MIPS® PDtrace™ Specification

Document Number: MD00439
Revision 6.11
Nov 06, 2008

MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043-1353

Template: nB1.03, Built with tags: 2B MIPS32

Table of Contents

List of Figures

List of Tables

Chapter 1

About This Book

1.1 Typographical Conventions

This section describes the use of *italic*, **bold** and `courier` fonts in this book.

1.1.1 Italic Text

- is used for *emphasis*
- is used for *bits*, *fields*, *registers*, that are important from a software perspective (for instance, address bits used by software, and programmable fields and registers), and various *floating point instruction formats*, such as *S*, *D*, and *PS*
- is used for the memory access types, such as *cached* and *uncached*

1.1.2 Bold Text

- represents a term that is being **defined**
- is used for **bits** and **fields** that are important from a hardware perspective (for instance, **register** bits, which are not programmable but accessible only to hardware)
- is used for ranges of numbers; the range is indicated by an ellipsis. For instance, **5..1** indicates numbers 5 through 1
- is used to emphasize **UNPREDICTABLE** and **UNDEFINED** behavior, as defined below.

1.1.3 Courier Text

`Courier` fixed-width font is used for text that is displayed on the screen, and for examples of code and instruction pseudocode.

1.2 UNPREDICTABLE and UNDEFINED

The terms **UNPREDICTABLE** and **UNDEFINED** are used throughout this book to describe the behavior of the processor in certain cases. **UNDEFINED** behavior or operations can occur only as the result of executing instructions in a privileged mode (i.e., in Kernel Mode or Debug Mode, or with the CP0 usable bit set in the Status register). Unprivileged software can never cause **UNDEFINED** behavior or operations. Conversely, both privileged and unprivileged software can cause **UNPREDICTABLE** results or operations.

1.2.1 UNPREDICTABLE

UNPREDICTABLE results may vary from processor implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. Software can never depend on results that are **UNPREDICTABLE**. **UNPREDICTABLE** operations may cause a result to be generated or not. If a result is generated, it is **UNPREDICTABLE**. **UNPREDICTABLE** operations may cause arbitrary exceptions.

UNPREDICTABLE results or operations have several implementation restrictions:

- Implementations of operations generating **UNPREDICTABLE** results must not depend on any data source (memory or internal state) which is inaccessible in the current processor mode
- **UNPREDICTABLE** operations must not read, write, or modify the contents of memory or internal state which is inaccessible in the current processor mode. For example, **UNPREDICTABLE** operations executed in user mode must not access memory or internal state that is only accessible in Kernel Mode or Debug Mode or in another process
- **UNPREDICTABLE** operations must not halt or hang the processor

1.2.2 UNDEFINED

UNDEFINED operations or behavior may vary from processor implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. **UNDEFINED** operations or behavior may vary from nothing to creating an environment in which execution can no longer continue. **UNDEFINED** operations or behavior may cause data loss.

UNDEFINED operations or behavior has one implementation restriction:

- **UNDEFINED** operations or behavior must not cause the processor to hang (that is, enter a state from which there is no exit other than powering down the processor). The assertion of any of the reset signals must restore the processor to an operational state

1.3 Special Symbols in Pseudocode Notation

In this book, algorithmic descriptions of an operation are described as pseudocode in a high-level language notation resembling Pascal. Special symbols used in the pseudocode notation are listed in Table 1.1.

Table 1.1 Symbols Used in Instruction Operation Statements

Symbol	Meaning
←	Assignment
=, ≠	Tests for equality and inequality
‖	Bit string concatenation
x^y	A *y*-bit string formed by *y* copies of the single-bit value *x*
b#n	A constant value *n* in base *b*. For instance 10#100 represents the decimal value 100, 2#100 represents the binary value 100 (decimal 4), and 16#100 represents the hexadecimal value 100 (decimal 256). If the "b#" prefix is omitted, the default base is 10.
$x_{y..z}$	Selection of bits *y* through *z* of bit string *x*. Little-endian bit notation (rightmost bit is 0) is used. If *y* is less than *z*, this expression is an empty (zero length) bit string.

Table 1.1 Symbols Used in Instruction Operation Statements (Continued)

Symbol	Meaning
+, −	2's complement or floating point arithmetic: addition, subtraction
*, ×	2's complement or floating point multiplication (both used for either)
div	2's complement integer division
mod	2's complement modulo
/	Floating point division
<	2's complement less-than comparison
>	2's complement greater-than comparison
≤	2's complement less-than or equal comparison
≥	2's complement greater-than or equal comparison
nor	Bitwise logical NOR
xor	Bitwise logical XOR
and	Bitwise logical AND
or	Bitwise logical OR
GPRLEN	The length in bits (32 or 64) of the CPU general-purpose registers
GPR[x]	CPU general-purpose register *x*. The content of *GPR[0]* is always zero.
FPR[x]	Floating Point operand register *x*
FCC[CC]	Floating Point condition code CC. *FCC[0]* has the same value as *COC[1]*.
FPR[x]	Floating Point (Coprocessor unit 1), general register *x*
CPR[z,x,s]	Coprocessor unit *z*, general register *x*, select *s*
CCR[z,x]	Coprocessor unit *z*, control register *x*
COC[z]	Coprocessor unit *z* condition signal
Xlat[x]	Translation of the MIPS16 GPR number *x* into the corresponding 32-bit GPR number
BigEndianMem	Endian mode as configured at chip reset (0 →Little-Endian, 1 → Big-Endian). Specifies the endianness of the memory interface (see LoadMemory and StoreMemory pseudocode function descriptions), and the endianness of Kernel and Supervisor mode execution.
BigEndianCPU	The endianness for load and store instructions (0 → Little-Endian, 1 → Big-Endian). In User mode, this endianness may be switched by setting the *RE* bit in the *Status* register. Thus, BigEndianCPU may be computed as (BigEndianMem XOR ReverseEndian).
ReverseEndian	Signal to reverse the endianness of load and store instructions. This feature is available in User mode only, and is implemented by setting the *RE* bit of the *Status* register. Thus, ReverseEndian may be computed as (SR_{RE} and User mode).
LLbit	Bit of **virtual** state used to specify operation for instructions that provide atomic read-modify-write. *LLbit* is set when a linked load occurs; it is tested and cleared by the conditional store. It is cleared, during other CPU operation, when a store to the location would no longer be atomic. In particular, it is cleared by exception return instructions.

Table 1.1 Symbols Used in Instruction Operation Statements (Continued)

Symbol	Meaning
I:, **I+n**:, **I-n**:	This occurs as a prefix to *Operation* description lines and functions as a label. It indicates the instruction time during which the pseudocode appears to "execute." Unless otherwise indicated, all effects of the current instruction appear to occur during the instruction time of the current instruction. No label is equivalent to a time label of **I**. Sometimes effects of an instruction appear to occur either earlier or later — that is, during the instruction time of another instruction. When this happens, the instruction operation is written in sections labeled with the instruction time, relative to the current instruction **I**, in which the effect of that pseudocode appears to occur. For example, an instruction may have a result that is not available until after the next instruction. Such an instruction has the portion of the instruction operation description that writes the result register in a section labeled **I+1**. The effect of pseudocode statements for the current instruction labelled **I+1** appears to occur "at the same time" as the effect of pseudocode statements labeled **I** for the following instruction. Within one pseudocode sequence, the effects of the statements take place in order. However, between sequences of statements for different instructions that occur "at the same time," there is no defined order. Programs must not depend on a particular order of evaluation between such sections.
PC	The *Program Counter* value. During the instruction time of an instruction, this is the address of the instruction word. The address of the instruction that occurs during the next instruction time is determined by assigning a value to *PC* during an instruction time. If no value is assigned to *PC* during an instruction time by any pseudocode statement, it is automatically incremented by either 2 (in the case of a 16-bit MIPS16 instruction) or 4 before the next instruction time. A taken branch assigns the target address to the *PC* during the instruction time of the instruction in the branch delay slot.
PABITS	The number of physical address bits implemented is represented by the symbol PABITS. As such, if 36 physical address bits were implemented, the size of the physical address space would be $2^{PABITS} = 2^{36}$ bytes.
FP32RegistersMode	Indicates whether the FPU has 32-bit or 64-bit floating point registers (FPRs). In MIPS32, the FPU has 32 32-bit FPRs in which 64-bit data types are stored in even-odd pairs of FPRs. In MIPS64, the FPU has 32 64-bit FPRs in which 64-bit data types are stored in any FPR. In MIPS32 implementations, **FP32RegistersMode** is always a 0. MIPS64 implementations have a compatibility mode in which the processor references the FPRs as if it were a MIPS32 implementation. In such a case **FP32RegisterMode** is computed from the FR bit in the *Status* register. If this bit is a 0, the processor operates as if it had 32 32-bit FPRs. If this bit is a 1, the processor operates with 32 64-bit FPRs. The value of **FP32RegistersMode** is computed from the FR bit in the *Status* register.
InstructionInBranchDelaySlot	Indicates whether the instruction at the Program Counter address was executed in the delay slot of a branch or jump. This condition reflects the *dynamic* state of the instruction, not the *static* state. That is, the value is false if a branch or jump occurs to an instruction whose PC immediately follows a branch or jump, but which is not executed in the delay slot of a branch or jump.
SignalException(exception, argument)	Causes an exception to be signaled, using the exception parameter as the type of exception and the argument parameter as an exception-specific argument). Control does not return from this pseudocode function - the exception is signaled at the point of the call.

1.4 For More Information

Various MIPS RISC processor manuals and additional information about MIPS products can be found at the MIPS URL: http://www.mips.com

Comments or questions on the MIPS32™ Architecture or this document should be directed to

Director of MIPS Architecture
MIPS Technologies, Inc.

1225 Charleston Road
Mountain View, CA 94043

or via E-mail to architecture@mips.com.

Chapter 2

Overview of the MIPS® PDtrace™ Architecture

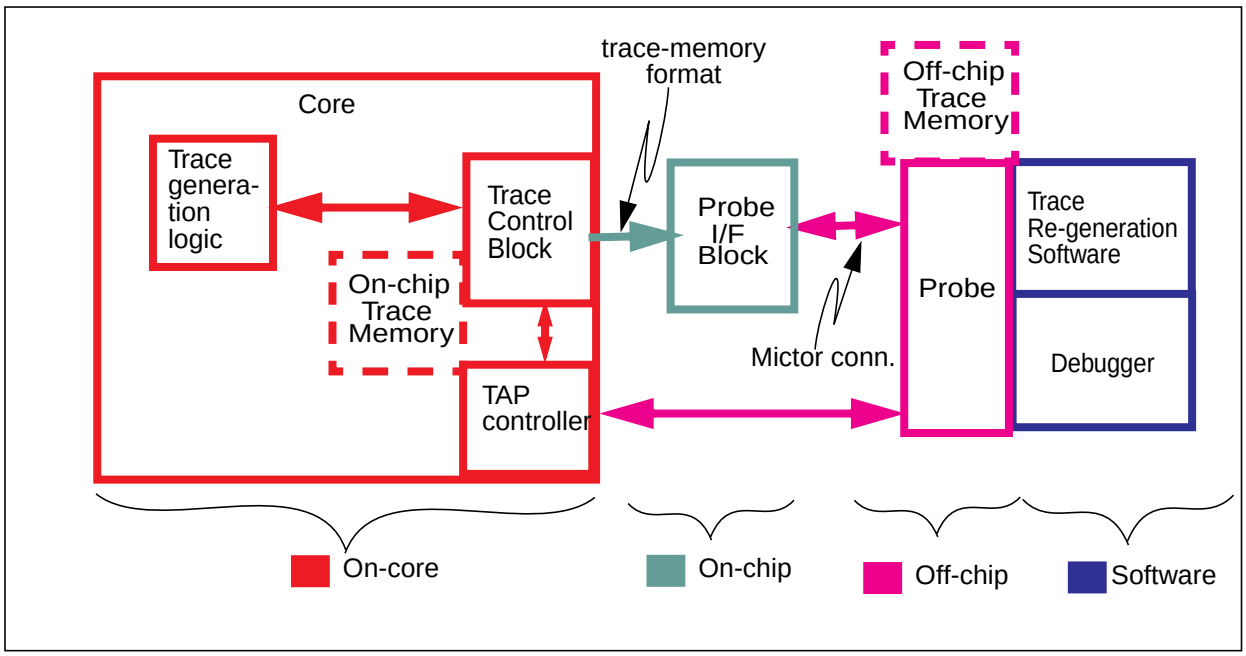
This document contains the MIPS® PDtrace™ specification which is the control and formats for tracing Program and Data from a MIPS® processor core or a System on Chip (SoC) that includes multiple MIPS processor cores. The SoC can include other important system elements such as a system bus or other IP (Intellectual Property) blocks that need to be traced for functional evaluation and performance measurements. In this situation, the trace formats provided by MIPS provide the ability to include trace from other blocks on the SoC and also provide the ability to be easily extended as needed by a customer. The processor-specific parts of the PDtrace specification captures information from each pipeline within the processor. The PDtrace specification includes trace control and formats for non-processor specific blocks provided by MIPS such as the CM (Coherence Manager) block in the CMP system. The specification lays out the details of how the trace from multiple blocks on-chip are to be combined to provide a single trace stream on the chip interface pins. Note that processor-implementation specific trace information and formats are included in the appendix of this document since this can be modified per implementation and does not necessarily constitute architecture.

What sort of information should be traced and captured into the trace stream going into trace memory is controlled in two ways. By CP0 control registers that can be programmed by user applications. Hence, an application can trace itself provided the needed hardware components and trace memory are present. Tracing can also be controlled via TCB (Trace Control Block) control registers in the PDtrace architecture that can be programmed by an external probe using the EJTAG TAP controller hardware or via software through the debug memory segment (this feature is only available cores that implement PDtrace revision 6.00 and higher). This allows users to control tracing at the execution time of applications using an external agent like the debugger that communicates to these control register using a debugger probe.

In most implementations, the trace information within a core, coming from the pipeline tracing logic is captured by a block called the Trace Control Block. This block holds registers used to control the trace information that is captured from within the core. This unit is also used to format the trace information into the architecturally specified trace formats, readying the information for writing into trace memory. The trace memory may be either on-chip or off-chip based on user requirements. The trace information written to memory is compressed and assumes that post-processing software has access to the static program image to reconstruct the dynamic program flow. Compression reduces the number of signals (hence pins) required to gather this information and also reduces the trace size.

Figure 2.1 illustrates one possible configuration for trace capture and post-analysis using software. The figure shows a core with trace generation logic and a TAP controller. This core is connected to a trace control block (TCB) via the TAP controller (since the TCB implements and uses TAP registers). The trace memory associated with the trace control block can either be located on-chip, or off-chip. An on-chip trace buffer will be smaller and will be writable by the TCB at higher speeds, while an off-chip trace memory can be much larger and is written via the potentially slower pin interface out of the core. Probe hardware and software connects to the TCB and the TAP controller via the chip's pin interface and allows debugger software to start, stop, and examine program execution traces.

Figure 2.1 Illustration of a PC and Data Trace Flow



The rest of this document describes the two aspects of the MIPS PDtrace Specification, the trace control and the trace formats. It also provides material in the appendix that are either implementation-specific or hints to implementation. This document serves three functions: (1) it provides a specification of the trace interface for the core designer, (2) it provides sufficient detail for an architecture licensee to build a trace control block that would work with existing probes from third parties, and (3) it provides sufficient details to design and code a post-processing software module for trace re-construction.

Implementation of PDtrace is optional for a given MIPS-compatible processor. Whether a core or processor implements PDtrace is indicated by a bit in the Coprocessor 0 Config3 register as shown in Figure 2.2 and Table 2.1.

Note that if a core or processor does not implement EJTAG, then the PDtrace tracing logic can still be implemented.

Figure 2.2 *Config3* Register Format

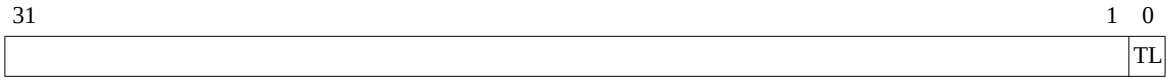


Table 2.1 *Config3* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
	31:1	As per the MIPS32 and MIPS64 Architecture specifications			
TL	0	This bit is used to indicate the presence of tracing logic in the processor. 0 : No tracing logic implemented 1 : Tracing logic implemented	R	Preset	Required

2.1 Processor Modes

The PDtrace specification allows tracing to be enabled or disabled based on various processor modes. This section precisely describes these modes, and the terminology is then used later in the document.

```
DebugMode ← (Debug_DM = 1)
ExceptionMode ← (not DebugMode) and ((Status_EXL = 1) or (Status_ERL = 1))
KernelMode ← (not (DebugMode or ExceptionMode)) and (Status_KSU = 2#00)
SupervisorMode ← (not (DebugMode or ExceptionMode)) and (Status_KSU = 2#01)
UserMode ← (not (DebugMode or ExceptionMode)) and (Status_KSU = 2#10)
```

2.2 Subsetting

The PDtrace specification allows four levels of subsetting. Within each level, all features required to support the level must be implemented. The allowable subsets are:

1. No PDtrace implemented
2. PDtrace with PC tracing only
3. PDtrace with PC and load and store address tracing only
4. PDtrace with PC, load and store address, and load and store data tracing

The specific subset implemented by a processor or core can be determined by reading the TL bit (0) of the ***Config3*** register (see Table 2.1) and the ImpSubset bits (6:5) in the ***TraceControl2*** register (see Table 4.14 on page 55).

In addition, Trace Trigger from EJTAG Hardware breakpoints (see 4.4 “Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints” on page 48) is optional. This feature depends on the EJTAG optional feature for hardware instruction and data breakpoints. The exact nature of this subsetting is described in later chapters.

2.3 Overview of the Trace Control Block

The tracing logic within the processor core (shown in Figure 2.1) outputs all trace information to the on-core trace control block (TCB) unit. The TCB is responsible for collecting the trace data and storing this trace data in an on-chip trace memory or an off-chip trace memory using the Probe Interface Block (PIB). The TCB is also responsible for taking user requests for trace as bits in its control registers. These bits determine what is traced and what is output. Chapter 3 of this document describes the general PDtrace specification and the CP0 trace control registers. This is followed by chapters that describe the TCB control registers in the TCB trace formats used to write out the trace information to memory. The description of the TCB includes:

- The details on the TCB’s internal architecture, i.e., registers, and how these registers are used to control tracing
- The trace formats used by the TCB to write trace information to memory
- The interface between the TCB and the TAP controller

This document does not include:

- The TCtrace Interface that connects the TCB to the Probe Interface Block which is off-core but on-chip.

- The PIB, and
- The external Probe interface including its electrical characteristics.

This information is available in a core-specific document.

Figure 2.3 shows the TCB, the PIB, and the trace data path from the TCB to the Probe IF. It is optional whether the TCB implements on-chip trace memory and/or the TCtrace IF with a PIB and off-chip trace memory.

Figure 2.3 TCB and Optional PIB Overview

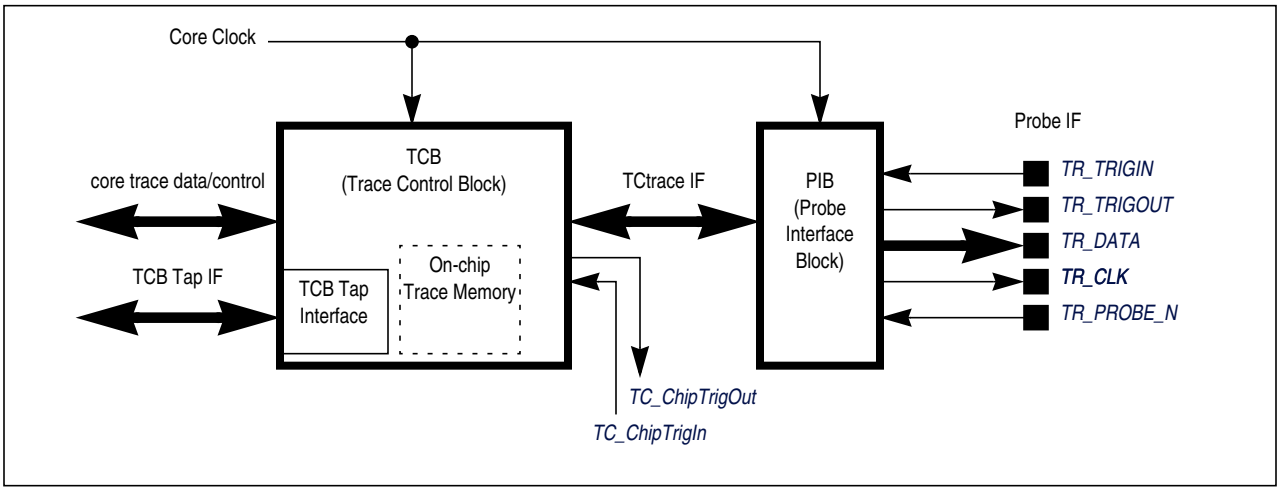


Figure 2.4 Illustration of the Core and TCB with External Trace Memory

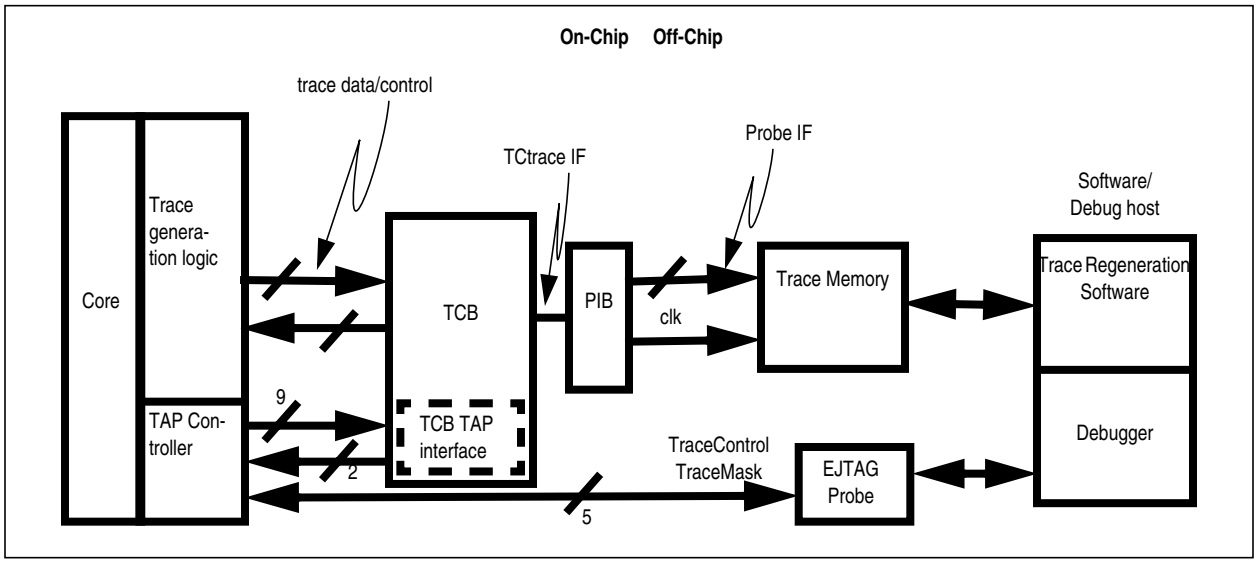
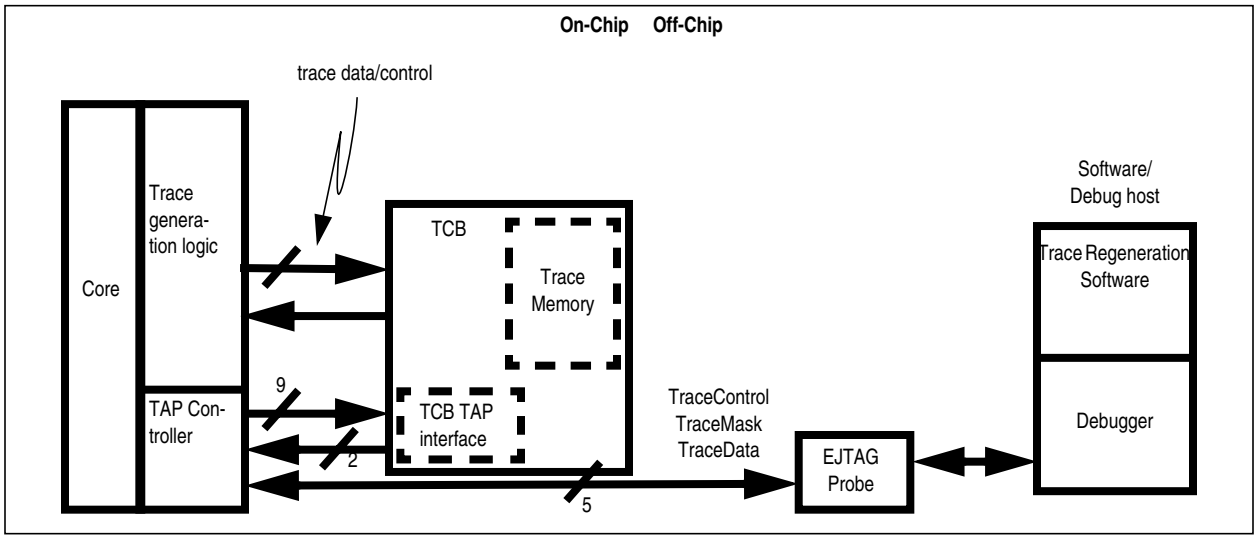


Figure 2.4 shows the full system configuration when the TCB is streaming data to off-chip trace memory through the PIB. The number of pins needed for trace data on the Probe IF is configurable to 4, 8, or 16. Note that the TCtraceIF is at the core interface boundary. The PIB is outside the core. Although cores from MIPS Technologies may include a sample PIB implementation, its design can be modified to suit the SOC vendor and the probe vendor. For example, whether or not a DDR memory interface is used on the ProbeIF is a decision made by the SOC vendor.

Figure 2.5 shows the configuration where the TCB is streaming data to an on-chip trace memory. The size of the on-chip trace memory is configurable. After trace capture has stopped, the trace data in the on-chip memory is accessed through the EJTAG probe by the Trace Regeneration Software.

Figure 2.5 Illustration of the Core and TCB with Internal Trace Memory



The TCB includes two primary interfaces:

- The TCB TAP interface, which connects the EJTAG TAP controller resident within the processor core to the TAP functionality present within the TCB.
- An optional TCtrace interface to the PIB. This interface is described along with the Probe IF in the core-specific document. If the TCB is configured with only on-chip trace memory, then the TCtrace IF and the PIB are not needed.

Chapter 3

PDtrace™ Description

A program executes sequentially through instructions within a basic block followed by a jump (or branch) to the head (first instruction) of the next basic block. To reconstruct the dynamic execution path of the program, it is sufficient to provide the post-analyzer with the PC address of the head of each basic block. Even this is not always necessary, since it may be possible in some instances to statically predict the value of the branch target, provided there is a separate indication for the taken branch. Hence, PC addresses need to be traced only when it is not possible to statically predict the branch target PC. For the MIPS32 and MIPS64 instruction sets, the statically unpredictable instructions are JR and JALR (for branch target address), and BEQ, BNE, BGEZ, etc. (for branch condition). Other statically unpredictable PC changes happen with taken exceptions and return from exceptions (ERET and DERET). To enable the post-analyzer to re-synchronize itself with the program execution, the PC value is also output at predictable intervals and synchronization periods.

The next sections of this chapter describe the various bits used in the output trace formats. This information indicates how tracing information is output and therefore is needed by the trace reconstruction software how to re-build the program execution for a user's benefit.

3.1 The Instruction Completion Indicator (InsComp)

Three bits are used as an indicator of completed instructions and their type in the processor's pipeline. Once tracing is initiated, a valid InsComp value is needed every cycle (except when the TCB has requested that the trace be stalled).

Table 3.1 Instruction Type Completion Indicator (InsComp)

Value	Mnemonic	Description
000	NI	No instruction completed this cycle. A "No Instruction" can happen due to a pipeline stall or when the instruction was killed (due to an exception).
001	I	Instruction completed this cycle
010	IL	Instruction completed this cycle was a load
011	IS	Instruction completed this cycle was a store
100	IPC	Instruction completed this cycle was a PC sync. The IPC value is used for the periodic output of the full PC value for synchronization. The tracing hardware should ensure that this is not done on an unpredictable branch, load, or store instruction.
101	IB	Instruction branched this cycle. The three encoding (101, 110, 111) for branched instruction indicates a discontinuity in the PC value for the associated instruction. Note that it is only when the new PC can not be predicted from the static program flow that it is traced.
110	ILB	Instruction branched this cycle was a load
111	ISB	Instruction branched this cycle was a store

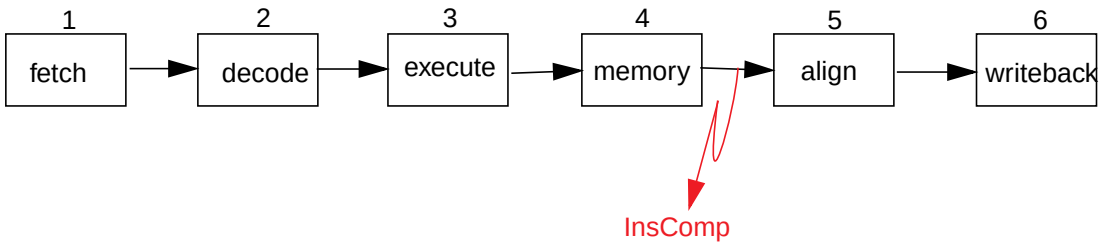
NI (No Instruction complete) is used when the internal pipe is stalled for some reason or the other, and no instruction completes in that cycle. It is also used when tracing has been turned off, but the internal FIFO is still emptying trace data out to the TCB that is data-related and not instruction-related, for example, data address or data values.

Instructions within a basic block are indicated with a I, IL, or IS value. The I is used to indicate a simple instruction that is neither a load nor a store. The IL is used to indicate a load instruction and the IS is used to indicate a store instruction.

Unpredictable (and predictable) changes in the PC value is indicated as a branch-type instruction, i.e., IB, ILB, or ISB. Note that the first instruction in the basic block is always indicated as a branch instruction. When this first instruction is a load or a store, then InsComp[2:0] takes values ILB or ISB respectively, to indicate the combined condition of the branch and load or store.

Implementation Notes: Figure 3.1 shows an example of when the InsComp value might be output by the processor tracing logic, with respect to the processor pipeline implementation. This example pipeline has six stages. They are: “fetch”, “decode”, “execute”, “memory”, “align”, and “write back”. In this example, the InsComp value is finalized after the memory stage. That is, the instruction goes through the pipeline and is captured after the last stage when the instruction **must** complete and can no longer be killed. In the example shown, this is after stage 4. This will differ, of course, with each pipeline implementation.

Figure 3.1 A Sample Pipeline And The InsComp Trace Point



Some instructions might have to provide more information for a complete picture of the program execution. For instance, a branch indicator might have to transmit the PC value if the unpredictability lies in the branch target address. If the unpredictability was in the branch condition (i.e., determining if the branch is taken or not), then the branch target PC value need not be transmitted; it suffices to indicate that it was a “taken” branch using the appropriate PDO_InsComp value.

The list below summarizes the three possible branching options, and the corresponding InsComp and PC tracing action:

- When the branch is unconditional and the branch target is predictable, IB, ILB, or ISB is used for the InsComp value, and the PC is not traced out.
- When the branch is conditional, and the branch target is predictable, IB, ILB, or ISB is used only when the branch is taken. The PC is not traced out.
- When the branch is conditional or unconditional, and the branch target is unpredictable, IB, ILB, or ISB is used and the PC is traced (using TPC for TType, to be discussed in section Section 3.2 “Trace Type and an Example Code Fragment”).

There are four possible circumstances that cause the value of the PC to be traced and they are listed here:

1. after a JR or JALR instruction.

2. after a control transfer to an exception handler.

3. after a return from exception (ERET or DERET instruction).

4. the PC is traced out periodically for software synchronization of trace with the static program image.

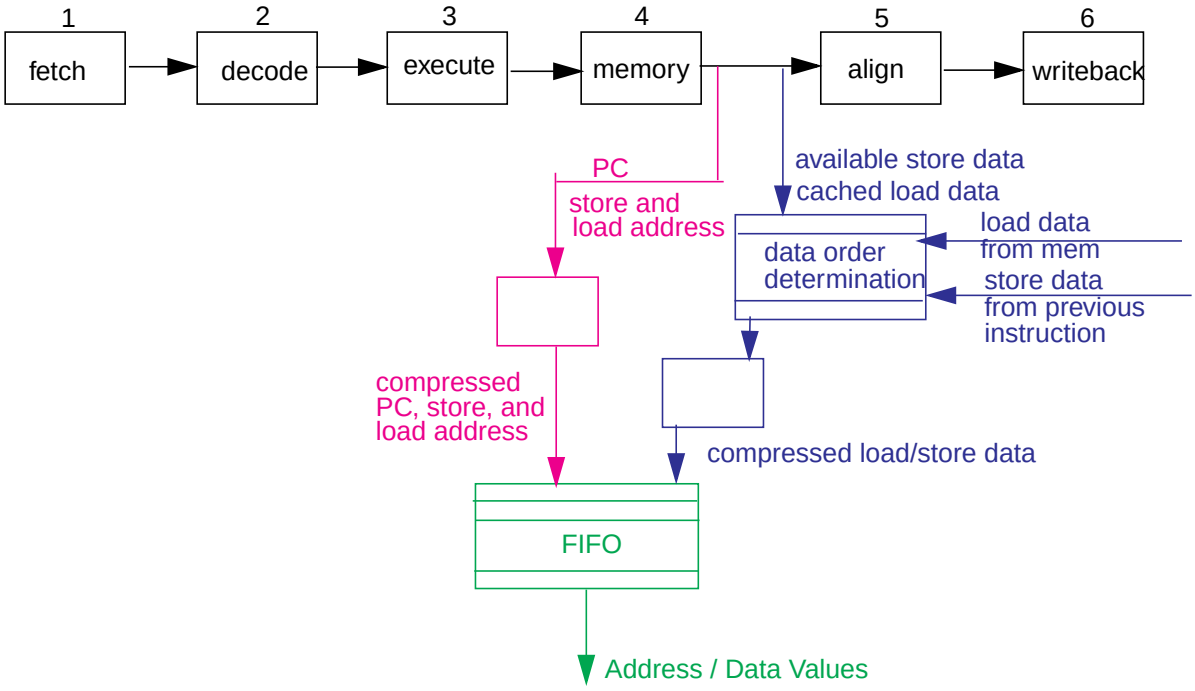
When the InsComp value indicates a store in the completing instruction with an IS, then the store address and data might have to be transmitted if the user requires these to be traced. With a ISB the PC value might also need to be traced out. In this situation, the PC value is sent first, followed by the store address, and finally the store data if it is immediately available.

An ILB is similar, and might require the tracing of the PC value as well as the load address and the load data. The PC value is sent first. If the load hits in the cache, then this works like the store described above, i.e., the PC value is sent, followed immediately by the load address and data.

The load or store data may not be immediately available. This can happen if the load misses in the cache and must be fetched from memory, or the store data is pending the completion of a previous (long latency) instruction that is computing the data value. In this situation, the load or store instruction is still indicated with the appropriate InsComp value of either IL, ILB, IS, or ISB. If the PC value needs to be traced, then it is traced first, followed by the load or store address, but the tracing of the corresponding data is deferred until it is available. While the processor is waiting for this data, other instructions may complete in the pipeline and are indicated by the appropriate InsComp values. When the data is available, it is traced out as soon as possible by the processor using the appropriate DataOrder value to indicate the out-of-orderness of the data (see 3.7 “Data Order Signal” on page 31).

Implementation Notes: Figure 3.2 shows, for the hypothetical pipeline, the points at which the different pieces of information could be tapped out to be traced. The PC value and the store address and load address are tapped out after stage 4. If the load hits in the primary cache, or the store data is available, then this information may be completely traced out at that point. If not, only the data’s address is sent and the data value is traced out when it becomes available.

Figure 3.2 Illustration of a Pipeline and Trace Tap Points



3.2 Trace Type and an Example Code Fragment

The TType[2:0] bits are used to indicate the type of information being traced.

Table 3.2 Trace Data Type Indicator (TType)

Value	Mnemonic	Description
000	NT	No data traced
001	TPC	Tracing the PC
010	TLA	Tracing the load address
011	TSA	Tracing the store address
100	TD	Tracing the load/store data value
101	TMOAS	Tracing the processor mode, the 8-bit ASID, and the SYNC bit. This is triggered by either a change in the processor mode, by a software write to the *EntryHi* register, or a trace synchronization operation. If the processor does not implement the standard TLB-based MMU, it is **UNPREDICTABLE** whether a write to the EntryHi register triggers a TMOAS operation. (See Figure 4.3).
110	TU1	Tracing the user-defined trace record - type 1
111	TU2	Tracing the user-defined trace record - type 2

An InsComp[2:0] value of IB, ILB, or ISB is traced when a branch instruction is taken, and the PC is traced this same cycle or later using a TType[2:0] value of TPC.

Implementation Notes: We will use Table 3.3 to illustrate these operation sequences. This table shows an example of a MIPS assembly fragment and the values of InsComp, TType, and TEnd that will be traced upon completion of each instruction of the code fragment in the pipeline. Assume that tracing was begun earlier, hence the start of tracing is not shown in this code fragment. The example also assumes a 32-bit processor and a 16-bit address/data trace width. This may imply that more than one type of a certain trace format is required to trace all out the address or data value bits if more than 16 bits are being traced. Hence, the TEnd bit is used to indicate the last format of a certain type needed to convey the same type of data. The trace formats are discussed later and they allow two widths of size 16 and 32 bits to be traced with a certain format type, as will be shown later.

As described earlier, a taken branch is always indicated with an IB value. But when the branch target address can be deduced from the static program image, then there is no accompanying TPC trace, that is the value of the current PC is not traced. An example of this can be seen in cycle 7, where the tracing of IB indicates the taken branch from the JAL instruction in cycle 5.

An example of an IB value traced for the InsComp value and accompanied by a corresponding TPC (to trace the statically unpredictable PC value), can be seen in cycle 10. This is triggered by the JR instruction in cycle 8. Cycle 10 is the branch target, also the first instruction of the new basic block. (Cycle 9 is the execution of the instruction in the branch delay slot). Note that the TPC trace could be directly started on cycle 10 since the implemented FIFO was empty.

The TEnd bit is used to indicate the end of any previously-started trace format. If the PC change value can be traced in a single cycle, then the TEnd bit may be traced in the same cycle as the TType value TPC. An example of this is seen in cycle 10. Otherwise, it may follow the required number of cycles later, for example in cycle 4, where it took 2 cycles to trace the store address value.

Note that at the processor's discretion, the TEnd bit may be used to cut off redundant sign bits from an address or data transmission. That is the tracing is curtailed and not all the upper bits of an address or data needs to be stored in trace memory. The reconstruction software must recognize this situation and sign-extend the address or data appropriately before use.

When a load instruction is executed, the InsComp value that indicates this is IL and ILB, and a store is indicated using IS and ISB. The user might have requested that load and store addresses (and data) be traced. In this situation, the load address and store address are traced using TLA or TSA respectively for the TType value.

Table 3.3 Example Code Fragment With Some PDtrace™ Trace Values

Cycle No.	PC	Instruction	InsComp[2:0]	TType[2:0]	TEnd
1	0x00400188	SW $6, 0xe170($1)	IS	TSA	1
2	0x0040018c	SW $4, 0xb134($28)	IS	TSA	1
3	0x00400190	SW $5, 0xb130($28)	IS	TSA	1
4	0x00400194	SW $0, 0x1c($29)	IS	TSA	0
5	0x00400198	JAL 0x418d9c	I	TSA	1
6	0x0040019c	OR $30, $0, $0	I	NT	x
7	0x00418d9c	NOP	IB	NT	x
8	0x00418da0	JR $31	I	NT	x
9	0x00418da4	NOP	I	NT	x

Table 3.3 Example Code Fragment With Some PDtrace™ Trace Values (Continued)

Cycle No.	PC	Instruction	InsComp[2:0]	TType[2:0]	TEnd
10	0x004001a0	JAL 0x411c40	IB	TPC	1
11	0x004001a4	NOP	I	NT	x
12	0x00411c40	JR $31	IB	NT	x
13	0x00411c44	NOP	I	NT	x
14	0x00414adc	LW $4, 0xb134($28)	ILB	TPC	0
15	0x00414ae0	BEQ $14, $0, 0x414af8	I	TPC	1
16	0x00414ae4	ADDIU $29, $29, 0xffe0	I	TLA	1
17	0x00414af8	OR $7, $0, $0	IB	TD	0
18	0x00414afc	NOP	IPC	TD	1
19	0x00414b00	ADDU $6, $6, $2	I	TMOAS	1
20	0x00414b04	OR $7, $2, $0	I	TPC	0
21	0x00414b08	SLTU $1, $2, $1	I	TPC	1

An example of store address tracing is seen in Table 3.3 at cycles 1, 2, 3, and 4. The store instruction in cycles 1, 2, and 3 take only 1 cycle to trace the store address. While the store address associated with the store in cycle 4 takes 2 cycles. (Perhaps it was not possible to compress the store address to less than 16 bits in this case). Note that in this case store data is not sent, only the store address is sent, as per the user request. If store data is also being traced, then the store data is sent immediately following the store address using a TD value for the TType bits. If the store data is not immediately available, it is sent later with the appropriate DataOrder value.

Assume that sometime between cycle 4 and cycle 14, the user changes the desired trace output, and wants load and store data to also be traced. Hence, the load instruction LW in cycle 14 will transmit not only the address, but also the associated data. Note that sometimes the load data is not immediately available since the load might miss in the first-level cache. In this situation, the load address is traced immediately and the load data is traced when it becomes available. The association of the load data with the corresponding load address is done using the DataOrder signal (not shown in the table).

The ILB in cycle 14 needs two cycles to trace the PC value, and then traces the load address using TLA in cycle 16. The load data is then traced using TD during cycles 17 and 18. The load must have hit the cache in this example, for otherwise, the associated load could have been separated from the instruction by an arbitrary number of cycles (required to satisfy the load miss from secondary memory).

An example of the periodic PC trace IPC for synchronization is shown in cycle 18. The required tracing for a synchronization includes sending a record of the process ASID and processor mode. This uses the TType[2:0] value of TMOAS, as seen in cycle 19 (traced as soon as the previous TD completes). This is followed by a tracing of the full PC value, which takes 2 cycles (cycles 20 and 21). As discussed in 3.5 “Trace Synchronization” on page 30, since load/store address tracing is turned on, the synchronization operation is not completed until a load and store full address trace is also sent (not shown in Table 3.3). A load or store address trace is always tied to a load or store instruction, respectively. The full load or store address is thus not sent until the next respective occurrence of a load or store instruction after the IPC trace.

The TMOAS trace is used to essentially track any modifications to the ASID and the processor mode. This tracking is enabled whenever tracing is on before the mode change takes place. If tracing is off when an ASID or mode change occurs, no mode transaction occurs. Figure 3.3 illustrates the bits that are traced in the right-most position for a TMOAS record.

Like other TType records, a TMOAS record can be split into two transactions in the trace formats, with the upper 16 bits sent in the second transaction. The first and the second (final) transaction can be defined using the TEND bit which is set to 0 in the first and to 1 in the second (refer to the section on TCB trace formats to understand the implication of TEND). Note that only the lower 16 bits of the TMOAS record are needed if the processor does not support multiple outstanding load instructions, hence an optimization on such processors would only send one TMOAS trace record with TEND set to 1.

Figure 3.3 A TMOAS Trace Record

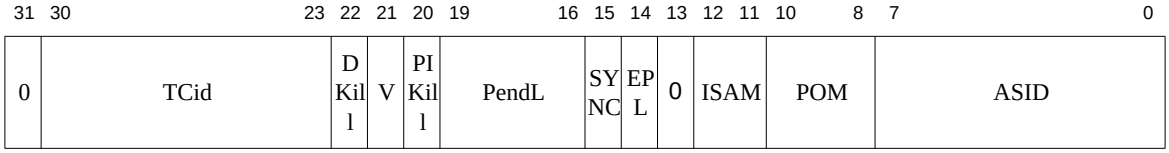


Table 3.4 A TMOAS Trace Record Field Descriptions

Fields		Description
Name	**Bits**	
TCid	30..23	Only required if the processor implements MT, otherwise reserved. Id of the TC that corresponds to the DKill signal assertion below.
DKill	22	Only required if the processor implements MT, otherwise reserved. When a ITC data instruction is killed for a given TC, this is indicated by asserting a TMOAS record with this bit set and a TCid value. When this bit is not set to one, then this indicates that no Data kill information is valid in this TMOAS record. **Implementation Notes:** Since a TC whose ITC data instructions was killed may not execute an instruction for a while, and data completion for other TCs may occur in the meantime, this TMOAS indication record is sent on an instruction that belongs to a different TC right after the late exception that killed the ITC instruction.
V	21	Only required if the processor implements MT, otherwise ignored. This bit determines whether or not only the DKill bits are valid in this TMOAS record or the entire TMOAS record is valid. That is, if V is 0, then all defined TMOAS bits are valid, and if V is 1, then only bits 30..22 are valid.
PIKill	20	Only required if the processor implements MT, otherwise ignored. This bit indicates that the instruction just previously traced was actually killed after it was traced. This scenario is possible in some situations where for example, an exception is taken after the ER stage of the ALU pipe. There are at least two cases to consider: • If an exception happens after ER when tracing a LW/SW accessing ITC memory in a core implementing MT. • If in an MT core, a TC is halted while executing Wait, Yield, or an instruction accessing ITC memory.

Table 3.4 A TMOAS Trace Record Field Descriptions

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>PendL</td><td>19:16</td><td>This field is valid only when SYNC is 1, see below. When SYNC is 1, this field indicates the number of outstanding loads at the IPC cycle. If the number of loads is zero, then all data transmissions TDs after that are ignored until the next load instruction, at which point counting is restarted. Such TD transmissions are from store instructions which could not complete before the IPC signal was sent.
Note that a sync happens with an InsComp value of IPC. Depending on whether or not there is data buffered up internally waiting to be sent out, the accompanying TMOAS may not be sent until several cycles later. In the meantime, any data sent in between the IPC and the TMOAS record may be ignored (at trace start or after an overflow) since this belongs to load and store instructions that happened before the sync. Now, if there are any load or store instructions between the IPC and the TMOAS, then the data for this will only be seen after the TMOAS is transmitted, since they would get buffered behind the TMOAS.</td></tr>
<tr><td>SYNC</td><td>15</td><td>When 0, this record was sent when the ASID, POM, or ISAM changed.
When 1, this record was sent for a synchronization event.</td></tr>
<tr><td>E PL</td><td>14</td><td>When 1, the PendL field is to be interpreted as (PendL + 16). When 0, the PendL field is interpreted by itself. This is introduced in PDtrace rev. 6.00</td></tr>
<tr><td>ISAM</td><td>12:11</td><td><table><tr><th>Value</th><th>In Architecture Mode</th></tr><tr><td>00</td><td>MIPS32</td></tr><tr><td>01</td><td>MIPS64</td></tr><tr><td>10</td><td>MIPS16e from MIPS32 mode</td></tr><tr><td>01</td><td>MIPS16e from MIPS64 mode</td></tr></table></td></tr>
<tr><td>POM</td><td>10:8</td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>000</td><td>Kernel Mode (EXL = 0, ERL = 0)</td></tr><tr><td>001</td><td>Exception Mode (EXL = 1, ERL = 0)</td></tr><tr><td>010</td><td>Exception Mode (EXL = don’t care, ERL = 1)</td></tr><tr><td>011</td><td>Debug Mode</td></tr><tr><td>100</td><td>Supervisor Mode</td></tr><tr><td>101</td><td>User Mode</td></tr><tr><td>110</td><td>Reserved</td></tr><tr><td>111</td><td>Reserved</td></tr></table></td></tr>
<tr><td>ASID</td><td>7:0</td><td>The ASID of the current process. If the processor does not implement the standard TLB-based MMU, this field is always traced as a zero because the EntryHi register, and hence the ASID, is not defined.</td></tr>
<tr><td>0</td><td>31,13</td><td>Reserved for future use</td></tr>
</table>

In addition to the TType values already discussed above, there are two, TU1 and TU2 which are used for user-triggered tracing. Whenever the user writes to a special register, the register values are traced out using one of the above respective TType values (depending on the exact register being written to).

3.3 Trace Mode

The TMode value is used to indicate the compression method used to transmit the address or data value. This is used by the external software to regenerate the program flow. The compression technique depends on the particular type of value being transmitted. A more detailed description is provided in Chapter 8, “Trace Compression” on page 97.

Table 3.5 Trace Mode Bits

TType	TMode
000 (NT) 101 (TMOAS)	Reserved
001 (TPC)	0 -> delta from last PC value 1 -> compression algorithm A (full address)
010 (TLA) 011 (TSA)	0 -> delta from last data address of that type 1 -> compression algorithm B (full address)
100 (TD) 110 (TU1) 111 (TU2)	0 -> Reserved 1 -> compression algorithm C (full data)

3.4 Start of Tracing

When tracing is first started, or when it is re-started after a break, some basic information is needed first to allow external software to identify the trace start point in the static program image, and make some reasonable conclusions about the processor mode at the start of tracing. The first record that is traced is a TMOAS. This trace record type shows the processor mode and the ASID value of the currently executing processor. This record is followed by a trace of the full PC value for the first instruction traced. This first traced instruction must use a IB, ILB, or ISB InsComp value so that the external software can correlate the PC transmission with the InsComp value. In addition, if load/store address tracing is turned on, then the first encountered load and store instruction will send the full address instead of a delta value. Note that the synchronization counter is reset to the value in $\text{TraceControl2}_{\text{SyP}}$ when tracing is started.

3.5 Trace Synchronization

Once the full PC value, or the full address for the load/store instruction has been sent during the start of tracing, subsequent traced addresses may all be delta values. Hence, it is possible that occasionally the external software will lose track of the current execution point in the static program image. To fix this potential problem, the tracing logic will send periodic synchronization information.

The synchronization tracing function is triggered when the internal synchronization counter overflows based on the synchronization period bits as set in the *TraceControl2* CP0 register. Similar to the start of tracing, when the synchronization period is reached, an IPC is sent for InsComp, accompanied by a TMOAS record, followed by a full PC value. To simplify this IPC transaction type, the hardware must ensure that the instruction used to synchronize the PC value is neither a branch, a load or a store instruction. Hence, the synchronization period is an approximate point, where the transmission of the IPC can be delayed by a few instructions until an instruction is found that is neither a

branch, load, or a store instruction. Note that the TMOAS associated with synchronization is sent only when the IPC instruction has been identified, to prevent other TType records between the TMOAS and the full PC trace for the synchronization. At this juncture, if load/store addresses are not being traced, then this completes all the transmissions needed for synchronization. If load/store addresses are being traced, then the first load and store instructions encountered after the IPC transmission trace a full address value, rather than a delta. This completes the synchronization process. Note that the synchronization counter is reset to the value in $\text{TraceControl2}_{\text{SyP}}$ once the IPC has been sent.

Note that the TMOAS record that is traced for synchronization uses a value of 1 for the SYNC bit field (see Figure 3.3). This is an aid used by external software to synchronize the InsComp stream and the data stream. To use this bit to synchronize, external software will look in the trace buffer for the first IPC entry, when it finds one, it starts looking in the trace buffer from the current cycle onwards for the first TMOAS record with the SYNC bit set to one. The first PC value following this TMOAS record will be a full PC transmission that corresponds to the IPC entry.

The TMOAS record also traces the number of outstanding loads and stores if data value tracing is underway. This ensures that in an out of order data return processor, the software using the record as a synchronization will know how many data values are still anticipated and count them correctly. See Figure 3.3.

3.6 Trace Overflow and Restart

In a real implementation, an internal FIFO or buffers may be used to hold address and data values waiting to be compressed, formatted, and traced out of the processor. It is possible to have a program sequence that overflows one or more of these FIFOs. In the situation that the FIFO overflows, the core is essentially losing trace data and hence the output gets illogical and no longer is a true representation of the program execution sequence. In this situation, the most natural thing for the core logic to do is abandon tracing in the current cycle, discard all entries in the FIFO, and restart tracing from the next completed instruction in the following cycle. Note that in this situation, the first new instruction to be traced after the overflow must have its full PC value, so this should be treated as a IB, ILB, or ISB. Similar to a trace start or re-start situation, a TMOAS record is first sent after the overflow, and before the full PC value is transmitted.

It should be possible for the entire program trace to be captured under all circumstances, and no trace records lost. This is done using the InhibitOverflow control bit from the program or the user debugger. When asserted, this bit implies that the processor core must back-pressure the pipeline and stall it without overflowing the FIFO. (Hence, if InhibitOverflow is asserted, the core must ensure that Overflow never gets asserted.) The pipeline is restarted as soon as the FIFO starts emptying again.

3.7 Data Order Signal

The data order bits DataOrder is used to indicate the out-of-order-ness of load and store data that is traced out. The main purpose of this is to allow load and store data to be traced out as and when it becomes available, and not maintain local storage that sequences it. This works by indicating the position of the traced load/store data in the list of current outstanding loads/stores starting at the oldest. For example, assume that the program issues 5 loads A, B, C, D, E, respectively.

Table 3.6 Load Order Example

Load	Cycle #	Cache Op	Load Data Available	Data Traced Out	DataOrder
A	1	Miss	-	-	-
B	2	Hit	B	B	1 (second oldest)
C	3	Hit	C	C	1 (second oldest)

Table 3.6 Load Order Example

Load	Cycle #	Cache Op	Load Data Available	Data Traced Out	DataOrder
D	4	Miss	-	-	-
E	5	Hit	E	E	2 (third oldest)
-	k	-	A	A	0 (oldest)
-	k+p	-	D	D	0 (oldest)

Table 3.6 shows an example of how these five loads may be traced. Load data that hits in the first-level cache is usually available at some fixed delay from instruction issue. So without loss of generality, we assume in the table that load data is available the same cycle as the issued instruction. The number of bits used to specify the DataOrder is processor implementation specific and depends on the number of possible outstanding loads and stores in that implementation. It is assumed that the default number for an implementation of a moderately complex processor is 4 bits, hence all the examples below use this value.

Table 3.7 Data Order with Four Bits

Value	Description
0000	data from oldest load/store instruction (is in-order)
0001	data from second-oldest load/store instruction
0010	data from third-oldest load/store instruction
0011	data from fourth-oldest load/store instruction
0100	data from fifth-oldest load/store instruction
0101	data from sixth-oldest load/store instruction
0110	data from seventh-oldest load/store instruction
0111	data from eighth-oldest load/store instruction
1000	data from ninth-oldest load/store instruction
1001	data from tenth-oldest load/store instruction
1010	data from eleventh-oldest load/store instruction
1011	data from twelfth-oldest load/store instruction
1100	data from thirteenth-oldest load/store instruction
1101	data from fourteenth-oldest load/store instruction
1110	data from fifteenth-oldest load/store instruction
1111	data from sixteenth-oldest load/store instruction

If the number of outstanding data supported by four bits is exceeded, then the processor simply issues the overflow signal, clears its internal buffers and restarts tracing. If the InhibitOverflow signal is asserted, then the processor must stall until at least some of the outstanding loads/stores are satisfied before continuing. Note that if data values are being traced, limits are being reached on other resources like the internal FIFO, and thus it is unlikely that the number-of-outstanding-data limit will be so easily reached.

Some processors will graduate a store instruction while still waiting for the store data to become available. Thus, a load can bypass a store and thus load data will be available before a preceding store's store data is available. An example is illustrated in Figure 3.4.

Figure 3.4 An Example of Load Data Bypassing an Earlier Store

(1)

Cycle	Program	InsComp	TType	Comments
m+0	DIV r3, r2	I	NT	multi-cycle instr
m+1	MFHI r1	I	NT	
m+2	SW r1, 0(r3)	ISa	TSAa	data not available
m+3	SW r4, 0(r7)	ISb	TSAb	data not available
m+4	LW r4, 0(r6)	ILc	TDb	store data
m+5	LW r5, 4(r6)	ILd	TLAc	cache hit
m+6			TDc	load data
m+7			TLAd	cache hit
m+8			TDd	load data
m+9+k			TDa	store data

(2)

Required Data Order	DataOrder
TDa	1
TDb	1
TDc	1
TDd	0

Block **(1)** in Figure 3.4 shows a small program fragment and the sequence of InsComp and TType traces. This processor will graduate and trace all instructions including the first store ISa. This store then waits for the data in r1 before it actually completes its execution. Some processors will order store data. Hence the second store ISb will wait for ISa before it can complete. But the following loads, ILc and ILd would complete without any delay. In this situation, the TType column of block **(1)** shows the sequence of data availability. But if the processor must trace data sequentially, then it is required to trace out data in-order as shown in the left column of block **(2)**. This sequential requirement can be avoided by using the DataOrder bits used to order both the loads and stores. The DataOrder values for the data is shown in the right column of block **(2)**.

Another example that illustrates the combined load/store ordering is shown in Figure 3.8. This table shows in column one, a sequence of only the loads and stores from a program fragment. The second column shows the sequence in which the data associated with the loads and stores become available, and the third column shows the DataOrder signal that is needed to trace out the sequence as available.

Table 3.8 Data (Load/Store) Order Example

Load/Store	Data Trace Order	DataOrder
Load-A	-	-
Load-B	-	-
Store-C	-	-

Table 3.8 Data (Load/Store) Order Example

Load/Store	Data Trace Order	DataOrder
Load-D	-	-
Store-E	-	-
Store-F	-	-
Store-G	-	-
Store-H	-	-
Load-I	I	8 (ninth oldest)
-	A	0 (oldest)
-	C	1 (second oldest)
-	E	2 (third oldest)
-	F	2 (third oldest)
-	G	2 (third oldest)
-	H	2 (third oldest)
-	B	0 (oldest)
-	D	0 (oldest)

3.8 Tracing During Processor Mode Changes

Note that during normal execution, the processor will change its operation mode frequently. For example, when executing user-level code, an interrupt may cause the processor to jump to kernel mode to service the interrupt. When the interrupt has been serviced, the processor will switch back to user mode. A mode change is indicated in the tracing logic by tracing out a TMOAS for TType.

In the situation that the mode change affects tracing, for example, the tracing system has been set up to trace only in user mode and not in kernel mode, then the interrupt service routine should not be traced. Upon jumping to kernel mode, the core tracing logic will add a TMOAS as the last record. In the meantime, all the accompanying InsComp values are traced as NI (No Instruction) until the TMOAS entry is traced. Once the TMOAS record has been output, nothing new is traced until execution jumps back into user mode. Note that pending information about outstanding loads and stores that were executed before the mode switch could still be traced. By knowing the static instruction stream in the user program, and using the TMOAS record, the external trace reconstruction software can figure out that tracing was suspended when the processor jumped to kernel mode.

When jumping from a non-tracing mode to a tracing mode, the first record output is TMOAS to indicate the mode change. This is followed by a full PC value of the first instruction in the tracing mode. This will enable the external trace reconstruction software to re-synchronize itself and track program execution in the desired mode.

When tracing is turned on and the processor enters Debug Mode where tracing is turned off, in the cycle-accurate tracing situation where every cycle is recorded including the ones where no instruction is executed, it is recommended that the processor turn off tracing as soon as it is detected that the DM bit is set. Otherwise, since it might take hundreds of cycles to fetch the first debug mode instruction through the TAP and the probe and execute it, the trace buffer will fill with records of idle cycles before the execution of the first debug instruction can be used to detect that tracing must be turned off. Thus, recording the entry into Debug mode as a processor mode change and then immediately stopping all tracing will prevent useful trace information in the trace buffer from being overwritten. In

this situation, in the presence of MIPS MT, it is recommended that all DM bits (in each VPE) be checked and when any one of them is set, tracing be immediately stopped when in cycle accurate mode).

3.9 Tracing Store Conditionals

A store conditional instruction part of an LL/SC operation may or may not perform the actual store operation. This section describes the tracing rules for such an instruction. A store conditional is always traced out as an IS or ISB for the InsComp value. If store address or data is being traced, then this associated information is traced as well. It is the responsibility of software to determine from context of the tracing and the program source whether the store conditional was successful or not. For typical uses of LL/SC pairs where the code executes in a loop until the SC succeeds, it should be easy to determine if the SC succeeded.

3.10 Tracing MIPS16e™ Macro Instructions

In the MIPS16e™ ASE, several single MIPS16e instructions expand to a fixed sequence of multiple 32-bit instructions. These include the SAVE and RESTORE and ASMACRO instructions. (See the *MIPS32® Architecture for Programmers Volume IV-a: The MIPS16e™ Application-Specific Extension to the MIPS32® Architecture*, document number MD00076).

When executing a Macro instruction, note that the PC address does not change for the instructions that comprise the macro instruction, hence the core does not output a PC value until it executes the first instruction outside the Macro. In fact, the core indicates the completion of the Macro instruction by outputting a full PC value for the first instruction executed after the macro instruction. This instruction could either have been reached sequentially by falling out of the macro sequence, or by executing a branch instruction from within the macro sequence. This full PC value is output using a branch indication e.g., IB for the InsComp value, even though this instruction is most likely not a branch target. The external re-construction software will note the preceding Macro instruction, and hence be able to handle this situation correctly.

Within the macro sequence, normal tracing is carried out. Note that the macro sequence can include, in addition to arithmetic and logical instructions, load and store instructions, which will be traced in a manner similar to loads/stores that are not in a macro instruction sequence. (Note that any branch instruction inside the Macro sequence can only branch out of the Macro sequence and not to any location within the sequence since all instructions within the sequence have the same PC value).

3.11 Tracing MIPS16e™ Extend Instructions

A MIPS16e extend instruction is considered a single instruction, and therefore the PC of the extend part is traced. Note that a branch to a MIPS16e extend instruction is to the extend part of the instruction. (For details, refer to the *MIPS32® Architecture for Programmers Volume IV-a: The MIPS16e™ Application-Specific Extension to the MIPS32® Architecture*, document number MD00076).

3.12 Tracing Instruction Cache and Data Cache Misses

With revision 4.0 of the specification, the PDtrace interface adds the feature where misses in the instruction cache and data cache are traced out by the Trace Control Block. This information is associated with the InsComp signal for instruction caches misses. The instruction cache miss and the data cache miss indicate a miss in the first level of the cache hierarchy. If instruction cache miss tracing is enabled, and PC tracing is disabled, the full PC of the instruction that missed in the cache must be traced.

Implementation Notes: In a processor that implements an instruction fetch buffer and does speculative execution and instruction prefetching, the instruction cache miss information may be duplicated for all instructions in a cache line, or the instruction cache miss could have been hidden from the back-end of the execution pipeline due to prefetching. And since tracing is done when an instruction graduates, at the back-end of the pipeline, the instruction cache miss statistics is not guaranteed to be 100% accurate. This problem is made worse in the presence of MIPS MT where due to frequently switching execution between thread contexts, cache miss information for a specific instruction of a specific thread context can be lost by the time that instruction is traced out.

Similarly, in an implementation it is possible that not all load/store misses might get traced due to certain constraints in how the pipeline timeline is implemented, or not all operations shown as misses might actually have been a miss. For example, in the MIPS32 24KE pipeline, a store operation that misses in the microTLB will show up as a hit in the output trace, whether or not this store actually missed in the cache. This is due to the fact that the store operation passes beyond the trace point in the pipeline before the cache miss or hit logic is done.

3.13 Tracing Potential Function Call/Return Instructions

To debug a program and understand its behavior, it is often sufficient to understand the execution call graph. When trace memory is limited, this can be an effective means for program debug for error or performance. To facilitate this feature, revision 4.0 of the PDtrace specification adds the ability for the processor to trace a potential function call instruction or a function return instruction.

Table 3.9 Possible Instructions for Function Call/Return s

ISA	Function Call Instructions	Function Return Instructions
MIPS32	JAL, JALR, JALR.HB, JALX	JR, JR.HB, ERET, DERET
MIPS64	JAL, JALR, JALR.HB	JR, JR.HB, ERET, DERET
MIPS16e	JAL, JALR, JALRC, JALX	JR, JRC

Note that the conditional procedure call instructions, BGEZAL, BGEZALL, BLTZAL, and BLTZALL are intentionally omitted from this list. Since executing these instructions does not automatically imply a procedure call, one would have to examine the PC trace to be certain whether or not a procedure call was invoked. When the TFCR bit is set and PC Tracing is enabled, the FCR bit should be set in the trace format used for the function call/return instruction. When PC Tracing is disabled, in addition to the FCR bit being set for the function call/return instruction, the full PC of the function call/return instruction must be traced.

3.14 Tracing with MIPS® MT ASE

The MIPS MT ASE adds several VPEs (Virtual Processing Elements) and TC (Thread Contexts). See the MIPS MT ASE specification for details. When tracing is in progress, the trace data needs to be qualified with the VPE and the TC number of the instruction being traced, for effective program debug and analysis.

The user via the debugger can request that only instructions from a particular VPE or TC be traced. The analogous function in the CP0 trace control register is provided via the TCNum and CPUId bits.

On the trace output, if MT is present on the processor, then every instruction traced is qualified with the TC identification. Software can tell from the TCid what VPE it belongs to by reading the appropriate MT CP0 registers. Each PC, address, and data delta computation is done on a per-TC basis. The processor is therefore expected to maintain per-TC delta values. The first time that a PC is traced for a thread, the full address is traced. This initiates the process whereby future instructions from that thread are done using delta PC values.

To clarify this further, each thread of instruction trace is independent, hence must carry its own output of the TMOAS record. That is, whenever trace is initiated for a new thread, a TMOAS is required for that thread. This is because each thread can have its own processor mode such as MIPS32 or MIPS16e, and this needs to be indicated correctly for that thread. In other words, every time trace is re-started for a thread, perhaps because of a FIFO overflow, a separate TMOAS is required on a per thread basis. A sync operation also requires a TMOAS on a per TC basis. If tracing is being initiated only for a single TC, then only a single TMOAS is required. But if tracing is being initiated for multiple TCs or per VPE, then separate TMOAS records are required per TC as described above. Also note that when multiple TCs are being traced, and a TMOAS is sent for the first TC after a sync, then the sync counter is restarted even though all TMOAS for other TCs have not yet been sent. Note that TMOAS is also sent on a TC restart, that is, a write to the TC restart register.

3.15 Tracing in WAIT State

A processor enters a WAIT or sleep state when transitioning to a low-power mode. In this situation, since the processor is not executing any instructions or doing any useful work, it is not necessary to continue tracing, which can be an unnecessary drain on power.

3.16 Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints

The MIPS EJTAG Specification describes the hardware data and instruction breakpoint feature. In brief, a core or processor can optionally implement up to 15 instruction and up to 15 data EJTAG hardware breakpoints. These breakpoints, when encountered during program execution, cause the processor to take a debug exception. Important to this discussion is that a bit (TE bit 2) in the breakpoint control register, when set, allows a trigger signal to be generated (instead of, or in addition to, causing a debug exception). The PC/Data tracing interface uses this trigger signal to trigger trace on/off and to enable other tracing modes. When a trigger is generated, this information is traced into the trace memory so the trace software can have knowledge of when trace triggers were generated.

3.16.1 The *TraceIBPC* and *TraceDBPC* Registers

Whether a particular hardware breakpoint triggers any action in the tracing logic is determined by bits in the *TraceIBPC* (Trace Instruction Break Point Control) register and the *TraceDBPC* (Trace Data Break Point Control) register. Each register uses 3 bits per breakpoint. A few bits are added for other information, hence each register hold information for 9 hardware breakpoints. If more than 9 breakpoints are implemented for either instructions or data, then another register is required. This is either *TraceIBPC2* or *TraceDBPC2*, for instruction or data. and these registers are available as memory-mapped registers in the EJTAG memory drseg space as shown in Table 3.10. PDtrace

Table 3.10 Mapping Trace Breakpoint Registers in CP0 Space or in drseg

CP0 register number/select or Offset in drseg	Register Mnemonic	Description
Register 23, Select 4	TraceIBPC	Holds information about the first 9 instruction breakpoints.
Register 23, Select 5	TraceDBPC	Holds information about the first 9 data breakpoints.
0x1FF8	TraceIBPC2	Holds information about the last 6 instruction breakpoints.
0x2FF8	TraceDBPC2	Holds information about the last 6 data breakpoints.

revision 05.00 and higher add the ability to simultaneously trigger tracing from other components in a coherent system, like the coherence manager in the MIPS 1004K core (Table 3.15 defines how tracing can be triggered in system

components). If a trigger that is set to enable CM tracing fires, the corresponding Core_CM_EN bit in TCBControlD is set to one. Similarly, if a trigger that is set to disable tracing fires on a core, the Core_CM_EN bit is set to zero.

The EJTAG control logic, upon encountering a hardware breakpoint, will signal the triggered breakpoint to the trace logic. If more than one breakpoint triggers every cycle, in the previous revision of the specification, even if one of the triggers turned on trace, then the trace turned on, and all triggers have to turn trace off to turn off tracing. Now, the possible trace modes generated by the triggers are more complex, hence if more than one trigger is generated in any given cycle, and there is at least one trigger from the Instruction side and one trigger from the data side, then the Data trigger is ignored. If there are multiple triggers, and all are either Instruction triggers or all are Data triggers, then all except the lowest numbered one are ignored.

The type of tracing that is triggered is determined by the tracing mode Mode[27:23] in the TCBCONTROLC register, or if in software control, by the Mode[11:7] bits in the *TraceControl2* register (described in "The TraceControl2 Register (CP0 Register 23, Select 2)" on page 53 of this document).

Note that it is possible for the tracing mechanism to globally disable the hardware breakpoint-enabled triggering of tracing using the disable bit in the *TraceIBPC* or the *TraceDBPC* register. One bit is used to disable instruction breakpoints, and the other is used to disable data breakpoints, as shown in Table 3.5 and Table 3.11.

It is possible that PDtrace tracing logic is implemented with no EJTAG implementation. Thus, it is the responsibility of (external or internal) software to read the Coprocessor 0 Config1 register to determine if EJTAG is implemented before assuming the presence of the *TraceDBPC* or *TraceIBPC* registers. Moreover, the EJTAG hardware breakpoints are optional for a core implementing EJTAG. The Debug Control Register (at offset 0x0000 in drseg) has bits DataBrk and InstBrk that specify whether any EJTAG data or instruction hardware breakpoints are implemented. If both these bits are set to 0, then no hardware breakpoints are implemented in EJTAG on that core, and the trace register specified in this section is also not implemented, i.e., the tracing logic does not implement the feature of trace triggering from EJTAG. Thus, one must first ensure that EJTAG is implemented, then examine the values of DataBrk and InstBrk in the Debug Control register, and ensure that at least one of them is not zero, before assuming the presence of the *TraceDBPC* or *TraceIBPC* registers.

Moreover, in a processor implementing the MIPS MT ASE, the EJTAG breakpoints can either be shared or not shared between any of the VPE (Virtual Processor Elements) in an MT environment. This sharing or not-sharing property for instruction breakpoint, if they exist, is determined by bit IBPshare of the IBS (Instruction Breakpoint Status) register located in EJTAG drseg memory. Similarly, the sharing or not sharing property for data breakpoints, if they exist is determined by bit DBPshare bit in the DBS (Data Breakpoint Status) register located in EJTAG drseg memory. If the breakpoints are not shared, then the TraceIBPC or the TraceDBPC registers are duplicated per VPE, otherwise, they are shared. When they are shared, the IE or the DE bit, see Table 3.11 and Table 3.12, are also shared, hence the breakpoints are enabled for PDtrace either for all VPEs or for none of them.

Figure 3.5 *TraceIBPC* Register Format

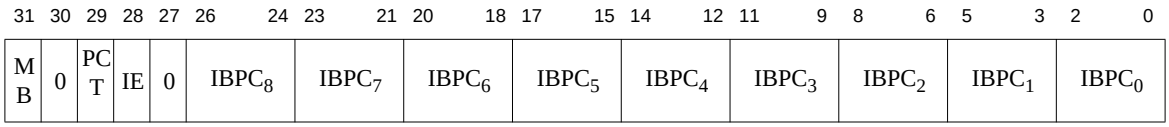


Table 3.11 *TraceIBPC* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
MB	31	Indicates that more instruction hardware breakpoints are present and register TraceIBPC2 should be used.	R	0/1	Required

Table 3.11 *TraceIBPC* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
0	30	Reserved. Reads as zero, and non-writable	R	0	Required
PCT	29	Specifies whether or not a performance counter trigger (PCT) signal is generated when an EJTAG hardware instruction breakpoint match occurs. This feature is enabled only if the IE bit is also set to 1.	R/W	0	Required after PDtrace revision 06.00 and higher if instruction breakpoints are implemented in EJTAG. Reserved otherwise.
IE	28	Used to specify whether the trigger signal from EJTAG instruction breakpoint should trigger tracing functions or not: 0 : disables trigger signals from instruction breakpoints 1 : enables trigger signals from instruction breakpoints Writes to this bit are ignored if instruction breakpoints are not implemented in EJTAG.	R/W	0	Required
0	27	The previously defined use of this bit is deprecated since values 2 through 7 of the trigger control bits have been taken over to support CMP tracing and performance counter tracing. It now reverts back to being reserved. Reads as zero, and non-writable.	R	0	Required
IBPCn	3n-1:3n-3	The three bits are decoded to enable different tracing modes. Table 3.15 shows the possible interpretations. Each set of 3 bits represents the encoding for the instruction breakpoint n in the EJTAG implementation, if it exists. If the breakpoint does not exist then the bits are reserved, read as zero and writes are ignored.	R/W	0	LSB required, Upper two bits are Optional. Required for breakpoints implemented in EJTAG

Figure 3.6 *TraceDBPC* Register Format

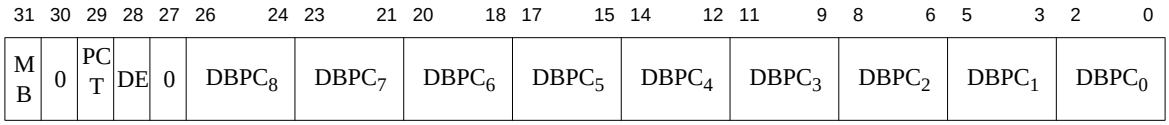


Table 3.12 *TraceDBPC* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
MB	31	Indicates that more instruction hardware breakpoints are present and register TraceDBPC2 should be used.	R	0/1	Required

Table 3.12 *TraceDBPC* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
0	30	Reserved. Reads as zero, and non-writable	R	0	Required
PCT	29	Specifies whether or not a performance counter trigger (PCT) signal is generated when an EJTAG hardware data breakpoint match occurs. This feature is enabled only if the DE bit is also set to 1.	R/W	0	Required after PDtrace revision 06.00 and higher if instruction breakpoints are implemented in EJTAG. Reserved otherwise
DE	28	Used to specify whether the trigger signal from EJTAG data breakpoint should trigger tracing functions or not: 0 : disables trigger signals from data breakpoints 1 : enables trigger signals from data breakpoints Writes to this bit are ignored if data breakpoints are not implemented in EJTAG.	R/W	0	Required
0	27	The previously defined use of this bit is deprecated since values 2 through 7 of the trigger control bits have been taken over to support CMP tracing and performance counter tracing. It now reverts back to being reserved. Reads as zero, and non-writable.	R	0	Required
DBPCn	3n-1:3n-3	The three bits are decoded to enable different tracing modes. Table 3.15 shows the possible interpretations. Each set of 3 bits represents the encoding for the data breakpoint n in the EJTAG implementation, if it exists. If the breakpoint does not exist then the bits are reserved, read as zero and writes are ignored.	R/W	0	LSB required, Upper two bits are Optional. Required for breakpoints implemented in EJTAG

Figure 3.7 *TraceIBPC2* Register Format

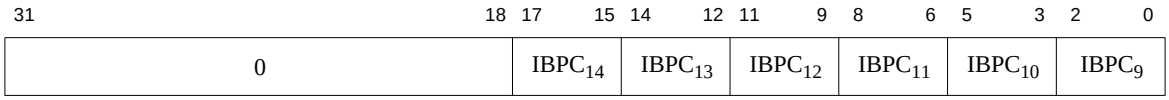


Table 3.13 *TraceIBPC2* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
IBPCn	3n-31:3n-33	The three bits are decoded to enable different tracing modes. Table 3.15 shows the possible interpretations. Each set of 3 bits represents the encoding for the instruction breakpoint n in the EJTAG implementation, if it exists.	R/W	0	Required

Figure 3.8 *TraceDBPC2* Register Format

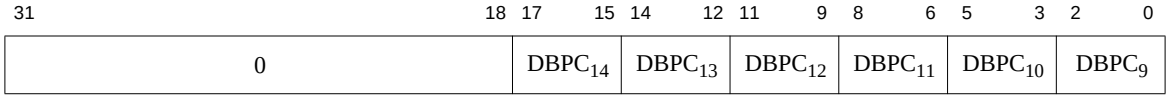


Table 3.14 *TraceDBPC2* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
DBPCn	3n-31:3n-33	The three bits are decoded to enable different tracing modes. Table 3.15 shows the possible interpretations. Each set of 3 bits represents the encoding for the data breakpoint n in the EJTAG implementation, if it exists.	R/W	0	Required

Table 3.15 BreakPoint Control Modes: IBPC and DBPC

Value	Trigger Action	Description
000	Unconditional Trace Stop	Unconditionally stop tracing (from the processor as well as the coherence components that support tracing) if tracing was turned on. If tracing is already off, then there is no effect. Tracing of the coherence components is tied to the processor tracing since otherwise the trace information might not be interpretable by external software if it not being matched or synced to the program execution trace.
001	Unconditional Trace Start in the processor	Unconditionally start tracing if tracing was turned off. If tracing is already turned off then there is no effect.
010	None	Reserved for future use
011	Unconditional Trace Start for both the processor and the components of the coherence system	Unconditionally start tracing if tracing was turned off. If tracing is already turned on then there is no effect. Do this for both the processor as well as the coherence components that support tracing.

Table 3.15 BreakPoint Control Modes: IBPC and DBPC

Value	Trigger Action	Description
100	Identical to trigger condition 000, and in addition, also dump the full performance counter values into the trace stream	As before, but also dump the full values of all the implemented performance counters into the trace stream. Note that this does not provide the ability to dump individual and/or specific performance counters for two reasons. One, there aren't sufficient bits available for this type of fine-grain control, and secondly, performance counter dumping on breakpoint trigger should be uncommon enough to not overwhelm the trace stream with bits.
101	Identical to trigger condition 001, and in addition, also dump the full performance counter values into the trace stream	As before, but also dump the full values of all the implemented performance counters into the trace stream.
110	None	Reserved for future use
111	Identical to trigger condition 011, and in addition, also dump the full performance counter values into the trace stream	As before, but also dump the full values of all the implemented performance counters into the trace stream.

3.17 Tracing Performance Counter Values

Dumping performance counter values through the trace stream provides the ability to correlate performance counter events to the specific instruction execution path. In fact, this provides is a non-intrusive read out of the performance counter values that does not alter execution behavior. In addition, with the right mechanisms in place, it can allow the ability to dynamically change the granularity of reading out the counter values without requiring recompilation of user code. For example, if a particular type of stall is suspected to be high in a particular function, that function can be traced individually, and the performance counter set to detect those stalls and dumped out periodically, thereby allowing a better correlation of that stall type to particular code blocks within that function to narrow down the performance bottleneck.

The performance counter trace feature has been defined in specification revision 06.00 and higher. A bit called PeC (bit 8 in *TraceControl3* and bit 0 in TCBControlE) defines whether or not this optional feature is implemented. Another single bit PeCE (bit 9 in *TraceControl3* and bit 1 in TCBControlE) indicates during program execution whether or not the feature is enabled. As before, the bit in the TCB register is used by external probe-based debugging and trace control, and the bit in *TraceControl3* is used when software-controlled tracing is in effect. Four other bits (bits 9, 10, 11, and 12 in *TraceControl3* register and bits 2, 3, 4, and 5 in TCBControlE register) are used to enable the specific events that can trigger a dump out of the performance counter values. These four events are:

1. Synchronization counter expiration trigger (PeCSync)
2. Hardware trace breakpoint trigger (PeCBP)
3. Function call, function return, or exception occurrence trigger (PeCFCR)
4. When any active performance counter in the processor overflows (PeCOvf)

Bit 15 PCTD (Performance Counter Trace Disable) in the Performance Counter Control Register is used to provide more detailed control over whether or not a particular performance counter value should be dumped with all the others. This bit is used to disable the specified performance counter from being traced when performance counter trace is

enabled and a performance counter trace event is triggered. Note that the reset and default value for this bit is 0, which enables tracing of this performance counter. Software must explicitly set this bit to 1 to disable tracing this counter value.

3.18 Filtered Data Trace Mode

This mode is added in PDtrace Specification revision 06.00 and higher. Bit 0 in TraceControl3 register (FDT bit) is used to either disable (value 0) or enable (value 1) this mode. In this mode, data load and store addresses are compared to the hardware data breakpoint address, if the addresses match, the data value associated with that match along with the address are traced out.

This mode works even when data address and/or value tracing is turned on. However, the general usage model is when both PC and Data trace are turned off since it may not always be possible to identify data that was traced due to a match vs. the regular data stream. This mode is used to shadow one or more static (fixed-address) variables. When there is a store to the variable, the store value is captured into the trace. Since there are generally two or more data triggers/watchpoints, the trace will need to uniquely identify the shadowed variable by also tracing out the associated address.

A main use of this filtered data trace is to support tracing of events in an application code on a Linux system. This type of instrumented code tracing is primarily used for performance analysis although it can also be used for event logging and debug. (This mode has been introduced to provide a mechanism to do low overhead event tracing from user application code since the User Trace Data registers require a kernel call from user mode.)

Another potential use of this mechanism is to set a watchpoint and track values written to an I/O or peripheral register. Off-chip trace probes can timestamp these values, thus providing valuable performance information on the delta between writes, assuming this was the intended use.

3.19 Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM

PDtrace specification revision 06.00 and higher provides mechanisms by which PDtrace can be controlled entirely through software and the on-chip trace memory can be accessed directly by software using load and store instructions. Access is provided by mapping most TCB registers to DRSEG space, which then allows software to access these registers in debug mode. Since all TCB registers are mapped directly to drseg, the TCBData register does not need to be mapped.

The mapped DRSEG registers are shown in Table 3.16. These mappings are “active” only when an external probe is

Table 3.16 Mapping TCB Registers in drseg

Offset in drseg	Register Name	Description
0x3000	*TCBControlA*	The TCBControlA register. See Section 5.1 “TCBCONTROLA Register” for more details about register contents.
0x3008	*TCBControlB*	The TCBControlB register. See Section 5.2 “TCBCONTROLB Register” for more details about register contents.
0x3010	*TCBControlC*	The TCBControlC register. See Section 5.3 “TCBCONTROLC Register” for more details about register contents.
0x3018	*TCBControlD*	The TCBControlD register. See Section 5.4 “TCBControlD Register” for more details about register contents.
0x3020	*TCBControlE*	The TCBControlE register. See Section 5.5 “TCBCONTROLE Register” for more details about register contents.
0x3028	*TCBConfig*	The TCBConfig register. See Section 5.7 “TCBCONFIG Register (Reg 0)” for more details about register contents.
0x3100	*TCBTW*	Trace Word read register. This register holds the Trace Word just read from on-line trace memory. See Section 5.8 “TCBTW Register (Reg 4)” for more details about register contents.
0x3108	*TCBRDP*	Trace Word Read pointer. It points to the location in the on-line trace memory where the next Trace Word will be read. A TW read has the side-effect of post-incrementing this register value to point to the next TW location. (A maximum value wraps the address around to the beginning of the trace memory). See Section 5.9 “TCBRDP Register (Reg 5)” for more details about register contents.
0x3110	*TCBWRP*	Trace Word Write pointer. It points to the location in the on-line trace memory where the next new Trace Word will be written. See Section 5.10 “TCBWRP Register (Reg 6)” for more details about register contents.
0x3118	*TCBSTP*	Trace Word Start Pointer. It points to the location of the oldest TW in the on-chip trace memory. See Section 5.11 “TCBSTP Register (Reg 7)” for more details about register contents.
0x3120	*BKUPRDP*	This is not a TCB register, but needed on a reset to save the TCBRDP value before that register is reset to 0. This allows the software that comes up after a (hard or soft) reset to know the last-known good value of TCBRDP before system crash, and potentially read the trace memory from or to the appropriate trace memory location.
0x3128	*BKUPWRP*	This is not a TCB register, but needed on a reset to save the TCBWRP value before that register is reset to 0. This allows the software that comes up after a (hard or soft) reset to know the last-known good value of TCBWRP before system crash, and potentially read the trace memory from or to the appropriate trace memory location.
0x3130	*BKUPSTP*	This is not a TCB register, but needed on a reset to save the TCBSTP value before that register is reset to 0. This allows the software that comes up after a (hard or soft) reset to know the last-known good value of TCBSTP before system crash, and potentially read the trace memory from or to the appropriate trace memory location.
0x3200-0x3238	*TCBTrigX*	The TCBTrigX set of registers. The number of implemented registers is determined by the value in $TCBCONFIG_{TRIG}$. See Section 5.12 “TCBTRIGx Register (Reg 16-23)” for more details about register contents.

either not present, or not enabled (i.e., the ProbEN bit in the EJTAG Control Register or ECR is set to zero). If the

mappings are active, writes to the TCB registers via drseg are enabled (so long as these writes are otherwise permitted). If the mappings are inactive, writes to the TCB registers via drseg are ignored. Note that a hardware probe could set the ProbEN bit to zero and still access the TCBControl registers. Writing the TCB registers via the probe and drseg simultaneously will result in unpredictable behavior. Software should not rely on reads from the TCB registers via drseg to return reliable data when the mappings are inactive. If the mappings are active on reset (i.e., ProbEN=0), software is responsible for initializing all control register fields, except for TCBControlA.On (bit 0 in TCBControlA) and TCBControlB.EN (bit 0 in TCBControlB). Those control bits must be set to zero on a core reset if the drseg mappings are active.

On chip trace memory can be read by doing a load instruction to TCBTW. On a 32-bit core, two load instructions must be executed to load a 64-bit trace word. These load instructions must target two different addresses. The first must target an offset of (+4) from the TCBTW register, and the second load instruction must target the TCBTW register. Accessing the TCBTW has the side effect of automatically incrementing the value of TCBRDP to the next trace word. The trace memory cannot be written to via this mechanism. Software can also do direct loads and stores to TCBRDP and TCBWRP at the beginning of the trace memory dump function. Note that writing to these registers in the middle of the trace logic writing into this memory can result in UNPREDICTABLE results and junked values in the trace memory.

Whether or not software has access to on-chip trace memory is controlled via one bit in TCBCONTROLB (TRPAD, bit 18). This is a control DISABLE bit. The bit in TCBCONTROLB is mirrored in *TraceControl3*.

Tracing is stopped when the system crashes and an exception handler is invoked on the crash. The last known good values of TCBRDP, TCBWRP, and TCBSTP are saved in the backup registers shown in the table. Software should not rely on TCBRDP, TCBWRP and TCBSTP holding their last known good values across a reset, and should use the backup registers for this purpose.

3.20 Trace Enabling/Disabling Condition

There are several input control values to the core that enable tracing, this can be from the TCB registers or from the CP0 control registers. In addition, trace can also be triggered on and off by the EJTAG hardware instruction and data breakpoint settings, as seen in 3.16 "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 37. The equations specified here clarify the conditions under which different input factors will enable or disable tracing.

Trace enabling and disabling from software is similar to the hardware method, with the exception that the bits in the control register are used instead of the input enable signals from the TCB. The TraceControl_{TS} bit controls whether hardware (via the TCB), or software (via the *TraceControl* register) controls tracing functionality.

In any given cycle n, an instruction is traced if the following equation evaluates true:

TraceOn & (TriggerOn(n) | MatchEnable | TriggerEnable | DQEnable | FilterDataTraceActive) (EQ 1)

And every cycle, the following state variable is set and then used in the next cycle:

TriggerOn(n+1) <- TraceOn & (TriggerEnable | (TriggerOn(n) & (~TriggerDisable)) (EQ 2)

The various expressions used in (EQ 1) and (EQ 2) are defined here.

$\text{TraceOn} \leftarrow ((\text{TraceControl}_{TS}\ \&\ \text{TraceControl}_{On}) \mid ((\sim\text{TraceControl}_{TS})\ \&\ \text{TCBCONTROLA}_{On}))$

$\text{MatchEnable} \leftarrow (\text{TraceControl}_{TS}\ \&$

```
 MTEnableR                                                                    &
 (TraceControl_G | (( (TraceControl_ASID ^ EntryHi_ASID) & (~TraceControl_ASID_M)) =0 )) &
 ((TraceControl_U & UserMode)        |
  (TraceControl_K & KernelMode)      |
  (TraceControl_S & SupervisorMode)  |
  (TraceControl_E & ExceptionMode)   |
  (TraceControl_D & DebugMode)))              |
((not TraceControl_TS)                      &
 MTEnable                                    &
 (TCBCONTROLA_G or (TCBCONTROLA_ASID = EntryHi_ASID))      &
 ((TCBCONTROLA_U & UserMode)        |
  (TCBCONTROLA_K & KernelMode)      |
  (TCBCONTROLA_S & SupervisorMode)  |
  (TCBCONTROLA_E & ExceptionMode)   |
  (TCBCONTROLA_D & DebugMode)))

MTEnableR ← (MT ASE not present) |
            ((MT ASE present) &
              ((!TCV) |
               (TCV & TCNum=current_TC_context) |
               (!VPEV) |
                (VPEV & VPENum=current_VPE)))

MTEnable ← (MT ASE not present) |
           ((MT ASE present) &
             ((!TCBCONTROLC_TCValid) |
              (TCBCONTROLC_TCnum & TCNum=current_TC_context) |
              (!TCBCONTROLC_CPUValid) |
               (TCBCONTROLC_CPUValid & VPENum=current_VPE)))

TriggerEnable ← Lowest Numbered Trigger turns on tracing either because the DBPC
or IBPC value was 001, 011, 101, or 111

TriggerDisable ← Lowest Numbered Trigger turns off tracing because the DBPC or
IBPC value was 000 or 100

FilterDataTraceActive ← TraceControl3_FDT &
                        (Load_Address_Matches_Hardware_Breakpoint_Address |
                          Store_AddresS_Matches_Hardware_Breakpoint_Address)
```

As seen in the (EQ 1), trace can be turned on only if the master switch On or PDI_TraceOn is first asserted (TraceOn). Once asserted, there are three ways in which instruction tracing can occur:

1. A trigger had occurred in the past that turned on tracing, but no trace disabling trigger had occurred since then (TriggerOn(n)).

2. The input enable signals from the TCB or the trace control register indicate a tracing condition (MatchEnable). This type of tracing is done over general program areas. For example, all of user-level code for a particular process (ASID specified), or some such conditions.

3. The third method to turn on tracing is from the processor side using the EJTAG hardware breakpoint triggers (TriggerEnable). If EJTAG is implemented, and hardware breakpoints can be set, then using this method, fine grain tracing control is possible. It is possible to send a trigger signal that turns on tracing at a particular instruction. For example, it would be possible to trace a single procedure in a program by triggering on trace at the first instruction, and triggering off trace at the last instruction.

Trace is turned off when (EQ 1) evaluates false. Note that tracing can be unilaterally turned off by de-asserting the On bit or the PDI_TraceOn signal.

Chapter 4

PDtrace™ Control Using CP0 Registers

Obviously, a lot of trace information can be obtained from the core, so the user can decide to a certain extent what should be traced at a given time so that the trace bandwidth can be minimized to the useful set. Obviously, the PDtrace architecture has some predetermined capabilities and the user is constrained to operate within these constraints. This chapter therefore detailed both the predetermined constraints of what can be traced using the mechanics of how this trace information may be controlled.

The user has two types of control. The first uses CP0 registers that can be set by the executing program. This allows for statically determined trace capability on the part of a program. For example, if the user wants to trace a program only if it enters a certain section of code, then at the top of this section of code, there are instructions to set the trace control CP0 registers to begin tracing. And at the end of this section of code, the program will reset the CP0 registers appropriately to stop tracing if that is what is required. This chapter is a discussion of this CP0 control of tracing

A second way to control trace output is via the registers in the Trace Control Block (TCB) that can be set by an external agent using an external probe hooked to the EJTAG TAP controller port of the core or via drseg register mappings as of revision 6.00 of the PDtrace architecture. This allows a user who is operating a debugger to interactively decide during the dynamic execution of the program whether or not to start and stop tracing. A discussion of the TCB and its registers is available in the next chapter.

As already noted, the tracing mechanism can be controlled either by hardware via the input signals from the external trace block or controlled by software. Software control is possible via bits in a Coprocessor 0 register. There is one extra bit in the CP0 register used to select control between the hardware and software mechanisms. The reset value of this bit selects hardware tracing control. If software wants to take over tracing, it can set all the tracing control bits in the Coprocessor 0 register to the desired value and then set the select bit to transfer tracing control to the bits in the register.

4.1 Trace Controls Overview

The majority of the trace control bits are used to specify the conditions under which tracing is to be enabled. The list below briefly explains the various types of trace controls:

- An overall trace control bit $TraceControl_{On}$ controls whether tracing is turned on or not. If this bit is asserted, then the control bits that control the per instruction decision of whether the core should trace or not include bits in *TraceControl* such as G, ASID, U, S, K, E, and D. These bits are expected to be modified only when the processor is not tracing. That is, if tracing is currently on, then tracing must be turned off, a change made to one of these bits and then tracing turned back on again. If not done this way, then the trace output obtained from the TCB is not guaranteed to be parse-able by the reconstruction software.

  For processors that implement MIPS MT ASE or in a multi-processor configuration, there are other control bits such as TCNum and CPUid that control which thread context or VPE (virtual processing element) or cpu in the configuration is currently tracing. The same rule as above applies here with respect to changing the control bits only when tracing is turned off.

- When tracing is turned on, one needs to specify what kind of information is to be traced, i.e., just the PC, or also the load/store addresses and data. This is done using the Mode bits in *TraceControl2*. In addition to this, another signal, $TraceControl_{TB}$ asks that the PC of all taken branches be traced, not just the ones that are statically unpredictable. When asserted, this will generate a lot of trace data, since in a RISC architecture like MIPS, typically every third or fourth instruction is a branch instruction. The main purpose of this all-branches tracing is to enable the TCB to track the execution addresses on the core without referring to the static program image, if needed. This knowledge can be used by the TCB to provide additional filtering on the trace data.

- $TraceControl_{IO}$ (InhibitOverflow) is used to ensure that trace data is never lost because of implementation-specific internal FIFO or buffer overflow. (This condition would result when a large number of bits are traced each cycle on the average while the bandwidth out of the core or TCB is far less. If this bit is asserted and an internal FIFO is in imminent danger of overflowing, then the core must stall its pipe while the FIFO is emptied.

4.2 Software Trace Control

Just as the TCB hardware can control tracing functionality using the input PDI_ signals, the PDtrace architecture allows software to control tracing with similar enables and with the same flexibility. This is done by setting bits in the Coprocessor 0 *TraceControl* register to appropriate values. To ensure that only one of hardware or software can control tracing at any given point in time, a trace select bit is used in the trace control register (*TraceControl*). A processor reset sets the trace select bit to default trace input select from the TCB hardware.

4.2.1 Coprocessor 0 Trace Registers

This section describes all the Coprocessor 0 trace registers needed for implementing PDtrace tracing logic in the core (with the exception of *TraceIBPC and TraceDBPC*, which were described in Section 3.16 “Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints”).

Table 4.1 shows a list of all the Coprocessor 0 tracing-related registers. The compliance level is specified assuming that tracing is implemented, i.e., the TL bit in Coprocessor 0 Config3 is 1 (Table 2.1).

Note that the *UserTraceData* register was renamed to *UserTraceData1* in PDtrace specification revision 06.00 and higher because of the introduction of the *UserTraceData2* register. This revision of the specification also introduces a new trace control register *TraceControl3* which needs to be implemented whether or not performance counter tracing, an optional feature, is implemented.

Table 4.1 A List of Coprocessor 0 Trace Registers

Register Number	Sel	Register Name	Reference	Compliance
23	1	TraceControl	4.2.1.1 “The TraceControl Register (CP0 Register 23, Select 1)” on page 50	Required
23	2	TraceControl2	4.2.1.2 “The TraceControl2 Register (CP0 Register 23, Select 2)” on page 53	Required
23	3	UserTraceData1	4.2.1.4 “The UserTraceData1 and UserTraceData2 Registers (CP0 Register 23 Select 3 and CP0 Register 24 Select 3)” on page 57	Required
23	4	TraceIBPC	3.16.1 “The TraceIBPC and TraceDBPC Registers” on page 37	Required

Table 4.1 A List of Coprocessor 0 Trace Registers

Register Number	Sel	Register Name	Reference	Compliance
23	5	TraceDBPC	3.16.1 "The TraceIBPC and TraceDBPC Registers" on page 37	Required
24	2	TraceControl3	4.2.1.3 "The TraceControl3 Register (CP0 Register 24, Select 2)" on page 56	Required for PDtrace spec revision 06.00 and higher
24	3	UserTraceData2	4.2.1.4 "The UserTraceData1 and UserTraceData2 Registers (CP0 Register 23 Select 3 and CP0 Register 24 Select 3)" on page 57	Required for PDtrace spec revision 06.00 and higher

4.2.1.1 The *TraceControl* Register (CP0 Register 23, Select 1)

The *TraceControl* register configuration is shown in Figure 4.1 and Table 4.2. Note the special behavior of the ASID_M, ASID, and G fields if the processor does not implement the standard TLB-based MMU.

Figure 4.1 *TraceControl* Register Format

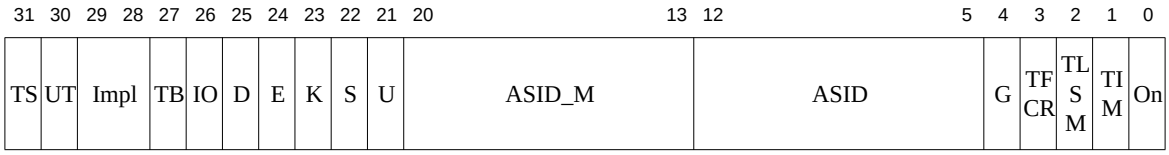


Table 4.2 *TraceControl* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
TS	31	The trace select bit is used to select between the trace control block and the software trace control bits. A value of zero selects the trace control block signals, and a value of one selects the trace control bits in the *TraceControl* register.	R/W	0	Required
0	30	*The previously defined use of this bit to specify user trace formats (as type 1 or type 2), is deprecated in PDtrace specification revisions 05.00 and higher. This functionality is now provided by the use of two explicit registers UserTraceData1 and UserTraceData2.*	0	0	Reserved
Impl	29:28	Reserved for implementation use; refer to the core-specific implementation document for usage details.	Impl-specific	Impl-specific	Reserved fir implementation
TB	27	Trace All Branch. When set to 1, this tells the processor to trace the PC value for all taken branches, not just the ones whose branch target address is statically unpredictable.	R/W	Undefined	Required

Table 4.2 *TraceControl* Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
IO	26	Inhibit Overflow. This signal is used to indicate to the core trace logic that slow but complete tracing is desired. Hence, the core tracing logic must not allow a FIFO overflow and discard trace data. This is achieved by stalling the pipeline when the FIFO is nearly full, so that no trace records are ever lost.	R/W	Undefined	Required
D	25	When set to one, this enables tracing in Debug Mode (see 2.1 "Processor Modes" on page 19). For trace to be enabled in Debug mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Debug Mode, irrespective of other bits.	R/W	Undefined	Required
E	24	When set to one, this enables tracing in Exception Mode (see 2.1 "Processor Modes" on page 19). For trace to be enabled in Exception mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Exception Mode, irrespective of other bits.	R/W	Undefined	Required
K	23	When set to one, this enables tracing in Kernel Mode (see 2.1 "Processor Modes" on page 19). For trace to be enabled in Kernel mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Kernel Mode, irrespective of other bits.	R/W	Undefined	Required
S	22	When set to one, this enables tracing in Supervisor Mode (see 2.1 "Processor Modes" on page 19). For trace to be enabled in Supervisor mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Supervisor Mode, irrespective of other bits. If the processor does not implement Supervisor Mode, this bit is ignored on write and returns zero on read.	R/W	Undefined	Required (if Supervisor Mode is implemented, is Reserved otherwise)
U	21	When set to one, this enables tracing in User Mode (see 2.1 "Processor Modes" on page 19). For trace to be enabled in User mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in User Mode, irrespective of other bits.	R/W	Undefined	Required

Table 4.2 *TraceControl* Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
ASID_M	20:13	This is a mask value applied to the ASID comparison (done when the G bit is zero). A “1” in any bit in this field inhibits the corresponding ASID bit from participating in the match. As such, a value of zero in this field compares all bits of ASID. Note that the ability to mask the ASID value is not available in the hardware signal bit; it is only available via the software control register. If the processor does not implement the standard TLB-based MMU, this field is ignored on write and returns zero on read.	R/W	Undefined	Required
ASID	12:5	The ASID field to match when the G bit is zero. When the G bit is one, this field is ignored. If the processor does not implement the standard TLB-based MMU, this field is ignored on write and returns zero on read.	R/W	Undefined	Required
G	4	When set, this implies that tracing is to be enabled for all processes, provided that other enabling functions (like U, S, etc.,) are also true. If the processor does not implement the standard TLB-based MMU, this field is ignored on write and returns 1 on read. This causes all match equations to work correctly in the absence of an ASID.	R/W	Undefined	Required
TFCR	3	When set, this indicates to the PDtrace interface that the optional Fcr bit must be traced in the appropriate trace formats. If PC tracing is disabled, the full PC of the function call (or return) instruction must also be traced. Note that function call/return information is only traced if tracing is actually enabled for the current mode.	R/W	Undefined	Required
TLSM	2	When set, this indicates to the PDtrace interface that information about data cache misses should be traced. If PC, load/store address and data tracing are disabled (see the TraceControl2$_{Mode}$ field), the full PC and load/store address are traced for data cache misses. If load/store data tracing is enabled, the LSm bit must be traced in the appropriate trace format. Note that data cache miss information is only traced if tracing is actually enabled for the current mode.	R/W	Undefined	Required
TIM	1	When set, this indicates to the PDtrace interface that the optional Im bit must be traced in the appropriate trace formats. If PC tracing is disabled, the full PC of the instruction that missed in the I-cache must be traced. Note that instruction cache miss information is only traced if tracing is actually enabled in the current mode.	R/W	Undefined	Required
On	0	This is the master trace enable switch in software control. When zero, tracing is always disabled. When set to one, tracing is enabled whenever the other enabling functions are also true.	R/W	0	Required

4.2.1.2 The *TraceControl2* Register (CP0 Register 23, Select 2)

The TraceControl2 register provides additional control and status information. It is described here in Figure 4.2 and Table 4.3. Note that some fields in the *TraceControl2* register are read-only, but have a reset state of “Undefined”. This is because these values are loaded from TCB Register bits.

Figure 4.2 *TraceControl2* Register Format

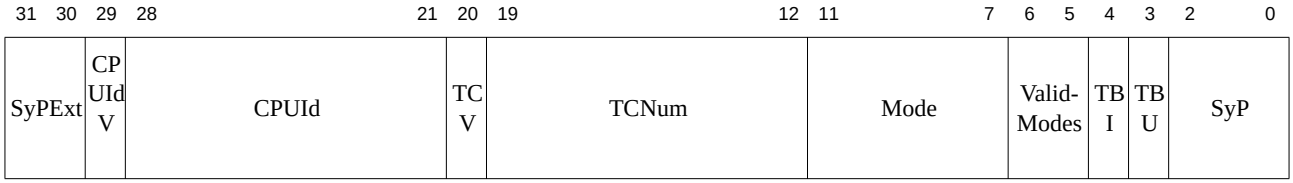


Table 4.3 *TraceControl2* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
SyPExt	31:30	This field is optional for PDtrace revisions 06.00 and higher and preset to 0 for earlier revisions. This is used to optionally extend the length of the synchronization period field SyP (bits 2:0) in this register. The value of SyP is extended by assuming that these two bits are juxtaposed to the left of the three bits of SyP (SyPExt.SyP). When only SyP was used to specify the synchronization period, the value was 2^x, where x was computed from SyP by adding 5 to the actual value represented by the bits. A similar formula is applied to the 5 bits just obtained by the juxtaposition of SyPExt and SyP. Sync period values greater than 2^{31} are UNPREDICTABLE. Since the value of 11010 represents the value of 31 (with +5), all values greater than 11010 are UNPREDICTABLE. Note that with these new bits, a sync period range of 2^5 to 2^{31} cycles can now be obtained.	0 or R/W (for PDtrace revisions 06.00 and higher)	0	Required for PDtrace rev 06.00 and higher.
CPUIdV	29	Only implemented on a processor with MT or multi-core SOC. Otherwise, this field must be written as zero; returns zero on read. When set, the CPUId field specifies the number of the VPE or CPU that must be traced. Otherwise, instructions from all VPEs are traced when other conditions for tracing are valid. On an MT system, this bit is ignored if TCV is asserted.	R/W	Undefined for a multi-VPE MT or multi-core processor, 0 otherwise	Required if MT ASE is implemented, otherwise reserved
CPUId	28:21	Only implemented on a processor with MT or multi-core SOC. Otherwise, this field must be written as zero; returns zero on read. On a MT core, specifies the number of the VPE to trace when CPUIdV is set. On a multi-core system, this is the Ebase.CPUId value. On an MT system, this bit is ignored if TCV is asserted.	R/W	Undefined for a multi-VPE MT or multi-core processor, 0 otherwise	Required if MT ASE is implemented, otherwise reserved

Table 4.3 *TraceControl2* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>TCV</td><td>20</td><td>Only implemented on a processor with MT. Otherwise, this field must be written as zero; returns zero on read. When set, the TCNum field specifies the number of the TC that must be traced. Otherwise, instructions from all TCs are traced when other conditions for tracing are valid.</td><td>R/W</td><td>Undefined for a MT processor, 0 otherwise</td><td>Required if MT ASE is implemented, otherwise reserved</td></tr>
<tr><td>TCNum</td><td>19:12</td><td>Only implemented on a processor with MT. Otherwise, this field must be written as zero; returns zero on read. Specifies the number of the TC to trace when TCV is set. For any given MT implementation only the appropriate number of bits encoding the TC number are used in the right-most position of this field. The upper bits are ignored.</td><td>R/W</td><td>Undefined for a MT processor, 0 otherwise</td><td>Required if MT ASE is implemented, otherwise reserved</td></tr>
<tr><td>Mode</td><td>11:7</td><td>These 5 bits provide the trace mode values.
It is optional for an implementation to allow PC tracing to be turned off. This must be clearly documented by the core implementation-specific document. When it is optional, bit 11 is tied to a value of 1 and setting bit 11 to 0 is simply ignored by the processor. Reading this bit always returns a value of one.
<table>
<tr><th>Bit</th><th>Trace The Following</th></tr>
<tr><td>0</td><td>PC</td></tr>
<tr><td>1</td><td>Load address</td></tr>
<tr><td>2</td><td>Store address</td></tr>
<tr><td>3</td><td>Load data</td></tr>
<tr><td>4</td><td>Store data</td></tr>
</table></td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
<tr><td>Valid-Modes</td><td>6:5</td><td>This field specifies the subset of tracing that is supported by the processor (see 2.2 “Subsetting” on page 19).
<table>
<tr><th>Encoding</th><th>Meaning</th></tr>
<tr><td>00</td><td>PC tracing only</td></tr>
<tr><td>01</td><td>PC and load and store address tracing only</td></tr>
<tr><td>10</td><td>PC, load and store address, and load and store data</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table></td><td>R</td><td>Preset</td><td>Required</td></tr>
</table>

Table 4.3 *TraceControl2* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>TBI</td><td>4</td><td>This bit indicates how many trace buffers are implemented by the TCB, as follows:
<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>Only one trace buffer is implemented, and the TBU bit of this register indicates which trace buffer is implemented</td></tr><tr><td>1</td><td>Both on-chip and off-chip trace buffers are implemented by the TCB and the TBU bit of this register indicates to which trace buffer the traces is currently written.</td></tr></table>This bit is loaded when the $TCBCONTROLB_{OfC}$ bit is set.</td><td>R</td><td>Undefined</td><td>Required</td></tr>
<tr><td>TBU</td><td>3</td><td>This bit denotes to which trace buffer the trace is currently being written and is used to select the appropriate interpretation of the $TraceControl2_{SyP}$ field.
<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>0</td><td>Trace data is being sent to an on-chip trace buffer</td></tr><tr><td>1</td><td>Trace Data is being sent to an off-chip trace buffer</td></tr></table>This bit is loaded from the $TCBCONTROLB_{OfC}$</td><td>R</td><td>Undefined</td><td>Required</td></tr>
<tr><td>SyP</td><td>2:0</td><td>The period (in cycles) to which the internal synchronization counter is reset when tracing is started, or when the synchronization counter has overflowed.
<table><tr><th>SyP</th><th>Sync Period</th></tr><tr><td>000</td><td>2^5</td></tr><tr><td>001</td><td>2^6</td></tr><tr><td>010</td><td>2^7</td></tr><tr><td>011</td><td>2^8</td></tr><tr><td>100</td><td>2^9</td></tr><tr><td>101</td><td>2^{10}</td></tr><tr><td>110</td><td>2^{11}</td></tr><tr><td>111</td><td>2^{12}</td></tr></table>This field is loaded from the $TCBCONTROLB_{SyP}$ bits.</td><td>R</td><td>Undefined</td><td>Required</td></tr>
</table>

4.2.1.3 The *TraceControl3* Register (CP0 Register 24, Select 2)

The TraceControl3 register provides additional control and status information. It is described here in Figure 4.3 and Table 4.3. Note that some fields in the *TraceControl3* register are read-only, but have a reset state of “Undefined”. This is because these values are loaded from TCB Register bits.

Figure 4.3 *TraceControl3* Register Format

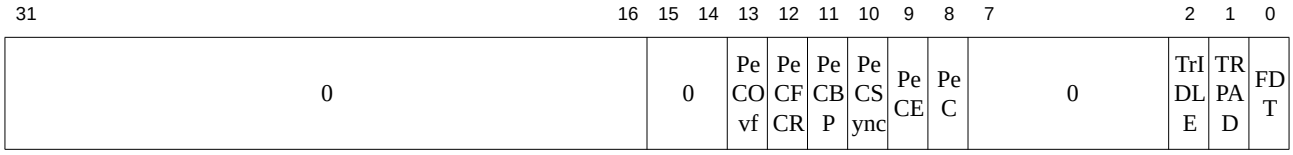


Table 4.4 *TraceControl3* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
0	31:16	Reserved for future use; Must be written as zero; returns zero on read.	0	0	Reserved
0	15:14	Reserved for future use; Must be written as zero; returns zero on read. (Hint to architect, reserved for future expansion of performance counter trace events).	0	0	Reserved
PeCOvf	13	Trace performance counters when one of the performance counters overflows its count value. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCFCR	12	Trace performance counters on function call/return or on an exception handler entry. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCBP	11	Trace performance counters on hardware breakpoint match trigger. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCSync	10	Trace performance counters on synchronization counter expiration. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCE	9	Performance counter tracing enable. When set to 0, the tracing out of performance counter values as specified is disabled. To enable, this bit must be set to 1. This bit is used under software control. When trace is controlled by an external probe, this enabling is done via the TCB control register.	R/W	0	Required after revision 06.00 and higher
PeC	8	Specifies whether or not Performance Control Tracing is implemented. This is an optional feature that may be omitted by implementation choice. See 3.17 “Tracing Performance Counter Values” on page 42 for details.	R	Preset	Required after revision 06.00 and higher
0	7:2	Reserved for future use; Must be written as zero; returns zero on read.	0	0	Required after revision 06.00 and higher

Table 4.4 *TraceControl3* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
TrIDLE	2	Trace Unit Idle. This bit indicates if the trace hardware is currently idle (not processing any data). This can be useful when switching control of trace from hardware to software and vice versa. The bit is read-only and updated by hardware.	R	1	Required after revision 06.00 and higher
TRPAD	1	Trace RAM access disable bit, disables program software access to the on-chip trace RAM using load/store instructions. This bit is a copy of the TRPAD bit (bit 18) in TCBCONTROLB.	R	0	Required after revision 06.00 and higher
FDT	0	Filtered Data Trace Mode enable bit. When the bit is 0, this mode is disabled, reset value is disable. When set to 1, this mode is enabled. This mode is described in Section 3.18 on page 43	R/W	0	Required after revision 06.00 and higher

4.2.1.4 The *UserTraceData1* and *UserTraceData2* Registers (CP0 Register 23 Select 3 and CP0 Register 24 Select 3)

A software write to any bits in the *UserTraceData1* register will trigger a trace record to be written indicating a type 1 user format. Similarly, a write by software to any bits in the *UserTraceData2* register will trigger a trace record to be written indicating a type 2 user format. *The UT bit in the TraceControl register was used to dictate the type of trace record, but the use of this bit has been deprecated in the PDtrace architecture revisions 06.00 and higher.* It is implementation dependent whether or not writes to this register cause dependency stalling, or the latency between writes to the register and the subsequent generation of the trace record. Please read the core-specific implementation specification for this information. Please note that since these two registers are in CP0 register space, the access to these registers is ruled by CP0 access restrictions imposed by the system. For example, when a processor is under the control of an operating system such as Linux, these registers cannot be written by code executing in user-level privilege mode.

Figure 4.4 *UserTraceData1* and *UserTraceData2* Register Format

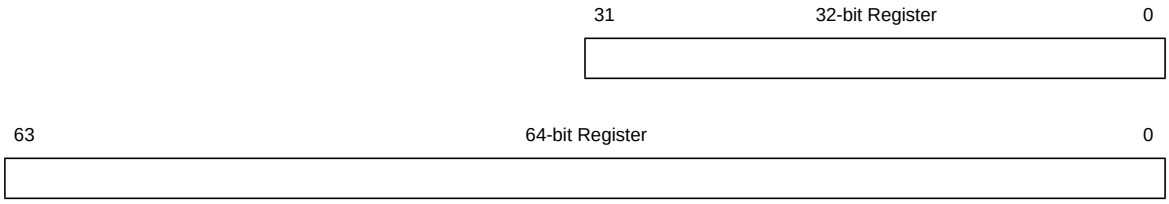


Table 4.5 *UserTraceData1* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
Data	31:0 or 63:0	Software readable/writable data. When written, this triggers a user format trace record type1 out into the trace stream to be written into the trace memory.	R/W	0	Required

Table 4.6 *UserTraceData2* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
Data	31:0 or 63:0	Software readable/writable data. When written, this triggers a user format trace record type 2 out into the trace stream to be written to trace memory.	R/W	0	Required after PDtrace spec 06.00 and higher

Chapter 5

Trace Control Block (TCB) Registers

The TCB uses several registers to control its operation. These registers are accessed via the EJTAG TAP interface. This chapter describes these registers in detail. These registers are shown in Table 5.1 and Table 5.2.

Table 5.1 Registers in the Trace Control Block

Register Name	EJTAG TAP controller instruction value	Description
TCBCONTROLA	0x10	Control register in the TCB mainly used for controlling the trace input signals to the core on the PDtrace interface.
TCBCONTROLB	0x11	Control register in the TCB that is mainly used to specify what to do with the trace information. The REG [25:21] field in this register specifies the TCB internal register to be accessed by the TCBDATA register. A list of all the registers that can be accessed by the TCBDATA register is shown in Table 5.2.
TCBDATA	0x12	This is used to access registers specified by the REG field in the TCBCONTROLB register.
TCBCONTROLC	0x13	Control Register in the TCB used to control and hold tracing information.
TCBCONTROLD	0x15	Added to support tracing coherent cores like the MIPS 1004K in PDtrace revision 05.00 and higher.
TCBCONTROLE	0x16	Added to support new features in PDtrace revision 06.00 and higher. New features include for example, performance counter tracing, etc.

Table 5.2 Registers selected by *TCBCONTROLB*$_{REG}$(accessed through TCBDATA).

REG[4:0]	Register Selected	Register Description	Compliance
0	*TCBCONFIG*	TCB Configuration register--holds information about the TCB hardware configuration.	Required
1-3	Reserved	Reserved for future use.	Reserved
4	*TCBTW*	Trace Word read register. This register holds the Trace Word just read from on-line trace memory.	Required if on-chip memory exists.
5	*TCBRDP*	Trace Word Read pointer. It points to the location in the on-line trace memory where the next Trace Word will be read. A TW read has the side-effect of post-incrementing this register value to point to the next TW location. (A maximum value wraps the address around to the beginning of the trace memory).	
6	*TCBWRP*	Trace Word Write pointer. It points to the location in the on-line trace memory where the next new Trace Word will be written.	
7	*TCBSTP*	Trace Word Start Pointer. This points to the location of the oldest TW in the on-line trace memory.	
8-15	Reserved	Reserved for future use.	Reserved
16-23	*TCBTRIGx*	Trigger Control registers 0-7 are used to specify some conditions that cause the firing of triggers, and to control the resulting action.	Optional

Table 5.2 Registers selected by *TCBCONTROLB*$_{REG}$(accessed through TCBDATA).

REG[4:0]	Register Selected	Register Description	Compliance
24-31	Reserved	Reserved for future use.	Reserved

5.1 *TCBCONTROLA* Register

The trace output from the processor on the PDtrace interface can be controlled by the trace input signals to the processor from the TCB. The TCB uses a control register, TCBCONTROLA, whose values are used to change the signal values on the PDtrace input interface. External software (i.e., debugger), can therefore manipulate the trace output by writing the TCBCONTROLA register.

The *TCBCONTROLA* register is written by an EJTAG TAP controller instruction, *TCBCONTROLA* (0x10). See the MIPS EJTAG Specification (MD00047) for more details regarding new TAP instructions. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3000 in drseg. See Section 3.19 "Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM" for details on how this register can be accessed via drseg.

Compliance: This register is required.

The format of the *TCBCONTROLA* register is shown below, and the fields are described in Table 5.3.

Figure 5.1 *TCBCONTROLA* Register Format

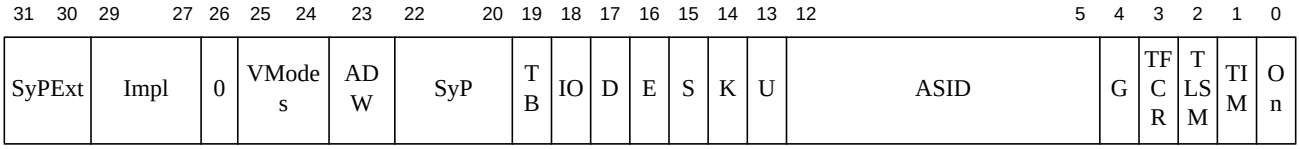


Table 5.3 *TCBCONTROLA* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
SyPExt	31:30	These two bits used to be Implementation specific until PDtrace spec revision 06.00 when it reverts to architecturally defined bits to extend the SyP (sync period) field for implementations that need higher numbers of cycles between synchronization events. The value of SyP is extended by assuming that these two bits are juxtaposed to the left of the three bits of SyP (SyPExt.SyP). When only SyP was used to specify the synchronization period, the value was 2^x, where x was computed from SyP by adding 5 to the actual value represented by the bits. A similar formula is applied to the 5 bits just obtained by the juxtaposition of SyPExt and SyP. Sync period values greater than 2^{31} are UNPREDICTABLE. Since the value of 11010 represents the value of 31 (with +5), all values greater than 11010 are UNPREDICTABLE. Note that with these new bits, a sync period range of 2^5 to 2^{31} cycles can now be obtained.	0 or R/W (for spec revisions 06.00 and higher)	0	Required after PDtrace revision 06.00 and higher

Table 5.3 *TCBCONTROLA* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>Impl</td><td>29:27</td><td>This field is reserved for implementation specific use. Refer to the processor specification for the format and definition of this field.</td><td></td><td>Undefined</td><td>Optional</td></tr>
<tr><td>0</td><td>26</td><td>Reserved for future use. Must be written as zero; returns zero on read.</td><td>0</td><td>0</td><td>Required</td></tr>
<tr><td>VModes</td><td>25:24</td><td>This field specifies the type of tracing that is supported by the processor, as follows:
<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>00</td><td>PC tracing only</td></tr><tr><td>01</td><td>PC and load and store address tracing only</td></tr><tr><td>10</td><td>PC, load, and store address, and load and store data.</td></tr><tr><td>11</td><td>Reserved</td></tr></table>This field is preset to the TCB register value ValidModes</td><td>R</td><td>Preset</td><td>Required</td></tr>
<tr><td>ADW</td><td>23</td><td>The address and data value width used in the trace formats.
0: The width is 16 bits.
1: The width is 32 bits.</td><td>R</td><td>Preset</td><td>Required</td></tr>
<tr><td>SyP</td><td>22:20</td><td>Used to indicate the synchronization period.
The period (in cycles) between which the periodic synchronization information is to be sent is defined as shown in the table below, when the trace buffer is either on-chip or off-chip (as determined by the TCBCONTROLB_{OfC} bit).
<table><tr><th>SyP</th><th>Sync Period</th></tr><tr><td>000</td><td>2^5</td></tr><tr><td>001</td><td>2^6</td></tr><tr><td>010</td><td>2^7</td></tr><tr><td>011</td><td>2^8</td></tr><tr><td>100</td><td>2^9</td></tr><tr><td>101</td><td>2^{10}</td></tr><tr><td>110</td><td>2^{11}</td></tr><tr><td>111</td><td>2^{12}</td></tr></table></td><td>R/W</td><td>000</td><td>Required</td></tr>
<tr><td>TB</td><td>19</td><td>Trace All Branches. This signal is used to indicate that the core must trace either full or incremental PC values for all branches. Not just the unpredictable ones.</td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
</table>

Table 5.3 *TCBCONTROLA* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
IO	18	Inhibit Overflow. This signal is used to indicate to the core trace logic that slow but complete tracing is desired. Hence, the core tracing logic must not allow a FIFO overflow and discard trace data. This is achieved by stalling the pipeline when the FIFO is nearly full so that no trace records are ever lost.	R/W	Undefined	Required
D	17	When set to one, this enables tracing in Debug mode, i.e., when the DM bit is one in the *Debug* register. For trace to be enabled in Debug mode, the On bit must be one and either the G bit must be one, or the current process must match the ASID field in this register. When set to zero, trace is disabled in Debug mode, irrespective of other bits.	R/W	Undefined	Required
E	16	This controls when tracing is enabled. When set, tracing is enabled when either the EXL or ERL bits in the *Status* register is one, provided that the On bit (bit 0) is also set, and either the G bit is set, or the current process ASID matches the ASID field in this register.	R/W	Undefined	Required
S	15	When set, this enables tracing when the core is in Supervisor mode as defined in the MIPS32 or MIPS64 architecture specification. This is provided the On bit (bit 0) is also set, and either the G bit is set, or the current process ASID matches the ASID field in this register.	R/W	Undefined	Required
K	14	When set, this enables tracing when the On bit is set and the core is in Kernel mode. Unlike the usual definition of Kernel Mode, this bit enables tracing only when the ERL and EXL bits in the *Status* register are zero. This is provided the On bit (bit 0) is also set, and either the G bit is set, or the current process ASID matches the ASID field in this register.	R/W	Undefined	Required
U	13	When set, this enables tracing when the core is in User mode as defined in the MIPS32 or MIPS64 architecture specification. This is provided the On bit (bit 0) is also set, and either the G bit is set, or the current process ASID matches the ASID field in this register.	R/W	Undefined	Required
ASID	12:5	The ASID field to match when the G bit is zero. When the G bit is one, this field is ignored.	R/W	Undefined	Required
G	4	When set, this implies that tracing is to be enabled for all processes, provided that other enabling functions (like U, S, etc.,) are also true.	R/W	Undefined	Required
TFCR	3	When set, this indicates to the PDtrace interface that the optional Fcr bit must be traced in the appropriate trace formats. If PC tracing is disabled, the full PC of the function call (or return) instruction must also be traced. Note that function call/return information is only traced if tracing is actually enabled for the current mode.	R/W	Undefined	Required for PDtrace revisions 4.00 and higher

Table 5.3 *TCBCONTROLA* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
TLSM	2	When set, this indicates to the PDtrace interface that information about data cache misses should be traced. If PC, load/store address and data tracing are disabled (see the $TraceControl2_{Mode}$ field), the full PC and load/store address are traced for data cache misses. If load/store data tracing is enabled, the LSm bit must be traced in the appropriate trace format. Note that data cache miss information is only traced if tracing is actually enabled for the current mode.	R/W	Undefined	Required for PDtrace revisions 4.00 and higher
TIM	1	When set, this indicates to the PDtrace interface that the optional Im bit must be traced in the appropriate trace formats. If PC tracing is disabled, the full PC of the instruction that missed in the I-cache must be traced. Note that instruction cache miss information is only traced if tracing is actually enabled in the current mode.	R/W	Undefined	Required for PDtrace revisions 4.00 and higher
On	0	This is the global trace enable switch to the core. When zero, tracing from the core is always disabled, unless enabled by core internal software override When set to one, tracing is enabled whenever the other enabling functions are also true.	R/W	0	Required

5.2 *TCBCONTROLB* Register

The TCB includes a second control register, *TCBCONTROLB* (0x11). This register generally controls what happens to the trace information once it arrives at the TCB. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3008 in drseg. See Section 3.19 "Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM" for details on how this register can be accessed via drseg.

Compliance: This register is required.

The format of the *TCBCONTROLB* register is shown below, and the fields are described in Table 5.4.

Figure 5.2 *TCBCONTROLB* Register Format

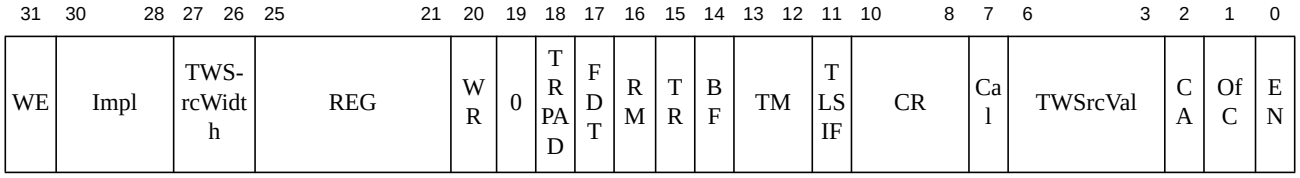


Table 5.4 *TCBCONTROLB* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
WE	31	Write Enable. Only when set to 1 will the other bits be written in *TCBCONTROLB*. This bit will always read 0.	R	0	Required

Table 5.4 *TCBCONTROLB* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
Impl	30:28	This field is reserved for implementations. Refer to the processor specification for the format and definition of this field.		Undefined	Optional
TWSrc-Width	27:26	Used to indicate the number of bits used in the source field of the Trace Word, this is a configuration option of the core that cannot be modified by software. 00 - zero source field width 01 - two bit source field width 10 - four bit source field width 11 - reserved for future use	R	Preset	Required for PDtrace revisions 4.00 and higher
REG	25:21	Register select: This field specifies the register, (one among the set of registers in Table 5.2) that can be accessed through the *TCBDATA* register.	R/W	0	Required
WR	20	The write register field, when set, allows the register selected by the REG field to be written as well as read when *TCBDATA* is accessed. Otherwise, the selected register is only read. Note that a JTAG register cannot be only written, it is always read and written. Therefore, a register that has a side-effect on read (see 5.9 "TCBRDP Register (Reg 5)") will have the same side-effect when written, since a read also happens on a write. Hence, it is specified that when this field WR is set, it is implementation dependent whether a side-effect of a read will occur when writing.	R/W	0	Required
0	19	Reserved for future use. Must be written as zero; returns zero on read.	0	0	Reserved
TRPAD	18	Trace RAM access disable bit, disables program software access to the on-chip trace RAM using load/store instructions. When this bit is set, this takes priority over the bit in *TraceControl3* (TRPAD bit 1). That is, if the TRPAD bit in *TraceControl3* is set to 0, that is, software access to on-chip trace memory is enabled, and this bit in this register is 0, that is the access is disabled, then software access to the on-chip memory is disabled. If probe access is not provided in the implementation, then this register bit must be tied to zero value to allow software to control access.	R/W	0	Required after revision 06.00 and higher
FDT	17	Filtered Data Trace Mode enable bit. When the bit is 0, this mode is disabled, reset value is disable. When set to 1, this mode is enabled. This mode is described in Section 3.18 on page 43	R/W	0	Required after revision 06.00 and higher

Table 5.4 *TCBCONTROLB* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>RM</td><td>16</td><td>Read on-chip trace memory.
When written to 1, the read address-pointer of the on-chip memory in register TCBRDP is set to point to the oldest memory location written since the last reset of pointers. Subsequent access to the TCBTW register (through the TCBDATA register), will automatically increment the read pointer in register TCBRDP after each read.
When the write pointer is reached, this bit is automatically reset to 0, and the TCBTW register will read all zeros.
Once set to 1, writing 1 again will have no effect. The bit is reset by setting the TR bit or by reading the last Trace word in TCBTW.</td><td>R/W</td><td>0</td><td>Required if on-chip memory exists. Otherwise reserved.</td></tr>
<tr><td>TR</td><td>15</td><td>Trace memory reset.
When written to one, the address pointers for the on-chip trace memory TCBRDP and TCBWRP are reset to zero. Also the RM and BF bits are reset to 0.
This bit is automatically reset back to 0, when the reset specified above is completed.</td><td>R/W1</td><td>0</td><td>Required if on-chip memory exists. Otherwise reserved.</td></tr>
<tr><td>BF</td><td>14</td><td>Buffer Full indicator that the TCB uses to communicate to external software that the on-chip trace memory is full. Note that this applies only in the situation that the on-chip trace memory is being deployed in the trace-from and trace-to mode. (See Chapter 1, “On-Chip Trace Memory” on page 55)
This bit is cleared when writing a 1 to the TR bit</td><td>R</td><td>0</td><td>Required if on-chip memory exists. Otherwise reserved.</td></tr>
<tr><td>TM</td><td>13:12</td><td>Trace Mode. This field determines how the trace memory is filled when using the simple-break control in the PDtrace™ IF to start or stop trace.
<table><tr><th>TM</th><th>Trace Mode</th></tr><tr><td>00</td><td>Trace-To</td></tr><tr><td>01</td><td>Trace-From</td></tr><tr><td>10</td><td>Reserved</td></tr><tr><td>11</td><td>Reserved</td></tr></table>In Trace-To mode, the on-chip trace memory is filled, continuously wrapping around, overwriting older Trace Words, as long as there is trace data coming from the core.
In Trace-From mode, the on-chip trace memory is filled from the point that the core starts tracing until the on-chip trace memory is full.
In both cases, de-asserting the EN bit in this register will also stop fill to the trace memory.
If a TCBTRIGx trigger control register is used to start/stop tracing, then this field should be set to Trace-To mode.</td><td>R/W</td><td>0</td><td>Required if on-chip memory exists. Otherwise reserved.</td></tr>
</table>

Table 5.4 *TCBCONTROLB* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
TLSIF	11	When set, this indicates to the TCB that information about Load and Store data cache miss, instruction cache miss, and function call are to be taken from the PDtrace interface and trace them out in the appropriate trace formats as the three optional bits LSm, Im, and Fcr.	R/W	0	Required for PDtrace revisions 4.00 and higher
CR	10:8	Off-chip Clock Ratio. Writing this field, sets the ratio of the core clock to the off-chip trace memory interface clock. The clock-ratio encoding is shown in Table 5.5. **Remark:** For example, a clock ratio of 1:2 implies a two times slow down of the Probe interface clock to the core clock. But, one data packet is sent per core clock rising edge, while a data packet is sent on every edge of the Probe interface clock, since the Probe interface works in double data rate (DDR) mode.	R/W	100	Required if off-chip trace interface exists. Otherwise reserved.
Cal	7	Calibrate off-chip trace interface. If set, the off-chip trace pins will produce the following pattern in consecutive trace clock cycles. If more than 4 data pins exist, the pattern is replicated for each set of 4 pins. The pattern repeats from top to bottom until the Cal bit is de-asserted. Calibrations pattern This pattern is replicated for every 4 bits of *TR_DATA* pins. 3 2 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 **Note:** The clock source of the TCB and PIB must be running.	R/W	0	Required if off-chip trace interface exists. Otherwise reserved.
TWSrcVal	6:3	These bits are used to indicate the value of the TW source field that will be traced if TWSrcBits indicates a source bit field width of 2 or 4 bits. Note that if the field is 2 bits, then only bits 4:3 of this field will be used in the TW.	R/W	0	Required for PDtrace revisions 4.00 and higher.

Table 5.4 *TCBCONTROLB* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
CA	2	Cycle accurate trace. When set to 1 the trace will include stall information. When set to 0 the trace will exclude stall information, and remove bit zero from all transmitted TF's. The stall information included/excluded is: • TF6 formats with TCBcode 0001 and 0101. • All TF1 formats except within the context of multi-pipe processor tracing (when it is used for individual pipes within the sequence of pipe outputs).	R/W	0	Required
OfC	1	If set to 1, trace is sent to off-chip memory using *TR_DATA* pins. If not set, trace info is sent to on-chip memory. This bit is read only if either off-chip or on-chip option exists.	R/W	Preset	Required
EN	0	Enable trace. This is the master enable for trace to be generated from the TCB. This bit can be set or cleared, either by writing this register or from a start/stop trigger. When set to "1", trace information is sampled on the output pins or written into the onchip trace memory. Trace Words are generated and sent to either on-chip memory or to the Trace Probe. The target of the trace is selected by the OfC bit. When set to "0", trace information on the output trace pins are ignored. A potential TF6-stop (from a stop trigger) is generated as the last information, and the TCB pipe-line is flushed, and trace output is stopped.	R/W	0	Required

Table 5.5 Clock Ratio encoding of the CR field

CR/CRMin/CRMax	Clock Ratio
000	8:1 (Trace clock is eight times that of core clock)
001	4:1 (Trace clock is four times that of core clock)
010	2:1 (Trace clock is double that of core clock)
011	1:1 (Trace clock is same as core clock)
100	1:2 (Trace clock is one half of core clock)
101	1:4 (Trace clock is one fourth of core clock)
110	1:6 (Trace clock is one sixth of core clock)
111	1:8 (Trace clock is one eighth of core clock)

5.3 *TCBCONTROLC* Register

The trace output from the processor on the PDtrace interface can be controlled by the trace input signals to the processor from the TCB. The TCB uses a control register, TCBCONTROLC, whose values are used to change the signal values on the PDtrace input interface. External software (i.e., debugger), can therefore manipulate the trace output by writing the TCBCONTROLC register.

The *TCBCONTROLC* register is written by an EJTAG TAP controller instruction, *TCBCONTROLC* (0x13). See the MIPS EJTAG Specification (MD00047) for more details regarding new TAP instructions. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3010 in drseg. See Section 3.19 "Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM" for details on how this register can be accessed via drseg.

Compliance: This register is required for PDtrace revisions 4.00 and higher.

The format of the *TCBCONTROLC* register is shown below, and the fields are described in Table 5.3.

Figure 5.3 *TCBCONTROLC* Register Format

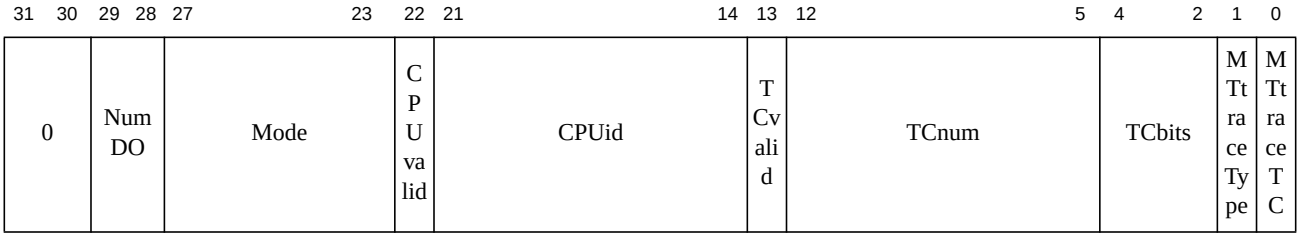


Table 5.6 *TCBCONTROLC* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
0	31:30	Reserved for future use. Must be written as zero; returns zero on read.	0	0	Reserved
NumDO	29:28	Specifies the number of bits needed by this implementation to specify the DataOrder: 00 - Four bits 01 - Five bits 10 - Six bits 11 - Eight bits	R	Preset	Required for PDtrace revision 5.0 and higher

Table 5.6 *TCBCONTROLC* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>Mode</td><td>27:23</td><td>When tracing is turned on, this signal specifies what information is to be traced by the core. It uses 5 bits, where each bit turns on a tracing of a specific tracing mode. The
<table><tr><th>Bit # Set</th><th>Trace The Following</th></tr><tr><td>0</td><td>PC</td></tr><tr><td>1</td><td>Load address</td></tr><tr><td>2</td><td>Store address</td></tr><tr><td>3</td><td>Load data</td></tr><tr><td>4</td><td>Store data</td></tr></table>table shows what trace value is turned on when that bit value is a 1. If the corresponding bit is 0, then the Trace Value shown in column two is not traced by the processor. This implementation is required for all processors using PDtrace specification 4.00 and higher.
Obviously, the processor has to support the tracing mode that is being requested for this input signal to have any effect. For example, if the processor only supports PC tracing, then only bit 0 is read by the processor, and other bits are ignored, and so on. Which bits are ignored and which are read can be obtained by reading the ValidModes bits.
It is optional for an implementation to allow PC tracing to be turned off. This must be clearly documented by the core implementation-specific document. When it is optional, bit 23 is tied to a value of 1 and setting bit 23 to 0 is simply ignored by the processor. Reading this bit always returns a value of one.</td><td>R/W</td><td>0</td><td>Required for PDtrace revisions 4.00 and higher</td></tr>
<tr><td>CPUvalid</td><td>22</td><td>This signal indicates whether or not to use the value in the CPUid field as the CPU to be traced. Only implemented on a processor with MT or multi-core SOC. Otherwise, this field must be written as zero; returns zero on read. When set, the CPUId field specifies the number of the VPE or CPU that must be traced. Otherwise, instructions from all VPEs are traced when other conditions for tracing are valid. On an MT system, this bit is ignored if TCV is asserted.</td><td>R/W</td><td>0</td><td>Required for PDtrace revisions 4.00 and higher if MT is present or multi-core</td></tr>
<tr><td>CPUId</td><td>21:14</td><td>This signal indicates the value of the CPU id to be traced if CPUvalid is set.</td><td>R/W</td><td>Undefined</td><td>Required for PDtrace revisions 4.00 and higher if MT is present or multi-core</td></tr>
<tr><td>TCvalid</td><td>13</td><td>This signal indicates whether or not to use the value in the TCnum field as the TC to be trace.</td><td>R/W</td><td>0</td><td>Required for PDtrace revisions 4.00 and higher if MT is present</td></tr>
</table>

Table 5.6 *TCBCONTROLC* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
TCnum	12:5	This signal indicates the value of the TC to be traced if TCvalid is set.	R/W	Undefined	Required for PDtrace revisions 4.00 and higher if MT is present
TCbits	4:2	This value is used by the TCB to determine the number of bits needed to represent the TC value for this MT ASE core configuration. This value can range from 1 to 8 bits when the value is 0 to 7. This determines the number of bits that will be used in the trace formats generated by this core.	R	Preset	Required for PDtrace revisions 4.00 and higher if MT is present
MTtrace-Type	1	This bit indicates the type of implemented multi-threading: fine-grained, i.e., switch threads every cycle (bit value 0), or coarse-grained, which is also referred to as block multithreading (bit value 1).	R	Preset	Required for PDtrace revisions 4.00 and higher if MT is present
MTtraceTC	0	This bit is used by the TCB to either disable or enable TC tracing. A value of 0 implies that a TC value is not traced, and a value of 1 implies that a TC value is traced. Whether or not the TC value is traced usingTF7 format or augmented TF formats is determined by the type of multi-threading, that is, the MTtraceType field. If the type bit is 0, that is, fine-grained multi-threading, then each TF format is augmented by the TC information. If the type bit is 1, then a TF7 format is used and each TF format is not augmented.	R/W	Undefined	Required for PDtrace revisions 4.00 and higher if MT is present

5.4 TCBControlD Register

The trace control block adds a new register - TCBControlD to control trace output from the coherence manager in the MIPS 1004K core. **Note:** The value of the TCBControlB field TWSrcWidth must be set to ‘10’ on a 1004K core, to indicate that the source ID field is 4 bits wide. The *TCBCONTROLD* register is written by an EJTAG TAP controller instruction, *TCBCONTROLD* (0x15). See the MIPS EJTAG Specification (MD00047) for more details regarding new TAP instructions. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3018 in drseg. See Section 3.19 “Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM” for details on how this register can be accessed via drseg.

Compliance: This register is required for PDtrace revisions 05.00 and higher. In a non-coherent core that implements PDtrace rev 5.00 or higher, all bit-fields are read-only.

The format of the *TCBCONTROLD* register is shown below, and the fields are described in Table 5.7.

Figure 5.4 *TCBCONTROLD* Register Format

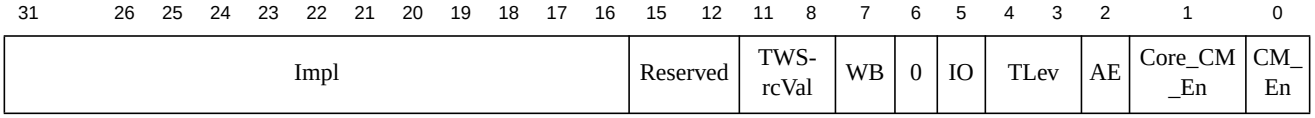


Table 5.7 *TCBCONTROLD* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
Impl	31:16	Reserved for implementations. Check core documentation		Undefined	Optional
Reserved	15:12	Reserved for future use. Must be written as 0, and read as 0	0	0	Required for PDtrace revision 05.00 or higher
TWSrcVal	11:8	The source ID of the CM.	0	0	Required for PDtrace revision 05.00 or higher
WB	7	When this bit is set, Coherent Writeback requests are traced. If this bit is not set, all Coherent Writeback requests are suppressed from the CM trace stream	R/W	0	Required for PDtrace revision 05.00 or higher
Reserved	6	Reserved for future use. Must be written as 0, and read as 0	0	0	Required for PDtrace revision 05.00 or higher
IO	5	Inhibit Overflow on CM FIFO full condition. Will stall the CM until forward progress can be made	R/W	Undefined	Required for PDtrace revision 05.00 or higher
TLev	4:3	This defines the current trace level being used by CM tracing Encoding / Meaning: 00 – No Timing Information 01 – Include Stall Times, Causes 10 – Reserved 11 – Reserved	R/W	Undefined	Required for PDtrace revision 05.00 or higher
AE	2	When set to 1, address tracing is always enabled for the CM. This affects trace output from the serialization unit of the CM.	R/W	0	Required for PDtrace revision 05.00 or higher
Core_CM_En	1	Each core can enable or disable CM tracing using this bit. This bit is not routed through the master core, but is individually controlled by each core. Setting this bit can enable tracing from the CM even if tracing is being controlled through software, if all other enabling functions are true.	R/W	0	Required for PDtrace revision 05.00 or higher

Table 5.7 *TCBCONTROLD* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
CM_EN	0	This is the master trace enable switch to the CM. When zero, tracing from the CM is always disabled. When set to one, tracing is enabled if other enabling functions are true.	R/W	0	Required for PDtrace revision 05.00 or higher

Since each core in the system has its own set of TCBControl registers, one core must be made the ‘master’ core that controls trace functionality for the CM. This can be done using a CMP GCR to designate a core as the master trace control for the CM. This control register is located in the global debug block within the GCR address space of the CM, at offset 0x0000. The format of the register is given below.

Figure 5.5 *PDtrace Control Configuration Register* Format

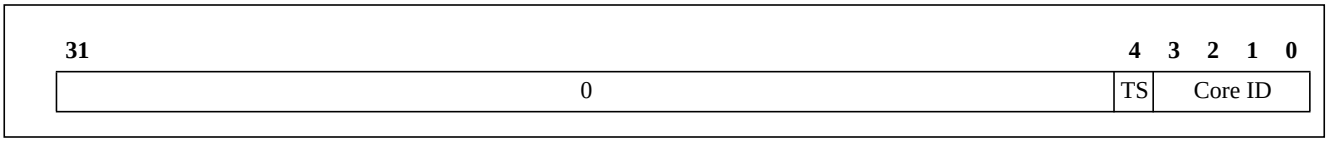


Table 5.8 PDtrace Control Configuration Register

Name	Bits	Description	Read / Write	Reset State	Compliance
0	31-5	Reserved for future use. Must be written as zero; returns zero on read.	R	0	Required
TS	4	The trace select bit is used to select between the hardware and the software trace control bits. A value of zero selects the external hardware trace block signals, and a value of one selects the trace control bits in the CMTrace-Control register	R/W	0	Required
CoreID	3:0	ID of core that controls PDtrace configuration for the coherent subsystem	R/W	0	Required

Each core in the coherent multiprocessor system has independent control over the Core_CM_EN bit. *(i.e., this field is not muxed using the GCR control register. Each core can turn on or turn off trace by setting this bit. The signal will be a wire-or of the N core signals and the SW_Trace_ON signal).*

5.5 *TCBCONTROLE* Register

The trace output from the processor on the PDtrace interface can be controlled by the trace input signals to the processor from the TCB. The TCB uses a control register, TCBCONTROLE, whose values are used to change the signal values on the PDtrace input interface. External software (i.e., debugger), can therefore manipulate the trace output by writing the TCBCONTROLE register. This register was added for PDtrace specification revision 06.00 and higher.

The *TCBCONTROLE* register is written by an EJTAG TAP controller instruction, *TCBCONTROLE* (0x16). See the MIPS EJTAG Specification (MD00047) for more details regarding new TAP instructions. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3020 in drseg. See Section 3.19 “Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM” for details on how this register can be accessed via drseg.

Compliance: This register is required for PDtrace revisions 06.00 and higher.

The format of the *TCBCONTROLE* register is shown below, and the fields are described in Table 5.9.

Figure 5.6 *TCBCONTROLE* Register Format

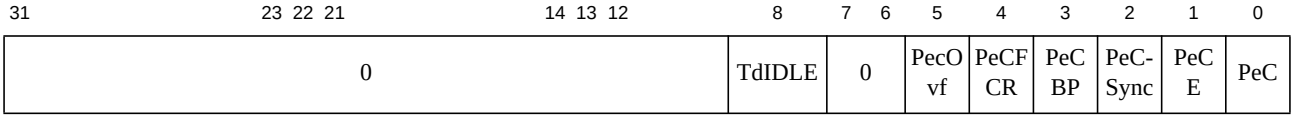


Table 5.9 *TCBCONTROLE* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
0	31:9	Reserved for future use. Must be written as zero; returns zero on read.	0	0	Reserved
TrIDLE	8	Trace Unit Idle. This bit indicates if the trace hardware is currently idle (not processing any data). This can be useful when switching control of trace from hardware to software and vice versa. The bit is read-only and updated by the trace hardware.	R	1	Required after revision 06.00 and higher
0	7:6	Reserved for future use; Must be written as zero; returns zero on read. (Hint to architect, reserved for future expansion of performance counter trace events).	0	0	Reserved
PeCOvf	5	Trace performance counters when one of the performance counters overflows its count value. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCFCR	4	Trace performance counters on function call/return or on an exception handler entry. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCBP	3	Trace performance counters on hardware breakpoint match trigger. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCSync	2	Trace performance counters on synchronization counter expiration. Enabled when set to 1.	R/W	0	Required after revision 06.00 and higher
PeCE	1	Performance counter tracing enable. When set to 0, the tracing out of performance counter values as specified is disabled. To enable, this bit must be set to 1. This bit is used under software control. When trace is controlled by an external probe, this enabling is done via the TCB control register.	R/W	0	Required after revision 06.00 and higher
PeC	0	Specifies whether or not Performance Control Tracing is implemented. This is an optional feature that may be omitted by implementation choice. See Section XXX for details.	R	Preset	Required after revision 06.00 and higher

5.6 *TCBDATA* Register

The *TCBDATA* register (0x12) is used to access the registers defined by the $TCBCONTROLB_{REG}$ field, see Table 5.2. Regardless of which register or data entry is accessed through *TCBDATA*, the register is only written if the $TCBCONTROLB_{WR}$ bit is set. For read only registers the $TCBCONTROLB_{WR}$ is a don't-care.

Compliance: This register is required.

The format of the *TCBDATA* register is shown below, and the field is described in Table 5.10. The width of *TCBDATA* is 64 bits when on-chip trace words (TWs) are accessed (*TCBTW* access).

Figure 5.7 *TCBDATA* Register Format

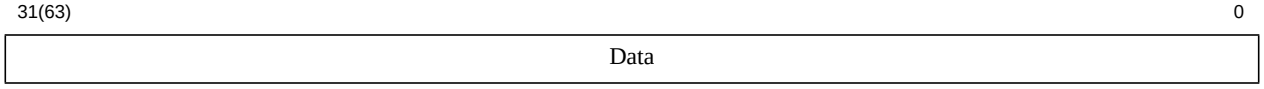


Table 5.10 *TCBDATA* Register Field Descriptions

Fields		Description	Read/Write	Reset State	Compliance
Names	Bits				
Data	31:0 63:0	Register fields or data as defined by the $TCBCONTROLB_{REG}$ field	Only writable if $TCBCONTROLB_{WR}$ is set	0	Required

5.7 *TCBCONFIG* Register (Reg 0)

The *TCBCONFIG* register holds hardware configuration information in the TCB. This register is also mapped to offset 0x3028 in drseg. See Section 3.19 "Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM" for details on how this register can be accessed via drseg.

Compliance: This register is required.

Figure 5.8 *TCBCONFIG* Register Format

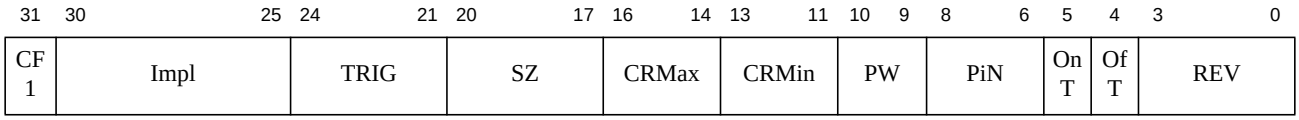


Table 5.11 *TCBCONFIG* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
CF1	31	This bit is set if a *TCBCONFIG1* register exists. In this revision, *TCBCONFIG1* does not exist, and this bit reads zero.	R	0	Required
Impl	30:25	This field is reserved for implementations. Refer to the processor specification for the format and definition of this field.	0	Undefined	Optional
TRIG	24:21	Number of triggers implemented. This also indicates the number of *TCBTRIGx* registers that exist.	R	Legal values are 0 - 8	Required
SZ	20:17	On-chip trace memory size. This field holds the encoded size of the on-chip trace memory. The size in bytes is given by $2^{(SZ+8)}$. I.e., the lowest value is 256 bytes, and the highest is 8Mb.	R	Preset	Required if on-chip memory exists. Otherwise reserved.

Table 5.11 *TCBCONFIG* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>CRMax</td><td>16:14</td><td>Off-chip Maximum Clock Ratio.
This field indicates the maximum ratio of the core clock to the off-chip trace memory interface clock. The clock-ratio encoding is shown in Table 5.5.</td><td>R</td><td>Preset</td><td>Required if off-chip trace interface exists. Otherwise reserved.</td></tr>
<tr><td>CRMin</td><td>13:11</td><td>Off-chip Minimum Clock Ratio.
This field indicates the minimum ratio of the core clock to the off-chip trace memory interface clock. The clock-ratio encoding is shown in Table 5.5.</td><td>R</td><td>Preset</td><td>Required if off-chip trace interface exists. Otherwise reserved.</td></tr>
<tr><td>PW</td><td>10:9</td><td>Probe Width: Number of bits available on the off-chip trace interface TR_DATA pins. The number of TR_DATA pins is encoded, as shown in the table.
<table><tr><th>PW</th><th>Number of bits used on TR_DATA</th></tr><tr><td>00</td><td>4 bits</td></tr><tr><td>01</td><td>8 bits</td></tr><tr><td>10</td><td>16 bits</td></tr><tr><td>11</td><td>reserved</td></tr></table>This field is preset based on input signals to the TCB and the actual capability of the TCB.</td><td>R</td><td>Preset</td><td>Required if off-chip trace interface exists. Otherwise reserved.</td></tr>
<tr><td>PiN</td><td>8:6</td><td>Pipe number.
For single-pipeline processors this field must read 0.
For multi-pipeline processor, this field indicates the number of pipes which are traced. If non-zero this also indicates, that the 3-bit PgmOrder field is included in the TF2, TF3 and TF4 Trace Formats, as shown in Figure 2-7, Figure 2-8 and Figure 2-9 on page 6.
The table below indicates the number of bits in PgmOrder for the possible values of PiN.
<table><tr><th>PiN</th><th>Number of Pipes traced</th><th>PgmOrder field included in the TF2, TF3 and TF4 Trace Formats</th></tr><tr><td>000</td><td>1</td><td>No</td></tr><tr><td>001</td><td>2</td><td rowspan="8">Yes</td></tr><tr><td>010</td><td>3</td></tr><tr><td>011</td><td>4</td></tr><tr><td>100</td><td>5</td></tr><tr><td>101</td><td>6</td></tr><tr><td>110</td><td>7</td></tr><tr><td>111</td><td>8</td></tr></table></td><td>R</td><td>Preset</td><td>Required</td></tr>
<tr><td>OnT</td><td>5</td><td>When set, this bit indicates that on-chip trace memory is present. This bit is preset based on the selected option when the TCB is implemented.</td><td>R</td><td>Preset</td><td>Required</td></tr>
</table>

Table 5.11 *TCBCONFIG* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
OfT	4	When set, this bit indicates that off-chip trace interface is present. This bit is preset based on the selected option when the TCB is implemented, and on the existence of a PIB module (*TC_PibPresent* asserted).	R	Preset	Required
REV	3:0	Revision of TCB. An implementation that conforms to the described architecture in this document (PDtrace revision 4.xx) must have revision 1. An implementation that conforms to PDtrace specification revision 05.00 must have this field set to integer value 2. An implementation that conforms to PDtrace specification 06.00 must have this field set to integer value 3.	R	0	Required

5.8 *TCBTW* Register (Reg 4)

The *TCBTW* register is used to read Trace Words from the on-chip trace memory. The TW read is the TW pointed to by the *TCBRDP* register. A side effect of reading the *TCBTW* register is that the *TCBRDP* register increments to the next TW in the on-chip trace memory. If *TCBRDP* is at the max size of the on-chip trace memory, the increment wraps back to address zero. Starting with PDtrace rev 6.00, the TCBTW register is mapped to offset 0x3100 in drseg. To read a 64-bit trace word from memory on a 32-bit processor, the user is required to execute two load word instructions. The first instruction targets offset 0x3104 in drseg, and the second one accesses offset 0x3100. An access to offset 0x3100 automatically causes the read pointer to be incremented. The use of load half-word or load byte instructions can lead to unpredictable results, and is not recommended. The results of attempting a write to trace memory by an explicit store instruction targeting TCBTW are unpredictable. See Section 3.19 "Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM" for details on how this register can be accessed via drseg.

Compliance: Required if on-chip trace memory is implemented.

The format of the *TCBTW* register is shown below, and the field is described in Table 5.12.

Figure 5.9 *TCBTW* Register Format

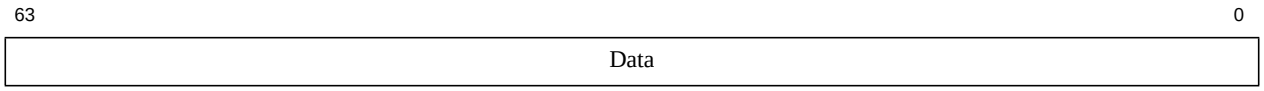


Table 5.12 *TCBTW* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
Data	63:0	Trace Word	R/W	0	Required

5.9 *TCBRDP* Register (Reg 5)

The *TCBRDP* register is the address pointer to on-chip trace memory. It points to the TW read when reading the *TCBTW* register. When writing the $TCBCONTROLB_{RM}$ bit to 1, this pointer is reset to the current value of *TCBSTP*. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3108 in drseg. See Section

3.19 “Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM” for details on how this register can be accessed via drseg.

Compliance: Required if on-chip trace memory is implemented.

The format of the *TCBRDP* register is shown below, and the field is described in Table 5.12. The value of n depends on the size of the on-chip trace memory. As the address points to a 64-bit TW, lower three bits are always zero.

Figure 5.10 *TCBRDP* Register Format

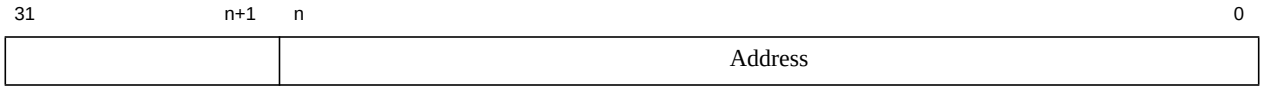


Table 5.13 *TCBRDP* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
Data	31:(n+1)	Reserved. Must be written zero, reads back zero.	0	0	Required
Address	n:0	Byte address of on-chip trace memory word.	R/W	0	Required

5.10 *TCBWRP* Register (Reg 6)

The *TCBWRP* register is the address pointer to on-chip trace memory. It points to the location where the next new TW for on-chip trace will be written. Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3110 in drseg. See Section 3.19 “Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM” for details on how this register can be accessed via drseg.

Compliance: Required if on-chip trace memory is implemented.

The format of the *TCBWRP* register is shown below, and the field is described in Table 5.12. The value of n depends on the size of the on-chip trace memory. As the address points to a 64-bit TW, lower three bits are always zero.

Figure 5.11 *TCBWRP* Register Format

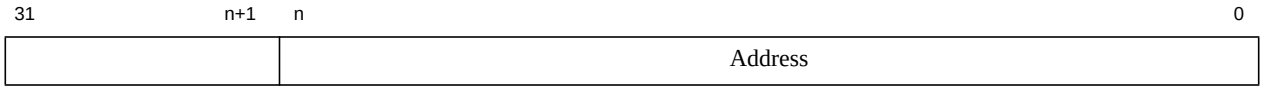


Table 5.14 *TCBWRP* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
Data	31:(n+1)	Reserved. Must be written zero, reads back zero.	0	0	Required
Address	n:0	Byte address of on-chip trace memory word.	R/W	0	Required

5.11 *TCBSTP* Register (Reg 7)

The *TCBSTP* register is the start pointer register. This register points to the on-chip trace memory address at which the oldest TW is located. This pointer is reset to zero when the $TCBCONTROLB_{TR}$ bit is written to 1. If a continuous trace to on-chip memory wraps around the on-chip memory, *TSBSTP* will have the same value as *TCBWRP*.

Starting with PDtrace rev 6.00, this register is also mapped to offset 0x3118 in drseg. See Section 3.19 “Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM” for details on how this register can be accessed via drseg.

Compliance: Required if on-chip trace memory is implemented.

The format of the *TCBSTP* register is shown below, and the field is described in Table 5.12. The value of n depends on the size of the on-chip trace memory. As the address points to a 64-bit TW, lower three bits are always zero.

Figure 5.12 *TCBSTP* Register Format

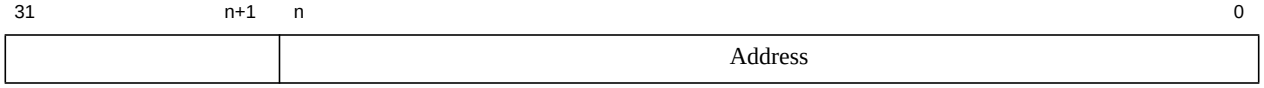


Table 5.15 *TCBSTP* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
Data	31:(n+1)	Reserved. Must be written zero, reads back zero.	0	0	Required
Address	n:0	Byte address of on-chip trace memory word.	R/W	0	Required

5.12 *TCBTRIGx* Register (Reg 16-23)

Eight Trigger Control registers are defined. Each register is named *TCBTRIGx*, where *x* is a single digit number from 0 to 7 (*TCBTRIG0* is Reg 16). The actual number of trigger registers implemented is defined in the $TCBCONFIG_{TRIG}$ field. An unimplemented register will read all zeros and writes are ignored. Starting with PDtrace rev 6.00, these registers are also mapped from offset 0x3200 to 0x3238 in drseg. See Section 3.19 “Software Control of Trace and Memory-mapped Access to On-Chip Trace RAM” for details on how these registers can be accessed via drseg.

Each Trigger Control register controls when an associated trigger is fired and the resulting action. Please also read Chapter 5, “TCB Trigger Logic,” on page 33, for a detailed description of trigger logic issues.

Compliance: The number of implemented trigger registers must be equal to the number in $TCBCONFIG_{TRIG}$.

Figure 5.13 *TCBTRIGx* Register Format

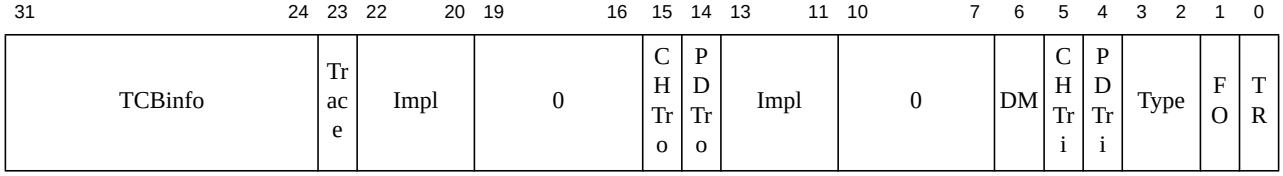


Table 5.16 *TCBTRIGx* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
TCBinfo	31:24	TCBinfo to be used in a possible TF6 trace format when this trigger fires.	R/W	0	Required

Table 5.16 *TCBTRIGx* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
Trace	23	When set, generate a TF6 trace information when this trigger fires. Use TCBinfo field for the TCBinfo of TF6 and use Type field for the two MSB of the TCBtype of TF6. The two LSB of TCBtype are 00. The write value of this bit always controls the action from the firing of this trigger. When this trigger fires, if another higher priority trigger fires simultaneously, then the action of this trigger can be suppressed. That is, the issue of the TF6 format would be suppressed. If this ever happens, this can be detected by reading the value of this field. If the Trace field was set to 1, and this trigger action was suppressed, then the read of this Trace field will return a 0. (Note that the read value is always 0 if the write value was 0). The read value of 0 indicating a suppressed trigger action is valid until the TCBTRIGx register is again written. That is, the read value is 0 if the trigger fires but the trigger action was ever suppressed, since the last write.	R/W	0	Required
Impl	22:20	These bits are reserved for implementation specific trigger actions (internal to the TCB). Refer to the processor specification for the format and definition of this field.		0	Optional
0	19:16	Reserved. Must be written zero, reads back zero	0	0	Reserved
CHTro	15	When set, when this trigger fires, generate a single cycle strobe on *TC_ChipTrigOut*.	R/W	0	Required
PDTro	14	When set, when this trigger fires, generate a single cycle strobe on *TC_ProbeTrigOut*.	R/W	0	Required
Impl	13:11	These bits are reserved for implementation specific trigger sources (internal or external to the TCB). Refer to the processor specification for the format and definition of this field.		0	Optional
0	10:7	Reserved. Must be written zero, reads back zero	0	0	Reserved
DM	6	When set, this Trigger will fire when a rising edge on the Debug mode indication from the core is detected. The write value of this bit always controls when this trigger will fire. If this trigger fires because this DM field is set, i.e., this is the cause of the trigger firing, then this can be determined by reading this DM field. If the DM field was written 1, then a read value of 1 indicates that this trigger has fired since the last write. Note that the action from a firing trigger could have been suppressed, and therefore, reading this field would be the only definite way to tell if the trigger fired and whether this was the cause. This special read value is valid until the *TCBTRIGx* register is written. Note that if the write value was 0 the read value is always 0.	R/W	0	Optional

Table 5.16 *TCBTRIGx* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Names	Bits				
CHTri	5	When set, this Trigger will fire when a rising edge on *TC_ChipTrigIn* is detected. The write value of this bit always controls when this trigger will fire. If this trigger fires because this CHTri field is set, i.e., this is the cause of the trigger firing, then this can be determined by reading this CHTri field. If the CHTri field was written 1, then a read value of 1 indicates that this trigger has fired since the last write. Note that the action from a firing trigger could have been suppressed, and therefore, reading this field would be the only definite way to tell if the trigger fired and whether this was the cause. This special read value is valid until the *TCBTRIGx* register is written. Note that if the write value was 0 the read value is always 0.	R/W	0	Required
PDTri	4	When set, this Trigger will fire when a rising edge on *TC_ProbeTrigIn* is detected. The write value of this bit always controls when this trigger will fire. If this trigger fires because this PDTri field is set, i.e., this is the cause of the trigger firing, then this can be determined by reading this PDTri field. If the PDTri field was written 1, then a read value of 1 indicates that this trigger has fired since the last write. Note that the action from a firing trigger could have been suppressed, and therefore, reading this field would be the only definite way to tell if the trigger fired and whether this was the cause. This special read value is valid until the *TCBTRIGx* register is written. Note that if the write value was 0 the read value is always 0.	R/W	0	Required

Table 5.16 *TCBTRIGx* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read / Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Names</th><th>Bits</th></tr>
<tr><td>Type</td><td>3:2</td><td>Trigger Type: The Type indicates the action to take when this trigger fires. The table below show the Type values and the corresponding Trigger action.
<table><tr><th>Type</th><th>Trigger action</th></tr><tr><td>00</td><td>Trigger Start: Trigger start-point of trace.</td></tr><tr><td>01</td><td>Trigger End: Trigger end-point of trace.</td></tr><tr><td>10</td><td>Reserved. Has no effect</td></tr><tr><td>11</td><td>Trigger Info: No action trigger, only for trace info.</td></tr></table>The action is to set or clear the TCBCONTROLB$_{EN}$ bit. A Start trigger will set TCBCONTROLB$_{EN}$, an End trigger will clear TCBCONTROLB$_{EN}$. Prior to revision 6.00 of the PDtrace architecture, the value of ‘10’ indicated a Center trigger. This has been deprecated as of revision 6.00 and is reserved for future use.
If Trace is set, then a TF6 format is added to the trace words. For Start and Info triggers this is done before any other TFs in that same cycle. For End triggers, the TF6 format is added after any other TFs in that same cycle.
If the TCBCONTROLB$_{TM}$ field is implemented it must be set to Trace-To mode (00), for the Type field to control on-chip trace fill.
The write value of this bit always controls the behavior of this trigger.
When this trigger fires, the read value will change to indicate if the trigger action was ever suppressed. If so the read value will be 11. If the write value was 11 the read value is always 11. This special read value is valid until the TCBTRIGx register is written.
If the condition is not true, i.e., either the trigger did not fire, or it fired and the action was not suppressed, then it is valid for the read value to read anything but 11.</td><td>R/W</td><td>0</td><td>Required</td></tr>
<tr><td>FO</td><td>1</td><td>Fire Once. When set, this trigger will not re-fire until the TR bit is de-asserted. When de-asserted this trigger will fire each time one of the trigger sources indicates trigger.</td><td>R/W</td><td>0</td><td>Required</td></tr>
<tr><td>TR</td><td>0</td><td>Trigger happened. When set, this trigger fired since the TR bit was last written 0.
This bit is used to inspect if the trigger fired since this bit was last written zero.
When set, all the trigger source bits (bit 4 to 13) will change their read value to indicate if the particular bit was the source to fire this trigger. Only enabled trigger sources can set the read value, but more than one is possible.
Also, when set, the Type field and the Trace field will have read values which indicate if the trigger action was ever suppressed by a higher priority trigger.</td><td>R/W0</td><td>0</td><td>Required</td></tr>
</table>

5.13 Reset State

Reset state for all register fields is entered when one or both of the following two things happen:

1. *ETT_SoftReset* input is set high.

2. *ETT_TRST_N* input is set low.

Most fields can be reset synchronously on the next rising edge of *ETT_TCK*.

The fields $TCBCONTROLA_{On}$ and $TCBCONTROLB_{EN}$ should be reset asynchronously on any of the above two events. Internal registers in the core-clock domain that need to reset must treat *ETT_SoftReset* and *ETT_TRST_N* as asynchronous reset inputs. It is not guaranteed that the core-clock is running when either of the two resets are asserted. For synchronous register reset, the reset event must be remembered until the core-clock starts running.

Please see Chapter 1, “Trace Control Block TAP Interface” on page 39 for a description of the *ETT_* pins.

5.14 TCB Registers in a Processor Implementing the MT ASE

In the presence of MT (Multi-Threading), there are potentially multiple TAP controllers in the processor, one per VPE (Virtual Processing Element). But such a processor only has a single pipeline, hence only implements a single PDtrace interface and a single TCB (Trace Control Block). Hence, there is only a single copy of the TCB registers as well on the core. In this situation, the TCB registers may be written from any one of the TAP controllers on the core. In the situation that more than one TAP controller is instructed to write a TCB register in the same instruction sequence from a probe, the write from TAP0 will succeed.

Chapter 6

PDtrace™ Output Trace Formats

One of the two main functions of the TCB is to capture trace information and send it to on-chip or off-chip trace memory. This trace information is then analyzed by the trace reconstruction software in the debugger. Since tracing the entire run of a program can require a lot of storage, compression of trace information is a desirable goal. While the trace information undergoes one level of compression in the core, further compression is possible before the trace information is stored to trace memory by the TCB. The TCB achieves this compression using a number of trace formats which eliminate the storage of unnecessary trace bits in each cycle. This section describes these formats. In PDtrace revision 05.00 and higher, a mechanism is added to decouple load/store cache miss indications from data tracing. This is done by augmenting existing trace formats with load/store hit/miss information (see Section 6.3.2.1). The load/store miss indication information is always sent at a fixed offset from the INSCOMP message for that particular instruction. The offset is implementation dependent (2 for the MIPS 34K core family). Since the offset is always fixed, we do not need a mechanism at startup to identify the offset.

Note that the description of the trace formats refers to PDtrace interface signals. Hence, to fully understand the intent of some of these trace formats, the reader must have a basic understanding of these signals or have access to the PDtrace specification document.

6.1 Single-Pipe Tracing Formats

The formats discussed in this section are relevant only when the core or processor being traced is a single-issue, i.e., single pipeline implementation. The multi-pipeline case is discussed in 6.3 “Multi-Pipe Tracing Formats”.

6.1.1 Trace Format 1 (TF1)

A processor stall is identified when *InsComp[2:0]* is 000, *TType[2:0]* is 000, and no overflow happens. When the processor is stalled, no execution trace information needs to be recorded except that this was a stall cycle. This can be done efficiently using a single bit “1” for this format. This is Trace Format 1 (TF1) as show in Figure 6.1. Note that this stall information is needed only when tracing is used to account for all execution cycles, i.e., cycle-accurate tracing ($TCBCONTROLB_{CA}$ = 1, see 5.2 “TCBCONTROLB Register”). However, TF1 generation is suppressed if the processor executes a WAIT instruction. Once the processor exits the WAIT state, TF1 messages resume.

Note that when parsing a trace format sequence, if the first bit of the trace format is a one, then this is TF1 and the next bit is the first bit of the next trace format.

Figure 6.1 TF1 (Trace Format 1)

6.1.2 Trace Format 2 (TF2)

A study of program traces shows that with only PC tracing enabled, nothing of significance needs to be captured a large percentage of the time. For instance, when *TType[2:0]* is **NT** (000), i.e., **N**o data **T**race, there is nothing to be traced. So, when *TType[2:0]* is **NT** and *InhibitOverflow* is 0, the only significant trace output is *InsComp[2:0]*. Having used a single bit value of “1” for TF1, we indicate the combination of non-zero *InsComp[2:0]*, zero *TType[2:0]*, and zero overflow in two bits (10_2). The next three bits of the format are the value of *InsComp[2:0].* This trace format with five bits is called Trace Format 2 (TF2), as shown in Figure 6.2.

Figure 6.2 TF2 (Trace Format 2 Single-Pipe)

Revision 4.00 (and higher) of the PDtrace specification introduces the ability to trace instruction fetch misses and to tag an instruction that might be a function call or return. These are fundamental properties that could impact most instructions in the stream that are represented by a non-zero InsComp value. Therefore, TF2 can optionally be augmented by two bits to trace out this information. These bits are optional and only traced when specifically requested by the user via bit TLSIF (bit 11) in TCBCONTROLB register. Hence, for correct interpretation of the trace formats, the trace reconstruction software must be told whether or not these 2 bits are present in each relevant trace format. This impacts other formats well, and will be discussed in each sub-section separately. The two optional bits of the TF2 format are shown in Figure 6.3.

- The Im bit indicates the Instruction miss for this instruction in the instruction cache.
- The optional Fcr bit indicates that this instruction is potentially a function call or return instruction.

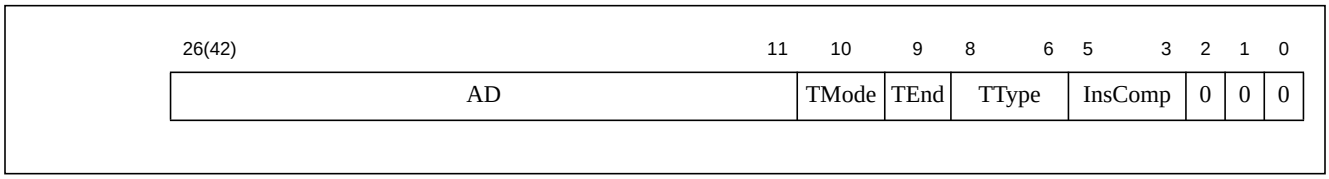
Figure 6.3 TF2 with Optional Bits (Trace Format 2 Single-Pipe)

6.1.3 Trace Format 3 (TF3)

When *TType[2:0]* is not **NT** (000) and there is no overflow, all trace information needs to be captured. This is the TF3 format shown in Figure 6.4. The *DataOrder[2:0]* value is an exception in that it only needs to be captured on the last cycle of a **D**ata **T**race (**DT** for the *TType[2:0]* value). Hence, a slight distinction is made between this format TF3 (which excludes the *LoadOrder[2:0]* value, see Figure 6.4), and the format TF4 (which includes the *DataOrder[2:0]* value, see Figure 6.7). This shows a data order value of 4 bits, but this is implementation-dependent and the number of bits in the DataOrder field is preset by the implementation in the $TCBCONTROLC_{NumDO}$ bits. The total length of the format increases by the corresponding number of bits.

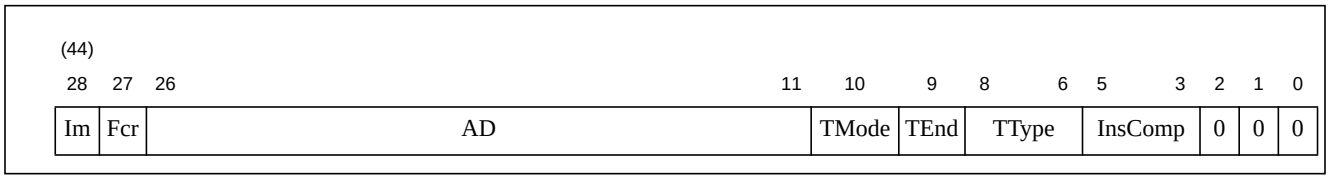
TF3 is distinguished from TF2 by having 000_2 on the first three bits. TF3 may be either 27 or 43 bits wide, depending on whether 16 or 32 bits is specified by the $TCBCONTROLA_{ADW}$ field. (See 5.1 “TCBCONTROLA Register”).

Figure 6.4 TF3 (Trace Format 3 Single-Pipe)



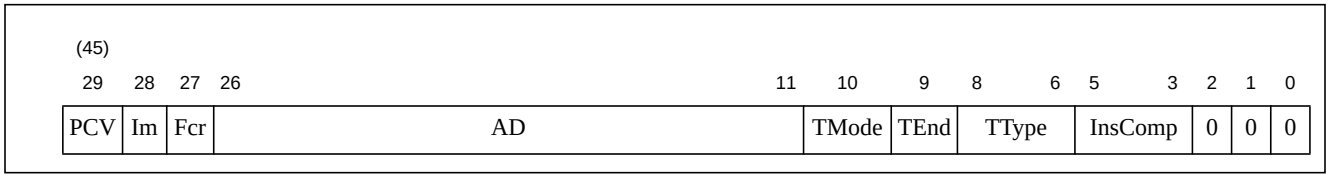
Revision 4.00 (and higher) of the PDtrace specification introduces the ability to trace instruction fetch misses and to tag an instruction that might be a function call or return. These are fundamental properties that could impact most instructions in the stream that are represented by a non-zero InsComp value. Therefore, TF3 can optionally be augmented by two bits to trace out this information. These bits are optional and only traced when specifically requested by the user. Hence, the trace reconstruction software must be told whether these bits are present. This impacts other formats well, and will be discussed in each sub-section separately. The two optional bits of the TF3 format are shown in Figure 6.5.

Figure 6.5 TF3 with Optional Bits (Trace Format 3 Single-Pipe)



Revision 6.00 (and higher) of the PDtrace specification introduces the ability to trace performance counter values. If this feature is enabled by the user, this information is traced through TF3, which can be optionally augmented by one bit. This expanded version of the TF3 format is show in Figure 6.6. If the PCV bit is set to zero, reconstruction software must interpret the trace format as before. If the PCV bit is set to one, reconstruction software must interpret the AD bits of the format as the value of the performance counter. In addition, the TType must be set to DT, and TEnd must be set to zero.

Figure 6.6 TF3 with Optional Performance Counter and other bits (Trace Format 3 Single-Pipe)

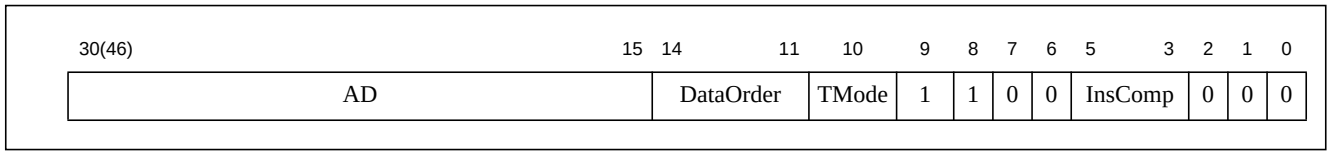


6.1.4 Trace Format 4 (TF4)

The TF4 format is shown in Figure 6.7. TF4 covers the case when *TType[2:0]* is set to **DT** and *TEnd* is set to 1, that is, the last cycle of the current data trace. This is shown in Figure 6.7, where the pattern on bits [9:6] distinguishes TF4 from TF3. Bits [8:6] are equal to 001_2 for a *Type[2:0]* value of **DT** and bit 9 has a value of 1 for *TEnd*.

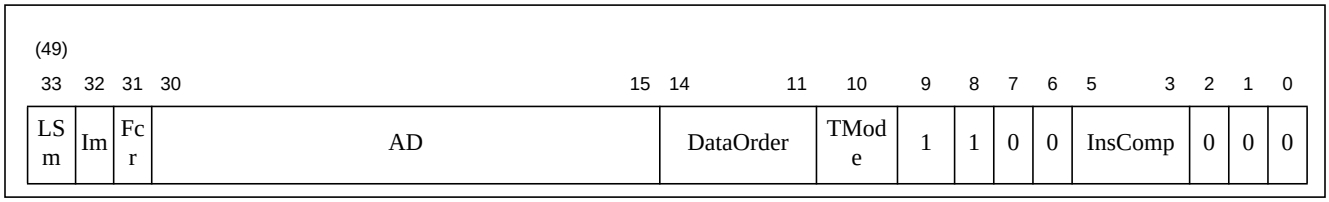
Note that the TF4 format will be used for the last cycle of both Load and Store Data transmission, a small inefficiency.

Figure 6.7 TF4 (Trace Format 4 Single-Pipe)



Revision 4.00 (and higher) of the PDtrace specification introduces the ability to trace instruction fetch misses, load/store data misses, and to tag an instruction that might be a function call or return. Therefore, TF4 can optionally be augmented by three bits to trace out this information. These bits are optional and only traced when specifically requested by the user. Hence, the trace reconstruction software must be told if these 3 bits are present. The optional bits of the TF4 format are shown in Figure 6.8. For non-coherent MIPS cores, only this format includes the LSm bit, that is the bit that indicates a possible load/store data cache miss. This is because a data miss is associated with the transmitted data rather than the instruction that caused the miss.

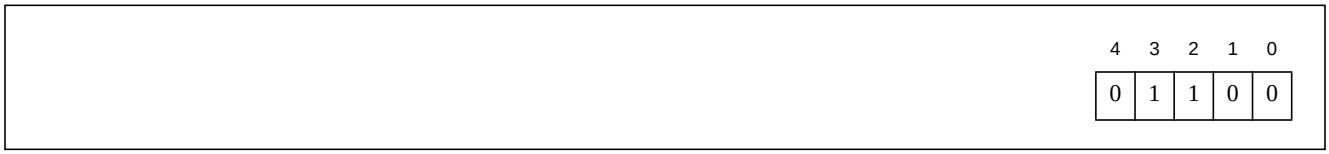
Figure 6.8 TF4 with Optional Bits (Trace Format 4 Single-Pipe)



6.1.5 Trace Format 5 (TF5)

When an overflow happens all other trace values are undefined and hence all current cycle trace values can be discarded. (When an overflow does occur, the trace always sends a full PC value in the next cycle. This is used for resynchronizing to the execution path.) The Trace Format 5 (TF5) shown in Figure 6.9 indicates an overflow. Revision 4.00 (and higher) of the PDtrace specification adds one bit to this format for a total of 5 bits.

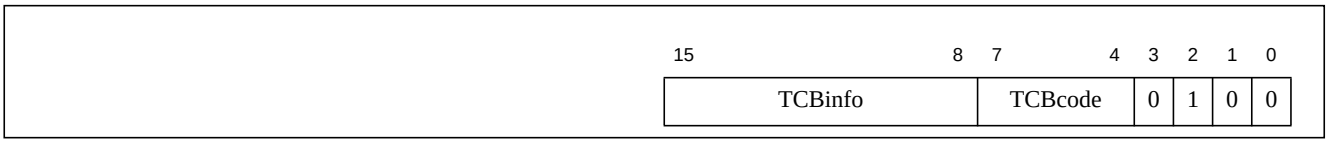
Figure 6.9 TF5 (Trace Format 5)



6.1.6 Trace Format 6 (TF6)

Trace Format 6 (TF6) shown in Table 6.10 is provided to the TCB to transmit information that does not directly originate from the cycle by cycle trace data. That is TF6 can be used by the TCB to store any information it wants in the trace memory, within the constraints of the specified format. This information can then be used by software for any purpose. For example, TF6 can be used to indicate a special condition, trigger, semaphore, breakpoint, or break in tracing that is encountered by the TCB.

Figure 6.10 TF6 (Trace Format 6)



The definition of TCBcode and TCBinfo is shown in Table 6.1.

Revision 4.00 (and higher) of the PDtrace specification uses two of the TCBcode fields to indicate that Instruction or Data Hardware Breakpoints were caused by the instruction in the trace format immediately preceding this TF6 format. Whether the trigger caused by the breakpoint turned trace off or on is indicated by the appropriate TCBinfo field value. Note that if the processor is tracing and trace is turned off this would be passed on to the external trace memory appropriately. If the processor is not tracing, and trace is turned on by a hardware breakpoint, then this record would show up in trace memory as the first instruction to be traced (it is also the one that triggered trace on). If tracing is ongoing and other triggers continue to keep turning on trace, then this would show up as a TF6 in trace memory. Revision 5.00 (and higher) of the PDtrace specification define an additional bit in TCBInfo, when TCBcode ‘1001’ is used (see Table 6.1).

Table 6.1 TCBcode and TCBinfo fields of Trace Format 6 (TF6)

TCBcode	Description	TCBinfo
0000	**Trigger Start:** Identifies start-point of trace. TCBinfo identifies what caused the trigger.	Cause of trigger. Taken from the Trigger control register generating this trigger.
0100	**Trigger End:** Identifies end-point of trace. TCBinfo identifies what caused the trigger.	
1000	**Reserved.** This value used to indicate a trigger center. Starting from PDtrace rev 6.00, this value is reserved for future use.	
1100	**Trigger Info:** Information-point in trace. TCBinfo identifies what caused the trigger.	
0001[1]	**No trace cycles:** Number of cycles where the processor is not sending trace data, but a stall is not requested by the TCB. This can happen when the processor, during its execution, switches modes internally that take it from a trace output required region to one where trace output was not requested. For example, if it was required to trace in User-mode but not in Kernel-mode, then when the processor jumps to Kernel-mode from User-mode, and an internal FIFO is emptied, then the processor stops sending trace information. In order to maintain an accurate account of total execution cycles, the number of such no-trace cycles have to be tracked and counted. This TCBcode does this tracking.	Number of cycles (All zeros is equal to 256). If more than 256 is needed, the TF6 format is repeated.
0101[1]	**Back stall cycles:** Number of cycles when no trace information was sent, for whatever reason.	
1001	Instruction or Data Hardware Breakpoint Trigger: Indicates that one or more of the instruction or data breakpoints were signalled and caused a trace trigger. Bit 8 of the TCBinfo field indicates whether it was an instruction (0) or data (1) breakpoint that caused the trigger. Bit 9 indicates whether or not trace was turned off (0) or on (1) by this trigger. Bits 13:10 encodes the hardware breakpoint number. Bit 14 indicates if tracing from the coherence manager was affected (1) or not (0). When tracing is turned off, a TF6 will be the last format that appears in the trace memory for that tracing sequence. The next trace record should be another TF6 that indicated a trigger on signal. It is important to note that a trigger that turns on tracing when tracing is already on will not necessarily get traced out, and is optional depending on whether or not there is a free slot available during tracing. Similarly, when tracing is turned off, then a trigger that turns off tracing will not necessarily appear in trace memory. Finally, if multiple breakpoints cause trigger actions, only one of the matching breakpoints is encoded in bits 13:10, and the trigger action is reported in bit 9.	Values are as described.

Table 6.1 TCBcode and TCBinfo fields of Trace Format 6 (TF6) (Continued)

TCBcode	Description	TCBinfo
1101	Reserved for future use	Undefined
0010, 0110 1010		
1110	Used for processors implementing MIPS MT ASE, see format TF7	TC value
xx11	TCB implementation dependent	Implementation dependent

1. TF6 formats with this TCBcode is not transmitted when $TCBCONTROLB_{CA}$ is 0

6.2 Format Enhancements for the MT ASE

In the presence of hardware-based multi-threading such as that provided by the MT ASE, there needs to be a method to indicate the thread id (or TC, thread context) for every traced instruction. This is possible in one of two ways.

1. The first method would typically be used when the multithreading method is coarse-grained or block-based, that is, instructions from a single thread are executed for a while before switching to another thread. In this type of scheme, it would suffice to trace out the thread id every time it changes and continue tracing instructions until there is a context switch. At which point, the new thread id is traced, and so on. The thread id thus traced is done using trace format 7 (TF7) illustrated in this section.

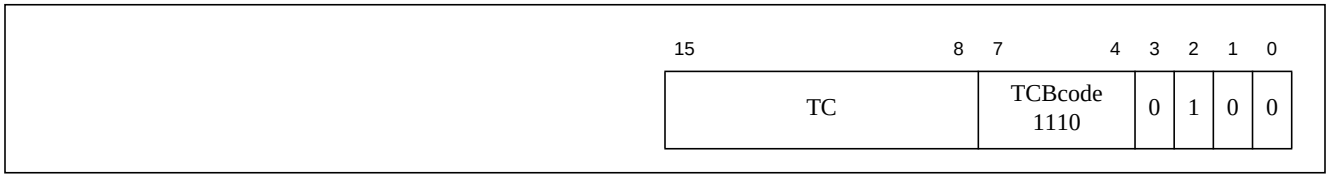
2. The second method is used when instructions from multiple threads are interleaved with a finer granularity. In this situation, the thread id might change every cycle, or in the case of a scheme like SMT (SImultaneous multi-threading), different instructions issued every cycle might belong to different thread contexts. In this situation, the thread id must be traced with every traced instruction. This might add significantly to the amount of trace data, but there is no avoiding this extra burden. In this situation, every trace format discussed this far, with the exception of TF1, TF5 and TF7, will be prefixed with a number of bits needed for the thread context value.

Bits in the TCB control register MTtrace determine which method is chosen, as well as the option to not trace the thread id in a processor implementing the MT ASE.

6.2.1 Trace Format 7 (TF7)

Trace Format 7 (TF7) shown in Table 6.11 is provided to the TCB to transmit information about the current Thread ID and is only used by a processor that implements the MT ASE. This format is used to indicate that the formats being sent following this one all belong to the indicated Thread id. Note that this is a sub-format of TF6. with a TCB-code value of 1110 and with the TC id value in the TCBinfo field.

Figure 6.11 TF7 (Trace Format 7)



6.2.2 TF2--TF4 Augmented for MT ASE

Up to 8 bits for the TC value is pre-pended to formats TF2, TF3, and TF4. The figure below illustrates the example of pre-pending 4 bits to format TF2 to support a hypothetical 34K core. Three bits in the TCB control register TCbits indicates how many TC id bits are needed for a particular core. In this example, the TCbit value would be set to four.

Figure 6.12 TF2 with Optional Bits and TC id Bits (Trace Format 2 Single-Pipe)

6.3 Multi-Pipe Tracing Formats

A processor with multiple pipelines requires additional support for sending trace information to trace memory. The TCB can perform some combining and the kind of format crunching as shown in the single-pipe case to reduce the number of bits that are sent out each cycle. If there are k pipelines within the core, 1, 2,... k, then for each cycle, the TCB generates a trace format from each pipeline, in that respective order. The external software programmer must refer to the User's Guide for that core to determine the order of the pipelines as hooked up to the PDtrace™ interface.

The trace format TF1 is usable by the TCB without change for multi-pipe tracing. The TF1 format indicates that the specific pipe did not complete an instruction and had no data to send.

TF5 is a common format. That is, all the pipes have to flush the trace buffer when just one of them has overflowed. Hence, a single instance of TF5 will suffice to cover all the 1..k pipeline stages. The trace reconstruction software must take this into account as it parses the trace formats in trace memory.

The TF6 format is also usable by the TCB without change, and as a common format. A TF6 format can be used after all the formats for the respective pipelines have been sent. Note that if needed, pipeline-specific information can be encoded within the TF6 format bits.

6.3.1 Multi-Pipe Trace Format 2-4 (TF2, TF3, TF4)

The TF2, TF3, and TF4 formats need the additional *PgmOrder[2:0]* value for multi-pipeline tracing. The **PgmOrder** field is added to all of them, right after the **InsComp** field, as shown in Figure 6.13, Figure 6.14, and Figure 6.15. The **PgmOrder** field is 3 bits wide to allow up to 8 pipelines. The number of processor pipelines is specified in the $TCBCONFIG_{PiN}$ field. See 5.7 "TCBCONFIG Register (Reg 0)" on page 74.

Figure 6.13 TF2 (Trace Format 2 Multi-Pipe)

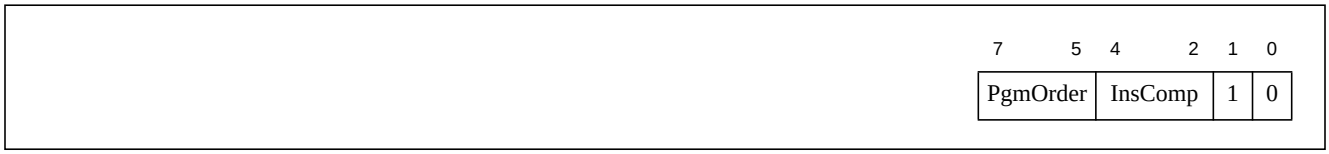
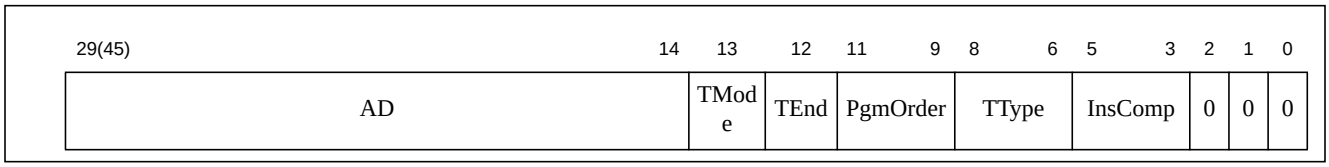
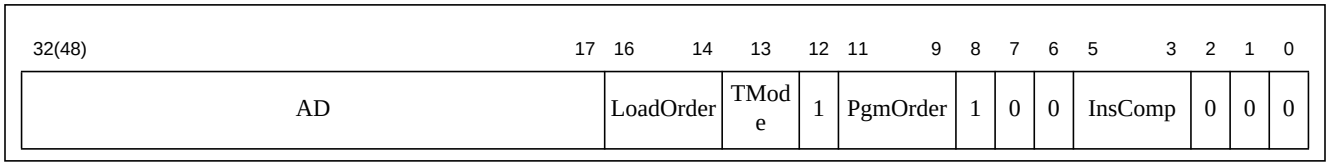


Figure 6.14 TF3 (Trace Format 3 Multi-Pipe)



TF4 for multi-pipe trace is defined as was the case for single-pipe trace. In the example in Figure 6.15, the TEnd bit (bit 12) is set, and the TType field (bits 8:6) is set to **DT** (100_2).

Figure 6.15 TF4 (Trace Format 4 Multi-Pipe)



6.3.2 Trace Format Extensions for Coherent Systems

The PDtrace architecture requires the coherent synchronization Id to be traced out from each core in a coherent system to allow correlation between requests from a core with transactions at the CM. The exact implementation of how this information is made available is highly dependent on the particular core on which it is implemented. We describe one mechanism that is implemented in the 34K core for CMP and extends every existing trace format by between 1 and 4 bits in Section 6.3.2.1.

6.3.2.1 Expanding existing trace formats

The first mechanism expands trace formats TF2,TF3,and TF4. Each of these formats is expanded by one to four bits. Each instruction that is capable of generating a bus request (“LSU” instruction) adds at least two bits. All non-LSU instructions add a single bit (0) to the end of the trace formats. An LSU instruction that hits in the cache adds two bits a “10”. If the instruction misses in the cache, it adds four bits - 11XY, where XY represent the COSId. The hit/miss/COSId information for an LSU instruction is always sent after the instruction completion message for that instruction has been sent. Specifically, it is always attached to the second LSU instruction after the original instruction. For the 34K, this guarantees that the hit/miss information is available at the time it needs to be sent out. Note: An implementation may choose to treat the LSm bit in a TF4 packet as a ‘don’t care’. Reconstruction software must not rely on the accuracy of this value to get a data cache hit/miss indication.

Chapter 7

TCB Trace Word

After compression of data into the Trace Formats described in Chapter 6, “PDtrace™ Output Trace Formats” on page 83, the trace information must be streamed to either on-chip or off-chip dedicated trace memory. As seen previously, each of the major Trace Formats are a different size. This complicates the efficient storage of this information into a fixed-width on-chip memory. It also complicates the transmission of this data through a fixed width interface to off-chip memory. To simplify the memory overhead and pin bandwidth issues, the Trace Formats are first gathered into Trace Words of regular width. This section describes the format of these Trace Words.

7.1 Trace Word

A Trace Word (TW) is defined to be 64 bits wide. A TW has a 4 bit type indicator on bits [3:0], an optional 2 or 4 bits to indicate the origin or source of this trace word, and regular TF’s stacked up in the remaining 58 or 56 respective bits of the word. Figure 7.1, Figure 7.2, and Figure 7.3 show the 64-bit wide TW. The source bits are valuable and used in an environment where trace from multiple cores or different sources need to be combined and written into a single trace memory. The trace re-generation software can then use these bits to sort out which trace words belong to which core or other traced logic in the chip or SOC.

It is recommended to allow the number of source bits be a configuration option for a core. The value thus chosen is written to the two-bit TWSrcWidth field in the TCB Control register (Figure 5.2 on page 63). For non-zero source options, the value of source to be used is preset to 0 during configuration in the TWSrcVal field in the TCB Control Register. This value can be over-written by software if needed and changed from the default value of zero.

Note that in all Trace Word examples illustrated later in this chapter, it is assumed that the source field is zero. But those examples could have been constructed in a similar manner for source field widths of 2 and 4 bits without any loss of generality.

Figure 7.1 Trace Word with Zero Source Bits

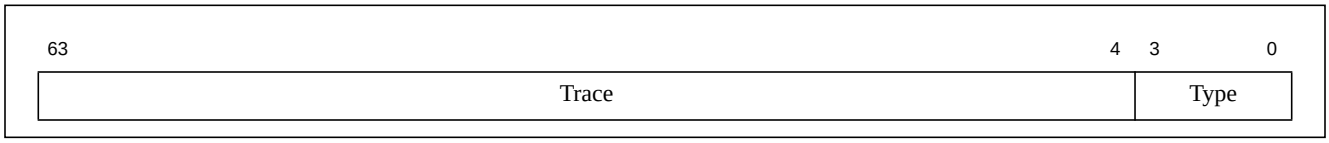


Figure 7.2 Trace Word with Two Source Bits

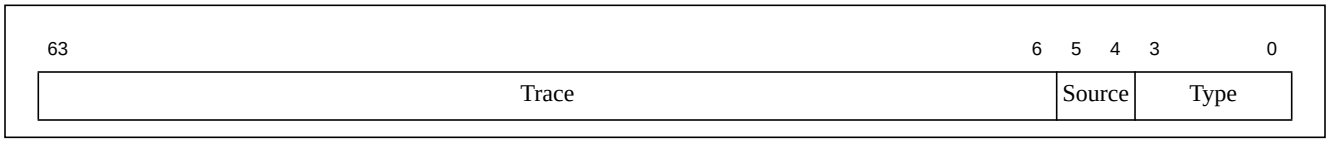
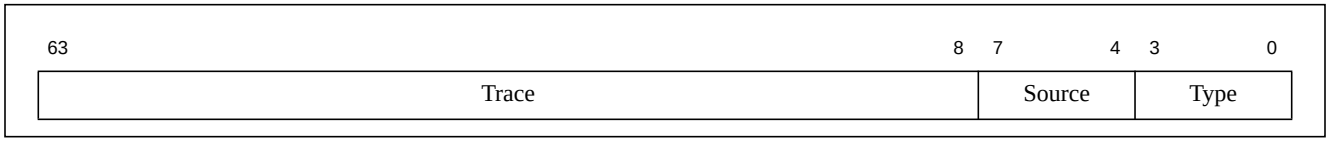


Figure 7.3 Trace Word with Four Source Bits



The **Trace** portion of a TW consists of one or more Trace Formats, TF1 through TF6. Note that trace formats TF1, TF2, TF5, and TF6 have a fixed size, while TF3 and TF4 can vary in size. The size of formats TF3 and TF4 is based on the value of the ADW bits. A further optimization is possible with an address value.That is, the redundant sign bits (in the upper address bits) can be optionally chopped from the formats, especially if the format straddles two TWs. This happens when *TType* is set to **TPC**, **TLA**, or **TSA**, TEnd is set to 1, and TMode is set to 0.

When *Type* is set to the **TMOAS** processor mode, this is traced as a TF3 with the **TMOAS** information in the AD field of that trace format type.

A TW is built by pushing in the TF's back to back until all 60 bits of the **Trace** field are used. If the last TF does not fit in **Trace**, it spills to the first bits of the **Trace** field in the next TW. The **Type** indicator is used to indicate where the first new TF starts in the new **Trace** field. This indirectly indicates the number of bits used to complete the TF from the previous TW.

Sometimes, when a TF cannot be completed in the remaining bits of a TW_n, it is more efficient to discard those bits of the TW_n and simply repeat all of them in the following TW_{n+1}. This is indicated in TW_{n+1} by setting **Type** to 1. When **Type** is 1, the first new TF of a TW starts at bit 0 in the **Trace** field. Since the previous TW_n ended with an uncompleted TF, a Type of 1 in TW_{n+1} instructs the decode software to discard the uncompleted TF in TW_n. Table 7.1 describes the word types for the TW.

Table 7.1 Trace Word Type field description

Decimal value of the Type field	The first new TF starts at this bit in the Trace field	Description
0	N/A	This TW does not carry any trace information. The **Trace** field is set to all zeroes. In the off-chip interface, the **Trace** field can be truncated to make the TW fit the bit-width of the off-chip interface. For on-chip trace, this TW is not to be stored in memory.
1	0	This indicates a situation where a new TF is started at the beginning of this TW. This can happen when: (1) a new trace is begun, (2) the TF in the previous TW was completed, and (3) an incomplete TF at the end of the previous TW is discarded. If the last trace format of the previous TW was a TF3 with: TType set to **TPC**, **TLA** or **TSA**, TEnd set to 1 and TMode set to 0, and with at least one AD bit, then that is considered a completed TF format, and no bits are discarded from the previous TW.
2 - 14	(Type - 1) * 4	The partial TF from the previous TW is completed in this TW in the bits available before the first new TF, i.e., bits 0..((Type -1)*4)-1) in the Trace field. If extra bits are available after completing the straddling TF, the rest of the bits until the first new TF start are undefined. TF3 formats sending the last part of a relative address are allowed to cut the AD bits to only show the needed sign bits. This enables compression of sign-extended address or data bits when the TF3 straddles a TW.
15	No new TF[1]	The TF started in the previous TW could not be completed within 54 bits[2]. It might complete in this TW. But if it does not complete, then the next TW will have a **Type** value higher than one.

1. This will never happen with the currently defined TF's, as the longest TF is only 49 bits wide.
2. 54 bits is the maximum allowable bits used to complete a TF from a previous TW_{n-1}, if a new one is to start in TW_n. This is so because a **Type** value of 14 indicates the maximum bit position (bit 54) in the **Trace** field, where a new TF will start.

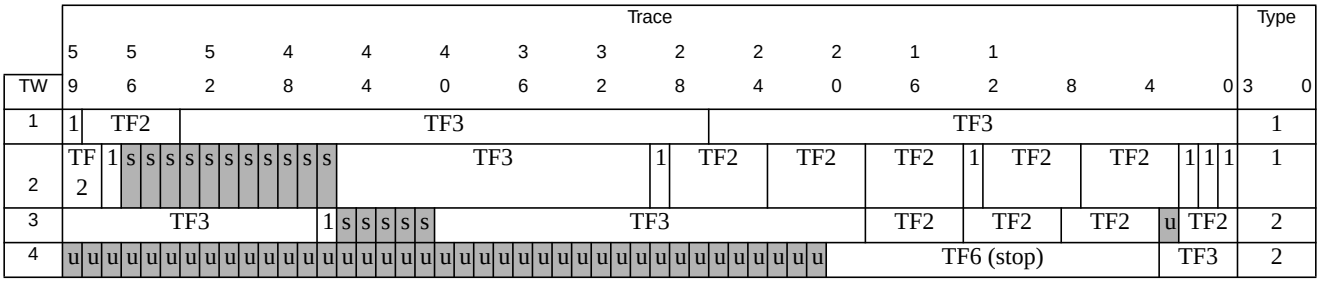
As an example of how a TW is built, consider the trace sequence shown in Table 7.2. In this example, the ADW value is assumed to be 16 bits wide (a zero value for $TCBCONTROLA_{ADW}$).

Table 7.2 Example Trace sequence

Cycle #	Trace Format	Cycle #	Trace Format
1	TF3 (16 significant AD bits)	2	TF3 (16 significant AD bits)
3	TF2	4	TF1
5	TF1	6	TF1
7	TF1	8	TF2
9	TF2	10	TF1
12	TF2	11	TF2
13	TF2	14	TF1
15	TF3 (5 significant AD bits)	16	TF1
17	TF2	18	TF2
19	TF2	20	TF2
21	TF3 (11 significant AD bits)	22	TF1
23	TF3 (6 significant AD bits)	24	TF6 (Stop indicator)

The TF sequence in Table 7.2 will create the set of TW's shown in Figure 7.4. The grayed out boxes with a "u" are unused bits. Grayed out boxes with an "s", show redundant sign-bits from a TF3 format. These sign-bits could not be compressed out, and must be included as part of the AD field. A "1" indicates the single bit of 1 in a TF1 format.

Figure 7.4 Trace Word from Example Trace in Table 7.2



In the example in Figure 7.4, the TF3 straddling TW_3/TW_4 have had insignificant sign bits cut from the full TF3 format. It is optional for TCB hardware to do this extra compression of TF3 formats, but TW decode software must always be designed to handle this extra compression.

7.1.1 Cycle Inaccurate Trace

The TF1 format is needed only when a sequence of the trace must show cycle-by-cycle behavior of the processor without missing any cycles. When the trace regeneration software only needs to show the sequence of instructions executed, the TF1 format which shows processor stall cycles can be omitted. In this latter situation, an additional optimization removes bit zero on the other TF's before storing to trace memory. The example trace sequence in Table 7.2

will then produce the TW's shown in Figure 7.5. Note that, to reconstruct the trace accurately, external software must know what type of tracing was enabled at the TCB.

Figure 7.5 Trace Word from Example Trace in Table 7.2 (No TF1 trace)

TW	Trace (bits 59 … 0)	Type (bits 3 … 0)
1	TF2 | TF2 | TF3 | TF3	1
2	TF3 | TF2 | TF2 | TF2 | TF2 | s s s s s s s s s s s | TF3 | TF2 | TF2 | TF2 | TF2	1
3	TF6 (stop) | s s s s s s s s s s | TF3 | s | TF3	6
4	u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u | TF6	2

In the example shown in Figure 7.5, the TF3 straddling TW_2/TW_3 have had insignificant sign bits cut from the full TF3 format. It is optional for TCB hardware to make this extra compression of TF3 format, but TW decode software must be able to handle this.

Additionally when not tracing for cycle accurate information, the TF6 formats TCBcode 0001 and 0101 are omitted from the Trace Words (not shown in Figure 7.4 and Figure 7.5). Cycle accurate versus cycle inaccurate tracing in controlled by the $TCBCONTROLB_{CA}$ bit.

7.1.1.1 Trace Word collection.

Figure 7.6 shows how the TCB builds the Trace Words using the Trace Formats cycle by cycle, using the trace information. Trace Words from Figure 7.4 are used.

Figure 7.6 Cycle-by-cycle Trace Word from Example Trace in Table 7.2

Cycle	TW	Trace (bits 59 … 0)	Type (bits 3 … 0)
1		free | TF3	1
2		free | TF3 | TF3	1
3		f | TF2 | TF3 | TF3	1
4	1	1 | TF2 | TF3 | TF3	1

Trace bit positions: 59, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20, 16, 12, 8, 4, 0; Type bit positions: 3, 0. Trace fields are listed from left (bit 59) to right (bit 0).

Cycle	TW	Trace	Type
5		free | 1	1
6		free | 1 | 1	1
7		free | 1 | 1 | 1	1
8		free | TF2 | 1 | 1 | 1	1
9		free | TF2 | TF2 | 1 | 1 | 1	1
10		free | 1 | TF2 | TF2 | 1 | 1 | 1	1
11		free | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
12		free | TF2 | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
13		free | TF2 | TF2 | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
14		free | 1 | TF2 | TF2 | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
15		free | s | s | s | s | s | s | s | s | s | s | s | TF3 | 1 | TF2 | TF2 | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
16		free | 1 | s | s | s | s | s | s | s | s | s | s | s | TF3 | 1 | TF2 | TF2 | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
	2	TF2 | 1 | s | s | s | s | s | s | s | s | s | s | s | TF3 | 1 | TF2 | TF2 | TF2 | 1 | TF2 | TF2 | 1 | 1 | 1	1
17		free | u | TF2	2
18		free | TF2 | u | TF2	2
19		free | TF2 | TF2 | u | TF2	2
20		free | TF2 | TF2 | TF2 | u | TF2	2
21		free | s | s | s | s | s | TF3 | TF2 | TF2 | TF2 | u | TF2	2
22		free | 1 | s | s | s | s | s | TF3 | TF2 | TF2 | TF2 | u | TF2	2
	3	TF3 | 1 | s | s | s | s | s | TF3 | TF2 | TF2 | TF2 | u | TF2	2
23		free | TF3	2
24	4	u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | TF6 (stop) | TF3	2

7.2 End of Trace Indication

In the examples in the previous section, the Trigger TF6 (stop: TCBcode == 0100) was used to indicate an End Trigger and this implied an end to the tracing as well. This stop trigger deasserts *TCBCONTROLB*$_{EN}$ and the TCB flushes out the current TW. However, the *TCBCONTROLB*$_{EN}$ bit can be deasserted for other reasons, and this trace end must be indicated externally using a different mechanism to distinguish this from the end-trigger case. The recommended method to accomplish this is to let the TCB fill the un-used bits in the last TW with zeroes. Note that nine bits of consecutive zeroes in the Trace field will be identified as a TF3 with no information, that is, InsComp and TType are both zero. This will never be ordinarily generated by the Trace Format generator, and can therefore be used as a end-of-trace indicator.

If less than nine bits remain in the last TW, then an incomplete TF is detected by trace software. After that no additional TW are generated by the TCB. This should not be a problem for trace re-generating software, as this is just like any other arbitrary cut in the trace stream.

7.3 On-chip Trace Memory Format

The on-chip trace memory is defined to be a 64-bit wide memory. The TW's defined in 7.1 "Trace Word", are stored in consecutive address locations. The trace memory is only written when a full TW is available, hence a new TW might not be written each cycle, since a new TW might not be created each cycle.

The memory image will exactly match the TW sequence shown in Figure 7.4 or Figure 7.5, depending on whether TF1 formats are included.

7.4 Probe Trace Word transmission

The Probe interface can support a *TR_DATA* bus width of 4, 8, or 16 bits. When a TW is ready to be sent, it is put on the *TC_Data* pins to the PIB. The PIB will feed the TW through on the available *TR_DATA* pins, starting with *TC_Data[n:0]* on the *TR_DATA[n:0]* utilized pins. Depending on the value of n, this will take 16, 8, or 4 transmissions. If a clock multiplier is used in the PIB, then 2, 4, 8, or 16 transmissions can be completed in one core clock cycle.

As long as no new TW is available for transmission, the *TC_Data* bus will show all zeros, allowing the PIB to keep transmitting this on the *TR_DATA* bits to also show all zeros.

On an 8 pin *TR_DATA* trace interface, running at core-clock frequency, the trace from the TW's in Figure 7.6 will look as shown in Figure 7.7 on the Probe IF. This assumes sufficient buffering to hold the TW's in the TCB when they become available for transmission, and a latency of one clock before the first part of an available TW on the *TC_data* bus, appears on the *TR_DATA* pins.

Figure 7.7 Cycle-by-Cycle *TR_DATA* (8-bit) of Example Trace in Table 7.2

Cycle	TR_DATA[7:0]
1	zero
2	zero
3	zero
4	zero
5	TW_1[7:0]
6	TW_1[15:8]
7	TW_1[23:16]
8	TW_1[31:24]
9	TW_1[39:32]
10	TW_1[47:40]

Cycle	TR_DATA[7:0]
11	TW_1[55:48]
12	TW_1[63:56]
13	zero
14	zero
15	zero
16	zero
17	zero
18	TW_2[7:0]
19	TW_2[15:8]
20	TW_2[23:16]

Cycle	TR_DATA[7:0]
21	TW_2[31:24]
22	TW_2[39:32]
23	TW_2[47:40]
24	TW_2[55:48]
25	TW_2[63:56]
26	TW_3[7:0]
27	TW_3[15:8]
28	TW_3[23:16]
29	TW_3[31:24]
30	TW_3[39:32]

Cycle	TR_DATA[7:0]
31	TW_3[47:40]
32	TW_3[55:48]
33	TW_3[63:56]
34	TW_4[7:0]
35	TW_4[15:8]
36	TW_4[23:16]
37	zero
38	zero
39	zero
40	zero

The probe sampling the *TR_DATA* pins should look for a non-zero transmission. When that happens, the following bits up to a collective count of 64-bits (i.e. including the first non-zero 4/8/16-bit value) will form a TW. After 64-bits, the probe should re-start looking for a new non-zero transmission. A non zero transmission can start at any time after a full TW is received.

Chapter 8

Trace Compression

This section is a discussion of compression techniques that may be used when tracing different values. The methods used are quite different for each “type" of value. For example, the PC may be sent as a delta from the previous PC address. Sometimes the full PC value needs to be sent when the trace process starts either at the beginning of tracing, after a buffer overflow, or for synchronization. In this case, the PC can be sent un-compressed, or some method such as bit-block compression can be used. The sections below discuss these various techniques as they correspond to the TMode bit value. Note that the single-bit TMode bit allows two ways in which to send the information being currently traced.

8.1 PC tracing

When TMode is zero, this implies that the delta of the PC value is transmitted. This delta is computed from the PC value of the instruction executed just before the branch target instruction (e.g., the instruction executed in the branch delay slot after a branch instruction). The computed delta is then right-shifted by one bit, since this bit is never used. Note that the value can be negative or positive, hence is a signed 16-bit value, and the upper bits need to be sign-extended before transmission.

PC_delta = (new_PC - last_PC) >> 1 (EQ 1)

If the width of the computed delta value is bigger than the width of the data field (ADW), then the lower bits are sent first, followed by the upper bits.

When the TMode value is one, this implies that the full PC value is transmitted. Depending on the width of the bus, this may take multiple cycles. Again, the first cycle transmits the least significant bits, and so on.

8.2 Load or Store Address Tracing

With a TMode zero value, the load address transmitted is a delta from the last transmitted load address. Stores are similar, where the computed delta is from the last transmitted store address. Note that the last load instruction can be a load instruction of any type, i.e., LB, LW, etc. The same is true for stores.

load_address_delta = current_load_address - last_load_address (EQ 2)

store_address_delta = current_store_address - last_store_address (EQ 3)

Note that the delta transmission is quite effective when the load or store addresses are increasing or decreasing sequentially.

With a TMode value of one, the value transmitted is the full address of either the load or the store. Depending on the width of the trace bus and the processor data width, this could take multiple cycles to transmit.

8.3 Load or Store Data Tracing

The data values are less prone to good compression techniques. But delta values and bit-block compression techniques might be useful in achieving some compression ratio. This revision of the PDtrace specification does not dictate any compression for data values. The TMode value of zero is reserved for a future compression scheme. And the TMode value of one is used to transmit the full data value.

8.4 Using Early TEnd Assertion

This technique was discussed in Table D.1. When tracing data address or value the tracing logic can optionally make a decision to cut off the trailing sign bits of the data and assert TEnd early, before all the bits of the address or data has been traced. For example, redundant sign bits need not be transmitted for accurate reconstruction of the data. Note that this data compression technique can be applied to any value traced in the AD field in the trace formats, be it PC address, load/store address, or load/store data. Also note that this technique is optional, but the software must be capable of handling this situation for implementations with PDtrace Specification 03.00 and higher.

Chapter 9

TCB Trigger Logic

The TCB is defined to optionally feature a trigger unit. Most of the actual implementation and functionality is implementation dependent, but if implemented, the base-line structure must be as defined in this section.

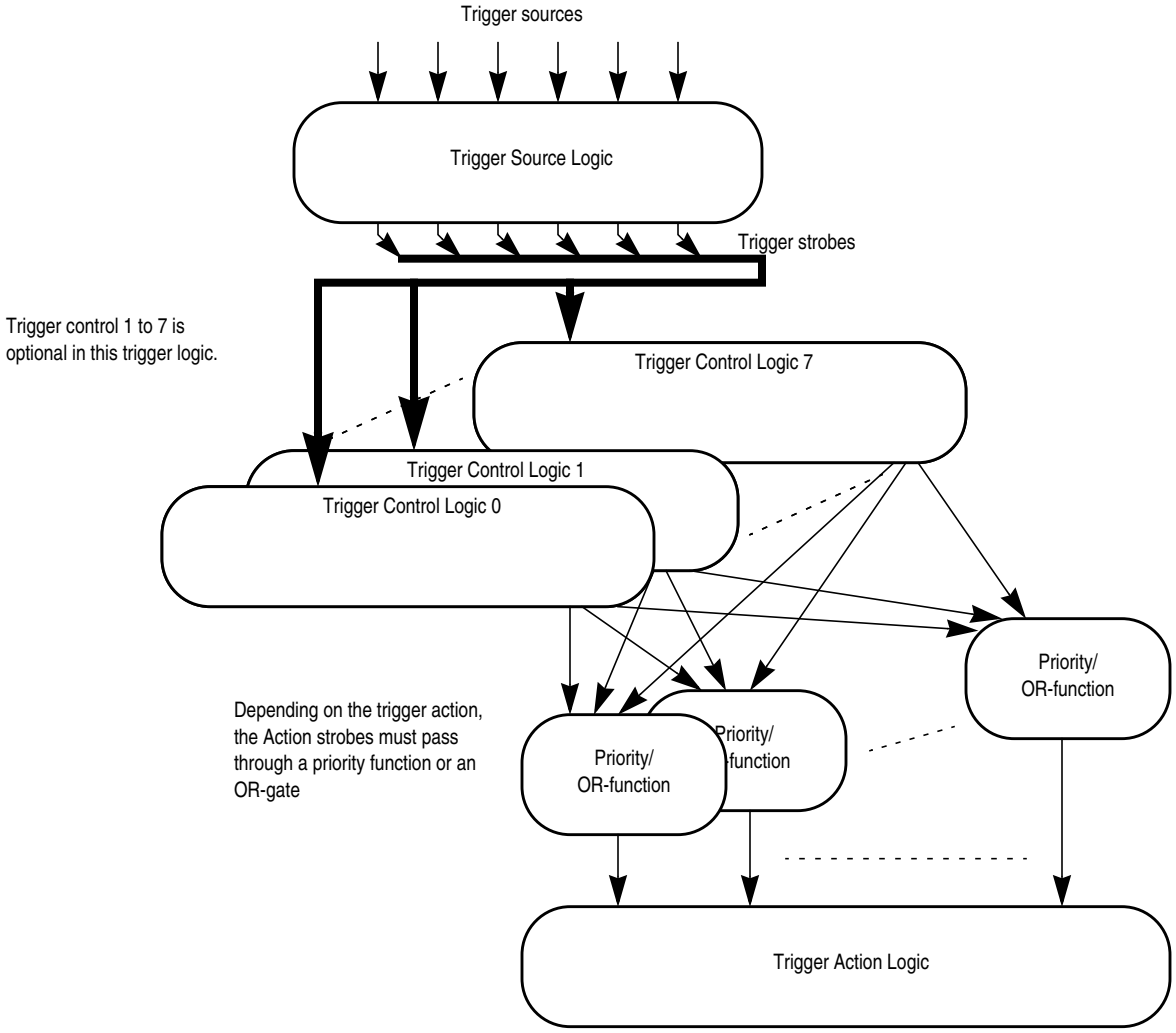
9.1 Trigger Logic Overview

The trigger logic is functionally split in three parts.

- Trigger Source logic.
- Trigger Control logic
- Trigger Action logic.

Figure 9.1 shows the functional overview of the trigger flow in the TCB.

Figure 9.1 TCB Trigger Processing Overview



9.1.1 Trigger Source Logic

A number of source events can be defined that cause a trigger to fire when the corresponding source condition is satisfied.

In this version of the TCB, three sources have been defined. These are the two trigger inputs *TC_ChipTrigIn* and *TC_ProbeTrigIn* (see 9.3 "TCB Trigger Input/Output Signals"), and the Debug Mode (DM) indication from the processor core. The input triggers are all rising-edge triggers, and the Trigger Source logic must convert the edge into a single cycle strobe to the Trigger Control logic.

9.1.2 Trigger Control Logic

Eight possible Trigger Control registers (*TCBTRIGx, x={0..7}*) are defined. Each of these registers controls a trigger fire mechanism. They can have each of the Trigger Sources as the trigger event and they can fire one or more of the Trigger Actions. This is defined in the Trigger Control register *TCBTRIGx* (see 4.9 "TCBTRIGx Register (Reg 16-23)").

9.1.3 Trigger Action logic

A number of possible trigger actions in this version of the TCB are:

- Two output trigger strobes, *TC_ChipTrigOut* and *TC_ProbeTrigOut*. These are explained in 9.3 “TCB Trigger Input/Output Signals”.
- The TF6 trace format as information output into trace memory. This is explained in 4.9 “TCBTRIGx Register (Reg 16-23)”. Also read 9.2 “Simultaneous Triggers”.
- The Start and End trigger actions. These are also explained in the sectionspointed to above. Earlier revisions of the PDtrace architecture (prior to revision 6.00) defined a center trigger action. This trigger action is no longer defined.

9.2 Simultaneous Triggers

Two or more triggers can fire simultaneously. The resulting behavior depends on trigger action set for each of them, and whether they should produce a TF6 trace information output or not. There are two groups of trigger actions: Prioritized and OR’ed.

9.2.1 Prioritized Trigger Actions

For prioritized simultaneous trigger actions, the trigger control register which has the lowest number takes precedence over the higher numbered *TCBTRIGx* registers. The oldest trigger takes precedence over everything.

The following trigger actions are prioritized when two or more *TCBTRIGx* registers fire simultaneously:

- Trigger Start and End type triggers (*TCBTRIGx*$_{\text{Type}}$ field set to 00 or 01), which will assert/deassert the *TCBCONTROLB*$_{\text{EN}}$ bit.
- Triggers which produce TF6 trace information in the trace flow (*TCBTRIGx*$_{\text{Trace}}$ bit is set).

Regardless of priority, the *TCBTRIGx*$_{\text{TR}}$ bit is set when the trigger fires, even if the trigger action was suppressed. If the trigger is set to only fire once (the *TCBTRIGx*$_{\text{FO}}$ bit is set), then the suppressed trigger action will not be possible until after *TCBTRIGx*$_{\text{TR}}$ is written 0.

If a Trigger action is suppressed by a higher priority trigger, then the read value, when the *TCBTRIGx*$_{\text{TR}}$ bit is set, for the *TCBTRIGx*$_{\text{Trace}}$ field will be 0 for suppressed TF6 trace information actions. The read value in the *TCBTRIGx*$_{\text{Type}}$ field for suppressed Start/End/ triggers will be 11. This indication of a suppressed action is sticky. If any of the two actions (Trace and Type) are ever suppressed for a multi-fire trigger (the *TCBTRIGx*$_{\text{FO}}$ bit is zero), then the read values in *TCBTRIGx*$_{\text{Trace}}$ and/or *TCBTRIGx*$_{\text{Type}}$, are set to indicate a suppressed action.

9.2.2 OR’ed Trigger Actions

The simple trigger actions CHTro, PDTro and CTATrg from each *TCBTRIGx* register’s action logic, are effectively OR’ed together to produce the final trigger. For example, one or more expected trigger strobes on *TC_ChipTrigOut* can disappear. External logic should therefore not rely on counting of strobes to predict a specific event unless simultaneous triggers are known not to occur.

9.3 TCB Trigger Input/Output Signals

Two sets of trigger input/outputs are defined on the TCB. One set is defined to be chip internal, and the other set is defined to be part of the probe interface. Table 9.1 shows the TCB signal names, and the related probe pin name for the probe trigger signals.

Table 9.1 TCB Trigger input and output

TCB pin name	Probe pin name	Description
TC_ChipTrigIn	N/A	Rising edge trigger input. The source should be on-chip. The input is considered async. I.e. double registered in the TCB.
TC_ChipTrigOut	N/A	Single cycle (relative to core clock) high strobe, trigger output. The target is supposed to be an on-chip unit.
TC_ProbeTrigIn	*TR_TRIGIN*	Rising edge trigger input. The source should be the Probe Trigger input. The input is considered async. I.e. double registered in the TCB.
TC_ProbeTrigOut	*TR_TRIGOUT*	Single cycle (relative to probe clock *TC_ProbeClk*) high strobe, trigger output. The target is supposed to be the Probes Trigger output.

Appendix A

Implementation-Specific PDtrace™ Enhancements for the MIPS32® 74K™ Cores

A.1 Tracing the 74K to Illuminate Pipeline Details and Execution Inefficiencies

The 74K core implements PDtrace revision 06.00 and higher. A bit in the trace control register specifies that 74K-specific trace information will be included in the trace stream. This is bit 28 in *TraceControl* and bit 28 in TCBCONTROLA. Setting this bit to a value of 1 implies that the 74K-specific tracing, that is described in this section will be output into the trace stream.

The following 74K-specific inefficiencies are traced to determine the cause of lost performance. This information is encoded into an expanded version of the INSCOMP field of TF2, TF3 and TF4. The field is expanded by one bit, and the expanded encodings identify potential performance bottlenecks. This increases the length of TF2, TF3 and TF4 by one bit.

- "Load/store cache miss information (INSCOMP=1000)
- "Branch/return mispredict information (INSCOMP=1001)
- "Replay (load consumer or branch likely or cacheop) (INSCOMP=1010)
- "Graduation stall due to backpressure (stall due to LSGB full and other) (INSCOMP=1011)

The re-encoded INSCOMP field is illustrated in Table A.1. The updated versions of the three trace formats are described next.

Table A.1 Expanded Instruction Type Completion Indicator (InsComp)

Value	Mnemonic	Description
0000	NI	No instruction completed this cycle. A "No Instruction" can happen due to a pipeline stall or when the instruction was killed (due to an exception).
00001	I	Instruction completed this cycle
0010	IL	Instruction completed this cycle was a load
0011	IS	Instruction completed this cycle was a store
0100	IPC	Instruction completed this cycle was a PC sync. The IPC value is used for the periodic output of the full PC value for synchronization. The tracing hardware should ensure that this is not done on an unpredictable branch, load, or store instruction.
0101	IB	Instruction branched this cycle. The three encoding (101, 110, 111) for branched instruction indicates a discontinuity in the PC value for the associated instruction. Note that it is only when the new PC can not be predicted from the static program flow that it is traced.
0110	ILB	Instruction branched this cycle was a load

Table A.1 Expanded Instruction Type Completion Indicator (Continued)(InsComp)

Value	Mnemonic	Description
0111	ISB	Instruction branched this cycle was a store
1000	NI74_LSM	No instruction completed this cycle - 74K - Load/Store Miss
1001	NI74_BMP	No instruction completed this cycle - 74K - Branch/return Mispredict
1010	NI74_RPL	No instruction completed this cycle - 74K - Instruction Replay
1011	NI74_GST	No instruction completed this cycle - 74K - Backpressure stall
1100-1111	-	Reserved for future use

A.1.1 Updated Trace Format 2 (TF2) for 74K specific information

If bit 28 in TraceControl (if trace is being controlled by software), or bit 28 in TCBControlA (if trace is being controlled through a probe) is set on a 74K core, TF2 is expanded by 1 bit. The two variants of TF2 in their expanded form are shown below. The difference between the regular TF2 and the expanded TF2 is the extra bit in the InsComp field.

Figure A.1 Expanded TF2 (Trace Format 2 Single-Pipe)

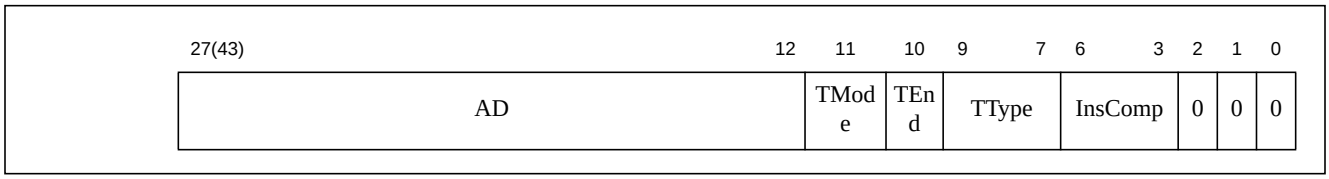
Revision 4.00 (and higher) of the PDtrace specification introduces the ability to trace instruction fetch misses and to tag an instruction that might be a function call or return. These are fundamental properties that could impact most instructions in the stream that are represented by a non-zero InsComp value. Therefore, TF2 can optionally be augmented by two bits to trace out this information. These bits are optional and only traced when specifically requested by the user. Hence, the trace reconstruction software must be told whether these bits are present. This impacts other formats well, and will be discussed in each sub-section separately. The two optional bits of the TF2 format are shown in Figure A.2.

Figure A.2 Expanded TF2 with Optional Bits (Trace Format 2 Single-Pipe)

A.1.2 Trace Format 3 (TF3)

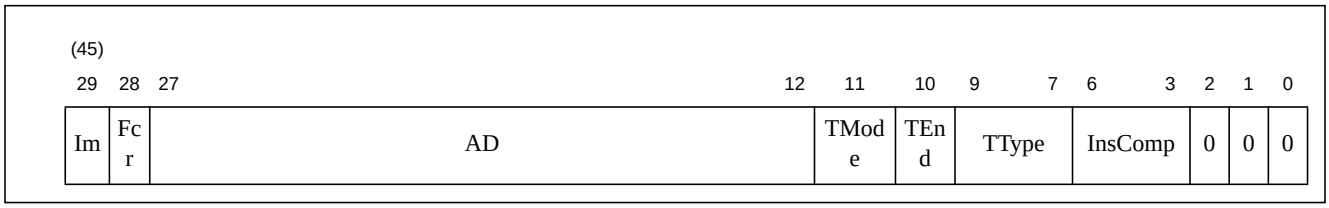
If bit 28 in TraceControl (if trace is being controlled by software), or bit 28 in TCBControlA (if trace is being controlled through a probe) is set on a 74K core, TF2 is expanded by 1 bit. The two variants of TF3 in their expanded form are shown below. The difference between the regular TF3 and the expanded TF2 is the extra bit in the InsComp field. The expanded TF3 may be either 28 or 44 bits wide, depending on whether 16 or 32 bits is specified by the *TCBCONTROLA*$_{\text{ADW}}$ field. (See 5.1 “TCBCONTROLA Register”).

Figure A.3 TF3 (Trace Format 3 Single-Pipe)



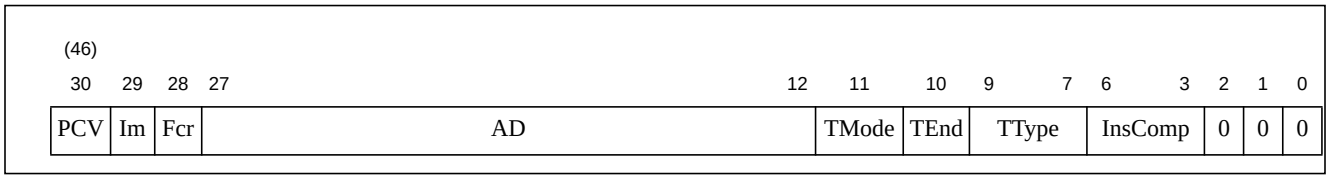
Revision 4.00 (and higher) of the PDtrace specification introduces the ability to trace instruction fetch misses and to tag an instruction that might be a function call or return. These are fundamental properties that could impact most instructions in the stream that are represented by a non-zero InsComp value. Therefore, TF3 can optionally be augmented by two bits to trace out this information. These bits are optional and only traced when specifically requested by the user. Hence, the trace reconstruction software must be told whether these bits are present. This impacts other formats well, and will be discussed in each sub-section separately. The two optional bits of the TF3 format are shown in Figure A.4.

Figure A.4 TF3 with Optional Bits (Trace Format 3 Single-Pipe)



Revision 6.00 (and higher) of the PDtrace specification introduces the ability to trace performance counter values. If this feature is enabled by the user, this information is traced through TF3, which can be optionally augmented by one bit. This version of the TF3 format is show in Figure A.5. If the PCV bit is set to one, reconstruction software must interpret the AD bits of the format as the value of the performance counter. In addition, the TType must be set to DT, and TEnd must be set to zero.

Figure A.5 Expanded TF3 with Optional Performance Counter and other bits (Trace Format 3 Single-Pipe)



A.2 Updated TF4 TCB Format to Handle 74K Core-Specific DataOrder and Inefficiency Information

The 74K core can have up to 21 outstanding loads and many other store operations in the system at any given time, hence the 4 bits currently being used in trace format TF4 is inadequate. Hence TF4 will be redone to use an additional fifth bit for the DataOrder field.

If bit 28 in TraceControl (if trace is being controlled by software), or bit 28 in TCBControlA (if trace is being controlled through a probe) is set on a 74K core, TF2 is expanded by 1 bit. The two variants of TF3 in their expanded form are shown below. The difference between the regular TF3 and the expanded TF2 is the extra bit in the InsComp field.

The TF4 format is shown in Figure A.6. TF4 covers the case when *TType[2:0]* is set to **DT** and *TEnd* is set to 1, that is, the last cycle of the current data trace. This is shown in Figure A.6, where the pattern on bits [9:6] distinguishes TF4 from TF3. Bits [8:6] are equal to 001_2 for a *Type[2:0]* value of **DT** and bit 9 has a value of 1 for *TEnd*.

Note that the TF4 format will be used for the last cycle of both Load and Store Data transmission, a small inefficiency.

PDtrace revision 06.00 and higher introduces an alteration to the number of bits needed for the DataOrder field. Since the 74K core can have up to 21 outstanding memory transactions, the original TF4 format with 4 bits for DataOrder would not suffice. Hence, if the core type is identified to be a 74K implementation, then the TF4 format is recognized as shown in the figures in this section.

Figure A.6 TF4 (Trace Format 4 Single-Pipe)

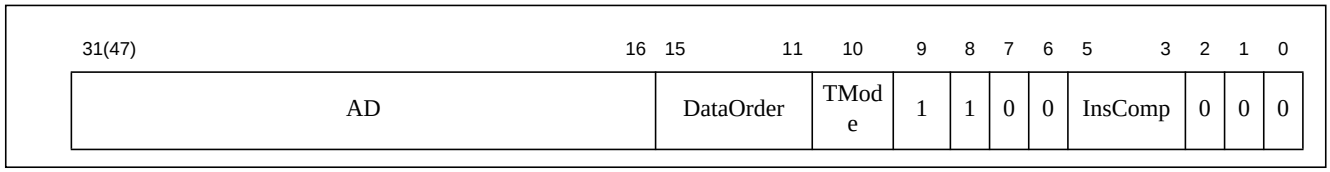
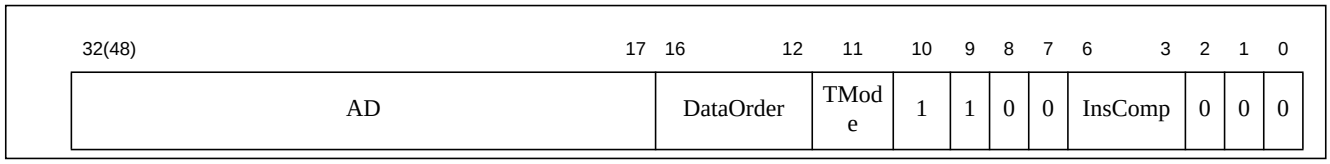


Figure A.7 Expanded TF4 (Trace Format 4 Single-Pipe)



Revision 4.00 (and higher) of the PDtrace specification introduces the ability to trace instruction fetch misses, load/store data misses, and to tag an instruction that might be a function call or return. Therefore, TF4 can optionally be augmented by three bits to trace out this information. These bits are optional and only traced when specifically requested by the user. Hence, the trace reconstruction software must be told if these 3 bits are present. The optional bits of TF4 are shown in Figure A.8. For non-coherent MIPS cores, only this format includes the LSm bit, that is the bit that indicates a possible load/store data cache miss. This is because a data miss is associated with the transmitted data rather than the instruction that caused the miss.

Figure A.8 TF4 with Optional Bits (Trace Format 4 Single-Pipe)

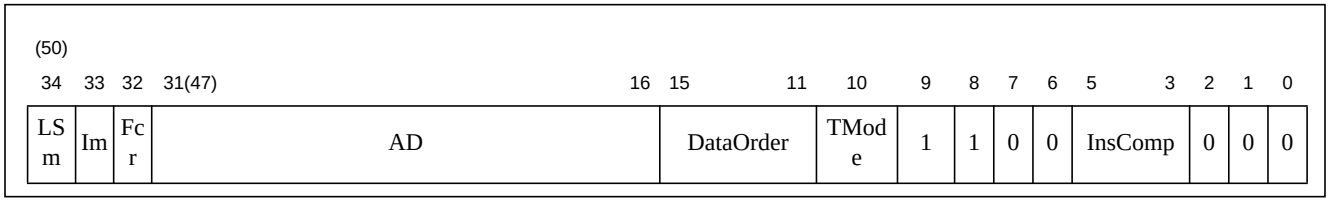
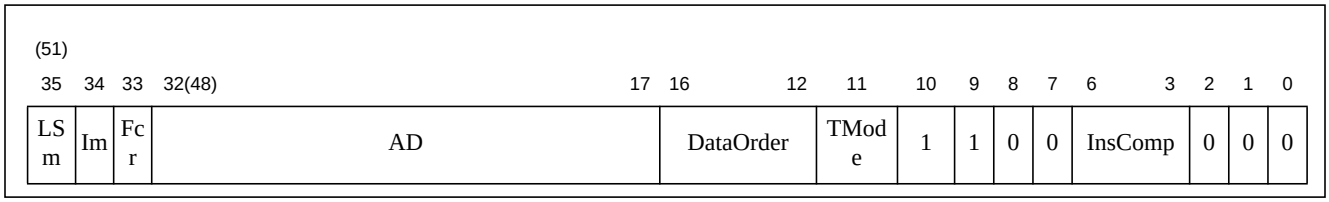


Figure A.9 Expanded TF4 with Optional Bits (Trace Format 4 Single-Pipe)



A.3 Tracing 74K in Cycle Accurate Mode

The 74K core can graduate zero, one, or two instructions in any given cycle. When tracing normally, that is, not in cycle accurate mode, the stream of graduated instructions that are traced cannot be tracked back to whether or not any pair graduated together, or how many cycles apart did they graduate. This is the typical behavior for other cores as well, with the exception that most other cores do not graduate two instructions in any given cycle. Hence, it is not necessary to do anything different for the 74K in this regard.

If cycle-accurate tracing is used, it is assumed that all graduation slots, whether empty or not are traced with NI or some other InsComp value. In this case, the assumption made is that graduation slot0 and slot1 are traced in that order.

A.4 Compressing Addresses in TF3 and TF4

The 74K implementation of the PDtrace architecture includes an additional optimization that allows the use of 16-bit addresses even when the ADW bit (bit 23 in TCBControlA) is set to 1, indicating a 32-bit address/data value width. If an address can be represented in 16 bits, the TF3 and TF4 formats are shortened to their 16-bit data variants. This is indicated by using a TMode value of 0 to indicate a delta value for the address. If the address cannot be represented in 16 bits (i.e., it requires 32 bits), TMode is set to 1 in the TF3 and TF4 formats.

Appendix B

Implementation-Specific PDtrace™ Enhancements for the MIPS32® 1004K™ Cores

B.1 Tracing a Coherent Subsystem

Memory operations in a coherent multiprocessor system can involve more than a single processor core and memory, since cache subsystems of other processors in the coherence domain must participate in coherent read and write transactions. In addition, each valid block of memory in a local processor cache can now be in one of multiple states (Modified, Exclusive, Shared, or in case of a MOESI protocol, Owned). Since each coherent transaction can involve one of two paths (get data from memory, or get data from another processor), the latency of the operation is not fixed. Finally, the coherence system as implemented by MIPS Technologies, and defined in OCP v3.0 introduces a new port, known as an 'intervention port' that deals with coherence requests from other processors which can affect the state of local cache lines. The coherence system introduces a new hardware block called the 'Coherence Manager' (CM) which is a system block responsible for queueing, ordering, processing and responding to all memory requests.

B.1.1 Trace Requirements

Trace data is gathered at two places - at each core, and at the coherence manager. This data must then be combined together to recreate an execution trace. There are two primary operations at the core that are affected by coherence - load/store instruction execution, and memory port transactions. It is useful to trace main memory port transactions since this provides a method of establishing a global order of memory instructions (by correlating a memory instruction with its main port transaction in case of a miss, and finally the request being serialized at the CM). To allow post-processing software to correlate CPU transactions with their corresponding CM transactions, we use a small identifier called the coherent synchronization ID (COSId) to synchronize transactions that is periodically updated to allow software to align transactions. The first CPU and CM transactions to use the new COSID are used to align transactions. There is sufficient detail in a trace to enable the reconstruction of program execution across multiple coherent cores. For performance debugging, timing information is collected to help determine potential bottlenecks in the system. It is possible to trace a coherent (or other) request as it is processed by the system, gathering information about the transaction type, cache hit/miss status, etc. Tracing support allows a user to gather different levels of detail from the CM. The CM is connected to a set of CPUs and IO devices and the user can selectively trace transactions that belong to only some of those cores or IO devices (applicable for external interface tracing).

B.1.1.1 Gathering Subsets of Trace Data

To reduce the information that a user must examine to find potentially interesting behavior, it is possible to trace only a subset of trace data from various sources[1]. The table below describes trace data subsets at various sources.

Table B.1 Coherent Trace Subset Options

Source	Trace Subset Options	Source	Trace Subset Options
Core	Trace All Instructions	CM - RQU	Trace Request
	Trace All Memory Instructions		Trace Request + Address
	Trace Instructions that Miss in the I-Cache		Trace Request + Stall Information
	Trace Instructions that Miss in the D-Cache		Trace Request + Address + Stall Information

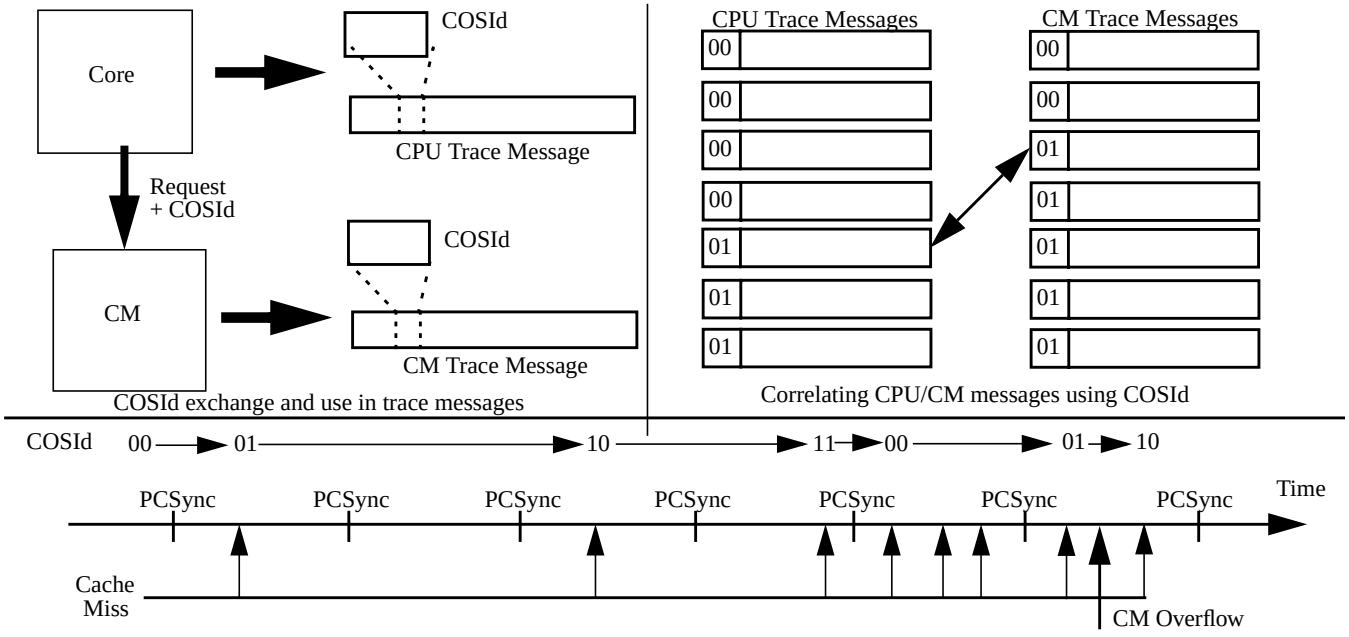
Table B.1 Coherent Trace Subset Options

Source	Trace Subset Options	Source	Trace Subset Options
Memory Instructions	Trace Address + Data	CM - IVU	Trace Request
	Trace Address Only		Trace Request + Stall Information

B.1.1.2 Synchronizing CPU and coherent interconnect trace messages

To synchronize CPU trace messages with those gathered from the coherent interconnect, a new identifier (COSId) is added in the core trace block. The COSId is included in trace messages from the core and is sent to the coherent interconnect for inclusion in trace messages. Each core maintains an independent COSId, and trace messages from the coherent interconnect always hold the COSId of the originating core. The COSId is updated periodically when a load or store miss occurs at the processor. The first CPU trace message using the new COSId corresponds to the first CM message that will use the new COSId, thus providing reconstruction software an exact point at which the two traces match. Using this match point, other trace messages from the CPU and CM can be correlated. PDtrace supports periodic synchronization messages in the form of PC syncs or TMOAS records (A PC sync is a TMOAS record with a SYNC value of 1). The new COSIds will be updated at most as frequently as the CPU sends out a PC sync/TMOAS record, with one added restriction - the COSId cannot be updated unless a miss occurs. Thus, the CPU can send out multiple PC syncs without updating the COSId, if no load/store misses occur during that time.

In case of overflow at the core, a new PCsync message must be sent. At the same time, the COSId must be updated (so long as there is a corresponding cache miss). In case of overflow at the CM, a signal must be sent back to all CPUs in the system requesting new COSIds. Figure B-1 illustrates the use of the COSId.



COSId updates: (a) PCSync followed by cache miss (first five changes), (b) CM overflow followed by cache miss

Figure B-1 COSId - Creation, Correlation and Updates

[1] This can also be used to reduce off-chip bandwidth requirements, but that is not the primary intent.

B.2 On-Chip Trace Memory

On-chip trace memory is supported in coherent cores that implement revision 6.10 and higher of the PDtrace architecture. This memory is shared by all processor cores and the coherence manager and is accessed as previously defined in the PDtrace architecture specification via the TCBTW register. To ensure consistent read/write behavior, the Core_ID field in the PDtrace Control Configuration Register is used to grant on-chip memory access to a single core. Read/Write requests from other cores (via drseg or via the TAP controller) are ignored.

B.3 Software Control of CMP Trace

Software control is enabled through the CMTraceControl register in the GCR register space (Debug Control Block, offset 0x0010). This register is very similar to the TCBControlD register, and is described below. A coherent core that implements revision 6.00 and above of the PDtrace architecture also provides software access to the TCBControlD register via drseg.

Figure B-2 *CMTraceControl* Register Format

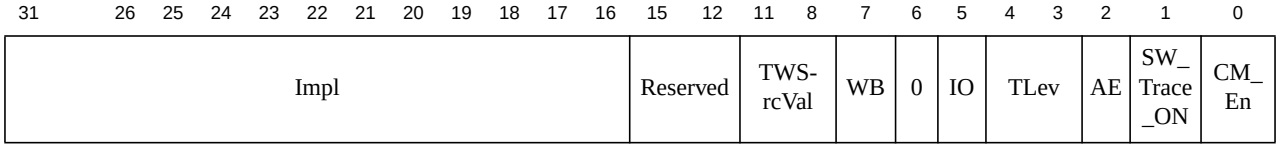


Table B.2 *CMTraceControl* Register Field Descriptions

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
Impl	31:16	Reserved for implementations. Check core documentation		Undefined	Optional
Reserved	15:13	Reserved for future use. Must be written as 0, and read as 0	0	0	Required
TWSrcVal	11:8	The source ID of the CM.	0	0	Required
WB	7	When this bit is set, Coherent Writeback requests are traced. If this bit is not set, all Coherent Writeback requests are suppressed from the CM trace stream	R/W	0	Required
Reserved	6	Reserved for future use. Must be written as 0, and read as 0	0	0	Required
IO	5	Inhibit Overflow on CM FIFO full condition. Will stall the CM until forward progress can be made	R/W	Undefined	Required
TLev	4:3	This defines the current trace level being used by CM tracing Encoding / Meaning: 00 — No Timing Information 01 — Include Stall Times, Causes 10 — Reserved 11 — Reserved	R/W	Undefined	Required
AE	2	When set to 1, address tracing is always enabled for the CM. This affects trace output from the serialization unit of the CM.	R/W	0	Required
SW_Trace_ON	1	Setting this bit to 1 enables tracing from the CM as long as the CM_EN bit is also enabled.	R/W	0	Required

Table B.2 *CMTraceControl* Register Field Descriptions (Continued)

Fields		Description	Read / Write	Reset State	Compliance
Name	Bits				
CM_EN	0	This is the master trace enable switch to the CM. When zero, tracing from the CM is always disabled. When set to one, tracing is enabled whenever the other enabling functions are also true.	R/W	0	Required

B.4 CM Trace Formats

Trace data can have two sources within the CM - the serialization response handler (SRH) or the Intervention Unit (IVU). The SRH uses two trace formats (CM_TF1, CM_TF2), and the IVU uses one format (CM_TF3). One trace format (CM_TF4) is used to indicate that overflow has occurred. The first one to four bits of a trace packet can be used to determine the packet type.

B.4.1 CM Trace Format 1

When request addresses are not being traced, the CM_TF1 trace format, shown in Figure B-3 and Figure B-4, is used. If the TLev field in TCBControlD (or CMTraceControl) is set to 1, each packet also includes the SRH_WaitTime field. The packet width varies from 14 bits (trace level 0) to 22 bits (trace level 1). Trace reconstruction software must determine the total packet length by examining the appropriate control bits in TCBControlD or the CMTraceControl register.

Figure B-3 CM Trace Format 1 (CM_TF1) - Trace Level 0

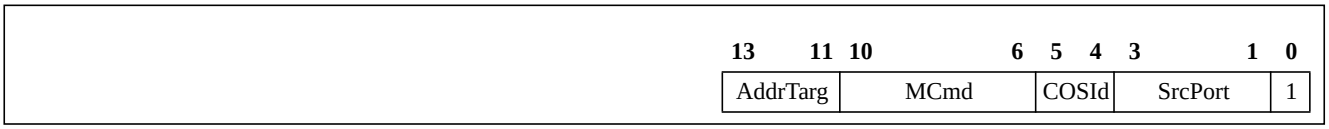
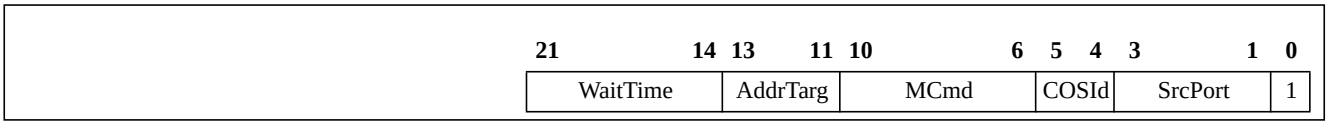


Figure B-4 CM Trace Format 1 (CM_TF1) - Trace Level 1



B.4.2 CM Trace Format 2

When request addresses are being traced, the CM_TF2 trace format, shown in Figure B-5 and Figure B-6, are used. Since each core sets the lowest three address bits to zero, only address bits [31:3] are traced. If the TLev field in TCBControlD (or CMTraceControl) is set to 1, each packet also includes the SRH_WaitTime field. The packet width varies from 45 bits (trace level 0) to 53 bits (trace level 1). Trace reconstruction software must determine the total packet length by examining the appropriate control bits in TCBControlD or the CMTraceControl register.

Figure B-5 CM Trace Format 2 (CM_TF2) - Trace Level 0

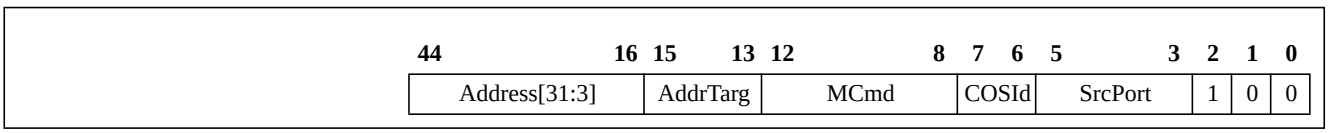
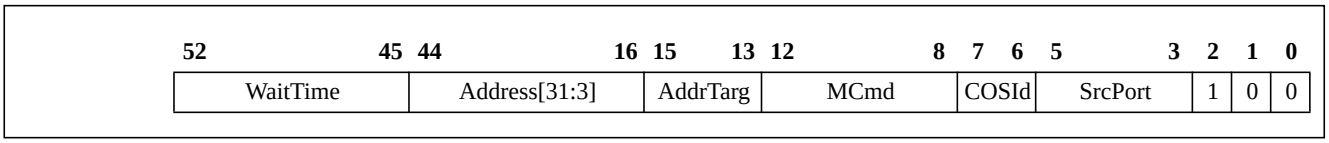


Figure B-6 CM Trace Format 2 (CM_TF2) - Trace Level 1



B.4.3 CM Trace Format 3 (CM_TF3)

Trace data from the IVU uses the CM_TF3 trace format, shown in Figure B-7 and Figure B-8. If the trace level (TLev in TCBControlD or CMTraceControl) is set to 1, each packet also includes two additional fields (WaitTime and StallCause). Each packet is 18 bits (trace level 0), or 29 bits (trace level 1). The SCF field indicates if a Store Conditional Failed, and the SCC field indicates if a Store Conditional was cancelled. Trace reconstruction software must determine the trace level being used by examining the TCBControlD register or the CMTraceControl register.

Figure B-7 CM Trace Format 3 (CM_TF3) with Trace Level 0

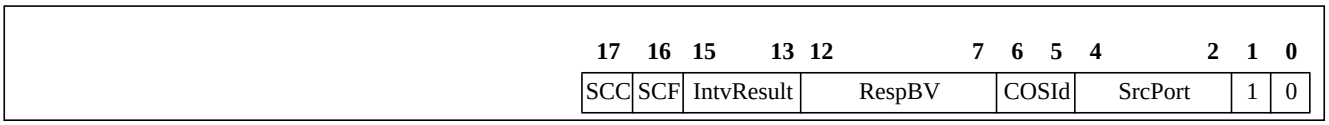
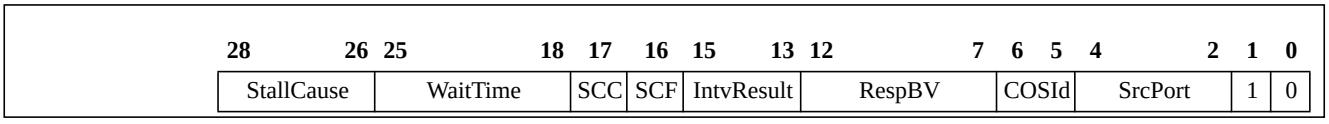


Figure B-8 CM Trace Format 3 (CM_TF3) with Trace Level 1



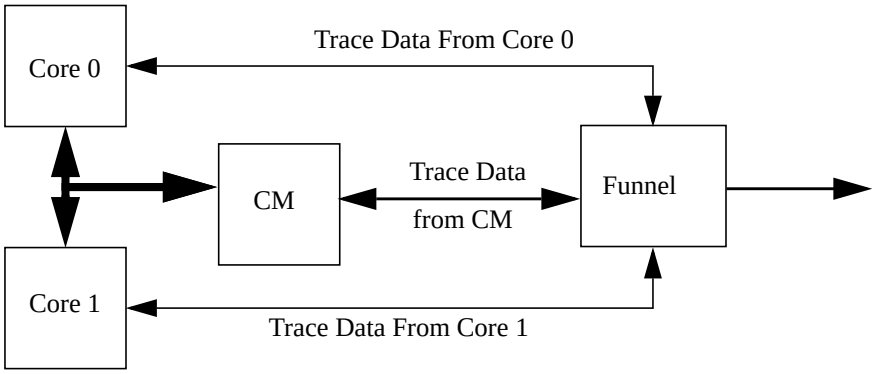
B.4.4 CM Trace Format 4 (CM_TF4)

If the CM_IO (Inhibit Overflow) bit is not set, it is possible for trace packets to be lost if internal trace buffers are filled. The CM indicates trace buffer overflow by outputting a CM_TF4 packet. Regular packets resume after the CM_TF4 packet. The coherence manager must also resynchronize with all cores by requesting a new COSId.

Figure B-9 CM TF_4 - Overflow Format

B.5 Consolidating Trace Information

The coherence manager and each core in the system can generate trace data. This data must be passed through some hardware block (the ‘Funnel’) that merges it into a single output stream and sends it to the user. A block diagram is given below:



The Funnel must allow the user control over which sources contribute to the final trace data being sent to the on-chip trace buffer, or the an external probe. For example, it should be possible to disable tracing data from the CM while gathering data from Core0 and Core1. Since each load/store miss can be traced at the core and the CM, this provides one possible method to determine execution order. Each message from the CM can act as a synchronization point for instruction execution at CPUs. Some external software that is aware of the potentially different clock domains under which the CPUs and the CM operate must be used to establish execution order.

Appendix C

Tracing Multi-Issue and High-Performance Processors

This section of the PDtrace specification is now designated to an appendix chapter since it is not clear if this method for tracing multi-issue pipelines is useful and will ever be implemented. This may very well be deprecated from the specification in a future release.

C.1 Background on High Performance Processors

This section addresses the tracing needs of multi-issue pipeline processors and describes a mechanism that allows a workable and efficient tracing of program execution on such processors. The features of high performance processors are not in general, very suitable for effectively tracing the sequential execution of a program. Such processor features include, but are not limited to:

- Superscalarity or multi-issue
- Aggressive, out-of-order dynamic scheduling with big fetch and issue windows
- Deep pipelines
- Multi-latency pipelines
- Multiple outstanding load misses

A processor that is designed to issue multiple instructions, and moreover out of order from the original program sequence, will also implement what is typically known as the re-order buffer. This re-order buffer and its control logic is responsible for putting the issued instructions back in-order (of the original program sequence). There is a stage in the pipeline when instructions are graduated from the re-order buffer, i.e., the point where it is certain that the instruction will not stop due to an exception (or any other reason), and can proceed to completion. This graduation of instructions from the re-order buffer is done in program sequence.

There are several things to note here, one, the graduated instructions have not completed their execution and will proceed to do so further in the pipeline, for example, the register write-back of the computed result of an arithmetic instruction will happen later in the pipeline. The second thing to note, is that, typically, the number of graduating instructions will not exceed the number of issue slots of the processor. But the number can vary from a minimum of zero up to the number of issue slots at the front of the pipe plus the number load miss completions from the bus and cache units, etc.

C.2 The Basic Tracing Methodology

The trace methodology in this document proposes that instructions be traced at the point of graduation. It is recommended that a number of instructions be simultaneously traced, the recommended number is the number of issue slots of the processor, let us call this the “number of instruction trace slots”. It is possible that in some cycles the number of graduating instructions is greater than the number of instruction trace slots. In these cases, the processor’s trace control logic must buffer the instruction(s) that could not be traced earlier, and trace them at the beginning of the next

cycle, still maintaining the program sequence order. Note that the size of such a buffer need not be very large, since over time, the number of issued instructions will equal the number of graduated instructions. The size of this buffer can be calculated based on the maximum number of instructions that can graduate from the re-order buffer on any given cycle, and this number is based on the processor's pipeline depths and other pipeline-related information.

All the signals marked "Out" are signals output from the processor core and represent the activity of a single instruction within the core. Most of these signals need to be duplicated as many times as the number of instruction trace slots within the core. Signals that must be duplicated are shown in Table D.1 also with signal names appended with a "_n", where n is used to designate the instruction trace slot number. For example, a two-issue core can trace two instructions and use InsComp_0 and InsComp_1 to represent the completion status values of two simultaneously graduating instructions. If only one instruction graduates on any given cycle, then InsComp_1 has a value of 000. When no instruction graduates on a given cycle, then both InsComp_0 and InsComp_1 have 000 values.

The same example code fragment from before is shown in Table C.1 and this table shows the graduation cycle of each instruction and the number of the instruction trace slot that actually traces that instruction. This example assumes a simple two-issue processor that allows up to one load/store instruction per issue and one branch instruction per cycle.

Table C.1 Example Code Fragment Showing the Graduation Cycle and Trace Bus Number

Instr No.	PC	Instruction	Graduation Cycle	Slot Number
1	0x00400188	SW $6, 0xe170($1)	n+0	0
2	0x0040018c	SW $4, 0xb134($28)	n+1	0
3	0x00400190	SW $5, 0xb130($28)	n+2	0
4	0x00400194	SW $0, 0x1c($29)	n+3	0
5	0x00400198	JAL 0x418d9c	n+4	0
6	0x0040019c	OR $30, $0, $0	n+4	1
7	0x00418d9c	NOP	n+5	0
8	0x00418da0	JR $31	n+5	1
9	0x00418da4	NOP	n+6	0
10	0x004001a0	JAL 0x411c40	n+7	0
11	0x004001a4	NOP	n+7	1
12	0x00411c40	JR $31	n+8	0
13	0x00411c44	NOP	n+8	1
14	0x00414adc	LW $4, 0xb134($28)	n+9	0
15	0x00414ae0	BEQ $14, $0, 0x414af8	n+9	1
16	0x00414ae4	ADDIU $29, $29, 0xffe0	n+10	0
17	0x00414af8	OR $7, $0, $0	n+10	1
18	0x00414afc	NOP	n+11	0
19	0x00414b00	ADDU $6, $6, $2	n+11	1
20	0x00414b04	OR $7, $2, $0	n+12	0
21	0x00414b08	SLTU $1, $2, $1	n+12	1

C.3 Coordinating the Instruction Completion Trace with the Address/Data Trace

When an instruction is traced on a particular instruction trace slot, say using InsComp_k, then all other information for that instruction is sent on the signals of the "k"th instruction trace slot. For example, the address and data, if any, associated with that instruction is sent on the same slot. Thus, once an instruction begins its trace life on a particular instruction trace slot, it must complete its life on the same slot. The exception to this occurs when the data is not immediately available. In this situation, the data can be sent on any of the slots that is temporarily free and hence chosen by the processor to send that data. See C.4 "Out-of-Order Loads and Stores in the Multi-Pipe Core".

The process of identifying the data associated with particular instructions has been simplified by making it a requirement that all the data associated with instructions traced on the same cycle be in lock-step. Specifically, all the data associated with instructions that are traced together on the different InsComp_n are such that their end points (i.e., the last data cycle) are synchronized to be traced together. This requirement makes it easier for an external block to sequence all the data operations into the program sequence. An example that illustrates this behavior is shown in Figure C.1.

Figure C.1 An Example Showing the Coordination of Instructions and Their Data

(1)

Program Sequence
ILBa
ILb
ISc
ILd

(2)

InsComp_0	InsComp_1	cycle
ILBa	ILb	n
ISc	ILd	n+1

(3)

Cycle	TType_0	TType_1	TEnd_0	TEnd_1	Comments
m+0	TPCa1	NT	0	x	
m+1	TPCa2	NT	1	x	
m+2	TLAa1	NT	0	x	
m+3	TLAa2	TLAb1	1	1	
m+4	TDa1	TDb1	0	0	
m+5	TDa2	TDb2	1	1	completion of all TType transfers for instructions traced in cycle n
m+6	TSAc1	NT	0	x	
m+7	TSAc2	TLAd1	1	1	
m+8	TDc1	TDd1	0	0	
m+9	TDc2	TDd2	1	1	completion of all TType transfers for instructions traced in cycle n+1

(4)
Data in Program Sequence
TPCa1, TPCa2,
TLAa1, TLAa2,
TLAb1,
TDa1, TDa2
TDb1, TDb2
TSAc1, TSAc2
TLAd1
TDc1, TDc2
TDd1, TDd2

Figure C.1 shows four blocks of information. The first one **(1)** shows the instruction complete (InsComp) values in the program sequence. The second block **(2)** shows these values as they would be transmitted on the two instruction trace slots, i.e., InsComp_0 and InsComp_1. The third block **(3)** shows the TType and TEnd values for the two trace slots. Note that the data trace information for the instructions that were simultaneously traced on InsComp_0 and InsComp_1 are traced such that their TEnd is coordinated. For the InsComp values traced in cycle n (in block **(2)**), the data transmission ends in cycle m+5 (in block **(3)**). And for the InsComp values traced in cycle n+1 (in block **(2)**), the data transmission ends in cycle m+9 (in block **(3)**).

The external block reading the signals on the interface can then take the data values, and knowing the program sequence order (in block **(1)**), can put the data trace in order, as shown in block **(4)**.

C.4 Out-of-Order Loads and Stores in the Multi-Pipe Core

When a multi-pipe core needs to send out-of-order data, it uses the DataOrder signal just like the single-pipe core. When an out-of-order data is returned, it can be traced on any free slot, not necessarily the one that traced the corresponding instruction. This is because, instruction tracing is sequentialized by the InsComp_n order and therefore the data can be associated with the correct instruction once the DataOrder value is known. Note that since the slots are implicitly ordered, for data transmissions that end on the same cycle, the data on TType_k is before the data on TType_k+1.

C.5 Tagging Instructions that Issue Together

With the method of tracing graduating instructions in sequence, it is not possible to know which instructions issue together without additional information. This information might be invaluable to tune a code optimizer for high performance processors. In order to trace this information, the processor tags all the instructions that issue together, using the signal IssueTag_n. This tag value is also traced out with each InsComp_n value. A tag value of 6 bits is being initially proposed, assuming an issue window of about 64 instructions. Note that this tag information can be traced out of the TCB only if the user requires it, hence it will not incur bandwidth on the external pins unless there is a real need for this information. Thus, it is recommended that the TCB allow the external tracing of this information under user discretion.

C.6 Miscellaneous

Tracing from each one of the multiple pipelines is controlled by the same set of bits, either in CP0 or in the TCB, as well be described in other chapters.

When tracing is first started (or re-started after a break), InsComp_0 is the first traced instruction in the static program image and this will output the TMOAS record and the full PC.

When there is a need for synchronization, the core can choose any InsComp_n to send the TMOAS record and the full PC value, as long as these two are both done on the same instruction in the trace slot. Note that if load/store addresses are also being traced, then a full load/store address value is part of the synchronization tracing. This may not always be possible on the instruction chosen by the core. But these should be sent on the next sequential load/store instruction. This is a situation that the external software has to take into account when recognizing synchronization transmissions in the multi-pipeline core or processor.

Appendix D

The PDtrace™ Interface Signals (The Interface is now Deprecated as Architecture and this Chapter is here Solely for Historical Reasons)

All signals are assumed to be asserted high unless otherwise noted. The signal direction “Out” refers to a signal that is output from the processor core or coherence manager, and “In” signals are those that are input to the processor core or coherence manager. The “PDO_” prefix to the signal names is used to uniquely identify the signals as belonging to the PDtrace™ Output interface. And the “PDI_” prefix is used to identify the PDtrace Input signals. Signals that have been repeated in the “Signal Name” column with a “_n” prefix are PDO_ signals that are to be duplicated for multi-issue processors.

D.1 PDtrace™ Core Interface Signal List

Table D.1 PDtrace™ Core Interface Signals

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>Pclk</td><td></td><td>Processor clock, used by the core and the trace control block.</td></tr>
<tr><td>PDO_IamTracing</td><td>Out</td><td>The core uses this signal to validate all the other Out signals. The external trace control block cannot always predict if the trace data from the core is valid or not valid since tracing depends on core execution status such as the processor mode and also since tracing can be controlled by software running on the core.
This signal is used for all the _n signals, and is not duplicated.</td></tr>
<tr><td>PDO_InsComp[2:0]
PDO_InsComp_n[2:0]</td><td>Out</td><td>Instruction completion status signal. The values are interpreted as follows:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>000</td><td>No instruction completed this cycle (NI)</td></tr>
<tr><td>001</td><td>Instruction completed this cycle (I)</td></tr>
<tr><td>010</td><td>Instruction completed this cycle was a load (IL)</td></tr>
<tr><td>011</td><td>Instruction completed this cycle was a store (IS)</td></tr>
<tr><td>100</td><td>Instruction completed this cycle was a PC sync (IPC)</td></tr>
<tr><td>101</td><td>Instruction branched this cycle (IB)</td></tr>
<tr><td>110</td><td>Instruction branched this cycle was a load (ILB)</td></tr>
<tr><td>111</td><td>Instruction branched this cycle was a store (ISB)</td></tr>
</table>
A "No Instruction" (NI) can happen due to a pipeline stall or when the instruction was killed (due to an exception).
The three encoding (101, 110, 111) for branched instruction indicates a discontinuity in the PC value for the associated instruction. Note that it is only when the new PC can not be predicted from the static program flow that it is traced.
The IPC value is used for the periodic output of the full PC value for synchronization. The tracing hardware should ensure that this is not done on an unpredictable branch, load, or store instruction.</td></tr>
<tr><td>PDO_MIPS16
PDO_MIPS16_n</td><td>Out</td><td>When asserted, this signal indicates that the current instruction specified in PDO_InsComp is a MIPS16e instruction. When de-asserted, the processor is not executing a MIPS16e instruction.
This signal (along with the PDO_MIPS16Ins signal) is used by the TCB to compute the current PC value. Hence this is irrelevant externally and not traced to memory. Note that since external software has access to the program image, it can always know whether an instruction is a MIPS16e instruction or not.
This is an optional signal for PDtrace specification revisions less than 03.00. This signal is only relevant if the processor also implements the MIPS16e ASE, and is not required otherwise. If a processor provides this signal, it is optional whether a TCB accepts this signal and uses it.</td></tr>
</table>

Table D.1 PDtrace™ Core Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_MIPS16Ins[1:0]
PDO_MIPS16Ins_n[1:0]</td><td>Out</td><td>This signal accompanies the PDO_MIPS16 signal and is used to indicate the type of MIPS16e instruction. Like PDO_MIPS16 this is optional, but must be implemented if PDO_MIPS16 is implemented.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00</td><td>Is executing a MIPS16e instruction that is not a MACRO instruction and is not extended.</td></tr>
<tr><td>01</td><td>Is executing a MIPS16e instruction that is not a MACRO instruction and is extended.</td></tr>
<tr><td>10</td><td>Is executing a MIPS16e MACRO instruction.</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>PDO_AD[15:0] or
PDO_AD[31:0]
PDO_AD_n[15:0] or
PDO_AD_n[31:0]</td><td>Out</td><td>The address or data value is transmitted on this bus. The actual values must be correlated using the PDO_TType signal described below. It is recommended that a 64-bit processor core implement at least 32 bits for improved tracing capability.
A multi-cycle transaction sends the least-significant bits first, followed by the more-significant bits.
When the transmitted data width is less than the width of the bus, the data is transmitted on the least-significant bits of the bus. There is no necessity to indicate the validity since the post-analyzing software knows the width of the data. (For example, a LB implies one byte of data). The upper bits of the bus must be sign extended to allow the TCB to truncate the upper bits and hence avoid tracing unneeded bits to memory.</td></tr>
<tr><td>PDO_TType[2:0]
PDO_TType_n[2:0]</td><td>Out</td><td>Specifies the transmission type for the transaction on the PDO_AD lines. The valid types are:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>000</td><td>No transmission this cycle (NT)</td></tr>
<tr><td>001</td><td>Transmitting the PC (TPC)</td></tr>
<tr><td>010</td><td>Transmitting the load address (TLA)</td></tr>
<tr><td>011</td><td>Transmitting the store address (TSA)</td></tr>
<tr><td>100</td><td>Transmitting the load/store data value (TD)</td></tr>
<tr><td>101</td><td>Transmitting the processor mode, the 8-bit ASID, and the SYNC bit. This is triggered by either a change in the processor mode, by a software write to the EntryHi register, or a trace synchronization operation. (TMOAS). If the processor does not implement the standard TLB-based MMU, it is UNPREDICTABLE whether a write to the EntryHi register triggers a TMOAS operation. (See Figure 4.3).</td></tr>
<tr><td>110</td><td>Transmitting user-defined trace record - type 1 (TU1)</td></tr>
<tr><td>111</td><td>Transmitting user-defined trace record - type 2 (TU2)</td></tr>
</table></td></tr>
</table>

Table D.1 PDtrace™ Core Interface Signals (Continued)

Signal Name	Direction	Description
PDO_TEnd PDO_TEnd_n	Out	Indicates the last cycle of the current transaction on the PDO_AD bus. This signal can be asserted in the same cycle that a transaction is started, implying that the particular transaction only took one cycle to complete. In a multi-issue core, the PDO_TEnd signals are synchronized for all the PDO_AD_n transmissions associated with instructions that graduate together. See 4.3.3 "Coordinating the Instruction Completion Trace with the Address/ Data Trace" on page 45 for details. In PDtrace revision 3.00 and higher, the processor is allowed to assert this signal early if the tracing logic determines that the upper bits of the address or data being sent on the PDO_AD bus are redundant. For example, redundant upper sign bits may be omitted and software could easily reconstruct these bits. Note that the TCB must therefore be capable of accepting an early PDO_TEnd signal for any transmission type. This early assertion of PDO_TEnd is allowed, for all the values of PDO_TMode.
PDO_TMode PDO_TMode_n	Out	Indicates the transmission mode for the bits transmitted on PDO_AD. The mode depends on the transmission type.

PDO_TType	PDO_TMode
000 (NT) 101 (TMOAS)	Reserved
001 (TPC)	0 -> delta from last PC value 1 -> compression algorithm A (full address)
010 (TLA) 011 (TSA)	0 -> delta from last data address of that type 1 -> compression algorithm B (full address)
100 (TD) 110 (TU1) 111 (TU2)	0 -> Reserved 1 -> compression algorithm C (full data)

Table D.1 PDtrace™ Core Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_DataOrder[3:0]
PDO_DataOrder_n[3:0]</td><td>Out</td><td>This signal is used to indicate the degree of out-of-order-ness of load and store data. Using this order value allows load and store data to be traced out as it becomes available, thus avoiding the need to internally buffer data. Note that only sixteen outstanding data values are allowed because of the limitation imposed by the signal width of 4 bits. This signal takes on the following values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0000</td><td>data from oldest load/store instruction (is in-order)</td></tr>
<tr><td>0001</td><td>data from second-oldest load/store instruction</td></tr>
<tr><td>0010</td><td>data from third-oldest load/store instruction</td></tr>
<tr><td>0011</td><td>data from fourth-oldest load/store instruction</td></tr>
<tr><td>0100</td><td>data from fifth-oldest load/store instruction</td></tr>
<tr><td>0101</td><td>data from sixth-oldest load/store instruction</td></tr>
<tr><td>0110</td><td>data from seventh-oldest load/store instruction</td></tr>
<tr><td>0111</td><td>data from eighth-oldest load/store instruction</td></tr>
<tr><td>1000</td><td>data from ninth-oldest load/store instruction</td></tr>
<tr><td>1001</td><td>data from tenth-oldest load/store instruction</td></tr>
<tr><td>1010</td><td>data from eleventh-oldest load/store instruction</td></tr>
<tr><td>1011</td><td>data from twelfth-oldest load/store instruction</td></tr>
<tr><td>1100</td><td>data from thirteenth-oldest load/store instruction</td></tr>
<tr><td>1101</td><td>data from fourteenth-oldest load/store instruction</td></tr>
<tr><td>1110</td><td>data from fifteenth-oldest load/store instruction</td></tr>
<tr><td>1111</td><td>data from sixteenth-oldest load/store instruction</td></tr>
</table></td></tr>
<tr><td>PDO_TrigI[N:0]</td><td>Out</td><td>This vector indicates which of the N+1 implemented EJTAG hardware instruction breakpoints caused a trigger. The instruction causing the trigger is indicated on the corresponding PDO_InsComp bus, if tracing has been turned on. Note that EJTAG restricts the maximum number of implementable hardware instruction breakpoints to 15.</td></tr>
<tr><td>PDO_TrigD[N:0]</td><td>Out</td><td>This vector indicates which of the N+1 implemented EJTAG hardware data breakpoints caused a trigger. The instruction causing the trigger is not necessarily the one on the PDO_InsComp bus since data triggers may be imprecise. Note that EJTAG restricts the maximum number of implementable hardware data breakpoints to 15.</td></tr>
<tr><td>PDO_TrigOn</td><td>Out</td><td>This bit is asserted if at least one trigger in PDO_TrigI[N:0] or PDO_TrigD[N:0] turns trace on. (See 4.4 “Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints” on page 48).</td></tr>
<tr><td>PDO_TrigOff</td><td>Out</td><td>This is asserted if no trigger turns trace on (i.e., PDO_TrigOn is not asserted), and at least one of the indicated triggers in PDO_TrigI[N:0] or PDO_TrigD[N:0] turns trace off. (See 4.4 “Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints” on page 48).</td></tr>
</table>

Table D.1 PDtrace™ Core Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_Overflow</td><td>Out</td><td>This signals an internal FIFO overflow error in the core and implies the following:
• the current transmission is to be abandoned in the current cycle
• the FIFO is emptied so that previously collected trace information in the FIFO is lost
• a new transmission begins in the next cycle with a TMOAS and a full PC address</td></tr>
<tr><td>PDO_ValidModes[1:0]</td><td>Out</td><td>This signal specifies the subset of tracing that is supported by the processor (see 2.2 “Subsetting” on page 19).
<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>00</td><td>PC tracing only</td></tr><tr><td>01</td><td>PC and load and store address tracing only</td></tr><tr><td>10</td><td>PC, load and store address, and load and store data</td></tr><tr><td>11</td><td>Reserved</td></tr></table></td></tr>
<tr><td>PDO_IssueTag_n[5:0]</td><td>Out</td><td>This signal is used in multi-issue processors and it is signaled with PDO_InsComp_n. In multi-issue processors, instructions that issue together are assigned a matching tag value, specified by this signal value.
A six bit internal counter increments each cycle, and the instructions that issue in that cycle are assigned the counter value. When the maximum counter value is reached, it simply restarts at zero.
This feature facilitates the performance debugging of code schedulers for high-end processors. These tag values are available every cycle, but it is anticipated that the TCB will trace this to memory only when specially requested by the user.</td></tr>
<tr><td>PDO_IMiss
PDO_IMiss_n</td><td>Out</td><td>When asserted, this signals whether the load or store instruction specified by PDO_InsComp in this cycle missed in the instruction cache during the fetch operation. This signal is ignored if PDO_InsComp indicated that no instruction completes this cycle (i.e., when it is 000).</td></tr>
<tr><td>PDO_LSMiss
PDO_LSMiss_n</td><td>Out</td><td>When asserted, this signals whether the load or store data specified by PDO_TType of TD in this cycle missed in the data cache during the data load or store operation. The data cache miss is indicated with the transmitted data rather than the instruction that caused the miss because in the pipeline a data cache miss cannot often be detected at the time that the instruction is transmitted with the appropriate PDO_InsComp value. It is the reconstruction software which needs to associate the data with the corresponding PC and data address.</td></tr>
<tr><td>PDO_FuncCR
PDO_FuncCR_n</td><td>Out</td><td>When asserted, this signal indicates that this instruction can potentially be either a function call instruction or a function return instruction. See Chapter D, “The PDtrace™ Interface Signals (The Interface is now Deprecated as Architecture and this Chapter is here Solely for Historical Reasons)” on page 121 for details.
Note that it is possible for a single instruction to assert both PDO_IMiss as well as this signal.</td></tr>
<tr><td>PDO_TC[7:0]
PDO_TC_n[7:0]</td><td>Out</td><td>For a processor that implements multithreading (MIPS MT ASE), and for a valid PDO_InsComp value (when not NI), this signal indicates the thread context number of the traced instruction. A given implementation only needs to use as many encoded bits for this signal as the total TCs implemented. For example, the 34Kc core with maximum 9 possible TCs will only require 4 bits. The PC delta value that is transmitted by the core is now maintained on a per-TC basis.</td></tr>
</table>

Table D.1 PDtrace™ Core Interface Signals (Continued)

Signal Name	Direction	Description
PDO_CPUid[7:0] PDO_CPUid_n[7:0]	Out	Optional output for a processor indicating the processor number (Ebase.CPUNum) of the traced instruction. It would be used in a multi-core design, where the Ebase.CPUNum field for the core is used to tag the trace from different cores in the multi-core environment. If the core were not implemented in a multi-core design, then the TCB would simply ignore the value on this signal.
PDO_COSId[1:0]	Out	Coherent Synchronization ID. This is a 2-bit value used to synchronize core trace messages with trace messages received from the coherent interconnect. **Required for all processors using PDtrace specification 5.00 and higher.**
PDI_TCBPresent	In	When asserted this indicates that the TCB hardware is present and connected to the core's tracing logic. Hence the core can consider the other PDI_ signals to be valid.
PDI_TraceOn	In	This is the signal asserted by the external trace block into the core that states whether tracing is globally turned on or off. It is expected that this signal be continuously asserted to turn on tracing. 0 : tracing off 1 : tracing is turned on
PDI_TraceMode[4:0]	In	When tracing is turned on, this signal specifies what information is to be traced by the core. It uses 5 bits, where each bit turns on tracing for a specific tracing mode. The table shows what trace value is turned on when that bit value is a 1. **Bit # Set** / **Trace The Following** 0 – PC 1 – Load address 2 – Store address 3 – Load data 4 – Store data If the corresponding bit is 0, then the Trace Value shown in column two is not traced by the processor. This implementation is required for all processors using PDtrace specification 4.00 and higher. Obviously, the processor has to support the tracing mode that is being requested for this input signal to have any effect. For example, if the processor only supports PC tracing, then only bit 0 is read by the processor, and other other bits are ignored, and so on. Which bits are ignored and which are read can be obtained by reading the PDO_ValidModes output signal. It is optional for an implementation to allow PC tracing to be turned off. This must be clearly documented by the core implementation-specific document. When it is optional, bit 0 is tied to a value of 1 and setting bit 0 to 0 is simply ignored by the processor.
PDI_G	In	The global bit, which if asserted to 1, implies that all processes are to be traced. If 0, then trace data is sent only for a process that matches PDI_ASID[7:0]. If the processor does not implement the standard TLB-based MMU, this signal is ignored by the processor and is treated as if it were asserted.
PDI_ASID[7:0]	In	When the global bit is 0, only the process whose ASID matches this ASID value will be traced. If the processor does not implement the standard TLB-based MMU, this signal is ignored by the processor.

Table D.1 PDtrace™ Core Interface Signals (Continued)

Signal Name	Direction	Description
PDI_U	In	Enables tracing in User Mode (see 2.1 "Processor Modes" on page 19). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.
PDI_S	In	Enables tracing in Supervisor Mode (for those processors that implement Supervisor Mode), otherwise, this signal is not required (see 2.1 "Processor Modes" on page 19). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.
PDI_K	In	Enables tracing in Kernel Mode (see 2.1 "Processor Modes" on page 19). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.
PDI_E	In	Enables tracing when in Exception Mode (see 2.1 "Processor Modes" on page 19). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.
PDI_DM	In	Enables tracing in Debug Mode (see 2.1 "Processor Modes" on page 19). This feature is useful to debug the debug handler code via the EJTAG and TAP controller port.
PDI_InhibitOverflow	In	This signal is used by the external trace block to indicate to the core that the core pipeline should be back-pressured (and stalled) instead of allowing the trace FIFO to overflow and hence lose trace information.
PDI_StallSending	In	When asserted, this signal is used by the external trace block to indicate to the core that it must stop transmitting trace information in the next cycle. This request may be essential when the trace control block is in imminent danger of over-running its internal trace buffer. In the cycle when the signal is asserted, the value on all the PDO_ signals are valid and must be captured by the TCB. In the cycle after the one where the core sees an assertion of this signal the core must not transmit any valid trace information on **any** of the PDO_ output signal bits (including PDO_InsComp). In the cycle after the TCB de-asserts this signal again, PDtrace PDO_ signals are valid and must be captured by the TCB. (Note that some processors cannot arbitrarily stall their pipeline on any given cycle. In this situation, the implementation on the processor side must provide sufficient buffering to hold trace information until the pipeline can be stalled).
PDI_SyncOffEn	In	This signal is an enable signal for the PDI_SyncPeriod, PDI_TBImpl, and PDI_OffChipTB signals. When asserted, the core latches these values. This signal, and the signals which it controls must be asserted before tracing can begin.

Table D.1 PDtrace™ Core Interface Signals (Continued)

Signal Name	Direction	Description
PDI_SyncPeriod[2:0]	In	This signal is used to set the synchronization period bits in the *TraceControl2* register. The value specifies the period (in cycles) for sending synchronization information. **SyncPeriod** / **Period (in cycles) for sending sync records** 000 — 2^5 001 — 2^6 010 — 2^7 011 — 2^8 100 — 2^9 101 — 2^{10} 110 — 2^{11} 111 — 2^{12}
PDI_TBImpl	In	When this signal is a 1, the TCB has implemented both an on-chip and an off-chip trace buffer, and the PDI_OffChipTB signal indicates to which the trace is currently being written. When this signal is a 0, the PDI_OffChipTB signal indicates which buffer is implemented. This value is written into the *TraceControl2* CP0 register (as the TBI bit). It is optional for the TCB to provide this signal to the core logic for all TCB implementations compatible to PDtrace specifications less than 03.00.
PDI_OffChipTB	In	When one, this signal indicates that the trace data is being sent off-chip to an external trace memory. When zero, this indicates an on-chip trace buffer. The value of this signal to the core changes how the core interprets the trace synchronization period bits. This signal value is written into the *TraceControl2* CP0 register (as the TBU bit).
PDI_TraceAllBranch	In	When asserted, the core's tracing logic will emit PC values for all taken branches encountered in the execution stream, including all conditional and unconditional, predictable and unpredictable branches. When de-asserted, the core reverts to normal tracing mode.
PDI_TraceIMiss	In	When asserted, PDO_IMiss is set when the processor detects an instruction cache miss for the current instruction being traced. Like all other trace signals, this input signal causes active tracing only when tracing is currently turned on. If PDI_TraceMode[0], i.e., bit 0 is turned off, that is, no PC tracing has been requested, then a PDO_IMiss assertion is accompanied by a full PC value. Otherwise there is no special action taken for this instruction other than asserting the PDO_IMiss bit.
PDI_TraceLSMiss	In	When asserted, when PDO_LSMiss is set when the processor detects a data cache miss on a load or a store. Like all other trace signals, this input signal causes active tracing only when tracing is currently turned on. If PDI_TraceMode[0] is turned off, that is PC tracing is disabled, then the PC of the missed instruction will not be available to the reconstruction software. If PDI_TraceMode[1] or PDI_TraceMode[2] is turned off, that is no data address tracing is enabled, then no data address is traced for the address that missed in the cache. Even if PDI_TraceMode[3] or PDI_TraceMode[4] is turned off, the full data value for the missed instruction is traced out.

Table D.1 PDtrace™ Core Interface Signals (Continued)

Signal Name	Direction	Description
PDI_TraceFuncCR	In	When asserted, PDO_FuncCR is set when the current instruction could be a function call or return instruction. Like all other trace signals, this input signal causes active output tracing only when tracing is turned on. If PDI_TraceMode[0], i.e., bit 0 is turned off, that is, no PC tracing has been requested, then a PDO_IMiss assertion is accompanied by a full PC value. Otherwise there is no special action taken for this instruction other than asserting the PDO_IMiss bit.
PDI_TCNum[7:0]	In	Only implemented in a processor with MT. When PDI_TCNumValid is asserted, this signal gives the number of the Thread Context that is to be traced. Only the number of bits required to encode the total TC number is implemented. As long as PDI_TraceTCValid is asserted, no instruction from any other thread is traced. If the required TC does not execute any instructions, then no instructions are traced, the PDO_InsComp value will remain NI. When instructions from other TCs are executed, these are marked as NI on the PDtrace interface.
PDI_TCNumValid	In	Only implemented in a processor with MT. When asserted, the PDI_TCNum signal is taken by the processor and used as the number of the TC whose instructions are to be traced. This signal must remain asserted as long as tracing is required from a specific TC. If not asserted, then tracing reverts to other conditions being fulfilled.
PDI_CPUId[7:0]	In	Implemented in a processor with MT, where this signal gives the number of the VPE that is to be traced, if PDI_CPUIdValid is asserted. As long as PDI_VPENumValid is asserted, no instruction from any other VPE is traced. If the required VPE does not execute any instructions, then no instructions are traced, the PDO_InsComp value will remain NI. When instructions from other VPEs are executed, these are marked as NI on the PDtrace interface. This bit is ignored if TCNumValid is asserted. In a multi-core processor SOC environment, this specifies the id of the processor that is to be traced if PDI_CPUIdValid is asserted.
PDI_CPUIdValid	In	Only implemented in a processor with MT or in a multi-core SOC implementation. When asserted, the PDI_CPUId signal is taken by the processor and used as the number of the VPE (or core) whose instructions are to be traced. This bit is ignored if TCNumValid is asserted.

D.1.0.1 PDtrace Coherence Manager Interface Signals

Table D.2 PDtrace Coherence Manager Interface Signals

Signal Name	Direction	Width	Description
COSId_n	Input	2	COSId input received from processor core. One copy of signal per core
SRH_SrcPort	Output	3	Source of the request that was serialized
SRH_COSId	Output	2	Coherent Sync ID of transaction. Used to correlate CPU and CM transactions
SRH_MCmd	Output	5	Command in the request that was serialized (See Table D.3)
SRH_WaitTime	Output	8	This is active only in timing mode. Tracks how many cycles the transaction spent stalled in the SRH. Saturates at 255 cycles.
SRH_Address	Output	29	This is active when we are tracing addresses from the SRH - provides the address corresponding to the request being traced.
SRH_AddrTarg	Output	3	Target of the current request (see Table D.4) (Indicates speculative reads as well)
IVU_COSId	Output	2	Coherent Sync ID at the Intervention Unit
IVU_SrcPort	Output	3	Which core made the original request that resulted in this intervention?
IVU_RespBV	Output	6	Bit vector of intervention port responses. Bit corresponding to a core is set to ‘1’ if the intervention hit, and set to ‘0’ if the intervention missed.
IVU_IntvResult	Output	3	Global Intervention State for this cache line (see Table D.5)
IVU_SC_Cancel	Output	1	This transaction was cancelled due to a previous SC Fail
IVU_SC_Failed	Output	1	This intervention will cause a future SC to fail
IVU_PIQ_WaitTime	Output	8	Cycle count that each transaction spends at the top of the PIQ. Saturates at 255
IVU_PIQ_StallCause	Output	3	What was the last reason this transaction was stalled on top of the PIQ. (see Table D.6)

Table D.3 MCmd - OCP Commands

Value	Command	Description	Value	Command	Description
0x00	IDLE		0x0C	COH_UPGRADE	Coherent Upgrade (SC bit == 0)
0x01	LEGACY_WR_UC	Uncached legacy write, CCA=UC, UCA, WT	0x0D	COH_WB	Coherent Writeback
0x02	LEGACY_RD_UC	Uncached legacy read, CCA = UC	0x10	COH_COPYBACK	Coherent Copyback
0x03	LEGACY_WR_WB	Cached legacy write, CCA = WB	0x11	COH_COPYBACKINV	Coherent Copyback Invalidate
0x04	LEGACY_RD_WB	Cached legacy read, CCA = WB, WT	0x12	COH_INV	Coherent Invalidate
0x05	LEGACY_SYNC	Uncached legacy read with MReqInfo[3] == 1	0x13	COH_WR_INV	Coherent Write Invalidate
0x06	L2_L3_CACHEOP_WR	Uncached legacy write with MAddrSpace != 0	0x14	COH_CMPL_SYNC	Coherent Completion Sync with MReqInfo[3] == 0
0x07	L2_L3_CACHEOP_RD	Uncached legacy read with MAddrSpace != 0	0x15	COH_CMPL_SYNC_MEM	Coherent Completion Sync with MReqInfo[3] == 1
0x08	COH_RD_OWN	Coherent Read Own	0x17	COH_WR_INV_FULL	Coherent Invalidate due to a full line
0x09	COH_RD_SHR	Coherent Read Shared	0x18	COH_RD_OWN_SC	Coherent Read own with SC bit == 1
0x0A	COH_RD_DISCARD	Coherent Read Discard	0x1C	COH_UPGRADE_SC	Coherent Upgrade with SC bit == 1
0x0B	COH_RD_SHR_ALWAYS	Coherent Read Share Always			

Table D.4 Cmd_AddrTarg

Value	Target	Value	Target
0x0	Memory/L2 with no speculation. L2 allocation bit = 0	0x1	Memory/L2 with no speculation. L2 allocation bit = 1
0x2	Memory/L2 with peculation. L2 allocation bit = 0	0x3	Memory/L2 with speculation. L2 allocation bit = 1
0x4	GCR	0x5	GIC
0x6	MMIO	0x7	Reserved

Table D.5 Global Intervention State

Value	State
0x0	Invalid
0x1	Shared
0x2	Modified
0x3	Exclusive
0x4-0x7	Reserved

Table D.6 PIQ Stall Causes

Value	Cause	Value	Cause
0x0	No Stall	0x1	Awaiting Intervention Results from CPU(s)
0x2	Waiting for IMQ empty (for Sync only) or IMQ full (for other request types)	0x3	IWDB Full
0x4	TRSQ Full	0x5	IRTQ Full
0x6	Waiting for speculative request to clear RMQ	0x7	PDtrace Stall

Appendix E

Revision History

Revision	Date	Description
1.6	August 29, 2000	Changes in this revision: Add the requirement that the data address be also periodically gathered for synchronization purposes, per FS2. Modify Figure 3 to show that the load data is picked up after alignment, per lhh. typo fixes
1.7	September12, 2000	Changes in this revision: Add a separate input signal that says whether to trace in Debug mode or not (i.e., DM = 1 in the *Debug* register), per Scott who wants to be able to debug the debug handler code. Put back Figure 3 to tap load/store data pre-alignment, per Franz. Add a section (3.17) to show when tracing is enabled. Allow the ASID to be masked under software control, per Scott. Amend Figure 1 to show the EJTAG/TAP controller and its connection to the debugger. Add to Table 2, to show the use of the PDO_InsComp signal value IPC (100). Add a chapter (6) on the trace capture block and its interaction with the external debugger software. Add TOC Fix typos, grammar, sentence construction.
1.8	October 27, 2000	Changes in this revision: Change the way loads are tracked and traced out. Add the tracing out of ASID and processor mode as part of the periodic synchronization. Add details to the multi-issue tracing section. The above changes require a modification to the output format section. Add a chapter to discuss the trace capture block (TCB), that includes: a definition of the control registers within the TCB, and the mechanism to write these registers from the external probe (or debugger). Define tracing with an on-chip trace buffer versus off-chip trace buffer. Add another Out signal from the core, PDO_IamTracing, that the core uses to signal to the TCB that it is actually sending valid trace data.
1.9	November 20, 2000	Changes in this revision: Add tracing of processor ISA mode, and whether processor is in Debug mode or not. Get rid of the TCBTraceMask register, is not really needed. Allocate some bits in the TraceControl register as implementation dependent. Specify that full addresses are used for on-chip trace memory. Change the encoding of bits from the EJTAG logic to the tracing logic, send all 30 bits of breakpoint trigger. Fix the logical expression in 3.1.8.

Revision	Date	Description
2.0	December 19, 2000	Changes in this revision: Add a signal from the TCB to the core tracing logic, PDI_StallSending, that inhibits the core from sending trace data. Note that the core does not stop tracing, only stops sending trace information to the TCB. Used by the TCB when its internal buffer is in imminent danger of overflowing. (The core will stall if its internal FIFO will overflow). Make the synchronization period programmable, by using some bits in a register to hold this value. These bits can be updated by either software or by the TCB (based on the trace buffer size). Add a signal from the TCB to the core tracing logic that signals whether the TCB is using an on-chip or off-chip trace buffer. This changes the way in which the core interprets the synchronization period bits in the register. The chapter on trace control block (TCB) has been cut off into another document, since it is not directly relevant to the PDtrace architecture.
2.01	January 25, 2001	Changes in this revision: Add a signal PDI_TCBPresent to indicate that the TCB hardware is present. Clearer explanation of how the PDI_StallSending signal works. Change in how the PDI_EXL and the corresponding X bit in the *TraceControl* register works. Coding change in the PDI_TraceMode[2:0] signal.
2.02	February 12, 2001	Changes in this revision: Change in how the PDI_EXL and the corresponding X bit in the *TraceControl* register works. Tracing triggers on when either EXL or the ERL bit is a 1, this enables tracing after a cold reset.
2.03	March 22, 2001	Changes in this revision: • Add a register description table for *UserTraceData.* • Add a PDI_TraceAllBranch signal to indicate that all branches (conditional, unconditional, predictable, and unpredictable) are to be traced. • Change the PDO_InsComp definition for unconditional predictable branches (jumps), so that these trace out as IB, ILB, and ISB (rather than I, IL, and IS). • Document how tracing is handled within MACRO instructions and the SAVE/RESTORE instruction. • Document what happens when a mode change happens within the processor and this changes the tracing mode, i.e., either turns it off or on. • Fix typos.

Revision	Date	Description
2.04	June 20, 2001	Changes in this revision: • Converted document to new template • PDO_TMode's reserved bit field of 100 is now used for tracing PC values and load data (this is optional for all PDtrace specifications less than 03.00 and conforming TCB implementations. • Three PDO_ signal bits have been added, PDO_MIPS16 and PDO_MIPS16Ins that are used only by processors implementing the MIPS16 ASE, and are optional. • The sense of EQ1, EQ2, and EQ3 used to compute the delta address values have been reversed. • Add the PDI_TraceAllBranch to the Trace Control Register. • Note that the select position of the COP0 registers implemented for tracing have all been changed, so that the control registers are together and the optional register *TraceBPC* is the last one. • Note that the end of a MIPS16 Macro instruction was indicated by the transmission of a full PC value. This was more fully specified so that this full PC value is accompanied by an PDO_InsComp value that indicates a branch, e.g., IB, ILB, etc. • The PDI_EXL has been changed to PDI_E, and similarly in the *TraceControl* register, X has been changed to E. • Bits 22 and 23 in the *TraceControl* register (K and S), have switched places. • The *TraceControl2* register has been re-arranged, and instead of the bit OfC, two new bits TBU and TBI have been added. • The TMOAS record has been augmented with an extra bit for the POM field and with a new bit called the SYNC bit. • Add an Input signal PDI_TBImpl from the TCB to the core tracing logic to say whether on-chip, off-chip, or both buffers are implemented by the TCB. This signal is optional for all TCB implementations that are compatible to PDtrace specifications less than 03.00.
2.05	June 28, 2001	Changes in this revision: • Convert the stand-alone document to a book format and add LOF and LOT pages. • Add trademark symbol to PDtrace • Fix minor typos.

Revision	Date	Description
2.06	August 8, 2001	Changes in this revision: • Define the behavior if the processor implements a fixed mapping MMU, rather than the standard TLB-based MMU. • Define the polarity of the TraceControl$_{ASID_M}$ field. • Precisely define the processor modes which for which tracing may be enabled. See Section 2.1, "Processor Modes" on page 19 for these definitions. • Make the equations for turning on and off trace more precise and convert to standard notation. • Add the standard “About This Book” chapter to define syntax and conventions. • Eliminate the R/W fields in TraceControl2. • More fully describe the synchronization counter, including when it must be restarted. • Make it explicit that ASID and processor mode changes are not traced if tracing is off when the change occurs. That is, ASID and processor mode changes are not traced if tracing is currently off. • Add subsetting rules for PDtrace (see Section 2.2, "Subsetting" on page 19) • Add the PDO_ValidModes signal and the ValidModes field in the *TraceControl2* register to specify which tracing modes the processor supports.
2.07	March 21, 2002	Changes in this revision: (RT) • Change the name of the TraceControl2 register field ValidModes to ImpSubset since this field indicated the implemented subset of tracing. • Get ready for commercial release, breakup the single file into individual chapter files, fix typos, cross-references, etc.
3.00	November 26, 2002	Changes in this revision: (RT) • Change the way multi-issue tracing is done (see Section C.1, "Background on High Performance Processors" on page 116). • Change the use of PDO_LoadOrder signal to PDO_DataOrder (see Section C.4, "Out-of-Order Loads and Stores in the Multi-Pipe Core" on page 119). • Increase the width of PDO_DataOrder signal to 4 bits. • Add a new signal called PDO_DataPerIns[7:0]. • Allow PDO_TEnd to be asserted early to cut off redundant upper bits of an address or data. • Add a section to clarify how tracing is handled for store conditional instructions (see Section 3.9, "Tracing Store Conditionals" on page 35). • Make the PDO_TMode bit 0 value for PDO_TType values of TD, TU1, and TU2 to be Reserved. • Add PDO_Trig signals on the PDtrace interface that transmit trace trigger information to the TCB. See Section 3.16, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 37. • Add MIPS16 in MIPS64 option to ISAM in TMOAS. See Table 3.4 on page 28. • Rewrite the trace enable equation to fix errors in the first version. See Section 3.20, "Trace Enabling/Disabling Condition" on page 45. • Fix grammatical errors and typos.
3.01	May 14, 2003	Removed the trace slot-specific signals PDO_TrigI_n, PDO_TrigD_n, PDO_TrigOn, and PDI_TrigOff, since these are superfluous. Fix minor typos.

Revision	Date	Description
4.10	July 4, 2005	Changes in this revision: • Merged the PDtrace and most of the TCB document • Modified how PDI_TraceMode works. This necessitated moving the Mode bits from *TraceControl* to the *TraceControl2* register. • Simplified the SyncPeriod values to range from 2^5 to 21^2 for both types of trace memory, on-chip or off-chip. • Revamped how the EJTAG hardware breakpoint trigger impacts tracing. This has an impact on what used to be the *TraceBPC* control register. See spec for change details. Added the ability to trace based on a ARM-trace-DISARM feature, as well as to allow data qualified tracing. • Added the ability to trace instruction and data cache misses • Added the ability to trace instructions that are potential function calls or function returns. • Added support to trace multi-threaded processors that implement the MIPS MT ASE. • Added a PendL, pending load field to the TMOAS record • Added a TCBControlC TCB register to deal with the added features
4.20	September 14, 2005	Changes include clarification of behavior under MT and removal of the MC bit in TCBControlB register.
4.30	January 30, 2006	Update the TMOAS record to add the V, DKill, and TCid fields for a MT-specific processor and remove a bad reference in chapter 7.
4.40	July 17, 2006	Change the drseg addresses of the TraceIBPC2 (0x1F28 to 0x1FF8) and TraceDBPC2 (0x2F28 to 0x2FF8) registers.
5.00	November 15, 2007	Significant change in the PDtrace architecture, the PDtrace interface is no longer architecture and the only externally software-visible parts are the control registers in CP0, in the TCB, and the TCB trace bits using the defined TCB formats. Adds CMP support.
6.00	June 23, 2008	Add Performance counter support, Filtered data trace mode, and software access to on-chip trace memory. Expanded PEndL in TMOAS record.
6.10	November 06, 2008	Added 74K specific updates, on-chip trace memory updates for the 1004K