

Debug Code for ARM Cortex-M3 MCUs

Jon Titus, Senior Technical Editor

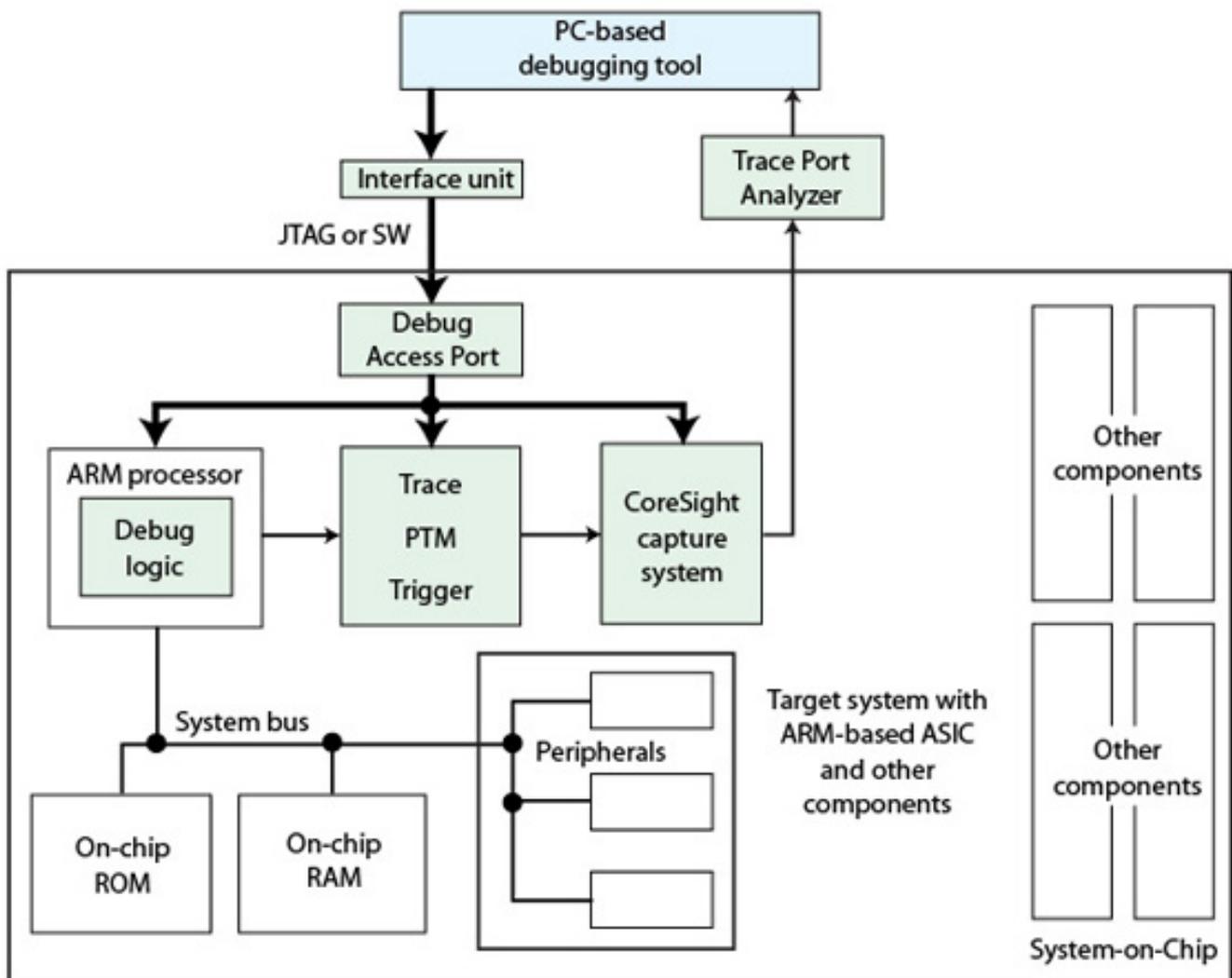


ARM counts six large silicon suppliers as licensees for the Cortex-M3 processor core, which attests to its popularity. Microcontroller manufacturers license the Cortex-M3 core and its attendant debug-and-trace macrocells, called CoreSight. The CoreSight block includes many capabilities, and hardware and software engineers should know how to take advantage of them.

On a Cortex-M3, a debug-port (DP) block gives you access to the ARM core (for clarity, I'll call it the CPU) via either a JTAG or a serial-wire debug interface. Both offer the same debug capabilities, but the serial-wire interface requires fewer pins than a JTAG port. Engineers often refer to the serial-wire interface as a 1-pin interface, but the interface specifies as many as three signals, serial-wire output (SWO), serial-wire data I/O (SWDIO), and serial-wire clock (SWCLK).

"The debug-port block connects to the processor, but it also gives you direct access to memory," explained Javier Orensanz, product manager for debug and profiling tools at ARM. "So software downloads occur faster, and debug tools can access memory at the same time the CPU runs code. If you want to look at variables in memory, for example, the debugger can obtain this information with only a slight increase in the memory system's latency."

Depending on the manufacturer, a Cortex-M3 MCU also may include an eEmbedded-trace Trace macrocell Macrocell (ETM), which generates instruction-trace packets as either serial data or as parallel data on four data lines. The ETM lets tools, such the ULINKpro and Microcontroller Development Kit--MDK--from Keil, run a long-term trace so you can view the "history" of instructions executed by the CPU.



Art 1. This diagram of an ARM Cortex-M3 core highlights the key debug portions of the microcontroller. Courtesy of ARM.

"Then you can generate profile and code-coverage reports," explained Orensanz. "You don't have to 'instrument' your code when you use this hardware capability." Instrumenting code involves placing a few extra instructions in the code to save information in locations you can access later. Because the instrumenting process adds code, it has an effect on performance and timing, and sometimes on the behavior of the software.

"Customers who have any concerns about code safety and reliability can take advantage of the profiling capabilities of the Cortex-M3 CPU and third-party development software," said Jean Anne Booth, director of worldwide Stellaris marketing at Texas Instruments. "These people must prove their code executed all paths through a program. The serial-wire trace [SWT] in a Cortex-M3 core gives you instruction information every few cycles, and when watchpoint matches occur. It doesn't give you a list of every executed instruction, but by using your object code software tools can reconstruct what occurred between the samples."

The Cortex-M3 also provides an instrumentation-trace macrocell (ITM), a central part of the debug logic. "The ITM provides selected trace data over a low speed

Debug Code for ARM Cortex-M3 MCUs

Published on Electronic Component News (<http://www.ecnmag.com>)

access port," said Tomas Hedqvist, global account manager at IAR Systems. "But you don't need a separate external trace probe. A probe such as our J-Link handles the trace functions. The ITM provides 32 stimulus registers, so application software can create short packets of information to transmit via these registers to debug-and-trace software through the SWO output. Software, such as C-SPY on a host PC, sorts packets from the 32 possible registers and displays information for you that relates to specific sections of your code."

"A simple write instruction in your code quickly transfers information to one of the 32 channels," continued Hedqvist. "Unlike a C-language printf function, the write instruction requires little CPU overhead because the CPU does not call a lengthy UART or Ethernet routine to output the register data. Our IAR Embedded Workbench tools use one register to implement a non-intrusive 'printf' type of operation that outputs messages to the terminal I/O window on the host PC. And as an option, the channel can timestamp each packet it sends." A C-language printf function would dramatically increase the size of your object code, too.

"Another ARM component, the data watchpoint and trace (DWT), collects information from the system buses and generates events that the ITM time stamps and transmits on the SWO channel," explained Hedqvist. "Four independent comparators or watchpoints can trigger an event based on an address or data match. You can set a data 'breakpoint,' for example, so an event that affects data in a watched address triggers an event. In the IAR Embedded Workbench tools you point to a location in the memory window or use a dialog window to edit more complex conditions for the breakpoint such as setting the condition to detect when a variable reaches a specific value." The ARM real-time trace capabilities also go by the name Serial Wire Viewer, or SWV.

"Engineers can graph memory data over time to look for unexpected changes or to determine whether the data exceeds a range of values," said Hedqvist. "They also can trace and log interrupt events so you get a good overview of interrupt activities as your code runs."

"Not long ago, you couldn't trace interrupt activity," said Booth of Texas Instruments. "When the CPU branched into an interrupt-service routine [ISR], it masked off or disabled other operations. So to debug an ISR, you halted the CPU and jumped to a debug routine. In a system that ran motors and actuators in real-time, that debugging approach might severely damage equipment."

"When you work with a Stellaris or Cortex-M3 MCU, the interrupt performance is always deterministic and requires either six or 12 cycles," said Booth. "So programmers can write their code with interrupt service routines rather than a polling routine. You use interrupts for all events because it is easy to debug them with the serial-wire viewer."

"Let's say you work with our brushless-DC-motor reference design controlled via CAN bus or Ethernet communications," continued Booth. "You need three software priorities. At the highest level, code controls the pulse-width modulators that drive the motor. Next you have the CAN- or Ethernet-communication code and at the

Debug Code for ARM Cortex-M3 MCUs

Published on Electronic Component News (<http://www.ecnmag.com>)

lowest level you have your main.c code that basically sits in an infinite loop and doesn't have much to do. Turn on profiling in your software tools and you can see that the PWM duty cycles are correct just by looking at the output of the chip's serial-wire viewer. If you need to debug the software, you don't have to stop the CPU, so the motor continues to run and you avoid damaging it."

ARM included a memory-protection unit (MPU) in its core and many MCU vendors have implemented it in their chips. Although not meant specifically for use during debugging, many engineers use it as such. "Say you have a 16 kbyte block of code memory you want to protect from unintended access," explained Booth. "If a pointer in your code goes astray and tries to write to the protected code space--a common problem--the MPU generates a 'fault' condition. Unlike debug capabilities that require little or no code, the MPU requires a programmer to set up registers to control how the CPU handles faults and write the code to report a fault. The Stellaris Peripheral Driver Library, for example, includes information about how to set up and control the MPU. Not only does the MPU protect the code in memory, it also can help you determine why and when your code tried to perform an 'illegal' access." Some development tools report the source of faults and help you interpret them.

For further reading

--, "Application Note 209, Using Cortex-M3 and Cortex-M4 Fault Exceptions."
www.keil.com/appnotes/files/apnt209.pdf [1].

Boys, Robert, "Real-Time Trace: Serial Wire Viewer," Information Quarterly.
www.iqmagazineonline.com/IQ/IQ22/pdfs/IQ22_pgs47-49.pdf [2].

Williams, Michael, "Low Pin-count Debug Interfaces for Multi-device Systems,"
www.arm.com/files/pdf/Low_Pin-Count_Debug_Interfaces_for_Multi-device_Systems.pdf [3].

Yiu, Joseph, "The Definitive Guide to the ARM Cortex-M3," Newnes. ISBN: 978-0-7506-8534-4

Source URL (retrieved on 12/06/2013 - 3:01am):

<http://www.ecnmag.com/articles/2010/07/debug-code-arm-cortex-m3-mcus>

Links:

[1] <http://www.keil.com/appnotes/files/apnt209.pdf>

[2] http://www.iqmagazineonline.com/IQ/IQ22/pdfs/IQ22_pgs47-49.pdf

[3] http://www.arm.com/files/pdf/Low_Pin-Count_Debug_Interfaces_for_Multi-device_Systems.pdf