Debugging Embedded Systems

Advanced RISC Machines Inc (ARM)

Introduction

A significant portion of product design time is spent on integration and debugging of both hardware and software. This is especially true for systems such as engine management, hard disk control and modems due to their real time constraints.

Deeply embedded cores (i.e. microprocessors which are buried within the heart of an ASIC or custom chip) have made the debug of systems progressively more difficult as there is often limited access to the processors' busses and signals. This is worsened by the use of multi-processor systems such as the controller-DSP architectures common to hard disk drives, pagers and cellphones.

This article briefly examines some of the traditional ways of debugging processor systems. It then considers a new approach that ARM has developed to help solve some of the problems with In Circuit Emulators, Monitor Programs and Logic Analysers.

In Circuit Emulators (ICE)

An ICE contains real-time event detection, real-time tracing and memory emulation, all integrated behind a unified user interface. This often provides the software engineer with a layer of protection from the hardware. In addition, an ICE does not require the system around it to be functional, allowing a degree of parallelism in software and hardware development to reduce time to market.

What problems are associated with standard ICEs?

- An ICE's pod interferes with the normal timing of the target system and may therefore reduce maximum speed.
- Physical processor replacement requires intricate pods which are notoriously unreliable. Replacing the processor also changes electrical characteristics meaning that problems can emerge which cannot be replicated when the ICE is used.
- ICE's lag processor availability. Typically 6-9 months can be expected between processor release and ICE availability.
- A deeply-embedded CPU requires a large "bond-out" chip to bring internal signals out for the ICE to see. Given the effort required to produce an ICE, custom variants of processors may not be supported by ICE at all.
- ICE can be expensive

Debug Monitors

The Debug Monitor resident on the target system is an alternative to ICE, providing the user with many of the features needed to test and debug software such as setting breakpoints, uploading data from target memory and downloading application programs.

An advantage of this approach is that the software being developed can run on the same processor and associated hardware as the final system. In addition, the cost of a Debug Monitor system is low and so every engineer can have one.

However, the need to have the Debug Monitor in ROM on the target system is a significant problem since it must either be removed from the final product or left in ROM at extra cost.

A communication channel to a host computer running the user interface of the debugging software is also required. A UART and suitable line driver in the target system are normally needed for this. The software driver for the UART must be written and successfully ported to the target system before the Debug Monitor will work, and before any system debugging can start.

In addition, the Debug Monitor code must be ported to the target system hardware, meaning that in many systems from a hardware perspective, the majority of the system must be functional before the Debug Monitor can be used at all.

Logic Analysers

These are often useful as a debug tool in addition to either of the above debug methodologies, although on their own they are usually insufficient. This is because they offer only a historical view of the action of code. The user can not change variables or jump to different parts of the program, so "what-if" tests are impossible without recompiling code. In addition, most analysers only have a fixed amount of memory, so the amount of trace per run is limited.

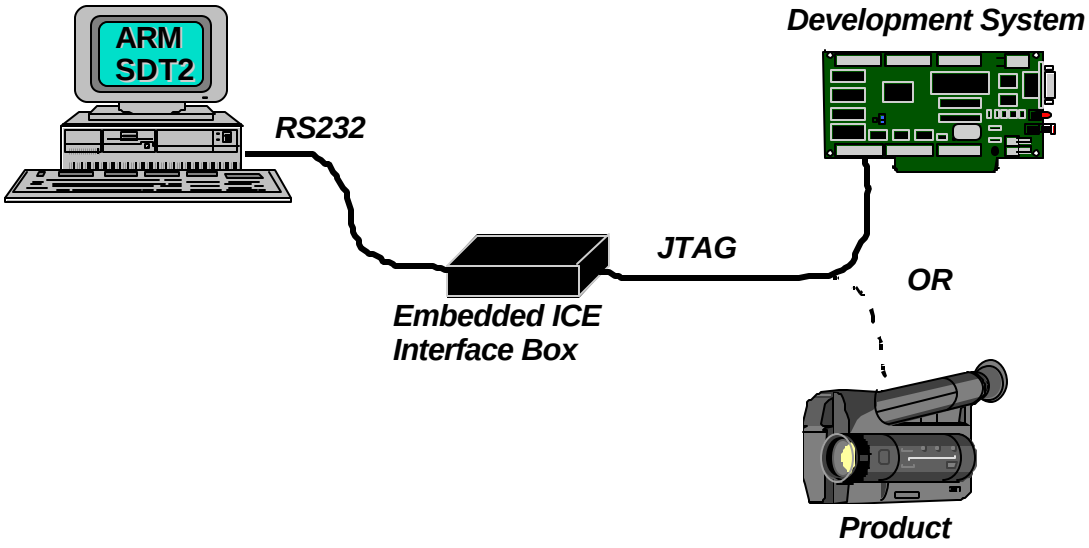
ARM's EmbeddedICE solution

As well as supporting the above traditional debugging techniques ARM has developed a new methodology which solves many of the problems with the traditional debugging routes. In order to simplify debug, this methodology is not limited to either hardware or software development, but provides an integrated approach to system debug.

The "EmbeddedICE" Architecture comprises:

- An EmbeddedICE-compatible ARM core (such as ARM7DI) with a boundary scan interface and debug enhancements.
- An external EmbeddedICE Interface Box which links the host development machine with the debug compatible ARM core.
- The ARM SDT2.01 Host software development and debug tools

The components of the system are interconnected as below:



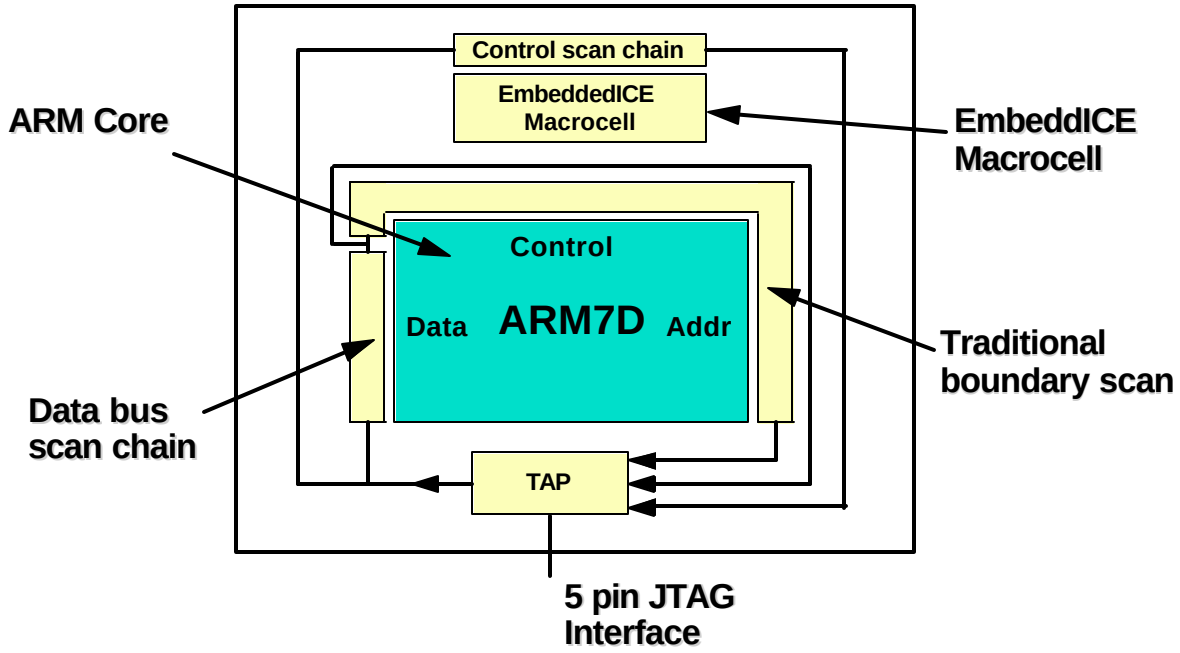
Embedded ICE is a JTAG-based debugging channel for ARM microprocessors. It provides an interface between ARM's Windows toolkit debugger and an ARM microprocessor deeply-embedded within an ASIC.

EmbeddedICE provides ICE functions such as real time address and data dependant breakpoints, single stepping, full control of the ARM CPU and access to the rest of the ASIC system, as well as access to host system peripherals for screen display, keyboard input and disk storage. The latter enables the developer to send debug messages from the target to the host for display on the host screen.

The benefits of ARM's EmbeddedICE solution are:

- No target resources or special hardware, such as ICE pod connectors or serial ports are required. No RAM or ROM in the target system need be set aside for debugging, and no special software need be incorporated (thus there is no need for the software on the target to be modified to work with the ARM EmbeddedICE Architecture).
- Reuse of boundary scan pins means no pin count overhead
- Low cost solution that does not require dedicated ICE silicon.
- Debug can be performed at full processor speed.
- Full host system access including screen, keyboard and storage for the target.
- Processor removal is not required. Debug can happen "in-situ". This solves the problems of expensive pods, unreliability and the change in processor electrical characteristics.
- Requires no extra communication channel to debug
- Works on any deeply embedded ARM system
- Supports multi-processor debug

The ARM7DI Core



An EmbeddedICE-compatible ARM7DI macrocell core comprises an ARM7 core, a small amount of core debug control logic (D), a JTAG test access port (TAP) controller and the EmbeddedICE macrocell (I) as shown above.

The EmbeddedICE macrocell contains breakpoint registers which compare the value on the address, data and control outputs against values programmed into the registers. If a match occurs, a breakpoint signal is generated which is passed to the processor. For example, the macrocell may be programmed to generate a breakpoint when an instruction is loaded from a particular address or a particular data value is stored to a given location.

If the breakpoint was on an instruction fetch, and the instruction reaches the execute stage of the pipeline, then instead of executing the instruction, the processor will enter debug state. From here, the processor state and the memory system may be examined through the JTAG interface via the TAP controller.

Once the processor enters debug state, it stops fetching instructions from the data bus and isolates itself from the memory system. EmbeddedICE may now scan instructions into the pipeline via scan chain 1 and clock the processor. Registers and memory contents can also be examined from debug state. This operation may be repeated in reverse also allowing the user to download code from the debugger into memory, saving the inconvenience of having to blow EPROMs.

Using EmbeddedICE for Multiprocessor Debug

The debug features offered by the EmbeddedICE macrocell allow the ARM processor to be debugged in a multiprocessor environment. When the ARM processor encounters a breakpoint, its execution is halted and control is passed to the debugger through the JTAG interface. At this time the ARM processor signals to the memory system that it is being debugged by asserting a ‘Debug

Acknowledge' signal. Simultaneously it stops requesting memory thus allowing other processors or DMA channels to continue operation and gain access to the memory system. At the end of the debug session, the ARM processor will assert its 'memory request' signal which the system controller will then use to arbitrate for the memory system.

If the user wishes to examine the state of more than one processor during a debugging session, then this may be achieved synchronously. The 'debug halt' conditions may be set up within each processor independently while the system is running. Through use of the TAP controller state machine, all the processors may then be stopped simultaneously. Similarly, at the end of the debugging session, the 'debug restart' conditions may be set up in each processor and again through use of the state machine, the processors may be restarted simultaneously.

The EmbeddedICE Interface Box

The EmbeddedICE Interface Box, converts between the ARM software toolkit's debugger protocol and the protocol required for the JTAG interface. Requests such as "set a watchpoint on this address" are converted to a sequence of JTAG TAP controller state transitions, instructions and data sequences.

The Converter is configurable for different target systems, allowing the user to accommodate, for example, an ASIC with different scan chain layouts to the ARM7DI.

The EmbeddedICE macrocell also supports a communications channel. The Comms channel provides a 'UART-like' serial port on the target system. This is tightly coupled to the processor and requires no extra pins as it re-uses the JTAG port. This approach does require software on the target and removes the need for the UART.

Code development route

Code is developed on a host PC using ARM WindowsTools 2.0. This toolkit comprises C compilers, assemblers and linkers required for code writing.

Debugging support is provided by a full windowing debugger on Microsoft Windows platforms and by a command line debugger for Unix and DOS. These tools provide full C source or assembler level debugging. ARM's debuggers can either debug code running on an instruction accurate simulator (ARMulator) or on target hardware. Switching between software emulation and real silicon is simply a button option on a dialog box. The software tools interface remains the same giving the user seamless switching between targets.

ARMulator can be configured to emulate target hardware with fragmented memory maps of differing speeds. Used in conjunction with the C profiling tool, designers can choose optimal memory configurations for the three critical factors of speed, space and memory cost.

ARMsd, a symbolic debugger allows the user to set breakpoints (on instruction fetches) and watchpoints (on data loads and stores) and examine and modify the state of the processor and memory. This is done in a high level manner independent of the target being debugged which may either be a chip such as ARM7DMI, or the ARMulator.

Semihosting of the target application can also be supported. This means that a program which relies on an ANSI C Library can be ported straight to the target, and calls which cannot be supported on the target can be intercepted by the host. For example C library calls to print status information to the screen can be intercepted and the information can be printed on the host's screen instead.

Summary

There are a number of techniques which are available for debugging, each with differing levels of price and performance. While these tools remain a valuable resource for system designers, more complex and highly integrated systems demand a new level of development environment.

In system designs where the processor core is embedded within an ASIC design the traditional methods of debugging are often no longer suitable and an embedded debug architecture, such as that developed at ARM becomes necessary to ensure that the system can beat the time to market requirements of today's complex systems.

To ensure that a product is developed in the shortest possible timescale it is vital that a complete testing and debugging environment exists, with a toolchain capable of taking the designer from initial product benchmarking, through system design simulation on to final product testing.