

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

1	Synopsis	2
2	Technology Overview	2
	Debug and Trace Tools Overview	
	Using the Right Tools for the Job	
	Training Resources	

List of Figures

1	Typical Embedded Development Cycle	. 2
2	Code Composer Studio	
3	ARM Advanced Features View	
4	PMU Profiling in CCS	. 7
5	Trace Setup and Configuration	. 9
6	CCS With Trace Output Viewer	. 9
7	Trace-Based Function Profiling	10
8	Trace Based Execution Graph	11
9	Trace-Based Code Coverage	12



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)



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.

📑 = 🔄 🕲 🐨 🖉 = 🖄 🖄 🔹 = 🕹 💣 = 😂 🔜 🐽 👧 🔊										
	🕨 🔝 📕 🛛 🖣	* • 🔗	- 0	Þ•≎•			Qui	ck Access	😭 🛛 🐯 CCS Edit 🛚 🎭 CC	CS Debu
🏇 Debug 🛿 🦓 📴 🗟 🥱 🔊 🤌	🐟 🛛 🕹 🗢		(×)= Vari	ables 🏘 Expressi	ons IIII Register	s 💁 Breakpoir	its 🖾	💣 🕶 🙁 🍇 🍪		
a 😳 conqueror_lockstep_pro.ccxml [Code Composer Studio - Device Debugging]			Identity	,	Name		Condition	Count	Action	
ØWon Debuggable Devices Spectrum Digital XDSPRO LAN Emulator, 0/LePick Spectrum Digital XDSPRO LAN Emulator, 0/LePick Spectrum Digital XDSPRO LAN Emulator, 0/LoneAS (Suspended - HW Breakpoint Emulator) at HL_systematical 80.00003310 Cint000() at HL_systematical 80.00003310 (the entry point was reached)	t)			친 Count Event	Count Event			0		
□ startup_ccs.c □ HL_sys_main.c ☆ □ HL_sys_star ⓒ memcpy() at	» ₅		🔛 Disa	issembly 🛛 🚺 I	Nemory Browser		Enter location here	0 2 6 30)	7 - E
36 STATUSER CODE END */		^	0000 0000 0	main():	00 STMFD	R13!,	[P14]			^
37 void main(void)			0000				13, #68			E
38 (44		nit();					
39 /* USER CODE BEGIN (3) */ 40 char buf1[32];			0000 47			rtiIni				
41 char buf2[32];		=	47		etDirection(h	R0, \$C				
42			0000			R1, #0	pconi			
43 /* Initialize RTI driver */			0000				Direction			
<pre>44 rtiInit();</pre>			50				,rtiNOTIFICATION_CO	MPARE0);		
45			0000	3328: E59F00	34 LDR	R0, \$C	\$CON2			
4		,								
	📑 🖻 🕆 📑 🔻	-		mory Map 🏼 *T		texto 23	🔐 👻 🔛	v 🕶 🗄 🔡 🕨 🌰	🛱 🏇 🕶 🔂 🔚 🛸	~ - 6
conqueror_lockstep_pro.ccxml	E ⊑ + 📑 י			d by user : 1% full,	1% read			≥ ▼ ⊞ <u> </u> ▶ ●		
© Console ⊠ conqueror_lockstep_pro.ccxml CortexR5: GEL Output: Memory Map Setup for Flash @ Address 0x0			Stoppe	d by user : 1% full, Filename	1% read Function P	rogram Address	Disassembly		Source	Сус
conqueror_lockstep_pro.ccxml	1 (j) ¥ [j] Y	^		d by user : 1% full, Filename	1% read		Disassembly	¥ ▼ 🖽 🔛 Þ ● x318C		Сус
onqueror_lockstep_pro.ccxml			Stoppe	d by user : 1% full, Filename c:\windows\	1% read Function P	rogram Address	Disassembly BNE 0		Source	Cyc 3574
onqueror_lockstep_pro.ccxml		*	Stopper 3034	d by user : 1% full, Filename c:\windows\ c:\windows\	1% read Function P _TI_deco	rogram Address 0x00003210	Disassembly BNE 0 LDMFD R	x318C	Source <source file="" no<="" td=""/> <td>Cyc 3574 3575</td>	Cyc 3574 3575
onqueror_lockstep_pro.ccxml			Stopper 3034 3035	by user : 1% full, Filename c:\windows\ c:\windows\ c:\windows\	1% read Function F _TI_deco _TI_deco	rogram Address 0x00003210 0x00003214	Disassembly BNE 0 LDMFD R B 0	x318C 13!, {R4, PC}	Source <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576</td>	Cyc 3574 3575 3576
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036	filename c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function F _TI_deco _TI_deco _TI_auto_i	rogram Address 0x00003210 0x00003214 0x00002F78	Disassembly BNE 0 LDMFD R B 0 CMP R	x318C 13!, {R4, PC} x2F58	Source <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576 3576</td>	Cyc 3574 3575 3576 3576
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036 3037	d by user: 1% full, Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function F _TI_deco _TI_deco _TI_auto_i _TI_auto_i	rogram Address 0x00003210 0x00003214 0x00002F78 0x00002F58	Disassembly BNE 0 LDMFD R B 0 CMP R BCS 0	x318C 13!, {R4, PC} x2F58 5, R7	Source <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576 3576 3576</td>	Cyc 3574 3575 3576 3576 3576
onqueror_lockstep_pro.ccxml			3034 3035 3036 3037 3038	by user: 1% full Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function F _Tl_deco	rogram Address 0x00003210 0x00003214 0x00002F78 0x00002F58 0x00002F55	Disassembly BNE 0 LDMFD R B 0 CMP R BCS 0 LDR R	x318C 13!, (R4, PC) x2F58 5, R7 x2F7C	Source <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576 3576 3576 3576 3579</td>	Cyc 3574 3575 3576 3576 3576 3576 3579
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036 3037 3038 3039	d by user: 1% full Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function P _Tl_deco	rogram Address 0x00003210 0x00003214 0x00002F78 0x00002F58 0x00002F5C 0x00002F7C	Disassembly BNE 0 LDMFD R B 0 CMP R BCS 0 LDR R LDR R	x318C 13!, (R4, FC) x2F58 5, R7 x2F7C 5, 0x2FFC	Source <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576 3576 3576 3576 3579 3582</td>	Cyc 3574 3575 3576 3576 3576 3576 3579 3582
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036 3037 3038 3039 3040	d by user: 1% full, Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function F _Tl_deco	rogram Address 0x00003210 0x00003214 0x00002F78 0x00002F58 0x00002F5C 0x00002F7C 0x00002F80	Disassembly BNE 0 LDMFD R B 0 CMP R BCCS 0 LDR R LDR R CMP R	x318C 13!, (R4, PC) x2F58 5, R7 x2F7C 5, 0x2FFC 7, 0x3000	Source <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576 3576 3576 3576 3579 3582 3582 3582</td>	Cyc 3574 3575 3576 3576 3576 3576 3579 3582 3582 3582
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036 3037 3038 3039 3040 3041	by user: 1% full, Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function F	rogram Address 0x00003210 0x00003214 0x00002278 0x00002258 0x0000255C 0x0000257C 0x00002580 0x00002580	Disassembly BNE 0 LDMFD R B 0 CMP R BCS 0 LDR R LDR R LDR R BCS 0	x318C 13!, (R4, PC) x2F58 5, R7 x2F7C 5, 0x2FFC 7, 0x3000 5, R7	Source <pre><source file="" no<br=""/><source file="" no<br=""/><source file="" no<br=""/><source file="" no<br=""/><source file="" no<br=""/><source file="" no<br=""/><source file="" no<="" pre=""/></pre>	Cyc 3574 3575 3576 3576 3576 3576 3579 3582 3582 3582 3582
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036 3037 3038 3039 3040 3041 3042	by user: 1% full, Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function Function T_deco T_deto T_auto_i	rogram Address 0x00003210 0x00003214 0x00002578 0x00002558 0x00002570 0x00002570 0x00002580 0x00002584	Diassembly BNE 0 LDMFD R B 0 CMP R BCS 0 LDR R LDR R LDR R LDR R LDR R LDR R LDR R LDR R LDR R LDR R	x318C 131, {R4, PC} x2F58 5, R7 x2F7C 5, 0x2FFC 7, 0x3000 5, R7 x2F98	Source <source file="" no<br=""/> <source file="" no<="" td=""/> <td>Cyc 3574 3575 3576 3576 3576 3576 3579 3582 3582 3582 3582 3582 3582</td>	Cyc 3574 3575 3576 3576 3576 3576 3579 3582 3582 3582 3582 3582 3582
onqueror_lockstep_pro.ccxml			Stopper 3034 3035 3036 3037 3038 3039 3040 3041 3042 3043	by user: 1% full, Filename c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\ c:\windows\	1% read Function Function T_deco T_deto T_auto_i	rogram Address 0x00003210 0x00003214 0x00002F78 0x00002F5C 0x00002F70 0x00002F70 0x00002F80 0x00002F84 0x00002F88 0x00002F98	Diassembly BNE 0 LDMFD R B 0 CMP R BCS 0 LDR R LDR R LDR R LDR R LDR R LDR R LDR R LDR R LDR R LDR R	x318C 13!, [R4, FC] x2F58 5, R7 x2F7C 5, 0x2FFC 7, 0x3100 5, R7 x2F98 13!, [R4, R5, R6	Source (Source file no (Source file no	Cyc 3574 3575 3576 3576 3576 3576 3579 3582 3582 3582 3582 3582 3582

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.



Debug and Trace Tools Overview

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.

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.

Table 1. XDS Debug Probes and Trace Receivers

Refer	ences:	
۲	XDS100v2 Details:	http://processors.wiki.ti.com/index.php/XDS100
۲	XDS200 Details:	http://processors.wiki.ti.com/index.php/XDS200
۲	XDS560v2 Details:	http://processors.wiki.ti.com/index.php/XDS560v2_System_Trace
۲	XDS560v2 Pro Trace Details:	http://processors.wiki.ti.com/index.php/XDS_Pro_Trace
۲	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.

Refer	rences:	
۲	CCS:	http://processors.wiki.ti.com/index.php/Category:CCS_Training#Getting_Started_Guides
۲	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



Using the Right Tools for the Job

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.

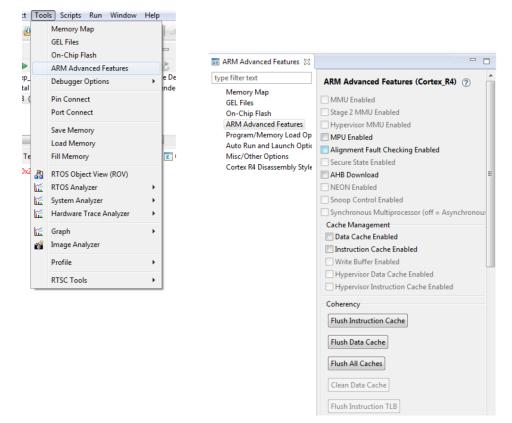


Figure 3. ARM Advanced Features View



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.

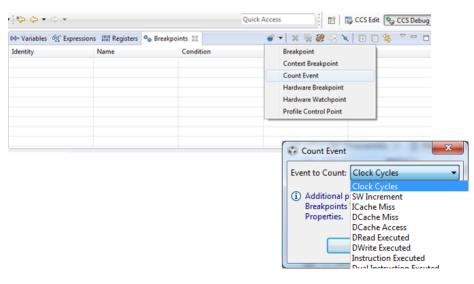


Figure 4. PMU Profiling in CCS



Using the Right Tools for the Job

www.ti.com

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

8

- 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.



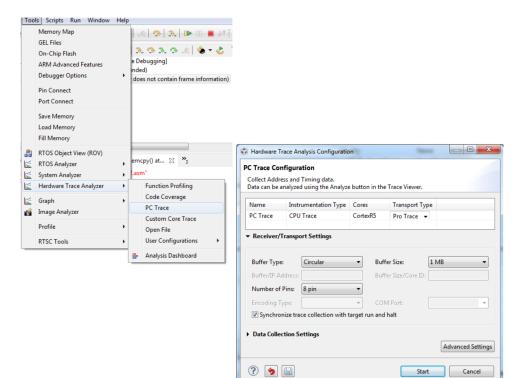


Figure 5. Trace Setup and Configuration

Eile Edit View Project Tools	Scripts <u>R</u> un <u>W</u> indow <u>H</u> elp									
📬 🕶 🔛 🌚 🖃 🗣 🕫) 🔀 🔹 • 🕹 🔮 • 🖾 i 🕾 i 🔊 i 😣 🔿 🕹 🕷 🚳	🌣 • 🔗	• 9	↓ ↓ ↓			Q	iick Access	😭 📑 CCS Edit 🛚 🎭 CC	CS Debu
🏇 Debug 🛙	🎕 🕨 🗉 🔳 🔍 👁 🔊 Le 💊 * 🕹	~	00+ Var	riables 🍕 Expressio	? 😔 🗙 🖽 🖽 🍣 💎					
 Image: Spectrum Digital XDS Spectrum Digital XDS Spectrum Digital XDSPR(Image: Spectrum D	iPRO LAN Emulator_0/IcePick iPRO LAN Emulator_0/Dap D LAN Emulator_0/CortexR5 (Suspended - HW Breakpoint)		Identit	y 参 Count Event	Name Count Event		Condition	Count 0	Action	
35 /* USER CODE END */ 36 void main(void) 38 { 39 /* USER CODE BEGIN (3) 40 char buf1[32]; 41 char buf1[32]; 42 /* Initialize RTI di 43 /* Initialize RTI di 44 rtiInit();			 0000 0000 44 0000 47 0000 0000 0000 50 	3318: EBFFF7 gioS 331c: E59F00 3320: E3E010 3324: EBFFFA	00 STNFD 44 SUB hit(); 7F BL etDirection(h 3C LDR 30 MVN 8B BL nableNotifica	rtiIni netPORT1, 0xF R0, \$C R1, #0 gioSet	13, #68 t FFFFFF); \$CON1 Direction ,rtiNOTIFICATION_C)	7
€		•		+						•
Console 🔀	🔒 🚮 🛃 🖉 🖛 🗌	•••		emory Map 🏾 *Tr		texR5 🔀	🔐 👻 🔛	🛠 = 🖽 🔛 🕨 🖷	🛱 🐎 🛛 🕹 🛛 🖯	~ - 6
onqueror_lockstep_pro.ccxml	ory Map Setup for Flash 🕯 Address 0x0		Stoppe	ed by user: 1% full,						
ortexes, dec output. Hem	ing hap becap for Frasin @ Audress exe	· · · · · ·				Program Address			Source	Cyc
			3034		_TI_deco	0x00003210		0x318C	<source file="" no<="" td=""/> <td></td>	
			3035		_TI_deco	0x00003214		R13!, {R4, PC}	<source file="" no<="" td=""/> <td></td>	
			3036		_TI_auto_i	0x00002F78		0x2F58	<source file="" no<="" td=""/> <td></td>	
			3037		_TI_auto_i	0x00002F58		R5, R7		
			3038		_TI_auto_i	0x00002F5C		0x2F7C	<source file="" no<="" td=""/> <td></td>	
			3039		_TI_auto_init()	0x00002F7C		R5, 0x2FFC	<source file="" no<="" td=""/> <td></td>	
			3040		_TI_auto_i	0x00002F80		R7, 0x3000	<source file="" no<="" td=""/> <td></td>	
			3041		_TI_auto_i	0x00002F84		R5, R7	<source file="" no<="" td=""/> <td></td>	
			3042		_TI_auto_i	0x00002F88		0x2F98	<source file="" no<="" td=""/> <td></td>	
			3043		_TI_auto_i	0x00002F98		R13!, {R4, R5, R6	<source file="" no<="" td=""/> <td></td>	
			3044	C:\Users\a03	_c_int00()	0x00003118	BL	0x3310	<source file="" no<="" td=""/> <td>358</td>	358
			4							
				fo: Trace data conta	ins no gaps or da				Showing 3,04	44 reco

Figure 6. CCS With Trace Output Viewer

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



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.

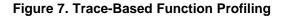
Real-time tracing is an important tool for debugging such issues where stopping the processor is undesired and affects application. ARM Cortex R ETM trace with XDS560v2 Pro Trace can be used to get run-time execution visibility. The ETM program trace (PC trace) with cycle accurate tracing provides execution 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.

itoppe	d by user : 2% fu	ll, 2% read											
	Filename	Function	Program A	Address Di	sassembly			Source		Cycle	Delta Cycles	Micro Secs	Trace Status
3040	c:\windows\	TI_auto_i	0x000	02F80 LI)R	R7, 0x3000		<source< td=""><td>file no</td><td>35941</td><td>2</td><td>950.2444</td><td></td></source<>	file no	35941	2	950.2444	
3041	c:\windows\	TI_auto_i	0x000	02F84 C1	1P	R5, R7		<source< td=""><td>file no</td><td>35943</td><td>0</td><td>950.2973</td><td></td></source<>	file no	35943	0	950.2973	
3042	c:\windows\	_TI_auto_i	0x000	02F88 B0	cs	0x2F98		<source< td=""><td>file no</td><td>35943</td><td>16</td><td>950.2973</td><td></td></source<>	file no	35943	16	950.2973	
3043	c:\windows\	TI_auto_i	0x000	02F98 LI	OMFD	R13!, {R4, R	5, R6	<source< td=""><td>file no</td><td>35959</td><td>8</td><td>950.7203</td><td></td></source<>	file no	35959	8	950.7203	
3044 ∢	C:\Users\a03	c int000	0x000	03118 BI		0x3310		<source< td=""><td>file no</td><td>35967</td><td></td><td>950.9318</td><td>End of trace.</td></source<>	file no	35967		950.9318	End of trace.
			1.1									Showin	ng 3,044 records
i In	fo: Trace data con	tains no gaps	or data errors	4								3110101	ing 5,044 records
	fo: Trace data con unction Profiler: S								١	- 🔡 🍫	- 🖽 🛔		
	unction Profiler: S	ummary - Cor		Excl (%) Incl (%)	Stalls (%)	Excl Avg	Incl Avg		▼ 🔡 🏠 Excl Total	▼ 🖽 🚔 Incl Total		
₩ *Fi Funct	unction Profiler: S	ummary - Cor	texR5 ⊠ Partial Calls		%) Incl (%) 16.08 =	Stalls (%) 0.00	Excl Avg 281.0	Incl Avg 5,785.0				≱ - 🔠 🛛	
Funct syster	unction Profiler: S ion	ummary - Cor Calls	texR5 🛛 Partial Calls	Excl (, , ,			-	Stalls Avg	Excl Total	Incl Total	🐝 ▾ 🚺 🛿 Stalls Total	
Funct Funct syster setupl	unction Profiler: S ion nInit()	ummary - Cor Calls 1.00	texR5 🔀 Partial Calls	Excl (0.78	16.08	0.00	281.0	5,785.0	Stalls Avg 0.0	Excl Total 281	Incl Total 5,785	Stalls Total	
Funct Funct syster setupl TI_d	unction Profiler: S ion nInit() Flash()	Calls 1.00 1.00	texR5 🔀 Partial Calls ((Excl (0.78 0.66	16.08 • 0.66	0.00	281.0 239.0	5,785.0 239.0	Stalls Avg 0.0 0.0	Excl Total 281 239	Incl Total 5,785 239	Stalls Total 0	
₩ *Fi Funct syster setupl _TI_d gioSel	unction Profiler: S ion nInit() Flash() lecompress_rl	Calls 1.00 1.00	texR5 🔀 Partial Calls ((2 (Excl (0.78 0.66 2.57	16.08 ■ 0.66 2.57	0.00 0.00 0.00	281.0 239.0 925.0	5,785.0 239.0 925.0	Stalls Avg 0.0 0.0 0.0	Excl Total 281 239 925	Incl Total 5,785 239 925	Stalls Total 0 0	
Funct Funct syster setupl _TI_d gioSel	unction Profiler: S ion nInit() Flash() lecompress_rl tPort(struct gi ification(struc	Calls 1.00 1.00 1.00 1.00	Partial Calls	Excl (0.78 0.66 2.57 0.17	16.08 • 0.66 2.57 • 0.17	0.00 0.00 0.00 0.00 0.00	281.0 239.0 925.0 60.0	5,785.0 239.0 925.0 60.0	Stalls Avg 0.0 0.0 0.0 0.0	Excl Total 281 239 925 60	Incl Total 5,785 239 925 60	Stalls Total 0 0 0 0	





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.

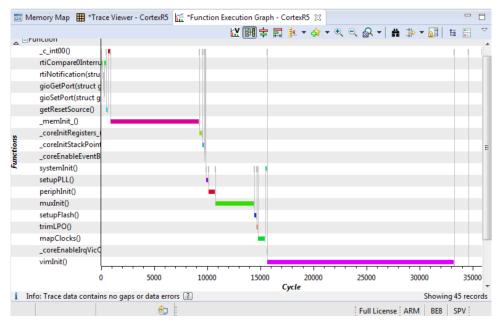


Figure 8. Trace Based Execution Graph



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



								11	r 🔠 🍫	- 🖽	# 🔅	- 🗕 🔂			۲ ک	7				
	File Name	Function		Line Numb	er Addres	s Disa	assembly	Cove	ered							^				
1	C:\Users	resetEntry()		2	1 0x) B		NO												
2	C:\Users	resetEntry()		2	3 0x	4 B		NO												
3	C:\Users	resetEntry()		2	5 0x	B		NO												
4	C:\Users	resetEntry()		2	7 0x0	В		NO												
5	C:\Users	resetEntry()		2	9 0x1	B		NO												
6	C:\Users	resetEntry()	TH M	emory Map	I Trace	linuar.	CortexPS		Code Co		unction	Covera	~	m •c.	ode Ce		Instructi	on Cours		
7	C:\Users	resetEntry()	NT IVI	entory map	inace	viewer	Contexio		CODECO	relage. Fi	unction					-				
8	C:\Users	resetEntry()										11	▼ I	· 🗞 ا	- 🖽	A.	🏞 🔻 🙆) 🗄 🛛		
9	C:\Users	vimInit()		File Name	Function	Instra	Covered ((%)												
10	C:\Users	vimInit()	13	C:\Users	_coreE	100.0		-												
11 ,	C-\Ilcore	vimInit() contains no gar	14	C:\Users	_coreE	0.0														
	nro: Trace data	contains no gap	15	C:\Users	_coreG	0.0														
			16	C:\Users	_coreG	0.0														
			17	C:\Users	_coreG	0.0														
			18	C:\Users	_coreG	0.0														
			19	C:\Users	_coreG	0.0														
			20	C:\Users	_coreG	0.0														
			21	C:\Users	_coreIn	100.0		-												
			22	C:\Users	_coreIn	68.4														
			22 i In	C-\Ilcerc fo: Trace data	dCar contains n	0.0 o gaps	or data err	rors 🛛	2									Showing	114 rec	ord
			1 10	IO. HACE GAL	Contains n	o gaps	en data en		0						1.6.		se : ARM	1 1		SPV :

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.

Refe	References:								
۲	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:		
۱	CCStudio Training:	http://processors.wiki.ti.com/index.php/Category:CCS_Training

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2022, Texas Instruments Incorporated