shadows can be optionally added to minimize latency for interrupt handlers)
    - Bit field manipulation instructions
    - Virtual memory support (smaller page sizes and hooks for more extensive page table manipulation)
- microMIPS-Compatible Instruction Set
    - microMIPS ISA is a build-time configurable and run-time convertible ISA to improve code size density over MIPS32, while maintaining MIPS32 performance.
    - microMIPS supports all MIPS32 instructions (except for branch-likely instructions) with new optimized 32-bit encoding. Frequent MIPS32 instructions are available as 16-bit instructions.
    - Added seventeen new and thirty-five MIPS32 corresponding commonly-used instructions in 16-bit opcode format.
    - Stack pointer implicit in instruction.
    - MIPS32 assembly and ABI compatible.
    - Supports ASEs and User-defined Instructions (UDI).
- MCU™ ASE
    - Increases the number of interrupt hardware inputs from 6 to 8 for Vectored Interrupt (VI) mode, and from 63 to 255 for External Interrupt Controller (EIC) mode.
    - Separate priority and vector generation. 16-bit vector address is provided.
    - Hardware assist, combined with the use of Shadow Register Sets to reduce interrupt latency during the prologue and epilogue of an interrupt.
    - An interrupt return with automated interrupt epilogue handling instruction (IRET) improves interrupt latency.
    - Supports optional interrupt chaining.
    - Two memory-to-memory atomic read-modify-write instructions (ASET and ACLR) eases commonly used semaphore manipulation in microcontroller applications. Interrupts are automatically disabled during the operation to maintain coherency.
- Programmable Cache Sizes
    - Individually configurable instruction and data caches
    - Sizes from 0 - 64KB
    - Direct Mapped, 2-, 3-, or 4-Way Set Associative
    - Loads block only until critical word is available
    - Write-back and write-through support

    - 16-byte cache line size
    - Virtually indexed, physically tagged
    - Cache line locking support
    - Non-blocking prefetches
- Scratchpad RAM (SPRAM) Support
    - Can optionally replace 1 way of the I- and/or D-cache with a fast scratchpad RAM
    - Independent external pin interfaces for I- and D-scratchpads
    - 20 index address bits allow access of arrays up to 1MB
    - Interface allows back-stalling the core
- MIPS32 Privileged Resource Architecture (PRA)
    - Count/Compare registers for real-time timer interrupts
    - I and D watch registers for SW breakpoints
- Memory Management Unit
    - Simple Fixed Mapping Translation (FMT) mechanism, or
    - 4-entry instruction and data Translation Lookaside Buffers (ITLB/DTLB) and a 16 or 32 dual-entry joint TLB (JTLB) with variable page sizes. Read, write, and execute page-protection attributes individually programmable.
- Bus Interface Unit (BIU)
    - Support AMBA-3 AHB-Lite protocol
    - All I/O's fully registered
    - Separate unidirectional 32-bit address and data buses
    - Two 16-byte collapsing write buffers
- Parity Support
    - The I-cache, D-cache, ISPRAM, and DSPRAM support optional parity detection.
- CorExtend® User-Defined Instruction Set Extensions
    - 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
- Multiply/Divide Unit (high-performance configuration)
    - Maximum issue rate of one 32x16 multiply per clock
    - Maximum issue rate of one 32x32 multiply every other clock
    - Early-in iterative divide. Minimum 11 and maximum 34 clock latency (dividend (*rs*) sign extension-dependent)
- Multiply/Divide Unit (area-efficient configuration)
    - 32 clock latency on multiply
    - 34 clock latency on multiply-accumulate
    - 33-35 clock latency on divide (sign-dependent)
- Coprocessor 2 interface
    - 32-bit interface to an external 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
    - Optional power-saving mode in organizing individual cache memory array per way
- EJTAG Debug/Profiling and iFlowtrace™ Mechanism
    - Support for single stepping
    - Virtual instruction and data address/value breakpoints
    - Complex breakpoint unit allows more detailed specification of break conditions
    - TAP controller is chainable for multi-CPU debug
    - Cross-CPU breakpoint support
    - iFlowtrace support for real-time instruction PC and special events
    - PC and/or load/store address sampling for profiling
    - Performance Counters
    - Support Fast Debug Channel (FDC)
- Testability
    - Full scan design achieves test coverage in excess of 99% (dependent on library and configuration options).
    - Optional memory BIST for internal SRAM arrays. Two memory BIST algorithms are provided and selectable by input pin.

Architecture Overview

The M14Kc core contains both required and optional blocks, as shown in Figure 1. Required blocks must be implemented to remain MIPS-compliant. Optional blocks can be added to the M14Kc core based on the needs of the implementation.

The required blocks are as follows:

- Execution Unit

- General Purposed Registers (GPR)
- Multiply/Divide Unit (MDU)
- System Control Coprocessor (CP0)
- Memory Management Unit (MMU)
- I/D Cache Controllers
- Bus Interface Unit (BIU)
- Power Management

Optional or configurable blocks include:

- Instruction Cache
- Data Cache
- Scratchpad RAM interface
- microMIPS
- Coprocessor 2 interface
- CorExtend® User-Defined Instruction (UDI) interface
- Enhanced JTAG (EJTAG) Controller and Trace logic

The section "MIPS32® M14Kc™ Core Required Logic Blocks" on page 5 discusses the required blocks. The section "MIPS32® M14Kc™ Core Optional or Configurable Logic Blocks" on page 11 discusses the optional blocks.

Pipeline Flow

The M14Kc core implements a 5-stage pipeline with a performance similar to the 4KE pipeline. The pipeline allows the processor to achieve high frequency while minimizing device complexity, reducing both cost and power consumption.

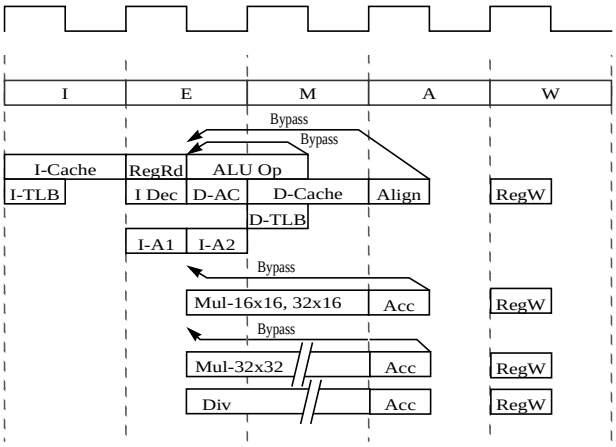
The M14Kc core pipeline consists of five stages:

- Instruction (I Stage)
- Execution (E Stage)
- Memory (M Stage)
- Align (A Stage)
- Writeback (W stage)

The M14Kc core implements a bypass mechanism that allows the result of an operation to be forwarded directly to the instruction that needs it without having to write the result to the register and then read it back.

Figure 2 shows a timing diagram of the M14Kc core pipeline (shown with the-high performance MDU and TLB used in the MMU).

Figure 2 MIPS32® M14Kc™ Core Pipeline



I Stage: Instruction Fetch

During the Instruction fetch stage:

- An instruction is fetched from instruction cache.
- microMIPS instructions are recoded into MIPS32 instructions if microMIPS mode is selected.

E Stage: Execution

During the Execution stage:

- Operands are fetched from the register file.
- The arithmetic logic unit (ALU) begins the arithmetic or logical operation for register-to-register instructions.
- The ALU calculates the virtual data address for load and store instructions.
- The ALU determines whether the branch condition is true and calculates the virtual branch target address for branch instructions.
- Instruction logic selects an instruction address.
- All multiply and divide operations begin in this stage.

M Stage: Memory Fetch

During the Memory fetch stage:

- The arithmetic ALU operation completes.
- The data cache access and the data virtual-to-physical address translation are performed for load and store instructions.
- Data cache look-up is performed and a hit/miss determination is made.
- A 16x16 or 32x16 multiply calculation completes (high-performance MDU option).

- A 32x32 multiply operation stalls the MDU pipeline for one clock in the M stage (high-performance MDU option).
- A multiply operation stalls the MDU pipeline for 31 clocks in the M stage (area-efficient MDU option).
- A multiply-accumulate operation stalls the MDU pipeline for 33 clocks in the M stage (area-efficient MDU option).
- A divide operation stalls the MDU pipeline for a maximum of 34 clocks in the M stage. Early-in sign extension detection on the dividend will skip 7, 15, or 23 stall clocks (only the divider in the fast MDU option supports early-in detection).

A Stage: Align

During the Align stage:

- Load data is aligned to its word boundary.
- A multiply/divide operation updates the HI/LO registers (area-efficient MDU option).
- A 16x16 or 32x16 multiply operation performs the carry-propagate-add. The actual register writeback is performed in the W stage (high-performance MDU option).
- A MUL operation makes the result available for writeback. The actual register writeback is performed in the W stage.
- EJTAG complex break conditions are evaluated.

W Stage: Writeback

During the Writeback stage:

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

MIPS32® M14Kc™ Core Required Logic Blocks

The required logic blocks of the M14Kc core (Figure 1) are defined in the following subsections.

Execution Unit

The M14Kc core execution unit implements a load/store architecture with single-cycle ALU operations (logical, shift, add, subtract) and an autonomous multiply/divide unit.

The execution unit includes:

- 32-bit adder used for calculating the data address
- Address unit for calculating the next instruction address
- Logic for branch determination and branch target address calculation
- 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

General Purpose Registers

The M14Kc core contains thirty-two 32-bit general-purpose registers used for integer operations and address calculation. Optionally, one, three, seven or fifteen 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.

Multiply/Divide Unit (MDU)

The M14Kc core includes a multiply/divide unit (MDU) that contains a separate pipeline for multiply and divide operations. This pipeline operates in parallel with the integer unit (IU) pipeline and does not stall when the IU pipeline stalls. This allows the long-running MDU operations to be partially masked by system stalls and/or other integer unit instructions.

Two configuration options exist for the MDU: an area-efficient iterative block, and a higher performance 32x16 array. The selection of the MDU style allows the implementor to determine the appropriate trade-off for his application.

Area-Efficient MDU Option

With the area-efficient option, multiply and divide operations are implemented with a simple 1-bit-per-clock iterative algorithm. Any attempt to issue a subsequent MDU instruction while a multiply/divide is still active causes an MDU pipeline stall until the operation is completed.

Table 1 lists the latency (number of cycles until a result is available) for the M14Kc core multiply and divide instructions. The latencies are listed in terms of pipeline clocks.

Table 1 Area-Efficient Integer Multiply/Divide Unit Operation Latencies

Opcode	Operand Sign	Latency
MUL, MULT, MULTU	any	32
MADD, MADDU, MSUB, MSUBU	any	34
DIVU	any	33
DIV	pos/pos	33
	any/neg	34
	neg/pos	35

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

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, respectively. 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.

High-Performance MDU

The M14Kc core includes a multiply/divide unit (MDU) that contains a separate pipeline for multiply and divide operations. This pipeline operates in parallel with the integer unit (IU) pipeline and does not stall when the IU pipeline stalls. This setup allows long-running MDU operations, such as a divide, to be partially masked by system stalls and/or other integer unit instructions.

The high-performance MDU consists of a 32x16 booth recoded multiplier, result/accumulation registers (*HI* and *LO*), a divide state machine, and the necessary multiplexers and control logic. The first number shown (‘32’ of 32x16) represents the *rs* operand. The second number (‘16’ of 32x16) represents the *rt* operand. The M14Kc core only checks the value of the *rt* operand to determine how many times the operation must pass through the multiplier. The 16x16 and 32x16 operations pass through the multiplier once. A 32x32 operation passes through the multiplier twice.

The MDU supports execution of one 16x16 or 32x16 multiply operation every clock cycle; 32x32 multiply operations can be issued every other clock cycle. Appropriate interlocks are implemented to stall the issuance of back-to-back 32x32 multiply operations. The multiply operand size is automatically determined by logic built into the MDU.

Divide operations are implemented with a simple 1-bit-per-clock iterative algorithm. 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 an IU pipeline stall until the divide operation has completed.

Table 2 lists the repeat rate (peak issue rate of cycles until the operation can be reissued) and latency (number of cycles until a result is available) for the M14Kc 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 M14Kc® Processor Core Family Software User’s Manual*.

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. By avoiding the explicit MFLO instruction, required when using the *LO* register, and by supporting multiple destination registers, the throughput of multiply-intensive operations is increased.

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 diagnostics capability, the operating modes (kernel, user, and debug), and whether interrupts are enabled or disabled. Configuration information, such as cache size and set associativity, presence of build-time options like microMIPS, CorExtend ASE or Coprocessor 2 interface, is also available by accessing the CP0 registers.

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.

Interrupt Handling

The M14Kc core includes support for eight hardware interrupt pins, two software interrupts, and a timer 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 M14Kc core, so the VInt bit will always read as a 1 for the M14Kc 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. The presence of this mode denoted by the VEIC bit in the *Config3* register. Again, this mode is architecturally optional. On the M14Kc 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 interrupt compatibility mode, such that a processor supporting Release 2 of the Architecture, the M14Kc core for example, 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 on entry to a particular vector. The shadow registers further improve interrupt latency by avoiding the need to save context when invoking an interrupt handler.

In the M14Kc core, interrupt latency is greatly improved over the 4KE by:

- Speculative interrupt vector prefetching during the pipeline flush
- Interrupt Automated Prologue (IAP) by hardware: Shadow Register Sets remove the need to save GPRs, and IAP removes the need to save specific Control Registers when handling an interrupt.
- Interrupt Automated Epilogue (IAE) by hardware: Shadow Register Sets remove the need to restore GPRs, and IAE removes the need to restore specific Control Registers when returning from an interrupt.
- Allow interrupt chaining. When servicing an interrupt and interrupt chaining is enabled, there is no need to return from the current Interrupt Service Routine (ISR) if there is another valid interrupt pending to be serviced. The control of the processor can jump directly from the current ISR to the next ISR without IAE and IAP.

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 may be a build-time option on some MIPS core. Although Release 2 of the Architecture defines a maximum of 16 shadow sets, the M14Kc core allows 1 (the normal GPRs), 2, 4, 8, or 16 shadow sets. The highest number actually implemented is indicated by the $SRSCtl_{HSS}$ 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, references 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, and 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

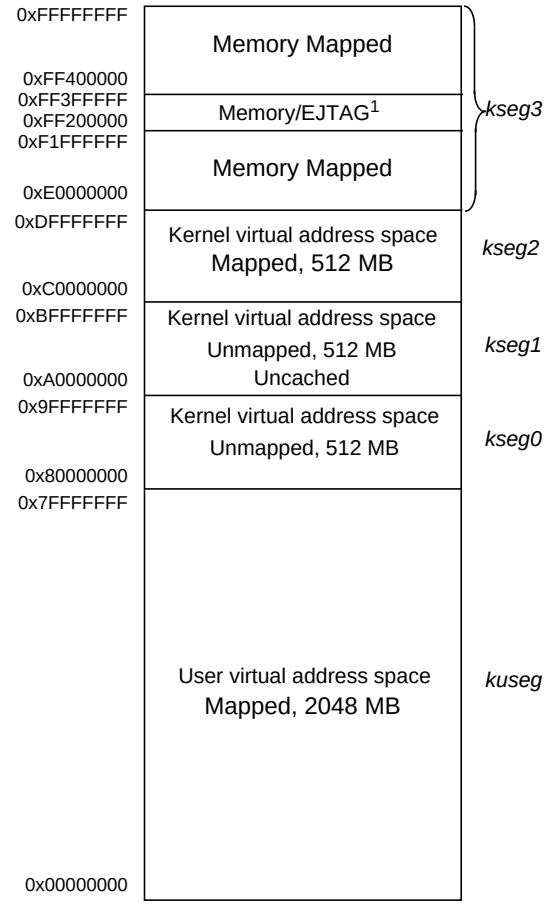
the PSS field of the *SRSCtl* register provides the number of the previous shadow register set (that which was current before the last exception or interrupt occurred).

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 $SRSCtl_{CSS}$ is copied to $SRSCtl_{PSS}$, and $SRSCtl_{CSS}$ is set to the value taken from the appropriate source. On an ERET, the value of $SRSCtl_{PSS}$ is copied back into $SRSCtl_{CSS}$ to restore the shadow set of the mode to which control returns.

Modes of Operation

The M14Kc core supports three modes of operation: user mode, kernel mode, and debug mode. User mode is most often used for applications programs. 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 the EJTAG section for more information on debug mode.

Figure 3 M14Kc Core Virtual Address Map



1. This space is mapped to memory in user or kernel mode, and by the EJTAG module in debug mode.

Memory Management Unit (MMU)

The M14Kc core offers one of the two choices of MMU that interfaces between the execution unit and the cache controller, namely Translation Lookaside Buffer (TLB) and Fixed Mapping Translation (FMT).

Translation Lookaside Buffer (TLB)

A TLB-based MMU consists of three translation buffers: a 16 or 32 dual-entry fully associative Joint TLB (JTLB), a 4-entry fully associative Instruction TLB (ITLB) and a 4-entry fully associative data TLB (DTLB).

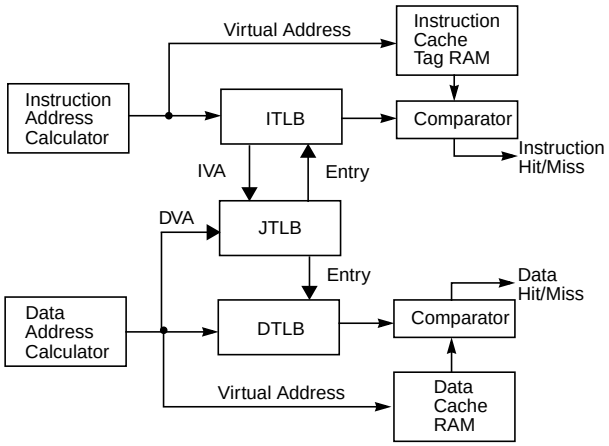
When an instruction address is calculated, the virtual address is compared to the contents of the 4-entry ITLB. If the address is not found in the ITLB, the JTLB is accessed. If the entry is found in the JTLB, that entry is then written into the ITLB. If the address is not found in the JTLB, a TLB refill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

When a data address is calculated, the virtual address is compared to both the 4-entry DTLB and the JTLB. If the address is not found in the DTLB, but is found in the JTLB, that address is immediately written to the DTLB. If the address is not found in the JTLB, a TLB refill exception is taken.

The M14Kc core TLB allows pages to be protected by a read inhibit and an execute inhibit attribute in addition to the write protection attribute defined by MIPS32 PRA.

Figure 4 shows how the ITLB, DTLB, and JTLB are implemented in the M14Kc core.

Figure 4 Address Translation During a Cache Access



The TLB consists of three address translation buffers:

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

Joint TLB (JTLB)

The M14Kc core implements a 16 or 32 dual-entry, fully associative JTLB that maps 32 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-Kbytes (or 1-Kbyte) to 256-Mbytes into the 4-Gbyte physical address space. By default, the minimum page size is normally 4-Kbytes on the M14Kc core; as a build time option, it is possible to specify a minimum page size of 1-Kbyte.

The JTLB is organized in page pairs to minimize the overall size. Each *tag* entry corresponds to 2 data entries: an even page entry and an odd page entry. The highest order virtual address bit not participating in the tag comparison is used to determine which of the data entries is used. Since page size can vary on a page-pair basis, the determination of which address bits participate in the comparison and which bit is used to make the even-odd determination is decided dynamically during the TLB 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 minimum sized pages/subpages. The minimum page size is either 1-Kbyte or 4-Kbyte, depending on the *PageGrain* and *Config3* registers.

The ITLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing store 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. If successful, the translation information is copied into the ITLB for future use. There is a two-cycle ITLB miss penalty.

Data TLB (DTLB)

The DTLB is a small 4-entry, fully associative TLB dedicated to performing translations for loads and stores. Similar to the ITLB, the DTLB only maps either 1-Kbyte or 4-Kbyte pages/subpages depending on the *PageGrain* and *Config3* registers.

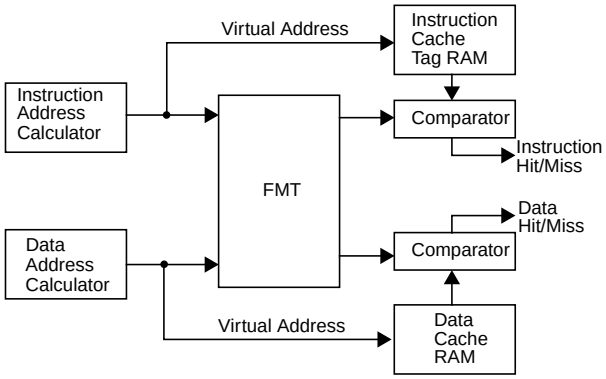
The DTLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing store for the DTLB. The JTLB is looked-up in parallel with the DTLB to minimize the DTLB miss penalty. If the JTLB translation is successful, the translation information is copied into the DTLB for future use. There is a one cycle DTLB miss penalty.

Fixed Mapping Translation (FMT)

An FMT is smaller and simpler than a 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 translated identically by the FMT.

Figure 5 shows how the FMT is implemented in the M14Kc core.

Figure 5 Address Translation During Cache Access



Cache Controllers

The M14Kc core instruction and data cache controllers support caches of various sizes, organizations, and set-associativity. For example, the data cache can be 2 Kbytes in size and 2-way set associative, while the instruction cache can be 8 Kbytes in size and 4-way set associative. Each cache can each be accessed in a single processor cycle. In addition, each cache has its own 32-bit data path, and both caches can be accessed in the same pipeline clock cycle. Refer to the section entitled "MIPS32® M14Kc™ Core Optional or Configurable Logic Blocks" on page 11 for more information on instruction and data cache organization.

The cache controllers also have built-in support for replacing one way of the cache with a scratchpad RAM. See the section entitled "Scratchpad RAM" on page 12 for more information on scratchpad RAMs.

Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) serves as the interface between the M14Kc core and the outside world. Primarily, the BIU receives read/write requests from the cache controller. These requests will be arbitrated and turned into bus transactions via the AMBA-3 AHB-lite protocol. The characteristics of the BIU are:

- AHB-Lite is a subset of the AHB bus protocol that supports a single bus master. It does not support complex Split/Retry operations.
- Shared 32-bit read/write address bus
- Two unidirectional 32-bit data buses for read and write operations
- Single read/write and burst (WRAP mode) read/write are supported.

Hardware Reset

For historical reasons within the MIPS architecture, the M14Kc core has two types of reset input signals: *SI_Reset* and *SI_ColdReset*.

Functionally, these two signals are ORed together within the core and then used to initialize critical hardware state. Both reset signals can be asserted either synchronously or asynchronously to the core clock, *SI_ClkIn*, and will trigger a Reset exception. The reset signals are active high and must be asserted for a minimum of 5 *SI_ClkIn* cycles. The falling edge triggers the Reset exception. The primary difference between the two reset signals is that *SI_Reset* sets a bit in the *Status* register; this bit could be used by software to distinguish between the two reset signals, if desired. The reset behavior is summarized in Table 2.

Table 2 Reset Types

SI_Reset	SI_ColdReset	Action
0	0	Normal operation, no reset.
1	0	Reset exception; sets $Status_{SR}$ bit.
X	1	Reset exception.

One (or both) of the reset signals must be asserted at power-on or whenever hardware initialization of the core is desired. A power-on reset typically occurs when the machine is first turned on. A hard reset usually occurs when the machine is already on and the system is rebooted.

In debug mode, EJTAG can request that a soft reset (via the *SI_Reset* pin) be masked. It is system-dependent whether this functionality is supported. In normal mode, the *SI_Reset* pin cannot be masked. The *SI_ColdReset* pin is never masked.

Power Management

The M14Kc 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 M14Kc 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, $Status_{EXL}$, $Status_{ERL}$, and $Debug_{DM}$ support the power management function by allowing the user to change the power state if an exception or error occurs while the M14Kc 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 four 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*, *SI_Reset*, and *SI_ColdReset*) 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 M14Kc core asserts the *SI_Sleep* signal, which is part of the system interface bus, whenever the WAIT instruction is executed. The assertion of *SI_Sleep* indicates that the clock has stopped and the M14Kc core is waiting for an interrupt.

Local clock gating

The majority of the power consumed by the M14Kc core is 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.

MIPS32® M14Kc™ Core Optional or Configurable Logic Blocks

The M14Kc core contains several optional or configurable logic blocks, shown as shaded in the block diagram in Figure 1.

Instruction Cache

The instruction cache is an optional on-chip memory block of up to 64 Kbytes. 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. The tag holds 22 bits of physical address, a valid bit, and a lock bit. The LRU replacement bits (0-6b per set depending on associativity) 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 M14Kc 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 optional on-chip memory block of up to 64 Kbytes. This virtually indexed, physically tagged cache is protected. Because the data cache is virtually indexed, the virtual-to-physical address translation occurs in parallel with the cache access. The tag holds 22 bits of physical address, a valid bit, and a lock bit. There is an additional array holding dirty bits and LRU replacement algorithm bits (0-6b depending on associativity) for each set of the cache.

In addition to instruction-cache locking, the M14Kc 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 cannot 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 M14Kc core incorporates on-chip instruction and data caches that can each be accessed in a single processor cycle. Each cache has its own 32-bit data path and can be accessed in the same pipeline clock cycle. Table 3 lists the M14Kc core instruction and data cache attributes.

Table 3 Instruction and Data Cache Attributes

Parameter	Instruction	Data
Size	0 - 64 Kbytes	0 - 64 Kbytes
Organization	1 - 4 way set associative	1 - 4 way set associative
Line Size	16 bytes	16 bytes
Read Unit	32 bits	32 bits
Write Policies	NA	write-through with write allocate, write-through without write allocate, write-back with write allocate
Miss restart after transfer of	miss word	miss word
Cache Locking	per line	per line

Cache Protocols

The M14Kc 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-through, write allocate**: Similar to above, but stores missing in the cache will cause a cache refill. The store data is then written to both the cache and main memory.
- **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.

Scratchpad RAM

The M14Kc core also supports replacing up to one way of each cache with a scratchpad RAM. Scratchpad RAM is accessed via independent external pin interfaces for instruction and data scratchpads. The external block which connects to a scratchpad interface is user-defined and can consist of a variety of devices. The main requirement is that it must be accessible with timing similar to an internal cache RAM. Normally, this means that an index will be driven one cycle, a tag will be driven the following clock, and the scratchpad must return a hit signal and the data in the second clock. The scratchpad can easily contain a large RAM/ROM or memory-mapped registers. Unlike the fixed single-cycle cache timing, however, the scratchpad interface can also accommodate back-stalling the core pipeline if data is not available in a single clock. This back-stalling capability can be useful for operations which require multi-cycle latency. It can also be used to enable arbitration of external accesses to a shared scratchpad memory.

The core's functional interface to a scratchpad RAM is slightly different from the interface to a regular cache RAM. Additional index bits allow access to a larger array, with 1MB of scratchpad RAM versus 4KB for a cache way. These bits come from the virtual address, so on a M14Kc core care must be taken to avoid virtual aliasing. The core does not automatically refill the scratchpad way and will not select it for replacement on cache misses.

microMIPS™ ISA

The M14Kc core supports the microMIPS ISA, which contains all MIPS32 ISA instructions (except for branch-likely instructions) in a new 32-bit encoding scheme, with some of the commonly used instructions also available in 16-bit encoded format. This ISA improves code density through the additional 16-bit instructions while maintaining a performance similar to MIPS32 mode. In microMIPS mode, 16-bit or 32-bit instructions will be fetched and recoded to legacy MIPS32 instruction opcodes in the pipeline's I stage, so that the M14Kc core can have the same 4KE microarchitecture. Because the microMIPS instruction

stream can be intermixed with 16-bit halfword or 32-bit word size instructions on halfword or word boundaries, additional logic is in place to address the word misalignment issues, thus minimizing performance loss.

microMIPS is MIPS32 assembly and ABI compatible with tool chain and ASE's support.

Depending on the optimization preference, the microMIPS can be configured in performance mode, with multiple recoding blocks being executed in parallel with Tag compare for each Way Associativity, or with a single recoding block after the Tag compare logic to improve area usage.

Coprocessor 2 Interface

The M14Kc core can be configured to have an interface for an on-chip coprocessor. This coprocessor can be tightly coupled to the processor core, allowing high-performance solutions integrating a graphics accelerator or DSP, for example.

The coprocessor interface is extensible and standardized on MIPS cores, allowing for design reuse. The M14Kc core supports a subset of the full coprocessor interface standard: 32b data transfer, no Coprocessor 1 support, single issue in-order data transfer to coprocessor, one out-of-order data transfer from coprocessor.

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 signals on the interface.

CorExtend® User-defined Instruction Extensions

An 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 execution unit. The interface to the UDI block is external to the M14Kc core.

Such 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 or local UDI state. The operation may complete in one cycle or multiple cycles, if desired.

EJTAG Debug Support

The M14Kc core provides for an Enhanced JTAG (EJTAG) interface for use in the software debug of application and kernel code. In addition to standard user mode and kernel modes of operation, the M14Kc 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.

The EJTAG interface operates through the Test Access Port (TAP), a serial communication port used for transferring test data in and out of the M14Kc core. In addition to the standard JTAG instructions, special instructions defined in the EJTAG specification specify which registers are selected and how they are used.

Debug Registers

Four debug registers (*DEBUG*, *DEBUG2*, *DEPC*, and *DESAVE*) have been added to the MIPS Coprocessor 0 (CP0) register set. The *DEBUG* and *DEBUG2* registers show the cause of the debug exception and are used for setting up single-step operations. The *DEPC* (Debug Exception Program Counter) register holds the address on which the debug exception was taken, which is used to resume program execution after the debug operation finishes. Finally, the *DESAVE* (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 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 M14Kc core: Instruction breakpoints and Data breakpoints. Additionally, *complex* hardware breakpoints can be included, which allow detection of more intricate sequences of events.

The M14Kc core can be configured with the following breakpoint options:

- No data, instruction, or complex breakpoints
- One data and two instruction breakpoints, without complex breakpoints
- Two data and four instruction breakpoints, without complex breakpoints
- Two data and six instruction breakpoints, with or without complex breakpoints
- Four data and eight instruction breakpoints, with or without complex breakpoints

Instruction breakpoints occur on instruction execution operations, and the breakpoint is set on the virtual address. Instruction breakpoints 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 binary range of instructions.

Data breakpoints occur on load/store transactions, and the breakpoint is set on a set of virtual address and ASID values, with the same single address or binary address range as the Instruction breakpoint. Data breakpoints can be set on a load, a store, or both. Data breakpoints can also be set to match on the operand value of the load/store operation, with byte-granularity masking. Finally, masks can be applied to both the virtual address and the load/store value.

Complex breakpoints utilize the simple instruction and data breakpoints and break when combinations of events are seen. Complex break features include:

- Pass Counters - Each time a matching condition is seen, a counter is decremented. The break or trigger will only be enabled when the counter has counted down to 0.
- Tuples - A tuple is the pairing of an instruction and a data breakpoint. The tuple will match if both the virtual address of the load or store instruction matches the instruction breakpoint, and the data breakpoint of the resulting load or store address and optional data value matches.
- Priming - This allows a breakpoint to be enabled only after other break conditions have been met. Also called sequential or armed triggering.
- Qualified - This feature uses a data breakpoint to qualify when an instruction breakpoint can be taken. Once a load matches the data address and the data value, the instruction break will be enabled. If a load matches the address, but has mis-matching data, the instruction break will be disabled.

Performance Counters

Performance counters are used to accumulate occurrences of internal predefined events/cycles/conditions for program analysis, debug, or profiling. A few examples of event types are clock cycles, instructions executed, specific instruction types executed, loads, stores, exceptions, and cycles while the CPU is stalled. There are two, 32-bit counters. Each can count one of the 64 internal predefined events selected by a corresponding control register. A counter overflow can be programmed to generate an interrupt, where the interrupt handler software can maintain larger total counts.

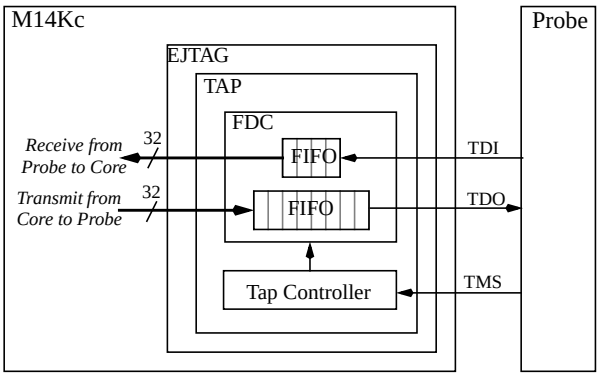
PC/Address Sampling

This sampling function is used for program profiling and hot-spots analysis. Instruction PC and/or Load/Store addresses can be sampled periodically. The result is scanned out through the EJTAG port. The Debug Control Register (*DCR*) is used to specify the sample period and the sample trigger.

Fast Debug Channel (FDC)

The M14Kc core includes optional FDC as a mechanism for high bandwidth data transfer between a debug host/probe and a target. FDC provides a FIFO buffering scheme to transfer data serially, with low CPU overhead and minimized waiting time. The data transfer occurs in the background, and the target CPU can either choose to check the status of the transfer periodically, or it can choose to be interrupted at the end of the transfer.

Figure 6 FDC Overview



iFlowtrace™

The M14Kc core has an option for a simple trace mechanism called iFlowtrace. This mechanism only traces the instruction PC, not data addresses or values. This simplification allows the trace block to be smaller and the trace compression to be more efficient. iFlowtrace memory can be configured as off-chip, on-chip, or both.

iFlowtrace also offers special-event trace modes when normal tracing is disabled, namely:

- Function Call/Return and Exception Tracing mode to trace the PC value of function calls and returns and/or exceptions and returns.
- Breakpoint Match mode traces the breakpoint ID of a matching breakpoint and, for data breakpoints, the PC value of the instruction that caused it.

- Filtered Data Tracing mode traces the ID of a matching data breakpoint, the load or store data value, access type and memory access size, and the low-order address bits of the memory access, which is useful when the data breakpoint is set up to match a binary range of addresses.
- User Trace Messages. The user can instrument their code to add their own 32-bit value messages into the trace by writing to the Cop0 UTM register.
- Delta Cycle mode works in combination with the above trace modes to provide a timestamp between stored events. It reports the number of cycles that have elapsed since the last message was generated and put into the trace.

Testability

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

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 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, using BIST controllers and logic that are 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 special side-band signal buses of configurable width are provided within the core to facilitate this approach.

Build-Time Configuration Options

The M14Kc core allows a number of features to be customized based on the intended application. Table 4 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 checking an appropriate register field. Refer to the *MIPS32® M14Kc™ 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 4 Build-time Configuration Options

Option	Choices	Software Visibility
Integer register file sets	1, 2, 4, 8 or 16	*SRSCtl*$_{HSS}$
Integer register file implementation style	Flops or generator	N/A
MMU type	FMT or TLB	*Config0*$_{MT}$
TLB Size	16 or 32 dual entries	*Config1*$_{MMUSize}$
microMIPS support	Present or not	*Config1*$_{CA}$
microMIPS implementation style	High performance or min area	N/A
Multiply/divide implementation style	High performance or min area	*Config*$_{MDU}$
EJTAG TAP controller	Present or not	N/A
EJTAG TAP Fast Debug Channel (FDC)	Present or not (even when TAP is present)	*DCR*$_{FDCI}$
* These bits indicate the presence of an external block. Bits will not be set if interface is present, but block is not.		

Table 4 Build-time Configuration Options (Continued)

Option	Choices	Software Visibility
EJTAG TAP FDC FIFO Size	Two TX/two RX or eight TX/four RX 32-bit registers	*FDCFG*
Instruction/data hardware breakpoints	0/0, 2/1, 4/2, 6/2, or 8/4	DCR_{IB}, IBS_{BCN} DCR_{DB}, DBS_{BCN}
Complex breakpoints	0/0, 6/2, or 8/4	DCR_{CBT}
Performance Counters	Present or not	$Config1_{PC}$
iFlowtrace hardware	Present or not	$Config3_{ITL}$
iFlowtrace memory location	On-core or off-chip	$IFCTL_{ofc}$
iFlowtrace on-chip memory size	256B - 8MB	N/A
Watch registers	0 - 8	$WatchHi_{M}$
CorExtend interface	Present or not	$Config_{UDI}$*
Coprocessor2 interface	Present or not	$Config1_{C2}$*
Instruction ScratchPad RAM interface	Present or not	$Config_{ISP}$*
Data ScratchPad RAM interface	Present or not	$Config_{DSP}$*
I-cache size	0 - 64 KB	$Config1_{IL}$, $Config1_{IS}$
I-cache associativity	1, 2, 3, or 4	$Config1_{IA}$
D-cache size	0 - 64 KB	$Config1_{DL}$, $Config1_{DS}$
D-cache associativity	1, 2, 3, or 4	$Config1_{DA}$
Memory BIST	Integrated (March C+ or IFA-13), custom, or none	N/A
Scan options for improved coverage around cache arrays	Present or not	N/A
Cache & ScratchPad RAM Parity	Present or not	$ErrCtl_{PE}$
Interrupt synchronizers	Present or not	N/A
Interrupt Vector Offset	Compute from Vector Input or Immediate Offset	N/A
Clock gating	Top-level, integer 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.

Revision History

Revision	Date	Description
01.00	July 12, 2009	Initial release
01.01	March 8, 2010	Removed protection of TLB pages by read and execute inhibit
01.02	March 16, 2010	Fixed typo in Fig. 1
02.00	December 17, 2010	Maintenance release 1

Template: nDb1.03, Built with tags: 2B