

Microsemi® SmartFusion®2 Lab: Cortex™-M3: using ARM® Keil™ MDK toolkit



Spring 2013 Version 1.1

featuring Serial Wire Viewer and ETM Trace

by Robert Boys bob.boys@arm.com

Hands-on lab using the Microsemi SmartFusion2 (SF2) and Keil MDK

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_238.asp

Introduction:

This note describes the process of operating the ARM® Keil™ MDK toolkit featuring μVision® and Microsemi's (Actel™) new 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 needs MDK® 4.70 or later.** You can use a Microsemi SF2-DEV-KIT or an EmCraft® SF2-Starter-Kit-ES-2. You can also adapt it to your own target board. A similar lab exists for the previous SmartFusion1 (SF1) processor A2Fxx03: www.keil.com/appnotes/docs/apnt_208.asp

Keil μVision:

MDK-ARM™ toolkit components include the μVision4 IDE, ARM C/C++ compiler, debugger and Keil RTX™ RTOS.

Keil supports all 8051, ARM7, Cortex-M1 and Cortex-M3 processors as used in various Microsemi (Actel) products.

The Keil ULINK® JTAG/SWD adapter family includes the ULINK2, ULINK-ME and the ULINKpro.

RTOS now comes with a BSD type license. Source code is provided with all versions of MDK. See www.arm.com/cmsis.

Why Use Keil MDK ?

MDK provides these features particularly suited for Microsemi MSS products: www.keil.com/microsemi

1. μVision IDE with Integrated Keil Debugger, eNVM (Flash) programmer and the ARM compiler.
2. Support for Microsemi ARM7, Cortex-M1 and Cortex-M3. Nothing more to buy. 8051 support with PK51 is a separate product and it also uses μVision as its IDE.
3. A full feature RTOS is included with MDK: RTX by Keil. No royalty payments are required.
4. RTX Kernel Awareness windows. They are updated in real-time and use no system resources.
5. Serial Wire Viewer capability is included.
6. ETM trace is supported with a Keil ULINKpro.
7. Choice of debug adapters: ULINK2, ULINK-ME, ULINKpro, Segger J-Link (black version) and CMSIS-DAP.
8. Keil Technical Support is included for one year. This helps you get your project completed faster. It is easy to renew.
9. 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 which includes SWV and ETM trace.



Keil ULINKpro connected to DEV KIT ETM

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, ULINKpro or a Segger J-Link (black case, V 6 or later).

Embedded Trace Macrocell (ETM Instruction Trace):

ETM adds a record of all instructions executed for program flow debugging and analysis. Code Coverage, Performance Analysis and Execution Profiling are provided with ETM. ETM requires a ULINKpro debug adapter and the Cortex ETM debug connector as found on the SF2-DEV-KIT. An ETM demo is shown using the Keil Simulator and the SF2-DEV-KIT.

Part 1: Introduction and Getting Ready:	3
1. CoreSight Definitions:	3
2. SmartFusion2 Development Boards:	3
3. STAPL File:	3
4. Obtaining Keil MDK toolkit 4.70 and special example files:	3
5. Microsemi SF2-DEV-KIT and EmCraft Starter-Kit Connection Issues:	4
6. Connecting the Hardware to a Keil ULINK:	5
7. Selecting JTAG or SWD (SW):	6
Part 2: Examples and Useful Information:	7
1. The Keil Blinky example: Setting a Hardware Breakpoint:	7
2. Call Stack + Locals Window:	8
3. Watch windows: Updated in Real-Time !	9
4. Memory Windows:	9
Part 3: Configuring the Serial Wire (SWV) Trace:	10
1. Configuring SWV Trace:	10
2. Testing SWV:	10
3. Trace Configuration Fields:	11
Part 4: The Rest of the Tracing Examples:	13
1. Logic Analyzer: (LA)	13
2. Data Writes in the Trace Records window:	14
3. Watchpoints (Access Breaks):	15
4. Exceptions Tracing:	16
5. PC Samples Tracing:	17
6. Debug (printf) Viewer:	18
Part 5: RTX RTOS:	19
1. Running the RTX_Blinky example:	19
2. RTX Kernel Awareness: The RTX Viewer:	20
3. Configuring the Serial Wire Viewer (SWV) Trace:	21
4. Logic Analyzer Window: View variables real-time in a graphical format:	22
Part 6: DSP Example:	23
1. DSP Sine example using the ARM CMSIS-DSP Libraries:	23
2. RTX Tasks and System Awareness window:	24
3. RTX Event Viewer window:	25
Part 7: ETM Trace Examples:	27
1. ETM Trace Examples Using the Keil Simulator:	27
2. Code Coverage	28
3. Performance Analysis:	29
4. Execution Profiling:	30
5. In-the-weeds Example:	31
6. ETM Trace Examples Using the SF2-DEV-KIT and ULINK _{pro} :	32
7. Trace Triggering with ULINK _{pro} :	33
8. More ETM Examples:	34
9. Configuring Target Options for ULINK _{pro} :	34
Part 8: ...more Useful Information	35
1. Programming the SmartFusion2 eNVM Flash with a STPL file:	35
2. Creating a new μ Vision project from scratch:	37
3. JTAG/SWD/SWO Debug Adapter Connector Schematics:	40
4. Acronym List:	41
5. Overloading the SWO pin:	41
6. Top Seven Reasons why you can't get SWV working:	42
7. Serial Wire Viewer and ETM Trace Summary:	43
8. Keil products and Contact Information:	44

This document details these features *and more*:

1. Serial Wire Viewer (SWV).
2. Real-time Read and/or Write to memory locations for Watch and Memory windows.
3. No-skid Hardware Breakpoints and Watchpoints (Access Breaks).
4. RTX RTOS operation using RTX Viewer: a kernel awareness program for the Keil RTOS – RTX.
5. A DSP example using the ARM DSP libraries. Uses the Serial Wire Viewer to display results.
6. ETM instruction trace. ETM is demonstrated using the Keil simulator and with a ULINK_{pro} on the DEV KIT.

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

- **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
- **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except for no Boundary Scan. SWD is referenced as SW in the μ Vision Cortex-M Target Driver Setup. See page 6, second screen in the Port: box. The SWJ box must be selected to be able to enable SWD.
- **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf. SWV must use SWD because of the TDIO and SWO conflict when JTAG is used.
- **DAP:** Debug Access Port. 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. μ Vision uses the DAP to update Memory, Watch and a RTOS kernel awareness window in real-time 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 in your sources. You do not need to configure or activate DAP. μ Vision does this automatically when you select the function.
- **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
- **ITM:** Instruction Trace Macrocell: 32 32 bit registers that output data to the outside world using SWV. μ Vision uses ITM 0 for a printf feature and ITM 31 for the RTX Event Viewer window. The others are not used.
- **Trace Port:** A 4 bit port (4 data and 1 clock) that ULINK_{pro} uses to collect ETM frames and optionally SWV (rather than with the SWO pin). These signals are available on the 20 pin ETM Header found on the DEV KIT.
- **ETM:** Embedded Trace Macrocell: Provides all the program counter values. Only ULINK_{pro} provides ETM.

2) SmartFusion2 Development Boards:

This document details operation with either the **1)** Microsemi SF2-DEV-KIT or the **2)** EmCraft StarterKit SF2-Starter-Kit. Both of these boards used to create this lab contain REV B of the M2S050 silicon.

The Microsemi board itself is REV B and EmCraft is Rev2A. Different versions of the board and /or silicon may need different STAPL (.stp) files or instructions. Contact Keil or Microsemi technical support. Contact the author if you find an error.

All examples will run on either board but the EmCraft board does not have access to the ETM 4 bit trace port. Both boards support Serial Wire Viewer.

3) STAPL File:

A STPL file must be programed into the SF2 eNVM in order to configure the PLL clock and other hardware aspects. This is different from the SF1 which could run with no STPL file. An example .stp file is provided by Keil to run with the examples.

4) Obtaining the Keil MDK toolkit and special example files:

This tutorial requires MDK 4.70 or later. MDK 4.70 contains all necessary files except for the DSP and ETM examples and the STAPL file described above. These are available here: www.keil.com/appnotes/docs/apnt_238.asp

MDK is downloaded here: www.keil.com/arm/demo/eval/arm.htm Install this into the default directory on your PC. No license is required. You also need a ULINK2, ULINK-ME, ULINK_{pro} or a J-Link.

Keil Contact Information: www.keil.com See the last page of this document for details.

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

5) Microsemi SF2-DEV-KIT Rev B Debug Connection Issues:

1) This board has a R555 1K Ω pulldown resistor on the nTRST pin 3 on the RVI Header J34 and FP4 Header J59. This causes the SmartFusion JTAG TAP RESET to always be asserted. This causes: a) JTAG debug will not work (SWD functions correctly) and b) the Serial Wire Viewer will not work. This is because the TAP controller outputs SWV on the SWO/TDO pin and when in RESET, SWO is disabled. nTRST must be high to release the TAP controller out of RESET and release the SWO.

Solution: The fix is to remove this R555 1K Ω resistor and move it to between pins 8 and 10 on FP4 Header J59.

This difficult to do because R555 is very small and it is located on the board bottom underneath the SmartFusion2 chip. It is possible that Microsemi has modified your board. Measure the resistance on J34 to ground to check this.

If R555 is present, you can still do all of the examples except for Serial Wire Viewer and JTAG. Just use SWD.

A Faster Option: Pull nTRST high with a jumper from J34 RVI Header Pin 1 (3.3v) to pin 3 (nTRST) OR from pin 7 FP4 Header to pin 10 on ETM Header while using a Keil ULINK. nTRST is pulled up releasing the TAP controller from RESET.

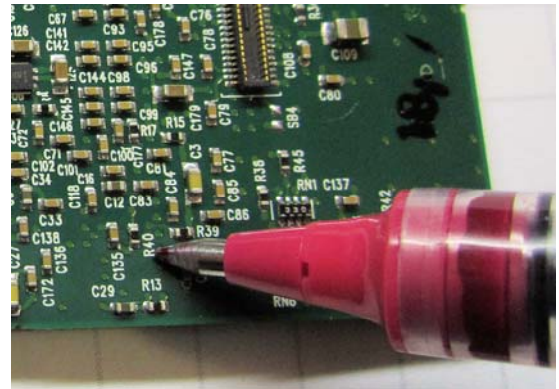
2) On the ETM Header, nTRST is connected to pin 10. This is incorrect. This pin should be connected to nSRST. This is not normally a problem because most debug adapters do not use this signal by default to RESET the SF2 processor.

EmCraft Starter-Kit Debug Connection Issues:

1) This board also has a 1K Ω pulldown resistor (R40) on nTRST with similar results as above. This resistor is easier to remove. It is on the bottom of the CPU board and is labeled R40 as shown here:

It is **not** recommended to pull nTRST high with a jumper as this could damage a FlashPro4[®] programmer. **It is best to remove resistor R40. It is easy to remove.** Small pliers will work. Or pry it off with a fingernail.

2) This board does not have the ETM header as in the DEV KIT to provide ETM trace signals. Everything else will work correctly.



For more information on Connectors:

Go to www.arm.com and search for **cortex_debug_connectors.pdf**.

Debug Jumper Summary:

Microsemi SF2 DEV KIT J93

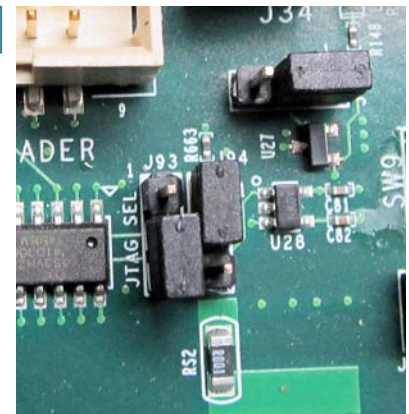
Microsemi SF2-DEV-KIT:

JTAG SEL J93:

Keil ULINK (for debug): Position 2-3 as shown here:

For Microsemi FlashPro4: Position 1-2:

All other board jumpers are set to default.



EmCraft SF2 Starter-Kit:

EmCraft SF2 Starter-Kit JP2

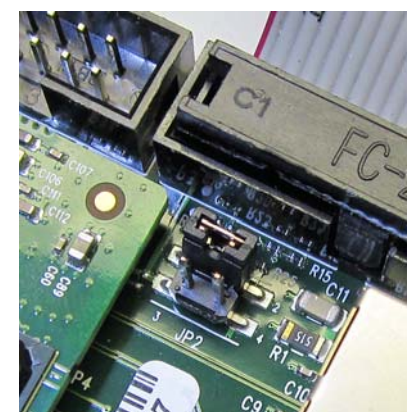
JP2:

Keil ULINK (for debug): Position 1-2 as shown below:

For Microsemi FlashPro4: No jumper is fitted.

Note: JP2 3-4 is don't care.

All other board jumpers are set to default.



6) Connecting the hardware to a Keil ULINK:

We will examine two SmartFusion2 boards and will focus on the smaller one: the EmCraft SF2 StarterKit.

Microsemi SF2-DEV-KIT: The board used in this lab was a REV B and the SF2 processor was a REV B M2S050.

See Figure 1. The ULINK2 is connected to RVI Header J34. On page 1, a ULINK_{pro} is connected to the 20 pin ETM HEADER. Below is a ULINK-ME connected to P3. A ULINK-ME is only available with an OEM board.

This board has two debug connections: RVI HEADER (J34) and ETM HEADER. RVI HEADER is the standard legacy JTAG connector. Any Keil ULINK can connect to this. A ULINK_{pro} comes with an adapter to connect to RVI HEADER.

ETM HEADER has all the necessary JTAG signals plus the Trace Port. This is 4 bits data and one clock. All ULINKs can connect using an appropriate cable, but only ULINK_{pro} can use the Trace Port for instruction trace and optionally SWV.

Jumper JTAG SEL J93: Must be set to position 2-3 as shown on the previous page to activate JTAG or SWD for a debug adapter such as a Keil ULINK2 or a ULINK_{pro}.

Position 1-2 is used to activate Microsemi FlashPro4 programmer. All other jumpers are set to default.

See the preceding page for a photo of J93.

EmCraft SF2 Starter-Kit: The board used in this lab was a REV 2A and the SF2 processor was a REV B M2S050.

The CPU board is designated M2S-SOM and the baseboard SOM-BSB-EXT by EmCraft.

See Figure 2. A Keil ULINK2 is attached to P3. The board is powered by connecting a USB cable to P1 as shown.

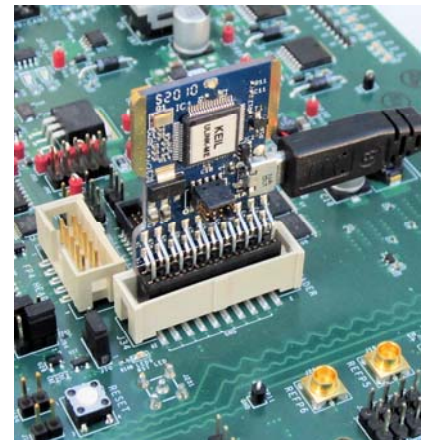
This board has only one 20 pin legacy JTAG connector P3: You must change one jumper to activate a debug adapter such as a ULINK2 or to program the FPGA Flash with a FlashPro4. A FlashPro4 is connected to P5: the 10 pin connector.

JP2 must be set to position 1-2 for a debug adapter. This is shown in the photo on the preceding page. For a FlashPro4, remove the jumper from JP2. JP2 3-4 is Don't Care.

Revisions of M2S050:

REV B: ES

REV C: PP



Microsemi SF2 DEV KIT with ULINK-ME



Figure 1: Microsemi SF2 DEV KIT with ULINK2




Figure 2: EmCraft SF2 Starter-Kit with ULINK2

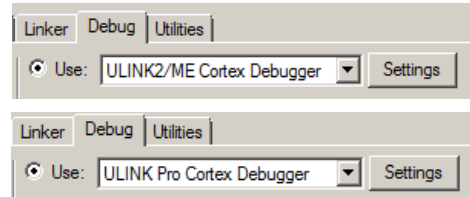
7) Selecting SWD or JTAG:

You must use Single Wire Debug (SWD) mode to use Serial Wire Viewer. This is configured in μ Vision as shown below. You must first have a ULINK connected to a Cortex-M3 target. It is useful to be able to test the JTAG or SWD connection. If you do not want to use SWV or will use a ULINK $_{pro}$ with Trace Port selected: you can use JTAG.

- 1) Connect a ULINK2, ULINK-ME or ULINK $_{pro}$ to the hardware as described. Set the JTAG jumper. See page 4.
- 2) Power the board as appropriate.
- 3) Start μ Vision by clicking on its desktop icon.



- 4) Click on the Target Options icon  or select “Project/Options for Target” or pressing Alt+F7.
- 5) Click on the Debug tab. Here is where you select your debug adapter:
- 6) Select either ULINK2/ME or ULINK $_{pro}$ Cortex Debugger:
- 7) Click Settings to open the window below:



The box labeled SW Device (or labeled JTAG Device Chain if JTAG is selected) will display the CoreSight devices detected in the processor or an error if none are found.

- 8) Alternate between JTAG and SW in the Port: box and confirm correct operation. SW-DP or JTAG will be displayed. Finally select SW as shown below for proper SWV operation. You can use SWD for all debugging operations.

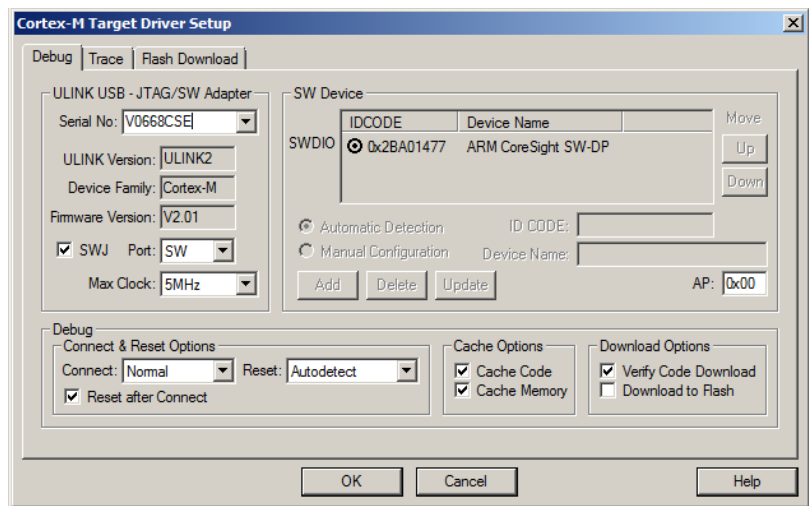
TIP: If you are unable to select JTAG yet SWD (SW) works properly, the issue might be the 1K Ω pulldown on the nTRST pin. See page 4 for the solution.

The ULINK is now successfully been connected to the CoreSight SWD port. The Serial No: box shows the link between your PC and the ULINK.

Urgent TIP: If you don’t see either SW-DP or JTAG-DP in this window and get an error message or device not found – stop right now and fix this. This section **must** work before you can continue any debugging. **Make sure the jumpers are set correctly** and the board is properly powered. Try re-powering the board.


- 9) Click on OK twice to return.
- 10) Run Blinky on the next page.

TIP: To refresh this window switch between JTAG and SW in the Port: drop down menu.



TIP: If you select ULINK or ULINK $_{pro}$, and have the opposite ULINK actually connected to your PC; the error message will say “No ULINK device found”. This message actually means that μ Vision found the wrong Keil adapter connected, and not that no ULINK was attached. Select the correct ULINK.

TIP: You can also use a Segger J-Link (black case). MDK completely supports this J-Link.

Super TIP: If you change the debug adapter here, you must also change it in the Utilities tab. This selects the eNVM flash programming algorithm. This tab is visible when you open the Target Options window. 






1) *Blinky* example program using Microsemi SmartFusion2:

Now we will run the Keil MDK development system using a SmartFusion2 DEV KIT or EmCraft evaluation board.

A STAPL file needs to be programmed into the SmartFusion2 processor. Instructions are on page 35. You can try the Blinky example using the STAPL file that is probably loaded into your SF2 by the board manufacturer. It might work.

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

Compile, Load and Run the example Blinky program:

1. Select Project/Open Project from the main menu.
2. Open the file C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\Blinky\M2S050T_MSS_CM3.uvmpw.
3. In the Project window, right click on M2S050T_MSS_CM3_hw_platform and click Set as Active Project: This project will take on a black highlight background as shown here:
4. Compile the source files by clicking on the Rebuild icon.  Ignore the one warning. It is inconsequential.
5. In the Project window, right click on M2S050T_MSS_CM3_app and click Set as Active Project. It will take on a black background.
6. Compile the source files by clicking on the Rebuild icon. 
7. Program the SmartFusion2 eNVM flash by clicking on the Load icon:  Progress is indicated in bottom left corner.
8. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.
9. Click on the RUN icon to start the Blinky program.  **Blinky is now running !**

Leds DS2 and DS3 on EmCraft or LED1 through LED8 on the DEV KIT will blink if the FPGA is programmed to configure the SF2 properly. If not, continue to step 11. Also see **What if this does not work ?** below.



Now you know how to compile a program, load it into the SmartFusion2 eNVM Flash and run it.

Confirm the program is running: Set a hardware breakpoint:

10. Bring Blinky.c in focus by clicking on its tab. If it is not visible, double-click on it in the Project window.
11. Scroll down to near Line 68: `Delay(200);`



```
67 LED_On (idx); /* Turn on LED 'idx'
68 Delay(200); /* Delay 200ms
69 LED_Off(idx); /* Turn off LED 'idx'
70 Delay(100); /* Delay 100ms
71 }
```

12. 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.
13. *While the program is still running*, click to the left of the 68 and a red circle will be created. This is a hardware breakpoint. SmartFusion2 has six you can use. μ Vision will warn you if you exceed this limit.
14. Note that the program will soon stop at line 68 or the one 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.
15. If the program does not stop: go to **What if this does not work ?** below.
16. This is proof the program is running properly even if the LEDs do not blink.
17. Run the program again.  Stop the program with the STOP icon.  Leave the breakpoint enabled.
18. Note this program spends most of its time in the Delay function.

What if this does not work ?

1. See page 35 for FPGA fabric programming. An stp file needs to be programmed into the FPGA fabric. The EmCraft board stp is available from www.emcraft.com and normally is pre-programmed with this file. The stp file for the SF2-DEV-KIT is available at www.keil.com/appnotes/docs/apnt_238.asp. See page 35 for programming instructions.
2. See also the JTAG and SWD configuration test on the previous page. JTAG works only if R40 is removed. SWD will work with this resistor installed.


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.

1. We will use the breakpoint you set on the previous page near line 68 at Delay(500) in Blinky.c.

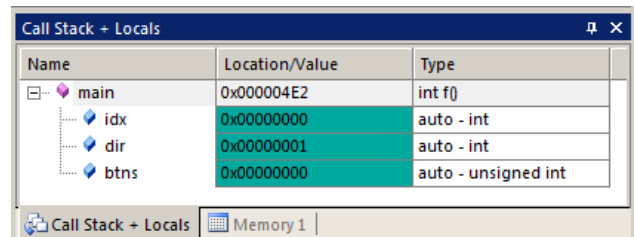
2. Run Blinky.  The program will soon stop on this breakpoint.

3. Click on the Call Stack + Locals tab if necessary to open it.


4. Shown is the Call Stack + Locals window. 

5. The contents of the local variables are displayed as well as function names.


6. In this example, three local variables **idx**, **dir** and **btns** are displayed in the window here with their values:




Name	Location/Value	Type
main	0x000004E2	int f()
idx	0x00000000	auto - int
dir	0x00000001	auto - int
btns	0x00000000	auto - unsigned int

7. Click on the Step icon or F11:  Make sure the Blinky.c window is in focus else you will step in assembly.

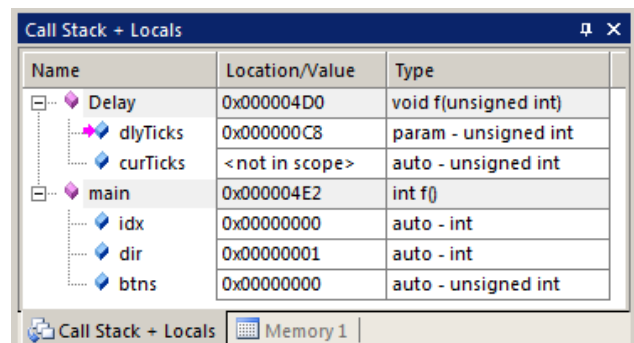
8. The program enters the **Delay** function. The Call Stack + Locals window will now display this event as shown here:

9. Click on the Step icon  or F11 multiple times until you re-enter the Delay function. Note along the way various other functions are displayed.

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.

10. 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.

11. **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.**



Name	Location/Value	Type
Delay	0x000004D0	void f(unsigned int)
dlyTicks	0x000000C8	param - unsigned int
curTicks	<not in scope>	auto - unsigned int
main	0x000004E2	int f()
idx	0x00000000	auto - int
dir	0x00000001	auto - int
btns	0x00000000	auto - unsigned int

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 ULINKpro in conjunction with the Call Stack provides much more debugging power.


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





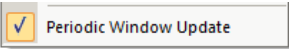
TIP: You can access the Hardware Breakpoint table by clicking on Debug/Breakpoints or Ctrl-B. This is also where Watchpoints (also called Access Points) are configured. You can temporarily disable entries in this table by unchecking them. You can permanently delete them by using one of the Kill buttons.

3) Watch Window: Updated in Real-time !

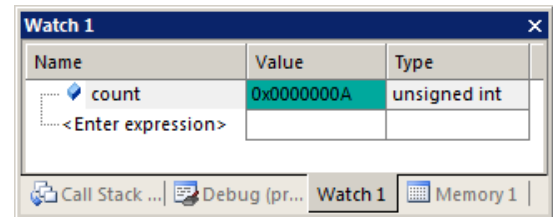
We want to display a variable. The Watch and Memory windows can be displayed while the program is running. These must be global, static, a physical address, a structure or array or anything that visible in all functions. Locals will not be displayed until the program is stopped in the function they exist 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 Blinky.c, declare a global variable near line 22: `unsigned int count = 0;`
3. Just after the C source line `Delay(200);` near line 68 add these lines:

```
count++;
if (count > 0x0F) count = 0;
```
4. Compile the source files by clicking on the Rebuild icon. .
5. Program the SmartFusion2 eNVM flash by clicking on the Load icon: . Progress is indicated in bottom left corner.
6. Enter Debug mode by clicking on the Debug icon. . Select OK if the Evaluation Mode notice appears.
7. Click on the RUN icon to start the Blinky program. .
8. Select View and select Periodic Windows Update: .
9. Right-click on the variable name `count` and select Add 'count' to... and then select Watch 1.
10. The value of `count` will be displayed and updated in real-time without stealing any CPU cycles as shown here:
11. The value of `count` displayed depends on when it is sampled.

TIP: To Drag 'n Drop into a tab that is not active such as the Logic Analyzer, Watch or Memory windows, pick up the variable by blocking it, click and hold, and move it over the tab you want to open; when the tab opens, move your mouse into the window and release the variable.



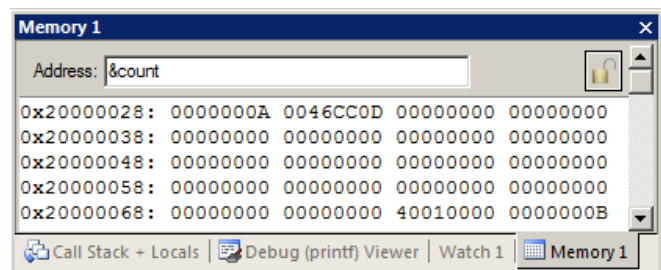
4) Memory Window: it is also updated in Real-time:

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.
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.

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.

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.




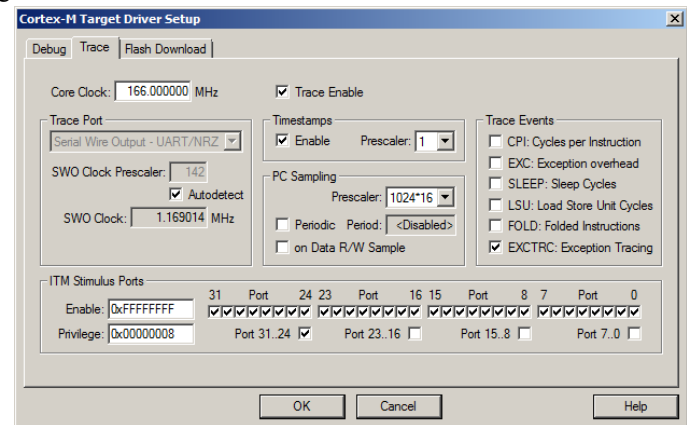
1) Configuring the Serial Wire Viewer (SWV) Trace with ULINK2/ME:



In order to see the Serial Wire Viewer features, the Trace must first be configured.

- 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.
- 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:
- 6) Set Core Clock to 50, 100 or 166 MHz.

TIP: The Core Clock: value is set in the STAPL file. This value has to be set correctly so SWV will work. ULINKpro detects this value automatically. It uses the Core Clock: value you enter for some timing values.


- 7) Select Trace Enable and EXCTRC.
- 8) Unselect Periodic and on Data R/W Sample. The window will look similar to this: 
- 9) Click on OK twice to return to the main screen.
- 10) The Serial Wire Viewer is now activated.
- 11) Click on File/Save All to save these settings.



- 12) Enter Debug mode  and click on RUN .

TIP: ULINKpro and J-Link has a slightly different configuration window. If using a ULINKpro, do not select ETM now.

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

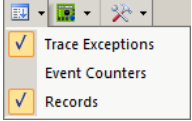
- 1) **In Edit mode:** The Target Options icon as already described. .
- 2) **In Debug Mode:** Select Debug/Debug Settings from the main menu.

TIP: You can 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.

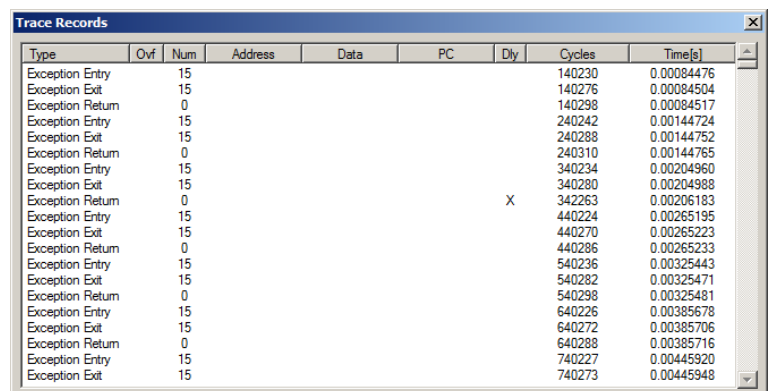
ULINKpro processes SWV information faster than a ULINK2. It uses Manchester encoding on the SWO pin or the Trace Port. The next page describes in detail the various fields of the Trace Configuration window.

2) Testing SWV:

1. Open the Trace Windows by selecting the small arrow beside its icon.
2. Select Trace Exceptions and repeat for Records as shown here: 
3. The Trace Records window will display Exception 15 which is the SysTick timer.
4. Double-click anywhere inside the window to clear it.
5. The Exceptions window will display Exceptions. Click the NUM column to order the interrupts. Note both windows update while the program is running.

What if it doesn't work ?

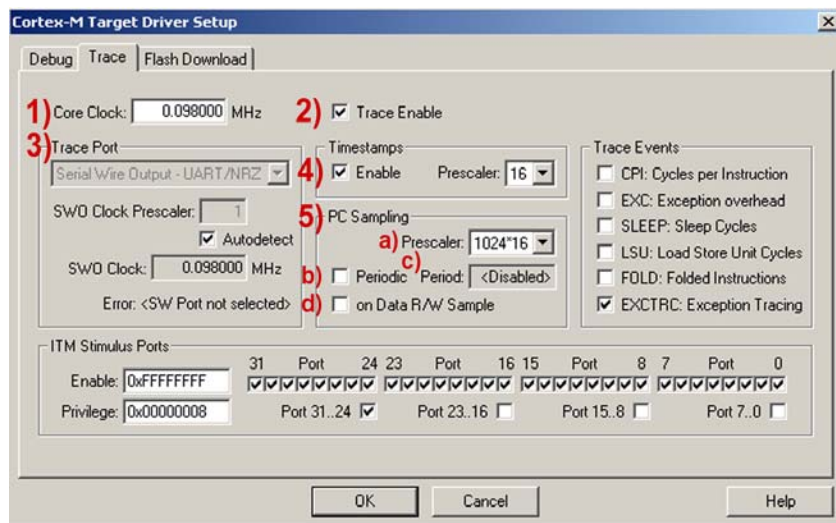
1. If you see either nothing or spurious frames such as ITM frames with numbers other than 0 or 31, or other frames that do not make sense, the Core Clock: value is probably wrong. Try entering 100 MHz. You need to know your CPU clock frequency.
2. If nTRST is pulled down with a 1KΩ resistor, the SWO pin will not output any data even though you can control the processor with SWD. See page 4 for information on fixing this problem.



Type	Out	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					140230	0.00084476
Exception Exit		15					140276	0.00084504
Exception Return		0					140298	0.00084517
Exception Entry		15					240242	0.00144724
Exception Exit		15					240288	0.00144752
Exception Return		0					240310	0.00144765
Exception Entry		15					340234	0.00204960
Exception Exit		15					340280	0.00204988
Exception Return		0				X	342263	0.00206183
Exception Entry		15					440224	0.00265195
Exception Exit		15					440270	0.00265223
Exception Return		0					440286	0.00265233
Exception Entry		15					540236	0.00325443
Exception Exit		15					540282	0.00325471
Exception Return		0					540298	0.00325481
Exception Entry		15					640226	0.00385678
Exception Exit		15					640272	0.00385706
Exception Return		0					640288	0.00385716
Exception Entry		15					740227	0.00445920
Exception Exit		15					740273	0.00445948

3) Trace Configuration Fields: *For reference...for ULINK2/ME*

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



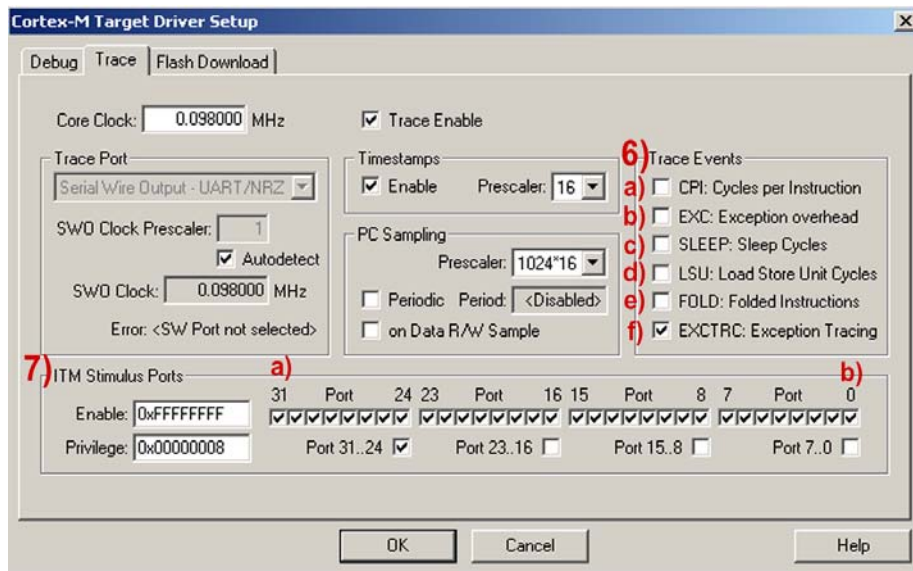
Trace Configuration window Fields 1) through 5).

- 1) **Core Clock:** The CPU clock speed for SWV. SWO Clock signal is derived from and is a ratio of the Core Clock. Setting for EmCraft.stp is 166 MHz and the DEV-KIT .stp is 100 MHz. This frequency is determined by the .stp file.
- 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, ULINK-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.



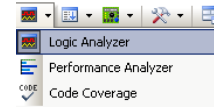
Trace Configuration window Fields 6) through 8)


- 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 Trace window. The event created when a counter wraps around is displayed in the Trace Records window or the Instruction Trace window (ULINKpro).
Event Counters are updated using the DAP and not SWV. These events are memory mapped and can be read by your program or the μ Vision 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 μ Vision. 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 μ Vision 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.


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

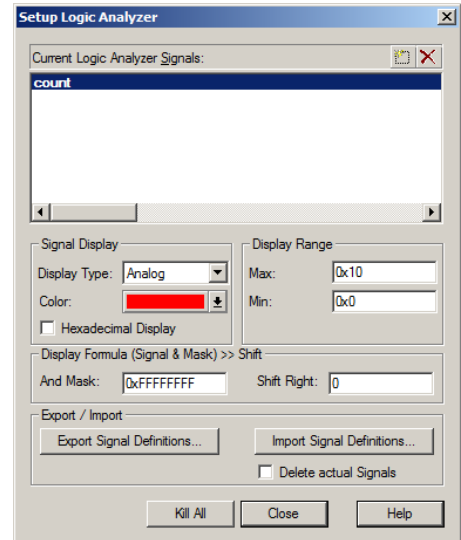
The Logic Analyzer (LA) displays up to four variables in a graphical format. When a variable is placed in the LA, it is also displayed in the Trace Records window. The LA needs SWV properly configured for its operation.

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



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

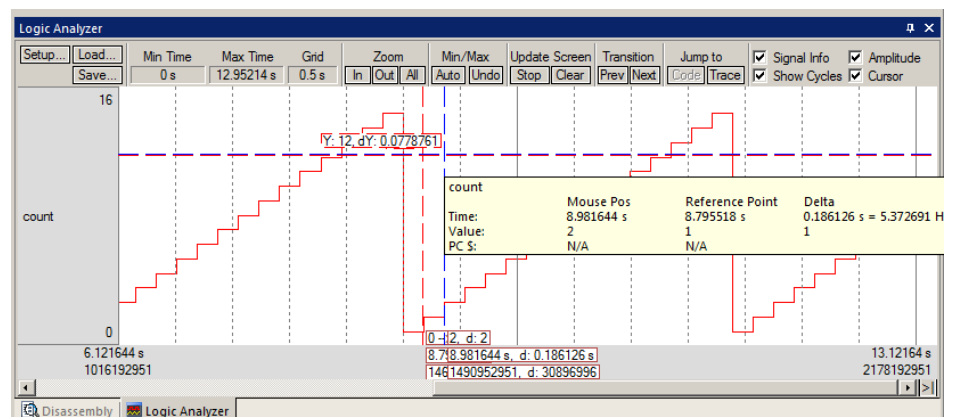
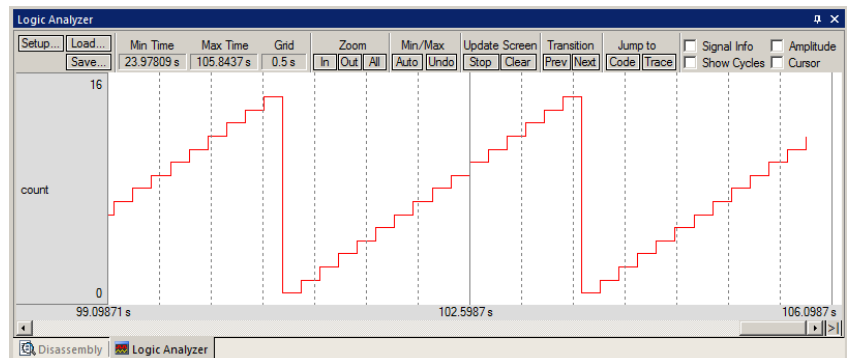
5. In the LA, click on Setup... This window opens up: 
6. Select the Display Range max = 0x10 and Min = 0. Click on Close.
7. Click on RUN if the program is not already running. The LA window will be updating as shown below:
8. If necessary, click on Zoom: Out for an appropriate X axis. Try 0.5 seconds or so.
9. Add `&count` to the Memory window if it is not already there.
10. 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.
11. Select the four boxes on the right of the LA.
12. Select Stop in Update Screen. This stops the LA but not your program. This is a very useful feature.
13. Hover the mouse over the waveform. By clicking and setting the cursor, you can determine the various timings of this program.
14. In my example, each step is 0.1891 seconds long. See the bottom screen:



The actual value depends on the board and STAPL file you are using, hence the CPU clock speed.

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.




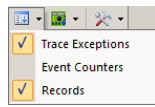

Logic Analyzer Notes:

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 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 μ Vision.

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. Double-click in the Trace Records window to clear it.
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} Data Trace window, double-clicking on a data write will take you to 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.

3) Watchpoints: (also known as Access Points)


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

Watchpoints are also known as Access 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 4 Watchpoints. Watchpoints must be configured in Debug mode and with the CPU halted. They are shared with the Logic Analyzer.

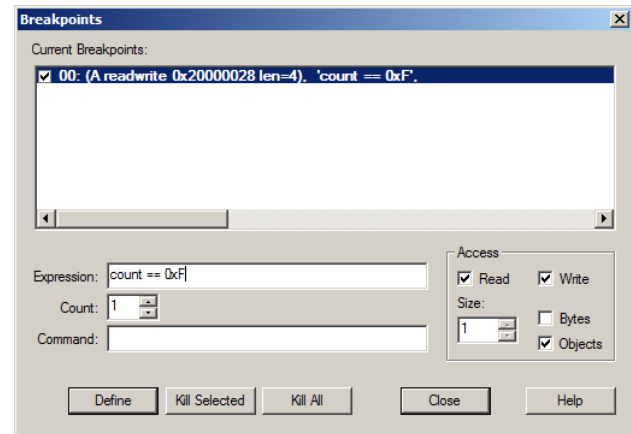
TIP: You can configure one Watchpoint if only one or two variables are listed in the LA. If you have three or four variables in the LA you are not able to configure a Watchpoint. This is a CoreSight Cortex-M3 limitation.

How to set a Watchpoint:

1. Stop the program  and remain in Debug mode.
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 0xF 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 0xF 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 (0xF) is displayed as shown below: This is the Trace Data from a ULINKpro 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.



Trace Data				
Display: All in All				
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	
0.688 377 406 s	W: 0x20000028	0x00000004	X: 0x00000582	
0.875 877 394 s	W: 0x20000028	0x00000005	X: 0x00000582	
1.063 377 356 s	W: 0x20000028	0x00000006	X: 0x00000582	
1.250 877 294 s	W: 0x20000028	0x00000007	X: 0x00000582	
1.438 377 344 s	W: 0x20000028	0x00000008	X: 0x00000582	
1.625 877 344 s	W: 0x20000028	0x00000009	X: 0x00000582	
1.813 377 319 s	W: 0x20000028	0x0000000A	X: 0x00000582	
2.000 877 350 s	W: 0x20000028	0x0000000B	X: 0x00000582	
2.188 377 344 s	W: 0x20000028	0x0000000C	X: 0x00000582	
2.375 877 381 s	W: 0x20000028	0x0000000D	X: 0x00000582	
2.563 377 381 s	W: 0x20000028	0x0000000E	X: 0x00000582	
2.750 877 356 s	W: 0x20000028	0x0000000F	X: 0x00000582	


4) Exception Tracing:

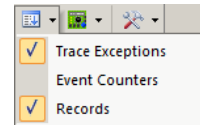
Serial Wire Viewer displays exceptions and interrupts in real-time without stealing any CPU cycles or needing instrumentation code. Note that ARM interrupts are a subset of exceptions.

The Keil examples often have the SysTick timer running. We will use this exception as an example.

1. Assume Blinky.c is loaded and μ Vision is in debug mode. The processor can be running or stopped.
2. Open the Trace Configuration window by clicking on Debug/Debug Settings. Select the Trace tab.
3. Enable EXCTRC: Unselect Periodic and on R/W Sampling. Click on OK twice to return.
4. Click on RUN to start the program.

Exceptions Trace Window:

5. Open the Exception Trace window and ensure the Trace Records window is still open.
6. The Trace Exceptions window will display the SysTick timer as shown below. Click the Clear icon: 
7. 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									
N	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NonMaskable	0	0 s						
3	HardFault	0	0 s						
4	MemoryManagem...	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	0	0 s						
12	DebugMonitor	0	0 s						
14	PendSV	0	0 s						
15	SysTick	69266	20.399 ms	277.108 ns	602.530 us	601.952 us	602.307 us	8.17976036	49.90686899
16	ExtIRQ_0	0	0 s						
17	ExtIRQ_1	0	0 s						
18	ExtIRQ_2	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.

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-M3 tail-chaining.

Trace Records with ULINK2 or a ULINK-ME:

Trace Records									
Type	Ofv	Num	Address	Data	PC	Dly	Cycles	Time[s]	
Exception Entry		15					11065040249	66.65686897	
Exception Exit		15					11065040295	66.65686925	
Exception Return		0					11065040311	66.65686934	
Exception Entry		15					11065140239	66.65747132	
Exception Exit		15					11065140285	66.65747160	
Exception Return		0					11065140301	66.65747169	
Exception Entry		15					11065240251	66.65807380	
Exception Exit		15					11065240297	66.65807408	
Exception Return		0					11065240313	66.65807417	
Exception Entry		15					11065340241	66.65867615	
Exception Exit		15					11065340287	66.65867643	
Exception Return		0					11065340303	66.65867652	
Exception Entry		15					11065440253	66.65927863	
Exception Exit		15					11065440299	66.65927891	
Exception Return		0					11065440315	66.65927901	
Exception Entry		15					11065540243	66.65988098	
Exception Exit		15					11065540289	66.65988126	
Exception Return		0					11065540305	66.65988136	
Exception Entry		15					11065640244	66.66048340	
Exception Exit		15					11065640290	66.66048367	

4. Shown is the Trace Data window using a ULINK pro . Currently, you must stop the program to update this window.
5. Select filter options in the Display: menu.

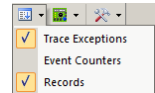
Trace Records with ULINK pro

Trace Data				
Display:	All			
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	
2.936 501 569 s		Exception Exit - SysTick		
2.936 501 669 s		Exception Return		
2.937 126 356 s		Exception Entry - SysTick		
2.937 126 644 s		Exception Exit - SysTick		
2.937 126 744 s		Exception Return		
2.937 751 294 s		Exception Entry - SysTick		
2.937 751 581 s		Exception Exit - SysTick		
2.937 751 681 s		Exception Return		

5) PC Samples Tracing:

µVision can display a sampling of the program counter values in the Trace Records window. PC Samples is enabled in the Trace Configuration window. This window is opened with View/Trace/Records or this drop down menu here:

1. Open the Trace Configuration window by clicking on Debug/Debug Settings. Select the Trace tab.
2. Enable PC Samples by checking Periodic in the PC Sampling area.
3. Enable EXCTRC:
4. Click on OK twice to return.
5. Click on RUN to start the program.
6. Open the Trace Records window. PC Samples will be displayed as shown below.



ULINK2 Trace Records Window:

The memory values displayed in the PC column gives you an indication where the program is spending its time. Note in this window we have PC Samples, Exception 15 (SysTick) and the data write from the global variable `count`. Note the data write causes a time delay ("x" in Dly column). This is an enormous amount of information to send out the one pin SWO pin. Probably, this is too much. It is always a good idea to display only that data you really need or corrupt data might result.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					00000452H		6572288445	65.72288445
PC Sample					00000456H		6572304829	65.72304829
PC Sample					00000456H		6572321213	65.72321213
PC Sample					00000456H		6572337597	65.72337597
PC Sample					00000456H		6572353981	65.72353981
Exception Entry	15				6572359790		65.72359790	65.72359790
Exception Exit	15				6572359824		65.72359824	65.72359824
Exception Return	0				6572359840		65.72359840	65.72359840
Data Write			2000001CH	0000000DH		X	6572363542	65.72363542
PC Sample					00000452H		6572370365	65.72370365
PC Sample					00000456H		6572386749	65.72386749
PC Sample					00000456H		6572403133	65.72403133
PC Sample					00000456H		6572419517	65.72419517
PC Sample					00000456H		6572435901	65.72435901
PC Sample					00000456H		6572452285	65.72452285
Exception Entry	15				6572459803		65.72459803	65.72459803
Exception Exit	15				6572459837		65.72459837	65.72459837
Exception Return	0				6572459853		65.72459853	65.72459853
PC Sample					00000452H		6572468669	65.72468669
PC Sample					00000452H		6572485053	65.72485053

PC Samples from the ULINK2 or ULINK-ME

TIP: If you want to see a record of all the instructions executed: a ULINK_{pro} does this using ETM trace.

ULINK_{pro} Instruction Trace Window: The ULINK_{pro} provides the Instruction Trace window shown below. The program was stopped, as you can see, just after the printf statement was entered. Note disassembled instructions are shown and if available, source code will also be displayed. If you double-click on a PC Sample you will be taken to that instruction in the Disassembly and/or source window. The ULINK_{pro} can process SWV data very quickly and therefore is much less susceptible to data overruns. This was taken on the SF2-DEV-KIT. The EmCraft board does not support ETM trace or the Trace Port.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x00000618	AND r4,r2,#0x01	gpio_setting = (value & 0x01u) << gpio_idx;
	X: 0x0000061C	LSL r4,r4,r3	
	X: 0x00000620	ORR r1,r4,r1	
1.250 159 637 s	X: 0x00000624	LDR r4,[pc,#296] ; @0x00000750	GPIO->GPIO_OUT = gpio_setting;
	X: 0x00000626	STR r1,[r4,#0x88]	
1.250 159 813 s	X: 0x0000062A	POP {r4,pc}	}
1.250 159 913 s	X: 0x00000490	POP {r4,pc}	}
	X: 0x000004B0	ADDS r4,r4,#1	
	X: 0x000004B2	CMP r4,#0x08	
1.250 159 938 s	X: 0x000004B4	*BLT 0x0000049A	
1.250 160 037 s	X: 0x000004B6	POP {r4-r6,pc}	}
1.250 160 063 s	X: 0x000005B0	ADR r0,[pc]+4 ; @0x000005CC	printf ("Hello World\n\r");

Instruction Trace window (ETM) from ULINK_{pro}.

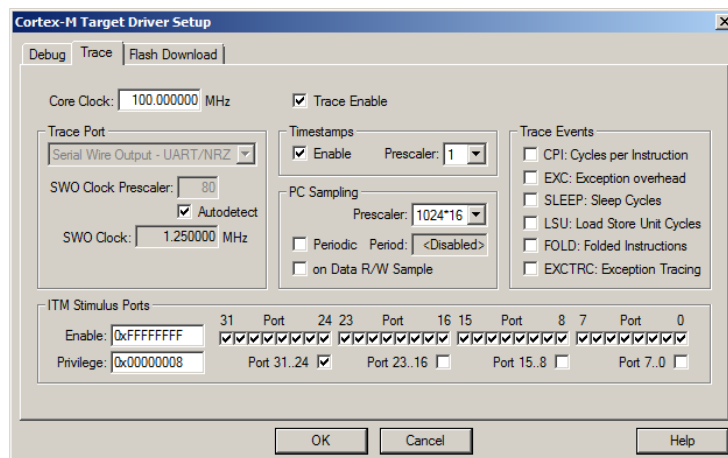
6) Debug (printf) Viewer:

ITM Stimulus Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the ASCII data write to the ITM port, zero CPU cycles are required to get the data out of the processor and into the μ Vision Debug (printf) Viewer for display. You write ASCII values to ITM Port 0.

1. Stop the program and exit debug mode if necessary.
2. Add this code to Blinky.c. A good place is right after the place where you declared `count` variable: (the following code is a pointer to ITM Port 0)

```
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
```
3. Near line 71 enter these three lines: Just after the line `count++;` that you entered is a good place.

```
ITM_Port8(0) = count + 0x30;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
```
4. Rebuild the source files, program the eNVM Flash memory and enter Debug mode.
5. Open Debug/Debug Settings and select the Trace tab. Confirm ITM Port 0 is selected. Unselect Periodic, EXCTRC and on Data R/W Sample to save SWO bandwidth. Core Clock: is determined by the STAPL file. Try 50, 100 or 166 MHz until you get valid SWV frames in the Trace Records window. Click OK twice.



SWV Trace Configuration for printf

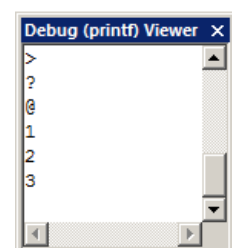
6. Click on View/Serial Windows and select Debug (printf) Viewer.
7. Make sure View/Periodic Update is enabled or the window will only update when you stop the program.
8. Click on RUN.
9. In the Debug (printf) Viewer you will see the value of `count` appear every few seconds as shown here: You could add more ASCII characters and CR and LF to create your own messages.

Trace Records

These writes to ITM 0 will also be displayed in the Trace Records window.

ITM Conclusion

The writes to ITM Stimulus Port 0 are intrusive and are usually one cycle. It takes zero additional CPU cycles to get the data displayed in the Debug (printf) Viewer window.



TIP: ITM_SendChar is a Keil supplied function you can use to send characters to the Debug Viewer. It is found in the header `core_CM3.h` in the directory `C:\Keil\ARM\CMSIS\Include`.






TIP: `retarget.c` contains routines to write a `printf` "Hello World" to J198, the RS232 connector on the DEV KIT. To get this to work on the EmCraft board USB port P1, define `MICROSEMI_STDIO_THRU_MMUART0`. This is easily entered in Target Options: open the C/C++ tab and place the line in the Define box. It might be at 191,232 baud.

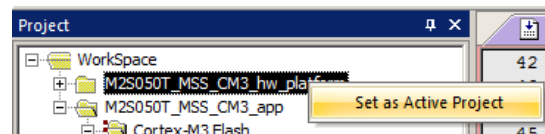
1) Running the RTX_Blinky example:

Now we will run the Keil MDK development system using either the DEV KIT or EmCraft board. RTX_Blinky is a stepper motor controller example. The program is developed using RTX to control four tasks to output the motor signals.

A STAPL file needs to be programmed into the SmartFusion2 processor. Usually one is already programmed by default.


Compile, Load and Run the example Blinky program:

1. Select Project/Open Project from the main menu.
2. Open the project: C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\RTX_Blinky\M2S050T_MSS_CM3.uvmpw.
3. In the Project window, right click on M2S050T_MSS_CM3_hw_platform and click Set as Active Project: This project will take on a black highlight background as shown here:
4. Compile the source files by clicking on the Rebuild icon.  Ignore the one warning. It is inconsequential.
5. In the Project window, right click on M2S050T_MSS_CM3_app and click Set as Active Project. It will take on a black background.
6. Compile the source files by clicking on the Rebuild icon. 
7. Program the SmartFusion2 eNVM flash by clicking on the Load icon:  Progress is indicated in bottom left corner.
8. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.
9. Click on the RUN icon to start the RTX_Blinky program.  **RTX_Blinky is now running !**
10. Leds DS2 and DS3 on the EmCraft board or LED1 through LED8 on the DEV KIT will blink indicating the four motor phases if the SF2 FPGA fabric is programmed to configure the SF2 properly.



If the Leds do not blink: Confirming RTX_Blinky is running:

Blinky.c contains four tasks, phasea through phased, that turn on and off various leds on the development boards.

1. Set a breakpoint in one of these tasks.
2. The program will soon stop at this point if it is running correctly as shown in Blinky.c and Disassembly windows.
3. Set another breakpoint in a different task and click on RUN. 
4. As you click on RUN, the program will stop at each task in turn.
5. This confirms RTX_Blinky is running properly. If the Leds do not blink: the STAPL file is probably the wrong one.
6. When you are done, remember to remove the breakpoints.

RTX Notes:

RTX is a full feature RTOS. It now has a BSD type license and source code is provided with Keil or at www.arm.com/cmsis.

Developers often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure by the RTOS. μ Vision's RTX Viewer is a kernel awareness feature with live update. It is a two window system. They respectively use the Serial Wire Viewer ITM Stimulus Port and the DAP (Debug Access Port). If μ Vision detects RTX running, the configuration of the RTX windows is automatic. You need only to select them for viewing. See the next page.

RTX Viewer windows are available only while in Debug mode by selecting Debug/OS Support. RTX is easy to configure.

Configuring μ Vision to activate RTX Kernel Awareness:




The RTX Viewer system has two components: the RTX Tasks and System window which uses the DAP for access and the Event Viewer which uses SWV (Serial Wire Viewer) through the ITM Stimulus Port # 31. The table below outlines the mandatory configuration items.

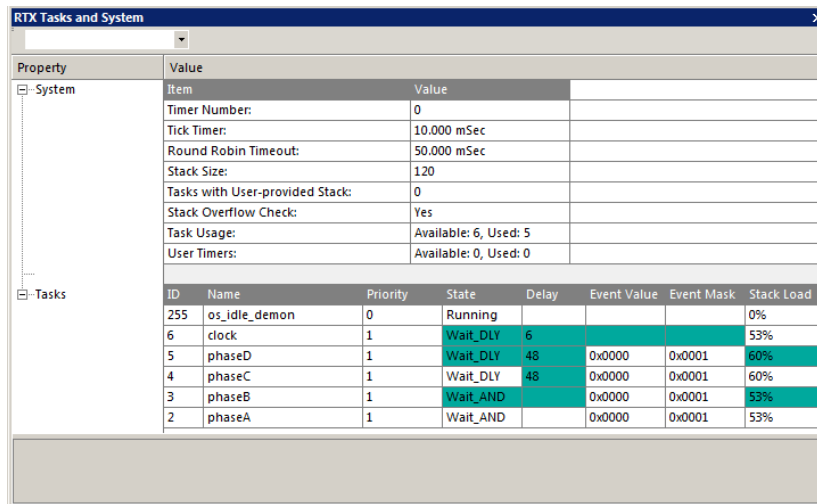
μ Vision configuration items needed for RTX Viewer:

Window\setting	RTX Running	Periodic Update	SWD	SWV	ITM
RTX Tasks & System	Yes	Yes	Yes	No	No
Event Viewer	Yes	No	Yes	Yes	# 31

2) RTX Kernel Awareness windows RTX Viewers:

RTX Tasks and System Window:

1. Remove any breakpoints you may have set (Ctrl-B Kill All) and run the program.
2. Run the RTX_Blinky program by clicking on RUN. 
3. Select View and ensure Update Periodic Window is highlighted.  Periodic Window Update
4. Select Debug/OS Support and choose RTX Tasks and System.
5. The window below will open up and various elements will change. You can drag it into the middle of your screen.
6. If the RTX Tasks and System window is updated only when you stop the program: check View/Update Periodic Window. This setting will not affect the Event Viewer.
7. Note the tasks are listed. No task, other than the idle daemon, ever shows as running or waiting. This is because the program spends nearly all of its time in idle. Set a breakpoint in one of the 4 tasks and it will be display as running.
8. Delete all the breakpoints you set. Ctrl-B and Kill All. Click on RUN. 



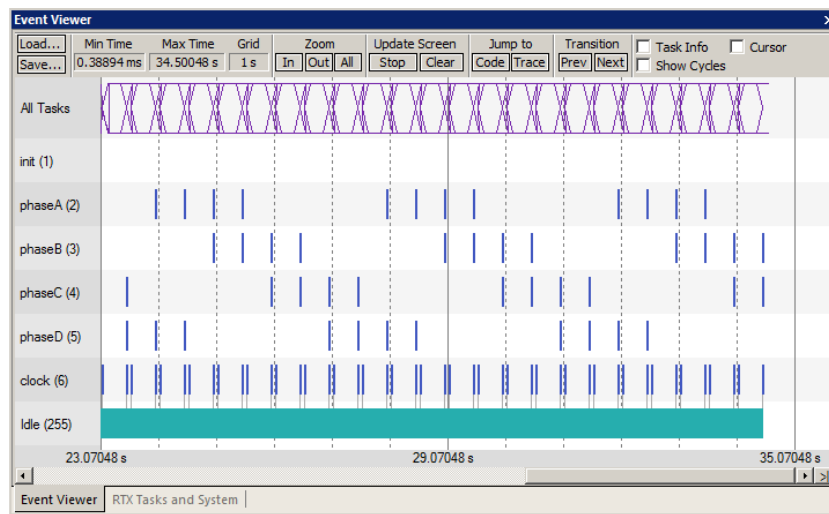
Property	Value
System	
Item	Value
Timer Number:	0
Tick Timer:	10.000 mSec
Round Robin Timeout:	50.000 mSec
Stack Size:	120
Tasks with User-provided Stack:	0
Stack Overflow Check:	Yes
Task Usage:	Available: 6, Used: 5
User Timers:	Available: 0, Used: 0

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
6	clock	1	Wait_DLY	6			53%
5	phaseD	1	Wait_DLY	48	0x0000	0x0001	60%
4	phaseC	1	Wait_DLY	48	0x0000	0x0001	60%
3	phaseB	1	Wait_AND		0x0000	0x0001	53%
2	phaseA	1	Wait_AND		0x0000	0x0001	53%

RTX Viewer: The RTX Tasks and System window.

Event Viewer Window:





1. Select Debug/OS Support and choose Event Viewer.
2. The window below opens up and it is probably blank. This is because the SWV must be configured. We will do this on the next page. If you see data as shown below, then SWV is working and you can see each task running.



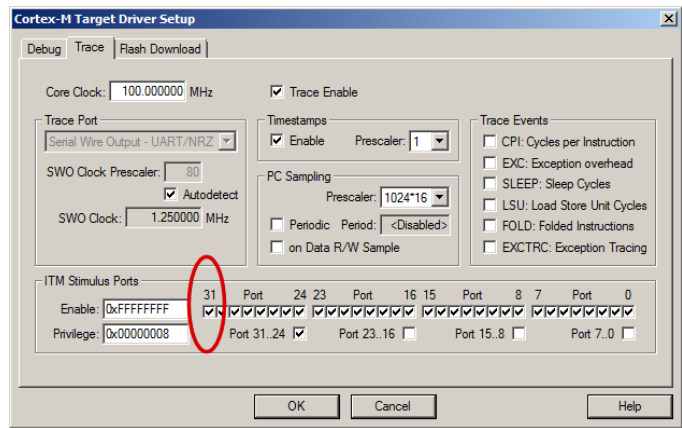
RTX Viewer: The Event Viewer window.



3) Configuring the Serial Wire Viewer (SWV) Trace:

TIP: The Trace Configuration must be configured for each Target Options. It is not set globally in μ Vision.

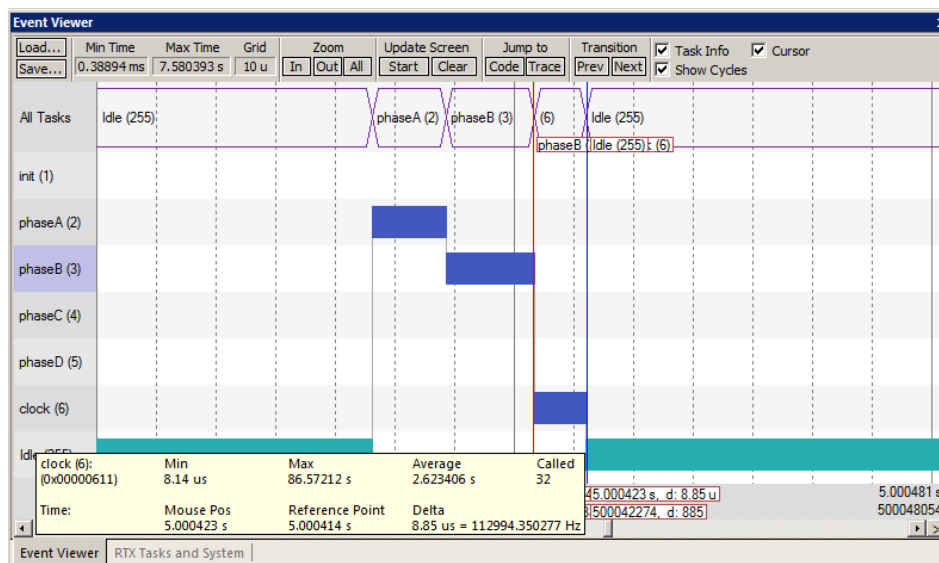
1. Stop the program  and exit Debug mode. 
- 3) Click on the Target Options icon  or ALT-F7.
- 4) Click on the Debug tab.
- 5) Click Settings beside the USE: ULINK Cortex Debugger box. Confirm SWJ and SW are selected.
- 6) Click on the Trace tab to open this window: 
- 7) Set Core Clock to 100 MHz. (or 166 on EmCraft).
You might have to try various clock settings if you are unsure of the value to use.

TIP: The Core Clock: value is set in the STAPL file. This value has to be set correctly in order for SWV to work. ULINK pro detects this value automatically. It uses Core Clock: value you enter for some timing values.



- 8) Select Trace Enable.
- 9) Unselect Periodic, EXCTRC and on Data R/W Sample. ITM bit 31 must be enabled as shown here:
- 10) Click on OK twice to return to the main screen.
- 11) The Serial Wire Viewer is now configured.
- 12) Click on File/Save All to save these settings.
- 13) Enter Debug mode  and click on RUN .
- 14) The Event Viewer should now work. Adjust the range by clicking on All and IN or Out.
- 15) If it isn't running: change the Core Clock: in the Trace Configuration window setting to 50, 100 or 166 MHz.
- 16) Stop the data collection by selecting Stop under Update Screen. The Blinky program keeps running.
- 17) Click on one of the tasks to create a red line as shown below. Click on In under Zoom to expand on this point.
- 18) Select the three boxes on the right of this window. Hover your mouse over the window and see the data provided:
- 19) In this case, the clock task ran for 8.85 μ s. The value you get will depend on the CPU speed set by the stp file.
- 20) Stop the program. Close both RTX awareness windows and leave debug mode.

TIP: It is possible to configure RTX (in the file RTX_Config_CM.c and in your sources) to match your timing needs. The Event Viewer will help you understand how the tasks are switched to ensure they are doing what you intend. As you have seen, the Event Viewer is very easy to enable and use.

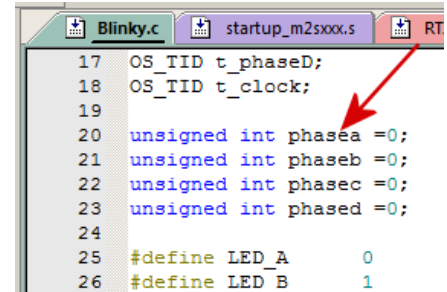


4) Logic Analyzer Window: View variables real-time in a graphical format:

µVision has a graphical Logic Analyzer window. Up to four variables can be displayed in real-time using the Serial Wire Viewer in the SmartFusion2. RTX_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.





1. Close the RTX Viewer windows. Stop the program and exit debug mode.
2. Add 4 global variables **unsigned int phasea** through **unsigned int phased** to Blinky.c as shown here:
3. Add 2 lines to each of the four tasks Task 1 through Task 4 in Blinky.c as shown below: **phasea=1;** and **phasea=0;** the first two lines are shown added at lines 051 and 054 (just after LED_On and LED_Off function calls). For each of the four tasks, add the corresponding variable assignment statements phasea, phaseb, phasec and phased.
4. We do this because in this simple program there are not enough suitable global or static variables to connect to the Logic Analyzer.

TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static or global. To see peripheral registers merely write to them and enter them into the Logic Analyzer.





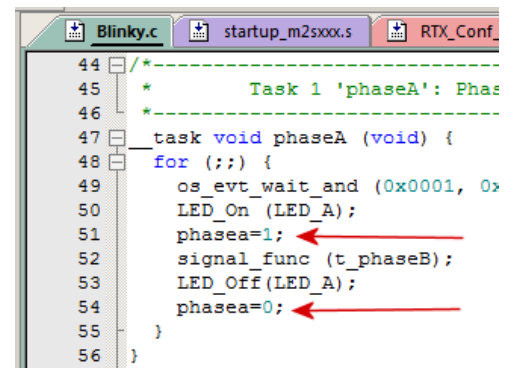
```

17 OS_TID t_phaseD;
18 OS_TID t_clock;
19
20 unsigned int phasea =0;
21 unsigned int phaseb =0;
22 unsigned int phasec =0;
23 unsigned int phased =0;
24
25 #define LED_A      0
26 #define LED_B      1
  
```

5. Rebuild the project.  Program the Flash .
6. Enter debug mode . Click RUN. .

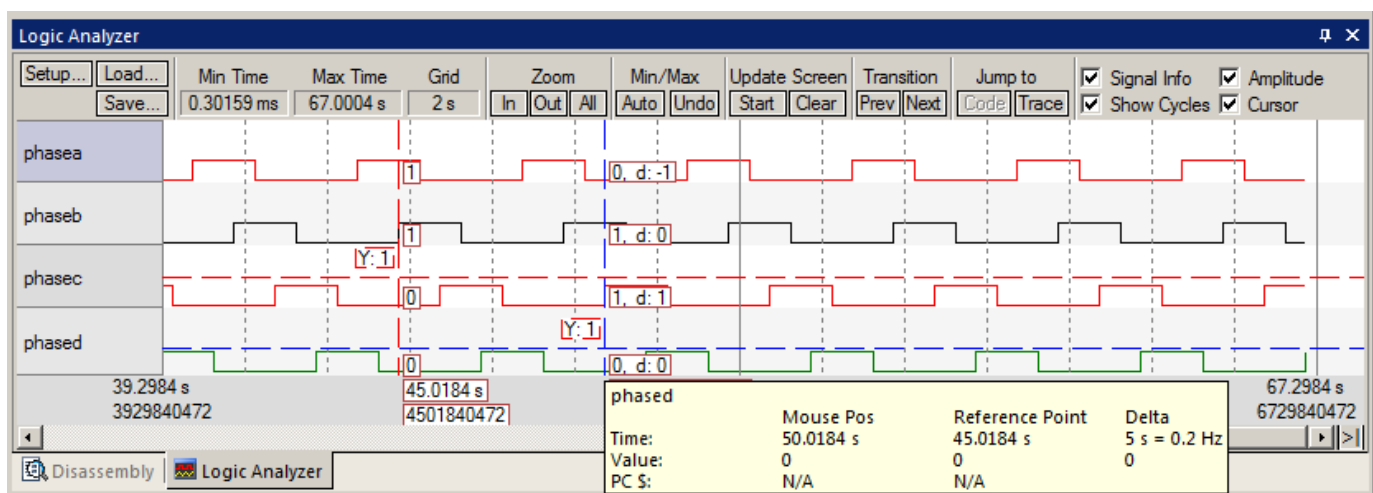
Enter the Variables into the Logic Analyzer:

7. Right-click on **phasea** and select Add 'phasea to ... and select Logic Analyzer. This will open the LA window if necessary.
8. Repeat for **phaseb**, **phasec** and **phased**. These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.
9. Click on the Setup icon and click on each of the four variables in turn and set Max. in the Display Range: to 0x3.
10. Click on Close to go back to the LA window.
11. Add each variable in turn to Watch 1. Note the Watch 1 starts to display these four variables and update them.
12. Using the OUT and In buttons set the range to 1 second or so. You will see the following waveforms appear:
13. Click Stop under Update Screen. Click to mark a place as shown below. Select Signal Info and Cursor. Hover over one of the waveforms and get timing and other information as shown in the inserted box labeled phased:
14. Stop the program  and leave Debug mode. .



```

44 /*
45 *      Task 1 'phaseA': Phase
46 *
47 _task void phaseA (void) {
48     for (;;) {
49         os_evt_wait_and (0x0001, 0);
50         LED_On (LED_A);
51         phasea=1;
52         signal_func (t_phaseB);
53         LED_Off(LED_A);
54         phasea=0;
55     }
56 }
  
```



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

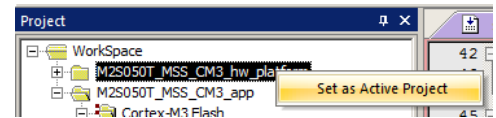
ARM CMSIS-DSP libraries are offered for ARM Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil\ARM\CMSIS. README.txt describes the location of various CMSIS components. See www.arm.com/cmsis and forums.arm.com for more information. CMSIS is an acronym for Cortex Microcontroller 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.

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

To obtain this example file, go to www.keil.com/appnotes/docs/apnt_238.asp Extract \DSP to ... \Microsemi\SF2-DEV-KIT\

1. Open the multi-project file sine: C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\DSP\DSP_Demo.uvmpw
2. In the Project window, right click on M2S050T_MSS_CM3_hw_platform and click Set as Active Project: This project will take on a black highlight background as shown here:



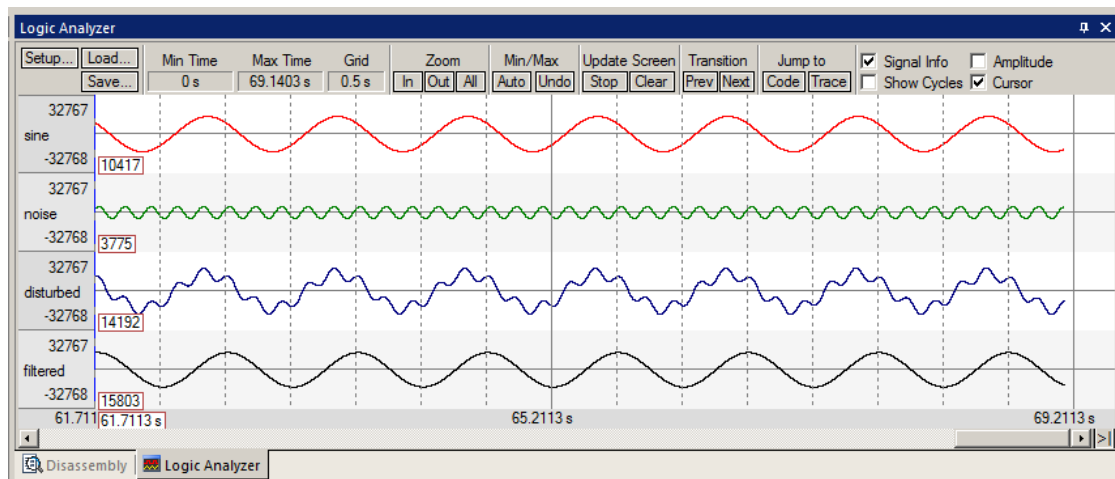
3. Compile the source files by clicking on the Rebuild icon. Ignore the one warning. It is inconsequential.
4. In the Project window, right click on DSP and click Set as Active Project. It will take on a black background.
5. Compile the source files by clicking on the Rebuild icon.
6. Program the SmartFusion2 eNVM flash by clicking on the Load icon. Progress is indicated in bottom left corner.
7. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode notice appears.

TIP: The default Core Clock: is 100 MHz for the DEV KIT. For the EmCraft board, change this to 166 MHz. See page 18.

8. Click on the RUN icon. Open the Logic Analyzer window.
9. 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 shows no waveform, disable the ITM Stimulus Port 31 in the Trace Config window. The SWO pin is probably overloaded.

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



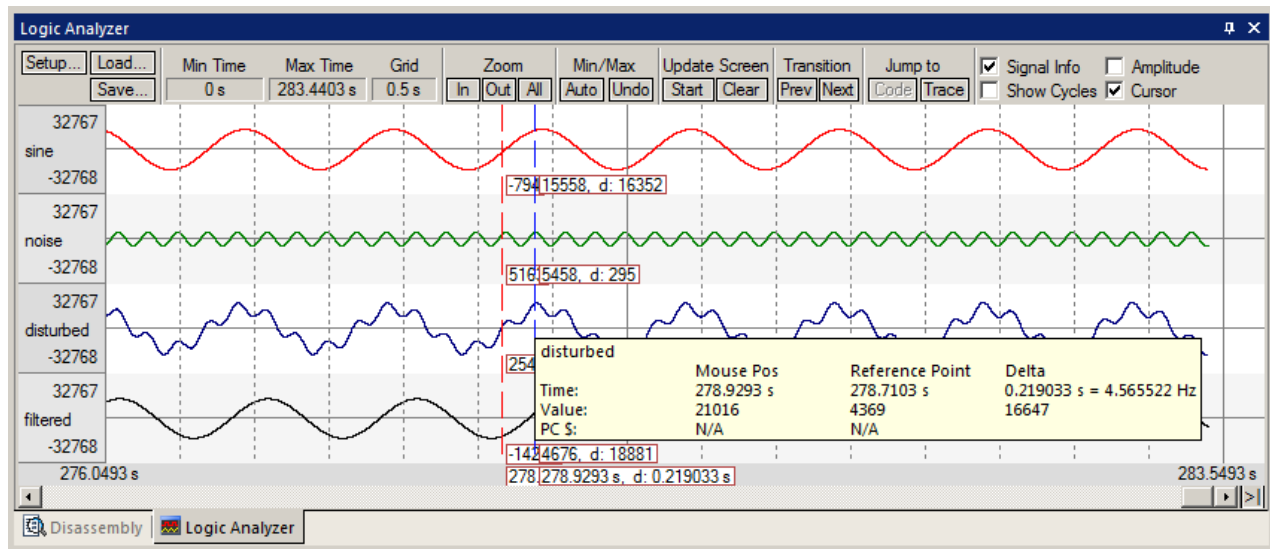
11. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below: They are pre-configured in this project.
12. Open the Trace Records window and the Data Writes to the four variables are displayed.
13. Leave the program running.
14. Close the Trace Records window.

TIP: The ULINK_{pro} trace display is different and the program must be stopped to update it.

Watch 1		
Name	Value	Type
..... sine	0xCF0E	short
..... noise	0x0800	short
..... disturbed	0xD336	short
..... filtered	0xC34F	short
< Enter expression >		

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:



2) 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.

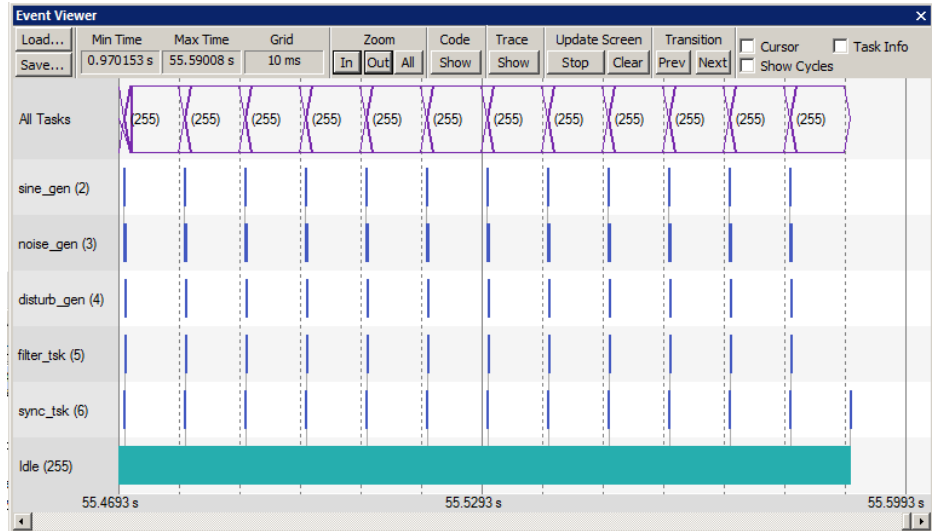
TIP: Recall this window uses the CoreSight DAP read and write technology to update this window. Serial Wire Viewer is not used and is not required to be activated for this window to display and be updated.

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

RTX Tasks and System									
Property		Value							
System		Item	Value						
		Timer Number:	0						
		Tick Timer:	10.000 mSec						
		Round Robin Timeout:							
		Stack Size:	200						
		Tasks with User-provided Stack:	0						
		Stack Overflow Check:	Yes						
		Task Usage:	Available: 7, Used: 5						
		User Timers:	Available: 0, Used: 0						
ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load		
255	os_idle_demon	0	Ready				32%		
6	sync_task	1	Wait_DLY	1			32%		
5	filter_task	1	Wait_AND		0x0000	0x0001	32%		
4	disturb_gen	1	Wait_AND		0x0000	0x0001	32%		
3	noise_gen	1	Running		0x0000	0x0001	0%		
2	sine_gen	1	Wait_AND		0x0000	0x0001	32%		

3) RTX Event Viewer:

1. Stop the program. 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. If you like – you can leave the LA loaded with the four variables to see what the Event Viewer will look like. Later, delete them to see the effect, if any, on the Event Viewer.
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. The window here opens up:
9. Note there is no Task 1 listed. Task 1 is main_tsk and is found in DirtyFilter.c near line 169. It runs some RTX initialization code at the beginning and then deletes itself with os_tsk_delete_self(); found near line 188.



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 Logic Analyzer to free up some SWO or Trace Port bandwidth. It depends on how much data is sent to the ports.

ULINKpro is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows.

ULINKpro uses the faster Manchester format than the slower UART mode that ST-Link, ULINK2 and J-Link uses.

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

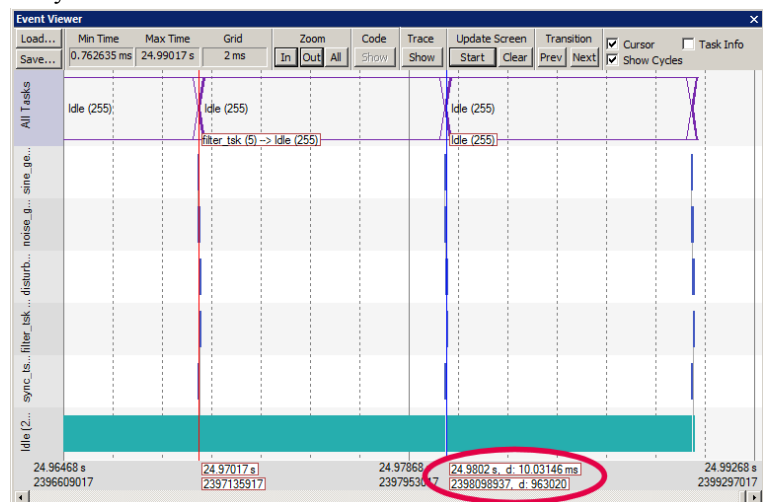
10. Note on the Y axis each of the 5 running tasks plus the idle daemon. Each bar is an active task and shows you what task is running, when and for how long.
11. Click Stop in the Update Screen box.
12. Click on Zoom In so three or four tasks are displayed.
13. Select Cursor. Position the cursor over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

The 10 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: The EmCraft board will show ~ 6 ms because it is running at 166 MHz versus 100 in the program. Change Timer Clock Value from 100 to 166 MHz in RTX_Conf_CM.c will fix this.

TIP: ITM Port 31 enables 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.

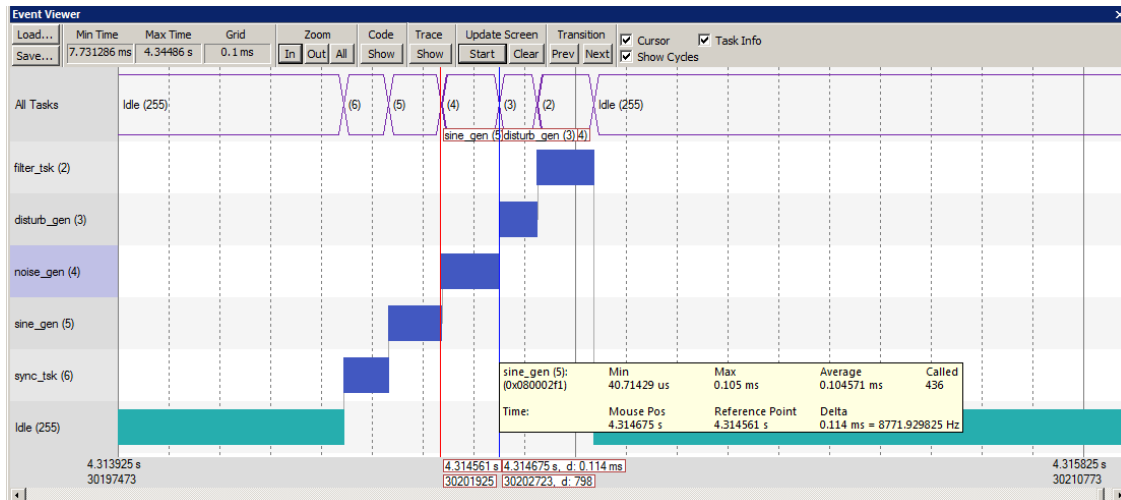


Event Viewer Timing:

1. Click on In under Zoom until one set of tasks is visible as shown below:
2. Enable Task Info (as well as Cursor and Show Cycles from the previous exercise).
3. Note one entire sequence is shown. This screen was taken with a ULINK2 with LA cleared of variables.
4. Click on a task to set the cursor and move it to its end. The time difference is noted. The Task Info box will appear.

TIP: If the Event Viewer does not display correctly, the display of the variables in the Logic Analyzer window might be overloading the SWO pin. In this case, stop the program and delete all LA variables (Kill All) and click on Run.

The Event Viewer can give you a good idea if your RTOS is configured correctly and running in the right sequence.



Changing the SysTick Timer:

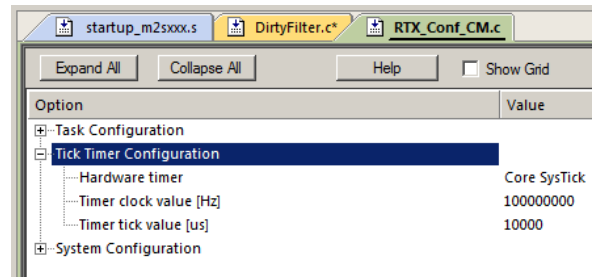
1. Stop the processor and exit debug mode.
2. Open the file RTX_Conf_CM.c from the Project window. You can also select File/Open in C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\DSP.
1. Select the Configuration Wizard tab at the bottom of the window. This scripting language is shown below in the Text Editor as comments starting such as a `</h>` or `<i>`. This scripting language is shown below in the Text Editor as comments starting such as a `</h>` or `<i>`. 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 10,000 μ s or 10 ms.
5. Change this value to 20,000.

TIP: The 100,000,000 is the CPU speed and is correct for the SF2-DEV-KIT. 166 MHz would be correct for the EmCraft board.

6. Rebuild the source files and program the Flash.
7. Enter debug mode and click on RUN .
8. When you check the timing of the tasks in the Event Viewer window as you did on the previous page, they will now be spaced at 20 msec.

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 10 μ s 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.

9. Set the SysTick timer back to 10,000. You will need to recompile the source files and reprogram the Flash.
10. Stop the processor and exit Debug mode to precompile the source files.





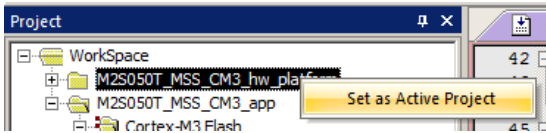


1) ETM Trace Examples: Using the Keil Simulator:

A ULINK_{pro} is required for ETM operation with a board. The Keil Simulator works but does not support any SF2 peripherals. ETM provides serious debugging power as shown on the next few pages. It is worth the modest added cost.


The example project files for Blinky_ETM are provided here: www.keil.com/appnotes/docs/apnt_238.asp

Extract these files into C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\ This will create a sub-directory \Blinky_ETM.

ETM With Keil Simulator:

1. Start μ Vision by clicking on its desktop icon. 
2. Select Project/Open Project: C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\Blinky_ETM\Blinky_ETM.uvmpw (do not select Blinky_ETM.uvproj).
3. You do not need any hardware connected to use the Keil Simulator.
4. Select Simulator in the Target Options box as shown here: 
5. In the Project window, right click on M2S050T_MSS_CM3_hw_platform and click Set as Active Project: This project will take on a black highlight background as shown here: This project was created by Libero. 
6. Compile the source files by clicking on the Rebuild icon. 
7. In the Project window, right click on Blinky and click Set as Active Project. It will take on a black background.
8. Compile the source files by clicking on the Rebuild icon. 
9. Do not use the Load icon – this is not used with the Simulator or when running in RAM.

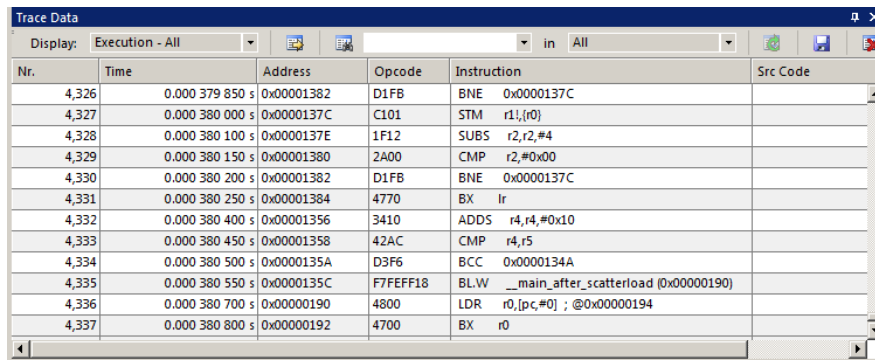
10. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.

11. Click on the small arrow beside the Trace Windows icon and select these two entries: 

12. Optionally exit and re-enter Debug mode. This starts your project from the beginning.


13. The Trace Data window will be filled with frames as shown below:

14. Examine the Instruction Trace window as shown below: This is a complete record of all the instructions executed since RESET. μ Vision halted the program at the start of main() because Run to Main is selected in Target Options.



Nr.	Time	Address	Opcode	Instruction	Src Code
4,326	0.000 379 850 s	0x00001382	D1FB	BNE 0x0000137C	
4,327	0.000 380 000 s	0x0000137C	C101	STM r1,[r0]	
4,328	0.000 380 100 s	0x0000137E	1F12	SUBS r2,r2,#4	
4,329	0.000 380 150 s	0x00001380	2A00	CMP r2,#0x00	
4,330	0.000 380 200 s	0x00001382	D1FB	BNE 0x0000137C	
4,331	0.000 380 250 s	0x00001384	4770	BX lr	
4,332	0.000 380 400 s	0x00001356	3410	ADDS r4,r4,#0x10	
4,333	0.000 380 450 s	0x00001358	42AC	CMP r4,r5	
4,334	0.000 380 500 s	0x0000135A	D3F6	BCC 0x0000134A	
4,335	0.000 380 550 s	0x0000135C	F7FEFF18	BLW __main_after_scatterload (0x00000190)	
4,336	0.000 380 700 s	0x00000190	4800	LDR r0,[pc,#0] ; @0x00000194	
4,337	0.000 380 800 s	0x00000192	4700	BX r0	

15. In this case, Nr. 4,337 shows the last instruction to be executed. (BX r0). In the Register window the PC will display the value of the next instruction to be executed (0x0000 040A in my case) which is MOV R4, #0xFFFFFFFF.

16. Click on the Disassembly window to bring it into focus. Click on Single Step once.  The MOV will now execute.



17. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.








18. This is 0x0000 0019C which is a LDR instruction.

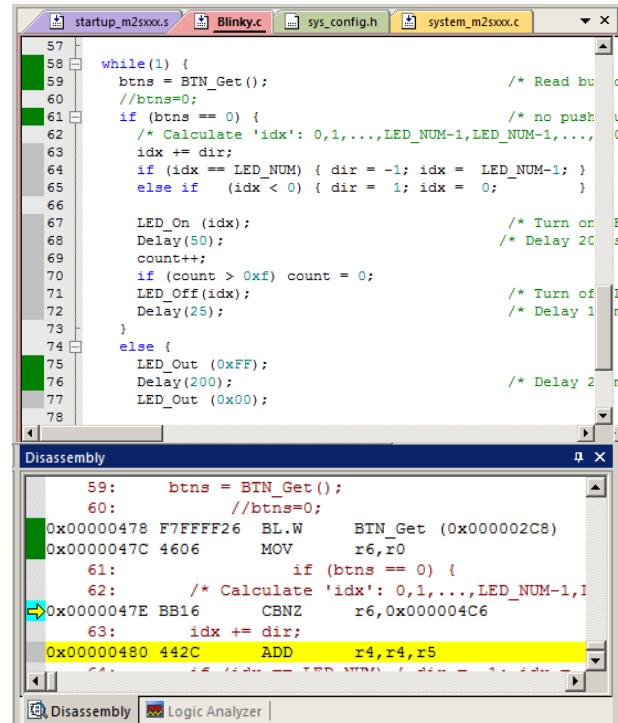
19. In the Memory 1 window, enter 0. Right click and select Unsigned, Long. Memory location 0 contains the initial stack pointer and location 4 contains the initial PC+1 which is 0x019D. (0x019D – 1 = 0x019C)

20. The Disassembly and C source windows display code as it is written. The ETM trace window displays *as it happened*.

2) Code Coverage: (works with the Simulator and ULINKpro)

1. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
2. Examine Blinky.c window near line as shown here: The green blocks indicate code that was executed. Note the section of gray at lines 63 to 72 that was never executed.
3. This is never executed because BTN_Get() never returns 0 as the simulator is not configured with GPIO input signals..
4. Click on the line `if (btns == 0)` and the instruction near (or at) 0x47E is highlighted. Its block is cyan indicating this branch was always taken bypassing lines 63 to 72.
5. This is Code Coverage provided by ETM trace.






- | | |
|---|--|
|  | 1. Green: this assembly instruction was executed. |
|  | 2. Gray: this assembly instruction was not executed. |
|  | 3. Orange: a Branch is always not taken. |
|  | 4. Cyan: a Branch is always taken. |
|  | 5. Light Gray: there is no instruction at this point. |
|  | 6. RED: A Breakpoint is set here (is really a circle). |
|  | 7. Indicates the next instruction to be executed. |



If you scroll in the Disassembly window, 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: It is possible that a C/C++ source line shows green while it contains a branch not fully exercised. You must check all windows.

Why was the branch CBNZ always taken resulting in 0x0480 never being executed? There are other such examples that you can easily display. You should devise tests to execute these instructions so you can test them.

1. Remove the comment on `btns=0;` near line 60. This will override the issue of returning 0xFF from the GPIO port.
2. Exit Debug mode  Rebuild the files  Enter Debug mode  Click on RUN  and then STOP .
3. Scroll down in Blinky.c and you will see the lines 63 through 72 are now executed and 75 through 77 are not.

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 any unexecuted instructions have not been tested. Some agencies such as the US FDA require Code Coverage for certification.

A Code Coverage window is available as shown here:

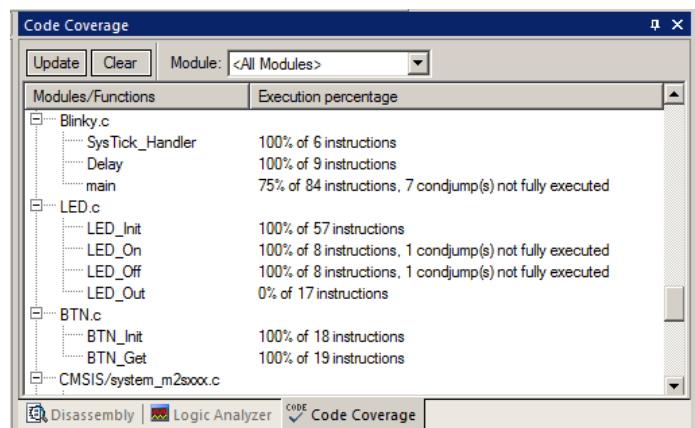
This window is available in View/Analysis/Code Coverage. Note your display may look different due to different compiler options.

You can display either Modules or Functions.

This window is updated while the program is running.

TIP: You may save and restore coverage statistics for multiple debug sessions using a command line in the Output Window. For more details see:

www.keil.com/uvision/db_and_codecoverage.asp or
www.keil.com/support/man/docs/uv4/uv4_cm_coverage.htm

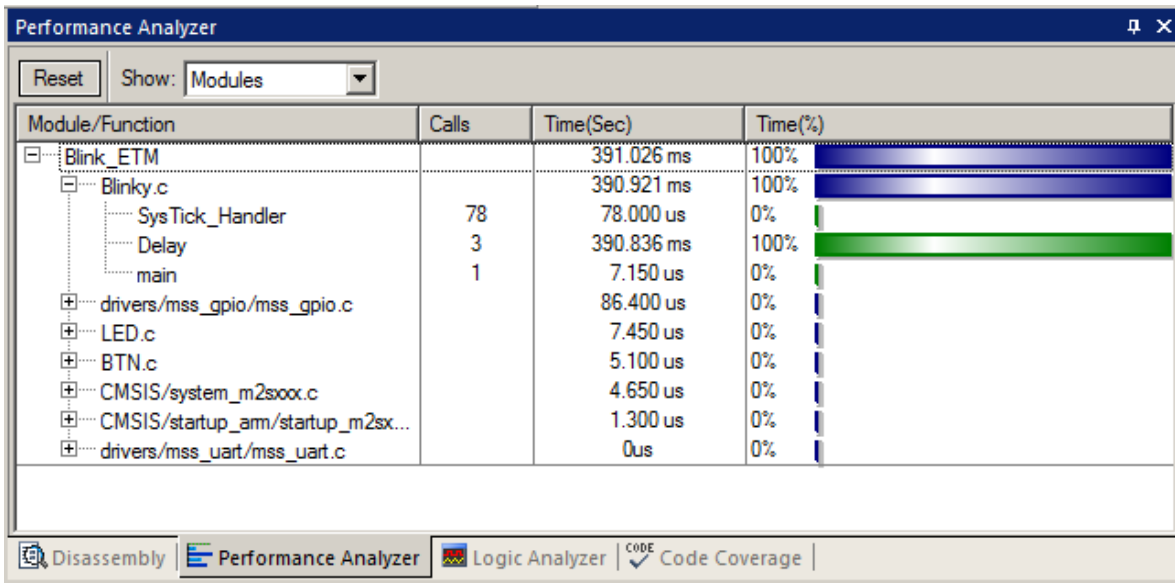


3) Performance Analysis (PA): (works with the Simulator and ULINKpro)

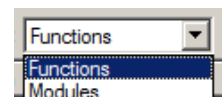
Performance Analysis tells you how much time was spent in each function. The data is provided by ETM trace. ETM provides complete Performance Analysis. Keil provides only ETM PA.

Keil provides Performance Analysis with the μ Vision Simulator or with ETM and the ULINKpro. 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 setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Expand some of the module names as shown below.
4. Note the execution information that has been collected in this initial short run. Both times and number of calls is displayed.
5. We can tell that most of the time at this point in the program has been spent in the Delay routine.



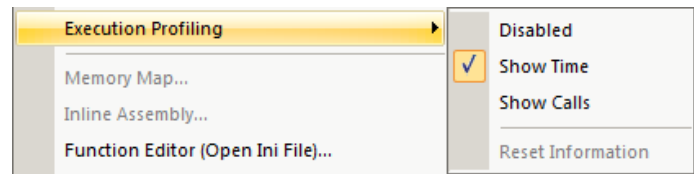
6. Click on the RUN icon.
7. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the information. No code stubs are needed in your source files.
8. Select Functions from the pull down box as shown here and notice the difference in the PA:
9. Click on Reset in the PA window and note the effect and that this window is updated while the program is running.
10. When you are done, Stop the program and exit Debug mode.



4) Execution Profiling: (works with the Simulator and ULINKpro)

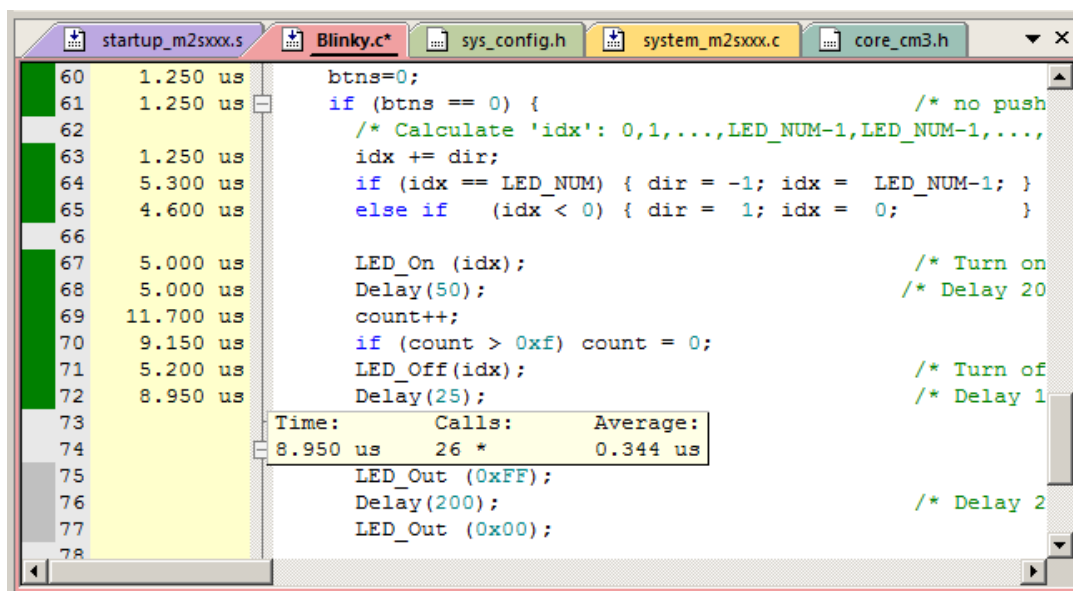
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace and in the Keil Simulator. It is possible to group source lines (called collapse) to get combined times and number of calls. This is called Outlining.

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



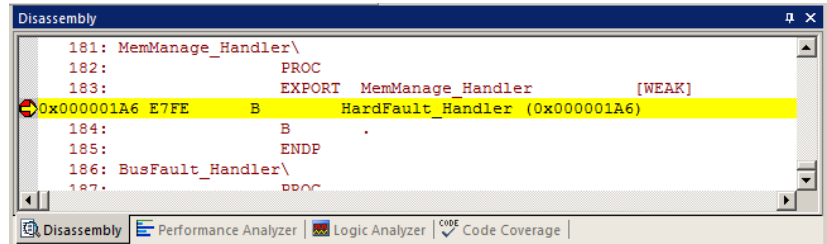
5) In-the-Weeds Example: (works with Simulator and ULINKpro)


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 task 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 is not hard to use.

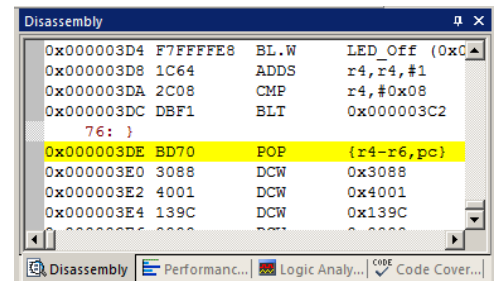
If a Hard Fault occurs, the CPU will end up at the address specified in the Hard Fault vector located at 0x00 01A6. This address points to the Hard Fault handler. This is usually a branch to itself and this Branch instruction will run forever. The trace buffer will save millions of this same branch instruction. This is not useful. We need to stop the CPU at this point.

This 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 Hard Fault handler (in this case located at 0x0000 01A6 as shown to the right) and stop.

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



1. Use the same Blinky setup from the previous page.
2. Locate the Hard fault vector near line 183 in the disassembly window as shown above or in startup_m2sxxx.s.
3. Set a breakpoint at this point. A red circle will appear as shown above.
4. Run the Blinky example for a few seconds and click on STOP. It will probably stop in the Delay function.
5. In the Disassembly window, scroll up until you find a POP instruction. I found one at 0x0000 03DE as shown below:
6. Right click on the POP instruction (or on the ADDS at 0x0000 03D8 as shown below for a more interesting display) and select Set Program Counter. This will now be the next instruction executed.
7. Optionally, clear the Data Trace window to make things more clear. 
8. Click on RUN and immediately the program will stop on the Hard Fault exception branch instruction.
9. Examine the Data Trace window and you find this POP plus everything else that was previously executed. In the bottom screen are the ADDS, CMP, BLT (branch not taken), and POP instructions and one DCW word.
10. Note the Branch at the Hard Fault does not show in the trace window because a hardware breakpoint does execute the instruction it is set to therefore it is not recorded in the trace buffer. Click on Step (F11) and this instruction will be added as it is executed.




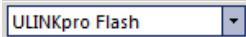
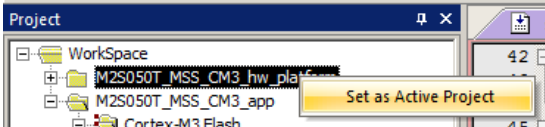





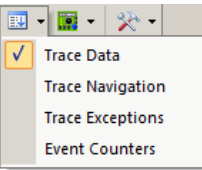
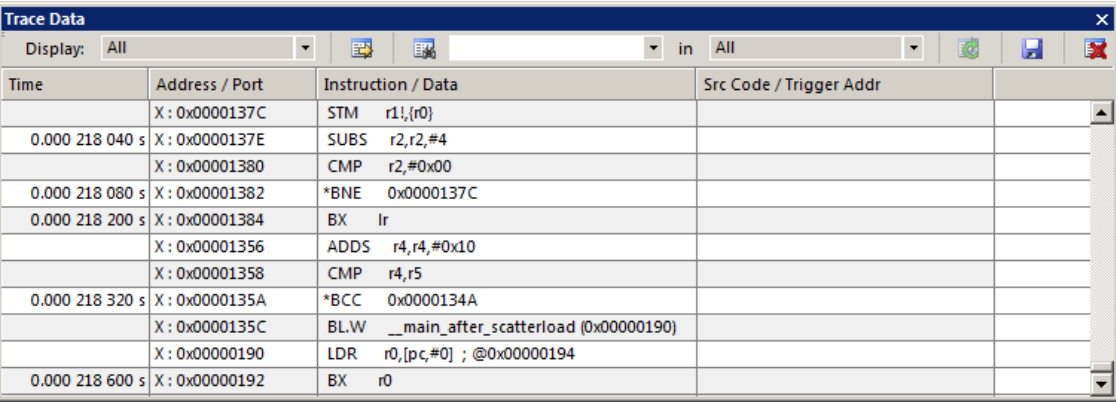
This ends the exercises using the Keil Simulator and trace. Next are some ETM exercises using a Keil ULINKpro and the Microsemi SF2-DEV-KIT.


Trace Data					
Display: Execution - All					
Nr.	Time	Address	Opcode	Instruction	Src Code
178	0.503 007 450 s	0x00000528	EA210104	BIC r1,r1,r4	
179	0.503 007 500 s	0x0000052C	F0020401	AND r4,r2,#0x01	gpio_setting = (value & 0x01u) ...
180	0.503 007 550 s	0x00000530	FA04F403	LSL r4,r4,r3	
181	0.503 007 600 s	0x00000534	EA440101	ORR r1,r4,r1	
182	0.503 007 650 s	0x00000538	4C4A	LDR r4,[pc,#296] ; @0x00000664	GPIO->GPIO_OUT = gpio_setting;
183	0.503 007 750 s	0x0000053A	F8C41088	STR r1,[r4,#0x88]	
184	0.503 007 850 s	0x0000053E	BD10	POP {r4,p,c}	}
185	0.503 008 100 s	0x000005B8	BD10	POP {r4,p,c}	}
186	0.503 008 350 s	0x000003D8	1C64	ADDS r4,r4,#1	
187	0.503 008 400 s	0x000003DA	2C08	CMP r4,#0x08	
188	0.503 008 450 s	0x000003DC	DBF1	BLT 0x000003C2	
189	0.503 008 500 s	0x000003DE	BD70	POP {r4-r6,p,c}	}
190	0.503 008 850 s	0x00000000	10A8	DCW 0x10A8	

6) ETM Trace Examples: Using the SF2-DEV-KIT and ULINKpro:

A ULINKpro is required for ETM operation. The Keil Simulator also works but does not support any SF2 peripherals. ETM provides serious debugging power as shown on the next few pages. It is certainly worth the modest added cost.

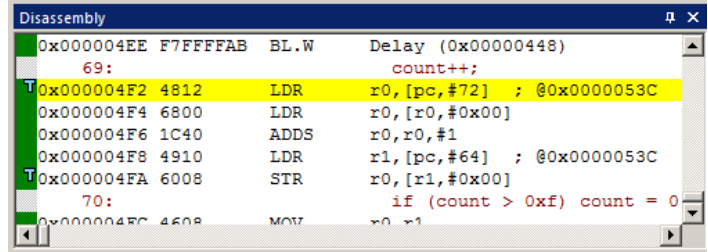
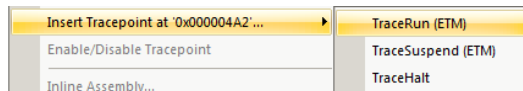
ETM With Keil ULINKpro:

1. Start μ Vision by clicking on its desktop icon. 
2. Select Project/Open Project: C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\Blinky_ETM\Blinky_ETM.uvmpw (do not select Blinky_ETM.uvproj). This is the same one used for the Simulator ETM demonstration.
3. Connect the ULINKpro to the DEV Kit as shown on page 1. Set J93 to position 2-3.
4. Select ULINKpro in the Target Options box as shown here: 
5. In the Project window, right click on M2S050T_MSS_CM3_hw_platform and click Set as Active Project: This project will take on a black highlight background as shown here: This project was created by Libero. 
6. Compile the source files in this project by clicking on the Rebuild icon. 
7. In the Project window, right click on Blinky and click Set as Active Project. It will take on a black background.
8. Comment out line 60 `btms=0;` in Blinky.c.
9. Compile these source files in the Blinky project by clicking on the Rebuild icon. 
10. Program the SmartFusion2 eNVM flash by clicking on the Load icon:  Progress is indicated in bottom left corner.
11. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.
12. Click on the small arrow beside the Trace Windows icon and select Trace Data: 
13. The Trace Data window will be filled with frames as shown below:
14. Examine the Instruction Trace window as shown below: This is a complete record of all the program flow since RESET and until μ Vision halted the program at the start of main(). This is because Run To Main is selected in Target Options. 
15. In this case, 218 600 s shows the last instruction to be executed. (BX r0). **Note:** your time might be different. In the Register window the PC will display the value of the next instruction to be executed (0x0000 040A in my case) which is MOV R4, #0x...FFFF. Timestamps for all instructions are not provided by ETM and is normal.
16. 

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x0000137C	STM r1!,{r0}	
0.000 218 040 s	X: 0x0000137E	SUBS r2,r2,#4	
	X: 0x00001380	CMP r2,#0x00	
0.000 218 080 s	X: 0x00001382	*BNE 0x0000137C	
0.000 218 200 s	X: 0x00001384	BX lr	
	X: 0x00001356	ADDS r4,r4,#0x10	
	X: 0x00001358	CMP r4,r5	
0.000 218 320 s	X: 0x0000135A	*BCC 0x0000134A	
	X: 0x0000135C	BL.W __main_after_scatterload (0x00000190)	
	X: 0x00000190	LDR r0,[pc,#0] ; @0x00000194	
0.000 218 600 s	X: 0x00000192	BX r0	
17. Click on the Disassembly window to bring it into focus. Click on Single Step once.  The MOV will now execute.
18. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.
19. This is located at memory 0x0019C which is a LDR instruction.
20. In the Memory 1 window, enter 0. Right click and select Unsigned, Long. Memory location 0 contains the initial stack pointer and location 4 contains the initial PC+1 which is 0x019D.
21. The Disassembly and C source windows display code as it was written. ETM trace displays code *as it happened*.

7) Trace Triggering with ULINK^{pro}:

1. CoreSight provides a mechanism to start and stop the trace collection to allow you to focus on frames of interest.
2. In Blinky.c, set a breakpoint on count++; near line 69. Run the program: it will stop on this line.
3. Remove this breakpoint and set it on the Delay(25) near line 70. Click on RUN several times. You can see the Data Trace window filling with many frames. We will focus on the line count++;.
4. In Blinky.c, click once on the count++; line to highlight it. It will get a cyan arrow and in the Disassembly window 0x4F2 will have a yellow highlight as shown here:
5. In the Disassembly window, right click on line 0x4F2 and select Insert Tracepoint .. TraceRun as shown below:
6. A cyan “T” will appear as shown here: Trace collection will start at this instruction.



7. Right-click on 0x4FA and similarly select TraceSuspend. A cyan “T” will be displayed as shown above:
8. The trace has now been configured as such:
 - a. ETM Trace collection will start when the instruction 0x4F2 is executed.
 - b. ETM Trace collection will stop when the instruction at 0x4FA is executed.
 - c. Everything in between will be collected. This includes all function calls, interrupts or other branches.
 - d. This does NOT always mean that only instructions between 0x4F2 and 0x4FA will be collected.
 - e. The exact addresses you obtain can vary due to compiler optimization settings and other variances.
9. Click on RUN. The program will run to the breakpoint and stop.
10. Shown here are two runs. Clearly it can be seen count++; is captured in the trace and everything else (almost) is discarded as shown by the red parenthesis.

TRACE RUN indicators shown divide the various runs making it more clear what is happening.

TIP: If you see numerous SysTick timer frames filling the trace, filter them out by selecting ETM – Code Exec as shown:

Trace Data			
Display: ETM - Code Exec			
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
0.575 386 170 s	X: 0x0000050E	MOVS r0, #0x19	Delay(25); ...
	TRACE RUN		
	X: 0x000004F2	LDR r0, [pc, #72] ; @0x0000053C	count++;
	X: 0x000004F4	LDR r0, [r0, #0x00]	
	X: 0x000004F6	ADDS r0, r0, #1	
	X: 0x000004F8	LDR r1, [pc, #64] ; @0x0000053C	
	X: 0x000004FA	STR r0, [r1, #0x00]	
0.650 385 940 s	X: 0x000004FC	MOV r0, r1	if (count > 0xf) count = 0;
	TRACE RUN		
0.650 385 960 s	X: 0x0000050E	MOVS r0, #0x19	Delay(25); ...
	TRACE RUN		
	X: 0x000004F2	LDR r0, [pc, #72] ; @0x0000053C	count++;
	X: 0x000004F4	LDR r0, [r0, #0x00]	
	X: 0x000004F6	ADDS r0, r0, #1	
	X: 0x000004F8	LDR r1, [pc, #64] ; @0x0000053C	
	X: 0x000004FA	STR r0, [r1, #0x00]	
0.725 386 190 s	X: 0x000004FC	MOV r0, r1	if (count > 0xf) count = 0;

TraceRUN: Starts ETM collection. This is mandatory for the next two commands to be valid.

TraceSuspend: Stops only ETM frames. SWV frames continue.

TraceHalt: Stops collection of both ETM and SWV frames.

8) More ETM Examples:

Now that you have the ETM working with a ULINK_{pro}, you can try the exercises on these preceding pages that were originally used with the Simulator.

Code Coverage: 28

Performance Analysis (PA): 29

Execution Profiling: 30

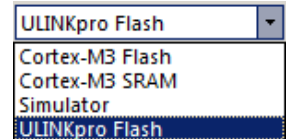
In-the-Weeds Example: 31


9) Configuring Target Options for ULINK_{pro}:

The Multi-Project Blinky_ETM.uvmpw contains the project Blinky.uvproj. This has several target options as shown here:


Each of these entries represents a different Target Options configuration. These were preconfigured for your convenience. It is useful to be able to create these and configure the Debug adapters yourself. These are created and managed in Projects/Manage/Project Blinky Components.

When you select one of these targets, the associated Target Options window will be activated.




It can then be opened in the usual fashion by selecting the Target Options icon  or ALT-F7.

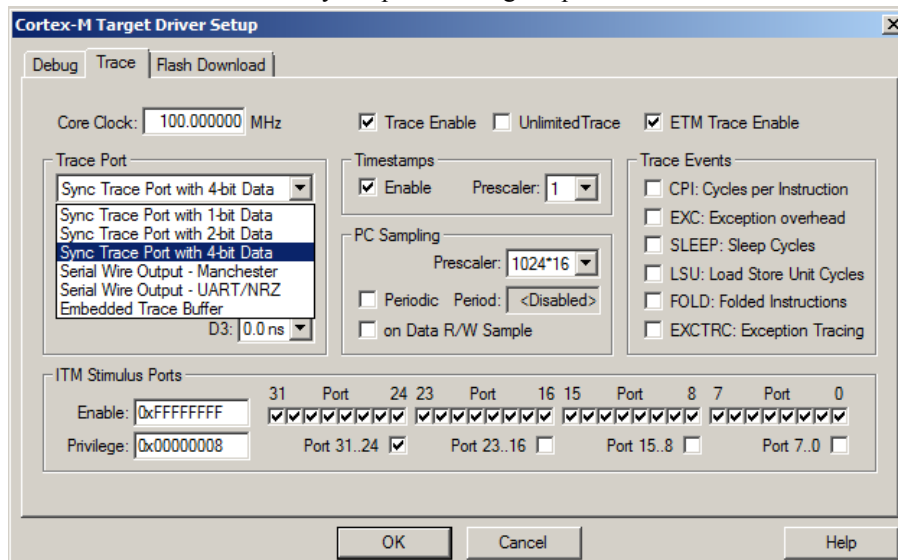
How to Configure a ULINK_{pro}:

1. You do not need to be connected to a target or a ULINK to set the Trace Configurations.
2. Open the Target Options window by clicking on its icon  or ALT-F7.
3. Select the Debug tab. Select ULINK Pro Cortex Debugger and select Settings:
4. Select the Trace tab and the window below opens:
5. Select Trace Enable and ETM Trace Enable. Enter a valid Core Clock: value for certain timing displays.
6. Select Sync Trace Port with 4-bit Data as shown. ETM and SWV trace is now configured.
7. Click OK twice to return to the main μ Vision menu.

TIP: ULINK_{pro} can use either Serial Wire Output – Manchester or Sync Trace Port with 4-bit Data for SWV. It cannot use the UART/NRZ option. Selecting this will cause an error. For ETM *and* SWV Trace, the 4 bit Trace Port is the only option. ULINK2 or ULINK-ME can use only the Serial Wire Output – UART/NRZ option. Same for the Segger J-Link.

TIP: JTAG or SWD can be used with the Trace Port option. With Serial Wire Output pin option, with either ULINK2/ME or ULINK_{pro}, only SWD is allowed. This is because of the TDIO and SWO conflict. JTAG is not allowed in this case. This is normally an issue of the 20 pin Cortex ETM connector is not available for the ULINK_{pro} to access the Trace Port.

Super TIP: If you change the debug adapter here, you must also change it in the Utilities tab. This selects the eNVM flash programming algorithm. This tab is visible when you open the Target Options window. 



1) Programming the SmartFusion2 FPGA Flash eNVM with a .stp file:

It is necessary to program the FPGA fabric with a STAPL file which has a .stp extension. This stp file also has an option to program the Cortex-M3 eNVM flash with Cortex-M3 executable code. Once you have programmed the STAPL file into the fabric, you can use μ Vision to program your executable Cortex-M3 code multiple times into the eNVM flash without conflict.


Usually a board comes pre-programmed with a STAPL file.

The EmCraft board is pre-programmed. The stp file is available here: www.emcraft.com/jdownloads/som/m2s/m2s-som-2a.stp

The DEV KIT needs to be programmed with a stp file provided by Keil. See www.keil.com/appnotes/docs/apnt_238.asp

Libero creates stp files with a name such as M2S050T_top.stp. You can change this name to keep track of various versions.

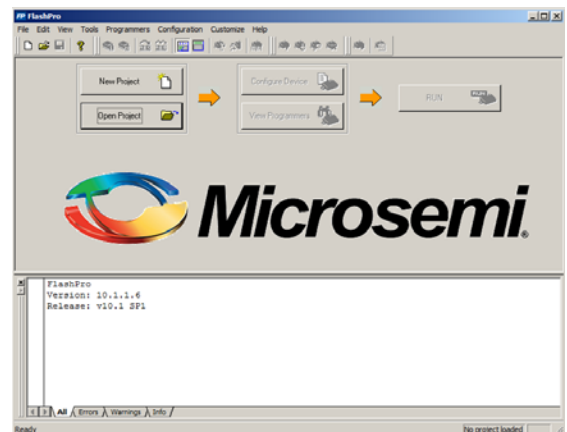
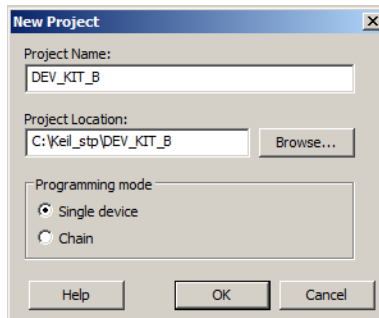
Hardware Connect:

1. Store the .stp file somewhere on your PC. I used C:\Keil_stp.
2. Connect the FlashPro4 to your SF2 board and its USB to your PC.
3. Set the JTAG jumper: DEV KIT J93: 1-2, EmCraft remove JP2
See the images below for details. Power the board.
4. Open the FlashPro4 Programming Software. 

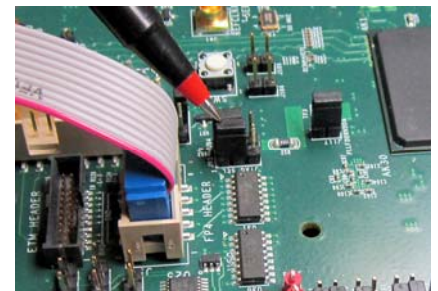
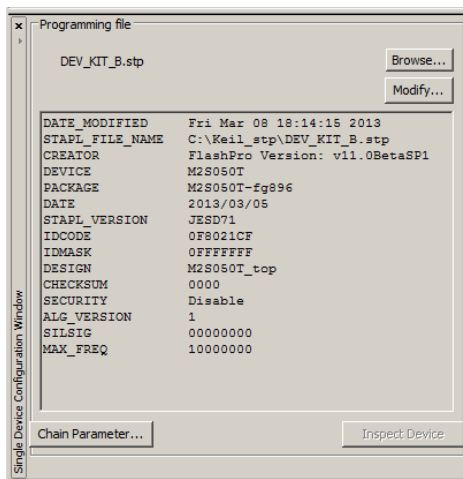
FlashPro4 is available stand-alone or as a component of the Libero IDE

FPGA development software. See www.microsemi.com.

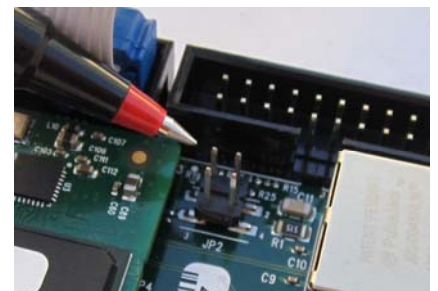
5. Create a new programming project by clicking the New Project:
6. Name your project and pick a directory to save it as shown below: A .pro file will be created. Click on OK.



7. Click on “Configure Device”. This will open the Load Programming File window as shown below:
8. Use the Browse... button to locate your stp file.
9. Highlight it and select Open.
10. Information about your stp file will be displayed as shown below:



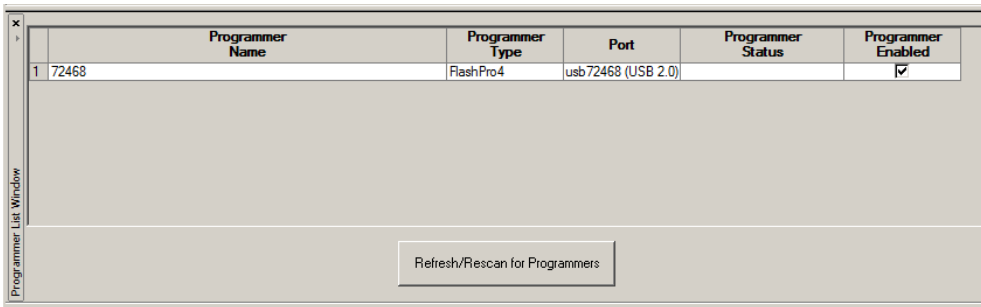
DEV KIT J93 set for FlashPro4:



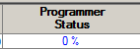
EmCraft JP2 set for FlashPro4:

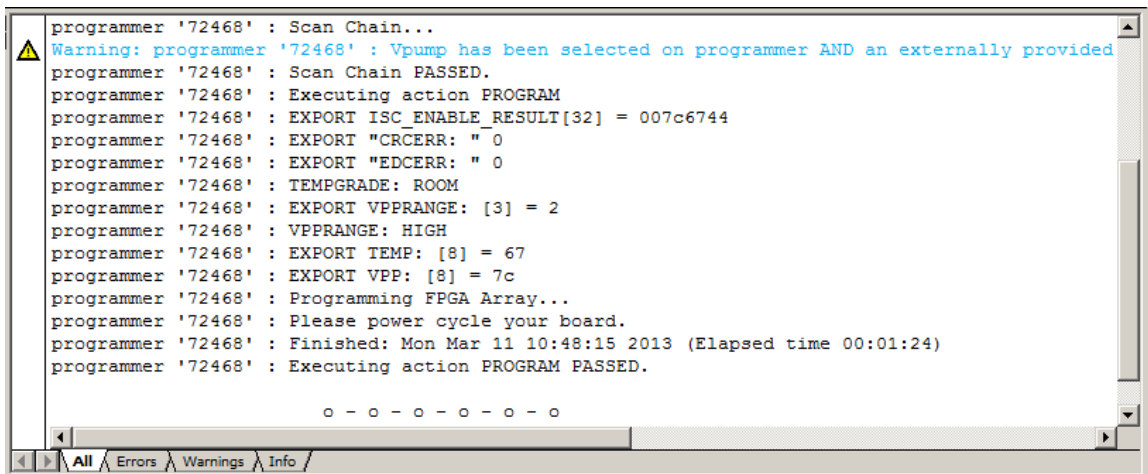
TIP: After you have created a project with FlashPro, you can retrieve it the next time by selecting the Open Project button. Select the *.pro project file. You can select a different STAPL (stp) file by clicking on the Configure Device: button. Then click on Browse: and select the new stp file.

11. At this time, the FlashPro software will connect to the FlashPro4 Device Programmer. As shown here:



Program the eNVM with the STAPL file:

12. Select the option Single Device. In the Action box, select Program.
13. Click the Program button to program the SmartFusion2 device.
14. If there is a warning about Vpump – you can ignore this message.
15. The Programmer Status % does not increment.  The hourglass does rotate.
16. It took FlashPro about 80 seconds to complete programming the STAPL file I was using. If FlashPro stops in a few seconds: there is probably an error. Look in the information window for the cause. You might have to scroll up a bit to see the error. Here is the text of a successful FPGA array programming:



17. If the programming is successful RUN PASSED will display as shown:

Programmer Name	Programmer Type	Port	Programmer Status	Programmer Enabled
72468	FlashPro4	usb72468 (USB 2.0)	RUN PASSED	<input checked="" type="checkbox"/>

18. If the board is not connected or powered properly: A Scan Chain Failed error will result:

Programmer Name	Programmer Type	Port	Programmer Status	Programmer Enabled
72468	FlashPro4	usb72468 (USB 2.0)	SCAN CHAIN FAILED	<input checked="" type="checkbox"/>

19. Replace the jumper as it was before in order to program, run and debug the with a Keil ULINK adapter. See page 4.
20. Remove the FlashPro4 and repower the board. The SF2 has been successfully programmed with the STAPL file.

2) Creating a new project: Using the Blinky source files:

In order for SmartFusion2 to operate, a STAPL (.stp) file must be programmed into it. A .stp file can be created by you using Microsemi Libero IDE or you can use a pre-built one that is provided by a third party. Keil provides such a .stp with this lab. The design flow created with Libero configures the SF2 clocks and peripherals. Various components are connected to the SF2 pins. This includes the ETM Trace Port and GPIO ports that are used in the Keil example files.

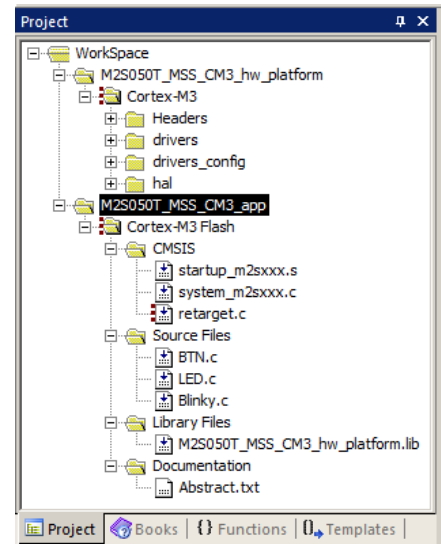
Libero also creates two uVision projects: one is for the hardware (M2S050T_MSS_CM3_hw_platform.uvproj) and the other is for your application: (M2S050T_MSS_CM3_hw_app.uvproj). Both are contained in a uVision Multi-Project (uvmpw). You can see this in the two Keil example projects included with MDK. Each uvproj project is selected as needed by right-clicking on its name and select Set as the Active Project.

1. You first compile the hw_platform project and it creates a library that is used as input by the hw_app project. You do this only once unless you make changes in Libero that would be reflected in the library file.
2. Second, you make the hw_app active and compile it. This is where you develop, compile and debug your own user code for the SF2 MSS.
You can change the names of these projects to your own preferences.

Shown here is the Multi-Project uvmpw file from the Keil Blinky example:

Two methods you can use to create your own custom project:

- a. Copy the Libero created projects into uVision. The *hw_app project contains an empty main.c. You must add startup files and other code to create a real project. The Main() function is a simple while() loop that you can add your own code to.
- b. Use one of the two Keil examples as a template for your own project. This is the simplest method. A disadvantage is the STAPL file might not be appropriate for your project. The DSP example was created this way. The DSP example files were moved from a generic Cortex-M3 processor into the SF2 RTX_Blinky example with minimal work.




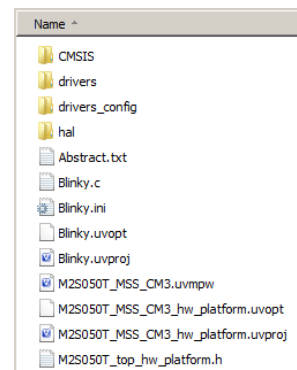
Use the Blinky example for a general bare metal template and if you prefer to use RTX, choose RTX_Blinky. In this way, most of the work is already done for you.

All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects.

However, we will start this example project from the beginning to illustrate how easy this process is. The second method above will be used. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run the Blinky example as usual. You can use this process to create any new project from your own source files created with uVision's editor or any other editor.

A) Copy existing Blinky files into a new directory called FAE:

1. Using Windows Explorer, got to C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT
2. Make a copy of the entire Blinky directory and rename it FAE. You can name it anything.
3. If you want to rename your project: now is a good time to do it. Rename everything from M2S050T_MSS_CM3_app.* to your choice. I chose Blinky as in Blinky.uvproj and so on. See below for examples:
4. Do not change the hw_platform filenames. You could, but let us keep things simple for now.
5. Delete all .bak, .dep, BTN.*, LED.* files and the lst and obj directories if they exist. This is optional, but cleans up excess files. The lst and obj directories will be re-created later.
6. You will have a directory that looks similar to this: 
7. This will be located the directory: C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\FAE\
8. If you have any extra files you put there such as sim.ini, you can leave them there.
9. You can modify the abstract file for your own needs. It is for information only.

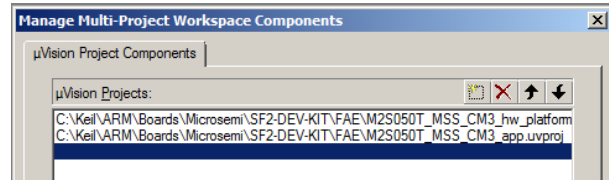


TIP: You don't need to delete all the files if you want to learn how to use them. It is interesting to implement the buttons or the leds. They are good examples of input and output routines.

Next, we will create a new user project in uVision...

B) Create a new project called Blinky from Scratch: *continued...*

1. Start μ Vision. Select Project/Open Project and in the directory C:\Keil\ARM\Boards\Microsemi\SF2-DEV-KIT\FAE and select M2S050T_MSS_CM3.uvmpw. (don't select a uvproj by mistake – you want the Multi-Project uvmpw file)
2. μ Vision will report it can't find the old project (assuming you renamed it): Click OK. We will fix this soon.
3. Select Project/Manage and select Multi-Project Workspace and this window opens up: (partially shown):
4. Double-click on the 2nd line: M2S050T_MSS_CM3_app.uvproj.
5. Change M2S050T_MSS_CM3_app.uvproj to Blinky.uvproj. Use the browse icon if you prefer.
6. Click OK to accept your change and close this window..
7. In the Project Workspace in the upper left hand of μ Vision, Blinky will now appear. Expand all the folders by clicking on the "+" beside each folder.



Compile the hw hardware files:

1. Right-click on M2S050T_MSS_CM3_hw_platform in the project window and set it as the active project.
2. Click on Rebuild . Ignore the one warning. It is inconsequential and will not affect your program.
3. This creates the library M2S050T_MSS_CM3_hw_platform.lib. It is now in the \obj directory that you deleted before.

Clean up the Blinky.uvproj files:

4. Right-click on Blinky in the project window and set it as the active project. It will take a black highlight.
5. Right-click on BTN.c and select Remove BTN.c. Click on OK when asked if you are sure.
6. Right-click on LED.c and select Remove LED.c. Click on OK when asked if you are sure. You do not need these.
7. Double-click on Blinky.c to open it in the Source windows. Now we will clean this file up.
8. Near line 19 remove the two lines: `#include "LED.h"` and `#include "BTN.h"`.
9. Remove (or comment out) all lines not needed. See the listing on the next page. Note we left the SysTick timer in. You can remove it if you do not want it. You will probably want to slow the program down with a delay loop.
10. Just before the printf statement add: `Delay(500);`
11. Select File/Save All.

Compile the source files, program Flash, RUN:

1. Click on Rebuild . It should compile with no errors or warnings. If you get some, carefully check your program.

TIP: This example will be configured to work with ULINK2/ME. If you have a ULINK_{pro} or J-link, configure it now.

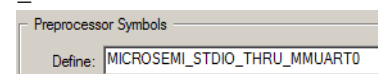
2. Program the SmartFusion2 eNVM flash by clicking on the Load icon: Progress is indicated in bottom left corner.
3. Enter Debug mode by clicking on the Debug icon: Select OK if the Evaluation Mode notice appears.
4. Click on RUN. Your program will now be running.
5. Set a breakpoint on the printf statement and the program should stop here. This is a good test it is running.
6. If it does not: click on STOP to see where it is. Check for standard programing and configuration errors.

If you have a SF2-DEV-KIT:

7. J198 DB-9 RS232 connector will be outputting the printf statement "Hello World" at 115,200 baud.
8. You can get the SWV working with a Core Clock: set to 100 MHz.

If you have a EmCraft board:

9. Open Target Options and select the C/C++ tab. Enter MICROSEMI_STUDIO_THRU_MMUART0 in the Define: box.
10. P1, the USB plug will be outputting the printf statement "Hello World".
11. You can get the SWV working with a Core Clock: set to 166 MHz.



You will be able to add more code and files to this basic template very easily.
This completes creating your own project from scratch.

Here is the example Blinky.c used in the preceding page:

```

/*-----
 * Name:    Blinky.c
 * Purpose: LED Flasher
 *-----*/

#include <stdio.h>
#include <M2Sxxx.h>
#include "system_m2sxxx.h"

volatile uint32_t msTicks;          /* counts 1ms timeTicks */

/*-----
   SysTick_Handler
 *-----*/
void SysTick_Handler(void) {
    msTicks++;                      /* increment counter necessary in Delay() */
}

/*-----
   delays number of tick Systicks (happens every 1 ms)
 *-----*/
void Delay (uint32_t dlyTicks) {
    uint32_t curTicks;
    curTicks = msTicks;
    while ((msTicks - curTicks) < dlyTicks);
}

/*-----
   MAIN function
 *-----*/
int main (void) {

    SystemCoreClockUpdate();        /* Get Core Clock Frequency */
    SysTick_Config(SystemCoreClock / 1000); /* SysTick 1 msec interrupts */

    while(1) {

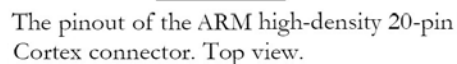
        Delay(500);
        printf ("Hello World\n\r");

    }
}

```

Search www.arm.com for cortex_debug_connectors.pdf for description of the new Cortex debug connectors including ETM.

The 10 pin part number that Keil uses in its boards is Samtec FTSH-105. A 20 pin is available: FTSH-110.



Keil μ Vision and Microsemi SmartFusion2

4) Acronym List

BP	Break Point
DAP	Debug Access Port
eNVM	SmartFusion Flash - embedded nonvolatile memory
ETM	Embedded Trace Macrocell
GPIO	General Purpose Input/Output
ITM	Instrumentation Trace Macrocell
SWD	Serial Wire Debug
SWV	Serial Wire Viewer
SWO	Serial Wire Output
TPIU	Trace Port Interface Unit
WP	Watch Points. They are also called Access Breaks.

Macrocell: A macrocell is a pre-built “black box” of gates used by the chip designer to add a function or peripheral to a processor. Examples are ETM, ITM and memory. Macrocells can also be thought of as “building blocks”. This is a term used by chip designers and not end users.

Hardware Design: This is Microsemi’s terminology for the FPGA program. It is similar to a microcontroller project.

5) 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 μ Vision.
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 μ Vision 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. μ Vision recovers easily and painlessly from data overruns. You are made aware when frames are lost.

6) 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 166 mHz. Recall the ULINKpro calculates Core Clock: automatically. It uses the figure you enter for certain timing displays.
2. 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. μ Vision will complain about this if you try it. SWV needs SWD (or SW) selected if the UART SWO pin is selected. If using a ULINKpro 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.



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 μ Vision.
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 μ Vision via a ULINK.

These are the types of problems that can be found with a quality 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.

8) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite (Evaluation version – 32K code/data limit) \$0
- **NEW !!** MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit) - \$3,200
- MDK-Standard (unlimited compile and debug code and data size) - \$4,895
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) \$9,500

USB-JTAG/SWD adapter (for Flash programming too)

- ULINK2 - \$395 (ULINK2 and ME - SWV only – no ETM)
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINKpro - \$1,250 – Cortex-Mx SWV & ETM trace.
- **For special promotional or quantity pricing and offers, please contact Keil Sales.**



The Keil RTX RTOS is now provided under a Berkeley BSD type license. This makes it free.

All versions, including MDK-Lite, includes Keil RTX RTOS *with source code* !

Keil provides free DSP libraries for the Cortex-M0, Cortex-M3 and Cortex-M4.

Call Keil Sales for details on current pricing, specials and quantity discounts.

Sales will provide advice about the various tools options available to you. They will help you find various labs and appnotes that are useful.

All products are available from stock.

All products include Technical Support for 1 year. This is easily renewed.

Call Keil Sales for special university pricing. Go to www.arm.com and search for university to view various programs and resources.

See the Keil Device Database® on www.keil.com/dd for the complete list of ARM supported devices. This information is also included in MDK in Target Options.

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.

Prices are for reference only and are subject to change without notice.

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



For more information:

Keil Sales: In the USA: sales.us@keil.com or 800-348-8051. Outside the USA: sales.intl@keil.com

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

International Distributors: www.keil.com/distis/ See www.embeddedsoftwarestore.com

For more Microsemi specific information: please visit www.keil.com/microsemi

CMSIS information: www.arm.com/cmsis and forums.arm.com

