

Freescal Kinetis K60N512: Cortex™-M4 Lab

ARM® Keil™ MDK Toolkit *featuring Serial Wire Viewer and ETM Trace*

Winter 2013 Version 3.0

for the K60N512 Board

by Robert Boys, bob.boys@arm.com



Introduction:

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_239.asp

The purpose of this lab is to introduce you to the Freescale Cortex™-M4 processor by using the ARM® Keil™ MDK toolkit featuring µVision® IDE. We will use the Serial Wire Viewer (SWV) and ETM™ trace on the Kinetis processor. For the latest version of this lab or the Freedom boards see www.keil.com/freescale. MDK includes example projects for many other Tower, KwikStik and Freedom boards. This lab can be adapted for use with any of these boards.

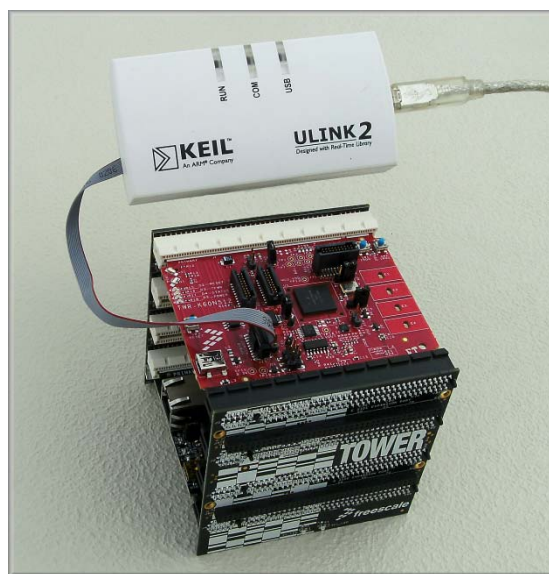
Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. The addition of a license number will turn it into the full, unrestricted version. MDK-Freescale is an economical US \$745 toolkit for Kinetis processors.

Linux and Android: For Linux, Android, bare metal (no OS) and other OS support on Freescale i.MX and Vybrid series processors please see DS-5™ at www.arm.com/ds5/.

Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M users:

1. µVision IDE with Integrated Debugger, Flash programmer, the ARM® Compiler toolkit and example projects.
2. **MDK-Freescale is available for \$745.** See the last page.
3. A full feature Keil RTOS called RTX is included with MDK.
4. The RTX Kernel Awareness windows are updated live.
5. **MQX:** An MQX port for MDK is available including Kernel Awareness windows. See www.keil.com/freescale.
6. **Processor Expert** compatible. For more information see www.keil.com/appnotes/files/apnt_235_pe2uv.pdf.
7. Serial Wire Viewer and ETM trace capability is included.
8. Choice of adapters: ULINK2™, ULINK-ME™, ULINKpro™, Segger J-Link (version 6 or later) and P&E OSJTAG.
9. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.



This document details these features and more:

1. Serial Wire Viewer (SWV) and ETM Instruction Trace.
2. Real-time Read and Write to memory locations for Watch, Memory and RTX Tasks windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also called Access Breaks).
4. RTX Viewer: two real-time kernel awareness windows for the Keil RTX RTOS.
5. A DSP example using ARM CMIS-DSP libraries which are included with MDK with all source code included.

Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and a timestamp. RTX Viewer uses SWV. This information comes from the ARM CoreSight™ debug module integrated into the Cortex-M4. SWV is output on the Serial Wire Output (SWO) pin found on the JTAG connector.

SWV does not steal any CPU cycles and is completely non-intrusive (except for ITM Debug printf Viewer). SWV is provided by the Keil ULINK family and the Segger J-Link. Most complete results are with the ULINK family.

Embedded Trace Macrocell™ (ETM):

ETM records all executed instructions in addition to the features provided by SWV. ETM provides advanced features including Code Coverage, Performance Analysis and Execution Profiling providing both time and calls. ETM requires a special adapter such as the ULINKpro to capture and display ETM trace frames.

Introduction:

1. Freescale Eval boards, Keil software install, Examples, Debug Adapters, CoreSight Definitions: 3
2. Notes for Freescale Tower CPU Boards: 4

Part A: Connecting and Configuring Debug Adapters to the Kinetis Tower board: 5

1. P&E OSJTAG Configuration for the Freescale Tower board: 5
2. Connecting ULINK2, ULINK-ME, J-Link or ULINK*pro* to the Freescale Tower board: 6
3. Configuring ULINK2 or ULINK-ME and μ Vision: 7
4. Configuring ULINK*pro* and μ Vision: 8

Part B: Example Projects: 9

1. Blinky example program using the Kinetis and ULINK2 or ULINK*pro*: 9
2. Hardware Breakpoints: 9
3. Call Stack + Locals Window: 10
4. Watch and Memory Windows and how to use them: 11
 - a. Watch window: 11
 - b. Memory window: 11
 - c. How to view Local Variables in the Watch or Memory windows: 12
5. Getting the Serial Wire Viewer (SWV) working: 13
 - a. For ULINK2 or ULINK-ME: 13
 - b. For ULINK*pro*: 14
6. Using the Logic Analyzer (LA) with ULINK2 or ULINK-ME: 15
7. Watchpoints: Conditional Breakpoints: 16
8. RTX_Blinky example program with Keil RTX RTOS: 17
9. RTX Kernel Awareness using Serial Wire Viewer (SWV): 18
10. Logic Analyzer Window: View variables real-time in a graphical format: 19
11. Serial Wire Viewer (SWV) and how to use it: (with ULINK2): 20
 - a. Data Reads and Writes: 20
 - b. Exceptions and Interrupts: 21
 - c. PC Samples: 22
12. ITM (Instruction Trace Macrocell) a printf feature: 23
13. Trace Configuration Fields (for reference): 24

Part C: DSP Example Project: 26

1. DSP Example using ARM CMSIS-DSP Libraries: 26
2. Signal Timings using Logic Analyzer: and the RTX Tasks window: 27
3. RTX Event Viewer: 28
4. Event Viewer Timing: and Changing the SysTick timer: 29

PART D: ETM Trace with the ULINK*pro*: 30

1. Configuring the ULINK*pro* ETM Trace: 30
2. Blinky Example: ETM Frames starting at RESET and beyond: 32
3. Finding the Trace Frames you are looking for: 33
4. Trace Triggers: 34
5. Code Coverage: and how to save Code Coverage: 35
6. Performance Analyzer (PA): 37
7. Execution Profiler: 38
8. In-The-Weeds Example: 39
9. Serial Wire Viewer summary: 40
10. Keil Products and contact information: 41

FreescalE Evaluation Boards:

This document uses the Freescale Kinetis TWR-K60N512. The K21, K40, K53, K70, KWIKSTIK or other boards can also be used. Please note LEDs E1 through E4 are on Port C on the K40 board and on Port A on the K60 board.

Software Installation:

This document was written using Keil MDK 4.72a. The evaluation copy of MDK (MDK-Lite) is available free on www.keil.com. Do not confuse μ Vision 4 with MDK 4.0. The “4” is a coincidence. MDK 5.0 is released.

To obtain a copy of MDK go to www.keil.com/arm or www.keil.com/freescale and select “Download”.

You can use MDK-Lite and a ULINK2, ULINK-ME, ULINK*pro* or a J-Link for this lab. You can use the OS-JTAG but Serial Wire Viewer (SWV) is not supported. QuickStik has a built-in J-Link Lite. SWV works best with any ULINK.

ULINK*pro* adds Cortex-M4 ETM trace support. It also adds faster programming time and a more sophisticated trace display.

Example Programs:

Two Blinky example programs are included in C:\Keil\ARM\Boards\Freescale\TWR-K60N512. More examples are in various projects for other boards. The DSP example is here: www.keil.com/apnotes/docs/apnt_239.asp

- **Blinky:** blinks 3 LEDs. This is a great starting point and can be adapted to your own project.
- **RTX_Blinky:** A motor controller with RTX (the Keil RTOS) implemented. An easy introduction to RTX.
- **FlexMem_Cfg** and **ProgOnce_Cfg:** See Keil Appnote 220 to help you easily configure the Kinetis FlexMem. It can be found here: www.keil.com/apnotes/docs/apnt_220.asp or www.keil.com/freescale.
- **RL:** Some projects contain an /RL folder. This is where examples for Keil middleware are stored. See <http://www.keil.com/arm/mdk.asp>. These projects can be compiled only with a licensed version of MDK-PROFESSIONAL. Contact your ARM sales office to obtain a temporary license. See the last page for contact info.

USB Debug Adapters:

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

1. **ULINK2 and ULINK-ME:** ULINK2 is pictured on page 1 and ULINK-ME on page 6. ULINK-ME is offered only as part of the TWR-K60N512-KEIL or MCBTWRK60-UME evaluation board packages. ULINK2 can be purchased separately. These are electrically the same and both support Serial Wire Viewer (SWV), Run-time memory reads and writes for the Watch and Memory windows and hardware breakpoint set/unset on-the-fly.
2. **ULINK*pro*:** This is pictured on page 5. ULINK*pro* supports all SWV features and adds ETM Trace support. This means it records all executed instructions. ETM provides complete Code Coverage, Execution Profiling and Performance Analysis features. ULINK*pro* also provides the fastest Flash programming times.

Keil supports more adapters:

1. **CMSIS-DAP:** An extra processor on your board becomes a debug adapter compliant to CMSIS-DAP. The Freedom boards incorporate CMSIS-DAP. μ Vision communicates via USB to the CMSIS-DAP processor and this mode is selected like any adapter in the Target Options menu under the Debug tab. ULINK2 supports CMSIS-DAP.
2. **P&E OSJTAG:** μ Vision running on your PC directly connects to the Kinetis Tower board via a USB connection without any debugging hardware. OSJTAG is good for general debugging but advanced debugging features such as SWV are not implemented. These limitations are listed on page 5 along with the configuration instructions.
3. **Segger J-Link and J-Trace:** J-Link Version 6 (black) or later supports Serial Wire Viewer. J-Trace provides ETM but has not been tested in this document. Data reads and writes are not currently supported with a J-Link.

Serial Wire Viewer and ETM Trace is best supported with one of the Keil ULINKs.

JTAG and SWD Definitions: It is useful to have an understanding of these terms.

JTAG: JTAG provides access to the CoreSight debugging module located on the Kinetis 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 no Boundary Scan. SWD is referenced as SW in the μ Vision Cortex-M Target Driver Setup. SWJ must also be selected.

SWV: Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf (ITM).

SWO: Serial Wire Output: SWV frames come out this 1 bit pin output. It is multiplexed with the JTAG signal TDO.

Trace Port: A 4 bit port that ULINK*pro* uses to output ETM frames and optionally SWV (rather than out the SWO pin).

ETM: Embedded Trace Macrocell: Records all executed instructions. ULINK*pro* provides reliable ETM support.

Notes for Freescale Tower CPU Boards:

K60N512 CPU Clock: In MDK 4.71a and earlier the CPU clock frequency in the examples is set to 47.97 MHz but is actually 56.2 as evidenced by the Serial Wire Viewer (SWV) Core Clock: setting for ULINK2. This is determined in the file `system_MK60N512MD100.c`. A copy of this file is in each example project. In this file, near line 80 is this line: `#define CLOCK_SETUP 0` with corresponding clock configuration starting at line 96 through 114. “0” is the default up to 4.71a.

In the next MDK after 4.71a, a new version of this system file adds an additional option. The reason is the K60N512 board derives its clock from a 50 MHz external crystal. `#define CLOCK_SETUP 3` will result in a Core Clock: of 96 MHz. The new version of this system file is included in the DSP example provided here: www.keil.com/appnotes/docs/apnt_239.asp. If you are using MDK 4.71a or earlier, you can copy the new system file into the example projects if you want to run them at 96 MHz. For 56.2, just use the version supplied. From an example perspective, the only issue you have to worry about when changing the CPU frequency is the Core Clock: setting needed to get SWV working.

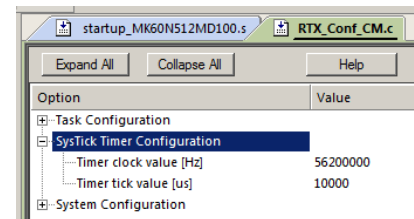
`#define CLOCK_SETUP 0` results in a Core Clock: of 56.2 MHz for SWV and the CPU speed.

`#define CLOCK_SETUP 3` results in a Core Clock: of 96 MHz for SWV and the CPU speed.

The DSP example must run at 56.2 MHz with a ULINK2 or J-Link or the SWV will encounter numerous overrun errors. The ULINK_{pro} is able to run the DSP example at 96 MHz and adequately process the SWV information.

If RTX is in a project where the clock speed is changed, the `#define OS_CLOCK` in the file `RTX_Conf_CM.c` must be changed to reflect the different CPU speed: in this case either 56.2 (shown on the right) or 96 MHz:

You can modify this value using the Configuration Wizard or directly to the file.



Using Tower boards other than the K60N512:

MDK has many examples for Tower boards. See `C:\Keil\ARM\Boards\Freescale` for the list. Some projects contain target options for various USB debug adapters. You can adapt these settings including which flash algorithms to use to your board.

For the K60D100M see www.keil.com/appnotes/docs/apnt_249.asp.

You can adapt this lab to any of these boards. Here are some items you might need to adjust for:

- 1) Core Clock: for SWV trace. Check the file `abstract.txt` for this value. With the ULINK2, this value must be correct in order to collect SWV frames. ULINK_{pro} detects this speed. It is used then to set the value of various timing displays.
- 2) The DSP example is designed for the K60N512 board and runs at 56.2 MHz. If you run it at a much higher speed, SWV might not be able to display all the data without dropping data frames when using a ULINK2 or a J-Link. To solve this issue, use a ULINK_{pro}, slow the clock or select fewer SWV elements to display.

You might have to switch the system and/or startup files to match your particular processor. These files are in the projects.

For the Vybrid Tower board, see www.arm.com/ds5 for a suitable development system.

For general information: www.keil.com/freescale

KL25Z Freedom

www.keil.com/appnotes/docs/apnt_232.asp

K20D50M Freedom Board

www.keil.com/appnotes/docs/apnt_243.asp

Kinetis K60N512 Tower

www.keil.com/appnotes/docs/apnt_239.asp

Kinetis K60D100M Tower

www.keil.com/appnotes/docs/apnt_249.asp

Export Freescale Processor Expert Projects to μ Vision™ Projects

www.keil.com/appnotes/docs/apnt_235.asp

FlexMemory configuration using MDK

www.keil.com/appnotes/files/apnt220.pdf

Part A: Connecting and Configuring Debug Adapters to the Kinetis Tower board:

1) P&E OSJTAG configuration for the Freescale Tower board:

If you are using any ULINK or J-Link: you can skip this page:

µVision supports OSJTAG. This allows debugging the Kinetis Tower with a USB cable. No external adapter is required. You can use the free evaluation version of Keil MDK (MDK-Lite) with OSJTAG to evaluate programs up to 32K.

If you decide to use a ULINK2 or ULINK-ME, you will get Serial Wire Viewer (SWV). With a ULINKpro, ETM Trace is added which records all executed instructions and provides Code Coverage, Execution Profiling and Performance Analysis.


OSJTAG Limitations: (Any ULINK provides these options)

1. Hardware Breakpoints can't be set on-the-fly while the program is running.
2. No Watchpoints.
3. No Serial Wire Viewer or ETM Trace support.
4. No on-the-fly memory read or write updates to the Watch and Memory windows.
5. No RTX Kernel Awareness window updating.

Install P&E Drivers: (you do not need to do these steps for any ULINK or J-Link adapters)

1. Download the drivers from www.keil.com/download/docs/408.asp. Filename is currently fslkinetisdriversv114.exe
2. Disconnect the Kinetis board from the USB cable and close µVision if it is running.
3. Run fslkinetisdriversv114.exe (or a later version if available) to install the OSJTAG drivers. This action will add P&E OSJTAG in the adapter window as shown here:

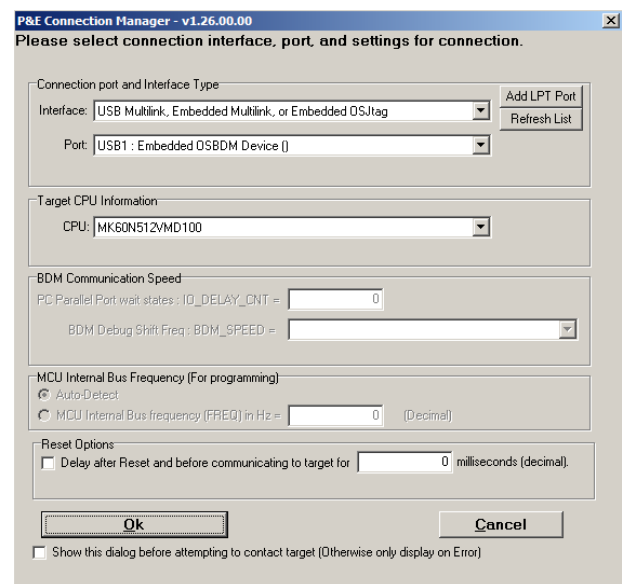
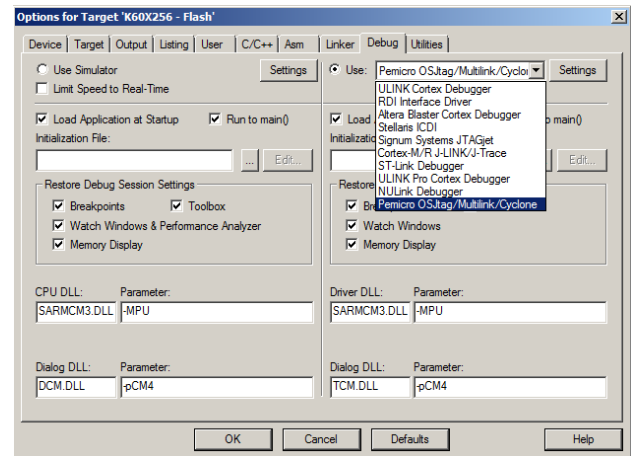
Configure µVision:

1. Start µVision by clicking on its desktop icon.
2. Leave it in Edit mode (do not enter Debug mode).
3. Select Project/Open Project.
4. Open the Blinky project for your board in C:\Keil\ARM\Boards\Freescale\
5. Plug a USB cable to J13 on the Kinetis K60 board.
6. Allow the USB driver initialization as required.
7. Select Options for Target  or ALT-F7. Click on the Debug tab to select a debug adapter.
8. Select Pemicro OSJtag/... as shown here:
9. Click on Settings: and the next window opens.
10. Select your exact processor in the Target CPU Information dialog box. This step is very important.
11. If you get any errors you will be notified with the probable cause of the problem.
12. Click on OK.
13. Click on the Utilities tab to select a Flash programmer.
14. Select Pemicro OSJtag/... as before.
15. No other settings are necessary. Click on OK.
16. Select File/Save All.

OSJTAG is now completely configured.

You can compile, program Flash or RAM, enter Debug mode and run/stop your program at this time.

TIP: Select Update Target before Debugging in the Utilities tab to program the Kinetis Flash when Debug mode is entered.



2) Connecting a ULINK2, ULINK-ME or ULINKpro to the Freescale Tower board:

Freescale provides the ARM standard 20 pin Cortex Debug connector for JTAG/SWD and ETM connection as shown here:

Pins 1 through 10 provide JTAG, SWD and SWO signals. Pins 11 through 20 provide the ETM trace signals.

ARM also provides a 10 pin standard connector that provides the first 10 pins of the 20 pin but this is not installed on the Kinetis board. ARM recommends that both the 10 and 20 pin connectors be placed on target boards.

Pin 7 on both the 10 and 20 pin is a key. On the male connector, pin 7 is supposed to be absent. This is not the case on the Kinetis board.

Keil cables might have pin 7 filled with a plastic plug and if so this will need to be removed before connecting to the Kinetis target. This is easily done with a sharp needle. Merely pry the pin out.

Alternatively, you can cut pin 7 off the target connector. This is more difficult to do. Cable orientation is provided by the socket itself and there is no chance for reverse orientation.

It is impossible to plug a 10 pin plug into the 20 pin socket without bending two pins on the socket.

Connecting ULINK2 or ULINK-ME:

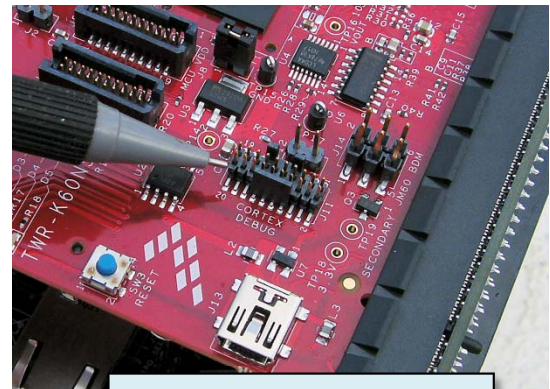
Pictured is the 10 pin to 20 pin Keil connector. The arrows point to pin 1. This cable is supplied with the ULINK2 and ULINK-ME. This cable can also be ordered by contacting Keil sales or tech support. The part number is ULC-2010A or B.

You will need to take the case off the ULINK2 and install the special cable. The ULINK-ME does not have a case and the cable can be directly installed on the 10 pin connector. The ULINK-ME is pictured bottom right and the arrow points the 10 pin connector.

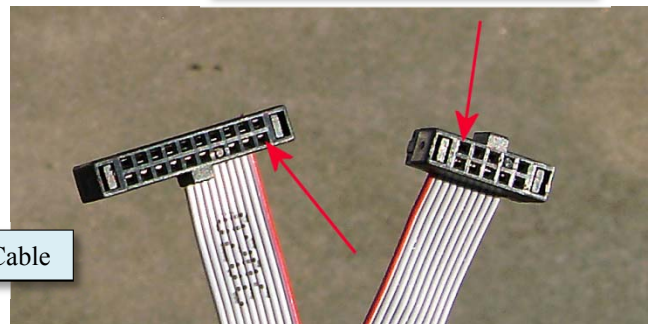
Connecting a ULINKpro:

The ULINKpro connects directly to the Kinetis board with its standard 20 pin connector. You might need to remove the key pin to connect to the Kinetis target as described above.

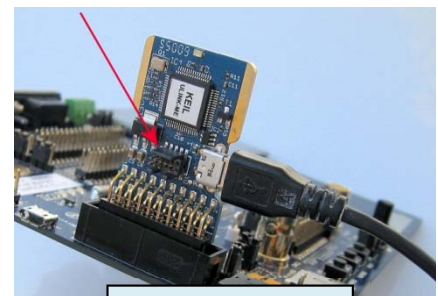
Power: Power the board with a USB cable as shown below (J13) or the method specified for your board.



20 Pin Cortex Debug Connector



10 to 20 Pin Cable



Keil ULINK-ME



Segger Cortex Adapter



ULINKpro connected to a K60 Tower

J-Link: Segger provides an adapter to go from the large 20 pin connector to the 10 and 20 pin Cortex connectors as shown above. Contact Segger to purchase this adapter: www.segger.com

Connector Part Numbers: The 10 pin male connector as shown on the ULINK-ME is Samtec part number FTSH-105. The 20 pin ETM connector as used on the Kinetis boards is: FTSH-110. You will have to add appropriate suffixes for guide options.

TIP: Want to purchase some of the connectors used on the Tower system? They are actually standard 32 bit PCI sockets as used on personal desktop computers. These connectors are easy to find.

3) Configuring ULINK2 or ULINK-ME and µVision: (similar instructions for J-Link)


If you are using a ULINKpro or OSJTAG, you can skip this page:

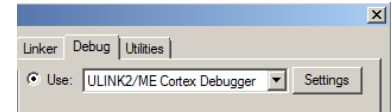
It is easy to select a USB debugging adapter in µVision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are selected using the Debug and Utilities tabs.

This document will use a ULINK2 or ULINK-ME as described. You can substitute a ULINKpro with suitable adjustments.

Serial Wire Viewer is supported by all three adapters. ULINK2 and ULINK-ME are essentially the same devices electrically and any reference to ULINK2 here includes the ME. The ULINKpro, which is a Cortex-M ETM trace adapter, can be used like a ULINK2 or ULINK-ME with the advantages of faster programming time and an enhanced instruction trace window.

1) Select the debug connection to the target:

1. Assume the ULINK2 is connected to a powered up Kinetis target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK2 is shown connected to the Freescale K60 Tower board on page 1.
2. Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select ULINK2/ME Cortex Debugger as shown here:
3. Select Settings and the next window below opens up. This is the control panel for the ULINK 2 and ULINK-ME (they are the same).
4. In **Port:** select SWJ and SW. Serial Wire Viewer (SWV) will not work with JTAG selected.



TIP: J-Link does not have a SWJ setting. This is enabled automatically by the J-Link.

5. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board.

TIP: To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.

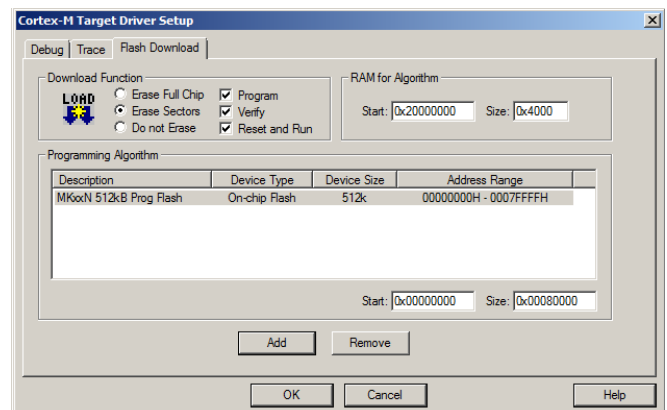
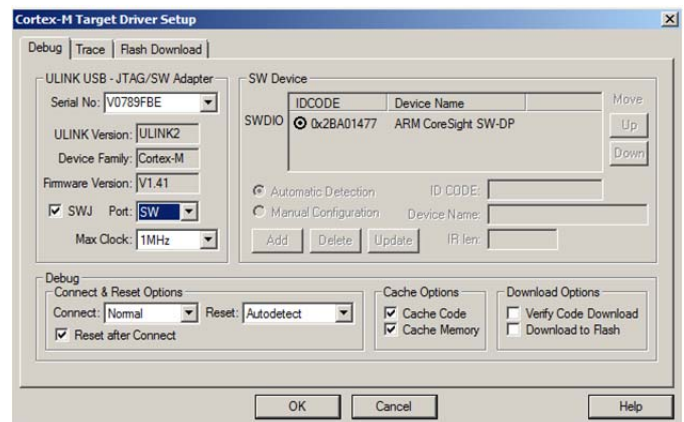
TIP: You can do regular debugging using either JTAG or SWD. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will operate only in SW (SWD) mode.

2) Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ULINK similar to Step 2 above.
8. Click Settings to select a programming algorithm.
9. If an algorithm is not already selected, select Add and select MKxxN 512kB Prog Flash as shown below or the one for your processor:
10. Click on Add to select your chosen algorithm.
11. Click on OK once.
12. Click on OK to return to the main screen.
13. You have successfully connected to Kinetis.
14. At this point you can compile source code, program it into Flash, enter Debug mode, start and stop your program and set/unset breakpoints plus much more.

TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this later.

TIP: If you select ULINK or ULINKpro, and have the opposite ULINK physically connected to your PC; the error message will say “No ULINK device found”. This message actually means that µVision found the wrong Keil adapter connected. Either select or connect the right ULINK.




4) Configuring ULINKpro and µVision:

KN60N512 examples are configured for ULINKpro by default. These steps are included for reference.

1. Assume a ULINKpro is connected to a powered up Kinetis target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINKpro is shown connected to the Freescale K60 Tower board on page 6.

1) Select the debug connection to the target:

2. Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select the ULINK Pro Cortex as shown here:
3. Select Settings and Target Driver window below opens up:
4. In **Port:** select SWJ and SW. SWV will not work with JTAG selected unless you are using the 4 bit Trace Port. See the second TIP: below:
5. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board.

TIP: To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.

TIP: You can do regular debugging using JTAG or SWD. SWD (also called SW) and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode unless the ULINKpro is using the Trace Port with ULINKpro to output the trace frames rather than the 1 bit SWO pin. This is selected in the Trace tab and is discussed later.

2) Configure the Keil Flash Programmer:

1. Click on OK once and select the Utilities tab.
2. Select the ULINKpro similar to Step 2 above.
3. Click Settings to select a programming algorithm.
4. If one is not visible click Add and select MKxxN 512kB Prog Flash as shown below or the appropriate one for your processor:
5. Click on Add to select your chosen algorithm.
6. Click on OK once.

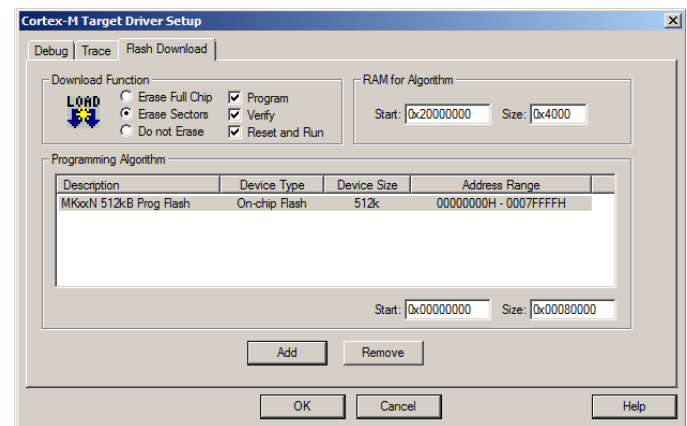
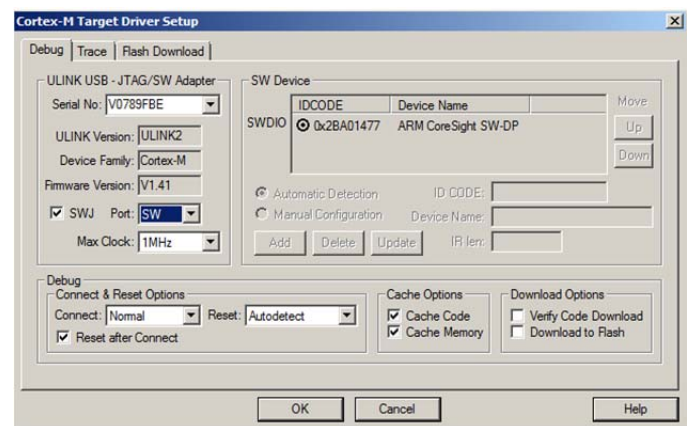
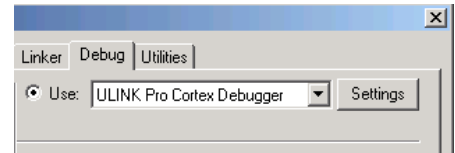
TIP: To program the Flash every time you enter Debug mode, check Update target before Debugging.

7. Click on OK to return to the µVision main screen.
8. You have successfully connected to the Kinetis target processor.
9. At this point you can compile source code, program then into Flash, enter Debug mode, start and stop these programs and set/unset breakpoints and much more.

TIP: If you select ULINK or ULINKpro, and have the opposite ULINK physically connected; the error message will say “No ULINK device found”. This message actually means that µVision found the wrong Keil adapter connected.

TIP: A ULINKpro will act very similar to a ULINK2. The trace window (Trace Data) is quite different from the ULINK2 Trace Records as it offers additional features.

TIP: µVision windows can be floated anywhere. You can restore them by setting Window/Reset Views to default. Using two monitors is also supported.



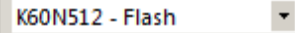
Part B: Keil Example Projects

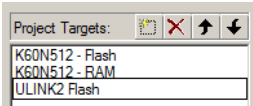
1) *Blinky* example program using the Kinetis and ULINK2 or ULINK-ME:

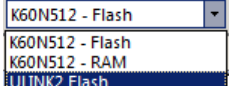
We will run the example program Blinky on a Kinetis processor using a ULINK2 or ULINK-ME. These instructions use a K60N512 Tower board. If you are using another Tower board or the Kwikstik, select the appropriate Blinky project. If you are unable to get this example working with your Kinetis Tower board, please contact the author or Keil tech support.



It is possible to use the ULINK^{pro} or the P&E OSJTAG with this example. The Segger J-Link is also supported. ULINK^{pro} is selected by default. We will add a Target Option for the ULINK2. You could also just modify the existing target option.


1. Connect the ULINK2 as pictured on the first page. Use the special 10 to 20 pin cable for both the ULINK2 and ULINK-ME. See page 5 for P&E. ULINK^{pro} is configured with this Blinky project. Connect it as on page 6.
2. Start µVision by clicking on its desktop icon. 
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky\Blinky.uvproj.





4. Make sure “K60N512 Flash” is selected:  This is where you select different target configurations such as to execute a program in RAM or Flash. We will make a copy of this one and configure it to use a ULINK2. You can create many target options and select them. Projects for other Kinetis processors contain various target options that you can use as a template for your projects.
5. In the main µVision menu, click on Project/Manage and select Components, Environment, Books...

6. Select the NEW icon or press the INSERT key on your keyboard.
7. In the space that appears: enter the name of your target option. I chose ULINK2 Flash: 

8. Click on OK to enter it and close this window.
9. Select your new Target Option as shown here: 

10. At this point, you can select Options for Target  and select the adapter you are using. See instructions on previous pages to accomplish this. ULINK2 is on page 7. Remember to select the Flash programmer too.
11. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.

TIP: Select Options for Target  and select the Target tab: select MicroLIB to make your compilations smaller.

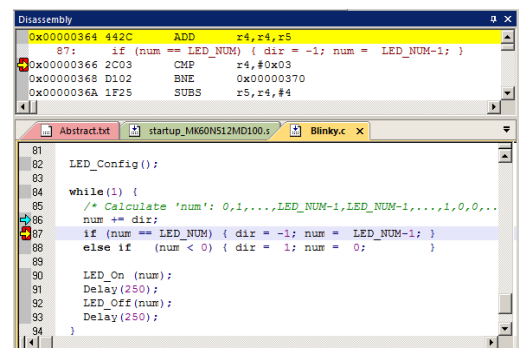
12. Program the Kinetis flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
13. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
Note: You only need to use the Load icon to download to FLASH and not for RAM operation or the simulator.
14. Click on the RUN icon.  Note: you stop the program with the STOP icon. 

The LEDs E1 through E3 on the Kinetis board will now blink in sequence.

Now you know how to compile a program, load it into the Kinetis processor Flash, run it and stop it.

2) Hardware Breakpoints:

1. With Blinky running, click in the margin in the source file Blinky.c on a darker gray block somewhere in the while(1) loop as shown below:
2. A red circle is created and soon the program will stop at this point.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows.
4. Recall you can set and unset hardware breakpoints while the program is running.
5. The Kinetis has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set on.
6. Remove the breakpoint by clicking on the red circle.



8) Call Stack + Locals Window:

Local Variables:

The Call Stack and Local 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.

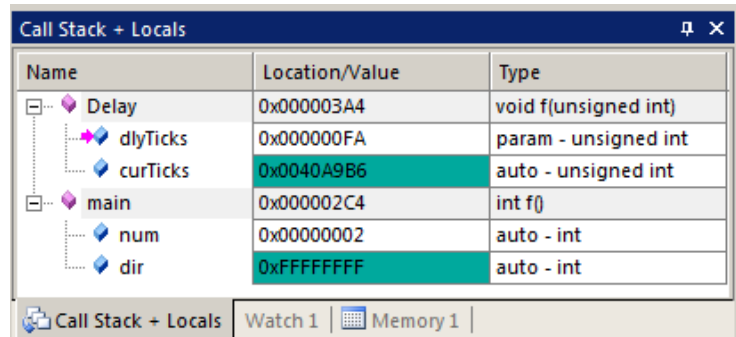
1. Run and Stop Blinky. Click on the Call Stack + Locals tab.
2. Shown is the Call Stack + Locals window.

The contents of the local variables are displayed as well as names of active functions. Each function name will be displayed as it is called from the function before it or from an interrupt or exception.




When a function exits, it is removed from the list.

The first called function is at the bottom of this table.

This table is active only when the program is stopped.



Name	Location/Value	Type
Delay	0x000003A4	void f(unsigned int)
dlyTicks	0x000000FA	param - unsigned int
curTicks	0x0040A9B6	auto - unsigned int
main	0x000002C4	int f()
num	0x00000002	auto - int
dir	0xFFFFFFFF	auto - int

3. Click on the Step In icon or F11: 
4. Note the function different functions displayed as you step through them. If you get trapped in the Delay function, use Step Out  or Ctrl-F11 to exit it immediately.
5. If available, clicking numerous times on Step In will display other functions. This Blinky example is too simple to display many deep levels of function calls.
6. Right click on a function name and try the Show Callee Code and Show Caller Code options as shown here:
This will be indicated in the appropriate source file and disassembly windows.
7. Click on the StepOut icon  to exit all functions to return to main().

TIP: If single step (Step In) doesn't work as you expect it to, click on the Disassembly window to bring it into focus. If needed, click on a disassembly line to step through assembly instructions. If a source window is in focus, you will step through the source lines instead.

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

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 or structures updated in real-time in the Watch and Memory windows while the program is running. You can modify variables in the Memory window. 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. Structures are always available as are peripheral and raw memory addresses.

If you have a ULINK_{pro} and ETM trace, you can see a record of all the instructions executed. The Disassembly and Source windows show your code in the order it was written. The ETM trace shows it in the order it was executed. ETM also provides Code Coverage, Performance Analysis and Execution Profiling.

Changing a local variable to a static or global normally means it is moved from a CPU register to RAM. CoreSight can view RAM or other memory mapped locations but not CPU registers when the program is running.

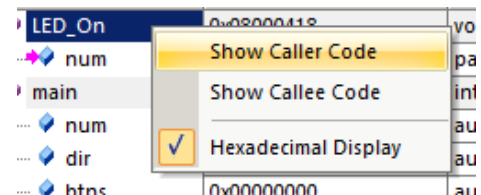
Call Stack:

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 what return data is stored on the stack.

TIP: You can modify a variable value when the program is stopped. You can also modify a CPU register at this time.

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.


Selecting Debug/Kill All Breakpoints in the Breakpoints window (Ctrl-B) deletes all Breakpoints but no Watchpoints.




4) Watch and Memory Windows and how to use them:

The Watch and Memory windows will display updated variable values in real-time. It does this using ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to “put” or insert values into memory locations using the Memory window in real-time. It is possible to “drag and drop” variables into windows or enter them manually.

a) Watch window:

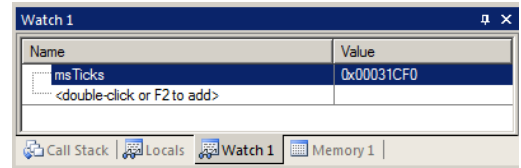
1. You can do the following steps while the CPU is running. Click on RUN if desired.
2. In the file Blinky.c locate the volatile variable **msTicks**. It will be near line 19.
3. In Blinky.c, right click on **msTicks** and select Add msTicks to ... and select Watch 1. **msTicks** will be displayed
4. This action will open the Watch window if it is not already open and display **msTicks** as shown below:
5. If Blinky is running, **msTicks** will update in real-time. If not, click on RUN  to start Blinky.

TIP: You can also block **msTicks**, click and hold and drag it into Watch 1. Release it and it will be displayed updating as shown here: 


You can also enter a variable manually by double-clicking or pressing F2 and using copy and paste or typing the variable name in.

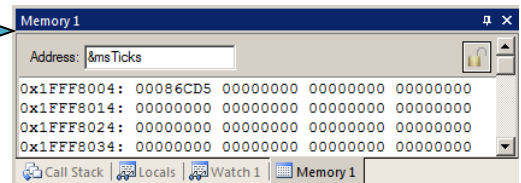
These three methods of entering variables also work with the Memory and Logic Analyzer windows. Variables can be selected from the Symbol table. This also has the benefit of fully qualifying them (where they are located).

TIP: To Drag ‘n Drop into a tab that is not active, pick up the variable and hold it over the tab you want to open; when it opens, move your mouse into the window and release the variable.



b) Memory window:

1. In Blinky.c, right click on **msTicks** and this time select Add msTicks to ... and select Memory 1.
2. Note the value of **msTicks** is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to, but this is not what we want to see at this time.
3. Add an ampersand “&” in front of the variable name. Now the physical address is shown (0x1FFF 8004). Note: the address where **msTicks** is located might be slightly different in your program but it will be displayed the same way.
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **msTicks** is displayed as shown here: 
6. Both the Watch and Memory windows are updated in real-time.
7. Right-click on a memory location and select Modify Memory. You can change the value of this location while the program is running. The Watch window can be modified when stopped.



TIP: You are able to configure the Watch and Memory windows while the program is still running in real-time without stealing any CPU cycles. You are able to modify the variable values in a Memory window in real-time.

How It Works:

µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. Remember the Cortex-M4 is a Harvard architecture. This means it has separate instruction and data buses. While the CPU is fetching instructions at full speed, there is plenty of time for the CoreSight debug module to read or write values without stealing any CPU cycles.

This can be slightly intrusive in the unlikely event the CPU and µVision reads or writes to the same memory location at exactly the same time. Then the CPU will be stalled for one clock cycle. In practice, this cycle stealing never happens.

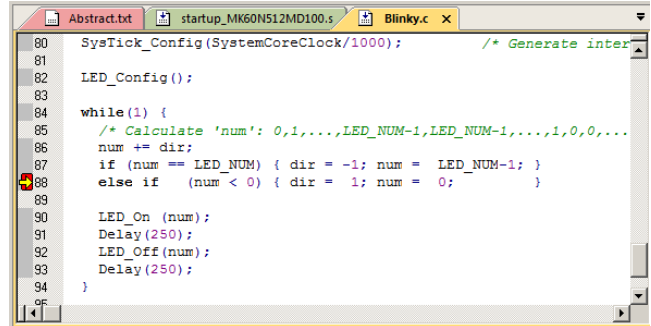
TIP: You are not able to view local variables while the program is running. To view local variables in real-time: convert them to static or global variables.

c) How to view Local Variables in the Watch or Memory windows:

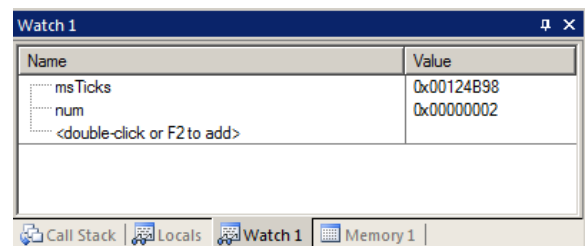
1. Make sure Blinky.c is running.
2. Locate where the local variable `num` is declared in Blinky.c near line 76, at the start of the main function.
3. In Blinky.c, right click on `num` and select Add `num` to ... and select Watch 1 where `num` will now be displayed.
4. The value of `num` is displayed as <cannot evaluate> because μ Vision can't read locals while the program is running.
5. Click in the left margin, somewhere in the while (1) loop in main to set a hardware breakpoint such as at line 88 shown here:
6. The program will soon stop here. Each time you click RUN, this value is updated.

TIP: Remember: you can set/unset hardware breakpoints on-the-fly in the Cortex-M processors !

7. Note the `num` value now displays its correct value because it is now in scope. This is shown in the second screen below:
8. Remove the breakpoint by clicking on it to remove the red circle.
9. μ Vision is unable to determine the value of `num` when the program is running since it exists only when main is running. `num` disappears in functions and handlers outside of main. In addition, `num` is a local variable and therefore is normally temporarily stored in a CPU register. μ Vision cannot read these while the program is running.
10. Start the program by clicking on the Run icon.
11. `num` changes to <cannot evaluate>.



```
80 SysTick_Config(SystemCoreClock/1000); /* Generate inter
81
82 LED_Config();
83
84 while(1) {
85     /* Calculate 'num': 0,1,...,LED_NUM-1,LED_NUM-1,...,1,0,0,...
86     num += dir;
87     if (num == LED_NUM) { dir = -1; num = LED_NUM-1; }
88     else if (num < 0) { dir = 1; num = 0; }
89
90     LED_On (num);
91     Delay(250);
92     LED_Off(num);
93     Delay(250);
94 }
```






Name	Value
msTicks	0x00124B98
num	0x00000002
<double-click or F2 to add>	


How to view these variables updated in real-time:




All you need to do is to make `num` static ! This changes it from existing in a CPU register to a RAM location.

1. In the declaration for `num` add `static` like this:

```
int main (void) {
    static int num = -1;
```

2. Stop  the program and exit Debug mode . **TIP:** You can edit files in edit or debug mode, but can compile them only in edit mode.
3. Compile the source files by clicking on the Rebuild icon. Hopefully they compile with no errors or warnings.
4. To program the Flash, click on the Load icon . A progress bar will be displayed at the bottom left.

TIP: To program the Flash automatically when you enter Debug mode select Options For Target , select the Utilities tab and select the “Update Target before Debugging” box.

5. Enter Debug mode. 
6. Click on RUN.
7. `num` is now updated in real-time. This is ARM CoreSight technology working.
8. Stop the CPU  and exit debug mode  for the next step.

TIP: View/Periodic Window Update must be selected. Otherwise, variables update only when the program is stopped.


TIP: Sometimes when you change a variable type, μ Vision will not be able to find it. You can re-enter it as previously described. Sometimes it helps if the variable is fully qualified. In this case it would be `\Blinky\main\num`. You can drag a variable from the Symbols window into a Watch or Memory window and it will be entered fully qualified.

5) Getting the Serial Wire Viewer (SWV) working:

Serial Wire Viewer provides data trace information including interrupts in real-time without any code stubs in your sources.




a) For ULINK2 or ULINK-ME: Configuration must be set as on page 7 (ULINK_{pro} instructions are on page 8). Segger J-Link instructions are very similar to ULINK2. The differences are intuitive to work around.

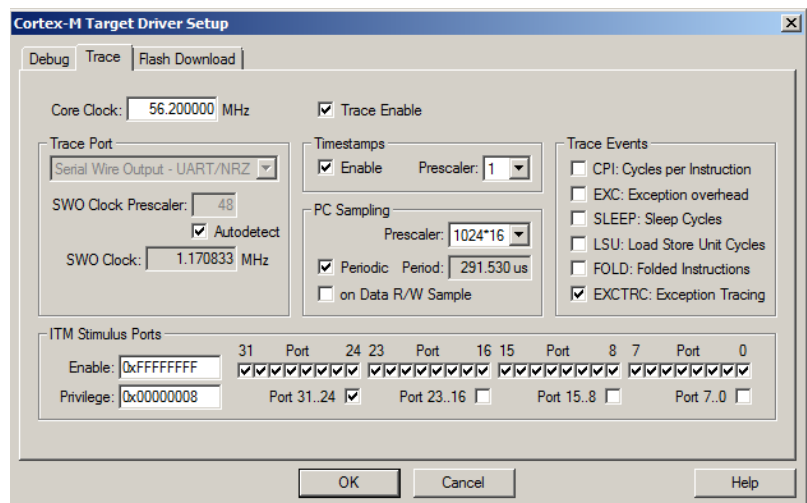
Configure SWV:

1. µVision must be stopped and in edit mode (not debug mode).
2. Select Options for Target  or ALT-F7 and select the Debug tab.
3. Click on Settings: beside the name of your adapter on the right side of the window.
4. Confirm Port: is set to SW and SWJ box is selected for SWD operation instead of JTAG.
5. Click on the Trace tab. The window below is displayed.
6. In Core Clock: enter 56.2 for MDK 4.71a and earlier or 96 for later versions of MDK. Select Trace Enable.
7. Select Periodic and leave everything else at default. Periodic activates PC Samples and is a good test for SWV.
8. Click on OK twice to return to the main µVision menu. SWV is now configured and ready to use.

TIP: This window can also be opened in Debug mode by selecting Debug/Debug Settings.

Display Trace Records:

9. Enter Debug mode. 
10. Click on the RUN icon. 
11. Open Trace Records window by clicking on the small arrow beside the Trace icon: 
12. The Trace Records window will open and display Exceptions and PC Samples as shown:



TIP: Remember the Core Clock: must be correct for ULINK2 and ULINK-ME or you will see spurious frames or none at all. If you see the Num column contain any numbers other than 15 or 0, the trace is not configured correctly. The most probable cause is the Core Clock: frequency is wrong. Try 56.2, 47.97 or 96 MHz. Use a scope on the SWO pin if needed to determine the value of Core Clock: Measure the time period of the narrowest pulse and invert it to get Core Clock:.

ULINK_{pro} uses this only for timing purposes since Manchester encoding is used out the SWO pin which is self-clocking.

Exception 15 is the SYSTICK timer. It is a dedicated timer for use with an RTOS.

All frames have a timestamp displayed.

Exception Return means all exceptions have returned. This can be used to detect Cortex exception tail-chaining.

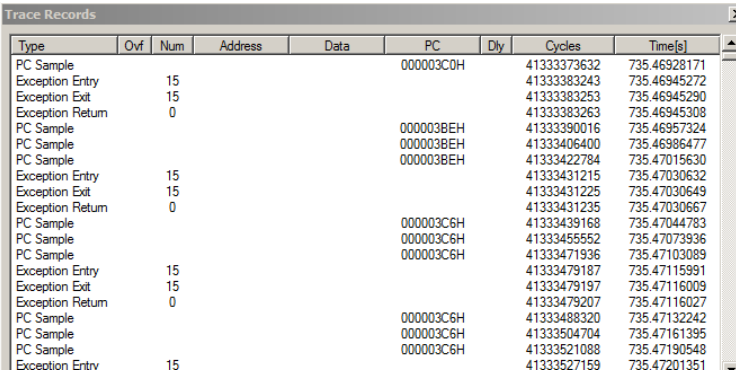
If you open the Trace Exceptions window you will see SYSTICK displayed by Exception number.

Double-click inside any of these two windows to clear them.

Right click in the Trace Records window and you can filter out various types of frames for easier viewing.

This is an easy way to see when your interrupts are occurring and how often.


There are more things you can do with SWV which you will see in the next few pages.



Type	Ofv	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					000003C0H		4133373632	735.46928171
Exception Entry		15					41333383243	735.46945272
Exception Exit		15					41333383253	735.46945290
Exception Return		0					41333383263	735.46945308
PC Sample					000003BEH		41333390016	735.46957324
PC Sample					000003BEH		41333406400	735.46986477
PC Sample					000003BEH		41333422784	735.47015630
Exception Entry		15					41333431215	735.47030632
Exception Exit		15					41333431225	735.47030649
Exception Return		0					41333431235	735.47030667
PC Sample					000003C6H		41333439168	735.47044783
PC Sample					000003C6H		41333455552	735.47073936
PC Sample					000003C6H		41333471936	735.47103089
Exception Entry		15					41333479187	735.47115991
Exception Exit		15					41333479197	735.47116009
Exception Return		0					41333479207	735.47116027
PC Sample					000003C6H		41333488320	735.47132242
PC Sample					000003C6H		41333504704	735.47161395
PC Sample					000003C6H		41333521088	735.47190548
Exception Entry		15					41333527159	735.47201351

b) For **ULINKpro**: This is not ETM trace. See page 30 for ETM trace.

1) Configure SWV:





1. μ Vision must be stopped and in edit mode (not debug mode).
2. Confirm ULINKpro is configured as found on page 8: 4) Configuring ULINKpro and μ Vision: SW MUST be selected as well as SWJ.
3. Select Options for Target  or ALT-F7 and select the Debug tab.
4. Click on Settings: beside the name of your adapter on the right side of the window.
5. Click on the Trace tab. The window below is displayed.
6. Core Clock: ULINKpro determines this automatically. It uses this only to calculate timings. Enter 56.2 MHz for MDK 4.71a and earlier and 96 MHz for later versions. The DSP example always uses 56.2 MHz.
7. Select the Trace Enable box.
8. In the Trace Port box select Serial Wire Output – Manchester. Selecting UART/NRZ will cause an error.
9. Select Periodic and leave everything else at default. Periodic activates PC Samples and is a good test.
10. Click on OK twice to return to the main μ Vision menu. SWV is now configured and ready to use.

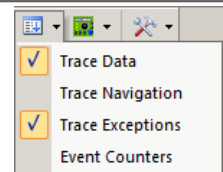
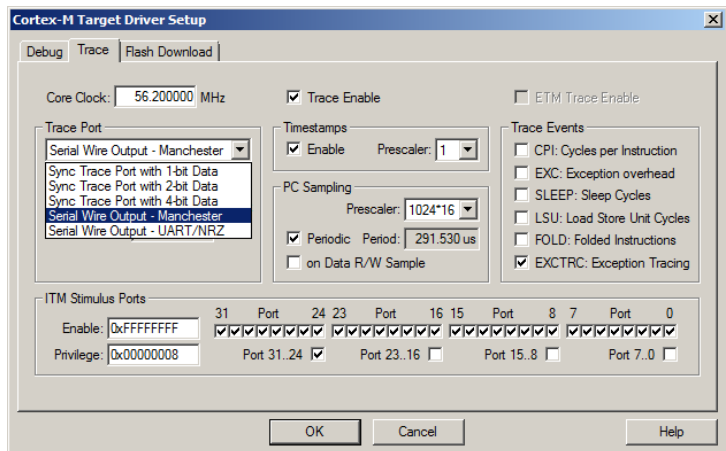
TIP: Sync Trace Port with 4 bit Data field sends the trace and SWV records out the 4 bit trace port rather than the single pin SWO. The Trace Port is faster and must be selected for ETM trace. It is available only with the ULINKpro.

We will examine this setting later on page 31.


TIP: You can also access this window in Debug mode by selecting Debug/Debug Settings and selecting the Trace tab.


2) Display the Trace Data window:

1. Enter Debug mode. 
2. Click on the RUN icon. 
3. Open Data Trace window by clicking on the small arrow beside the Trace Data icon:  You can also open the Trace Exceptions window at this time.
4. The Trace Data window will open.
5. STOP  the program to display the Exceptions and PC Samples as shown below:
Currently, ULINKpro does not update the Trace Data window while the program is running.
The exceptions window does update in real-time.



TIP: The Instruction Trace window is different than the Trace Records window provided with the ULINK2. Note the disassembled instructions are displayed and if available, the source code is also displayed. Clicking on a PC Sample line will take you to that place in the source and disassembly windows. All the executed instructions are displayed with ETM trace.

Clear the Trace Data window by clicking the red X. 







The contents of the Trace Data window can be saved to a file. 

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
11.951 005 658 s	X: 0x000003BE	BCC 0x000003B6	
11.951 297 189 s	X: 0x000003BE	BCC 0x000003B6	
11.951 588 719 s	X: 0x000003BE	BCC 0x000003B6	
11.951 771 530 s		Exception Entry - SysTick	
11.951 771 708 s		Exception Exit - SysTick	
11.951 771 886 s		Exception Return	
11.951 880 249 s	X: 0x000003BA	SUBS r2,r2,r1	
11.952 171 779 s	X: 0x000003BA	SUBS r2,r2,r1	
11.952 463 310 s	X: 0x000003BA	SUBS r2,r2,r1	
11.952 625 125 s		Exception Entry - SysTick	
11.952 625 302 s		Exception Exit - SysTick	
11.952 625 480 s		Exception Return	
11.952 754 840 s	X: 0x000003B8	LDR r2,[r2,#0x00]	
11.953 046 370 s	X: 0x000003B8	LDR r2,[r2,#0x00]	
11.953 337 900 s	X: 0x000003B8	LDR r2,[r2,#0x00]	
11.953 478 719 s		Exception Entry - SysTick	
11.953 478 897 s		Exception Exit - SysTick	
11.953 479 075 s		Exception Return	

6) Using the Logic Analyzer (LA) with ULINK2 or ULINK-ME:


This example will use the ULINK2 with the Blinky example. Please connect a ULINK2 or ULINK-ME to your Kinetis board and configure it for Serial Wire Viewer (SWV) trace. If you want to use a ULINK_{pro} you will have to make appropriate modifications to the configuration instructions. This exercise does not require ETM configuration.

µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using the Serial Wire Viewer as implemented in the Kinetis. This is shared with the Watchpoints.

1. SWV must be configured as found on page 13 or page 14 for ULINK_{pro}. Exit debug mode if not already. 
2. In Blinky.c, near line 32, add the line: `msTicks=0;` just after `uint32_t curTicks;` and before `curTicks = msTicks;`
3. Compile the source files by clicking on the Rebuild icon. 
4. Program the Kinetis flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
6. Select Debug/Debug Settings and select the Trace tab.
7. Unselect Periodic and EXCTRC. This is to prevent SWV overrun. Click OK twice to return to the main menu.
8. Run the program.  **TIP:** Recall you can configure the LA while the program is running or stopped.
9. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 
10. Locate the volatile variable `msTicks` in Blinky.c. It is declared near line 19.
11. Right click on `msTicks` and select Add `msTicks` to... and select Logic Analyzer.
12. Click on Setup and set Max: in Display Range to 0xFF. Click on close. The LA is completely configured now.
13. `msTicks` should still be in the Watch and Memory windows. It should be incrementing if the program is running.
14. Adjust the Zoom OUT icon in the LA window to about 50 msec or so to get a nice ramp as shown below.
15. In the Memory 1 window, right click on the `msTicks` value and enter 0 and press Enter.
16. This modified value will be displayed in the LA window as shown here: You can enter any value into `msTicks`.

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

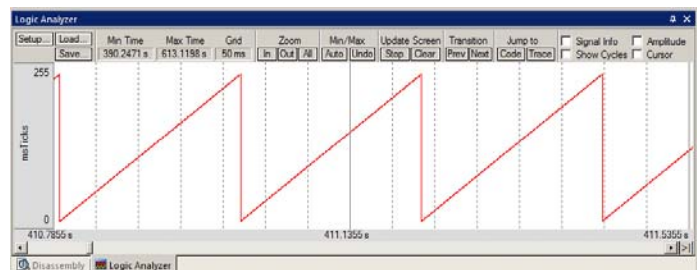
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers enter them into the Logic Analyzer and write to them.

1. Select Debug/Debug Settings and select the Trace tab.
2. Select On Data R/W Sample. Click OK. This adds the PC column shown below.
3. Run the program. 
4. Open the Trace Records window.
5. The window similar to below opens up:
6. The first line in Trace Records here means:
7. The instruction at 0x2C4 caused a write of data 0x0000_F07D to address 0x1FFF 8008 at the listed time in Cycles and Time.

TIP: The PC column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin.

TIP: The ULINK_{pro} will give a more sophisticated Instruction Trace window.

Watchpoints are described on the next page.



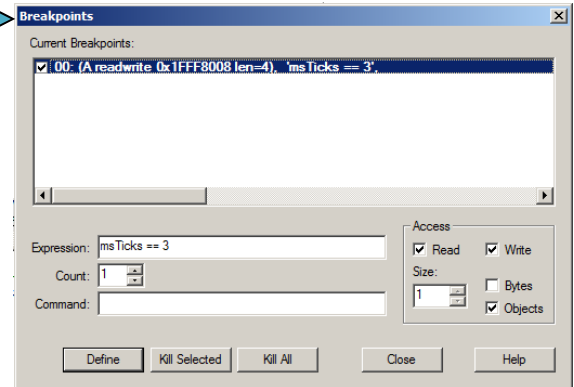
Type	Ofv	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			1FFF8008H	0000F07DH	00002C4H		5734717783	103.63255022
Data Write			1FFF8008H	0000F07EH	00002C4H		5734765755	103.63340382
Data Write			1FFF8008H	0000F07FH	00002C4H		5734813727	103.63425741
Data Write			1FFF8008H	0000F080H	00002C4H		5734861699	103.63511101
Data Write			1FFF8008H	0000F081H	00002C4H		5734909671	103.63596460
Data Write			1FFF8008H	0000F082H	00002C4H		5734957643	103.63681820
Data Write			1FFF8008H	0000F083H	00002C4H		5735005615	103.63767179
Data Write			1FFF8008H	0000F084H	00002C4H		5735053587	103.63852538
Data Write			1FFF8008H	0000F085H	00002C4H		5735101559	103.63937898
Data Write			1FFF8008H	0000F086H	00002C4H		5735149531	103.64023257
Data Write			1FFF8008H	0000F087H	00002C4H		5735197503	103.64108617
Data Write			1FFF8008H	0000F088H	00002C4H		5735245475	103.64193976
Data Write			1FFF8008H	0000F089H	00002C4H		5735293447	103.64279336
Data Write			1FFF8008H	0000F08AH	00002C4H		5735341419	103.64364695
Data Write			1FFF8008H	0000F08BH	00002C4H		5735389391	103.64450054
Data Write			1FFF8008H	0000F08CH	00002C4H		5735437363	103.64535414
Data Write			1FFF8008H	0000F08DH	00002C4H		5735485335	103.64620773
Data Write			1FFF8008H	0000F08EH	00002C4H		5735533307	103.64706133
Data Write			1FFF8008H	0000F08FH	00002C4H		5735581279	103.64791492
Data Write			1FFF8008H	0000F090H	00002C4H		5735629251	103.64876852

7) Watchpoints: Conditional Breakpoints

The Kinetis Cortex-M4 processors have four Watchpoint comparators. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same CoreSight components as Watchpoints in its operations. This means you must have at least two variables out of the four not used in the Logic Analyzer to use Watchpoints.

Each Watchpoint uses two comparators out of four. μ Vision warns you if you have set too many Watchpoints and/or breakpoints. SWV or ETM do not need to be configured and any ULINK or J-Link can be used for Watchpoints.


1. Using the Blinky example from the previous page, stop the program. Stay in Debug mode.
2. Click on Debug and select Breakpoints or press Ctrl-B.
3. The Trace does not need to be configured for Watchpoints. However, we will use it in this exercise.
4. In the Expression box enter: "msTicks == 3" without the quotes. Select both the Read and Write Access.
5. Click on Define and it will be accepted as shown here:
6. Click on Close.
7. Double-click in the Trace Records window to clear it.
8. Set `msTicks` in the Watch 1 or Memory 1 window to 4. This is to allow the program to run for a bit.
9. Click on RUN.
10. When `msTicks` equals 3, the program will stop. This is how a Watchpoint works.
11. You will see `msTicks` incremented in the Logic Analyzer as well as in the Watch window if you choose a Watchpoint with a higher value.



This is not so easy to see since `msTicks` rolls over very fast in this example.

12. Note the three data writes in the Trace Records window shown below. 1, 2 and 3 in the Data column. Plus the address the data written to and the PC of the write instruction. This is with a ULINK2 or ULINK-ME. The ULINK_{pro} will display a different window and the program must be stopped to display the trace frames.
13. There are other types of expressions you can enter and they are detailed in the Help button in the Breakpoints window.

Type	Ovt	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			1FFF8008H	00000001H	000002C4H		27572147844	492.19892142
Data Write			1FFF8008H	00000002H	000002C4H		27572195816	492.19977501
Data Write			1FFF8008H	00000003H	000002C4H		27572243788	492.20062860

14. To repeat this exercise, set `msTicks` in the Watch 1 window to 4 (anything but 3) and select RUN.
15. When finished, open the Breakpoints window and either use Kill All to delete the watchpoint or deselect it by unchecking it. Having undeleted Watchpoints activate unexpectedly can be rather confusing while debugging.
16. Leave Debug mode  for the next exercise.

TIP: You cannot set Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

TIP: To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint. You should delete the old one by highlighting it and click on Kill Selected or try the next TIP:

TIP: The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it.

J-Link does not currently display Data reads or writes in its trace window.

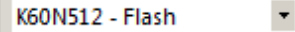
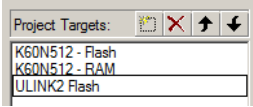
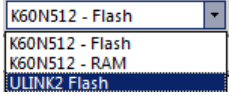






P&E OSJTAG does not currently support Watchpoints.

8) RTX_Blinky Example Program with Keil RTX RTOS: A Stepper Motor example

Keil provides RTX, a full feature RTOS is a component of MDK.

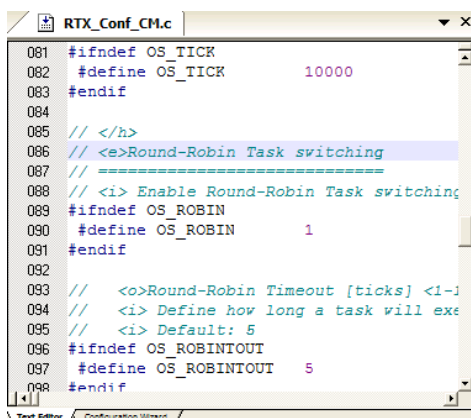
RTX now comes with a BSD license and is incorporated in MDK or is available from www.keil.com/demo/eval/rtx.htm.

RTX_Blinky uses a ULINK_{pro} as default. If you are using a ULINK2 or J-Link please configure it in steps 5 through 10. We will add a Target Option for the ULINK2. You could also just modify the existing target option.

1. Connect the ULINK2 as pictured on the first page. Use the special 10 to 20 pin cable for both the ULINK2 and ULINK-ME. See pages 5 for P&E. ULINK_{pro} is configured with this Blinky project. Connect it as on page 6.
2. With uVision in Edit mode Select Project/Open Project.
3. Open C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky\Blinky.uvproj.
4. Make sure "K60N512 Flash" is selected:  This is where you select different target configurations such as to execute a program in RAM or Flash. We will make a copy of this one and configure it to use a ULINK2.
5. In the main µVision menu, click on Project/Manage and select Components, Environment, Books...
6. Select the NEW icon or press the INSERT key on your keyboard.
7. In the space that appears: enter the name of your target option. I chose ULINK2 Flash: 
8. Click on OK to enter it and close this window.
9. Select your new Target Option as shown here:  →
10. At this point, you must select Options for Target  and select the adapter you are using. See instructions on previous pages to accomplish this. ULINK2 is on page 7. Remember to configure the Flash programmer too.
11. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
12. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
13. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
14. Three LEDs will blink indicating the four waveforms of a stepper motor driver changing. Click on STOP .

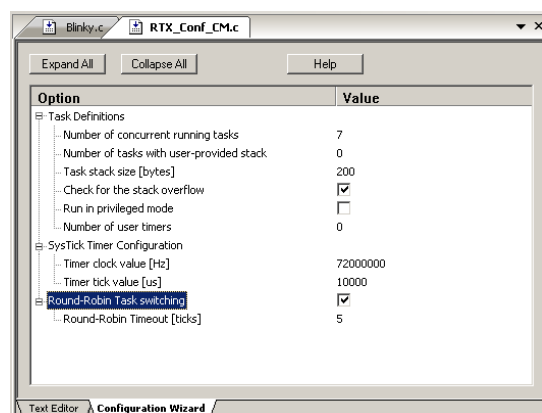
The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open.
2. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available. Do not change anything !
4. See www.keil.com/support/docs/2735.htm for instructions on how to add this feature to your own source code.



```
081 #ifndef OS_TICK
082 #define OS_TICK      10000
083 #endif
084
085 // </h>
086 // <e>Round-Robin Task switching
087 // =====
088 // <i> Enable Round-Robin Task switching
089 #ifndef OS_ROBIN
090 #define OS_ROBIN      1
091 #endif
092
093 // <o>Round-Robin Timeout [ticks] <1-1
094 // <i> Define how long a task will ex
095 // <i> Default: 5
096 #ifndef OS_ROBINTOUT
097 #define OS_ROBINTOUT  5
098 #endif
```


Text Editor: Source Code



Configuration Wizard




9) RTX Kernel Awareness using Serial Wire Viewer

Users 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 or memory area by the RTOS. Keil provides Task Aware windows for RTX. Other RTOS companies also provide awareness plug-ins for μ Vision.

1. Run RTX_Blinky again by clicking on the Run icon. 
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. Note these values are updated in real-time using the same technology as used in the Watch and Memory windows.
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.

RTX Viewer: Configuring Serial Wire Viewer (SWV):

We must activate Serial Wire Viewer to get the Event Viewer working. These steps are for ULINK2. For ULINKpro, see page 14.

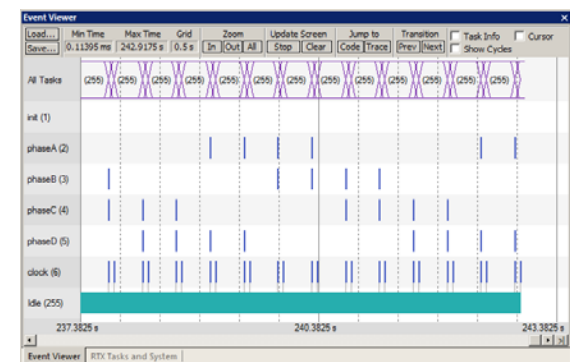
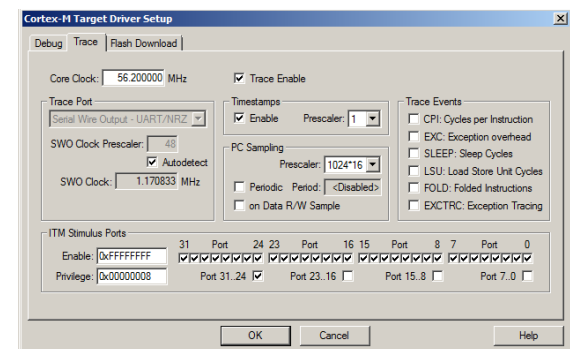
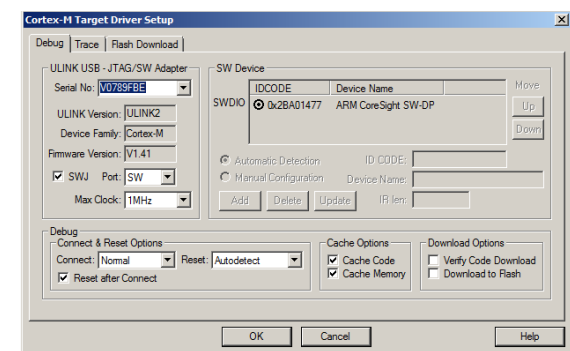
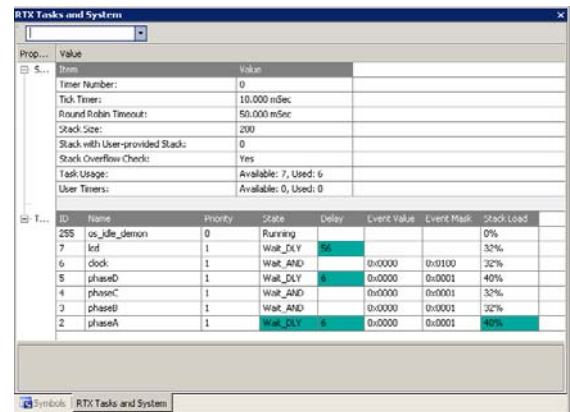
1. Stop the CPU and exit debug mode.  
2. Click on the Options icon  next to the target box.
3. Select the Debug tab and then click the Settings box next to ULINK2/ME Cortex Debugger dialog.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 56.2 MHz and select Trace Enable. For MDK after 4.71a, set this to 96 MHz. With a ULINK2 or ULINK-ME, this must be set exactly to the CPU frequency.
7. Unselect the Periodic and EXCTRC boxes as shown here:
8. ITM Stimulus Port 31 must be checked. This is the port used to output the kernel awareness information to the Event Viewer. ITM is slightly intrusive.
9. Click on OK twice to return to the μ Vision main menu.
The Serial Wire Viewer is now configured in μ Vision.
10. Enter Debug mode and click on RUN to start the program.
11. Select “Tasks and System” tab: note the display is updated.
12. Click on the Event Viewer tab.
13. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 0.5 seconds by clicking on the ALL and then the + and – icons.

TIP: View/Periodic Window Update must be selected !

TIP: If Event Viewer doesn't work, open up the Trace Records and confirm there is good ITM 31 frames present. The most probably cause for no valid ITM frames is the Core Clock: is not set correctly

Cortex-M4 Alert: μ Vision will update all RTX information in real-time on a target board due to its read/write capabilities as already described. The Event Viewer uses ITM and is slightly intrusive.

You do not have to stop the program to view this data. Your program runs at nearly full speed. No instrumentation code need be inserted into your source. You will find this feature very useful to see if RTX is behaving as you expect it to !








10) 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 Kinetis. 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 Task1 through Task4 in Blinky.c as shown below: **phasea=1;** and **phasea=0;** :the first two lines are shown added at lines 081 and 084 (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 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. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

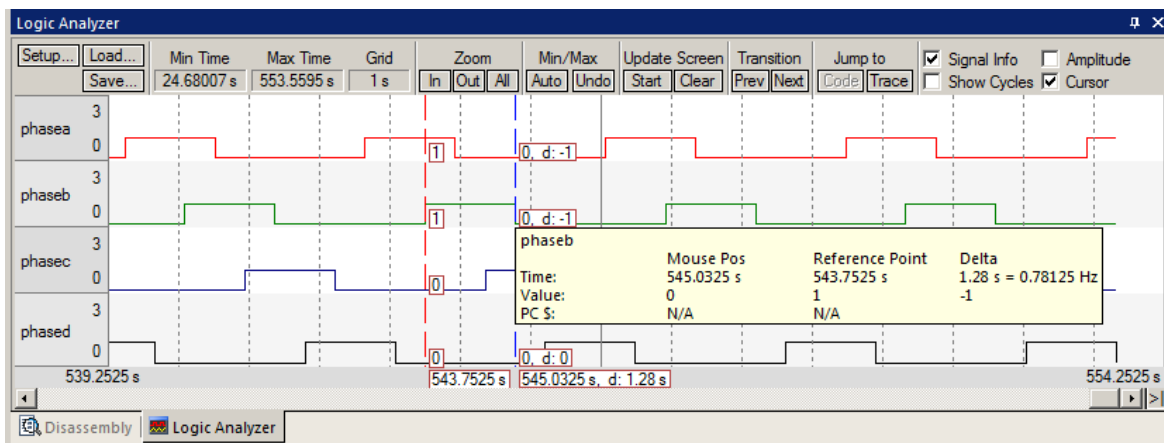
```
028 #define LED_D 0
029 #define LED_CLK LED_1
030
031 unsigned int phasea;
032 unsigned int phaseb;
033 unsigned int phasec;
034 unsigned int phased;
035
036 /*-----
037 * Function 'signal_fu
038 */
```

5. Rebuild the project.  Program the Flash .
6. Enter debug mode .
7. You can run the program at this point. .
8. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. .

```
074 /*-----
075 * Task 1 'phaseA': Phase A output
076 *-----
077 task void phaseA (void) {
078     for (;;) {
079         os_evt_wait_and (0x0001, 0xffff); /*
080         LED_On (LED_A);
081         phasea = 1;
082         signal_func (t_phaseB); /*
083         LED_Off(LED_A);
084         phasea=0;
085     }
086 }
```

Enter the Variables into the Logic Analyzer:

9. Right-click on the variable **phasea** and select Add phasea to... and then select Logic Analyzer. The LA will open.
10. Repeat for **phaseb**, **phasec** and **phased**. These variables will be listed on the left side of the LA window as shown. You might have to adjust the LA margins. Now we have to adjust the scaling.
11. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
12. Click on Close to go back to the LA window.
13. Using the OUT and In buttons set the range to 1 or 2 seconds. Move the scrolling bar to the far right if needed.
14. You will see the following waveforms appear. Click on STOP in the Update Screen box. The program will keep running. Select Signal Info and Cursor.
15. Click to mark a place See 252 s below. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phaseb: Note each LED is on for about 1.28 seconds.
16. Stop the program and exit Debug mode when you are finished.



TIP: You can also enter these variables into the Watch and Memory windows to display and modify them in real-time.


11) Serial Wire Viewer (SWV) and how to use it:

a) Data Reads and Writes: (Note: Data Reads but not Writes are disabled in the current version of μ Vision).

You have configured Serial Wire Viewer (SWV) in Page 6 under **RTX Viewer: Configuring the Serial Wire Viewer:**

Now we will examine some of the features available to you. SWV works with μ Vision and a ULINK2, ULINK-ME, ULINKpro or a Segger J-Link V6 or higher. SWV is included with MDK and no other components must be purchased.


Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. A user program runs at full speed and needs no code stubs or instrumentation software added to your programs.

1. Use RTX_Blinky from the last exercise. Enter Debug mode and run the program if not already running.
2. Select View/Trace/Records or click on the arrow beside the Trace icon  and select Records.
3. The Trace Records window will open up as shown here:
4. The ITM frames are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num.
5. To turn this off select Debug/Debug Settings and click on the Trace tab. Unselect ITM Stimulus Port 31. **TIP:** Port 0 is used for Debug printf Viewer.
6. Unselect EXCTRC and Periodic.
7. Select On Data R/W Sample.
8. Click on OK twice to return.
9. Click on the RUN icon.
10. Double-click anywhere in the Trace records window to clear it.
11. Only Data Writes will appear now.
TIP: You could also have right clicked on the Trace Records window to filter the ITM frames out.

What is happening here ?

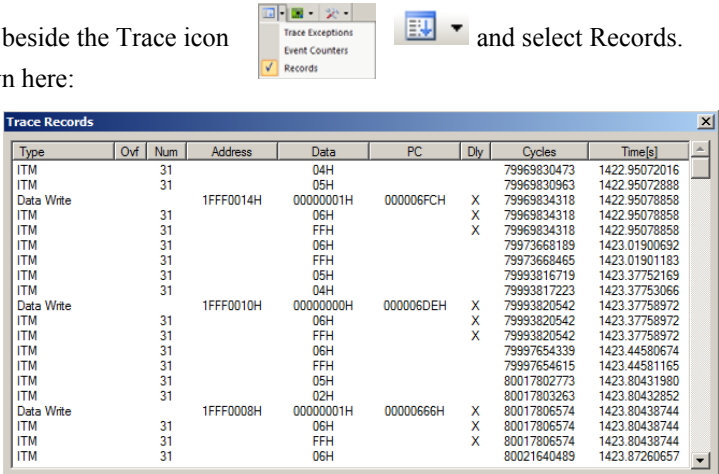
1. When variables are entered in the Logic Analyzer (remember phasea through phased ?), the reads and/or writes will also appear in Trace Records.
2. The Address column shows where the four variables are located.
3. The Data column displays the data values written to phasea through phased.
4. PC is the address of the instruction causing the writes. You activated it by selecting On Data R/W Sample in the Trace configuration window.
5. The Cycles and Time(s) columns are when these events happened.

TIP: You can have up to four variables in the Logic Analyzer and subsequently displayed in the Trace Records window. Remember these are shared by the Watchpoints.

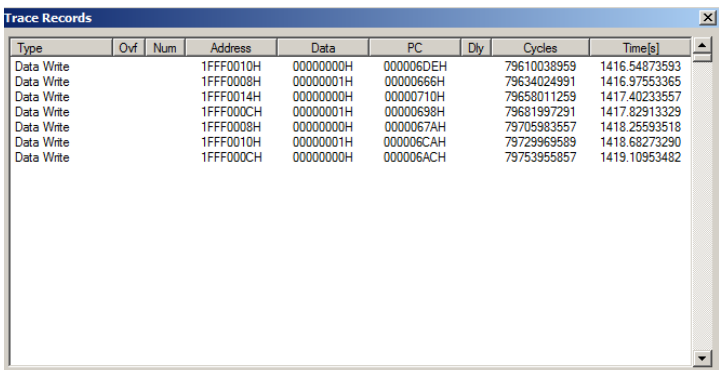
TIP: If you select View/Symbol Window you can see where the addresses of the variables are. 

Note: You must have Browser Information selected in the Options for Target/Output tab to see the Symbol Browser.

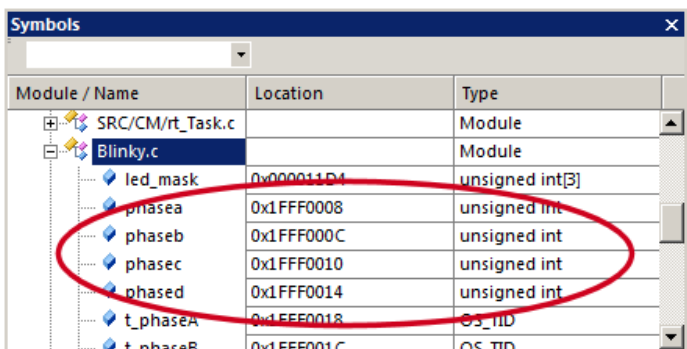
TIP: ULINKpro, and J-Link adapters display the trace frames in a different style trace window. You must stop the program to view their trace windows.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		04H			79969830473	1422.95072016
ITM		31		05H			79969830963	1422.95072888
Data Write			1FFF0014H	00000001H	000006FCH	X	79969834318	1422.95078858
ITM		31		06H		X	79969834318	1422.95078858
ITM		31		FFH		X	79969834318	1422.95078858
ITM		31		06H			79973668189	1423.01900692
ITM		31		FFH			79973668465	1423.01901183
ITM		31		05H			79993816719	1423.37752169
ITM		31		04H			79993817223	1423.37753066
Data Write			1FFF0010H	00000000H	000006DEH	X	79993820542	1423.37758972
ITM		31		06H		X	79993820542	1423.37758972
ITM		31		FFH		X	79993820542	1423.37758972
ITM		31		06H			79997654339	1423.44580674
ITM		31		FFH			79997654615	1423.44581165
ITM		31		05H			80017802773	1423.80431980
ITM		31		02H			80017803263	1423.80432852
Data Write			1FFF0008H	00000001H	00000666H	X	80017806574	1423.80438744
ITM		31		06H		X	80017806574	1423.80438744
ITM		31		FFH		X	80017806574	1423.80438744
ITM		31		06H			80021640489	1423.87260657



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			1FFF0010H	00000000H	000006DEH		79610038959	1416.54873593
Data Write			1FFF0008H	00000001H	00000666H		79634024991	1416.97553365
Data Write			1FFF0014H	00000000H	00000710H		79658011259	1417.40233557
Data Write			1FFF000CH	00000001H	00000698H		79681997291	1417.82913329
Data Write			1FFF0008H	00000000H	0000067AH		79705983557	1418.25593518
Data Write			1FFF0010H	00000001H	000006CAH		79729969589	1418.68273290
Data Write			1FFF000CH	00000000H	000006ACH		79753955857	1419.10953482

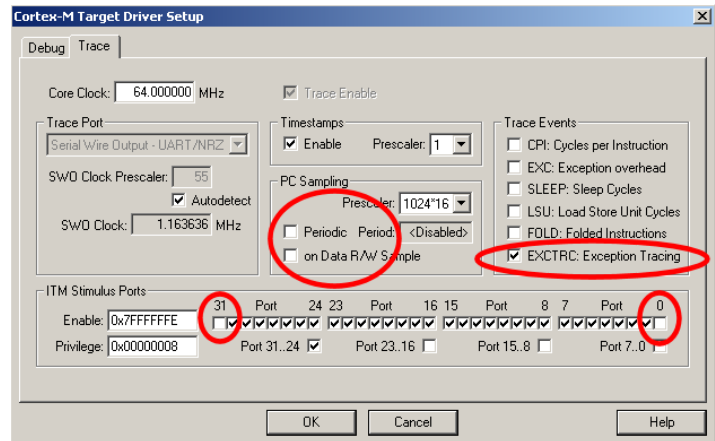


Module / Name	Location	Type
SRC/CM/rt_Task.c		Module
Blinky.c		Module
led_mask	0x000011D4	unsigned int[3]
phasea	0x1FFF0008	unsigned int
phaseb	0x1FFF000C	unsigned int
phasec	0x1FFF0010	unsigned int
phased	0x1FFF0014	unsigned int
t_phaseA	0x1FFF0018	OS_TID
t_phaseB	0x1FFF001C	OS_TID

b) Exceptions and Interrupts:

The Kinetis family using the Cortex-M4 processor has many interrupts and it can be difficult to determine when and how often they are being activated. SWV on the Cortex-M4 processor makes this easy.

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample and ITM Ports 31 and 0. (this is to minimize overloading the SWO port)
3. Select EXCTRC as shown here:
4. Click OK twice.
5. The Trace Records should still be open. Double click on it to clear it.
6. Click RUN to start the program.
7. You will see a window similar to the one below with Exceptions frames displayed.



What Is Happening ?

1. You can see two exceptions happening.
 - **Entry:** when the exception enters.
 - **Exit:** When it exits or returns.
 - **Return:** When all the exceptions have returned and is useful to detect tail-chaining.
2. Num 11 is SVCcall from the RTX calls.
3. Num 15 is the SysTick timer.
4. In my example you can see one data write from the Logic Analyzer.
5. Note everything is timestamped.
6. The “X” in Ovf is an overflow and some data was lost. The “X” in Dly means the timestamps are delayed because too much information is being fed out the SWO pin.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					1163020698	18.17219841
Exception Exit		15					1163020952	18.17220238
Exception Return		0				X	1163022877	18.17223245
Exception Entry		15					1163660698	18.18219841
Exception Exit		15					1163661343	18.18220848
Exception Return		0				X	1163664507	18.18225792
Exception Entry	X	11				X	1163664507	18.18225792
Exception Exit		11					1163674947	18.18242105
Exception Return		0				X	1163675072	18.18242300
Data Write			20000024H	00000000H		X	1163680402	18.18250628
Exception Return	X	0				X	1163680402	18.18250628
Exception Entry		11					1163688581	18.18263408
Exception Exit		11					1163688706	18.18263603
Exception Return		0				X	1163691952	18.18268675
Exception Return	X	0				X	1163691952	18.18268675
Exception Entry		15					1164300698	18.19219841
Exception Exit		15					1164300958	18.19220247
Exception Return		0				X	1164302892	18.19223269
Exception Return		15					1164940698	18.20219841

TIP: The SWO pin is one pin on the Cortex-M4 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate. ULINKpro has the option of sending this data out the 4 bit Trace Port or the SWO pin. ULINKpro handles SWV data faster than the ULINK2.

1. Select View/Trace/Exceptions or click on the Trace icon and select Trace Exceptions.
2. The next window opens up and more information about the exceptions is displayed as shown. Click on Num.
3. Note the number of times these have happened. This is very useful information in case interrupts come too fast or slow.
4. Note the peripheral exceptions are listed by name.

5. You can clear this trace window by clicking on the clear icon.
6. All this information is displayed in real-time and without stealing CPU cycles !

N	Name	Count	Total Time	Min Time In	Max Time...	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
1	Reset	0	0 s						
2	NonMaskableInt	0	0 s						
3	HardFault	0	0 s						
4	MemoryManage...	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCcall	158	0 s					56.76458557	90.12321731
12	DebugMonitor	0	0 s						
14	PendSV	0	0 s						
15	SysTick	3928	15.487 ms	3.523 us	9.288 us	8.527 ms	8.532 ms	56.70473158	90.22558959
19	INT_Hard_Fault	0	0 s						
21	INT_Bus_Fault	0	0 s						
22	INT_Usage_Fault	0	0 s						

TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 - 16 = ExtIRQ 0.

c) PC Samples:

Serial Wire Viewer can display a sampling of the program counter. If you need to see all the PC values, use the ETM trace with a Keil ULINKpro. ETM trace also provides Code Coverage, Execution Profiling and Performance Analysis.

SWV can display at best every 64th instruction but usually every 16,384 is more common. It is best to keep this number as high as possible to avoid overloading the Serial Wire Output (SWO) pin. This is easily set in the Trace configuration.

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample. Select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear it.
5. Click on RUN and this window opens:
6. Most of the PC Samples are 0x0040_05E2 which is a branch to itself in a loop forever routine.
7. Stop the program and the Disassembly window will show this Branch:
8. Not all the PCs will be captured. Still, PC Samples can give you some idea of where your program is; especially if it is caught in a tight loop like in this case.
9. **Note:** you can get different PC values displayed here depending on the optimization level set in μ Vision.
10. Set a breakpoint in one of the tasks.
11. Run the program and when the breakpoint is hit, you might see another address at the bottom of the Trace Records window. See the screen below:
12. Scroll to the bottom of the Trace Records window and you might see the correct PC value displayed. Usually, it will be a different PC depending on when the sampling took place.
13. Remove the breakpoint and exit Debug mode for the next step.

TIP: ULINKpro gives much more ! All executed instructions are captured and the Trace data window is much more sophisticated.

The screenshot displays the Keil IDE interface. The Disassembly window shows the following assembly code:

```
116: /*-----
117: *      Task 4 'phased': Phase D output
118: *-----*/
119: __task void phased (void) {
0x004007C2 2000  MOVs    r0,#0x00
0x004007C4 4978  LDR     r1,[pc,#480] ; 0x004009A8
0x004007C6 6008  STR     r0,[r1,#0x00]
0x004007C8 E7E7  B       0x0040079A
```

The Trace Records window is overlaid on the Disassembly window, showing a table of trace data:


Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					004005E2H		86035838109	1344.30997045
PC Sample					004005E2H		86035854493	1344.31022645
PC Sample					004005E2H		86035870877	1344.31048245
PC Sample					004005E2H		86035887261	1344.31073845
PC Sample					004005E2H		86035903645	1344.31099445
PC Sample					004005E2H		86035920029	1344.31125045
PC Sample					004005E2H		86035936413	1344.31150645
PC Sample					004005E2H		86035952797	1344.31176245
PC Sample					004005E2H		86035969181	1344.31201845
PC Sample					004005E2H		86035985565	1344.31227445
PC Sample					004005E2H		86036001949	1344.31253045
PC Sample					004005E2H		86036018333	1344.31278645
PC Sample					004005E2H		86036034717	1344.31304245
PC Sample					004005E2H		86036051101	1344.31329845
PC Sample					004005E2H		86036067485	1344.31355445
PC Sample					004005E2H		86036083869	1344.31381045
PC Sample					004005E2H		86036100253	1344.31406645
PC Sample					004005E2H		86036116637	1344.31432245
PC Sample					004005E2H		86036133021	1344.31457845
PC Sample					004005E2H		86036149405	1344.31483445

12) ITM (Instrumentation Trace Macrocell) printf without using a UART:

Recall that we showed how you can display information about the RTOS in real-time using the RTX Viewer. This is done through ITM Stimulus Port 31. ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into μ Vision for display in the Debug (printf) Viewer window.

1. Open the project Blinky.uvproj (not RTX Blinky).
2. Add this code to Blinky.c. A good place is near line 16.

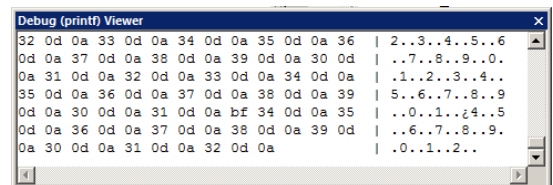
```
unsigned int count = 0;
#define ITM_Port8(n)  (*((volatile unsigned char *) (0xE0000000+4*n)))
```
3. In the main function in Blinky.c right after the second Delay(250) near Line 95, enter these lines:

```
count++;
if (count >9) count=0;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = count + 0x30;    /* displays count value: +0x30 converts to ASCII */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```
4. Rebuild the source files, program the Flash memory and enter debug mode.
5. Open Debug/Debug Settings and select the Trace tab.
6. Unselect On Data R/W Sample, PC Sample and ITM Port 31. (this is to help not overload the SWO port)
7. Select EXCTRC and ITM Port 0. ITM Stimulus Port “0” enables the Debug (printf) Viewer.
8. Click OK twice.
9. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
10. In the Debug (printf) Viewer you will see the ASCII value of count appear. 
11. Right click on the Debug window and select Mixed Hex ASCII mode. Note the other settings.



Trace Records

1. Open the Trace Records if not already open. Double click in it to clear it.
2. You will see a window such as the one below displaying both ITM and Exception frames.



What Is This ?

1. You can see Exception 15 Entry, Exit, Return and the three ITM writes. You probably have to scroll down.
2. ITM 0 frames (Num column) are our ASCII characters from `count` with carriage return (0D) and line feed (0A) as displayed the Data column.
3. All these are timestamped in both CPU cycles and time in seconds.
4. Note the “X” in the Dly column. This means the timestamps might not be correct due to SWO pin overload.
5. Right click in the Trace Records window and deselect Exceptions to see only ITM writes.

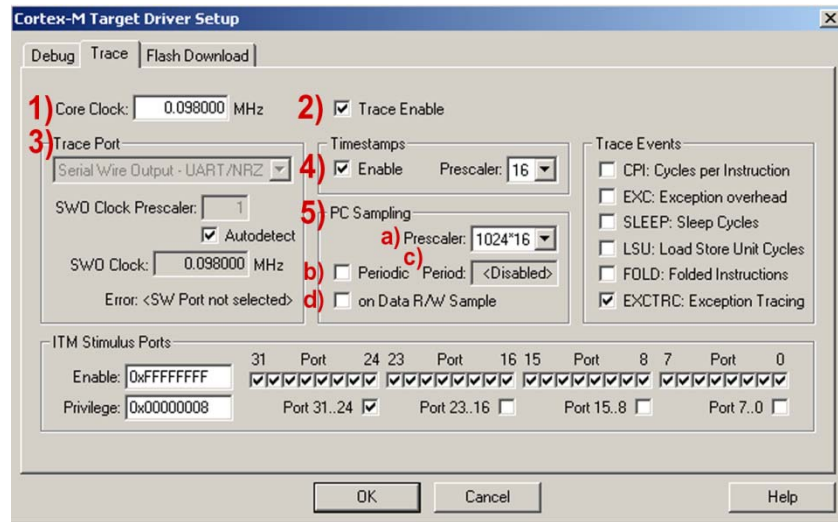
Type	Dly	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Return	0					X	2047810917	31.99704558
Exception Entry	15						2047873261	31.99801970
Exception Exit	15						2047873275	31.99801992
Exception Return	0					X	2047874882	31.99804503
Exception Entry	15						2047937261	31.99901970
Exception Exit	15						2047937275	31.99901992
Exception Return	0					X	2047938902	31.99904534
Exception Entry	15						2048001261	32.00001970
Exception Exit	15						2048001275	32.00001992
Exception Return	0					X	2048006167	32.00009636
ITM	0		34H			X	2048006167	32.00009636
ITM	0		0DH			X	2048006167	32.00009636
ITM	0		0AH			X	2048006167	32.00009636
Exception Entry	15						2048065261	32.00101970
Exception Exit	15						2048065275	32.00101992
Exception Return	0					X	2048066887	32.00104511
Exception Entry	15						2048129261	32.00201970
Exception Exit	15						2048129275	32.00201992
Exception Return	0					X	2048130907	32.00204542
Exception Entry	15						2048193261	32.00301970

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enter only those features that you need.

Super TIP: ITM_SendChar is a useful function you can use to send characters out ITM. It is found in core.CM3.h. To send ITM data to your Windows application, see www.keil.com/appnotes/docs/apnt_240.asp.

13) 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}.



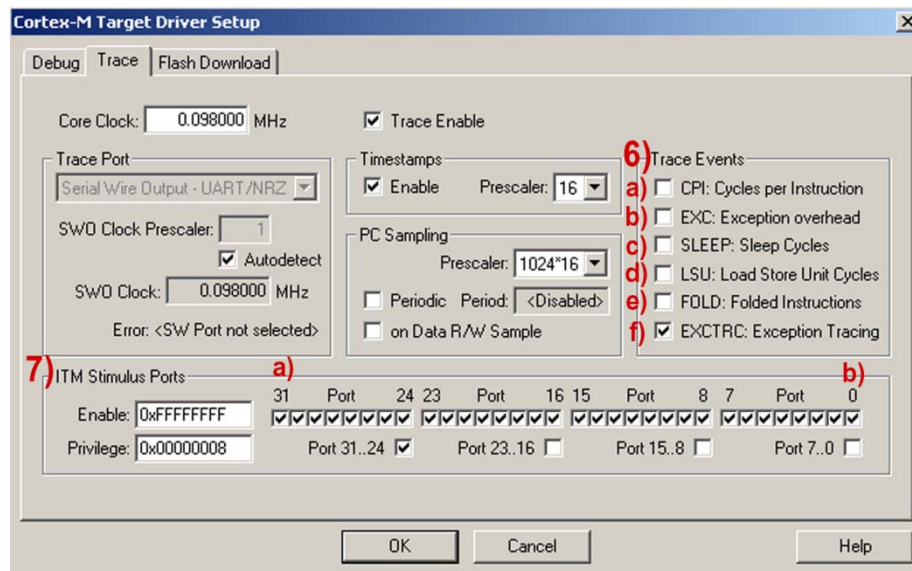
Trace Configuration window Fields 1) through 5).

- 1) **Core Clock:** The CPU clock speed for SWV. SWO Clock signal is derived from and is a ratio of the Core Clock. CPU frequency is set by the systemxxx.c file. It is either 56.2 or 96 MHz depending on the version of this file used.
- 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} or J-link.

TIP: It is important to ensure the Serial Wire Output (SWO) pin is not overloaded. μ Vision 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. μ Vision 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 better 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 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 23 for information on using this feature.

PART C): DSP Example using ARM CMSIS-DSP Libraries:

1) DSP SINE example:

ARM CMSIS-DSP libraries are offered for ARM Cortex-M0, 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.

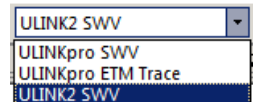
You can use this example with other Kinetis boards. You might need to make some changes to the startup and system files.

This example creates a *sine* wave, then creates 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. Source code is provided.



To obtain this DSP example project, go to www.keil.com/appnotes/docs/apnt_239.asp

1. Extract DSP.zip to C:\Keil\ARM\Boards\Freescale\TWR-K60N512\ to create the folder \DSP.
1. Open the project file sine.uvproj with µVision. Connect a ULINK2 or ULINKpro to the K60 board.
2. If you are using a ULINK2 or ME, select ULINK2 SWV from the target drop menu:
If using a ULINKpro, select either ULINKpro SWV to send SWV data out the SWO pin or ULINKpro ETM Trace to send it out the 4 bit Trace port. Either works with this example.



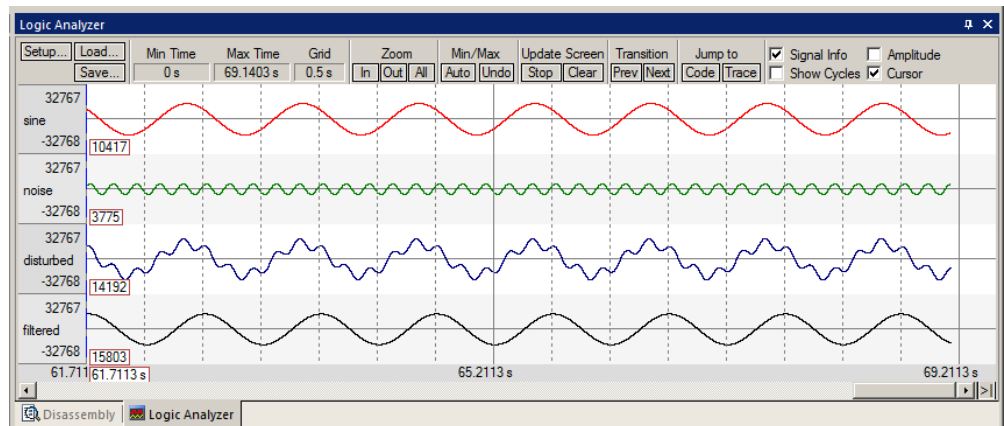
3. Compile the source files by clicking on the Rebuild icon.
4. Program the K60 flash by clicking on the Load icon:  Progress is indicated in bottom left corner.
5. Enter Debug mode by clicking on the Debug icon:  Select OK if the Evaluation Mode notice appears.

TIP: The default Core Clock: is 56.2 MHz for use by the Serial Wire Viewer configuration window in the Trace tab.

6. Click on the RUN icon:  Open the Logic Analyzer window: .
7. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom 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 if you are using a ULINK2. ULINKpro handles SWV data faster than a ULINK2 or J-Link can.

8. This project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.
9. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below: They are also pre-configured in this project.



10. Open the Trace Records window and the Data Writes to the four variables are displayed using Serial Wire Viewer. When you enter a variable in the LA, its data write is also displayed in the Trace window.
11. Open View/Serial Windows/Debug (printf) Viewer. printf data is displayed from printf statements in DirtyFilter.c near lines 174 through 192 using ITM using SWV.

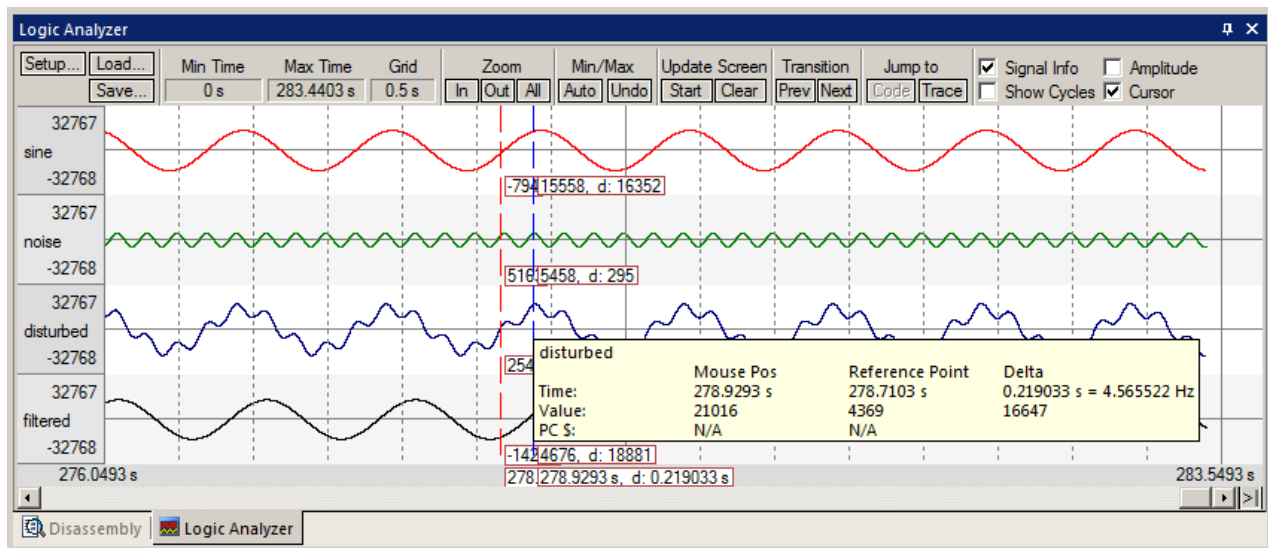
TIP: The ULINKpro trace display is different and the program must currently be stopped to update it.

12. Leave the program running.
13. Close the Trace Records window.

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

2) Signal Timings in Logic Analyzer (LA):

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



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. This window does not change much: nearly all the processor time is spent in the idle daemon: it shows as 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 each of the four tasks in DirtyFilter.c by clicking in the left margin on a grey area. Do select while(1) as this will not stop the program.
9. Click on Run and the program will stop here and the Task window will be updated accordingly. In the screen below, the program stopped in the noise_gen task:
10. Clearly you can see that noise_gen was running when the breakpoint was activated.
11. Each time you click on RUN, the next task will display as Running.
12. Remove all the breakpoints. You can use Ctrl-B and select Kill All.

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

System



Tasks

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

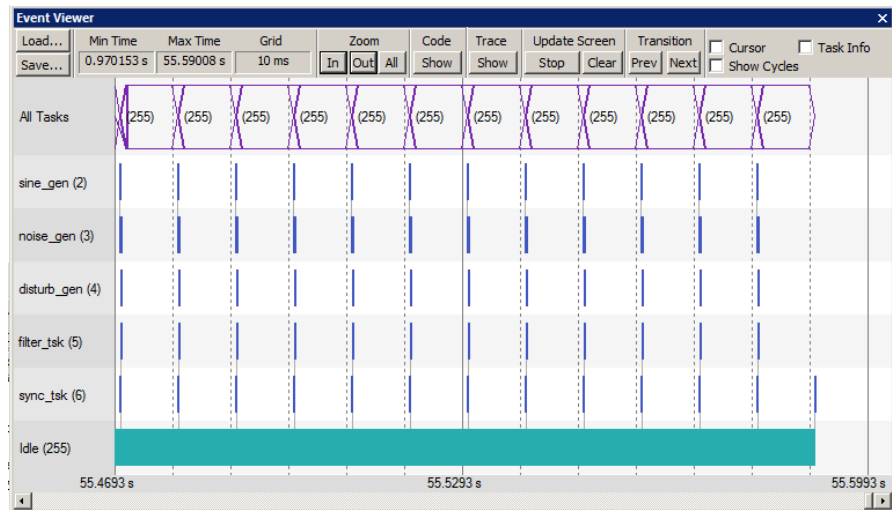
ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Ready				32%
6	sync_tsk	1	Wait_DLY	1			32%
5	filter_tsk	1	Wait_AND		0x0000	0x0001	32%
4	disturb_gen	1	Wait_AND		0x0000	0x0001	32%
3	noise_gen	1	Running		0x0000	0x0001	0%
2	sine_gen	1	Wait_AND		0x0000	0x0001	32%

3) RTX Event Viewer:

1. **If you are using a ULINKpro, skip this step:** Stop the program. Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables and select Close. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might occur. You can also leave the LA loaded with the four variables to see what the Event Viewer will look like. Or you can remove just one or two. Later, delete them to see the effect 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 195.

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.

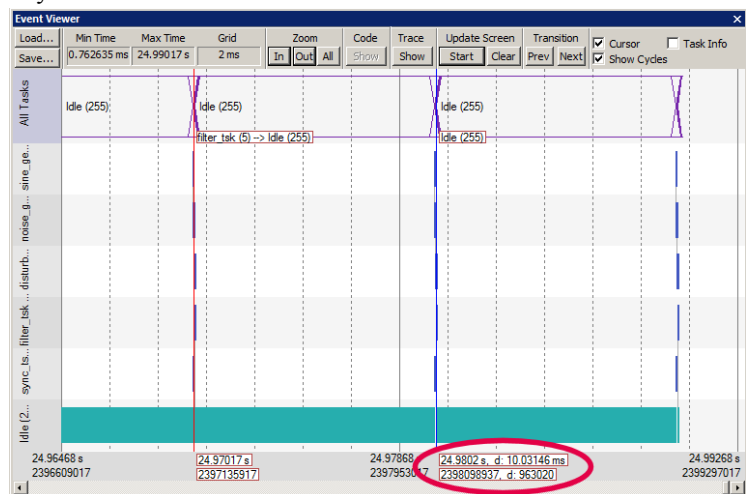
ULINKpro can also use the 4 bit Trace Port for even faster operation for SWV. Trace Port use is mandatory for ETM trace.

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 as shown below:
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: 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. This is normally a good idea if you are running RTX with a high SWO data rate.

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

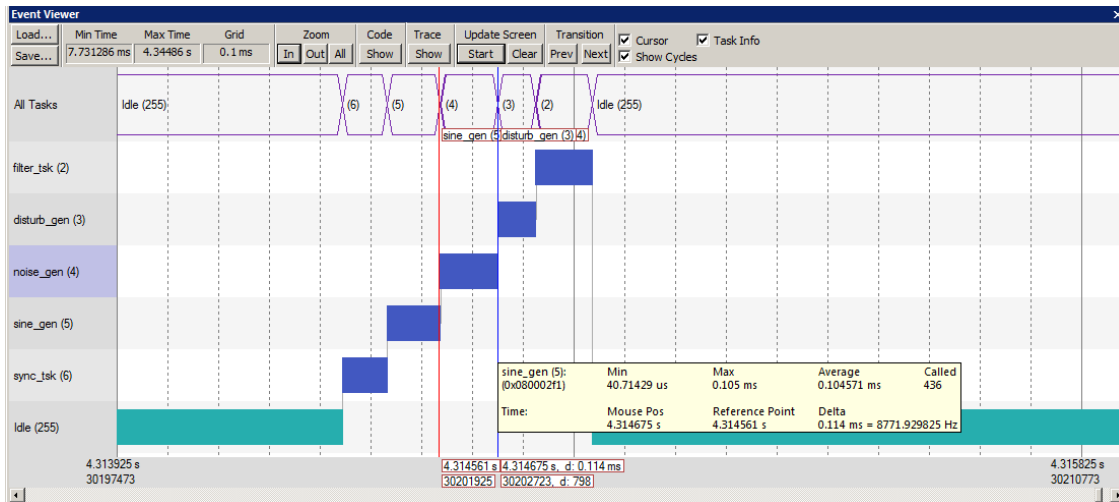


4) Event Viewer Timing:



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 all 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.





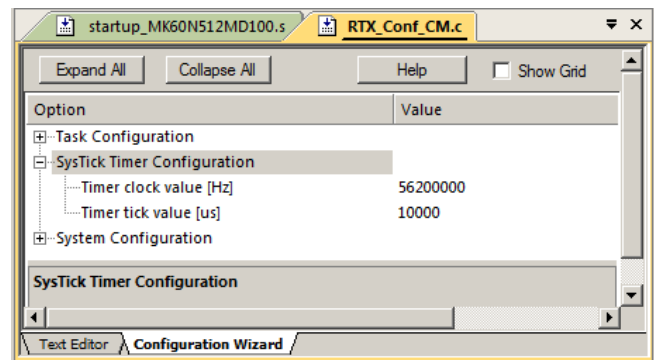
Changing the SysTick Timer:

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 and select it in C:\Keil\ARM\Boards\Freescale\TWR-K60N512\DSP.
3. Select the Configuration Wizard tab at the bottom of the window. This scripting language is shown in the Text Editor as comments starting such as a </h> or <i>. See www.keil.com/support/docs/2735.htm for instructions.
4. This window opens up. Expand the SysTick Timer Configuration as shown here:
5. Note the Timer tick value is 10,000 μ s or 10 ms.
6. Change this value to 20,000.

TIP: The 56,200,000 is the CPU speed and is correct for this DSP example.

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. You can use SysTick for other purposes.

9. Set the SysTick timer back to 10,000. You will need to recompile the source files and reprogram the Flash.
10. Click on File/Save All.
11. Stop the processor and exit Debug mode to compile the source files.

Part D) ETM Trace with ULINKpro:

Introduction:

The examples shown previously with the ULINK2 will also work with the ULINKpro. There are two major differences:

- 1) The window containing the trace frames is now called Trace Data. More complete filtering is available.
- 2) The SWV (Serial Wire Viewer) data is sent out the 1 bit SWO pin with the ULINK2 using UART encoding. The ULINKpro can send SWV data *either* out this same SWO pin using Manchester encoding or through the 4 bit Trace Port. This allows ULINKpro to support those Cortex-M processors that have SWV but not ETM and have no Trace Port. The Trace Port is found on the 20 pin Cortex connector and is configured in the Trace configuration window. ETM frames are always sent out the Trace Port and if this is the case, SWV data is also be sent out this port.

ULINKpro offers:

- 1) Faster Flash programming than the ULINK2.
- 2) All the Serial Wire Viewer features as the ULINK2 provides.
- 3) Adds ETM Instruction Trace which provides a record of all executed instructions. Trace Data window has Trace start and stop, filtering and ability to save records to a file.
- 4) **Code Coverage:** were all the assembly instructions executed ?
- 5) **Performance Analysis:** where the processor spent its time.
- 6) **Execution Profiling:** How long instructions, ranges of instructions, functions or C source code took in both time and CPU cycles as well as number of times these were executed.

1) Configuring ULINKpro ETM Trace:

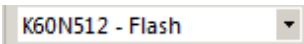

Configuring the Connection:

The ULINKpro was configured for SWV operation using the SWO pin and Manchester encoding on page 14. We will activate ETM trace in the next few pages. We will output the trace frames, including SWV, out the 4 bit Trace Port.

.ini File:

A script must be executed upon entering Debug mode to configure the ETM registers and GPIO ports. This script is provided as TracePort.ini and is found in C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky. This is an ASCII file. This file is specific to Kinetis processors as their proprietary GPIO ports must be configured.

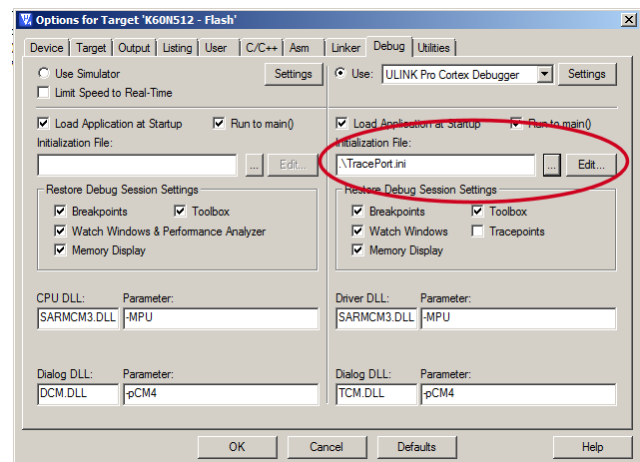
Entering the Initialization File: *Note:* Blinky is pre-configured for ULINKpro. We need to modify it slightly.

- 1) Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky\Blinky.uvproj.
- 2) Select "K60N512 Flash": 
- 3) Click on the Target Options icon  or select "Project/Options for Target" or by pressing Alt+F7.
- 4) Click on the Debug tab. We will enter TracePort.ini in the Initialization file: box and shown below:
- 5) You can type it in or use the browse icon to select it from C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky\
- 6) If you click on the Edit icon, TracePort.ini will be opened with the other source files. You can then view and edit it.
- 7) Leave this window open for the next page.
- 8) Select File/Save All.

TIP: This ini file will be executed every time you enter Debug mode. In the case of this file, a µVision RESET will run it again because of the function OnResetExec. See www.keil.com/support/man/docs/uv4/uv4_db_trace_init.htm for more information.

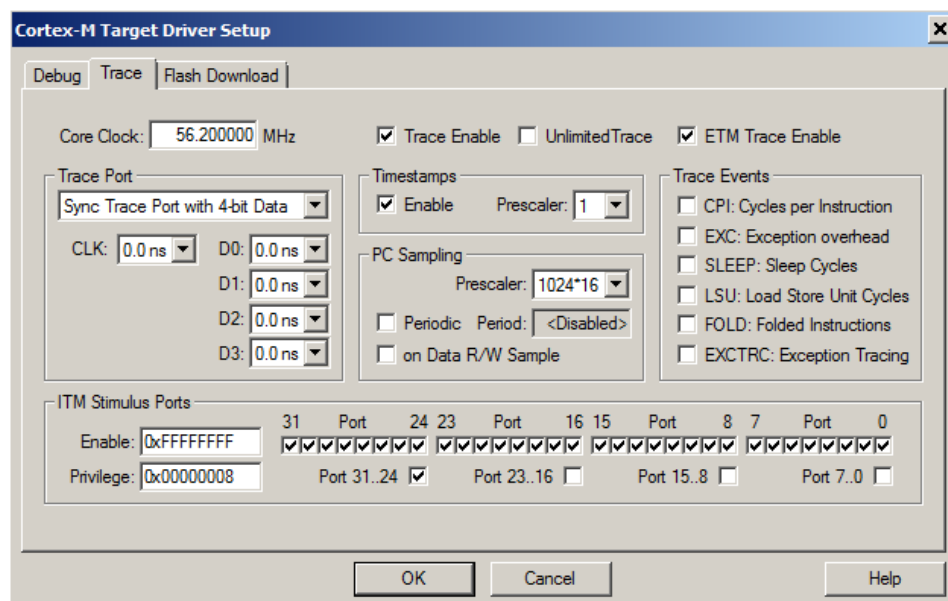
TIP: If you are adapting this lab for other Kinetis boards, you may have to configure µVision for operation with a ULINKpro. See page 8 for directions.

The next page describes how to configure ETM.



Configuring ETM Trace (and SWV):

- 1) These instructions assume you are continuing from the previous page.
- 2) In the Target Options window left open from the previous page, click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side of the window.
- 3) Click on the Trace tab. The window below is displayed.
- 4) Core Clock: ULINK_{pro} calculates this from the Trace Port. μ Vision uses this to display timing values so it is still a good idea to insert a valid clock speed. 56.2 for MDK 4.71a or 96 MHz for later versions.
- 5) In Trace Port select Sync Trace Port with 4 bit data. It is best to use the widest size.
- 6) Select CLK: 0.0 ns clock delay. D0 through D3 should be set to zero also.
- 7) Select Trace Enable and ETM Trace Enable. Unselect Periodic and EXCTRC and leave everything else at default as shown below. Only ITM 31 and 0 need to be selected. The others are Don't Care.
- 8) Click on OK twice to return to the main μ Vision menu. Both ETM and SWV are now configured through the 4 bit Trace Port.
- 9) Select File/Save All.



TIP: We said previously that you must use SWD (also called SW) in order to use the Serial Wire Viewer. With the ULINK_{pro} and with the Trace Port selected, you can also select the JTAG port as well as the SWD port.

With ULINK2, ULINK-ME and Segger J-Link, you must select SW. There is a conflict using JTAG signal TDO and the SWO signal.




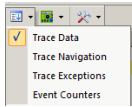

ULINK_{pro} can send the SWV signals out the 4 bit Trace Port which does not share pins with JTAG. It has its own dedicated pins on the Kinetis processor.

Be aware these pins are usually multiplexed with GPIO pins or other peripherals. You should make appropriate allowances for the use of these shared ports during debugging with ETM trace.

It is good engineering practice during system design to not use those pins shared with ETM for important purposes that will preclude normal operation with ETM enabled.

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

The project in C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky has now been modified on the previous page to provide ETM Trace and all the features it provides using a ULINKpro.

1. Compile the Blinky source files by clicking on the Rebuild icon.  You can also use the Build icon beside it.
2. Program the Kinetis flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
3. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
4. DO NOT CLICK ON RUN YET !!! If you did, simply exit and re-enter Debug mode.
5. Open the Trace Data window by clicking on the small arrow beside the Trace icon as shown here: 
6. Examine the Trace Data window as shown below: This is a complete record of all the program flow since RESET until μ Vision halted the program at the start of main() since Run To main is selected in μ Vision.
7. In this case, Time 014 306 shows the last instruction to be executed. (BL.W). In the Register window the PC will display the value of the next instruction to be executed (0x000 02C4 in my case). Click on Single Step once. 

Trace Data			
Display:	All		
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x000005D0	MOV r8,#0x00	
	X: 0x000005D4	MOV r11,#0x00	
	X: 0x000005D8	BIC r1,r1,#0x07	
	X: 0x000005DC	MOV r12,r5	
	X: 0x000005DE	STM r12!,[r6-r8,r11]	
	X: 0x000005E2	STM r12!,[r6-r8,r11]	
	X: 0x000005E6	STM r12!,[r6-r8,r11]	
	X: 0x000005EA	STM r12!,[r6-r8,r11]	
	X: 0x000005EE	MOV sp,r1	
0.000 013 879 s	X: 0x000005F0	BX lr	
	X: 0x00000260	MOV r1,r2	
	X: 0x00000262	BL.W __rt_lib_init (0x00000254)	
	X: 0x00000254	PUSH {r0-r4,lr}	
0.000 014 235 s	X: 0x00000256	POP {r0-r4,pc}	
0.000 014 306 s	X: 0x00000266	BL.W main (0x000002C4)	

8. The instruction MOV will display at 014 413:
9. Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is a LDR as shown below:
10. If you use the Memory window to look at location 0x4, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0000 027C + 1 = 0x027D (+1 says it is a Thumb instruction). These first instructions after RESET are shown below: Note the source information that is displayed: If you double-click on any line, this will be highlighted in both the Disassembly and source windows.

0.000 014 306 s	X: 0x00000266	BL.W main (0x000002C4)	
		TRACE RUN	
0.000 014 413 s	X: 0x000002C4	MOV r4,#0xFFFFFFFF	int num = -1;

Trace Data			
Display:	All		
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
		TRACE RUN	
	X: 0x0000027C	LDR r0,[pc,#36] ; @0x000002A4	LDR R0,=SystemInit
0.000 000 214 s	X: 0x0000027E	BLX r0	BLX R0
	X: 0x00000410	MOVW r0,#0xC520	WDOG->UNLOCK = (uint16_t)0xC520u; /* Key 1 */
	X: 0x00000414	LDR r1,[pc,#380] ; @0x00000594	
	X: 0x00000416	STRH r0,[r1,#0x0E]	
	X: 0x00000418	MOVW r0,#0xD928	WDOG->UNLOCK = (uint16_t)0xD928u; /* Key 2 */

TIP: If you unselect Run to main() in the Debug tab, no instructions will be executed when you enter Debug mode. The PC will equal 0x027C. You can run or single-step from that point and this will be recorded in the Trace Data window.

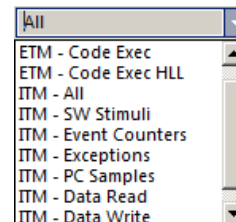
ETM trace provides a powerful tool for finding nasty bugs not easily found any other way. See page 40 for problems ETM and Serial Wire Viewer can help solve.

3) Finding the Trace Frames you are looking for:

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

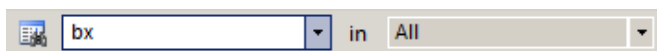
Trace Filters:

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




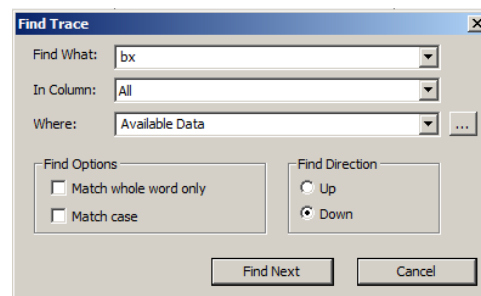
Find a Trace Record:

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



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

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



Trace Triggering:

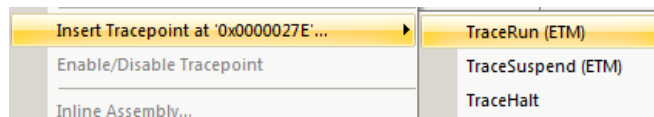
μ Vision has three trace triggers currently implemented:

TraceRun: Starts ETM trace collection when encountered.

TraceSuspend: Stops ETM trace collection when encountered. TraceRun has to have been encountered for this to have an effect.

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


TraceHalt: Stops ETM trace, SWV and ITM. Can be resumed only with a STOP/RUN sequence. TraceStart will not restart this.



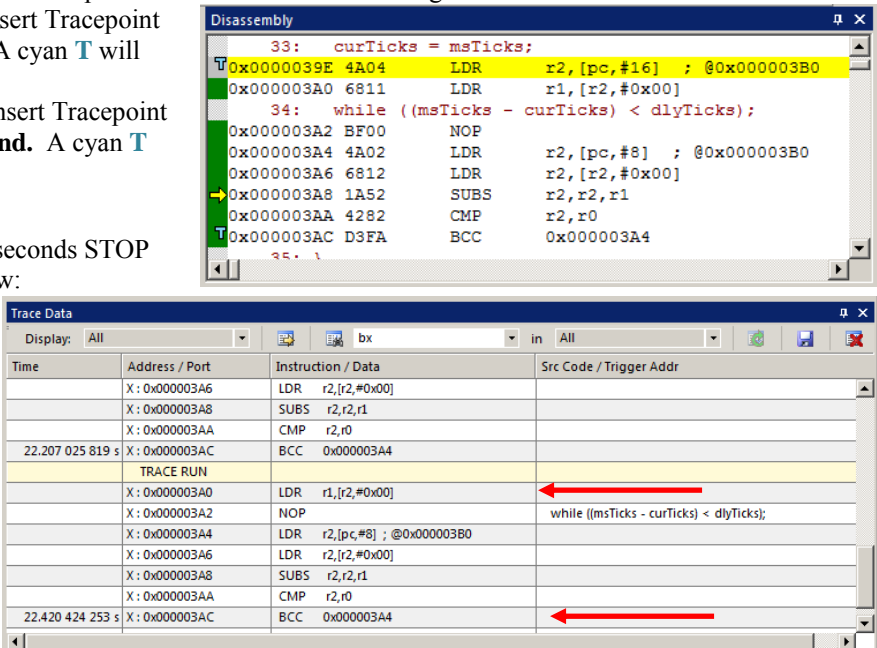
How it works:

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

4) Setting Some Trace Triggers:

1. With Blinky in Debug mode, click in the Blinky.c on `curTicks = msTicks;` near line 33. This will be highlighted in the Disassembly window. This window will open: Position so lines 33: through 35: are visible.
2. Right-click on 0x39E and select **Insert Tracepoint** at 0x039E and select **TraceRun**. A cyan **T** will appear as shown:
3. Right-click on 0x3AC and select **Insert Tracepoint** at 0x03AC and select **TraceSuspend**. A cyan **T** will appear.
4. Clear the Trace Data window. 
5. RUN the program and after a few seconds STOP it. Examine the Trace Data window:

You can see where the trace started on 0x3A0 and stopped on 0x3AC with the red arrows.



The Disassembly window shows the following code:

```
33: curTicks = msTicks;
0x0000039E 4A04 LDR r2,[pc,#16] ; @0x000003B0
0x000003A0 6811 LDR r1,[r2,#0x00]
34: while ((msTicks - curTicks) < dlyTicks);
0x000003A2 BF00 NOP
0x000003A4 4A02 LDR r2,[pc,#8] ; @0x000003B0
0x000003A6 6812 LDR r2,[r2,#0x00]
0x000003A8 1A52 SUBS r2,r2,r1
0x000003AA 4282 CMP r2,r0
0x000003AC D3FA BCC 0x000003A4
```

The Trace Data window shows the following execution flow:



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x000003A6	LDR r2,[r2,#0x00]	
	X: 0x000003A8	SUBS r2,r2,r1	
	X: 0x000003AA	CMP r2,r0	
22.207 025 819 s	X: 0x000003AC	BCC 0x000003A4	
	TRACE RUN		
	X: 0x000003A0	LDR r1,[r2,#0x00]	←
	X: 0x000003A2	NOP	while ((msTicks - curTicks) < dlyTicks);
	X: 0x000003A4	LDR r2,[pc,#8] ; @0x000003B0	
	X: 0x000003A6	LDR r2,[r2,#0x00]	
	X: 0x000003A8	SUBS r2,r2,r1	
	X: 0x000003AA	CMP r2,r0	
22.420 424 253 s	X: 0x000003AC	BCC 0x000003A4	←

TIP: Sometimes, as in this case, the addresses the triggers are set to and the ones actually captured do not quite match. You will need to move your trigger points a bit to try and compensate.








Trace Configuration Skid:

The trace triggers use the same CoreSight hardware as the Watchpoints. This means that it is possible a program counter skid will happen. The program might not start or stop on the exact location you set the trigger to. You might have to adjust the trigger point location to minimize this effect.

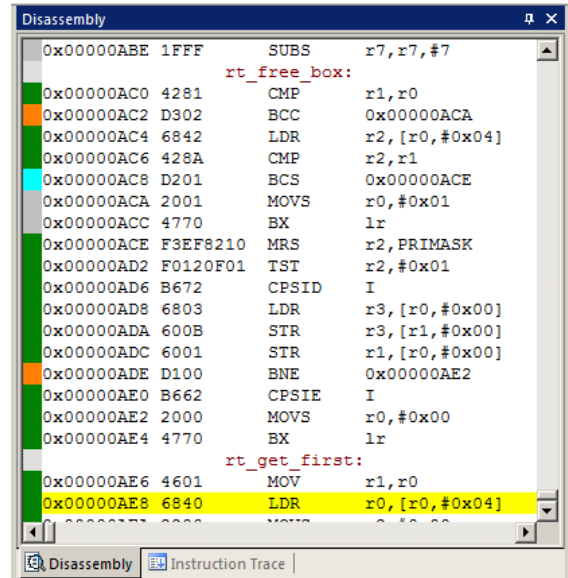
5) Code Coverage:

1. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
2. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin:
3. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not.

Colour blocks indicate which assembly instructions have been executed.

-  1. Green: this assembly instruction was executed.
-  2. Gray: this assembly instruction was not executed.
-  3. Orange: a Branch is always not taken.
-  4. Cyan: a Branch is always taken.
-  5. Light Gray: there is no assembly instruction here.
-  6. RED: Breakpoint is set here. (is actually a circle)
-  7. Next instruction to be executed.

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



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

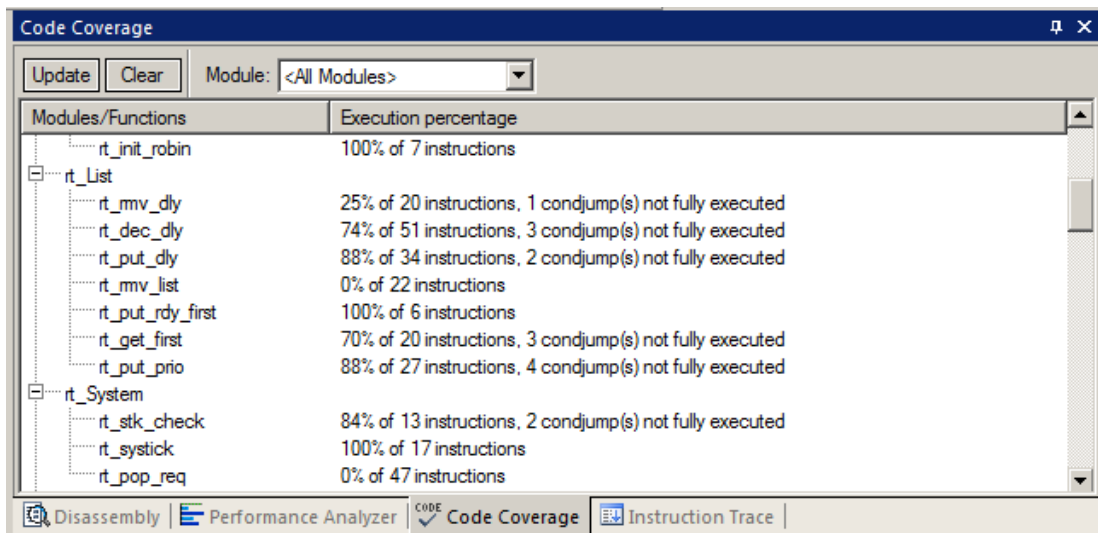
Why was 0x0000_0ACA never executed? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed?

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

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

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

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage. The next page describes how you can save Code Coverage information to a file.



Saving Code Coverage information:

Code Coverage information is temporarily saved during a run and is displayed in various windows as already shown. It is possible to save this information in an ASCII file for use in other programs.

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

You can Save Code Coverage in two formats:

1. In a binary file that can be later loaded back into µVision. Use the command Coverage Save *filename*.
2. In an ASCII file. You can either copy and paste from the Command window or use the log command:
 - 1) log > c:\cc\test.txt ; send CC data to this file. The specified directory must exist.
 - 2) coverage asm ; you can also specify a module or function.
 - 3) log off ; turn the log function off.

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

```
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
  3 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\Blinky.c\Delay - 100% (9 of 9 instructions executed)
\\Blinky\system_MK60N512MD100.c\SystemInit - 100% (34 of 34 instructions executed)
  2 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\system_MK60N512MD100.c\SystemCoreClockUpdate - 31% (36 of 116 instructions executed)
  5 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\startup_MK60N512MD100.s\__asm_0x27C - 47% (9 of 19 instructions executed)
\\Blinky\startup_MK60N512MD100.s\Reset_Handler - 100% (4 of 4 instructions executed)
\\Blinky\startup_MK60N512MD100.s\NMI_Handler - 0% (0 of 1 instructions executed)
\\Blinky\startup_MK60N512MD100.s\HardFault_Handler - 0% (0 of 1 instructions executed)
\\Blinky\startup_MK60N512MD100.s\MemManage_Handler - 0% (0 of 1 instructions executed)
```

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

```
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
EX | 0x000002B8 SysTick_Handler:
EX | 0x000002B8 483D      LDR      r0,[pc,#244] ; @0x000003B0
EX | 0x000002BA 6800      LDR      r0,[r0,#0x00]
EX | 0x000002BC 1C40      ADDS     r0,r0,#1
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
  3 condjump(s) or IT-bcock(s) not fully executed
EX | 0x000002C4 main:
EX | 0x000002C4 F04F34FF MOV      r4,#0xFFFFFFFF
EX | 0x000002C8 2501      MOVS     r5,#0x01
EX | 0x000002CA F000F8CB BL.W     SystemCoreClockUpdate (0x00000464)
```

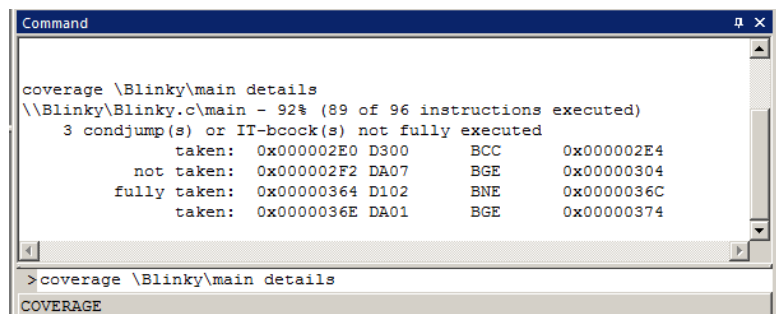
The first column above describes the execution as follows:

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

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

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

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





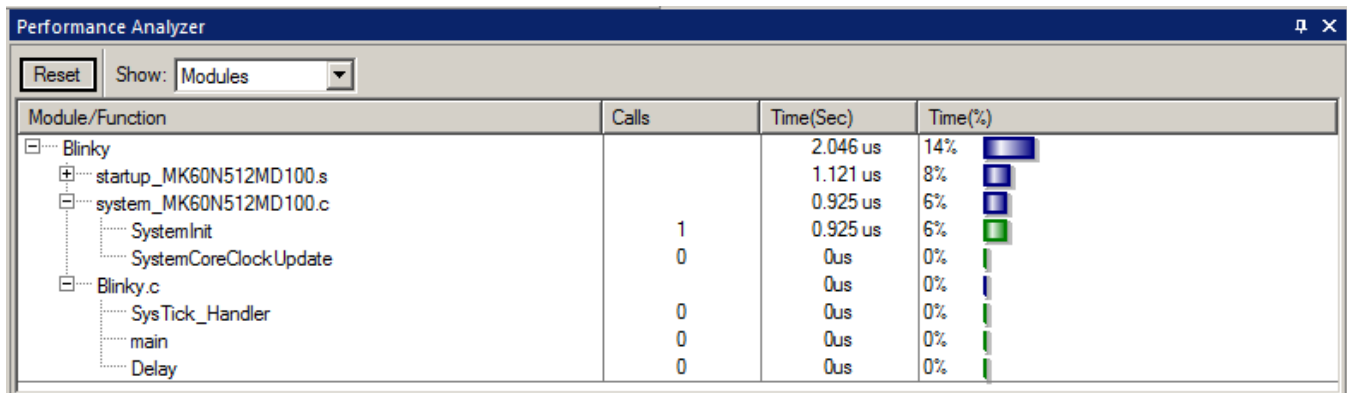
```
Command
coverage \Blinky\main details
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
  3 condjump(s) or IT-bcock(s) not fully executed
    taken: 0x000002E0 D300      BCC      0x000002E4
    not taken: 0x000002F2 DA07      BGE      0x00000304
    fully taken: 0x00000364 D102      BNE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


6) Performance Analysis (PA):


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

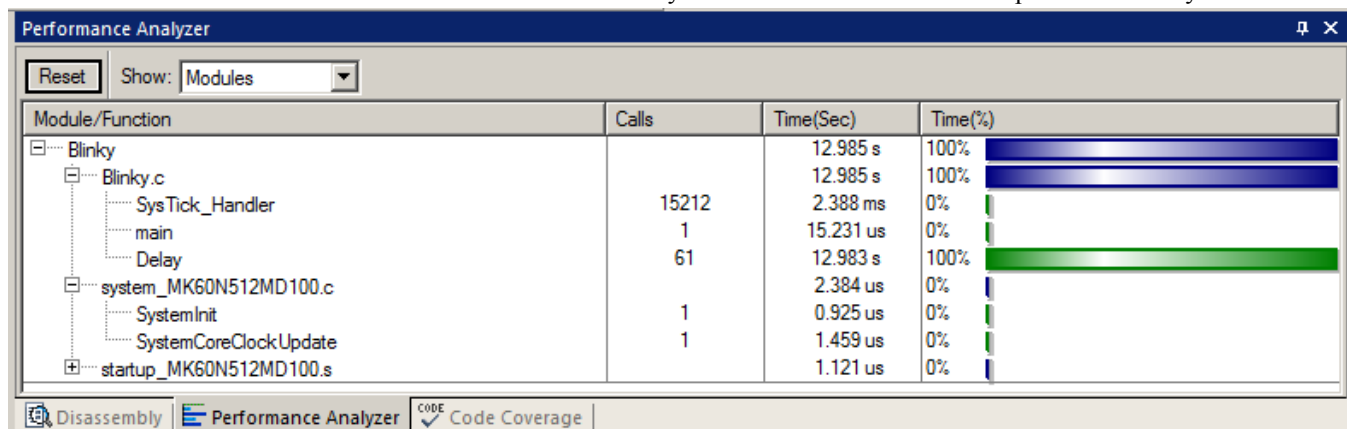
Keil provides Performance Analysis with the μ Vision simulator or with ETM and the ULINK pro . The number of total calls made as well as the total time spent in each function is displayed. A graphical display is generated for a quick reference. If you are optimizing for speed, work first on those functions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

1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it.  This clears the PA window and resets the Kinetis processor and reruns it to main(). You can also click on the RESET icon  and enter g,main in the Command window.
4. Do **not** click on RUN yet Expand some of the module names as shown below.
5. Times and number of calls has been collected in this short run from RESET to main().



Module/Function	Calls	Time(Sec)	Time(%)
Blinky		2.046 us	14%
startup_MK60N512MD100.s		1.121 us	8%
system_MK60N512MD100.c		0.925 us	6%
SystemInit	1	0.925 us	6%
SystemCoreClockUpdate	0	0us	0%
Blinky.c		0us	0%
SysTick_Handler	0	0us	0%
main	0	0us	0%
Delay	0	0us	0%


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. Most time is spent in the Delay function.



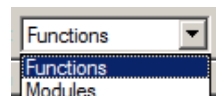
Module/Function	Calls	Time(Sec)	Time(%)
Blinky		12.985 s	100%
Blinky.c		12.985 s	100%
SysTick_Handler	15212	2.388 ms	0%
main	1	15.231 us	0%
Delay	61	12.983 s	100%
system_MK60N512MD100.c		2.384 us	0%
SystemInit	1	0.925 us	0%
SystemCoreClockUpdate	1	1.459 us	0%
startup_MK60N512MD100.s		1.121 us	0%

8. Select Functions from the pull down box as shown here and notice the difference.
9. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.

TIP: You can also click on the RESET icon  but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

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

TIP: The Performance Analyzer uses ETM to collect its raw data.

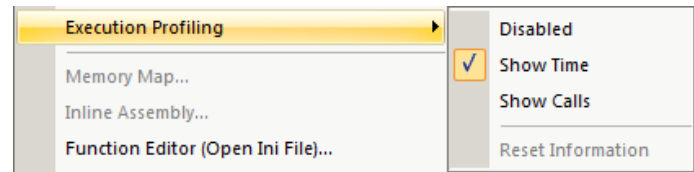


7) Execution Profiling:

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

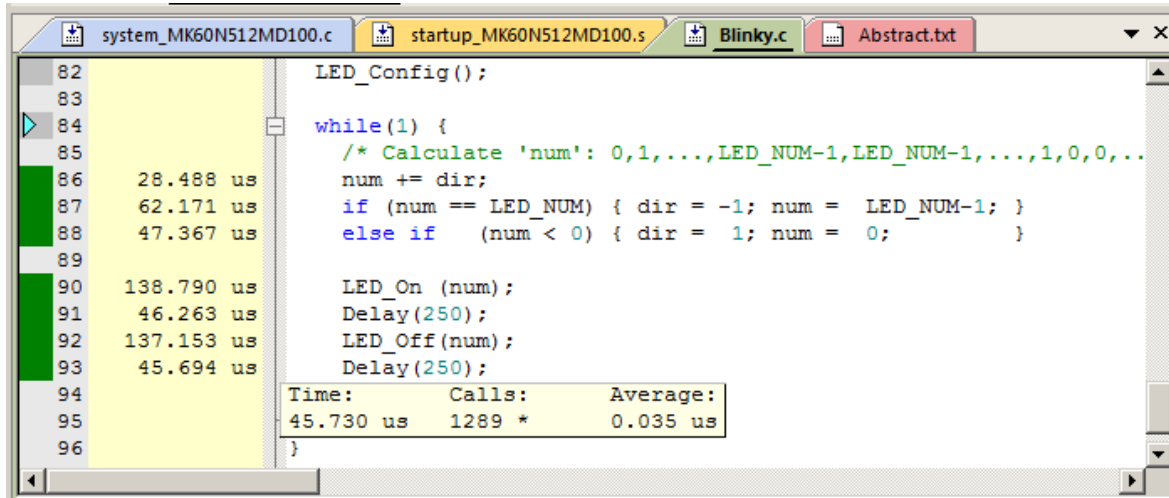
The μ Vision simulator also provides Execution Profiling.

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




Time:	Calls:	Average:
19.599 s	139910257 *	0.140 μ s

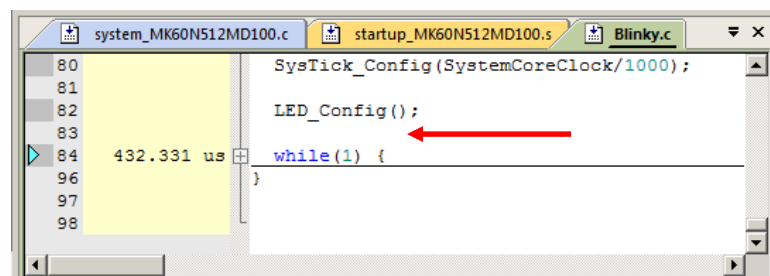
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



Outlining:

Each place there is a small square with a “-“ sign  can be collapsed down to compress the associated source files together.

- 1) Click in the square near the while(1) loop near line 84 as shown here:
- 2) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
- 3) Click on the + to expand it.
- 4) Stop the program and exit Debug mode.



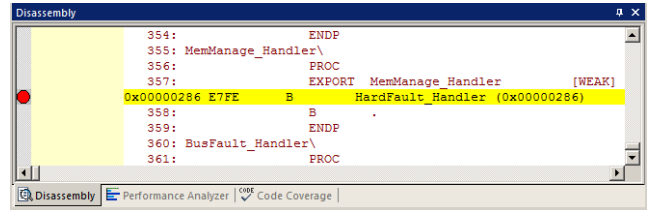
8) In-the-Weeds Example:


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 it is not hard to use.

If a Bus Fault occurs in our example, the CPU will end up at 0x0286 as shown in the disassembly window below. This is the Bus Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. The Trace Data window at the bottom shows this branch forever. This is not very useful.

This exception vector is found in the file startup_MK60N512MD100.s. If we set a breakpoint by clicking on the Hard Fault handler and run the program: at the next Bus Fault event the CPU will again jump to its handler.

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

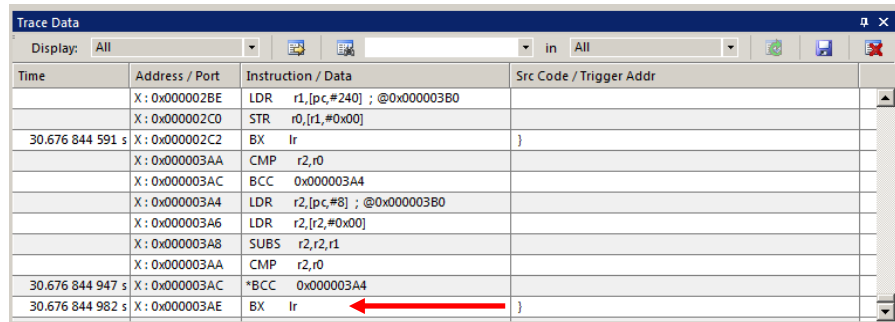


1. Use the Blinky example from the previous exercise, enter Debug mode.
2. Locate the Hard fault vector near address 0x0286 in the disassembly window or near line 353 startup_MK60N512MD100.s.
3. Set a breakpoint at this point. A red circle will appear.
4. Run the Blinky example for a few seconds and click on STOP. Repeat until it stops in the Delay() function.
5. Clear the Trace data window by clicking on the Clear Trace icon:  This is to help clarify what is happening.
6. In the Disassembly window, scroll down to the end of the Delay function to the BX LR instruction near 0x03AE.
7. In the register window, double-click on R14 (LR) register and set it to zero. BX is a branch to the address contained in a register, in this case LR. LR contains the return address from a subroutine. This is guaranteed to cause a hard fault when the processor tries to execute an instruction at 0x1FFF 0468. (initial SP). This will cause a real problem.
8. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
9. At the end of the Trace Data window you will find the remainder of the Delay function with the BX LR at the end.
10. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.
11. Remove the breakpoint and click on RUN and then STOP.
12. You will now see all the Hard Fault branches as shown in the bottom screen:

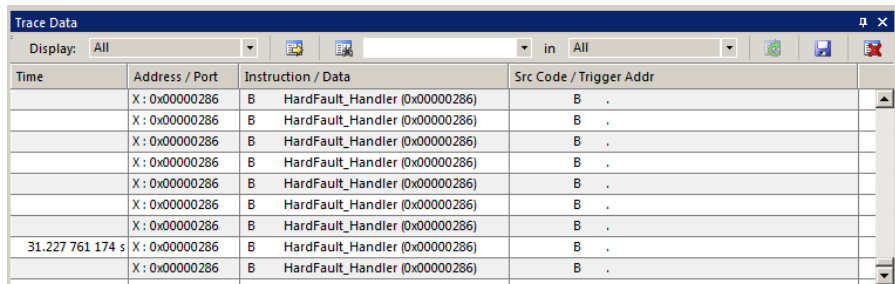
This is admittedly a contrived example but it clearly shows how quickly and easily ETM trace can show you program flow problems.

These types of problems are very hard to solve and usually take a long time.

TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 353... and select TraceHalt. When Hard Fault is reached, trace collection will halt but the program will keep executing the B instructions forever.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x000002BE	LDR r1,[pc,#240] ; @0x000003B0	
	X: 0x000002C0	STR r0,[r1,#0x00]	
30.676 844 591 s	X: 0x000002C2	BX lr	}
	X: 0x000003AA	CMP r2,r0	
	X: 0x000003AC	BCC 0x000003A4	
	X: 0x000003A4	LDR r2,[pc,#8] ; @0x000003B0	
	X: 0x000003A6	LDR r2,[r2,#0x00]	
	X: 0x000003A8	SUBS r2,r2,r1	
	X: 0x000003AA	CMP r2,r0	
30.676 844 947 s	X: 0x000003AC	*BCC 0x000003A4	
30.676 844 982 s	X: 0x000003AE	BX lr	}



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
31.227 761 174 s	X: 0x00000286	B HardFault_Handler (0x00000286)	B .
	X: 0x00000286	B HardFault_Handler (0x00000286)	B .

9) Serial Wire and ETM Trace Summary:

We have several basic debug systems as implemented in Kinetis Cortex-M4 devices:

1. SWV and ITM data output on the SWO pin located on the JTAG/SWD 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 all 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 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. SWV and ETM can only see CPU actions.

10) Keil Products and contact information: See www.keil.com/freescale

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

- MDK-Lite™ (Evaluation version) 32K Code and Data Limit - \$0
- **MDK-Freescale™ For all Kinetis Cortex-M3/4, 1 year term license. RTX included. - \$745**
- **NEW !! MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit) - \$3,200**
- MDK-Standard™ (with included RTX RTOS with source code) - \$4,895
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) \$9,500

USB-JTAG 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

A Keil ULINK must be purchased or you can use the OS-JTAG on the Kinetis Tower board. For Serial Wire Viewer (SWV), a ULINK2, ULINK-ME or a J-Link is needed. For ETM support, a ULINKpro is needed. OS-JTAG does not support either SWV or ETM debug technology.

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

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

For the ARM University program: go to www.arm.com and search for university. Email: university@arm.com

All products include Technical Support for 1 year. This can be renewed.

Keil RTX™ Real Time Operating System

- RTX is provided free as part of Keil MDK.
- No royalties are required and is very easy to use. It has a BSD license.
- RTX source code is included with all versions of MDK.
- Kernel Awareness visibility integral to µVision.

MQX: See www.keil.com/freescale for a descriptive movie.

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

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor.

For Linux, Android, bare metal (no OS) and other OS support on Freescale i.MX and Vybrid series processors please see DS-5 at www.arm.com/ds5/.



For more information:

Keil Sales In North and South America: sales.us@keil.com or 800-348-8051. Outside the US: sales.intl@keil.com

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

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

For the latest version of this document, contact the author, Keil Technical support or www.keil.com/freescale.

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

