E600COREPB
Rev. 0, 11/2004

e600 Core Product Brief

This product brief provides an overview of the e600 core features, including a block diagram showing the major functional components. The e600 is a PowerPC™ core.

This document also provides information about how the e600 implementation complies with the PowerPC and AltiVec™ architecture definitions.

1 e600 Core Overview

This section describes the features and general operation of the e600 core and provides a block diagram showing the major functional units. The e600 implements the PowerPC architecture and is a reduced instruction set computer (RISC) core. The e600 core consists of 32-Kbyte separate L1 instruction and data caches and a 1-Mbyte L2 cache. The core is a high-performance design supporting multiple execution units, including four independent units that execute AltiVec instructions.

The e600 core implements the 32-bit portion of the PowerPC architecture, which provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits. The core provides virtual memory support for up to 4 Petabytes (2^{52}) of virtual memory and real memory support for up to 64 Gigabytes (2^{36}) of physical memory.

The e600 core also implements the AltiVec instruction set architectural extension. The e600 core can dispatch and complete three instructions simultaneously. It incorporates the following execution units:

- 64-bit floating-point unit (FPU)
- Branch processing unit (BPU)
- Load/store unit (LSU)

- Four integer units (IUs):
  - Three shorter latency IUs (IU1a–IU1c)—execute all integer instructions except multiply, divide, and move to/from special-purpose register (SPR) instructions.
  - Longer latency IU (IU2)—executes miscellaneous instructions including condition register (CR) logical operations, integer multiplication and division instructions, and move to/from SPR instructions.
- Four vector units that support AltiVec instructions:
  - Vector permute unit (VPU)
  - Vector integer unit 1 (VIU1)—performs shorter latency integer calculations
  - Vector integer unit 2 (VIU2)—performs longer latency integer calculations
  - Vector floating-point unit (VFPU)

The ability to execute several instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for e600-based systems. Most integer instructions (including VIU1 instructions) have a one-clock cycle execution latency.

Several execution units feature multiple-stage pipelines; that is, the tasks they perform are broken into subtasks executed in successive stages. Typically, instructions follow one another through the stages, so a four-stage unit can work on four instructions when its pipeline is full. So, although an instruction may have to pass through several stages, the execution unit can achieve a throughput of one instruction per clock cycle.

AltiVec computational instructions are executed in four independent, pipelined AltiVec execution units. A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). This means an instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions. The VPU has a two-stage pipeline; the VIU2 and VFPU each have four-stage pipelines. As many as ten AltiVec instructions can be executing concurrently.

Note that for the e600 core, double- and single-precision versions of floating-point instructions have the same latency. For example, a floating-point multiply-add instruction takes 5 cycles to execute, regardless of whether it is single (**fmadds**) or double precision (**fmadd**).

The e600 core has independent on-chip, 32-Kbyte, eight-way set-associative, physically addressed L1 (level one) caches for instructions and data, and independent instruction and data memory management units (MMUs). Each MMU has a 128-entry, two-way set-associative translation lookaside buffer (DTLB and ITLB) that saves recently used page address translations. Block address translation is implemented with the eight-entry instruction and data block address translation (IBAT and DBAT) arrays defined by the PowerPC architecture. During block translation, effective addresses are compared simultaneously with all BAT entries, as described in Chapter 5, “Memory Management,” of the *MPC7450 RISC Microprocessor Family User’s Manual*. For information about the L1 caches, see Chapter 3, “L1, L2, and L3 Cache Operation,” of the *MPC7450 RISC Microprocessor Family User’s Manual*.

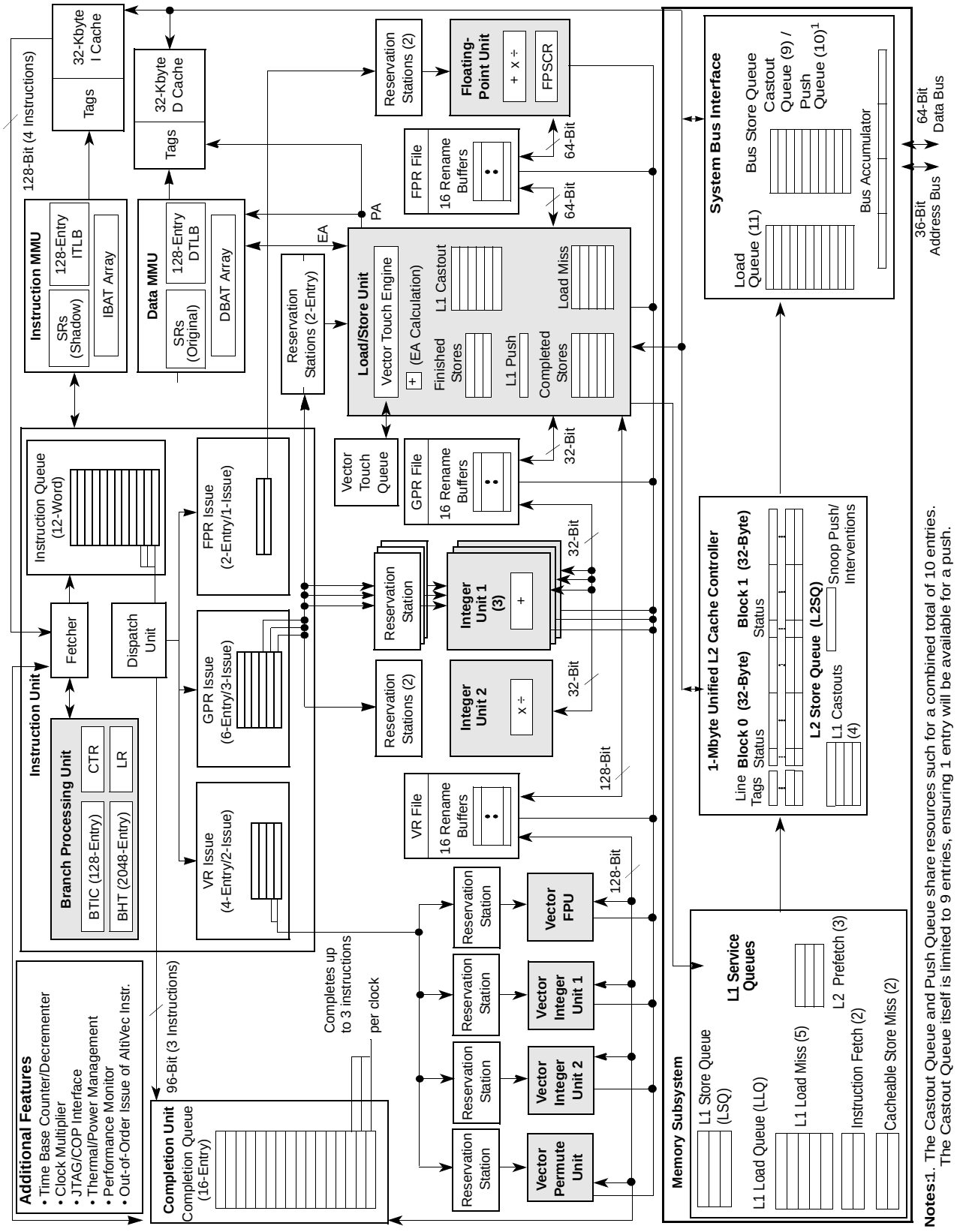
The L2 cache is implemented with an on-chip, 1-Mbyte, eight-way set-associative physically addressed memory available for storing data, instructions, or both. The L2 cache supports parity generation and checking for both tags and data. If ECC is disabled, it responds with an 11-cycle load latency for an L1 miss that hits in L2; if ECC is enabled, the L2 load access time is 12 cycles. The L2 cache is fully pipelined for two-cycle throughput. For information about the L2 cache implementation, see Chapter 3, “L1, L2, and L3 Cache Operation,” of the *MPC7450 RISC Microprocessor Family User’s Manual*.

The e600 core has three power-saving modes, nap, sleep, and deep sleep, which progressively reduce power dissipation. When functional units are idle, a dynamic power management mode causes those units to enter a low-power mode automatically without affecting operational performance, software execution, or external hardware. Section 2.9, “Power and Thermal Management,” describes how power management can be used to reduce

power consumption when the processor, or portions of it, are idle. It also describes how the instruction cache throttling mechanism reduces the instruction dispatch rate. The information in these sections are described more fully in Chapter 10, “Power and Thermal Management,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

The performance monitor facility provides the ability to monitor and count predefined events such as processor clocks, misses in the instruction cache, data cache, or L2 cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (which may be an approximation) can be used to trigger the performance monitor exception. Section 2.10, “Performance Monitor,” describes the operation of the performance monitor diagnostic tool. This functionality is fully described in Chapter 11, “Performance Monitor,” of the *MPC7450 RISC Microprocessor Family User’s Manual*

Figure 1 shows the parallel organization of the execution units (shaded in the diagram). Note that this is a conceptual model showing basic features, rather than an attempt at showing how features are implemented physically.



Notes:1. The Castout Queue and Push Queue share resources such for a combined total of 10 entries. The Castout Queue itself is limited to 9 entries, ensuring 1 entry will be available for a push.

Figure 1. e600 Core Block Diagram

2 e600 Core Features

This section describes the features of the e600 core. The interrelationships of these features are shown in Figure 1.

2.1 Features on the e600 Core

Major features of the e600 core are as follows:

- High-performance e600 core
  - As many as 4 instructions can be fetched from the instruction cache at a time.
  - As many as 3 instructions can be dispatched to the issue queues at a time.
  - As many as 12 instructions can be in the instruction queue (IQ).
  - As many as 16 instructions can be at some stage of execution simultaneously.
  - Single-cycle execution for most instructions
  - One-instruction throughput per clock cycle for most instructions
  - Seven-stage pipeline control
- Eleven independent execution units and three register files
  - Branch processing unit (BPU) features static and dynamic branch prediction
    - 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC), a cache of branch instructions that have been encountered in branch/loop code sequences. If a target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, a fetch that hits the BTIC provides the first 4 instructions in the target stream.
    - 2048-entry branch history table (BHT) with 2 bits per entry for four levels of prediction—not-taken, strongly not-taken, taken, strongly taken
    - Up to three outstanding speculative branches
    - Branch instructions that do not update the count register (CTR) or link register (LR) are often removed from the instruction stream.
    - Eight-entry link register stack to predict the target address of Branch Conditional to Link Register (**bclr**) instructions
  - Four integer units (IUs) that share 32 GPRs for integer operands
    - Three identical IUs (IU1a, IU1b, and IU1) can execute all integer instructions except multiply, divide, and move to/from special-purpose register instructions.
    - IU2 executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions.
  - 64-bit floating-point unit (FPU)
    - Five-stage FPU
    - Fully IEEE 754-1985 compliant FPU for both single- and double-precision operations
    - Supports non-IEEE mode for time-critical operations
    - Hardware support for denormalized numbers
    - Thirty-two 64-bit FPRs for single- or double-precision operands

  - Four vector units and 32-entry vector register file (VRs)
    - Vector permute unit (VPU)
    - Vector integer unit 1 (VIU1) handles short-latency AltiVec integer instructions, such as vector add instructions (for example, **vaddsbs**, **vaddshs**, and **vaddsws**)
    - Vector integer unit 2 (VIU2) handles longer-latency AltiVec integer instructions, such as vector multiply add instructions (for example, **vmhaddshs**, **vmhraddshs**, and **vmladduhm**).
    - Vector floating-point unit (VFPU)
  - Three-stage load/store unit (LSU)
    - Supports integer, floating-point and vector instruction load/store traffic
    - Four-entry vector touch queue (VTQ) supports all four architected AltiVec data stream operations
    - Three-cycle GPR and AltiVec load latency (byte, half word, word, vector) with single-cycle throughput
    - Four-cycle FPR load latency (single, double) with single-cycle throughput
    - No additional delay for misaligned access within double-word boundary
    - Dedicated adder calculates effective addresses (EAs)
    - Supports store gathering
    - Performs alignment, normalization, and precision conversion for floating-point data
    - Executes cache control and TLB instructions
    - Performs alignment, zero padding, and sign extension for integer data
    - Supports hits under misses (multiple outstanding misses)
    - Supports both big- and little-endian modes, including misaligned little-endian accesses
- Three issue queues, FIQ (floating point issue queue), VIQ (vector issue queue), and GIQ (general purpose issue queue), can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - Instructions can be dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - Space must be available in the completion queue (CQ) for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not in an issue queue).
- Rename buffers
  - 16 GPR (general purpose register) rename buffers
  - 16 FPR (floating point register) rename buffers
  - 16 VR (vector register) rename buffers
- Dispatch unit—The decode/dispatch stage fully decodes each instruction.
- Completion unit
  - The completion unit retires an instruction from the 16-entry CQ when all instructions ahead of it have been completed, the instruction has finished execution, and no exceptions are pending.
  - Guarantees sequential programming model (precise exception model)
  - Monitors all dispatched instructions and retires them in order
  - Tracks unresolved branches and flushes instructions after a mispredicted branch
  - Retires as many as three instructions per clock cycle

- L1 cache has the following characteristics:
  - Two separate 32-Kbyte instruction and data caches (Harvard architecture)
  - Instruction and data caches are eight-way set-associative
  - Instruction and data caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes—it corresponds to a cache line for the L1 data cache.
  - Cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
  - The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way.
  - Cache write-back or write-through operation is programmable on a per-page or per-block basis.
  - Instruction cache can provide four instructions per clock cycle; data cache can provide four words per clock cycle
    - Two-cycle latency and single-cycle throughput for instruction or data cache accesses
  - Caches can be disabled in software
  - Caches can be locked in software
  - Supports a four-state modified/exclusive/shared/invalid (MESI) coherency protocol
    - A single coherency status bit for each instruction cache block allows encoding for the following two possible states:

      Invalid (INV)

      Valid (VAL)
    - Two status bits (MESI[0–1]) for each data cache block allow encoding for coherency, as follows:

      00 = invalid (I)

      01 = shared (S)

      10 = exclusive (E)

      11 = modified (M)
  - Separate copy of data cache tags for efficient snooping
  - Both L1 caches support parity generation and checking (enabled through bits in the ICTRL register) as follows:
    - Instruction cache—one parity bit per instruction
    - Data cache—one parity bit per byte of data
  - No snooping of instruction cache except for **icbi** instruction
  - Caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way
  - Data cache supports AltiVec LRU and transient instructions
  - Critical double- and/or quad-word forwarding is performed as needed. Critical quad-word forwarding is used for AltiVec loads and instruction fetches. Other accesses use critical double-word forwarding.
- On-chip level 2 (L2) cache has the following features:
  - Integrated 1-Mbyte, eight-way set-associative unified instruction and data cache
  - Fully pipelined to provide 32 bytes every other clock cycle to the L1 caches
  - Total latency of 11 processor cycles for L1 data cache miss that hits in the L2 with ECC disabled, 12 cycles when ECC is enabled
  - Uses one of two random replacement algorithms (selectable through L2CR)
  - Cache write-back or write-through operation programmable on a per-page or per-block basis

  - Organized as 32 bytes/block and 2 blocks (sectors)/line (a cache block is the block of memory that a coherency state describes).
  - Supports error correction and detection using a SECDED (single-error correction, double-error detection) protocol. Every 64 bits of data comes with 8 bits of error detection/correction, which can be programmed as ECC across the 64 bits of data, byte parity, or no error detection/correction.
  - Supports parity generation and checking for both tags and data (enabled through L2CR). Tag parity is enabled separately in the L2ERRDIS register, and data parity can be enabled through L2CR only when ECC is disabled.
  - Error injection modes provided for testing
- Separate memory management units (MMUs) for instructions and data
  - 52-bit virtual address; 32- or 36-bit physical address
  - Address translation for 4-Kbyte pages, variable-sized blocks, and 256-Mbyte segments
  - Memory programmable as write-back/write-through, caching-inhibited/caching-allowed, and memory coherency enforced/memory coherency not enforced on a page or block basis
  - Separate IBATs and DBATs (eight each) also defined as SPRs
  - Separate instruction and data translation lookaside buffers (TLBs)
    - Both TLBs are 128-entry, two-way set-associative, and use LRU replacement algorithm
    - TLBs are hardware or software reloadable (that is, on a TLB miss a page table search is performed in hardware or by system software).
- Efficient data flow
  - Although the VR/LSU interface is 128 bits, the L1/L2 bus interface allows up to 256 bits.
  - The L1 data cache is fully pipelined to provide 128 bits/cycle to or from the VRs.
  - L2 cache is fully pipelined to provide 32 bytes every other processor clock cycle to the L1 cache
  - As many as eight outstanding, out-of-order cache misses are allowed between the L1 data cache and L2 bus.
  - As many as 16 out-of-order transactions can be present on the MPX bus.
  - Store merging for multiple store misses to the same line. Only coherency action taken (address-only) for store misses merged to all 32 bytes of a cache block (no data tenure needed)
  - Support for a second cacheable store miss
  - Three-entry finished store queue and five-entry completed store queue between the LSU and the L1 data cache
  - Separate additional queues for efficient buffering of outbound data (such as castouts and write-through stores) from the L1 data cache and L2 cache
- Multiprocessing support features include the following:
  - Hardware-enforced, MESI cache coherency protocols for data cache
  - Load/store with reservation instruction pair for atomic memory references, semaphores, and other multiprocessor operations
- Power and thermal management
  - The following three power-saving modes are available to the system:
    - Nap—Instruction fetching is halted. Only those clocks for the time base, decrementer, and JTAG logic remain running. The part goes into the doze state to snoop memory operations on the bus and then back to nap using a $\overline{\text{QREQ}}$/$\overline{\text{QACK}}$ processor-system handshake protocol.

  - Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.
  - Deep sleep—When the part is in deep sleep state, the system can disable the PLL. The system can then disable the SYSCLK source for greater system power savings. Power-on reset procedures for restarting and relocking the PLL must be followed upon exiting the deep sleep state.
- DFS (dynamic frequency switching) capability conserves power by lowering processor operating frequency. Divide-by-two and divide-by-four modes (DFS2 and DFS4) provided.
- Instruction cache throttling provides control of instruction fetching to limit device temperature.

- Performance monitor helps to debug system designs and improve software efficiency
- In-system testability and debugging features through JTAG boundary-scan capability
- Reliability and serviceability
  - Parity checking on system bus
  - Parity checking on L1 and L2 cache arrays

2.2 Instruction Flow

As shown in Figure 1, the e600 core instruction unit provides centralized control of instruction flow to the execution units. The instruction unit contains a sequential fetcher, 12-entry instruction queue (IQ), dispatch unit, and branch processing unit (BPU). It determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

See Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual* for a detailed discussion of instruction timing.

The sequential fetcher loads instructions from the instruction cache into the instruction queue. The BPU extracts branch instructions from the sequential fetcher. Branch instructions that cannot be resolved immediately are predicted using either e600-specific dynamic branch prediction or architecture-defined static branch prediction.

Branch instructions that do not affect the LR or CTR are often removed from the instruction stream. Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual* describes when a branch can be removed from the instruction stream.

Instructions dispatched beyond a predicted branch do not complete execution until the branch is resolved, preserving the programming model of sequential execution. If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are fetched from the correct path.

2.2.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 1 holds as many as 12 instructions and loads as many as 4 instructions from the instruction cache during a single processor clock cycle.

The fetcher attempts to initiate a new fetch every cycle. The two fetch stages are pipelined, so as many as four instructions can arrive to the IQ every cycle. All instructions except branch (**b***x*), Return from Exception (**rfi**), System Call (**sc**), Instruction Synchronize (**isync**), and no-op instructions are dispatched to their respective issue queues from the bottom three positions in the instruction queue (IQ0–IQ2) at a maximum rate of three instructions per clock cycle. Reservation stations are provided for the three IU1s, IU2, FPU, LSU, VPU, VIU2, VIU1, and VFPU. The dispatch unit checks for source and destination register dependencies, determines whether a position is available in the CQ, and inhibits subsequent instruction dispatching as required.

Branch instruction can be detected, decoded, and predicted from entries IQ0–IQ7.

2.2.2 Branch Processing Unit (BPU)

The BPU receives branch instructions from the IQ and executes them early in the pipeline, achieving the effect of a zero-cycle branch in some cases.

Branches with no outstanding dependencies (CR, LR, or CTR unresolved) can be processed and resolved immediately. For branches in which only the direction is unresolved due to a CR or CTR dependency, the branch path is predicted using either architecture-defined static branch prediction or e600-specific dynamic branch prediction. Dynamic branch prediction is enabled if HID0[BHT] is set. For **bclr** branches where the target address is unresolved due to a LR dependency, the branch target can be predicted using the hardware link stack. Link stack prediction is enabled if HID0[LRSTK] is set.

When a prediction is made, instruction fetching, dispatching, and execution continue from the predicted path, but instructions cannot complete and write back results to architected registers until the prediction is determined to be correct (resolved). When a prediction is incorrect, the instructions from the incorrect path are flushed from the processor and processing begins from the correct path.

Dynamic prediction is implemented using a 2048-entry branch history table (BHT), a cache that provides two bits per entry that together indicate four levels of prediction for a branch instruction—not-taken, strongly not-taken, taken, strongly taken. When dynamic branch prediction is disabled, the BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the e600 core executes instructions from the predicted target stream although the results are not committed to architected registers until the conditional branch is resolved. Unresolved branches are held in a three-entry branch queue. When the branch queue is full, no further conditional branches can be processed until one of the conditions in the branch queue is resolved.

When a branch is taken or predicted as taken, instructions from the untaken path must be flushed and the target instruction stream must be fetched into the IQ. The BTIC is a 128-entry, four-way set associative cache that contains the most recently used branch target instructions (up to four instructions per entry) for **b** and **bc** branches. When a taken branch instruction of this type hits in the BTIC, the instructions arrive in the IQ two clock cycles later, a clock cycle sooner than they would arrive from the instruction cache. Additional instructions arrive from the instruction cache in the next clock cycle. The BTIC reduces the number of missed opportunities to dispatch instructions and gives the processor a 1-cycle head start on processing the target stream.

The BPU contains an adder to compute branch target addresses and three user-accessible registers—the link register (LR), the count register (CTR), and the condition register (CR). The BPU calculates the return pointer for subroutine calls and saves it in the LR for certain types of branch instructions. The LR also contains the branch target address for Branch Conditional to Link Register (**bclr***x*) instructions. The CTR contains the branch target address for Branch Conditional to Count Register (**bcctr***x*) instructions. Because the LR and CTR are SPRs, their contents can be copied to or from any GPR. Also, because the BPU uses dedicated registers rather than GPRs or FPRs, execution of branch instructions is largely independent from execution of integer and floating-point instructions.

2.2.3 Completion Unit

The completion unit operates closely with the instruction unit. Instructions are fetched and dispatched in program order. At the point of dispatch, the program order is maintained by assigning each dispatched instruction a successive entry in the 16-entry CQ. The completion unit tracks instructions from dispatch through execution and retires them in program order from the three bottom CQ entries (CQ0–CQ2).

Instructions cannot be dispatched to an execution unit unless there is a CQ vacancy.

Branch instructions that do not update the CTR or LR are often removed from the instruction stream. Those that are removed do not take a CQ entry. Branches that are not removed from the instruction stream follow the same dispatch and completion procedures as non-branch instructions but are not dispatched to an issue queue.

Completing an instruction commits execution results to architected registers (GPRs, FPRs, VRs, LR, and CTR). In-order completion ensures the correct architectural state when the e600 core must recover from a mispredicted branch or any exception. An instruction is retired as it is removed from the CQ.

For a more detailed discussion of instruction completion, see Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual*.

2.2.4 Independent Execution Units

In addition to the BPU, the e600 core provides the ten execution units described in the following sections.

2.2.4.1 AltiVec Vector Permute Unit (VPU)

The VPU executes permutation instructions such as pack, unpack, merge, splat, and permute on vector operands.

2.2.4.2 AltiVec Vector Integer Unit 1 (VIU1)

The VIU1 executes simple vector integer computational instructions, such as addition, subtraction, maximum and minimum comparisons, averaging, rotation, shifting, comparisons, and Boolean operations.

2.2.4.3 AltiVec Vector Integer Unit 2 (VIU2)

The VIU2 executes longer-latency vector integer instructions, such as multiplication, multiplication/addition, and sum-across with saturation.

2.2.4.4 AltiVec Vector Floating-Point Unit (VFPU)

The VFPU executes all vector floating-point instructions.

A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

2.2.4.5 Integer Units (IUs)

The integer units (three IU1s and IU2) are shown in Figure 1. The IU1s execute shorter latency integer instructions, that is, all integer instructions except multiply, divide, and move to/from special-purpose register instructions. IU2 executes integer instructions with latencies of three cycles or more.

IU2 has a 32-bit integer multiplier/divider and a unit for executing CR logical operations and move to/from SPR instructions. The multiplier supports early exit for operations that do not require full 32×32-bit multiplication.

2.2.4.6 Floating-Point Unit (FPU)

The FPU, shown in Figure 1, is designed such that double-precision operations require only a single pass, with a latency of 5 cycles. As instructions are dispatched to the FPUs reservation station, source operand data can be

accessed from the FPRs or from the FPR rename buffers. Results in turn are written to the rename buffers and made available to subsequent instructions. Instructions start execution from the bottom reservation station only and execute in program order.

The FPU contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the e600 core to implement multiply and multiply-add operations efficiently. The FPU is pipelined so that one single- or double-precision instruction can be issued per clock cycle.

Note that an execution bubble occurs after four consecutive, independent floating-point arithmetic instructions execute to allow for a normalization special case. Thirty-two 64-bit floating-point registers are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by automatic allocation of the 16 floating-point rename registers. The e600 core writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The e600 core supports all IEEE 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines.

2.2.4.7 Load/Store Unit (LSU)

The LSU executes all load and store instructions as well as the AltiVec LRU and transient instructions and provides the data transfer interface between the GPRs, FPRs, VRs, and the cache/memory subsystem. The LSU also calculates effective addresses and aligns data.

Load and store instructions are issued and translated in program order; however, some memory accesses can occur out of order. Synchronizing instructions can be used to enforce strict ordering. When there are no data dependencies and the guarded bit for the page or block is cleared, a maximum of one out-of-order cacheable load operation can execute per clock cycle from the perspective of the LSU. Loads to FPRs require a 4-cycle total latency. Data returned from the cache is held in a rename register until the completion logic commits the value to a GPR, FPR, or VR. Stores cannot be executed out of order and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The e600 core executes store instructions with a maximum throughput of one per clock cycle and a 3-cycle total latency to the data cache. The time required to perform the load or store operation depends on the processor: bus clock ratio and whether the operation involves the on-chip caches, system memory, or an I/O device.

2.3 Memory Management Units (MMUs)

The e600 core MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 64 Gigabytes (2^{36}) of physical memory for instructions and data. The MMUs control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to support demand-paged virtual memory systems. The memory management units are contained within the load/store unit.

The LSU calculates effective addresses for data loads and stores; the instruction unit calculates effective addresses for instruction fetching. The MMU translates the effective address to determine the correct physical address for the memory access.

The e600 core supports the following types of memory translation:

- Real addressing mode—In this mode, translation is disabled by clearing bits in the machine state register (MSR): MSR[IR] for instruction fetching or MSR[DR] for data accesses. When address translation is disabled, the physical address is identical to the effective address. When extended addressing is disabled (HID0[XAEN] = 0) a 32-bit physical address is used, PA[4–35]. For more details see Chapter 5, “Memory Management Unit,” of the *MPC7450 RISC Microprocessor Family User’s Manual*.

- Page address translation—translates the page frame address for a 4-Kbyte page size
- Block address translation—translates the base address for blocks (4 Gbytes)

If translation is enabled, the appropriate MMU translates the higher-order bits of the effective address into physical address bits. Lower-order address bits are untranslated and are the same for both logical and physical addresses. These bits are directed to the on-chip caches where they form the index into the eight-way set-associative tag array. After translating the address, the MMU passes the higher-order physical address bits to the cache and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits; the resulting 32- or 36-bit physical address is used by the memory subsystem and the bus interface unit, which accesses external memory.

The TLBs store page address translations for recent memory accesses. For each access, an effective address is presented for page and block translation simultaneously. If a translation is found in both the TLB and the BAT array, the block address translation in the BAT array is used. Usually the translation is in a TLB and the physical address is readily available to the on-chip cache. When a page address translation is not in a TLB, hardware or system software searches for one in the page table following the model defined by the PowerPC architecture.

Instruction and data TLBs provide address translation in parallel with the on-chip cache access, incurring no additional time penalty in the event of a TLB hit. The e600 core instruction and data TLBs are 128-entry, two-way set-associative caches that contain address translations. The core can initiate a hardware or system software search of the page tables in memory on a TLB miss.

2.4 On-Chip L1 Instruction and Data Caches

The e600 core implements separate L1 instruction and data caches. Each cache is 32-Kbyte eight-way set-associative. As defined by the PowerPC architecture, they are physically indexed. Each cache block contains eight contiguous words from memory that are loaded from an eight-word boundary (that is, bits EA[27–31] are zeros); thus, a cache block never crosses a page boundary. An entire cache block can be updated by a four-beat burst load across a 64-bit system bus. Misaligned accesses across a page boundary can incur a performance penalty. The data cache is a non-blocking, write-back cache with hardware support for reloading on cache misses. The critical double word is transferred on the first beat and is forwarded to the requesting unit, minimizing stalls due to load delays. For vector loads, the critical quad word is handled similarly but is transferred on the second beat. The cache being loaded is not blocked to internal accesses while the load completes.

The e600 core L1 cache organization is shown in Figure 2.

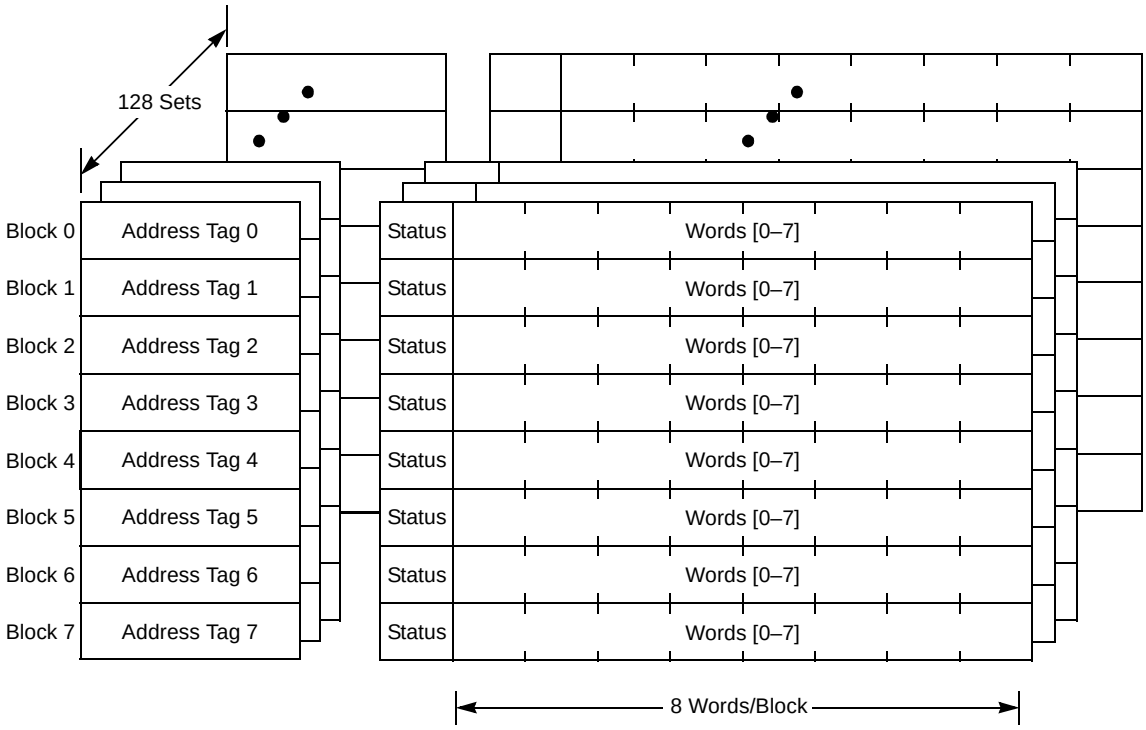


Figure 2. L1 Cache Organization

The instruction cache provides up to four instructions per clock cycle to the instruction queue. The instruction cache can be invalidated entirely or on a cache-block basis. It is invalidated and disabled by setting HID0[ICFI] and then clearing HID0[ICE]. The instruction cache can be locked by setting HID0[ILOCK]. The instruction cache supports only the valid/invalid states.

The data cache provides four words per clock cycle to the LSU. Like the instruction cache, the data cache can be invalidated all at once or on a per-cache-block basis. The data cache can be invalidated and disabled by setting HID0[DCFI] and then clearing HID0[DCE]. The data cache can be locked by setting HID0[DLOCK]. The data cache tags are dual-ported, so a load or store can occur simultaneously with a snoop.

The e600 core also implements a 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC). The BTIC is a cache of branch instructions that have been encountered in branch/loop code sequences. If the target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, the BTIC contains the first four instructions in the target stream.

The BTIC can be disabled and invalidated through software. As with other aspects of e600 core instruction timing, BTIC operation is optimized for cache-line alignment. If the first target instruction is one of the first five instructions in the cache block, the BTIC entry holds four instructions. If the first target instruction is the last instruction before the cache block boundary, it is the only instruction in the corresponding BTIC entry. If the next-to-last instruction in a cache block is the target, the BTIC entry holds two valid target instructions, as shown in Figure 3.

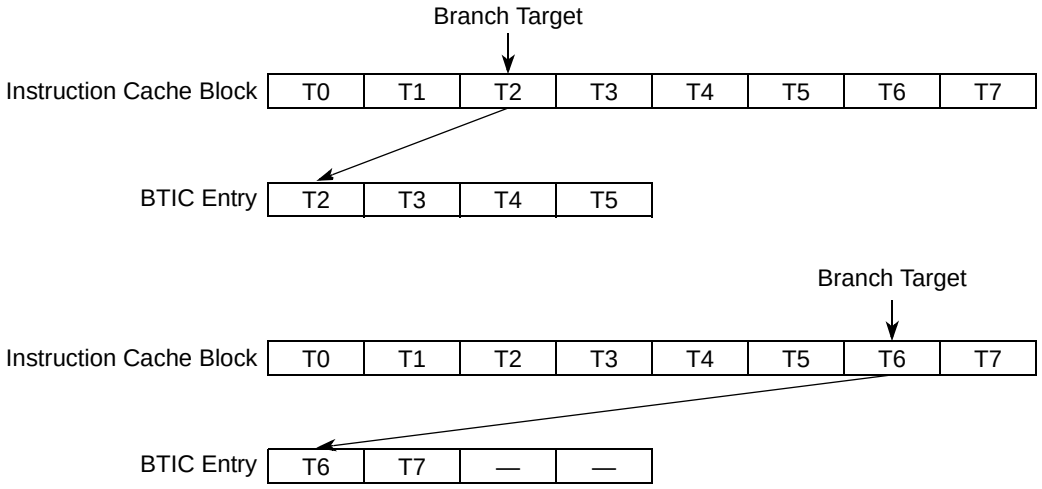


Figure 3. Alignment of Target Instructions in the BTIC

BTIC ways are updated using a FIFO algorithm.

For more information and timing examples showing cache hit and cache miss latencies, see Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual*.

2.5 L2 Cache Implementation

The L2 cache is a unified cache that receives memory requests from both the L1 instruction and data caches independently. The integrated L2 cache on the e600 core is a unified (containing both instructions and data) 1-Mbyte on-chip cache. It is eight-way set-associative and organized with 32-byte blocks and two blocks/line.

Each line consists of 64 bytes of data organized as two blocks (also called sectors). Although all 16 words in a cache line share the same address tag, each block maintains the three separate status bits for the 8 words of the cache block, the unit of memory at which coherency is maintained. Thus, each cache line can contain 16 contiguous words from memory that are read or written as 8-word operations.

The integrated L2 cache organization of the e600 core is shown in Figure 4.

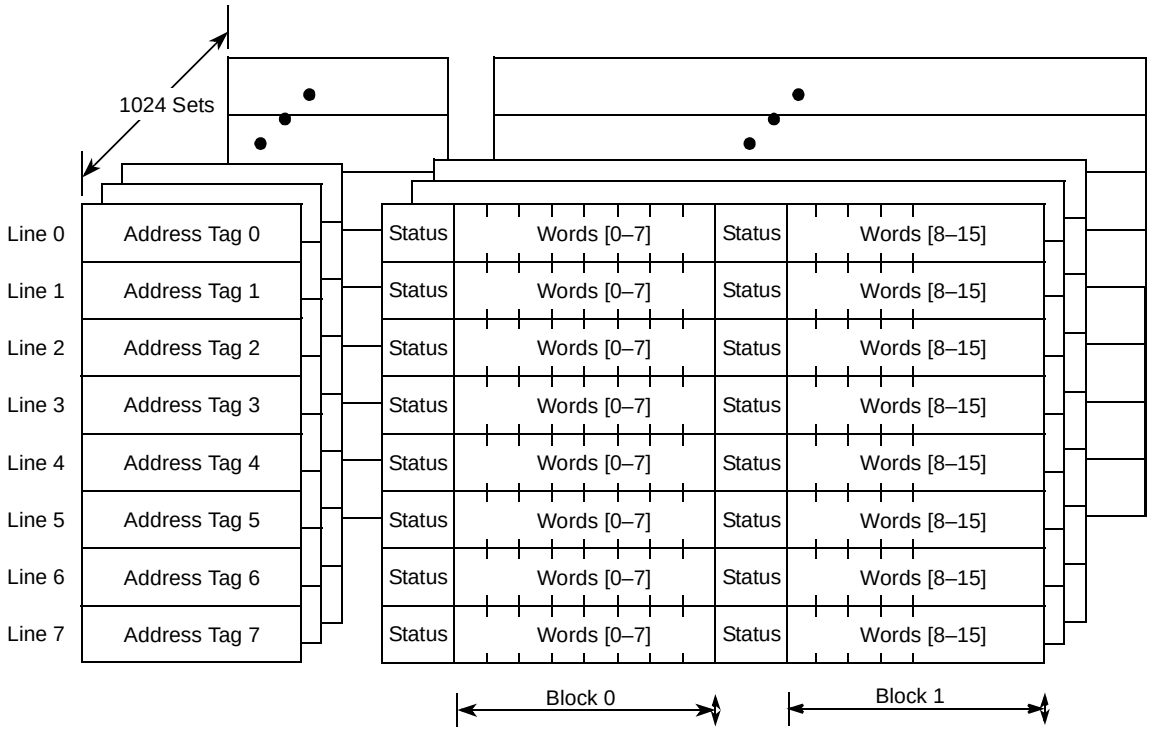


Figure 4. L2 Cache Organization

The L2 cache controller contains the L2 cache control register (L2CR), which does the following:

- Includes bits for enabling parity checking on the L2
- Provides for instruction-only and data-only modes
- Provides hardware flushing for the L2
- Selects between two available replacement algorithms for the L2 cache

The L2 implements the MESI cache coherency protocol using three status bits per sector.

Requests from the L1 cache generally result from instruction misses, data load or store misses, write-through operations, or cache management instructions. Requests from the L1 cache are compared against the L2 tags and serviced by the L2 cache if they hit; if they miss in the L2 cache, they go to main memory.

The L2 cache tags are fully pipelined and non-blocking for efficient operation. Thus the L2 cache can be accessed internally while a load for a miss is pending (allowing hits under misses). A reload for a cache miss is treated as a normal access and blocks other accesses for only 1 cycle.

For more information, see Chapter 3, “L1, L2, and L3 Cache Operation,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

2.6 System Interface

The e600 core supports two interface protocols—the MPX bus protocol and a subset of the 60x bus protocol. Note that although this protocol is implemented by the MPC603e, MPC604e, MPC740, and MPC750 processors, it is referred to as the 60x bus interface. The MPX bus protocol is derived from the 60x bus protocol. The MPX bus interface includes several additional features that provide higher memory bandwidth than the 60x bus and more

efficient use of the system bus in a multiprocessing environment. Because the performance of the e600 core is optimized for the MPX bus, use of the MPX is recommended over use of the the 60x bus.

The e600 bus interface includes a 64-bit data bus with 8 bits of data parity, a 36-bit address bus with 5 bits of address parity, and additional control signals to allow for unique system level optimizations.

The bus interface protocol is configured using the $\overline{\text{BMODE0}}$ configuration signal at reset. If $\overline{\text{BMODE0}}$ is asserted at the negation of $\overline{\text{HRESET}}$, the e600 core uses the MPX bus protocol; if $\overline{\text{BMODE0}}$ is negated during the negation of $\overline{\text{HRESET}}$, the e600 core uses a limited subset of the 60x bus protocol. Note that the inverse state of $\overline{\text{BMODE}}$[0:1] at the negation of $\overline{\text{HRESET}}$ is saved in MSSCR0[BMODE].

2.7 e600 Core Bus Operation Features

The e600 core has a separate address and data bus, each with its own set of arbitration and control signals. This allows for decoupling the data tenure from the address tenure of a transaction and provides for a wide range of system-bus implementations including:

- Non-pipelined bus operation
- Pipelined bus operation
- Split transaction operation

The e600 core supports only the normal memory-mapped address segments defined in the PowerPC architecture. Access to direct store segments results in a DSI exception.

2.7.1 MPX Bus Features

The MPX bus has the following features:

- Extended 36-bit address bus plus 5 bits of odd parity (41 bits total)
- 64-bit data bus plus 8 bits of odd parity (72 bits total); a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- On-chip snooping to maintain L1 data cache and L2 cache coherency for multiprocessing applications and DMA environments
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 16 out-of-order transactions using 4 data transaction index (DTI[0:3]) signals
- Full data streaming
- Support for data intervention in multiprocessor systems

2.7.2 60x Bus Features

The following list summarizes the 60x bus interface features:

- Extended 36-bit address bus plus 5 bits of odd parity (41 bits total)
- 64-bit data bus plus 8 bits of odd parity (72 bits total); a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- On-chip snooping to maintain L1 data cache and L2 cache coherency for multiprocessing applications and DMA environments

- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 16 outstanding transactions. No reordering is supported.

2.8 Overview of System Interface Accesses

The system interface includes address register queues, prioritization logic, and a bus control unit. The system interface latches snoop addresses for snooping in the L1 data cache and the L2 cache, the memory hierarchy address register queues, and the reservation controlled by the Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions. Accesses are prioritized with load operations preceding store operations.

Instructions are automatically fetched from the memory system into the instruction unit where they are issued to the execution units at a peak rate of three instructions per clock cycle. Conversely, load and store instructions explicitly specify the movement of operands to and from the integer, floating-point, and AltiVec register files and the memory system.

When the e600 core encounters an instruction or data access, it calculates the effective address and uses the lower-order address bits to check for a hit in the on-chip, 32-Kbyte L1 instruction and data caches. During L1 cache lookup, the instruction and data memory management units (MMUs) use the higher-order address bits to calculate the virtual address, from which they calculate the physical (real) address. The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred in the L1 instruction or data cache. If the access misses in the corresponding cache, the transaction is sent to L1 load miss queue or the L1 store miss queue. L1 load miss queue transactions are sent to the internal 1-Mbyte L2 cache. Store miss queue transactions are queued up in the L2 cache controller. If no match is found in the L2 tags, the physical address is used to access system memory.

In addition to loads, stores, and instruction fetches, the e600 core performs hardware table search operations following TLB misses; L1 and L2 cache castout operations; and cache-line snoop push operations when a modified cache line detects a snoop hit from another bus master.

2.8.1 System Interface Operation

The primary activity of the e600 core system interface is transferring data and instructions between the processor and system memory. There are three types of transfer accesses:

- Single-beat transfers—These memory accesses allow transfer sizes of 1, 2, 3, 4, or 8 bytes in one bus clock cycle. Single-beat transactions are caused by uncacheable read and write operations that access memory directly (that is, when caching is disabled), cache-inhibited accesses, and stores in write-through mode.
- Two-beat burst (16-byte) data transfers—Generated to support caching-inhibited or write-through AltiVec loads and stores (only generated in MPX bus mode) and for caching-inhibited instruction fetches in MPX mode.
- Four-beat burst (32-byte) data transfers—Initiated when an entire cache block is transferred into or out of the internal caches. Because the first-level caches on the e600 core are write-back caches, burst-read memory operations are the most common memory accesses, followed by burst-write memory operations, and single-beat (caching-inhibited or write-through) memory read and write operations.

Memory accesses can occur in single-beat (1, 2, 3, 4, and 8 bytes), double-beat (16 bytes), and four-beat (32 bytes) burst data transfers. For memory accesses, the address and data buses are independent to support pipelining and split transactions. The bus interface can pipeline as many as 16 transactions and, in MPX bus mode, supports full

out-of-order split-bus transactions. The e600 core bursts out of reset in MPX bus mode, fetching eight instructions on the MPX bus at a time.

Access to the system interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the e600 core to be integrated into systems that implement various fairness and bus-parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered to maximize the efficiency of the bus without sacrificing coherency of the data. The e600 core allows load operations to bypass store operations (except when a dependency exists). Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

Note that the synchronize (**sync**) and enforce in-order execution of I/O (**eieio**) instructions can be used to enforce strong ordering.

The system interface is synchronous. All e600 core inputs are sampled and all outputs are driven on the rising edge of the bus clock cycle. The hardware specifications gives timing information. The system interface is specific for each microprocessor that implements the PowerPC architecture.

2.8.2 Signal Groupings

Signals are provided for implementing the bus protocol and clocking. Test and control signals provide diagnostics for selected internal circuits.

The e600 core MPX and 60x bus interface protocol signals are grouped as follows:

- Address arbitration—The e600 core uses these signals to arbitrate for address bus mastership.
- Address transfer start—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer—These signals include the address bus and address parity signals. They are used to transfer the address and to ensure the integrity of the transfer.
- Transfer attribute—These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or cache-inhibited.
- Address transfer termination—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.
- Data arbitration—The e600 core uses these signals to arbitrate for data bus mastership.
- Data transfer—These signals, which consist of the data bus and data parity signals, are used to transfer the data and to ensure the integrity of the transfer.
- Data transfer termination—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, data termination signals also indicate the end of the tenure. In burst accesses, data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. Data termination signals also indicate whether a condition exists that requires the data phase to be repeated.

Many other e600 core signals control and affect other aspects of the device, aside from the bus protocol. They are as follows:

- Interrupts/resets—These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the processor.
- Processor status and control—These signals enable the time-base facility and are used to select the bus mode and control sleep mode.
- Clock control—These signals determine the system clock frequency. They are also used to synchronize multiprocessor systems.

- Test interface—The JTAG (IEEE 1149.1a-1993) interface and the common on-chip processor (COP) unit provide a serial interface to the system for performing board-level boundary-scan interconnect tests.
- Voltage selection—These signal control the electrical characteristics of the I/O circuitry of the device as appropriate to support various signaling levels.

NOTE

Active-low signals are shown with overbars. For example, $\overline{\text{ARTRY}}$ (address retry) and $\overline{\text{TS}}$ (transfer start). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as AP[0:4] (address bus parity signals) and TT[0:4] (transfer type signals) are referred to as asserted when they are high and negated when they are low.

2.8.3 MPX Bus Mode Functional Groupings

Figure 5 illustrates the signal configuration in MPX bus mode for the e600 core, showing how the signals are grouped. A pinout diagram and tables showing pin numbers are included in the hardware specifications. Note that the left side of the figure depicts the signals that implement the MPX bus protocol and the right side of the figure shows the remaining signals on the e600 core (not part of the bus protocol).

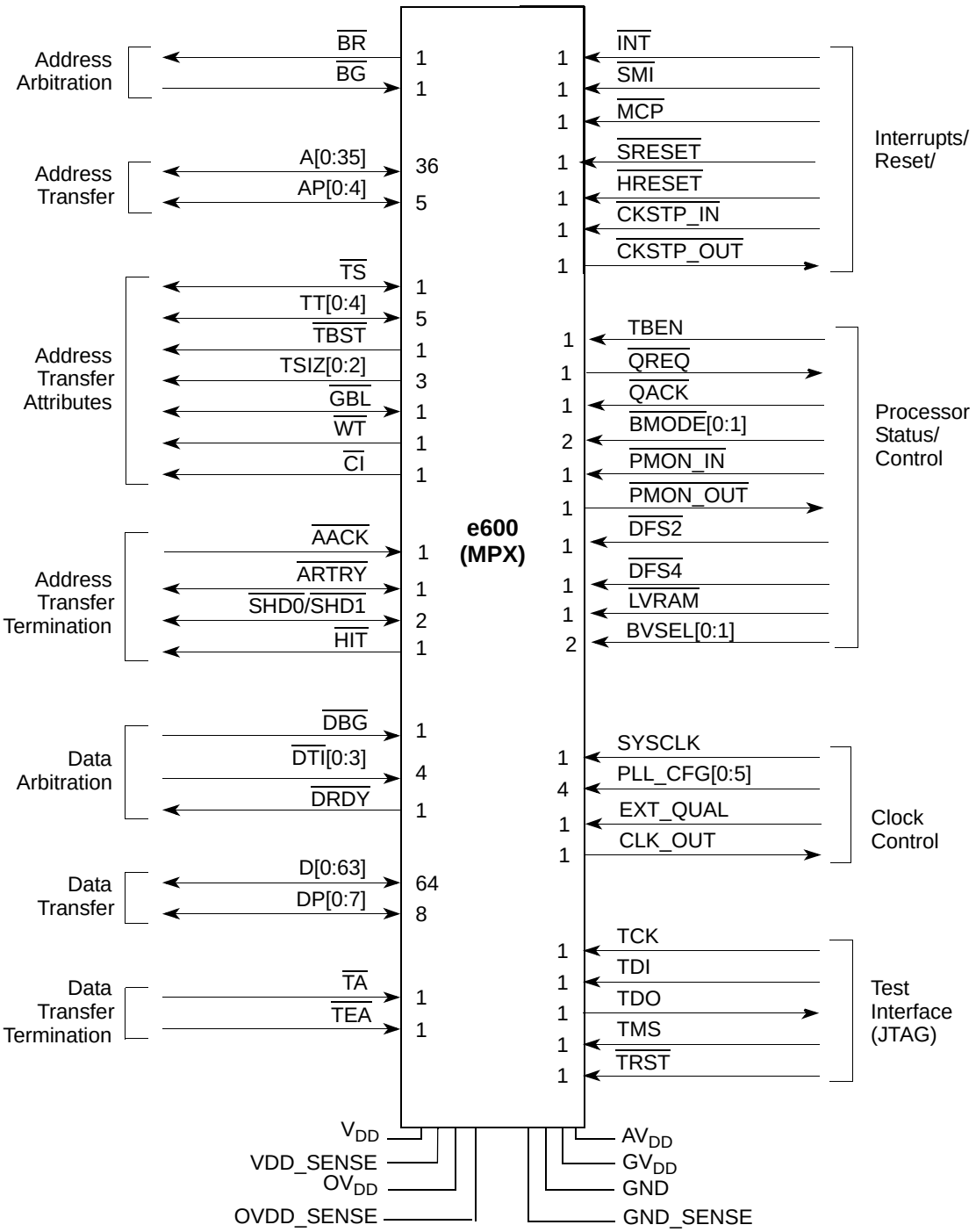


Figure 5. MPX Bus Signal Groups in the e600 Core

Signal functionality is described in detail in Chapter 8, “Signal Descriptions,” and Chapter 9, “System Interface Operation,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

2.8.3.1 Clocking

For functional operation, the e600 core uses a single clock input signal, SYSCLK, from which clocking is derived for the processor core and the MPX bus interface. Additionally, internal clock information is made available at the pins to support debug and development.

The e600 core clocking structure supports a wide range of processor-to-bus clock ratios. The internal processor core clock is synchronized to SYSCLK with the aid of a VCO-based PLL. The PLL_CFG[0:5] signals are used to program the internal clock rate to a multiple of SYSCLK as defined in the hardware specifications. The bus clock is maintained at the same frequency as SYSCLK. SYSCLK does not need to be a 50% duty-cycle signal.

2.9 Power and Thermal Management

The e600 core is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. If an e600 core functional unit is idle, it automatically goes into a low-power mode. This mode does not affect operational performance. Dynamic power management automatically supplies or withholds power to execution units individually, based upon the contents of the instruction stream. The operation of dynamic power management is transparent to software or any external hardware.

The following three programmable power modes are available to the system:

- Nap—Instruction fetching is halted. Only those clocks for time base, decrementer, and JTAG logic remain running. The e600 core goes into the doze state to snoop memory operations on the bus and then back to nap using a $\overline{\text{QREQ}}$/$\overline{\text{QACK}}$ processor-system handshake protocol.
- Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.
- Deep sleep—The system can disable the PLL. The system can then disable the SYSCLK source for greater system power savings. Power-on reset procedures for restarting and relocking the PLL must be followed upon exiting deep sleep.

The dynamic frequency switching (DFS) feature in the e600 core conserves power by lowering processor operating frequency. The e600 has the ability to divide the processor-to-system bus ratio by two or four during normal functional operation. Chapter 10, “Power and Thermal Management,” of the *MPC7450 RISC Microprocessor Family User’s Manual* provides information on power saving with DFS.

The e600 core also provides an instruction cache throttling mechanism to effectively reduce the instruction execution rate without the complexity and overhead of dynamic clock control. When used with the dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating. For thermal management, the e600 core provides a supervisor-level instruction cache throttling control register (ICTC). Chapter 10, “Power and Thermal Management,” of the *MPC7450 RISC Microprocessor Family User’s Manual* provides information about how to configure the ICTC register for the e600 core.

2.10 Performance Monitor

The e600 core incorporates a performance monitor facility that system designers can use to help bring up, debug, and optimize software performance. The performance monitor counts events during execution of instructions related to dispatch, execution, completion, and memory accesses.

The performance monitor incorporates several registers that can be read and written to by supervisor-level software. User-level versions of these registers provide read-only access for user-level applications. Performance monitor

control registers, MMCR0, MMCR1, and MMCR2 can be used to specify which events are to be counted and the conditions for which a performance monitoring exception is taken. Additionally, the sampled instruction address register, SIAR (USIAR), holds the address of the first instruction to complete after the counter overflowed.

Attempting to write to a user-level read-only performance monitor register causes a program exception, regardless of the MSR[PR] setting.

When a performance monitor exception occurs, program execution continues from vector offset 0x00F00.

Chapter 11, “Performance Monitor,” of the *MPC7450 RISC Microprocessor Family User’s Manual* describes the operation of the performance monitor diagnostic tool incorporated in the e600 core.

3 e600 Core Architectural Implementation

The PowerPC architecture consists of three layers. Adherence to the PowerPC architecture can be described in terms of which of the following levels of the architecture is implemented:

- PowerPC user instruction set architecture (UISA)—Defines the base user-level instruction set, user-level registers, data types, floating-point exception model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment
- PowerPC virtual environment architecture (VEA)—Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA, but may not necessarily adhere to the OEA.
- PowerPC operating environment architecture (OEA)—Defines the memory management model, supervisor-level registers, synchronization requirements, and the exception model. Implementations that conform to the OEA also adhere to the UISA and the VEA.

The e600 core implementation supports the three levels of the architecture described above. For more information about the PowerPC architecture, see the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture.* Specific e600 core features are listed in Chapter 1, “Overview,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

This section describes the PowerPC architecture in general, and specific details about the implementation of the e600 core as a low-power, 32-bit device that implements this architecture. The structure of this section follows the user’s manual organization. Each subsection provides an overview of that chapter.

- Registers and programming model—Describes the registers for the operating environment architecture common among processors of this family and describes the programming model. It also describes the registers that are unique to the e600 core.

  Instruction set and addressing modes—Describes the PowerPC instruction set and addressing modes for the PowerPC operating environment architecture, and defines and describes the PowerPC instructions implemented in the e600 core. The information in this section is described more fully in Chapter 2, “Programming Model,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*
- Cache implementation—Describes the cache model that is defined generally by the virtual environment architecture. It also provides specific details about the e600 core cache implementation. The information in this section is described more fully in Chapter 3, “L1, L2, and L3 Cache Operation,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*
- Exception model—Describes the exception model of the PowerPC operating environment architecture and the differences in the e600 core exception model. The information in this section is described more fully in Chapter 4, “Exceptions,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

- Memory management—Describes generally the conventions for memory management. This section also describes the e600 core implementation of the 32-bit PowerPC memory management specification. The information in this section is described more fully in Chapter 5, “Memory Management,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*
- Instruction timing—Provides a general description of the instruction timing provided by the parallel execution supported by the PowerPC architecture and the e600 core. The information in this section is described more fully in Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*
- AltiVec implementation—Points out that the e600 core implements AltiVec registers, instructions, and exceptions as described in the *AltiVec Technology Programming Environments Manual.* Chapter 7, “AltiVec Technology Implementation,” of the *MPC7450 RISC Microprocessor Family User’s Manual* provides complete details.

3.1 PowerPC Registers and Programming Model

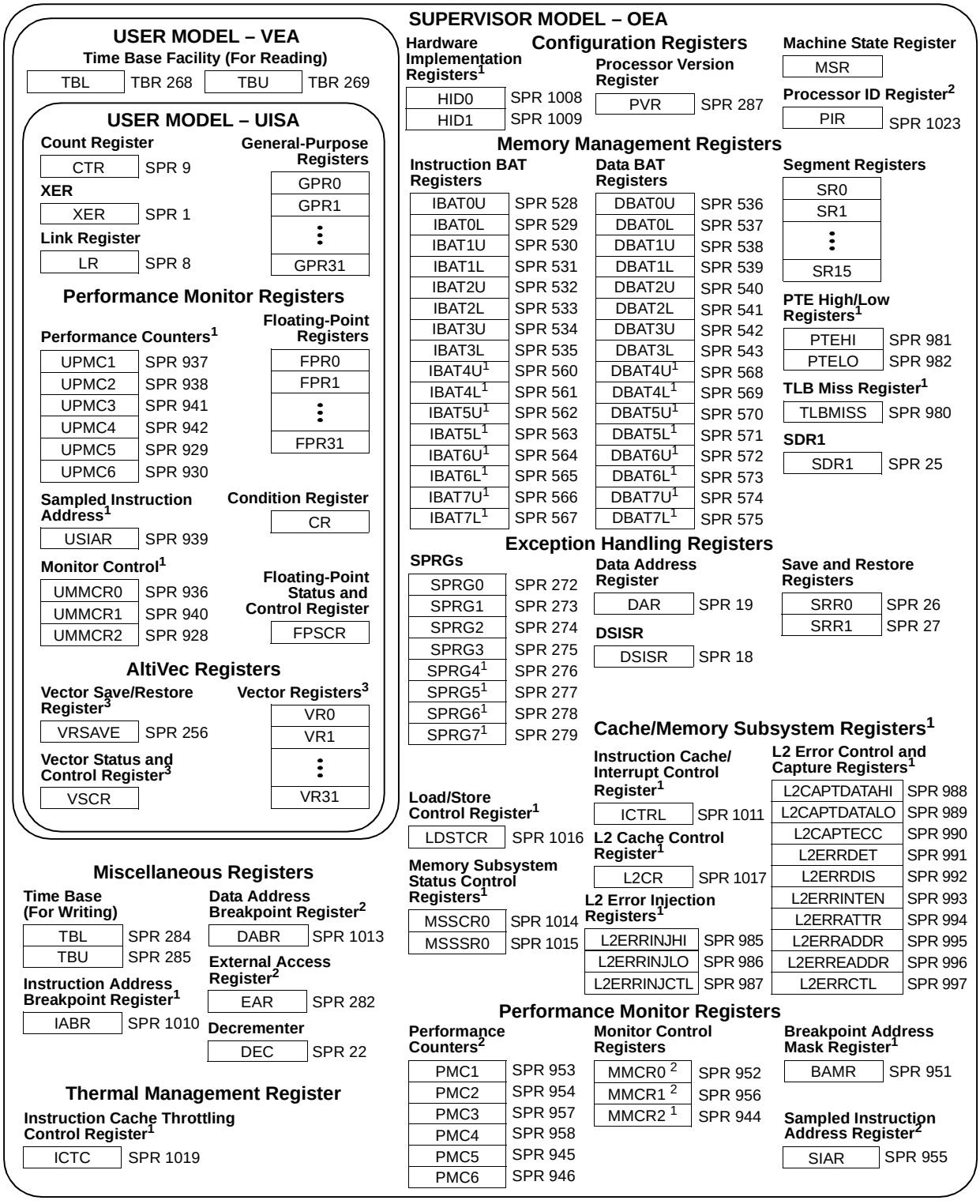
The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction opcode. The three-register instruction format allows specification of a target register distinct from the two source operands. Load and store instructions transfer data between registers and memory.

The PowerPC architecture also defines two levels of privilege—supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, SPRs, and several miscellaneous registers. The AltiVec extensions to the PowerPC architecture augment the programming model with 32 VRs, one status and control register, and one save and restore register. Each processor that implements the PowerPC architecture also has a unique set of implementation-specific registers to support functionality that may not be defined by the PowerPC architecture.

Having access to privileged instructions, registers, and other resources allows the operating system to control the application environment (providing virtual memory and protecting operating-system and critical machine resources). Instructions that control the state of the processor, the address translation mechanism, and supervisor registers can be executed only when the processor is operating in supervisor mode.

Figure 6 shows all the e600 core registers available at the user and supervisor level. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands to access the register. For more information, see Chapter 2, “Programming Model,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

The OEA defines numerous SPRs that serve a variety of functions, such as providing controls, indicating status, configuring the processor, and performing special operations. During normal execution, a program can access the registers shown in Figure 6, depending on the program’s access privilege (supervisor or user, determined by the privilege-level bit, MSR[PR]). GPRs, FPRs, and VRs are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit, as the part of the execution of an instruction.



1 e600-specific register may not be supported on other processors or cores that implement the PowerPC architecture.
2 Register defined as optional in the PowerPC architecture.
3 Register defined by the AltiVec technology.

Figure 6. Programming Model—e600 Core Registers

Some registers can be accessed both explicitly and implicitly. In the e600 core, all SPRs are 32 bits wide. Table 1 describes registers implemented by the e600 core. For the full table, see Table 1-1 in Chapter 1, “Overview,” of the *MPC7450 RISC Microprocessor Family User’s Manual*.

Table 1. Register Summary for the e600 Core

Name	SPR	Description
UISA Registers		
CR	—	Condition register. The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching.
CTR	9	Count register. Holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction.
FPR0– FPR31	—	Floating-point registers (FPR*n*). The 32 FPRs serve as the data source or destination for all floating-point instructions.
FPSCR	—	Floating-point status and control register. Contains floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits for compliance with the IEEE 754 standard.
GPR0– GPR31	—	General-purpose registers (GPR*n*). The thirty-two GPRs serve as data source or destination registers for integer instructions and provide data for generating addresses.
LR	8	Link register. Provides the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can be used to hold the logical address of the instruction that follows a branch and link instruction, typically used for linking to subroutines.
UMMCR0 [1] UMMCR1 [1] UMMCR2 [1]	936, 940, 928	User monitor mode control registers (UMMCR*n*). Used to enable various performance monitor exception functions. UMMCRs provide user-level read access to MMCR registers.
UPMC1– UPMC6 [1]	937, 938 941, 942 929, 930	User performance monitor counter registers (UPMC*n*). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to PMC registers.
USIAR [1]	939	User sampled instruction address register. Contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. USIAR provides user-level read access to the SIAR.
VR0–VR31 [2]	—	Vector registers (VR*n*). Data source and destination registers for all AltiVec instructions.
VRSAVE [2]	256	Vector save/restore register. Defined by the AltiVec technology to assist application and operating system software in saving and restoring the architectural state across process context-switched events. The register is maintained only by software to track live or dead information on each AltiVec register.
VSCR [2]	—	Vector status and control register. A 32-bit vector register that is read and written in a manner similar to the FPSCR.
XER	1	Indicates overflows and carries for integer operations. **Implementation Note**—To emulate the POWER architecture **lscbx** instruction, XER[16–23] are be read with **mfspr**[XER] and written with **mtspr**[XER].

Table 1. Register Summary for the e600 Core (continued)

Name	SPR	Description
VEA		
TBL, TBU (For reading)	TBR 268, TBR 269	Time base facility. Consists of two 32-bit registers, time base lower and upper registers (TBL/TBU). TBL (TBR 268) and TBU (TBR 269) can only be read from and not written to.TBU and TBL can be read with the move from time base register (**mftb**) instruction. **Implementation Note**—Reading from SPR 284 or 285 using the **mftb** instruction causes an illegal instruction exception.
OEA		
BAMR [1]	951	Breakpoint address mask register. Used in conjunction with the events that monitor IABR hits.
DABR [3]	1013	Data address breakpoint register. Optional register implemented in the e600 core and used to cause a breakpoint exception if a specified data address is encountered.
DAR	19	Data address register. After a DSI or alignment exception, DAR is set to the effective address (EA) generated by the faulting instruction.
DEC	22	Decrementer register. A 32-bit decrementer counter used with the decrementer exception. **Implementation Note**—In the e600 core, DEC is decremented and the time base increments at 1/4 of the system bus clock frequency.
DSISR	18	DSI source register. Defines the cause of DSI and alignment exceptions.
EAR	282	External access register. Used with **eciwx** and **ecowx**. Note that the EAR and the **eciwx** and **ecowx** instructions are optional in the PowerPC architecture.
HID0 [1] HID1 [1]	1008, 1009	Hardware implementation-dependent registers. Control various functions, such as the power management features, and locking, enabling, and invalidating the instruction and data caches. The HID1 includes bits that reflects the state of PLL_CFG[0:5] clock signals and control other bus-related functions.
IABR [1]	1010	Instruction address breakpoint register. Used to cause a breakpoint exception if a specified instruction address is encountered.
IBAT0U/L IBAT1U/L IBAT2U/L IBAT3U/L IBAT4U/L [1] IBAT5U/L [1] IBAT6U/L [1] IBAT7U/L [1] DBAT0U/L DBAT1U/L DBAT2U/L DBAT3U/L DBAT4U/L [1] DBAT5U/L [1] DBAT6U/L [1] DBAT7U/L [1]	528, 529 530, 531 532, 533 534, 535 560, 561 562, 563 564, 565 566, 567 536, 537 538, 539 540, 541 542, 543 568, 569, 570, 571 572, 573 574, 575	Block-address translation (BAT) registers. The PowerPC OEA includes an array of block address translation registers that can be used to specify four blocks of instruction space and four blocks of data space. The BAT registers are implemented in pairs: eight pairs of instruction BATs (IBAT0U–IBAT7U and IBAT0L–IBAT7L) and eight pairs of data BATs (DBAT0U–DBAT7U and DBAT0L–DBAT7L). Sixteen additional BAT registers are enabled by setting HID0[HIGH_BAT_EN]. When HID0[HIGH_BAT_EN] = 1, the 16 additional BAT registers, organized as four pairs of instruction BAT registers(IBAT4U–IBAT7U paired with IBAT4L–IBAT7L) and four pairs of data BAT registers (DBAT4U–DBAT7U paired with DBAT4L–DBAT7L) are available. Thus, the e600 core can define a total of 16 blocks implemented as 32 BAT registers. Because BAT upper and lower words are loaded separately, software must ensure that BAT translations are correct during the time that both BAT entries are being loaded. The e600 core implements IBAT[G]; however, attempting to execute code from an IBAT area with G = 1 causes an ISI exception.

Table 1. Register Summary for the e600 Core (continued)

Name	SPR	Description
ICTC [1]	1019	Instruction cache throttling control register. Has bits for enabling instruction cache throttling and for controlling the interval at which instructions are fetched. This controls overall junction temperature.
ICTRL [1]	1011	Instruction cache and interrupt control register. Used in configuring interrupts and error reporting for the instruction and data caches.
L2CR [1]	1017	L2 cache control register. Includes bits for enabling parity checking, setting the L2 cache size, and flushing and invalidating the L2 cache.
L2ERRINJHI [1] L2ERRINJLO [1] L2ERRINJCTL [1] L2CAPTDATAHI [1] L2CAPTDATALO [1] L2CAPTDATAECC [1] L2ERRDET [1] L2ERRDIS [1] L2ERRINTEN [1] L2ERRATTR [1] L2ERRADDR [1] L2ERREADDR [1] L2ERRCTL [1]	985 986 987 988 989 990 991 992 993 994 995 996 997	L2 error registers. The L2 cache supports error injection into the L2 data, data ECC or tag, which can be used to test error recovery software by deterministically creating error scenarios. L2ERRINJHI, L2ERRINJLO, and L2ERRINJCTL are error injection registers. The rest of the registers, error control and capture registers, control the detection and reporting of tag parity, ECC, and L2 configuration errors.
LDSTCR [1]	1016	Load/store control register. Controls data L1 cache way-locking.
MMCR0 [3], MMCR1 [3], MMCR2 [1]	952, 956, 944	Monitor mode control registers (MMCR*n*). Enable various performance monitor exception functions. UMMCR0–UMMCR2 provide user-level read access to these registers.

Table 1. Register Summary for the e600 Core (continued)

<table>
<tr><th>Name</th><th>SPR</th><th>Description</th></tr>
<tr><td>MSR</td><td>—</td><td>Machine state register. Defines the processor state. The MSR can be modified by the mtmsr, sc, and rfi instructions. It can be read by the mfmsr instruction. When an exception is taken, MSR contents are saved to SRR1. See Chapter 4, “Exceptions,” of the MPC7450 RISC Microprocessor Family User’s Manual. The following bits are optional in the PowerPC architecture.
Note that setting MSR[EE] masks decrementer and external interrupt exceptions and e600-specific system management, and performance monitor exceptions.
<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>6</td><td>VEC</td><td>AltiVec available. e600 and AltiVec technology specific; optional to the PowerPC architecture.
0 AltiVec technology is disabled.
1 AltiVec technology is enabled.
Note: When a non-stream AltiVec instruction accesses VRs or the VSCR when VEC = 0 an AltiVec unavailable exception is generated. This does not occur for data streaming instructions (dst(t), dstst(t), and dss); the VRs and the VSCR are available to data streaming instructions even if VEC = 0. VRSAVE can be accessed even if VEC = 0.</td></tr>
<tr><td>13</td><td>POW</td><td>Power management enable. e600-specific and optional to the PowerPC architecture.
0 Power management is disabled.
1 Power management is enabled. The processor can enter a power-saving mode determined by HID0[NAP,SLEEP] when additional conditions are met.</td></tr>
<tr><td>29</td><td>PMM</td><td>Performance monitor marked mode. e600-specific and optional to the PowerPC architecture. See Chapter 11, “Performance Monitor,” of the MPC7450 RISC Microprocessor Family User’s Manual.
0 Process is not a marked process.
1 Process is a marked process.</td></tr>
</table></td></tr>
<tr><td>MSSCR0 [1]</td><td>1014</td><td>Memory subsystem control register. Used to configure and operate many aspects of the memory subsystem.</td></tr>
<tr><td>MSSSR0 [1]</td><td>1015</td><td>Memory subsystem status register. Used to configure and operate the parity functions in the L2 cache for the e600 core.</td></tr>
<tr><td>PIR</td><td>1023</td><td>Processor identification register. Provided for system use. e600 core does not change PIR contents.</td></tr>
<tr><td>PMC1–
PMC6 [3]</td><td>953, 954
957, 958
945, 946</td><td>Performance monitor counter registers (PMCn). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to these registers.</td></tr>
<tr><td>PTEHI,
PTELO</td><td>981,
982</td><td>The PTEHI and PTELO registers are used by the tlbld and tlbli instructions to create a TLB entry. When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the bits of the page table entry (PTE) for this access are located by software and saved in the PTE registers.</td></tr>
<tr><td>PVR</td><td>287</td><td>Processor version register. Read-only register that identifies the version (model) and revision level of the processor.</td></tr>
</table>

Table 1. Register Summary for the e600 Core (continued)

Name	SPR	Description
SDAR, USDAR	—	Sampled data address register. The e600 core does not implement the optional registers (SDAR or the user-level, read-only USDAR register) defined by the PowerPC architecture. Note that in previous processors the SDA and USDA registers could be written to by boot code without causing an exception, this is not the case in the e600. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception**.**
SDR1	25	Sample data register. Specifies the base address of the page table entry group (PTEG) address used in virtual-to-physical address translation. **Implementation Note**—The SDR1 register has been modified (with the SDR1[HTABEXT] and SDR1[HTMEXT] fields) for the e600 core to support the extended 36-bit physical address (when HID0[XAEN] = 1]).
SIAR [3]	955	Sampled instruction address register. Contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. USIAR provides user-level read access to the SIAR.
SPRG0–SPRG3 SPRG4–SPRG7 [1]	272–275 276–279	SPRG0–3. Provided for operating system use. The SPRG4–7 provide additional registers to be used by system software for software table searching.
SR0–SR15	—	Segment registers (SR*n*). Note that the e600 core implements separate instruction and data MMUs. It associates architecture-defined SRs with the data MMU. It reflects SRs values in separate, shadow SRs in the instruction MMU.
SRR0, SRR1	26, 27	Machine status save/restore registers (SRR*n*). Used to save the address of the instruction at which execution continues when **rfi** executes at the end of an exception handler routine. SRR1 is used to save machine status on exceptions and to restore machine status when **rfi** executes. **Implementation Note**—When a machine check exception occurs, the e600 core sets one or more error bits in SRR1. Refer to the individual exceptions for individual SRR1 bit settings.
SVR [1]	286	System version register. Read-only register provided for future product compatibility.
TBL, TBU (For writing)	284, 285	Time base. A 64-bit structure (two 32-bit registers) that maintains the time of day and operating interval timers. The TB consists of two registers—time base upper (TBU) and time base lower (TBL). The time base registers can be written to only by supervisor-level software. TBL (SPR 284) and TBU (SPR 285) can only be written to and not read from. TBL and TBU can be written to, with the move to special purpose register (**mtspr**) instruction. **Implementation Note**—Reading from SPR 284 or 285 causes an illegal instruction exception.
TLBMISS [1]	980	The TLBMISS register is automatically loaded when software searching is enabled (HID0[STEN] = 1) and a TLB miss exception occurs. Its contents are used by the TLB miss exception handlers (the software table search routines) to start the search process.

1 e600-specific register that may not be supported on other processors that implement the PowerPC architecture.

2 Register is defined by the AltiVec technology.

3 Defined as optional register in the PowerPC architecture.

3.2 Instruction Set

All PowerPC instructions are encoded as single-word (32-bit) opcodes. Instruction formats are consistent among all instruction types, permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format greatly simplifies instruction pipelining.

For more information, see Chapter 2, “Programming Model,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

3.2.1 PowerPC Instruction Set

The PowerPC instructions are divided into the following categories:

- Integer instructions—These include computational and logical instructions.
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR.
  - Floating-point arithmetic instructions
  - Floating-point multiply/add instructions
  - Floating-point rounding and conversion instructions
  - Floating-point compare instructions
  - Floating-point status and control instructions
- Load and store instructions—These include integer and floating-point load and store instructions.
  - Integer load and store instructions
  - Integer load and store multiple instructions
  - Floating-point load and store
  - Primitives used to construct atomic memory operations (**lwarx** and **stwcx.** instructions)
- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow.
  - Branch and trap instructions
  - Condition register logical instructions
- Processor control instructions—These instructions are used for synchronizing memory accesses and management of caches, TLBs, and the segment registers.
  - Move to/from SPR instructions
  - Move to/from MSR
  - Synchronize
  - Instruction synchronize
  - Order loads and stores
- Memory control instructions—These instructions provide control of caches, TLBs, and SRs.
  - Supervisor-level cache management instructions
  - User-level cache instructions

— Segment register manipulation instructions
— Translation lookaside buffer management instructions

This grouping does not indicate the execution unit that executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (one double word) floating-point operands. The PowerPC architecture uses instructions that are four bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 GPRs. It also provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (FPRs).

Computational instructions do not modify memory. To use a memory operand in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written back to the target location with distinct instructions.

Processors that implement the PowerPC architecture follow the program flow when they are in the normal execution state. However, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either kind of exception may cause one of several components of the system software to be invoked.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored in 32-bit implementations.

3.2.2 AltiVec Instruction Set

The AltiVec instructions are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate, and shift instructions.
- Vector floating-point arithmetic instructions—These include floating-point arithmetic instructions, as well as a discussion on floating-point modes.
- Vector load and store instructions—These include load and store instructions for vector registers. The AltiVec technology defines LRU and transient type instructions that can be used to optimize memory accesses.
  — LRU instructions. The AltiVec architecture specifies that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave cache entries in a least-recently-used (LRU) state instead of a most-recently-used state.
  — Transient instructions. The AltiVec architecture describes a difference between static and transient memory accesses. A static memory access should have some reasonable degree of locality and be referenced several times or reused over some reasonably long period of time. A transient memory reference has poor locality and is likely to be referenced a very few times or over a very short period of time.

    The following instructions are interpreted to be transient:
    – **dstt** and **dststt** (transient forms of the two data stream touch instructions)
    – **lvxl** and **stvxl**
- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions.

- Processor control instructions—These instructions are used to read and write from the AltiVec status and control register.
- Memory control instructions—These instructions are used for managing of caches (user level and supervisor level).

3.2.3 e600 Core Microprocessor Instruction Set

The e600 core instruction set is defined as follows:

- The e600 core provides hardware support for all 32-bit PowerPC instructions.
- The e600 core implements the following instructions optional to the PowerPC architecture:
  - External Control In Word Indexed (**eciwx**)
  - External Control Out Word Indexed (**ecowx**)
  - Data Cache Block Allocate (**dcba**)
  - Floating Select (**fsel**)
  - Floating Reciprocal Estimate Single-Precision (**fres**)
  - Floating Reciprocal Square Root Estimate (**frsqrte**)
  - Store Floating-Point as Integer Word (**stfiwx**)
  - Load Data TLB Entry (**tlbld**)
  - Load Instruction TLB Entry (**tlbli)**

3.3 On-Chip Cache Implementation

The following subsections describe the PowerPC architecture's treatment of cache in general, and the e600-specific implementation, respectively. A detailed description of the e600 core cache implementation is provided in Chapter 3, "L1, L2, and L3 Cache Operation," of the *MPC7450 RISC Microprocessor Family User's Manual.*

3.3.1 PowerPC Cache Model

The PowerPC architecture does not define hardware aspects of cache implementations. For example, processors that implement the PowerPC architecture can have unified caches, separate L1 instruction and data caches (Harvard architecture), or no cache at all. These microprocessors control the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited/caching-allowed mode
- Memory coherency required/memory coherency not required mode

The caches are physically addressed, and the data cache can operate in either write-back or write-through mode as specified by the PowerPC architecture.

The PowerPC architecture defines the term 'cache block' as the cacheable unit. The VEA and OEA define cache management instructions a programmer can use to affect cache contents.

3.3.2 e600 Core Cache Implementation

The e600 core has the same cache implementation as the MPC7450, which is described in Chapter 1, “Overview,” of the *MPC7450 RISC Microprocessor Family User’s Manual*. The BPU also contains a 128-entry BTIC that provides immediate access to cached target instructions.

3.4 Exception Model

The following sections describe the PowerPC exception model and the e600 core implementation. A detailed description of the e600 core exception model is provided in Chapter 4, “Exceptions,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

3.4.1 PowerPC Exception Model

The OEA portion of the PowerPC architecture defines the mechanism by which processors that implement the PowerPC architecture invoke exceptions. Exception conditions may be defined at other levels of the architecture. For example, the UISA defines conditions that may cause floating-point exceptions; the OEA defines the mechanism by which the exception is taken.

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of unusual conditions arising in the execution of instructions and from external signals, bus errors, or various internal conditions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions begins in supervisor mode.

Although multiple exception conditions can map to a single exception vector, often a more specific condition may be determined by examining a register associated with the exception—for example, the DSISR and the floating-point status and control register (FPSCR). Also, software can explicitly enable or disable some exception conditions.

The PowerPC architecture requires that exceptions be taken in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are handled strictly in order with respect to the instruction stream. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the exception is taken. In addition, if a single instruction encounters multiple exception conditions, those exceptions are taken and handled sequentially. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

To prevent loss of state information, exception handlers must save the information stored in the machine status save/restore registers, SRR0 and SRR1, soon after the exception is taken to prevent this information from being lost due to another exception event. Because exceptions can occur while an exception handler routine is executing, multiple exceptions can become nested. It is the exception handler’s responsibility to save the necessary state information if control is to return to the excepting program.

In many cases, after the exception handler handles an exception, there is an attempt to execute the instruction that caused the exception. Instruction execution continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

The following terms are used to describe the stages of exception processing: recognition, taken, and handling.

- Recognition—Exception recognition occurs when the condition that can cause an exception is identified by the processor.
- Taken—An exception is said to be taken when control of instruction execution is passed to the exception handler; that is, the context is saved and the instruction at the appropriate vector offset is fetched and the exception handler routine begins executing in supervisor mode.
- Handling—Exception handling is performed by the software at the appropriate vector offset. Exception handling is begun in supervisor mode.

The term ‘interrupt’ describes the external interrupt, the system management interrupt, and sometimes the asynchronous exceptions. Note that the PowerPC architecture uses the word ‘exception’ to refer to IEEE-defined floating-point exception conditions that may cause a program exception to be taken. The occurrence of these IEEE exceptions may or may not cause an exception to be taken. IEEE-defined exceptions are referred to as IEEE floating-point exceptions or floating-point exceptions.

3.4.2 e600 Core Exceptions

As specified by the PowerPC architecture, exceptions can be either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions are caused by events external to the processor’s execution; synchronous exceptions are caused by instructions.

The types of exceptions are shown in Table 2. Note that all exceptions except for the performance monitor, AltiVec unavailable, instruction address breakpoint, system management, AltiVec assist, and the three software table search exceptions are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

Table 2. e600 Core Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Types
Asynchronous, nonmaskable	Imprecise	System reset, machine check
Asynchronous, maskable	Precise	External interrupt, system management interrupt, decrementer exception, performance monitor exception
Synchronous	Precise	Instruction-caused exceptions

The exception classifications are discussed in greater detail in Chapter 4, “Exceptions,” of the *MPC7450 RISC Microprocessor Family User’s Manual.* For a better understanding of how the e600 core implements precise exceptions, see Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual.* Table 3 lists the exceptions implemented in the e600 core, and conditions that cause them. Table 3 also notes the e600-specific exceptions.

The three software table search exceptions support software page table searching and are enabled by setting HID0[STEN]. See Chapter 4, “Exceptions,” and Chapter 5, “Memory Management Unit,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

Table 3. Exceptions and Conditions

Exception Type	Vector Offset	Causing Conditions
Reserved	0x00000	—
System reset	0x00100	Assertion of either $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ or at power-on reset
Machine check	0x00200	Assertion of $\overline{\text{TEA}}$ during a data bus transaction, assertion of $\overline{\text{MCP}}$, an address bus parity error on the MPX bus, a data bus parity error on the MPX bus, an L1 instruction cache error, an L1 data cache error, and a memory subsystem detected error including the following: • L2 data parity error • L2 tag parity error • Single-bit and multiple-bit L2 ECC errors MSR[ME] must be set.
DSI	0x00300	As specified in the PowerPC architecture. Also includes the following: • A hardware table search due to a TLB miss on load, store, or cache operations results in a page fault • Any load or store to a direct-store segment (SR[T] = 1) • A **lwarx** or **stwcx.** instruction to memory with cache-inhibited or write-through memory/cache access attributes.
ISI	0x00400	As specified in the PowerPC architecture
External interrupt	0x00500	MSR[EE] = 1 and $\overline{\text{INT}}$ is asserted
Alignment	0x00600	• A floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned. • A multiple/string load/store operation is attempted in little-endian mode • An operand of a **dcbz** instruction is on a page that is write-through or cache-inhibited for a virtual mode access. • An attempt to execute a **dcbz** instruction occurs when the cache is disabled or locked.
Program	0x00700	As specified in the PowerPC architecture
Floating-point unavailable	0x00800	As specified in the PowerPC architecture
Decrementer	0x00900	As defined by the PowerPC architecture, when the msb of the DEC register changes from 0 to 1 and MSR[EE] = 1
Reserved	0x00A00–00BFF	—
System call	0x00C00	Execution of the System Call (**sc**) instruction
Trace	0x00D00	MSR[SE] =1 or a branch instruction is completing and MSR[BE] =1. The e600 core operates as specified in the OEA by taking this exception on an **isync**.

Table 3. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
Reserved	0x00E00	The e600 core does not generate an exception to this vector. Other processors that implement the PowerPC architecture may use this vector for floating-point assist exceptions.
Reserved	0x00E10–00EFF	—
Performance monitor	0x00F00	The limit specified in PMC*n* is met and MMCR0[ENINT] = 1 (e600-specific)
AltiVec unavailable	0x00F20	Occurs due to an attempt to execute any non-streaming AltiVec instruction when MSR[VEC] = 0. This exception is not taken for data streaming instructions (**dst*x***, **dss,** or **dssall**). (e600-specific)
ITLB miss	0x01000	An instruction translation miss exception is caused when HID0[STEN] = 1 and the effective address for an instruction fetch cannot be translated by the ITLB (e600-specific).
DTLB miss-on-load	0x01100	A data load translation miss exception is caused when HID0[STEN] = 1 and the effective address for a data load operation cannot be translated by the DTLB (e600-specific).
DTLB miss-on-store	0x01200	A data store translation miss exception is caused when HID0[STEN] = 1 and the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs, and the changed bit in the PTE must be set due to a data store operation (e600-specific).
Instruction address breakpoint	0x01300	IABR[0–29] matches EA[0–29] of the next instruction to complete and IABR[BE] = 1 (e600-specific).
System management interrupt	0x01400	MSR[EE] = 1 and $\overline{\text{SMI}}$ is asserted (e600-specific).
Reserved	0x01500–015FF	—
AltiVec assist	0x01600	This e600-specific exception supports denormalization detection in Java mode as specified in the *AltiVec Technology Programming Environments Manual* in Chapter 3, “Operand Conventions.”
Reserved	0x01700–02FFF	—

3.5 Memory Management

The following subsections describe the memory management features of the PowerPC architecture, and the e600 core implementation, respectively.

3.5.1 PowerPC Memory Management Model

The primary function of the MMU in a processor or core that implements the PowerPC architecture is the translation of logical (effective) addresses to physical addresses (referred to as real addresses in the architecture specification) for memory accesses and I/O accesses (I/O accesses are assumed to be memory-mapped). In addition, the MMU provides access protection on a segment, block, or page basis. Note that the e600 core does not implement the optional direct-store facility.

Two general types of memory accesses generated by processors that implement the PowerPC architecture require address translation—instruction accesses and data accesses generated by load and store instructions. In addition, the addresses specified by cache instructions and the optional external control instructions also require translation. Generally, the address translation mechanism is defined in terms of the segment descriptors and page tables that the processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the virtual address to a physical address.

The segment descriptors, used to generate the interim virtual addresses, are stored as on-chip segment registers on 32-bit implementations (such as the e600 core). In addition, two translation lookaside buffers (TLBs) are implemented on the e600 core to keep recently used page address translations on-chip. Although the PowerPC OEA describes one MMU (conceptually), the e600 core hardware maintains separate TLBs and table search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the e600 core is described as having two MMUs, one for instruction accesses (IMMU) and one for data accesses (DMMU).

The block address translation (BAT) mechanism is a software-controlled array that stores the available block address translations on-chip. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor special-purpose registers (SPRs). There are separate instruction and data BAT mechanisms. In the e600 core, they reside in the instruction and data MMUs, respectively.

The MMUs, together with the exception processing mechanism, provide the necessary support for the operating system to implement a paged virtual memory environment and for enforcing protection of designated memory areas. Chapter 4, “Exceptions,” of the *MPC7450 RISC Microprocessor Family User’s Manual* describes how the MSR controls critical MMU functionality.

3.5.2 e600 Core Memory Management Implementation

The e600 core implements separate MMUs for instructions and data. It maintains a copy of the segment registers in the instruction MMU; however, read and write accesses to the segment registers (**mfsr** and **mtsr**) are handled through the segment registers in the data MMU. The e600 core MMU is described in Chapter 5, “Memory Management Unit,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

The e600 core implements the memory management specification of the PowerPC OEA for 32-bit implementations but adds capability for supporting 36-bit physical addressing. Thus, it provides 4 Gbytes of physical address space accessible to supervisor and user programs, with a 4-Kbyte page size and 256-Mbyte segment size. In addition, the e600 core MMUs use an interim virtual address (52 bits) and hashed page tables in the generation of 32- or 36-bit physical addresses (depending on the setting of HID0[XAEN]). Processors that implement the PowerPC architecture also have a BAT mechanism for mapping large blocks of memory. Block range from 128 Kbytes to 256 Mbytes and are software programmable.

The e600 core provides table search operations performed in hardware. The 52-bit virtual address is formed and the MMU attempts to fetch the PTE that contains the physical address from the appropriate TLB on-chip. If the translation is not found in either the BAT array or in a TLB (that is, a TLB miss occurs), the hardware performs a table search operation (using a hashing function) to search for the PTE. Hardware table searching is the default mode for the e600 core; however, if HID0[STEN] = 1, a software table search is performed.

The e600 core also provides support for table search operations performed in software (if HID0[STEN] is set). In this case, the TLBMISS register saves the effective address of the access that requires a software table search. The PTEHI and PTELO registers and the **tlbli** and **tlbld** instructions are used in reloading the TLBs during a software table search operation.

The following exceptions support software table searching if HID0[STEN] is set and a TLB miss occurs:

- For an instruction fetch, an ITLB miss exception
- For a data load, an DTLB miss-on-load exception
- For a data store, an DTLB miss-on-store exception

The e600 core implements the optional TLB invalidate entry (**tlbie**) and TLB synchronize (**tlbsync**) instructions that can be used to invalidate TLB entries.

3.6 Instruction Timing

This section describes how the e600 core performs operations defined by instructions and reports the results of instruction execution. The e600 core design minimizes average instruction execution latency, which is the number of clock cycles it takes to fetch, decode, dispatch, issue, and execute instructions and make results available for subsequent instructions. Some instructions, such as loads and stores, access memory and require additional clock cycles between the execute phase and the write-back phase. Latencies depend on whether an access is to cacheable or noncacheable memory, whether it hits in the L1 or L2 cache, whether a cache access generates a write back to memory, whether the access causes a snoop hit from another device that generates additional activity, and other conditions that affect memory accesses.

To improve throughput, the e600 core implements pipelining, superscalar instruction issue, branch folding, removal of fall-through branches, three-level speculative branch handling, and multiple execution units that operate independently and in parallel.

As an instruction passes from stage to stage, the subsequent instruction can follow through the stages as the preceding instruction vacates them, allowing several instructions to be processed simultaneously. Although it may take several cycles for an instruction to pass through all the stages, when the pipeline is full, one instruction can complete its work on every clock cycle. Figure 7 represents a generic four-stage pipelined execution unit, which when filled has a throughput of one instruction per clock cycle.

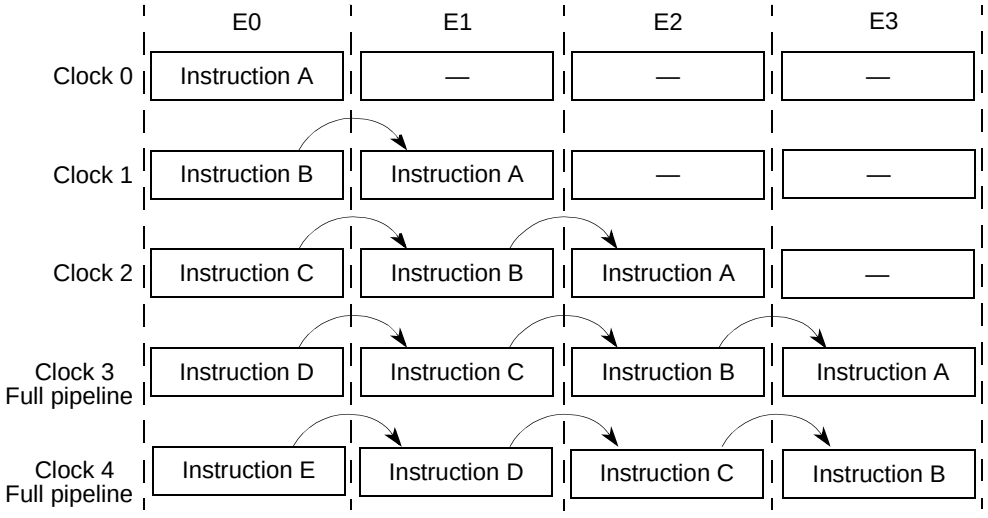


Figure 7. Pipelined Execution Unit

Figure 8 shows the entire path that instructions take through the fetch1, fetch2, decode/dispatch, execute, issue, complete, and write-back stages, which is considered the master pipeline of the e600 core. The FPU, LSU, IU2, VIU2, VFPU, and VPU are multiple-stage pipelines.

The e600 core contains the following execution units:

- Branch processing unit (BPU)
- Three integer unit 1s (IU1a, IU1b, and IU1c)—execute all integer instructions except multiply, divide, and move to/from SPR instructions.
- Integer unit 2 (IU2)—executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions
- 64-bit floating-point unit (FPU)
- Load/store unit (LSU)
- The AltiVec unit contains the following four independent execution units for vector computations
  - AltiVec permute unit (VPU)
  - AltiVec integer unit 1 (VIU1)
  - Vector integer unit 2 (VIU2)
  - Vector floating-point unit (VFPU)

  A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

The e600 core can complete as many as three instructions on each clock cycle. In general, the e600 core processes instructions in seven stages—fetch1, fetch2, decode/dispatch, issue, execute, complete, and write-back, as shown in Figure 8. Note that the pipeline example in Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual* is similar to the four-stage VFPU pipeline in Figure 8.

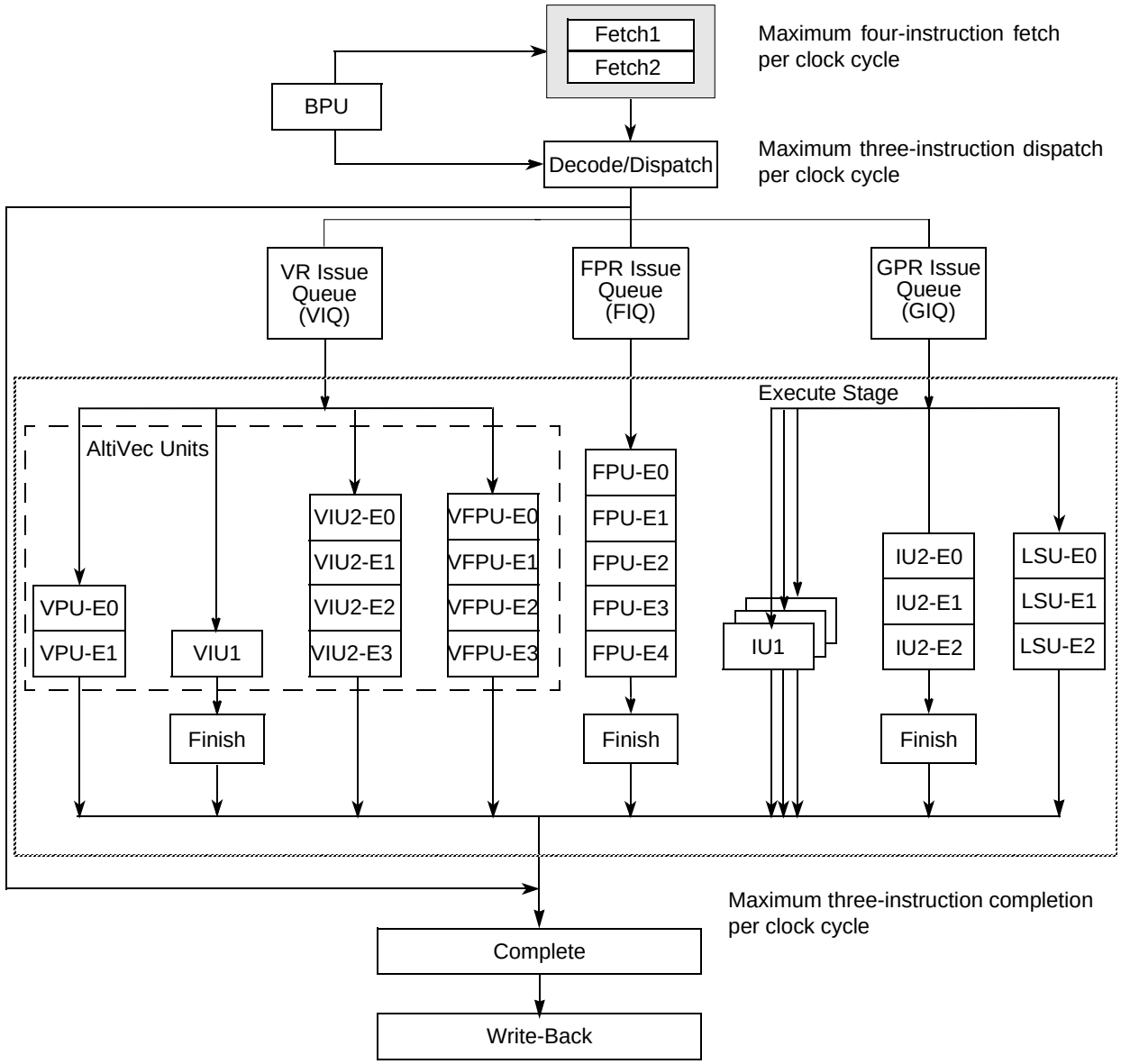


Figure 8. Superscalar/Pipeline Diagram

The instruction pipeline stages are described as follows:

- Instruction fetch—Includes the clock cycles necessary to request an instruction and the time the memory system takes to respond to the request. Instructions retrieved are latched into the instruction queue (IQ) for subsequent consideration by the dispatcher.

  Instruction fetch timing depends on many variables, such as whether an instruction is in the branch target instruction cache (BTIC), the on-chip instruction cache, or the L2 cache. Those factors increase when it is necessary to fetch instructions from system memory and include the processor-to-bus clock ratio, the amount of bus traffic, and whether any cache coherency operations are required.
- The decode/dispatch stage fully decodes each instruction; most instructions are dispatched to the issue queues (branch, **isync**, **rfi**, and **sc** instructions do not go to issue queues).

- The three issue queues, FIQ, VIQ, and GIQ, can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - Instructions are dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - Space must be available in the CQ for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not an issue queue).
- The issue stage reads source operands from rename registers and register files and determines when instructions are latched into the execution unit reservation stations. The GIQ, FIQ, and VIQ (AltiVec) issue queues have the following similarities:
  - Operand lookup in the GPRs, FPRs, and VRs, and their rename registers.
  - Issue queues issue instructions to the proper execution units.
  - Each issue queue holds twice as many instructions as can be dispatched to it in one cycle; the GIQ has six entries, the VIQ has four, and the FIQ has two.

  The three issue queues are described as follows:
  - The GIQ accepts as many as three instructions from the dispatch unit each cycle. IU1, IU2, and all LSU instructions (including floating-point and AltiVec loads and stores) are dispatched to the GIQ.
  - Instructions can be issued out-of-order from the bottom three GIQ entries (GIQ2–GIQ0). An instruction in GIQ1 destined for an IU1 does not have to wait for an instruction in GIQ0 that is stalled behind a long-latency integer divide instruction in the IU2.
  - The VIQ accepts as many as two instructions from the dispatch unit each cycle. All AltiVec instructions (other than load, store, and vector touch instructions) are dispatched to the VIQ. In the MPC7450, as many as two instructions can be issued to the four AltiVec execution units, but unlike the GIQ, instructions in the VIQ cannot be issued out of order. In the MPC7448, a maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). This means an instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability.
  - The FIQ can accept one instruction from the dispatch unit per clock cycle. It looks at the first instruction in its queue and determines if the instruction can be issued to the FPU in this cycle.
- The execute stage accepts instructions from its issue queue when the appropriate reservation stations are not busy. In this stage, the operands assigned to the execution stage from the issue stage are latched.

  The execution unit executes the instruction (perhaps over multiple cycles), writes results on its result bus, and notifies the CQ when the instruction finishes. The execution unit reports any exceptions to the completion stage. Instruction-generated exceptions are not taken until the excepting instruction is next to retire.

  Most integer instructions have a 1-cycle latency, so results of these instructions are available 1 clock cycle after an instruction enters the execution unit. The FPU, LSU, IU2, VIU2, VFPU, and VPU units are pipelined, as shown in Chapter 7, “AltiVec Technology and Implementation,” in the *MPC7450 RISC Microprocessor Family User’s Manual*.

  Note that AltiVec computational instructions are executed in the four independent, pipelined AltiVec execution units. The VPU has a two-stage pipeline, the VIU1 has a one-stage pipeline, and the VIU2 and VFPU have four-stage pipelines. As many as 10 AltiVec instructions can be executing concurrently.
- The complete and write-back stages maintain the correct architectural machine state and commit results to the architected registers in the proper order. If completion logic detects an instruction containing an exception status, all following instructions are cancelled, their execution results in rename buffers are discarded, and the correct instruction stream is fetched.

The complete stage ends when the instruction is retired. Three instructions can be retired per clock cycle. If no dependencies exist, as many as three instructions are retired in program order. Chapter 6, “Instruction Timing,” of the *MPC7450 RISC Microprocessor Family User’s Manual* describes completion dependencies.

The write-back stage occurs in the clock cycle after the instruction is retired.

3.7 AltiVec Implementation

The e600 core implements the AltiVec registers and instruction set as they are described in the *AltiVec Technology Programming Environments Manual* in Chapter 2, “AltiVec Register Set,” and in Chapter 6, “AltiVec Instructions.” Two additional implementation specific exceptions have been added; they are as follows:

- The AltiVec assist exception, which is used in handling denormalized numbers in Java mode.
- An alignment exception for cache-inhibited AltiVec loads and stores and write-through stores that execute when in 60x bus mode

Both exceptions are described fully in Chapter 4, “Exceptions,” of the *MPC7450 RISC Microprocessor Family User’s Manual.* Also, the default setting for the VSCR[NJ] bit is Java–compliant (VSCR[NJ] = 0) in the e600 core. The AltiVec implementation is described fully in Chapter 7, “AltiVec Technology Implementation,” of the *MPC7450 RISC Microprocessor Family User’s Manual.*

4 Document Revision History

Table 4 provides a revision history for this product brief.

Table 4. Document Revision History

Revision Number	Substantive Changes
0	Initial revision.

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120 191014
+81 3 3440 3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@
hibbertgroup.com

E600COREPB
Rev. 0
11/2004

freescale™
semiconductor