Non-intrusive On-chip Debug Hardware Accelerates Development for MIPS RISC Processors

Special for EE Times

Author: Morten Zilmer, Design Engineer, MIPS Technologies, Inc.

Overview

The proliferation of high performance 32- and 64-bit RISC processors have been a boon to designers of next generation systems in digital consumer electronics, information appliances, set-top boxes, and office automation applications. With new design methodologies and advances in manufacturing processes, powerful RISC processors can now be easily integrated into a variety of custom ASICs. But with this new capability, comes the traditional burden of providing effective debug and development tools.

On the one hand, the powerful software and development tools that come with the MIPS RISC 32-bit and 64-bit processor architecture, such as highly optimized compilers and integrated development environments, and modular real-time operating systems software make software development easier. But, unlike traditional workstation or PC-oriented application software that executes in the same platform in which the software is developed, embedded system software will execute on a separate target system with extremely limited memory and I/O resources. Debugging and hardware/software integration can become a significant hurdle to fast prototype development and ultimately to hitting the market opportunity window.

MIPS Technologies, Inc. and the MIPS architecture licensees have developed the industry¹s broadest range of new 32-bit and 64-bit processor implementations spanning the spectrum of price, performance and on-chip peripheral options. Through the MIPS semiconductor licensees, system OEMs can choose an appropriate processor configuration and select from a wide variety of peripheral options. They can then quickly incorporate that design into custom ASICs that provide the optimum power consumption and compact size for their application. But, how do you quickly and inexpensively debug a one-of-a-kind system-on-chip? Typically, hardware development tools lag the appearance of new generation architectures by a significant amount of time and generally only follow when there is a large design community. When you are designing a one-of-a-kind system, you need a more immediate option.

MIPS Technologies, Inc. and the MIPS RISC consortium have developed a specialized hardware debug technology called MIPS EJTAG. EJTAG is inexpensive, easy to implement and provides non-intrusive debug capabilities for any MIPS RISC processor-based system-on-a-chip that incorporates it. It has been published (EJTAG Specification, version 2.x) for use as an industry-wide standard and is being implemented in almost all the MIPS processor products by MIPS licensees, and all the processor core products from MIPS Technologies, Inc.

EJTAG Re-Uses IEEE JTAG Boundary Scan Pins for Basic Debug Interface

To keep on-chip costs low, and to minimize any target system overhead, the MIPS EJTAG utilizes the widely used IEEE JTAG pins for its debug functions. Using special debug circuitry on-chip, the EJTAG provides run control, breakpoints on both data and instructions, real-time Program Counter trace. In addition, individual licensees can add additional features when desired. Such features could include complex breakpoints and execution profiling features.

On-chip debug provides some new tools for debugging embedded CPUs that avoid the limitations of traditional hardware debug tools. For example, it is not possible to use a logic analyzer to track operations that take place between the CPU and the on-chip data and instruction caches. But, on-chip EJTAG can track these operations. Also, using In-Circuit Emulators with high-speed systems is often problematical because they affect the bus loading characteristics of the system and can induce "tool-related" bugs into the system. In addition, they are rarely available for on-of-a-kind system-on-a-chip. Finally, some solutions require special bond-out chips that provide extra control signals and busses. But, this is additional design overhead in both chip and board design and it adds more precious time to the product cycle. EJTAG obtains the same results without the additional time and cost.

EJTAG utilizes the 5-pin IEEE 1149.1 JTAG specification for off-chip communications. These signals, called the Test Access Port or TAP include TRST (reset), TCK (clock), TMS (Test Mode Select), TDI (Data In) and TDO (Data Out). Internally as shown in Figure 1, EJTAG provides for a set of instruction, data and control registers and circuitry to access the address and data busses.

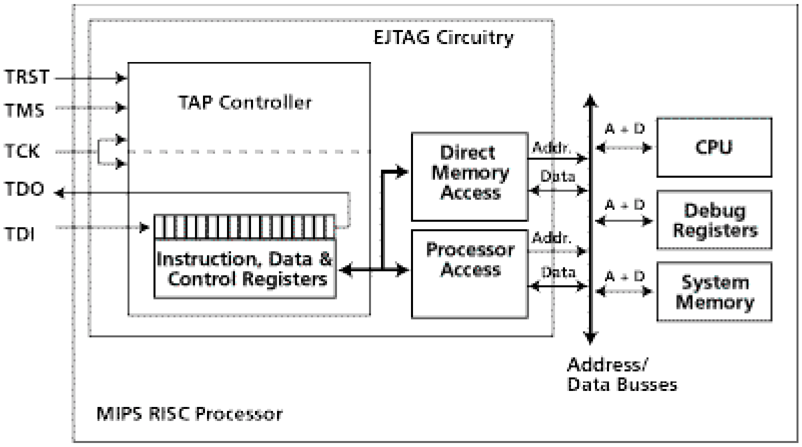

Figure 1 - EJTAG uses the 5-pin JTAG interface to access internal debug registers, and circuitry that monitors and controls the address and data busses of the processor (Note: Direct Memory Access is proposed but not implemented as of today)

The EJTAG interface through the TAP is a serial communication port with a clocking frequency of 40 MHz or more. The Control pins switch between data registers and instruction registers whose contents are send in and out serially. Special instructions defined in the EJTAG specification defines what registers are selected and how they are used. The following table shows the EJTAG instruction and the description of each.

EJTAG Instruction	Description of Register Usage
IDCODE	Device Identification Register with manufacturer, part number, and version ID for this specific chip.
ImpCode	Implementation Register indicating implemented EJTAG features in this device.
EJTAG_ADDRESS_IR	EJTAG Address Register used to access the on-chip address bus.
EJTAG_DATA_IR	EJTAG Data Register used to access the on-chip data bus.
EJTAG_CONTROL_IR	EJTAG Control Register used for setup and status information.
EJTAG_ALL_IR	Access to EJTAG Address, Data and Control registers in one chain.
PC Trace	Setup for start of PC Trace.
BYPASS	One-bit register with no operation.

EJTAG registers are generally 32-bit wide (depending on implementation) and are used to set up the debug resources and capture debug status information during the debug operation.

Operation of the EJTAG circuitry is controlled through a EJTAG probe that interfaces between the host development system and the target device as shown in Figure 2.

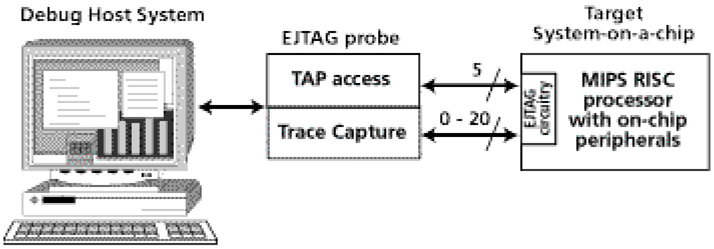

Figure 2 - A simple EJTAG debug probe provides easy access to internal processor resources for any host development system.

Debug Operations

The Processor Access is used to setup and monitor the processor internal busses and to execute the code from the EJTAG interface. In order to provide debug code without integrating it into in the application code, the EJTAG Processor Access circuitry shares a specific memory location (0xFF20.0000 to 0xFF2F.FFFF in the 32-bit address space) that can replace system memory in debug mode. When the processor accesses this memory space, the EJTAG circuitry can feed it debug instructions not resident in the application code.
When an access is detected, the EJTAG circuitry makes the transaction address available in the EJTAG Address Register, and the appropriate data available in the EJTAG Data Register available if the operation is a write, and the appropriate data is inserted into the EJTAG Data Register if the operation is a read. It takes about 200 TCK periods to access 32-bit address and data registers in this fashion, so with a 40 MHz TCK frequency, the access time is in the range of 5 us.
An added benefit of the EJTAG implementation is that it can be used to provide reprogramming of any in-system programmable FLASH memory that may be in the system. This makes it possible to upgrade in final production and even in the field.

CPU Debug Features

The EJTAG debug features do require high integration with the on-board CPU. The CPU must provide a special debug mode, registers, and instructions to support the debug process. One of the most important features is a high priority debug exception that has a higher priority over all other exceptions. The debug exception can occur when a Software Debug Breakpoint instruction is encountered, a Single Step instruction takes occurs, a JtagBrk debug event is registered by the EJTAG circuit, or a Hardware Breakpoint occurs.

The Software Debug Breakpoint (SDBBP) instruction is defined for MIPS Instruction Set Architecture and for the code compression Application Specific Extension MIPS16. For simple breakpoints, the debug system can replace application code instructions with software breakpoint instructions and generate a debug exception.

When the debug exception occurs, the CPU switches into debug mode where there are no restrictions on access to coprocessors and memory, and where the usual exceptions like address error and interrupt are masked.

The debug exception handler is provided by the debug system and can be executed through the EJTAG port using the processor access circuitry or it can be placed in application code space if the system developer so chooses.

Debug registers are DEBUG, DEPC, and DESAVE registers which are added to the MIPS Coprocessor 0 (CP0). The DEBUG register shows the cause of the debug exception and any other standard exceptions that may have occurred at the same time. Also, it is used for the setting up of single step operations. The DEPC or Debug Exception Program Counter Register holds the address of where the debug exception occurred. This is used to resume program execution after the debug operation finishes. Finally, DESAVE or Debug Exception Save Register is a scratch pad for one of the general-purpose 32-bit registers of the processor. This frees up the general purpose registers from duty in handling the debug exception handler. This lets the debug exception handler execute without affecting the contents of any of the general-purpose registers.

EJTAG Hardware Breakpoints

There are several types of simple hardware breakpoints defined in the EJTAG specification. These stop the normal operation of the CPU and force the system into debug mode. The break occurs when certain activities take place on the CPU address, data and control busses. The debug exception occurs before the bus transaction occurs preserving any contents in the register file or memory. Hardware breaks, unlike software breaks, can be made based on the address on the memory bus, so breakpoints can be set for access to any area of memory (not just legitimate software accesses). Hardware breaks also enable breaks on load/store operations.

Generally, breakpoints are setup in debug mode and become operational when normal operational mode is re-entered. There are three types of simple hardware breakpoints: Instruction breakpoints, Data breakpoints, and Processor Bus breakpoints.

Instruction breaks occur on instruction fetch operations and the break is set on virtual address on the bus between the CPU and the instruction cache. Instruction breaks can also be set on the Address Space ID or ASID value used by the Memory Management Unit. Finally, a mask can be applied to the virtual address to set breakpoints on a range of instructions.

Data breakpoints occur on load/store transactions and the breakpoint is set on virtual address and ASID values in similar fashion as the Instruction breakpoint. Data breakpoints can be set on a load, a store or both. Data breakpoints can also be set based on the value of the load/store operation. Finally, masks can be applied to both the virtual address and the load/store value.

Processor bus breakpoints occur on transactions on the system bus. Processor bus breakpoints are set on physical address values and/or on the fetch/load/store data value. Masks can also be applied to both the physical address and the fetch/load/store data value.

There are up to 45 break channels (15 of each type) defined in the specification. The actual number available will depend upon the amount of debug hardware circuitry added to the chip. With the maximum number of breakpoint hardware, it is possible to have up to 45 concurrent breakpoints set all independent of each other with separate breakpoint values.

Real-time Program Counter Trace

One of the major difficulties in debugging a high performance CPU with on-chip caches, is obtaining real-time access to the program counter. EJTAG real-time PC trace makes it possible to monitor the operation of embedded CPUs with no real-time impact on code execution.

Real-time Program Counter Trace does require some additional pins on the chip however. The number of pins required depends upon the CPU speed. Depending upon the number of available pins and the clock frequency for outputting the PC trace values, the processor may continue at full speed, or it may be forced to stall while the debug data transfer is completed. An implementation for the JADE 32-bit processor cores from MIPS Technologies, Inc, a very compact, but powerful 32-bit MIPS RISC processor implementation can utilize from 4 to 11 pins for PC trace. At 4 pins, the trace captures around 40 percent of the available data before delays occur, while with 11 pins, 100 percent capture is achieved with no stalls. The trace provides program counter information from a specified anchor point. When the instructions flow sequentially there is no need for continuous updating of the PC value. Only when the program counter changes via a jump, branch or exception is there a need to provide updated program counter values. At the occurrence of a jump, branch or exception, information indicating what kind of change in the program flow occurred, and indication of the resulting new value of the program counter is provided. In the case of an exception, the type of exception is also provided.

Only a small number of pins are required because only changes in the program counter need be transferred. Depending upon the speed of the processor, it may be possible to capture sufficient information with the minimum four-pin interface. At higher speeds, it may require more to maintain real-time trace capabilities.

Conclusion

EJTAG is a powerful new non-intrusive development and debug technology that can provide many of the high performance, real-time debug features needed to rapidly develop high performance embedded systems, at a very low system cost.

Using the pre-existing JTAG Boundary Scan interface, the 5-pin EJTAG interface provides hardware breakpoints, unlimited software breakpoints, and real-time Program Counter trace with a minimum of hardware overhead. It can be easily incorporated into MIPS RISC processors used in system-on-a-chip devices and while adding very little system cost; it can greatly accelerate system design.