

M•CORE Architecture Implements Real-Time Debug Port based on Nexus Consortium Specification

*David Ruimy Gonzales
Senior Member of Technical Staff
Motorola M•CORE™ Technology Center
Austin, Texas*

INTRODUCTION

Real-time embedded microcontrollers introduce new challenges to designers as they become more highly integrated. Most embedded controllers execute code from internal memory and provide little to no visibility of how programs flow. Prior techniques for providing visibility of deeply pipelined instruction execution units have cost performance hits as well as increased pin count. Because each silicon vendor may choose different trade-offs between performance and visibility, it is very difficult for tool vendors to produce tools that offer consistent functionality across architectures.

A consortium of competing companies has developed a specification which describes a standard for developing highly embedded microprocessor applications. Originally started by five founding companies (Motorola, Siemens, Hitachi, Etas and Hewlett Packard), there are now twenty companies participating in this consortium. Officially named the Global Embedded Processor Debug Interface Standard (GEPDIS) the project has been code named Nexus.

The objective of the consortium is to define a common set of microcontrollers

on-chip debug features, protocols, pins and interfaces to external tools which may be used by real-time embedded application developers. The consortium has been organized into four committees which address business issues, the technical specification, software and validation. The goal is to pass the specification to a governing body of the Institute of Electrical and Electronic Engineers (IEEE) for institution as a standard. At this time the Nexus Consortium has finalizing revision 1.0 of the specification and is currently working on the IEEE standardization process.

One major objective of the consortium is to help development tool vendors more easily provide a standard set of tools which may be used on a number of embedded microcontrollers. In the spirit of reusability, it has been recognized that many semiconductor vendors currently have debug ports and tool sets that sufficiently address the static debug requirements of their architectures.

Providing a cost effective yet powerful migration path to a standard set of dynamic debug features is a goal of the consortium. More than 70% of leading embedded microcontrollers vendors have dedicated circuits and pins which assist in

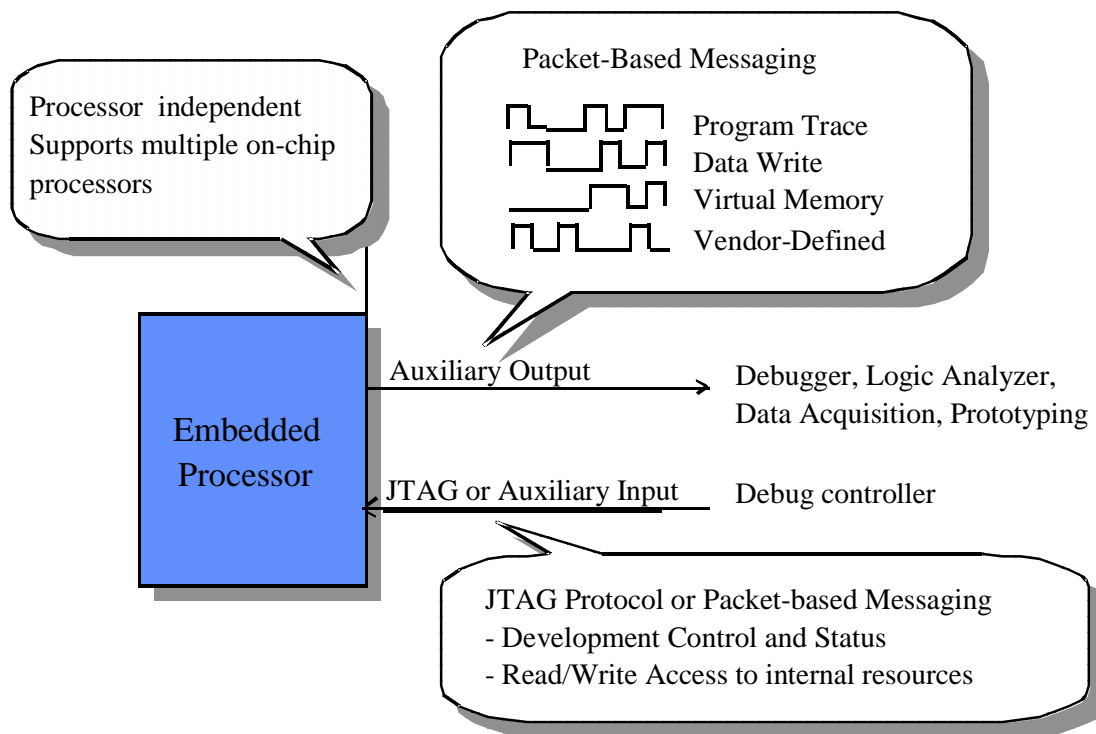


Figure 1 - Nexus Development Port

new product development based on the IEEE 1149.1 Joint Test Action Group (JTAG) 4 wire serial interface.

The JTAG pins and protocol help developers with static debug methodologies in a master-slave mode but there is no means for the embedded microcontroller to initiate real-time information transfers to a host computer.

SCALABLE DEBUG PORT FEATURES

The Nexus standard addresses this need with a scalable set of features whereby existing debug blocks may be used with an extensible auxiliary port. The features associated with this new auxiliary port focus on real-time transfer of information to and from the embedded microcontroller

as illustrated in Figure 1.

To address various levels of development needs the Nexus consortium has categorized static and dynamic debug features according to class levels. These class levels provide a means for implementing a scalable debug block which addresses different market segment requirements. Also, it should be noted that when a product is in development, it is desirable to have as many debug features available because of time to market constraints.

Once a device is put into production, it may not be necessary to have all the development features and pins. A cost savings may be realized by implementing a scalable debug port which meets only requirements needed for specific stages of

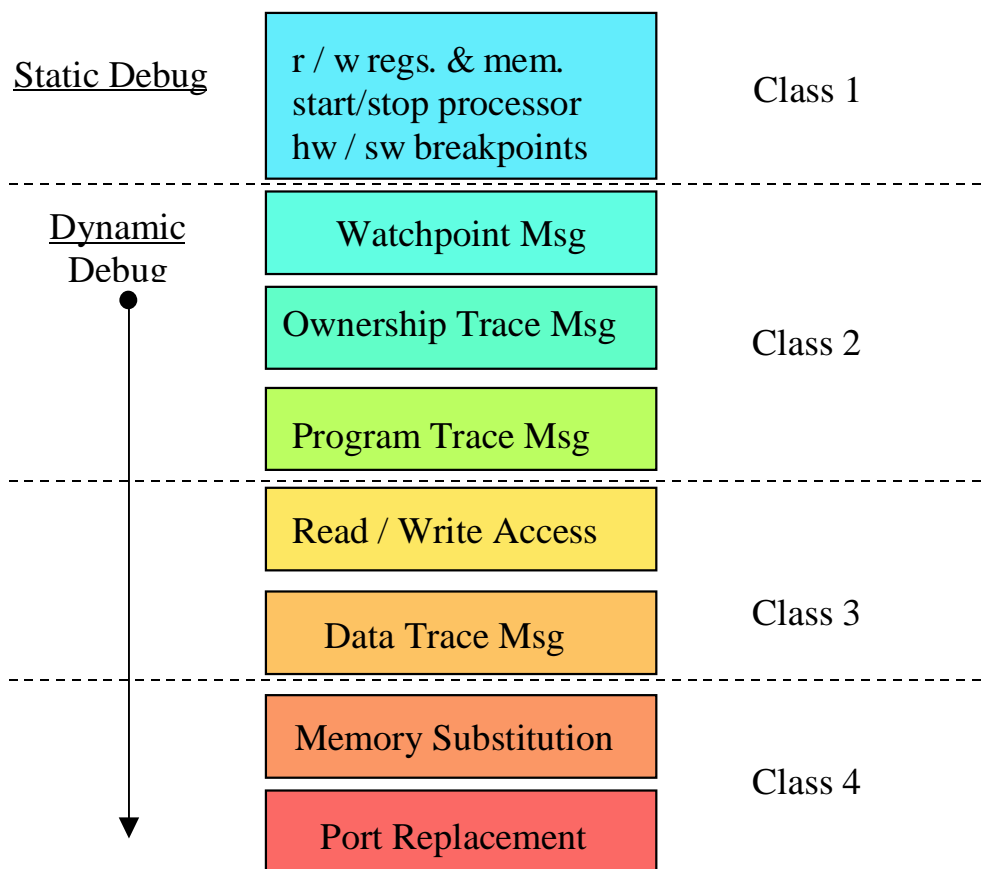


Figure 2 - Nexus Features and Classes of Compliance

the product life cycle.

Figure 2 illustrates the 4 classes of compliance based on features in the proposed global development standard. Each class level increases in complexity from the classical static debug capabilities to complex dynamic debug and each higher numbered class is inclusive of its respective lower numbered class. Another unit of measure for classification is throughput performance of the debug port. Half duplex operation is limited to Class 1 while all other classes require full duplex communication.

Each class level has increasing feature complexity. Class 1 encompasses vendor defined static debug features that exist today and would not require redesign of the respective debug blocks provided they implement the baseline set of features described.

Class 2 and above addresses fundamental real-time debug needs for program flow behavior. Class 3 adds real-time data flow behavior and real-time read/write access to programmer model memory. Class 4 addresses both static and dynamic debug requirements and adds virtual memory access capabilities. It

should be noted that Class 4 is intended for new integrated circuit designs where the debug block is implemented without legacy constraints.

Each class of compliance will require specific protocol packets to transfer information to and from an external host computer. Also, there is a minimum number of Nexus debug registers at each class level as well as pins to meet throughput requirements.

APPLYING REAL-TIME FEATURES USING PUBLIC MESSAGES

A set of data packets, commonly referred to as Public Messages, has been defined for efficient transfer of debug information between the embedded processor and a development system. Public Messages consist of a transfer code or TCODE, source processor identification number, and the data associated with the particular feature being accomplished. A key requirement in the definition of the Public Messages is efficiency, thus packets may be variable in length depending on the TCODE.

Monitoring Program Flow

Three types of public messages have been designed to address program flow behavior. For software architectures which incorporate a real-time operating system the ownership message provides visibility of the process identification. This message can also be used as a poor man's data trace value. The objective of the Ownership Trace Message is to give the most current value of the data bus when a pro-

cess writes to a special address. This address is called the User Base Address where comparators on the Nexus block will trigger a capture of the data bus when this address is written to.

If you need to monitor real-time program flow, there are a group of Branch Trace Messages that give visibility of the program counter's change of flow. The key goal is to efficiency of message information so only changes of program flow will be reported. For more specific types of change of flow, there are messages that tell you if a direct branch or an indirect branch was taken. The difference in the messages is that in a direct change of flow, the only information needed is the number of instructions executed since the last change of flow. This is initiated by a reference address which is normally transmitted to establish where the program counter currently is. After that, all references are made to that address until an indirect change of flow occurs.

If you need to evaluate accesses to data or program memory locations, the Watchpoint Message does the job. This message triggers off existing hardware breakpoint logic a silicon vendor may already have implemented. This is good reuse of existing on-chip hardware so you don't have to redesign a new block. The idea is to set a watchpoint trigger where a signal as well as a message may be transmitted. The message tells which of the watchpoint triggers occurred. This is especially valuable for debugging variable writes. For example, if you have a global variable which is being modified by a number of processes and you want to pinpoint

which of those processes is accessing that variable, the watchpoint message is the tool to use.

Data Trace Messages

Data Trace messages provide a means for reporting real-time data accesses to memory locations. This feature is quite useful for monitoring specific parameters stored in data memory or values being accessed by a memory mapped peripheral port. Data trace qualifiers include the access type, i.e., read/write or either, as well as a start and stop address range. If the data address and access type qualifiers are met, a data messages are generated and sent to the debug port.

Output bandwidth requirements for the debug port are reduced by only sending the unique portion of the data address instead of the complete address. Consequently a data trace message is reconstructed relative to each prior message using a synchronization message as a reference address to begin with.

Auxiliary Access Messages

Auxiliary Access Messages are used to read and write data to the auxiliary control and status registers through the auxiliary port. It provide a means for transferring information to and from the target system at relatively high speed. Configuration of debug port registers is a classical use of the auxiliary access messages. For example, to enable program trace messaging, a write to the auxiliary port development control register (DCR) must be initiated using an auxiliary access message.

Memory Substitution Messages

Memory Substitution Messages are used to emulate a bus where opcodes and data may be accessed through the auxiliary port. Currently the Nexus standard only requires reading of data and/or fetching instructions via the auxiliary port. Discussion is currently underway for definition of a full memory emulation capability for the next specification release.

Memory Substitution Messages support run-time patching for portions of internal ROM memory, with the patch provided via the auxiliary port. A classical example of this is to begin reading instructions upon occurrence of a watchpoint qualifier. Once activated, memory substitution accesses continue until an external development tool disables memory substitution.

M•CORE M340 NEXUS CLASS 2+

The first processor to implement the GEP-DIS standard will be an M•CORE M340 architecture with cache and memory management unit. M•CORE based microcontrollers are widely being used in automotive and hand-held portable applications at an astounding rate. A key element to its success is its low pin count and low power consumption. Common to all M•CORE based microcontrollers is a OnCE™ debug block for rapid new product development utilizing the JTAG protocol for communication. This debug block contains a superset of the features required for Nexus Class 1 compliance. It serves very well for static debug control and contains limited observation of real-time program flow.

To enhance the M•CORE debug block with minimal impact to new designs, features for Nexus Class 2 compliance and Read/Write Access for Class 3 were modeled. Providing real-time program trace visibility and high speed DMA-like accesses to memory mapped resources would add significant capability with low power consumption penalty. Also, by using the OnCE controller to access 5 required Nexus registers, only a small amount of additional controller logic was needed.

Nexus Class 2+ port utilizing the JTAG/OnCE port for configuration of registers and for performing Read/Write Access features. This approach is a hybrid of a 6 wire JTAG/OnCE port with a 6 wire Nexus Auxiliary Port. All static debug features are conducted through the JTAG port including real-time access to memory. Utilizing the JTAG port removed the requirement of adding MDI and MSEI pins as well as a Nexus register decoder.

Figure 3 illustrates an implementation of a

The data path for Ownership Trace Messages, Program Trace Messages and

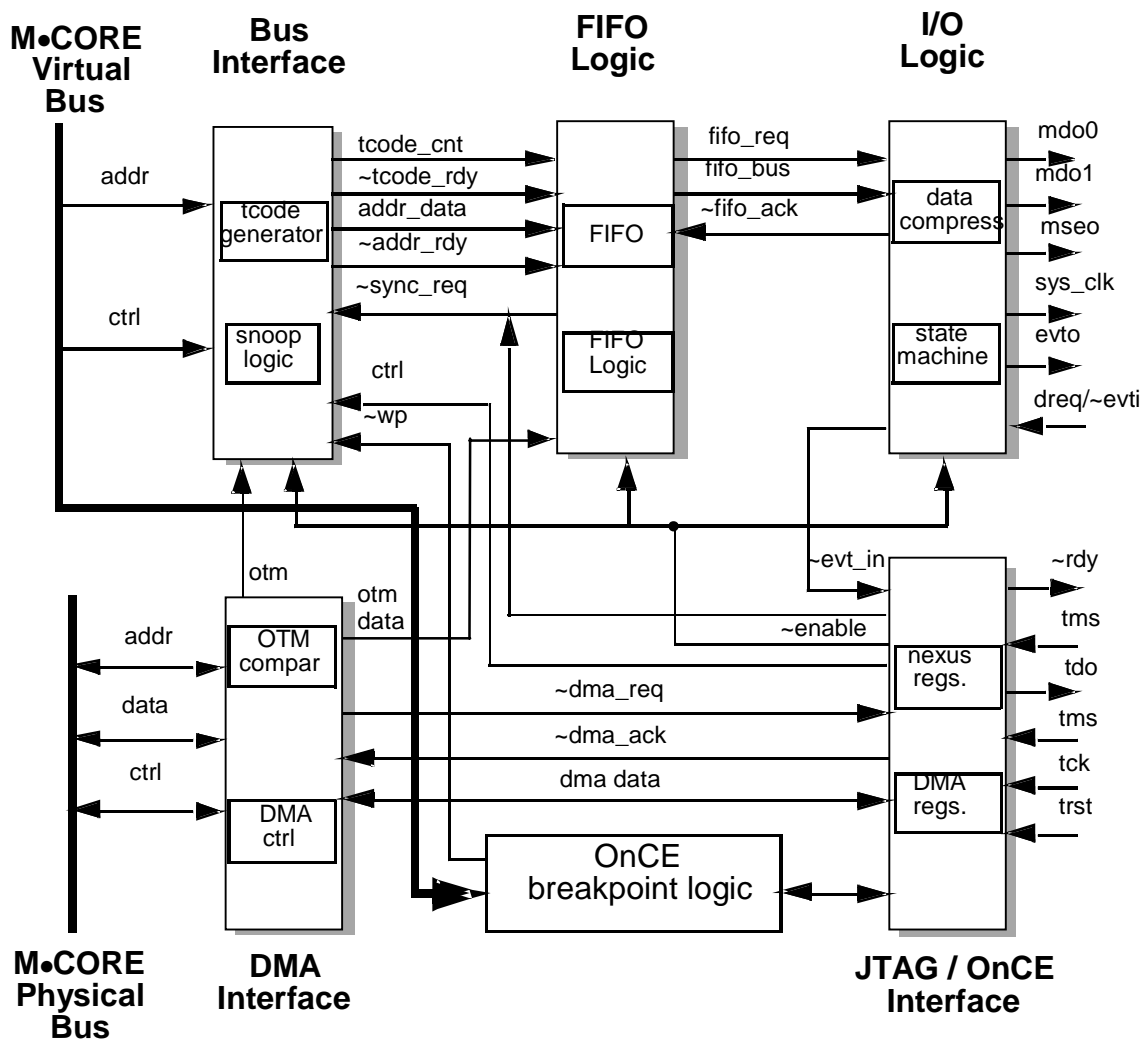


Figure 3 Nexus Compliant Class 2+ on M-CORE Based Architecture

Watchpoint Messages occur through a dedicated Nexus block consisting of 3 sub-blocks. The Bus Interface block snoops the core virtual bus so that program flow within a cache unit could be monitored. The Nexus DCR register in the OnCE interface enables messaging via the auxiliary port. Real-time program change of flow addresses are stored in a FIFO block so that no messages are lost. The FIFO block in turn sends message packets to the I/O block for transmission to the MDO and MSEO pins.

The existing OnCE breakpoint logic was slightly modified to add watchpoint messaging capability. This approach allowed for more sophisticated watchpoint capability as well as removed the task of adding any new comparators to qualify watchpoint messages.

Real-time DMA accesses are performed using the JTAG/OnCE serial interface to the Nexus Read/Write Access circuits. A Ready for Transfer pin (RDY) was added to increase the transfer rate. Calculations show that accesses to the Read/Write Data Register allow for a throughput of 1 megabyte per second on an M•CORE based microcontroller operating at 40mhz system clock.

M•CORE BEHAVIORAL ANALYSIS

A group of automotive/industrial algorithms written in C from the EDN Embedded Microprocessor Benchmark Consortium (EEMBC) were compiled using a Diab Data Compiler and loaded onto a verilog behavioral model of an M•CORE based architecture.

The test bench instantiated the Nexus block so that it was evaluated for throughput capability. Perl scripts were written to take the output data generated by a verilog monitor of the MDO pins. The TCODES were then reconstructed into program counter information to evaluate whether each change of flow was detected and reported.

The idea was to evaluate what the output pin count coupled with the FIFO depth would be required to transfer the messages in real-time Using the protocol defined by the Nexus consortium, it was realized that coherent program flow could be accomplished using only 2 MDO pins, 1 MSEO pin and a 16 message deep FIFO. The FIFO is used to queue all messages for output port.

It was realized that the worst case algorithm had 12% of the executed code that was change of flow. A large portion of that code was short direct branches for interactive loop sequences. Since short branches require a small number of bits to report the distance between branches, the messages were shorter, thus helping the FIFO to stay within a reasonable size. Figure 4 illustrates the maximum FIFO utilization per algorithm.

NEXUS PORT CONNECTORS

To assist in the standardization of development tool interfaces, a group of connectors have been defined which accommodate scalable debug requirements. By providing well defined connectors with fixed pin functions, tool developers as well as system developers

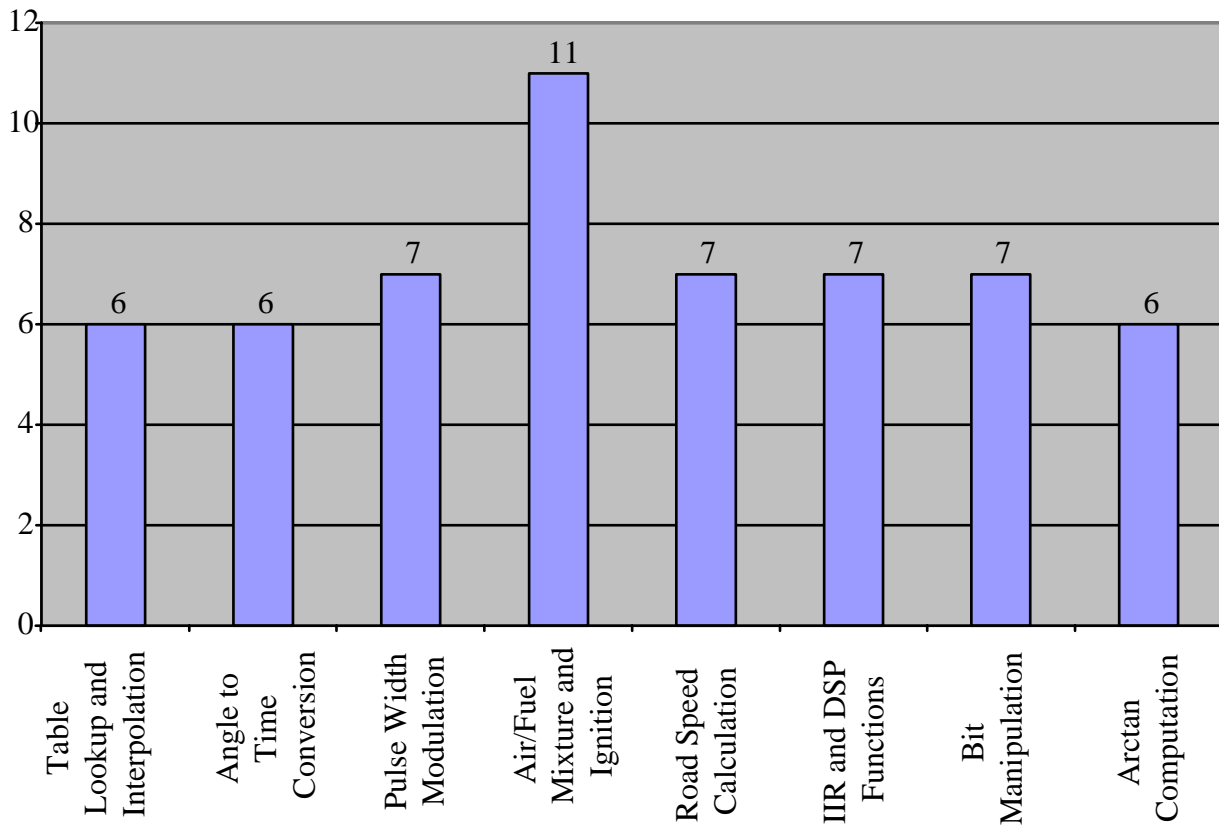


Figure 4 M-CORE M340 Nexus FIFO Utilization for Program Messages

can reuse connectors from one processor design to another. Considerations for low power, high speed interfaces with vendor specific pins have been implemented in design of the connectors.

There are three connectors which may be used where each connector increases in pin count as the development port increases with features. Connector A provides an interface to IEEE 1149 JTAG pins, Connector B provides JTAG and/or a low pin count Auxiliary Port, while Connector C is designed for a full featured Auxiliary port only.

Figure 5 illustrates the interface to a hybrid

of an existing JTAG debug port and the high speed Auxiliary port. The connector selected is the Nexus Standard Connector B (option 3) as illustrated in the Table 1. Each pin is isolated by a ground wire to reduce crosstalk. Logic analyzer vendors are also participating in the consortium and will be supporting the post processing of the Nexus message streams. The processed data will then be presented to some sort of graphical interface which will present the program's behavior as it was executed in real-time. One alternative that has been presented is the current design of logic analyzers today will not efficiently store 2-4 bits of message information in their respective trace buffers. One logic

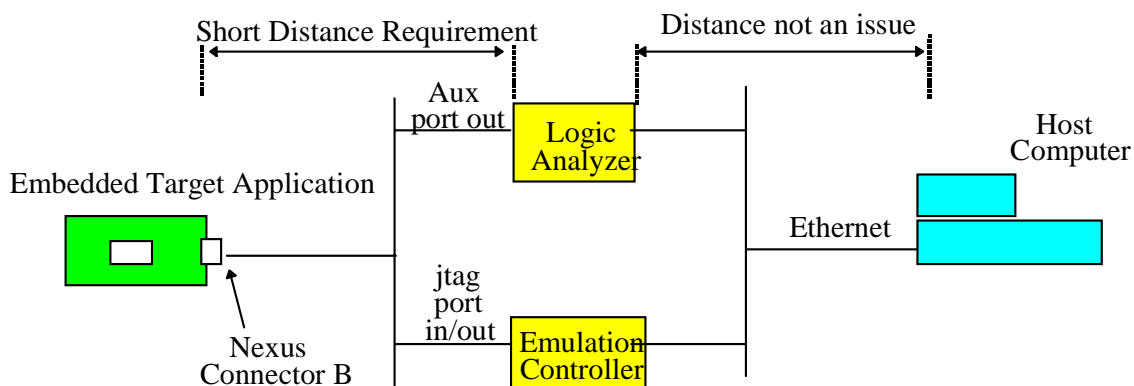


Figure 5 M-CORE M340 Debug Environment

Signal Name	Signal Name	Signal Name	I/O	Pin	Pin	I/O	Signal Name
1) JTAG Mode	2) Aux Mode	3) Mixed Mode					
RESET	RESET	RESET	IN	1	2	OUT	VREF
EVTI*	EVTI*	EVTI*	IN	3	4	-	GND
TRST*	RSTI	TRST*	IN	5	6	-	GND
TMS	RESERVED	TMS	IN	7	8	-	GND
RESERVED	MDI1*	RESERVED	IN	9	10	-	GND
TDI	MDI0	TDI	IN	11	12	-	GND
TCK	MCKI	TCK	IN	13	14	-	GND
RESERVED	MSEI	RESERVED	IN	15	16	-	GND
TDO	MDO3*	TDO	OUT	17	18	-	GND
RDY*	MDO2*	RDY*	OUT	19	20	-	GND
RESERVED	MDO1*	MDO1*	OUT	21	22	-	GND
RESERVED	MDO0	MDO0	OUT	23	24	-	GND
CLOCKOUT*	MCKO	MCKO	OUT	25	26	-	GND
RESERVED	MSEO	MSEO	OUT	27	28	-	GND
EVTO*	EVTO*	EVTO*	OUT	29	30	I or O	Vendor Defined*

Table 1 Nexus Recommended Connector (Option B)

analyzer vendor recommended putting a serial to parallel converter between the target processor and the logic analyzer to improve the trace buffer depth when capturing data. This will require a second method for post processing the information.

SUMMARY

Significant effort is underway throughout the electronics industry to improve tools and methods for designing complex embedded systems. The Global Embedded Processor Debug Interface Standard Consortium, code named Nexus, is a tes-

tament to this and demonstrates that there is a dire need to standardize on a set of features, protocols, pins, interfaces and tools for rapid development of real-time microcontroller based products.

The consortium has visited many customers to educate the design community and there is considerable enthusiasm from the feedback received. The main question asked by customers is "When will this become available on the products I design with?". As with all industry standards, this will take time to be accepted but the momentum, desire and energy is there to accomplish this effort.

Originally targeted to solve automotive engine controller design problems, the Nexus consortium has extended the scope of this effort to encompass telecommunications, industrial and portable hand-held products.

The problems of real-time visibility to deeply embedded microcontrollers is very similar if not identical to most product types. Each will have special cases for solving specific design issues but the proposed global standard has addressed this by allowing for vendor defined blocks for special features, all addressed by a common protocol. The Nexus consortium frequently updates its progress, solicits feedback and comments and places the most current specification on internet web site www.nexus-standard.org.

M•CORE and OnCE are trademarks of Motorola