Application Report
SLAA674A–August 2015–Revised February 2016

MSP432™ Debugging Tools: Using Serial Wire Output With Code Composer Studio™ Hardware Trace Analyzer

Jiacheng Jason He and Dung Dang

ABSTRACT

This application note introduces an ARM® hardware-based debugging tool, Serial Wire Output (SWO) Trace. The discussion starts with background information on what happens at a hardware level, to explain what the many capabilities are. Then, it focuses on how the tools are implemented in the TI Code Composer Studio™ (CCS) integrated development environment (IDE), compared to other IDEs.

In CCS, the SWO Trace tools are presented in the form of three main use cases: Statistical Function Profiling, Data Variable Tracing, and Interrupt Profiling. A fourth, Custom Core Trace, lets the user customize what triggers are set and what events are recorded by the hardware.

This application note explains how to use SWO Trace in CCS (called Hardware Trace Analyzer), demonstrate with a simple Out of Box example, and explain further configuration and customization. By using this application note as a guide, users should be able to implement the Hardware Trace Analyzer debugging tools in CCS to view the large projects in smaller parts to fully understand what is happening.

Software examples and related collateral can be downloaded from http://www.ti.com/lit/zip/slaa674.

Contents

List of Figures

List of Tables

1 Introduction to Serial Wire Output (SWO)

The TI Code Composer Studio (CCS) IDE and other IDEs provide many debugging tools to help with software development. The MSP432™ MCUs use the ARM® Cortex®-M4F processor with FPU, which introduces a number of advanced debugging capabilities. To effectively understand and use these debugging tools, this application note introduces background information on ARM functions, and then describes what the MSP432 and CCS specifically can do with these tools.

1.1 *ARM® CoreSight™ Components Overview*

CoreSight components are the ARM debugging tools. The available debugging hardware modules vary depending on the processor selected. Detailed comparison of processor models can be found in the ARM CoreSight Components Technical Reference Manual (see Reference 1). This application note focuses on how to leverage the SWO Trace debugging tools with the specific ARM Cortex-M4F implementation on MSP432 MCUs.

1.2 *ARM Cortex-M4F Components*

Of the CoreSight components, the ARM Cortex-M4F specifically offers many options to add on to the basic core. Figure 1 shows all of the possible add-ons.

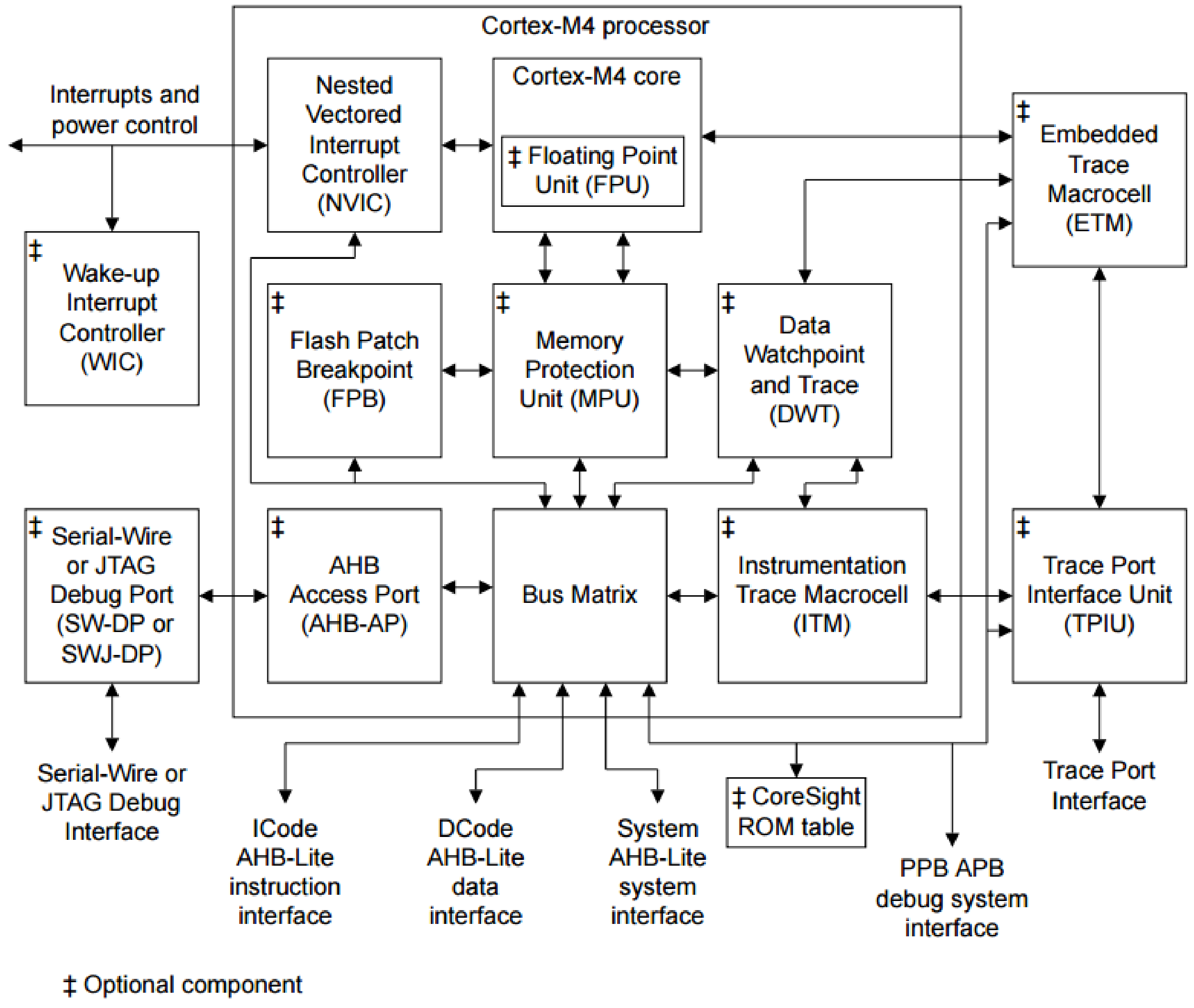

Cortex-M4 processor
Interrupts and power control
Nested Vectored Interrupt Controller (NVIC)
Cortex-M4 core
‡ Floating Point Unit (FPU)
‡ Embedded Trace Macrocell (ETM)
‡ Wake-up Interrupt Controller (WIC)
‡ Flash Patch Breakpoint (FPB)
‡ Memory Protection Unit (MPU)
‡ Data Watchpoint and Trace (DWT)
‡ Serial-Wire or JTAG Debug Port (SW-DP or SWJ-DP)
‡ AHB Access Port (AHB-AP)
Bus Matrix
‡ Instrumentation Trace Macrocell (ITM)
‡ Trace Port Interface Unit (TPIU)
Serial-Wire or JTAG Debug Interface
ICode AHB-Lite instruction interface
DCode AHB-Lite data interface
System AHB-Lite system interface
‡ CoreSight ROM table
PPB APB debug system interface
Trace Port Interface
‡ Optional component

Figure 1. Cortex-M4F Block Diagram

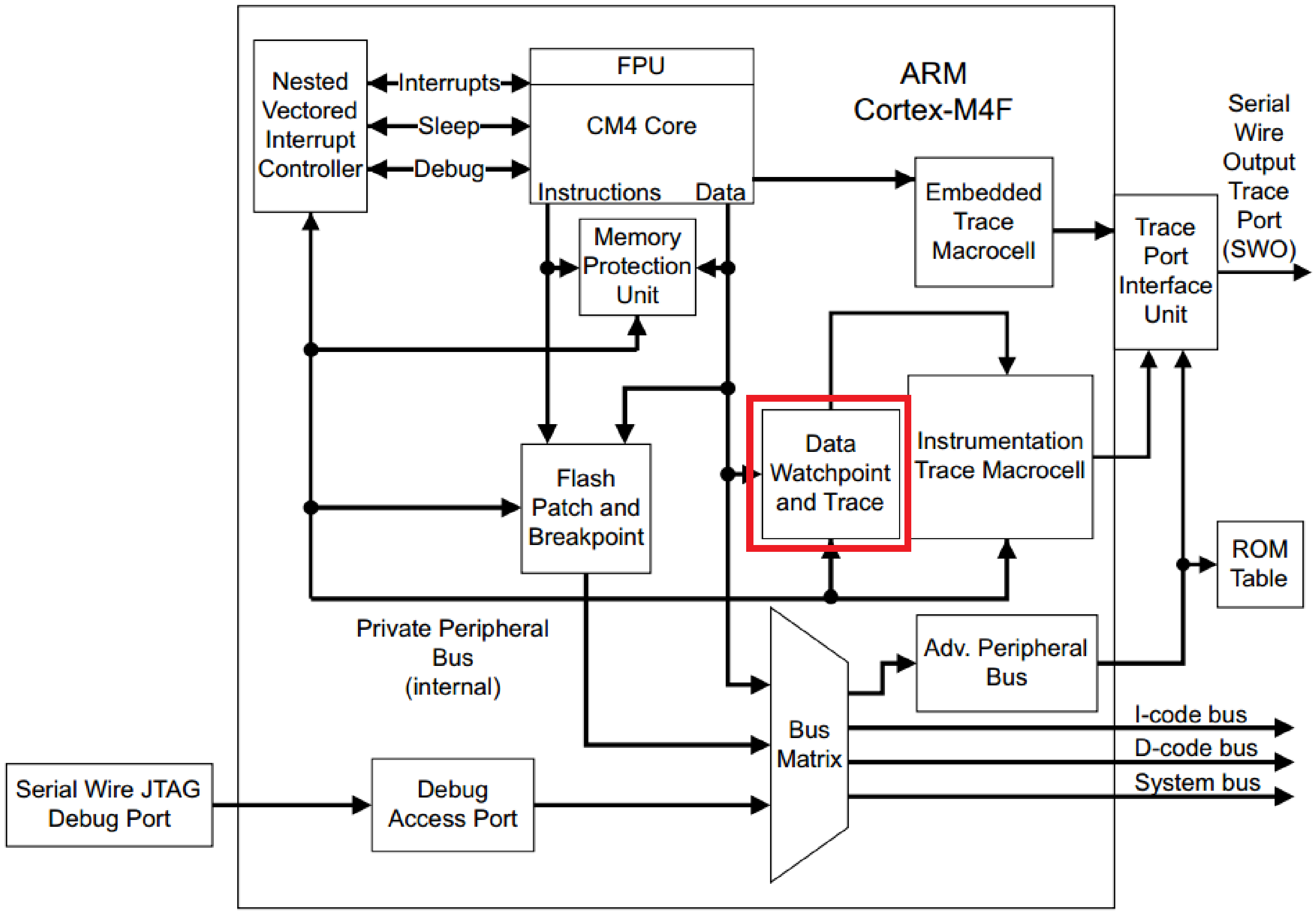
Three particular components can be used for debugging purposes:

- Embedded Trace Macrocell (ETM)
- Instrumentation Trace Macrocell (ITM)
- Data Watchpoint and Trace Unit (DWT)

Each of these three components can do a variety of things. This application note describes the features that are in the MSP432 Cortex-M4F processor.

1.3 MSP432-Specific Components

The MSP432 specifically contains the ARM Cortex-M4F processor with the DSP extension instruction set and Floating Point Unit functionality. Additionally, it includes many core modules including Memory Protection Unit (MPU), Nested Vector Interrupt Controller (NVIC), and SysTick (see Figure 2).



Nested Vectored Interrupt Controller
Interrupts
Sleep
Debug
FPU
CM4 Core
Instructions
Data
ARM Cortex-M4F
Memory Protection Unit
Embedded Trace Macrocell
Serial Wire Output Trace Port (SWO)
Trace Port Interface Unit
Data Watchpoint and Trace
Instrumentation Trace Macrocell
Flash Patch and Breakpoint
ROM Table
Private Peripheral Bus (internal)
Adv. Peripheral Bus
Bus Matrix
I-code bus
D-code bus
System bus
Serial Wire JTAG Debug Port
Debug Access Port

Figure 2. MSP432 Block Diagram

Focusing on the debugging modules, the MSP432 features the ITM and DWT (ETM is not present), both of which are configured through the Trace Port Interface Unit (TPIU) and output through the Serial Wire Output (SWO) pin. The CCS Hardware Trace Analyzer tools use a Serial Wire Viewer (SWV) to collect the trace data on the SWO, and thus the feature is referred to as SWO Trace.

The DWT provides the watchpoints to trace data and profile the system. On the MSP432, the DWT contains three configurable comparators: a hardware watchpoint, a PC sampler event trigger, and a data address sampler event trigger. The DWT also consists of the following counters: clock cycles (CYCCNT), folded instructions, Load Store Unit (LSU) operations, sleep cycles, all instructions except the first cycle (CPI), and interrupt overhead.

The ITM generates packets of trace information, which are output through the SWO. Software is one source that the ITM uses to generate packets; packets are generated when software directly writes to the ITM ports. This is discussed further in the case study. ITM can also generate packets from hardware, the DWT. In this case, the DWT counters and triggers produce the information, and the ITM sends it to the SWO. ITM can also generate time stamps.

CCS implementations of these hardware features are discussed in Section 3 and Section 4.

2 Comparing IDE Debugging Tools

Different IDEs offer individual implementations of the ITM and DWT functions. Table 1 compares the methods for accessing ITM and DWT in the ARM® Keil® µVision® debugger, IAR Embedded Workbench® IDE, and TI CCS IDE.

Table 1. IDE Debugging Tools Comparison

Feature	Keil µVision	IAR Embedded Workbench	TI CCS
Required Hardware	Keil ULINK2™, Keil ULINKpro™, or SEGGER J-Link	SEGGER J-Link, SEGGER J-Trace, or IAR i-Jet®	TI XDS110 (LaunchPad™) or TI XDS200
SWO Trace Use Cases	• Code Coverage • Performance Analyzer • Execution Profiler • Logic Analyzer	• PC sampling • Interrupt logs • Data log events • ITM Stimulus Ports	• Statistical Function Profiling • Data Variable Tracing • Interrupt Profiling • Custom Core Trace

All three IDEs offer similar tool sets. The customizability of the CCS use cases makes it unique. More information on using SWO Trace tools in Keil µVision and IAR Embedded Workbench can be found in Reference 2, Reference 3, and Reference 4.

3 Introduction to CCS Use Cases

The way CCS exposes the ITM and DWT functionalities is through three preconfigured use cases, plus one fully customizable one. They are named Statistical Function Profiling, Data Variable Tracing, Interrupt Profiling, and Custom Core Trace. These four use cases together are present in the *Hardware Trace Analyzer* menu in CCS. This section explains how the four use cases work, and Section 4 demonstrates the use cases and guides users through using the tools.

3.1 Statistical Function Profiling

Statistical Function Profiling displays how often each function is called and what percentage of total program CPU cycles each function consumed. The ITM is used to sample the program counter periodically (adjustable by the user), and the debugger checks the program counter to determine which function is being executed.

3.2 Data Variable Tracing

Data Trace tracks the value at a user-specified address. This address can be a specific memory location or a pointer to a variable. Note that if a variable local to a particular function is traced, no memory address is allocated for the variable until it has been initialized in the code, so the user must step past the initialization of the variable in order for the system to know what the address of the variable is. Then, when the program is run, ITM tracks any reads and writes to the address space. Thus, the result is a graph showing the value of the variable over time.

3.3 Interrupt Profiling

The Interrupt Analyzer shows when interrupts occurred during program runtime, and which interrupts preempted each other. Every time an interrupt is received or completed, ITM logs the number and time. The Interrupt Analyzer can then visualize the interrupt processes and calculate minimum, maximum, and average times per interrupt.

3.4 Custom Core Trace

The most basic setup of Custom Core Trace lets the user see what data has been passed to the ITM ports through software. Every time strings or binary pass through one of the ITM channels, ITM logs the time and message, which is then output through SWO. The ITM library, which is discussed in the case study in Section 4, passes data to the ITM ports. The ITM library allows the user to use predefined functions to easily transmit formatted data (number, string) though one of the ITM ports back to the PC through SWO.

Custom Core Trace is essentially the base of the other three use cases; it does not have any triggers set up, so it does not look at any functions, data addresses, or interrupts. Through the CCS interface, users can fully customize Custom Core Trace to incorporate one or more triggers to analyze many things. This customization is discussed in Section 4.8.1.

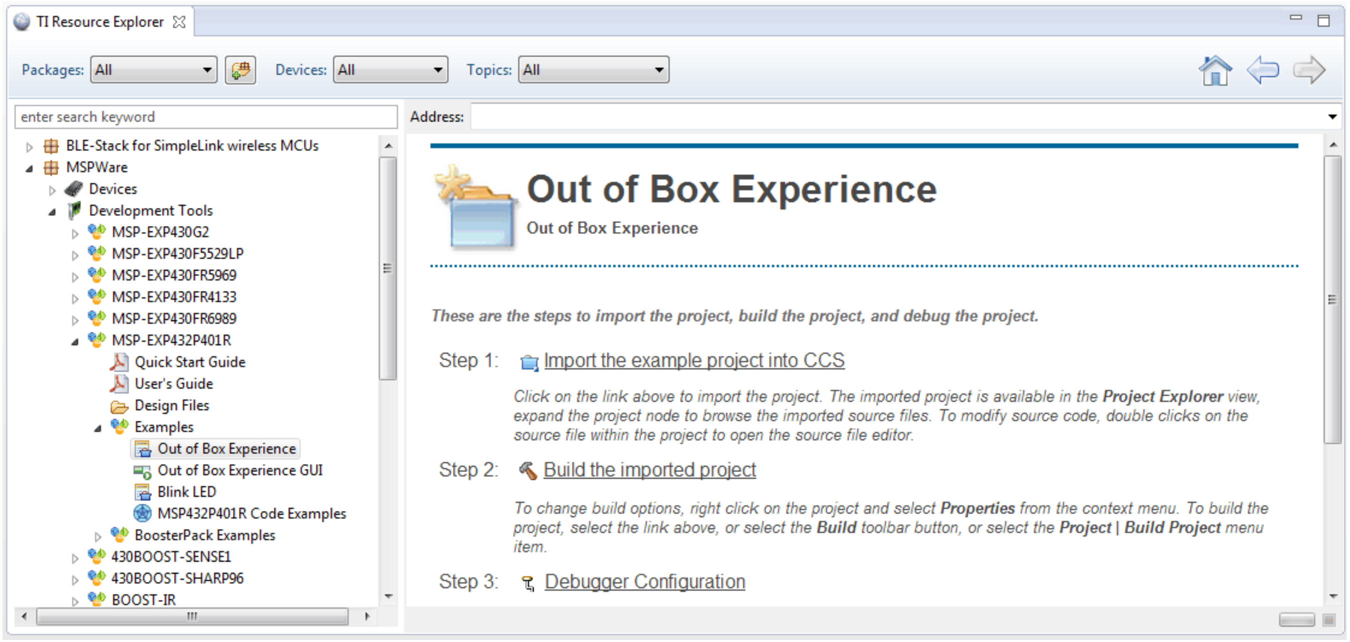
4 Case Study

This section explains how to use the different use cases available in the CCS SWO Trace tool, also referred to as the Hardware Trace Analyzer. The MSP432P401R LaunchPad development kit (MSP-EXP432P401R) is required for this case study. Only one file is needed for this demo: the Out of Box Experience project, available on the TI Resource Explorer in CCS. This project starts by blinking the red LED; the rate of the blinking matches the rate at which the user toggles switch one (SW1), while the color cycles through red, green, blue, and white when the user toggles switch two (SW2).

The case study begins by looking at the general steps to set up any use case of Hardware Trace Analyzer, then the specific configuration possibilities for each use case.

4.1 Import the Out of Box Experience

1. Install MSPWare in CCS version 6.1.0 or later. If you already have this installed, go on to the next step.
   (a) Click *View → CCS App Center*.
   (b) Under *Code Composer Studio Add-ons* find *MSPWare* and click the *Select* box.
   (c) Select *Install Software* and accept the license agreement to begin installation.
   (d) Restart CCS if prompted, to complete the installation process.
2. Open TI Resource Explorer by clicking *View → Resource Explorer (Examples)*.
3. Find the Out of Box Experience under the MSP-EXP432P401R Development Tools Examples, as shown in Figure 3.


TI Resource Explorer

Packages: All | Devices: All | Topics: All

enter search keyword

- BLE-Stack for SimpleLink wireless MCUs
- MSPWare
  - Devices
  - Development Tools
    - MSP-EXP430G2
    - MSP-EXP430F5529LP
    - MSP-EXP430FR5969
    - MSP-EXP430FR4133
    - MSP-EXP430FR6989
    - MSP-EXP432P401R
      - Quick Start Guide
      - User's Guide
      - Design Files
      - Examples
        - Out of Box Experience
        - Out of Box Experience GUI
        - Blink LED
        - MSP432P401R Code Examples
      - BoosterPack Examples
    - 430BOOST-SENSE1
    - 430BOOST-SHARP96
    - BOOST-IR

Address:

Out of Box Experience

Out of Box Experience

These are the steps to import the project, build the project, and debug the project.

Step 1: Import the example project into CCS

*Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.*

Step 2: Build the imported project

*To change build options, right click on the project and select **Properties** from the context menu. To build the project, select the link above, or select the **Build** toolbar button, or select the **Project | Build Project** menu item.*

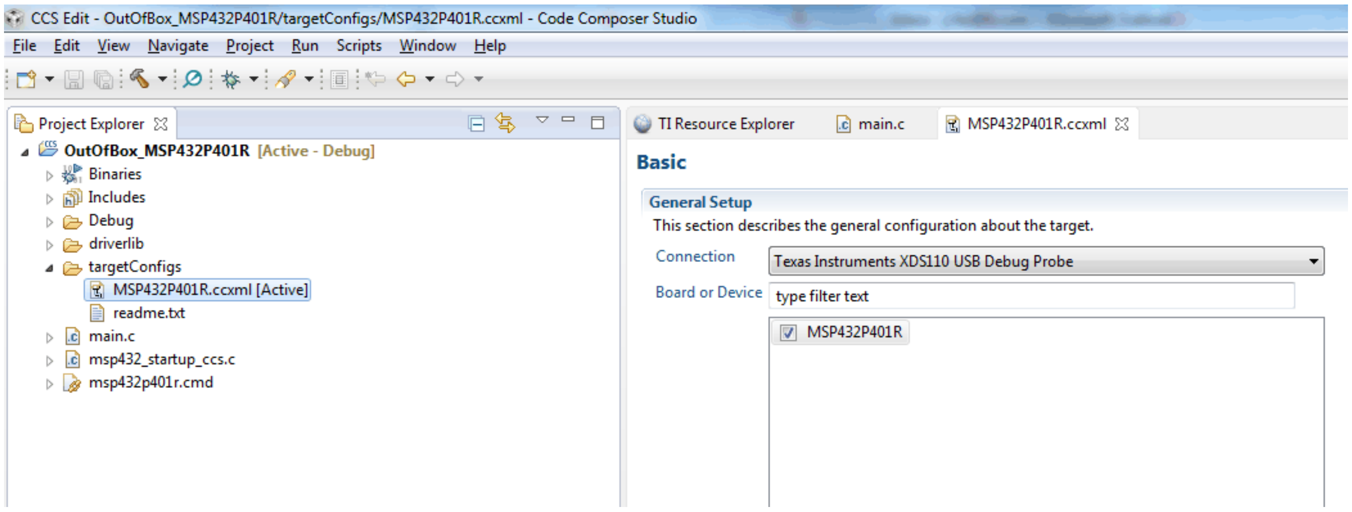
Step 3: Debugger Configuration

Figure 3. Out of Box Experience in MSPWare

4. Click *Import the example project into CCS*, listed as *Step 1*.

4.2 *Configure the Project for SWO Trace*

1. Expand the project in *Project Explorer*, and open *MSP432P401R.ccxml* in the *targetConfigs* folder (see Figure 4).


CCS Edit - OutOfBox_MSP432P401R/targetConfigs/MSP432P401R.ccxml - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer

- OutOfBox_MSP432P401R [Active - Debug]
  - Binaries
  - Includes
  - Debug
  - driverlib
  - targetConfigs
    - MSP432P401R.ccxml [Active]
    - readme.txt
  - main.c
  - msp432_startup_ccs.c
  - msp432p401r.cmd

TI Resource Explorer | main.c | MSP432P401R.ccxml

Basic

General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS110 USB Debug Probe

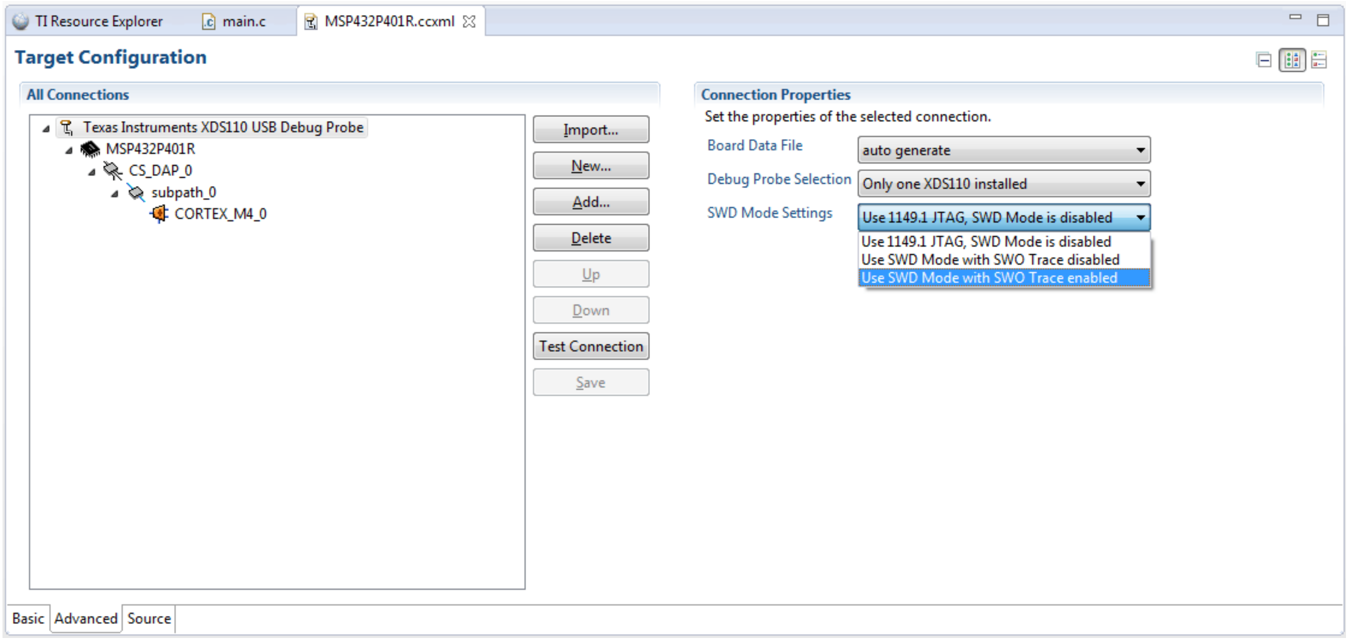
Board or Device: type filter text

☑ MSP432P401R

Figure 4. Target Configuration File for MSP432P401R

2. Enable SWD mode with SWO Trace enabled.
   (a) Check to make sure the correct connection is displayed. *Texas Instruments XDS110 USB Debug Probe* should be selected, and *MSP432P401R* should be checked.
   (b) Click the *Advanced* tab at the bottom of the Target Configuration window.
   (c) Select *Texas Instruments XDS110 USB Debug Probe*.
   (d) Under *Connection Properties → SWD Mode Settings* choose *Use SWD Mode with SWO Trace enabled*.

(e) Press the *Save* button under the column of buttons, including *Import…*, *New…*, and so on



Connection Properties

Set the properties of the selected connection.

Board Data File	auto generate
Debug Probe Selection	Only one XDS110 installed
SWD Mode Settings	Use 1149.1 JTAG, SWD Mode is disabled

Use 1149.1 JTAG, SWD Mode is disabled
Use SWD Mode with SWO Trace disabled
Use SWD Mode with SWO Trace enabled

Figure 5. Changing SWD Mode Settings

4.3 Build the Project

1. Right click the project name, and click *Build Project* in the popup menu. Alternatively, click the hammer icon.

4.4 Enter Debug Mode

1. Enter a debug session. Right click the project name, select *Debug As* → *Code Composer Studio Debug Session*. Alternatively, click the bug icon.
2. In debug mode, go to *Tools* → *Hardware Trace Analyzer*, and choose the use case to use. Note that only one SWO Trace use case can be open at a time.

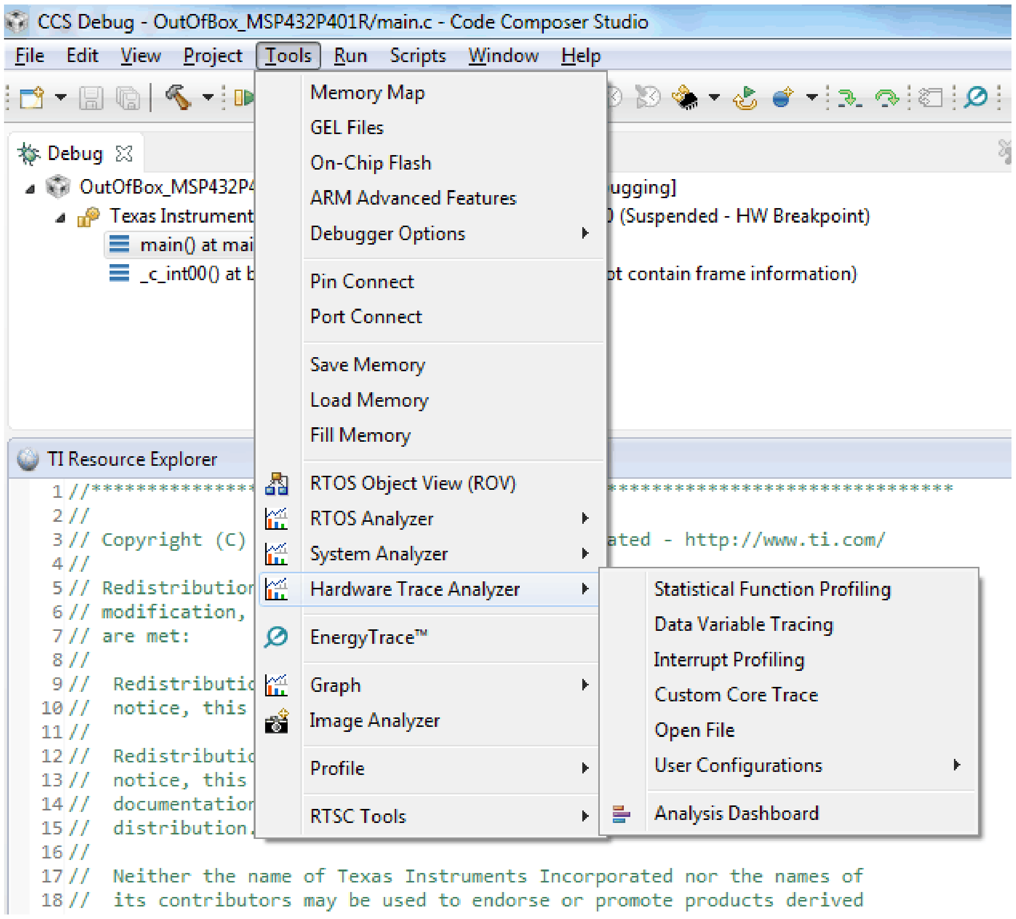
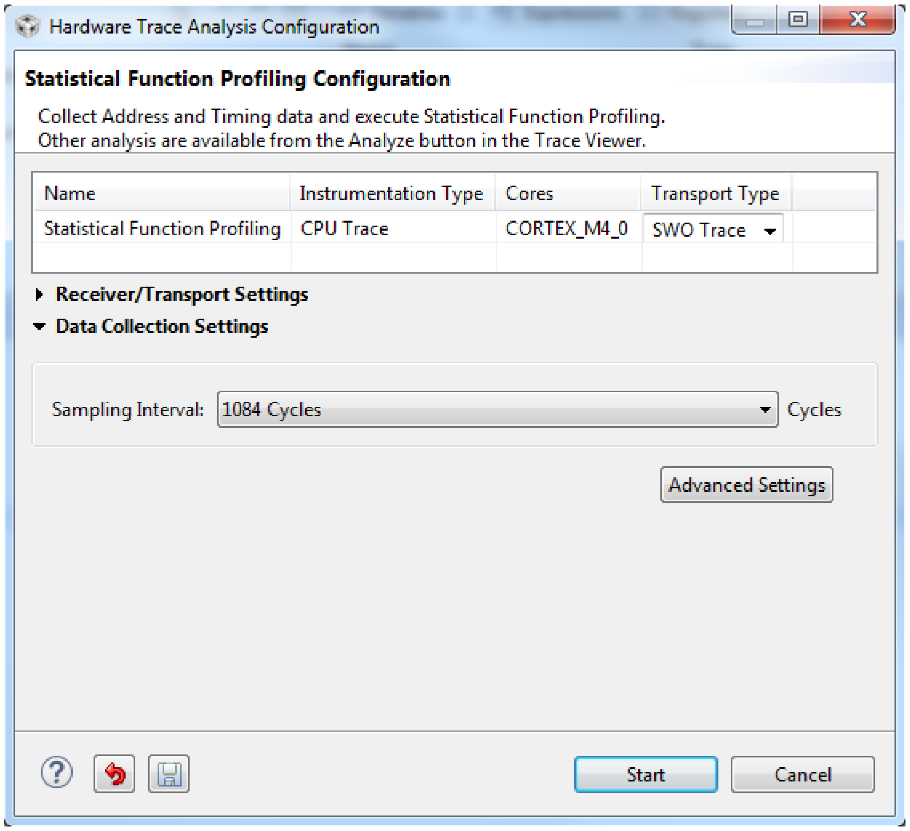


Figure 6. Choosing a Hardware Trace Analyzer Use Case

4.5 Statistical Function Profiling

4.5.1 Setting Up Statistical Function Profiling

1. Under *Tools → Hardware Trace Analyzer*, click *Statistical Function Profiling*. If another use case is open, a popup prompt you to close the current use case.
2. Use the settings in the Statistical Function Profiling Configuration window to configure the use case (see Figure 7). For statistical function profiling, choose how often the ITM samples the PC. The more often the PC (Program Counter) is sampled, the more precise the list of functions. More samples also take more time. The fastest rate that the ITM can sample is one sample per 64 clock cycles, which is a hardware limitation. For the demo, leave the default setup and select *Start*.


Hardware Trace Analysis Configuration
Receiver/Transport Settings
Data Collection Settings
Advanced Settings
Start
Cancel

Statistical Function Profiling Configuration

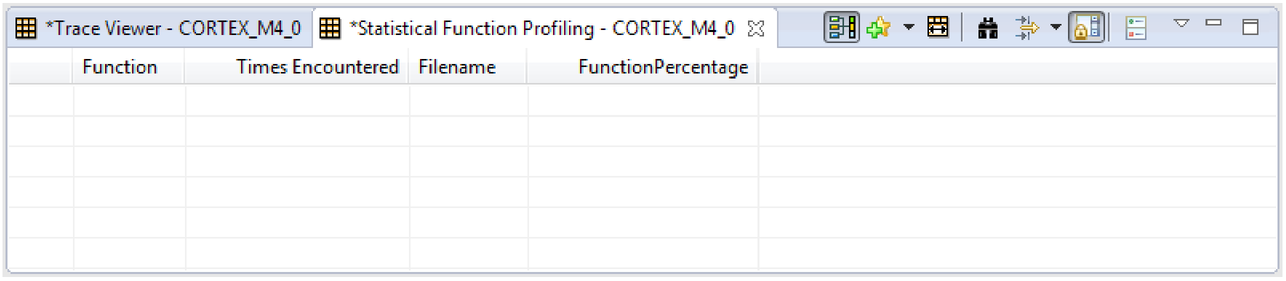
Collect Address and Timing data and execute Statistical Function Profiling.
Other analysis are available from the Analyze button in the Trace Viewer.

Name	Instrumentation Type	Cores	Transport Type
Statistical Function Profiling	CPU Trace	CORTEX_M4_0	SWO Trace

Sampling Interval: 1084 Cycles Cycles

Figure 7. Statistical Function Profiling Setup

3. A new window appears in the bottom right corner of the debug view, as shown in Figure 8. The tab on the left traces the SWO trace output. The right tab displays the functions profiled. You are now ready to run the use case.

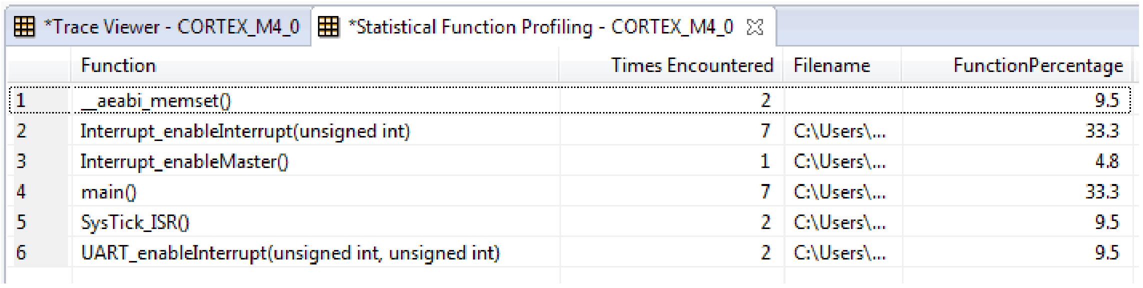

*Trace Viewer - CORTEX_M4_0
*Statistical Function Profiling - CORTEX_M4_0

Function	Times Encountered	Filename	FunctionPercentage

Figure 8. Statistical Function Profiling Window

4.5.2 Using Statistical Function Profiling

1. Click *Run* → *Resume* to start running the program. Alternatively, press F8.
2. Either press *Suspend* (Alt + F8) when you want to pause the program and see the results, or set a breakpoint where you want it to stop.
   (a) To demonstrate, start the program with no breakpoints, let the red LED blink 10 times without touching any switches, and suspend the program. Compare your Statistical Function Profiling window with Figure 9.


*Trace Viewer - CORTEX_M4_0
*Statistical Function Profiling - CORTEX_M4_0

	Function	Times Encountered	Filename	FunctionPercentage
1	__aeabi_memset()	2		9.5
2	Interrupt_enableInterrupt(unsigned int)	7	C:\Users\...	33.3
3	Interrupt_enableMaster()	1	C:\Users\...	4.8
4	main()	7	C:\Users\...	33.3
5	SysTick_ISR()	2	C:\Users\...	9.5
6	UART_enableInterrupt(unsigned int, unsigned int)	2	C:\Users\...	9.5

Figure 9. Statistical Function Profiling Demo

3. The options in the Statistical Function Profiling window next to the tabs give you the ability to sort, filter, and find specific functions.

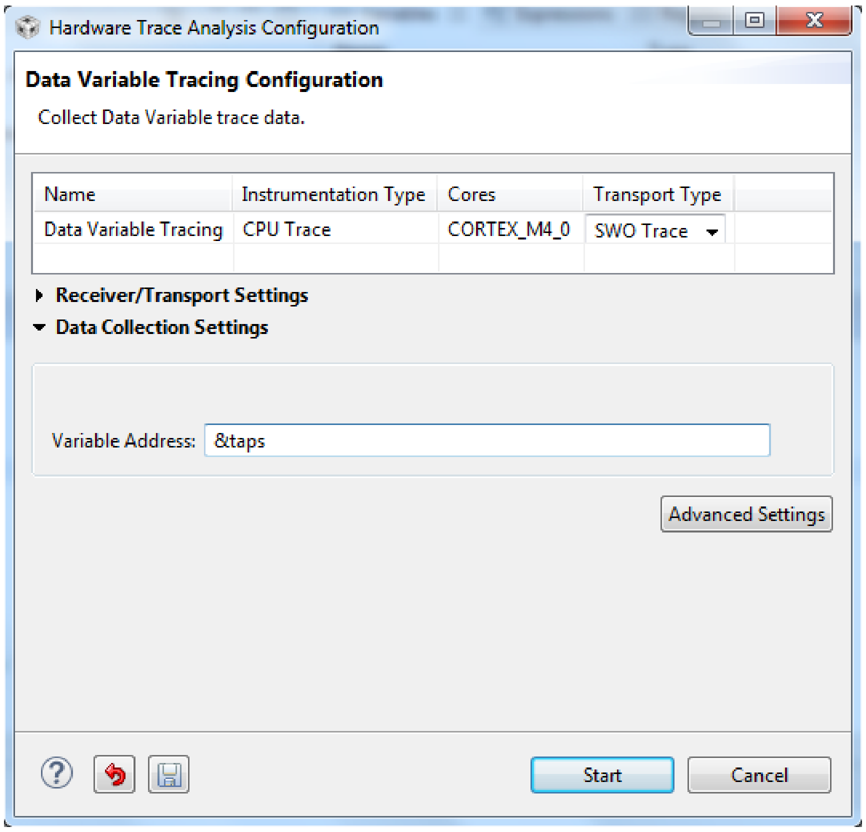
4.6 Data Variable Tracing

4.6.1 Setting Up Data Variable Tracing

1. Under *Tools* → *Hardware Trace Analyzer*, click *Data Variable Tracing*. If a different use case is open, a popup prompts you to close the current use case.
2. The pop-up window lets you configure the use case. For the demo, the only thing you must do is choose a location to trace.

   For the location, you can select an exact memory address in hex, or use a pointer to a variable. Use a global variable, or step past the initialization of the variable so the debugger knows the address of the variable.

   For the demo, type "&taps" to point to the global variable "taps." This variable counts the number of times SW1 is toggled.


Hardware Trace Analysis Configuration

Data Variable Tracing Configuration

Collect Data Variable trace data.

Name	Instrumentation Type	Cores	Transport Type
Data Variable Tracing	CPU Trace	CORTEX_M4_0	SWO Trace

Receiver/Transport Settings

Data Collection Settings

Variable Address: &taps

Figure 10. Data Variable Tracing Setup

3. Press *Start* to continue.
4. A new window appears in the bottom right corner of the debug view, as shown in Figure 11. The tab on the left traces the SWO trace output. The tab on the right is a graph of the value of the specified variable. You are now ready to run the use case.

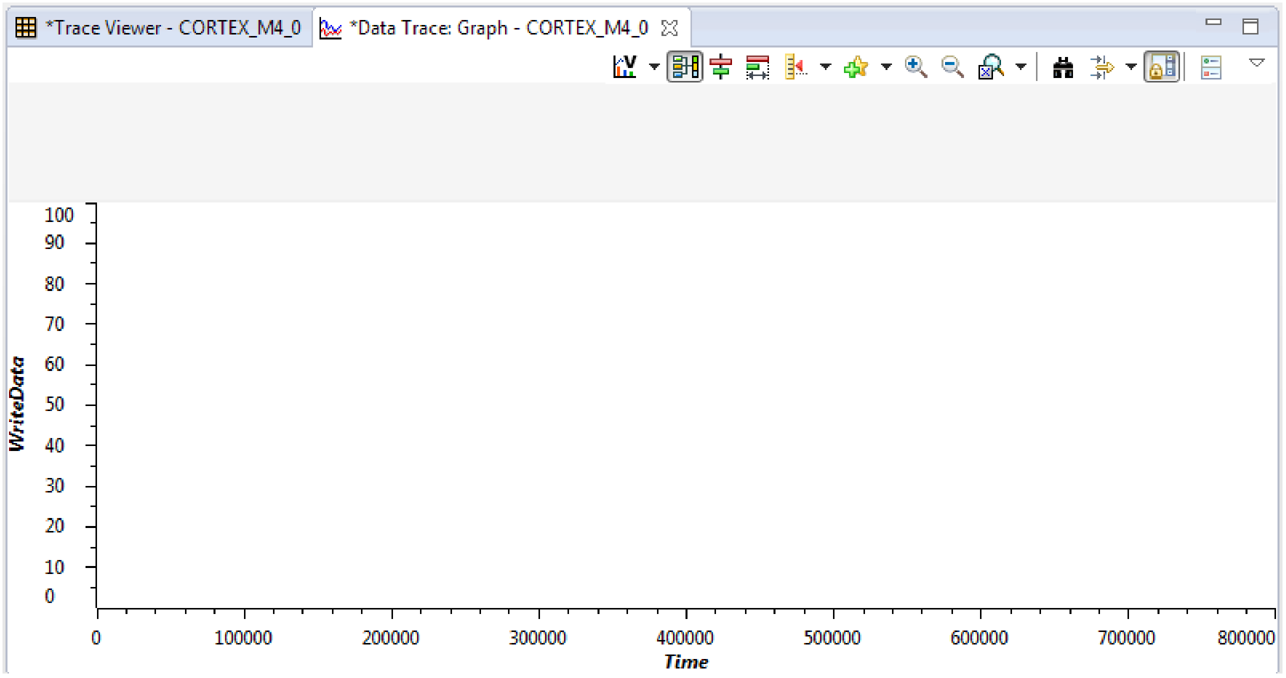


Figure 11. Data Variable Tracing Window

4.6.2 Using Data Variable Tracing

1. Click *Run* → *Resume* to start running the program. Alternatively, press F8.
2. Either press *Suspend* (Alt + F8) when you want to pause the program and see the results, or set a breakpoint where you want it to stop.
   (a) To demonstrate, start the program with no breakpoints. Push the SW1 however many times you would like, keeping track of how many times you have pushed it. In Figure 12, SW1 was pushed 8 times, each time when the LED blinked, so the time between each press was approximately the same each time.

   Zoom out as many times as necessary to see the full window.

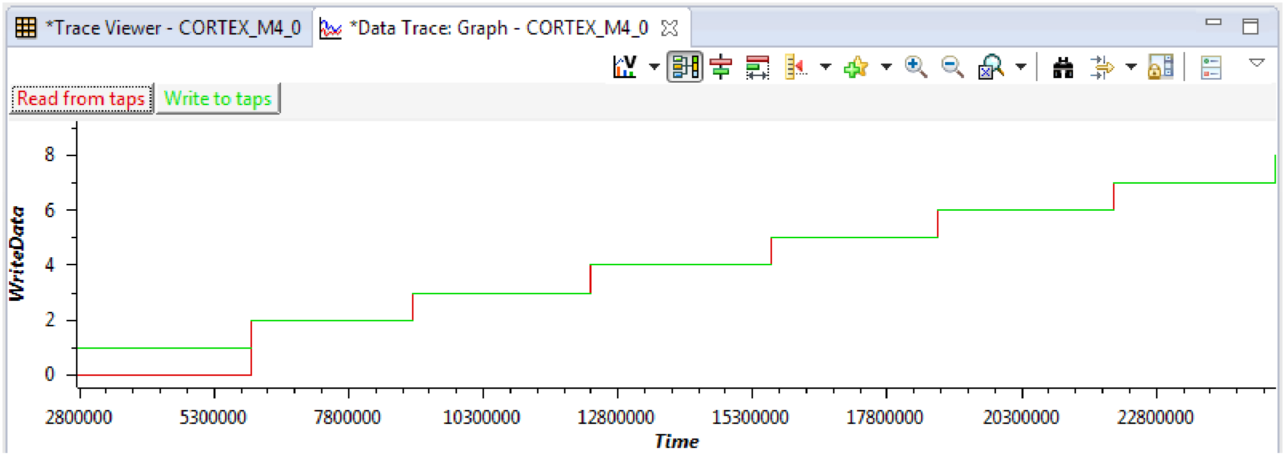


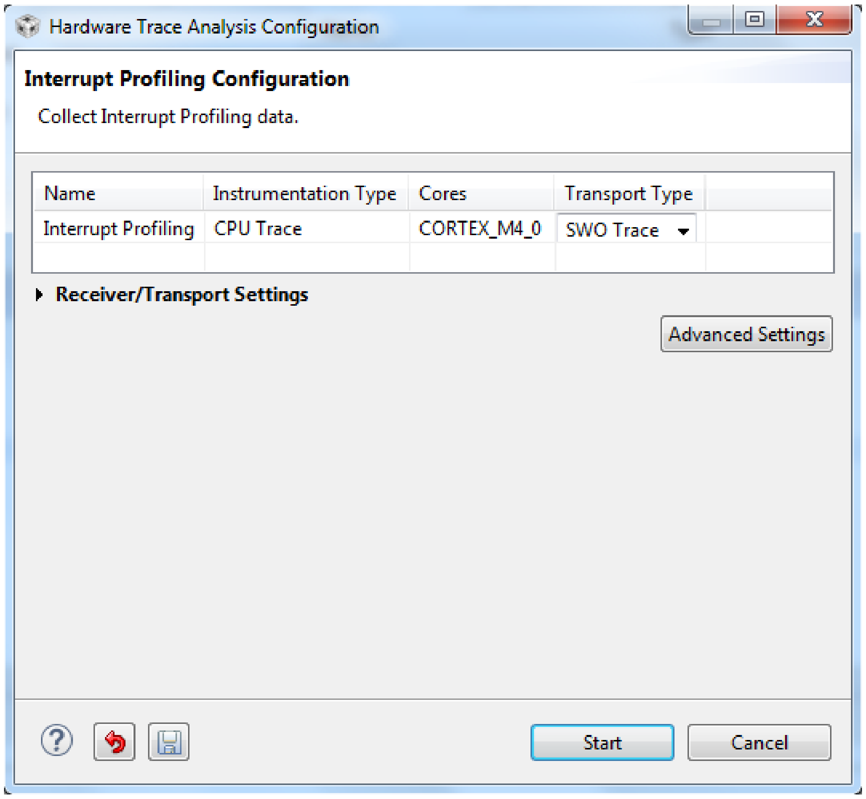
Figure 12. Data Variable Tracing Demo

3. The options in the Data Trace Graph window give you the ability to focus on reads and writes to the variable, look at specific points in time, and other functions.

4.7 Interrupt Profiling

4.7.1 Setting Up Interrupt Profiling

1. Under *Tools* → *Hardware Trace Analyzer*, click *Interrupt Profiling*. If a different use case is open, a popup prompts you to close the current use case.
2. There is no configuration necessary for Interrupt Profiling for normal use and or when viewing the demo.



Hardware Trace Analysis Configuration

Interrupt Profiling Configuration

Collect Interrupt Profiling data.

Name	Instrumentation Type	Cores	Transport Type
Interrupt Profiling	CPU Trace	CORTEX_M4_0	SWO Trace

Receiver/Transport Settings

Figure 13. Interrupt Profiling Setup

3. Press *Start* to continue.
4. A new window appears in the bottom right corner of the debug view, as shown in Figure 14. The tab on the left traces the SWO trace output. The middle tab holds an interrupt graph; the right tab contains a detailed summary about the interrupts. You are now ready to run the use case.

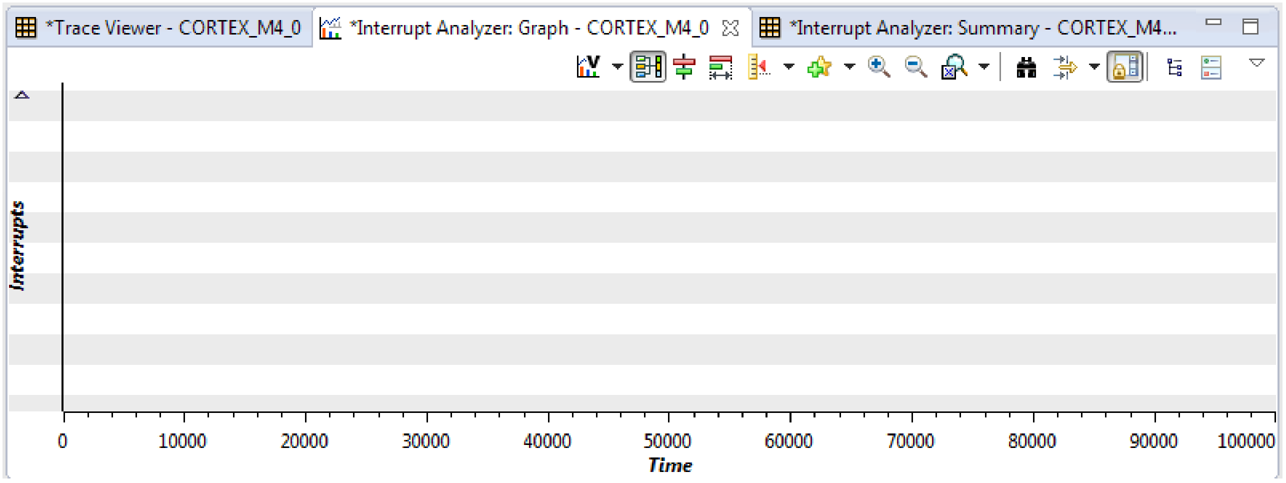

*Trace Viewer - CORTEX_M4_0
*Interrupt Analyzer: Graph - CORTEX_M4_0
*Interrupt Analyzer: Summary - CORTEX_M4...
Interrupts
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Time

Figure 14. Interrupt Profiling Window

4.7.2 Using Interrupt Profiling

1. Click *Run* → *Resume* to start running the program. Alternatively, press F8.
2. Either press *Suspend* (Alt + F8) when you want to pause the program and see the results, or set a breakpoint where you want it to stop.
   (a) To demonstrate, start the program with no breakpoints, let the red LED blink 10 times without touching any switches, and suspend the program. Compare your Interrupt Profiling graph with Figure 15.

   Zoom out as many times as necessary to see the full window.

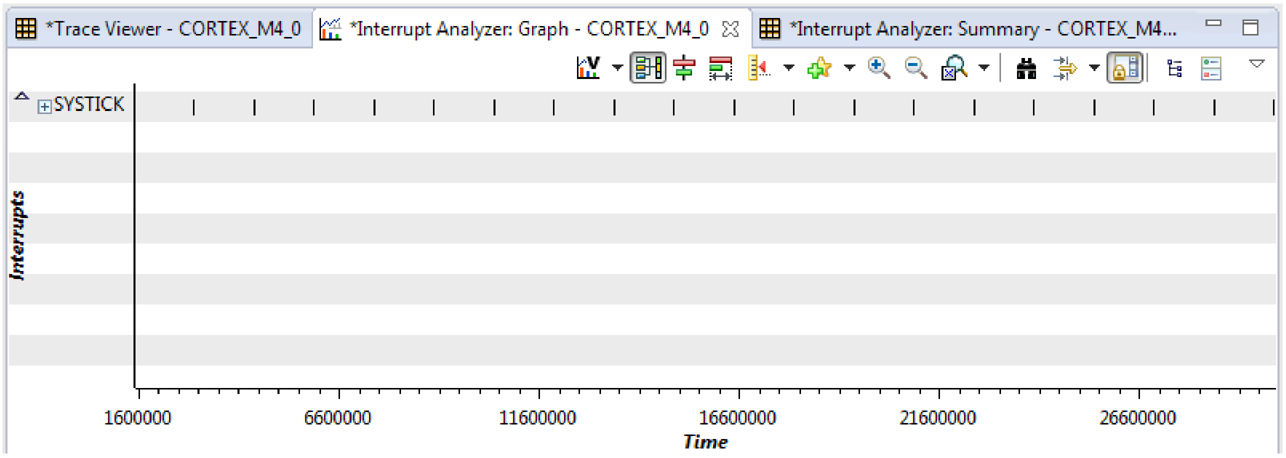

*Trace Viewer - CORTEX_M4_0
*Interrupt Analyzer: Graph - CORTEX_M4_0
*Interrupt Analyzer: Summary - CORTEX_M4...
SYSTICK
Interrupts
1600000
6600000
11600000
16600000
21600000
26600000
Time

Figure 15. Interrupt Profiling Demo

   (b) To see a more elaborate demonstration with interrupts preempting each other, start the program and toggle the switches quickly and randomly. This is an example of what it might look like.

   Zoom out as many times as necessary to see the full window.

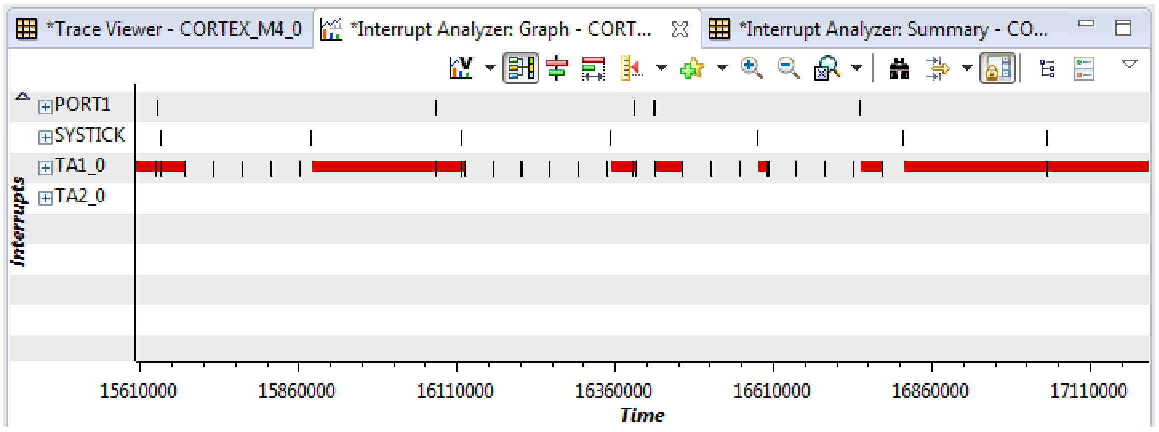

*Trace Viewer - CORTEX_M4_0
*Interrupt Analyzer: Graph - CORT...
*Interrupt Analyzer: Summary - CO...
PORT1
SYSTICK
TA1_0
TA2_0
Interrupts
15610000
15860000
16110000
16360000
16610000
16860000
17110000
Time

Figure 16. Interrupt Profiling Advanced Example

3. The options in the Interrupt Profiling windows allow you to look at specific points in time, see if and when interrupts preempt each other, and look at how often specific interrupts were serviced.

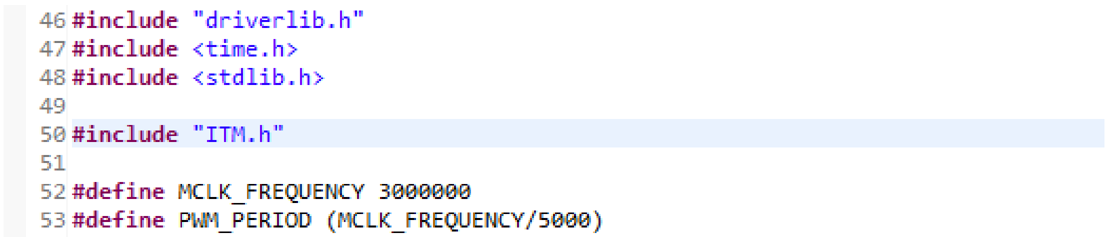
4.8 Custom Core Trace

4.8.1 Setting Up Custom Core Trace

1. Custom Core Trace is unique in that ITM does not use hardware packets to trace. Rather, for the most basic version, users must send software messages to the ITM port manually.

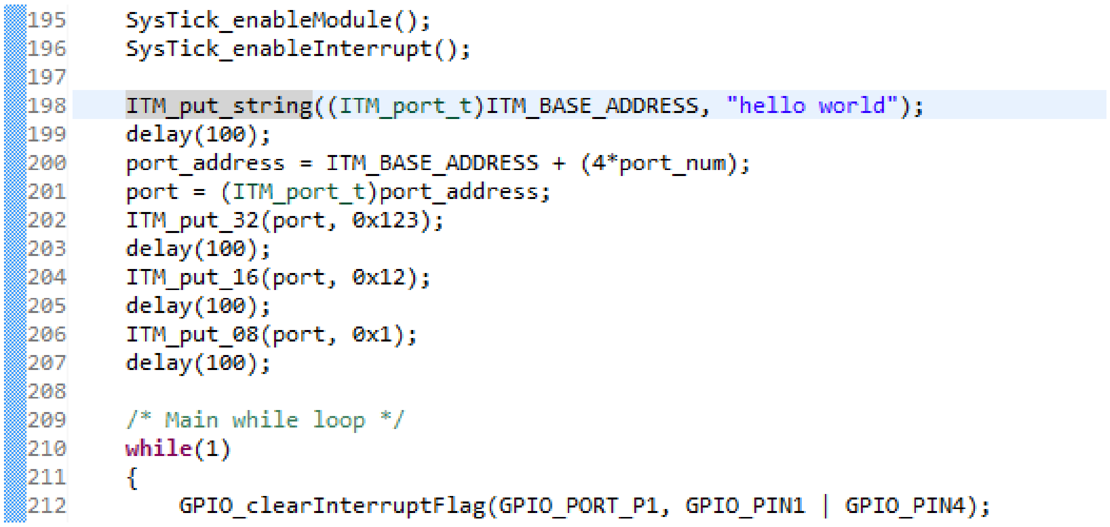
   For this use case, use the same sample project imported in Section 4.2, and insert the custom ITM library and code. Before building the project and entering debug mode, open *main.c*. Insert the following lines of code at the specified locations.

   Line 50:



```
46 #include "driverlib.h"
47 #include <time.h>
48 #include <stdlib.h>
49
50 #include "ITM.h"
51
52 #define MCLK_FREQUENCY 3000000
53 #define PWM_PERIOD (MCLK_FREQUENCY/5000)
```

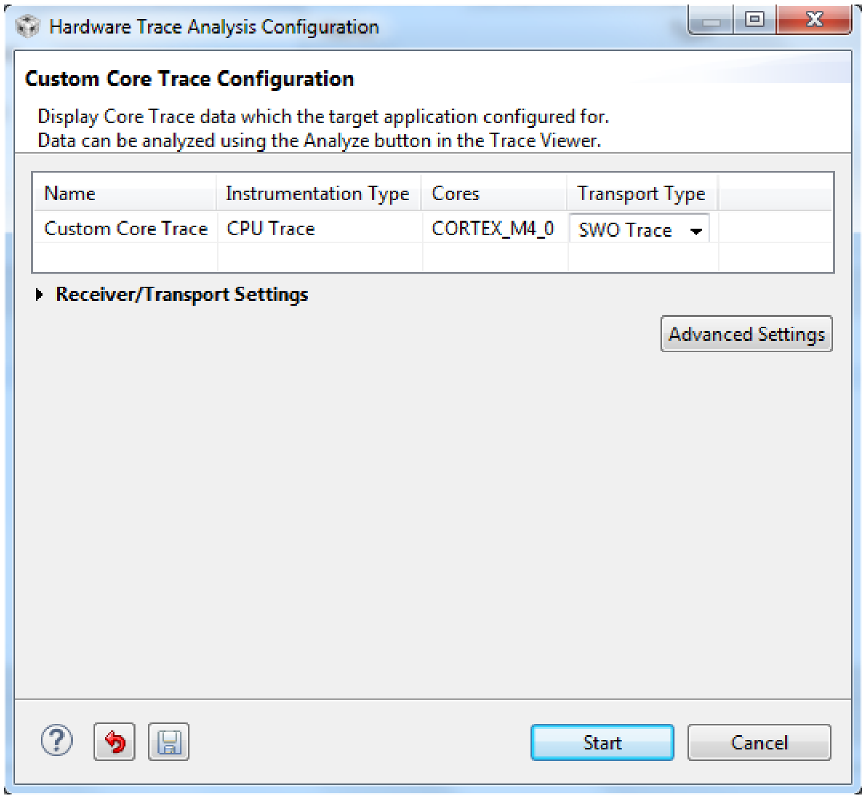
   Lines 198-207:



```
195     SysTick_enableModule();
196     SysTick_enableInterrupt();
197
198     ITM_put_string((ITM_port_t)ITM_BASE_ADDRESS, "hello world");
199     delay(100);
200     port_address = ITM_BASE_ADDRESS + (4*port_num);
201     port = (ITM_port_t)port_address;
202     ITM_put_32(port, 0x123);
203     delay(100);
204     ITM_put_16(port, 0x12);
205     delay(100);
206     ITM_put_08(port, 0x1);
207     delay(100);
208
209     /* Main while loop */
210     while(1)
211     {
212         GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1 | GPIO_PIN4);
```

   The first line includes the ITM library that includes functions to write to the ITM port, which consists of 32 available channels. The next group of lines first sends the string "hello world" to the first ITM channel, followed by the numbers 123, 12, and 1, written to the second ITM channel, with a delay between each.

2. Now, build the project and enter debug mode.
3. Under *Tools → Hardware Trace Analyzer*, click *Custom Core Trace*. If a different use case is open, a popup prompts you to close the current use case.
4. There is no configuration necessary for Custom Core Trace when using the basic features shown in this example.



Hardware Trace Analysis Configuration

Custom Core Trace Configuration

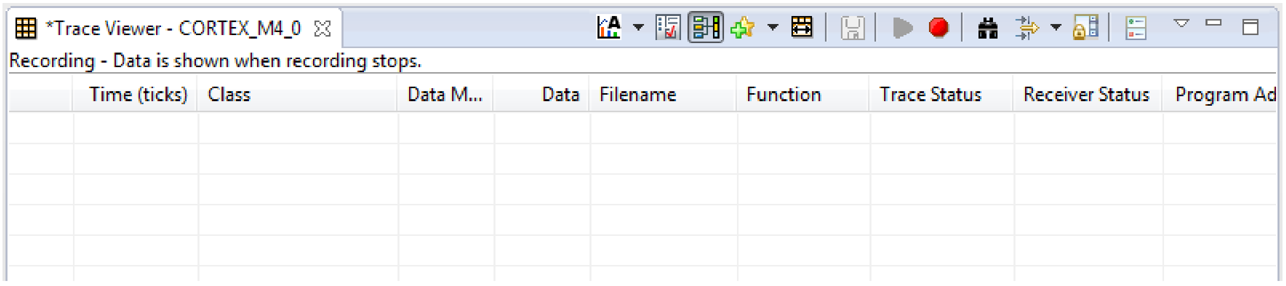
Display Core Trace data which the target application configured for.
Data can be analyzed using the Analyze button in the Trace Viewer.

Name	Instrumentation Type	Cores	Transport Type
Custom Core Trace	CPU Trace	CORTEX_M4_0	SWO Trace

Receiver/Transport Settings

Figure 17. Custom Core Trace Setup

5. Press *Start* to continue.
6. A new window appears in the bottom right corner of the debug view, as shown in Figure 18. You are now ready to run the use case.



*Trace Viewer - CORTEX_M4_0

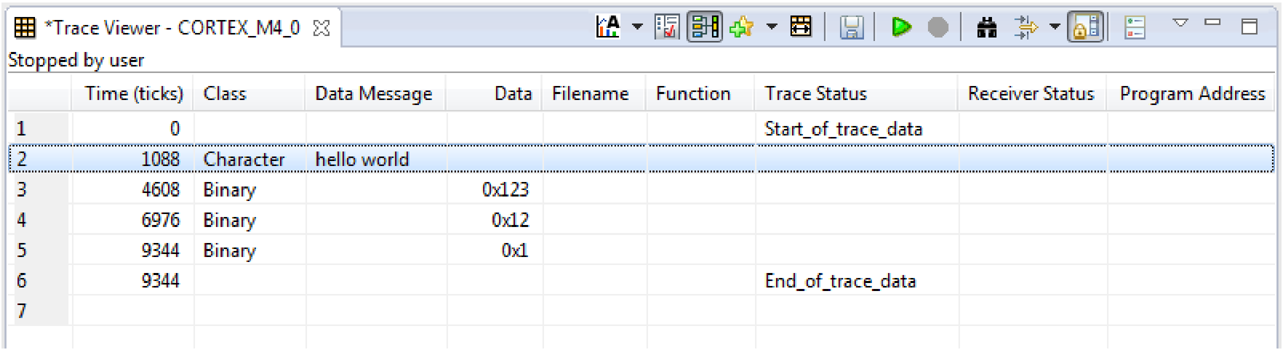
Recording - Data is shown when recording stops.

Time (ticks)	Class	Data M...	Data	Filename	Function	Trace Status	Receiver Status	Program Ad

Figure 18. Custom Core Trace Window

4.8.2 Using Custom Core Trace

1. Click *Run* → *Resume* to start running the program. Alternatively, press F8.
2. Either press *Suspend* (Alt + F8) when you want to pause the program and see the results, or set a breakpoint where you want it to stop.
   (a) To demonstrate, start the program with no breakpoints. Wait a few seconds, suspend the program, and compare your Trace Viewer window with Figure 19.



*Trace Viewer - CORTEX_M4_0

Stopped by user

	Time (ticks)	Class	Data Message	Data	Filename	Function	Trace Status	Receiver Status	Program Address
1	0						Start_of_trace_data		
2	1088	Character	hello world						
3	4608	Binary		0x123					
4	6976	Binary		0x12					
5	9344	Binary		0x1					
6	9344						End_of_trace_data		
7									

Figure 19. Custom Core Trace Demo

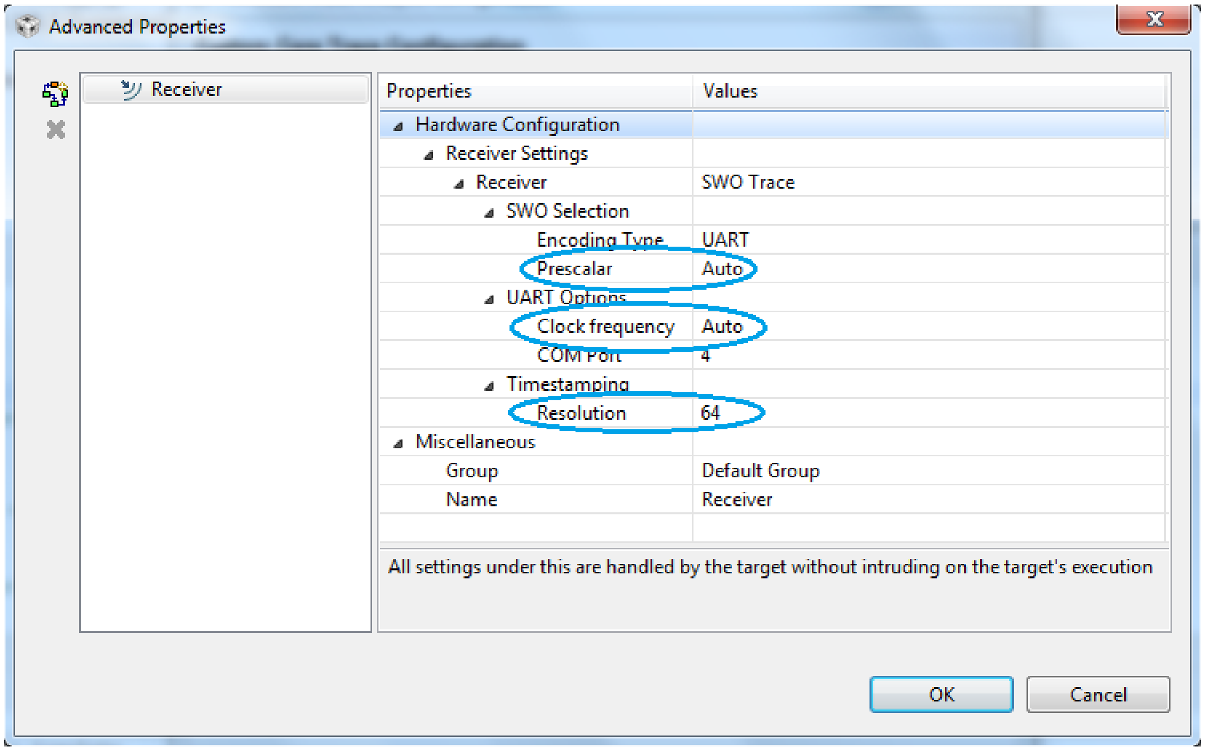
3. The options in the Trace Viewer display various details about the SWO Trace output.

4.8.3 Configuring Trace

The configuration of Custom Core Trace allow the user to fully maximize the functions of the SWO Trace hardware capabilities. To access the customization screen, choose *Custom Core Trace*, click *Advanced Settings*, and a popup like the one in Figure 20 opens.

On this screen, click this button to add triggers, which tell CCS what it should be trying to trace. The triggers use the hardware counters—clock cycle, sleep cycles, and so on—to determine when to react. The next few sections discuss some useful capabilities of triggers.

The UART encoding type listed in this dialog box specifies the encoding type for the serial data transmitted through the SWO, not from the MSP432 device's UART peripheral.



Advanced Properties

Receiver

Properties	Values
Hardware Configuration	
Receiver Settings	
Receiver	SWO Trace
SWO Selection	
Encoding Type	UART
Prescalar	Auto
UART Options	
Clock frequency	Auto
COM Port	4
Timestamping	
Resolution	64
Miscellaneous	
Group	Default Group
Name	Receiver

All settings under this are handled by the target without intruding on the target's execution

Figure 20. Custom Core Trace Default Advanced Settings

4.8.4 Receiver Settings

The configurable settings in Figure 20 are Prescalar, Clock frequency, and Timestamping Resolution. These configure the clock necessary for successful SWO data transfer.

Prescalar – this is the value that the trace clock frequency is divided by, since the trace receiver (debugging probe) cannot process data as fast as the original trace clock frequency.

Clock frequency – this is the system clock frequency, which the trace receiver uses to determine the trace clock frequency.

Timestamping Resolution – the rate at which timestamps should be created.

For MSP432, it is highly recommended to leave them with their default values (auto). If the user decides to input custom values, then either keep the Prescalar at auto or set it to a value such that (system clock / (prescalar + 1)) is a standard baud rate value.

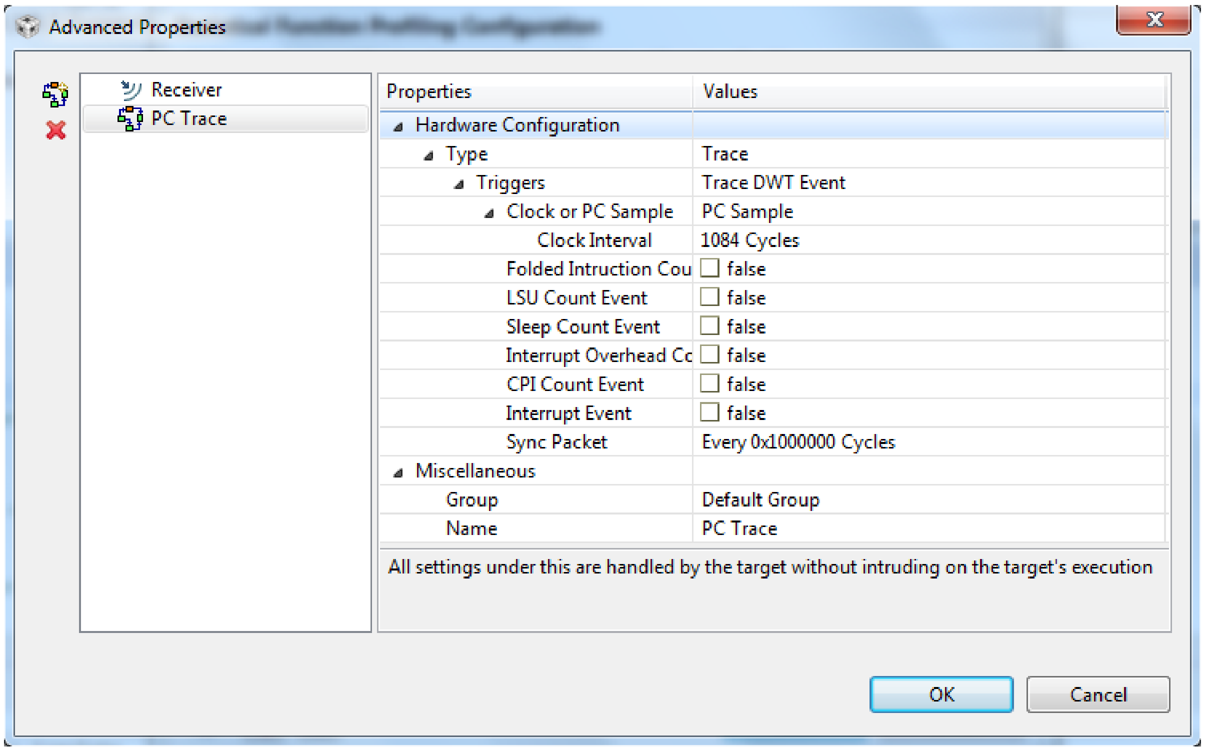
Also, the user should not start any trace use cases until after all system clock configurations. If the code changes the system clock after the code reaches the main function, then step past the clock changes before beginning SWO Trace, or else there will be corrupted data in the trace.

The only time Timestamp Resolution should be altered is if there are many timestamp overflows in the trace data. The messages can be reduced by increasing the Timestamp Resolution.

4.8.5 Use Case Triggers

One of the most useful capabilities of the *Advanced Settings* is the ability to run more than one of the aforementioned use cases at once, or use the Data Variable Tracing use case on more than one variable at a time.

To create a trigger for each of the use cases, reference and replicate the setting shown in Figure 21 through Figure 23.

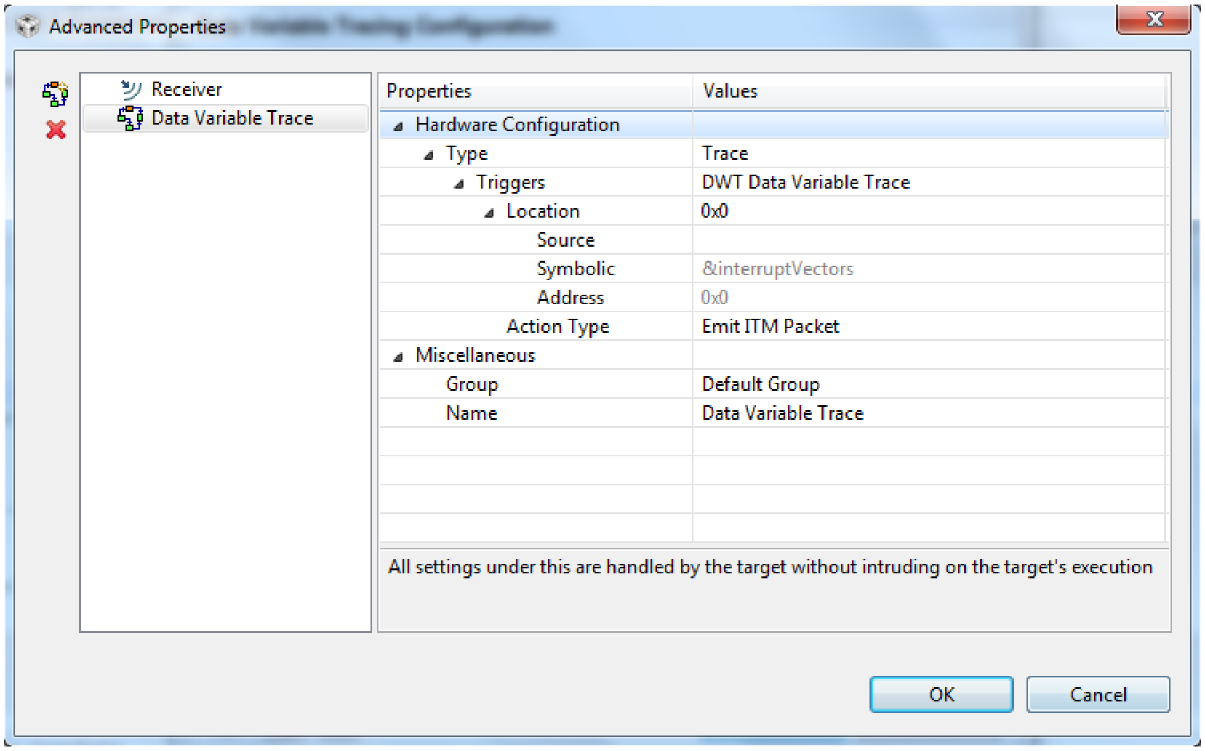


Properties	Values
Hardware Configuration	
Type	Trace
Triggers	Trace DWT Event
Clock or PC Sample	PC Sample
Clock Interval	1084 Cycles
Folded Intruction Cou	false
LSU Count Event	false
Sleep Count Event	false
Interrupt Overhead Cc	false
CPI Count Event	false
Interrupt Event	false
Sync Packet	Every 0x1000000 Cycles
Miscellaneous	
Group	Default Group
Name	PC Trace

All settings under this are handled by the target without intruding on the target's execution

Figure 21. Custom Core Trace PC Trace Trigger

Adjust the *Clock Interval* setting to determine how often the system should sample the PC, similar to the Statistical Function Profiling use case (see Section 4.5).



Advanced Properties

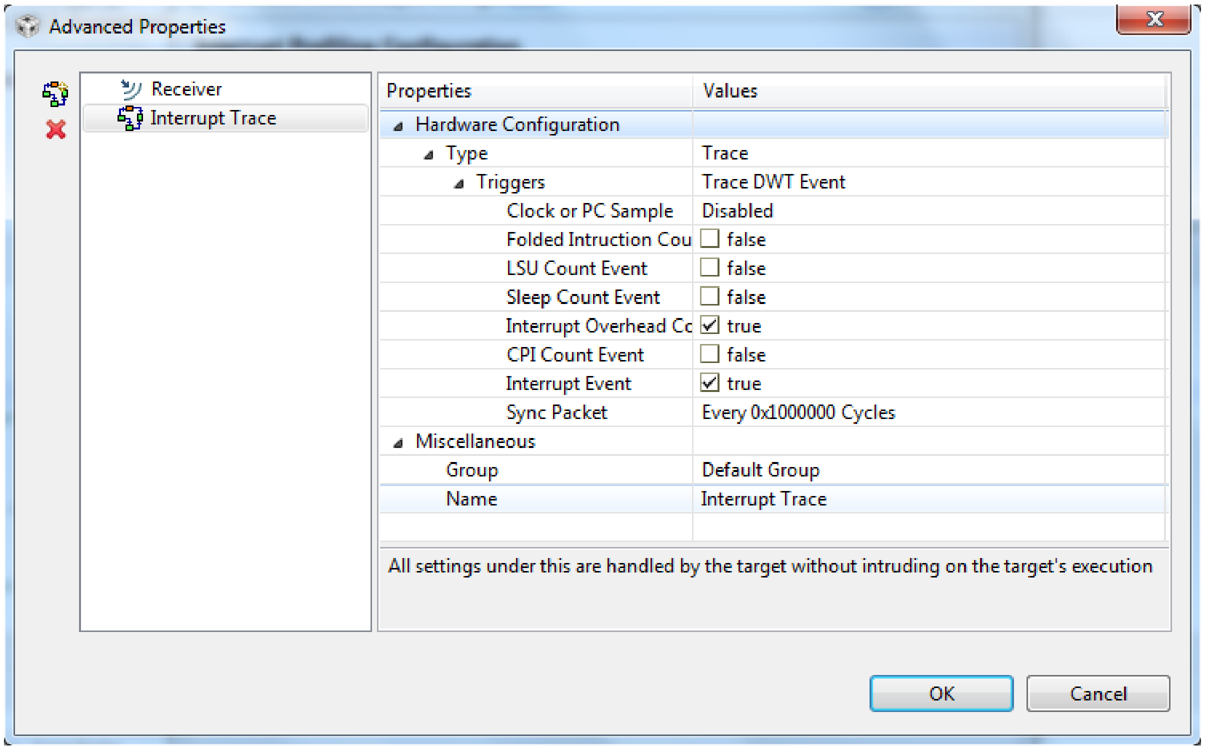
- Receiver
- Data Variable Trace

Properties	Values
Hardware Configuration	
Type	Trace
Triggers	DWT Data Variable Trace
Location	0x0
Source	
Symbolic	&interruptVectors
Address	0x0
Action Type	Emit ITM Packet
Miscellaneous	
Group	Default Group
Name	Data Variable Trace

All settings under this are handled by the target without intruding on the target's execution

Figure 22. Custom Core Trace Data Variable Tracing Trigger

Decide the *Location* the DWT should watch. To trace more than one variable, create multiple triggers with the pictured configuration. The only difference between them should be the location that the DWT watches.



Advanced Properties

- Receiver
- Interrupt Trace

Properties	Values
Hardware Configuration	
Type	Trace
Triggers	Trace DWT Event
Clock or PC Sample	Disabled
Folded Intruction Cou	☐ false
LSU Count Event	☐ false
Sleep Count Event	☐ false
Interrupt Overhead Cc	✓ true
CPI Count Event	☐ false
Interrupt Event	✓ true
Sync Packet	Every 0x1000000 Cycles
Miscellaneous	
Group	Default Group
Name	Interrupt Trace

All settings under this are handled by the target without intruding on the target's execution

Figure 23. Custom Core Trace Interrupt Trace Trigger

There are no configuration options for the Interrupt Trace trigger.

After setting the triggers to use, run the program and then pause it. Only the Trace Viewer is visible initially. To view the other windows, such as the variable value graph and interrupt graph, click this button . The other windows display information only if the corresponding use case has a trigger set (for example, if no Interrupt Trace trigger is set, the interrupt graph is empty).

4.8.5.1 Trace DWT Event Type Configuration

When creating a new Trace trigger (under *Type)*, there are two options for *Triggers*. The first option is Trace DWT Event (the other option is described in Section 4.8.5.2). Under this option, there are many customizable features:

Clock or PC Sample – Depending on the selected option, the ITM samples either the clock or the PC for events. You can input how often the ITM should sample, limited by hardware to a minimum of 64 cycles. The Statistical Function Profiling use case uses this to sample the PC.

Folded Instruction Count Events – Every 256 times a folded instruction occurs, this counter sends an event packet to the ITM. The Out of Box does not have any folded instructions.

LSU Count Event – Every 256 times the load-store unit is operated, this counter sends an event packet to the ITM. The Out of Box does use the LSU, so this event occurs multiple times if the program runs for a few seconds.

Sleep Count Event – Every 256 times the processor is sleeping, this counter sends an event packet to the ITM. The Out of Box does have the processor sleeping at points in time, so this event occurs if the program runs for a few seconds.

Interrupt Overhead Count Event – Every 256 cycles of interrupt overhead. When this is coupled with the Interrupt Event trigger, it is equivalent to the Interrupt Trace trigger (for the Interrupt Profiling use case).

CPI Count Event – Every 256 cycles of multi-cycle instructions, this counter sends an event packet to the ITM. The Out of Box has a lot of multi-cycle instructions, so this counter tends to always overflow.

Interrupt Event – Every entrance into and exit out of an interrupt, this trigger sends an event packet notifying the ITM. This is essential for the Interrupt Trace trigger (for the Interrupt Profiling use case).

4.8.5.2 DWT Data Variable Trace Type Configuration

The other trigger option is the DWT Data Variable Trace option. This is the trigger specifically for tracing the data of a variable location in memory. The only customization is the location or a pointer.

4.8.5.3 Other Types Configuration

Custom Core Trace shares a configuration window with other debugging capabilities, such as breakpoints, watchpoints, and other things. Configuring these capabilities is not discussed in this application note, as they do not pertain to SWO Trace.

4.9 Known Issues

In CCS releases up to CCS v6.1.0, starting any one of the use cases may trigger the error shown in Figure 24.

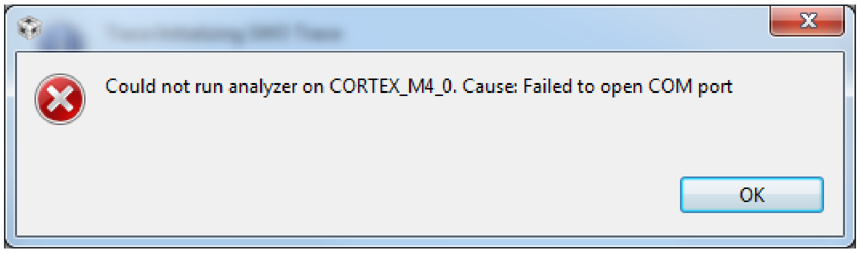

Could not run analyzer on CORTEX_M4_0. Cause: Failed to open COM port

Figure 24. Failed to Open COM Port

The reason for this bug is related to a specific open source library used by CCS, and a fix is planned for CCS v6.1.1.

5 Summary

In summary, the MSP432 ARM Cortex-M4F processor offers advanced hardware debugging functionality. Paired with the implementation of these resources in the TI Code Composer Studio IDE, users can analyze programs at high and low levels. The predefined use cases make implementation of these tools quick and direct, while customizable options cater to more specific needs.

6 References

1. ARM CoreSight Components Technical Reference Manual
   http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/index.html
2. ARM Advanced Analysis Tools
   http://www.keil.com/uvision/db_anl.asp
3. Using the MSP432 LaunchPad: ARM Keil MDK 5 Toolkit
   http://www.keil.com/appnotes/files/apnt_276.pdf
4. IAR Embedded Workbench IDE User Guide for ARM: Using trace
   http://supp.iar.com/FilesPublic/UPDINFO/005405/arm/doc/EWARM_UserGuide_AddOn1.ENU.pdf
5. MSP432P4xx Family Technical Reference Manual (SLAU356)
6. Code Composer Studio 6.1 for MSP432 User's Guide (SLAU575)

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from August 15, 2015 to February 23, 2016	**Page**
• Updated Figure 2, *MSP432 Block Diagram*	4
• Updated description in Section 3.4, *Custom Core Trace*	6
• Updated description in Section 4.8.1, *Setting Up Custom Core Trace*	17
• Added the sentence that starts "The UART encoding type..." in Section 4.8.3, *Configuring Trace*	19
• Updated Figure 20, *Custom Core Trace Default Advanced Settings*	19

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		