MIPS® iFlowtrace™ Architecture Specification

Document Number: MD00526
Revision 1.01
March 16, 2009

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

Table of Contents

Chapter 1: Introduction to the iFlowtrace™ Architecture and Methodology 7
- 1.1: Overview of the iFlowtrace™ Method 7
- 1.2: Architectural Indicator for the Presence of the iFlowtrace™ Method 7
- 1.3: A Block-Level Overview of the iFlowtrace™ Method 7

Chapter 2: iFlowtrace™ Interface Signals 9
- 2.1: Trace Inputs 9
  - 2.1.1: Implementation Notes 9
- 2.2: Trace Outputs 9
- 2.3: iFlowtrace™ To ITCB Signal Interface 10

Chapter 3: iFlowtrace™ Control Block (ITCB) 13
- 3.1: ITCB Storage Representation 13
- 3.2: ITCB Register Interface for Software Configurability 14
- 3.3: ITCB iFlowtrace Off-Chip Interface 14

Chapter 4: Breakpoint-Based Triggering of Tracing 17

Appendix A: References 19

Appendix B: Revision History 21

List of Figures

Figure 1.1: An Overview of iFlowtrace, ITCB, and Other Core Blocks.. 8

List of Tables

Table 2.1: Data Bus Encoding 11
Table 3.1: Registers in the ITCB.. 14
Table 4.1: drseg Registers that Enable/Disable Trace from Breakpoint-Based Triggers.. 17

Chapter 1

Introduction to the iFlowtrace™ Architecture and Methodology

The goal of this trace specification is to describe the information needed to reconstruct a simple instruction trace from an execution stream. Such a simple mechanism enables a small, efficient tracing methodology for any core. The PDtrace™ scheme that is currently supported by MIPS® core families can sometimes be too heavy weight for certain applications such as low-end microcontrollers or when it is desired to have a simpler tracing scheme in production chips.

1.1 Overview of the iFlowtrace™ Method

The light-weight instruction-only tracing scheme proposed here is sufficient to reconstruct the execution flow in a core under conditions as stated in this document. This tracing scheme has been kept very simple to minimize the impact on die size.

The iFlowtrace tracing scheme is not a strict subset of the PDtrace tracing methodology, and its trace format outputs differ from those of PDtrace. New trace formats, using simplified instruction state descriptors, were designed for the iFlowtrace trace to simplify the trace mechanism and to obtain better compression.

1.2 Architectural Indicator for the Presence of the iFlowtrace™ Method

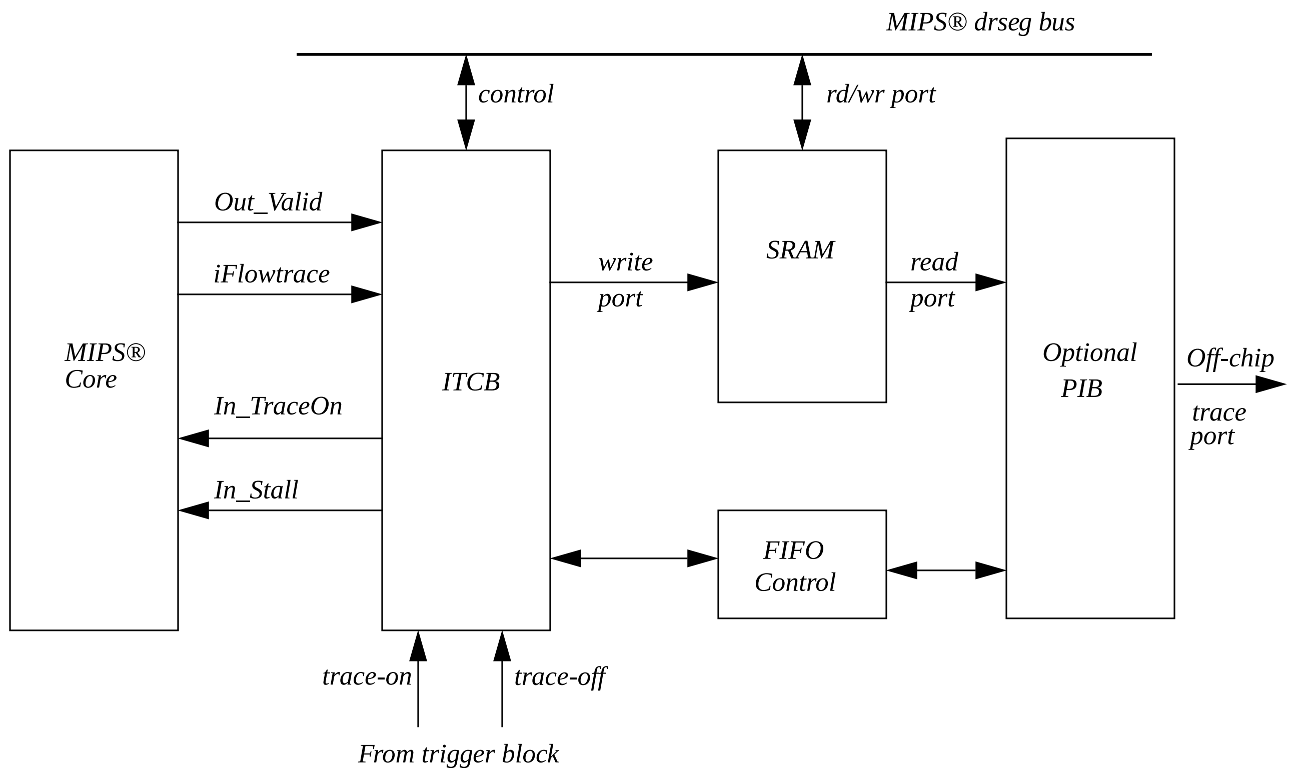
The presence of the iFlowtrace scheme in a core is indicated by a configuration bit, bit 8 (*ITL*) of *Config3* CP0 register [4].

1.3 A Block-Level Overview of the iFlowtrace™ Method

A trace methodology can often be defined mainly by its inputs and outputs. Hence this scheme is described by the inputs to the core tracing logic and by the trace output format from the core. We assume here that the execution flow of the program is traced at the end of the execution path in the core similar to the PDtrace scheme. Figure 1.1 shows a block diagram of the flow of the trace from the tracing logic near the core to the iFlowtrace Control Block (ITCB).

The ITCB is responsible for accepting trace signals from the CPU core, formatting them, and storing them in an on-chip memory organized as a circular buffer. The figure shows the Probe Interface Block (PIB) as capable of emptying the SRAM/circular buffer and outputing the memory contents through a narrow off-chip trace port.

Figure 1.1 An Overview of iFlowtrace, ITCB, and Other Core Blocks



Chapter 2

iFlowtrace™ Interface Signals

This chapter describes iFlowtrace input and output signals and the iFlowtrace interface to the iFlowTrace Control Block (ITCB).

2.1 Trace Inputs

1. *In_TraceOn*: When on, legal trace words are coming from the core and at the point when it is turned on, that is for the first traced instruction, a full PC value is output. ?? When off, it cannot be assumed that legal trace words are available at the core interface.

2. *In_Stall*: This indicates that the processor should be stalled in order to avoid a buffer overflow that can lose trace information. When off, a buffer overflow will simply throw away trace data and start over again. When on, the processor is signalled from the tracing logic to stall until the buffer is sufficiently drained and then restart the pipeline.

2.1.1 Implementation Notes

Depending on the core pipeline and the ease with which the pipe can be stalled, the trace control block needs to know the latency between the assertion of the *In_Stall* signal and the maximum number of cycles before the pipe can be halted. This information is then used to determine how many empty trace FIFO entries are needed to store potential trace information after the stall is asserted and the *Out_Valid* signal will be de-asserted.

For a given core implementation, the maximum pipeline stall latency is known and this will be used by the ITCB (iFlowtrace Control Block, shown in Figure 1.1) for its worst case calculations on FIFO space requirements. Note that if tracing is turned on, stalls are enabled to ensure no lost trace data, and the code being run has a particularly large number of unpredictable jumps, then for a given FIFO size, it is possible to make the core stall quite often, and this will affect the use of the M4K and the performance that one would see on the core when running under these conditions. If it is anticipated that tracing will be enabled and stalls will be enabled for full traces as the default configuration, then it is essential to take typical code that will run under these situations and characterize the number of bits of trace that will be needed for say 100 instructions and correlate that back to both the size of the FIFO in the ITCB as well the expected rate at which this FIFO will be cleared, in order to prevent an excessive amount of stalling.

2.2 Trace Outputs

1. Stall cycles in the pipe are ignored by the tracing logic and are not traced. This is indicated by the signal *Out_Valid* that is turned off when no valid instruction is being traced. When *Out_Valid* is asserted, instructions are traced out as described in the rest of this section. The traced instruction PC is a virtual address.

2. In the output format, every sequentially executed instruction is traced as bit 0.

3. Every instruction that is not sequential to the previous one is traced as either a 10 or an 11. This implies that the target instruction of a branch or jump is traced this way, not the actual branch or jump instruction (this is similar to PDtrace):

4. A 10 instruction implies a taken branch for a conditional branch instruction whose condition is unpredictable statically, but whose branch target can be computed statically and hence the new PC does not need to be traced out. Note that if this branch was not taken, it would have been indicated by a 0 bit, that is sequential flow.

5. A 11 instruction implies a taken branch for an indirect jump-like instruction whose branch target could not be computed statically and hence the taken branch address is now given in the trace. This includes, for example, instructions like jr, jalr, and interrupts:

   - 11 00 - followed by 8 bits of 1-bit shifted offset from the last PC. The bit assignments of this format on the bus between the core tracing logic and the ITCB is:

     [3:0] = 4'b0011
     [11:4] = PCdelta[8:1]

   - 11 01 - followed by 16 bits of 1-bit shifted offset from the last PC. The bit assignments of this format on the bus between the core tracing logic and the ITCB is:

     [3:0] = 4'b1011
     [19:4] = PCdelta[16:1]

   - 11 10 - followed by 31 of the most significant bits of the PC value, followed by a bit (NCC) that indicates no code compression. Note that for a MIPS32 or MIPS64 instruction, NCC=1, and for MIPS16e instruction NCC=0. This trace record will appear at all transition points between MIPS32/MIPS64 and MIPS16e instruction execution.
     This form is also a special case of the 11 format and it is used when the instruction is not a branch or jump, but nevertheless the full PC value needs to be reconstructed. This is used for synchronization purposes, similar to the Sync in PDtrace. A preset sync period of 256 instructions is counted down and when an internal counter runs through all the values, this format is used. The bit assignments of this format on the bus between the core tracing logic and the ITCB is:

     [3:0] = 4'b0111
     [34:4] = PC[31:1]
     [35] = NCC

   - 11 11 - Used to indicate trace resumption after a discontinuity occurred. The next format is a 1110 that sends a full PC value. A discontinuity might happen due to various reasons, for example, an internal buffer overflow, and at trace-on/trace-off trigger action.

2.3 iFlowtrace™ To ITCB Signal Interface

For the given input and output bits defined in the previous section, the iFlowtrace interface consists of 36 data signals plus a valid signal. The 36 data signals encode information about what the CPU is doing in each clock cycle. Valid indicates that the CPU is executing an instruction in this cycle and therefore the 36 data signals carry valid execution information. The iFlowtrace data bus is encoded as shown in Table 2.1. Note that all the non-defined upper bits of the bus are zeroes.

Table 2.1 Data Bus Encoding

Valid	Data (LSBs)	Description
0	X	No instructions executed in this cycle
1	0	Sequential instruction executed
1	01	Branch executed, destination predictable from code
1	<8>0011	Discontinuous instruction executed, PC offset is 8 bit signed offset
1	<16>1011	Discontinuous instruction executed, PC offset is 16 bit signed offset
1	<NCC><31>0111	Discontinuous instruction or synchronization record, No Code Compression (NCC) bit included as well as 31 MSBs of the PC value
1	1111	Internal overflow

The ITCB controls trace using the *In_TraceOn* signal. When 0, all data appearing on the iFlowtrace outputs is considered invalid. To turn on trace, the ITCB switches *In_TraceOn* from 0 to 1. A 1011 record represents the first instruction executed thereafter, with a full PC indicating the current execution point.

Chapter 3

iFlowtrace™ Control Block (ITCB)

The ITCB is responsible for accepting the data stream from the iFlowtrace output and sending it to the memory or appropriate core output interface.

3.1 ITCB Storage Representation

Records from iFlowtrace are inserted into a memory stream exactly as they appear in the iFlowtrace data output. Records are concatenated into a continuous stream starting at the LSB. When a trace word is filled, it is written to memory along with some tag bits. Each record consists of a 64-bit word, which comprises 58 message bits and 6 tag bits or header bits that clarify information about the message in that word.

The ITCB includes a 58-bit shift register to accumulate trace messages. Once 58 or more bits are accumulated, the 58 bits and 6 tag bits are sent to the memory write interface. Messages may span a trace word boundary. In this case, the 6 tag bits indicate the bit number of the first full trace message in the 58-bit data field.

The tag bits are not strictly binary because they serve a secondary purpose of indicating to off-chip trace hardware when a valid trace word transmission begins. At least one of the 4 LSBs of the tag is always a 1. The longest trace message is 36 bits, so the starting position indicated by the tag bits is always between 0 and 35.

When trace stops (ON set to zero), any partially filled trace words are written to memory. Any unused space above the final message is filled with 1's. The decoder distinguishes 1111 patterns used for fill in this position from an 1111 overflow message by recognizing that it is the last trace word.

These trace formats are written to a trace memory that is either on-chip or off-chip. No particular size of SRAM is specified; the size is user selectable based on the application needs and area trade-offs. Each trace word can typically store about 20 to 30 instructions, so a 1 KWord trace memory could store the history of 20K to 30K executed instructions.

The on-chip SRAM or trace memory is drseg mapped to a maximum size of 16KB, and starting at drseg offset of 0x4000 and ending at 0x7FFF.

3.2 ITCB Register Interface for Software Configurability

The ITCB includes a drseg memory interface to allow software to set up tracing and read the current status. There are two drseg register locations in the ITCB as shown in Table 3.1.

Table 3.1 Registers in the ITCB

drseg Location Offset	Register	Defined Bits	Code	Description
0x3FC0	Control/Status	0	ON	Software control of trace collection. 0 disables all collection and flushes out any partially filled trace words.
		1	EN	Trace enable. This bit may be set by software or by Trace-on/Trace-off action bits from the Complex Trigger block. Software writes EN with the desired initial state of tracing when the ITCB is first turned on and EN is controlled by hardware thereafter. EN turning on and off does not flush partly filled trace words.
		2	IO	Inhibit overflow. If set, the CPU is stalled whenever the trace memory is full. Ignored unless OfC is also set.
		3	OfC	Offchip. 1 enables the PIB (if present) to unload the trace memory. 0 disables the PIB and would be used when on-chip storage is desired or if a PIB is not present. This bit is settable only if the design supports both on-chip and off-chip modes. Otherwise is a read-only bit indicating which mode is supported.
		4	OfClk	Controls the Off-chip clock ratio. When the bit is set, this implies 1:2, that is the trace clock is running at 1/2 the core clock, and when the bit is clear, implies 1:4 ratio, that is the trace clock is at 1/4 the core clock
0x3FC8	Trace write address pointer	N:0	WAddr	This register is used only if the SRAM is supported in on-chip mode. The current write pointer for trace memory. Each completed trace word is written to memory, then WAddr increments. When trace concludes, WAddr contains the first address in trace memory not yet written.
		31	Wrap	Trace wrapped. This bit indicates that the entire trace depth has been written at least once. After trace concludes, this bit along with WAddr is used by software to determine the oldest and youngest words in the buffer.

3.3 ITCB iFlowtrace Off-Chip Interface

The off-chip interface consists of a 4-bit data port (*TR_DATA*) and a trace clock (*TR_CLK*). *TR_CLK* can be a DDR clock; that is, both edges are significant. *TR_DATA* and *TR_CLK* follow the same timing and have the same output structure as the PDtrace TCB described in MIPS specifications. The trace clock is the same as the system clock or related to the system clock as either divided or multiplied. The *OfClk* bit in the *Control/Status* register is of the form X:Y, where X is the trace clock and Y is the core clock. The Trace clock is always 1/2 of the trace port data rate, hence the “full speed” ITCB outputs data at the CPU core clock rate but the trace clock is half that, hence the 1:2 OfClk value is the full speed, and the 1:4 OfClk ratio is half-speed.

When a 64-bit trace word is ready to transmit, the PIB reads it from the FIFO and begins sending it out on *TR_DATA*. It is sent in 4-bit increments starting at the LSBs. In a valid trace word, the 4 LSBs are never all zero, so a probe listening on the *TR_DATA* port can easily determine when the transmission begins and then count 15 additional cycles to collect the whole 64-bit word. Between valid transmissions, *TR_DATA* Is held at zero and *TR_CLK* continues to run.

TR_CLK runs continuously whenever a probe is connected. An optional signal *TR_PROBE_N* may be pulled high when a probe is not connected and could be used to disable the off-chip trace port. If not present, this signal must be tied low at the Probe Interface Block (PIB) input.

The following encoding is used for the 6 tag bits to tell the PIB receiver that a valid transmission is starting:

```
//   if (srcount == 0), EncodedSrCount = 111000 = 56
//   else if (srcount == 16) EncodedSrCount = 111001 = 57
//   else if (srcount == 32) EncodedSrCount = 111010 = 58
//   else EncodedSrCount = srcount
```

Chapter 4

Breakpoint-Based Triggering of Tracing

Each hardware breakpoint in the EJTAG block [2] has a control bit associated with it that enables a trigger signal to be generated on a break match condition. This trigger signal can be used to turn trace on or off, thus allowing a user to control the trace on/off functionality using breakpoints. For the simple hardware breakpoints, there are defined registers TraceIBPC, TraceDBPC, etc in PDtrace that are used to control tracing functionality. Similar registers need to be defined to control the start and stop of iFlowtrace. The details on the actual register names and drseg addresses are shown in Table 4.1.

Table 4.1 drseg Registers that Enable/Disable Trace from Breakpoint-Based Triggers

Register Name	drseg Address	Reset Value	Description
ITrigiFlowTrcEn	0x3FD0	0	Register that controls whether or not hardware instruction breakpoints can trigger iFlowtrace tracing functionality
DTrigiFlowTrcEn	0x3FD8	0	Register that controls whether or not hardware data and tuple breakpoints can trigger iFlowtrace tracing functionality

The bits in each register are defined as follows:

- Bit 28 (IE/DE) : Used to specify whether the trigger signal from EJTAG simple or complex instruction (data or tuple) break should trigger iFlowTrace tracing functions or not. A value of 0 disables trigger signals from EJTAG instruction breaks, and 1 enables triggers for the same.

- Bits 14..0 (IBrk/DBrk): Used to explicitly specify which instruction (data or tuple) breaks enable or disable iFlowTrace. A value of 0 implies that trace is turned off (unconditional trace stop) and a value of 1 specifies that the trigger enables trace (unconditional trace start).

Chapter A

References

This appendix lists other documents available from MIPS Technologies that are referenced elsewhere in this document. These documents may be included in the `$MIPS_HOME/$MIPS_CORE/doc` area of a typical CoreName soft or hard core release, or in some cases may be available on the MIPS web site, http://www.mips.com.

1. MIPS® EJTAG Specification
   MIPS document: MD00047

2. MIPS® EJTAG Trace Control Block Specification
   MIPS document: MD00148

3. MIPS® PDtrace™ Interface Specification
   MIPS document: MD00136

4. MIPS32® Architecture For Programmers, Volume I: Introduction to the MIPS32® Architecture
   MIPS document: MD0082

Chapter B

Revision History

Revision	Date	Description
1.00	June 26, 2006	Initial release.
1.01	July 15, 2008	Add Reference appendix.