"Nexus - Addressing On-Chip Debug Support Requirements for Embedded Controllers"

Ron Stence
Sr. Strategic Marketing and Systems Engineering
32-Bit Embedded Controller Division
Transportation & Standard Products Division
Motorola Semiconductor Products Sector

Technology improvements in the area of embedded microprocessors (MPUs) have changed the methods of debugging and calibration with instruction reconstruction and data collection. This transition has accelerated in part, due to the System-on-a-Chip (SoC) methodology from silicon vendors. The result in systems is the removal of the external address and data bus or the acceptance that the on-chip bus is two to ten times faster than external accesses. Embedded microprocessors with integrated flash memory allow applications to function without an external memory bus. The demands of real-time applications, such as automotive engine management and transmission systems, require embedded MPUs to provide real time information. Semiconductor vendors have been working with the IEEE-ISTO Nexus 5001 Consortium to provide a real-time debug and calibration specification and standard to address these issues.

The use of run control debug ports such as On-Chip Emulation (OnCE), Background Debug™ Mode (BDM), or JTAG have been around for more than a dozen years. However, run control is an intrusive debug technique that relies on the external address and data bus for instruction and data tracing. The integration of on-chip memory systems such as cache, flash and random access memory (RAM) can make the external address and data bus ineffective in providing the necessary visibility to the internal bus activity. The run control debug ports can make tracking down a run time bug very difficult or impossible without specialized debug busses and significant extra expense.

One of the largest issues facing the embedded market is ever-shortening systems development time, with small focused teams developing multiple generations of products in parallel. To enable the engineering teams to hit the shorter cycle times and reduced development cost, a new tools methodology is required. The development tools methodology used in designing and debugging microprocessor-based systems of today will have to improve to become more effective. Over the last decade, the embedded market has seen a proliferation of new MPUs and microcontrollers (MCUs) from 8-bit to digital signal processors (DSPs) to high performance 32-bit RISC MPUs. The development tools used to debug embedded control applications are not capable of providing non-intrusive visibility into the highly integrated embedded processors being deployed today. This problem is further exaggerated with the ever increasing frequency and complexity of newer 32-bit MPUs and MCUs.

Traditional Debug Techniques

The traditional debug techniques found on virtually all embedded MPUs over the past decade depend on visibility to the address and data bus to perform partial real time instrumentation. A brief discussion below of the traditional debugging techniques includes: in-circuit emulation, special bond out chips, logic analyzer pre-processor probes and intrusive run control debug ports (BDM, OnCE, JTAG and others).

In-Circuit Emulators

In-circuit emulation often removes the target MPU from the system logically if not physically. In-circuit emulators suffer from a limitation in the clock speed of around 30-50MHz due to loading and transmission delays. They can be expensive and often have difficulty keeping current silicon in the emulator, which can result in tracking down bugs that are not there any longer. Additionally, an in-circuit emulator might not be able to emulate on-chip features such as floating point units, caches, on-chip peripherals and on-chip memory.

Bond Out Chips

The bond out chips have several issues, including: out-of-date silicon, limitations such as clock speed and bus loading, difficulty in full test and characterization relative to the production versions of embedded MPUs, and can be expensive to maintain.

Logic Analysis

The standard approaches used by logic analyzer companies are clip-on probes or soldered down adapters to gain access to the external signals. These techniques may have been useful several years ago, but with higher levels of integration, faster clock speeds and very fine pin spacing, this approach is changing. Embedded systems without an external bus will make the debugging or even probing the embedded MPU more difficult, if not impossible.

Run Control Debug Ports

Run control debug ports have been on embedded MPUs for almost 15 years. One of the first run control debug ports for a 32-bit embedded MPU was the BDM on the Motorola 68332. Run control debug ports provide an engineer the ability to run-stop the MPU, and examine registers and memory in the system. The common characteristic in these debug ports is that they require the MPU to be halted before the port can transmit the requested information. Some of the more recent run control debug ports have moved to cycle stealing in order to allow limited amounts of information to be sent over the debug port while the MPU is running. The difficulty with this technique is the limited amount of information, reliance on the external address and data bus for information, and the impacts to the MPU performance during run time. The major advantages of on-chip debug ports tools are the relatively low cost and the simplicity of the interface.

Real Time Debug and Instrumentation

Members of the Nexus 5001 consortium are working to make significant improvements in the debug interface for multiple architectures by setting a standard for real-time debug and for instrumentation interface to embedded MPUs. The Nexus 5001 standard will specify common connectors, low-level protocols and on-chip features such as debug registers, pins and signals.

The Nexus 5001 specification includes standard features for setting breakpoints and watchpoints on data and instructions for intrusive and non-intrusive run control debugging. The specification will deploy several unique features for tracking down the most difficult-to-find hardware and software bugs. Some of these new features include ownership trace messaging, data trace, memory substitution, port replacement, program trace, overrun and error messaging. While many of these

features have been deployed on microprocessors over the years, none have implemented all of these together in a real-time debug interface.

The Motorola MPC565 was the first Nexus 5001 Class 3 MCU designed for the automotive market. Tool venders, module suppliers and car companies are using the Nexus 5001 interface on the MPC565 to debug and calibrate engine management systems. Figure 1 is a picture of the die with the two major nexus blocks highlighted. The Nexus debug logic is known as the READI block. The Overlay static RAM (SRAM) is used to overlay other SRAM, internal flash, or external memory. The Overlay SRAM logic will match the number of clock cycles to access the memory being overlaid. This allows the system to have the exact bus performance characteristics during calibration and debug of the system. The READI block contains the internal bus snooping control, watchpoints and breakpoint logic, TCODE translation logic, the transmit FIFO and several other functions.

Slide Presentation - Figure 1: Current Nexus 5001 Implementation on MPC565

The current instrumentation and debugging methodology deploys a “must see every cycle” mindset towards the debug of an MPU-based system. The Nexus 5001 methodology makes four assumptions about the debugging scenario that frees it from the “must see every cycle” approach.

First, the source code and object code are available to the debugging tools. This allows the host-based tools to track or make calculations as to the program flow without any direct address or data bus visibility.

Second, change of flow instructions would be required to maintain synchronization of the host-based debugging tool. When the host instrumentation and/or debugging tool has access to the object code, proper synchronization can be maintained between the MPU and the host tool with change of flow instruction addresses being transmitted over the debugging interface. The Nexus 5001 specification will send out a synchronization message if a change of flow has not forced a synchronization address within 255 instructions. This would be a rare event since most C compiled programs generate a change of flow once every 5 to 10 instructions for embedded MPUs.

Third, a reasonable number of the data locations have to be displayed in real time, while most can be examined during a halted condition or updated due to a special event. The ability of the Nexus 5001 interface to trace on data values will be new for many system design engineers. Often this function is accomplished with a modern logic analyzer tracing the address bus and triggering on the data bus writes to a specific memory location. This is a very cumbersome task that is made nearly impossible with the advent of large data caches and system SRAM on the MPU.

Fourth, if or when an error occurs, the user is informed by the instrumentation tool. The Nexus 5001 interface actively checks for errors and will automatically notify the engineer of the error while re-synchronizing with the target. One error condition that can occur on a serial debug interface is when the FIFO overflows. When this occurs, the user has the option to enforce a stall of the MPU or to continue with a new synchronization message.

Nexus 5001 Features

The Nexus 5001 debug interface is a standard debug methodology across multiple MPU architectures and development tools. Its features come from a mixture of industry best practices in debug interfaces over the past 20 years. Each feature has proven its worth on one or more MPUs architectures in the past. The unique aspect to the Nexus 5001 is bringing the best features all together in a single industry standard debug interface. This will provide the industry with distinct

advantages such as run control, real time instruction and data trace information, RTOS support, memory substation, breakpoints and watchpoints, among others.

The protocol implements a standard packet messaging interface that allows each MPU architecture to remain unique while still communicating with common commands and debug information. This will allow the design engineer to debug multiple architectures with very similar capabilities and tools.

The interface uses the JTAG pins for the basic functions. For higher bandwidth and dedicated communications, the auxiliary port can be added with variable widths. To reduce the information being sent over the interface three features were added. These features were designed to increase the performance of the debug interface without significantly impacting the system cost.

First, the optional auxiliary pins can be removed by the silicon vendor to minimize the cost of the interface when the application can tolerate a reduced bandwidth debug port. The lower pin count implementation might be found on 8-bit and 16-bit devices or in market segments where real-time debugging is less critical.

Second, a packet based messaging scheme has been developed to minimize the overhead of the information between the target MPU and the host-debugging tool. The packet information is called a Transfer Code (TCODE) and is used in conjunction with the auxiliary port pins.

The third method of reducing the information from the debug port to the host development tool is to limit transmissions by sending the required addresses to trace the instruction flow. For example, if the target MPU does not have a change of flow, then the 32-bit address is not transmitted. When a change of flow such as an interrupt or branch occurs, the Nexus 5001 development interface would send the new beginning address. In this environment, it is a requirement that the source code and the linked code are available to the host tool at the time of the debugging session. Also, if the debugging session must be real time, then some information must be derived by the development tools.

Figure 2 provides a high level view of the four areas or classifications the Nexus Debug port. Class 1 is the basic run control that is similar to current BDM, OnCE and JTAG based debug interfaces. Class 2 contains all of the features and functions of Class 1 with the additional features of watchpoint messaging, ownership trace and program trace messaging. Class 3 provides many of the functions required to perform calibration of a powertrain system. Class 3 builds on Class 2 with the additional features of read-write memory access while the MPU is running and data trace messages. The highest level of real time debug capabilities is reserved for Nexus Class 4 with memory substitution and port replacement capabilities. The Nexus 5001 specification directly outlines these capabilities for each Class.

Slide Presentation - Figure 2: Nexus 5001 Development Interface Classes

The interface between the MCU and the development tool uses a packet based messaging protocol to make the interface processor specific independent. Additionally the packet based information is minimized to transmit the changed LSB for addresses. The engineer selects what data is critical to the debug session. Figure 3 is a high level view of the bi-directional interface using the JTAG for commands and the Auxiliary port for high-speed data out information.

Slide Presentation - Figure 3: Nexus 5001 Development Protocol and Pin Interface

Run Control
Architecture specific Run control features are very unique to each MPU's architecture, but the basic types of capabilities are fairly universal and have been for many years. These include the ability to halt and start the MPU; read and write the MPU registers; single step the MPU; and modify memory.

Read-Write Access
Read-Write access allows the debugging tool to access, read and write registers and memory without impacting the MPU's execution of instructions. The Nexus 5001 standard goes beyond what has historically been made available in the area of read-write access by allowing access while the MPU is running. The Nexus 5001 standard requires that memory accesses be allowed while the core MPU is running, which is along the approach of a debug direct memory access (DMA) channel.

Breakpoints
The breakpoint features facilitate the software development process by allowing the developer to halt the MPU core at a specific state. If there is internal ROM or flash or if a breakpoint or trap instruction does not exist in the vendor's architecture then this feature becomes a valuable tool for development. The Nexus 5001 requires at least two address comparison and two data comparison registers be implemented in silicon which can be programmed as breakpoint registers, with up to eight of each.

Watchpoints
The watchpoint features are very similar to the breakpoint features. They use the same address and data comparison registers as do the breakpoints. The difference is that watchpoints do not halt the MPU core but instead send a watchpoint message to the external tools monitoring the Nexus 5001 trace pins. This is very useful in sequencing and triggering external logic analysis collection, both of Nexus 5001 trace information and external I/O signals.

Instruction Trace
The Nexus 5001 standard protocol for instruction trace visibility is microprocessor independent and significantly reduces the amount of information bandwidth needed to transmit the information off chip. The instruction trace feature implements a Program Flow Change Model in which the instruction trace is synchronized at each program flow discontinuity. The address information is compressed to further reduce the bandwidth needed. A program flow discontinuity occurs at taken branches and exceptions. Development tools can interpolate what transpires between program flow discontinuities by correlating information from Instruction Trace information and static source or object code files.

Data Trace
For data trace visibility of accesses to memory locations and memory mapped device-specific internal peripherals, the Nexus 5001 specification defines a minimum of two data trace range resources which can be controlled to emit data trace messages on reads, writes or both within a specified address range. As with the instruction trace feature, the address information is compressed to reduce bandwidth requirements. Additionally, data trace can be globally enabled and disabled by using watchpoint features to further reduce required bandwidth of the Nexus port.

The Nexus Class 3 requirement for data trace information allows a host tool to reconstruct the target's data based on data visibility. A properly configured Nexus Class 3 tool will send data write information over the Nexus port and allow the host tool to reconstruct the total data image. With

only data write information, the host tools can display to the user all of the modified data during a program run. Figure 4 is a screen shot from an Ashling Microsystems tool showing the MPC565 data trace for write only. When the Nexus port is configured to send all data access, it can more easily be overrun or force a stall, depending on the configuration by the user. In addition to data trace information, time stamping, address and actual values can be attached to the data trace stream and the host tool can attach the global or local variable name for ease of debugging and calibration by the user.

Slide Presentation - Figure 4: Data Trace Screen-shot

Ownership Trace

The ownership trace feature allows the software that is currently executing to send messages to external development tools. It is typically used to provide a macroscopic view into the software that is running. For example, it can be used for task or procedure flow reconstruction for debugging software that is written in a high-level language. Ownership trace offers the highest level of abstraction for tracking software execution and is especially useful for embedded MPUs with a memory management unit, in which all processes can use the same logical program and data spaces. Ownership trace offers development tools a mechanism to decipher which set of symbolic and sources are associated with lower-level program instructions. It is expected that a Real Time Operating System (RTOS) will be the primary tool that enables other external tools to determine what is transpiring within the operating system.

Data Messaging

Data messaging provides the capability for operating software to send identified packets of data, which can be arbitrarily large, to external tools. Due to the construction of the data messaging protocol, it is a much more efficiently packed message than data trace messages. A common usage of this feature would be for communicating with rapid prototyping tools, which are widely deployed in the automotive industry, to perform a Remote Procedure Call to an external hardware tool and send all the parameters to the development tool.

Memory Substitution

The memory substitution feature enables memory accesses and instruction fetches from a development tool rather than from the system memory. This allows non-real time execution of software before a system's memory subsystem is fully debugged. Additionally, single stepping with instruction and data fetches via the auxiliary port can be used for a non-real time read only memory (ROM) monitor.

The feature can also be activated upon the occurrence of a watchpoint. It can support easy run-time patching for portions of ROM memory, with the patch provided via the auxiliary port running at a reduced rate. Another option is to activate memory substitution upon the occurrence of a data access or an instruction fetch from a device-specific address range.

Port Replacement and Sharing

For embedded applications, MPU vendors typically scrutinize the use of every physical pin. Often, there are not enough pins available on the MPU to meet both the application and development needs. Port replacement and sharing support is intended to solve this problem by using common MPU ports for a secondary development support function.

Vendor Defined Extensions

The developers of the Nexus 5001 standard understand that there is no way to perfectly predict all of the possible needs for development features for unique MPUs or future MPU features. The vendor-defined extensions allow for additional functionality or optimization without disrupting the ability of standard tools to support the standard features. If a tool does not support a vendor-specific feature, it at least can safely ignore it without disruption of normal functionality.

Multi-Core Debugging

A new PowerPC ISA compatible architecture from Motorola, the MPC5500, will allow multi-core debugging with the family members that are designed for powertrain applications. Figure 5, is a high level block diagram of the first multi-client Nexus 5001 debug port. The first version will allow for a debug port to be enabled for four processing elements. The four processing elements are the e500 PowerPC ISA compatible core, the dual Enhanced Timer Processor Units (E-TPU) and DMA unit. With the single connection over the Class 3 Nexus 5001 interface, the host tool can collect data from any or all of the four elements simultaneously. In addition, the debug and calibration tool could establish breakpoints and/or watchpoints on any of the four processing elements. This will be useful when the user is tracking an issue between a timed event over the E-TPU and the code running on the PowerPC ISA-compatible Book-e core.

Slide Presentation - Figure 5: MPC5500 Multi-Client Nexus Implementation

The Nexus auxiliary port can transfer data on both clock edges for high-speed applications. Another feature of the Nexus auxiliary port is that it can be various port widths depending on the amount of throughput required. The maximum auxiliary port width is 16-bit wide.

Conclusion

Embedded systems hardware and software engineers and calibration engineers will have to adopt new debug and instrumentation strategies with the proliferation of highly integrated MPUs. These changes will be analogous to the transition from assembly language to high-level languages such as C, over which many experienced software engineers in the mid 1980's said was "a loss of control." The embedded MPUs already in the market and coming to market over the next 12 months will make the traditional run-control debug methodology obsolete. The external address and data buses have virtually disappeared from the 8-bit microprocessors and microcontrollers of today. The 16-bit and 32-bit MPUs are, in many cases, reducing the external buses in favor of more specialized I/O functions while integrating the more memory on-chip which leaves little room for the external buses.

The Nexus 5001 consortium has published the current revision of the specification (version 1.0 1999) on the consortium Web site (www.ieee-isto.org).