Microsemi® SmartFusion®2 Lab: M2S090TS-EVAL-KIT with ARM® Keil™ MDK toolkit

featuring Serial Wire Viewer and ETM Instruction Trace

Spring 2017 Version 1.1

by Robert Boys bob.boys@arm.com

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_292.asp

Introduction:

This note describes the process of operating the ARM® Keil™ MDK toolkit featuring µVision® and Microsemi's (Actel™) SmartFusion2 (SF2) family which contains an embedded ARM® Cortex™-M3 processor. SmartFusion2 implements ARM Serial Wire Viewer (SWV) and Embedded Trace Macrocell (ETM) CoreSight debug technology.

This note describes how to get all the components of this important technology working with µVision. **This article uses MDK® 5.23 or later** and a Microsemi M2S090TS-EVAL-KIT. You can adapt this document to your own target board. Keil supports 8051, ARM7, Cortex-M1 and Cortex-M3 processors as used in various Microsemi (Actel) products. For more information see www.keil.com/Microsemi

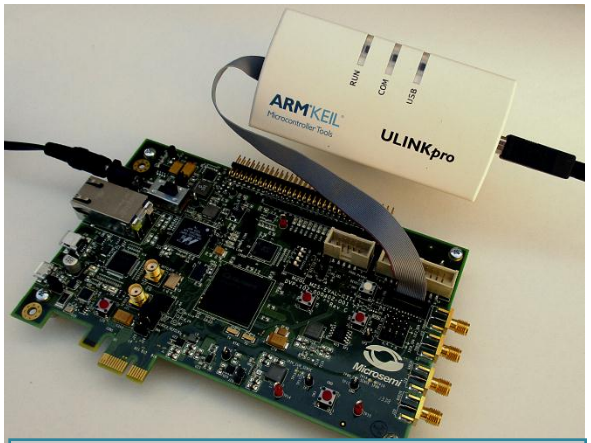
Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into an unrestricted commercial version.

CMSIS is an ARM standard. CMSIS 5 has an Apache 2.0 license. GitHub: https://github.com/ARM-software/CMSIS_5

Why Use Keil MDK ?

MDK provides these features suited for Microsemi products:

1. µVision IDE with Debugger, Flash programmer and the ARM® Compiler. MDK is turn-key "out-of-the-box".
2. ARM Compiler 5 and ARM Compiler 6 (LLVM) included. GCC is supported. https://launchpad.net/gcc-arm-embedded
3. Dynamic Syntax Checking on C source lines.
4. **Compiler Safety Certification Kit:** www.keil.com/safety/
5. **TÜV** certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
6. ARM Compiler 5 and Compiler 6 (LLVM) are included. GCC is supported. https://launchpad.net/gcc-arm-embedded
7. MISRA C/C++ support using PC-Lint. www.gimpel.com
8. Keil RTX is included. This full feature RTOS has a BSD license and sources are provided. Two kernel aware windows are provided which update in real time. www.keil.com/RTX.
9. CoreSight™ Serial Wire Viewer and ETM trace capability.
10. Choice of debug adapters: ULINK2™, ULINK-ME™, ULINK*pro*™, CMSIS-DAP.
11. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
12. Affordable perpetual and term licensing. Contact Keil sales for pricing, options and current special offers.
13. Support for Microsemi ARM7, Cortex-M1 and Cortex-M3. 8051 is supported with PK51 which also uses µVision.
14. MDK is compatible with Microsemi development products. Microsemi Libero® can create µVision projects. µVision can debug an executable ELF file generated with SoftConsole®. Normally, projects will be compiled using the ARM C/C++ compiler/assembler and debugged with µVision's integrated debugger. µVision includes SWV and ETM trace.



Keil ULINK*pro* connected to the CoreSight ETM connector on the M2S090TS-EVAL-KIT.

Serial Wire Viewer (SWV):

Serial Wire Viewer displays PC Samples, Exceptions (including interrupts), data writes, ITM, CPU counters and a timestamp. SWV (and ETM) does not steal any CPU cycles, is non-intrusive and requires no stubs in your source code. SWV will work with a ULINK2, ULINK-ME or ULINK*pro*.

Embedded Trace Macrocell (ETM Instruction Trace):

ETM records all instructions executed for program flow debugging and analysis. Code Coverage, Performance Analysis and Execution Profiling are provided with ETM. ETM requires a ULINK*pro* debug adapter and the CoreSight 20 ETM debug connector as found on the Microsemi M2S090TS-EVAL-KIT. ETM exercises are provided Instruction Tracing, Code Coverage, Performance Analysis and a Gone-In-The-Weeds scenario showing a Hard Fault.

Part 1) Introduction and Getting Ready:

1) This document details these features *and more*:

1. Serial Wire Viewer (SWV). This is a data trace.
2. ETM instruction trace. A ULINK*pro* is needed.
3. Real-time Read and/or Write to memory locations for Watch, Memory and System Viewer SV (Peripheral) windows.
4. No-skid Hardware Breakpoints and Watchpoints (Access Breaks).
5. RTX RTOS operation using two kernel awareness windows.
6. A DSP example using ARM DSP libraries. Uses Serial Wire Viewer to display results.

2) SmartFusion2 Development Boards:

This document details operation with the Microsemi M2S090TS-EVAL-KIT as pictured on page 1. All examples will run on the EmCraft board but it does not have access to the ETM 4 bit trace port. There is a lab available for the EmCraft board. Both boards support Serial Wire Viewer. The examples can be adapted for custom boards.

3) STP File:

An STP file must be programed into the SF2 eNVM in order to configure the PLL clock and other hardware aspects. If you want to use ETM trace, the Trace Port must also be activated. An example .STP file is provided by Keil to run with the examples. An STP file is created by Microsemi Libero SoC Design Suite. This process is also used for custom boards.

4) General Information and Introduction to Keil MDK:

Keil Software: MDK 5 This document used MDK 5.23 and M2Sxxx Software Pack 1.0.61. You can use later versions.

MDK 5 uses Software Packs to distribute processor specific software, examples, documentation and middleware. MDK 5 Core is first installed and you then download the Software Pack required for your processor(s) from the web. A Pack can also be imported manually. You no longer need to wait for the next version of MDK or install patches to get the latest files. Software Packs are an ARM CMSIS standard. See www.keil.com/cmsis and https://github.com/ARM-software/CMSIS_5

Libero SoC Design Suite provides software projects in MDK format consistent with Software Packs.

Summary of the Keil software installation: This is a simple three step process:

A. Download and install MDK Core. This is done in the next paragraph.

B. Download and install the appropriate Software Pack for the processor you are using. This is done on the next page.

C. In addition, you need to download and install the examples used in this tutorial. See page 5.

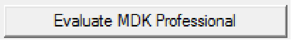
Part 2: Software Installation:

1) Keil MDK Core Software Download and Installation:

1. Download MDK Core from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5
3. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
4. If you install MDK or the examples into different folders, you will have to adjust for the folder location differences.

Licensing:

1. You can use the evaluation version (MDK-Lite) for this lab. No license is needed for this lab.
2. You can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible:
3. This gives you unlimited code size compilation. Contact Keil Sales to extend this license for evaluation purposes.



USA: North and South America:
Keil, An ARM Company
Plano, Texas
800-348-8051 (Toll Free)
sales.us@keil.com support.us@keil.com

Europe and Asia:
Keil, An ARM Company
Grasbrunn, Germany
+49 89/456040-20
sales.intl@keil.com support.intl@keil.com

2) Software Pack Installation Process:

1) Start µVision and open Pack Installer:

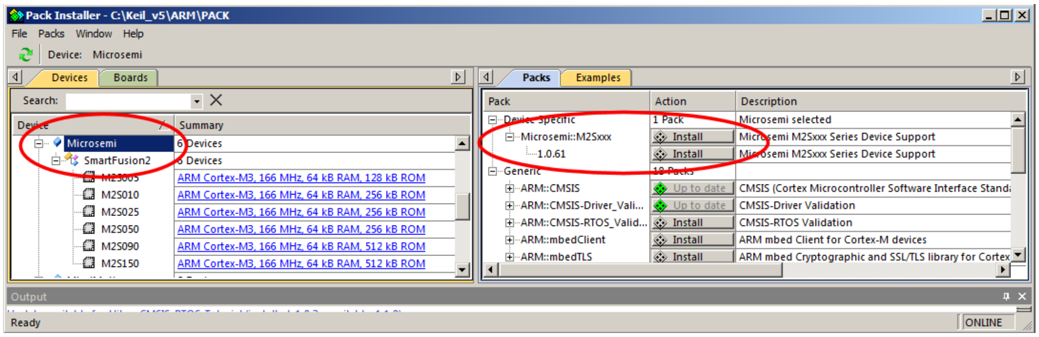
When the first MDK install is complete and if you are connected to the Internet, µVision and Software Packs will automatically start. Otherwise, follow Steps 1 and 2 below. Initially, the Pack master list will be downloaded automatically from the web.

1. Connect your computer to the Internet. This is normally needed to download the Software Packs.
2. Start µVision by clicking on its desktop icon. The Pack descriptions will download on the initial µVision run.
3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
4. If there are any Updates available, you can download them now if they are applicable. Update
5. The window below opens up: Select the Devices tab. Scroll down and select Microsemi as shown below. You could select a specific Microsemi processor but in this case one Pack supports all of them at this time.

TIP: The Devices and Boards tabs are used to filter the items displayed on the right side in the Packs and Examples tabs.

6. Note: "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet.

TIP: If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or to refresh once you have connected to the Internet.



Device	Summary
Microsemi	6 Devices
SmartFusion2	6 Devices
M2S005	ARM Cortex-M3, 166 MHz, 64 kB RAM, 128 kB ROM
M2S010	ARM Cortex-M3, 166 MHz, 64 kB RAM, 256 kB ROM
M2S025	ARM Cortex-M3, 166 MHz, 64 kB RAM, 256 kB ROM
M2S050	ARM Cortex-M3, 166 MHz, 64 kB RAM, 256 kB ROM
M2S090	ARM Cortex-M3, 166 MHz, 64 kB RAM, 512 kB ROM
M2S150	ARM Cortex-M3, 166 MHz, 64 kB RAM, 512 kB ROM

Pack	Action	Description
Device Specific	1 Pack	Microsemi selected
Microsemi::M2Sxxx	Install	Microsemi M2Sxxx Series Device Support
1.0.61	Install	Microsemi M2Sxxx Series Device Support
Generic	18 Packs	
ARM::CMSIS	Up to date	CMSIS (Cortex Microcontroller Software Interface Standa
ARM::CMSIS-Driver_Vali...	Up to date	CMSIS-Driver Validation
ARM::CMSIS-RTOS_Valid...	Install	CMSIS-RTOS Validation
ARM::mbedClient	Install	ARM mbed Client for Cortex-M devices
ARM::mbedTLS	Install	ARM mbed Cryptographic and SSL/TLS library for Cortex

2) Install the Microsemi SmartFusion2 Device Family Pack (M2Sxxx):

1. Click on the Packs tab. Initially, the Software Pack ARM::CMSIS is installed by default.
7. Select the Install icon beside Microsemi M2Sxxx as shown above.
2. The latest Pack will download and install to C:\Keil_v5\ARM\PACK\Microsemi\ by default. This download can take two to four minutes depending on your Internet connection speed.
3. The Pack's status will then be indicated by the "Up to date" icon: Up to date
1. Close Pack Installer.

TIP: You can also install a Pack manually. A Pack has a file extension .pack. It is an ordinary zip file with the extension changed so it is recognized by µVision. You can download the Pack from the web or transfer the file in any other way. Double click on this file and it will automatically be recognized (.pack) and installed by Pack Installer.

TIP: You can create your own Pack to distribute a DFP, BSP and an SDK. This is a good way to distribute confidential material.

For CMSIS-Pack documentation: www.keil.com/pack/doc/CMSIS/Pack/html/

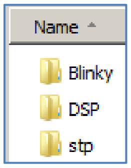
For complete CMSIS documentation: www.keil.com/CMSIS/

For CMSIS 5 on GitHub: https://github.com/ARM-software/CMSIS_5

3) Copy the Blinky and DSP Examples and the STP files to your PC:

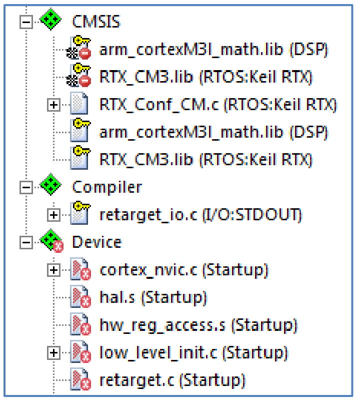
Copy the Blinky and DSP Examples and stp files from www.keil.com:

1. Obtain the example software zip file from www.keil.com/appnotes/docs/apnt_292.asp.
2. Create this folder: C:\00MDK\Boards\Microsemi\M2S090TS\
3. Extract the example zip into this folder. This is what the folders will look like:



4) Updating CMSIS-Packs Files:

You may see red marks on various files in the Project window as shown here: This is because you have installed a newer version of a Software Pack. See page 8 for descriptions of what these red icons mean.

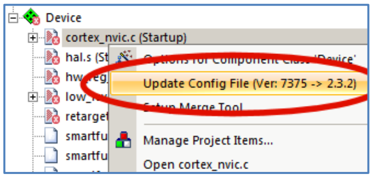

CMSIS
arm_cortexM3l_math.lib (DSP)
RTX_CM3.lib (RTOS:Keil RTX)
RTX_Conf_CM.c (RTOS:Keil RTX)
arm_cortexM3l_math.lib (DSP)
RTX_CM3.lib (RTOS:Keil RTX)
Compiler
retarget_io.c (I/O:STDOUT)
Device
cortex_nvic.c (Startup)
hal.s (Startup)
hw_reg_access.s (Startup)
low_level_init.c (Startup)
retarget.c (Startup)

Some files stored in the .\RTE folder are now older. You have the option of updating these files. To use the older files see **A) Select Software Pack Version: on page 7.** You can select the appropriate file version you want to use. It is recommended top update the files for this tutorial.

Updating the Files:

1. Right click on each filename in turn and select Update Config File as shown below:
2. Each file will be updated in turn. Repeat until they are all updated.
3. If you see any messages concerning updating or replacing files or Packs, you can select OK or Yes.

TIP: When you use another stp file that you created, you normally copy over several files from the Libero project. This means these files will not be the same as found in the SmartFusion 2 Pack. This is why you see these red notices.



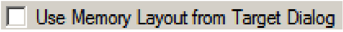
If you have moved these CMSIS like files into your µVision project, you might not want to update these files. This causes the issue described and fixed below:

Modifying low_level_init.c and startup_m2sxxx.s:

If you select to update these two files, a compilation will produce errors. There are two solutions. Either these two files must be modified as described below or use an external scatter file produced by Libero as opposed to the one created by µVision.

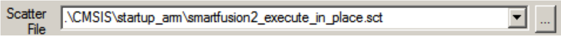
A) Use Libero Scatter File: (this allows external DDR memory to be used)

1) Click on the Target Options icon or select Project/Options for Target.
2) Select the Linker tab.
3) Unselect Use Memory Layout from Target Dialog:



☐ Use Memory Layout from Target Dialog

4) In the Scatter File box, find and enter this scatter file:



Scatter File .\CMSIS\startup_arm\smartfusion2_execute_in_place.sct

5) Click OK and build the project.

B) Modify Files:

1) Open low_level_init.c by clicking on its tab or double click it in the project window.
2) Comment out these two lines by using a semi-colon **;**:

```
18      Image$$ER_RO$$Base
19      Image$$ER_RW$$Base
```

3) The function low_level_init is located in low_level_init.c starting at line 25. Comment out this entire function from lines 25 through 43.
4) Open startup_m2sxxx.s by clicking on its tab or double click it in the Project window.
5) Comment out these three lines (by using a semi-colon ;)

```
163    ;   IMPORT   low_level_init
210    ;   LDR      R0, =low_level_init
211    ;   BLX      R0
```

6) Click on File/Save All or

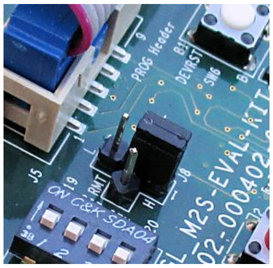
Build the project and there will be no errors or warnings.

5) Programming the SmartFusion2 FPGA eNVM Flash with the example STP file:

It is necessary to program the FPGA fabric with a STP file. An example stp file created with Libero is provided for this lab. This configures the SmartFusion FPGA fabric and configures the clocks. Once you have programmed the STP file into the fabric, you can use µVision to program your executable Cortex-M3 code multiple times into the eNVM flash without conflict.

The .stp file has been copied to C:\00MDK\Boards\Microsemi\M2S090TS\stp\ on page 5 of this tutorial.

Connect FlashPro4 and Configure Jumper J8:

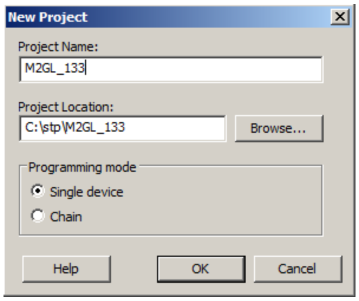


1. Connect the FlashPro4 to J5 on your board and to your PC USB. Photo on page 10.
2. Set the jumper J8 JTAG_SEL to position H as shown here.
3. If you have a Rev E of the board and hence have a J35, set this to FP4 JTAG. Rev D does not have J35.
4. Power the board with the supplied 12 volt AC adapter to J6.
5. Turn power on with SW7.

Select the .stp file:

6. Open the FlashPro4 Programming Software. The window below right opens:

   TIP: FlashPro4 is available stand-alone or as a component of the Libero IDE FPGA development software.
7. Create a new programming project by clicking the New Project: New Project
8. Name your project and pick a directory to save it as shown here:
9. Click on OK.



New Project

Project Name: M2GL_133

Project Location: C:\stp\M2GL_133 Browse...

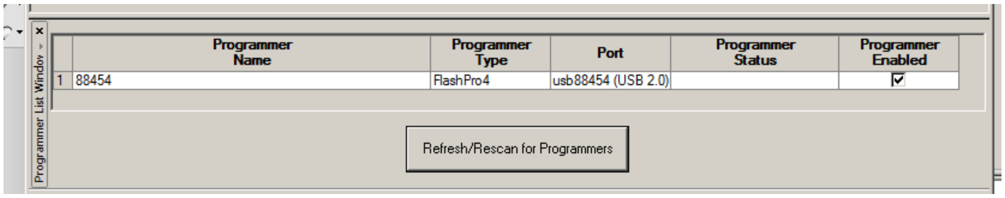
Programming mode

● Single device

○ Chain

Help OK Cancel

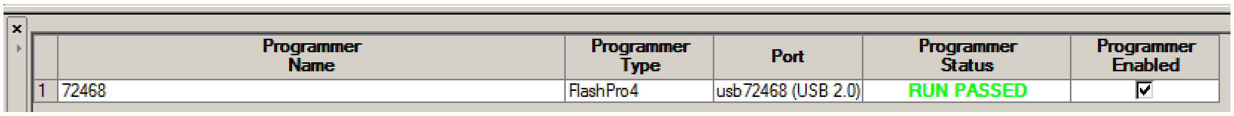
10. Click on "Configure Device". Configure Device The Load Programming File window will open. This window is not shown here.
11. Use the Browse… button to locate your stp file.
12. Highlight it and select Open.
13. Information about your stp file will be displayed in the Programming File box.
14. Click View Programmers: View Programmers
15. The FlashPro4 software will connect to the FlashPro4 Device Programmer. As shown below:



	Programmer Name	Programmer Type	Port	Programmer Status	Programmer Enabled
1	88454	FlashPro4	usb88454 (USB 2.0)		☑

Program the eNVM Flash with the STP file:

16. Click the Program button to program the SmartFusion2 device. PROGRAM
17. If there is a warning about Vpump – you can safely ignore this message. If it fails, check J8 and J35.
18. It took FlashPro about 80 seconds to complete programming the STP file I was using. If FlashPro stops in a few seconds: there is probably an error. Look in the information window for the cause.
19. If the programming is successful RUN PASSED will display as shown below: Status information will display in a window in the ALL tab including any programming error messages.



	Programmer Name	Programmer Type	Port	Programmer Status	Programmer Enabled
1	72468	FlashPro4	usb72468 (USB 2.0)	RUN PASSED	☑

20. The SF2 has been successfully programmed with the STP file. Turn the power off and remove the FlashPro4.
21. Set the jumper: J8 JTAG_SEL to position L to activate Debug mode. See the page 10 for a photo.
22. At this point, everything is configured to allow a debug adapter such as any ULINK to be connected to J4 or J10.

6) Software Pack Version Selection and Manage Run-Time Environment:

These three sections are provided only for reference.

This section contains three parts on this and the next page:

A) Select Software Pack Version:

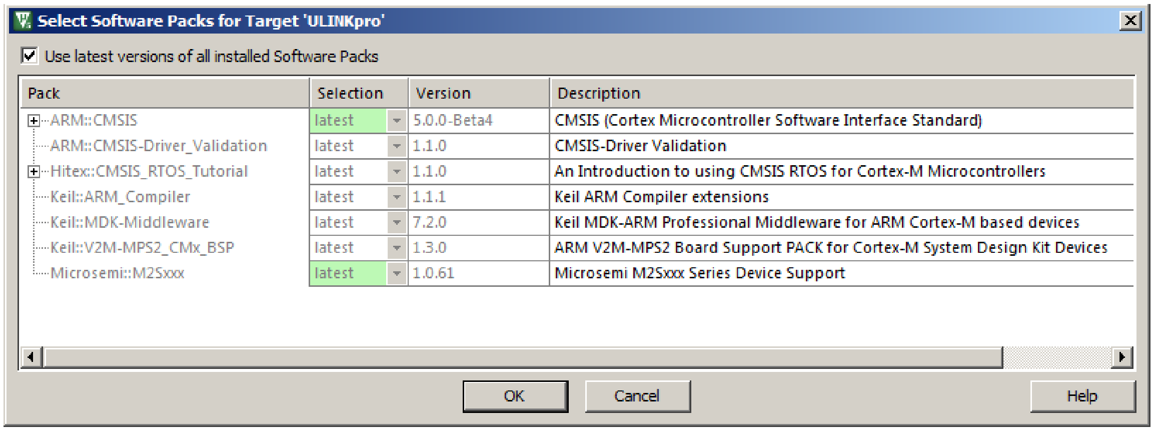
B) Manage Run-Time Environment:

C) Updating Source Files:

A) Select Software Pack Version: *this section is provided for reference:*

This µVision utility provides the ability to choose various Software Pack versions installed in your computer. You can select the versions you want to use. You must have µVision running and any project open for the following exercises:

1. Select Project/Open Project and navigate to C:\00MDK\Boards\Microsemi\Blinky.
2. Open Blinky.uvprojx. Blinky will load into µVision.
3. Open Select Software Packs by clicking on its icon:
4. This window opens up. Note **Use latest versions …** is selected. The latest version of the Pack will be used.
5. Unselect this setting and the window changes as shown similar to the one below right:

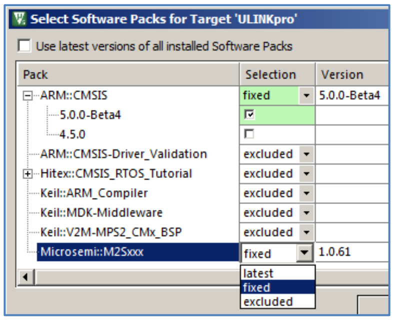


Select Software Packs for Target 'ULINKpro'

☑ Use latest versions of all installed Software Packs

Pack	Selection	Version	Description
⊞ ARM::CMSIS	latest	5.0.0-Beta4	CMSIS (Cortex Microcontroller Software Interface Standard)
ARM::CMSIS-Driver_Validation	latest	1.1.0	CMSIS-Driver Validation
⊞ Hitex::CMSIS_RTOS_Tutorial	latest	1.1.0	An Introduction to using CMSIS RTOS for Cortex-M Microcontrollers
Keil::ARM_Compiler	latest	1.1.1	Keil ARM Compiler extensions
Keil::MDK-Middleware	latest	7.2.0	Keil MDK-ARM Professional Middleware for ARM Cortex-M based devices
Keil::V2M-MPS2_CMx_BSP	latest	1.3.0	ARM V2M-MPS2 Board Support PACK for Cortex-M System Design Kit Devices
Microsemi::M2Sxxx	latest	1.0.61	Microsemi M2Sxxx Series Device Support

6. Expand the header ARM::CMSIS as shown here:
7. Expand the Microsemi heading too;
8. You will see only one version – the one you installed. If you had installed others, you would see them listed like this:

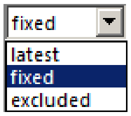


Select Software Packs for Target 'ULINKpro'

☐ Use latest versions of all installed Software Packs

Pack	Selection	Version
⊟ ARM::CMSIS	fixed	5.0.0-Beta4
5.0.0-Beta4	☑	
4.5.0	☐	
ARM::CMSIS-Driver_Validation	excluded	
⊞ Hitex::CMSIS_RTOS_Tutorial	excluded	
Keil::ARM_Compiler	excluded	
Keil::MDK-Middleware	excluded	
Keil::V2M-MPS2_CMx_BSP	excluded	
Microsemi::M2Sxxx	fixed	1.0.61

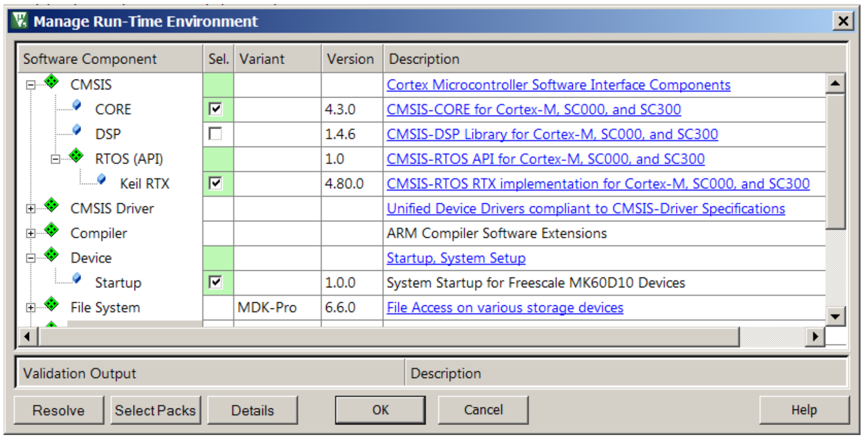
9. Select the fixed pull-down menu and see the three options as shown below:
10. If you wanted to use a different version, you would select fixed and then select the check box opposite the version you wanted to use in your project.



11. Re-select Use latest versions… **Do not make any changes at this time.**
12. Click OK or Cancel to close this window.

B) Manage Run-Time Environment: *this section is provided for reference:*

1. Click on the Manage RTE icon: ◆ The window below opens: This includes Keil Middleware, selected open source software, RTX and CMSIS drivers for various peripherals. Not all Packs offer all options but more are being added.
2. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project. You can now use them in your program.
3. Note CMSIS/Core (system.c), Keil RTX and Device/Startup (startup.s) files are selected. You can see these files in the µVision Project window.
4. **Do not make any changes.**
   Click Cancel to close this window.



Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CORE	☑		4.3.0	CMSIS-CORE for Cortex-M, SC000, and SC300
DSP	□		1.4.6	CMSIS-DSP Library for Cortex-M, SC000, and SC300
RTOS (API)			1.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
Keil RTX	☑		4.80.0	CMSIS-RTOS RTX implementation for Cortex-M, SC000, and SC300
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler				ARM Compiler Software Extensions
Device				Startup, System Setup
Startup	☑		1.0.0	System Startup for Freescale MK60D10 Devices
File System		MDK-Pro	6.6.0	File Access on various storage devices

Validation Output	Description

TIP: Different colors represent messages:

🟩☑ Green: all required files are located.

🟨☑ Yellow: some files need to be added. Click the Resolve button to add them automatically or add them manually.

🟥☑ Red: some required files could not be found. Obtain these files or contact Keil technical support for assistance.

The Validation Output area at the bottom of this window gives some information about needed files and current status.

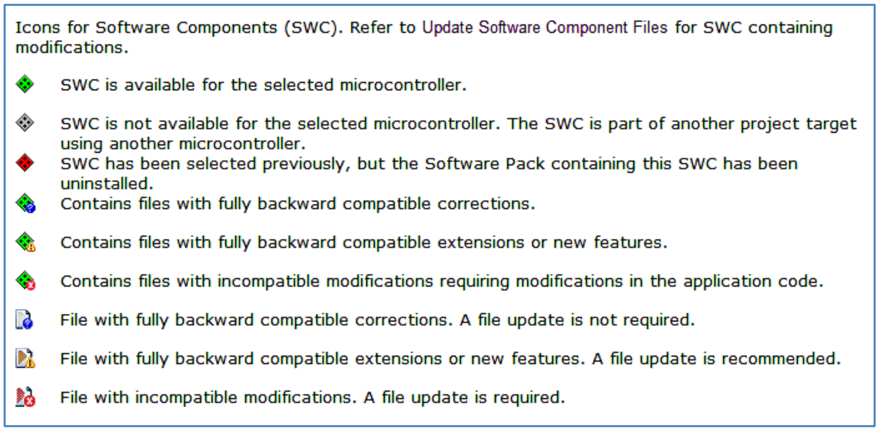
C) Updating Source Files: *this section is provided for reference:*

Some of the files provided by a Software Pack are stored in your project in .\RTE

If you update a Software Pack, some of these files might need to be updated from the new Pack. These files will show new icons as shown below and described here: www.keil.com/support/man/docs/uv4/uv4_ca_updswcmpfiles.htm

Updating Source Files:

1. Right click on a file you want to update. A window similar to the one below right opens:
2. Select the appropriate Update selection. Any changes you made to the original file are lost.
3. This procedure is described here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm



Icons for Software Components (SWC). Refer to Update Software Component Files for SWC containing modifications.

◆ SWC is available for the selected microcontroller.

◇ SWC is not available for the selected microcontroller. The SWC is part of another project target using another microcontroller.

◆ SWC has been selected previously, but the Software Pack containing this SWC has been uninstalled.

◆ Contains files with fully backward compatible corrections.

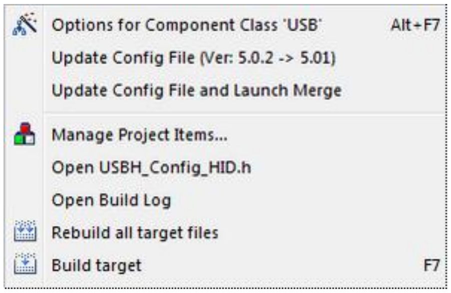
◆ Contains files with fully backward compatible extensions or new features.

◆ Contains files with incompatible modifications requiring modifications in the application code.

📄 File with fully backward compatible corrections. A file update is not required.

📄 File with fully backward compatible extensions or new features. A file update is recommended.

📄 File with incompatible modifications. A file update is required.



Options for Component Class 'USB' Alt+F7
Update Config File (Ver: 5.0.2 -> 5.01)
Update Config File and Launch Merge
Manage Project Items...
Open USBH_Config_HID.h
Open Build Log
Rebuild all target files
Build target F7

7) CoreSight Definitions: It is useful to have a basic understanding of these terms:

Cortex-M0 and Cortex-M0+ may have only features 2) and 4) plus 11), 12) and 13) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 can have all features listed implemented. MTB is normally found on Cortex-M0+. It is possible some processors have all features except ETM Instruction trace and the trace port. Consult your specific datasheet.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except Boundary Scan is not possible. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup.
   The SWJ box must be selected in ULINK2/ME or ULINK*pro*. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDO shares the same pin as SWO. The SWV data normally comes out the SWO pin or Trace Port.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **DAP:** Debug Access Port. This is a component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. µVision uses the DAP to update Memory, Watch, Peripheral and RTOS kernel awareness windows while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed.
   You do not need to configure or activate DAP. µVision configures DAP when you select a function that uses it.
   Do not confuse this with CMSIS_DAP which is an ARM on-board debug adapter standard.
5. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
6. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
7. **Trace Port:** A 4 bit port that ULINK*pro* uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by µVision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK*pro* provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis. ETM is output on the Trace Port or accessible in the ETB (ETB has no Code Coverage or Performance Analysis).
10. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK*pro*. ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. Cortex-M3/M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 usually have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs. The CPU is halted before the instruction is executed.
13. **Watchpoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. There also referred to as Access Breaks in Keil documentation.

Read-Only Source Files:

Some source files in the Project window will have a yellow key on them: 🔑 This means they are read-only. This is to help unintentional changes to these files. This can cause difficult to solve problems. These files normally need no modification.

If you need to modify one, you can use Windows Explorer to modify its permission.

1. In the Projects window, double click on the file to open it in the Sources window.
2. Right click on its source tab and select Open Containing folder.
3. Explorer will open with the file selected.
4. Right click on the file and select Properties.
5. Unselect Read-only and click OK. You are now able to change the file in the µVision editor.
6. It is a good idea to make the file read-only when you are finished modifications.

Super TIP: µVision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

8) Connecting a Keil ULINK2 or a ULINK*pro* to the M2S090TS-EVAL-KIT:

A. The Microsemi FlashPro software with the FlashPro4 hardware is used to program the SmartFusion2 FPGA eNVM Flash with an stp file. FlashPro is available stand-alone program or integrated into Libero. Libero creates the stp file.

B. ULINK2, ULINK-ME or a ULINK*pro* is used to program the Cortex-M3 executable into eNVM Flash. µVision also supports CMSIS-DAP. µVision is used is create, compile, run and debug your software project.

a) The Debug Adapters:

1. Use a Keil ULINK2 or a ULINK-ME. Serial Wire Viewer is supported. ETM is not. Connect to RVI J4.
2. A ULINK*pro* connects into the CoreSight Trace DBG 20 pin connector J10. ETM instruction trace is then provided.

Keil manufactures several adapters. These are listed below with a brief description.

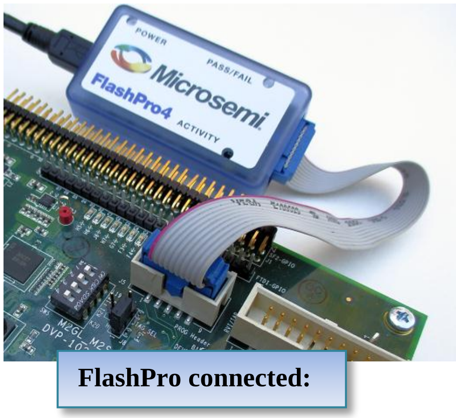
1. **ULINK2 and ULINK-ME:** ULINK2 is pictured below. ULINK-ME is offered only as part of certain evaluation board packages. ULINK2 can be purchased separately. They are electrically the same and both support Serial Wire Viewer (SWV), run-time memory reads and writes for the Watch, Memory and SVD Peripheral windows, Watchpoints and hardware breakpoints can be set/unset on-the-fly while the program is running..
2. **ULINK*pro*:** This is pictured on page 1. ULINK*pro* supports all ULINK2 features and adds ETM Trace support. ULINKpro provides higher SWV throughput on the 4 bit Trace Port as well as the 1 bit SWO pin. ETM records all executed instructions. ETM provides complete instruction flow debugging, Code Coverage, Execution Profiling and Performance Analysis. ULINK*pro* also provides the fastest Flash programming times.

Jumper **JTAG_SEL J8:** Must be selected to switch between FlashPro and JTAG/SWD debug modes.

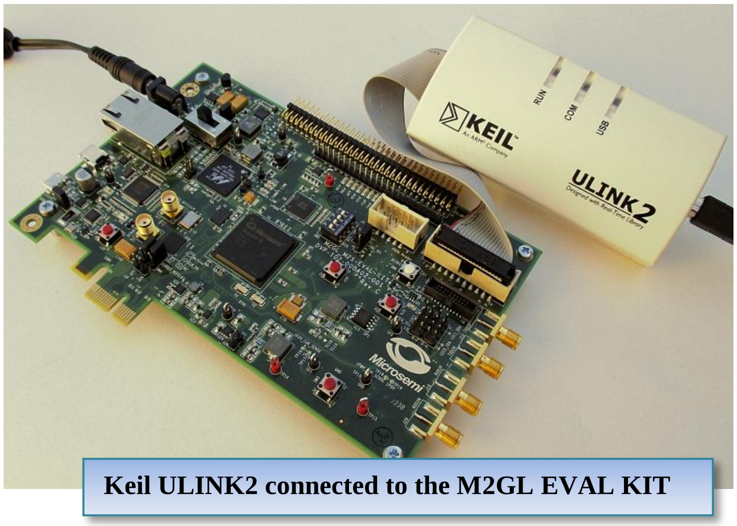
Position **H** is used to activate Microsemi FlashPro4 programmer as shown below. All other board jumpers are set to default.

Position **L** is used to activate JTAG/SWD mode for use with any Keil ULINK, CMSIS-DAP or J-Link. See below:

Jumper J35: If you have a Rev E board, you must always have J35 set to FP4 JTAG. Rev D does not have a J35.

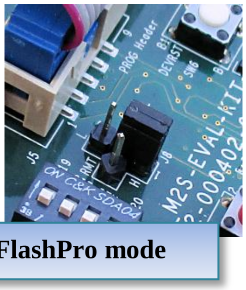


FlashPro connected:

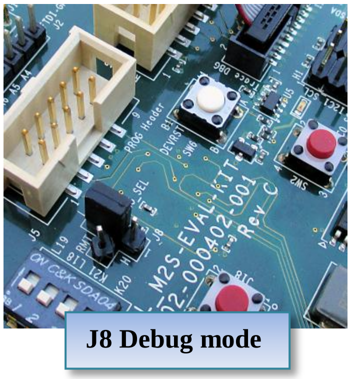


Keil ULINK2 connected to the M2GL EVAL KIT

<span style="color:red">**Super TIP:**</span> It is best to not plug both the FlashPro4 and a ULINK to the board at the same time. This avoids any conflicts.



J8 FlashPro mode



J8 Debug mode

Part 3: Running Blinky and Testing µVision Debug Features:

1) *Blinky* example program using Microsemi SmartFusion2:

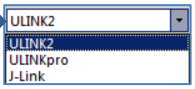
Now we will run the Keil MDK development system using a SmartFusion2 M2S090TS-EVAL-KIT.

An STP file needs to be programmed into the SmartFusion2 processor. Instructions are on page 6.

Connection: Connect your board with a suitable debug adapter and power it as described on the previous page.

Compile, Load and Run the example Blinky program:

1. Select Project/Open Project from the main menu.
2. Open the file C:\00MDK\Boards\Microsemi\M2S090TS \Blink.uvprojx.
3. Select your debug adapter. This example program has been preconfigured:



4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode notice appears. The SmartFusion2 eNVM flash will be programmed with the executable. Progress is indicated in bottom left corner.

ERROR TIP: If you get an error "No JTAG Devices Found", the most likely reason is J8 is set incorrectly. Set J8 to L.

6. Click on the RUN icon to start the Blinky program. . ***Blinky is now running !***

> ***The eight LEDs will now blink.***
>
> **Now you know how to compile a program, load it into the SmartFusion eNVM Flash, run it and stop it.**
>
> **Note:** The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

UART Message:

A message is sent out the USB Serial port on J18 FTDI connector. You will need to use a terminal program such as PuTTY to view this message. You can also type in letters and these will display. **Caution:** make sure a source window is not in focus when you type any letters. You may inadvertently change a source file and find it difficult to find the error later. Your terminal program must be in focus for this to function properly and not cause problem.

µVision provides a printf utility that displays messages in a window and does not require a UART. This uses SWV.

Setting/Unsetting Hardware Breakpoints:

7. Bring main.c in focus by clicking on its tab. If it is not visible, double-click on it in the Project window.
8. Scroll down to inside the for(;;) loop found starting near line 104.
9. Note on the left of the line numbers are darker grey blocks. This indicates there is assembly code present and you can set a hardware breakpoint on these lines. You can also see these blocks in the Disassembly window.
10. *While the program is still running*, click to the left of a suitable line in the for loop (but not exactly on the for loop statement) and a red circle will be created. This is a hardware breakpoint. SmartFusion2 has six you can use. µVision will warn you if you exceed this limit.
11. Note that the program will soon stop at the line where you set the breakpoint on. The yellow arrow is the current program counter position. This will be the next instruction executed when the CPU is started again. The cyan arrow is a placeholder you can use to explore the relationship between a source window and the Disassembly window.
12. Each time you click on RUN, the program will cycle to the next breakpoint. .
13. Remove all breakpoints by clicking on them or Ctrl-B and choose Kill All.

TIP: You can set/unset hardware breakpoints with µVision while the program is running or stopped. Coretx-M3 has 6 hardware breakpoints.

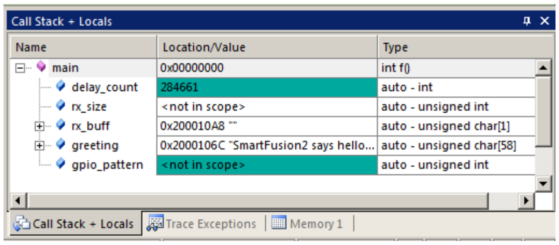
TIP: You can also select Debug/Breakpoints or Ctrl-B to view and delete the breakpoint table.

2) Call Stack + Locals Window:

Local Variables:

The Call Stack and Locals windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function.

If possible, the values of the local variables will be displayed and if not, the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

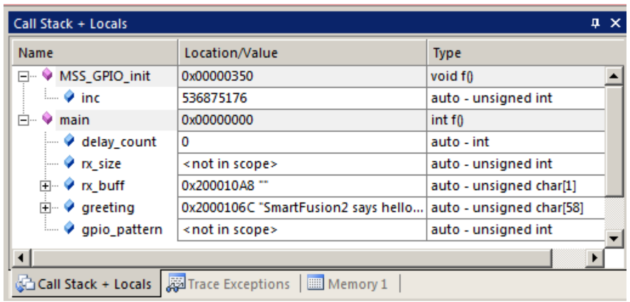


Call Stack + Locals

Name	Location/Value	Type
main	0x00000000	int f()
delay_count	284661	auto - int
rx_size	<not in scope>	auto - unsigned int
rx_buff	0x200010A8 ""	auto - unsigned char[1]
greeting	0x2000106C "SmartFusion2 says hello...	auto - unsigned char[58]
gpio_pattern	<not in scope>	auto - unsigned int

1. Run Blinky for a few seconds and then stop the CPU .
2. Click on the Call Stack + Locals tab if necessary to open it.
3. Shown is the Call Stack + Locals window.
4. The contents of the local variables are displayed as well as function names. Only main() is displayed in this case.
5. Each time you RUN and STOP, the variables will be updated. They do not update while the program is running.
6. Set a breakpoint in main.c on MSS_GPIO_init(); which is near line 54.
7. Click on RESET to reset the CPU. Click RUN and the program will run to MSS_GPIO_init(): and stop.
8. Click on the Step icon or F11: Make sure the main.c window is in focus else you will step in assembly.
9. The program enters the MSS_GPIO_init(); function. The Call Stack + Locals window will display as shown here:
10. This function is displayed. Click on the Step icon or F11 multiple times. Note the way various other functions are displayed. This is a very simple example but the operation is clear.

TIP: If the Disassembly window is in focus the steps will be by assembly instruction. If a source window is in focus: the steps will be by C or C++ source line.



Call Stack + Locals

Name	Location/Value	Type
MSS_GPIO_init	0x00000350	void f()
inc	536875176	auto - unsigned int
main	0x00000000	int f()
delay_count	0	auto - int
rx_size	<not in scope>	auto - unsigned int
rx_buff	0x200010A8 ""	auto - unsigned char[1]
greeting	0x2000106C "SmartFusion2 says hello...	auto - unsigned char[58]
gpio_pattern	<not in scope>	auto - unsigned int

11. Click on the Step Out icon or CTRL-F11 to exit all function(s) to return to main(). This will be indicated in the Call Stack window.
12. **When you are ready to continue, remove the hardware breakpoint by clicking on its red circle ! You can also type Ctrl-B and select Kill All.**

TIP: You can modify a variable value in the Call Stack & Locals window when the program is stopped.

TIP: You can set and unset breakpoints on-the-fly. No need to stop the CPU. A CoreSight hardware breakpoint does not execute the instruction it breaks on. These are rather important features.

TIP: This is standard "Stop and Go" debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables updated in real-time while the program is running. No additions or changes to your code are required. Variable update while the program is running is not possible with local variables because they are usually stored in a CPU register. They must be converted to global or static variables so they always remain in scope: normally in RAM.

Call Stack:

The list of stacked functions is displayed when the program is stopped as you have seen. This is useful when you need to know which functions have been called and are stored on the stack. This can be quite useful for finding program crashes. Using the ETM trace with a ULINK*pro* in conjunction with the Call Stack provides much more debugging power.

!! Do not forget to remove the hardware breakpoint(s) before continuing.

TIP: You can access the Hardware Breakpoint table by clicking on Debug/Breakpoints or Ctrl-B. This is also where Watchpoints (also called Access Points) are configured. You can temporarily disable entries in this table by unchecking them. You can permanently delete them by using one of the Kill buttons.

3) Watch Window: Updated in Real-time !

We want to display a variable. The Watch and Memory windows can be displayed while the program is running. These must be global, static, a physical address, a structure or array or anything that visible in all functions. Locals will not be displayed until the program is stopped in the function they exist in.

Create a Global Variable:

1. Exit Debug mode . You can edit source code while in Debug mode but you must be in Edit mode to compile.
2. In main.c, declare a global variable near line 28: **`unsigned int count = 0;`**
3. At the end of the for(;;) loop, near line 125, just after the line MSS_GPIO_set_outputs(gpio_pattern); add these lines:

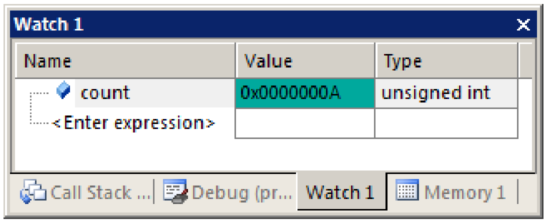
```
count++;
if (count > 0x0F) count = 0;
```

4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode by clicking on the Debug icon. The SmartFusion2 eNVM flash will be programmed.
6. Click on the RUN icon to start the Blinky program. .
7. Select View and confirm Periodic Windows Update is enabled:

☑ Periodic Window Update

8. Right-click on the variable name **`count`** and select Add ‘count’ to… and then select Watch 1.
9. The value of **`count`** will be displayed and updated in real-time without stealing any CPU cycles as shown here:
10. The value of **`count`** displayed depends on when it is sampled.

TIP: The value of **`count`** displayed periodically, note when it is written or changed. This means you will not see all values you know must exist as this variable is changing faster than the periodic refresh rate.



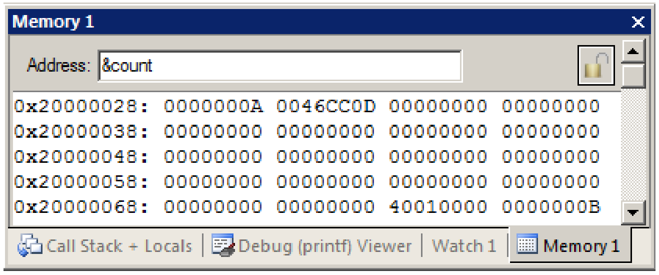
Watch 1

Name	Value	Type
count	0x0000000A	unsigned int
<Enter expression>		

4) Memory Window: it is also updated in Real-time:

1. Click on the Memory 1 tab to open it.
2. Enter **`count`** in the space Address: provided. The program can be running.
3. Right click anywhere inside the Memory data display area and select Unsigned Long.
4. Note that as **`count`** increments, the address it points to changes. This is useful for work with pointers.
5. Add an ampersand (&) in front of **`count`**. i.e. **`&count`** The window below opens:
6. Note the memory address changes to 0x2000 0028 in this case. You will probably have a different address.
7. This is the physical address in RAM where the variable **`count`** is stored.
8. Right-click on the data field (0x0A in this case) and select Modify Memory at 0x2…28 and enter 0x0 and press Enter.
9. This value is inserted into **`count`** in real time using CoreSight DAP technology. You might not see the result due to the high update rate.

How It Works: Read and Write to Memory Locations: µVision has the ability to read and write memory locations on-the-fly and without stealing CPU cycles by using the Debug Access Port (DAP). The Cortex-M3 is a Harvard architecture: this means it has separate code and data address busses. While the CPU is fetching instructions at full speed from the code space, µVision can access the data space via JTAG or SWD. This feature is active even while the CPU is halted. SWV is not used for this feature and can be disabled or not.



Memory 1

Address: &count

```
0x20000028: 0000000A 0046CC0D 00000000 00000000
0x20000038: 00000000 00000000 00000000 00000000
0x20000048: 00000000 00000000 00000000 00000000
0x20000058: 00000000 00000000 00000000 00000000
0x20000068: 00000000 00000000 40010000 0000000B
```

This feature is used to display data in the Watch and Memory windows, Event Counters as well as an RTX Awareness window. Not only can values of memory locations be displayed and updated in real-time: you can also insert values while the program is running in the Memory window as demonstrated above. The only instance real-time will be violated is in the unlikely event the CPU accesses a memory location the exact same time as µVision does.

5) System Viewer (SV) Peripherals:

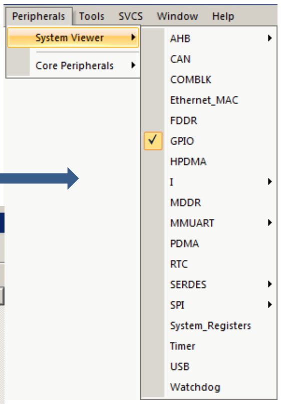
System Viewer provides the ability to view certain registers in the CPU core and in peripherals. In many cases, these Views are updated in real-time while your program is running. These Views are available only while in Debug mode. There are two ways to access these Views: **a)** View/System Viewer and **b)** Peripherals/System Viewer. In the Peripheral/Viewer menu, the Core Peripherals are also available:

In our Blinky example, the four LEDs blinking are connected to MSS_GPIO.

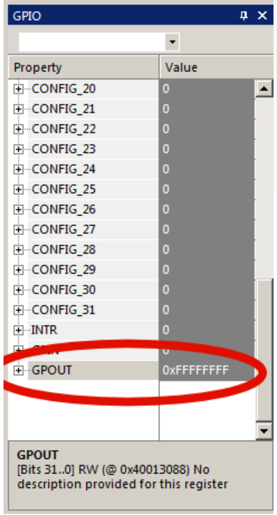
1. Click on RUN . You can open SV windows when your program is running.

GPIO Port A:

2. Select Peripherals/System Viewer then select GPIO as shown here:



3. The GTPIO window opens up:



Property	Value
CONFIG_20	0
CONFIG_21	0
CONFIG_22	0
CONFIG_23	0
CONFIG_24	0
CONFIG_25	0
CONFIG_26	0
CONFIG_27	0
CONFIG_28	0
CONFIG_29	0
CONFIG_30	0
CONFIG_31	0
INTR	0
[illegible]	0
GPOUT	0xFFFFFFFF

GPOUT
[Bits 31..0] RW (@ 0x40013088) No description provided for this register

4. You can now see GPOUT update as the LEDs blink:
5. You can change the values in the System Viewer while the program is running or stopped. It will be difficult to see this as these values are updated so often that your changes will be overwritten.
6. This window is updated using the same CoreSight DAP technology as the Watch and Memory windows.
7. Look at other Peripherals contained in other System Viewer windows to see what else is available.

TIP: If you click on a register in the properties column, a description about this register will appear at the bottom of the window.

TIP: It is true: you can modify values in the SV while the program is running. This is very useful for making slight timing value changes instead of the usual modify, compile, program, run cycle.

You must make sure a given peripheral register allows for and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.

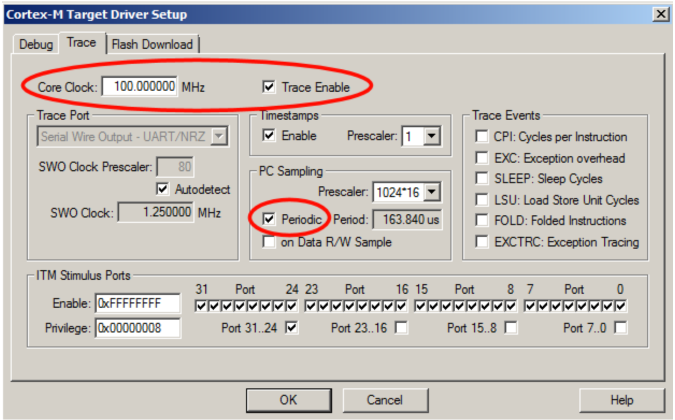
Part 4) Serial Wire Viewer (SWV): A Data Trace:

1) Configuring the Serial Wire Viewer (SWV) Trace for ULINK2/ME *and J-Link*:

These instructions are not for the ULINK*pro*, they are on the next page. In order to see the Serial Wire Viewer features, the Trace must first be configured. These instructions will also work with a J-Link. Some windows are different.

J-Link USB drivers are here: C:\Keil_v5\ARM\Segger. They must be installed manually.

6) Stop the CPU and exit Debug mode.
7) Click on the Target Options icon or select Project/Options for Target.
8) Click on the Debug tab.
9) Click Settings beside the USE: ULINK Cortex Debugger box. Confirm SWJ and SW are selected.
10) Click on the Trace tab to open this window:



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 100.000000 MHz ☑ Trace Enable

Trace Port: Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 80 ☑ Autodetect
SWO Clock: 1.250000 MHz

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16
☑ Periodic Period: 163.840 us
☐ on Data R/W Sample

Trace Events:
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☐ EXCTRC: Exception Tracing

ITM Stimulus Ports:
Enable: 0xFFFFFFFF
31 Port 24 ☑☑☑☑☑☑☑☑ 23 Port 16 ☑☑☑☑☑☑☑☑ 15 Port 8 ☑☑☑☑☑☑☑☑ 7 Port 0 ☑☑☑☑☑☑☑☑
Privilege: 0x00000008
Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

11) Set Core Clock to 100 MHz (or your CPU speed).

TIP: The Core Clock: value is set in the STP file. This value ***must*** be set correctly so SWV will work.

12) Select Trace Enable and EXCTRC.
13) Select Periodic. The window will look like this:
14) We will use Periodic to test SWV shown below. If you ***are not*** doing this test: do not select Periodic.
15) Leave everything at default settings.
16) Click on OK twice to return to the main screen.
17) The Serial Wire Viewer is now activated.
18) Click on File/Save All to save these settings.
19) The Serial Wire Viewer is enabled and ready to use. You can test it below.

TIP: How to open this window again to modify trace selections:

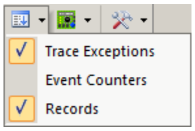
1) **In Edit mode:** The Target Options icon as already described.
2) **In Debug Mode:** Select Debug/Debug Settings from the main menu.

TIP: You can change Trace Enable only in Edit mode. You can change the rest of the options while in either mode.

You select various SWV elements in the Trace Configuration window. It is important to note that it is easy to overload the single pin Serial Wire Output (SWO). The general rule is to activate only those features you need.

Testing Serial Wire Viewer (SWV):

1. Enter Debug mode and click on RUN .
2. Open the Trace Records windows by selecting the small arrow beside its icon.
3. The Trace Records will display PC Samples as shown below if it is working correctly.
4. Double-click anywhere inside the window to clear it.
5. Trace Records updates while the program runs. This is non-intrusive to your program. No CPU cycles are stolen.

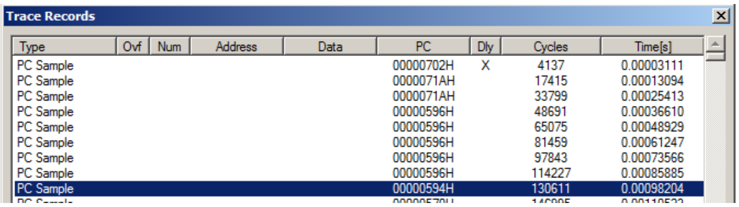


What if SWV doesn't work ?

If you see either nothing or spurious frames such as ITM frames with numbers other than 0 or 31, or other frames that do not make sense, the Core Clock: value is probably wrong. You need to know your CPU clock frequency. For help in determining the CPU frequency see www.keil.com/appnotes/docs/apnt_297.asp

Unselect Periodic: We only needed Periodic selected to provide some SWV frames for testing. This is rarely used.

6. Select Debug/Debug Settings.
7. Select the Trace tab.
8. Unselect Periodic and click OK.
9. Click OK if a dialog box states Configuration has changed. This occurs when you change trace configuration when the program runs.



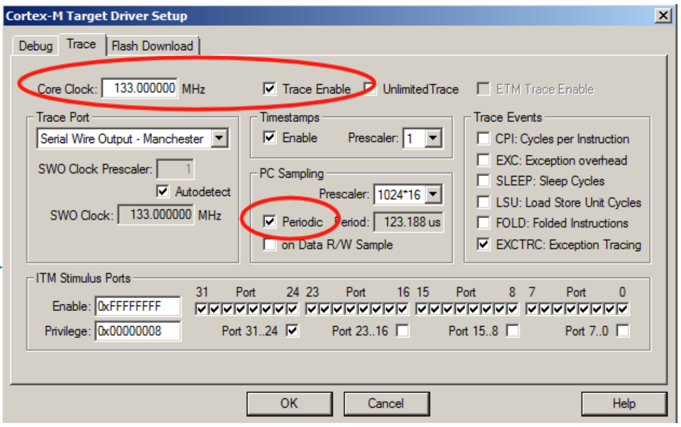
Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					00000702H	X	4137	0.00003111
PC Sample					0000071AH		17415	0.00013094
PC Sample					0000071AH		33799	0.00025413
PC Sample					00000596H		48691	0.00036610
PC Sample					00000596H		65075	0.00048929
PC Sample					00000596H		81459	0.00061247
PC Sample					00000596H		97843	0.00073566
PC Sample					00000596H		114227	0.00085885
PC Sample					00000594H		130611	0.00098204

2) Configuring the Serial Wire Viewer (SWV) Trace for ULINK*pro*:

In order to see the Serial Wire Viewer features, the Trace must first be configured. These instructions are not for the ULINK2/ME, these are on the previous page. We will connect using UART Manchester mode. You can also use the 4 bit Trace Port mode for even faster SWV performance and also gain use of ETM instruction trace.

1) Stop the CPU and exit Debug mode.
2) Click on the Target Options icon or select "Project/Options for Target".
3) Click on the Debug tab. Confirm ULINK Pro Cortex Debugger is selected.
4) Click Settings beside the USE: ULINK Cortex Debugger box. Confirm SWJ and SW are selected.
5) Click on the Trace tab to open this window: →



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 133.000000 MHz ☑ Trace Enable ☐ UnlimitedTrace ☐ ETM Trace Enable

Trace Port: Serial Wire Output - Manchester
SWO Clock Prescaler: 1 ☑ Autodetect
SWO Clock: 133.000000 MHz

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16
☑ Periodic Period: 123.188 us
☐ on Data R/W Sample

Trace Events:
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

ITM Stimulus Ports:
Enable: 0xFFFFFFFF (31 Port 24: ☑☑☑☑☑☑☑☑ 23 Port 16: ☑☑☑☑☑☑☑☑ 15 Port 8: ☑☑☑☑☑☑☑☑ 7 Port 0: ☑☑☑☑☑☑☑☑)
Privilege: 0x00000008 Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

6) Set Core Clock to 100 or 133 MHz (or your CPU speed).

TIP: The Core Clock: value is set in the STP file. ULINK*pro* detects this value automatically. It uses the Core Clock: value you enter for various timing values to display.

7) Select Trace Enable and EXCTRC.
8) Select Periodic. The window will look like this: →
9) We will use Periodic to test SWV shown below. If you ***are not*** doing this test: do not select Periodic.
10) Leave everything at default settings.
11) Click on OK twice to return to the main screen.
12) The Serial Wire Viewer is now activated.
20) Click on File/Save All to save these settings
21) Serial Wire Viewer (SWV) is enabled and ready to use. You can test it below.

TIP: How to open this window again to modify trace selections:

1) **In Edit mode:** The Target Options icon as already described.
2) **In Debug Mode:** Select Debug/Debug Settings from the main µVision menu.

TIP: You can only enable/disable trace in Edit mode. You can change the rest of the options while in either mode.

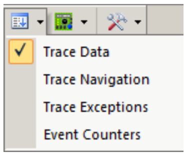
You select various SWV elements in the Trace Configuration window. It is important to note that it is easy to overload the single pin Serial Wire Output (SWO). The general rule is to activate only those features you need.

ULINK*pro* processes SWV information faster than a ULINK2. It uses Manchester encoding on the SWO pin or the Trace Port.

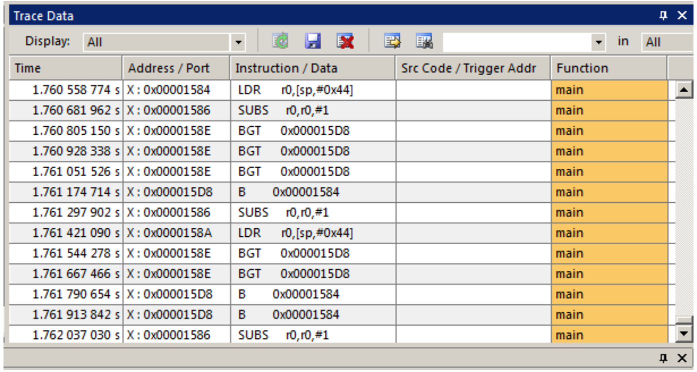
The next page describes in detail the various fields of the Trace Configuration window.

3) Testing Serial Wire Viewer (SWV):

1. Enter Debug mode and click on RUN .
2. Open the Trace Data window by selecting the small arrow beside its icon. →



3. Stop the CPU ULINK*pro* updates the Trace Data window only when the program is stopped.
4. Trace Data will display PC Samples: →
5. Double click on a line, this will be highlighted in the source and Disassembly windows.



Trace Data

Display: All in All

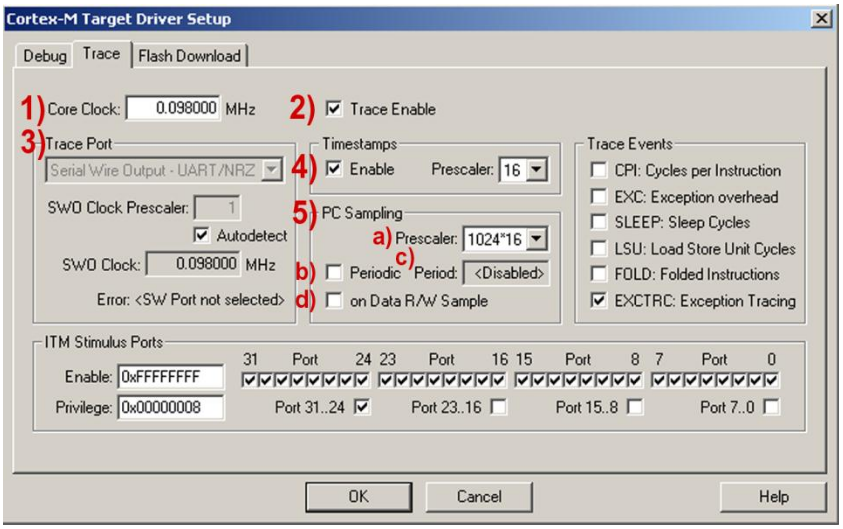
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
1.760 558 774 s	X : 0x00001584	LDR r0,[sp,#0x44]		main
1.760 681 962 s	X : 0x00001586	SUBS r0,r0,#1		main
1.760 805 150 s	X : 0x0000158E	BGT 0x000015D8		main
1.760 928 338 s	X : 0x0000158E	BGT 0x000015D8		main
1.761 051 526 s	X : 0x0000158E	BGT 0x000015D8		main
1.761 174 714 s	X : 0x000015D8	B 0x00001584		main
1.761 297 902 s	X : 0x00001586	SUBS r0,r0,#1		main
1.761 421 090 s	X : 0x0000158A	LDR r0,[sp,#0x44]		main
1.761 544 278 s	X : 0x0000158E	BGT 0x000015D8		main
1.761 667 466 s	X : 0x0000158E	BGT 0x000015D8		main
1.761 790 654 s	X : 0x000015D8	B 0x00001584		main
1.761 913 842 s	X : 0x000015D8	B 0x00001584		main
1.762 037 030 s	X : 0x00001586	SUBS r0,r0,#1		main

Unselect Periodic: We only needed Periodic selected to provide some SWV frames for testing.

6. Select Debug/Debug Settings.
7. Select the Trace tab.
8. Unselect Periodic and click OK.
9. Click OK if a dialog box states Configuration has changed. This occurs when you change trace configuration when the program runs.

4) Trace Configuration Fields: *For reference…*

TIP: ULINK*pro* is similar but with a few features (ETM) added. You can use these instructions for ULINK*pro*.



1) Core Clock: 0.098000 MHz

2) ☑ Trace Enable

3) Trace Port: Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 1

☑ Autodetect

SWO Clock: 0.098000 MHz

Error: <SW Port not selected>

Timestamps: 4) ☑ Enable Prescaler: 16

5) PC Sampling

a) Prescaler: 1024*16

b) □ Periodic c) Period: <Disabled>

d) □ on Data R/W Sample

Trace Events:

□ CPI: Cycles per Instruction

□ EXC: Exception overhead

□ SLEEP: Sleep Cycles

□ LSU: Load Store Unit Cycles

□ FOLD: Folded Instructions

☑ EXCTRC: Exception Tracing

ITM Stimulus Ports

Enable: 0xFFFFFFFF

31 Port 24 23 Port 16 15 Port 8 7 Port 0

☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑

Privilege: 0x00000008

Port 31..24 ☑ Port 23..16 □ Port 15..8 □ Port 7..0 □

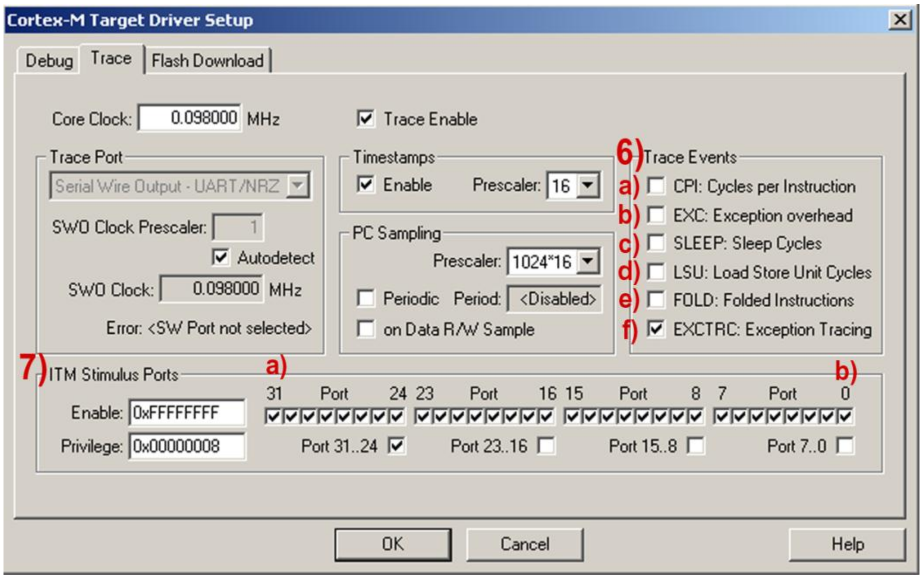
Trace Configuration window Fields 1) through 5).

1) **Core Clock:** The CPU clock speed for SWV. SWO Clock signal is derived from and is a ratio of the Core Clock. This frequency is determined by the .stp file. It must be accurately set for ULINK2/ME and J-Link.
2) **Trace Enable:** Enables SWV and ITM which is essentially everything on this window except Trace Port and ETM. This does not affect the DAP Watch and Memory window display updates.
3) **Trace Port:** Selects the SWO trace output UART or Manchester protocol.
   a. **Serial Wire Output – UART/NRZ:** This is set by default with ULINK2, ULINLK-ME and J-Link.
   b. **Serial Wire Output – Manchester**: Use Manchester encoding. ULINK*pro* only. UART/NRZ encoding is not supported by ULINK*pro*. An error will result when you enter debug mode with ULINK*pro*.
4) **Timestamps:** Enables timestamps and selects a Prescaler. 1 is the default. Selecting a higher value can, but not always lessen SWO overloads. Completely disabling the timestamps can lessen data overruns but can disable other SWV features. It is worth a try if you are having overload problems.
5) **PC Sampling:** Samples the program counter and displays them in the Trace Records window.
   a. **Prescaler** 1024*16 (the default) means every 16,384th PC is displayed. The rest are lost.
   b. **Periodic:** Enables PC Sampling.
   c. **Period:** Automatically derived from Prescaler and Core Clock settings.
   d. **On Data R/W Sample:** Displays the address of the instruction that made a data read or write of a variable entered in the Logic Analyzer in the Trace Records window. This is not connected with PC Sampling.

Note: This window is slightly different for ULINK*pro* and J-link. The resulting trace windows are also different. Currently, the program must be stopped to display the trace records with ULINK*pro* and J-link.

TIP: It is important to ensure the Serial Wire Output (SWO) pin is not overloaded. µVision will alert you when an overflow occurs with a "X" in the Trace Records window or with a "D" or a "O" in the ULINK*pro* Instruction Trace window. µVision easily recovers from these overflows and immediately continues displaying the next available trace frame. Dropped frames are somewhat the normal situation especially with many data reads and/or writes.

TIP: ULINK*pro* can process SWV information much faster than the ULINK2 or ULINK-ME can. This results in fewer dropped frames especially with higher data transfer rates out the SWO pin. ULINK*pro* has the option of collecting information from the 4 bit Trace Port instead of the 1 bit SWO pin. Data overruns are often associated with a fast stream of data reads and writes which are set in the Logic Analyzer. Minimize these issues by displaying only the information you really need.



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 0.098000 MHz

☑ Trace Enable

Trace Port

Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 1

☑ Autodetect

SWO Clock: 0.098000 MHz

Error: <SW Port not selected>

Timestamps

☑ Enable Prescaler: 16

PC Sampling

Prescaler: 1024*16

☐ Periodic Period: <Disabled>

☐ on Data R/W Sample

Trace Events

☐ CPI: Cycles per Instruction

☐ EXC: Exception overhead

☐ SLEEP: Sleep Cycles

☐ LSU: Load Store Unit Cycles

☐ FOLD: Folded Instructions

☑ EXCTRC: Exception Tracing

ITM Stimulus Ports

Enable: 0xFFFFFFFF

31 Port 24 23 Port 16 15 Port 8 7 Port 0

☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑

Privilege: 0x00000008

Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

OK | Cancel | Help

Trace Configuration window Fields 6) through 8)

6) **Trace Events:** Enables various CPU counters. All except EXCTRC are 8 bit counters. Each counter is cumulative and an event is created when this counter overflows every 256 cycles. These values are displayed in the Counter window. The event created when a counter wraps around is displayed in the Trace Records window or the Instruction Trace window (ULINK*pro*).
   Event Counters are updated using the DAP and not SWV. These events are memory mapped and can be read by your program or the µVision Memory window.
   a. **CPI: Cycles per Instruction:** The cumulative number of extra cycles used by each instruction beyond the first one plus any instruction fetch stalls.
   b. **Fold:** Cumulative number of folded instructions**.** This will result from a predicted branch instruction removed and flushed from the pipeline giving a zero cycle execution time.
   c. **Sleep:** Cumulative number of cycles the CPU is in sleep mode. Uses FCLK for timing.
   d. **EXC:** Cumulative cycles CPU spent in exception overhead not including total time spent processing the exception code. Includes stack operations and returns.
   e. **LSU:** Cumulative number of cycles spent in load/store operations beyond the first cycle.
   f. **EXCTRC:** Exception Trace. This is different than the other items in this section. This enables the display of exceptions in the Instruction Trace and Exception windows. It is not a counter. This is a very useful feature and is often used in debugging.
7) **ITM Stimulus Ports**: Enables the thirty two 32 bit registers used to output data in a *printf* type statement to µVision. Port 0 is used for the Debug (printf) Viewer and Port 31 is used for the Keil RTX real-time kernel awareness window. Only Ports 0 and 31 are currently implemented in µVision and should normally be checked. Ports 1 through 30 are not currently implemented and are Don't Care.
   a. **Port 31:** Enables the ITM port used for the RTX Viewer.
   b. **Port 0:** Enables the ITM port used for the Debug (printf) Viewer. A small amount of instrumentation code is needed in your project. See Debug (printf) Viewer: page 18 for information on using this feature.

For more Information on Configuring Trace:

For ULINK2 see www.keil.com/support/man/docs/ulink2/ulink2_ctx_trace.htm

For ULINK*pro* see www.keil.com/support/man/docs/ulinkpro/ulinkpro_ctx_trace.htm

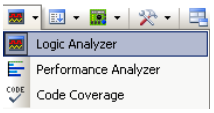
For Segger J-Link see www.keil.com/support/man/docs/jlink/jLink_cortexTrace.htm

Part 5) Examples Using Serial Wire Viewer (SWV):

1) Logic Analyzer Window (LA): Updated in Real-time !

The Logic Analyzer (LA) displays up to four variables in a graphical format. When a variable is placed in the LA, it is also displayed in the Trace Records window. The LA needs SWV properly configured for its operation.

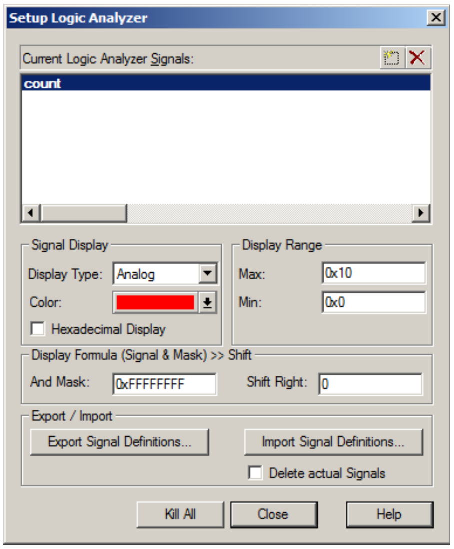
1. µVision must be in Debug mode. The program can be running or not.
2. Find the global variable **`count`** you created in main.c.
3. Right-click on it and select Add 'count' to… and select Logic Analyzer.
4. The LA will open up with **`count`** displayed.



What an error or no display of count: The most common problem of not entering a variable in the LA is SWV is not configured or the Core Clock: setting is incorrect. In both cases, SWV is not operating and the LA uses /needs SWV. If Core Clock: is not set correctly with ULINK*pro*, SWV and the LA will still work but the timestamps displayed will be incorrect.

TIP: You can also drag and drop the variable into the LA window or select Setup… and use the Insert icon .

5. In the LA, click on Setup… This window opens up:
6. Select the Display Range max = 0x10 and Min = 0. Click on Close.
7. Click on RUN if the program is not already running. The LA window will be updating the variable count as shown below:
8. Note the peak when count is high. This is useful to determine problems when a variable is tested at the wrong time in your program.
9. If necessary, click on Zoom: Out for an appropriate X axis. Try around 0.5 seconds.
10. Add **`&count`** to the Memory window if it is not already there.
11. In the Memory window: change the value of **`count`** to various values to change LA values. Right click on the memory address you want to change and select Modify …. You will see the effect in the LA.
12. Select Signal Info and Cursor on the right of the LA.
13. Select Stop in Update Screen. This stops the LA but not your program. This is a very useful feature. You can also stop your program.
14. Hover the mouse over the waveform. By clicking and setting the cursor, you can determine the various timings of this program.



Setup Logic Analyzer

Current Logic Analyzer Signals:
count

Signal Display
Display Type: Analog
Color:
☐ Hexadecimal Display

Display Range
Max: 0x10
Min: 0x0

Display Formula (Signal & Mask) >> Shift
And Mask: 0xFFFFFFFF
Shift Right: 0

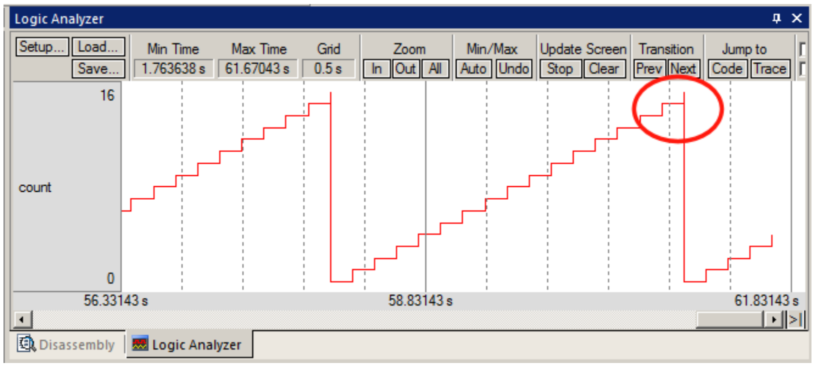
Export / Import

☐ Delete actual Signals

15. In my example, each step is 0.181 seconds long. See the bottom screen: The actual value depends on the board and STP file you are using, hence the CPU clock speed. This is 133 MHz.

TIP: You can enter up to four variables in the LA window.

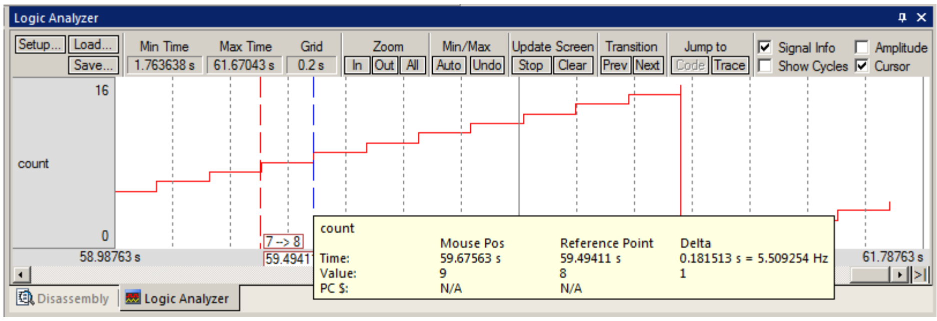
Watchpoints also use part of this CoreSight debug technology. There are four comparators that must be shared.



TIP: Raw addresses can also be entered into the Logic Analyzer. An example is:

***((unsigned long *)0x20000000)**

This can be useful to detect any write operations to a peripheral.



count

	Mouse Pos	Reference Point	Delta
Time:	59.67563 s	59.49411 s	0.181513 s = 5.509254 Hz
Value:	9	8	1
PC $:	N/A	N/A	

Logic Analyzer Notes:

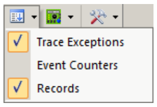
1. The Logic Analyzer displays up to four variables in a graphical format.
2. SWV is used to obtain the data displayed. SWV must be configured and it comes out the Serial Wire Output pin. The ULINK*pro* has the option of sending this data out the 4 bit Trace Port and this is more efficient.
3. You will not be able to enter variables if the SWV Trace configuration is not enabled and configured properly. The Core Clock needs to be accurately set except for the ULINK*pro*. It uses Core Clock: for timing display values.
4. Each variable will have its write operation displayed in the Trace Records window. (not with J-Link)
5. You must enter a variable name. Raw memory addresses are allowed. See the TIP: below.
6. You might have to fully qualify a variable. An example is \Blinky\AD_dbg.
7. Number 1 reason you can't enter a variable is the Trace not properly configured. Test SWV using PC Samples.
8. Number 2 reason is too many SWV items are enabled and this overloads the SWO pin. It helps to use the Trace Port.
9. You can drag and drop variables into the LA. Variables must be global, static or a structure.
10. The LA is updated at a fixed rate of approximately 500 msec.

TIP: Raw addresses can also be entered into the Logic Analyzer. An example is: ***((unsigned long *)0x20000000)**

2) Data Writes in the Trace Records for the global variable count:

When a variable is entered into the Logic Analyzer, the associated data writes are displayed in the Trace Records window. This is the method used to enter writes in the Trace Records window. Reads are not currently implemented in µVision.

1. Select Debug/Debug Settings and then the Trace tab. Unselect EXCTRC. Select on Data R/W Sample.
2. Click on OK twice to return to the main menu.
3. Click on RUN.
4. Open the Trace Records window with View/Trace/Records or the pull-down menu:



5. The window below opens up. Data Writes to variable count (at 0x2000_0028 in this case) are displayed.
6. If Trace Records is already open and full of SysTick frames, double-click anywhere inside it to clear it.
7. Note the data values are incrementing.
8. Click on RUN.

TIP: The ULINK*pro* will display a different window and you must stop the program to see the writes.

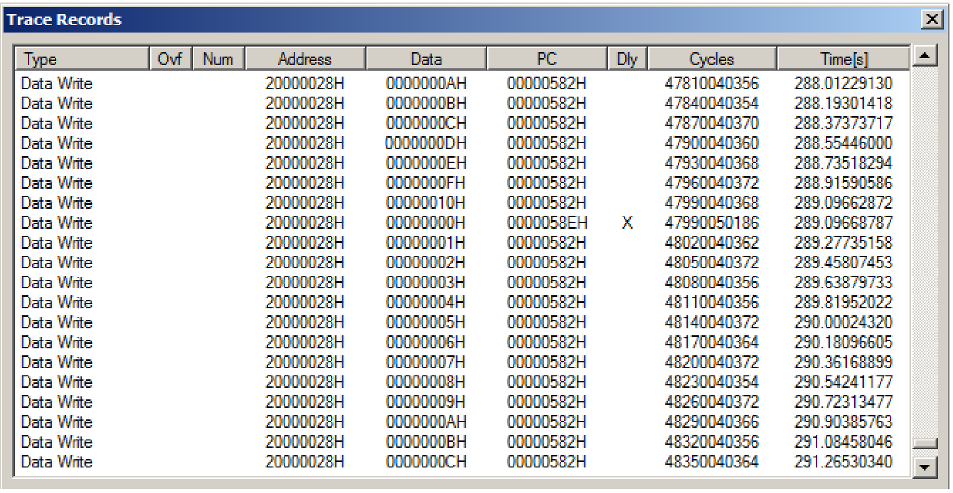
TIP: Double-clicking on a line in the ULINK*pro* Trace Data window will highlight this instruction in the Disassembly and Source windows.

Explanation:

1st Line: Data Write to `count` address 0x2000_0028 the data value 0x0A by the instruction located at 0x0000_0582.

The PC column was added when you selected on Data R/W sample in the Trace Configuration. It is optional.

With the ULINK*pro* Trace Data window, double-clicking on a data write will highlight that instruction in the Disassembly and source windows.



Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000028H	0000000AH	00000582H		47810040356	288.01229130
Data Write			20000028H	0000000BH	00000582H		47840040354	288.19301418
Data Write			20000028H	0000000CH	00000582H		47870040370	288.37373717
Data Write			20000028H	0000000DH	00000582H		47900040360	288.55446000
Data Write			20000028H	0000000EH	00000582H		47930040368	288.73518294
Data Write			20000028H	0000000FH	00000582H		47960040372	288.91590586
Data Write			20000028H	00000010H	00000582H		47990040368	289.09662872
Data Write			20000028H	00000000H	0000058EH	X	47990050186	289.09668787
Data Write			20000028H	00000001H	00000582H		48020040362	289.27735158
Data Write			20000028H	00000002H	00000582H		48050040372	289.45807453
Data Write			20000028H	00000003H	00000582H		48080040356	289.63879733
Data Write			20000028H	00000004H	00000582H		48110040356	289.81952022
Data Write			20000028H	00000005H	00000582H		48140040372	290.00024320
Data Write			20000028H	00000006H	00000582H		48170040364	290.18096605
Data Write			20000028H	00000007H	00000582H		48200040372	290.36168899
Data Write			20000028H	00000008H	00000582H		48230040354	290.54241177
Data Write			20000028H	00000009H	00000582H		48260040372	290.72313477
Data Write			20000028H	0000000AH	00000582H		48290040366	290.90385763
Data Write			20000028H	0000000BH	00000582H		48320040356	291.08458046
Data Write			20000028H	0000000CH	00000582H		48350040364	291.26530340

The time these events occurred is timed by both CPU cycles and in seconds (Time(s)).

The X in the DLY column indicates the timestamp is delayed from its true value because of overflow.

These entries are caused by the associated data write. If there are too many such writes and are causing too many overloads, you might have to sample the variable and send this to the LA. µVision and CoreSight recover nicely from overloads.

3) Watchpoints: (also known as Access Points) *Does not actually use SWV.*

This is an excellent opportunity to show how the Watchpoints work in Keil µVision.

Watchpoints are also known as **A**ccess Breaks and are indicated by an **(A** in the Breakpoints window shown below. They are set in the Breakpoints Window. Watchpoints can be thought of as conditional breakpoints when compared to an Execution breakpoint. Execution breakpoints stop the CPU when a specified instruction is fetched (but not executed). Watchpoints stop the CPU when a specified data access occurs and any specified expression becomes true. There might be some Program Counter skid. This is normal. Watchpoints are useful for locating read and write operations you do not expect.

SmartFusion2 has 2 Watchpoints and currently µVision can activate one. A Watchpoint must be configured in Debug mode and the program can be running. They are shared with the Logic Analyzer.

TIP: You can configure one Watchpoint if only one or two variables are listed in the LA. If you have three or four variables in the LA you are not able to configure a Watchpoint. This is a CoreSight Cortex-M3 limitation.

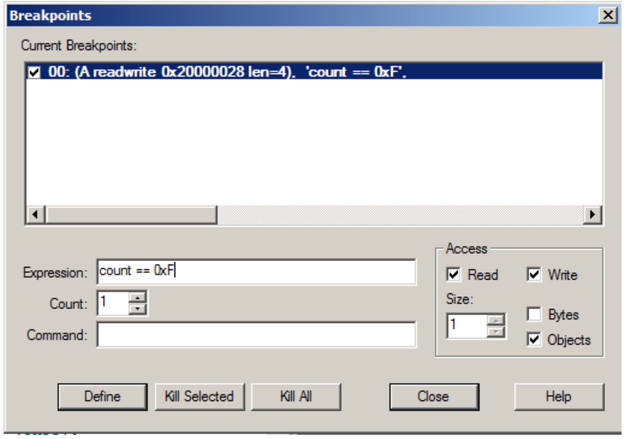
How to set a Watchpoint:

1. µVision must be in Debug mode and the program can be running or halted.
2. Open the Breakpoint window through View/Breakpoints or Ctrl-B. The Breakpoints window below opens up empty.
3. Watchpoints are created in the bottom half and are then defined and listed in the top half.
4. In the Expression: box: enter **`count == 0xF`**
5. Click both Read and Write (for simplicity).
6. Click on Define and the Watchpoint moves to the upper half as shown below in this composite screen: Click on Close.
7. This window says the program will be halted if the value of 0x0F is written to or read ***once*** from the variable **`count`**. (Count = 1) The other entries are the default settings.

TIP: Note: The count: box is not our variable **`count`**. They are quite different.

8. Set **`count`** to less than 0x0F in the Watch window otherwise it will break immediately.
9. The Trace Records window must be open. Double-click on it to clear it.
10. Click on RUN and wait for the program to stop.
11. Scroll to the bottom of the Trace Records window and the data write (0x0F) is displayed as shown below: This is the Trace Data from a ULINK*pro* but the ULINK2/ME Trace Records is similar.
12. Delete (kill) this Watchpoint when you are done otherwise it will interfere with your other examples.

TIP: You can easily use this to find the instruction causing a spurious write to a specified memory location.



Current Breakpoints:

☑ 00: (A readwrite 0x20000028 len=4), 'count == 0xF',

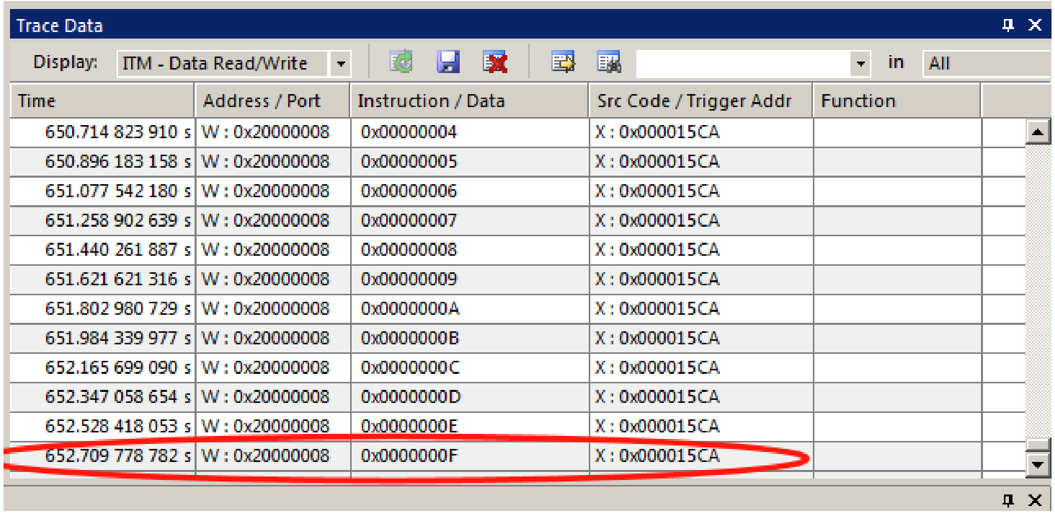
Expression: count == 0xF

Count: 1

Command:

Access: ☑ Read ☑ Write

Size: 1 □ Bytes ☑ Objects



Display: ITM - Data Read/Write in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
650.714 823 910 s	W : 0x20000008	0x00000004	X : 0x000015CA	
650.896 183 158 s	W : 0x20000008	0x00000005	X : 0x000015CA	
651.077 542 180 s	W : 0x20000008	0x00000006	X : 0x000015CA	
651.258 902 639 s	W : 0x20000008	0x00000007	X : 0x000015CA	
651.440 261 887 s	W : 0x20000008	0x00000008	X : 0x000015CA	
651.621 621 316 s	W : 0x20000008	0x00000009	X : 0x000015CA	
651.802 980 729 s	W : 0x20000008	0x0000000A	X : 0x000015CA	
651.984 339 977 s	W : 0x20000008	0x0000000B	X : 0x000015CA	
652.165 699 090 s	W : 0x20000008	0x0000000C	X : 0x000015CA	
652.347 058 654 s	W : 0x20000008	0x0000000D	X : 0x000015CA	
652.528 418 053 s	W : 0x20000008	0x0000000E	X : 0x000015CA	
652.709 778 782 s	W : 0x20000008	0x0000000F	X : 0x000015CA	

4) Exception Tracing:

Serial Wire Viewer displays exceptions and interrupts in real-time without stealing any CPU cycles or needing instrumentation code. Note that ARM interrupts are a subset of exceptions.

Cortex-M processors have a timer called SysTick. It is normally used to an RTOS but it can be used for other uses. We will activate SysTick in main.c and use SWV to trace it. This is using CMSIS compliant software.

Add source to configure and enable the SysTick Timer:

1. Stop the CPU and exit Debug mode.
2. In main.c near line 23, declare the variable Syscounter:
3. Before main(), add this code starting at near line 32:
4. In main(), just after MSS_GPIO_init();, add this line:

```
unsigned int Syscounter = 0;
```

```
/*----------------------------
  SysTick IRQ Handler
 *----------------------------*/
     void SysTick_Handler (void) {
       static uint32_t ticks;
       Syscounter=!Syscounter;
       ticks++;
           if (ticks > 10) {
             ticks = 0;
           }
       }
```

```
SysTick_Config(133000000/100);
```

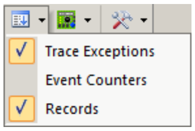
5. Select File/Save All or click .

Compile main.c, Configure the SWV Trace and RUN:

6. Compile the source files by clicking on the Rebuild icon. .
7. Select Options for Target or ALT-F7.
8. Select the Trace tab.
9. Enable EXCTRC: Unselect Periodic and on R/W Sampling. Click on OK twice to return.
10. Enter Debug mode and click on RUN .

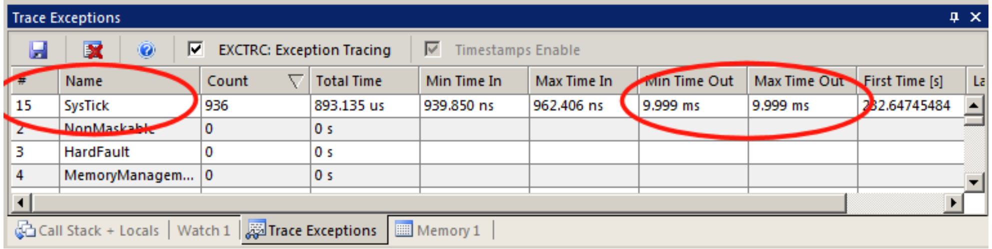
Exceptions Trace Window:

11. Open the Exception Trace and Trace Records windows.



☑ Trace Exceptions
Event Counters
☑ Records

12. The Trace Exceptions window will display the SysTick timer as shown below. Click the Clear icon:
13. Scroll up and down and you can see the listing of all potential exceptions available in the NVIC. ExtIRQ are peripheral exceptions. Consult your device datasheet to determine what they are.



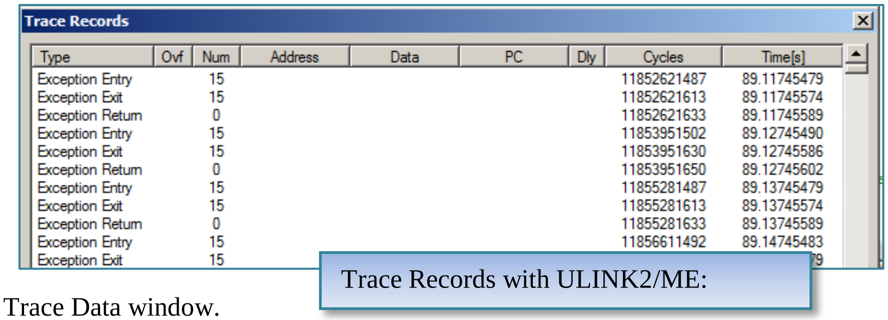
Trace Exceptions

☑ EXCTRC: Exception Tracing ☑ Timestamps Enable

#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	La
15	SysTick	936	893.135 us	939.850 ns	962.406 ns	9.999 ms	9.999 ms	232.64745484	
2	NonMaskable	0	0 s						
3	HardFault	0	0 s						
4	MemoryManagem...	0	0 s						

Trace Records Window:

1. A ULINK2 Trace Records window opens up and displays the Exception 15 which is the SysTick timer.
2. Right click on the Trace Records window and filter out Exceptions frames. Only Data writes are now left
3. Double-click inside this window to clear it.



Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					11852621487	89.11745479
Exception Exit		15					11852621613	89.11745574
Exception Return		0					11852621633	89.11745589
Exception Entry		15					11853951502	89.12745490
Exception Exit		15					11853951630	89.12745586
Exception Return		0					11853951650	89.12745602
Exception Entry		15					11855281487	89.13745479
Exception Exit		15					11855281613	89.13745574
Exception Return		0					11855281633	89.13745589
Exception Entry		15					11856611492	89.14745483
Exception Exit		15						79

Trace Records with ULINK2/ME:

Explanation of Exception frames:

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned including any Cortex-M tail-chaining.

4. When using a ULINK*pro*, you must stop the program to update the Trace Data window.

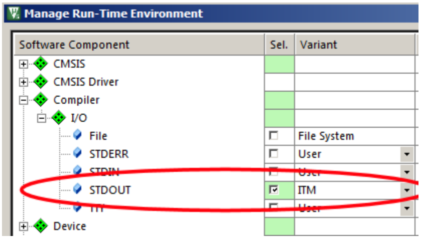
5) Debug (printf) Viewer: SWV is required:

ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in the Debug (printf) Viewer window. It is possible to send ITM data to a file: www.keil.com/appnotes/docs/apnt_240.asp.

1. Stop the program and exit Debug mode .

Add STDOUT File (retarget_io.c):

1. Open the Manage Run-Time Environment window (MRTE) .
2. Expand Compiler and I/O as shown here:
3. Select STDOUT and ITM. This adds the file retarget_io.c to the project.
4. Ensure all blocks are green and click OK to close the MRTE.



Manage Run-Time Environment

Software Component	Sel.	Variant
CMSIS		
CMSIS Driver		
Compiler		
I/O		
File	□	File System
STDERR	□	User
STDIN	□	User
STDOUT	☑	ITM
TTY	□	User
Device		

Add printf and #include <stdio.h> to main.c:

1. In main.c near line 14, add this line: #include <stdio.h>
2. Near the end of the for(;;) loop near line 143, add this line: if (gpio_pattern == 0) printf("LEDs are ON\n");
3. Near the end of the for(;;) loop near line 144, add this line: if (gpio_pattern != 0) printf("LEDs are OFF\n");

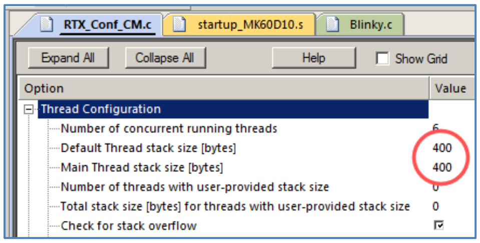
Enable Microlib and Configure Serial Wire Viewer:

1. Select Options for Target or ALT-F7. Select the Target tab.
2. Select Use MicroLIB. If you don't want to use MicroLIB add 200 bytes to the Heap in startup_M2Sxxx.s.
3. Select the Debug tab. Select Settings and then the Trace tab.
4. Unselect On Data R/W Sample and ITM Port 31. (this is to help not overload the SWO pin so this step is optional)
5. Select ITM Port 0. ITM Stimulus Port "0" enables the Debug (printf) Viewer. All ports 1 through 30 are unused.
6. Click OK twice to return to the main µVision menu.

Increasing RTX stack size: This step is *only* if you are using RTX: You are not using RTX so skip these three steps !

When a printf statement is added to a thread, you must increase the RTX stack size as follows:

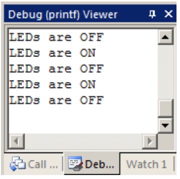
1. Select the RTX_Conf_CM.C tab. Click on the Configuration Wizard tab at the bottom of this window.
2. Set the Default and Main stack size to 400 bytes as shown here:
3. In general, if you experience trouble with RTX operation, try increasing the number of threads and size of the RTX stack. This stack is not the same as the CPU system stack as referenced by the Stack Pointer but it is still in RAM.



Option	Value
Thread Configuration	
Number of concurrent running threads	6
Default Thread stack size [bytes]	400
Main Thread stack size [bytes]	400
Number of threads with user-provided stack size	0
Total stack size [bytes] for threads with user-provided stack size	0
Check for stack overflow	☑

Compile and Run the Project:

1. Select File/Save All or click .
2. Rebuild the source files and enter Debug mode .
3. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
4. In the Debug (printf) Viewer you will see the printf statements appear.
5. Right click on the Debug window and select Mixed Hex ASCII mode. Note other settings.



```
LEDs are OFF
LEDs are ON
LEDs are OFF
LEDs are ON
LEDs are OFF
```

Obtaining a character typed into the Debug printf Viewer window from your keyboard:

It is possible for your program to input characters from a keyboard with the function ITM_ReceiveChar in core.CM3.h.

This is documented here: www.keil.com/pack/doc/CMSIS/Core/html/group__ITM__Debug__gr.html

A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer.

TIP: ITM_SendChar is a useful function you can use to send characters out ITM. It is found in core.CM3.h.

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enable only those SWV features that you need. If you need high performance SWV, a ULINK*pro* using 4 bit Trace Port is fastest.

6) Overloading the SWO pin:

The Serial Wire Output pin (SWO) is just that – a single bit high speed line that is expected to handle all the data sent to it. A Cortex-M3 can read and write data much faster than the SWO can process.

Overloads have an X in the Ovf or Dly (overflow or Delay) column in the Trace Records window: but not always. Sometimes the frames have corrupt, missing or have wrong information and this is easy to spot.

ITM frames with IDs other than 0 or 31 are a sure sign of either overloading or an incorrect Core Clock setting.

Here are some hints on reducing traffic to minimize SWO overruns:

1. Reduce the number of variables in the Logic Analyzer.
2. Reduce the number of items selected in the Trace Configuration window: These include:
   a. PC Sampling. Turn this off or increase the Prescaler to reduce the sampling rate.
   b. On Data R/W Sample.
   c. EXCTRC: Exception Tracing
   d. Any Trace Events. (the Counters)
   e. ITM Stimulus Ports 31 and 0. (Viewer and RTX Awareness respectively)
3. Set the Timestamp Prescaler to 1.
4. Turn the timestamps off. Sometimes this will turn the trace off though. But it is worth a try.
5. Lower the rate of reads, writes and exception calls in your software.
6. Turn off the SysTick timer in your software if you are not using it.
7. Reduce the Range setting in the Logic Analyzer. Sometimes a very large number overloads µVision.
8. The General Rule is: activate only those SWV features you need.
9. Using a faster computer can help in some cases. SWV can create a great deal of information that must be processed quickly. This is especially true where the Logic Analyzer window is concerned.
10. We will be making continuous improvements to µVision to provide you with a better SWV experience.
11. Use a ULINK*pro*. It uses the SWO pin in Manchester mode which is faster than the UART mode used in ULINK2 or J-Link. It can also access SWV via the 4 bit ETM Trace Port and this is even faster.

TIP: If you are seriously overloading the SWO, there will usually be at least some data coming out. µVision recovers easily and painlessly from data overruns. You are made aware when frames are lost.

7) Top Seven Reasons why you can't get SWV working:

Symptoms can range from an inability to enter a known variable in the Logic Analyzer, corrupted trace frames or no information at all.

1. Core Clock is wrong. This is the number 1 reason. Normally, the SWO speed is derived from the Core Clock speed. Common speeds at this time with various projects have been 50, 100 and 133 MHz. Recall the ULINK*pro* calculates Core Clock: automatically. It uses the figure you enter for certain timing displays.
2. View/Periodic Update is not enabled. Your windows update only when the processor is stopped.
3. Trace Enable is not checked.
4. Using JTAG instead of SWD. µVision will complain about this if you try it. SWV needs SWD (or SW) selected if the UART SWO pin is selected. If using a ULINK*pro* and the Trace Port is used, JTAG is permitted.
5. SWD speed too high. Try a slower speed. This is not usually the problem.
6. You have selected too many SWV items and it is seriously overloaded. This effect often manifests itself when you stop the program and the Trace Records window is updated or a variable in the Logic Analyzer is does not change.

Part 6: DSP Example:

1) DSP SINE example using ARM CMSIS-DSP Libraries:

ARM CMSIS-DSP libraries are offered for ARM Cortex-M processors.

See www.arm.com/cmsis and forums.arm.com for more information. CMSIS is an acronym for Cortex Microcontroller Software Interface Standard.

This example creates a sine wave, then a second to act as noise, which are then added together (disturbed), and then the noise is filtered out (filtered). The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer and Watch 1.

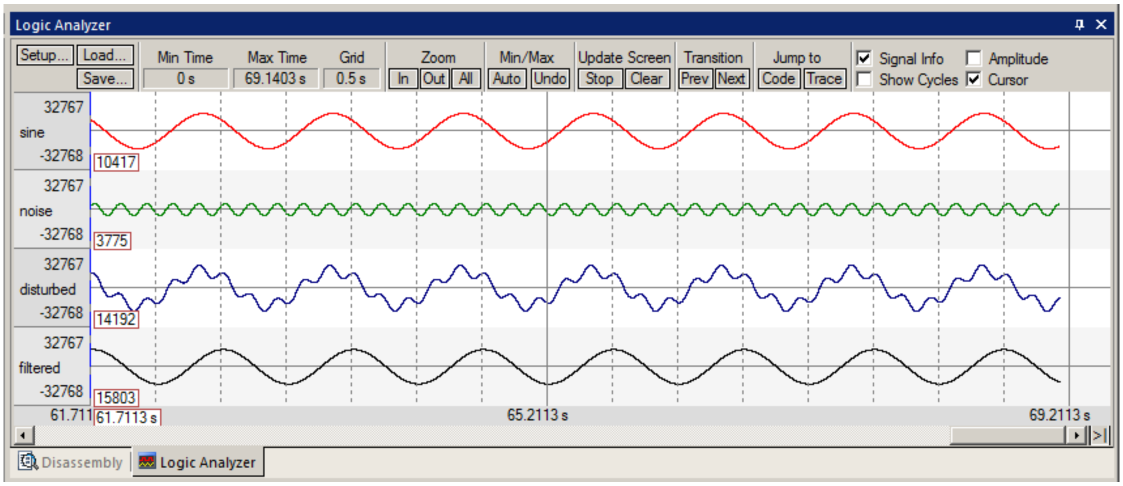
This example incorporates the Keil RTOS RTX. RTX is available free with a BSD type license. RTX source code is provided.

This example file was installed on page xyz. It is provided with this document.

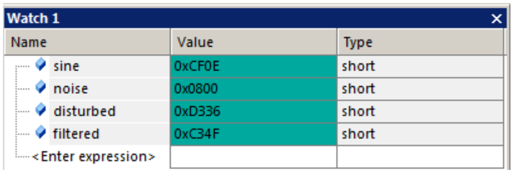
1. Open the project file sine: C:\00MDK\Boards\Microsemi\M2S090TS\DSP\sine.uvprojx
2. Compile the source files by clicking on the Rebuild icon. .
3. Program the SmartFusion2 eNVM flash by clicking on the Load icon: Progress is indicated in bottom left corner.
4. Enter Debug mode by clicking on the Debug icon. the SmartFusion 2 eNVM flash will be programmed.
5. Click on the RUN icon. Open the Logic Analyzer window if necessary:
6. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom Out for an appropriate display. Displayed are 4 global variables: **`sine`**, **`noise`**, **`disturbed`** and **`filtered`**.

TIP: If one or two variables displays no waveform or is severely distorted, disable the ITM Stimulus Port 31 in the Trace Config window. The SWO pin is probably overloaded. ULINKpro displays SWV the best.

7. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.



8. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below: They are pre-configured in this project.
9. Open the Trace Records window and the Data Writes to the four variables are displayed. Not with J-Link.
10. If you are using a ULINKpro, you must stop the program to refresh the Trace Data window.
11. Leave the program running.
12. Close the Trace Records window.



Watch 1

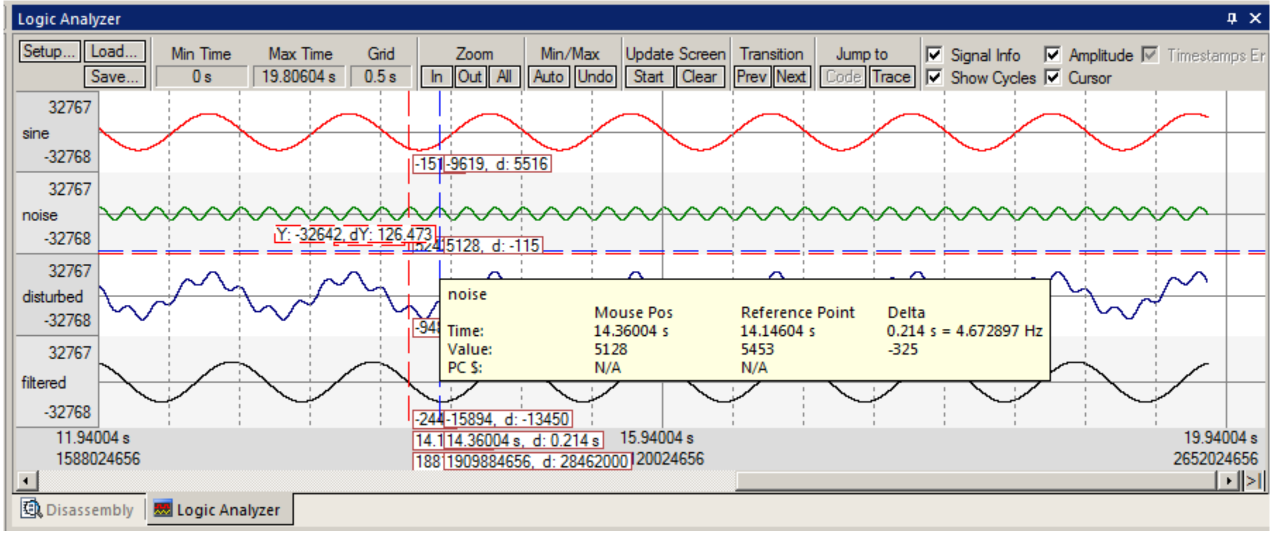
Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short
<Enter expression>		

CMSIS-DSP documentation: C:\Keil_v5\ARM\PACK\ARM\CMSIS\5.0.0-Beta4\CMSIS\Documentation\DSP\html.
CMSIS-DSP Libraries: C:\Keil_v5\ARM\PACK\ARM\CMSIS\5.0.0-Beta4\CMSIS\Lib\
CMSIS-DSP source code: C:\Keil_v5\ARM\PACK\ARM\CMSIS\5.0.0-Beta4\CMSIS\DSP_Lib\

A DSP library is easily added to your project by selecting it in the Manage Run-Time Environment window (MRTE) .

2) Signal Timings in Logic Analyzer (LA):

1. In the LA window, select Signal Info, Show Cycles, Amplitude and Cursor.
2. Click on STOP in the Update Screen box. You could also stop the program but leave it running in this case.
3. Click somewhere in the LA to set a reference cursor line.
4. Note as you move the cursor various timing information is displayed as shown below:



noise

	Mouse Pos	Reference Point	Delta
Time:	14.36004 s	14.14604 s	0.214 s = 4.672897 Hz
Value:	5128	5453	-325
PC $:	N/A	N/A	

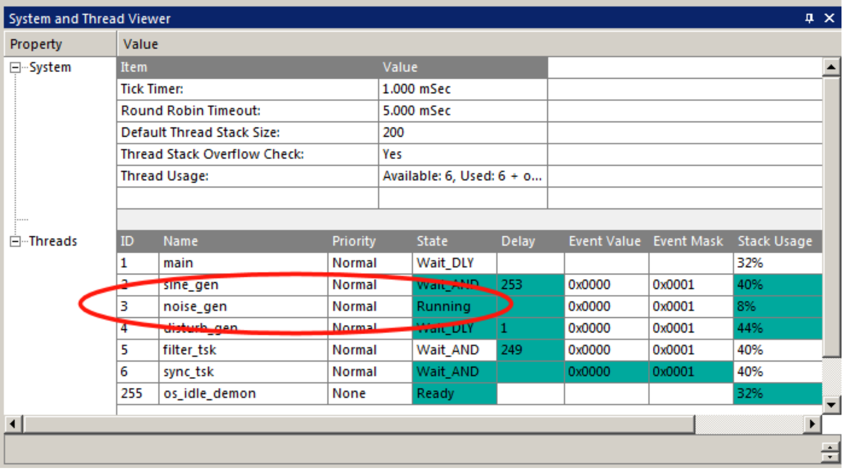
3) RTX Tasks and System Awareness window:

5. Click on Start in the Update Screen box to resume the collection of data.
6. Open Debug/OS Support and select RTX Tasks and System. A window similar to below opens up. You probably have to click on its header and drag it into the middle of the screen.
7. Note this window does not update: nearly all the processor time is spent in the idle daemon: it shows it is Running. The processor spends relatively little time in other tasks. You will see this illustrated clearly on the next page.
8. Set a breakpoint in one of the four tasks in DirtyFilter.c by clicking in the left margin on a grey area.
9. Click on Run and the program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the noise_gen task:
10. Clearly you can see that noise_gen was running when the breakpoint was activated.
11. Remove the breakpoint.

TIP: You can set hardware breakpoints while the program is running.

TIP: Recall this window uses the CoreSight DAP read and write technology to update this window. Serial Wire Viewer is not used and is not required to be activated for this window to display and be updated.

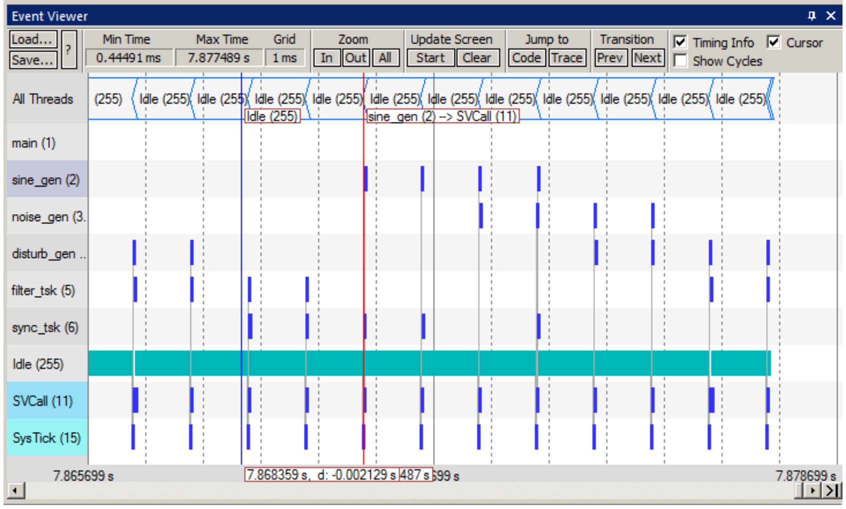
The Event Viewer does use SWV and this is demonstrated on the next page.



Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 6, Used: 6 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	main	Normal	Wait_DLY				32%
2	sine_gen	Normal	Wait_AND	253	0x0000	0x0001	40%
3	noise_gen	Normal	Running		0x0000	0x0001	8%
4	disturb_gen	Normal	Wait_DLY	1	0x0000	0x0001	44%
5	filter_tsk	Normal	Wait_AND	249	0x0000	0x0001	40%
6	sync_tsk	Normal	Wait_AND		0x0000	0x0001	40%
255	os_idle_demon	None	Ready				32%

4) RTX Event Viewer:



1. Stop the program.
2. Select Debug/Debug Settings.
3. Click on the Trace tab.
4. Enable ITM Stimulus Port 31. Event Viewer uses this to collect its information.
5. Click OK twice.
6. Exit and re-enter Debug mode to refresh the Trace Configuration.
7. Click on RUN.
8. Open Debug/OS Support and select Event Viewer. This window opens:
9. Thread 1 is main(1) and is found in DirtyFilter.c near line 169. It runs some RTX initialization code at the beginning. Scroll the Event Viewer to the left to view when main() ran.

Interrupt Handler Timing: Note the SVCall and SysTick entries at the bottom. These display the execution times of any exception handlers active. This could be DMA, CAN or other peripheral. This is useful to optimize your exception handler times. This feature is available only with a ULINK*pro* and RTX.

TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the LA to free up some SWO or Trace Port bandwidth. It depends on how much data is sent to the ports.

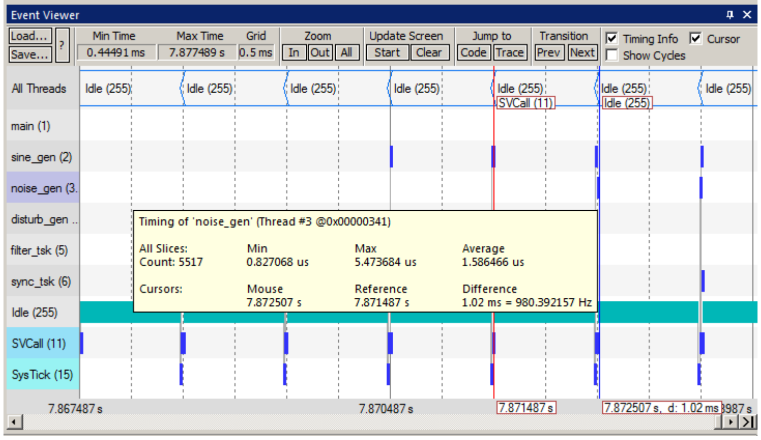
To remove the variables in the Logic Analyzer (LA):

Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might occur.

ULINK*pro* is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows. ULINK*pro* uses the faster Manchester format than the slower UART mode that ULINK2/ME and J-Link uses.

ULINK*pro* can also use the 4 bit Trace Port for faster operation for SWV. Trace Port use is mandatory for ETM trace.

1. Note on the Y axis each of the 5 running tasks plus the idle daemon. Each bar is an active task and shows you what task is running, when and for how long.
2. Click Stop in the Update Screen box.
3. Click on Zoom In so three or four tasks are displayed.
4. Select Cursor. Position the cursor over one set of bars and click once. A red line is set here:
5. Move your cursor to the right over the next set and total time and difference are displayed.
6. Note, since you enabled Show Cycles, the total cycles and difference is also shown.



```
Timing of 'noise_gen' (Thread #3 @0x00000341)

All Slices:     Min             Max             Average
Count: 5517     0.827068 us     5.473684 us     1.586466 us

Cursors:        Mouse           Reference       Difference
                7.872507 s      7.871487 s      1.02 ms = 980.392157 Hz
```

The 1 msec shown is the SysTick timer value. This value is set in RTX_Conf_CM.c. The next page describes how to change this.

TIP: ITM Port 31enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port if you are not using the Event Viewer and this is normally a good idea if you are running RTX with high SWO use.

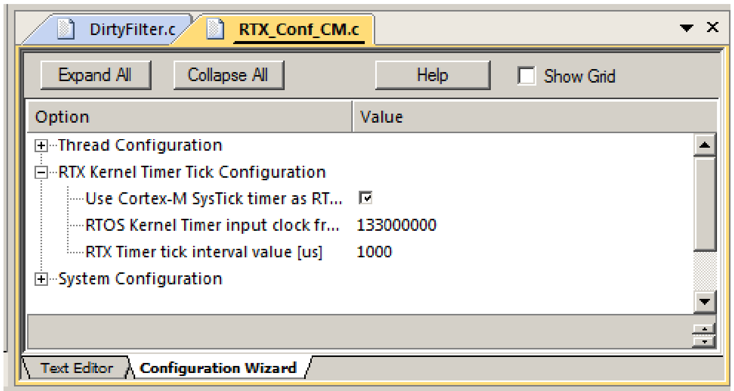
Even if the Event Viewer is closed, the data is still being sent out the SWO pin or the Trace Port. This can contribute to SWV overloading.

If you have consistent SWV overloading, a ULINK*pro* is a good investment.

How to Change the SysTick Timer in RTX:

1. Stop the processor and exit debug mode.
2. Open the file RTX_Conf_CM.c from the Project window.
1. Select the Configuration Wizard tab at the bottom of the window. This scripting language is shown in the Text Editor as comments starting such as a </h> or <i>.
2. You can use this in your own source code files. See www.keil.com/support/docs/2735.htm for instructions.
3. This window opens up. Expand SysTick Timer Configuration.
4. Note the Timer tick value is 1,000 µs or 1 ms.
5. This will be the Round Robin thread switching time period.
6. The RTOS Kernel Timer is set to 133 MHz for this project.
7. If you change the CPU frequency in your project and use RTX, you must enter the new frequency here.

TIP: The SysTick is a dedicated timer on Cortex-M processors that is used to switch tasks in an RTOS. It does this by generating an Exception 15 periodically every 1,000 µs or to what you set it to. You can view these exceptions in the Trace Records window by enabling EXCTRC in the Trace Configuration window.



Option	Value
⊞ Thread Configuration	
⊟ RTX Kernel Timer Tick Configuration	
Use Cortex-M SysTick timer as RT...	☑
RTOS Kernel Timer input clock fr...	133000000
RTX Timer tick interval value [us]	1000
⊞ System Configuration	

RTX Documentation:

MDK 5 Getting Started Guide: Obtain this free, useful book here: www.keil.com/gsg/.
General CMSIS Documentation is found here: www.keil.com/pack/doc/CMSIS/General/html/
RTX Documents: www.keil.com/pack/doc/CMSIS/RTOS/html/

RTX is CMSIS-RTOS compliant.
RTX currently has a BSD license. There are ports for Keil/ARM and GCC. Other companies also have their own ports.
RTX 2 will soon have an Apache 2.0 license. It is hosted on GitHub: https://github.com/ARM-software/CMSIS_5

Part 7) ETM Instruction Trace with ULINK*pro*:

1) Configuring ULINK*pro* ETM Instruction Trace: A Keil ULINK*pro* is needed for ETM:

A ULINK*pro* is required for ETM operation. A 20 pin CoreSight 20-Pin Cortex Debug + ETM Connector is needed: the M2S090TS-EVAL-KIT has this connector. See www.keil.com/coresight/coresight-connectors/ for debug connectors.

ETM provides serious debugging power as shown on the next few pages. It is worth the modest added cost.

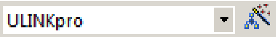
The ETM signals consist of 4 data bits plus a clock. ETM signals runs at high speed so appropriate PC board design techniques should be used for custom boards. At a CPU speed of 133 MHz, the Smartfusion2 ETM clock runs at ½ of this.

Connecting and configuring the physical connection to the Microsemi M2S090TS-EVAL-KIT:

1. Connect a ULINK*pro* to the J9 Trace DBG connector. See page 1 for a photo.
2. Set J8 JTAG_SEL to L position. Power the board and ULINK*pro*.
3. Start µVision and open Blinky.uvprojx in C:\00MDK\Boards\Microsemi\M2S090TS\Blinky\
4. The Blinky example is preconfigured for ULINK*pro* in UART Manchester mode. We will now add ETM support.

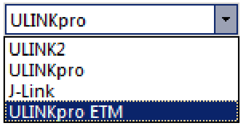
Create a new Target Selection for ULINK*pro*:

1. Select ULINKpro in the Target Option select box to use as a template:



ULINKpro

2. Select Project/Manage/Project Items… or select:
3. In the Project Targets area, select NEW or press your keyboard INSERT key.
4. Enter **ULINKpro ETM** and press Enter. Click OK to close this window.
5. In the Target Selector menu, select the ULINKpro ETM selection you just made:

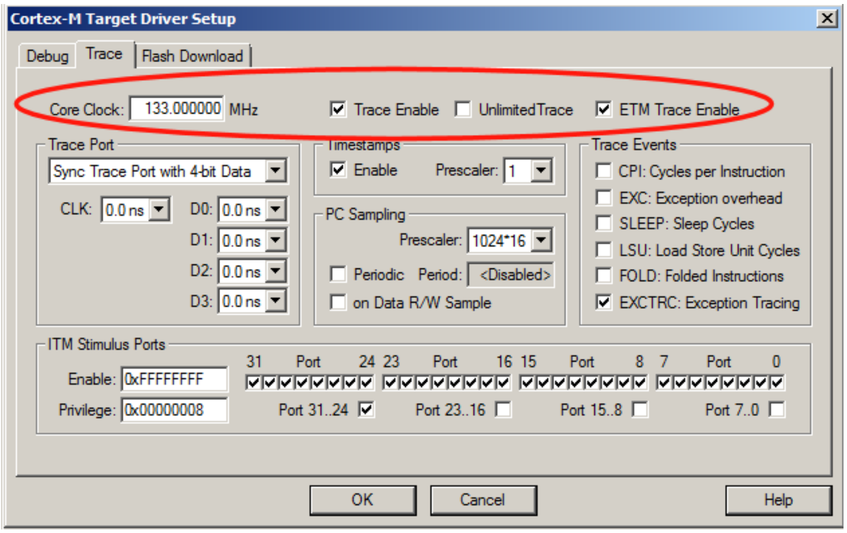


ULINKpro

ULINK2
ULINKpro
J-Link
ULINKpro ETM

Configuring ETM Trace *(and SWV using the 4 bit Trace port)*:

1. Select Options for Target or ALT-F7.
2. Select the Debug tab and click on Settings:
3. Click on the Trace tab. The window below is displayed.
4. Select Trace Enable. This selects SWV.
5. Set Core Clock: to 133 MHz. ULINK*pro* uses this to calculate timings in various windows.
6. In Trace Port, select Sync Trace Port with 4-bit data. It is best to use the widest size which is 4 bits for Cortex-M.
7. Select ETM Trace Enable. This activates ETM Instruction Trace.
8. Select EXCTRC and leave everything else at default as shown below. Only ITM 31 and 0 are used in µVision and need to be selected.
9. Click on OK twice to return to the main µVision menu.
10. ETM and SWV are now configured through the 4 bit Trace Port.
11. Select File/Save All.



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 133.000000 MHz ☑ Trace Enable ☐ Unlimited Trace ☑ ETM Trace Enable

Trace Port: Sync Trace Port with 4-bit Data

CLK: 0.0 ns D0: 0.0 ns D1: 0.0 ns D2: 0.0 ns D3: 0.0 ns

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16 ☐ Periodic Period: <Disabled> ☐ on Data R/W Sample

Trace Events:
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

ITM Stimulus Ports

31 Port 24 23 Port 16 15 Port 8 7 Port 0

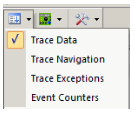
Enable: 0xFFFFFFFF ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑

Privilege: 0x00000008 Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

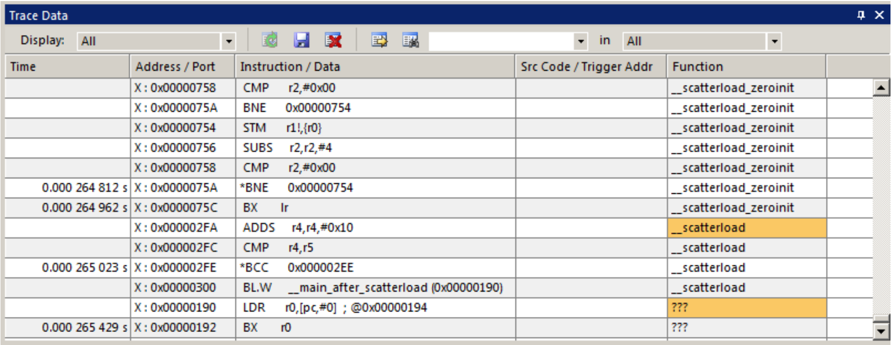
2) Blinky Example: ETM Frames starting at RESET and beyond:

The Blinky project has now been configured on the previous page to provide ETM Trace.

1. Compile the Blinky source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
2. Enter Debug mode. Select OK if the Evaluation Mode box appears.
3. DO NOT CLICK ON RUN YET !!! If you did, simply exit and re-enter Debug mode.
4. Open the Trace Data window by clicking on the small arrow beside the Trace icon as shown here:

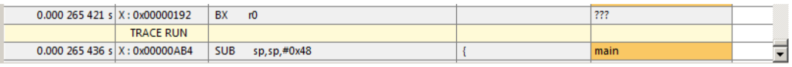


5. Right click anywhere in the Trace Data window and select Show Functions. Size the Trace Data window.
6. Examine the Trace Data window as shown below: This is a complete record of all the program flow since RESET until µVision halted the program at the start of main() since Run To main is selected in µVision.
7. In this case, Time 265 421 shows the last instruction to be executed is (BX r0). This is the jump to main().



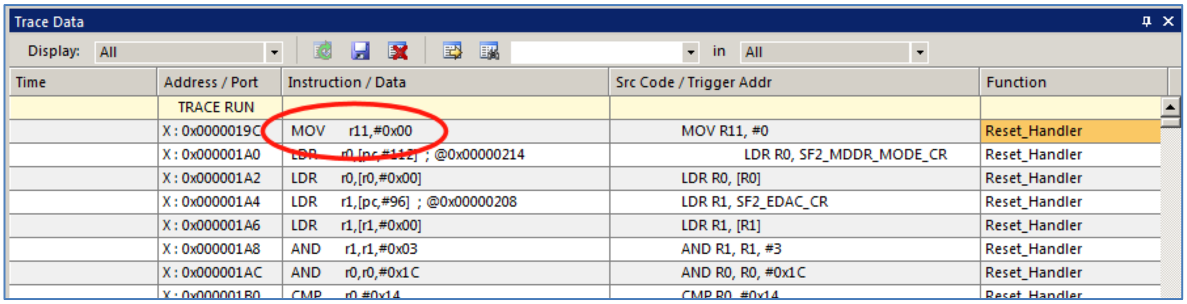
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x00000758	CMP r2,#0x00		__scatterload_zeroinit
	X : 0x0000075A	BNE 0x00000754		__scatterload_zeroinit
	X : 0x00000754	STM r1!,{r0}		__scatterload_zeroinit
	X : 0x00000756	SUBS r2,r2,#4		__scatterload_zeroinit
	X : 0x00000758	CMP r2,#0x00		__scatterload_zeroinit
0.000 264 812 s	X : 0x0000075A	*BNE 0x00000754		__scatterload_zeroinit
0.000 264 962 s	X : 0x0000075C	BX lr		__scatterload_zeroinit
	X : 0x000002FA	ADDS r4,r4,#0x10		__scatterload
	X : 0x000002FC	CMP r4,r5		__scatterload
0.000 265 023 s	X : 0x000002FE	*BCC 0x000002EE		__scatterload
	X : 0x00000300	BL.W __main_after_scatterload (0x00000190)		__scatterload
	X : 0x00000190	LDR r0,[pc,#0] ; @0x00000194		???
0.000 265 429 s	X : 0x00000192	BX r0		???

8. In the Register window the PC will display the value of the next instruction to be executed (0x000 0Ab4 this case).
9. Click on Single Step once. One instruction will be executed and displayed in the Trace Data window:
10. The instruction SUB will display at 0x 07FE:
11. Double click on any line in the Trace Data window and this instruction will be highlighted in the source and Disassembly windows.



0.000 265 421 s	X : 0x00000192	BX r0		???
	TRACE RUN			
0.000 265 436 s	X : 0x00000AB4	SUB sp,sp,#0x48	{	main

12. Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is a MOV instruction in the RESET_Handler function as shown below:
13. If you use the Memory window to look at location 0x4, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0000 0019C + 1 = 0x019D (+1 says it is a Thumb instruction). These first instructions after RESET are shown below: Note the source information that is displayed including which function and instruction belongs to. The first occurrence in a function is highlighted in orange to make the start of functions easier to find.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x0000019C	MOV r11,#0x00	MOV R11, #0	Reset_Handler
	X : 0x000001A0	LDR r0,[pc,#112] ; @0x00000214	LDR R0, SF2_MDDR_MODE_CR	Reset_Handler
	X : 0x000001A2	LDR r0,[r0,#0x00]	LDR R0, [R0]	Reset_Handler
	X : 0x000001A4	LDR r1,[pc,#96] ; @0x00000208	LDR R1, SF2_EDAC_CR	Reset_Handler
	X : 0x000001A6	LDR r1,[r1,#0x00]	LDR R1, [R1]	Reset_Handler
	X : 0x000001A8	AND r1,r1,#0x03	AND R1, R1, #3	Reset_Handler
	X : 0x000001AC	AND r0,r0,#0x1C	AND R0, R0, #0x1C	Reset_Handler
	X : 0x000001B0	CMP r0,#0x14	CMP R0, #0x14	Reset_Handler

TIP: If you unselect Run to main() in the Debug tab, no instructions will be executed when you enter Debug mode. The PC will equal 0x019C. You can run or single-step from that point and this will be recorded in the Trace Data window. You can try this by clearing the Trace Data , click RESET and then Single Step .

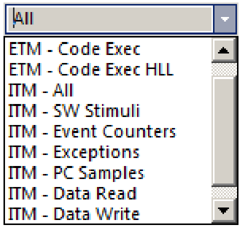
TIP: If you see mostly the same B instructions, these are in the RTX Idle daemon code. To filter these out, select ETM – Code Exec HLL: Only instructions with C source will be displayed. ETM - Code Exec HLI

3) Finding the Trace Frames:

Capturing all the instructions executed is possible with ULINK*pro* but this might not be practical. It is not easy sorting through millions and billions of trace frames or records looking for the ones you want. You can use Find, Trace Triggering, Post Filtering or save everything to a file and search with a different application program such as a spreadsheet.

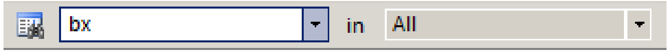
Trace Filters:

In the Trace Data window you can select various types of frames to be displayed. Open the Display: box and you can see the various options available as shown here: These filters are post collection. Future enhancements to µVision will allow more precise filters to be selected.



Find a Trace Record:

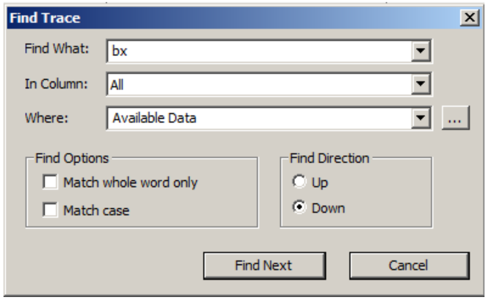
In the Find a Trace Record box, enter **bx** as shown here:



You can select properties where you want to search in the “in” box. "All" is shown in the screen above:

Select the Find a Trace Record icon and the Find Trace window screen opens as shown here: Click on Find Next and each time it will step through the Trace records highlighting each occurrence of the instruction bx.

TIP: Or you can press Enter to go to the next occurrence of the search term.



Find What: bx

In Column: All

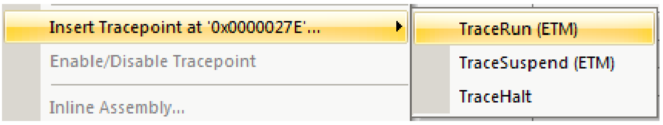
Where: Available Data

Trace Triggering:

µVision has three trace triggers currently implemented:

1. **TraceRun:** Starts ETM trace collection when encountered.
2. **TraceSuspend:** Stops ETM trace collection when encountered. TraceRun has to have first been set and encountered to start the trace collection for this trigger to have an effect.
3. **TraceHalt:** Stops ETM trace, SWV and ITM. Trace collection can be resumed only with a STOP/RUN sequence. TraceStart will not restart it.

They are selected from the menu shown here:
This menu is accesses by right-clicking on a valid assembly instruction in the Disassemble window.


Insert Tracepoint at '0x0000027E'...
Enable/Disable Tracepoint
Inline Assembly...
TraceRun (ETM)
TraceSuspend (ETM)
TraceHalt

TIP: These trace commands have no effect on SWV or ITM. TraceRUN starts the ETM trace and TraceSuspend and TraceHalt stops it.

How it works:

When you set a TraceRun point in assembly language point, ULINK*pro* will start collecting trace records. When you set a TraceSuspend point, trace records collection will stop there. EVERYTHING in between these two times will be collected. This includes all instructions through any branches, exceptions and interrupts.

Sometimes there is some skid past the trigger point which is normal.

Trace Commands:

There are a series of Trace Commands you can enter in the Command window while in Debug mode.

See **www.keil.com/support/man/docs/uv4/uv4_debug_commands.htm**

TL - Trace List: list all tracepoints.

TK - Trace Kill: tk* kills all tracepoints or tk *number* only a specified one i.e. tk 2.

TIP: TK* is very useful for deleting tracepoints when you can't locate them in the source or Disassembly windows.

TL is useful for finding any tracepoints set. Results are displayed in the Command window.

4) Setting Trace Triggers:

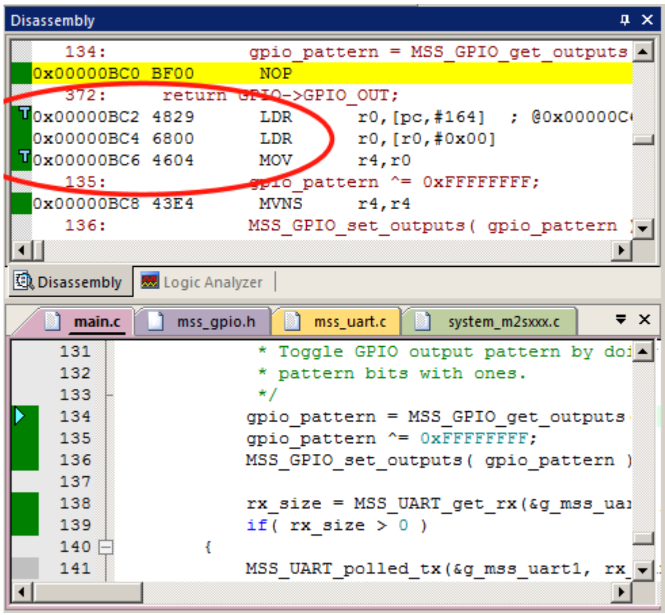
1. Stop the program and stay in Debug mode.
2. In main.c, click on the line gpio_pattern = MSS_GPIO_get_outputs. This line is located near line 134.
3. This highlights the assembly code in the Disassembly window.
4. Right-click on LDR at 0x6D2 in the Disassembly window (just after line 63). (your addresses might be different)
5. Select Insert Tracepoint at 0x0BC2 and select **TraceRun (ETM).** A cyan **T** will appear as shown below:
6. Right-click on the MOV in the left margin at 0x0BC6 as shown below in the Disassembly window:
7. Select Insert Tracepoint at 0x0BC6 and select **TraceSuspend (ETM).** A cyan **T** will appear.
8. Clear the Trace Data window for convenience. This is an optional step.
9. Click RUN and after a few seconds click STOP.
10. Filter exceptions out by selecting ETM – Code Exec in the Display in the Trace Data window:

Display: ETM - Code Exec

11. Examine the Trace Data window as shown below:
12. You can clearly see where the triggers started and stopped the collection of trace.

You can see below where the trace started on 0xBC2 and stopped on 0xBC8: The MVNS is trace skid. All other frames are discarded.

13. In the Command window, enter TL and press Enter. The two Trace points are displayed.
14. **Enter TK* and press Enter to delete all Tracepoints.**



```
   134:                gpio_pattern = MSS_GPIO_get_outputs
0x00000BC0 BF00         NOP
   372:        return GPIO->GPIO_OUT;
0x00000BC2 4829         LDR      r0,[pc,#164]  ; @0x00000C
0x00000BC4 6800         LDR      r0,[r0,#0x00]
0x00000BC6 4604         MOV      r4,r0
   135:                gpio_pattern ^= 0xFFFFFFFF;
0x00000BC8 43E4         MVNS     r4,r4
   136:                MSS_GPIO_set_outputs( gpio_pattern )
```

```
 131                   * Toggle GPIO output pattern by doi
 132                   * pattern bits with ones.
 133                   */
 134                  gpio_pattern = MSS_GPIO_get_outputs
 135                  gpio_pattern ^= 0xFFFFFFFF;
 136                  MSS_GPIO_set_outputs( gpio_pattern )
 137
 138                  rx_size = MSS_UART_get_rx(&g_mss_ua
 139                  if( rx_size > 0 )
 140          {
 141                  MSS_UART_polled_tx(&g_mss_uart1, rx_
```

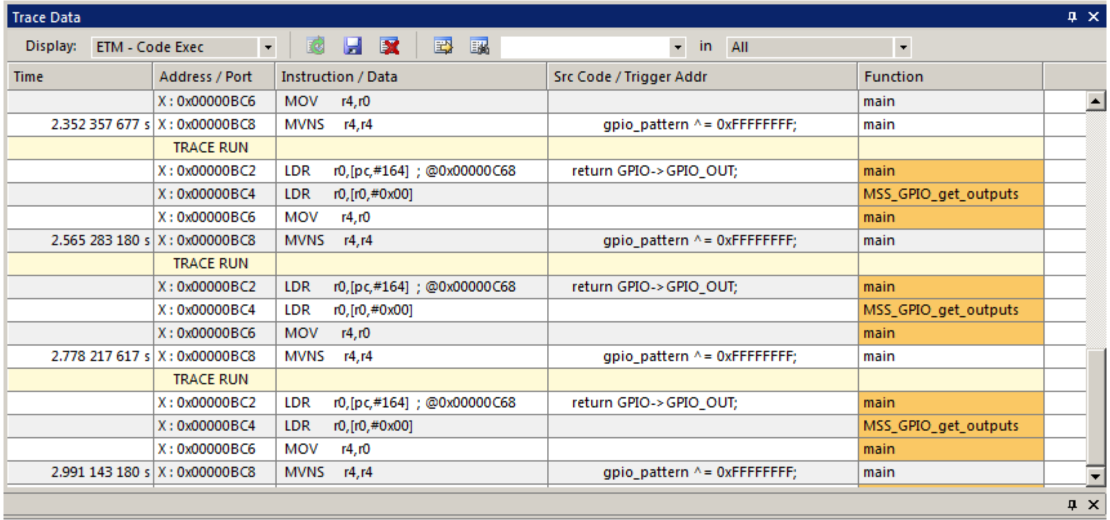
Trace Skid:

In this example, the instruction MVNS was included even though you did not select it when the trace triggers were set. This is trace skid. The trace triggers use the same CoreSight hardware as the Watchpoints. This means that it is possible a program counter skid might happen. The program might not start or stop on the exact location you set the trigger to.

You might have to adjust the trigger point location to minimize this effect.

This is because of the nature of the comparators in the CoreSight module and it is normal behavior.

TIP: If any exceptions or threads switching occurred when the trace collection is on, these instructions will also be collected.



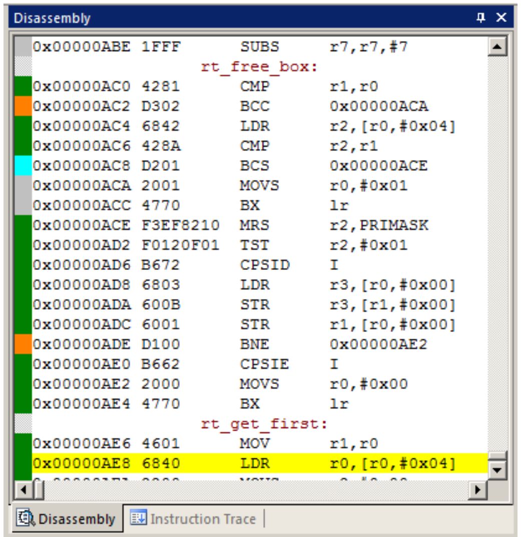
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x00000BC6	MOV r4,r0		main
2.352 357 677 s	X : 0x00000BC8	MVNS r4,r4	gpio_pattern ^= 0xFFFFFFFF;	main
	TRACE RUN			
	X : 0x00000BC2	LDR r0,[pc,#164] ; @0x00000C68	return GPIO->GPIO_OUT;	main
	X : 0x00000BC4	LDR r0,[r0,#0x00]		MSS_GPIO_get_outputs
	X : 0x00000BC6	MOV r4,r0		main
2.565 283 180 s	X : 0x00000BC8	MVNS r4,r4	gpio_pattern ^= 0xFFFFFFFF;	main
	TRACE RUN			
	X : 0x00000BC2	LDR r0,[pc,#164] ; @0x00000C68	return GPIO->GPIO_OUT;	main
	X : 0x00000BC4	LDR r0,[r0,#0x00]		MSS_GPIO_get_outputs
	X : 0x00000BC6	MOV r4,r0		main
2.778 217 617 s	X : 0x00000BC8	MVNS r4,r4	gpio_pattern ^= 0xFFFFFFFF;	main
	TRACE RUN			
	X : 0x00000BC2	LDR r0,[pc,#164] ; @0x00000C68	return GPIO->GPIO_OUT;	main
	X : 0x00000BC4	LDR r0,[r0,#0x00]		MSS_GPIO_get_outputs
	X : 0x00000BC6	MOV r4,r0		main
2.991 143 180 s	X : 0x00000BC8	MVNS r4,r4	gpio_pattern ^= 0xFFFFFFFF;	main

5) Code Coverage:

1. Click on the RUN icon. After a second or so stop the program with the STOP icon.
2. Examine the Disassembly and main.c windows. Scroll and notice the different color blocks in the left margin:
   Note: This example program is very simple and it might be hard to find any cyan or orange blocks.
3. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not.

Colour blocks indicate which assembly instructions have been executed.

1. 🟩 Green: this assembly instruction was executed.
2. ⬜ Gray: this assembly instruction was not executed.
3. 🟧 Orange: a Branch is never taken.
4. 🟦 Cyan: a Branch is always taken.
5. ▨ Light Gray: there is no assembly instruction here.
6. 🔴 RED: Breakpoint is set here.
7. ⇦ The next instruction to be executed.



```
0x00000ABE 1FFF      SUBS     r7,r7,#7
                 rt_free_box:
0x00000AC0 4281      CMP      r1,r0
0x00000AC2 D302      BCC      0x00000ACA
0x00000AC4 6842      LDR      r2,[r0,#0x04]
0x00000AC6 428A      CMP      r2,r1
0x00000AC8 D201      BCS      0x00000ACE
0x00000ACA 2001      MOVS     r0,#0x01
0x00000ACC 4770      BX       lr
0x00000ACE F3EF8210  MRS      r2,PRIMASK
0x00000AD2 F0120F01  TST      r2,#0x01
0x00000AD6 B672      CPSID    I
0x00000AD8 6803      LDR      r3,[r0,#0x00]
0x00000ADA 600B      STR      r3,[r1,#0x00]
0x00000ADC 6001      STR      r1,[r0,#0x00]
0x00000ADE D100      BNE      0x00000AE2
0x00000AE0 B662      CPSIE    I
0x00000AE2 2000      MOVS     r0,#0x00
0x00000AE4 4770      BX       lr
                 rt_get_first:
0x00000AE6 4601      MOV      r1,r0
0x00000AE8 6840      LDR      r0,[r0,#0x04]
```

In the window on the right you can easily see examples of each type of Code Coverage block and if they were executed or not and if branches were taken (or not).

TIP: Code Coverage is visible in both the Disassembly and source code windows. Click on a line in one and this place will be matched in the other window.

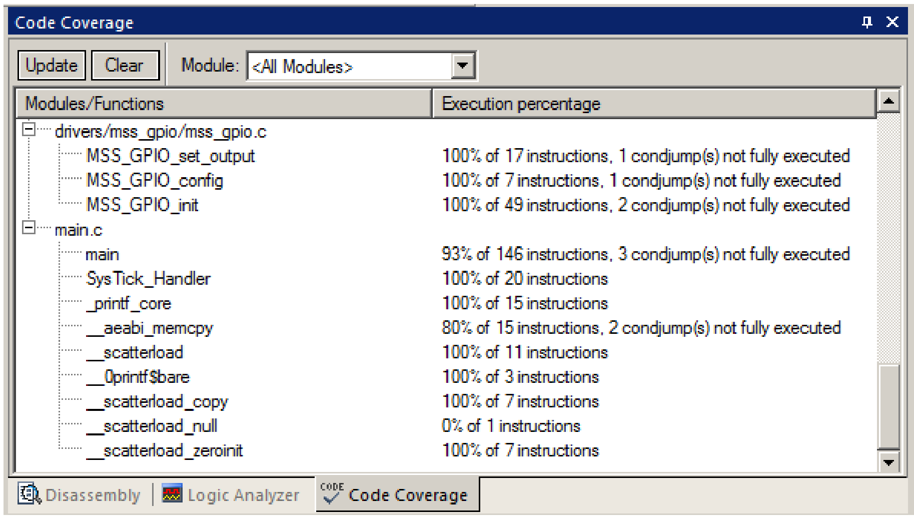
In the window above, why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

Code Coverage tells what assembly instructions were executed. It is important to ensure all assembly code produced by the compiler is executed and tested. You do not want a bug or an unplanned circumstance to cause a sequence of untested instructions to be executed. The result could be catastrophic as unexecuted instructions have not been tested. Some agencies such as the US FDA and FAA require Code Coverage for certification. This is provided in MDK µVision using ULINK*pro*.

Good programming practice requires that these unexecuted instructions be identified and tested.

Code Coverage is captured by ETM. Code Coverage is also available in the Keil Simulator.

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage.
The next page describes how to save Code Coverage information to a file. Click the Update button to refresh this window.



Modules/Functions	Execution percentage
drivers/mss_gpio/mss_gpio.c	
MSS_GPIO_set_output	100% of 17 instructions, 1 condjump(s) not fully executed
MSS_GPIO_config	100% of 7 instructions, 1 condjump(s) not fully executed
MSS_GPIO_init	100% of 49 instructions, 2 condjump(s) not fully executed
main.c	
main	93% of 146 instructions, 3 condjump(s) not fully executed
SysTick_Handler	100% of 20 instructions
_printf_core	100% of 15 instructions
__aeabi_memcpy	80% of 15 instructions, 2 condjump(s) not fully executed
__scatterload	100% of 11 instructions
__0printf$bare	100% of 3 instructions
__scatterload_copy	100% of 7 instructions
__scatterload_null	0% of 1 instructions
__scatterload_zeroinit	100% of 7 instructions

Saving Code Coverage:

Code Coverage information is temporarily saved during a run and is displayed in various windows as already shown.

It is possible to save this information in an ASCII file for use in other programs.

You can Save Code Coverage in two formats:

1. As a binary file that can be later loaded back into µVision. Use the command Coverage Save *filename*.
2. As an ASCII file. You can either copy and paste from the Command window or use the log command:

```
1)  log > c:\cc\test.txt    ; send CC data to this file.  The specified directory must exist.
2)  coverage asm            ; you can also specify a module or function.
3)  log off                 ; turn the log function off.
```

HELP TIP: To get help on Code Coverage, type Coverage in the Command window and press the F1 key.

1) Here is a partial display using the command **`coverage.`** This displays and optionally saves everything.

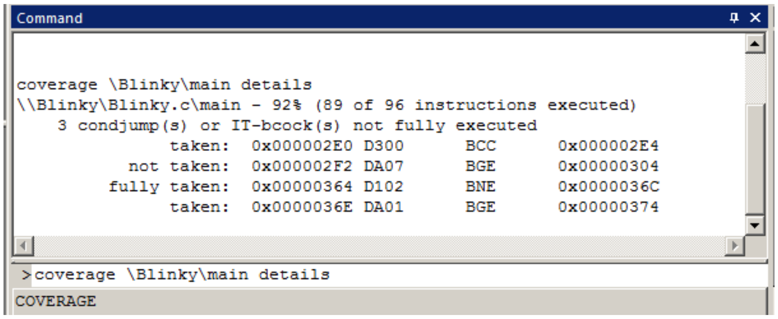
```
coverage
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\SystemCoreClockUpdate - 44% (30 of 67 instructions executed)
    2 condjump(s) or IT-block(s) not fully executed
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\get_rcosc_25_50mhz_frequency - 0% (0 of 8 instructions
executed)
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\set_clock_frequency_globals - 0% (0 of 16 instructions
executed)
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\SystemInit - 100% (9 of 9 instructions executed)
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\silicon_workarounds - 43% (7 of 16 instructions executed)
    1 condjump(s) or IT-block(s) not fully executed
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\get_silicon_revision - 68% (11 of 16 instructions executed)
    2 condjump(s) or IT-block(s) not fully executed
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\m2s050_rev_a_workarounds - 0% (0 of 11 instructions
executed)
\\SF2_Blink\RTE/Device/M2S090/startup_m2sxxx.s\Reset_Handler - 48% (20 of 41 instructions executed)
    4 condjump(s) or IT-block(s) not fully executed
\\SF2_Blink\RTE/Device/M2S090/startup_m2sxxx.s\fill_memory - 0% (0 of 42 instructions executed)
\\SF2_Blink\RTE/Device/M2S090/startup_m2sxxx.s\NMI_Handler - 0% (0 of 1 instructions executed)
```

The command **`coverage asm produces this listing (partial is shown):`**

```
coverage asm
\\SF2_Blink\RTE/Device/M2S090/system_m2sxxx.c\SystemCoreClockUpdate - 44% (30 of 67 instructions executed)
    2 condjump(s) or IT-block(s) not fully executed
 EX |  0x00000620 SystemCoreClockUpdate:
 EX |  0x00000620 E92D41FC  PUSH     {r2-r8,lr}
 EX |  0x00000624 4827      LDR      r0,[pc,#156]  ; @0x000006C4
 EX |  0x00000626 6800      LDR      r0,[r0,#0x00]
 EX |  0x00000628 F0006480  AND      r4,r0,#0x4000000
 EX |  0x0000062C 2C00      CMP      r4,#0x00
 NT |  0x0000062E D141      BNE      0x000006B4
 EX |  0x00000630 4824      LDR      r0,[pc,#144]  ; @0x000006C4    3 condjump(s) or IT-bcock(s) not fully
executed
```

The first column above describes the execution as follows:

NE	Not Executed
FT	Branch is fully taken
NT	Branch is not taken
AT	Branch is always taken.
EX	Instruction was executed (at least once)



```
coverage \Blinky\main details
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
    3 condjump(s) or IT-bcock(s) not fully executed
               taken:  0x000002E0 D300      BCC       0x000002E4
           not taken:  0x000002F2 DA07      BGE       0x00000304
         fully taken:  0x00000364 D102      BNE       0x0000036C
               taken:  0x0000036E DA01      BGE       0x00000374
>coverage \Blinky\main details
```

2) Shown here is an example using:
`coverage \Blinky\main details`

If the log command is run, this will be saved/appended to the specified file.

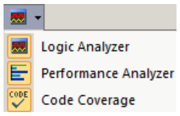
You can enter the command coverage with various available options to see what is displayed.

6) Performance Analysis (PA):

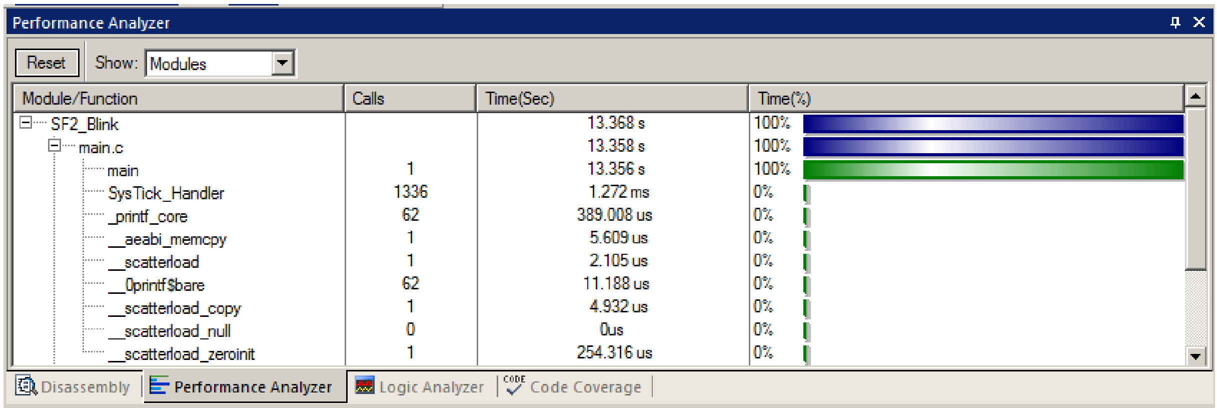
Performance Analysis tells you how much time was spent in each function. It is useful to optimize your code for speed.

Keil provides Performance Analysis with ETM and the ULINK*pro*. The number of total calls made as well as the total time spent in each function is displayed. A graphical display is generated for a quick reference. If you are optimizing for speed, work first on those functions taking the longest time to execute.

1. Use the same Blinky program as used with Code Coverage.
2. The program can be running. PA updates while the program runs.
3. Select View/Analysis Windows/Performance Analysis or with this icon:



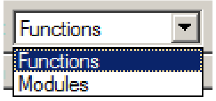
4. A window similar to the one below will open up.
5. Expand some of the module names as shown below.



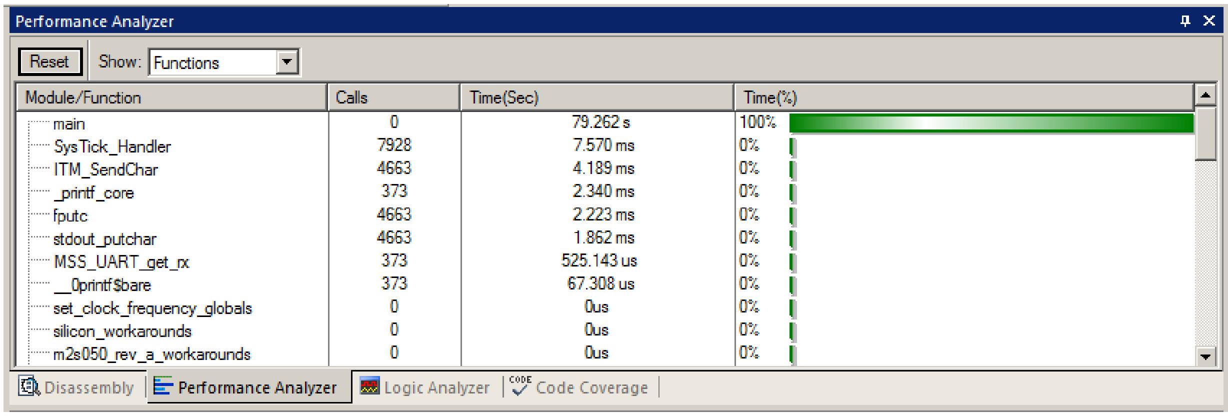
Show: Modules

Module/Function	Calls	Time(Sec)	Time(%)
SF2_Blink		13.368 s	100%
main.c		13.358 s	100%
main	1	13.356 s	100%
SysTick_Handler	1336	1.272 ms	0%
_printf_core	62	389.008 us	0%
__aeabi_memcpy	1	5.609 us	0%
__scatterload	1	2.105 us	0%
__0printf$bare	62	11.188 us	0%
__scatterload_copy	1	4.932 us	0%
__scatterload_null	0	0us	0%
__scatterload_zeroinit	1	254.316 us	0%

6. Shown is the number of calls and percentage of total time in this short run from RESET to the beginning of main().
7. Note the display updates in real-time while the program is running. There is no need to stop the processor to collect the data. No code stubs are needed in your source files. This program is very simple. Try the DSP if you like.
8. Select Functions from the pull down box as shown here and notice the difference.



9. Click on the PA RESET icon. Reset Watch as new data is displayed in the PA window.
10. When you are done, STOP and exit Debug mode .



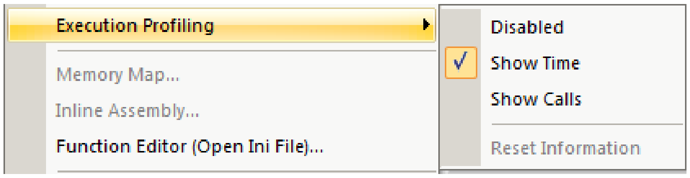
Show: Functions

Module/Function	Calls	Time(Sec)	Time(%)
main	0	79.262 s	100%
SysTick_Handler	7928	7.570 ms	0%
ITM_SendChar	4663	4.189 ms	0%
_printf_core	373	2.340 ms	0%
fputc	4663	2.223 ms	0%
stdout_putchar	4663	1.862 ms	0%
MSS_UART_get_rx	373	525.143 us	0%
__0printf$bare	373	67.308 us	0%
set_clock_frequency_globals	0	0us	0%
silicon_workarounds	0	0us	0%
m2s050_rev_a_workarounds	0	0us	0%

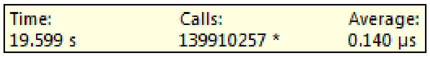
.

7) Execution Profiling:

Execution profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.

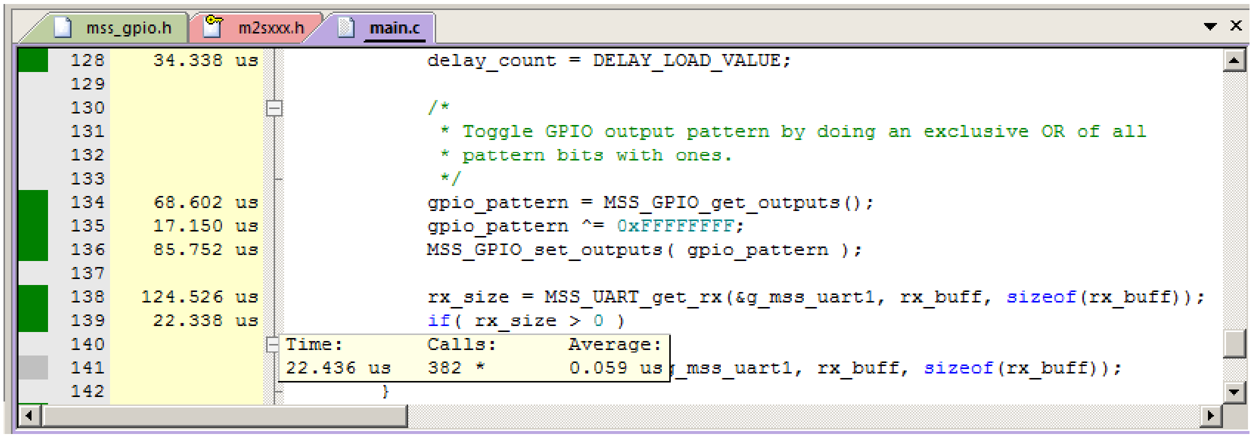


1. Enter Debug mode.
2. Select Debug/Execution Profiling/Show Time.
3. Click on RUN.
4. In the left margin of the Disassembly and C source windows will display various time values.
5. The times will start to fill up as shown below:
6. Click inside the yellow margin of main.c to refresh it.
7. This is done in real-time and without stealing CPU cycles.
8. Hover the cursor over a time and ands more information appears as in the yellow box here:



```
Time:        Calls:         Average:
19.599 s     139910257 *    0.140 µs
```

9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



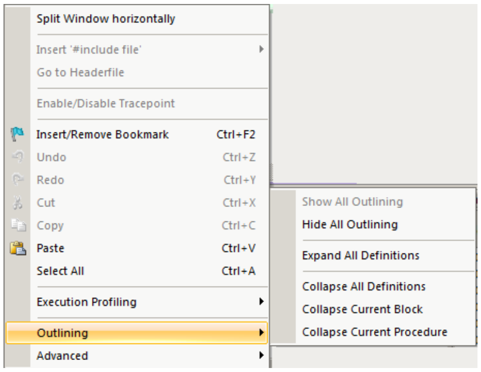
```
 34.338 us          delay_count = DELAY_LOAD_VALUE;

                    /*
                     * Toggle GPIO output pattern by doing an exclusive OR of all
                     * pattern bits with ones.
                     */
 68.602 us          gpio_pattern = MSS_GPIO_get_outputs();
 17.150 us          gpio_pattern ^= 0xFFFFFFFF;
 85.752 us          MSS_GPIO_set_outputs( gpio_pattern );

124.526 us          rx_size = MSS_UART_get_rx(&g_mss_uart1, rx_buff, sizeof(rx_buff));
 22.338 us          if( rx_size > 0 )
           Time:       Calls:     Average:
           22.436 us   382 *      0.059 us _mss_uart1, rx_buff, sizeof(rx_buff));
                 }
```

Outlining:

Each place there is a small square with a "-" sign ⊟ that can be collapsed down to compress the associated source lines.



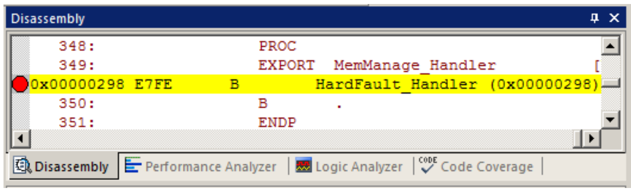
1) Right click in a place in main.c. This window opens up:
1) You can collapse sections of the C source using these menu options.
2) This can be useful to hide sections of code to simplify the window you are reading.
3) Click on a + to expand it.
4) Stop the program
5) Exit Debug mode .

For more information see:
http://www.keil.com/support/man/docs/uv4/uv4_ui_outline.htm

8) In-the-Weeds Example: (your addresses might not be the same as shown here)

Some of the hardest problems to solve are those when a crash has occurred and you have no clue what caused this – you only know that it happened and the stack is corrupted or provides no useful clues. Modern programs tend to be asynchronous with interrupts and RTOS thread switching plus unexpected and spurious events. Having a recording of the program flow is useful especially when a problem occurs and the consequences are not immediately visible. Another problem is detecting race conditions and determining how to fix them. ETM trace handles these problems and others easily and it is easy to use.

If a Hard Fault occurs in our example, the CPU will end up at 0x0298 as shown in the Disassembly window below. This is the Hard Fault handler. This is a branch to itself and will run this instruction forever. The trace buffer will save millions of the same branch instructions. This is not very useful.

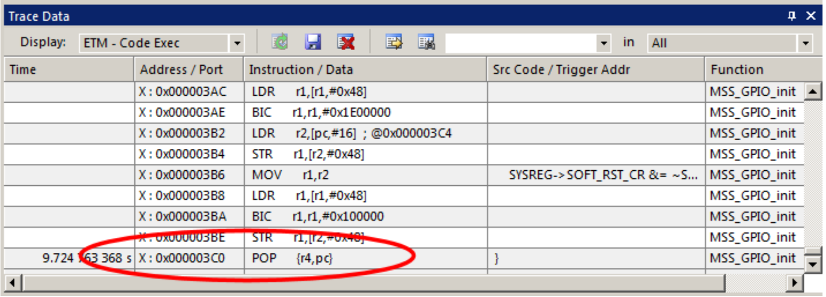


```
    348:                    PROC
    349:                    EXPORT  MemManage_Handler          [
0x00000298 E7FE      B          HardFault_Handler (0x00000298)
    350:                    B       .
    351:                    ENDP
```

The Hard Fault handler exception vector is found in the file startup_M2Sxxx.s. If we set a breakpoint by clicking on the Hard Fault Handler and run the program: at the next Hard Fault event the CPU will jump to the HardFault_Handler and stop.

The difference this time is the breakpoint will stop the CPU and also the trace collection. The trace buffer will be visible and is useful to investigate and determine the cause of the crash.

1. Use the Blinky example from the previous exercise, enter Debug mode.
2. Locate the Hard Fault near address 0x0298 in the Disassembly window or near line 345 in startup_M2Sxxx.s.
3. Set a breakpoint at this point. A red circle will appear. You can do this in Edit or Debug mode in the .s file.
4. In main.c, set a breakpoint near line 60: at MSS_GPIO_init();. We need to enter a function using a LR branch.
5. Click on RESET: This ensures the hardware breakpoint at MSS_GPIO_init(); will be hit.
6. RUN the program. It will halt on this breakpoint. Single-step once to enter the function MSS_GPIO_init().
7. The assembly and sources in the Disassembly window do not always match up and this is caused by anomalies in ELF/DWARF specification. In general, scroll downwards in this window to provide the best match.
8. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
9. In the Register window, double-click on R14 (LR) register and set it to zero. This is guaranteed to cause a Hard Fault when the processor attempts to execute an instruction at 0x2000 10B0 which is the initial SP address.
10. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
11. In the Trace Data window you will find the POP in the function MSS_GPIO_init that created the Hard Fault. Plus all the instructions leading to this point. This is what caused the Hard Fault since you set LR = 0. When the function tried to return, the bogus value of LR caused a Hard Fault. See the Trace Data window below:
12. The Call Stack shows the Hard Fault
13. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.
14. Click on Single Step. You will now see the Hard Fault branch.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x000003AC	LDR r1,[r1,#0x48]		MSS_GPIO_init
	X : 0x000003AE	BIC r1,r1,#0x1E00000		MSS_GPIO_init
	X : 0x000003B2	LDR r2,[pc,#16] ; @0x000003C4		MSS_GPIO_init
	X : 0x000003B4	STR r1,[r2,#0x48]		MSS_GPIO_init
	X : 0x000003B6	MOV r1,r2	SYSREG->SOFT_RST_CR &= ~S...	MSS_GPIO_init
	X : 0x000003B8	LDR r1,[r1,#0x48]		MSS_GPIO_init
	X : 0x000003BA	BIC r1,r1,#0x100000		MSS_GPIO_init
	X : 0x000003BE	STR r1,[r2,#0x48]		MSS_GPIO_init
9.724 [illegible]63 368 s	X : 0x000003C0	POP {r4,pc}	}	MSS_GPIO_init

This example clearly shows how quickly ETM trace can help debug program flow bugs.

Exception Entry: Note at the top the Hard fault exception is listed. How can this happen before the Hard Fault happens ? This entry is part of SWV – the timestamps are different for SWV and ETM instructions so they can be out of sync. Use Display: to filter out such events.

TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 489… and select TraceHalt. When Hard Fault is reached, trace collection will halt but the program will keep executing.

15. Exit Debug mode. This is the end of the ETM Instruction trace tutorial.

Part 8: Creating µVision projects:

1) Creating your own MDK 5 project from scratch: no RTOS (bare metal)

We will start this example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run a bare metal (no OS) Blinky example. It will have a simple incrementing counter to monitor. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using an RTOS. This tutorial used Software Pack 1.0.61.

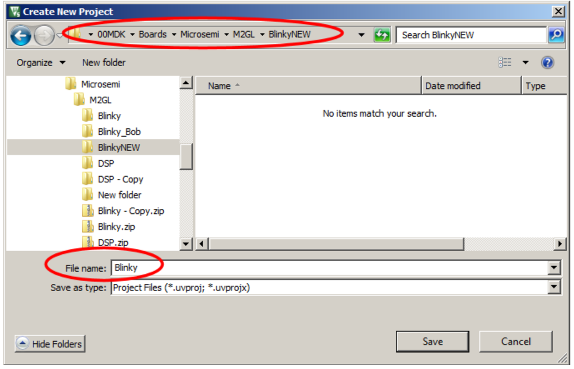
Three files from a Libero project are needed. You can use your own or the ones provided with this tutorial.

Install the Software Pack for your processor:

1. Start µVision and leave it in Edit mode. Do not enter Debug mode. A project must be loaded. Any project at all.
2. **Pack Installer:** The Microsemi::M2Sxxx Software Pack must be installed. This was done on page 4.

Create a new Directory and a New Project:

3. In the main µVision menu, select Project/New µVision Project…   Create New Project window opens:
4. In this window, shown here, navigate to the folder C:\00MDK\Boards\Microsemi\M2S090TS\
5. Right click in this window and select New (or click New Folder) and create a new folder. I called it BlinkyNEW.
6. Double click on BlinkyNew to open it or select Open.
7. In the File name: box, enter Blinky. Click on Save.
8. This creates the project Blinky.uvproj. (MDK 4 format)



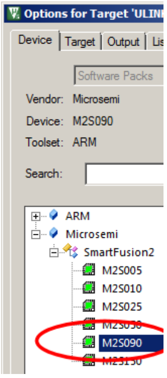
File name: Blinky

Save as type: Project Files (*.uvproj; *.uvprojx)

9. The Select Device for Target…opens:

Select the Device you are using:

1. Expand Microsemi/SmartFusion2. Select M2S090 as shown here:
2. Click OK and the Manage Run Time Environment window shown below right opens.



Vendor: Microsemi

Device: M2S090

Toolset: ARM

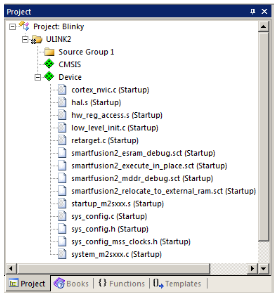
Search:

Select the CMSIS components you want:

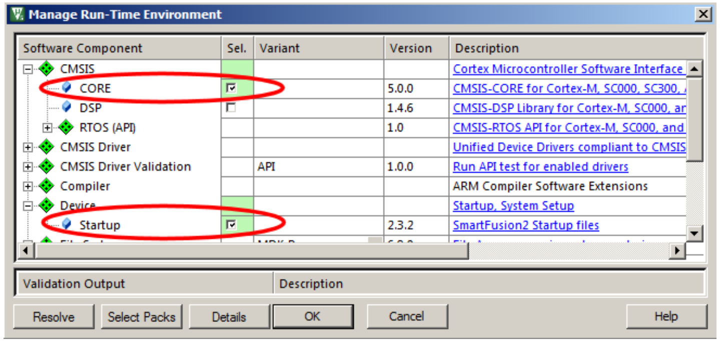
1. Expand CMSIS and Device. Select CORE and Startup as shown below. They will be highlighted in Green indicating there are no other files needed. Click OK to close.
2. Click on File/Save All or select the Save All icon:
3. The project Blinky.uvproj will now be changed to Blinky.uvprojx. (MDK 4 ➡ MDK 5 format)
4. You now have a new project list as shown on the bottom left below: The CMSIS files you selected have been automatically entered and configured into your project for your selected processor.
5. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
6. Click once on it and change its name to **ULINK2 or ULINKpro** and press Enter. Select Target will change.

What has happened to this point:

You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files and select your debug adapter. The Software Pack has pre-configured many settings for your convenience.



- Project: Blinky
  - ULINK2
    - Source Group 1
    - CMSIS
    - Device
      - cortex_nvic.c (Startup)
      - hal.s (Startup)
      - hw_reg_access.s (Startup)
      - low_level_init.c (Startup)
      - retarget.c (Startup)
      - smartfusion2_esram_debug.sct (Startup)
      - smartfusion2_execute_in_place.sct (Startup)
      - smartfusion2_mddr_debug.sct (Startup)
      - smartfusion2_relocate_to_external_ram.sct (Startup)
      - startup_m2sxxx.s (Startup)
      - sys_config.c (Startup)
      - sys_config.h (Startup)
      - sys_config_mss_clocks.h (Startup)
      - system_m2sxxx.c (Startup)



Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface
CORE	☑		5.0.0	CMSIS-CORE for Cortex-M, SC000, SC300,
DSP	☐		1.4.6	CMSIS-DSP Library for Cortex-M, SC000, ar
RTOS (API)			1.0	CMSIS-RTOS API for Cortex-M, SC000, and
CMSIS Driver				Unified Device Drivers compliant to CMSIS
CMSIS Driver Validation		API	1.0.0	Run API test for enabled drivers
Compiler				ARM Compiler Software Extensions
Device				Startup, System Setup
Startup	☑		2.3.2	SmartFusion2 Startup files

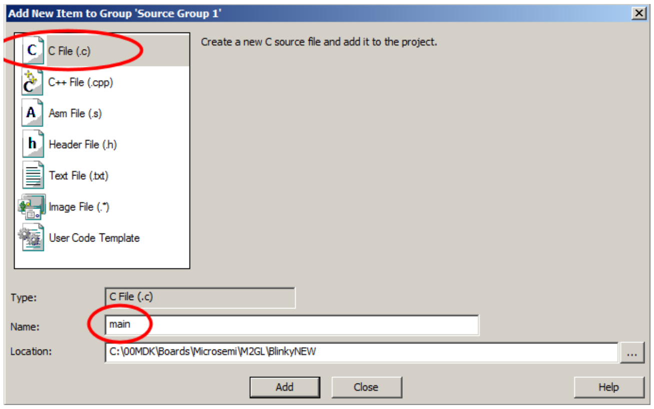
Copy Libero System Files:

An initial project has three dummy files in C:\00MDK\Boards\Microsemi\M2S090TS\BlinkyNEW\RTE\Device\M2S090\ They are sys_config.c, sys_config.h and sys_config_mss_clocks.h. These must be replaced with new ones Libero created when you created the stp file. We can use the files provided in the Blinky example and related to the stp file used. These are located. In C:\00MDK\Boards\Microsemi\M2S090TS\STP\.

1) If you build the files you will see errors in the Output window asking you to replace three files. The text from these errors are inside each of the three dummy source files.
2) Copy the three source files from C:\00MDK\Boards\Microsemi\M2S090TS\STP\ into .\BlinkyNEW\RTE\Device\M2S090\
3) Build the files. The only error now will be that main() is missing. We will add this now.

Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select Add New Item to Group 'Source Files'... .
2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter main.
5. Click Add to close this window.
6. Click on File/Save All or
7. Expand Source Group 1 in the Project window and main.c will now display. It is a blank file.


Add New Item to Group 'Source Group 1'

Create a new C source file and add it to the project.

Type: C File (.c)

Name: main

Location: C:\00MDK\Boards\Microsemi\M2GL\BlinkyNEW

Add Some Code to main.c:

1. Right click in main.c and select Insert '#include file'.
2. Select m2sxxx.h. This will be added to main.c
3. In the nearly blank main.c, add the C code below:
4. Click on File/Save All or
5. Build the files. There will be two errors that are easy to take care of.

```
#include "m2sxxx.h"        // Device header
unsigned int counter = 0;
/*----------------------------------------------
  MAIN function
 *----------------------------------------------*/
int main (void) {

  while(1) {
      counter++;
          if (counter > 0x0F) counter = 0;
}
}           //make sure you add a CR Carriage Return or Enter after the last parentheses.
```

TIP: You could also add existing source files: Add Existing Files to Group 'Source Files'... But not at this time.

Modify low_level_init.c and startup_m2sxxx.s:

7) Open low_level_init.c by clicking on its tab or double click it in the project window.

8) Comment out these two lines:

```
18       Image$$ER_RO$$Base
19       Image$$ER_RW$$Base
```

9) The function low_level_init is located in low_level_init.c starting at line 25. Comment out this entire function from lines 25 through 43.

10) Open startup_m2sxxx.s by clicking on its tab or double click it in the Project window.

11) Comment out these three lines (by using a semi-colon ;)

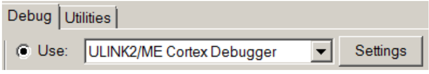
```
163    ;    IMPORT   low_level_init
210    ;    LDR      R0, =low_level_init
211    ;    BLX      R0
```

12) Click on File/Save All or

13) Rebuild the project and there will be no errors or warnings.

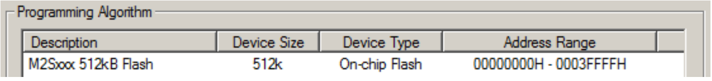
Configure the Target Flash: ***Please complete these instructions carefully to prevent unusual problems…***

1. Select the Target Options icon. Select the **Target** tab. Note the Flash and RAM addresses are already entered.
2. Select Use MicroLIB to optimize for smaller code size. An error will be generated if you cannot use this feature.
3. Select the **Linker** tab. Select Use Memory Layout from Target Dialog. The addresses in the Target tab will be used. If unselected, you can add your own custom scatter file.
4. Click on the **Debug** tab. Select the debugger you are using in the Use: box: You can use a ULINK2, ULINK-ME, ULINK*pro* or a J-Link.



● Use: ULINK2/ME Cortex Debugger

5. Connect your debug adapter to the debug connector. Power your board. Make sure J8 JTAG_SEL is set to L
6. Select Settings: box beside Use ULINK2/ME Debugger or whatever adapter you are using as shown above.
7. Set SW and SWJ as shown here: ☑ SWJ Port: SW If your hardware is connected and powered correctly, you should now see a valid IDCODE and Device Name in the SW Device box. If you do not, you **must** correct this before continuing.
8. Click on the **Flash Download** tab. Confirm the correct eNVM Flash algorithms are present: Shown here is the correct one for the SmartFusion2 processors:



Programming Algorithm

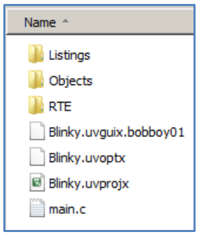
Description	Device Size	Device Type	Address Range
M2Sxxx 512kB Flash	512k	On-chip Flash	00000000H - 0003FFFFH

9. Click on OK twice to return to the main menu.
10. Click on File/Save All or
11. Build the files. There will be no errors or warnings if all was entered correctly. If there are, please fix them !

The Next Step ? First we will do a summary of what we have done so far and then …. you will run the program !

What we have so far ?

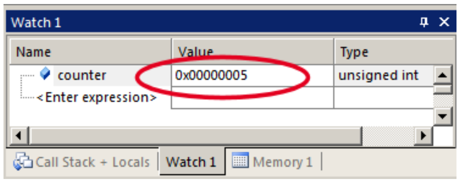
1. A project has been created in C:\00MDK\Boards\Microsemi\M2GL\BlinkyNEW\
2. The folders have been created as shown here:
3. RTE contains the CMSIS-Core startup and system files.
4. The Software Pack has pre-configured many items in this new project for your convenience.



Name
- Listings
- Objects
- RTE
- Blinky.uvguix.bobboy01
- Blinky.uvoptx
- Blinky.uvprojx
- main.c

Running Your Program:

1. Enter Debug mode by clicking on the Debug icon . The eNVM Flash will be programmed.
2. Click on the RUN icon. Note: you stop the program with the STOP icon.
3. Right click on counter in main.c and select Add 'counter' to … and select Watch 1.
4. counter should be updating as shown here:
5. You can also set a breakpoint in main.c and the program should stop at this point if it is running properly. If you do this, remove the breakpoint.
6. You are now able to add your own source code to create a meaningful project. You can select software components in the Manage Run-time Environment window. You can experiment with this later.



Name	Value	Type
counter	0x00000005	unsigned int
<Enter expression>		

TIP: Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip some sequential values that you know must exist.

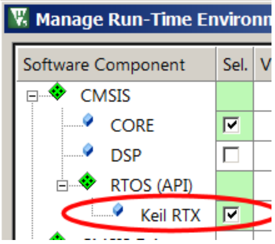
What else can we do ?

5. You can create new source files using the Add New Item window. See the top of the previous page.
6. You can add existing source files by right clicking on a Group name and selecting Add Existing Files.
7. You can easily add Libero example files to your project. Libero provides MDK 4 projects.
8. If you use RTX or Keil Middleware, source and template files are provided in the Add New window.
9. Now, we will add RTX to your new project !

2) Adding RTX to your MDK 5 project:

Software Packs contain all the code needed to add RTX to your project. RTX is CMSIS-RTOS compliant. Configuring RTX is easy in MDK 5. These steps use the preceding Blinky example you constructed.

1. Using the same example from the preceding pages, Stop the program and Exit Debug mode.
2. Open the Manage Run-Time Environment window:
3. Expand all the elements as shown here:
4. Select Keil RTX as shown and click OK.
5. Appropriate RTX files will be added to your project. See the Project window.
6. In main.c, on the first line, right click and select Insert '# include file'. Select "cmsis_os.h". This will be added as the first line in main.c.

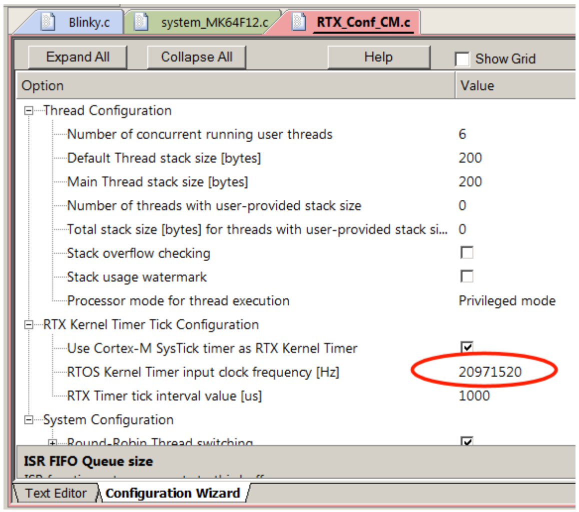


Manage Run-Time Environn

Software Component	Sel.	V
CMSIS		
CORE	☑	
DSP	□	
RTOS (API)		
Keil RTX	☑	

Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.
3. Select the Configuration Wizard tab at the bottom of this window: Select Expand All.
4. The window is displayed here:
5. Select Use Cortex-M SysTick Timer as RTX Kernel Timer.
6. Set Timer clock value: to 133000000 as shown: (133 MHz)
7. Use the defaults for the other settings.



Option	Value
Thread Configuration	
Number of concurrent running user threads	6
Default Thread stack size [bytes]	200
Main Thread stack size [bytes]	200
Number of threads with user-provided stack size	0
Total stack size [bytes] for threads with user-provided stack si...	0
Stack overflow checking	□
Stack usage watermark	□
Processor mode for thread execution	Privileged mode
RTX Kernel Timer Tick Configuration	
Use Cortex-M SysTick timer as RTX Kernel Timer	☑
RTOS Kernel Timer input clock frequency [Hz]	20971520
RTX Timer tick interval value [us]	1000
System Configuration	
Round-Robin Thread switching	☑

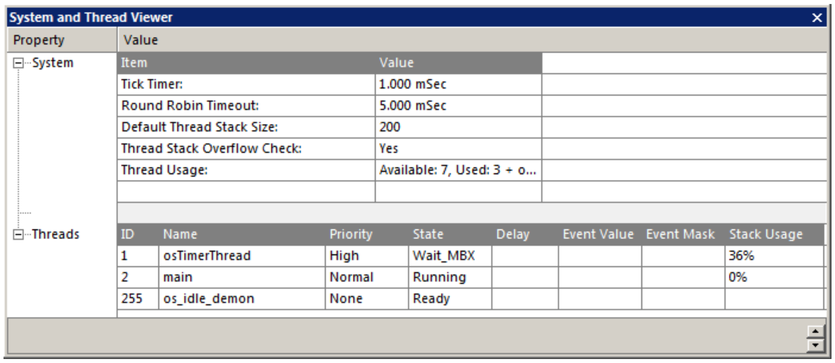
ISR FIFO Queue size

Build and Run Your RTX Program:

1. Click on File/Save All or
2. Build the files. There will be no errors or warnings.
3. Enter Debug mode: Click on the RUN icon.
4. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
5. You can see three threads: the main thread is the only one running. As you add more threads to create a real RTX program, these will automatically be added to this window.
6. Stop the program and Exit Debug mode.

What you have to do now:

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project.
2. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/gsg/. It has information on implementing RTX as well as other subjects.
3. You can use the Event Viewer to examine your threads graphically if you have a ULINK2, ULINK*pro* or a J-Link.



System and Thread Viewer

Property	Value
System	

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 3 + o...

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				36%
2	main	Normal	Running				0%
255	os_idle_demon	None	Ready				

TIP: The Configuration Wizard is a scripting language as shown in the Text Editor as comments starting such as a </h> or <i>. See www.keil.com/support/docs/2735.htm for instructions on how to add this feature to your own source code.

3) Adding a Thread:

TIP: RTX will be part of the new CMSIS 5 See https://github.com/ARM-software/CMSIS_5

We will create and activate a thread. We will add an additional variable counter2 that will be incremented by this new thread.

1. In main.c, add this line near line 5: unsigned int counter2 = 0;

```
5   unsigned int counter2 = 0;
```

Create the Thread job1:

2. Add this code before main():
   This will be the new thread named job1.

   osDelay(500) delays the program by 500 clock ticks to slow it down so we can easier see the values of counter and counter2 increment by 1.

```
7     void job1 (void  const *argument) {
8     for (;;) {
9               counter2++;
10                 if (counter2 > 0x0F) counter2 = 0;
11                osDelay(500);
12     }
13   }
```

Add another osDelay to main(): (main() is a thread !)

3. Add this line just after the if statement near line 21:

```
23    osDelay(500);
```

Define and Create the Thread:

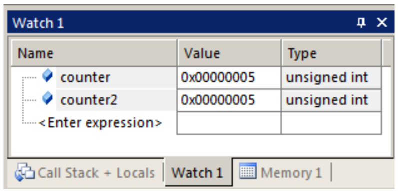
1. Add this line near line 15 just before main():

```
15    osThreadDef(job1, osPriorityNormal, 1, 0);
```

2. Create the thread job1 near line 18 just after main() and before the while(1) loop:

```
22    osThreadCreate(osThread(job1), NULL);
```

3. Click on File/Save All or
4. Build the files. There will be no errors or warnings. If there are, please fix them before continuing.
5. Enter Debug mode: Click on the RUN icon.
6. Right click on counter2 in main.c and select Add counter2 to … and select Watch 1.
7. Both counter and counter2 will increment but slower than before:
   The two osDelay(500) function calls each slow the program down by 500 msec. This makes it easier to watch these two global variables increment.



Watch 1

Name	Value	Type
counter	0x00000005	unsigned int
counter2	0x00000005	unsigned int
<Enter expression>		

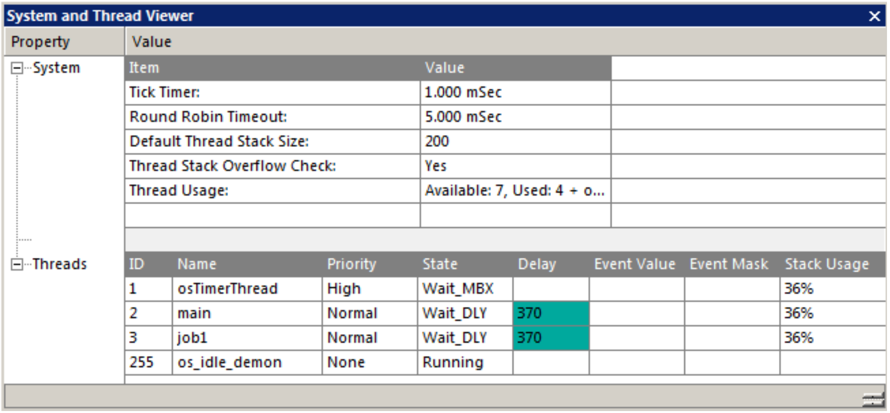
TIP: osDelay() is a function provided by RTX and is triggered by the SysTick timer.

8. Open the System and Thread Viewer by selecting Debug/OS Support.
9. Note that job1 has now been added as a thread as shown below:
10. Note os_idle_demon is always labelled as Running. This is because the program spends most of its time here.
11. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
12. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
13. There are many attributes of RTX you can add. RTX help files are located here depending on your CMSIS version: C:/Keil_v5/ARM/Pack/ARM/CMSIS/*x.x.x*/CMSIS/Documentation/RTX/html/index.html.
14. **Remove any breakpoints you have created.**

TIP: You can use this process to add more threads to your project.

On the next page, we will demonstrate the Event Viewer.

.



System and Thread Viewer

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 4 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				36%
2	main	Normal	Wait_DLY	370			36%
3	job1	Normal	Wait_DLY	370			36%
255	os_idle_demon	None	Running				

4) View RTX Thread Timing with Event Viewer:

The Event Viewer displays threads running in a graphical format. It is possible to make timing measurements.

The Event Viewer uses Serial Wire Viewer (SWV). A ULINK*pro* is best for SWV. A ULINK2/ME does a good job.

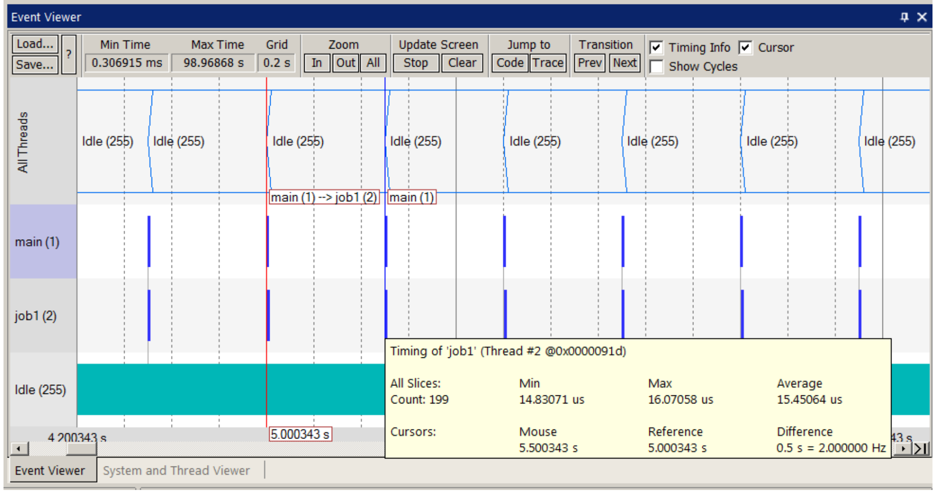
This page assumes you are running this Blinky with any ULINK or a J-Link connected to your Smartfusion2 board.

Configure SWV:

1. Stop the program and Exit Debug mode.
2. Select the Target Options icon .
3. Click on the Debug tab. Select the debugger you are using in the Use: box: This is for ULINK2: ULINK2/ME Cortex Debugger
4. Click on Settings on the right side of the target Options window. The Cortex-M Target Driver Setup window opens.
5. Set SW as shown here: Port SW JTAG does not support SWV. If your SmartFusion2 board is connected to your PC, you should now see a valid IDCODE and Device Name in the SW Device box.
   If you do not, you **must** correct this before continuing.
6. Select the Trace tab. Select Core Clock: to 133 MHz. Select Trace Enable.
7. Unselect EXCTRC to minimize potential SWO overflows. This might not be needed. You can experiment later.
8. Click OK twice to close the Target configuration windows.
9. Click on File/Save All or SWV is now configured.

Run Blinky and Open Event Viewer:

1. Enter Debug mode: Click on the RUN icon.
2. The program should be running as shown in the System and Thread Viewer as shown on the previous page. Viewing this window gives a good indication RTX is configured and running properly.
3. Select Debug/OS Support and select Event Viewer.
4. This window will open and if SWV is correctly configured, the threads will be visible.
5. Use the In and/or Out buttons to select an appropriate horizontal scale.



Min Time 0.306915 ms   Max Time 98.96868 s   Grid 0.2 s

☑ Timing Info ☑ Cursor
☐ Show Cycles

Timing of 'job1' (Thread #2 @0x0000091d)

All Slices: Count: 199	Min 14.83071 us	Max 16.07058 us	Average 15.45064 us
Cursors:	Mouse 5.500343 s	Reference 5.000343 s	Difference 0.5 s = 2.000000 Hz

6. The space between the same thread is 500 msec or 0.5 sec. This is 500 times the tick rate as set in the RTX config file. The tick rate is 1 msec.
7. Note the program spends most of its time in Idle (os demon). This can be modified in your program.

This is the end of the tutorial at this time.

Later we will add a section on using projects created with Libero.

7) Serial Wire and ETM Trace Summary:

We have three basic debug systems as implemented in SmartFusion Cortex-M3 devices:

1. SWV and ITM data output on the SWO pin located on the standard JTAG debug connector.
2. ITM is a printf type viewer. ASCII characters are displayed in the Debug printf Viewer in µVision.
3. Memory Reads and Writes in/out the JTAG/SWD ports. The Memory and Watch windows use this technology.
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.
5. ETM provides a record of the instructions executed.

These are all completely controlled through µVision via a ULINK.

These are the types of problems that can be found with a quality ETM trace:

SWV Trace adds significant power to debugging efforts. Problems which may take hours, days or even weeks in big projects can often be found in a fraction of these times with a trace. Especially useful is where the bug occurs a long time before the consequences are seen or where the state of the system disappears with a change in scope(s) or RTOS task switches.

Usually, these techniques allow finding bugs without stopping the program. These are the types of problems that can be found with a quality trace: Some of these items need ETM trace.

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes). How I did I get to this Fault vector ?
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs). ***How did I get here ?***
4) A corrupted stack.
5) Out of bounds data. Uninitialized variables and arrays.
6) Stack overflows. What causes the stack to grow bigger than it should ?
7) **Runaway programs:** your program has gone off into the weeds and you need to know what instruction caused this. ***This is probably the most important use of trace. Needs ETM to be most useful.***
8) Communication protocol and timing issues. System timing problems.

- Trace adds significant power to debugging efforts. Tells you where the program has been.
- Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time before any consequences are seen.
- Or where the state of the system disappears with a change in scope(s).
- Plus - don't have to stop the program. Crucial to some.
- A recorded history of the program execution ***in the order it happened.***
- Trace can often find nasty problems very quickly.
- Profile Analysis and Code Coverage is provided. Available only with ETM trace.

What kind of data can the Serial Wire Viewer display ?

1. Global variables.
2. Static variables.
3. Structures.
4. Can see Peripheral registers – just read or write to them. The same is true for memory locations.
5. Can see executed instructions. SWV only samples them.
6. CPU counters. Folded instructions, extra cycles and interrupt overhead.

What Kind of Data the Serial Wire Viewer can't display…

1. Can't see local variables. (just make them global or static).
2. Can't see register to register operations. PC Samples records some of the instructions but not the data values.
3. SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU and SWV can only see CPU actions.

1) Document Resources: See www.keil.com/Microsemi

Books:

1. ***NEW!*** **Getting Started with MDK 5:** Obtain this free book here: www.keil.com/gsg/
2. There is a good selection of books available on ARM: www.arm.com/support/resources/arm-books/index.php
3. µVision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab. Or select the Cortex-M processor you want in the Processor panel on the left.
5. Or search for the Cortex-M processor you want on www.arm.com.
6. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
7. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
8. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano
9. MOOC: Massive Open Online Class: University of Texas: http://users.ece.utexas.edu/~valvano/

Application Notes: www.keil.com/appnotes

1. ***NEW!*** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ 4 www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with µVision https://launchpad.net/gcc-arm-embedded
8. RTX CMSIS-RTOS Download https://github.com/ARM-software/CMSIS_5
9. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
10. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
11. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
12. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
13. ***NEW!*** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
14. ***NEW!*** ARMv8-M Architecture Technical Overview https://community.arm.com/docs/DOC-10896
15. ***NEW!*** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp

Useful ARM Websites:

1. ***NEW!*** CMSIS Standards: https://github.com/ARM-software/CMSIS_5 and www.keil.com/cmsis/
2. Forums: www.keil.com/forum http://community.arm.com/groups/tools/content https://developer.arm.com/
3. ARM University Program**:** www.arm.com/university. Email: university@arm.com
4. mbed™: http://mbed.org
5. CMSIS 5: https://github.com/ARM-software/CMSIS_5

Sales In Americas: sales.us@keil.com or 800-348-8051. **Europe/Asia:** sales.intl@keil.com +49 89/456040-20
Keil Distributors: See www.keil.com/distis/ **DS-5 Direct Sales Worldwide:** orders@arm.com
Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com
Global Inside Sales Contact Point: Inside-Sales@arm.com

For comments, additions or corrections please email bob.boys@arm.com

2) Keil Products and contact information: See **www.keil.com/Microsemi**

Keil Microcontroller Development Kit (MDK-ARM™) for Cortex-M processors:

- **MDK-Lite™** (Evaluation version) 32K Code and Data Limit - $0
- *New* **MDK-ARM-Essential™** For all Cortex-M series processors – unlimited code limit
- *New* **MDK-Plus™** MiddleWare Level 1. ARM7™, ARM9™, Cortex-M, SecureCore®.
- *New* **MDK-Professional™** MiddleWare Level 2. For details: www.keil.com/mdk5/version520.

For the latest MDK details see: www.keil.com/mdk5/selector/

Keil Middleware includes Network, USB, Graphics and File System. www.keil.com/mdk5/middleware/

USB-JTAG/SWD Debug Adapter (for Flash programming too)

- ULINK2 - *(ULINK2 and ME - SWV only – no ETM) ULINK-ME is equivalent to a ULINK2.*
- ***New*** ULINK*plus*: Hi-speed SWV, Power measurement.
- ULINK*pro* - Cortex-Mx SWV & ETM trace
- ULINK*pro* D - Cortex-Mx SWV no ETM trace ULINK*pro* also works with ARM DS-5

For Serial Wire Viewer (SWV), a ULINK2 or a J-Link is needed. For ETM support, a ULINK*pro* is needed.

Call Keil Sales for more details on current pricing. All products are available.

For the ARM University program: go to **www.arm.com/university** Email: university@arm.com

All software products include Technical Support and Updates for 1 year. This can easily be renewed.

Keil RTX™ Real Time Operating System

- RTX is provided free as part of Keil MDK.
- No royalties are required and is very easy to use. It has a BSD or an Apache 2.0 license.
- RTX source code is included with all versions of MDK.
- Kernel Awareness visibility windows are integral to µVision.
- See **https://github.com/ARM-software/CMSIS_5**

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor. For Microsemi support: www.keil.com/Microsemi

For Linux, Android, bare metal (no OS) and other OS support on Cortex-A processors, please see DS-5 and DS-MDK at www.arm.com/ds5/ and www.keil.com/ds-mdk.

For more information:

Sales In Americas: sales.us@keil.com or 800-348-8051. **Europe/Asia:** sales.intl@keil.com +49 89/456040-20

Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.

Global Inside Sales Contact Point: Inside-Sales@arm.com **World Wide Distributors:** www.keil.com/distis

For more information: www.arm.com/cmsis, www.keil.com/forum and http://community.arm.com/groups/tools/content

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™ Intelligent Processors by ARM®

ARM® KEIL® Microcontroller Tools