digital

Date: 12 April, 1996
From: Daniel Leibholz
Dept: Digital Semiconductor
MS: HLO2-3/J3
DTN: 225-6141
Enet: ad::leibholz

Topic: EV6 Chip Specification, Rev 2.0

Enclosed you will find the second revision of the EV6 specification.

This version includes a major rewrite of the external interface, substantial changes to the PAL/IPR sections, as well as inclusion of PAL coding restrictions and some electrical and packaging information.

As the EV6 design proceeds, we are filling in the details of the following topics:

- Electrical and packaging information
- Reset and initialization
- Test and debug features
- PLL Operation
- Error handling

We will send further documentation on these areas (plus errata/changes to rev. 2.0) when available.

Please note that the EV6 specification is Digital Confidential. Refer all requests for copies to Sue Jacquart (ad::jacquart/DTN 225-4967).

Please refer questions regarding the spec to the following individuals:

EV6 Microarchitecture/PALcode/IPR issues: Dan Leibholz
EV6 System Interface/Electrical issues: Rick Hetherington (ad::hetherington, DTN 225-4571)

EV6 Specification

Rev 2.0
April 10, 1996

1. EV6 AND THE ALPHA ARCHITECTURE 7

1.1 Alpha Architectural Extensions 7

1.2 Implementation-Specific Features 7

1.3 Instruction Set Features Defined as Optional 8

1.4 Arithmetic Exceptions 8

2. INTERNAL ARCHITECTURE 9

2.1 Chip Organization 9
2.1.1 Ebox 9
2.1.2 Fbox 10
2.1.3 Ibox 11
2.1.4 On-chip Caches 13
2.1.5 Mbox 14
2.1.6 Cbox 15

2.2 Pipeline Organization 16
2.2.1 Stage 0 - Instruction Fetch 16
2.2.2 Stage 1 - Instruction Slot 16
2.2.3 Stage 2 - Map 17
2.2.4 Stage 3 - Issue 17
2.2.5 Stage 4 - Register Read 17
2.2.6 Stage 5 - Execute 17
2.2.7 Stage 6 - Dcache Access 17
2.2.8 Instruction Retire 17
2.2.9 Retire of Operates into R31/F31 18
2.2.10 Pipeline Aborts 18

2.3 Memory And I/O Accesses 19
2.3.1 Memory Space Load Instructions 19
2.3.2 IO Space Load Instructions 19
2.3.3 Memory Space Store Instructions 20
2.3.4 IO Space Store Instructions 20

2.4 Replay Traps 21
2.4.1 Mbox Order Traps 21
2.4.2 Other Mbox Replay Traps 22

2.5 Software-Directed Prefetching and Loads into R31 & F31 22
2.5.1 Normal Prefetch: LDL, LDF, LDG, LDB, LDW 22
2.5.2 Prefetch with Modify Intent: LDS 22
2.5.3 Prefetch, Evict Next: LDQ 22
2.5.4 Prefetch, No Reuse: LDT 22

2.6 Special Cases **22**
2.6.1 Load Hit Speculation 23
2.6.2 Floating Point Stores 24
2.6.3 CMOV 24

2.7 Instruction Issue Rules **25**
2.7.1 Instruction Class Definitions 25
2.7.2 Ebox Slotting 25
2.7.3 Instruction Latencies 28

3. EXTERNAL INTERFACE **29**

3.1 Address Spaces **29**
3.1.1 I/O Ordering and Merge Rules 31

3.2 Cache Organization and Coherence **31**
3.2.1 Cache Block States 31
3.2.2 Cache Block State Transitions 32
3.2.3 System Knowledge of Bcache Contents 33
3.2.4 Dcache States & the Dcache Duplicate Tags 33
3.2.5 Memory Barrier (MB/WMB/TBfill flow) 33
3.2.6 Load/Locked and Store/Conditional 35

3.3 System Port **38**
3.3.1 System Port Pins 38
3.3.2 EV6 to System Address/Command Format 38
3.3.3 SysAdd Commands Generated by EV6 39
3.3.4 Probe Response Transfers 41
3.3.5 SysAck & System Port Flow Control 41
3.3.6 SysReadValid and Speculative Reads 41
3.3.7 SysAdd Commands Generated by the System 42
3.3.8 Two Cycle Commands For Data Transfers 44
3.3.9 Data Movement In and Out of EV6 45
3.3.10 Data ECC 50
3.3.11 Ordering of System Port Transactions 51
3.3.12 System Port Clocking 53

3.4 Bcache Port **66**
3.4.1 Bcache Port Pins 66
3.4.2 Pin Descriptions 66
3.4.3 Bcache Banking 67
3.4.4 Bcache Transactions 67
3.4.5 Bcache Clocking 67

3.5 Interrupts **70**

3.6 Pin List **70**

4. PRIVILEGED ARCHITECTURE LIBRARY CODE **72**

4.1 Use of Alpha Implementation-Specific Opcodes **72**
4.1.1 HW_LD Instruction 73

4.1.2 HW_ST Instruction 73
4.1.3 HW_RET Instruction 75
4.1.4 HW_MFPR and HW_MTPR Instructions 75

4.2 Internal Processor Register Access Mechanisms 77
4.2.1 IPR Scoreboard Bits 77
4.2.2 Hardware Structure of Explicitly Written IPRs 78
4.2.3 Hardware Structure of Implicitly Written IPRs 78
4.2.4 IPR Access Ordering 79
4.2.5 IPRs and HW_RET Stalls 79

4.3 PAL Shadow Registers 80

4.4 PALcode Emulation of FPCR 80
4.4.1 Status Flags 80
4.4.2 MF_FPCR 80
4.4.3 MT_FPCR 80

4.5 PALcode Entry Points 80
4.5.1 CALL_PAL entry 81
4.5.2 PALcode Exception Entry Points 81

4.6 TB Fill Flows 82
4.6.1 DTB Fill 82
4.6.2 ITB Fill 83

5. INTERNAL PROCESSOR REGISTERS 84

5.1 Ebox IPRs 86
5.1.1 CC 86
5.1.2 CC_CTL 86
5.1.3 VA 87
5.1.4 VA_FORM 88
5.1.5 VA_CTL 89

5.2 Ibox IPRs 90
5.2.1 ITB_TAG 90
5.2.2 ITB_PTE 90
5.2.3 ITB_IAP 91
5.2.4 ITB_IA 91
5.2.5 ITB_IS 91
5.2.6 EXC_ADDR 91
5.2.7 IVA_FORM 91
5.2.8 IER_CM 92
5.2.9 SIRR 93
5.2.10 ISUM 94
5.2.11 HW_INT_CLR 95
5.2.12 EXC_SUM 96
5.2.13 PAL_BASE 97
5.2.14 I_CTL 98
5.2.15 I_STAT 101
5.2.16 IC_FLUSH 101
5.2.17 CLR_MAP 101

5.2.18 SLEEP 101
5.2.20 Ibox Process Context IPR (PCTX) 102
5.2.21 PCTR_CTL 104

5.3 Mbox IPRs 105
5.3.1 DTB_TAG0 & DTB_TAG1 105
5.3.2 DTG_PTE0 & DTB_PTE1 106
5.3.3 DTB_ALTMODE 106
5.3.4 DTB_IAP 107
5.3.5 DTB_IA 107
5.3.6 DTB_IS0 & DTB_IS1 108
5.3.7 DTB_ASN0 & DTB_ASN1 108
5.3.8 MM_STAT 109
5.3.9 M_CTL 110
5.3.10 DC_CTL 111
5.3.11 DC_STAT 112

5.4 Cbox CSRs and IPRs 112
5.4.1 CBOX CSR Description 113
5.4.2 CBOX IPR Description 116

6. IEEE FLOATING POINT CONFORMANCE 117

6.1 Floating Point Control Register (FPCR) 120

7. ERROR DETECTION AND HANDLING 122

7.1 Icache Data or Tag Parity Error 122

7.2 Dcache Tag Parity Error 122

7.3 Dcache Data Correctable ECC Error 122
7.3.1 Load Instruction 122
7.3.2 Store Instruction (Less than Quadword Length) 123
7.3.3 Victim Reads 123

7.4 Dcache Triplicate Tag Parity Error 123

7.5 Bcache Tag Parity Error 123

7.6 Bcache Data Correctable ECC Error 124
7.6.1 Icache Fill from Bcache 124
7.6.2 Dcache Fill from Bcache 124
7.6.3 Victim Read 124

7.7 Bcache Data Uncorrectable ECC Error 124

7.8 Memory Data Correctable ECC Error 125
7.8.1 Icache Fill from Memory 125
7.8.2 Dcache Fill from Memory 125

7.9 Memory Data Uncorrectable ECC Error 125

7.10 System Port Read Errors 125

8. INITIALIZATION AND TEST 126

9. ELECTRICAL DATA 127

9.1 Electrical Characteristics 127

9.2 DC Characteristics 127
9.2.1 Power Supply 127
9.2.2 Input Signal Pins 127
9.2.3 Driven Signals From EV6 127

9.3 AC Characteristics 128
9.3.1 Clocking Scheme 128
9.3.2 Input Clocks 128
9.3.3 129
9.3.4 1.3.3Analog PLL 129
9.3.5 Timing 129

9.4 Power Supply Considerations 131
9.4.1 Decoupling 131
9.4.2 Power Supply Sequencing 131

10. PACKAGING INFORMATION 132

10.1 Introduction 132

10.2 Package Information 132

10.3 EV6 Pinout 133

11. APPENDIX 1: RESET AND SLEEP MODE 143

12. APPENDIX 2: PAL CODING RESTRICTIONS 144

12.1 Restriction: Reset Sequence Required by Retirator and Mapper 144

12.2 Restriction: No Multiple Writers to IPRs in Same Scoreboard Group 145

12.3 Restriction: (removed) 145

12.4 Restriction: No Writers and Readers to IPRs in Same Scoreboard Group 145

12.5 Restriction: PAL shadow enables 145

12.6 Guideline: Avoid Consecutive read-modify-write-read-modify-write sequences to IPRs in the Same Scoreboard Group 145

12.7 Restriction: Replay trap and interrupt code sequence and STF/ITOF 146

12.8 Restriction (removed) **147**

12.9 Restriction: PALmode I-Stream address ranges **147**

12.10 Restriction: Duplicate IPR mode bits **147**

12.11 Guideline: Ibox IPR update synchronization **147**

12.12 Restriction: HW_MFPR EXC_ADDR/IVA_FORM/EXC_SUM Usage **147**

12.13 Restriction: DTB FILL flow collision **147**

12.14 Restriction: HW_RET **147**

12.15 Restriction: (REMOVED) **147**

12.16 Restriction: JSR-BAD VA **147**

12.17 Restriction: MTPR to DTB_TAG0/DTB_PTE0/DTB_TAG1/DTB_PTE1 **148**

12.18 Restriction: No FP OPERATES or FP CONDITIONAL BRANCHES in same fetch block as MTPR **148**

12.19 Restriction: HW_RET/STALL after updating the FPCR via MT_FPCR in PALmode **148**

12.20 Guideline: I_CTL SBE Stream Buffer Enable **148**

12.21 Restriction: HW_RET/STALL after MT ASN0/ASN1 **148**

12.22 Restriction: HW_RET/STALL after MT IS0/IS1 **148**

12.23 Restriction: HW_ST/P/CONDITIONAL does not "clear" the lock flag **149**

12.24 Restriction: HW_RET/STALL after MT ITB_IA, ITB_IAP, IC_FLUSH **149**

12.25 Restriction: MT ITB_IA after Reset **149**

12.26 Guideline: Conditional branches in PALcode **149**

12.27 Restriction: Reset of 'Force-Fail Lock Flag' State in PALcode **150**

1. EV6 and the Alpha Architecture

This section describes the ways in which EV6 architecturally differs from prior Alpha implementations. These architectural differences fall into four classes:

1. Extensions to the Alpha Architecture, such as new instructions.
2. Architectural features which the Alpha SRM defines as implementation-specific, such as the size of the virtual or physical address space.
3. Instruction set features which the Alpha SRM defines as optional.
4. Arithmetic exceptions

Alpha SRM version 6.0 and appendix Z of that document are included by reference.

1.1 Alpha Architectural Extensions

EV6 includes the following instruction set extensions to the Alpha Architecture:

- Floating point square root for both VAX and IEEE formats
- Population Count - counts the number of ones in an integer register: *CTPOP*
- Leading and trailing zero count: *CTLZ, CTTZ*
- Cache Control Operations:
  - ⇒ Evict Data Cache Block: *ECB*
  - ⇒ Write Hint: *WH64*
- Integer to floating and floating to integer register transfers: *ITOFS, ITOFF, ITOFT, FTOIS & FTOIT*
- Graphics & MultiMedia instructions:
  - ⇒ Pixel Error: *PERR*
  - ⇒ Min and Max instructions: *MINUB8, MINSB8, MINUW4, MINSW4, MAXUB8, MAXSB8, MAXUW4, MAXSW4*
  - ⇒ Pack and Unpack instructions: *PKLB, PKWB, UNPKBL, UNPKBW*
- Software-directed prefetch instructions: *LDL/LDF/LDG/LDB/LDW/LDS/LDQ/LDT into R31/F31*
- Version and architecture extension instructions: *AMASK/IMPLVER*
- Power-saving feature/instruction: *CALL_PAL WTINT*

1.2 Implementation-Specific Features

- 8 KB page size
- 48-bit virtual address, with IPR-controlled 43-bit mode
- 44-bit physical address with MSB indicating IO space when set
- Loads into R31 and F31 are executed to completion, and memory access violations, alignment faults and fault-on-read errors generated by these instructions are reported by hardware. PALcode is expected to dismiss these exceptions as required by Alpha SRM ECO 95. See section 2.5 for more details on software prefetching with loads into R31F31.
- Integer operate instructions into R31 are dismissed; no arithmetic exceptions are reported.
- Floating point operate instructions into F31 are dismissed; no arithmetic exceptions are reported.
- Load-locked/Store Conditional semantics are, except for the waiver described below, compliant with ALPHA SRM ECO 102:
  - There must be no intervening memory operation between the LDx_L and STx_C; the presence of a memory operation (LDx,STx) will cause the STx_C to always fail. One exception (for which EV6 requires a waiver): if the memory operation is a WH64, the STx_C might succeed even in the presence of a store from another processor to the lock range.

- The physical address of STx_C must specify a location within the naturally aligned 16-byte block in physical memory accessed by the preceding LDx_L instruction (in processor issue sequence) from the same processor. Otherwise it is unpredictable whether the lock flag will be cleared by a store from another processor within the lock range.

1.3 Instruction Set Features Defined as Optional

This section describes instruction set features which the Alpha SRM defines as optional, and from which EV6 differs in comparison with prior implementations.

- *FETCH* and *FETCH_M* are not implemented
- IEEE floating point support
  - ⇒ NaN's and infinities are generated and propagated in hardware
  - ⇒ rounding to plus and minus infinity is supported in hardware (this is also true of EV5, but not of EV4)

1.4 Arithmetic Exceptions

In EV6 arithmetic exceptions are precise and reported as synchronous traps, and the TRAPB and EXCB instructions are processed as NOPs. This behavior is architecturally compliant, but means that the software completion rules as currently defined in the Alpha SRM are conservative relative to EV6. These rules could simply state that floating operates are not allowed to overwrite their own operands and should have their /S qualifier set.

2. Internal Architecture

EV6 is the third-generation implementation of Digital's Alpha RISC architecture. It is a superscalar CPU which performs register renaming, speculative execution and dynamic scheduling in hardware. It contains four integer execution units, two of which can perform memory address calculations for load and store instructions. It also contains dedicated execution units for floating point add, multiply, divide and square root. The on-chip instruction cache is a 64K byte, two-way set associative virtual cache with 64-byte blocks. The on-chip data cache is a 64K byte, two-way set associative, virtually indexed, physically tagged, write-back cache with 64-byte blocks.

The external interface consists of two ports - a Bcache port and a System port. The Bcache port is controlled entirely by the processor, and is used to interface to a module-level secondary cache which may be built from a range of standard synchronous SRAMs. The System port interfaces to the rest of the system. The processor contains two external data busses, one 16-bytes wide and the other 8-bytes wide. The 16-byte bus is used to support the Bcache port and the 8-byte bus is used to support the System port.

The chip will initially be fabricated in Digital's 0.35um CMOS-6 process. The speed distribution will center at an internal operating frequency of 550 MHz, though the final bin points are TBD. At 500 MHz, power dissipation is estimated to be 60 watts at 2.0 volts.

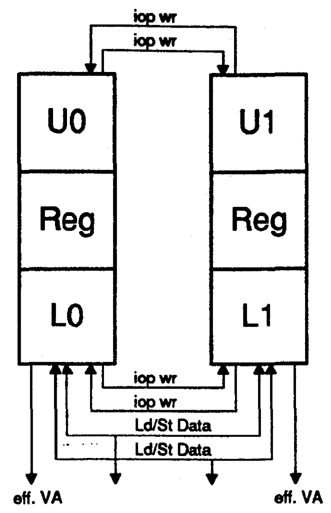
2.1 Chip Organization

EV6 consists of the following internal sections:

- Integer execution unit (Ebox)
- Floating point execution unit (Fbox)
- Instruction fetch, issue and retire unit (Ibox)
- Memory reference unit (Mbox)
- External cache and system interface unit (Cbox)
- Instruction cache (Icache)
- Data cache (Dcache)

2.1.1 Ebox

The Ebox is a four-wide integer execution unit which is implemented as two functional unit "clusters" - labeled 0 and 1. Each cluster contains a copy of an 80-entry physical register file and two "subclusters", named upper (U) and lower (L). Most instructions have one-cycle latency for consumers which execute within the same cluster. There is a one cycle delay associated with producing a value in onc cluster and consuming the value in the other cluster. The instruction issue queue minimizes the performance effect of this cross-cluster delay.



The Ebox contains the following resources:

- Four 64-bit adders, all of which are used to calculate results for integer ADD instructions.. The adders in subclusters L0 and L1 are used to generate the effective virtual address for load and store instructions.
- Four logic units
- Two barrel shifters and associated byte logic - U0 and U1
- two sets of conditional branch logic - U0 and U1
- two copies of an 80-entry register file
- one fully pipelined multiplier, with 7-cycle latency for all integer multiply operations - U1
- one fully pipelined unit with 3-cycle latency. This unit executes the following instructions:
  ⇒ POPC, LOC, TOC
  ⇒ PERR, MINxxx, MAXxxx, UNPKxx, PKxx

The 80 Ebox register file entries contain storage for the values of the 31 Alpha integer registers (the value of R31 is not stored), the values of 8 PAL shadow registers, and 41 results written by instructions that have not yet retired. Ignoring cross-cluster delay, the two copies of the Ebox register files contain identical values. Each copy of the Ebox register file contains four read ports and six write ports. The four read ports are used to source operands to each of the two subclusters within a cluster. Two write ports are used to write results generated within the cluster; two write ports are used to write results generated by the other cluster; and two write ports are used to write results from load instructions.

2.1.2 Fbox

The Fbox is a two-wide floating point execution unit which executes both VAX and IEEE floating point instructions. It support IEEE S_floating and T_floating data types and all rounding modes. It also supports VAX F_floating and G_floating data types, and provides limited support for D_floating format. It contains the following resources:

- a 72-entry physical register file
- a fully pipelined multiplier with four cycle latency
- a fully pipelined adder with four cycle latency
- a nonpipelined divide unit associated with the adder pipeline
- a nonpipelined square root unit associated with the adder pipeline

The 72 Fbox register file entries contain storage for the values of the 31 Alpha floating point registers other than F31, and 41 values written by instructions that are not yet retired. The Fbox register file contains six reads ports and four write ports. Four read ports are used to source operands to the add and multiply pipelines, and two read ports are used to source data for store instructions. Two write ports are used to write results generated by the add and multiply pipelines, and two write ports are used to write results from floating point load instructions.

2.1.3 Ibox

The Ibox consists of the following subsections:

- Virtual PC logic
- Instruction-stream translation buffer (ITB)
- Instruction fetch logic
- Register rename maps
- Integer and floating point issue queues
- Exception and interrupt logic
- Retire logic

2.1.3.1 Virtual PC Logic

The Virtual PC logic maintains the virtual program counter values for instructions that are in flight. There can be up to 80 instructions in 20 successive fetch slots in-flight between the mappers and the end of the pipeline, hence the VPC logic contains a 20-deep table to store these fetched VPCs.

2.1.3.2 Instruction Translation Buffer (ITB)

The Ibox includes a 128-entry, fully associative translation buffer used to store recently used I-stream address translations and page protection information. Each of the entries in the ITB can map 1, 8, 64 or 512 contiguous 8K byte pages. The allocation scheme is round-robin. The ITB supports an 8-bit ASN and contains an ASM bit. The Icache is virtual, hence the ITB is only accessed for I-stream references which miss the Icache. The Icache contains the access-check information so a fetch address translation is only made if the address missed in the Icache.

2.1.3.3 Instruction Fetch Logic

The instruction fetcher reads up to four naturally aligned instructions per cycle from the instruction cache. It uses both branch prediction and line prediction to maximize efficiency. It also contains a subroutine return prediction stack and an Icache stream controller. The stream controller generates fetch requests for additional icache lines and stores the istream data in the icache. There is no separate buffer to hold stream requests.

2.1.3.4 Register Rename Maps

The prefetcher forwards instructions to the integer and floating point register rename maps. The rename maps perform two functions. First, they serve to eliminate register WAR and WAW dependencies while preserving true RAW data dependencies, in order to allow instructions to be dynamically rescheduled. Second, they provide a means of speculatively executing instructions before the control flow previous to those instructions is resolved. Note that both exceptions and branch mispredicts represent deviations from the control flow predicted by the prefetcher.

The map logic translates each instruction's operand register specifiers from the "virtual" register numbers in the instruction to the "physical" register numbers which hold the corresponding architecturally correct values. The map logic also renames each instruction's destination register specifier from the virtual number in the instruction to a physical register number chosen from a list of free physical registers, and updates the register maps. The map logic can process four instructions per cycle.

The map logic does not return the physical register which holds the old value of an instruction's virtual destination register to the free list until the instruction retires, which means that the control flow up to that instruction has been resolved. If a branch mispredict or exception occurs, the map logic backs the contents of both maps up to the state associated with the instruction which triggered the condition, and the prefetcher restarts at the appropriate PC.

At most 20 valid fetch slots containing up to 80 instructions can be in flight between the register maps and the end of the machine's pipeline, where the control flow is finally resolved. The map logic is capable of backing the contents of the maps up to the state associated with any of these 80 instructions in a single cycle.

2.1.3.5 Instruction Issue Queues

The register rename logic places instructions into one of issue queues, from which they are later issued to functional units for execution.

2.1.3.5.1 Integer Queue (IQ)

The integer queue (IQ) is associated with the Ebox, is 20-deep, and issues instructions of the following types at a maximum rate of four operations per cycle:

- integer operates
- integer conditional branches
- unconditional branches - both displacement and memory format
- integer and floating point loads and stores
- PAL-reserved instructions: HW_MTPR, HW_MFPR, HW_LD, HW_ST, HW_RET
- ITOFx, FTOIx

Each queue entry physically produces four requests signals - one for each of the Ebox subclusters. A queue entry asserts a request when it contains an instruction that can be executed by the subcluster, if the instruction's operand register values are available within the subcluster. There are two arbiters - one for the upper subclusters and one for the lower subclusters. Each arbiter picks two of the possible 20 requesters for service each cycle. A given instruction only requests upper subclusters or lower subclusters, but since many instructions can only be executed in one type or another this is not too constraining. For example, loads and stores can only go to lower subclusters, and shifts can only go to upper subclusters. Instructions which can execute in either upper or lower subclusters, such as adds and logic operations, are statically assigned before being placed in the IQ.

The IQ arbiters pick between simultaneous requesters of a subcluster based on age - older instructions are given priority over newer instructions.

If a given instruction requests both lower subclusters and no older instruction requests a lower subcluster, then the arbiter assigns subcluster L0 to the instruction. If a given instruction requests both upper subclusters and no older instruction requests an upper subcluster, then the arbiter assigns subcluster U1 to the instruction. This asymmetry between the upper and lower subcluster arbiters is a circuit implementation optimization.

2.1.3.5.2 Floating Point Queue (FQ)

The floating point queue is associated with the Fbox, is 15-deep, and issues the following instruction types:

- floating point operates
- floating point conditional branches

- floating point stores
- floating point register to integer register transfers (ftoi)

Each queue entry physically produces three request wires - one for the add pipe, one for the mul pipe, and one for stores. There are three arbiters, one for each of the add, mul and store pipes. The add and mul arbiters pick one requester per cycle, and each of two store pipe arbiters picks one requester per cycle.

The FQ arbiters pick between simultaneous requesters of a pipe based on age - older instructions are given priority over new instructions. Floating stores and FTOI instructions in even-numbered queue entries arbitrate for one store port and floating stores and FTOI instructions in odd-numbered queue entries arbitrate for a second store port.

Floating stores and FTOI instructions are enqueued in both the integer and floating queues. They wait in the floating queue until their operand register values are available. They subsequently request service to the store arbiter. Upon issue from the floating queue, they signal the corresponding entry in the integer queue to request service. Upon issue from the integer queue, the operation is completed.

2.1.3.6 Exception and Interrupt Logic

There are two types of exceptions: faults and synchronous traps. Arithmetic exceptions are precise and reported as synchronous traps.

There are four sources of interrupts:

- Level sensitive hardware interrupts sourced by the **irq_h<5:0>** pins.
- Edge sensitive hardware interrupts generated by the serial line receive pin, performance counter overflows, and hardware corrected read errors.
- Software interrupts sourced by the software interrupt request (SIRR) register.
- Asynchronous system traps (ASTs).

Interrupt sources can be individually masked. In addition, AST interrupts are qualified by the current processor mode.

2.1.3.7 Retire Logic

The Ibox fetches instructions in program order, executes them out of order, and retires them in order. The retire logic maintains the correct architectural state of the machine by retiring an instruction only if all previous instructions have executed without generating exceptions or branch mispredicts. In effect, retiring an instruction commits the machine to any changes the instruction may have made to software-visible state, of which there are three classes:

- The integer and floating point registers
- Memory
- Internal processor registers (including control/status registers and translation buffers).

The retire logic can sustain a maximum retire rate of eight instructions per cycle, and can retire up to as many as eleven instruction in a single cycle.

2.1.4 On-chip Caches

EV6 contains two on-chip primary caches implemented with fully static, six transistor CMOS structures.

2.1.4.1 Instruction Cache

The instruction cache (Icache) is a 64K byte, virtual cache. Set prediction is used to approximate the performance of a two-set cache without slowing the cache access time. Each Icache block contains:

- 16 Alpha instructions (64 bytes)
- Virtual tag bits <47:15>
- An 8-bit address space number (ASN) field
- A 1-bit address space match (ASM) bit
- A 1-bit PALcode bit to indicate physical addressing
- A valid bit
- Data and tag parity protection
- Four access-check bits: K, E, S, U
- Additional predecoded information to assist with instruction processing and fetch control

2.1.4.2 Data Cache

The data cache (Dcache) is a 64K byte, two-way set associative, virtually indexed, physically tagged, write-back, read/write allocate cache with 64-byte blocks. Each cycle the Dcache can perform:

- two quadword (or shorter) reads to arbitrary addresses, or
- two quadword writes to the same aligned octaword, or
- two non-overlapping less-than-quadword writes to the same aligned quadword, or
- one sequential read and write of the same aligned octaword

Each Dcache block contains:

- 64 data bytes and associated quadword ECC
- Physical tags bits <42:13>
- *Valid*, *dirty*, *shared*, and *modified* bits
- A tag parity bit calculated across the tag, dirty, shared and modified bits
- A bit to control round-robin set allocation (one bit per two cache blocks)

The dcache contains two sets, each with 512 rows containing 64-byte blocks per row (i.e. 32K bytes of data per set). EV6 requires 2 additional bits of virtual address beyond the bits which specify an 8K byte page in order to specify a dcache row index. Conceptually, a given virtual address might be found in 4 distinct places in the dcache, depending on the virtual-to-physical translation for those two bits. EV6 prevents this aliasing by keeping only one of the four possible translated addresses in the cache at any particular time.

2.1.5 Mbox

The Mbox is responsible for controlling the Dcache and for ensuring architecturally correct behavior of load and store instructions. It contains the following structures:

- Load queue (LQ)
- Store queue (SQ)
- Miss address file (MAF)
- D-stream translation buffer (DTB)

2.1.5.1 Load Queue (LQ)

The load queue (LQ) is essentially a reorder buffer for load instructions. It contains 32 entries and maintains the state associated with load instructions which have been issued to the Mbox but which have not delivered their results to the CPU and been retired. The Mbox assigns loads to load queue slots based on the order in which they were fetched from the Icache and places them into the load queue after they are issued by the IQ. The load queue serves to help ensure correct Alpha memory reference behavior.

2.1.5.2 Store Queue (SQ)

The store queue (SQ) is essentially a reorder buffer and graduation unit for store instructions. It contains 32 entries and maintains the state associated with store instructions which have been issued to the Mbox but which have not both been retired and written to the Dcache. The Mbox assigns stores to store queue slots based on the order in which they were fetched from the Icache and places them into the store queue after they are issued by the IQ. The store queue holds data associated with stores issued from the IQ until they are retired, at which point the store can be allowed to update the Dcache. The store queue also serves to help ensure correct Alpha memory reference behavior.

2.1.5.3 Miss Address File (MAF)

The miss address file (MAF) holds physical addresses associated with pending Icache and Dcache fill requests and pending IO space reads. It contains eight entries.

2.1.5.4 D-stream Translation Buffer (DTB)

The Mbox includes a 128-entry, fully associative translation buffer used to store recently used D-stream address translations and page protection information. Each of the entries in the DTB can map 1, 8, 64 or 512 contiguous 8K byte pages. The allocation scheme is round-robin. The DTB supports an 8-bit ASN and contains an ASM bit.

2.1.6 Cbox

The CBOX controls the Bcache and System ports. It contains the following structures:

- Victim Address File (VAF)
- Victim Data File (VDF)
- IO Write Buffer (IOWB)
- Probe Queue (PQ)
- Duplicate Dcache Tags (DTAGS)

2.1.6.1 Victim Address File (VAF) and Victim Data File (VDF)

The VAF and VDF together form an 8-entry victim buffer used for holding:

- Dcache blocks to be written to the Bcache
- I-stream cache blocks from memory to be written to the Bcache
- Bcache blocks to be written to memory
- Cache blocks sent to the system in response to probe commands

2.1.6.2 IO Write Buffer (IOWB)

The IOWB consists of four 64-byte entries and associated address and control used for buffering IO write data between the store queue and the System port.

2.1.6.3 Probe Queue (PQ)

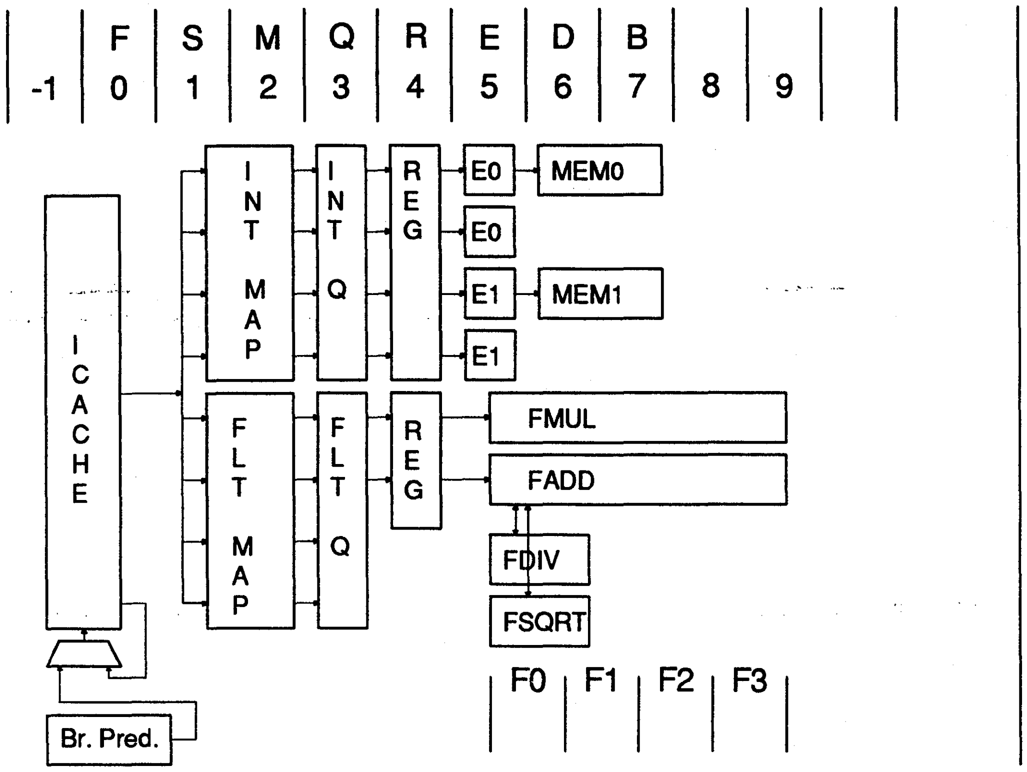
The probe queue (PQ) is an eight-deep queue which holds pending System port cache probe commands and addresses.

2.1.6.4 Duplicate Dcache Tag (DTAG) Array

The DTAG array holds a duplicate copy of the Dcache tags and is used by the Cbox when processing Dcache fills, Icache fills and System port probes. See section 3 for more details.

2.2 Pipeline Organization

The machine's basic pipeline is shown below:



2.2.1 Stage 0 - Instruction Fetch

In the fetch stage of the pipe, up to four aligned instructions are fetched from the Icache. The branch prediction tables are also accessed in this cycle. The branch tables produce a prediction for one branch or memory format JSR instruction per cycle, hence the prefetcher is limited to fetching through one branch per cycle. If there is more than one branch within the fetch line, and the branch predictor predicts that the first branch will not be taken, it will predict through subsequent branches at the rate of one per cycle, until it predicts a taken branch or predicts through the last branch in the fetch line.

The Icache array also contains a line prediction field, the contents of which are applied to the Icache in the next cycle. The purpose of the line predictor is to remove the pipeline bubble which would otherwise be created when the branch predictor predicts a branch to be taken. In effect, the line predictor attempts to predict the Icache line which the branch predictor will generate. On fills, the line predictor value at each fetch line is initialized with the index of the next sequential fetch line, and later retrained by the branch predictor if necessary.

2.2.2 Stage 1 - Instruction Slot

In the slot stage the branch predictor compares the next Icache index that it generates to the index that was generated by the line predictor. If there's a mismatch the branch predictor wins - the instructions fetched during that cycle are aborted, and the index predicted by the branch predictor is applied to the Icache the next cycle. Line mispredicts result in one pipeline bubble.

There is one case where the line predictor takes precedence over the branch predictor - memory format calls or jumps. If the line predictor was trained with a true (as opposed to predicted) memory format call or jump target, then its contents take precedence over the target hint field associated with these instructions. This allows dynamic calls or jumps to be correctly predicted.

The instruction fetcher produces the full VPC during the fetch stage of the pipe. The Icache produces the tags for both sets 0 and 1 each time it's accessed, which enables the fetcher to differentiate set mispredicts from true Icache misses. If the access was a set mispredict the fetcher aborts the last two fetched slots and re-fetches the slot in the next cycle. It also retrains the appropriate set prediction bits.

The instruction data is transferred from the icache to the integer and floating point register map hardware during this stage. In addition the integer instructions begin to pass through the slot logic, which determines whether they will use upper or lower eboxes.

2.2.3 Stage 2 - Map

Instructions are sent from the Icache to the integer and floating point register maps during the slot stage, and register renaming is performed during the map stage. Also, each instruction is assigned a unique 8-bit number, called an **inum**, which is used to identify the instruction and its program order with respect to other instructions during the time that it is in flight. Instructions are considered to be in flight between the time they are mapped and the time they are retired.

Mapped instructions and their associated inums are placed in the integer and floating point queues by the end of the map stage.

2.2.4 Stage 3 - Issue

Instructions are selected for execution by the IQ and FQ during the issue stage of the pipe. In general, instructions are deleted from the IQ or FQ two cycles after they issue - i.e. if an instruction issues in cycle N, it remains in the queue but does not request service in cycle N+1, and is gone in cycle N+2.

2.2.5 Stage 4 - Register Read

Instructions which are issued from the queues read their operands from the register files and receive bypass data.

2.2.6 Stage 5 - Execute

The Ebox and Fbox pipelines begin execution in this pipe stage.

2.2.7 Stage 6 - Dcache Access

Memory reference instructions access the Dcache and data translation buffers in this pipe stage. In general, loads access both the tag and data arrays in pipe stage 6, while stores only access the tag array. Store data is written into the store queue where it is held until the store instruction retires.

Most integer operate instructions write their register results in this cycle.

2.2.8 Instruction Retire

A given instruction retires when it has been executed to completion, and all previous instructions have been retired. The execution pipe stage in which a given instruction becomes eligible to be retired depends upon the type of instruction. The following table gives the minimum retire latencies (assuming that all previous instructions have been retired) for various classes of instructions:

Instruction Class	Retire Stage	Comments
INT Conditional Branch	7	

INT Multiply	7/13	(13 for MUL/V)
INT Operate	7	
Memory	10	
FP Add	11	
FP Mul	11	
FP DIV/SQRT	11 + L*	Add latency of instruction see section 2.7.3. Latency is 11 if hardware detects that no exception is possible (see section 2.2.8.1)
FP Conditional Branch	11	Branch mispredict is reported in stage 7
BSR/JSR	10	JSR mispredict is reposted in stage 8

2.2.8.1 *FP Divide/Square Root Early Retire*

The FP divider and square root unit can detect that, for many combinations of source operand values, no exception can be generated. Instructions with these operands can retire before the result is generated. When detected, they retire with the same latency as FP Add. Early retire is not possible for the following instruction/operand/architecture state conditions:

- Instruction is not a DIV or SQRT
- SQRT source operand is negative
- Divide operand exponent_a is 0
- Either operand is NaN or INF
- Divide operand exponent_b is 0
- Trapping mode is /I (inexact)
- INE status bit is 0

Early retire is also not possible for divides if the resulting exponent has any of the following characteristics (define EXP as the result exponent):

- DIVT,DIVG: EXP >= 0x3ff or EXP <= 0x2
- DIVS,DIVF: EXP >= 0x7f or EXP <= 0x382

2.2.9 Retire of Operates into R31/F31

Many instructions which have R31 or F31 as their destination are retired immediately upon decode (stage 3). These instructions do not produce a result and are 'squashed' from the pipeline as well -- they do not occupy a slot in the issue queues and do not occupy a functional unit.

Instruction Type	Notes
INTA, INTL, INTM, INTS	All with R31 as destination
FLTI, FLTL, FLTV	All with F31 as destination. MT_FPCR is not included because it has no destination -- it is never squashed
LDQ_U	All with R31 as destination
MISC	TRAPB and EXCB are always squashed. Others are never squashed.
FLTS	All (SQRT, ITOF) with F31 as destination

2.2.10 Pipeline Aborts

The following table lists the timing associated with each common source of pipeline abort. The abort penalty as given is measured from the cycle after the fetch stage of the instruction which triggers the abort to the fetch stage of the new target, ignoring any Ibox pipeline stalls or queuing delay which the triggering instruction might experience.

Abort Condition	Penalty (cycles)	Comments
Branch mispredict	7	integer or floating conditional branch mispredict

Abort Condition	Penalty (cycles)	Comments
JSR mispredict	8	memory format JSR or HW_RET
Mbox order trap	14	load-load order, store-load order
Other Mbox replay traps	13	
DTB miss	13	
ITB miss	7	
Integer arithmetic trap	12	
FP arithmetic trap	13+L	Add latency of instruction. See section 2.7.3 2.7.3for instruction latencies.

2.3 Memory And I/O Accesses

This section provides a brief overview of EV6 processing of memory and IO references.

The IQ may issue any combination of loads and stores to the Mbox at the rate of two per cycle. The two lower Ebox subclusters, L0 and L1, generate the 48-bit effective virtual address for these instructions.

In the discussions which follow, an instruction is said to be **newer** than another instruction if it follows that instruction in program order and is said to be **older** if it precedes that instruction in program order.

2.3.1 Memory Space Load Instructions

The Mbox begins execution of a load instruction by translating its virtual address to a physical address using the DTB and by accessing the Dcache. The Dcache is virtually indexed, allowing these two operations to be done in parallel. The Mbox puts information about the load, including its physical address, destination register and data format, into the load queue.

If the requested physical location is found in the Dcache (a hit) the data is formatted and written into the appropriate integer or floating register. If the location is not in the Dcache (a miss) then the physical address is placed in the miss address file (MAF) for processing by the Cbox. The MAF performs a merging function in which a new miss address is compared to miss addresses already held in the MAF. If the new miss address is to the same Dcache block as a miss address already held in the MAF, then the new miss address is discarded.

When Dcache fill data is returned to the Dcache by the Cbox, the Mbox satisfies the requesting loads in the load queue.

2.3.2 IO Space Load Instructions

Since IO space reads may have side effects, they can't be done speculatively. Hence, when the Mbox receives an IO space read it first places it in the load queue, where it is held until it retires. The Mbox replays retired IO space reads from the load queue to the MAF in program order at a rate of one per CPU cycle. 5.3.9

The MAF handles IO reads differently from memory reads, since for IO space reads the system requires an indication as to which bytes were actually accessed by the CPU. Each MAF entry contains 8 mask bits and a 2-bit length field to hold this information, and may thus hold:

- a single byte or word IO read (byte and word length IO reads are not merged), or
- up to eight longword IO reads within an aligned 32-byte region, or
- up to eight quadword IO reads within an aligned 64-byte region, or
- a single memory space read for an aligned 64-byte Dcache block

EV6 maintains IO reference ordering as follows (assume address X and address Y are different):

First Instruction In Pair	Second Instruction In Pair	Reference Order
LD-IO to address X	LD-IO to address X	maintained
LD-IO to address X	LD-IO to address Y	maintained
ST-IO to address X	ST-IO to address X	maintained
ST-IO to address X	ST-IO to address Y	maintained
LD-IO to address X	ST-IO to address X	maintained
LD-IO to address X	ST-IO to address Y	not maintained
ST-IO to address X	LD-IO to address X	maintained
ST-IO to address X	LD-IO to address Y	not maintained

When the Mbox allocates a new MAF entry to an IO read, it attempts to merge other IO reads into the same entry until one of the following conditions occur, at which point the entry may be serviced by the Cbox.

- an IO read which doesn't merge with the entry is replayed from the load queue
- Four cycles go by without an IO read merging with the entry
- an IO read which matches the entry but touches a mask bit which is already set is replayed from the load queue
- an IO write matches the entry

The Cbox sends IO read requests off-chip in the order in which they were received from the Mbox.

2.3.3 Memory Space Store Instructions

The Mbox begins execution of a store instruction by translating its virtual address to a physical address using the DTB and by probing the Dcache. The Mbox puts information about the store, including its physical address, its data and the results of the Dcache probe, into the store queue.

If the Mbox does not find the addressed location in the Dcache then it places the address into the MAF for processing by the Cbox. If the Mbox finds the addressed location in a Dcache block which isn't dirty, then it places a ChangoToDirty request into the MAF.

A given store instruction may write the Dcache when it is retired and when the Dcache block containing its address is dirty in the Dcache. Store queue entries which meet these two conditions may be placed into the **writeable** state, and are done so in program order at a maximum rate of two entries per cycle. The Mbox transfers writable store queue entries from the store queue to the Dcache in program order at a maximum rate of two stores per cycle. Dcache lines associated with writable store queue entries are locked down by the Mbox - System port probe commands cannot evict these blocks until their associated writable store queue entries have been transferred into the Dcache. This restriction assists in store-conditional and Dcache ECC processing.

Stores in the store queue which have not been transferred to the Dcache may source data to newer load instructions. The Mbox compares the virtual Dcache index bits of incoming loads to queued stores, and sources the data from the store queue, bypassing the Dcache, when necessary.

2.3.4 IO Space Store Instructions

The Mbox begins processing IO space stores just like memory space stores - by translating the virtual address and placing state associated with the store into the store queue.

The Mbox replays retired IO space stores from the store queue to the IOWB in program order at a rate of one per CPU cycle. Each IOWB entry may contain:

- a single byte or word IO write (byte and word length IO writes are not merged), or
- up to eight longword IO writes within an aligned 32-byte region, or
- up to eight quadword IO writes within an aligned 64-byte region

When the Mbox allocates a new IOWB entry to an IO write, it attempts to merge other IO writes into the same entry until one of the following conditions occur, at which point the entry may be serviced by the Cbox.

- an IO write which doesn't merge with the entry is replayed from the store queue
- Four cycles go by without an IO write merging with the entry
- an IO write which matches the entry but touches a mask bit which is already set is replayed from the store queue
- an IO read matches this entry
- a WMB instruction is replayed from the store queue

The Mbox never allows queued IO space stores to source data to subsequent loads. The Cbox sends IO space write requests off-chip in the order they were received from the Mbox.

2.4 Replay Traps

There are some situations in which a load or store instruction can not be executed due to a condition which is detected after that instruction issues from the IQ or FQ. The instruction is therefore aborted (along with all newer instructions) and restarted from the fetch stage of the pipeline. This mechanism is called a replay trap.

2.4.1 Mbox Order Traps

Load and store instructions may be issued from the IQ in a different order than they were fetched from the Icache, while architecturally, D-stream memory accesses to the same physical bytes must be completed in order. Generally, the Mbox manages the memory reference stream by itself to achieve architecturally correct behavior, but there are two cases in which replay traps are used to manage the memory stream.

The Mbox ensures that loads which reference the same physical byte(s) ultimately issue in order via the **load-load order trap**. The Mbox compares the address of each newly issued load to that of all loads in the load queue. If it finds a newer load instruction in the load queue then it invokes a load-load order trap on the newer instruction. This is a replay trap which aborts the target of the trap and all newer instructions from the machine and refetches instructions starting at the target of the trap.

The Mbox ensures that a load ultimately issues after an older store which writes some portion of the its memory operand via the **store-load order trap**. The Mbox compares the address of each newly issued store to that of all loads in the load queue. If it finds a newer load instruction in the load queue then it invokes a store-load order trap on the load instruction. This is a replay trap, just like the load-load order trap. The Ibox contains extra hardware to reduce the frequency of this trap. There is a one-bit by 1024-entry PC-indexed table in the Ibox called the **stWait** table. At Icache fetch time this table is accessed along with the Icache. The table produces one bit for each instruction accessed from the Icache. When a load instruction gets a store-load order replay trap its associated bit in the stWait table is set during the cycle that the load is re-fetched. Hence the trapping load's stWait bit will be set the next time it's fetched. The IQ will not issue load instructions whose stWait bit is set while there are older unissued stores in the queue. A load instruction whose stWait bit is set can issue the cycle immediately after the last older store issues from the queue. All the bits in the stWait table are unconditionally cleared every 16384 cycles.

2.4.2 Other Mbox Replay Traps

The Mbox also uses replay traps to flow control the load queue and store queue, and to ensure that is there are never multiple outstanding misses to different physical addresses which map to the same Dcache or Bcache line. Unlike the order traps, however, these replay traps are invoked on the incoming instruction which triggered the condition.

2.5 Software-Directed Prefetching and Loads into R31 & F31

This section describes how EV6 processes the various forms of load into R31/F31.

First, EV6 requires PALcode assistance to conform to ECO 95 - loads into R31/F31 may generate exceptions - these exceptions must be dismissed by PALcode.

2.5.1 Normal Prefetch: LDL, LDF, LDG, LDB, LDW

EV6 processes these instructions as "normal" cache line prefetches - if the load hits the Dcache, the instruction is dismissed, otherwise the addressed cache block is allocated into the Dcache.

2.5.2 Prefetch with Modify Intent: LDS

EV6 processes a LDQ into F31 as a prefetch with modify intent. If the load hits a dirty, modified DCache block the instruction is dismissed. Otherwise, the addressed cache block is allocated into the Dcache for write access - its dirty and modified bits are set.

2.5.3 Prefetch, Evict Next: LDQ

EV6 processes this like a "normal" prefetch, with one exception. If the load misses the Dcache, the addressed cache block is allocated into the Dcache, but the Dcache set allocation pointer is left pointing to this block. The next miss to the same Dcache line will evict the block. One example where this instruction might be used is when software is reading an array which is known to fit in the off-chip secondary cache, but will not fit in the on-chip Dcache. The use of the instruction in this case will ensure that hardware provides the desired prefetch function without displacing useful cache blocks stored the other set of the Dcache.

2.5.4 Prefetch, No Reuse: LDT

This instruction will indicate to EV6 that the addressed cache block will be accessed once and not accessed again for a long time. This instruction might be used when sweeping through the contents of an array which is known to be larger than the secondary cache, for example, and will inform EV6 to perform a cache line prefetch without displacing otherwise useful cache blocks.

EV6 will respond to this instruction as follows. If the load hit the Dcachc the instruction is dismissed. Otherwise the addressed cache block is fetched from the Bcache or memory, depending upon the result of the Bcache tag probe, and transmitted across EV6's internal data busses. This external reference will not result in a fill of either the Dcache or the Bcache, however. Any loads to the same cache block and which issue after the prefetch issues but before the block is transmitted across the internal busses will be satisfied when the prefetched block is transmitted across the internal data busses. Loads to this cache block which issue after the block is transmitted will miss the Dcache and result in another external read, either to memory or the BCache.

2.6 Special Cases

This section describes the mechanisms by which EV6 processes “irregular” instructions in the Alpha instruction set, or cases in which EV6 processes instructions in a non-intuitive way.

2.6.1 Load Hit Speculation

The latency of integer loads which hit in the Dcache is three cycles. Here is the pipeline timing:

Hit

cycle #	1	2	3	4	5	6	7	8
ILD	Q	R	E	D	B			
instr1				Q	R			
instr2					Q			

There are two cycles in which the IQ may speculatively issue instructions which consume load data before Dcache hit information is known. Any instructions which issue from the IQ within this two cycle "speculative window" are kept in the IQ with their requests inhibited until the load's hit condition is known, even if they are not dependent on the load. If the load hits then these instructions are removed from the queue. If the load misses then the execution of these instructions is aborted and the instructions are allowed to request service again. For example, in the above diagram, **instr1** and **instr2** are issued within the speculative window of the load. If the load hits then both instructions will be deleted from the queue by the start of cycle 7 - one cycle later than normal for **instr1** and at the normal time for **instr2.** If the load misses then both instructions are aborted from the execution pipelines and may request service again in cycle 6.

IQ-issued instructions are aborted if issued within the speculative window of an integer load which missed the Dcache, even if they are not dependent on the load. However, if software knows misses are likely, it can still benefit from scheduling the instruction stream for Dcache miss latency. EV6 includes a saturating counter which is incremented by load misses and decremented by load hits. When the upper bit of the counter is set the integer load latency is increased to five cycles, and the speculative window is removed. The counter is 5 bits wide, and increments by two on a miss and by one on a hit.

Since loads into R31 do not produce a result, they do not create a "speculative window" when they execute and therefore never waste IQ-issue cycles if they miss.

Floating loads which hit in the Dcache have a latency of four cycles. Here is the pipeline timing:

Hit

cycle #	1	2	3	4	5	6	7	8
FLD	Q	R	E	D	B			
instr1					Q	R		
instr2						Q		

For floating loads the speculative window is only one cycle wide, and FQ-issued instructions which issue within the speculative window of a missing floating load are only aborted if they depend on the load. For example, in the above diagram **instr1** is issued in the speculative window of the load. If it is not a consumer of the data returned by the load then it is removed from the queue at its normal time - just at the start of cycle 7. If it is dependent on the load data and the load hit it is removed from the queue one cycle later - at the start of cycle 8, while if the load missed then it is aborted from the Fbox pipeline and may request service again in cycle 7.

2.6.2 Floating Point Stores

Floating point store instructions are cloned and loaded into both the IQ and the FQ from the mapper. Each IQ entry contains a control bit called **fpWait**, which when set prevents that entry from asserting its requests. This bit is initially set for each floating store which enters the IQ unless it was the target of a replay trap. The instruction's FQ clone issues when its Ra register is about to become clean, resulting in its IQ clone's fpWait bit being cleared and allowing the IQ clone to issue and be executed by the Mbox. This mechanism ensures that floating stores are always issued to the Mbox along with their associated data without requiring the floating register dirty bits to be available within the IQ.

2.6.3 CMOV

For EV6, the Alpha CMOV instruction has three operands, and thus presents a special case. The required operation is to move either the value in register Rb or the value from the old physical destination register into the new destination register, based on the value in Ra. Since neither the mapper nor the Ebox and Fbox data paths are otherwise required to handle three operand instructions, the CMOV instruction is decomposed by the Ibox pipeline into two, two-operand instructions.

```
cmov    Ra,Rb -> Rc

cmov1   Ra,oldRc -> newRc1
cmov2   newRc1, Rb -> newRc2
```

The first instruction, **cmov1**, tests the value of Ra and records the result of this test in a 65th bit of its destination register, newRc1. It also copies the value of the old physical destination register, oldRc, to newRc1. The second instruction, **cmov2**, then copies either the value in newRc1 or the value in Rb into a second physical destination register, newRc2, based on the CMOV "predicate" bit stored in newRc1. In summary, the original CMOV instruction is decomposed into two dependent instructions which each consume a physical register from the free list.

In order to further simplify this operation the two component instructions of a CMOV instruction are driven through the mappers in successive cycles. Hence, if a given fetch line contains N CMOV instructions, it takes N+1 cycles to run that fetch line through the mappers. For example, the following fetch line:

```
add cmovx sub cmovy
```

results in the following three map cycles:

```
add cmovx1
cmovx2 sub cmovy1
cmovy2
```

Integer CMOVs are executed as two distinct one-cycle latency operations by the Ebox.

Floating CMOVs are executed as two distinct four-cycle latency operations by the Fbox add pipeline.

2.7 Instruction Issue Rules

This section defines instruction classes, the functional unit pipelines to which they are issued, and their associated latencies.

2.7.1 Instruction Class Definitions

The table below defines instruction classes as they apply to the issue rules, and for each class specifies which of the functional unit pipelines execute those instructions.

Class Name	Pipeline	Instruction List
ild	L0, L1	all integer loads
fld	L0, L1	all floating loads
ist	L0, L1	all integer stores
fst	FST0, FST1, L0, L1	all floating stores
lda	L0,L1,U0,U1	LDA, LDAH
mem_misc	L1	WH64, ECB, WMB
rpcc	L1	RPCC
rx	L1	RS, RC
mxpr	L0, L1 (depends on IPR)	HW_MTPR,HW_MFPR
ibr	U0, U1	integer conditional branches
jsr	L0	BR, BSR, JMP, CALL, RET, COR, HW_RET, CALL_PAL
iadd	L0, U0, L1, U1	opcode 10_{16}, except CMPBGE
ilog	L0, U0, L1, U1	AND, BIC, BIS, ORNOT, XOR, EQV, CMPBGE
ishf	U0, U1	opcode 12_{16}
cmov	L0, U0, L1, U1	integer CMOV - either clone
imul	U1	integer multiplies
imisc	U0	LOC, TOC, POPC, PERR, MINxxx, MAXxxx, PKxx,UNPKxx
fbr	FA	floating conditional branches
fadd	FA	all floating operates except multiply, divide, square root and conditional move
fmul	FM	floating multiply
fcmov1	FA	floating CMOV - first half
fcmov2	FA	floating CMOV - second half
fdiv	FA	floating divide
fsqrt	FA	floating square root
nop	none	TRAP, EXCB, UNOP - LDQ_U R31, 0(Rx)
ftoi	L0, L1	FTOIS, FTOIT
itof	L0, L1	ITOFS, ITOFF, ITOFT
mx_fpcr	FM	move from floating point control register

2.7.2 Ebox Slotting

Instructions which issue from the IQ and could execute in either upper or lower Ebox subclusters are slotted to one pair or the other during the map stage of the pipeline, based on the instruction mix in the fetch line. These slotting rules are defined in the table below. In the type column, "U" means the instruction only executes in an upper subcluster, "L" means the instruction only executes in a lower subcluster, and "E" means the instruction could execute in either an upper or lower subcluster. The numbers 3,2,1 and 0 identify each instruction's location in the fetch line by the value of bits <3:2> of its PC.

Instruction Type	Slotting	Instruction Type	Slotting
3210	3210	3210	3210
EEEE	ULUL	LLLL	LLLL
EEEL	ULUL	LLLU	LLLU
EEEU	ULLU	LLUE	LLUU
EELE	ULLU	LLUL	LLUL
EELL	UULL	LLUU	LLUU
EELU	ULLU	LUEE	LULU
EEUE	ULUL	LUEL	LUUL
EEUL	ULUL	LUEU	LULU
EEUU	LLUU	LULE	LULU
ELEE	ULUL	LULL	LULL
ELEL	ULUL	LULU	LULU
ELEU	ULLU	LUUE	LUUL
ELLE	ULLU	LUUL	LUUL
ELLL	ULLL	LUUU	LUUU
ELLU	ULLU	UEEE	ULUL
ELUE	ULUL	UEEL	ULUL
ELUL	ULUL	UEEU	ULLU
ELUU	LLUU	UELE	ULLU
EUEE	LULU	UELL	UULL
EUEL	LUUL	UELU	ULLU
EUEU	LULU	UEUE	ULUL
EULE	LULU	UEUL	ULUL
EULL	UULL	UEUU	ULUU
EULU	LULU	ULEE	ULUL
EUUE	LUUL	ULEL	ULUL
EUUL	LUUL	ULEU	ULLU
EUUU	LUUU	ULLE	ULLU
LEEE	LULU	ULLL	ULLL
LEEL	LUUL	ULLU	ULLU
LEEU	LULU	ULUE	ULUL
LELE	LULU	ULUL	ULUL
LELL	LULL	ULUU	ULUU
LELU	LULU	UUEE	UULL
LEUE	LUUL	UUEL	UULL
LEUL	LUUL	UUEU	UULU
LEUU	LLUU	UULE	UULL
LLEE	LLUU	UULL	UULL
LLEL	LLUL	UULU	UULU
LLEU	LLUU	UUUE	UUUL
LLLE	LLLU	UUUL	UUUL
		UUUU	UUUU

2.7.3 Instruction Latencies

After an instruction is placed in the IQ or FQ, its issue point is determined by the availability of its register operands, functional unit(s), and relationship to other instructions in the queue. There are register producer-consumer dependencies and dynamic functional unit availability dependencies which affect instruction issue. The mapper removes register producer-producer dependencies.

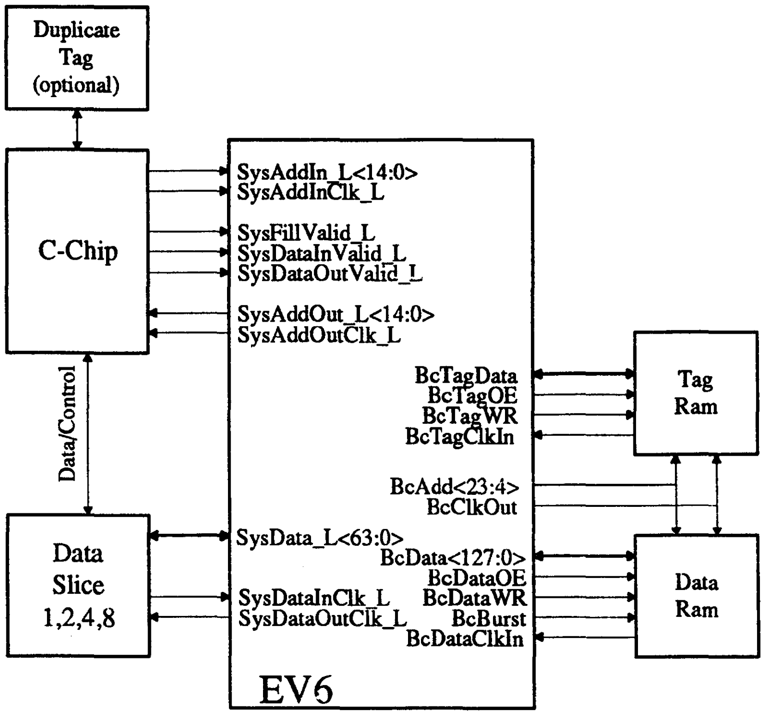
The latency to produce a register result is generally fixed. The exception is for loads which miss the Dcache.

Class	Latency	Comments
ild	3	Dcache hit
	13+	Dcache miss, latency with 6-cycle Bcache. Add additional bcache loop latency if bcache is slower than 6 cycles.
fld	4	Dcache hit
	8+	Dcache miss, latency with 0-cycle Bcache. Add Bcache loop latency.
ist		doesn't produce register value
fst		doesn't produce register value
rpcc	1	possible 1 cycle cross cluster delay
rx	1	
mxpr	1 or 3	HW_MFPR. Ebox IPRs: 1. Ibox & Mbox IPRs: 3. HW_MTPR doesn't produce a register value.
icbr		conditional branch; doesn't produce register value
ubr	3	unconditional branch
jsr	3	
iadd	1	possible 1-cycle Ebox cross-cluster delay
ilog	1	possible 1-cycle Ebox cross-cluster delay
ishf	1	possible 1-cycle Ebox cross-cluster delay
cmov1	1	only consumer is **cmov2**. possible 1-cycle Ebox cross-cluster delay
cmov2	1	possible 1-cycle Ebox cross-cluster delay
imul	7	possible 1-cycle Ebox cross-cluster delay
imisc	3	possible 1-cycle Ebox cross-cluster delay
fcbr		doesn't produce register value
fadd	4	consumer other than **fst** or **ftoi**
	6	consumer **fst** or **ftoi.** measured from **fadd** issuing from FQ to **fst** or **ftoi** issuing from IQ
fmul	4	consumer other than **fst** or **ftoi**
	6	consumer **fst** or **ftoi.** measured from **fmul** issuing from FQ to **fst** or **ftoi** issuing from IQ
fcmov1	4	only consumer is **fcmov2**
fcmov2	4	consumer other than **fst**
	6	consumer **fst** or **ftoi.** measured from **fcmov2** issuing from FQ to **fst** or **ftoi** issuing from IQ
fdiv	12	single precision - latency to consumer of result value
	10	single precision - latency to using divider again
	15	double precision - latency to consumer of result value
	13	double precision - latency to using divider again
fsqrt	16	single precision - latency to consumer of result value
	14	single precision - latency to using unit again
	32	double precision - latency to consumer of result value
	30	double precision - latency to using unit again
ftoi	3	

Class	Latency	Comments
itof	4	
nop		doesn't produce register value

3. External Interface

The external interface consists of two ports - a Bcache port and a System port. The Bcache port is controlled entirely by the processor, and is used to interface to a module-level secondary cache which may be built from a range of standard synchronous SRAMs. The System port interfaces to the rest of the System. The processor contains two external data busses, one 16-bytes wide for the Bcache and the other 8-bytes wide for the System.



3.1 Address Spaces

EV6 supports a 44-bit physical address space which is divided equally between Memory space and IO space. Memory space resides in the lower half of the physical address space (PA<43> clear) and IO space resides in the upper half of physical address space (PA<43> set). EV6 recognizes these spaces internally.

EV6-generated external references to Memory space are always of a fixed 64-byte size, though the internal access granularity is byte, word, longword or quadword. All EV6-generated external references to Memory or I/O space are physical addresses that are either successfully translated from a virtual address

or produced by PAL code. On rare occasions, speculative execution may cause a reference to non-existent memory. Systems must range check all addresses and report those events to EV6. See section 6.3.8.

EV6 does not cache IO space data, however it merges both reads and writes and supplies a mask to indicate the bytes which are actually accessed. EV6 merges IO space LW and QW Loads and Stores into Reads or Writes of up to 32 or 64 bytes respectively. Systems may limit I/O QW writes to 32 bytes maximum by setting the TLASER_STIO_MODE bit in the CBOX csr. Byte and word operations to I/O space are never merged in EV6. All LDB,LDW,STB and STW instructions (PA<43>set) generate an unique interface command. Finally, references of differing sizes are not merged.

3.1.1 I/O Ordering and Merge Rules

EV6 will adhere to the following rules of order when executing LD and ST instructions to I/O

1. Consecutive Loads from I/O space happen in the order specified by the programmer.
2. Consecutive Stores to I/O space happen in the order specified by the programmer.
3. Loads followed by stores to the same address (within the same 64 byte block) happen in the order specified by the programmer.
4. Stores followed by Loads to the same address (within the same 64 byte block) happen in the order specified by the programmer.
5. Loads followed by Stores (and vice versa) to different addresses (bits <43:6> not equal) may be UNORDERED and software can not depend on one occurring first.

The following matrix illustrates I/O merging rules in EV6. The intersection of two consecutive I/O operations contains the rule observed by EV6. Reads and writes will merge in ascending order only (obeys default ordering of a PCI device). Finally, merging can be terminated with a timer set to TBD CPU cycles. Collapsing (multiple I/O writes to the same location) does not occur in EV6.

	BYTE/WORD	LONGWORD	QUADWORD
BYTE/WORD	No Merge	No merge	No Merge
LONGWORD	No Merge	Merge up to 32 Bytes	No Merge
QUADWORD	No merge	No merge	Merge Up to 64 Bytes

A CBOX IPR mode bit that effect merging to I/O space is TLASER_STIO_MODE. When asserted will limit stores to I/O space to 32 bytes.

3.2 Cache Organization and Coherence

The EV6 cache hierarchy has the following attributes:

- I-stream data from both IO and Memory space may be cached in the Icache. Icache coherence is not maintained by hardware - it must be maintained by software using the IMB instruction.
- D-stream Memory space data may be cached in the Bcache and Dcache. EV6 ensures that the Dcache contents are a subset of the Bcache. This allows Memory requests from other agents in the System to be filtered using only a duplicate copy of the Bcache tags; external duplicate Dcache tags are not required.
- In Systems which use a Bcache, a Bcache duplicate tag store may be used to filter requests, but this is not required.
- System hardware is required to cooperate with EV6 to ensure coherence of the Bcache and Dcache.

3.2.1 Cache Block States

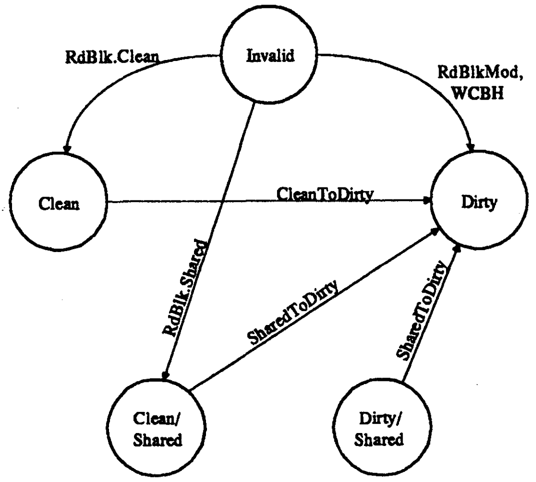
EV6 supports the following cache block states:

State Name	Description
Invalid	
Clean	This processor holds a copy of the block, but no other agent in the System holds a copy.
Clean/Shared	This processor and at least one other agent in the System may hold a copy of the block.
Dirty	This processor may write to the block and must write it to Memory after it's evicted from the cache. No other agent in the System holds a copy of the block.
Dirty/Shared	The dirty block may be shared - this processor must write it back to Memory when it's

evicted. The block may not be written by this processor.

3.2.2 Cache Block State Transitions

Cache block state transitions may be triggered by EV6-generated commands to the System or by System-generated commands to EV6. The latter are called **probes**. The diagram below shows the cache state transitions which are triggered by EV6's actions.



EV6 issues two types of reads to Memory space: **RdBlk** and **RdBlkMod.** EV6 will mark a cache block from a RdBlk command sent to the System either CLEAN or CLEAN/SHARED or even DIRTY depending on the response from the System during the cache fill. Systems can not transition a CLEAN block to DIRTY or DIRTY/SHARED with a probe command.

EV6 will send a **ChangeToDirty** command to the System against a block not in the DIRTY state when it wants to write that block. There are two types of ChangeToDirty commands which EV6 may use based on the initial state of the cache block: **CleanToDirty** and **SharedToDirty**. Having two flavors of ChangeToDirty relieves Systems with a duplicate tag and address CAM from having to do a read-modify-write of the tag store in response to the ChangeToDirty command. Also, Systems need not generate a System bus invalidate in response to a CleanToDirty command.

EV6 will send an **InvalToDirty** command to the System in response to the execution of a WCBH instruction (when enabled with INVALTODIRTY_ENABLE csr) which does not hit on a Bcache block (if the block is valid, it defaults to the ChangeTodirty command rules). This will cause the block to transition to the DIRTY state, and other agents in the System should invalidate their copies of the block. There is no data movement associated with this command. A success response can be from Systems by using the ChangeToDirty Success command encoding in SysDc<4:0>. See section 1.3.8 for details on data transfer commands that include responses to ChangeToDirty type commands.

EV6 will send a **WrVictimBlk** command to the System when evicting a dirty or dirty/shared cache block, and may also be configured to send a **CleanVictimBlk** to the System when evicting a clean or shared block.

The System sends probe commands to EV6 both to invoke data movement from EV6 to the System, and to change cache block states. Systems with duplicate tags can directly specify the cache block state transition which should occur, while Systems without duplicate tags specify a "transition type" which is combined with the results of the probe to determine the final state transition.

3.2.3 System Knowledge of Bcache Contents

EV6 will support Systems both with and without specialized hardware apparatus that track the state of the Bcache, and will take different actions on this basis. There are two principal differences related to the cache coherence protocol.

Systems with duplicate Bcache tags or Memory resident directory maps only send probes to EV6 for cache blocks that are relevant (Bcache hit). These Systems know the final state of the cache block and can specify it. Ending status is not conditioned by the probe lookup in EV6. In contrast, Systems that do not have knowledge of internal Bcache status do not know the result of the probe in advance, so both data movement and cache block state transitions are conditioned by the results of the probe.

Systems with duplicate Bcache tags or Memory resident directory maps will require CleanToDirty commands sent to the System port by EV6 to keep the external status tracking hardware up-to-date. This is not necessary in non-duplicate tag Systems. SharedToDirty commands to Shared blocks in either type of System must result in bus invalidates, and thus always appear on the System port.

3.2.4 Dcache States & the Dcache Duplicate Tags

Each Dcache block contains an extra state bit beyond those required to support the cache protocol. The **modified** bit, when set, indicates that the associated block should be written to the Bcache when it's evicted from the Dcache. The modified bit is set in two cases:

1. When a block is filled into the Dcache from Memory its modified bit is set, ensuring that it also gets filled into the Bcache.
2. When the processor writes to a dirty Dcache block the modified bit is set, indicating it should be written to the Bcache when evicted.

The DTAG array holds a physically indexed duplicate copy of the Dcache tags. Since the Dcache contains 64KB and is virtually indexed, a given physical address could reside in any one of eight places in the cache. The Cbox uses the DTAGS for the following situations.

1. When the Mbox requests a Dcache fill, the Cbox uses the DTAGS to see if the Dcache already contains the requested physical address in another virtually indexed Dcache line. If so, the Cbox invalidates that cache line after first writing the data back to the Bcache if it was in the modified state. The Cbox also checks to see if the Dcache contains an address different from the requested address but which maps to the same Bcache line. If so, the Dcache line is evicted in order to keep the Dcache a subset of the Bcache.
2. When the Ibox requests an Icache fill, the Cbox uses the DTAGS to see if the Dcache contains the requested physical address in the modifed state. If so, the Cbox forces the line to be written back to the Bcache before servicing the Icache fill request. The Cbox also checks to see if the Dcache contains an address different from the requested address but which maps to the same Bcache line. In this case the I-stream request will miss the Bcache, and the Cbox will service the request by launching a noncached fetch to the System port and will not put the I-stream block into the Bcache. This mechanism allows EV6 to use a cache resident "lock flag" for LDx_L/STx_C instructions.
3. The Cbox uses the DTAGS to determine whether probe addresses are held in the Dcache.

3.2.5 Memory Barrier (MB/WMB/TBfill flow)

There is a mode bit called SYS_MB in the CBOX Control CSR which controls whether MB instructions produce external System port transactions. EV6 will need to generate System port MB transactions in Systems which allow READ and ChangeToDirty responses to reach EV6 ahead of system probes (out of order with respect to the order that transactions reached the System's serialization point). An external system MB command is required in systems which do not compare incoming addresses as well as allow refills to be seen by EV6 out of order with respect to the order that commands reached the system serialization point.

A counter exists in the CBOX that contains the number of pending uncommited transactions. The counter will increment for the following commands:

- RdBlk, RdBlkMod, RdBlkI,
- valid RdBlkSpec, valid RdBlkModSpec, valid RdBlkSpecI,
- RdBlkVic, RdBlkModVic, RdBlkVicI
- CleanToDirty, SharedToDirty, STChangeToDirty, InvalToDirty
- FetchBlk, valid FetchBlkSpec, Evict, RdByte, RdLw, RdQw

The counter is decremented with the C (commit) bit in the Probe and SysDc commands described in Section 3.3.7. Systems can send the C bit in the SysDc fill-response to the commands which increment the counter or on the last probe seen by that command when it reached the system serialization point.

When an MB instruction is fetched, it stalls in the map stage of the pipeline. This also stalls all instructions after the MB until:

1. If SYS_MB is *clear* the EV6 CBOX waits for the integer issue queue to empty and performs the following actions: Sends all pending miss address file (MAF) and WRIO entries to the system port
   - Monitors a 4-bit counter of outstanding committed events. When the counter decrements from one to zero, CBOX marks the youngest probe queue entry
   - Waits until the miss address file contains no more dstream references, the store queue, load queue and I/O write buffers are empty

When all above have occurred and a probe response has been sent to the system for the marked probe queue entry, instruction execution continues with the instruction after the MB.

2. If SYS_MB is *set*, the EV6 CBOX performs the following actions:Sends all pending MAF entries to the system port,
   - Sends the MB command to the System port,
   - Waits until the MB command is acknowledged and marks the youngest entry in the probe queue
   - Waits until the miss address file contains no more dstream references, the store queue, load queue and I/O write buffers are empty

When all above have occurred and a probe response has been sent to the system for the marked probe queue entry, instruction execution continues with the instruction after the MB. Write Memory Barriers (WMB's) are issued into the MBOX store-queue, wait until they are retired and become writeable, and when the writeable pointer reaches the WMB, the MBOX freezes the writeable pointer and informs the CBOX. The CBOX closes the write buffer and responds based on SYS_MB.If SYS_MB is *clear* the EV6 CBOX performs the following actions:Marks the youngest entry in the probe queue

When a probe response has been sent to the system for the marked probe queue entry, the MBOX unfreezes and advances the writeable pointer.

If SYS_MB is *set* the EV6 CBOX performs the following actions:Sends the MB command to the System port,
   - Waits until the MB command is acknowledged and marks the youngest entry in the probe queue

When a probe response has been sent to the system for the marked probe queue entry, the MBOX unfreezes and advances the writeable pointer.Loads to a virtual page table entry (HW_LD/VPTE) are processed by EV6 so as to avoid litmus test problems associated with the ordering of memory accesses from another processor against load of a page table entry and the subsequent virtual-mode load from this processor. Consider the following :

P_i	P_j

Wr $Data_i$	LD/ST $Data_i$
MB	<TB Miss>
Wr PTE_i	LD-PTE
	<wr TB>
	LD/ST (restart>

P_j must get the updated $Data_i$ if it got the updated PTE_i. Also consider the related

P_i	P_j
Wr $Data_i$	I-stream read $Data_i$
MB	<TB Miss>
Wr PTE_i	LD-PTE
	<wr TB>
	I-stream read (restart) - will miss the Icache

In this case the Data could be cached in the Bcache; P_j should fetch $Data_i$ if it is using PTE_i. EV6 processes dstream loads to the page table entry by injecting, in hardware, some memory barrier processing between the access of the page table entry and any subsequent load or store. This is accomplished by the following mechanism:Integer queue issues a HW_LD/VPTE

- Integer queue issues a HW_MTPR DTB_PTE0 which is data-dependent on the HW_LD/VPTE and is required in order to fill the DTB's. The HW_MTPR, when enqueued, set IPR scoreboard bits <4> and <0>.
- On issue of HW_MTPR DTB_PTE0, IBOX signals CBOX that a HW_LD/VPTE has been processed and causes CBOX to begin "MB" processing. IBOX prevents issue of any subsequent memory operations by not clearing the IPR scoreboard bit <0> (one of the scoreboard bits associated with the HW_MTPR DTB_PTE0).
- When "MB" processing is complete (one of the above sequences, depending on SYS_MB), CBOX signals IBOX to clear IPR scoreboard bit <0>.

EV6 processes TB niss fills to the page table entry via a similar mechanism:Integer queue issues a HW_LD/VPTE

- Integer queue issues a HW_MTPR ITB_PTE which is data-dependent on the HW_LD/VPTE and is required in order to fill the ITB. The HW_MTPR, when enqueued, set IPR scoreboard bits <4> and <0>.
- On issue of HW_MTPR ITB_PTE, IBOX signals CBOX that a HW_LD/VPTE has been processed and causes CBOX to begin "MB" processing. The MBOX stalls off any IBOX fetching from the time that the HW_LD/VPTE finishes until the probe queue is drained.
- When "MB" processing is complete (one of the above sequences, depending on SYS_MB), CBOX signals IBOX to clear IPR scoreboard bit <0>. In addition, the MBOX signals the IBOX to begin fetching.

3.2.6 Load/Locked and Store/Conditional

EV6 doesn't contain a dedicated lock register, nor are System components required to do so. When a LDx_L instruction executes, data is accessed from the D or Bcache. If there is a cache miss, data is accessed from memory with a RdBlk command. . When the store-conditional executes, it is allowed to succeed if its associated cache line is still present in the Dcache and can be made writeable, otherwise it fails. This works since if another agent in the System wrote to the cache line between the load-lock and the store conditional then the cache line would have been invalidated. There are a host of further complications however.

Problem	A load-lock and its matching store-conditional must issue in program order.
Solution	The **stWait** logic in the IQ is used to ensure that a store conditional always issues after an older load-lock. The stWait logic treats load-locks like stores, and store conditionals are always loaded into the IQ with their associated stWait bit set.
Problem	I-stream references can't evict the locked cache line.
Solution	If an Icache fill request misses the Bcache but maps to the same Bcache line as an address which is held in the Dcache, then the I-stream request is sent to the System port as a non-cached fetch, and the I-stream line is not allocated into the Bcache.
Problem	Loads or stores that are older than the load-lock but issue after it can't evict the locked cache line.
Solution	The Mbox recognizes this case and invokes a replay trap on the incoming load or store, which also aborts the load-lock. These instructions issue in program order the next time down the pipe.
Problem	If the instruction fetcher predicts that a branch between a load-lock and a store conditional will be taken, and the branch is not taken, then a load or store executed on this mispredicted path can't evict the locked cache line.
Solution	There is a bit in the instruction fetcher which is set on a load-lock and cleared on any other Memory reference instruction. When this bit is set the branch predictor forces all branches to be predicted as fall through.
Problem	Loads or stores which are newer than the store-conditional can't evict the locked line
Solution	The Ibox ensures that a store-conditional issues before any newer load or store by placing the store-conditional into the IQ and stalling all subsequent instructions in the map stage of the pipe until the IQ is empty. This allows the Mbox to prevent newer loads and stores from evicting the cache line associated with the store conditional.
Problem	If two store-conditionals execute without an intervening load-lock, the second store-conditional must always fail. (Store conditionals to I/O will ALWAYS succeed)
Solution	The register map logic contains a bit which is set by load-locks and cleared by store-conditionals. If the bit is cleared when a store conditional instruction is mapped, then the store-conditional is forced to fail. The mapper updates the value of the bit as appropriate when pipeline aborts occur.
Problem	There must be no live-lock conditions in multiprocessor Systems.
Solution	If a store conditional misses the Dcache then no System port transaction is launched, and the store conditional fails. If the store conditional hits a block which isn't dirty, then a ChangeToDirty is launched only after the store conditional instruction retires and all older store queue entries are in the writable state . This ensures that once the ChangeToDirty is launched on behalf of the store-conditional that the store conditional will be executed to completion if the ChangeToDirty passes. If the ChangeToDirty passes, the store-conditional enters the writable state, and the Mbox locks down the Dcache line and does not release it until the store-conditional's data is transferred into the Dcache. If the Cbox launches a CleanToDirty command for the locked block to the System port and another agent reads the block before the CleanToDirty hits the serialization

point in the System, then the System will cause the CleanToDirty to fail.
In this case EV6 will launch a SharedToDirty command to the System against the locked block. This ensures that other agents do not cause the store-conditional to fail just by reading the locked block.

3.3 System Port

The System port is EV6's connection to either a local Memory/IO controller or a shared multiprocessor system controller. The System port consists of two uni-directional address and command busses (SysAddIn<14:0>, SysAddOut<14:0>), a bi-directional data bus (SysData<63:0>, SysCheck<7:0>), single-ended uni-directional clocks, and a few control pins. All SysAdd and SysData signals are driven from EV6 with low assertion levels. Systems must receive and drive low asserted signals.

3.3.1 System Port Pins

Pin Name	type	Count	Description
SysAddIn<14:0>_L	I	15	time-muxed Command/Address/ID/Ack System to EV6 bus
SysFillValid_L	I	1	validation for fill given in previous SysDC command
SysAddInClk_L	I	2	single-ended forwarded clock from System for above signals
SysAddOut<14:0>_L	O	15	time-muxed Command/Address/ID/Mask EV6 to System Bus
SysAddOutClk_L	O	2	single-ended forwarded clock output for above signals
SysData<63:0>_L	B	64	data bus for Memory and IO data
SysCheck<7:0>_L	B	8	QW ECC check bits for SysData
SysDataInClk_L	I	8	8 System generated clocks for clock forwarded SysData in
SysDataOutClk_L	O	8	8 EV6 generated clocks for clock forwarded SysData out
SysDataInValid_L	I	1	Marks a valid data cycle for data transfers to EV6 when asserted
SysDataOutValid_L	I	1	Marks a valid data cycle for data transfers from EV6 when asserted

3.3.1.1 Legend: I= input, O = output, B= Bi-directional

3.3.2 EV6 to System Address/Command Format

Command, Address, ID and Mask are sent in four consecutive cycles. EV6 can be configured to send two different combinations of PA bits in the four cycle command , the goal being to give the System the PA bits that let it do Memory bank select and RAS address drive as fast as possible. The ID is the miss address file (MAF), victim buffer or IO write buffer number associated with the command. The mask indicates the accessed bytes, longwords or quadwords for an IO space reference. Commands with Victims are sent as an atomic pair of standard format commands, if the CBOX IPR bc_rdvictim is set.

3.3.2.1 Bank Interleave On Cache Block Boundary

	SysAddOut<14:2>				SysAddOut<1>	SysAddOut<0>
Cycle 1	M1	Command<4:0>	PA<34:28>		PA<36>	PA<38>
Cycle 2	PA<27:22>, PA<12:6>				PA<35>	PA<37>
Cycle 3	M2	Mask<7:0>	CH	ID<2:0>	PA<40>	PA<42>
Cycle 4	RV	PA<21:13> , PA<5:3>			PA<39>	PA<41>

3.3.2.2 *Page Mode Hit*

	SysAddOut<14:2>				SysAddOut<1>	SysAddOut<0>
Cycle 1	M1	Command<4:0>	PA<31:25>		PA<32>	PA<33>
Cycle 2	PA<24:12>				PA<11>	PA<34>
Cycle 3	M2	Mask<7:0>	CH	ID<2:0>	PA<35>	PA<37>
Cycle 4	RV	PA<34:32>, PA<11:3>			PA<36>	PA<38>

Field definitions are:

SysAddOut Field	Definition
M1	reports a miss to the System for the oldest probe when =1. Has no meaning when = 0.
Command<4:0>	the five bit command field
SysAddOut<1:0>	is only required for Systems with greater than 32 Gbyte (up to a maximum of 8 Terabyte) memories. This will allow cost focused systems to use a 13 bit command/address field.
M2	reports a miss to the System for the oldest probe when = 1,additionally it is asserted for Invalidates or set shared commands that have no data movement. M2 has no meaning when = 0. Assertion of both M1 and M2 will not occur. (Reporting probe results is timing critical so when a result is known, EV6 will take the earliest opportunity to send a M signal to the system. M bit assertion can occur either in a valid command or a NZNOP)
ID<2:0>	the MAF, VDB, or Write I/O buffer id number associated with the command
RV	validates this command, in (optional) speculative read mode RV= 1 validates the command and RV=0 is a NOP. RV is a 1 for all non-speculative commands.
Mask<7:0>	the byte, LW or QW mask field for corresponding I/O commands
CH	cache hit bit that is asserted along with M2 when probes with no data movement hit in the D or B cache. A probe with no data movement can be an Invalidate or a ReadifDirty that hits on a valid but clean or shared block.

3.3.3 SysAdd Commands Generated by EV6

Command	Command <4:0>	Function
Nop	00000	EV6 drives this on idle cycles
ProbeResponse	00001	Returns probe status and Victim Buffer number holding the requested cache block.
NZNOP	00010	Non_zero NOP, helps parse command packet
VDBFlushRequest	00011	Victim Data Buffer Flush Request. EV6 sends this command to the System when an internally generated reference hits a Bcache victim

Command	Command <4:0>	Function
		or Probe in the VDB. The System should flush VDB entries associated with all probes and WrVictimBlks which occurred before this command.
MB	00111	Indicates a MB was issued, optional when SYS_MB is set
RdBlk	10000	Memory Read
RdBlkMod	10001	Memory Read, modify intent
RdBlkI	10010	Memory Read for I-stream, optional
FetchBlk	10011	Memory Uncached RdBlk
RdBlkSpec	10100	Memory Read speculative, optional
RdBlkModSpec	10101	Memory Read, speculative, modify intent,optional
RdBlkSpecI	10110	Memory Read for I-stream, optional
FetchBlkSpec	10111	Memory Uncached RdBlk, speculative
RdBlkVic	11000	Memory Read with a victim- optional
RdBlkModVic	11001	Memory Read, modify intent, victim - optional
RdBlkVicI	11010	Memory Read for I-stream with a victim - optional
WrVictimBlk	00100	Writeback of Dirty Block
CleanVictimBlk	00101	Address of a Clean Victim, optional mode used in directory Systems
Evict	00110	Duplicate Tag Invalidate, optional
RdBytes	01000	IO Read, Byte mask
RdLWs	01001	IO Read, LW mask
RdQWs	01010	IO Read, QW mask
WrBytes	01100	IO Write, Byte mask
WrLWs	01101	IO Write, LW mask
WrQWs	01110	IO Write, QW mask
CleanToDirty	11100	Sets a block dirty that was previously Clean, optional for duplicate Tags
SharedToDirty	11101	Sets a block dirty that was previously Shared, optional for MP Systems
STCChangeToDirty	11110	Sets a block dirty that was previously Clean or Shared for a STx_C , optional for MP Systems
InvalToDirty	11111	Acts like a RdBlkMod without the fill cycles, optional for MP Systems InvalToDirty has a victim - optional

Systems can optionally enable RdBlkVic and RdBlkModVic commands . In this mode the RdBlkxVic command cycles are always followed immediately by the WrVictimBlk commands. Also, when CleanVictimBlk commands are enabled they immeditaely follow RdBlkVic and RdBlkModVic commands. Speculative Rds in RdBlk victim mode will not create victims, this is useful for TurboLaser and TurboLaser follow-ons.

3.3.4 Probe Response Transfers

EV6 responds to System probes that did not miss with a four cycle transfer on the SysAddOut bus. The format of the probe response is shown below:

	SysAddOut<14:2>						SysAddOut<1>	SysAddOut<0>
Cycle 1	0	00001	Status<1:0>	DM	VS	VDB <2:0>	X	X
Cycle 2	0				MS	MAF <2:0>	X	X
Cycle 3	0	X					X	X
Cycle 4	X						X	X

Probe Response Field	Description
Command<4:0>	Identifies transfer as probe response
DM	Indicates that data movement should occur (copy of Probe valid bit)
VS	Write Victim Sent bit
VDB<2:0>	VDB (Victim Data Buffer) entry containing the requested cache block. this field is valid when either the DM bit or the VS bit = 1
MS	MAF address sent
MAF<2:0>	MAF entry which matched against the probe address
Status<1:0>	Result of probe:
	00 HitClean
	01 HitShared
	10 HitDirty
	11 HitSharedDirty

The System retrieves data from EV6 for probes that requested a cache block by using the SysDC wires. Probes which respond with M1 or M2 set will never be reported to the System in a Probe Response command.

3.3.5 SysAck & System Port Flow Control

Flow control of EV6-generated System port commands is done via the "A" bit , which is driven by the System, and a counter internal to EV6. EV6 increments its "command outstanding" counter every time it sends a command to the System. It increments this counter by two for RdBlkVic commands. EV6 decrements the counter by one each time the System asserts "A" (SysAddIn<14> cycle 4 of the probe command or cycle 2 of the SysDc command). EV6 stops sending new commands when the counter hits the maximum count specified by the sysbus_ack_limit field in the CBOX IPR. EV6 will not send a RdBlkxVic command if the counter is equal to one less than the maximum outstanding count. There is no mechanism for the System to reject a command that has been sent. ProbeResponse, VDBFlushReq, NOP, NZNOP and a RdBlkxSpec with a clear RV bit will not increment the "command outstanding" counter and will therefore not require an "ACK" from the system. Systems must provide adequate resources for responses to all probes sent to EV6. Additionally, there is a CBOX IPR that when set will not increment the outstanding command counter for RdBlkVic, RdBlkModVic and RdBlkVicI command. This is the "rdvic_ack_inhibit" bit.

3.3.6 SysReadValid and Speculative Reads

Systems can configure EV6 to send Memory space RdBlkSpec and RdBlkModSpec commands before EV6 has determined that the read has missed the Bcache. SysAddOut<14> of the fourth command address cycle contains the RV bit for that transaction. When configured for speculative reads, RV=0 indicates a NOP for that command and RV=1 validates that command. Systems not opting for speculative reads will always have RV=1. A RdBlkSpec or RdBlkModSpec with a clear RV bit will not increment the outstanding command counter and therefore systems must not send an ACK to EV6 for these commands.

3.3.7 SysAdd Commands Generated by the System

The commands driven by the System to EV6 are generically called probes and data movement commands. There are two formats for the SysAddIn bus that specify probes and data movement. Probes are always 4 cycle commands that also contain a field to include a valid SysDc command. The format of the four cycle command is shown below. Note that SysAddIn<1:0> are optional and are used for Memory designs greater than 32 Gbytes. The position of the address bits matches the selected format of the SysAddOut bus. The example below shows the bank interleave format.

	SysAddIn<14:2>						SysAddIn<1>	SysAddIn<0>
Cycle 1	1	Probe<4:0>	PA<34:28>				PA<36>	PA<38>
Cycle 2	PA<27:22>, PA<12:6>						PA<35>	PA<37>
Cycle 3	0	SysDc<4:0>	RVB	RPB	A	ID<3:0>	PA<40>	PA<42>
Cycle 4	C	PA<21:13>, PA<5:3>					PA<39>	PA<41>

SysAddIn Field	Description
Probe<4:0>	Probe Type and Next Tag State (See table below)
SysDc<4:0>	Controls data movement in out of EV6, See section 3.3.7 for details
RVB	Clears Victim Buffer or WRIO buffer valid bit specified in ID<3:0>
RPB	Clears Probe Buffer valid bit specified in ID<2:0>
A	Command Ack bit that decrements EV6 command outstanding counter
ID<3:0>	Identifies VDB number or WRIO buffer number, <3> is asserted for WRIO only
C (COMMIT)	Commit bit that decrements the uncommitted event counter used for Memory Barrier acknowledge.

The command field of a probe has 2 fields - the first sets the data movement, the second determines the next cache block state.

Probe<4:3>	Data Movement Function
00	Nop
01	Read if Hit, supply data to system if block is valid
10	Read if Dirty, supply data to system if block is valid/dirty
11	Read Anyway,supply data to system at index of probe

Probe<2:0>	Next Tag State
000	Nop
001	Clean
010	Clean/Shared
011	Transition3: Clean->Clean/Shared, Dirty->Invalid Dirty/Shared->Clean/Shared

100	Dirty/Shared
101	Invalid
110	Transition1: Clean->Clean/Shared, Dirty->Dirty/Shared
111	Transition2: Clean->Clean/Shared, Dirty->Clean/Shared

Next Tag State notes:

Transition1 is useful in non-duplicate tag Systems that do not update Memory on RdBlk hits to a dirty block.

Transition2 is useful in non-duplicate tag Systems that update Memory on RdBlk hits to a dirty block.

Transition3 is useful in non-duplicate tag Systems that want to give writeable status to the reader and do not know if the block is clean or dirty.

EV6 holds pending probe commands in a 8 entry deep probe queue. The System must keep track of how many probes were sent and not overrun EV6's queue. Probes are removed from the internal probe queue when the probe response is sent.

3.3.8 Two Cycle Commands For Data Transfers

As mentioned above, there are two formats for the SysAddIn bus. The second format is a two cycle transfer for data movement commands. The SysDC command field contained within a two cycle format control movement of data in and out of EV6, success/failure for ChangeToDirty and MB commands, and error conditions. The data transfers must begin in the first SysData cycle which occurs 9 CPU cycles after the start of the SysAdd cycle in which the ID command was received. The pattern of data is controlled by the SysDataInValid and SysDataOutValid signals. These signals valid each cycle of data transfer and are used to put gaps in the data pattern. The timing is described in Section 3.3.9.1. The format of the two cycle SysAddIn transfer is shown below:

	SysAddIn<14:2>						SysAddIn<1>	SysAddIn<0>
Cycle 1	0	SysDc<4:0>	RVB	RPB	A	ID<3:0>	X	X
Cycle 2	C	X					X	X

SysDC Command	SysDc <4:0>	Description
Nop	00000	Nop, SysData ignored by EV6
ReadDataError	00001	Data returned for Reads, System Drives SysData bus, I/O or Em NXM
ChangeToDirtySuccess	00100	no data, SysData ignored by EV6, also used for InvalToDirty response
ChangeToDirtyFail	00101	no data, SysData ignored by EV6, also used for Evict response
MBDone	00110	Memory barrier completed
ReleaseBuffer	00111	Command to alert EV6 that RVB, RPB and the ID field are valid.
ReadData (System Wrap)	100xx	Data Returned for Reads, System Drives SysData. Systems define wrap order using SysDc<1:0> See section 3.3.9.6 on Data Wrapping.
ReadDataDirty	101xx	Data Returned for Readx and ReadxMods, Ending Tag Status is Dirty, as above - System defines wrap order.
ReadDataShared(System Wrap)	110xx	Data Returned for Reads, System Drives Data, Tag marked Shared. Systems define wrap order using SysDc<1:0>.
ReadDataShared/Dirty	111xx	Data returned for ReadBlk, Ending Tag status is Shared/Dirty, as above - System defines wrap order
WriteData	010xx	Data sent for EV6 Writes or System Probe. EV6 drives SysData bus. Lower two bits of the command specify the quadword address around which EV6 should wrap the data.

There are 8 victim buffers in EV6. These victim buffers are used for both victims (fills that are replacing dirty cache blocks) and for System probes that require data movement. The CleanVictim command (optional) will also assign a victim data buffer. Each buffer will have two valid bits that denote the buffer is valid for a Victim or valid for a Probe or valid for both Victim and Probe. Probe commands that address match a VAF entry with an asserted Probe valid bit (P) will stall the EV6 probe queue. No probe responses will be returned until the P bit is clear. RVB(Release Victim Buffer), when asserted, will clear the Victim valid bit on the Victim Data Buffer (VDB) specified in the ID field. RVB bit will also clear the WRIO buffer when systems move data on I/O writes. RPB(Release Probe Buffer), when asserted, will clear the Probe valid bit on the Victim Data Buffer (VDB) specified in the ID field. Read data commands and victim write command use I.Ds 0-7 while Ids 8-11 are used to address the 4 IO write buffers.

"A" in the first cycle is command acknowledge used to decrement the EV6 "command outstanding" counter, but is not necessarily related to the current SysDc command.

Probe commands can have a combined SysDc command along with MBDone. In that event, the probe is considered ahead of the SysDc command . In particular, if the SysDc command allows EV6 to retire an instruction before an MB, or allows EV6 to retire an MB itself (SysDc is MBDone), that MB will not complete until the probe is executed.

Systems must assert appropriate SysDc command for correct ending Tag status. Systems may elect to return a dirty block of data to a RdBlk command. Systems that return a clean block in response to a RdBlkMod may cause a livelock; it is, therefore, not recommended. Finally, Systems may not cause a STx_C failure on any other processor in the system when returning a dirty block in response to a RdBlk.

3.3.9 Data Movement In and Out of EV6

There are two modes of operation that pertain to data movement in and out of EV6. These modes are selected with the CBOX csr called FAST_MODE_DISABLE. Fast data mode allows movement of data from EV6 to bypass protocol and achieve lowest possible latency for probes data, write victims and I/O writes. Rules and conditions for each mode is as follows:

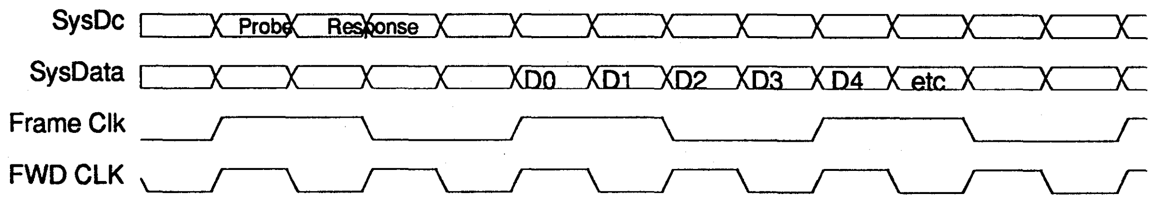
3.3.9.1 Fast Data Mode

EV6 is the default driver of the bi-directional SysData bus. As EV6 is processing WrVictim, Probe Response and WRIO commands to the system, data relative to this command is made available at the clock forwarded pin bus. SysDc commands that turn the SysData bus around may interrupt the successful completion of the 'fast' transfer. Systems are responsible to detect and replay all interrupted 'fast' transfers. There are no gaps in a 'fast' transfer and no wrapping (the first cycle contains QW0 addressed by 5:3 = 000#2).

Finally, systems must release victim buffers, probe buffers and WRIO buffers by sending a SysDc command with the appropriate RVB/RPB bit for both successful 'fast' transfers and for transfers that have been replayed. Fast transfers have two components, (1) the SysAddOut command with the probe response, WrVictim, or Wr(I/O) and (2) data. The command precedes data by, at least, one Framing Clock. The matrix below shows the number of Framing Clocks between SysAddOut and SysData for all System clock ratios (clock forwarded bit times) and Framing clock multiples.

	CLOCK FORWARD BIT TIME (System Clock Ratio)					
Bit Time/Framing Clock	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**
1	4	3	2	2	2	2
2	2	2	2	2	2	1
4	1	1	1	1	1	1

The timing diagram below show a simple example of a 'fast' transfer. This is an example of a system clock ratio of 1.5 and 4 bit times/Framing clock.



Movement of Data into EV6 involves careful timing to turnaround the SysData bus that is being driven by EV6. EV6 will respond to the SysDc command that always precedes the movement of data into EV6. Both the SysDc command and the first cycle of data are sent on System Framing clock boundaries(rising or falling edges). The total minimum number of Framing clocks between SysDc and data can be calculated as follows:

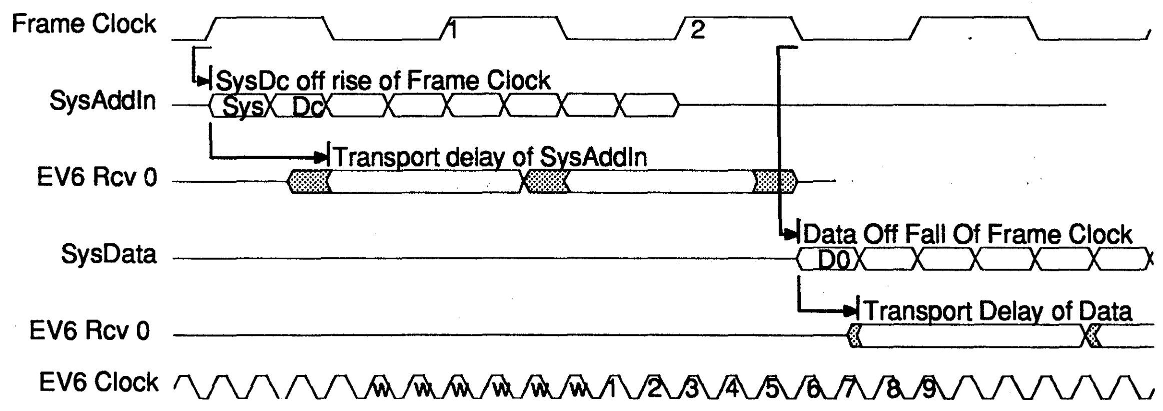
1. The fixed minimum delay in EV6 between the receipt of SysDc and the capture of the first piece of data is 9 processor cycles. So, Fixed Delay (FD) = (EV6 Cycle Time * 9).
2. Settle Time is the electrical bus settle time requirement that depends upon the maximum distance between EV6 and the furthest data chip. It is the round trip delay on the bi-directional data bus and can be calculated as : Settle Time (ST) =(2*max distance(in)) * 200 psec/in.
3. Clock skew between the EV6 Framing clock and the System Framing clock is a factor in the turnaround time. Total Skew (Tskew) = EV6 skew(4.0 nsec) + System skew(?).
4. To calculate the total number of Framing clocks between SysDc and data, take the sum of the three delay components above and divide that by the period of the Framing clock. and round up to the next half or whole Framing clock.. The Equation is as follows

# Framing Clocks = (FD +ST +TSkew)/Framing Clock period

Example:	CPU Cycle time	= 2 nsec
	Framing Clock	= 12 nsec
	Max Distance	= 10 inches
	Total Skew	= 4.5 nsec

# Framing Clocks = ((2.0 nsec *9) + (20 inches*200 psec.in) + (4.5 nsec) / 12 nsec
# Framing Clocks = (18 nsec + 4.0 nsec + 4.5 nsec) / 12 nsec = 26.5nsec /12 nsec
# Framing Clocks = 2.2 rounded up to either 2.5 or 3.

The following timing diagram illustrates data movement into EV6 using the results of the sample calculation shown above. The bottom trace in the diagram illustrates the EV6 internal clock with text indicating the 9 fixed processor cycles preceded by 6 delay (w) cycles. Systems use the wait cycles to delay the perception of SysDc so that the first piece of data arrives in time to be sampled by EV6. There is a 4 bit CBOX IPR called sysdc_delay that is used to fine tune the interface timing in the manner shown below. The delay does not effect SysData bandwidth.

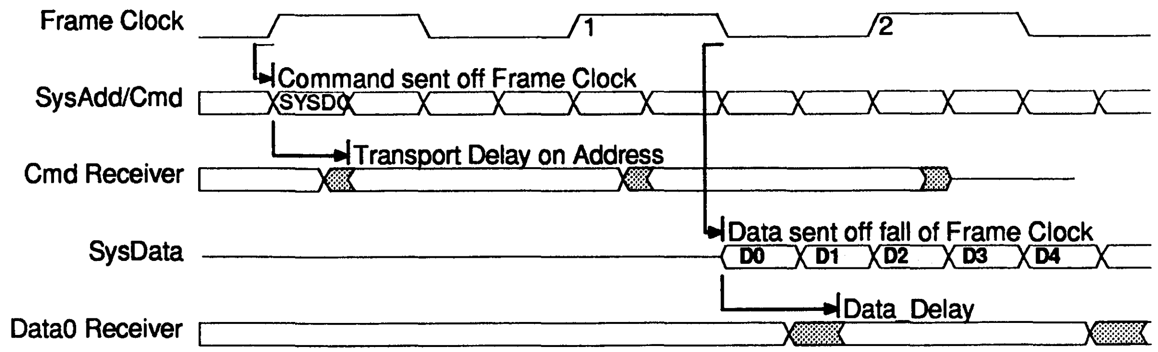

Frame Clock
1
2
SysDc off rise of Frame Clock
SysAddIn
Sys
Dc
Transport delay of SysAddIn
EV6 Rcv 0
Data Off Fall Of Frame Clock
SysData
D0
Transport Delay of Data
EV6 Rcv 0
EV6 Clock
w w w w w w 1 2 3 4 5 6 7 8 9

If a fast transfer is interrupted and fails to complete, the system must use the conventional protocol by sending EV6 a SysDc command of WriteData to removed the desired data buffer. The following section will describe the timing events for transferring data from EV6 to the system.

3.3.9.2 *Fast Data Disable Mode*

The system controls all data movement to and from EV6. Movement of data into and out of EV6 is preceded by a SysDc command. EV6 drivers are enabled only for the duration of an 8 cycle transfer of data from EV6 to the system. Systems must insure there is no overlap of enabled drivers and that there is adequate settle time on the SysData bus. As described above, systems must insure there is proper settle time when transitioning the SysData bus from write to read and from read to write. Settle time is measured from the point where EV6 sends the last quadword of data to the point when the system begins to transfer its first quadword (and vice versa). Settle time is an electrical constraint that can be calculated by multiplying the round trip distance of the furthest data chip from EV6 (in inches) times 200 psec/inch.

The diagram below shows the transferof data into EV6 on an idle SysData bus.



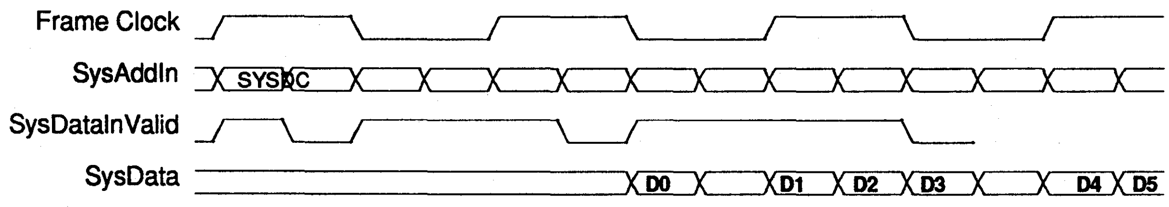
Frame Clock
1
2
Command sent off Frame Clock
SysAdd/Cmd
SYSDC
Transport Delay on Address
Cmd Receiver
Data sent off fall of Frame Clock
SysData
D0
D1
D2
D3
D4
Data_Delay
Data0 Receiver

When in Fast Data Disable mode, systems move data from EV6 with a WriteData SysDc command. This is used for removing data from a probe buffer, a victim buffer of a WRIO buffer. There is a fixed timing relationship from the point where EV6 receives the SysDc command until it drives the first QW on the SysData bus. Seven cycles after receiving the SysDc, EV6 looks for the rising edge of the next Framing clock. The example below shows the SysDc sent from the system on the rising edge of the first Framing clock with EV6 driving data two framing clocks later. This delay is fixed and uninterruptable much like reading a ram. Note there is some skew between the EV6 Frame clock and the System Frame clock.

3.3.9.4 SysDataIn/OutValid

There are two signals that are sourced by the system that control the rate of data delivery to and from EV6. These signals are associated with the address/cmd and have data bus timing attributes. Each signal represents a 64 bit quantity of data. For a complete transfer of data, EV6 must see the DataValid signals asserted for 8 data cycles. There can be any number of leading zero (deasserted cycles) and any pattern of gaps between valid cycles. Once the 8th cycle of an asserted DataValid signal is perceived by EV6, the transfer is considered complete. Minimal latency is achieved when the SysDataIn/OutValid signal is asserted in the same cycle as the SysDc command. Both SysDataIn/OutValid are 'don't cares' when not accompanied by a SysDc command. Systems may elect to drive and receive data at the lowest latency and highest bandwidth by asserting both SysDataIn/OutValid continuously.

EV6 expects to clock a valid data word on the 9th CPU clock after clocking the associated SysDataValidIn signal. Systems must ensure that data does not arrive too early. The following diagram illustrates a system transfer of a block of data into EV6.



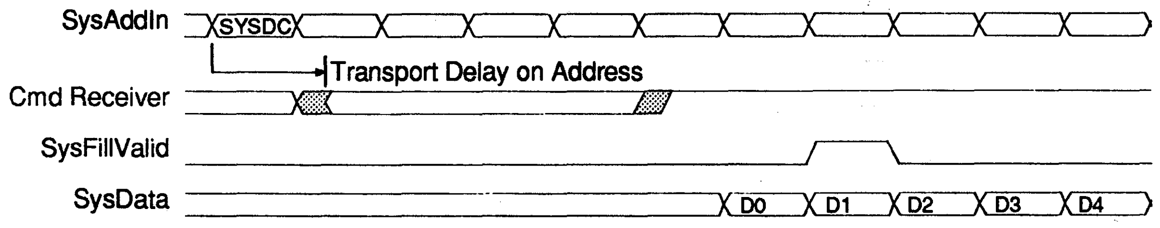
The timing relationship is slightly different for transfers out of EV6. EV6 will drive the first piece of data on the rise of the 'Framing' clock 7 CPU cycles after perceiving the first SysDataValidOut signal. From that point forward, every cycle of deasserted DataValidOut will cause a one cycle gap in the data transfer.

SysDc commands that do not move data into EV6 but modify cache tag state must allow a 2 data cycle window in the data bus by asserting the SysDataValidIn signal for 2 clock forwarded cycles. These commands are success acknowledge for ChangeToDirty and InvalToDirty commands. Systems that elect to tie this signal high (always asserted) must allow for a two cycle gap in the SysData bus when doing these commands.

Systems that elect to control the rate at which EV6 delivers data to the system, must set the add_frame_select register to 0. This means that all transmissions from EV6 to the system (address and data, will ignore framing clock edges and will commence on the earliest SysAddClkOut or SysDataClkOut.

3.3.9.5 SysFillValid

SysFillValid, when asserted validates the current memory and I/O data transfer into EV6. Systems may elect to tie this pin to a logical 1 to always assert valid fills or use it dynamically to enable or cancel fills as they progress. The net effect of this signal is to allow MP systems some additional time to attain probe results. EV6 will sample the value of SysFillValid at D1 time (the point at which EV6 samples the second data cycle). If SysFillValid is asserted at D1 time, the fill will continue uninterrupted. If it is unasserted, EV6 will cancel the fill but maintain the valid MAF entry until a successful fill occurs. The timing diagram below illustrates SysFillValid.



3.3.9.6 Data Wrapping

All data movement between EV6 and the system is one size only and that is 64 bytes or 8 complete cycles on the data bus. EV6 will generate memory read and write addresses that point to the desired octaword. All 64 bytes of memory data are valid . This applies to memory reads, memory writes and system probe reads.

I/O read and write addresses on the SysAddOut bus will point to the desired Byte, Word, Longword or Quadword , with a combination of address bits 5:3 and the mask field <7:0>. That combination is defined as follows:

Command	Significant Address Bits	Mask Type	Rules
RdQWs and WrQWs	SysAddOut<5:3>	QW	bits 5:3 will contain the exact PA bits of the first LDQ or STQ to the block. The Mask bits point to the valid Qws merged in ascending order
RdLWs and WrLWs	SysAddOut<5:3>	LW	bits <5:3> will contain the exact PA bits of the first LDL or STL to the block. The Mask bits point to the valid Lws merged in

			ascending order within one hexword.)
LDByte/Word and STByte/Word	SysAddOut<5:3>	BYTE	bits <5:3> will contain the exact PA bits of the LDByte/Word or STByte/Word. One byte mask for byte operation and two for word. No merging.

The order in which data is given to EV6 (in the case of a memory or I/O fill) or moved from EV6 (write victims or probe reads) is determined by the system. Systems can choose to reflect back the same low-order address bits and the corresponding octaword or any other starting point within the block.

SysDc commands for ReadData, ReadDataShared and WriteData require that systems define the position of the 1st QW by inserting the appropriate SysAddOut<5:3> into bits <1:0> of the command field. The recommended starting point is the quadword pointed to by EV6, however, some systems may find it more beneficial to begin the transfer elsewhere. The key point is that EV6 must always be told what the starting point is and the wrap order for all subsequent quadwords is always interleaved. The following table will define the method for systems to specify wrap and deliver data:

Source/Dest	SysDc<4:2>	SysDc<1:0>	Size	Rules
Memory	100 (ReadData)	SysAddOut<5:4>	Block (64 Bytes)	See Note 1
Memory	101(ReadDataDirty)	SysAddOut<5:4>	Block (64 Bytes)	See Note 1
Memory	110(ReadDataShared)	SysAddOut<5:4>	Block (64 Bytes)	See Note 1
Memory	111(Read DataShared/ Dirty)	SysAddOut<5:4>	Block (64 Bytes)	See Note 1
Memory	010(WriteData)	SysAddOut<5:4>	Block (64 Bytes)	See Note 1
I/O	100 (ReadData)	SysAddOut<5:4>	QW (8 -64Bytes)	See Note 1
I/O	100 (ReadData)	SysAddOut<4:3>	LW(4-32Bytes)	See Note 2
I/O	100 (ReadData)	SysAddOut<4:3>	Byte/Word	See Note 2
I/O	010(WriteData)	SysAddOut<5:4>	QW (8 -64Bytes)	See Note 1
I/O	010(WriteData)	SysAddOut<5:4>	LW(4-32Bytes)	See Note 1
I/O	010(WritcData)	SysAddOut<5:4>	Byte/Word	See Note 1

NOTE 1 Transfers to and from EV6 are 8 data cycles for 8 total Qws. The starting point is defined by the system. The preferred starting point is the one pointed to by SysAddOut <5:4>. Systems can insert the bits <5:4> into bit <1:0> of the SysDc command. The wrap order is 'interleaved' as defined by the table below.

	PA Bits <5:3> of Transferred QW			
1st QW	000	010	100	110
2nd QW	001	011	101	111
3rd QW	010	000	110	100
4th QW	011	001	111	101
5th QW	100	110	000	010
6th QW	101	111	001	011
7th QW	110	100	010	000
8th QW	111	101	011	001

Note 2 Longword and Byte/Word reads differ from all other transfers. Systems unload only 4 Qws of data into 8 data cycles by sending each QW twice. The first QW returned is determined by SysAddOut bits <4:3>. Systems again may elect to choose their own starting point for the transfer and insert that value into SysDc<1:0>. The wrap order for "double pumped" transfers is interleaved as defined by the table below.

	PA Bits <5:3> of Transferred QW			
1st QW	x00	x01	x10	x11
2nd QW	x00	x01	x10	x11
3rd QW	x01	x00	x11	x10
4th QW	x01	x00	x11	x10
5th QW	x10	x11	x00	x01
6th QW	x10	x11	x00	x01
7th QW	x11	x10	x01	x00
8th QW	x11	x10	x01	x00

3.3.10 Data ECC

EV6 supports a QW error correction code for the System Data bus. ECC is generated by the CPU for all Memory write transactions (WrVictimBlk) emitted from EV6 and for all probe data. ECC is also checked for every Memory read for single bit correction and double bit error detection. Bcache data is checked for fills to the Dcache and for all Bcache to system transfers (victims and probes).

I/O write data will not have a valid ECC (the ECC bits must be ignored by the System) and similarly, no checking is done on I/O read data.

If the System indicates that Memory data should not be checked via mode setting in ECC_DISABLE in the MBOX CSR, then no checking or correcting is performed.

3.3.10.1 ECC CODE

```
11 1111 1111 2222 2222 2233 3333 3333 4444 4444 4455 5555 5555 6666 cccc cccc
0123 4567 8901 2345 6789 0123 4567 8901 2345 7689 0123 4567 8901 2345 6789 0123 0123 4567
```

3.3.10.1.1 CB0 0111 0100 1101 0010 0111 0100 1101 0010 1000 1011 0010 1101 1000 1011 0010 1101 1000 0000

```
CB1 1110 1010 1010 1000 1110 1010 1010 1000 1110 1010 1010 1000 1110 1010 1010 1000 0100 0000
CB2 1001 1001 0110 0101 1001 1001 0110 0101 1001 1001 0110 0101 1001 1001 0110 0101 0010 0000
CB3 1100 0111 0001 1100 1100 0111 0001 1100 1100 0111 0001 1100 1100 0111 0001 1100 0001 0000
CB4 0011 1111 0000 0011 0011 1111 0000 0011 0011 1111 0000 0011 0011 1111 0000 0011 0000 1000
CB5 0000 0000 1111 1111 0000 0000 1111 1111 0000 0000 1111 1111 0000 0000 1111 1111 0000 0100
CB6 1111 1111 0000 0000 0000 0000 1111 1111 1111 1111 0000 0000 0000 0000 1111 1111 0000 0010
```

CB7 1111 1111 0000 0000 0000 0000 1111 1111 0000 0000 1111 1111 1111 1111 0000 0000 0000 0001

3.3.11 Ordering of System Port Transactions

This section details transaction ordering issues as they relate to the System port. There are two classes of ordering considerations:

EV6 commands and System probes
System probes and SysDC transfers

3.3.11.1 EV6 Commands and System Probes

The issue to be addressed involves EV6-generated commands and System port probes which reference the same cache block. First, a few points:

- EV6 commands reflect all probe responses sent and probe responses reflect all EV6 commands sent.
- VAF (Victim Address File) and VDB (Victim Data Buffer) entries each have independent valid bits for both a Victim and a Probe.
- Probe results indicate a hit on a VAF/VDB and whether or not the address has been sent. Systems can decide whether to move the buffer once or twice.
- Probe responses are issued in the order that they were received, however, there is no requirement for the system to retain order when issuing release buffer commands.
- Probe invalidates that match a valid VAF for which the address has been sent, will clear the VAF so that subsequent probes to this same cache block will NOT report a Hit VDB condition. The RVB is still required to release the VDB.

The following table lists all interactions between pending internal EV6 commands and probe commands, and shows EV6's response in each case.

	Probe: Next-State Command						
EV6 Command	Invalid	Clean	Clean/ Shared	Dirty/ Shared	Nop	Trans1	Trans2
RdBlk							
RdBlkMod							
FetchBlk							
InvalToDirty							
WrVictimBlk	Table 1	Table 1	Table 1	Table 1	Table 1	Table 1	Table 1
CleanToDirty	RdBlkMod /fail STx_C		Table 2			Table 2	Table 2
SharedToDirty	RdBlkMod /fail STx_C	Table 2					

- Notes:
- RdBlkVic and RdBlkModVic do not appear in the above table. If the interaction is between the probe and victim then it's the same as WrVictimBlk.
- Probes that invalidate locked blocks will not result in a RdBlkMod command. EV6 must fail the STx_C as defined in the Alpha SRM.
- All reads (RdBlk,RdBlkMod,Fetch, InvalToDirty) have no interaction as EV6 does not yet own the block.

Legend for Table 1 and Table 2 is as follows:

DM	= data movement (0 := probe does not need data, 1 := Systems requires data)
NsI	= Next State Invalid (0 :=next state NOP or shared etc., 1:=Next State is Invalid)
AS	= Address has been sent to the system

NOP = no buffer for probe
Status<1:0> = EV6 response on Probe (00=HitClean,01=Hit/Shared,10=HitDirty,11=Dirty/Shared)

Type	= Type of Hit (MAF = MAF hit and Address sent, VDB= VDB hit and Address sent)
sendV	= send victim to System as usual
killV	=block is no longer considered a victim by EV6
RVB	=release the Victim Valid bit on the VDB
RPB	=release the Probe Valid bit on the VDB
moveV	=move Victim data from EV6 to the System
moveP	=move Probe data from EV6 to the System
suppress victim	= don't moveV or (write victim to Memory and guarentee System DMA write is last)
cam	= Content address Memory ...a queue of addresses that are bit-for-bit compared with new entries

Table 1 : Probes that interact with WrVictimBlk

DM	NsI	AS	Status <1:0>	Type	EV6 Action		System Action
0	0	0	NOP	NOP	SendV	;wait RVB	moveV, RVB
0	0	1				;wait RVB	moveV, RVB
0	1	0	NOP	NOP	KillV		NOP
0	1	1				;wait RVB	RVB, suppress victim (VDB#cam)
1	0	0	HitDirty	NOP	SetP,SendV	;wait RPB,RVB	moveV/P, RPB, RVB
1	0	1	HitDirty	VDB	SetP	;wait RPB,RVB	moveV/P, RPB, RVB
1	1	0	HitDirty	NOP	killV, SetP	;wait RPB	moveP, RPB
1	1	1	HitDirty	VDB	SetP,	;wait RPB,RVB	moveP, RPB, RVB,suppress victim

Table 1 Notes:

1) Vafstate: SendV, Vvalid, Pvalid
2) moveV/P depends on Systems, blocks could be moved once or twice
3) Systems with address cams may clear both bits at the same time.

System Notes:

1) Tagless Uniprocessor -using P and V independently requires no address cams, the System can either compare VDB#s or observe the rule that Memory is written with the WrVictimBlk first followed by the DMA write.
2) Tagless MP - address cams are needed to fail SharedToDirty commands
3) Tagged MP
   If ChangeToDirty commands are failed by probing duplicate tag, no address cams are needed.

   If the VDB is not released until the Victim address is on the Bus, no address cams are needed for new probes versus victims.

Table 2 illustrates action taken by System and EV6 when a probe interacts with a ChangeToDirty command. A ChangeToDirty (XtoD) can be either a CleanToDirty (CtoD) or a SharedToDirty (StoD).

Table 2 : Probes that interact with ChangeToDirty

NsI	AS	Status <1:0>	Type	EV6 Action	System Action
0	0	NOP	NOP	Send StoD if Next state = S	
0	1	NOP	MAF		If CtoD, System can succeed or fail
1	0	NOP	NOP	Fail XtoD	
1	1	NOP	MAF		Fail XtoD

Table 3 illustrates the actions taken when a probe has a conflict with a pending fill.

PFAS = Probe is first to arrive at serialization point
PFT6 = Probe is sent first to EV6

Table 3: Probes that interact with pending Memory refills

PFAS	PFT6	ACTIONS
0	0	send probe after fill
0	1	tagged System option, read probes that hit in the MAF wait for fill to complete
1	0	N/A
1	1	Normal case, System waits for probe response and then send SysDC fill command.

Note: Probe commands that contain a SysDc fill to the same address, are considered unordered with repsect to the action they take on the cache. The Fill may occur before the probe or the probe may occur before the fill.

3.3.12 System Port Clocking

This chapter will define all aspects of clocking the EV6 processor. It will define the rules for input clock frequency, initialization and reset rules, system port clocking rules, and finally rules for entering and exiting low power.

3.3.12.1 Input Oscillator

EV6 has a nominal internal "CPU" clock rate of 500 MHz. It is produced as the result of a phase locked loop circuit with a frequency multiplying VCO generating 800 Mhz to 1.2Ghz that is divided by 2 (nominally) and distributed throughout EV6 (GCLK). Systems provide an input frequency or CLKIN_H/L that is used by the PLL for phase alignment. CLKIN_H/L can range from 80 to 200 Mhz.. Systems will input a differential sinusoidal signal preferrably from a PLL that is also the source of the clock for the system interface logic. The electrical, jitter and phase alignment specification for CLKIN_H/L are described in the PLL section of the Electrical data chapter.

There are three divisor circuits in the PLL loop. The X and Z divisor shown in the diagram below are controlled by an internal clock controller that steps up the frequency of the chip during power on/reset and also steps the frequency down and up for sleep mode. The Y divisor is set during reset by copying the values on IRQ<2:0> into a clock IPR. The Y divisor is never modified. Systems use the table below to select the appropriate Y divisor to establish the desired EV6 frequency. For example, if a system supplies a 100 Mhz CLK_IN and wants to run EV6 at 500 Mhz, it must establish a Y divisor of 5.

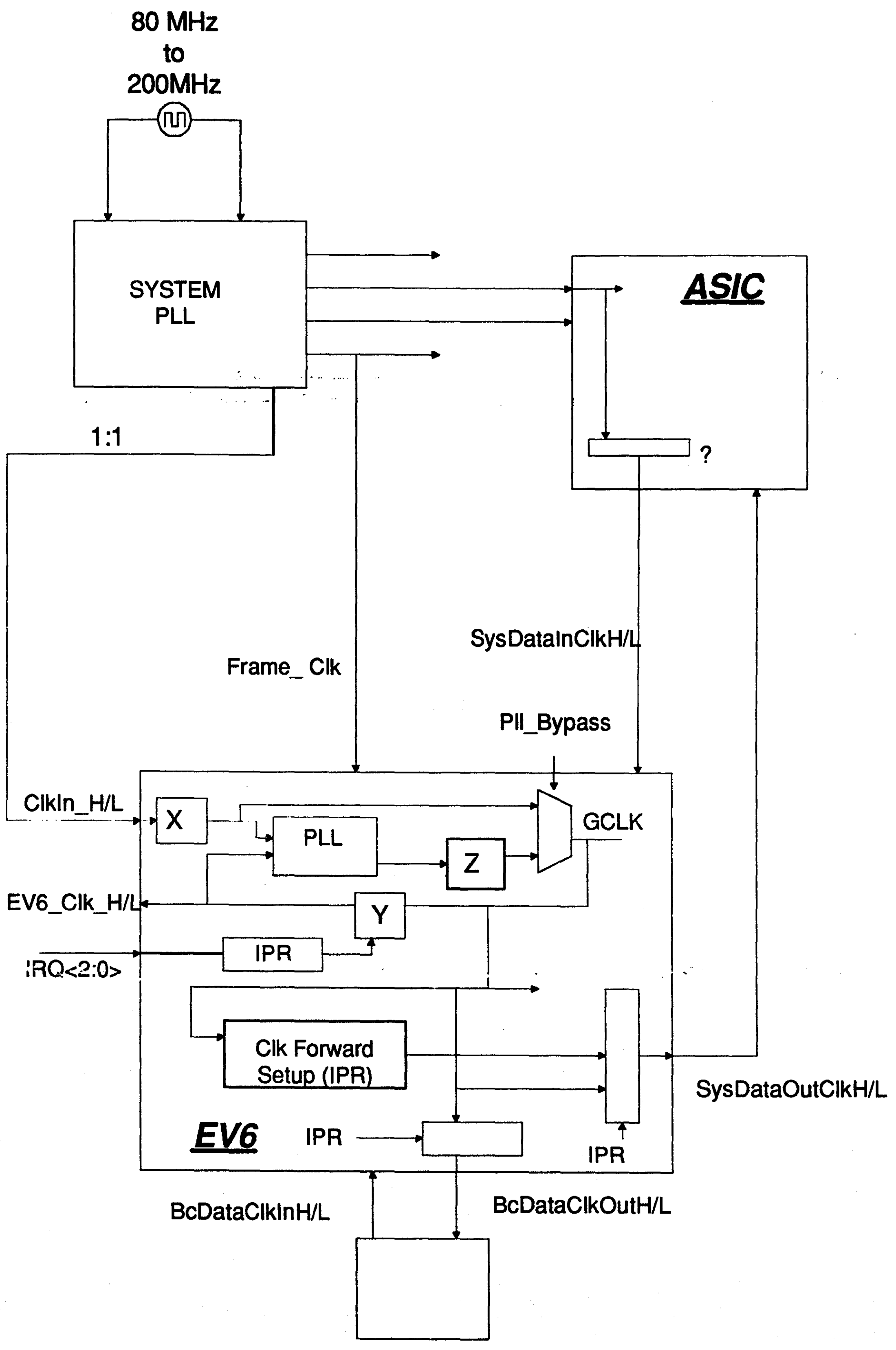
EV6 CYCLE TIME		Y DIVISOR VALUE					
NSEC	GCLK FREQ	3	4	5	6	7	8
2.5	400	133.33	100	80	na	na	na
2.4	416.66	138.889	104.167	83.333	na	na	na
2.3	434.8	144.928	108.696	86.956	na	na	na
2.2	454.54	151.15	113.636	90.91	na	na	na
2.1	476.19	158.73	119.048	95.238	na	na	na
2.0	500	166.67	125	100	83.333	na	na
1.9	526.31	175.439	131.579	105.263	87.719	na	na
1.8	555.555	185.185	138.889	111.111	92.593	na	na
1.7	588.24	196.078	147.059	117.647	98.04	84.034	na
1.666	600	200	150	120	100	85.72	na

1.5	666.667	na	166.667	133.333	111.11	95.238	83.3

Note that the lowest frequency applied to the input of the PLL in normal operating mode is 80 MHz.

3.3.12.2 System Clock or Framing CLock

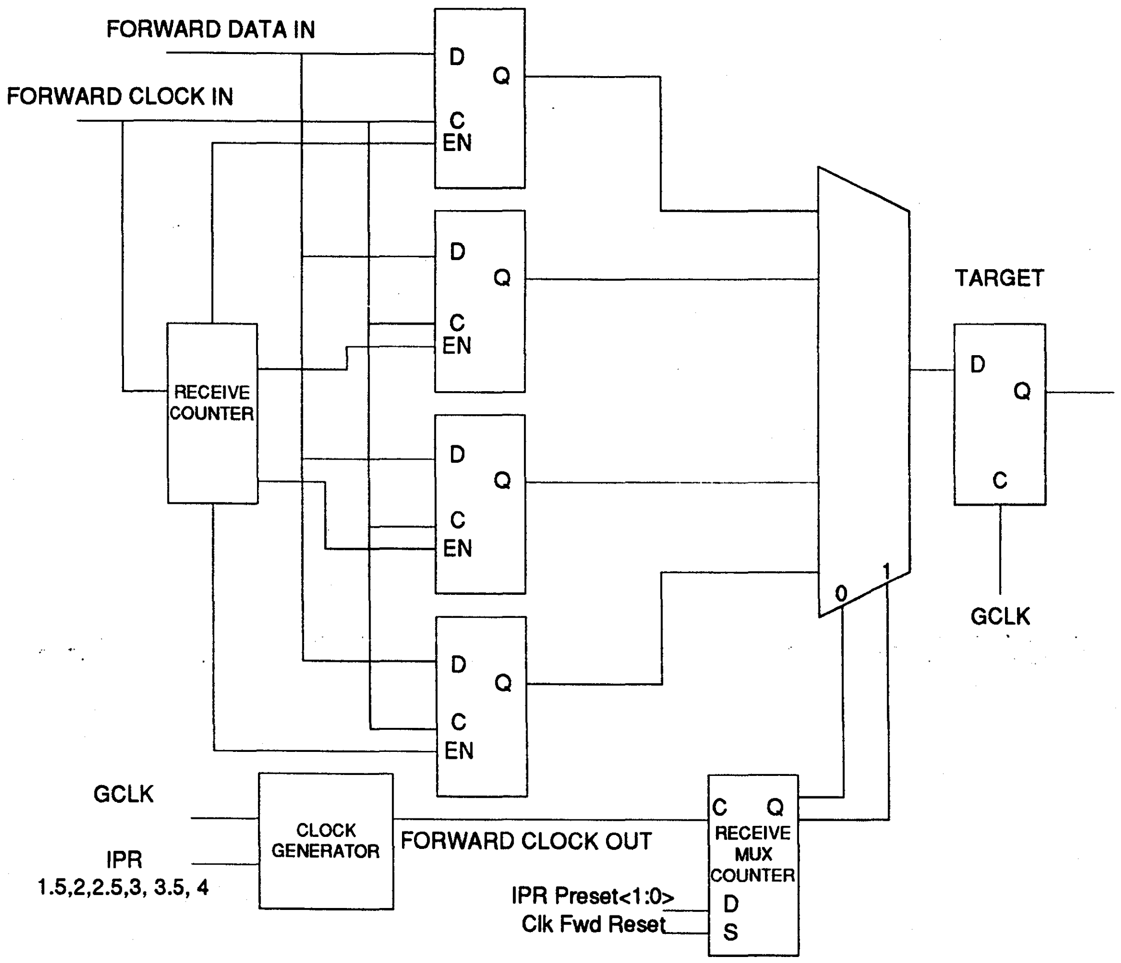
Systems are expected to run at an integer divisor of the oscillator input clock. EV6 requires a skew controlled copy of the system clock as a reference or a Framing Clock. This clock is a single-ended 50% duty cycle clock. This clock is captured by EV6 at the deassertion of reset. The captured framing clock will track the internal Gclk. EV6 uses this clock to determine the start of the system clock cycle for both clock forwarded transfers. Addititionally, EV6 uses the Framing clock to do a synchronous reset of the Clock Forwarding circuit. Systems must chose generate a Framing clock with a period that can insure proper synchronous transfer of the clock forward reset. The following block diagram illustrates a representative clock distribution scheme for EV6 systems.


80 MHz
to
200MHz
SYSTEM
PLL
ASIC
?
1:1
Frame_ Clk
SysDataInClkH/L
Pll_Bypass
ClkIn_H/L
X
PLL
Z
GCLK
EV6_Clk_H/L
Y
IPR
IRQ<2:0>
Clk Forward
Setup (IPR)
SysDataOutClkH/L
EV6
IPR
IPR
BcDataClkInH/L
BcDataClkOutH/L

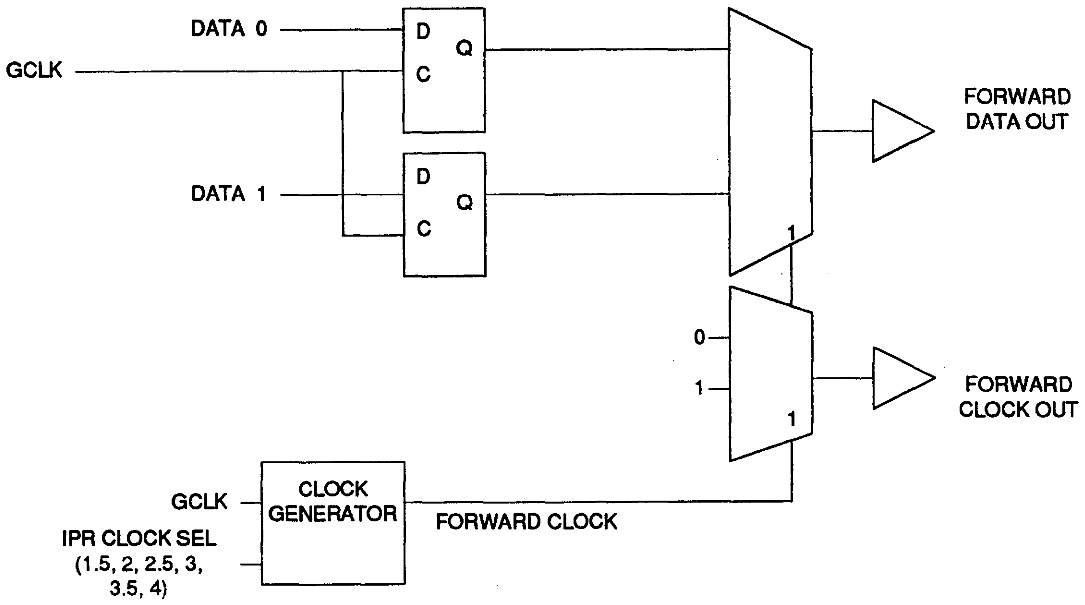
3.3.12.3 Clock Forwarding Definition

Clock forwarding is a well known communication technique that allows for transfers at higher speed than traditional synchronous point-to-point communication would allow. EV6 has very high address and data bandwidth requirements coupled with limited signal pins. Clock forwarding is the method that can overcome the limited pin availability and yet provide high bandwidth. Previous Alpha processors communicated to interface designs via a skew controlled synchronous clock. EV6 will send and receive data and address/cmd to/from systems accompanied by a single ended clock. There is one single ended clock for the address out (including SysFillValid and DataValid In\Out) and one single ended clock for the address in. Further, each byte of the data bus has a single ended clock for EV6 to System transfers and for System to EV6 transfers.

On the receiving end of the clock forwarded path, circuitry is sensitive to the forwarded clock such that it is used to strobe the flop of the data that it accompanies. Additionally the receive circuit has a counter that enables the receive flop and this counter is incremented by the received clock. What follows is a simple circuit illustrating a receive circuit for a single bit.



A simplified example of a sending circuit is shown below.



3.3.12.4 Glossary of Terms

There are a number of terms that will be used repeatedly in this specification and require defining.

BIT TIME - Specified in Nsec and pertains to the total time that a signal conveys a single valid piece of information. Since all data and command is associated with a clock and the receivers latch on both the rise and fall of the clock. Bit times are defined as a multiple of the EV6 clocks. Systems must produce a Bit time identical to EV6.

FORWARD CLOCK - A single ended signal that is aligned with its associated fields. Sourced and aligned by the sender with a period that is 2 times the bit time. Forwarded clocks must be 50-50 duty cycle clocks whose rising and falling edges are aligned with the changing edge of the data.

FRAMING CLOCK - The framing clock defines the start of a transmission either from the system to EV6 or from EV6 to the system. The Framing clock is a power-of-2 integer multiple of the EV6 CPU clock. and is usually the system clock. The Framing clock and the input oscillator can have the same frequency. The add_frame_select IPR sets that ratio of bit times to Framing clock. The Frame clock could have a period that is 4 times the bit time with a add_frame_select of 2X. Transfers begin on the rising and falling edge of the Frame clock. This is useful for systems that have system clocks with a period to small to perform the synchronous reset of the clock forward logic.

SYSTEM CLOCK - The primary skew controlled clock used throughout the interface components to clock transfer between ASICS, main memory and I/O bridges.

GCLK - Global clock within EV6, the 2 nsec globally distributed clock with EV6

INTERFACE RESET - A synchronously received reset signal that is used to preset and start the clock forwarding circuitry. During this reset, all forwarded clocks are stopped and the presettable count values are applied to the counters, than some number of cycles later the clocks are enabled and are free running.

RECEIVE COUNTER- Counter used to enable the receive flops. It is clocked by the incoming forwarded clock and reset by the Interface Reset.

RECEIVE MUX COUNTER- The Receive Mux counter is preset to a selectable starting point and incremented by the locally generated Forward Clock.

OUTPUT MUX COUNTER - Counter used to select the output mux that drives address and data. It is reset with the Interface Reset and incremented by a copy of the locally generated forwarded clock.

CORRELATED SKEW - Uncertainty contributors that are track commonly . Examples of correlated skew might be a signal sourced from the same chip and sent to the same destination chip. The total system clock skew is correlated among this group of signals. Intra-die process variations are also correlated.

UNCORRELATED SKEW - The mismatch between the delay of the forwarded clock and the forwarded data. There arc a number of contributors in the uncorrelated category whose total magnitude is a limit to the minimum bit time. Uncorrelated skew is what forces the forwarded clock out of alignment with respect to the data.

TARGET CLOCK - Skew controlled clock which receives the output of the RECEIVE MUX .

CLOCK OFFSET - or ClkOffset is the delay intentionally added to the forwarded clock to meet the setup and hold requirements at the Receive Flop

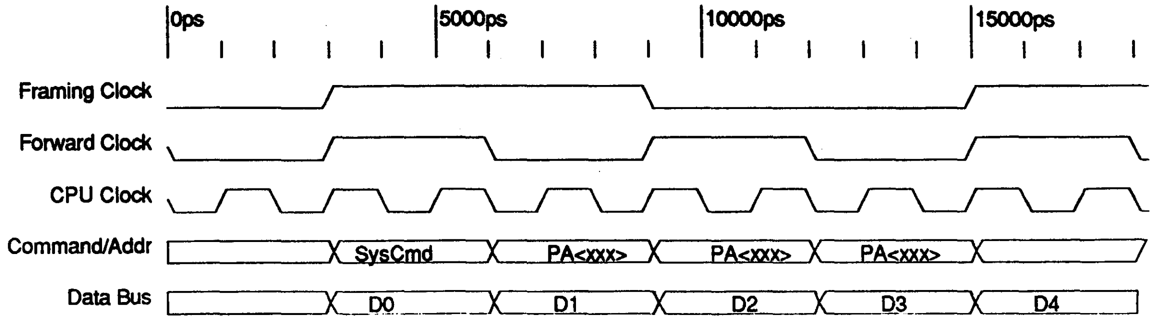
3.3.12.5 *Clock Forwarding Bit times*

EV6 will derive its forwarded clock from the internal CPU otherwise known as GCLK. Systems can choose from one of six GCLK multiples for the forwarded clock. Those value are 1.5, 2, 2.5, 3,3.5 and 4.

Systems must match their send and receive circuits with the BIT TIMES that it selects for EV6. If EV6 is setup to drive data at a 3 nsec BIT TIME, then the system must send and receive at the same 3 nsec rate. Below is a table that show the bit times for all possible clock multiples and the standard six GCLK frequencies.

EV6 FORWARD CLOCK MULTIPLIER VALUE	EV6 Internal Operating Frequency				
	450MHz	500MHz	550MHz	600MHz	700MHz
1.5	3.3 nsec	3 nsec	2.75 nsec	2.5 nsec	2.14 nsec
2	4.4 nsec	4 nsec	4.63 nsec	3.33 nsec	2.84 nsec
2.5	5.5 nsec	5 nsec	4.545 nsec	4.16 nsec	3.5 nsec
3	6.6 nsec	6 nsec	5.454 nsec	5 nsec	4.26 nsec
3.5	7.7 nsec	7 nsec	6.363 nsec	5.833 nsec	4.97 nsec
4.0	8.8 nsec	8 nsec	7.272 nsec	6.667 nsec	5.68 nsec

Below is a timing diagram showing an example of the timing relationship of the three clocks (Framing clock, forwarding clock and GCLK) and the data and address bus. The frequency of the clock matches that of the data. Receive circuits must be designed to latch on both the rise and fall of the forwarded clock. Note that it does not illustrate correct protocol.



3.3.12.6 *Principles Of Operation*

A single-ended clock accompanies at most 16 signals from a sender targeted at a receiver. All delay contributors that effect the propagation of the clock and the signal it represents are matched as closely as possible. The output stage and the drivers are closely matched to align the rise/fall of the single-ended clock with the front edge of the data signal. Systems must do a likewise matched circuit design for their sending circuits.

3.3.12.6.1 Number of RECEIVE FLOPS required

Receiving circuits for the system designs have one variable and that is the number of receive flops along with the associated size of the receive clock enable counter. The absolute minimum number of receive flops (N) is the Target Clock Period/ BIT TIME and this would assume that there is no skew, no setup and no hold. The minimum number of receive flops (N) can be determined by the following equation:

N = (Target Clock Period + (Max _delay - Min_delay) + TSkew + Tsetup + Thold) / BIT TIME

ex. Target Clock Period (System clock) = 12 nsec

Max_Delay(total worst case delay from sender to Target flop) = 9 nsec
Min_Delay(total best casc dclay from sender to Target flop) = 5 nsec
TSkew (Total clock uncertainty between GCLK and System Clock = 5 nsec
Tsetup (setup time of the Target flop) = 100 psec
Thold (hold time of the Target flop) = 500 psec
BIT TIME = 3 nsec

N = (12 + (9-5) + 5 + .1 + .5) / 3 = 21.6/3 = 7.2 rounded up to the next integer = 8 receive flops.

3.3.12.6.2 Maximum allowable skew and max to min delay difference.

EV6 will have 4 receive flops for both the SysAddIn bus and the SysDataIn bus. This number of receive flops combined with the bit time will determine the maximum allowable difference between the Min_Delay and the Max_Delay plus the total clock skew. The equation is as follows:

4 * BIT TIME > (EV6 CYCLE TIME +(Max_delay-Min_delay) + TSkew + Tsetup+ Thold)

BIT TIME = 3nsec

EV6 Cycle Time = 2

Max_delay = 9

Min_delay = 5

TSkew = 4

Tsetup = .100

Thold = 0

4 * 3 > (2 +(9-5) + 4 + .1) .....12 > 10.1 nsec

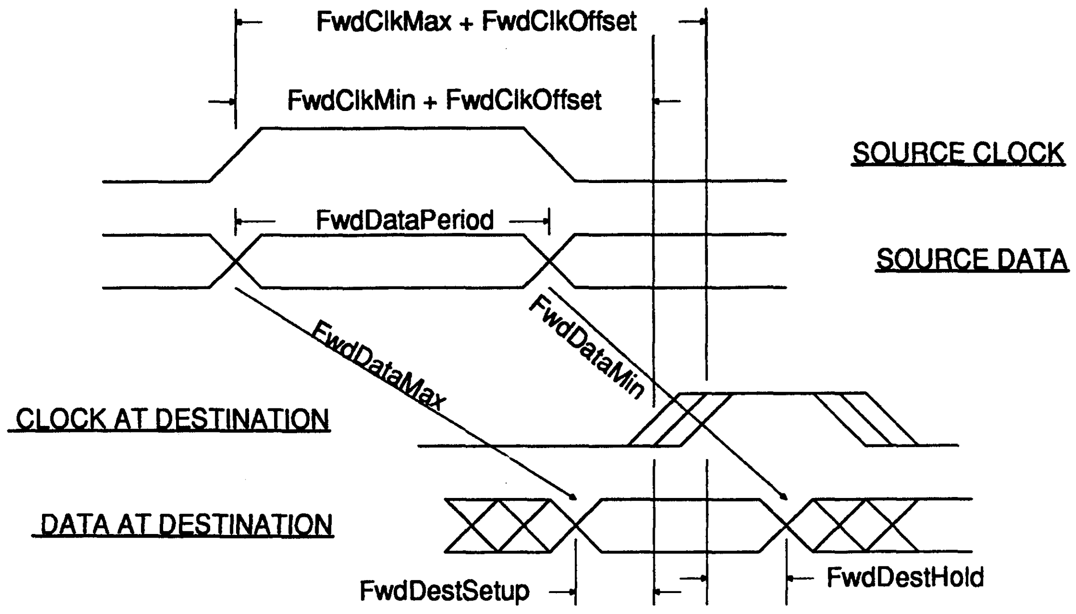
3.3.12.6.3 Receive Mux Counter Preset value

Another system selectable value is the preset value on the receive mux counter. The receive mux counter is incremented by a copy of the local Forward Clock so that it has a frequency equal to the bit rate and is skew aligned with GCLK. Since the counter must select a receive flop at the earliest point in its valid window, it is really determined by the maximum delay from the source clock within the senders ASIC to the output of the receive flop. The MAX_DELAY between the sender and receiver can be greater than the turnover of the receive mux counter. In fact, it can many times greater than the time taken to complete one cycle of the receive mux counter as long as all of the other requirements are met. The preset delay in nanoseconds is :

MAX_DELAY (nsec) mod (Bit Time * 4) = Preset Delay(nsec)

Then the preset value which is loaded into the counter during clock forward reset is the two's compliment of : ***PRESET DELAY (nsec) / BIT TIME (nsec)***
Systems will have differing delays between the SysAddIn/Out buses and the SysDatabus. Since EV6 has 4 receive flops for address and data, a wide valid window is created. This wide window can absorb delay differences due to placement up to 2 nsec or about 13 inches. Therefore, when systems calculate the receive mux counter preset value they should use the max delay of all the data and address busses.

3.3.12.6.4 Minimum Bit TimeThe minimum bit time or the period of the forwarded data and clock (recall that the clock switches at the same rate as the data) is vital in arriving at the maximum supportable bandwidth of the interface. The limits are local, meaning within the specific set of signals and the associated clock. For EV6, minimum bit time is established by examining the min max differences across a group of signals that have a common source and destination. For example, the SysAddOut bus is accompanied by a forwarded clock and collectively they are targeted at one destination. The following diagram illustrates the contributors that minimize the Bit time.



From the diagram, one can derive the following equations:

a) Min_BitRate = FwdClkPeriod

b) Min_BitRate = UncorrelatedSkew + FwdDestSetup + FwdDestHold

c) UncorrelatedSkew = (FwdDataMax - FwdDataMin) + (FwdClkMax - FwdClkMin) + (FwdClkOffsetMax - FwdClkOffsetMin)

There are a number of contributors to Uncorrelated skew. They are as follows:

- Delay variations dependent on previous history of data transitions on an individual bit line.
- Simultaneous switching of outputs causing clock and data pad cells to experience delay that depend on the switching patterns of nearby neighbors.
- Crosstalk between signal lines couple into adjacent signal nets causes signals to move up and down from their crosstalk-free positions.
- Differences between the desired length of the clock lines and the data lines on both the module and package
- Differences in impedance along path of clock and data lines.
- Differences in the propagation velocity of clock and data due etch runs on different signal layers.
- Differences in propagation delays of clock and data due to different cell types used in the two paths on both the source and target chip.
- Differences in termination techniques between the clock and data.
- Differences in loading of the clock and data networks at either the source or target chips.
- Differences in clocking times of different cells due to RC delays.
- Intra-die process variations

To determine how much one must delay the Forward clock relative to the forwarded data (FwdClkOffset), use the following equations:

d) FwdClkOffset > (FwdDataMax + FwdDestSetup - FwdClkMin)

e)FwdClkOffset < (FwdDataMin + FwdDataPeriod - FwdClkMax - FwdDestHold)

3.3.12.7 Power Down Mode

EV6 is designed to operate in computer systems that meet all the criteria specified in the EPA Energy Star worksheet. EV6 can automatically enter a low-power or "sleep" mode that enables the system to be 30 watts or less. EV6 will automatically "wake up" upon resumption of system activity and return to the same situation that existed prior to entering sleep mode.

1. EV6 will enter sleep mode by way of CALL_PAL WTINT. The following sequence of events will occur during the execution of this instruction.
2. EV6 writes a value into the TBD CSR external to the system which is the number of interval timer interrupts that the system ignores until threshold..
3. EV6 interval caches are swept. Clean block are invalidated(Evict commands are issued on systems with Duplicate Tag stores) and dirty caches blocks are written back to main memory.
4. EV6 then saves all architectural and readable state that would be needed upon returning to the wake state.
5. EV6 sets an IDLE in the interface that alerts the system that he interface is inactive. The system will 'ACK' this command when all outstanding probes have been serviced. The system will send no more probes until the IDLE bit is clear.
6. The routine now writes to an internal IPR that sets the divisor on the PLL output so that the GCLK now runs at less than 1/10th the nominal clock rate.
7. Upon receiving either an interval timer interrupt or a device interrupt (if enabled), EV6 will do a limited chip reset. That is, reset all but the configuration registers.
8. When nominal clock rate is achieved, EV6 receives a ClockFwdReset to reset its own clock forward circuitry as well as the systems.
9. EV6 will then read the external interval timer threshold register to determine the type of interrupt and to update the memory resident time-of-year clock. EV6 will also clear the IDLE bit in the system.
10. EV6 will now restore the processor state and return to normal operation.

3.3.12.8 PLL Bypass

EV6 testing requirements include the ability to bypass the internal PLL. An input pin known as PLLBypass, when asserted will apply the CLKIN_H/L frequency directly to the internal GCLK distribution. For nominal 500 MHz operation a 500 MHz sinusoidal differential signal must be applied to the CLKIN_H/L pins.

Additionally, EV6 can be operated in a system with PLLBypass asserted. Systems providing a frequency directly to EV6 in bypass mode will either provide an external PLL that performs that phase alignment to EV6CLK_H/L. An alternative would be to absorb the delay from CLKIN to EV6CLK as skew. The max delay from CLKIN to EV6CLK is TBD.

3.3.12.9 INITIALIZATION

This section will describe those features of the EV6 clock forwarding interface that are programmable.

1. Address and Data bus bit time :
   A 3 bit field in the CBOX IPR that defines the forwarding clock period of the SysData bus and the SysAddIn/Out Bus as a multiple of the EV6 GCLK. The table shows the clock multiple which is selected and the associated bit times for each of the 6 possible settings. The IPR is called sysclk_ratio.

Sysclk_ratio<2:0>	Multiple	Bit Time EV6 @2nsec
001	1.5	3nsec
010	2.0	4nsec
011	2.5	5nsec
100	3.0	6nsec
101	3.5	7nsec
110	4.0	8nsec

2) Address and Data Receive Mux Counter Preset
A field that is the preset value of the receive mux counter after deassertion of the synchronous clock forward reset. The preset value is chosen by careful analysis of the Max_delay of the forwarded clock and the earliest possible point to select the appropriate receive flop. The CBOX IPR is called sys_rcv_cnt_preset<1:0>. .

sys_rcv_cnt_preset<1:0>	Counter Preset
00	00
01	01
10	10
11	11

3) Y Divisor value select field
A three bit field used to select one of 8 Y divisor values in the PLL return loop. The Y divisor divides down the distributed GClk from the nominal 500 Mhz to match the CLKIN_H/L frequency. The Y divisor value is set during reset by copying the static held levels on IRQ<2:0> into a clock control register.

IRQ<2:0>	Y divisor value
000	3
001	4
010	5
011	6
100	7
101	8

4) Framing Clock Offset

A 2 bit field that changes the position of the framing clock relative to the framing clock seen at the input pins. It allows systems to adjust the start of EV6 generated commands and data so that it is guaranteed to be valid at the earliest system clock edge. This will help in reducing latency. Each bit equals one forward clock period of adjustment earlier than the nominal frame clock. This CBOX IPR is called fram_clk_offset<1:0>.

fram_clk_offset<1:0>	# Forward Clock Periods
00	0 (nominal)
01	1
10	2
11	3

3.3.12.10 Clock Forward Reset

Systems are required to generate Clock Forward Reset (ClkFwdReset_H). This signal must occur no earlier than TDB cycles after the deassertion of reset_L and TBD cycles after the interval timer interrupt is sent to wake up a powered down EV6. ClkFwdReset is a synchronous signal and is clocked into a register
in EV6 with the captured copy of Framing Clock. Systems must insure that the with +/-2.0 nsec of skew and set up time of ***psec, EV6 can safely capture the assertion of ClkFwdReset.

There is a one (framing clock) cycle of internal distribution delay on ClkFwdReset so that on the second rising egde of ClkFwdReset, it is applied to the target circuit. The forwarded clocks are disabled both at the system and within EV6. The receive counter is set to 0 and the sys_rcv_mux_cnt_preset<1:0> is applied to the Unload counter in EV6. ClkFwdReset should assert for a minimum of 3 framing clock cycles. The synchronous deassertion of ClkFwdReset will start the forwarded clocks. Clocks remain on and free running.

The diagram below shows the application of ClkFwdReset. Note that it is asserted for only two framing clock periods and the minimum is three.

3.4 Bcache Port

EV6 supports a second level cache from 1 to 16 MB in size, with 64-byte blocks. A 128-bit bus is used for data transfers between EV6 and the Bcache. The Bcache is fully synchronous, and the SRAMs must contains either one, two or three internal registers. All Bcache control and address pins are clocked synchronously on Bcache cycle boundaries. The Bcache clock rate can vary from 1.5 to 4 CPU clock cycles, in half cycle increments.

3.4.1 Bcache Port Pins

Pin Name	Type	Count	
BcAddress<23:4>_H	O	20	Bcache Index
BcDataOE<1:0>_H	O	1	Bcache data output enable
BcBurst_H	O	1	Bcache burst enable for burst mode SRAM's
BcDataWR_H	O	1	Bcache data write enable
BcData<127:0>_H	B	128	Bcache data
BcCheck<15:0>	B	16	ECC check bits for BcData
BcDataClkIn<7:0>_H	I	8	optional Bcache data input clocks
BcDataClkIn<7:0>_L	I	8	optional Bcache data input clocks
BcDataClkOut<3:0>_H	O	4	Bcache data clock outputs
BcDataClkOut<3:0>_L	O	4	Bcache data clock outputs
BcTag<42:20>_H	B	23	
BcTagValid_H	B	1	
BcTagDirty_H	B	1	
BcTagShared_H	B	1	
BcTagParity_H	B	1	
BcTagClkOut_H/L	O	2	
BcTagClkIn_H/L	I	2	optional BcTag input clocks
BcTagOE_H	O	1	tag ram output enable
BcTagWR_H	O	1	tag ram write enable

3.4.2 Pin Descriptions

3.4.2.1 BcAddress

BcAddress is a high drive output and supplies the index for the Bcache. EV6 supports the following Bcache sizes : 0,1MB,2MB,4MB,8MB, and 16MB.

3.4.2.2 BcClkOut

BcClkOut<3:0> are differential copies of the Bcache clock. BcClkOut may be configured such that its rising edge lags BcAddress by 0 to 2 CPU clock cycles. The BcClkOut is free-running and is derived from the internal GCLK. It's period is a multiple of the GCLK and is fixed for all operations.

EV6 supports only Synchronous SRAMS. Those Synchronous SRAMs can be from 3 different families. The first is a BurstRam with conventional Reg/Reg output and that is one piece of data for every rise of the clock. The second type is a non-burst REG/REG Late Write architecture. And finally, the third type is a BurstRam Reg/Reg Late Write with clock forwarded output with data on the rise and fall of the clock.

3.4.2.3 BcBurst

BcBurst is asserted on the first cycle of a read or write when BC_BURST is set in the **IPR. Tag stores are not bursted and can be accessed under a burst of the data BurstRam.

3.4.2.4 BcDataClkIn & BcTagClkIn

The BcDataClkIn and BcTagClkIn pins are to be used with high speed DDRs that provide a clock out with the data output pins to optimize Bcache read bandwidth. EV6 will internally sync the data to the its CPU with clock forward receive circuitry similar to the System interface. For non DDR devices systems will connect the .

3.4.3 Bcache Banking

Bcache banking is possible by the decode of the most significant address bit. Switching between cache banks may require Rd-Rd bubbles as well as the usual Rd-Wr bubbles; this will be programmed via the BC_RR_BUB field of the **IPR.

3.4.4 Bcache Transactions

The Bcache supports 4 transactions:

Data Read
Data Write
Tag Read
Tag Write

Data reads are always accompanied by tag reads in the first cycle of the data read. Similarly data writes include a tag write in the first cycle so the Bcache tag state can reflect any changes made to the block while it was in the Dcache. Tag reads and writes are used individually as the result of System PROBE commands. Tag reads will also be performed during a 4 cycle burst of the Data SRAMS. This allows system probes access to the Bcache Tag store without interrupting that private access by the processor.

EV6 supports late write SRAMs - write data can be delayed from the address by 0 to 4 Bcache clocks.

3.4.5 Bcache Clocking

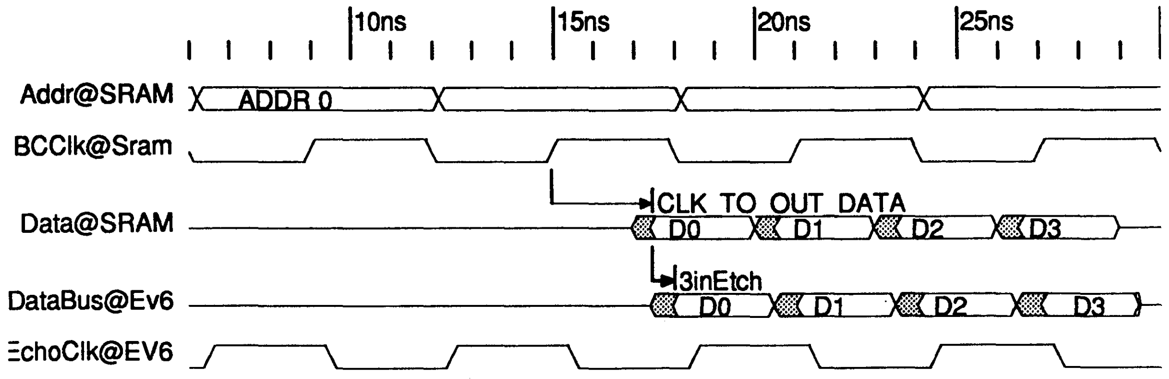
BcClkOut is used to synchronously clock address, control and data into and out of the Bcache SRAMs. BcClkOut and BcClkIn are free running clocks and they are derived from the internal processor clock The period and position of the edges of the clock is determined by setting appropriate fields in the CBOX IPS.

The period of the Bcache clock is established by setting the bcclk_ratio CBOX IPR. The ratio is a multiple of the processor clock and can range from 1.5 to 4.0 in .5 increments. This setting essentially defines the period of the data bit. In single data mode, the clock ratio established the period of the Bcache clock and in dual data mode the bclk_ratio defines one phase of the bcclk_ratio. Dual data SRAMs provide data on the rise and fall of a clock that accompanies the data back to EV6. Systems must enable dual data mode by asserting the bc_ddr_enable in the CBOX IPR.

The position of the clock relative to address and write data can be controlled down to a processor clock phase by setting the appropriate value in the 3 following registers:

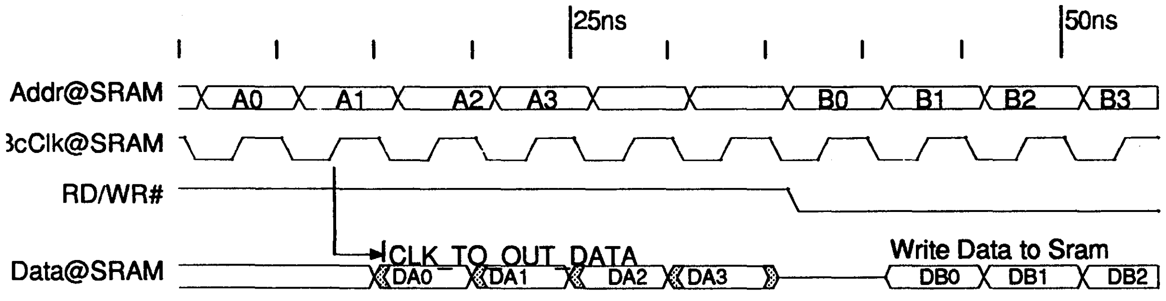
1. BC_LATE_WRITE_NUM delays write data relative to the address by one bccache clock period for each binary value in the register from 0 to 7 Bcache clocks. This is useful for late write SRAMs.
2. BC_CPU_LATE_WRITE_NUM delays write data relative to address by one CPU clock period for each binary value set in the register from 0 to 3 CPU cycles. The rising clock edge moves with write data.
3. BC_PHASE_LATE_WRITE_NUM delays the rising edge of the Bcache clock by 0-2 CPU clock phases. For a 2 nsec CPU clock, this allows for 1 nsec granularity setting clock the position of the clock.

3.4.5.1 *Dual Data SRAM Read*



The timing diagram above show a read from a bursting dual data SRAM that forwards data on the rising and falling edges of an echo clock. The echo clock is a reflected copy of the SRAM input clock provided by EV6. It is properly aligned with the data output so that it meets setup and hold requirements of the receive circuitry in EV6. The SRAM does a burst of 4 in interleaved burst order. Control signals not shown initiate the burst, control read and write and the direction of the output drivers.

3.4.5.1 *Standard SRAM Read*



The timing diagram above illustrates a late write SRAM that is non-bursting in single data mode. The ram provides one piece of data for every rise of the BcClk. For reads, the address is clocked into the part in cycle 'n' and the data appears at the pins on clock 'n + 1'. Each piece of data requires a new address. The diagram shows the transition to a write, beginning with address B0 and the fall of the RD/WR# control signal. Data relative to B0 address is clocked into the part on the next rising edge of the BcClk. Note that there are two bubbles in the address when transitioning from a read to a write. There are no bubbles when going from a write to a read.

3.5 Interrupts

The System may request interrupts via the irq_h<5:0> pins. These six interrupt sources are identical: they may be asynchronous, are level sensitive and can be individually masked via the EIE field of the IER IPR. The way these signals are used and their relative priority is completely general, and left to the System designer.

3.6 Pin List

Name	Type	Count
SysAddIn<14:0>_L	I	15
SysAddInClk_L	I	1
SysFillValid_L	I	1
SysAddOut<14:0>_L	O	15
SysAddOutClk_L	O	1
SysData<63:0>_L	B	64
SysCheck<7:0>_L	B	8
SysDataInClk<7:0>_L	I	8
SysDataOutClk<7:0>_L	O	8
SysDataInValid_L	I	1
SysDataOutValid_L	I	1
TOTAL		**123**
BcAddress<23:4>_H	O	20
BcDataOE_L	O	1
BcLoad_L	O	1
BcDataWR_L	O	1
BcData<127:0>_H	B	128
BcCheck<15:0>_H	B	16
BcDataInClk<7:0>_H	I	8
BcDataInClk<7:0>_L	I	8
BcDataOutClk<3:0>_H	O	4
BcDataOutClk<3:0>_L	O	4
BcTag<42:20>_H	B	23
BcTagValid_H	B	1
BcTagDirty_H	B	1
BcTagShared_H	B	1
BcTagParity_H	B	1
BcTagOE_L	O	1
BcTagWR_L	O	1
BcTagInClk_H/L	I	2
BcTagOutClk_H/L	O	2
TOTAL		**224**
IRQ<5:0>_H	I	6
RESET_L	I	1

Name	Type	Count
SromData_H	B	1
SromClk_H	O	1
SromEn_L	O	1
TestModeSelect_H	I	1
TestClk_H	I	1
TestReset_L	I	1
TestDataIn_H	I	1
TestDataOut_H	O	1
TestStatus_H	O	1
ClkIn_H	I	1
ClkIn _L	I	1
FrameClk_H	I	1
PllBypass_H	I	1
ClkFwdReset_H	I	1
EV6CLK_H	O	1
EV6CLK_L	O	1
PLLVDD	I	1
DCOk_H	I	1
VRefBcache	I	1
VRefSys	I	1
TOTAL		27
GRAND TOTAL		374

4. Privileged Architecture Library Code

This chapter describes the EV6 PALcode environment.

4.1 Use of Alpha Implementation-Specific Opcodes

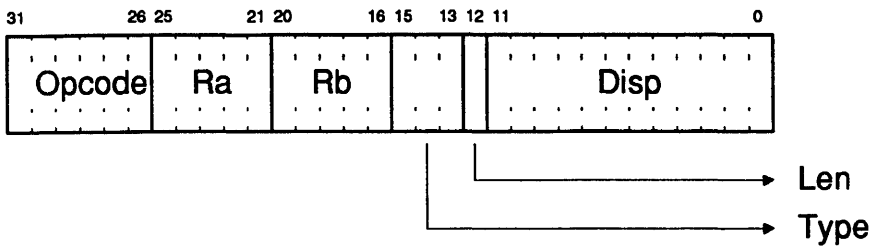
The Alpha architecture reserves five opcode points for implementation-specific PALcode use. The table below lists these opcodes and their use in EV6.

EV6 Mnemonic	Opcode_{16}	Function
HW_LD	1B	D-stream load instruction
HW_ST	1F	D-stream store instruction
HW_RET	1E	Return from PALcode routine
HW_MFPR	19	Reads the value of an IPR into a integer GPR
HW_MTPR	1D	Writes the value of an integer GPR into an IPR

These instructions generally produce an OPCDEC exception if executed while the processor is not in PALmode, however, if I_CTL<HWE> is set these instructions can be also be executed in kernel mode. Software which uses these instructions must adhere to the PALcode restrictions listed in this chapter.

4.1.1 HW_LD Instruction

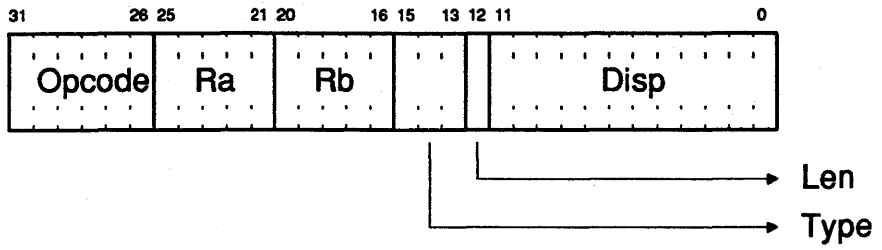
PALcode uses the HW_LD instruction to access memory outside the realm of normal Alpha memory management and to do special forms of D-stream loads. Data alignment traps are disabled for the HW_LD instruction.



Field	Value	Description
Opcode	$1B_{16}$	The Opcode value: $1B_{16}$
Ra		Destination register number
Rb		Base register for memory address
Type	000_2	Physical The effective address for the HW_LD is physical
	001_2	Physical/Lock The effective address for the HW_LD is physical. Load lock version of HW_LD.
	010_2	Virtual/VPTE Flags a virtual PTE fetch (LD_VPTE). Used by trap logic to distinguish single TB miss from double TB miss. Kernel mode access checks are performed
	100_2	Virtual The effective address for the HW_LD is virtual.
	101_2	Virtual/WrChk The effective address for the HW_LD is virtual. Access checks for FOR, FOW, read and write protection.
	110_2	Virtual/Alt The effective address for the HW_LD is virtual. Access checks use DTB_ALT_MODE IPR
	111_2	Virtual/WrChk/Alt The effective address for the HW_LD is virtual. Access checks for FOR, FOW, read and write protection. Access checks use DTB_ALT_MODE IPR
Len	0	Access length is longword
	1	Access length is quadword
Disp		Holds a 12-bit signed byte displacement

4.1.2 HW_ST Instruction

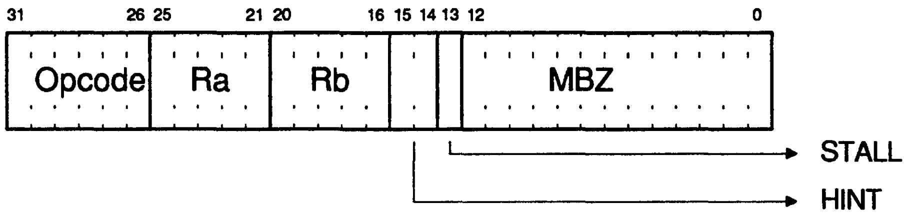
PALcode uses the HW_ST instruction to access memory outside the realm of normal Alpha memory management and to do special forms of D-stream store instructions. Data alignment traps are inhibited for HW_ST instructions.



Field	Value	Description
Opcode	$1F_{16}$	The Opcode value: $1F_{16}$
Ra		Write data register number
Rb		Base register for memory address
Type	000_2	Physical The effective address for the HW_ST is physical
	001_2	Physical/Cond The effective address for the HW_ST is physical. Store conditional version of HW_ST. The lock flag is returned in Ra. Refer to PAL restrictions for correct use of this function.
	010_2	Virtual The effective address for the HW_ST is virtual.
	110_2	Virtual/Alt The effective address for the HW_ST is virtual. Access checks use DTB_ALT_MODE IPR
	all others	Unused
Len	0	Access length is longword
	1	Access length is quadword
Disp		Holds a 12-bit signed byte displacement

4.1.3 HW_RET Instruction

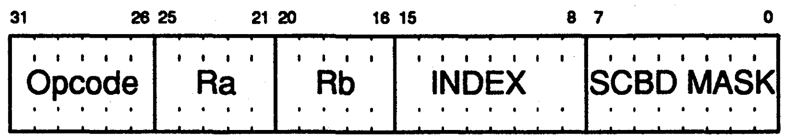
The HW_RET instruction is used to return instruction flow to a specified PC. The Rb field of the HW_RET instruction specifies an integer GPR which holds the new value of the PC. Bit <0> of this register provides the new value of PALmode after the HW_RET instruction is executed. Bits <15:14> of the instruction contain the stack action. Normally the HW_RET succeeds a CALL_PAL instruction or trap handler, which pushed its PC onto the prediction stack. In this mode, the HINT should be set to '10' to pop the PC and generate a predicted target address for the HW_RET. In certain circumstances, the HW_RET is used in the middle of a PAL flow to cause a group of instructions to retire. In these cases, if the HW_RET does not have a corresponding instruction which pushed a PC onto the stack, the HINT field should be set to '00' to keep the stack from being modified. In the rare circumstance that the HW_RET might be used like a JSR or JSR_COROUTINE, the stack can be managed by setting the HINT bits accordingly.



Field	Value	Description
Opcode	1E$_{16}$	The Opcode value: 1E$_{16}$
Ra		Register number. Should be R31.
Rb		Target PC of HW_RET. Bit<0> of the register's contents determines the new value of PALmode.
Hint	00	HW_JMP: PC is not pushed onto prediction stack; no predicted target
	01	HW_JSR: PC is pushed onto prediction stack; no predicted target
	10	HW_RET: prediction is popped off stack and used as target
	11	HW_COROUTINE: prediction is popped and used as target. PC is pushed onto stack
Stall		If set, the fetcher is stalled until the HW_RET is retired or aborted. EV6 will force a mispredict, kill instructions which were fetched beyond the HW_RET, refetch the target of the HW_RET and stall until the HW_RET is retired or aborted. Note that if instructions beyond the HW_RET have issued out-of-order they will be killed and refetched.

4.1.4 HW_MFPR and HW_MTPR Instructions

The HW_MFPR and HW_MTPR instructions are used to access internal processor registers. The HW_MFPR instruction reads the value from the specified IPR into the integer register specified by the Ra field of the instruction. The HW_MTPR instruction writes the value from the integer GPR specified by the Rb field of the instruction into the specified IPR.



Field	Value	Description
Opcode	19_{16}	The Opcode value for HW_MFPR: 19_{16}
	$1D_{16}$	The Opcode value for HW_MTPR: $1D_{16}$
Ra		Destination register for HW_MFPR. Should be R31 for HW_MTPR.
Rb		Source register for HW_MTPR. Should be R31 for HW_MFPR.
INDEX		IPR index.
SCBD MASK		Specifies which IPR scoreboard bits in the IQ are to be applied to this instruction. A set mask bit indicates that the corresponding IPR scoreboard bit should be applied to this instruction.

4.2 Internal Processor Register Access Mechanisms

Since the EV6 Ibox reorders instructions and executes instructions speculatively, extra hardware is required to provide software with the correct view of architecturally defined state. The Alpha architecture defines two classes of state - general-purpose registers and memory. Register renaming is used to provide architecturally correct register file behavior, while the Ibox and Mbox each have hardware dedicated to invisibly providing correct memory behavior to the programmer. Since the internal processor registers are implementation-specific state not defined by the Alpha architecture, access mechanisms for these registers may be defined which impose restrictions and limitations on the software which uses them. This section describes the hardware and software access mechanisms which are used for EV6's IPRs.

With respect to a particular IPR, each instruction type can be classified by how it affects and is affected by the value held by that IPR. **Explicit readers** are HW_MFPR instructions which explicitly read the value of the IPR. **Implicit readers** are instructions whose behavior is affected by the value of the IPR. For example, each load instruction is an implicit reader of the DTB. **Explicit writers** are HW_MTPR instructions which explicitly write a value into the IPR. **Implicit writers** are instructions which may write a value into the IPR as a side effect of execution. For example, a load instruction which generates an access violation is an implicit writer of the VA, MM_STAT, and EXC_ADDR IPRs. In EV6, only instructions which generate an exception will act as implicit IPR writers. Only certain IPRs, such as write-one-to-clear bits are both implicitly and explicitly written. The read-write semantics of these IPRs is controlled by software.

4.2.1 IPR Scoreboard Bits

In previous Alpha implementations, IPR registers were not scoreboarded in hardware, and software was required to schedule HW_MTPR and HW_MFPR instructions for each machine's pipeline organization in order to ensure correct behavior. This software scheduling task is more difficult in EV6 since the Ibox performs dynamic scheduling. Hence eight extra scoreboard bits are used within the IQ to help maintain correct IPR access order. The HW_MTPR and HW_MFPR instruction formats contain an eight-bit field which is used as an IPR scoreboard bit mask to specify which of the eight IPR scoreboard bits are to be applied to the instruction.

For HW_MTPR, if any of the unmasked scoreboard bits are set when the instruction is about to enter the IQ, then the instruction (and those behind it) is stalled outside the IQ until all the unmasked scoreboard bits are clear and the queue does not contain any implicit or explicit readers which were dependent on those bits when they entered the queue. When all the unmasked scoreboard bits are clear and the queue does not contain any of those readers, the instruction enters the IQ, and the unmasked scoreboard bits are set.

HW_MFPR instructions are stalled in the IQ until all their unmasked IPR scoreboard bits are clear.

Scoreboard bits <3:0> and <7:4> behave differently in regard to their effect on other instructions when set, and in regard to how they are cleared.

If any of scoreboard bits <3:0> are set when a load or store instruction enters the IQ, then that load or store will not issue from the IQ until those scoreboard bits are clear.

Scoreboard bits <3:0> are cleared when the HW_MTPR instructions which set them issue (or are aborted). Bits <7:4> are cleared when the HW_MTPR instructions which set them retire (or are aborted).

Bits <3:0> are used for the DTB_TAG and DTB_PTE register pairs within the DTB fill flows. These bits can be used to order writes to the DTB with respect to loads and stores. See sections 4.6.1 and 5.3.1. The assignment of IPRs to scoreboard bits is given in the next chapter.

Bit <0> is used in both DTB and ITB fill flows to trigger, in hardware, a light-weight memory barrier (TB-MB) to be inserted between a ld_vpte and the corresponding virtual-mode load which TB-missed.

4.2.2 Hardware Structure of Explicitly Written IPRs

IPRs which are written by software are physically implemented as two registers. When the HW_MTPR instruction which writes the IPR executes it writes its value to the first register. When the HW_MTPR instruction retires the contents of the first register are written into the second register. Instructions which either implicitly or explicitly read the value of the IPR do so from the second register. Read-after-write and write-after-write dependencies are managed using the IPR scoreboard bits. Write-after-read conflicts are avoided: the second register is not written until the writer retires, the writer won't retire before the previous reader retires, and the reader retires after it has read its value from the second register.

Some groups of IPRs are built using a single shared "first" register. To prevent write-after-write conflicts, IPRs which share a "first" register also share scoreboard bits.

4.2.3 Hardware Structure of Implicitly Written IPRs

Implicitly written IPRs are physically built using only a single level of register, however the IPR has two hardware states associated with it:

1. Default State: The contents of the register may be written when an instruction generates an exception. If an exception occurs, write a new value into the IPR and go to state 2.
2. Locked State: The contents of the register may only be overwritten by an excepting instruction which is older than the instruction associated with the contents of the IPR. If such an exception occurs, overwrite the value of thc IPR. When the triggering instruction, or instruction which is older than the triggering instruction, is killed by the Ibox, go to state 1.

4.2.4 IPR Access Ordering

IPR access mechanisms must allow values to be passed through each IPR from a producer to its intended consumers. The table below exhaustively list all the pair-wise instruction fetch orderings between instructions of the four IPR access types, specifies whether access order must be maintained, and if so, the mechanisms used to ensure correct ordering.

	First Instruction			
Second Instr.	Implicit Reader	Implicit Writer	Explicit Reader	Explicit Writer
Implicit Reader	Reads can be reordered	No IPRs in this class	Reads can be reordered	Scoreboard bits stall issue of reader until writer retires, or HW_RET/STALL is used to stall reader
Implicit Writer	No IPRs in this class	The hardware structure of implicitly written IPRs handles this case.	IPR-specific PALcode restrictions are required for this case. For example, reads might be required to be placed in certain locations in a PAL flow.	No IPRs in this class
Explicit Reader	Reads can be reordered	If the reader is in the PALcode routine invoked by the exception associated with the writer, then ordering is guaranteed.	Reads can be reordered	Scoreboard bits stall issue of reader until writer is retired.
Explicit Writer	Reader reads second latch. Writer can't write second latch until it retires.	Write-one-to-clear bits, or performance counter special case. For example, performance counter increments are typically not scoreboarded against reads.	Reader reads second latch. Writer can't write second latch until it retires	Scoreboard bits stall second writer in map stage until first writer retires.

4.2.5 IPRs and HW_RET Stalls

In some cases, correct ordering of an explicit write to an IPR followed by and implicit read of the IPR is guaranteed using the IPR scoreboard bits. However, if the instruction which implicitly reads the IPR does so before the issue stage of the pipeline then this method does not work. For example, modification of the ITB affects instructions before the issue stage of the pipeline. For this case PALcode must contain a

HW_RET instruction with its stall bit set before any instruction which implicitly reads the IPR(s) in question. This prevents instructions which are newer than the HW_RET from being successfully fetched, issued, and retired until after the HW_RET instruction is retired (or aborted).

4.3 PAL Shadow Registers

EV6 contains extra virtual integer registers which are available to PALcode for use as scratch space and storage for commonly used values. These registers are made available under the control the SDE<1:0> field of the I_CTL IPR.

Any PALcode which supports CALL_PAL instructions must leave one of SDE<1:0> set when the processor is native mode, since hardware writes a shadow PAL register with the return address of CALL_PAL instructions. See section 4.5.1.

4.4 PALcode Emulation of FPCR

The FPCR register contains two classes of bits, status and control, which are accessed via the MT_FPCR and MF_FPCR instructions. The register is physically implemented like an explicitly written IPR. It may be written with a value from the floating point register file via the MT_FPCR instruction. Architecturally compliant FPCR behavior requires PALcode assistance. There are three behaviors of the FPCR register which must be considered:

1. Correct operation of the status bits, which must be set when a floating point instruction encounters an exceptional condition, independent of whether a trap for the condition is enabled.
2. Correct values when read via the MF_FPCR instruction.
3. Correct behavior when written via the MT_FPCR instruction.

4.4.1 Status Flags

The FPCR status bits in EV6 are set with PALcode assistance. Floating point exceptions for which the associated FPCR status bit is clear, or for which the associated trap is enabled, result in a hardware trap to the ARITH PALcode routine. The EXC_SUM register contains information to allow this routine to update the FPCR appropriately, and to decide whether to report the exception to the operating system.

4.4.2 MF_FPCR

The MF_FPCR is issued from the floating point queue and executed by the Fbox. No PALcode assistance is required.

4.4.3 MT_FPCR

The MT_FPCR instruction is issued from the floating point queue. This instruction is implemented as an explicit IPR write: the value is written into the "first" latch, and when the instruction retires the value is written into the "second" latch. There is no IPR scoreboarding mechanism in the floating point queue, however, so PALcode assistance is required to ensure that subsequent readers of the FPCR get the updated value.

Subsequent to writing the "first latch," the MT_FPCR instruction invokes a synchronous trap to the MT_FPCR PALcode entry point. The PALcode can simply return using a HW_RET instruction with its STALL bit set. This sequence ensures that the MT_FPCR instruction will be correctly ordered with respect to subsequent readers of the FPCR.

4.5 PALcode Entry Points

PALcode is invoked at specific entry points, of which there are two classes: CALL_PAL and exceptions.

4.5.1 CALL_PAL entry

CALL_PAL entry points are used whenever the Ibox encounters a CALL_PAL instruction in the instruction stream. In order to speed the processing of CALL_PAL instructions, they do not invoke pipeline aborts, but are processed as normal jumps to the offset from the contents of the PAL_BASE register which is specified by the CALL_PAL's function field. The IBOX fetches a CALL_PAL instruction, bubbles one cycle, and then fetches the instructions at the CALL_PAL entry point. For convenience of implementation, returns from CALL_PAL are aided by a linkage register (much like JSR's). A PAL shadow register is used as the linkage register - the Ibox loads it with the PC of the instruction after the CALL_PAL instruction. Bit <0> of the linkage register is set if the CALL_PAL was executed while the processor was in PAL mode. If I_CTL<NT_MODE> is clear then PAL shadow R27 is the linkage register, otherwise PAL shadow R23 is used. The Ibox also pushes the value of the return PC onto the return prediction stack. CALL_PAL instructions start at the following offsets:

- Privileged CALL_PAL instructions start at offset 2000_{16}
- Nonprivileged CALL_PAL instructions start at offset 3000_{16}

Each CALL_PAL instruction includes a function field which is used to calculate the PC of the its associated PALcode entry point. The PALcode OPCDEC flow will be invoked if the CALL_PAL function field is:

- in the range of 40_{16} to $7F_{16}$ inclusive, or
- greater than BF_{16}, or
- Between 00_{16} and $3F_{16}$, inclusive, and PS<CUR_MODE> is not equal to kernel

If none of the above conditions are met, then the PALcode entry point PC is as follows:

- PC<64:15> = PAL_BASE<63:15>
- PC<14> = 0
- PC<13> = 1
- PC<12> = CALL_PAL function field <7>
- PC<11:6> = CALL_PAL function field <5:0>
- PC<5:1> = 0
- PC<0> = 1 (PALmode)

4.5.2 PALcode Exception Entry Points

When hardware encounters an exception the Ibox jumps to a PALcode entry point at a PC determined by the type of exception, and writes the PC of the instruction which triggered the exception into the EXC_ADDR register and onto the top of the return prediction stack.

The table below shows the PALcode exception entry points and their offset from the PAL_BASE IPR The entry points are listed in decreasing order of priority.

Entry Name	Type	Offset_{16}	Description
DTBM_DOUBLE_3	Fault	100	D-stream TB miss on virtual page table entry fetch. Use three-level flow
DTBM_DOUBLE_4	Fault	180	D-stream TB miss on virtual page table entry fetch. Use four-level flow.
FEN	Fault	200	Floating point disabled
UNALIGN	Fault	280	D-stream unaligned reference
DTBM_SINGLE	Fault	300	D-stream TB miss
DFAULT	Fault	380	D-stream fault or virtual address sign check error

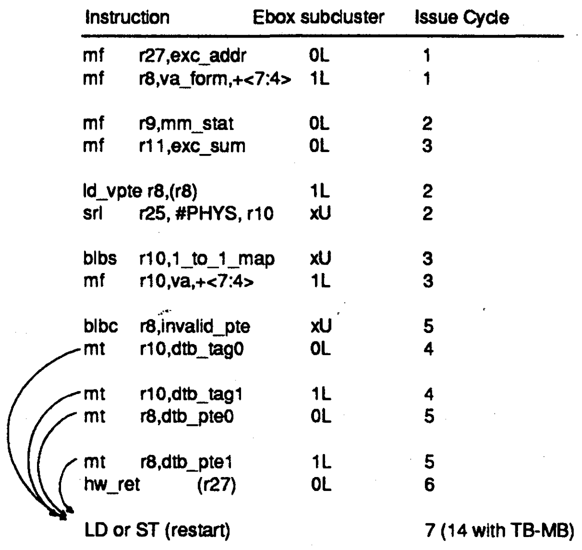
Entry Name	Type	Offset$_{16}$	Description
OPCDEC	Fault	400	Illegal opcode or function field: • opcode 1, 2, 3, 4, 5, 6 or 7 • opcode 19_{16}, $1B_{16}$, $1D_{16}$, $1E_{16}$ or $1F_{16}$, not PAL mode or not I_CTL<HWE> • extended precision IEEE format • unimplemented function field of opcodes 14_{16} or $1C_{16}$
IACV	Fault	480	I-stream access violation or virtual address sign check error
MCHK	Interrupt	500	Machine Check
ITB_MISS	Fault	580	I-stream TB miss
ARITH	Synch. Trap	600	Arithmetic exception or update to FPCR
INTERRUPT	Interrupt	680	Interrupts: hardware, software and AST
MT_FPCR	Synch. Trap	700	Invoked when a MT_FPCR instruction is issued.
RESET/WAKEUP	Interrupt	780	Chip reset or wakeup from sleep mode

4.6 TB Fill Flows

This section shows the expected PALcode flows for DTB miss and ITB miss. Familiarity with EV6's IPRs is assumed. See chapter 5.

4.6.1 DTB Fill

The single-miss DTB flow is shown below:



```
Instruction             Ebox subcluster   Issue Cycle

mf      r27,exc_addr        0L              1
mf      r8,va_form,+<7:4>   1L              1

mf      r9,mm_stat          0L              2
mf      r11,exc_sum         0L              3

ld_vpte r8,(r8)             1L              2
srl     r25, #PHYS, r10     xU              2

blbs    r10,1_to_1_map      xU              3
mf      r10,va,+<7:4>       1L              3

blbc    r8,invalid_pte      xU              5
mt      r10,dtb_tag0        0L              4

mt      r10,dtb_tag1        1L              4
mt      r8,dtb_pte0         0L              5

mt      r8,dtb_pte1         1L              5
hw_ret        (r27)         0L              6

LD or ST (restart)                          7 (14 with TB-MB)
```

Here are some notes with respect to this flow:

- r8, r9, r10, & r27 are PAL shadows.
- The arcs show issue order dependencies that are not related to register data.
- IPR scoreboard bits <3:0> are used to order the restarted load or store with respect to the DTB writes.
- MM_STAT and VA will not be overwritten if the LD_VPTE instruction misses the DTB - there is no issue order constraint here.
- The code is written to prevent a later execution of the DTB fill from issuing ahead of a previous execution and corrupting the previous write to the TB registers. This is accomplished by placing code

dependencies on scoreboard bits <7:4> in the path of the successive writers. This keeps the successive writers from issuing ahead of the retiring of the previous writers.

- The issue of MTPR DTB_PTE0 triggers, in hardware, a light-weight memory barrier (TB-MB) which enforces read-ordering of stores from another processor (I) to this processor's (J) page table and this processor's virtual memory area such that if this processor sees the write to the PTE from (I) it will see the new data.:

Processor I	Processor J
Wr Data	LD/ST
MB	<tb miss>
Wr PTE	LD-PTE, write TB
	LD/ST

- The conditional branch is placed in the code so that all of the MTPR's issue and retire or none of them issue and retire. This allows the TB fill hardware to update the TB whenever it sees the retiring of PTE1 and to ignore writes to TAG0/TAG1/PTE0/PTE1 in the interim between the issuing of those writes and a retire of PTE1.

4.6.2 ITB Fill

The ITB miss flow is shown below:

Instruction		Ebox subcluster	Issue Cycle	
mf	r8,iva_form	0L	1	
mf	r27,exc_addr	0L	x	;
ld_vpte r8,(r8)		xL	4	; get PTE
lda	r9,0x0fff	xU	2	; create mask for prot
and	r8,r9,r9	xx	7	; get prot bits
srl	r25,#PHYS,r10	xU	x	
blbs	r10,1_to_1	xU	x	
srl	r8,#19,r10	xU	7	
sll	r10 #PTE_PFN,r10	xU	8	; put PFN in place
and	r8,#foe_bit,r11	xL	8	; get FOE bit
blbc	r8,invalid_pte	xU	x	
bne	r11,foe_pte	xU	x	
bis	r9,r10,r10	xL	9	; PTE in ITB format
mt	r27,itb_tag	0L	6	
mt	r10,itb_pte	0L	10	
hw_ret/stall (r27)		0L	5	; hw_ret/stall

(istream restart)

Here are some notes with respect to this flow:

- The ITB is only accessed on Icache misses
- r8, r9, r10, r11 & r27 are PAL shadows.
- The arcs show issue order dependencies that are not related to register data.
- The HW_RET instruction should have its STALL bit set to ensure that the restarted I-stream does not read the ITB until the ITB is written.

5. Internal Processor Registers

This chapter describes EV6's internal processor registers (IPRs).

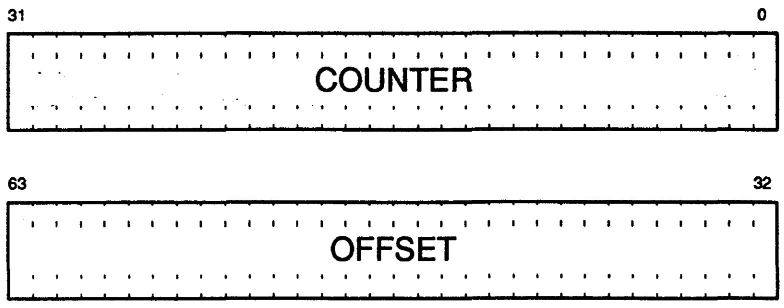
IPR Mnemonic	$Index_2$	Score-board Bit	Access Type	MT/MF Issued from Ebox Pipe:	Latency for MFPR
Ebox IPRs					
CC	1100 0000	5	RW	1L	1
CC_CTL	1100 0001	5	W	1L	
VA	1100 0010	4,5,6, & 7	R	1L	1
VA_FORM	1100 0011	4,5,6, & 7	R	1L	1
VA_CTL	1100 0100	5	W	1L	
Ibox IPRs					
ITB_TAG	0000 0000	6	W	0L	
ITB_PTE	0000 0001	4 & 0	W	0L	
ITB_IAP	0000 0010	4	W	0L	
ITB_IA	0000 0011	4	W	0L	
ITB_IS	0000 0100	4 & 6	W	0L	
EXC_ADDR	0000 0110		R	0L	3
IVA_FORM	0000 0111		R	0L	3
CM	0000 10x1	4	RW	0L	3
IER	0000 101x	4	RW	0L	3
SIRR	0000 1100	4	RW	0L	3
ISUM	0000 1101		R	0L	3
HW_INT_CLR	0000 1110	4	W	0L	
EXC_SUM	0000 1111		R	0L	3
PAL_BASE	0001 0000	4	RW	0L	3
I_CTL	0001 0001	4	RW	0L	3
IC_FLUSH	0001 0011	4	W	0L	
PCTR_CTL	0001 0100	4	RW	0L	3
CLR_MAP	0001 0101	4,5,6 & 7	W	0L	
SLEEP	0001 0111	4,5,6 & 7	W	0L	
I_STAT	0001 0110		RW	0L	3
ASN	01xx xxx1	4	RW	0L	3
ASTER	01xx xx1x	4	RW	0L	3
ASTRR	01xx x1xx	4	RW	0L	3
PPCE	01xx 1xxx	4	RW	0L	3
FPE	01x1 xxxx	4	RW	0L	3
Mbox IPRs					
DTB_TAG0	0010 0000	2 & 6	W	0L	
DTB_TAG1	1010 0000	1 & 5	W	1L	
DTB_PTE0	0010 0001	0 & 4	W	0L	
DTB_PTE1	1010 0001	3 & 7	W	1L	
DTB_IAP	1010 0010	7	W	1L	
DTB_IA	1010 0011	7	W	1L	
DTB_IS0	0010 0100	6	W	0L	
DTB_IS1	1010 0100	7	W	1L	
DTB_ASN0	0010 0101	4	W	0L	
DTB_ASN1	1010 0101	7	W	1L	

IPR Mnemonic	Index$_2$	Score-board Bit	Access Type	MT/MF Issued from Ebox Pipe:	Latency for MFPR
DTB_ALT_MODE	0010 0110	6	W	0L	
MM_STAT	0010 0111		R	0L	3
M_CTL	0010 1000	6	W	0L	
DC_CTL	0010 1001	6	W	0L	
DC_STAT	0010 1010	6	RW	0L	3
Cbox IPRs					
DATA	0010 1011	6	RW	0L	3
SHIFT_CONTROL	0010 1100	6	W	0L	
TBox IPRs					
SL_XMIT			W		
SL_RCV			R		3

5.1 Ebox IPRs

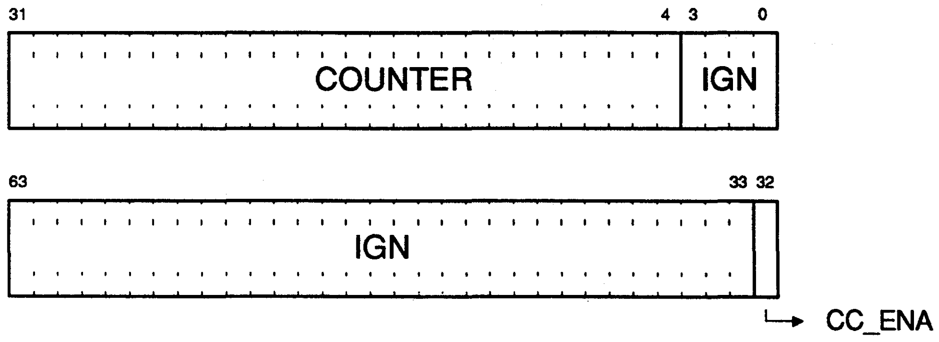
5.1.1 CC

The cycle counter register (CC) is a read/write register. The lower half of CC is a counter which, when enabled via CC_CTL<32>, increments once each CPU cycle. The upper half of the register is simply 32 bits of register storage which may be used as a counter offset as described in the Alpha SRM. A HW_MTPR to the CC register writes the upper half of the register and leaves the lower half unchanged. The RPCC instruction returns the full 64-bit value of the register.



5.1.2 CC_CTL

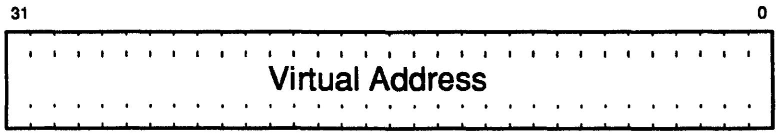
The cycle counter control register (CC_CTL) is a write only register through which the lower half of the CC register may be written and its associated counter enabled and disabled.

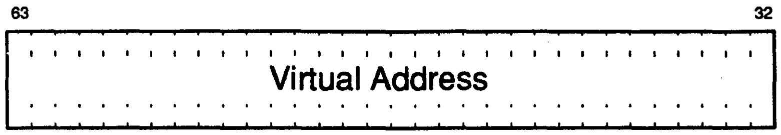


Name	Type	Description
Counter<31:4>	W	This is the field through which CC<31:4> may be written. Writes to CC_CTL result in CC<3:0> being cleared.
CC_ENA	W	Counter enable. When set, this bit allows the cycle counter to increment.

5.1.3 VA

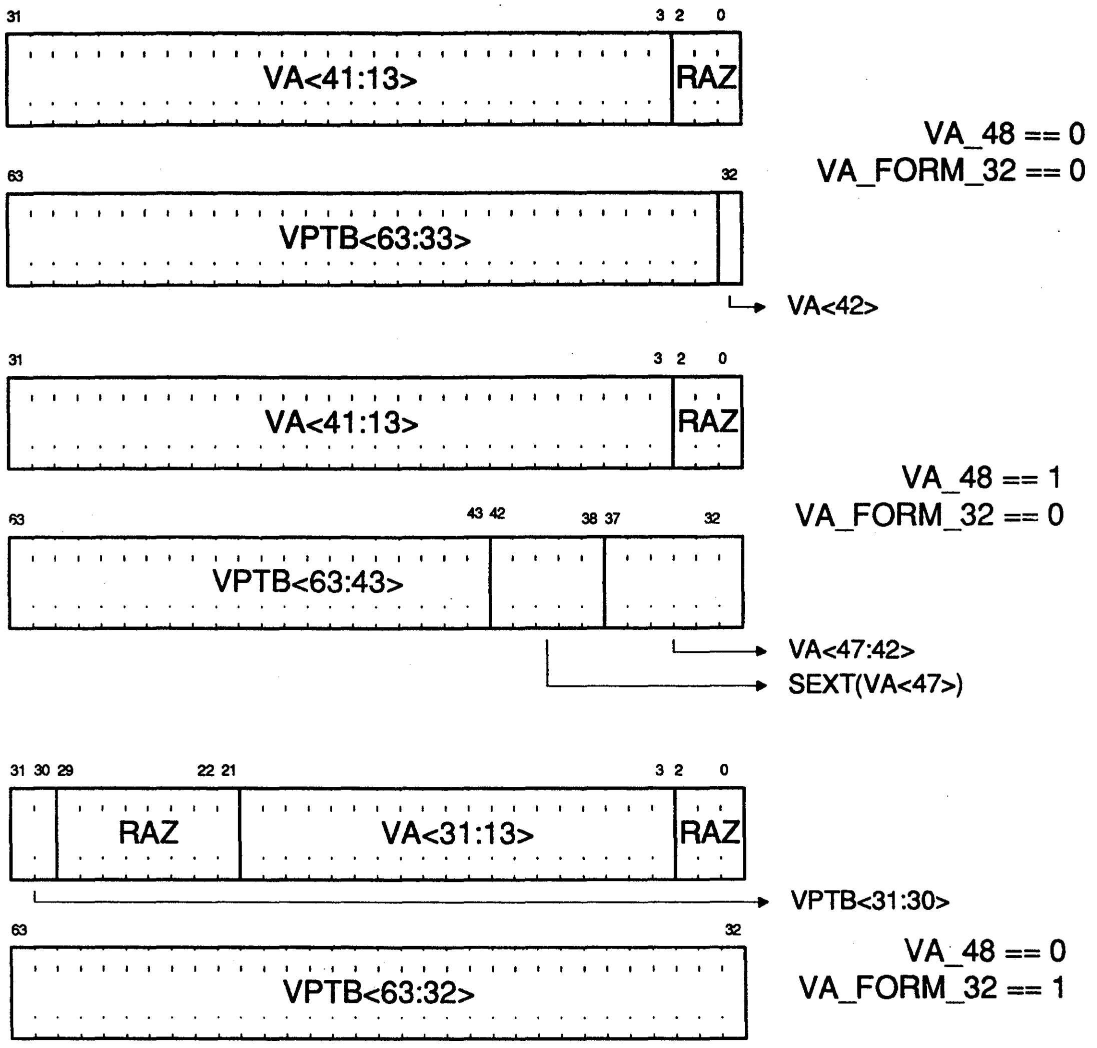
VA is a read-only register. When a D-stream TB miss or fault occurs the associated effective virtual address is written into the VA register. VA is not written when a LD_VPTE gets a DTB miss or D-fault.





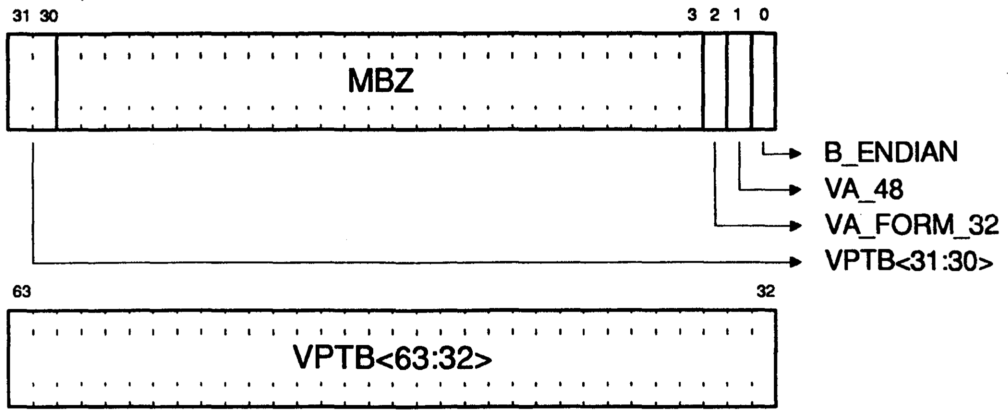
5.1.4 VA_FORM

VA_FORM is a read-only register containing the virtual page table entry address derived from the faulting virtual address stored in the VA register, and from the virtual page table base and associated control bits stored in the VA_CTL register.



5.1.5 VA_CTL

VA_CTL is a write-only register which controls the way in which the faulting virtual address stored in the VA register is formatted when read via the VA_FORM register. It also contains control bits which effect the behavior of the memory pipe virtual address sign extension checkers, and the behavior of the Ebox extract, insert and mask instructions.



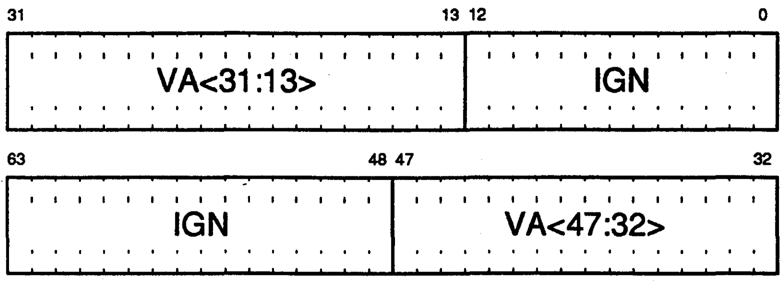
Name	Type	Description
B_ENDIAN	W,0	Big Endian Mode. When set • the shift amount (Rbv<2:0>) is inverted for EXTxx, INSxx and MSKxx instructions • the lower bits of the physical address for D-stream accesses are inverted based upon the length of the reference: ⇒ Byte: invert bits <2:0> ⇒ Word: invert bits <2:1> ⇒ Longword: inverts bit <2>
VA_48	W,0	This bit controls the format applied to effective virtual addresses by the VA_FORM register and the memory pipe virtual address sign extension checkers. When VA_48 is clear, 43-bit virtual address format is used, and when VA_48 is set, 48-bit virtual address format is used. The effect of VA_48 on the VA_FORM register is described above. When VA_48 is set the sign extension checkers generate an ACV if: va<63:0> != SEXT(va<47:0>) When VA_48 is clear and the sign extension checkers generate an ACV if: va<63:0> != SEXT(va<42:0>)
VA_FORM_32	W,0	This bit is used to control address formatting on a read of the VA_FORM register. See the section on the VA_FORM register for details.
VPTB<63:30>	W	Virtual Page Table Base. See the VA_FORM register section for details.

5.2 Ibox IPRs

This section describes the IPRs which control Ibox functions.

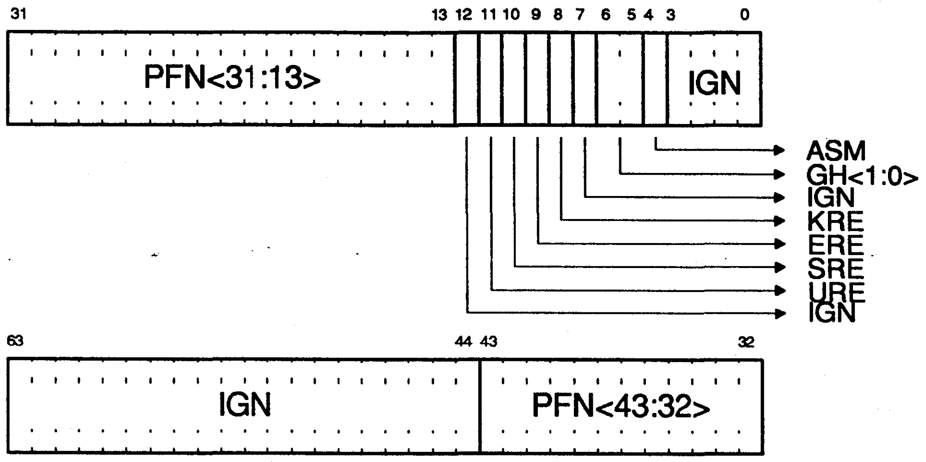
5.2.1 ITB_TAG

ITB_TAG is a write-only register through which the ITB tag array is written. A write to ITB_TAG actually writes a register outside the ITB array. When a write to the ITB_PTE register is retired, the contents of both the ITB_TAG and ITB_PTE registers are written into the ITB entry. The specific ITB entry that is written is determined by a round-robin mechanism; the mechanism writes to entry #0 as the first entry after chip reset.



5.2.2 ITB_PTE

ITB_PTE is a write-only register through which the ITB PTE array is written. A write to the ITB_PTE array, when retired, results in both the ITB_TAG and ITB_PTE arrays being written. The specific entry that is written is chosen by the round-robin mechanism described above.



5.2.3 ITB_IAP

ITB_IAP is a pseudo register which, when written to, invalidates all ITB entries and Icache blocks whose ASM bit is clear. The Icache flush will not occur until after the retire of the next encountered HW_RET/stall.

5.2.4 ITB_IA

ITB_IA is a pseudo register which, when written to, invalidates all ITB entries and invalidates the entire Icache. The Icache flush will not occur until after the retire of the next encountered HW_RET/stall.

5.2.5 ITB_IS

I-stream Translation Buffer Invalidate Single (ITB_IS) is a write-only register. Writing a virtual page number to this register invalidates any ITB entry which meets one of the following criteria:

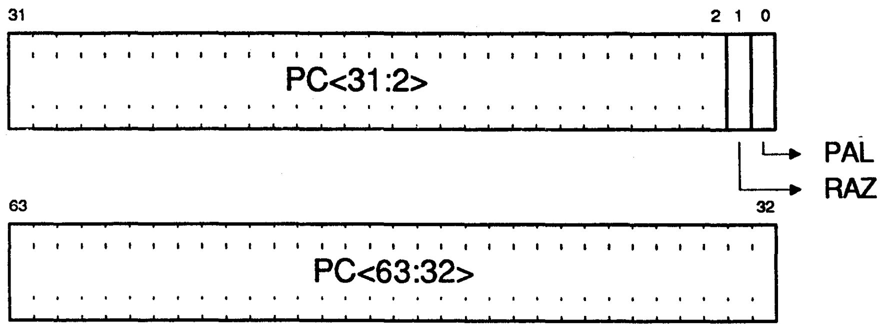
- the ITB entry's virtual page number matches ITB_IS<47:13> (or fewer bits if granularity hint bits are set in the ITB entry) and its ASN field matches the address space number supplied in the process context IPR: PCTX<46:39>.
- the ITB entry's virtual page number matches ITB_IS<47:13> and its ASM bit is set

Note that since the Icache is virtually indexed and tagged, it is normally not necessary to flush the icache when paging. Therefore a write to ITB_IS will not flush the icache.

5.2.6 EXC_ADDR

The Exception Address (EXC_ADDR) register is a read-only register that is updated by hardware when it encounters an exception or interrupt. If the exception was a fault, EXC_ADDR contains the PC of the instruction which triggered the fault. If the exception was a synchronous trap, EXC_ADDR contains the PC of the instruction after that which triggered the trap. For an interrupt, EXC_ADDR contains the PC of the next instruction which would have executed if the interrupt had not occurred.

EXC_ADDR<0> is set if the associated exception occurred in PAL mode.

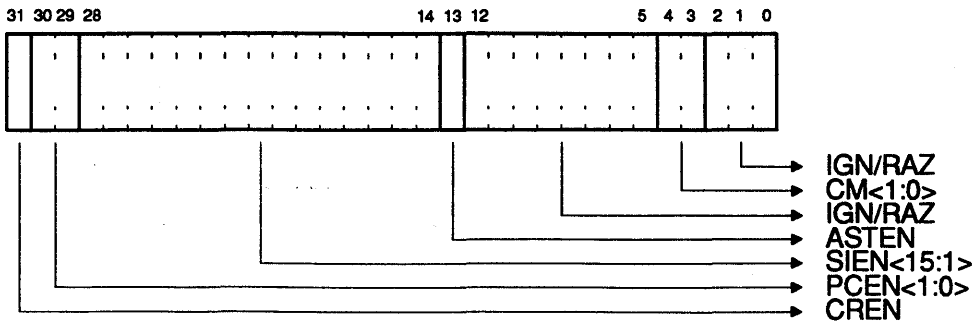


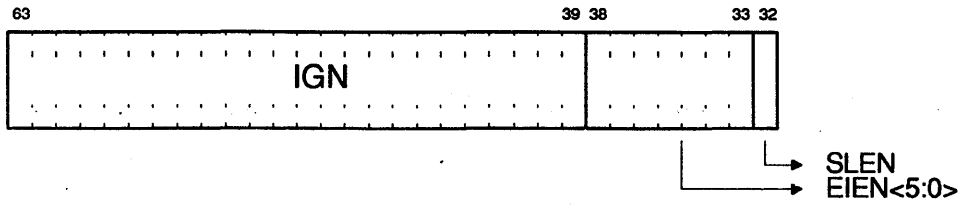
5.2.7 IVA_FORM

IVA_FORM is a read-only register containing the virtual page table entry address derived from the faulting virtual address stored in the EXC_ADDR register, and from the virtual page table base, VA_48 and VA_FORM_32 bits stored in the I_CTL register. The IVA_FORM bit format is identical to VA_FORM. See section 5.1.4

5.2.8 IER_CM

IER_CM is a register which contains the interrupt enable (all active fields of the register except CM<1:0>) and current processor mode (CM<1:0>) bit fields. These two bit fields may be written either individually or together with a single HW_MTPR instruction. When bits <7:2> of the IPR index field of a HW_MTPR instruction contain the value 000010_2, this register is selected. Bits <1:0> of the IPR index indicate which bit fields are to be written: bit<1> corresponds to the IER field, bit<0> corresponds to the processor mode field. A HW_MFPR of this register returns the values in both fields.

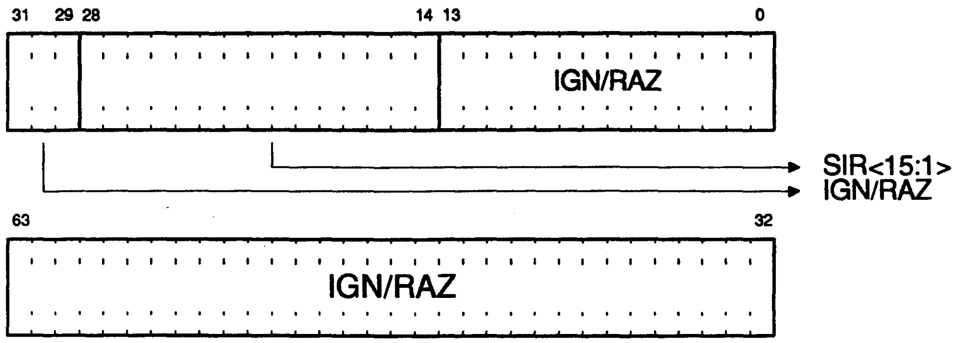




Name	Type	Description	
CM<1:0>	RW	Current Mode:	
		00	Kernel
		01	Executive
		10	Supervisor
		11	User
ASTEN	RW	AST Interrupt Enable. When set enables those AST interrupt requests which are also enabled by the value in ASTER.	
SIEN<15:1>	RW	Software Interrupt Enables	
PCEN<1:0>	RW	Performance Counter Interrupt Enables	
CREN	RW	Corrected Read Error Interrupt Enable	
SLEN	RW	Serial Line Interrupt Enable	
EIEN<5:0>	RW	External Interrupt Enable	

5.2.9 SIRR

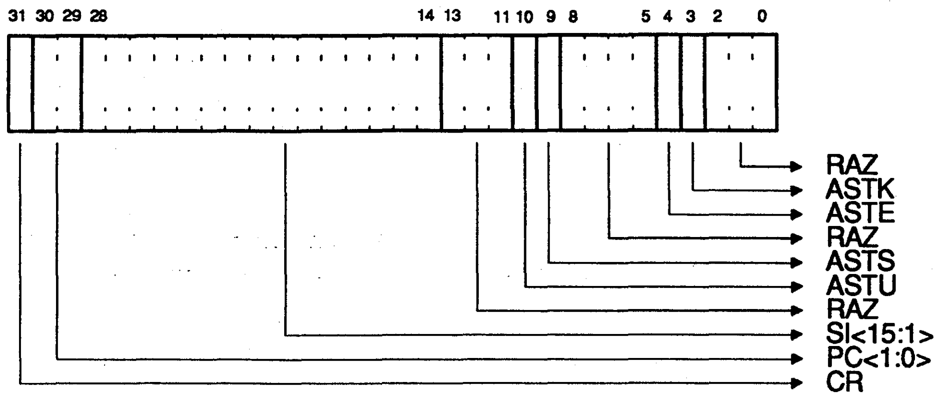
The Software Interrupt Request Register (SIRR) is a read/write register containing bits to request software interrupts. In order to generate a particular software interrupt, its corresponding bits in SIRR and IER<SIER> must both be set.

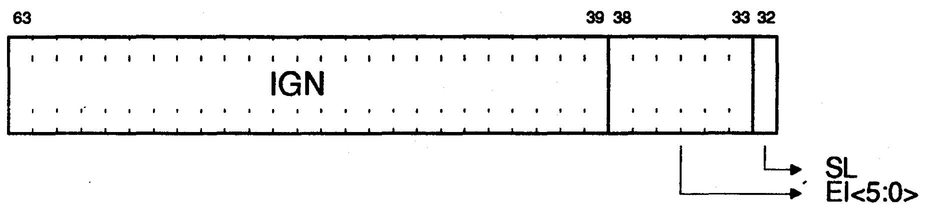


Name	Type	Description
SIR	RW	Software interrupt requests

5.2.10 ISUM

The Interrupt Summary (ISUM) register is a read-only register that records all pending hardware, software and AST interrupt requests.

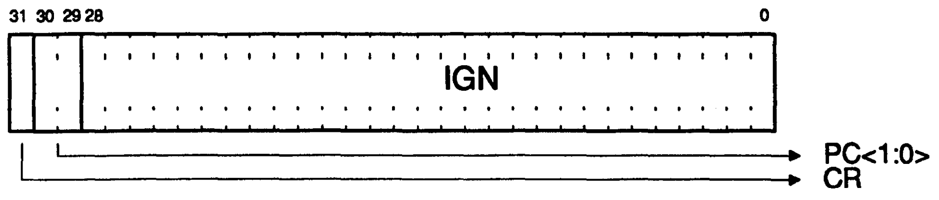


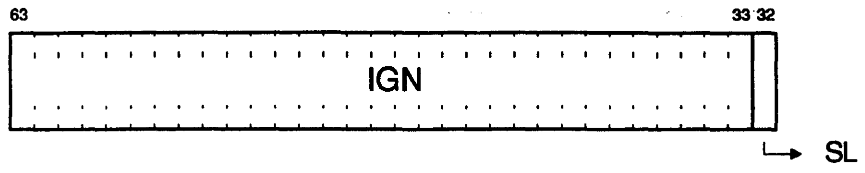


Name	Type	Description
ASTx	R	AST Interrupts. For each processor mode, records whether an associated AST interrupt is pending. This include the mode's ASTER and ASTRR bits, and whether the processor mode value held in the CM register is greater than or equal to the value for the mode.
SI<15:1>	R	Software Interrupts
PC<1:0>	R	Performance Counter Interrupts
CR	R	Corrected Read Error Interrupts
SL	R	Serial Line Interrupt
EI<5:0>	R	External Interrupts

5.2.11 HW_INT_CLR

HW_INT_CLR is a write-only register used to clear edge-sensitive interrupt requests



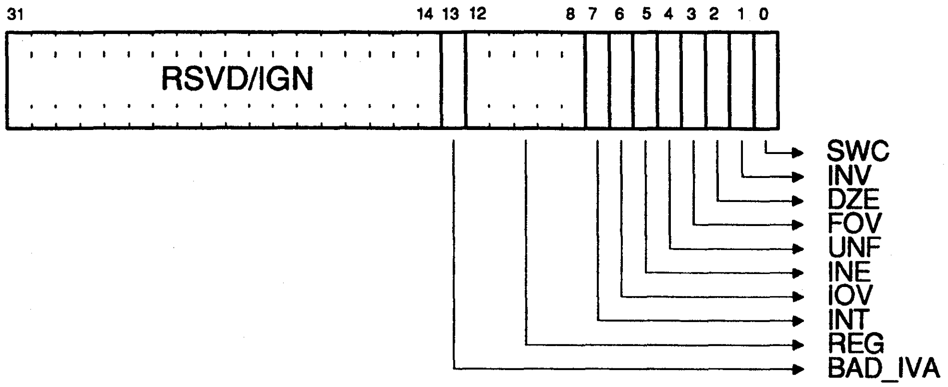


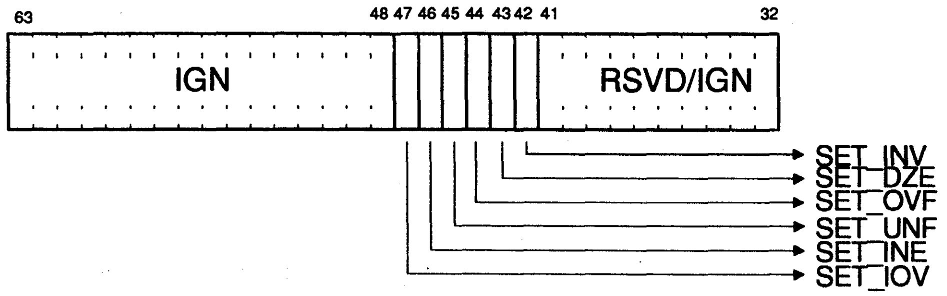
Name	Type	Description
PC<1:0>	W1C	Clears performance counter interrupt requests
CR	W1C	Clears corrected read error interrupt request
SL	W1C	Clears serial line interrupt request

5.2.12 EXC_SUM

The Exception Summary (EXC_SUM) register is a read-only register which records information about instructions which triggered traps. The register is updated at trap delivery time; its contents are only valid if it is read (via a HW_MFPR) in the first fetch block of the exception handler. There are three types of traps for which this register captures related information:

- Arithmetic traps: the instruction generated an exceptional condition which should be reported to the operating system, and/or the FPCR status bit associated with this condition is clear and should be set by PALcode. Additionally, the REG field contains the register number of the destination specifier for the instruction which triggered the trap.
- I-stream ACV: The BAD_IVA bit of this register indicates whether the offending I-stream virtual address is latched into the EXC_ADDR or VA register.
- D-stream Exceptions: The REG field contains the register number of either the source specifier (for stores) or the destination specifier (for loads) of the instruction which triggered the trap.



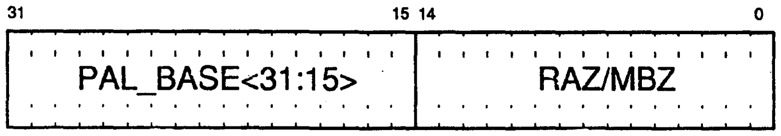


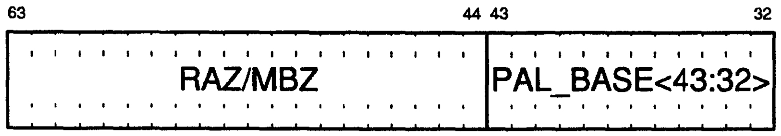
Name	Type	Description
SWC	R	Indicates software completion possible. This bit is set if the instruction which triggered the trap contained the /S modifier.
INV	R	Indicates invalid operation trap
DZE	R	Indicates divide by zero trap
FOV	R	Indicates floating point overflow trap
UNF	R	Indicates floating point underflow trap
INE	R	Indicates floating point inexact error trap
IOV	R	Indicates Fbox convert to integer overflow or Ebox integer overflow trap
INT	R	Set to indicate Ebox integer overflow trap, clear to indicate Fbox trap condition
REG	R	Destination register of load or operate which triggered the trap OR source register of store which triggered the trap. These bits may contain

		the Rc field of an operate instruction or the Ra field of a load or store instruction. The value is unpredictable if the trap was triggered by an ITB miss, interrupt, OPCDEC, or other non load/st/operate.
BAD_IVA	R	Bad I-stream VA. This bit should be used by the IACV PALcode routine to determine whether the offending I-stream virtual address is latched in the EXC_ADDR register or the VA register. If BAD_IVA is clear, then EXC_ADDR contains the address, if BAD_IVA is set then VA contains the address.
RSVD/IGN	R,0	Reserved for hardware use.
SET_INV	R	PALcode should set FPCR<INV>
SET_DZE	R	PALcode should set FPCR<DZE>
SET_OVF	R	PALcode should set FPCR<OVF>
SET_UNF	R	PALcode should set FPCR<UNF>
SET_INE	R	PALcode should set FPCR<INE>
SET_IOV	R	PALcode should set FPCR<IOV>

5.2.13 PAL_BASE

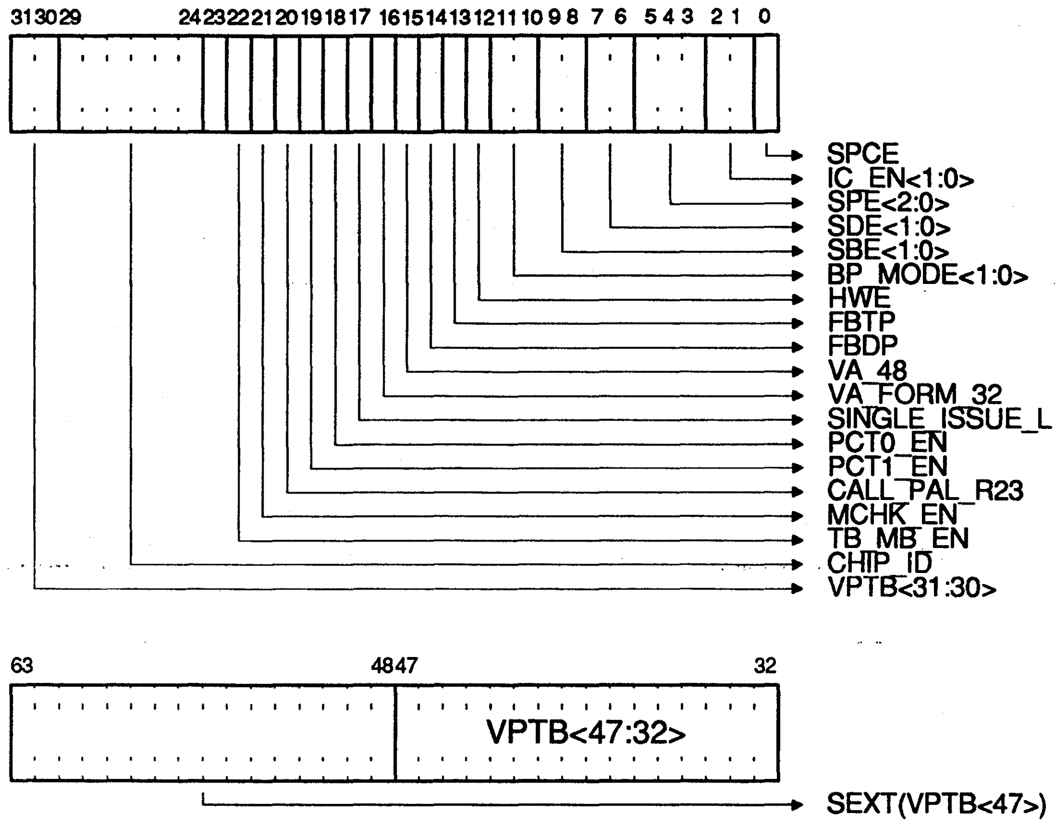
PAL_BASE is a read/write register which contains the base physical address for PALcode. Its contents are cleared by chip reset.





5.2.14 I_CTL

Ibox Control (I_CTL) is a read/write register which controls various Ibox functions. Its contents are cleared by chip reset.



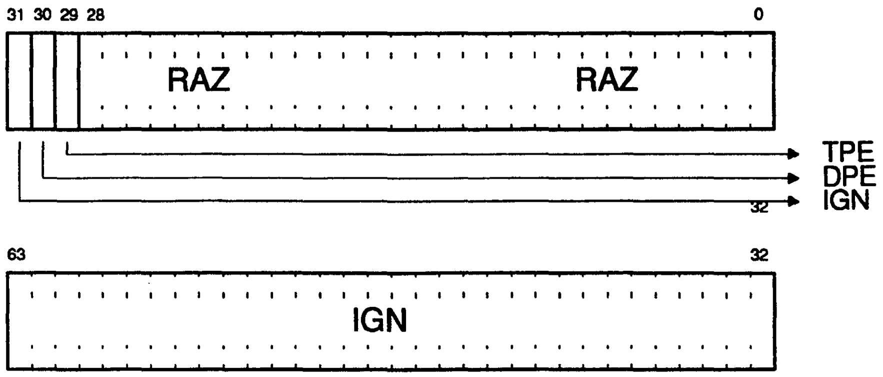
Name	Type	Description
SPCE	RW,0	System Performance counter enable. A performance counter is enabled if its individual enable is asserted (PCTR0 or PCTR1) and either SPCE or the PPCE bit of the Ibox process context IPR is set.
IC_ENABLE<1:0>	RW,3	Icache set enable. The entire cache may be enabled by setting both bits. Zero, one, or two icache sets can be enabled.
SPE<2:0>	RW,0	Super page mode enables - just like the SPE bits in the MBOX M_CTL IPR.
SDE<1:0>	RW,0	When set, enables access to the PAL shadow registers. If SDE<0> is set, R8-R11& R24-R27 are used as PAL shadows. If SDE<1> is set, R4-R7 & R20-R23 are used as PAL shadows. Both SDE<0> and SDE<1> may be set. However, this reduces the size of the physical integer register free pool, and may reduce overall system performance.
SBE<1:0>	RW,0	Stream Buffer Enable. The value in this bit field specifies the number of stream buffer prefetches (besides the demand-fill) which are launched after an Icache miss. If the value is zero, only demand requests are launched.
BP_MODE<1:0>	RW,0	Branch prediction mode selection:

Name	Type	Description
		BP_MODE<1>: If set, forces all branches to be predicted fall-thru. If clear, the dynamic branch predictor is chosen. BP_MODE<0>: If set, the dynamic branch predictor chooses local history prediction. If clear, the dynamic branch predictor chooses local or global prediction based on the state of the chooser.
HWE	RW,0	If set, allow PALRES intructions to be executed in kernel mode. Note that modification of the ITB while in kernel mode/native mode may cause unpredictable behavior.
FBTP	RW,0	When set, forces bad Icache tag parity on fills.
FBDP	RW,0	When set, forces bad Icache data parity on fills.
VA_48	RW,0	This bit controls the format applied to effective virtual addresses by the IVA_FORM register and the Ibox virtual address sign extension checkers. When VA_48 is clear, 43-bit virtual address format is used, and when VA_48 is set, 48-bit virtual address format is used. The effect of this bit on the IVA_FORM register is identical to the effect of VA_CTL<VA_48> on the VA_FORM register. See section 5.1.4 When VA_48 is set the sign extension checkers generate an ACV if: va<63:0> != SEXT(va<47:0>) When VA_48 is clear the sign extension checkers generate an ACV if: va<63:0> != SEXT(va<42:0>) This bit also affects three additional functions: (1) JSR return address: The address is sign-extended from bit 47. Otherwise it is sign-extended from bit 43. (2) PC adder: The PC incrementer generates addresses in a 48-bit virtual address space instead of a 44 bit virtual address space. (3) DTB_DOUBLE Traps: if set, the DTB double miss traps vector to the DTB_DOUBLE_4 entry point.
VA_FORM_32	RW,0	This bit controls address formatting on a read of the IVA_FORM register. See the section 5.1.4 5.1.4
SINGLE_ISSUE_L	RW,0	When clear, this bit forces instructions to issue only from the bottom-most entries of the IQ and FQ.
PCT0_EN	RW,0	Enable performance counter #0. If this bit is one, the performance counter will count if EITHER the system (SPCE) or process (PPCE) performance counter enable is set.
PCT1_EN	RW,0	Enable performance counter #1. If this bit is one, the performance counter will count if EITHER the system (SPCE) or process (PPCE) performance counter enable is asserted.
CALL_PAL_R23	RW,0	When set, the CALL_PAL linkage register is R23, when clear it's R27. This choice should correspond to SDE so as to ensure that a shadow register is used as the linkage register.
MCHK_EN	RW,0	Machine check enable - set to enable machine checks.
TB_MB_EN	RW,0	When set, the hardware ensures that the virtual-mode loads in DTB and ITB fill flows which access the page table and the subsequent virtual mode load or store which is being retried are 'ordered' relative to another processor's stores. This must be set for multiprocessor systems in which no MB instruction is present in the TB fill flow.

Name	Type	Description
CHIP_ID<5:0>	R	This is a read-only field which supplies the revision ID number for the EV6 part. EV6 pass 1 parts will have a chip ID of 000001.
VPTB<63:30>	RW,0	Virtual Page Table Base. See section 5.1.4 for details.

5.2.15 I_STAT

Ibox Status (I_STAT) is a read/write register which contains Ibox status information.



Name	Type	Description
TPE	R,W1C	If set, an Icache tag parity error occurred.
DPE	R,W1C	If set, an Icache data parity error occurred.

5.2.16 IC_FLUSH

IC_FLUSH is a pseudo register which, when written, results in all Icache blocks being invalidated. The cache is actually flushed at the retire of the next encountered HW_RET/STALL instruction.

5.2.17 CLR_MAP

CLR_MAP is a pseudo register which, when written, results in the clearing of the current map of virtual to physical registers. This register must only be written after there are no register-borne dependencies present and there are no unretired instructions. See PALcode restrictions for a usage example.

5.2.18 SLEEP

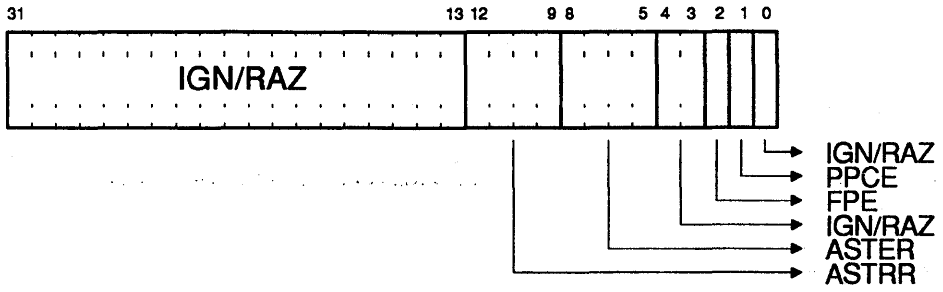
SLEEP is a pseudo register which, when written, results in the PLL speed being reduced and the chip entering a low-power mode. This register must only be written after a sequence of code has been run which saves all necessary state to DRAM, flushes the caches, and unmasks certain interrupts so the chip can be woken up. The details of this sequence are TBD .

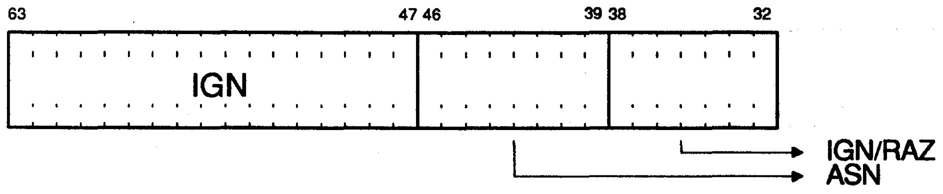
5.2.20 Ibox Process Context IPR (PCTX)

This register contains information associated with the context of a process. Any combination of the bit fields within this register may be written with a single HW_MTPR instruction. When bits <7:6> of the IPR index field of a HW_MTPR instruction contain the value 01_2, this register is selected. Bits <4:0> of the IPR index indicate which bit fields are to be written. The correspondence between register fields and IPR index bits is:

IPR index bit	Register Field
0	ASN
1	ASTER
2	ASTRR
3	PPCE
4	FPE

A HW_MFPR from this register returns the values in all of its component bit fields.



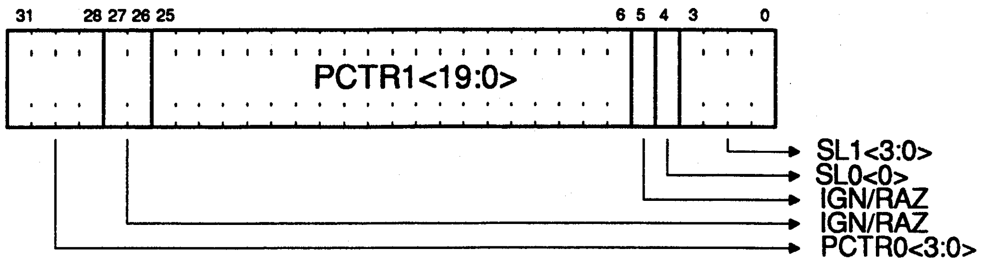


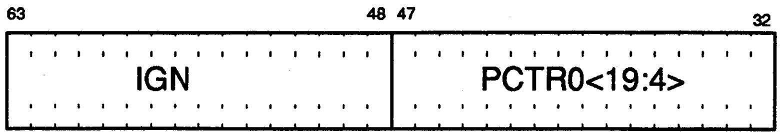
Name	Type	Description
ASN	RW	Address Space Number.
ASTER	RW	AST Enable Register - used to individually enable each of the four AST interrupt requests. The bit order with this field is: User Mode <11> Supervior Mode <10> Executive Mode <9> Kernel Mode <8>
ASTRR	RW	AST Request Register - used to request AST interrupts in each of the four processor modes. In order to generate a particular AST interrupt, its corresponding bits in ASTRR and ASTER must be set, along with the ASTE bit in IER. Further, the value of the current mode bits in the PS register must be equal to or higher than the value of the mode associated with the AST request. The bit order with this field is: User Mode <11> Supervior Mode <10> Executive Mode <9> Kernel Mode <8>
PPCE	RW	Process Performance Counter Enable. Both performance counters are

Name	Type	Description
		enabled if either this bit is set or the SPCE bit of the I_CTL register is set.
FPE	RW,0	Floating Point Enable - if clear, floating point instructions generate FEN exceptions.

5.2.21 PCTR_CTL

Performance counter control (PCTR_CTL) is a read/write register which controls the function of the performance counters.





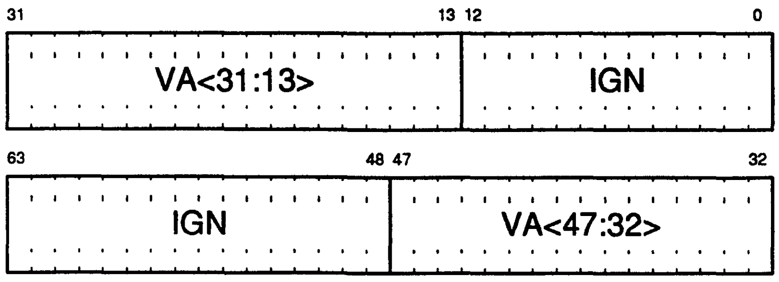
Name	Type	Description
SL1	RW	Select Input for Performance Counter #1 0000: Retired Instructions 0001: Retired Conditional Branches 0010: Retired Branch Mispredicts 0011: Retired ITB Misses 0100: Retired DTB Misses 0101: Retired Unaligned Traps 0110: Icache Misses 0111: MBOX Replay Traps 1000: Dcache Load Misses 1001: Dcache Misses 1010: Bcache Reads 1011: Bcache Writes 1100: SysPort Reads 1101: SysPort Writes 1110: 1111:
SL0	RW	Select Input for Performance Counter #1 0: Cycles 1: Retired instructions
PCTR1	RW	Performance Counter #1
PCTR0	RW	Performance Counter #0

5.3 Mbox IPRs

This section describes the IPRs which control Mbox functions.

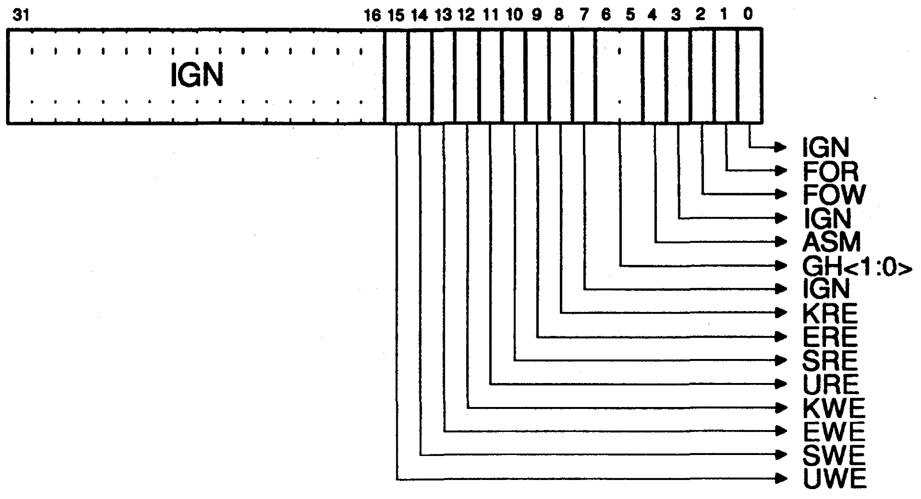
5.3.1 DTB_TAG0 & DTB_TAG1

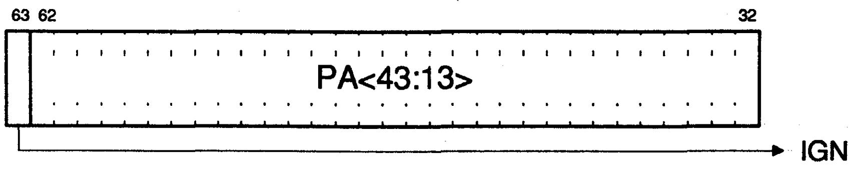
DTB_TAG0 and DTB_TAG1 are write-only registers through which the two memory pipe DTB tag arrays are written. Writes to DTB_TAG0 and DTG_TAG1 actually write registers outside the DTB arrays. When writes to the corresponding DTB_PTE registers are retired, the contents of both the DTB_TAG and DTB_PTE registers are written into their respective DTB arrays at locations determined by the round-robin allocation algorithm.



5.3.2 DTG_PTE0 & DTB_PTE1

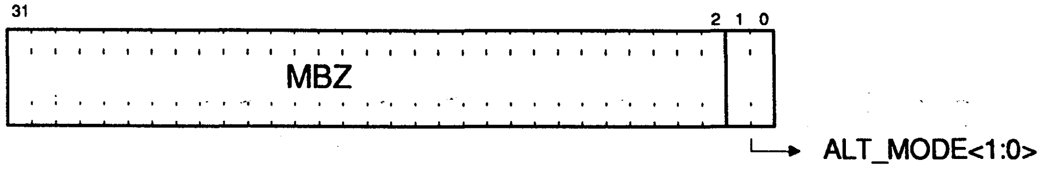
DTB_PTE0 and DTB_PTE1 are registers though which the DTB PTE arrays are written. The entries to be written are chosen by a round-robin allocation scheme. Writes to the DTB_PTE registers, when retired, result in both the DTB_TAG and DTB_PTE arrays being written.

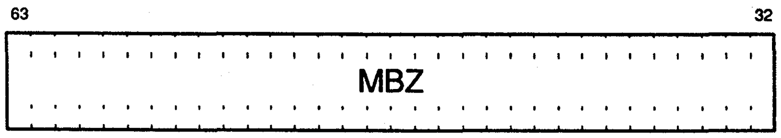




5.3.3 DTB_ALTMODE

DTB_ALTMODE is a write only register whose contents specify the alternate processor mode use by some HW_LD and HW_ST instructions.





Name	Type	Description	
ALT_MODE<1:0>	RW	Alt_Mode:	
		00	Kernel

Name	Type	Description	
		01	Executive
		10	Supervisor
		11	User

5.3.4 DTB_IAP

D-stream Translation Buffer Invalidate All Process (DTB_IAP) is a write-only pseudo register. Writes to this register invalidate all DTB entries in which the address space match (ASM) bit is clear.

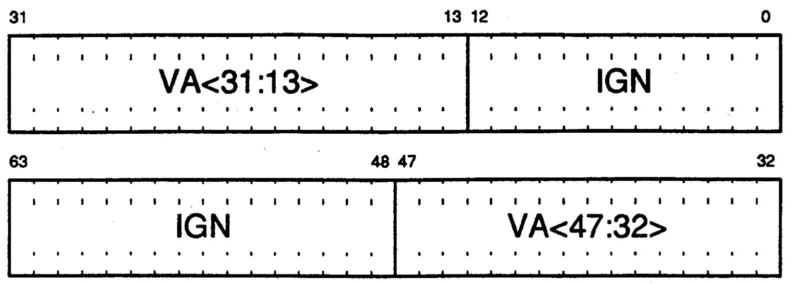
5.3.5 DTB_IA

D-stream Translation Buffer Invalidate All (DTB_IA) is a write-only pseudo register. Writes to this register invalidate all DTB entries and reset the DTB not-last-used pointer to its initial state.

5.3.6 DTB_IS0 & DTB_IS1

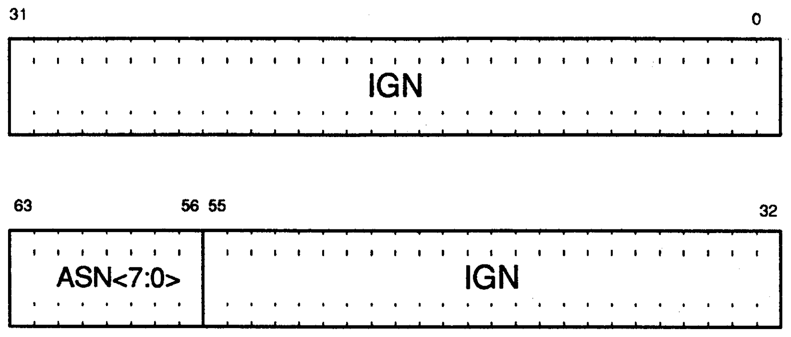
The D-stream Translation Buffer Invalidate Single registers (DTB_IS0 & DTB_IS1) are write-only pseudo registers through which software may invalidate a single entry in the DTB arrays. Writing a virtual page number to one of these registers invalidates any DTB entry in the corresponding memory pipeline which meets one of the following criteria:

- the DTB entry's virtual page number matches DTB_IS<47:13> and its ASN field matches DTB_ASN<63:56>
- the DTB entry's virtual page number matches DTBIS<47:13> and its ASM bit is set



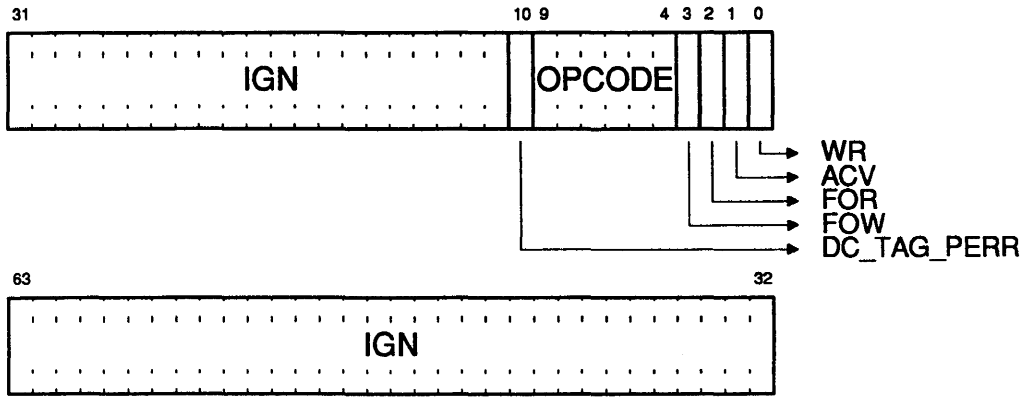
5.3.7 DTB_ASN0 & DTB_ASN1

The D-stream Translation Buffer Address Space Number registers (DTB_ASN0 & DTB_ASN1) are write-only registers which should be written with the address space number of the current process.



5.3.8 MM_STAT

MM_STAT is a read-only register. When a D-stream TB miss or fault occurs information about the error is latched in the MM_STAT register. MM_STAT is not locked by a LD_VPTE instruction.



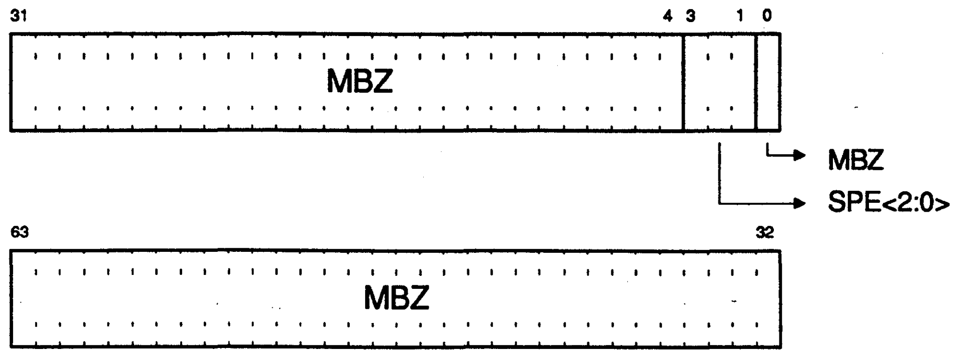
Name	Description
WR	Set if the reference which triggered the error was a write
ACV	Set if the reference caused an access violation. Includes bad virtual address.
FOR	Set if the reference was a read operation and the PTE FOR bit was set.
FOW	Set if the reference was a write operation and the PTE FOW bit was set.
OPCODE	Opcode of the instruction which triggered the error.
DC_TAG_PERR	Set to indicate that a Dcache tag parity error occurred during the initial tag probe of a load or store instruction. This error created a synchronous fault to the D_FAULT PALcode entry point, and is correctable. The virtual address associated with the error is available in the VA register.

Note:

The Ra field of the instruction which triggered the error can be obtained from the Ibox EXC_SUM register.

5.3.9 M_CTL

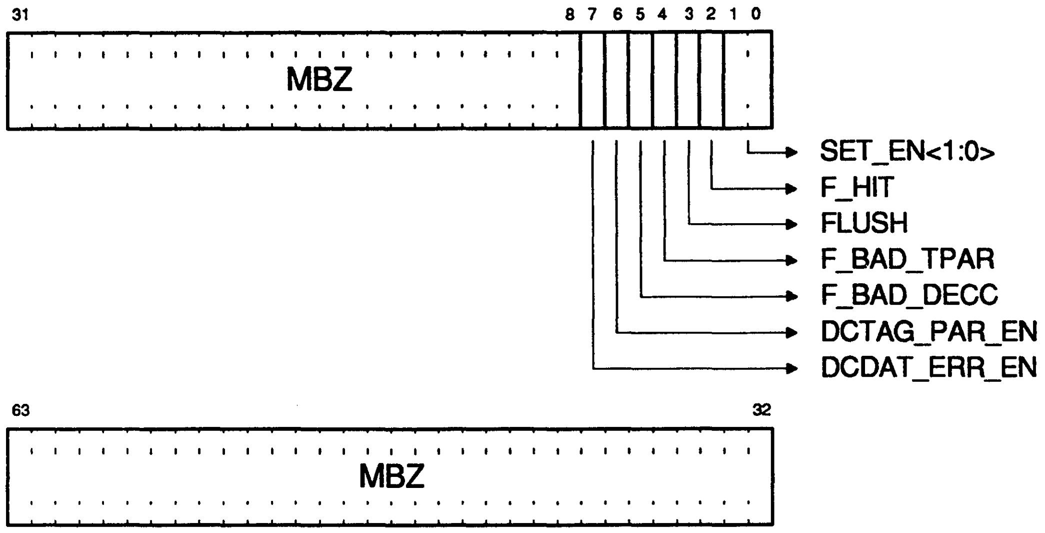
Mbox Control (M_CTL) is a write-only register, the contents of which are cleared by chip reset.



Name	Type	Description
SPE<2:0>	WO,0	Super Page mode enables. Only one (or none) may be set. SPE<2>, when set, enables super page mapping when VA<47:46> = 2. In this mode VA<43:13> are mapped directly to PA<43:13> and VA<45:44> are ignored. SPE<1>, when set, enables super page mapping when VA<47:41> = $7E_{16}$. In this mode VA<40:13> are mapped directly to PA<40:13> and PA<43:41> are copies of PA<40> (sign extension). SPE<0>, when set, enables super page mapping when VA<47:30> = $3FFFE_{16}$. In this mode VA<29:13> are mapped directly to PA<29:13> and PA<43:30> are cleared. Note: Super page accesses are only allowed in kernel mode. Non-kernel mode references to super pages result in access violations.

5.3.10 DC_CTL

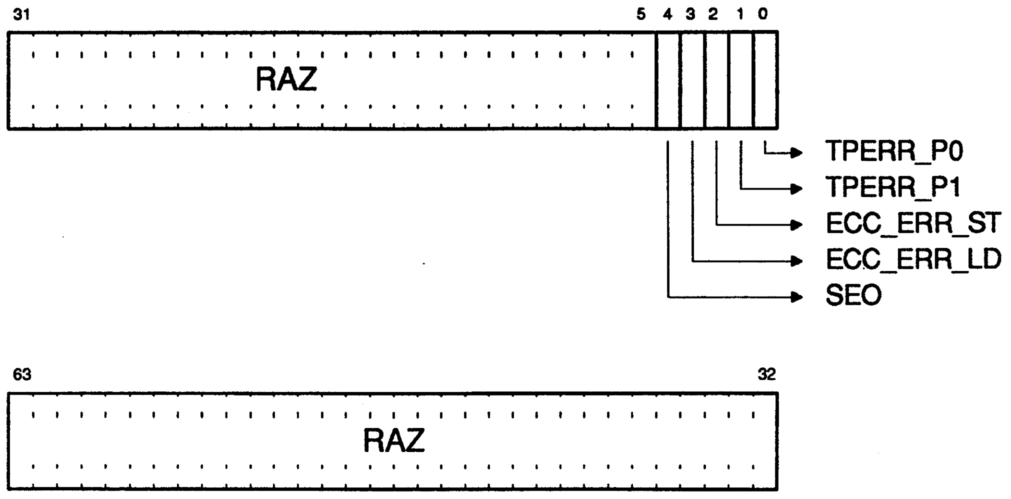
Dcache Control (DC_CTL) is a write-only register that controls Dcache activity. The contents of DC_STAT are initialized by chip reset as indicated.



Name	Type	Description
SET_EN<1:0>	W,3	Dcache Set Enable. At least one set must be enabled.
F_HIT	W,0	Force Hit. When set, this bit causes all memory space load and store instructions to hit in the Dcache, independent of the tag status bits. In this mode, only one of the two sets may be enabled, and tag parity checking must disabled (set DCTAG_PER_EN to zero).
FLUSH	W,0	When the value written into the DC_CTL register contains a one in this bit position all the Dcache tag valid bits are cleared.
F_BAD_TPAR	W,0	Force Bad Tag Parity. If set, this bit causes bad tag parity to be put into the Dcache tag array during Dcache fill operations. .
F_BAD_DECC	W,0	Force Bad Data ECC. If set, this bit ECC data to NOT be written into the cache along with the block that is loaded by a fill or store. This can be used to cause bad ECC to be present in the dcache by writing the same block with different data than is already present. Since the old ECC value will remain, it will be 'bad' relative to the new data.
DCTAG_PAR_EN	W,0	Dcache tag parity enable.
DCDAT_ERR_EN	W,0	Dcache data ecc and parity error enable

5.3.11 DC_STAT

Dcache Status (DC_STAT) is a read/write register. If a Dcache tag parity error or data ECC error occurs information about the error is latched in DC_STAT.

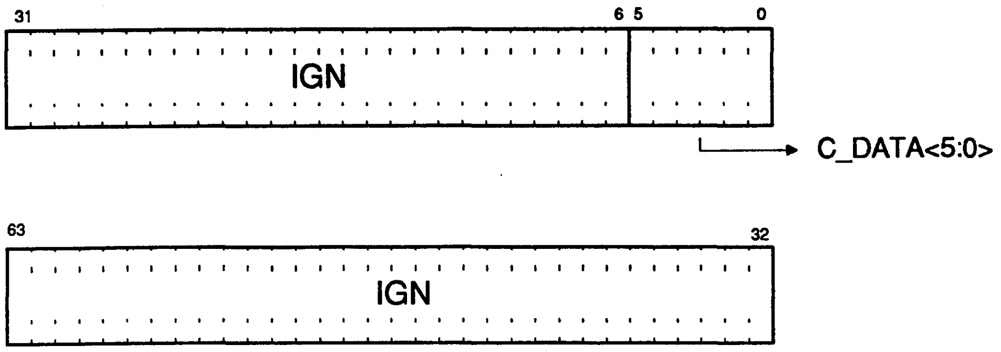


Name	Type	Description
TPERR_P0	R,W1C	Tag Parity Error - Pipe 0. When set, this bits indicate that a Dcache tag probe from pipe 0 resulted in a tag parity error. The error is uncorrectable and will result in a machine check.
TPERR_P1	R,W1C	Tag Parity Error - Pipe 1. When set, this bits indicate that a Dcache tag probe from pipe 1 resulted in a tag parity error. The error is uncorrectable and will result in a machine check.
ECC_ERR_ST	R,W1C	When set, this bit indicates that an ECC error occurred while processing a store.
ECC_ERR_LD	R,W1C	When set, this bit indicates that an ECC error occurred while processing a load (load data retrieved from dcache or bcache fill data).
SEO	R,W1C	Second Error Occurred. When set, this bit indicates that a tag parity or Store data ECC error occurred while the DC_STAT register was already locked; or that a Load data ECC error occurred while the DC_STAT register was already locked and error recovery was in progress.

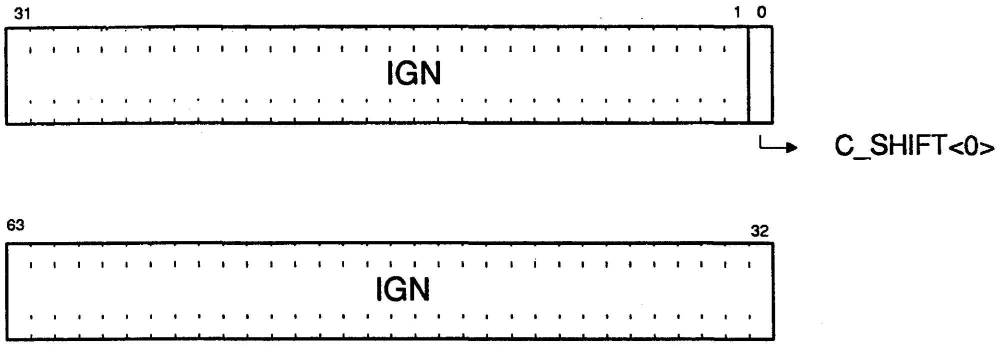
5.4 Cbox CSRs and IPRs

The CBOX Control/Status Registers (CSR's) are write-only registers which define system configuration, command processing, and timing parameters. They are written via a serial load/shift register. Six bits of data are written to CBOX_DATA IPR via a HW_MTPR instruction. When the instruction retires, the data in the register is shifted into the CBOX. The process is repeated until all CBOX data is shifted in.

The CBOX Internal Processor Registers (IPR's) are read-only registers which allow software access to system error information. They are read via a serial shift/read register. A SHIFT command is written to CBOX_SHIFT IPR. When the instruction retires, the CBOX_DATA IPR contains the first six bits of error information and can be read via a HW_MFPR instruction. The process is repeated until all CBOX data is shifted in.



Name	Type	Description
C_DATA<5:0>	RW	Cbox data register. Writes 6 bits of CSR data into serial shift register. When read (after C_SHIFT), allows access to 6 sequential bits of CBOX IPR data.



Name	Type	Description
C_SHIFT<0>	W1	When written (with a '1') causes 6 bits of CBOX IPR data to shift into CBOX_DATA register where it can be read by software (via a HW_MFPR instruction). All bits of the CBOX IPR data scan chain must be shifted.

5.4.1 CBOX CSR Description

Note that the precise order of these CSR's is TBD.

Name	Description
FRAME_SEL<2:0>	Sets ratio of framing clock to bit time. This in turn specifies the number of samples per framing clock. Allowed values: 0001: 1 (one sample per framing clock) 0010: 2 0100: 4

Name	Description
	1000: 8
VICTIM_THRESH<7:0>	Dcache victim threshold. Number of dcache read victims to allow to accumulate in VAF before writing them bcache. The number of victims is specified by a set bit in the vector. Allowed values: 00000001: 1 (write bcache on presence of one victim) 00000010: 2 (write bcache on presence of two victims) 00000100: 3 00001000: 4 00010000: 5 00100000: 6 01000000: 7 10000000: 8
BC_RDVICTIM<0>	When set, causes EV6 to abut victim writes with reads. Used for systems in which victim data and read data are on the same DRAM page.
SYSCLK_RATIO<15:0>	Ratio of CPU clock to SYSCLK. The final multiple of CPU clock period to SYSCLK period is calculated as 1 + (0.5 * SYSCLK_RATIO) Allowed values are: 0000000000000001: final multiple is 1.5 0000000000000010: final multiple is 2.0 0000000000000100: final multple is 2.5 0000000000001000: final multiple is 3 0000000000010000: final multiple is 3.5 0000000000100000: final multiple is 4.0
DUP_TAG_ENA<0>	When set, indicates to EV6 that external system has a duplicate tag
SET_DIRTY_ENA<2:0>	Enable sending set-dirty commands to system. Protocols supported: 000: EV6 sends no dirty commands off chip 001: EV6 sends clean-to-dirty commands 010: EV6 sends shared/clean 011: EV6 sends clean commands 100: EV6 sends shared/dirty commands 101: EV6 sends shared/dirty AND clean commands 110: EV6 sends all shared commands 111: EV6 sends all commands off chip
ZEROBLK_ENA<1:0>	Enable zero block processing and commands: ZEROBLK_ENA<1>: Enables zeroblk commands to system (Multiprocessor systems or duplicate tagged systems need to see zeroblk commands) ZEROBLK_ENA<0>: If set, enables EV6 processing zeroblk command as zero-block. If clear, EV6 converts zeroblk commands to read-modified commands.
SPEC_READ_ENA<0>	Enable speculative reads (read commands sent to system before bcache hit is known).
SYSBUS_FORMAT<0>	Format of physical address as it appears on system bus. Two allowed configurations: 0: Interleaved on bcache block boundries 1: Page mode-hit (refer to chapter 6)

Name	Description
SYSBUS_MB_ENA<0>	When set, sends memory barrier commands (MB) to system.
SYSBUS_ACK_LIMIT<4:0>	Encoded count of maximum number of outstanding commands the system can accept. Values are interpretted as: 00000: INF (system can accept an infinite number of outstanding commands) 00001: 1 (system can accept 1 outstanding command) 00010: 2 (system can accept 2 outstanding commands) ... 10110: 22 (system can accept 22 outstanding commands)
STIO_32_LIMIT<0>	If set, system is imposing a 32-byte limit for stores to IO space.
Bcache Port Control/Status	
BC_ENA<0>	If set, bcache is enabled
BC_CLEAN_VICTIM<0>	If set, causes EV6 to notify system that a clean block is being evicted. EV6 sends a CleanVictimBlk command along with the victim address.
BC_SIZE<3:0>	Encoded bcache size. Allowed values are: 0000: 1 MB 0001: 2 MB 0011: 4 MB 0111: 8 MB 1111: 16 MB
BC_RD_RD_BBL<1:0>	Number of CPU cycles to insert between reads to different SRAM banks. If the bcache is built as one bank, the value should be zero. Values are interpretted as: 00: 0 (no bubble cycles between reads to different SRAM banks) 01: 1 (one bubble cycle) 10: 2 (two bubble cycles) 11: 3 (three bubble cycles)
BC_RD_CLK_RATIO <15:0>	Ratio of bcache clock period to CPU clock period. The final multiple is computed as: $1 + (0.5 * N)$ where N is encoded in BC_RD_CLK_RATIO as one of the following allowed values: 0000000000000001: final multiple is 1.5 0000000000000010: final multiple is 2 0000000000000100: final multiple is 2.5 0000000000001000: final multiple is 3 0000000000010000: final multiple is 3.5 0000000000100000: final multiple is 4
BC_RD_WR_BBL<5:0>	Encoded number of bcache clock cycles between a bcache read and write. Allowed values are: 00000: zero clock cycles 01111: fifteen clock cycles
BC_LATE_WR_BC<2:0>	For *Late Write* synchronous SRAMs. The following three IPRs encode the total delay for which bcache data is delayed from bcache address. The total delay is calculated as the sum of the specified number of Bcache and CPU Clock cycles plus the CPU clock phase offset. LATE_WR_BC encodes the number of Bcache clock cycles to delay the write data. (000 = 0 cycles; 111 = 7 cycles)

Name	Description
BC_LATE_WR_CPU<1:0>	Encoded number of CPU Clock cycles to delay the write data (see above). (00 = 0 cycles; 11 = 3 cycles)
BC_LATE_WR_PHASE<0>	When set, delays write data by one CPU clock phase (see above).
BC_BURST_MODE_ENA<0>	When set, enables bcache burst mode.
Internal Cbox CSRs	
BC_RDCLK_VECTOR<15:0>	Vector describing bcache read clocks 1 bit per phase, 50% duty cycle 111000 for 1.5
MBZ<4:0>	Write zeros to last group to extend shift chain to a multiple of six

5.4.2 CBOX IPR Description

The CBOX IPR's are read 6 bits at a time: ERR_ADDR<43:38> comprises the first read-group; ERR_ADDR<43> is read in CBOX_DATA<5>.

Name	Description
ERR_ADDR<43:6>	Address of last reported ECC or parity error
ERR_CODE<2:0>	Summary of where error was detected: 000: No error 001: Bcache tag parity error 010: Triplicate tag parity error 011: Memory data ECC error 100: Bcache data ECC error 101: Dcache data ECC error
ECC_SYNDROME<7:0>	Syndrome of last reported ECC error
RAZ<4:0>	Padded zero's to extend shift chain to a multiple of 6

6. IEEE Floating Point Conformance

EV6 supports the IEEE floating-point operations defined in the Version 6 of the Alpha SRM. Support for a complete implementation of the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Standard 754-1985) is provided by a combination of hardware and software. EV6 provides several hardware features to facilitate complete support of the IEEE standard. These features are outlined in this section.

- EV6 implements precise exception handling in hardware.
- EV6 accepts both Signaling and Quiet NaNs as input operands and propagates them as specified by the Alpha Architecture. In addition, EV6 delivers a canonical Quiet NaN when an operation is required to produce a NaN value and none of its inputs are NaNs. Encodings for Signaling NaN and Quiet NaN are defined by the Alpha SRM, version 6.
- EV6 accepts infinity operands and implements infinity arithmetic as defined by the IEEE standard.
- EV6 implements SQRT for single (SQRTS) and double (SQRTT) precision in hardware.
- Denormal input operands produce an unmaskable Denorm Trap when used with arithmetic operations. CPYSE/CPYSN, FCMOVxx, and MF_FPCR/MT_FPCR are not arithmetic operations, and will pass Denormal values without initiating arithmetic traps
- EV6 implements the following disable bits in the Floating-Point Control Register (FPCR):
  - ⇒ Underflow Disable (UNFD)
  - ⇒ Overflow Disable (OVFD)
  - ⇒ Inexact Result Disable (INED)
  - ⇒ Division by Zero Disable (DZED)
  - ⇒ Invalid Operation Disable (INVD)

  If one of these bits is set and an instruction with the /S qualifier set generates the associated trapping result, EV6 produces the IEEE nontrapping result and supresses the trap. These nontrapping responses include correctly signed infinity, largest finite number, and Quiet NaNs as specified by the IEEE standard. EV6 will not produce a Denorm result for the underflow exception. Instead, a true zero (+0) is written to the destination register. In EV6 the FPCR Underflow to Zero (UNDZ) bit must be set if Underflow Disable (UNFD) bit is set. If desired, trapping on Underflow can be enabled by the instruction and the FPCR, and software may compute the Denorm value as defined in the IEEE Standard.

EV6 records floating-point exception information in two places:

- The FPCR status bits record the occurance of all exceptions that are detected whether or not the corresponding trap is enabled. The status bits are cleared only through a explicit clear command (MT_FPCR), hence the exception information they record is a summary of all exceptions that have occurred since the last time they were cleared.
- If an exception is detected and the corresponding trap is enabled by the instruction, and is not disabled by the FPCR control bits, EV6 will record the condition in the EXC_SUM register and initiate an arithmetic trap.

The following tables list all exceptional inputs and output conditions recognized by EV6, the result and exception generated for each condition. Notes:

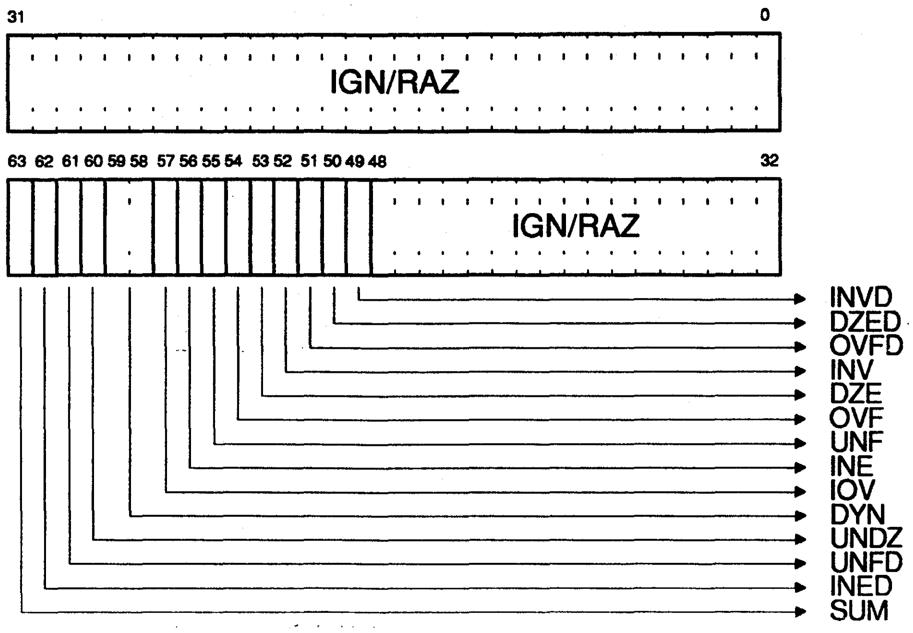
- EV6 will always trap on a Denormal input operand for all arithmetic operations.
- Input operand traps take precedence over arithmetic result traps.
- Abbreviations used in table:

⇒	Inf:	Infinity
⇒	QNaN:	Quiet NaN
⇒	SNaN:	Signalling NaN
⇒	CQNaN:	Canonical Quiet NaN

Alpha AXP Instructions	EV6 Hardware Supplied Result	Exception
ADDx SUBx INPUT		
Inf operand	+/-Inf	(none)
QNaN operand	QNaN	(none)
SNaN operand	QNaN	Invalid Op
Effective subtract of two Inf operands	CQNaN	Invalid Op
ADDx SUBx OUTPUT		
Exponent overflow	+/-Inf or +/-MAX	Overflow
Exponent underflow	+0	Underflow
Inexact result	Result	Inexact
MULx INPUT		
Inf operand	+/-Inf	(none)
QNaN operand	QNaN	(none)
SNaN operand	QNaN	Invalid Op
0 * Inf	CQNaN	Invalid Op
MULx OUTPUT		
(same as ADDx)		
DIVx INPUT		
QNaN operand	QNaN	(none)
SNaN operand	QNaN	Invalid Op
0/0 or Inf/Inf	CQNaN	Invalid Op
A/0 (A not 0)	+/-Inf	Div Zero
A/Inf	+/-0	(none)
Inf/A	+/-Inf	(none)
DIVx OUTPUT		
(same as ADDx)		
SQRTx INPUT		
+Inf operand	+Inf	(none)
QNaN operand	QNaN	(none)
SNaN operand	QNaN	Invalid Op
-A (A not 0)	CQNaN	Invalid Op
-0	-0	(none)
SQRTx OUTPUT		
Inexact result	root	Inexact
CMPTEQ CMPTUN INPUT		
Inf operand	True or False	(none)
QNaN operand	False for EQ, True for UN	(none)
SNaN operand	False for EQ,True for UN	Invalid Op
CMPTLT CMPTLE INPUT		
Inf operand	True or False	(none)
QNaN operand	False	Invalid Op
SNaN operand	False	Invalid Op
CVTfi INPUT		
Inf operand	0	Invalid Op
QNaN operand	0	Invalid Op
SNaN operand	0	Invalid Op

Alpha AXP Instructions	EV6 Hardware Supplied Result	Exception
CVTfi OUTPUT		
Inexact result	Result	Inexact
Integer overflow	Truncated result	Invalid Op
CVTif OUTPUT		
Inexact result	Result	Inexact
CVTff INPUT		
Inf operand	+/-Inf	(none)
QNaN operand	QNaN	(none)
SNaN operand	QNaN	Invalid Op
CVTff OUTPUT		
(same as ADDx)		
FBEQ FBNE FBLT FBLE FBGT FBGE LDS LDT STS STT CPYS CPYSN FCMOVx		

6.1 Floating Point Control Register (FPCR)



Name	Type	Description
SUM	RW	Summary bit. Records bit-wise OR of FPCR exception bits.
INED	RW	Inexact Disable. If this bit is set and a floating point instruction which enables trapping on inexact results generates an inexact value, the result is placed in the destination register and the trap is suppressed.
UNFD	RW	Underflow Disable. If UNFD and UNZD are set and a floating point instruction which enables trapping on underflow and which has the software completion qualifier set generates an underflow, then the trap is suppressed.
UNZD	RW	Underflow to zero. The Alpha architecture specifies that if this bit is set along with UNFD, then on underflow implementations place an appropriately signed zero value in the destination register rather than the denormal number specified by the IEEE standard, if they are capable of doing so. EV6 is not capable of generating IEEE compliant denormal results and always generates a positive zero (+0.0) on underflow. Hence this bit is only used along with UNFD to determine whether to suppress underflow traps.
DYN	RW	Dynamic rounding mode. Indicates the rounding mode to be used by an IEEE floating point instruction when the instruction specifies dynamic rounding mode: 00_2 Chopped 01_2 Minus infinity 10_2 Normal 11_2 Plus infinity
IOV	RW	Integer overflow. An integer arithmetic operation or a conversion from floating to integer overflowed the destination precision.

Name	Type	Description
INE	RW	Inexact result. A floating arithmetic or conversion operation gave a result that differed from the mathematically exact result.
UNF	RW	Underflow. A floating arithmetic or conversion operation gave a result that underflowed the destination exponent.
OVF	RW	Overflow. A floating arithmetic or conversion operation gave a result that overflowed the destination exponent.
DZE	RW	Divide by zero. An attempt was made to perform a floating divide with a divisor of zero.
INV	RW	Invalid operation. An attempt was made to perform a floating arithmetic operation and one or more of its operand values were illegal.
OVFD	RW	Overflow disable. If this bit is set and a floating arithmetic operation generates an overflow condition, then the appropriate IEEE non-trapping result is placed in the destination register and the trap is suppressed.
DZED	RW	Division by zero disable. If this bit is set and a floating divide by zero is detected, the appropriate IEEE non-trapping result is placed in the destination register and the trap is suppressed.
INVD	RW	Invalid operation disable. If this bit is set and a floating operate generates an invalid operation condition and EV6 is capable of producing the correct IEEE nontrapping result, that result is placed in the destination register and the trap is suppressed.

7. Error Detection and Handling

This section gives an overview of EV6's error detection and error handling mechanisms.

- The system port data bus is quadword ECC protected.
- The Bcache tag is parity protected.
- The Bcache data bus is quadword ECC protected.
- The Dcache tag array is parity protected.
- The Dcache data array is quadword ECC protected, however this mode of operation is only supported in systems in which ECC is enabled on both the system and Bcache ports.
- The Icache tag array is parity protected.
- The Icache data array is parity protected.
- The Dcache duplicate tag array is ECC protected.

The EV6 ECC implementation detects and corrects single bit errors in hardware. Multiple bit errors within a quadword are not detected.

7.1 Icache Data or Tag Parity Error

- The hardware detects the error, replay-traps the instructions which were fetched under the error, and flushes the entire icache so the re-fetched instructions are not sourced directly from the icache.
- I_STAT<TPE> or <DPE> is set.
- A CRD interrupt is posted.

7.2 Dcache Tag Parity Error

The primary copies of the Dcache tags are only used when servicing CPU-generated loads and stores, hence a Dcache tag parity error is processed as a fault.

- Machine check occurs before any machine state is changed.
- EXC_ADDR contains the PC of the load or store instruction which triggered the error.
- The TPERR_P0 and TPERR_P1 fields of the DC_STAT register are written to indicate the source of the error.
- The virtual address associated with the error is available in the VA register.
- Recovery: flush the errored block using the EDCB (Evict Data Cache Block) instruction. The on-chip duplicate tag provides the correct victim address and cache state.

7.3 Dcache Data Correctable ECC Error

The actions which may invoke Dcache data ECC errors are:

- Load instructions
- Stores of less than quadword length
- Dcache Victim Reads

The hardware flow used for Dcache data ECC errors depends the action which triggered the error.

7.3.1 Load Instruction

Load instructions only trigger Dcache ECC errors if they use the data, i.e. if they hit in the Dcache. Loads which read their data from the Dcache may do so either in the same cycle as the Dcache tag probe (typical case) or in some subsequent cycle (load-queue retry). The hardware flows for these two error cases differ slightly.

If an ECC error occurs when a load reads the Dcache data array in the same cycle as the tag array, then the Ibox stops retiring instructions before the offending load retires, and does not start retiring again until after hardware recovers from the error.

If an ECC error occurs when a load reads the Dcache data array after it read the Dcache tag array, then the load may already have retired.

In either case:

- The load's destination register is written with incorrect data, however the load queue will retain the state associated with the load instruction.
- A consumer of the load's data may issue before the error is recognized, however the Ibox will invoke a replay trap at an instruction which is older than (or equal to) any instruction which consumes the load's data, and then stalls the replayed I-stream in the map stage of the pipeline until the error is corrected.
- The Cbox scrubs the block in the Dcache, which it does by evicting the block into the victim buffer (thereby scrubbing it) and writing it back into the Dcache.
- The load queue retries the load and rewrites the register.
- A corrected read (CRD) interrupt is posted.
- DC_STAT register:
  - ⇒ DECC_ERR set
  - ⇒ DECC_COR set

7.3.2 Store Instruction (Less than Quadword Length)

A store of less than quadword length could invoke a Dcache ECC error since the original quadword must be read to calculate the new check bits.

- The Mbox scrubs the original quadword and replays the write.
- The Mbox posts a CRD interrupt:
- DC_STAT register:
  - ⇒ DECC_ERR set
  - ⇒ DECC_COR set

7.3.3 Victim Reads

- ECC-errored Dcache victims are scrubbed as they are written into the victim data buffer
- A CRD interrupt is posted.
- DC_STAT register:
  - ⇒ DECC_ERR set
  - ⇒ DECC_COR set

7.4 Dcache Triplicate Tag Parity Error

- Machine check
- C_STAT: TPERR is set.
- C_ADDR: contains bits <43:6> of the address associated with the error.

7.5 Bcache Tag Parity Error

- Machine check
- C_STAT: TPERR is set.
- C_ADDR: contains bits <43:6> of the address associated with the error.
- BC_TAG: contains the tag and tag control fields of the errored Bcache block.

- Bcache Tag Parity Errors are not recoverable.

7.6 Bcache Data Correctable ECC Error

The actions which may trigger Bcache data ECC errors are:

- Icache fill
- Dcache fill, data possibly used by load instruction.
- Victim read invoked by a system port probe or by the processor's own reference stream.

Independent of the action which triggered the error:

- A CRD interrupt is posted
- C_STAT: BC_ECC is set. ECC_CRD is set.
- C_ADDR: contains bits <43:6> of the address associated with the error.

The recovery mechanism depends on the action which triggered the error.

7.6.1 Icache Fill from Bcache

For an Icache fill, bad Icache data parity is generated for the octaword which contains the errored quadword.

- The hardware flushes the icache
- C_STAT: BC_ECC is set.
- A machine check is invoked.. The PAL machine check handler must scrub the block in the bcache.

7.6.2 Dcache Fill from Bcache

If the errored quadword is not used to satisfy a load instruction no hardware recovery flow is invoked - the errored quadword and its associated check bits are written into the Dcache

If the errored quadword is used to satisfy a load instruction then the flow is very similar to that used for a Dcache ECC error:

- The load's destination register is written with incorrect data, however the load queue will retain the state associated with the load instruction.
- A consumer of the load's data may issue before the error is recognized, however the Ibox will invoke a replay trap at an instruction which is older than (or equal to) any instruction which consumes the load's data, and then stalls the replayed I-stream in the map stage of the pipeline until the error is corrected.
- The Cbox scrubs the block in the Dcache, which it does by evicting the block into the victim buffer (thereby scrubbing it) and writing it back into the Dcache.
- The load queue retries the load and rewrites the register.

7.6.3 Victim Read

The errored quadword is written to the system port. It is not scrubbed.

7.7 Bcache Data Uncorrectable ECC Error

- Machine Check
- C_STAT: BC_ECC is set. ECC_CRD is clear.
- C_ADDR: contains bits <43:6> of the address associated with the error

7.8 Memory Data Correctable ECC Error

The actions which may trigger memory data ECC errors are:

- Icache fill
- Dcache fill, data possibly used by load instruction.

Independent of the action which triggered the error:

- A CRD interrupt is posted
- C_STAT: MEM_ECC is set. ECC_CRD is set.
- C_ADDR: contains bits <43:6> of the address associated with the error.

The recovery mechanism depends on the action which triggered the error.

7.8.1 Icache Fill from Memory

For an Icache fill, bad Icache data parity is generated for the octaword which contains the errored quadword.

- The hardware flushes the icache
- C_STAT: MEM_ECC is set.
- A machine check is invoked.. The PAL machine check handler must scrub the block in the bcache and memory.

7.8.2 Dcache Fill from Memory

If the errored quadword is not used to satisfy a load instruction no hardware recovery flow is invoked - the errored quadword and its associated check bits are written into the Dcache and later written into the Bcache.

If the errored quadword is used to satisfy a load instruction then the flow is very similar to that used for a Dcache ECC error:

- The load's destination register is written with incorrect data, however the load queue will retain the state associated with the load instruction.
- A consumer of the load's data may issue before the error is recognized, however the Ibox will invoke a replay trap at an instruction which is older than (or equal to) any instruction which consumes the load's data, and then stalls the replayed I-stream in the map stage of the pipeline until the error is corrected.
- The Cbox scrubs the block in the Dcache, which it does by evicting the block into the victim buffer (thereby scrubbing it) and writing it back into the Dcache.
- The load queue retries the load and rewrites the register.

7.9 Memory Data Uncorrectable ECC Error

- Machine Check
- C_STAT: BC_ECC is set. ECC_CRD is clear.
- C_ADDR: contains bits <43:6> of the address associated with the error

7.10 System Port Read Errors

- Machine Check
- C_STAT: SRDERR set.
- C_ADDR: contains bits <43:6> of the address assoiciated with the error.

8. Initialization and Test

**To be specified. Here are a few key points:

- The SROM port will work just like EV4 and EV5.
- The SROM pins can do double-duty as a software-controlled UART, just like EV4 and EV5.
- Unlike EV4 and EV5, systems are required to have an SROM - that will be the only way to configure the system port.
- There will be an IEEE 1149.1 compliant test access port.
- There will be Built-in-Self-Test (BIST) of all major storage arrays, and Built-in-Self-Repair (BISR) of the Icache and Dcache tag and data arrays.

9. Electrical Data

This chapter describes the electrical characteristics of EV6 and its interface pins. It will contain electrical characteristics, DC characteristics, AC characterisitcs and power supply considerations.

9.1 Electrical Characteristics

The following table lists the maximum ratings for EV6.

Characteristics	**Ratings**
Storage Temperature	-55C to +125C
Junction Temperature	?C to 100C
Supply Voltage	VSS 0V, VDD 2.0 V
Input or Output applied	-0.5 to TBD V
Maximum Power @ VDD=?	TBD W typical
Frequency=TBD MHz	TBD W maximum

9.2 DC Characteristics

9.2.1 Power Supply

The VSS pins are connected to 0.0V, and the VDD pins are connected to 2.0V, +/- 100mV.

9.2.2 Input Signal Pins

Nearly all input signals are CMOS inputs with 2.0V levels. The one exception is CLK_IN_H/L.

9.2.3 Driven Signals From EV6

EV6 requires a floating well type driver on the Bcache I/O interface, due to Bcache configurations that may drive voltages in excess of a threshold voltage above the 2V VDD. All I/O cells will use the same floating well design, however the drive strengths will not be the same.

The output only cells will not use a floating well design, but, will use a simple push/pull circuit. More than one drive strength may be required for this pin category.

The SROM pins must be truly TTL compatible. This is achieved by employing open drain pulldown circuits. A resistor must be placed on the module to the 3.3V supply to pull the signal past the TTLVih point.

Some lines will be either series or parallel terminated. For the parallel terminated lines, the chip will provide good Voh and Vol margins while sourcing or sinking the termination current, as defined in the table below.

(TBD)

The following table show the drive specifications for each category of driver(typical-typical process/ Tj=85 degrees C/VDD=2.0V).

EV6 IO SPECIFICATIONS

Parameter	Units	Current	Notes
Voh_ca	VDD - 0.25V	45 mA	1
Vol_ca	0.25V	25 mA	1
Voh_cd	VDD - 0.25V	2 mA	2
Vol_cd	0.25V	2 mA	2
Vih	Vref +/- 250mV	TBD mA	3
Vil	Vref +/- 250mV	TBD mA	3
Vol_OD	0.25V	75 mA	4
Vih_HV	VDD/2 + 250mV	TBD mA	5
Vil_HV	VDD/2 - 250mV	TBD mA	5
Vol_OD 3V	0.4 V	4mA	6

1. Applies to cache address drivers
2. Applies to cache data drivers
3. Applies to all 2.5V tolerant inputs
4. Applies to 'OPEN DRAIN' type outputs,(assumes a TDB ohm resistor connected to the 3.3V supply on the module.
5. Applies to 3V tolerant pins.
6. Applies to 3V open drain outputs (assumes resistor to 3.3V supply).

9.3 AC Characteristics

This section describes the ac timing specifications for the 21164.

9.3.1 Clocking Scheme

The System port clocking scheme is described in detail in section 3.3.12. It requires a differential input clock and a system or framing clock. There is one signal that is considered synchronous while all other signals employ a clock forwarding scheme that is described in section 3.3.12. The system or FrameClk_H is used to establish a starting point for multi-cycle transfers of command and data to the system. It is also used to perform a synchronous clock forward reset of the interface.

9.3.2 Input Clocks

The differential input clock signals CLK_IN_H/L are frequencies ranging from 80 to 200 Mhz. Systems choose the appropriate frequency within that range which matches the requirement for their own clock distribution. This input clock is used to compare against a divided down copy of the VCO output for phase alignment. The GCLK is not the product of this oscillator input so there is no requirement for specialized circuitry to detect the presence of CLK_IN_H/L.

One additional input clock is a single ended square wave clock called the framing clock. This is expected to be a skewed controlled copy of the exact clock distributed throughout the system. The period of this clock can be identical to the osc_clk_in_h/l or an integer multiple of that signal and it should be phase aligned with the osc_clk_h/l with distribution skew not in excess of TDB psec.

It has a two functions. First, it is used to provide a known starting point for all clock forwarding transfer that emanate from EV6. Second, it is the clock used for the only synchronous signal in the interface, the clock forward reset.

9.3.2.1 Clock Termination and Impedance Levels

9.3.2.2 AC Coupling

9.3.3

9.3.4 1.3.3Analog PLL

************* PUT PLL SPEC HERE***********

9.3.5 Timing

9.3.5.1 Synchronous Signals

There is one synchronous output signal driven from EV6. That is, **ClkFwdReset_H**. All clock forward circuits are reset using this signal. It is operationally specified in section 3.3.11.11. This signal is clocked by a copy of the Framing clock described above and is not derived from GCLK off the internal PLL. Therefore, there it does not have a skew component caused by the drift in the PLL. The timing specification is the delay from the input of the framing clock to the output pad driver and that is TBD min and TBD max.

9.3.5.2 Asynchronous Signals

The following is a list of asynchronous input signals:

IRQ<5:0> SROMDATA DC_OK_H RESET

9.3.5.3 Clock Forwarded Signals For System Interface

Clock forwarding is described in detail in section 3.3.11.4. The following is a list of input only signals that are accompanied by a clock and are open drain:

SysAddIn<14:0> SysFillValid SysDataInValid SysDataOutValid

Setup and Hold With Respect To SysAddInClkH			
	450 MHz	500MHz	600MHz
min Setup	-200 psec	-300 psec	-400 psec
min Hold	1005 psec	1005 psec	1005 psec

9.3.5.3.1

The following is a list of output signals that are accompanied by a clock and are open drain:

SysAddOut<14:0> SysAddOutClkH

Timing difference across all signals including clock			
	450 MHz	500MHz	600MHz
max output skew	560 psec	560 psec	560 psec

The following is a list of bi-directional signals that are accompanied by a clock, when Ev6 is driving the data bus there is one clock for 18 bits (ie SysClkOut<3> is associated with SysData<63:48> and SysCheck<7:6>, when EV6 is receiving there is on clock for every 9 bits. The drivers are open drain:

SysData<63:56>	**SysCheck<7>**	**SysDataInClk<7>**	**SysDataOutClk<3>**
SysData<55:48>	**SysCheck<6>**	**SysDataInClk<6>**	**SysDataOutClk<3>**
SysData<47:40>	**SysCheck<5>**	**SysDataInClk<5>**	**SysDataOutClk<2>**
SysData<39:32>	**SysCheck<4>**	**SysDataInClk<4>**	**SysDataOutClk<2>**
SysData<31:24>	**SysCheck<3>**	**SysDataInClk<3>**	**SysDataOutClk<1>**
SysData<23:16>	**SysCheck<2>**	**SysDataInClk<2>**	**SysDataOutClk<1>**
SysData<15:8>	**SysCheck<1>**	**SysDataInClk<1>**	**SysDataOutClk<0>**
SysData<7:0>	**SysCheck<0>**	**SysDataInClk<0>**	**SysDataOutClk<0>**

Incoming direction with respect to SysDataInClk			
	450 MHz	500MHz	600MHz
min Setup	-200 psec	-300 psec	-400 psec
min Hold	1005 psec	1005 psec	1005 psec

Timing difference across outputs including SysDataOutClk			
	450 MHz	500MHz	600MHz
max output skew	560 psec	560 psec	560 psec

9.3.5.4 Bcache Timing

The Bcache is entirely private to the EV6 pinbus. Address and control are directly driven from EV6 along with multiple differential clocks. There is internal adjustment which delay the clock relative to the address and control. All signals to the synchronous SRAM devices are directly driven to the device. Data is bidirectional. For writing, data is clocked in the SRAM by the same clock used for address and control. For reading, there are two styles of data delivery to EV6. First, the conventional REG/REG component drives data on the rising edge of its received clock and EV6 uses a copy of this same clock to capture this data at the pads. The second type of data from the SRAM is clock forwarded from the device with data supplied on the rising and falling edge of the clock.

The following signals are driven from EV6 to the devices and must meet the setup and hold constraints of the receiving device:

BcAddress<23:4> **BcDataOE_L** **BcLoad_L** **BcDataWr_L**

BcTagOE_L   BcTagWr_L

Clocks: BcDataOutClk<3:0>H/L

The following signals are Bidirectional. When EV6 is driving, these signals must meet the setup and hold constraints of the receiving device:

BcData<127:0>   BcCheck<7:0>

Clocks: BcDataInClk<7:0>H/L

Incoming direction with respect to BcDataInClk			
	450 MHz	500MHz	600MHz
min Setup	-200 psec	-300 psec	-400 psec
min Hold	1005 psec	1005 psec	1005 psec

BcTag<42:20> BcTagValid_H BcTagDirty_H BcTagShared_H   BcTagParity_H
Clocks: BcTagInClkH/L

Incoming direction with respect to BcTagInClk			
	450 MHz	500MHz	600MHz
min Setup	-200 psec	-300 psec	-400 psec
min Hold	1005 psec	1005 psec	1005 psec

9.4 Power Supply Considerations

9.4.1 Decoupling

9.4.2 Power Supply Sequencing

10. Packaging Information

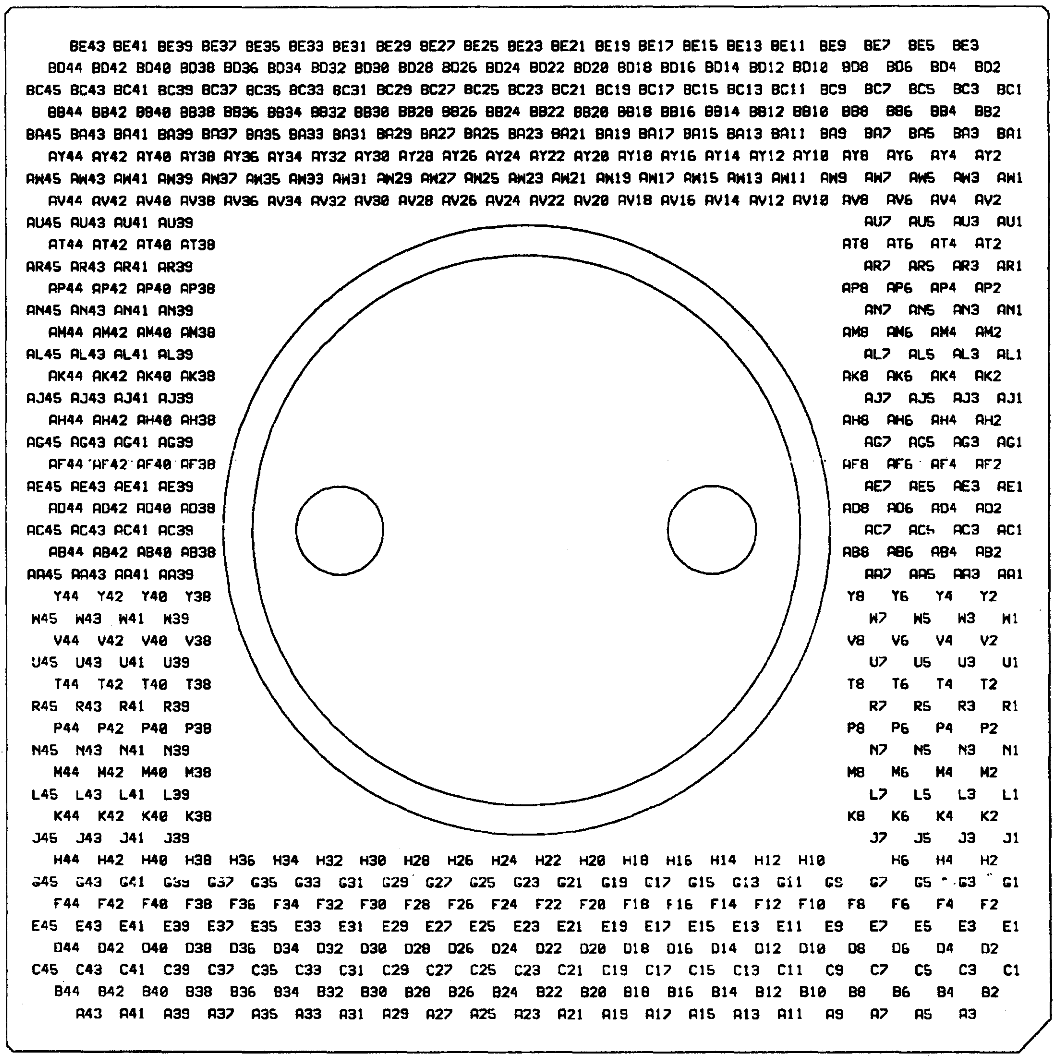
10.1 Introduction

This chapter provides detailed information on the chip package and the complete pinout for the 587 pin ceramic PGA for EV6.

10.2 Package Information

The following figure shows the pin location and the package dimensions.

10.3 EV6 Pinout


BE43 BE41 BE39 BE37 BE35 BE33 BE31 BE29 BE27 BE25 BE23 BE21 BE19 BE17 BE15 BE13 BE11 BE9 BE7 BE5 BE3
BD44 BD42 BD40 BD38 BD36 BD34 BD32 BD30 BD28 BD26 BD24 BD22 BD20 BD18 BD16 BD14 BD12 BD10 BD8 BD6 BD4 BD2
BC45 BC43 BC41 BC39 BC37 BC35 BC33 BC31 BC29 BC27 BC25 BC23 BC21 BC19 BC17 BC15 BC13 BC11 BC9 BC7 BC5 BC3 BC1
BB44 BB42 BB40 BB38 BB36 BB34 BB32 BB30 BB28 BB26 BB24 BB22 BB20 BB18 BB16 BB14 BB12 BB10 BB8 BB6 BB4 BB2
BA45 BA43 BA41 BA39 BA37 BA35 BA33 BA31 BA29 BA27 BA25 BA23 BA21 BA19 BA17 BA15 BA13 BA11 BA9 BA7 BA5 BA3 BA1
AY44 AY42 AY40 AY38 AY36 AY34 AY32 AY30 AY28 AY26 AY24 AY22 AY20 AY18 AY16 AY14 AY12 AY10 AY8 AY6 AY4 AY2
AW45 AW43 AW41 AW39 AW37 AW35 AW33 AW31 AW29 AW27 AW25 AW23 AW21 AW19 AW17 AW15 AW13 AW11 AW9 AW7 AW5 AW3 AW1
AV44 AV42 AV40 AV38 AV36 AV34 AV32 AV30 AV28 AV26 AV24 AV22 AV20 AV18 AV16 AV14 AV12 AV10 AV8 AV6 AV4 AV2
AU45 AU43 AU41 AU39 AU7 AU5 AU3 AU1
AT44 AT42 AT40 AT38 AT8 AT6 AT4 AT2
AR45 AR43 AR41 AR39 AR7 AR5 AR3 AR1
AP44 AP42 AP40 AP38 AP8 AP6 AP4 AP2
AN45 AN43 AN41 AN39 AN7 AN5 AN3 AN1
AM44 AM42 AM40 AM38 AM8 AM6 AM4 AM2
AL45 AL43 AL41 AL39 AL7 AL5 AL3 AL1
AK44 AK42 AK40 AK38 AK8 AK6 AK4 AK2
AJ45 AJ43 AJ41 AJ39 AJ7 AJ5 AJ3 AJ1
AH44 AH42 AH40 AH38 AH8 AH6 AH4 AH2
AG45 AG43 AG41 AG39 AG7 AG5 AG3 AG1
AF44 AF42 AF40 AF38 AF8 AF6 AF4 AF2
AE45 AE43 AE41 AE39 AE7 AE5 AE3 AE1
AD44 AD42 AD40 AD38 AD8 AD6 AD4 AD2
AC45 AC43 AC41 AC39 AC7 AC5 AC3 AC1
AB44 AB42 AB40 AB38 AB8 AB6 AB4 AB2
AA45 AA43 AA41 AA39 AA7 AA5 AA3 AA1
Y44 Y42 Y40 Y38 Y8 Y6 Y4 Y2
W45 W43 W41 W39 W7 W5 W3 W1
V44 V42 V40 V38 V8 V6 V4 V2
U45 U43 U41 U39 U7 U5 U3 U1
T44 T42 T40 T38 T8 T6 T4 T2
R45 R43 R41 R39 R7 R5 R3 R1
P44 P42 P40 P38 P8 P6 P4 P2
N45 N43 N41 N39 N7 N5 N3 N1
M44 M42 M40 M38 M8 M6 M4 M2
L45 L43 L41 L39 L7 L5 L3 L1
K44 K42 K40 K38 K8 K6 K4 K2
J45 J43 J41 J39 J7 J5 J3 J1
H44 H42 H40 H38 H36 H34 H32 H30 H28 H26 H24 H22 H20 H18 H16 H14 H12 H10 H6 H4 H2
G45 G43 G41 G39 G37 G35 G33 G31 G29 G27 G25 G23 G21 G19 G17 G15 G13 G11 G9 G7 G5 G3 G1
F44 F42 F40 F38 F36 F34 F32 F30 F28 F26 F24 F22 F20 F18 F16 F14 F12 F10 F8 F6 F4 F2
E45 E43 E41 E39 E37 E35 E33 E31 E29 E27 E25 E23 E21 E19 E17 E15 E13 E11 E9 E7 E5 E3 E1
D44 D42 D40 D38 D36 D34 D32 D30 D28 D26 D24 D22 D20 D18 D16 D14 D12 D10 D8 D6 D4 D2
C45 C43 C41 C39 C37 C35 C33 C31 C29 C27 C25 C23 C21 C19 C17 C15 C13 C11 C9 C7 C5 C3 C1
B44 B42 B40 B38 B36 B34 B32 B30 B28 B26 B24 B22 B20 B18 B16 B14 B12 B10 B8 B6 B4 B2
A43 A41 A39 A37 A35 A33 A31 A29 A27 A25 A23 A21 A19 A17 A15 A13 A11 A9 A7 A5 A3

Signal Pin

Signal Name		Pin Number	Type
BCADDRESS_4		D26	OUT
BCADDRESS_5		E27	OUT
BCADDRESS_6		C27	OUT
BCADDRESS_7		B28	OUT
BCADDRESS_8		D28	OUT
BCADDRESS_9		H28	OUT
BCADDRESS_10		G29	OUT
BCADDRESS_11		C29	OUT
BCADDRESS_12		A29	OUT
BCADDRESS_13		B30	OUT
BCADDRESS_14		A31	OUT
BCADDRESS_15		F30	OUT
BCADDRESS_16		H30	OUT
BCADDRESS_17		E31	OUT
BCADDRESS_18		G31	OUT
BCADDRESS_19		D32	OUT
BCADDRESS_20		F32	OUT
BCADDRESS_21		E33	OUT
BCADDRESS_22		C33	OUT
BCADDRESS_23		B34	OUT
BCCHECK_0		BC7	BI
BCCHECK_1		AV12	BI
BCCHECK_2		BC11	BI
BCCHECK_3		AY14	BI
BCCHECK_4		AY38	BI
BCCHECK_5		BE41	BI
BCCHECK_6		BB38	BI
BCCHECK_7		AW35	BI
BCCHECK_8		BB8	BI
BCCHECK_9		BE9	BI
BCCHECK_10		BB12	BI
BCCHECK_11		AW15	BI
BCCHECK_12		AW37	BI
BCCHECK_13		BD40	BI
BCCHECK_14		BA37	BI
BCCHECK_15		AV34	BI
BCDATAINCLK_0_	H	F8	IN
BCDATAINCLK_0_	L	E7	IN
BCDATAINCLK_1_	H	P4	IN
BCDATAINCLK_1_	L	R5	IN
BCDATAINCLK_2_	H	AH4	IN
BCDATAINCLK_2_	L	AJ3	IN
BCDATAINCLK_3_	H	AY8	IN
BCDATAINCLK_3_	L	AW9	IN
BCDATAINCLK_4_	H	E39	IN
BCDATAINCLK_4_	L	F38	IN
BCDATAINCLK_5_	H	R41	IN
BCDATAINCLK_5_	L	P42	IN
BCDATAINCLK_6_	H	AF40	IN
Signal Name		**Pin Number**	**Type**
BCDATAINCLK_6_	L	AG41	IN
BCDATAINCLK_7_	H	AV40	IN

BCDATAINCLK_7_	L	AW41	IN
BCDATAOE_L		E25	OUT
BCDATAOUTCLK_0	_H	J5	OUT
BCDATAOUTCLK_0	_L	K6	OUT
BCDATAOUTCLK_1	_H	AU5	OUT
BCDATAOUTCLK_1	_L	AV4	OUT
BCDATAOUTCLK_2	_H	J41	OUT
BCDATAOUTCLK_2	_L	K40	OUT
BCDATAOUTCLK_3	_H	AP42	OUT
BCDATAOUTCLK_3	_L	AR43	OUT
BCDATAWR_L		G25	OUT
BCDATA_0		E13	BI
BCDATA_1		B10	BI
BCDATA_2		G11	BI
BCDATA_3		D8	BI
BCDATA_4		C5	BI
BCDATA_5		K8	BI
BCDATA_6		E3	BI
BCDATA_7		M8	BI
BCDATA_8		G3	BI
BCDATA_9		J1	BI
BCDATA_10		P6	BI
BCDATA_11		L1	BI
BCDATA_12		T6	BI
BCDATA_13		R1	BI
BCDATA_14		U1	BI
BCDATA_15		Y6	BI
BCDATA_16		AA5	BI
BCDATA_17		AB6	BI
BCDATA_18		AC3	BI
BCDATA_19		AD8	BI
BCDATA_20		AE7	BI
BCDATA_21		AG3	BI
BCDATA_22		AG5	BI
BCDATA_23		AH8	BI
BCDATA_24		AN3	BI
BCDATA_25		AK8	BI
BCDATA_26		AR3	BI
BCDATA_27		AW3	BI
BCDATA_28		BA1	BI
BCDATA_29		BC1	BI
BCDATA_30		BC5	BI
BCDATA_31		AV10	BI
BCDATA_32		D34	BI
BCDATA_33		B36	BI
BCDATA_34		G35	BI
BCDATA_35		D38	BI
BCDATA_36		C41	BI
BCDATA_37		E43	BI
BCDATA_38		H40	BI
BCDATA_39		E45	BI
Signal Name		**Pin Number**	**Type**
BCDATA_40		H42	BI
BCDATA_41		P40	BI
BCDATA_42		J45	BI
BCDATA_43		N43	BI
BCDATA_44		T40	BI
BCDATA_45		T44	BI
BCDATA_46		W41	BI

Signal Name	Pin Number	Type
BCDATA_47	V44	BI
BCDATA_48	AB38	BI
BCDATA_49	AB40	BI
BCDATA_50	AC43	BI
BCDATA_51	AD44	BI
BCDATA_52	AE45	BI
BCDATA_53	AH44	BI
BCDATA_54	AK44	BI
BCDATA_55	AK38	BI
BCDATA_56	AL39	BI
BCDATA_57	AN43	BI
BCDATA_58	AR45	BI
BCDATA_59	AP38	BI
BCDATA_60	AW43	BI
BCDATA_61	AT38	BI
BCDATA_62	BA43	BI
BCDATA_63	BC41	BI
BCDATA_64	D12	BI
BCDATA_65	A9	BI
BCDATA_66	F10	BI
BCDATA_67	C7	BI
BCDATA_68	D6	BI
BCDATA_69	G5	BI
BCDATA_70	D2	BI
BCDATA_71	L7	BI
BCDATA_72	F2	BI
BCDATA_73	K2	BI
BCDATA_74	R7	BI
BCDATA_75	M2	BI
BCDATA_76	U7	BI
BCDATA_77	T2	BI
BCDATA_78	V2	BI
BCDATA_79	Y4	BI
BCDATA_80	AA1	BI
BCDATA_81	AB8	BI
BCDATA_82	AD2	BI
BCDATA_83	AE1	BI
BCDATA_84	AF4	BI
BCDATA_85	AH2	BI
BCDATA_86	AK2	BI
BCDATA_87	AJ7	BI
BCDATA_88	AP2	BI
BCDATA_89	AL7	BI
BCDATA_90	AT2	BI
BCDATA_91	AY2	BI
BCDATA_92	BA3	BI
Signal Name	**Pin Number**	**Type**
BCDATA_93	BE3	BI
BCDATA_94	BB6	BI
BCDATA_95	BE5	BI
BCDATA_96	A35	BI
BCDATA_97	A37	BI
BCDATA_98	F36	BI
BCDATA_99	C39	BI
BCDATA_100	D40	BI
BCDATA_101	D44	BI
BCDATA_102	J39	BI
BCDATA_103	F44	BI
BCDATA_104	L39	BI

BCDATA_105	L43	BI
BCDATA_106	L45	BI
BCDATA_107	R39	BI
BCDATA_108	U39	BI
BCDATA_109	R45	BI
BCDATA_110	Y40	BI
BCDATA_111	W43	BI
BCDATA_112	AA41	BI
BCDATA_113	AB44	BI
BCDATA_114	AD38	BI
BCDATA_115	AE39	BI
BCDATA_116	AF42	BI
BCDATA_117	AH38	BI
BCDATA_118	AL45	BI
BCDATA_119	AK40	BI
BCDATA_120	AM42	BI
BCDATA_121	AN41	BI
BCDATA_122	AT44	BI
BCDATA_123	AR39	BI
BCDATA_124	AY44	BI
BCDATA_125	AU39	BI
BCDATA_126	BB44	BI
BCDATA_127	BD42	BI
BCLOAD_L	F26	OUT
BCTAGDATA_20	F14	BI
BCTAGDATA_21	G15	BI
BCTAGDATA_22	H16	BI
BCTAGDATA_23	A11	BI
BCTAGDATA_24	B12	BI
BCTAGDATA_25	C13	BI
BCTAGDATA_26	D14	BI
BCTAGDATA_27	E15	BI
BCTAGDATA_28	F16	BI
BCTAGDATA_29	G17	BI
BCTAGDATA_30	H18	BI
BCTAGDATA_31	A15	BI
BCTAGDATA_32	B16	BI
BCTAGDATA_33	C17	BI
BCTAGDATA_34	D18	BI
BCTAGDATA_35	E19	BI
BCTAGDATA_36	A17	BI

Signal Name	**Pin Number**	**Type**
BCTAGDATA_37	F20	BI
BCTAGDATA_38	D20	BI
BCTAGDATA_39	G21	BI
BCTAGDATA_40	E21	BI
BCTAGDATA_41	A21	BI
BCTAGDATA_42	H22	BI
BCTAGDIRTY_H	G23	BI
BCTAGINCLK_H	C19	IN
BCTAGINCLK_L	B18	IN
BCTAGOE_L	H24	OUT
BCTAGOUTCLK_H	C23	OUT
BCTAGOUTCLK_L	B24	OUT
BCTAGPARITY_H	F22	BI
BCTAGSHARED_H	B22	BI
BCTAGVALID_H	F24	BI
BCTAGWR_L	A25	OUT
CLKFWDRESET_H	BC19	OUT

CLKIN_H	AN5	IN
CLKIN_L	AP4	IN
DCOK_H	BB20	IN
EV6CLK_H	AM8	OUT
EV6CLK_L	AP8	OUT
FRAMECLK_H	BA19	IN
IRQ_H_0	AY16	IN
IRQ_H_1	BA15	IN
IRQ_H_2	AV16	IN
IRQ_H_3	BB14	IN
IRQ_H_4	BC13	IN
IRQ_H_5	BD12	IN
PLLBYPASS_H	AT6	IN
PLLVDD	AT8	IN
RESET_L	AY20	IN
SROMCLK_H	AV18	BI
SROMDATA_H	AW17	BI
SROMEN_L	BE15	OUT
SYSADDINCLK_H	BE25	IN
SYSADDINCLK_L	AW25	IN
SYSADDIN_0_L	BD28	IN
SYSADDIN_1_L	BA27	IN
SYSADDIN_2_L	AY26	IN
SYSADDIN_3_L	BC27	IN
SYSADDIN_4_L	BB26	IN
SYSADDIN_5_L	BA25	IN
SYSADDIN_6_L	AV24	IN
SYSADDIN_7_L	AY24	IN
SYSADDIN_8_L	BD24	IN
SYSADDIN_9_L	AW23	IN
SYSADDIN_10_L	BC23	IN
SYSADDIN_11_L	AY22	IN
SYSADDIN_12_L	BD22	IN
SYSADDIN_13_L	BE21	IN
SYSADDIN_14_L	BA21	IN
SYSADDOUTCLK_H	BB32	OUT
Signal Name	**Pin Number**	**Type**
SYSADDOUTCLK_L	BA31	OUT
SYSADDOUT_0_L	BD36	OUT
SYSADDOUT_1_L	BC35	OUT
SYSADDOUT_2_L	BB34	OUT
SYSADDOUT_3_L	BA33	OUT
SYSADDOUT_4_L	AY32	OUT
SYSADDOUT_5_L	BE35	OUT
SYSADDOUT_6_L	BD34	OUT
SYSADDOUT_7_L	BC33	OUT
SYSADDOUT_8_L	AW31	OUT
SYSADDOUT_9_L	AV30	OUT
SYSADDOUT_10_L	AY30	OUT
SYSADDOUT_11_L	AW29	OUT
SYSADDOUT_12_L	AV28	OUT
SYSADDOUT_13_L	BE31	OUT
SYSADDOUT_14_L	BD30	OUT
SYSCHECK_0_L	AW11	BI
SYSCHECK_1_L	BD10	BI
SYSCHECK_2_L	BA13	BI
SYSCHECK_3_L	BE11	BI
SYSCHECK_4_L	AV36	BI
SYSCHECK_5_L	BC39	BI

SYSCHECK_6_L	AY36	BI
SYSCHECK_7_L	BE37	BI
SYSDATAINCLK_0_H	H6	IN
SYSDATAINCLK_0_L	J7	IN
SYSDATAINCLK_1_H	V4	IN
SYSDATAINCLK_1_L	W5	IN
SYSDATAINCLK_2_H	AL5	IN
SYSDATAINCLK_2_L	AM4	IN
SYSDATAINCLK_3_H	BA9	IN
SYSDATAINCLK_3_L	AY10	IN
SYSDATAINCLK_4_H	G41	IN
SYSDATAINCLK_4_L	F42	IN
SYSDATAINCLK_5_H	V42	IN
SYSDATAINCLK_5_L	U43	IN
SYSDATAINCLK_6_H	AH42	IN
SYSDATAINCLK_6_L	AJ43	IN
SYSDATAINCLK_7_H	BA39	IN
SYSDATAINCLK_7_L	BB40	IN
SYSDATAINVALID_L	BB28	IN
SYSDATAOUTCLK_0_H	M4	OUT
SYSDATAOUTCLK_0_L	N5	OUT
SYSDATAOUTCLK_1_H	AV6	OUT
SYSDATAOUTCLK_1_L	AW5	OUT
SYSDATAOUTCLK_2_H	M42	OUT
SYSDATAOUTCLK_2_L	N41	OUT
SYSDATAOUTCLK_3_H	AU41	OUT
SYSDATAOUTCLK_3_L	AV42	OUT
SYSDATAOUTVALID_L	BE29	IN
SYSDATA_0_L	C11	BI
SYSDATA_1_L	H12	BI
SYSDATA_2_L	E9	BI
Signal Name	**Pin Number**	**Type**
SYSDATA_3_L	B6	BI
SYSDATA_4_L	H10	BI
SYSDATA_5_L	F4	BI
SYSDATA_6_L	C1	BI
SYSDATA_7_L	H4	BI
SYSDATA_8_L	E1	BI
SYSDATA_9_L	L3	BI
SYSDATA_10_L	T8	BI
SYSDATA_11_L	N3	BI
SYSDATA_12_L	V8	BI
SYSDATA_13_L	U3	BI
SYSDATA_14_L	W3	BI
SYSDATA_15_L	AA7	BI
SYSDATA_16_L	AB2	BI
SYSDATA_17_L	AC7	BI
SYSDATA_18_L	AD6	BI
SYSDATA_19_L	AE5	BI
SYSDATA_20_L	AF6	BI
SYSDATA_21_L	AJ1	BI
SYSDATA_22_L	AL1	BI
SYSDATA_23_L	AK6	BI
SYSDATA_24_L	AR1	BI
SYSDATA_25_L	AM6	BI
SYSDATA_26_L	AU1	BI
SYSDATA_27_L	AY4	BI
SYSDATA_28_L	BB2	BI
SYSDATA_29_L	BD4	BI

SYSDATA_30_L	BA7	BI
SYSDATA_31_L	BD6	BI
SYSDATA_32_L	C35	BI
SYSDATA_33_L	H34	BI
SYSDATA_34_L	E37	BI
SYSDATA_35_L	B40	BI
SYSDATA_36_L	G37	BI
SYSDATA_37_L	C45	BI
SYSDATA_38_L	K38	BI
SYSDATA_39_L	G43	BI
SYSDATA_40_L	M38	BI
SYSDATA_41_L	K44	BI
SYSDATA_42_L	M44	BI
SYSDATA_43_L	T38	BI
SYSDATA_44_L	V38	BI
SYSDATA_45_L	U45	BI
SYSDATA_46_L	AA39	BI
SYSDATA_47_L	Y42	BI
SYSDATA_48_L	AA45	BI
SYSDATA_49_L	AC39	BI
SYSDATA_50_L	AD40	BI
SYSDATA_51_L	AE41	BI
SYSDATA_52_L	AG43	BI
SYSDATA_53_L	AJ45	BI
SYSDATA_54_L	AJ39	BI
SYSDATA_55_L	AL41	BI
Signal Name	**Pin Number**	**Type**
SYSDATA_56_L	AM40	BI
SYSDATA_57_L	AP44	BI
SYSDATA_58_L	AU45	BI
SYSDATA_59_L	AT40	BI
SYSDATA_60_L	BA45	BI
SYSDATA_61_L	AY42	BI
SYSDATA_62_L	BC45	BI
SYSDATA_63_L	BE43	BI
SYSFILLVALID_L	BC29	IN
TESTCLK_H	BC17	IN
TESTDATAIN_H	BB18	IN
TESTDATAOUT_H	BD18	OUT
TESTMODESELECT_H	BD16	IN
TESTRESET_L	BE17	IN
VREFBCACHE	AW21	IN
VREFSYS	AV22	IN

Ground Pins

BD44	D36	Y8	AV26
BB42	F34	Y2	AY28
AY40	H32	AB4	BB30
AV38	B32	AC1	BD32
AV44	D30	AD4	AV32
AT42	F28	AF2	AY34
AP40	H26	AF8	BB36
AM38	B26	AH6	BD38
AM44	D24	AK4	AV38
AK42	A23	AM2	BD26
AH40	D22	AP6	
AF38	B20	AR7	

AF44	H20	AU7
AD42	F18	AT4
AC45	D16	AV2
AB42	B14	AY6
Y38	H14	BB4
Y44	F12	BD2
V40	D10	AV8
T42	B8	AV14
P44	G9	AY12
P38	B4	BB10
M40	B2	BD8
K42	D4	AV20
H44	F6	AY18
H38	H2	BB16
F40	K4	BD14
D42	M6	BD20
B44	P8	BB22
B42	P2	BB24
B38	V6	BE23

VDD_2V

BC43	A19	AW13
BA41	G19	BE13
AW45	E17	BC15
AU43	C15	BA17
AR41	A13	AW19
AN39	G13	BE19
AN45	E11	BC21
AL43	C9	BA23
AJ41	A7	BC25
AG39	A5	BE27
AG45		AW27
AE43	A3	BA29
AC41	C3	BC31
AA43	E5	BE33
W45	G7	AW33
W39	G1	BA35
U41	J3	BC37
R43	L5	BE39
N45	N7	AW39
N39	N1	
L41	R3	
J43	U5	
G45	W7	
G39	W1	
E41	AA3	
C43	AC5	
A43	AE3	
	AG1	
A41	AG7	
H36	AJ5	
A39	AL3	
C37	AN1	
E35	AN7	
G33	AR5	

A33	AU3
C31	AW1
E29	BC3
G27	BA5
A27	AW7
C25	BE7
E23	BC9
C21	BA11

11. Appendix 1: Reset and Sleep Mode

This chapter contains reset and sleep mode information. It has not been written yet.

12. Appendix 2: PAL Coding Restrictions

12.1 Restriction: Reset Sequence Required by Retirator and Mapper

- (a) For convenience of implementation, the retirator "done" status bits are not initialized during reset. Instead, it relies on the first batch of valid instructions to sweep through inum-space and initialize these bits. The 80 status bits, corresponding to the maximum number of inflight instructions, must be marked "not done" by the first 80 instructions mapped after reset and subsequently marked "done" when those instructions retire. Therefore, the first 20 fetch blocks must contain 4 valid instructions apiece, and containing no retirator-nops (see previous guideline).

- Example:
```
reset:
ADDQ R31,#19,R0
ADDQ R31,R0,R0
ADDQ R31,R0,R0
ADDQ R31,R0,R0
loop:
SUBQ R0,#1,R0
ADDQ R31,R0,R0
ADDQ R31,R0,R0
BNE R0, loop
continue:
...
```

- Note that all four instructions in each fetch block are valid and none have R31 as a destination. (b) For convenience of implementation, the mapper requires that all virtual registers (architected and PAL shadow, excluding R31 and F31) be used as destinations before they are used as sources. In other words, the hardware does not create the "initial mapping" of virtual-to-physical registers; it relies on software. Since there is no hardware-created initial mapping, a virtual register cannot be used as a source operand before it is mapped. An example initial mapping sequence is as follows:

-
```
ADDQ R31,R31,R0
ADDQ R31,R31,R1
ADDQ R31,R31,R2
ADDQ R31,R31,R3

ADDQ R31,R31,R4
ADDQ R31,R31,R5
ADDQ R31,R31,R6
ADDQ R31,R31,R7

...

ADDQ R31,R31,R28
ADDQ R31,R31,R29
ADDQ R31,R31,R30
ADDQ R31,R31,R1 ; note that R31 need not be initialized as a destination

ADDF F31,F31,F0
ADDF F31,F31,F1
ADDF F31,F31,F2
ADDF F31,F31,F3

...

ADDF F31,F31,F4
ADDF F31,F31,F5
ADDF F31,F31,F6
ADDF F31,F31,F7

ADDF F31,F31,F28
ADDF F31,F31,F29
ADDF F31,F31,F30
ADDF F31,F31,F1 ; note that R31 need not be initialized as a destination
```

- Note that this sequence can be used to initialize the retirator staus bits as well.

12.2 Restriction: No Multiple Writers to IPRs in Same Scoreboard Group

- Only one explicit writer (HW_MTPR) to IPRs that are in the same group can appear in the same fetch block (octaword-aligned octaword). Multiple explicit writers to IPRs that are NOT in the same scoreboard group can appear. If this restriction is violated the IPR readers might not see the in-order state. Also, the IPR might ultimately end up with a bad value. This is for convenience of implementation.

12.3 Restriction: (removed)

12.4 Restriction: No Writers and Readers to IPRs in Same Scoreboard Group

- Within one fetch block (octaword-aligned octaword), an implicit or explicit reader of an IPR in a particular Scoreboard Group an not follow an explicit writer (HW_MTPR) to an IPR in that scoreboard group. This is for convenience of implementation. Note that implicit readers include all memory operations and JSR/HW_RET.

12.5 Restriction: PAL shadow enables

- Once pal shadows are enabled (via I_CTL<SDE>), the NT-mode (I_CTL<NT_MODE>) state must not be changed. Enabling PAL shadows will allow the assignment of 8 physical registers to the 8 additional general-purpose register specifier as determined by I_CTL<NT_MODE>. Subsequent changing of I_CTL<NT_MODE> will assign 8 additional physical registers to the specifiers in the new overlay range but will not deallocate the prior 8 registers. The net effect is that 8 physical register will be removed from the resource pool.

12.6 Guideline: Avoid Consecutive read-modify-write-read-modify-write sequences to IPRs in the Same Scoreboard Group

- The latency between the first write and the second read is determined by the retire latency of the IPR. For convenience of implementation, the latency between when the read issues and the final write issues depends on the runtime contents of the issue queue. It is somewhere between 4 and 9 cycles even if there is no data dependency between the read and write.

12.7 Restriction: Replay trap and interrupt code sequence and STF/ITOF

- On an MBOX replay trap, the EV6 Ibox guarantees that the refetched load or store that caused the trap will issue before any newer loads or stores. For loads and integer stores this is a consequence of the natural operation of the issue queue. The refetched instruction enter the age-prioritized queue ahead of newer loads and stores will not have any dependencies on dirty registers. Since there is no time-overhead for checking these register dependencies (i.e. it is known upon enqueueing that there are no dirty registers) The queue will issue it in priority order. For floating stores, there is normally some overhead associated with checking the floating point source register dirty status so the store would normally wait before issuing. This would have the undesired consequence of allowing newer loads and stores to issue out-of-order. A deadlock can occur if this out-of-order issue causes the floating store to continually replay trap. To avoid the deadlock, on a floating store replay trap, the source register dirty status is not checked (the source register is assumed to be clean because the store was issued before).
- The hardware mechanism which keeps track of replayed floating stores and cancels the dirty register check requires some software restrictions to guarantee that it is applied appropriately to the replayed instruction and not to other floating stores. It operates by marking the position in the fetch block (low two bits of the PC) where the replay trap occurred and then canceling the floating point dirty source register check of the next valid instruction enqueued to the integer queue (integer instructions, all loads and stores, and ITOF) which has the same position in the fetch block (normally the replayed STF). If the PC is somehow diverted to a PAL flow, this hardware might inadvertently cancel the register check of some other STF (or ITOF) instruction. Fortunately, there are a minimal number of reasons why the PC might be diverted during a replay trap. They are:
  Interrupts
  ITB Fill
  (others?)
- In these PAL flows, a STF or ITOF instruction in a given position in a fetch block must be preceded by a valid instruction that is issued out of the integer queue in the same position in an earlier fetch block. Acceptable instruction classes include loads, integer stores, integer operates that do not have R31 as a destination, branches.
- Example:

```
Bad_Interrupt_Flow_Entry:
ADDQ R31,R31,R0
STF Fa,(Rb) ; this STF might NOT undergo a dirty source register check and might give wrong
results
ADDQ R31,R31,R0
ADDQ R31,R31,R0
...
Good_Interrupt_Flow_Entry:
ADDQ R31,R31,R0 ; enables FP dirty source register check for (PC<1:0> == 00)
ADDQ R31,R31,R0 ; enables FP dirty source register check for (PC<1:0> == 01)
ADDQ R31,R31,R0 ; enables FP dirty source register check for (PC<1:0> == 10)
ADDQ R31,R31,R0 ; enables FP dirty source register check for (PC<1:0> == 11)

ADDQ R31,R31,R0
STF Fa,(Rb) ; this STF will successfully undergo a dirty source register check
ADDQ R31,R31,R0
ADDQ R31,R31,R0
```

12.8 Restriction (removed)

12.9 Restriction: PALmode I-Stream address ranges

- PALmode<physical> I-Stream addresses must insure proper sign extension for the selected value of I_CTL<VA_WIDE>. When I_CTL<VA_WIDE> is clear, indicating 43-bit virtual address format, PALmode<physical> I-Stream addresses must sign extend address bits above bit 42 although physical address range is 44 bits. An illegal address can only be generated by a PALmode JSR-type instruction or a HW_RET instruction returning to a PALmode address.

12.10 Restriction: Duplicate IPR mode bits

- Duplicate IPR mode bits I_CTL<VA_WIDE> and VA_CTL<VA_WIDE>, I_CTL<NT_MODE> and VA_CTL<NT_MODE> must be equal when executing in native(virtual) mode.

12.11 Guideline: Ibox IPR update synchronization

- When updating any Ibox IPR, a return to native(virtual) mode should use the HW_RET instruction with associated STALL bit set to insure that the updated IPR value affects all instructions following the return path. The new IPR value takes effect only after the associated HW_MTPR instruction retires.

12.12 Restriction: HW_MFPR EXC_ADDR/IVA_FORM/EXC_SUM Usage

- These three registers are sourced by non-renamed hardware registers that need to be available for subsequent traps. Hardware protects the values from overwrite by locking the registers, but only for a limited time. Their values can only be read reliably by a HW_MFPR within the first four instructions of a PALflow AND prior to any taken branch in that PALflow, whichever is earlier. After the delimiting instruction defined above retires, these registers are unlocked and may change due to new exception conditions.
- If a second exception occurs before the registers are unlocked, it will be either delayed or forced to replay trap until the register has been unlocked. After being unlocked, a subsequent, new path exception condition will be allowed to reload the register and trap to PAL. Note that the CPU may complete execution of the first PAL flow, encountering the second exception condition before the delimiting instruction retires, hence the need for the locking mechanism to insure visibility of the initial register value.

12.13 Restriction: DTB FILL flow collision

- Two DTB Fill flows might collide such that the HW_MTPR's in the second fill could issue before all of the HW_MTPR's in the first flow retired. This can be prevented by putting appropriate software scoreboard barriers in the PAL flow.

12.14 Restriction: HW_RET

- No hw_ret in the first fetch block of PAL routine. The HW_RET will be mispredicted and the JSR/RETURN stack might lose its synchronization.

12.15 Restriction: (REMOVED)

12.16 Restriction: JSR-BAD VA

- A JSR memory format instruction which generates a bad VA (IACV) trap requires PAL assistance to determine the correct exception address. If the EXC_SUM<BAD_IVA> bit is set, bits <63,1> of the exception address are valid in the VA IPR and not the EXC_ADDR as usual. The PALmode bit, however, is always located in EXC_ADDR<0> and must be combined, if necessary, by PALcode to determine the full exception address.

12.17 Restriction: MTPR to DTB_TAG0/DTB_PTE0/DTB_TAG1/DTB_PTE1

- These four writes must be executed atomically, i.e. either all four must issue and retire or none of them may issue and retire.

12.18 Restriction: No FP OPERATES or FP CONDITIONAL BRANCHES in same fetch block as MTPR

- For convenience of implementation, no floating point operate instructions or FP conditional branches in the same fetch block as any move-to-processor register instructions. This inludes ADDF/MULF/DIVF/FBxx but does not include LDF/STF or ITOF/FTOI.

12.19 Restriction: HW_RET/STALL after updating the FPCR via MT_FPCR in PALmode

- FPCR updating happens in hardware based on the retire of nontrapping version of MT_FPCR (in PALcode). Use a HW_RET/STALL after the nontrapping MT_FPCR to achieve minimum latency (4 cycles) between the retiring of the MT_FPCR and the first FLOP that uses the updated FPCR.

12.20 Guideline: I_CTL SBE Stream Buffer Enable

- The I_CTL(SBE) bits should not be enabled when running with the Icache disabled to avoid potentially long fill delays. When the Icache is disabled, the only method of supplying instructions is via a stream hit. If the fill is returned in non-sequential wrap order, the stream will continue fetching through the entire page while waiting for a hit. Normally the data will be found in the cache.

12.21 Restriction: HW_RET/STALL after MT ASN0/ASN1

- There must be a scoreboard bit -> register dependency chain to prevent MT ASN0 or MT ASN1 from issuing while ANY of scoreboard bits <7:4> are set. A code sequence which accomplishes this:

```
; assume Ra holds valuc to write to ASN0/ASN1
HW_MFPR IPR_VA,SCBD<7,6,5,4>,R0
XOR R0,R0,R0
BIS R0,R9,R9
BIS R31,R31,R31
HW_MTPR R9,ASN0,SCBD<4>
HW_MTPR R9,ASN1,SCBD<7>
```

- This sequence guarantees, through the register dependency on R0, that neither HW_MTPR are issued before scoreboard bits <7:4> are cleared. In addition, there must be a HW_RET/STALL after a MT ASN0/MT ASN1 pair. Finally, these two writes must be executed atomically, i.e. either both must issue and retire or neither may issuc and retire.

12.22 Restriction: HW_RET/STALL after MT IS0/IS1

- There must be a scoreboard bit -> register dependency chain to prevent MT IS0 or MT IS1 from issuing while ANY of scoreboard bits <7:4> are set. A code sequence which accomplishes this:

```
HW_MFPR IPR_VA,SCBD<7,6,5,4>,R0
XOR R0,R0,R0
BIS R0,R9,R9
BIS R31,R31,R31
HW_MTPR R9,IS0,SCBD<6>
HW_MTPR R9,IS1,SCBD<7>
```

- This sequence guarantees, through the register dependency on R0, that neither HW_MTPR are issued before There must be a HW_RET/STALL after a MT IS0/MT IS1 pair. Also, these two writes must be executed atomically, i.e. either both must issue and retire or neither may issue and retire.

12.23 Restriction: HW_ST/P/CONDITIONAL does not "clear" the lock flag

- A HW_ST/P/CONDITIONAL will not "clear" the lock flag such that a successive store-conditional (either STx_C or HW_ST/C) might succeed even in the absence of a load-locked instruction. In EV6 a store-conditional is forced to fail if there is an intervening memory operation between the store-conditional and its address-matching LDxL. The memory operations are:

LDL/Q/F/G/S/T
STL/Q/F/G/S/T
LDQ_U (not to R31)
STQ_U
Absent from this list are HW_LD (any type), HW_ST (any type), ECB, and WH64. Their absence implies that they will NOT force a subsequent store-conditional instruction to fail. PALcode MUST insert a memory operation from the above list after a HW_ST/CONDITIONAL in order to force a future store-conditional to fail if it was not preceded by a load-locked:

```
HW_LD/L
xxx
HW_ST/C -> R0
Bxx R0, try_again
STQ /*force next ST/C to fail if no preceding LDxL */
HW_RET
```

12.24 Restriction: HW_RET/STALL after MT ITB_IA, ITB_IAP, IC_FLUSH

- There must be a HW_RET/STALL after a MT ITB_IA, ITB_IAP or IC_FLUSH. The Icache flush associated with these instructions will not occur until the HW_RET stall occurs and all outstanding I-stream fetches have been completed.

12.25 Restriction: MT ITB_IA after Reset

- An MT ITB_IA is required in the reset PALcode to initialize the ITB. It is also required that PALcode not be exited, even via a mispredicted path until this MT ITB_IA has retired. PALmode can change temporarily after fetching a HW_RET, regardless of the STALL qualifier, down a mispredicted path leading to use of the ITB before it is actually initialized.
- Unexpected instruction fetch and execution can occur following misprediction of any memory format Control instruction (JMP,JSR,RET,JSR_CO, or HW_JMP,HW_JSR,HW_RET, HW_JSR_CO regardless of the STALL qualifier), or after any mispredicted conditional branch instruction. If the unexpected instruction flow contains a HW_RET instruction, PALmode may be exited prematurely.
- One way to insure that PALmode is not exited is to place the MT ITB_IA at least 80 instructions before any possible HW_RET instruction can be encountered via any fetch path. Since memory format Control instructions can mispredict to any cache location, they should also be avoided within these 80 instructions.

12.26 Guideline: Conditional branches in PALcode

- To avoid pollution of the branch predictors and improve overall branch prediction accuracy, conditional branch instructions in PALcode will be predicted not taken. The only exception to this rule are conditional branches within the first cache fetch (up to four instructions) of all pal flows except call_pal flows. It is advisable that conditional branches be avoided in this window.

12.27 Restriction: Reset of 'Force-Fail Lock Flag' State in PALcode

- A virtual mode load or store is required in PAL code before the execution of any load-locked or store-conditional instructions. The virtual-mode load or store may not be a HW_LD, HW_ST, LDx_L, ECB, or WH64.

1.