

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

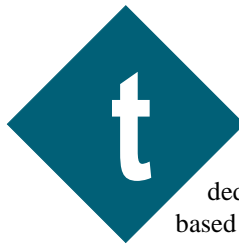
FEATURE ARTICLE

Daniel Mann

Helping the Environment

Embedded Software Development

Although embedded microprocessor applications are on the rise, the tools used for developing embedded software are having a hard time keeping up. ICES can't equal the complexity of today's processors, so Daniel shows us why manufacturers like AMD are moving to on-chip support.



The software used to support embedded microprocessor-based products has grown so complex that it almost rivals the complexity of desktop software. However, the productivity of tools used to develop embedded software has lagged behind their desktop counterparts.

Within the embedded industry, in-circuit emulators (ICES) are widely used for software development. ICES are effective tools, but rising processor complexity has reduced or delayed their availability and is also pushing to raise their cost. Additionally, ICES have not provided the embedded-software industry with a universally accepted software-development environment to rival the PC desktop environment.

To address this issue, processor manufacturers are moving to include on-chip support for software development. This article reviews traditional methods and introduces AMD's software development port.

TRADITIONAL EMBEDDED TOOLS

Traditionally, the most powerful piece of debug equipment available to an embedded project has been the ICE. They are most frequently (but not exclusively) used during the early stages of bringing up the hardware. In most cases, they are too expensive to be widely available to all project members. Unfortunately, rising processor complexity, higher clock speeds, use of on-chip cache, and packaging problems have reduced the availability of ICES.

Although AMDebug technology supports an on-chip trace capability (more on that later), an alternative solution to supporting some kind of on-chip trace buffering scheme is to support a trace port. A port offers a standard interface for external trace capture hardware and should simplify the design and cost of ICE equipment. In my opinion, program trace data is best captured on-chip and not unreliably or intrusively extracted via expensive trace-pins, which have inadequate bandwidth. An on-chip trace cache also offers a trace capability at a much lower tool-up cost.

HOST-TO-TARGET CONNECTION

Generally, and certainly at the early stages of software development, embedded (target) processors are controlled from remote (host) platforms. These remote machines are said to host the debug and development tools.

The method used to connect the target and host machine is a primary concern for any embedded project. A number of techniques have been used to achieve this connection. ROM monitors typically use a UART and support software located in the target system's memory. Using an ICE solves the problem because the target is connected via the ICE umbilical.

An ideal strategy would use an on-chip dedicated link for all communica-

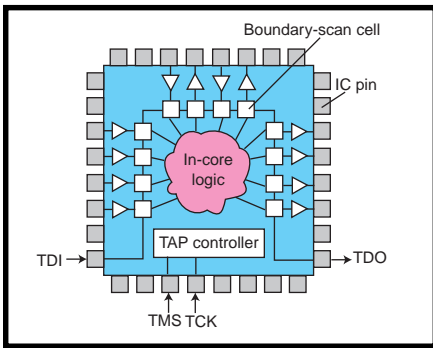


Figure 1—The boundary-scan test (BST) methodology is based on a test access port (TAP) that can be used to serially clock test data into a chip via the test data input (TDI) pin.

tion with the target. This arrangement greatly simplifies the connection problem. This development port should support high communication speeds—much faster than a UART. The port should support target bring-up, as well as kernel-mode debugging, application-mode debugging, and operating system communication needs.

WHAT IS JTAG?

Manufacturers of high-tech electronic products formed the Joint Test Action Group (JTAG) to develop a boundary-scan testing standard. The intent was to enable testing of board interconnections without the need for physical probing. For more information, see Jeff Bachiochi's article "JTAG—Working with CoolPLD" in *Circuit Cellar* 104.

Clocking the test clock (TCK) input pin moves the data along the scan cells making up the test register (see Figure 1). Test cells can be connected to input/output pins or other key internal test points. The test register is also commonly known as the scan path.

The test data output (TDO) pin serially clocks out the test register stream as data is clocked in. The fourth pin, known as the test mode select (TMS) is an input pin used to control the state transitions of the port control logic. By clocking in 0 or 1 on the TMS pin, the action taken by the controller can be directed. For example, the state of the chip's input pins can be sampled or different scan paths can be selected.

Conventional micropro-

cessor testing, based on JTAG technology, involves serially clocking test data into selected scan paths within the processor. Test results must also be clocked out serially and scan paths can be long.

Because multiple-scan steps must be carried out for even simple test actions, the process can be time consuming. Even with high JTAG clock rates, scan-intensive technology is too slow for use with software development. However, techniques based on short scan paths and additional hardware-assistance can enable a JTAG port to be used as a communications methodology for a processor port oriented to software development. Such techniques are included under the common title "enhanced JTAG."

BRINGING DOWN TOOL COSTS

The traditional full-function ICE solution is the most expensive in terms of hardware tool cost. It also satisfies the smallest number of software engineers, because its price restricts its availability. A low-cost BDM-style approach is preferred by the majority of software engineers. For this reason, the leading processor manufacturers have started to offer some low-cost debug tool approaches, based on on-chip debug support.

Most projects have one or two hardware/firmware engineers and six or eight software engineers. These companies can generally afford to buy one full-function ICE for the project, but not one for each software engineer. Most of the software engineers need to be supported by a low-cost option.

AMD is working on a debug-port technology designed to support all stages of embedded software development. This new debug port standard is based on an enhanced JTAG interface.

A JTAG-based port was chosen because many processors already have the necessary four or five pins allocated.

Connection pins are an expensive resource and there is considerable benefit to a solution that can make use of existing pins.

Although communication with the processor complies with JTAG signaling standards, conventional boundary-scan methods are not used. The JTAG port supports short scan paths into on-chip hardware within the AMDebug port, which efficiently supports the basic tasks required during software debugging. In effect, JTAG is used as a communication methodology.

The AMDebug port adds short scan paths but does not prevent traditional boundary-scan testing to co-exist with the port's new use. The JTAG specification supports the addressing (selection) of alternative scan paths. A CISC processor, such as an 'x86, can use microcode to implement the development port functionality. The basic functions supported by the AMDebug development port enable visibility and control of processor and system resources, such as registers and memory contents.

HARDWARE CONNECTION

Software development systems based on AMDebug technology should provide for a 12-pin connector on each board design. It shouldn't be difficult to assign the necessary tracking from the processor's pins to the standard 12-pin connector. To enable in-field debugging, it would be worthwhile including the small connector on production systems.

A PC parallel port-to-AMDebug connector can be built or acquired for as little as \$100. Connection to a target via this simple arrangement offers considerable advantages. There is no need to remove the processor to connect an ICE-like umbilical, connection is ensured no matter what the processor packaging technology, and debug communication is independent of processor or memory system clocking speeds.

For improved communication and greater functionality, the four basic JTAG signals (TCK clock, TMS state selection, TDI input, and TDO serial

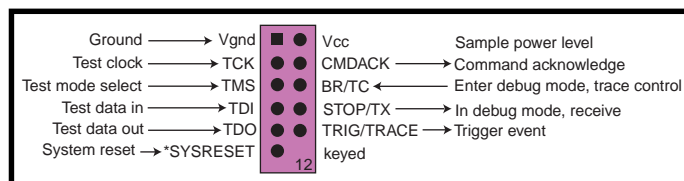


Figure 2—This setup requires four additional signal pins to be added to the processor.

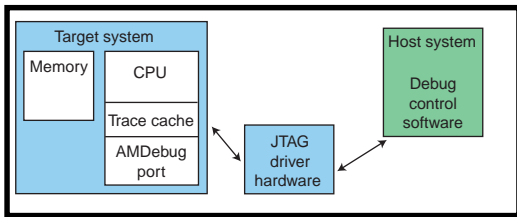


Figure 3—Software engineers need to know a program's address flow without turning off the cache or in any way intrusively monitoring the processor's operation.

SUPPORTED FUNCTIONS

The software development port provides commands to the processor that are processed by microcode. The port consists of command and data registers that are used to exchange information between the processor and the host system used to control the target. Alternative scan paths are provided for initializing a Soft-Address support register. Certain commands autoincrement the Soft-Address value when moving data between the Debug-Data register and memory. This feature is helpful when loading a program into memory.

Commands are provided for accessing all processor and system resources. However, while the processor is dealing with AMDebug utility commands, it cannot perform its normal task of fetching instructions from memory. Command processing only occurs while the processor is stopped and operating in debug mode. However, the AMDebug port can be used for application-mode communication while the processor is running.

Instructions are provided for the processor to read and write data into the receive and transmit registers—the technique supports bidirectional data transfer and enables an operating system or application to communicate with the host platform without stopping processor execution. Communication is achieved via the AMDebug channel and on-chip or target-system application resources are not stolen.

As you can see in Figure 3, commands and data are exchanged via serial JTAG-based clocking.

On-chip instruction cache memory makes it difficult to trace a program's execution path by merely observing the external pins. The use of clock scaling, superscalar technology, and high internal clock speeds, makes it difficult to provide information to the outside world.

The AMDebug strategy provides for a small on-chip trace cache that stores only critical information, such as the outcome of the branch decision. The compression techniques employed enable a good deal of the execution path to be re-

output) are extended by additional "side band" signals. A 12-pin connector format like the one in Figure 2 is used.

Let's look at what these pins offer the debug tool user. The command-acknowledge (CMDACK) output pin helps speed the communication channel. Normally after a command is serially clocked into the AMDebug port, the port must be polled to determine when the command has been processed and the port is ready for another command.

The CMDACK pin simplifies or eliminates the poll-loop. When a new command is entered, the CMDACK pin is cleared. It is set again when command processing completes.

Testing the CMDACK pin is much quicker than scanning the status of the AMDebug port. Alternatively, the CMDACK pin low-to-high transition can be used to generate a host interrupt. The benefits of the CMDACK pin are most apparent when moving large blocks of data, such as with program download.

The operation of the Break/Trace-Control input pin is determined by setting an operating mode. When set for Break, the pin can be used to interrupt normal processor operation. This is similar to a program encountering a breakpoint at a software location, only this breakpoint is activated by external hardware. It can be useful for off-chip hardware monitoring system operation to stop execution when a predefined condition has been identified.

When set for Trace-Control, the pin controls enabling and disabling of on-chip trace capture. This is explained later, but basically the strategy supports an on-chip Trace Cache that can be used to record a program's execution path. The BR/TC pin enables external logic to control the capturing of trace data.

The STOP/TX output pin

is also multiplexed. Under normal processor operation the pin is low, it transitions high when processor execution stops (i.e., when the processor enters debug mode). When properly configured, a processor will enter this mode when an exception is taken, such as a memory access violation, a software breakpoint, or a privilege violation. It's possible to determine if the processor has stopped by polling the AMDebug port with a serial command. Polling in this case isn't too bad of a performance loss. However, it's more convenient to test the STOP/TX pin or be interrupted by its activation.

The TX portion of the STOP/TX pin is used while the processor is running. The pin becomes active (high) when there is communication data waiting to be sent from the target processor to the host development platform. This process is known as Application Mode communication and is explained in more detail later. The AMDebug port can be polled to determine if the processor has stopped or if data is waiting to be serially clocked out. When the data is removed and processor continues normal execution, the STOP/TX pin transitions low.

The TRIG output pin is used to pulse-out an event signal (identified by the on-chip breakpoint control registers) or they can be arranged to generate pulse signals rather than stop program execution. These arrangements are sometimes known as data watchpoints. The method enables external debug logic or trace-capture hardware to initiate its debug support operation in synchronization with program execution.

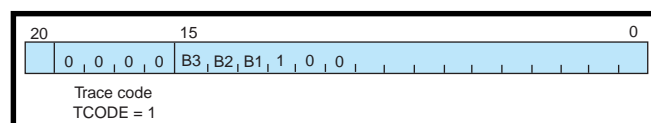


Figure 4—Here's an example trace entry showing three conditional branch instructions.

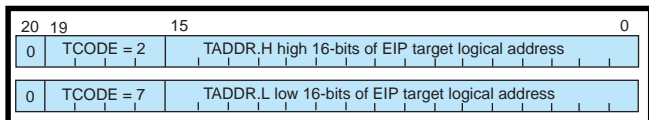


Figure 5—In a procedure return where the target address is supplied by the stack, tracing the target address of the new instruction sequence is a must.

tained in the on-chip cache. The cache can also log other information such as OS activity or performance critical parameters (new instructions enable the processor to write or read trace cache data).

The incorporation of on-chip trace memory allows tracing to be performed without intruding into (slowing down) program execution. There is no need to turn off on-chip caches to provide for visibility into processor operation and, of course, there are none of the packaging or connection problems associated with traditional ICE. Further, trace capture can be turned on and off under control of the target processor, which is useful when you are only tracing a single thread or interrupt handlers.

The trace cache is also used when a multitasking operating system is employed. It is possible to unobtrusively trace execution of a single task. This process extends the debug capability normally offered to debuggers incorporated with operating systems. The method overcomes the typically poor integration between operating systems and external trace capture hardware such as a logic analyzer or ICE.

TRACE CACHE DATA COMPRESSION

Only executed instructions need to be traced and trace entries should only contain the minimum of information. For this reason, branch instructions (which are unconditional) and where the target is supplied in immediate format (e.g., a CALL and JMP), are not traced. Conditional branch instructions only generate a single bit entry in a trace record (B1, B2, B3, etc.). The bit records the outcome of the branch decision (see Figure 4).

However, when a conditional branch or any branch instruction disturbs sequential instruction flow and the target address is data depen-

dent, the target address of the new instruction sequence must be traced, as shown in Figure 5.

One of the programs used to study trace cache operation is an LAPD (link access protocol-D) benchmark. It is large and representative of communication applications. The LAPD is the signaling protocol for the D-channel of ISDN. It is primarily used to perform such tasks as call set-up and tear-down.

Analysis of the LAPD program indicates that a trace cache of 256 entries typically holds the trace history of over 1,000 assembly-level instructions, and frequently a trace length of much more than 1,000 (as high as 6,000). This trace length, although limited, is of considerable value to the software engineer.

APPLICATION MODE COMMUNICATION

Suppose an application program is required to send data to the host platform controlling the target processor, this data may consist of a character text string from a *printf()* call or register information from a Task's Control Block (TCB). AMDebug supports data communication via a send and receive register that can be scanned from the AMDebug port (see Figure 6).

When the Host platform needs to send data to an application or the target operating system, it clocks the data into the Receive register. Once the register is full, the RX bit is set and an interrupt generated. The RX bit remains set until the processor removes the data from the Receive register. You are required to scan the RX bit and wait for its reset before scanning further data into the Receive register.

The technique enables an operating system or application to communicate

with the host platform without stopping processor execution. Communication is achieved via the AMDebug channel and on-chip application resources are not stolen. Where it is necessary to disable system interrupts, the RX and TX bits can be examined by the 'x86 processor, and hence the communication link driven in a polled mode.

FUTURE TOOL METHODOLOGIES

One thing is clear, ICE developers are having an increasingly difficult time developing equipment to keep pace with growing processor complexity. Processor manufacturers have responded with an array of on-chip support features that assist with tool development. It seems clear that the future of ICE lies with the widespread use of advanced on-chip debug ports.

These ports have three main characteristics. They need to support complete access to processor and system resources, at least while the processor is halted. This should be accomplished via an inexpensive, high-speed, link to the processor.

Secondly, they need to support some form of debug and communication capability while the processor is running. This supports application mode debugging and operating system communication needs.

Thirdly, they need to support program trace. Ideally (or possibly essentially in the embedded industry), this should be achieved at real-time speeds without any processor pipeline stalling as a result of trace generation. Most port designers have approached the problem by easing the construction of external trace capture logic. Via an on-chip trace cache, AMDebug technology provides for trace without requiring any external trace capture hardware. Providing for program trace is the greatest challenge, particularly as internal CPU clock frequencies continue to rise.

A superior tool environment can be achieved by integrating support into an OS, or debugger, for on-chip resources that specifically support embedded software development. In this way, a development environment is enabled which is as produc-

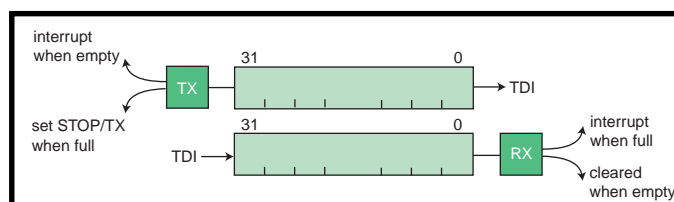


Figure 6—Instructions are provided for the processor to read and write data into the receive and transmit registers. This technique supports bidirectional data transfer.

tive and powerful as the current PC desktop development environment. Creating a seamless, low-cost, full-featured, easy start-up environment will be a compelling reason for selecting a processor for a widening range of embedded products.

Daniel Mann has worked extensively on software and hardware development tools for use with embedded RISC and CISC microprocessors. He has also worked on designing on-chip resources to assist with, and lower the cost of, software development. You may reach him at daniel.mann@amd.com.

SOURCE

AMDebug

Advanced Micro Devices, Inc.

(408) 732-2400

Fax: (408) 732-7216

www.amd.com