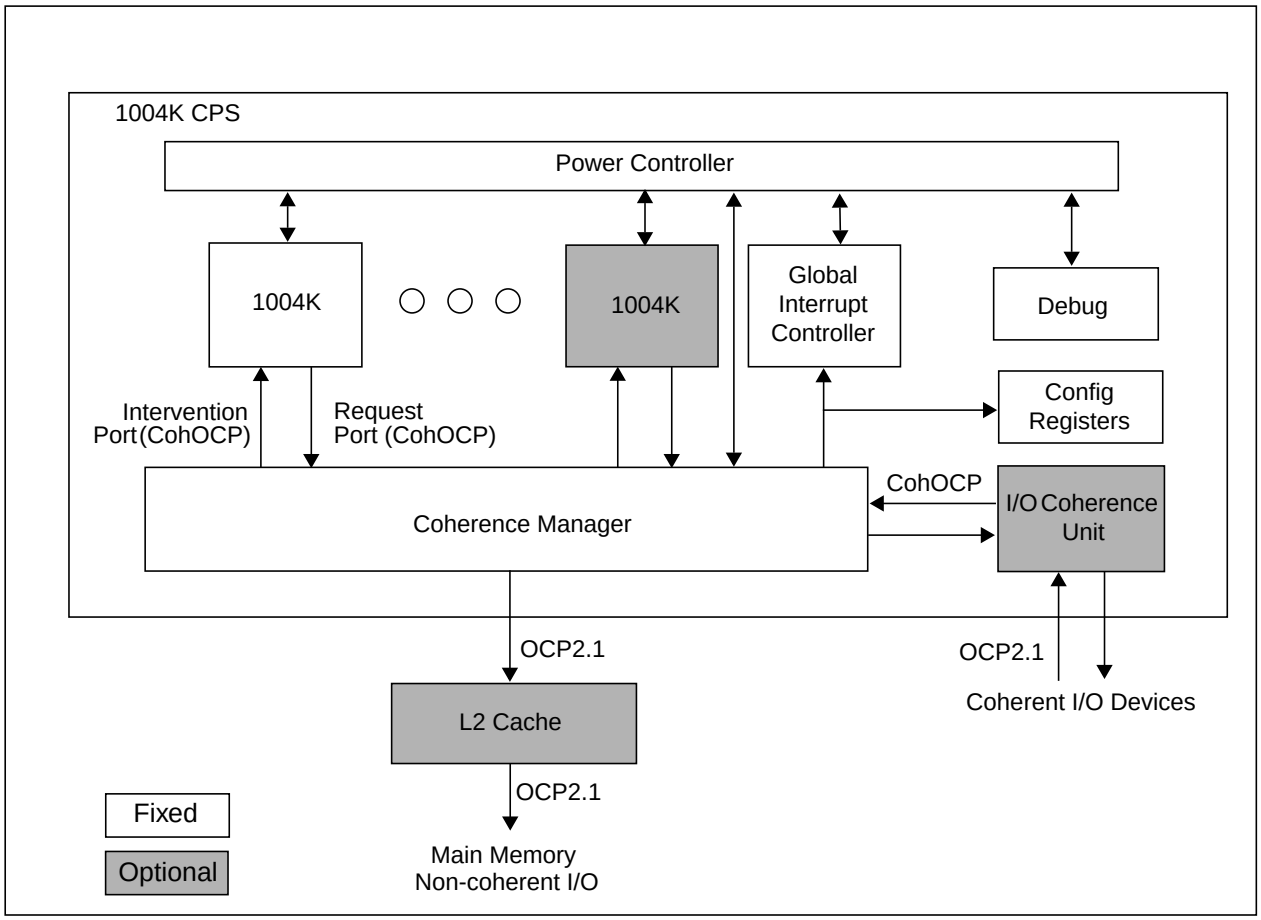
MIPS32® 1004K™ Coherent Processing System Datasheet January 21, 2011

The MIPS32® 1004K™ Coherent Processing System (CPS) from MIPS Technologies is a high-performance coherent multiprocessor cluster of one to four MIPS32® 1004K™ CPUs and an optional coherent I/O port. The 1004K CPU features Multi-Threading and Digital Signal Processing Application Specific Extensions and is based on the proven MIPS32® 34K® core. Multi-CPU coherence is managed in hardware by a Coherence Manager (CM), using extensions to the OCP-IP protocol and an OCP-based intervention port on each CPU. Figure 1 shows a block diagram of the Cluster.

The 1004K CPS Cluster can optionally be connected to the MIPS® SOC-it® L2 Cache Controller. When connected to the L2 cache, 256-bit datapaths are utilized to take advantage of the full bandwidth available from the L2 cache design. When the L2 cache is not present, the interface is restricted to 64-bit datapaths so that it matches the memory interface of existing non-coherent CPU core products from MIPS Technologies. The L2 or memory interface uses non-coherent OCP protocols.

The 1004K Cluster includes other modules that handle common system-level functions. The optional I/O Coherence Unit supports HW I/O coherence by bridging a non-coherent OCP I/O interconnect to the CM and handling ordering requirements. The Global Interrupt Controller handles the distribution of interrupts between and among the CPUs in the Cluster. Additionally, the Cluster Power Controller can gate off the clocks and power to idle cores to save power.

Figure 1 1004K™ Coherent Processing System Block Diagram



1004K™ CPS Features

- 1 - 4 coherent MIPS32 1004K™ CPU cores
- Power Controller to shut down idle CPU cores
- Optional hardware I/O coherence port
- MESI coherence states in L1 data caches
- Supports cache to cache data transfers
- Speculative memory reads to reduce latency
- Out-of-order data return
- Optimized 256b interface to SOC-it L2 cache controller
- Cluster EJTAG TAP included to control a shared on-chip PDtrace buffer

Supported Configurations

Refer to Section “Test Capability” for a complete list of configuration options.

The following configuration options are restricted:

- CPU:CM clock ratio fixed at 1:1
- CM:IOCU clock ratio fixed at 1:1

Area/Frequency

- The 1004K CPU is approximately 5% larger than an equivalent non-coherent 34K CPU, primarily because of the duplicated cache tags.
- The CM and other common logic adds 300-400K gates depending on configuration.
- The 1004K CPU does not have any frequency degradation relative to an equivalent non-coherent 34K.
- The CM runs at the same frequency as the CPUs.

Software Support

Modifications to the Linux operating system are available from MIPS Technologies that support SMP (for 1 VPE per CPU) and SMVP (for more than 1 VPE per CPU) build options.

The product is also supported by third party software releases as part of the on-going development of the ecosystem by MIPS.

Coherence Protocol

The coherence protocol is characterized by the following properties:

- Writeback cache - Uses a writeback cache to ensure high performance.
- Cache-line based - Coherence and ownership is maintained per cache line.
- Snoopy protocol - Each CPU snoops the stream of transactions and updates its cache state accordingly.
- Invalidate - A line is invalidated in the cache (possibly with a writeback to memory) when a store from another processor is seen.

Cache States

MESI cache states are used in the L1 data cache. A cache line can exist in one of the following states:

- *Modified:* The line has been modified. This is the only valid copy of the data in the system. No other L1 data caches will have a copy of this line in a valid state.
- *Exclusive*: No other L1 data caches have a copy of this line in a valid state. This CPU ‘owns’ the line and can modify it. The data for this line is consistent with memory.
- *Shared*: Cache line potentially exists in more than one L1 data cache. Read-only in all caches.
- *Invalid*: The line is not valid in this cache.

Ordering

Weak ordering is used within the Cluster. Reads and writes to different addresses can occur out of program order unless explicitly ordered through the use of a SYNC instruction.

Coherent OCP Extensions

The 1004K Cluster makes use of extensions to the OCP protocol to include information about coherent traffic. These extensions include:

- Coherent Request Types as shown in Table 1.
- Coherent state slave responses. On an intervention, indicates what MESI state the CPU had the line in. On a request, indicates what MESI state the line should be installed in.

- Dataless response - allows a dataless Upgrade transaction and is also a common response on the intervention port.

Table 1 Coherent Requests

Request	Description
(Legacy)	Non-coherent Reads/Writes.
CohReadShared	Request for a line in a Shared state (load miss). Data can remain in other data caches in the Shared state. Line may be upgraded to Exclusive if there are no other Sharers.
CohReadOwn	Request for Exclusive ownership of the line (store miss). Following this request, the line cannot remain in any other L1 data caches.
CohUpgrade	Request to upgrade line from shared to exclusive (store hit on shared). Following this request, the line will not remain in any other data caches. The upgrade can be done without transferring data.
CohWriteBack	Writeback of data (eviction). CPU is writing data back to memory. This request is not sent to other CPUs, as it is known they will not have valid copies of the line.
CohInvalidate	Invalidate the line in all caches (PREF Prepare for Store, CACHE HitInvalidate).
CohCopyBack	Write dirty data back to memory from any cache (CACHE HitWrite-Back).
CohCopyBackInv	Write dirty data back to memory from any cache and invalidate in all (CACHE HitWriteBackInvalidate).
CohWriteInvalidate	Write data to memory (I/O write). This write data is replacing the existing data. A partial line write will be merged with data from the caches and memory, and a full line write will invalidate the line in the caches and overwrite memory.

Table 1 Coherent Requests (Continued)

Request	Description
CohReadSharedDiscard	Read data from coherence domain (I/O read). Gets the most up-to-date data, but because the data is not going to a coherent cache, the state of the line does not need to change.
CohReadSharedAlways	Request line in Shared state only - cannot be installed as Exclusive or Modified. (Not used in this system - could be used by a caching I/O agent or for a coherent I-Cache fetch.)

Intervention Port

Devices with coherent caches include a second OCP port referred to as the *intervention port*. Coherency is maintained by sending all coherent requests to all devices via the intervention port. This includes requests by the device itself; referred to as a *self-intervention*, this provides a mechanism for the agent to determine the global ordering of its request with respect to requests from other agents.

Each device updates its cache state for the intervention and responds when the state transition has completed. The previous state of the line is indicated in the response. If a read type intervention hits on a line that the CPU has in a Modified or Exclusive state, the CPU returns the cache line with its response.

A cacheless device, such as the IOCU, does not require an intervention port.

MIPS32® 1004K™ CPU

The MIPS32® 1004K™ CPU is a high-performance, low-power, 32-bit MIPS RISC CPU designed for coherent system-on-silicon applications. It is based on the MIPS32® 34K® core, but features extensions for cache coherence. Two variants of the 1004K CPU exist: the 1004Kf CPU that features a floating point unit and the 1004Kc CPU that does not.

The 1004K CPU implements the MIPS32 Release 2 Architecture. In addition to the base architecture, it features the following application specific extensions (ASE):

- The MIPS MT ASE which defines multi-threaded operation.
- The MIPS DSP ASE which provides support for signal processing instructions.
- The MIPS16e™ ASE which reduces code size

This standard architecture allows support by a wide range of industry standard tools and development systems.

The MT ASE allows the CPU to operate more efficiently by executing multiple program streams concurrently. The CPU can be configured with 1 or 2 Virtual Processing Elements (VPEs), each of which contain much of the privileged coprocessor 0 state, including a full Memory Management Unit (MMU), to allow multiple OSes to operate concurrently on the processor. Additionally, the core can be configured to have from 1-9 Thread Contexts (TCs). A TC consists of a register file, a Program Counter, and a limited amount of privileged state. TCs offer lightweight multi-threading to allow cooperative or independent threads to run concurrently

The DSP ASE provides support for a number of powerful data processing operations. There are instructions for fractional arithmetic (Q15/Q31) and for saturating arithmetic. Additionally, for smaller data sizes, SIMD operations are supported, allowing 2x16b or 4x8b operations to occur simultaneously. Another feature of the ASE is the inclusion of additional HI/LO accumulator registers to improve the parallelization of independent accumulation routines.

The 1004Kf CPU also features an IEEE 754 compliant Floating Point Unit (FPU). The FPU supports both single and double precision instructions.

The synthesizable 1004K CPU includes a high performance Multiply/Divide Unit (MDU) by default. The MDU is fully pipelined to support a single cycle repeat rate for 32x32 MAC instructions. Further, in the 1004K Pro™ CPU, the optional CorExtend block can utilize the HI/LO registers in the MDU block. The CorExtend block allows specialized functions to be efficiently implemented.

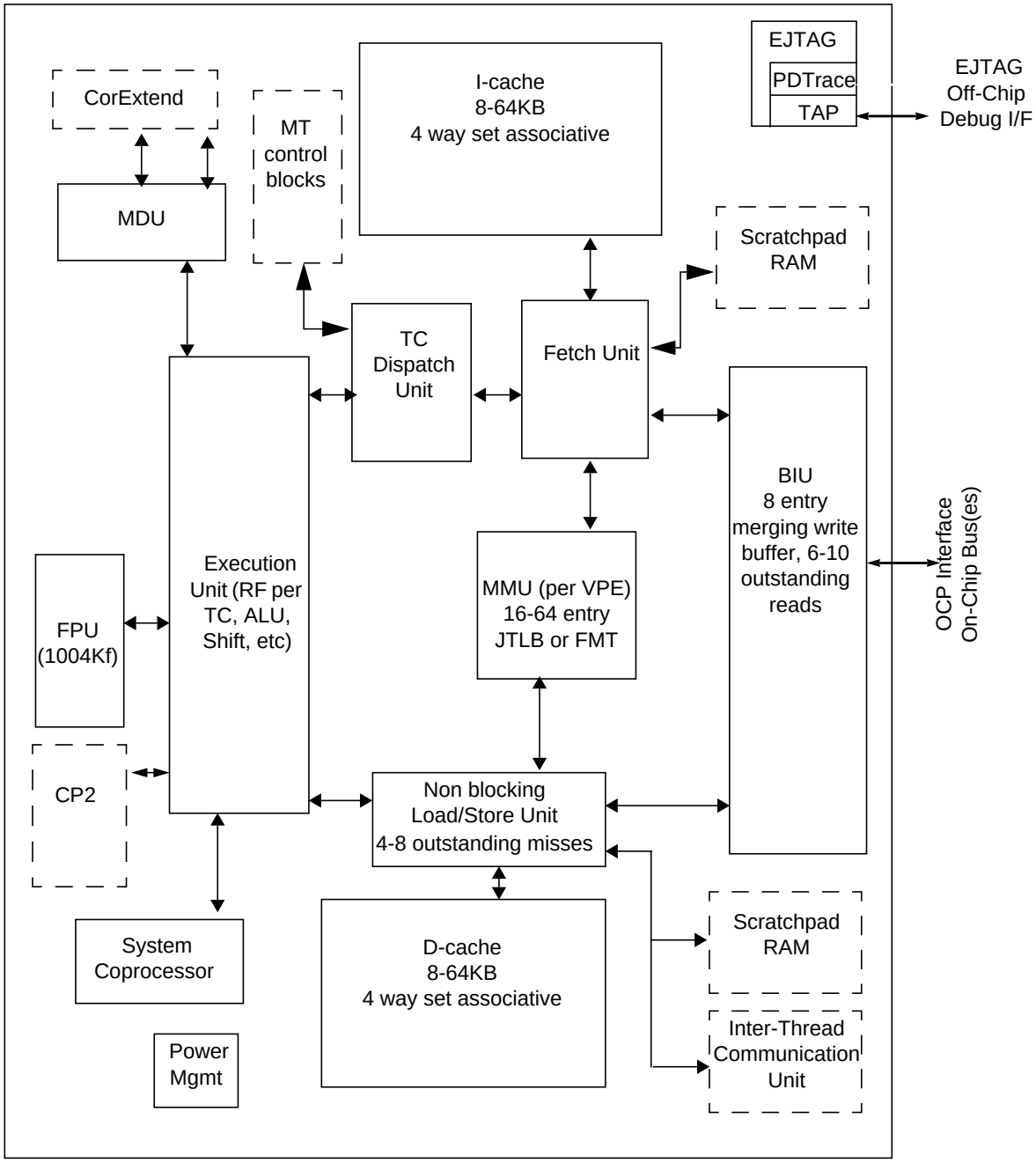
Instruction and data level one caches are configurable at 0(Instruction only), 8, 16, 32, or 64 KB in size. Each cache is organized as 4-way set associative by default. Data cache misses are non-blocking and up to 8 may be outstanding. Two instruction cache misses can be outstanding. To achieve high frequencies while using commercially available SRAM generators, the cache access is spread across two pipeline stages, dedicating 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.

An Enhanced JTAG (EJTAG) block allows for software debugging of the processor. This includes a TAP controller with PC sampling and Fast Debug Channel features. Optional features include instruction and data trace as well as instruction and data virtual address/value breakpoints.

Figure 2 shows a block diagram of the 1004K CPU. The dashed boxes indicate blocks that can be modified by the customer for specific applications.

Figure 2 1004K™ CPU Block Diagram



1004K™ CPU Features

- 8-9-stage pipeline (a thread selection stage is bypassed on single-TC CPUs, yielding 8 stages)
- 32-bit address paths
- 64-bit data paths to caches and external interface
- MIPS32 Release2 Instruction Set and Privileged Resource Architecture
- MIPS16e™ Code Compression (optional)
- MIPS MT Application Specific Extension (ASE)
  - Support for 1 or 2 Virtual Processing Elements (VPEs) each with 1 Thread Context (TC)
  - Inter-Thread Communication (ITC) memory for efficient communication & data transfer.
- MIPS DSP ASE (optional)
  - 3 additional pairs of accumulator registers.
  - Fractional data types (Q15, Q31)
  - Saturating arithmetic
  - SIMD instructions operate on 2x16b or 4x8b simultaneously.
- Programmable Memory Management Unit
  - 16/32/64 dual-entry JTLB per VPE
  - 4-5 entry MT-optimized ITLB
  - 8-entry DTLB
  - Optional simple Fixed Mapping Translation (FMT) mechanism
  - Programmable L1 Cache Sizes
  - Individually configurable instruction and data caches
  - 4-Way Set Associative sizes of 4/8/16/32/64 KB
  - Up to 9 outstanding load misses
  - MESI coherent cache states in L1 data cache
  - 32-byte cache line size
  - Virtually indexed, physically tagged
  - Cache line locking support
  - Non-blocking prefetches
  - Optional parity support
- Scratchpad RAM support
  - Separate RAMs for Instruction and Data
  - Independent of cache configuration
  - Maximum size of 1MB
  - Reference design available that features two 64 bit OCP interfaces for external DMA
- Bus Interface
  - OCP interface with 32-bit address and 64-bit data
  - Extensions for communicating coherence information about the request
  - Intervention port for receiving snoop requests from Coherence Manager
  - Burst size of four 64-bit beats
  - 8 entry write buffer
  - “Simple” byte enable mode allows easier bridging to other bus standards
  - Extensions for front-side L2 cache
- Multiply/Divide Unit (High Performance)
  - Maximum issue rate of one 32x32 multiply per clock
  - 5 cycle multiply latency
  - Early-in iterative divide. Minimum 11 and maximum 34 clock latency (dividend (*rs*) sign extension-dependent)
- Multiply/Divide Unit (Iterative)
  - Reduced area option that maintains full MIPS32 compatibility
  - Iterative 1 bit per cycle processing of multiplies and divides
  - Not available with DSP ASE or CorExtend access
- CorExtend™ User Defined Instruction Set Extensions
  - Allows user to define and add instructions to the CPU at build time
  - Maintains full MIPS32 compatibility
  - Supported by industry standard development tools
  - Single or multi-cycle instructions
  - Includes access to HI and LO registers
- Floating Point Unit (FPU) (1004Kf CPU only)
  - IEEE-754 compliant Floating Point Unit
  - Compliant to MIPS 64b FPU standards
  - Supports single and double precision data types
  - Optionally run at 1:1 or 2:1 CPU/FPU clock ratio
  - Supports Multiple thread contexts
- 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 fine-grained clock gating
- EJTAG Debug
  - Support for single stepping
  - Instruction address and data address/value breakpoints
  - TAP controller is chainable for multi-CPU debug
  - Cross-CPU breakpoint support

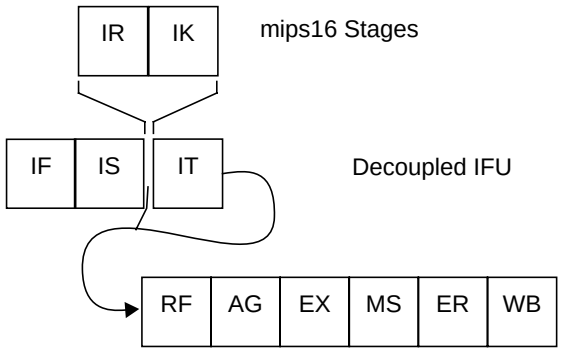
- MIPS Trace
  - PC, data address and data value tracing w/ trace compression
  - Includes features for correlation with CM trace
  - Support for on-chip and off-chip trace memory
- Testability
  - Full scan design achieves test coverage in excess of 99% (dependent on library and configuration options)
  - Optional memory BIST for internal SRAM arrays

Pipeline Flow

The 1004K CPU implements a 8-9-stage pipeline. One stage is bypassed if the CPU is configured with a single TC. Two 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.

Figure 3 shows a diagram of the 1004K CPU pipeline.

Figure 3 1004K™ CPU 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
- Branch prediction

IR - Instruction Recode

- MIPS16e instruction recode

IK - Instruction Kill

- MIPS16e instruction kill

IT - Instruction Fetch Third

- Instruction Buffer
- Thread selection
- This stage is bypassed on single TC configurations when the instruction buffer is empty.
- 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
- DCache SRAM access
- Branch Resolution
- Data watch and EJTAG break address compares

MS - Memory Access Second

- DCache hit detection
- Way select mux
- Load align

ER- Exception Resolution

- Instruction completion
- Register file write setup
- Exception processing

WB - Writeback

- Register file writeback occurs on rising edge of this cycle

1004K™ CPU Logic Blocks

The 1004K CPU consists of the following logic blocks, shown in Figure 2. These logic blocks are defined in the following subsections.

Fetch Unit

This block is responsible for fetching instructions for all Thread Contexts (TCs). Each TC has an 8-entry instruction buffer (IBF) that decouples the fetch unit from the execution unit. When executing instructions from multiple TCs, a portion of the IBF is used as a skid buffer. Instructions are held in the IBF after being sent to the execution unit. This allows stalled instructions to be flushed from the execution pipeline without needing to be refetched.

In order to fetch instructions without intervention from the execution unit, the fetch unit contains branch prediction logic. 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. The BHT is shared by all TCs on the processor, while the RPS is dynamically associated with a single TC.

Thread Schedule Unit (TSU)

This unit is responsible for dispatching instructions from different Thread Contexts (TCs). An external policy manager assigns priorities for each TC. The TSU determines which TCs are runnable and selects the highest priority one available. If multiple are available, a round-robin mechanism will select between them fairly.

The policy manager is a customer configurable block. Simple round-robin or fixed priority policies can be implemented by tying off signals on the interface. A reference policy manager is also included that implements a weighted round-robin algorithm for long-term distribution of execution bandwidth.

Execution Unit

The 1004K CPU execution unit implements a load/store architecture with single-cycle ALU operations (logical, shift, add, subtract) and an autonomous multiply/divide unit. Each TC on a 1004K CPU contains thirty-one 32-bit general-purpose registers used for integer operations and address calculation. 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

MIPS16e™ Application Specific Extension

The 1004K CPU includes support for the MIPS16e ASE. This ASE improves code density through the use of 16-bit encodings 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 1004K CPU 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 masked by instructions on other TCs and/or other integer unit instructions.

The standard 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 1 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 1004K CPU 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 9 of *Programming the MIPS32 1004K™ Core Family.*

Table 1 1004K™ CPU Integer Multiply/Divide Unit Latencies and Repeat Rates (High Performance MDU)

Opcode	Operand Size (mul *rt*) (div *rs*)	Latency	Repeat Rate
MULT/MULTU, MADD/MADDU, MSUB/MSUBU	32 bits	5	1
MUL	32 bits	5	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

* If there is no data dependency, a MUL can be issued every cycle.

For applications which will not use the MDU much, an iterative MDU is also available. This MDU saves area while still preserving MIPS32 compatibility. Both multiplies and divides are processed using a 1-bit per cycle iterative algorithm and have 34 cycle latencies.

Floating Point Unit (FPU) / Coprocessor 1

The 1004K CPU 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 per-thread 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 CPU. The FPU is not as deeply pipelined as the integer CPU so the maximum CPU frequency will only be attained with the FPU running at one-half the CPU 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 CPU 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 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

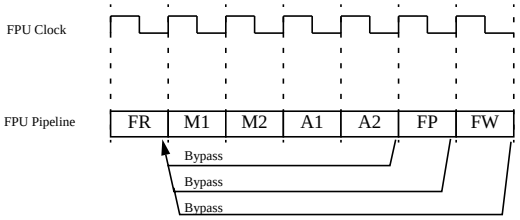
The FPU implements a high-performance 7-stage pipeline:

- 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 4 shows the FPU pipeline

Figure 4 FPU Pipeline



FPU Instruction Latencies and Repeat Rates

Table 2 contains the floating point instruction latencies and repeat rates for the 1004K CPU. 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 2 1004K™ 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
* Format: S = Single, D = Double, W = Word, L = Longword		

Table 2 1004K™ FPU Latency and Repeat Rate

Opcode*	Latency (FPU cycles)	Repeat Rate (FPU cycles)
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		

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.

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.

Most of CP0 is replicated per VPE. A small amount of state is replicated per TC and some is shared between the VPEs.

Interrupt Handling

Each 1004K VPE includes support for six hardware interrupt pins, two software interrupts, a timer interrupt, and a performance counter interrupt. These interrupts can be used in the following interrupt modes:

- 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 1004K CPU, so the VInt bit will always read as a 1 for the 1004K CPU.

If a TC is configured to be used as a shadow register set, the VI interrupt mode can 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.

Modes of Operation

The 1004K CPU 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 15 for more information on debug mode.

Memory Management Unit (MMU)

Each 1004K VPE contains a Memory Management Unit (MMU) that is primarily responsible for converting virtual addresses to physical addresses and providing attribute information for different segments of memory. At synthesis time, the type of MMU can be chosen independently for each VPE from the following options:

- Translation Lookaside Buffer (TLB)
- Fixed Mapping Translation (FMT)

In a dual-TLB configuration, each VPE contains a separate JTLB so that the translations for each are independent from each other.

The following sections explain the MMU options in more detail.

Translation Lookaside Buffer (TLB)

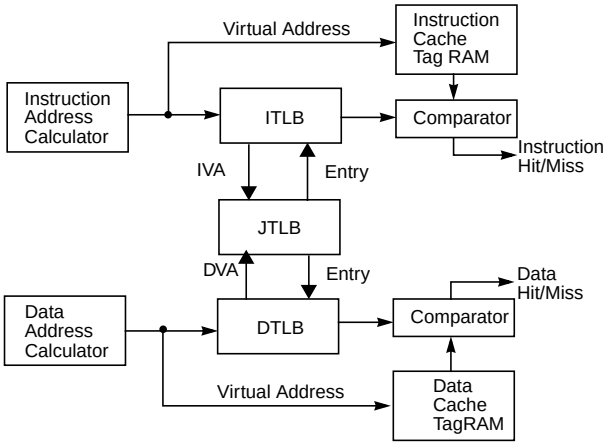
The basic TLB functionality is specified by the MIPS32 Privileged Resource Architecture. A TLB provides mapping and protection capability with per-page granularity. The 1004K implementation allows a wide range of page sizes to be present simultaneously.

The TLB contains a fully associative Joint TLB (JTLB). To enable higher clock speeds, two smaller micro-TLBs are also implemented: the Instruction Micro TLB (ITLB) and the 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 uTLB, 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 1004K CPU.

Figure 5 Address Translation During a Cache Access



Joint TLB (JTLB)

The JTLB is a fully associative TLB cache containing 16, 32, or 64-dual-entries mapping up to 128 virtual pages to their corresponding physical addresses. The address 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 dedicated to performing translations for the instruction stream. The ITLB is a hybrid structure having 3 entries that are shared by all TCs plus an additional entry dedicated to each TC.

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 an 8-entry, fully associative TLB dedicated to performing translations for loads and stores. All entries are shared by all TCs. 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.

Fixed Mapping Translation (FMT)

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. 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.

Data Cache

The data cache is an on-chip memory block of 4/8/16/32/64 KB, with 4-way associativity. Direct mapped caches of 0/1/2/4/8/16 KB are also supported, though not generally recommended for performance reasons. A tag entry holds 20 or 21 bits of physical address, two cache state bits, and an optional parity bit. The data entry holds 64 bits of data per way, with optional parity per byte. There are 4 data entries for

each tag entry. The tag and data entries exist for each way of the cache. There is an additional array that holds the dirty and LRU replacement algorithm bits for all 4 ways(6b LRU, 4b dirty, and optionally 4b dirty parity).

Using 4KB pages in the TLB and 32 or 64KB cache sizes it would normally be possible to get virtual aliasing. Because it is quite challenging for software to manage virtual aliases across multiple devices, these larger cache arrays are banked on the aliased 1 or 2 physical address bits to eliminate the virtual aliases.

When built with a 4-way cache, the 1004K CPU supports data-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 locked contents can be updated on a store hit, but will not be selected for replacement on a cache miss. Locked lines do not participate in the coherence scheme so processes which lock lines into a particular cache should be locked to that processor and prevented from migrating.

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.

Instruction Cache

The instruction cache is an on-chip memory block of 8/16/32/64 KB, with 4-way associativity.Direct mapped caches of 0/1/2/4/8/16 KB are also supported, though not generally recommended for performance reasons. A tag entry holds 20 or 21 bits of physical address, a valid bit, a lock bit, and an optional parity bit. The instruction data entry holds two instructions (64 bits), 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). There are four data entries for each tag entry. The tag and data entries exist for each way of the cache. The LRU replacement bits (6b) are shared among the 4 ways and 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.

Just like the data cache, with certain cache and TLB page sizes, it is possible to have virtual aliasing in the instruction cache. This is less of a problem because the instruction cache is not written so the aliases are always consistent. If instruction memory is modified, all of the aliases should be flushed from the instruction cache. The CPU can automatically check all possible aliases when invalidating an address from the instruction cache.

The 1004K CPU also supports instruction-cache locking when configured as 4-way set associative. 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.

Cache Memory Configuration

The 1004K CPU 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 1 lists the 1004K CPU instruction and data cache attributes.

Table 1 1004K™ CPU Instruction and Data Cache Attributes

Parameter	Instruction	Data
Size and Organization	4, 8, 16, 32, or 64 KB* 4-way set associative	4, 8, 16, 32, or 64 KB 4-way set associative
	0,1,2,4,8 or 16 KB* Direct Mapped	0,1,2,4,8 or 16 KB Direct Mapped
Line Size	32 Bytes*	32 Bytes
Read Unit	64 bits*	64 bits
Write Policies	N/A	coherent and non-coherent 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 1004K CPU supports the following cache protocols:

- **Non-coherent, 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.
- **Non-coherent, write-back, write allocate:** Loads and stores that miss in the cache will cause a cache refill. The memory read request will be marked as non-coherent. However, once the data is written to the cache, there is no distinction between coherent and non-coherent data. 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.
- **Non-coherent, 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.
- **Coherent, write-back, write allocate, exclusive on write:** Use coherent data. Load misses will bring the data into the cache in a shared state. Multiple caches can contain data in the shared state. Stores will bring data into the cache in an exclusive state - no other caches can contain that same line. If a store hits on a shared line in the cache, a request will be made to upgrade to exclusive.
- **Coherent, write-back, write allocate, exclusive:** Similar to the above, but load misses will bring data into the cache in an exclusive state rather than shared. This can be used if data is not shared and will eventually be written. This can reduce bus traffic because the line does not have to be refetched in an exclusive state when a store is done.

Intervention Processing

The CPU includes a duplicate set of tags for the data cache as well as the fill-store buffer and write-back buffer. Duplicate tags allow intervention lookups to be done in parallel with regular cache accesses for loads and stores. If an intervention hits and needs to change the cache state or read the cache data array, load and store accesses from the main pipe are stalled.

Interventions that ‘miss’ can be fully pipelined. Intervention lookups are speculatively started assuming a miss on previous interventions. If there is a hit, subsequent interventions are killed and then restarted after intervention processing has completed.

If a read-type intervention hits in the cache on a line that is Exclusive or Modified, the CPU reads the data out of the cache and returns it on the intervention port. This data is staged through the write-back buffer and at least one entry of the buffer will always be available for interventions to avoid deadlock conditions.

Intervention latencies are 5 cycles for a miss and 9 cycles to get a response and the first dword of data (if required) for a hit. While data is being returned, the CPU can resume processing subsequent interventions and provide responses in parallel.

Scratchpad RAM

The 1004K CPU allows blocks of scratchpad RAM to be attached to the load/store and fetch units. These allow low-latency access to a fixed block of memory.

These blocks can be modified by the customer. A reference design is provided which includes an SRAM array as well as an external DMA port to allow the system to directly access the array.

InterThread Communication Unit (ITU)

This block provides a mechanism for efficient communication between TCs using gating storage. This block has a number of locations that can be accessed using different views. These views provide the mechanisms to implement a number of useful communication methods such as mailboxes, FIFO mailboxes, mutexes, and semaphores.

This block can be modified by the customer to target a specific application. A reference multi-core ITU design is included with the CPS that implements some basic views and functionality.

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.

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 eight 32-byte buffers. Each buffer contains data from a single 32-byte aligned block of memory.

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.

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 or dword stores.

Gathering of uncached accelerated stores will start on cache-line aligned addresses, i.e. 32 byte aligned addresses. Once an uncached accelerated store starts 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 store that does not merge and does not go to an aligned address will be treated as a regular uncached store.

SimpleBE Mode

To aid in attaching the 1004K CPU 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.

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 a read 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.

EJTAG Debug Support

The 1004K CPU 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 1004K CPU 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 1004K CPU. In addition to the standard JTAG instructions, special instructions defined in the EJTAG specification define what registers are selected and how they are used.

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 1004K CPU: Instruction breakpoints and Data breakpoints.

During synthesis, the 1004K CPU can be configured to support the following breakpoint options per VPE:

- Zero instruction and zero data
- Two instruction and one data
- Four instruction and two data

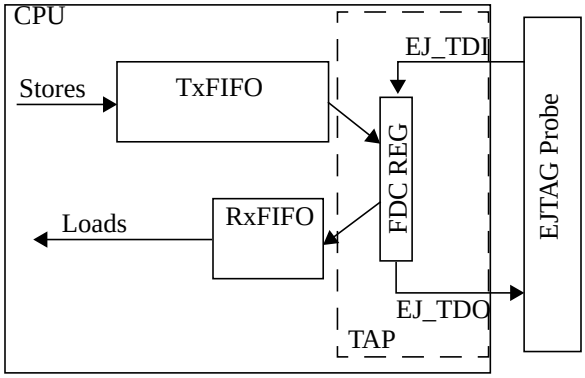
Instruction breaks occur on instruction fetch operations, and the break is set on the virtual address. Instruction breaks can also be made on the ASID value used by the MMU. A mask can be applied to the virtual address to set breakpoints on a range of instructions.

Data breakpoints occur on load and/or store transactions. Breakpoints are set on virtual address and ASID values, similar to the Instruction breakpoint. 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.

Fast Debug Channel

The 1004K CPU includes the EJTAG Fast Debug Channel (FDC) as a mechanism for efficient bidirectional data transfer between the CPU and the debug probe. Data is transferred serially via the TAP interface. A pair of memory- mapped FIFOs buffer the data, isolating software running on the CPU from the actual data transfer. Software can configure the FDC block to generate an interrupt based on the FIFO occupancy or can poll the status.

Figure 6 Fast Debug Channel



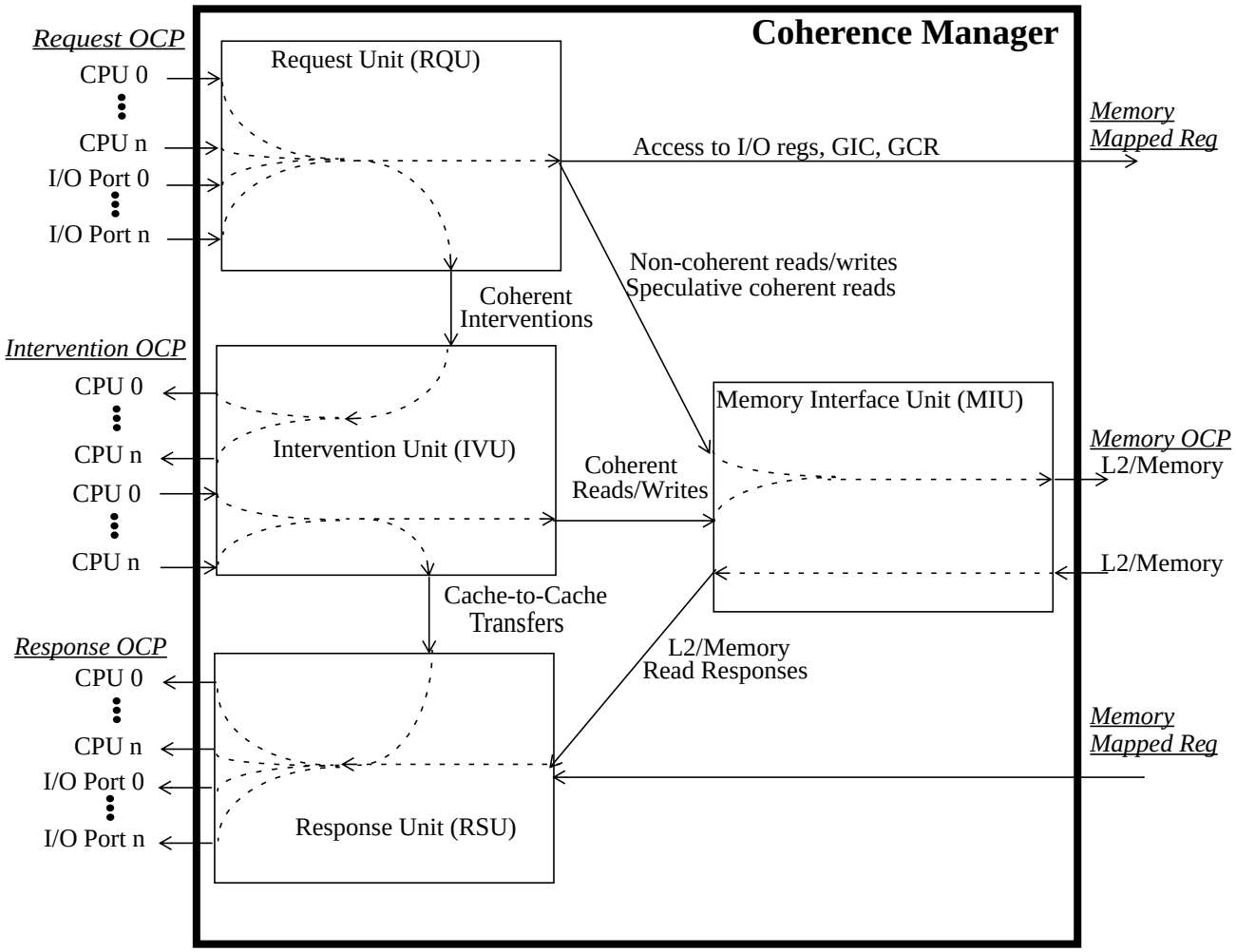
MIPS Trace

The 1004K CPU includes optional MIPS Trace support for real-time tracing of instruction addresses, data addresses and data values. The trace information is sent out of the CPU to a trace funnel where it is interleaved with trace data from the other CPUs and Coherence Manager. 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 1MB; 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.

Coherence Manager

The Coherence Manager (CM) is the glue that holds the coherent devices together. It is responsible for establishing the global ordering of requests as well as collecting the intervention responses and sending the correct data back to the requester. A high-level view of the request flow through the CM is shown in Figure 7. Each of the sub-units is described in more detail below.

Figure 7 Coherence Manager Transaction Flow



Request Unit (RQU)

This block receives requests from the coherent devices and serializes them. Non-coherent requests are forwarded to the Memory Interface Unit. Coherent requests are sent to the Intervention Unit.

The CM supports a speculative read. On a coherent read request, the memory request is started assuming that the line will not be available from another L1 cache. This avoids increasing the memory latency for the common case when it is not. In order to avoid a read after write hazard, the read address is compared against pending coherent requests that can generate writes. If a match is detected, the read will not be started speculatively.

Intervention Unit (IVU)

This block receives the serialized stream of coherent requests from the Request Unit. These requests are sent as interventions to each of the coherent caching agents. The caching agent updates its cache state appropriately for the intervention and gives a response. If the cache has the line in an Exclusive or Modified state, it returns the data with its response on a read type intervention. The Intervention Unit gathers the responses from each of the agents and takes care of the following actions:

- Speculative reads are resolved (confirmed or cancelled).
- Memory reads that are required because they were not speculated are issued to the Memory Interface Unit.
- Any Modified data returned from the CPU is sent to the Memory Interface Unit to be written back to memory.
- Any data returned from the CPU is forwarded to the Response Unit to be sent to the requester.
- The MESI state in which the line is installed by the requesting CPU is determined (the install state). If there are no other CPUs with the data, a Shared request is upgraded to Exclusive.

Memory Interface Unit (MIU)

This block handles the interface to the L2 cache or memory. Non-coherent reads and writes as well as speculative coherent reads are sent to the Memory Interface Unit from the Request Unit. Coherent writes and late reads are generated from the Intervention Unit.

The external interface may run at a lower frequency than the CM, and the external block may not be able to accept as many requests as multiple CPUs can generate, so there is some buffering of requests here.

This block is responsible for staging the read data back to each of the agents. There are independent staging registers for each agent, which allows concurrent data return to different agents. This block buffers the read data returned from the system if the read is still speculative or if the response unit is busy.

Response Unit (RSU)

This block is responsible for returning data to the requesting agent. It can send data from the Intervention Unit, the Memory Interface Unit, or from memory-mapped accesses to the GCR/GIC/MMIO.

Performance

The CM has a number of features that improve performance:

- Cache to Cache transfers: If a read request hits in another L1 cache in the Exclusive or Modified state, it will return the data to the CM and it will be forwarded to the requesting CPU, reducing latency on the miss.
- Speculative Reads: Coherent read requests are forwarded to the memory interface before they are looked up in the other caches. This is speculating that the cache line will not be found in another CPU's L1 cache. If another cache was able to provide the data, the memory request is not needed, and the CM will cancel the speculative request —dropping the request if it has not been issued or dropping the memory response if it has.

Table 3 provides a cycle-by-cycle description of the latency added by the CM to a read request (assuming the internal queues are all empty). In this case, the CM adds 6 cycles to the round-trip latency of the request.

Table 3 Read Request Timing

Cycle	Description
0	CPU sends out request, captured by input buffers in CM.RQU.
1	Serialization of requests in CM.RQU. One request is selected in Round-robin fashion from all sources.
2	Request is sent from CM.RQU to CM.MIU (bypassing internal queues).
3	Request is presented on L2/memory interface.
3+N	Response from L2 on bus. Captured by input flops in CM.MIU.
4+N	Lookup in table to determine what to do with response (drop, wait for intervention, etc.). Response data forwarded to CM.MIU output queue.
5+N	Response is forwarded to the CM.RSU.
6+N	Response is sent back to the CPU.

I/O Coherence Unit (IOCU)

The 1004K CPS cluster optionally supports hardware I/O coherence. This allows I/O devices to access memory while maintaining coherence with the caches in the CPUs. Coherent reads and writes from I/O devices will generate interventions to the CPU cores that will query the L1 data caches. Reads will be able to get the latest data values from the caches or memory. Writes will invalidate the stale data from the caches and will merge the newer write data with the older existing data as needed.

Hardware I/O coherence is supported through the I/O Coherence Unit (IOCU). The IOCU acts an interface block between the Coherence Manager and coherent I/O devices. A possible system topology is shown in Figure 8.

A reference design is provided for the IOCU. This block provides a legacy (without coherent extensions) OCP slave interface to the I/O interconnect for I/O devices to read and write system memory. Further, an OCP Master port to the I/O interconnect is included to allow the CPUs to access registers and memory on the I/O devices.

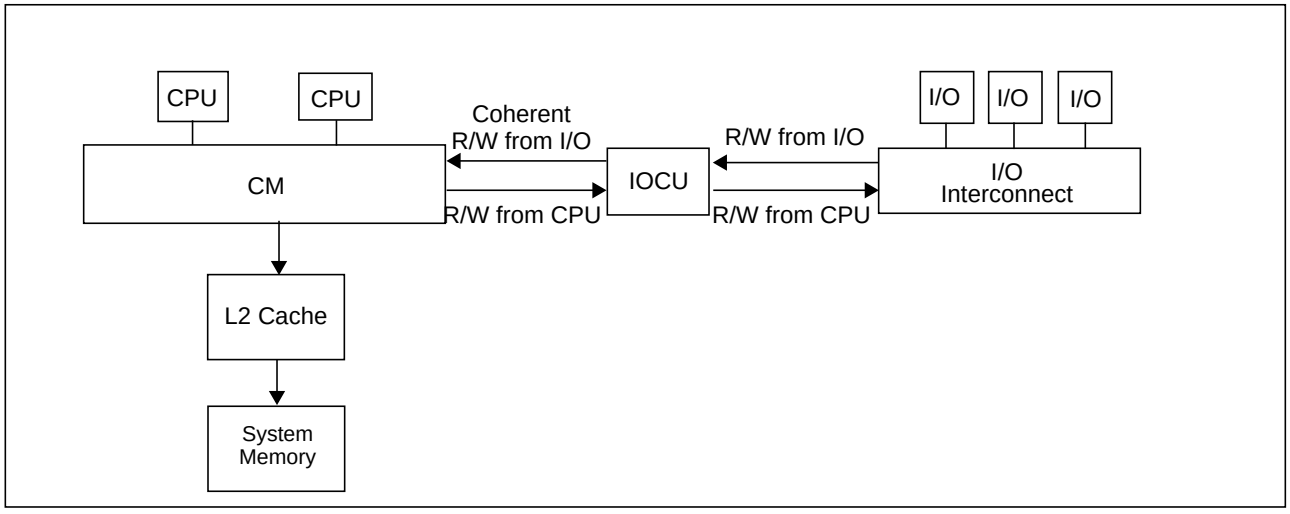
The reference IOCU design provides several features for easier integration:

- Customer-defined mapping unit can define cache attributes for each request—coherent or not, cacheable (in L2) or not, and L2 allocation policy.
- Supports incremental bursts up to 16 beats on I/O side. These requests are split into cache-line sized requests on the CM side.
- Ensures proper ordering of responses for the split requests and tagged requests.

In addition, the reference design has a number of features that support the producer-consumer ordering model and help ensure that transaction ordering can be enforced. These features include:

- Set-aside buffer: This buffer can hold up read responses from the I/O device until previous writes have completed.
- Writes are issued to the CM in the order they were received.
- CM provides ACK to the IOCU when writes are “visible” (guaranteed that a subsequent CPU read will receive that data):
    - non-coherent write is ACK'ed after serialization
    - coherent write is ACK'ed after intervention complete on all CPUs
- IOCU can be configured to treat incoming writes as non-posted and provide a write ACK when they become visible.

Figure 8 I/O Coherent System



Software I/O Coherence

Taking advantage of hardware I/O coherence may require some redesign of existing systems that may not always be feasible. The CPUs and Coherence Manager are designed to efficiently support software-managed I/O coherence as well. This support is through the globalization of Hit-type CACHE instructions. When a coherent address is used for the CACHE operations, the CPU makes a corresponding coherent request. The CM will send interventions for the request to all of the CPUs, allowing all of the L1 caches to be maintained together. The basic software coherence routines developed for single CPU systems can be reused with minimal modifications.

L2 Cache Interface

When the CM is used with the SOC-it L2 cache controller, an optimized 256b interface is used. The L2 cache controller is able to access 256b at a time, so expanding the interface allows that cache bandwidth to be better utilized.

This interface is not a true 256b OCP interface. Because the CM-CPU interface and L2-memory interfaces are 64b OCP, converting to a true 256b OCP would add two additional alignment steps and increase latency. The key differences are:

- Data can be returned either 64b or 256b at a time. On L2 misses, the data will be returning from the system 64b at a time and can be passed directly back, rather than forcing the L2 to gather all 256b before responding.
- Data is aligned such that the critical dword of the original request is always in bits [63:0], which eases timing pressure when the critical dword bypasses the CM's read buffer.

This interface can be thought of as a 64b OCP interface, where all 4 beats of a burst can be returned in a single cycle.

System Features

Global Configuration Registers (GCR)

The Cluster includes a set of memory-mapped registers that are used to configure and control various aspects of the Coherence Manager and the coherence scheme.

Reset Control

The reset input from the system will reset the entire Cluster, including all 1004K CPUs, IOCU, and the Coherence Manager. After reset is released, the Coherence Manager, IOCU, and CPU0 will be active, but the remaining CPUs will continue to be held in reset. This allows CPU0 to initialize the system resources and perform the bringup in a controlled manner. Software must explicitly enable each of the other CPUs by writing to a reset register.

In addition to controlling the deassertion of the CPU reset signals, there are memory-mapped registers that can set the value for each CPU's *SI_ExceptionBase* pins. This allows different boot vectors to be specified for each of the CPUs so they can execute unique code if required.

Following reset, the caches are uninitialized and coherent requests should not be looked up in the cache. A coherence-enable register can be set after the caches have been flushed and the CPU is ready to start participating in the coherence scheme.

Inter-CPU Debug Breaks

The 1004K Cluster includes registers that enable cooperative debugging across all CPUs. Each VPE features an *EJ_DebugM* output that indicates it has entered debug mode (possibly through a debug breakpoint). Registers are defined that allow CPUs to be placed into debug groups such that whenever one CPU within the group enters debug mode, a debug interrupt is sent to all CPUs within the group, causing them to also enter debug mode and stop executing non-debug mode instructions.

CM Control

Registers in the GCR allow software to configure and control various aspects of the operation of the Coherence Manager. Some of the control options include:

- *Address map*: the base address for the GCR and GIC address ranges can be specified. An additional 4 address ranges can be defined as well. These control whether non-coherent requests go to memory or to memory-mapped I/O. A default can also be selected for addresses that do not fall within any range.
- *Error reporting and control*: Logs information about errors detected by the CM and controls how errors are handled (ignored, interrupt, etc.)
- *Control Options*: Various features of the CM can be disabled or configured. Examples of this are disabling speculative reads and preventing ReadShared requests from being upgraded to Exclusive.

Global Interrupt Controller

The Cluster includes an interrupt controller is included in. This block has the following features:

- Software interface through relocatable memory-mapped address range.
- Configurable number of system interrupts - from 8 to 256 in multiples of 8.
- Support for different interrupt types:
    - Level-sensitive - active high or low.
    - Edge-sensitive - positive, negative, or double edge-sensitive.
- Ability to mask and control routing of interrupts to a particular CPU and VPE.
- NMI and Yield Qualifier routing is also supported.
- Standardized mechanism for sending inter-processor interrupts.
- Support for External Interrupt Controller (EIC) mode.

Clock and Power Considerations

The following sections describe clocking and power management features.

Clocking

The CPU has 3 primary clock domains:

- Core domain - This is the main CPU clock domain, controlled by the *SI_ClkIn* clock input.
- OCP domain - This domain controls the OCP bus interface logic. Because the Coherence Manager runs at the same frequency as the CPU, this is effectively the same as the rest of the core domain.
- 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*.

The CM, IOCU, and all CPU cores must operate at the same frequency. Additionally, the TAP signals are daisy chained through each of the CPUs and thus will operate at the same frequency.

Clock ratios are supported at the following interfaces:

- Between the CM and L2
- Between the CM and memory if the L2 is not present
- Between the L2 and memory
- Between the IOCU and the I/O interconnect.

Clock ratios are established by enabling the input and output registers in the appropriate cycles to match up with the slower device. The two clocks are still expected to be synchronous even though they operate at different frequencies.

Power Management

The 1004K Cluster offers a number of power management features, including low-power design, active power management, and power-down modes of operation. The logic features a static design style that supports slowing or halting the clocks, which reduces system power consumption during idle periods.

Cluster Power Controller

Individual CPUs within the cluster can have their clock and/or power gated off when they are not in use. This gating is managed by the Cluster Power Controller (CPC). The CPC handles the power shutdown and ramp-up of CPUs in the Cluster. Any 1004K CPU supporting power-gating features will be managed by the CPC. The CPC also organizes power cycling of the CM dependent on the individual core status and shutdown policy. Reset and root level clock gating of individual CPUs is considered part of this sequencing.

Local clock gating

A significant portion of the power consumed by the 1004K CPU is often in the clock tree and clocking registers. The CPU has support for extensive use of local gated clocks. Power-conscious implementors can use these gated clocks to significantly reduce power consumption within the CPU.

Instruction-Controlled Power Management

The primary 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 (for example *SI_Int[5:0], SI_Int_1[5:0]*, *SI_NMI, SI_NMI_1*, *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 1004K CPU asserts the *SI_Sleep* signal, which is part of the system interface, whenever it has entered low-power operation and gone to sleep. It will enter sleep mode when all bus transactions are complete and all TCs are not running instructions. This happens when a TC is:

- Blocked due to a WAIT instruction
- Blocked due to an outstanding ITC operation
- Yielded
- Halted
- Not Active

When the CPU enters low-power operation, it will monitor the intervention interface and will wake up to service coherence requests. Upon completion of the intervention processing, the CPU can go back to sleep. For a deep-sleep mode, software can flush the caches and write to a GCR to take the CPU out of the coherence domain before entering sleep mode via the WAIT instruction. This will allow the CPU to stay asleep while other CPUs continue to process coherent data.

Test Capability

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

The core provides an integrated memory BIST solution for testing the internal cache SRAMs, the on-chip trace memory, and SPRAM using BIST controllers and logic tightly coupled to the cache subsystem. These BIST controllers can be configured to utilize the following algorithms: March C+ or IFA-13.

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.

User-specified BIST signals are also provided for the other data arrays that can be implemented with generator based SRAM cells in place of the standard registers

Build-Time Configuration Options

The 1004K 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 querying an appropriate register field. Refer to the *MIPS32 1004K CPU 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
System Options		
Number of CPUs	1,2,3,4	GCR_CONFIG_{PCORES}
I/O Coherence Unit	Present or not	$GCR\_CONFIG_{NUMIOCU}$
MConnID mask	0-8b	N/A
MIPS Trace support	Present or not	$Config3_{TL}$
MIPS Trace memory location	On-core, off-chip, or both	$TCBCONFIG_{OnT}$, $TCBCONFIG_{OfT}$
MIPS Trace on-chip memory size	256B - 1MB	$TCBCONFIG_{SZ}$
Probe Interface Block - Number of data pins	4,8,16	N/A
IOCU Options		
IODB implementation style	Flops or generator	N/A
Advanced IOCU Config - fine tuning of buffer sizes, etc.	Varies	N/A
Coherence Manager Options		
RWDB implementation style	Flops or generator	N/A
RDB implementation style	Flops or generator	N/A
Number of Address Regions	0,4,6	$GCR\_CONFIG_{NUM\_ADDR\_REGIONS}$
Default GCR base address & writeability	Any 32KB-aligned physical address Hardwired or programmable	*GCR_BASE*
Default Exception Base for each CPU	Any 4KB-aligned physical address	*GCR_Cx_RESET_BASE*
* These bits indicate the presence of external blocks. Bit will not be set if interface is present, but block is not.		

Table 4 Build-time Configuration Options (Continued)

Option	Choices	Software Visibility
Advanced CM Config - fine tuning of buffer sizes, etc.	Varies	N/A
Global Interrupt Controller Options		
Number of system interrupts	8*[1-32]	$GIC\_SH\_CONFIG_{NUMINTERRUPTS}$
Local routing of CPU sourced interrupts (per VPE)	Present or not	N/A
Local routing of CPU Timer Interrupt	Enabled or not	$GIC\_VPEj\_CTL_{TIMER\_ROUTABLE}$
Local routing of CPU Performance Counter Interrupt	Enabled or not	$GIC\_VPEj\_CTL_{PERFCOUNT\_ROUTABLE}$
Local routing of CPU Fast Debug Channel Interrupt	Enabled or not	$GIC\_VPEj\_CTL_{FDC\_ROUTABLE}$
Local routing of CPU Software Interrupts	Enabled or not	$GIC\_VPEj\_CTL_{SWINT\_ROUTABLE}$
ITU Options		
Number of single entry mailboxes	0,1,2,4,8,16	$ITCAddressMap1_{NumEntries}$
Number of 4 entry FIFOs	0,1,2,4,8,16	
Cluster Power Controller Options		
Microstep delay in cycles	1-1024	
RailEnable delay	1-1024	
Power Gating Enabled	Enabled or not	
Clock Gating Enabled	Enabled or not	
CPU Options		
Number of VPEs	1 or 2	$MVPConf0_{PVPE}$
Number of Shadow Register Sets (Includes required 1 per VPE)	1-8	$MVPConf0_{PTC}$
Number of TCs	1-9	$MVPConf0_{PTC}$
Integer register file implementation style	Flops or generator	N/A
Number of outstanding data cache misses	4 or 8	N/A
Number of outstanding Loads	4 or 9	N/A
Memory Management Type (per VPE)	TLB or FMT	$Config_{MT}$
TLB Size (per VPE)	16, 32, or 64 dual entries	$Config1_{MMUSize}$
TLB data array implementation style	Flops or generator	N/A

* These bits indicate the presence of external blocks. Bit will not be set if interface is present, but block is not.

Table 4 Build-time Configuration Options (Continued)

Option	Choices	Software Visibility
MIPS16e Support	Present or not	$Config1_{CA}$
DSP ASE Support	Present or not	$Config3_{DSPP}$
MDU	High Performance or Iterative	$Config_{MDU}$
Watch Registers	Present or Not	$Config1_{WR}$
UserLocal Register	Present or Not	$Config3_{ULRI}$
Performance Counters	Present or Not	$Config3_{PC}$
Branch Prediction	Dynamic or Static	$Config7_{BHT}$ $Config7_{RPS}$
Instruction Buffer Depth	8 or 6	none
L2 Cache Support	Present or Not	$Config2_{SL}$*
Instruction hardware breakpoints (per VPE)	0, 2, or 4	$DCRI_{B}$, IBS_{BCN}
Data hardware breakpoints (per VPE)	0, 1, or 2	DCR_{DB}, DBS_{BCN}
Fast Debug FIFO Sizes	Min (2Tx,2Rx), Useful(12Tx, 4Rx)	$FDCFG$
MIPS Trace triggers	0 - 8	$TCBCONFIG_{TRIG}$
FPU clock ratio relative to integer CPU	1:1 or 1:2	$Config7_{FPR}$
Coprocessor2 interface	Present or not	$Config1_{C2}$*
Data ScratchPad RAM interface	Present or not	$Config_{DSP}$*
Instruction ScratchPad RAM interface	Present or not	$Config_{ISP}$*
I-cache size	0,1,2,4, 8, 16, 32, or 64 KB	$Config1_{IL}$, $Config1_{IA}$, $Config1_{IS}$
D-cache size	1,2,4,8, 16, 32, or 64 KB	$Config1_{DL}$, $Config1_{DA}$, $Config1_{DS}$
Cache parity	Present or not	$ErrCtl_{PE}$
PrID Company Option	0x0-0x7f	$PrID_{CompanyOption}$
General Options (applicable to multiple blocks)		
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 external blocks. Bit will not be set if interface is present, but block is not.

Document Revision History

Change bars (vertical lines) in the margins of this document indicate significant changes in the document since its last release. Change bars are removed for changes that are more than one revision old. This document may refer to Architecture specifications (for example, instruction set descriptions and EJTAG register definitions), and change bars in these sections indicate changes since the previous version of the relevant Architecture document.

Table 5 Revision History

Revision	Date	Description
00.90	December 7, 2007	• Updated document for initial product release
01.00	June 23, 2008	• Updated details for GA release and incorporated more details on the processor core
01.01	July 29, 2008	• Update nomenclature.
01.10	July 15, 2009	• Added Cluster Power Controller • Support of heterogeneous core configurations • IT bypass added for single TC configurations • Support for on-chip trace memory
01.20	January 21, 2011	• Fixed maximum size of on-chip trace buffer (1MB) • Added number of area reduction options