TEXAS INSTRUMENTS

Hercules™ TMS570LC/RM57Lx Safety Microcontrollers Development Insights Using Debug and Trace Tools

ABSTRACT

This application report provides an overview of the technology and tools, and their applicability for the fastest turnaround time for your embedded system.

Contents

List of Figures

1 Synopsis

The typical embedded product development cycle includes several stages. Debugging and tuning is an important stage of the product development cycle where scalable debugging tools are critical to accelerating the overall product development cycle.
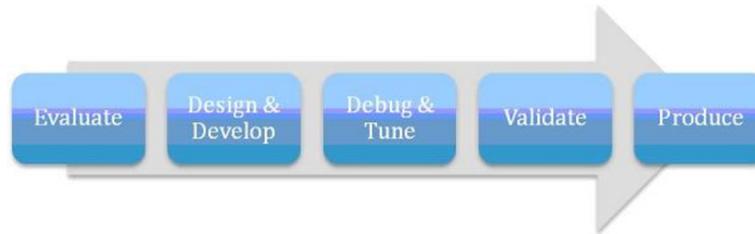

Figure 1. Typical Embedded Development Cycle

Efficient and bug-free software is crucial for taking full advantage of a microcontroller system. Texas Instruments believes that insightful, efficient, and powerful debug technology is critical for its customers' success. TI debugging technology and tools have evolved significantly to keep up with the emerging programming trends. These scale well to various RTOS and applications by providing an inexpensive and developer-friendly environment.

2 Technology Overview

Texas Instruments' Hercules safety TMS570LC/RM57Lx microcontrollers are designed specifically for IEC 61508 and ISO 26262 safety critical applications and provide advanced integrated safety features while delivering scalable performance, connectivity, and memory options. The Hercules microcontrollers have specialized debug and tracing technology for real-time debug controls, visibility into a processor's execution flow, memory system, and interrupts. This section talks about key debug technology available in Hercules microcontrollers.

- ARM® CoreSight™ Technology for ARM Cortex® Debug and Trace
  - Debug execution control (for example, run, step, and halt)
  - Register and memory visibility while halted
  - Debug Access Port (DAP) to directly access the entire memory space of the device without requiring the processor to enter the debug state or halt
  - Hardware breakpoints and data watchpoints for addresses and range
  - Performance measurement units (PMU) for sampling-based performance analysis of cycles and cache events
  - Embedded Trace Macrocell (ETM™) ARM Cortex-R trace with the following functions
    - PC and cycles
    - Data
    - Triggering control for tracing window, address range, and start/stop conditions
- Debug and Trace Interface
  - IEEE 1149.1 JTAG (5 pin) debug interface for stop mode debugging
  - TI ICEPick module for scan path management of all the cores and chip level, clock, and reset management
  - Trace port for off-chip trace collection (requires an external trace receiver)

Hercules, Code Composer Studio are trademarks of Texas Instruments.
CoreSight, ETM are trademarks of ARM Limited.
ARM, Cortex are registered trademarks of ARM.
All other trademarks are the property of their respective owners.

3   Debug and Trace Tools Overview

3.1   *Code Composer Studio*

Code Composer Studio™ (CCS) is an Eclipse-based integrated development environment (IDE) for TI embedded processor families. CCS comprises a suite of tools used to develop and debug embedded applications. It includes debugger, profiler, source code editor, compilers, project build environment and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow.
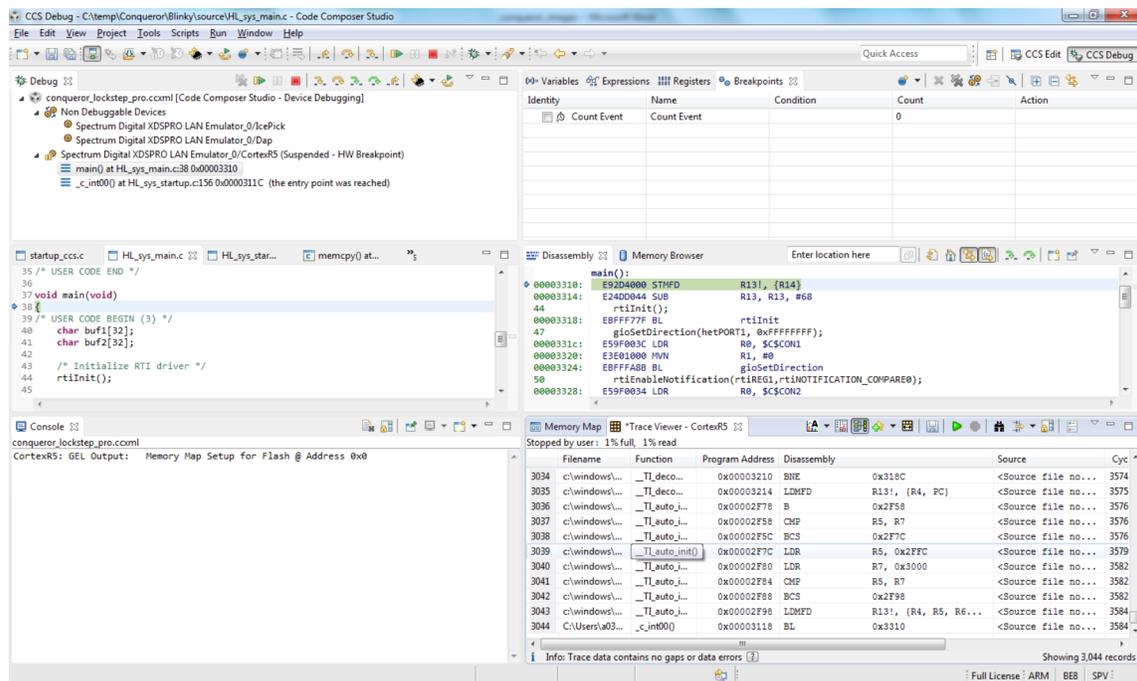
Figure 2. Code Composer Studio

From the Hercules debugging perspective, CCS supports the following functions:

- Debugging
  - JTAG debug for ARM Cortex
  - Source and assembly debugging
  - Advanced register (with CP14) and memory views
  - Integrated flash programmer
  - Software/hardware breakpoints and watchpoints
  - Profile counters/performance monitoring unit (PMU) support
  - Debug code from the reset vector
- Tracing and Profiling
  - ARM Cortex PC trace with code profiling and code coverage support

CCS also has rich scripting support for debugging and profiling via a set of Debug Server Scripting (DSS) APIs. DSS APIs enable scripting through Java, JavaScript, Python, and Tcl. Trace scripting APIs are also available for trace set up and collection to export trace output as comma separated values (CSV). The trace output can be sent to stdout, where it can be processed by piping it to other processing scripts. More details on the scripting APIs can also be found in the <CCS Install>\ ccsv6\ccs_base\scripting\docs directory.

References:		
	Code Composer Studio Details:	http://processors.wiki.ti.com/index.php/Category:CCS
	Debug Scripting:	http://processors.wiki.ti.com/index.php/Debug_Server_Scripting

3.2 XDS Debug Probes and Trace Receiver

The XDS product family includes various debug probes and real-time trace receivers covering needs for entry-level to professional users. XDS products come with various connectors suitable for most target boards.

Table 1. XDS Debug Probes and Trace Receivers

Product	Description
XDS100v2	Entry-level, low-cost JTAG debug probe (also called an emulator) for hobbyist and university applications. Supports a USB 2.0 host interface with TI14, CTI-20, and ARM-JTAG-20 target connectors. Average download speed in the CCS environment is around 30 KB/sec.
XDS2xx	Balanced price/performance JTAG for serious users. Supports USB 2.0 (XDS200) and ENET (XDS220) host interfaces with TI20, TI14, and ARM-JTAG-20 target connectors. The current measurement interface for power profiling requires XDS220. Average download speed in the CCS environment is around 300 KB/sec. Also comes with ARM Serial Wire Debug (SWD) and Serial Wire Output (SWO) for microcontrollers and for Cortex-M microcontrollers.
XDS560v2 STM	High-performance JTAG and cJTAG for professional users. USB 2.0 and ENET host interface with MIPI60, TI20, TI14, and ARM20 target connectors. Average download speed in the CCS environment is around 600 KB/sec. Low bandwidth system trace (STM) receiver with 4 pin, 100 MHz, and 128 MB receiver buffer.
XDS560v2 Pro Trace	High performance JTAG & cJTAG for professional users. USB 2.0 and ENET host interface with MIPI60, TI20, TI14, and ARM20 target connectors. Average download speed in the CCS environment is around 600 KB/sec. High bandwidth dual-channel trace receiver with 22-pin, 250 MHz DDR, and 2 GB storage buffer. Supports Cortex ETM/PTM, TI C6x DSP, and System Trace (STM) protocols.

References:		
	XDS100v2 Details:	http://processors.wiki.ti.com/index.php/XDS100
	XDS200 Details:	http://processors.wiki.ti.com/index.php/XDS200
	XDS560v2 Details:	http://processors.wiki.ti.com/index.php/XDS560v2_System_Trace
	XDS560v2 Pro Trace Details:	http://processors.wiki.ti.com/index.php/XDS_Pro_Trace
	Target Debug and Trace Headers Guidelines:	*Emulation and Trace Headers Technical Reference Manual* (SPRU655)

4 Using the Right Tools for the Job

This section describes when to apply specific tools and techniques to get the best results. These tools support cross development environments across various development stages.

4.1 Boot Code and Application Debugging

System bring-up can be a complex task involving connectivity issues, boot loader troubleshooting, system initialization sequencing, and register and memory visibility. Majority of times the complexity arises due to constrained low-level debug accesses and restrictive tools.

Using JTAG is essential during the bring-up stage, because it enables debug visibility without any software dependency. CCS and XDS JTAG debug probes provide register and memory visibility with or without code running on the target. For downloading or uploading memory, the block memory read/write feature can be very helpful during the bring-up stage. The XDS debug probe also provides the ability to disconnect and cause the debug subsystem to power down when not in use.

In a bare-metal or RTOS development environment, CCS with an XDS JTAG debug probe provides comprehensive debugging functions for code running in RAM and Flash. This includes support for registers, memory, breakpoints, and watchpoints with assembly and source level debugging. The Debug Access Port (DAP) also provides access to memory via AHB-AP and APB-AP via System View and APB View respectively in CCS. Access to runtime memory is done via AHB-AP and user needs to select System View to inspect application variables while the target is running.

DBGJTAG is another tool that comes with CCS for debugging issues such as JTAG scan chain problems, the length of the JTAG test access ports (TAPs), and the reliability of the scan chain. This can be useful when dealing with board-level JTAG connectivity issues.

The CCS General Extension Language (GEL) is a C-like expression language. It can be used to describe various hardware setups, memory initialization, and peripheral configuration routines. GEL routines can be invoked in the debugger interactively or automatically in response to debug events such as a connection or halt. GEL scripts do not need to be compiled or built. GEL is a powerful tool during bring-up, as it provides a way to program the hardware before the software is ready.

References:		
	CCS:	http://processors.wiki.ti.com/index.php/Category:CCS_Training#Getting_Started_Guides
	CCS GEL:	http://processors.wiki.ti.com/index.php/GEL
	Wait in Reset Support:	http://processors.wiki.ti.com/index.php/Wait_in_Reset
	DBGJTAG:	http://processors.wiki.ti.com/index.php/Dbgjtag

4.2 Performance Monitoring

Efficient usage of a device with caches can be a challenging task in the absence of the right tools providing visibility to multiple cache levels during development. This section talks about various tools that are at your disposal to understand cache behavior for their applications.

4.2.1 ARM Advanced Features View

The *ARM Advanced Features* view in CCS provides debugging controls and visibility related to MPU, cache, and coherency controls. These controls provide a direct mechanism to control low-level hardware features; they can be helpful during driver debugging. The controls allow you to change settings, such as enabling and disabling of the data cache, without making changes to the kernel code. Be careful about using these controls, as an incorrect operation may result in unexpected behavior, including a processor crash.
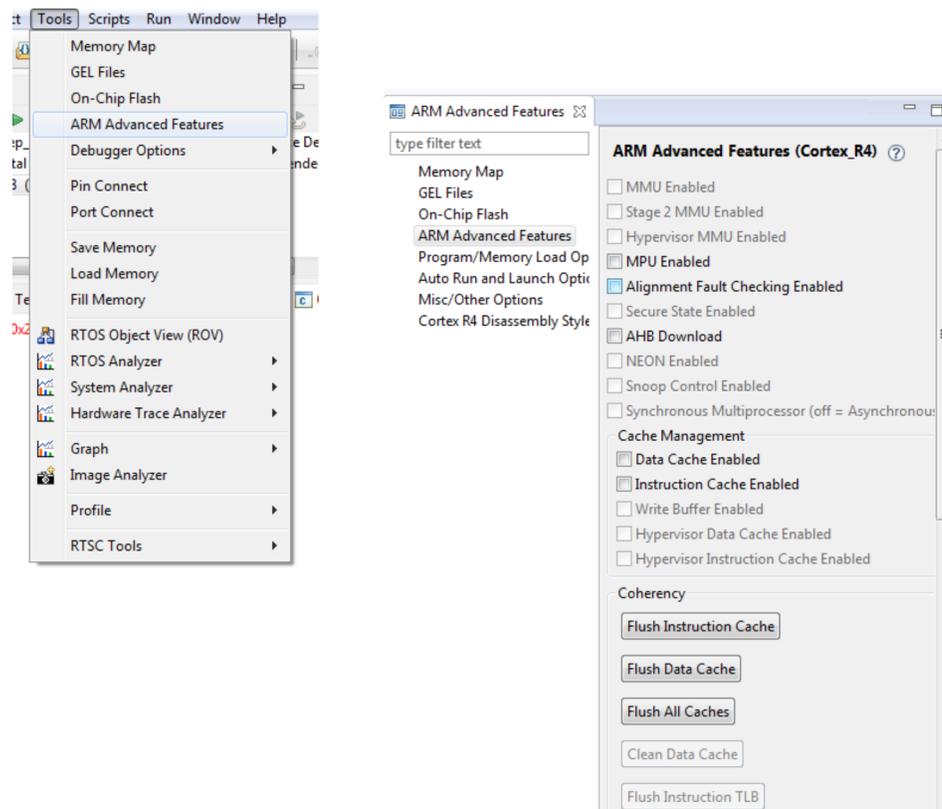
Figure 3. ARM Advanced Features View

4.3 ARM Performance Monitoring

ARM Performance Monitoring Unit (PMU) counters can be used for benchmarking and performance analysis. These counters provide statistical information on CPU cycles, exceptions, cache, and other events. CCS also provides a PMU-based profiling interface via JTAG. The general technique is as follows:

1. Stop the program where the analysis should begin.
2. Program the PMU to count the desired events and reset the counters.
3. Execute the program to the point where the analysis should end.
4. Read the PMU counters, check for overflows, and compute metrics.
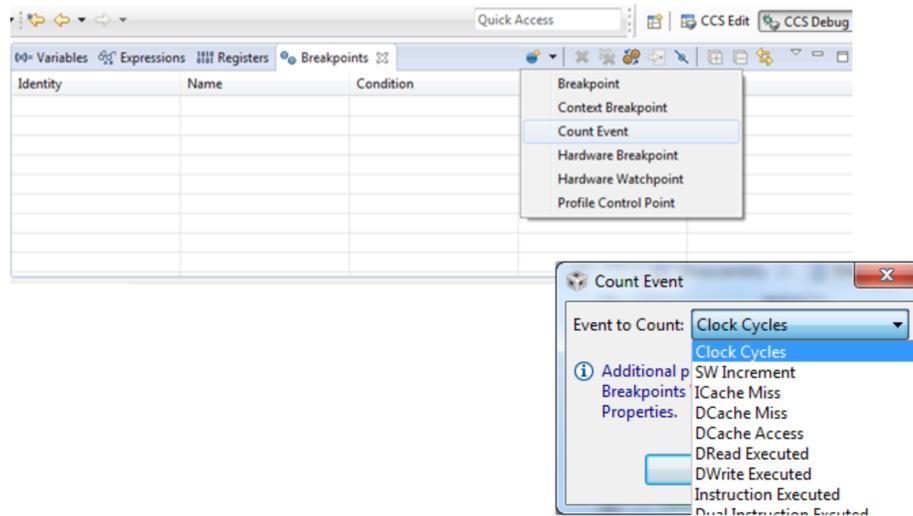
Figure 4. PMU Profiling in CCS

Copyright © 2014, Texas Instruments Incorporated

The following list summarizes some of the key PMU events available for TMS570LC/RM57Lx microcontroller. Information on more events is available from the CCS Cout Event dialog box.

- Clock Cycles
- Data Cache Miss
- Datac Cache Access
- Data Read Executed
- Data Write Executed
- Data Dependency Stall
- Data Cache Write Back
- Data Cache Data RAM Fatal ECC Error
- Data Cache Tag or Dirty RAM Fatal ECC Error
- External Memory Request
- Instruction Cache Miss
- Instruction Executed
- Dual Instruction Executed
- Instruction Buffer Full Stall
- Instruction Cache Tag ECC Error
- Instruction Cache Data ECC Error
- Non-Cacheable Access on AXI Bus
- Exception Taken
- Exception Return Executed
- Data Cache Tag ECC Error
- Data Cache Data ECC Error
- Change to ContextID Executed
- Software Change of PC
- B,BL and(or) BLX Immediate Executed
- Procedure Return Executed
- Unaligned Access Executed
- Branch Unpredicted (not predicted)
- LSU Busy Stall
- Cycles of Disabled FIQ
- Cycles of Disabled IRQ

4.4 Tracing Applications for Execution Visibility

Hard-to-find intermittent issues can evade developers as traditional debugging techniques can be frustrating and time consuming. Performance measurements in a real-time environment are also challenging, and often do not provide the needed granularity and non-intrusiveness. ARM Cortex R ETM trace presents itself as a powerful tool for complex debug, profiling, code coverage needs. While using ETM trace, no changes to the application are required and the trace data can be uploaded and analyzed without halting target. The trace output includes program counters (PC), cycles, and data trace (R/W) at full speed in a loss-less manner.

Hercules devices export the ETM trace over trace pins and the trace information can be captured via a trace receiver such as XDS560v2 Pro Trace. Trace Analyzer in Code Composer Studio (CCS) provides a unified interface to setup and analyze ETM trace for debugging and profiling needs. Various triggering options for trace include Trace Always or Trace On, Trace in Range, Start/End Trace, and Trace Variables.
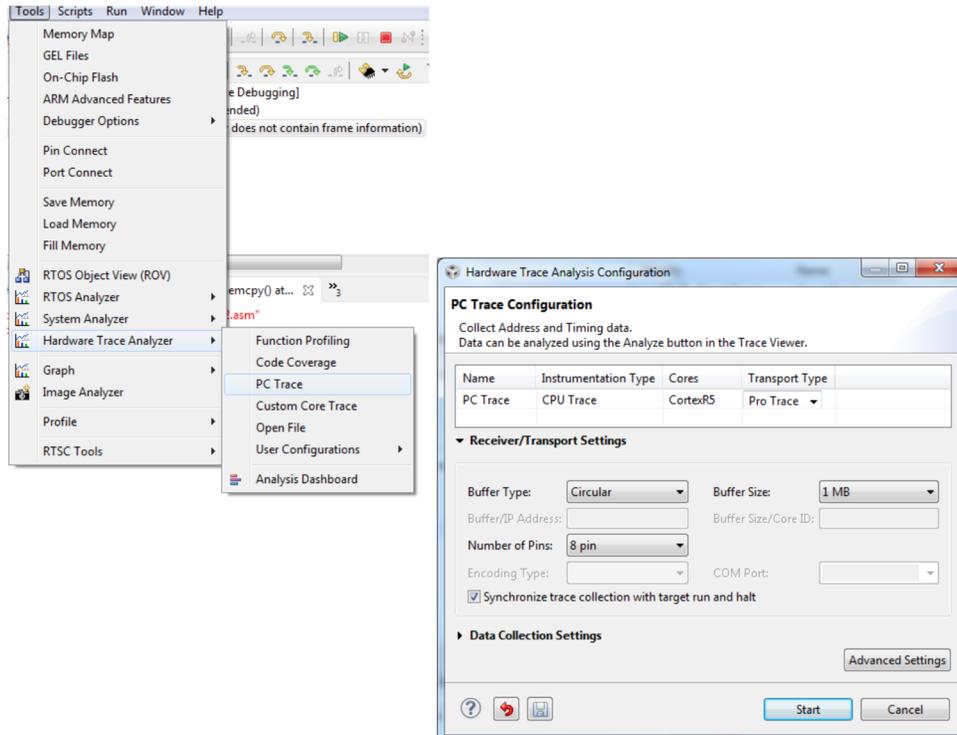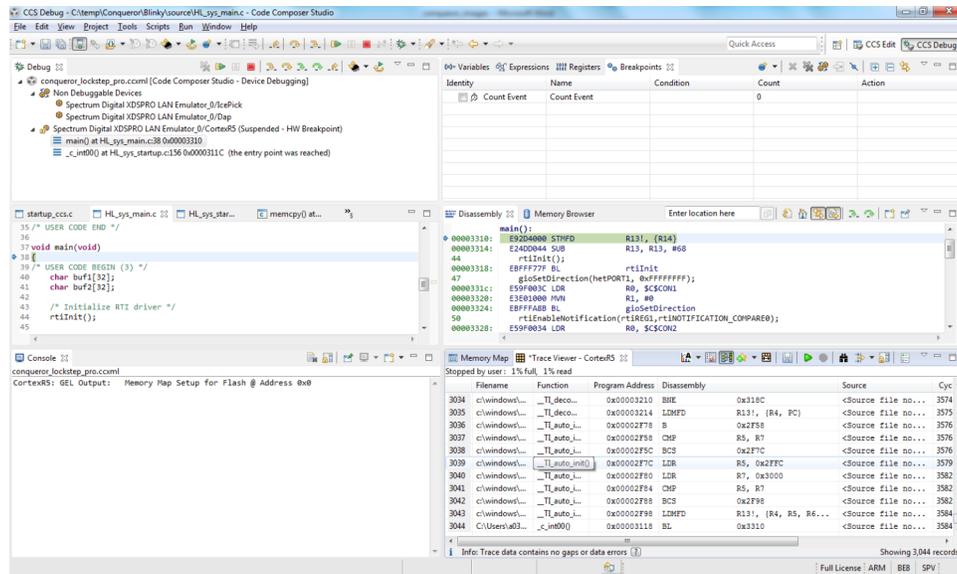
Figure 5. Trace Setup and Configuration

Figure 6. CCS With Trace Output Viewer

References:		
Using Trace:		http://processors.wiki.ti.com/images/3/36/Spruhm7.pdf

Copyright © 2014, Texas Instruments Incorporated

4.5 Detecting Memory Corruptions and Stack Overflows

While debugging programs, you may find that a certain location in memory is being corrupted, without knowing where it is being changed in the source code. Array overruns and memory overwrites are typical causes of such memory corruptions. These can happen when a data variable is assigned an illegal value or an unexpected write occurs due to a bug in some other part of the application. You can set a hardware watchpoint at a variable or an absolute address to catch such conditions.

For stack overflow detection, watchpoints can be used to trap the causing condition. You can set a watchpoint to watch a single address or range of addresses just below the last memory location of the stack. If the program accesses that memory, the watchpoint will trigger, and you will know that the stack has overflowed.

ARM Cortex ETM trace can also be used in conjunction with the watchpoints to captures PC execution sequence. On watchpoint trigger when processor halts, trace collection also stops. The collected trace information contains sequence of program execution until the stack corruption and provides next level of visibility into the potential stack corruption causing events.

4.6 Tracking Exceptions and Complex Real-Time Issues

Issues such as race conditions, intermittent glitches, code runaway, and false interrupts may not be easy to reproduce with stop mode debugging. Many times such issues are caused by spurious interrupts or bad ISR code. Halting the application during ISR execution can result in undesired or non-reproducible behavior.

Real-time tracing is an important tool for debugging such issues where stopping the processor is undesired and affects application. ARM Cortex R ETM trace with XDS560v2 Pro Trace can be used to get run-time execution visibility. The ETM program trace (PC trace) with cycle accurate tracing provides execution sequences and associated cycles. Intermittent glitches or code runaways cause processor exception with the program counter information that caused the exception. However, this does not tell about the execution and events sequence leading into the exception. Trace is the fastest way of getting such visibility in a very short amount of time with a certainty. When using trace, you can start tracing the program with a condition to end at exception vector. The trace includes PC sequences until the exception and information on spurious and nested interrupts to narrow down the cause of the issue.

4.7 Profiling

ARM Cortex ETM trace provides a mechanism for non-intrusive application profiling. This does not require any software instrumentation in the code. The trace-based profiling commonly provides function-level inclusive or exclusive profiling information.

Profiling setup can be defined for the entire program, address range, or with a defined start address and stop address points. The profiling configuration also allows you to choose whether you want this analysis to include TI libraries or not. The output includes inclusive and exclusive level of profiling results.
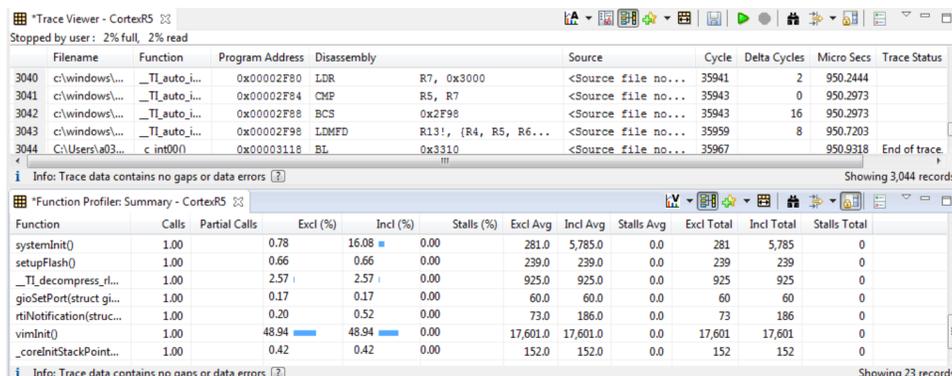

Figure 7. Trace-Based Function Profiling

Copyright © 2014, Texas Instruments Incorporated

The ETM trace information can also be processed to provide a function execution graph from the trace analysis view. The execution graph shows caller and callee relationship with cycle count spent in a given call. It can be used to measure the number of cycles between operations.
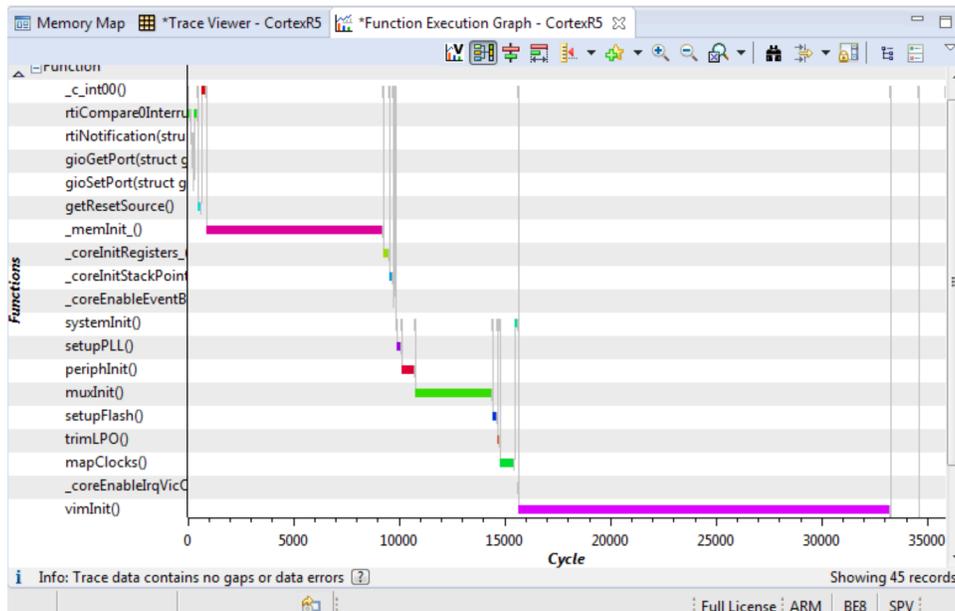

Figure 8. Trace Based Execution Graph

References:		
🌐	*Using ARM Trace for profiling*:	http://processors.wiki.ti.com/images/3/36/Spruhm7.pdf

4.8 Code Coverage

The ARM Cortex ETM trace can also be leveraged for getting non-intrusive code-coverage information. The trace-based code coverage analysis includes information on lines of code and functions that have been executed during a program run. You can use code coverage analysis to verify that your test cases exercise all portions of your code. Metrics for function coverage and statement (line) coverage are also provided. Code Composer studio also enables trace based statistical code coverage by providing an option to accumulate and merge trace collection from multiple trace collection sessions. There are four types of code coverage information is available.

• Function Coverage
• Line Coverage
• File Coverage
• Instruction Coverage
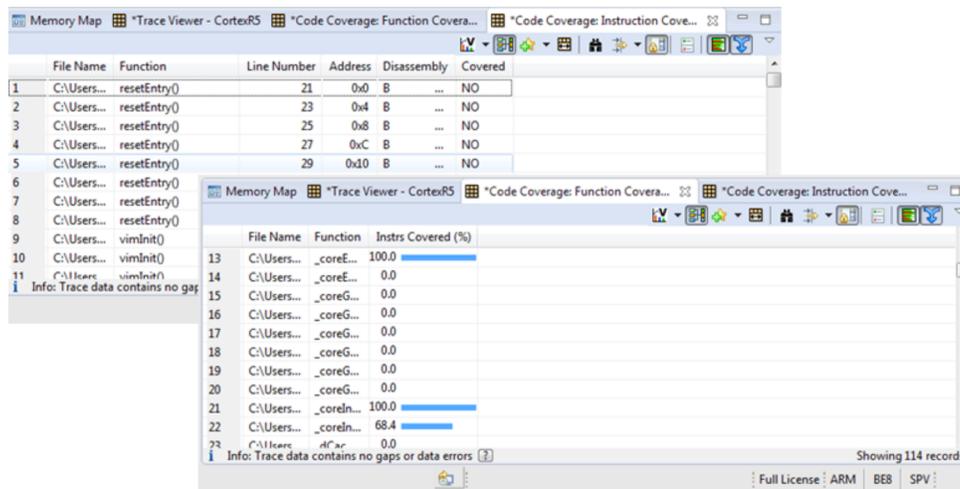
Figure 9. Trace-Based Code Coverage

4.9   Scripting

CCS Debug Server Scripting (DSS) is a set of Java APIs for debug session scripting. The CCS scripting APIs provide support for debug and trace scripting through Java, JavaScript, Python, and TCL, and so forth. Previously described debug and trace capabilities can be used from the scripting environment for testing or quality checks purpose. The trace scripting APIs provide trace, profiling, and code coverage output into command separated text files and the output can be further processed via external tools. More details on the existing scripting APIs can also be found in a CCS installation directory <CCS Install>\ ccsv5\ccs_base\scripting.

References:	
Debug Scripting:	http://processors.wiki.ti.com/index.php/Debug_Server_Scripting

5   Training Resources

Additional training material — including white papers, tutorials, and videos on Code Composer Studio - is available via TI websites and the Texas Instruments Wiki.

References:	
CCStudio Training:	http://processors.wiki.ti.com/index.php/Category:CCS_Training

IMPORTANT NOTICE AND DISCLAIMER