

### Introduction:

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 see [www.keil.com/freescale](http://www.keil.com/freescale). MDK has example projects for the TWR-K40N256, TWR-K53N512, TWR-K60N512 and KWIKSTIK boards. MDK supports the K10, K20, K30, K40, K50 and K60 series of Cortex-M4 processors.

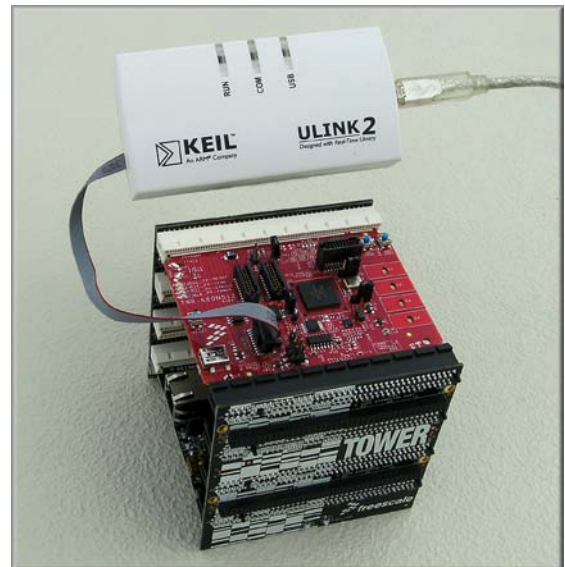
Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. The addition of a license number will turn it into the full, unrestricted version. MDK-Freescale is a low cost \$745 toolkit for Kinetis processors. MDK includes Keil RTX™, our own RTOS. No royalty payments are required and source code is included. RTX examples are provided.

**Linux and Android:** For Linux, Android and bare metal (no OS) support on Freescale i.MX series processors (ARM9™, Cortex-A8, A9 and A15), please see the ARM DS-5™ toolkit at [www.arm.com/ds5/](http://www.arm.com/ds5/).

### Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M4 users:

1. μVision IDE with Integrated Debugger, Flash programmer, the ARM® Compiler toolchain 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 window is updated live.
5. Kernel Awareness for Keil RTX, CMX, Quadros and Micrium. All RTOSs will compile with MDK.
6. **MQX:** An MQX port for MDK will be released August 2011. Kernel Awareness windows for MQX will also be available.
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 Signum Systems JtagJetTrace. P&E OSJTAG is also supported by μVision.
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:

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: a real-time kernel awareness program for the Keil RTX RTOS.

### 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 adapter. 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 adds all the program counter values to the features provided by SWV. This allows advanced debugging features including Code Coverage, Performance Analysis and Execution Profiling providing both time and calls. ETM requires a special adapter such as the ULINKpro or Segger J-Trace. This document uses ULINKpro for ETM trace.

## Freescale Evaluation Boards:

This document uses the Freescale Kinetis TWR-K60N512. The K40, K53, 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 4.20. MDK 4.21 has the best ETM support for Kinetis. The evaluation copy of MDK (MDK-Lite) is available free on [www.keil.com](http://www.keil.com). Do not confuse  $\mu$ Vision 4 with MDK 4.0. The “4” is a coincidence.

To obtain a copy of MDK go to [www.keil.com/arm](http://www.keil.com/arm) and select “Evaluation Software” from the left column.

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 you can use. SWV works best with a ULINK.

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

## Example Programs:

Two example programs plus two more for programming the Freescale FlexMem are included in either C:\Keil\ARM\Boards\Freescale\TWR-K40N256, \TWR-K53N512, \TWR-K60N512 or KWIKSTIK.

- **Blinky:** blinks 3 LEDs. 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/appnotes/docs/apnt\\_220.asp](http://www.keil.com/appnotes/docs/apnt_220.asp)
- For **DSP** examples see [www.keil.com/freescale](http://www.keil.com/freescale).

## 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 5. 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 setting 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 provides all the program counter values. 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. **P&E OSJTAG:**  $\mu$ 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 Serial Wire Viewer are not implemented. These limitations are listed on page 4 along with the configuration instructions. Keil MDK evaluation version can be used to evaluate programs under 32K.
2. **Segger J-Link and J-Trace:** J-Link Version 6 (black) or later supports Serial Wire Viewer. J-Trace for the Cortex-M provides ETM. Data reads and writes are not currently supported.
3. **Signum Systems JtagJetTrace:** This provides very good ETM support with its advanced triggers and filters for both ETM and SWV. Currently it does not support the  $\mu$ Vision Logic Analyzer, on-the-fly data reads and writes, Code Coverage, Performance Analyzer or Execution.Profiling. The Signum window is embedded in  $\mu$ Vision. The Signum Systems JtagJet, which has no ETM, does not support Serial Wire Viewer or ETM.

**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  $\mu$ Vision Cortex-M Target Driver Setup.

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

**SWO:** Serial Wire Output: SWV frames come out this one 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: Provides all the program counter values. ULINK*pro* provides ETM support.

## Index:

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

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

### Part B: Example Projects:

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

### PART C: ETM Embedded Trace Macrocell with the ULINK*pro*:

1. Configure the ULINK*pro* with ETM Trace: 22
  - a. Ini Initialization file 22
  - b. Configuring SWV and ETM 23
2. Blinky Example: ETM Frames from RESET and beyond: 24
3. Code Coverage: 25
4. Performance Analyzer (PA) 26
5. Execution Profiler 27
  - a. Outlining 27
6. In-The-Weeds Example 28
7. Serial Wire Viewer summary 29
8. Keil Products and contact information 30

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

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

$\mu$ Vision 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 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 ULINK<sub>pro</sub> ETM Trace will provide all PC values, Code Coverage, Execution Profiling, Performance Analysis and advanced program flow debugging.





**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 updates.

#### Install P&E Drivers:

1. Download the drivers from [www.keil.com/download/docs/408.asp](http://www.keil.com/download/docs/408.asp). Filename is fslkinetisdriversv102.exe
2. Disconnect the Kinetis board from USB and close  $\mu$ Vision if it is running.
3. Run fslkinetisdriversv102.exe (or a later version if available) to install the OSJTAG drivers.

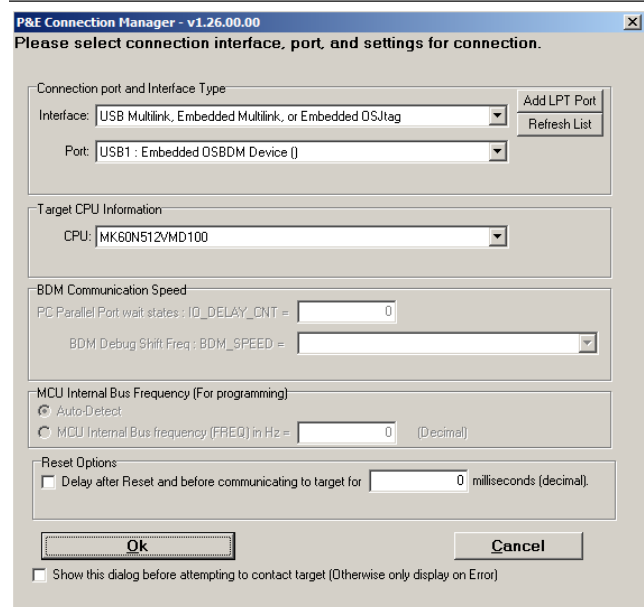
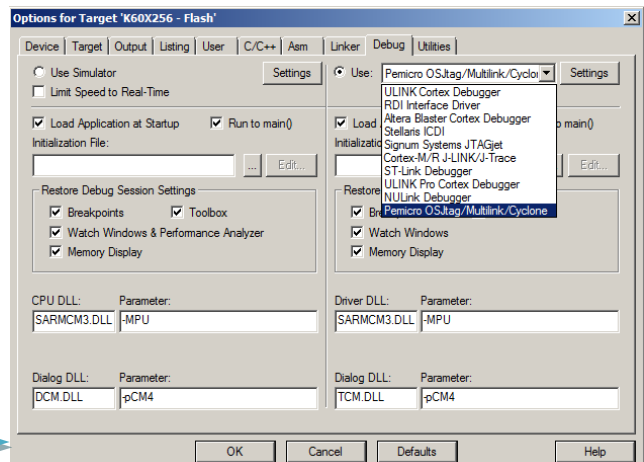
#### Configure $\mu$ Vision:

1. Start  $\mu$ Vision by clicking on its desktop icon. 
2. Leave 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 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 processor in the Target CPU Information dialog box.
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 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 there is a new image whenever Debug mode is entered.



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

Freescale provides the ARM standard 20 pin hi-density 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 provided 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:

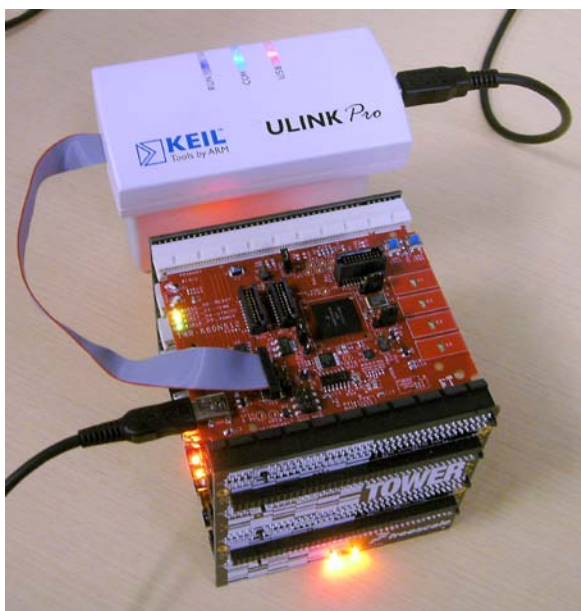
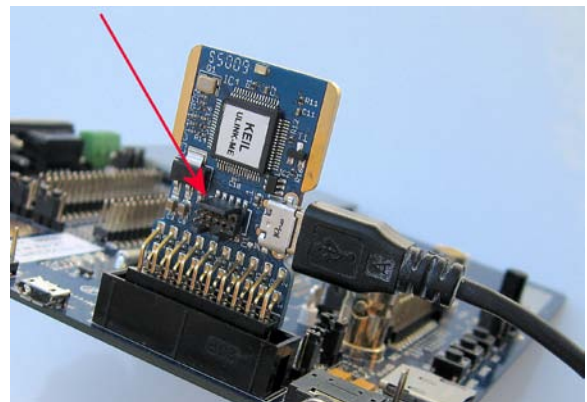
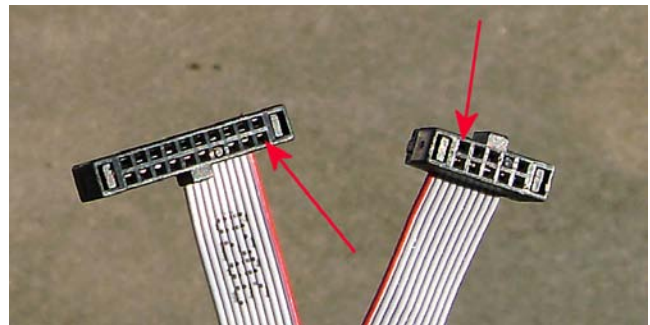
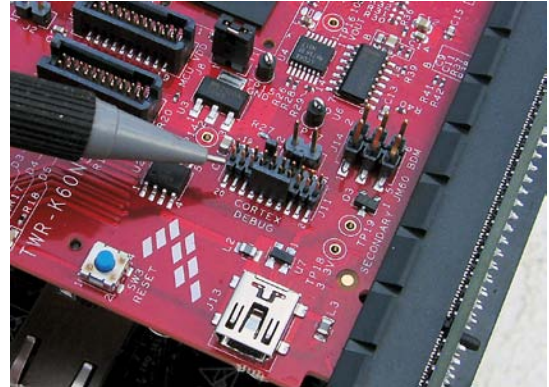
Pictured is the 10 pin to 20 pin Keil connector. The arrows point to pin 1. This cable is supplied with the ULINK2 and ULINK-ME. This cable can also be ordered by contacting Keil sales or tech support. The part number is ULC-2010A or B.

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 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.

The pictured 10 to 20 pin cable is supplied with the ULINKpro but is not needed for this target. ULINKpro is pictured below:



**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 $\mu$ Vision:

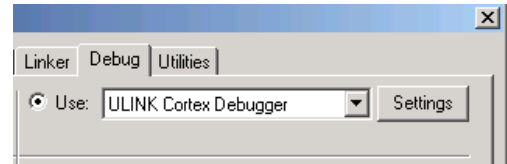
It is easy to select a USB debugging adapter in  $\mu$ Vision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are selected using the Debug and Utilities tabs.

This document will use a ULINK2 or ULINK-ME as described. You can substitute a ULINK $pro$  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 ULINK $pro$ , which is a Cortex-Mx 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.

#### Select the debug connection to the target:

1. Assume the ULINK2 is connected to a powered up Kinetis target board,  $\mu$ Vision 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 ULINK Cortex 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.
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. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will operate only in SW mode.

#### 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 the programming algorithm.
9. If an algorithm not already selected, select Add and select MKXX 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. In RAM for Algorithm, confirm Size: 0x8000

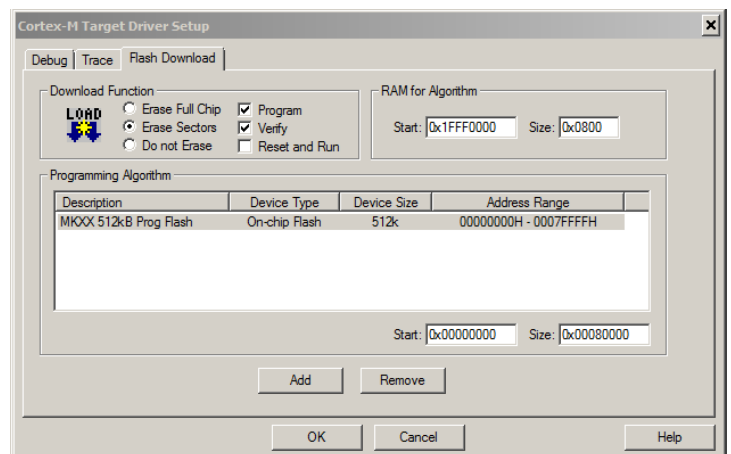
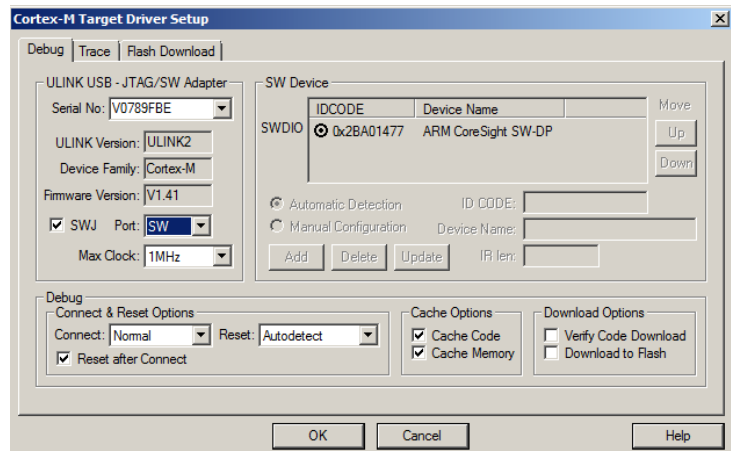
**TIP:** To program the Flash every time you enter Debug mode, check Update target before Debugging.

13. Click on OK to return to the  $\mu$ Vision main screen.
14. You have successfully connected to Kinetis.
15. At this point you can compile source code, program them into Flash, enter Debug mode and then run and stop your program.

**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 ULINK $pro$ , and have the opposite ULINK physically connected to your PC; the error message will say “No ULINK device found”.


This message actually means that  $\mu$ Vision found the wrong Keil adapter connected.

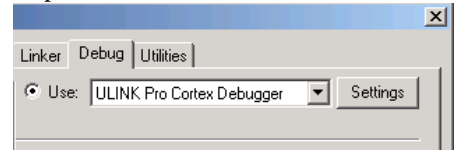


#### 4) Configuring ULINKpro and μVision:

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 5.

##### 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 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.
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.

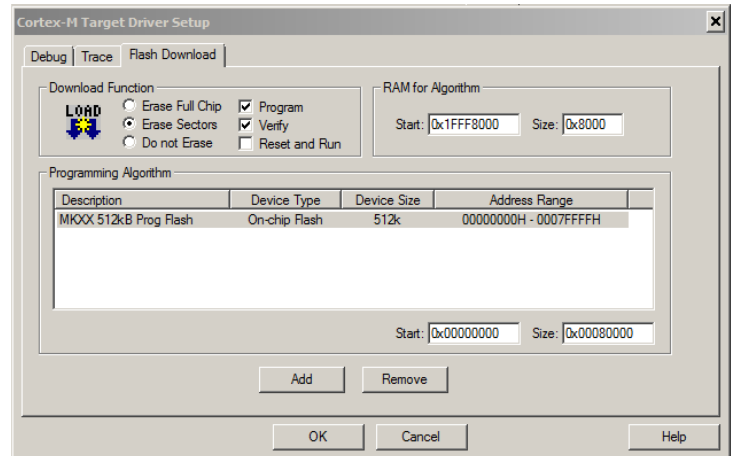
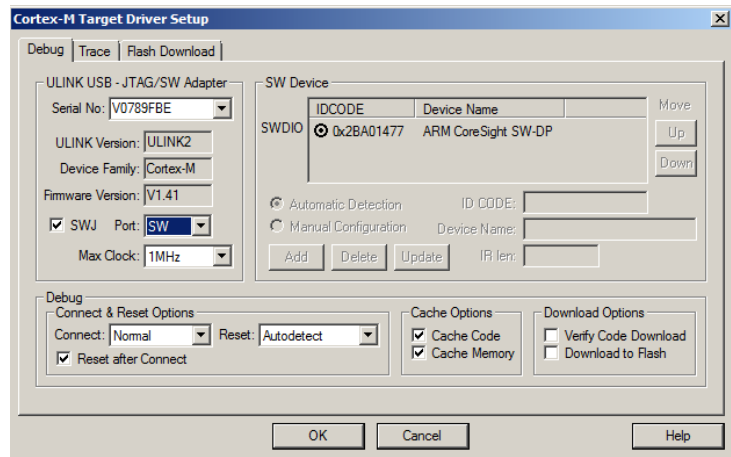
**TIP:** You can do regular debugging using JTAG. SW (also called SWD) and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode unless the ULINKpro is using the Trace Port to output the trace frames.

##### 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 the programming algorithm.
4. If one is not visible click Add and select MKXX 512kB Prog Flash as shown here 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 and run and stop these programs.



**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 (Instruction Trace) will be 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.

## 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 K60 Tower board. If you are using a K40, K53 board or the Kwikstik, select the appropriate Blinky project.

It is possible to use the ULINK $pro$  or the P&E OSJTAG. The Segger J-Link is also supported. Ulink2 is selected by default.

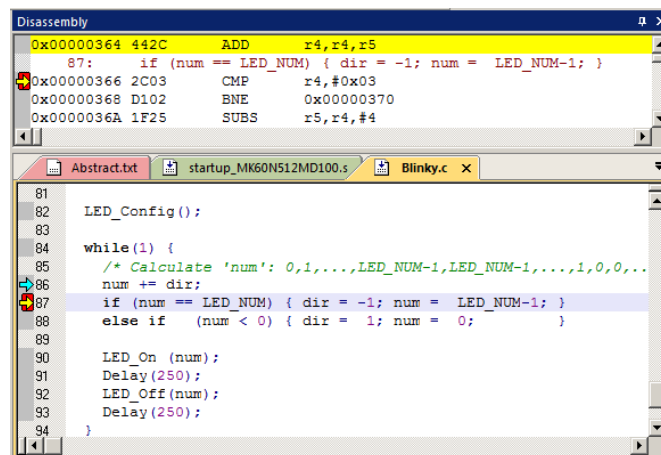
1. Connect the ULINK2 as pictured on the first page. Use the special 10 to 20 pin cable for both the ULINK2 and ULINK-ME. See pages 4 for P&E and 7 for ULINK $pro$ .
  2. Start  $\mu$ Vision by clicking on its desktop icon. 
  3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky\Blinky.uvproj.
  4. Make sure "K60X256 Flash" is selected (soon this will be K60X512).  This is where you create and select different target configurations such as to execute a program in RAM or Flash.
  5. 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.
6. Program the Kinetis flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
  7. 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.
  8. 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.***

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 Blink running, double-click in the margin on a darker gray block somewhere in the while(1) loop as shown below:
2. A red block 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. Note 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 double-clicking on the red block.



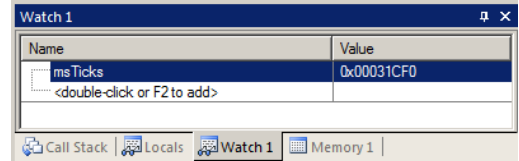
### 3) Watch and Memory Windows and how to use them:

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-Mx processors. It is also possible to “put” or insert values into these memory locations in real-time. It is possible to “drag and drop” variables into windows or enter them manually.

#### 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. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
4. In Blinky.c, block `msTicks`, click and hold and drag it into Watch 1. Release it and it will be displayed updating as shown here: 
5. You can also enter a variable manually by double-clicking or pressing F2 and using copy and paste or typing the variable.

**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.

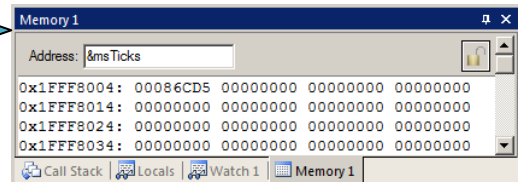


6. Double click on the value for `msTicks` in the Watch window. Enter the value 0 and press Enter. 0 will be inserted into memory in real-time. You can do this in the Memory window with a right-click and select Modify Memory.

#### Memory window:

1. Drag ‘n Drop `msTicks` into the Memory 1 window or enter it manually.
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 but will display the variable 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.

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




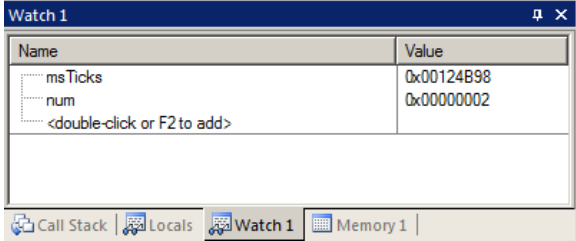
#### How It Works:

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.






## 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. Drag and Drop `num` into Watch 1 window. Note you are unable to accomplish this.
4. Double-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: 
5. The program will stop and you can now enter `num` into Watch 1. Remove the breakpoint by double-clicking on it to remove the red block.  
**TIP:** Remember: you can set hardware breakpoints on-the-fly in the Cortex-M4 !
6. Run the program and note `num` value displays as ????????
7. Stop the program and it will probably stop in the Delay function. `num` now displays as <out of scope> or ????????
8. These happen because  $\mu$ 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.
9. Start the program by clicking on the Run icon.
10. `num` changes to ????????. Set a breakpoint by double-clicking in the margin inside the while (1) loop as before in Step 4 above. The program will stop on this breakpoint.
11. `num` correctly displays its value as shown here: 
12. Each time you click RUN, this value is updated.

## 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. 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.  You will have to re-enter `num` in the Watch 1 window because it isn't the same variable anymore – it is a static variable now instead of a local. Drag 'n Drop is the fastest way.
6. Remove the breakpoint you previously set and click on RUN. You can use Debug/Kill All Breakpoints to do this.
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.  and 

**Note:** In MDK 4.20 you might have to run and stop the program to after the declaration of `num` (use a breakpoint to do this) and then drag and drop it into the Watch window. This is fixed in MDK 4.21 and later.

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


**TIP:** `num` will be deleted from the Watch window when you leave the debug window because it is in the middle of a function as opposed to being a global variable. To have it remembered the variable must be fully qualified. In this case it would be `\Blinky\main\num`. You can drag a variable from the Symbols window and it will be entered fully qualified.

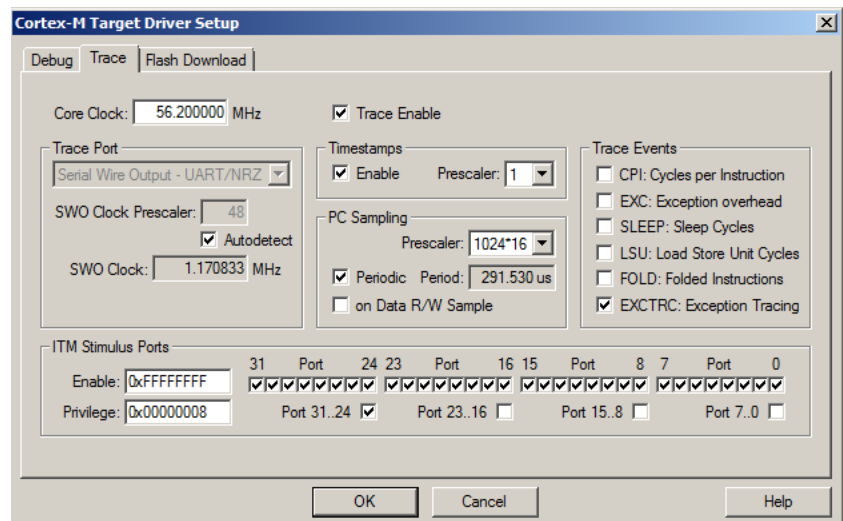
#### 4) Getting the Serial Wire Viewer (SWV) working:

Serial Wire Viewer provides program trace information in real-time.




A) For ULINK2 or ULINK-ME: Configuration must be set as on page 6 (ULINK<sub>pro</sub> instructions are on the next page)

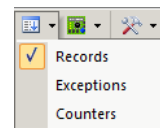
##### Configure SWV:

1.  $\mu$  Vision must be stopped and in edit mode (not debug mode).
2. Select Options for Target  or ALT-F7 and select the Debug tab. Confirm Port: is set to SW.
3. Click on Settings: beside the name of your adapter on the right side of the window.
4. Click on the Trace tab. The window below is displayed.
5. In Core Clock: enter 56.2 and select the Trace Enable box.
6. Select Periodic and leave everything else at default.
7. Click on OK twice to return to the main  $\mu$  Vision menu. SWV is now configured and ready to use.



##### Display Trace Records:

8. Enter Debug mode. 
9. Click on the RUN icon. 
10. Open Trace Records window by clicking on the small arrow beside the Trace icon: 
11. The Trace Records window will open and display Exceptions and PC Samples as shown below: You might have to stop the program or switch the Filter: dialog box.



**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 52.9 or 48 MHz. Use a scope on the SWO pin if needed.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					000003C0H		41333373632	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	

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

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 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.

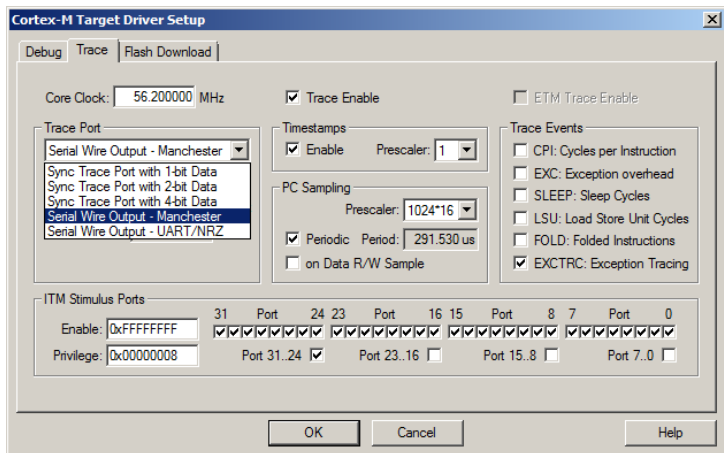
This is an easy way to see when your interrupts are occurring and how often.

B) For **ULINKpro**: This is not ETM trace. See page 23 for ETM trace.



**Configure SWV:**

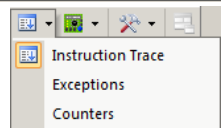
1.  $\mu$ Vision must be stopped and in edit mode (not debug mode).
2. Configure ULINKpro as found on page 7: 4) Configuring ULINKpro and  $\mu$ Vision:
3. Select Options for Target  or ALT-F7 and select the Debug tab.
4. Click on Settings: beside the name of your adapter on the right side of the window.
5. Click on the Trace tab. The window below is displayed.
6. Core Clock: ULINKpro determines this automatically. It uses this to calculate timings. Enter 56.2 MHz.
7. Select the Trace Enable box.
8. In the Trace Port 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.
10. Click on OK twice to return to the main  $\mu$ Vision menu. SWV is now configured and ready to use.

**TIP:** Sync Trace Port with 4 bit Data field sends the trace 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 23.



**Display Trace Records:**

1. Enter Debug mode. 
2. Click on the RUN icon. 
3. Open Instruction Trace window by clicking on the small arrow beside the Trace icon:
4. The Race Records window will open and display Exceptions and PC Samples as shown below:



**TIP:** The Instruction Trace window is different that 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 program counter values are displayed with ETM trace.




You cannot clear the Instruction Trace window by double-clicking inside it. To clear the trace, exit and re-enter debug mode. New trace windows are under development.

#	Type	Flag	Num	PC	Opcode	Instruction	Source Code	Address	Data	Cycles	Time[s]
174864	PC Sample			0x000003BE	4A03	LDR r2,[pc,#12] : @0x000003CC				1415085058	141.50850580
174865	PC Sample			0x000003BE	4A03	LDR r2,[pc,#12] : @0x000003CC				1415101442	141.51014420
174866	PC Sample			0x000003BE	4A03	LDR r2,[pc,#12] : @0x000003CC				1415117826	141.51178260
174867	Exception Entry		15							1415127027	141.51270270
174868	Exception Exit		15							1415127037	141.51270370
174869	Exception Return		0							1415127047	141.51270470
174870	PC Sample			0x000003BE	4A03	LDR r2,[pc,#12] : @0x000003CC				1415134210	141.51342100
174871	PC Sample			0x000003BE	4A03	LDR r2,[pc,#12] : @0x000003CC				1415150594	141.51505940
174872	PC Sample			0x000003BE	4A03	LDR r2,[pc,#12] : @0x000003CC				1415166978	141.51669780
174873	Exception Entry		15							1415174999	141.51749990
174874	Exception Exit		15							1415175009	141.51750090
174875	Exception Return		0							1415175019	141.51750190
174876	PC Sample			0x000003C4	4282	CMP r2,r0				1415183362	141.51833620
174877	PC Sample			0x000003C4	4282	CMP r2,r0				1415199746	141.51997460
174878	PC Sample			0x000003C4	4282	CMP r2,r0				1415216130	141.52161300
174879	Exception Entry		15							1415222971	141.52229710
174880	Exception Exit		15							1415222981	141.52229810
174881	Exception Return		0							1415222991	141.52229910

## 5) 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<sub>pro</sub> 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.

1. SWV must be configured as found on page 11 or page 12 for ULINK<sub>pro</sub>. Enter debug mode .
2. Select Debug/Debug Settings and select the Trace tab.
3. Unselect Periodic and EXCTRC. This is to prevent SWO pin frame overrun. Click OK to return to the main menu.
4. Run the program.  **TIP:** You can configure the LA while the program is running or stopped.
5. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. .
6. Locate the volatile variable msTicks in Blinky.c. It is declared near line 19.
7. Block msTicks and drag it into the LA window and release it. Or click on Setup in the LA and enter it manually.
8. Click on Setup and set Max: in Display Range to 0xFFFF. Click on close. The LA is completely configured now.
9. Drag and drop msTicks into the Watch 1 window. It should be incrementing if the program is running.
10. Adjust the Zoom OUT icon in the LA window to about 1 second or so to get a nice ramp as shown below.
11. In the Watch 1 window, double-click on the msTicks value and enter 0 and press Enter.
12. This 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)

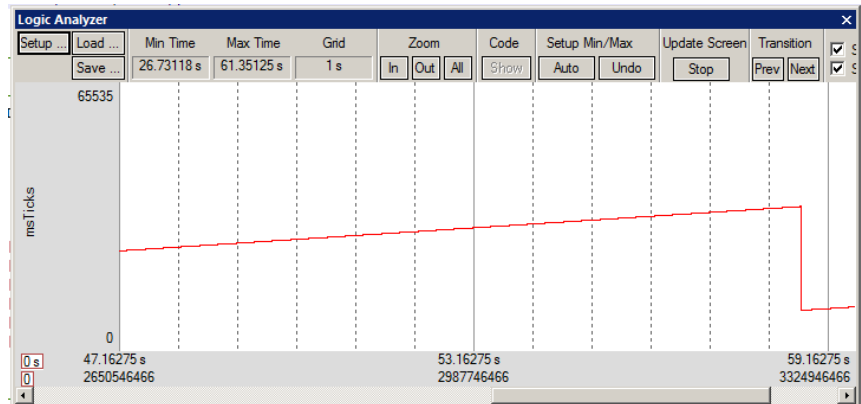
**TIP:** The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

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 below means:
7. The instruction at 0x2C4 caused a write of data 0x0000\_F07D to address 0x1FFF\_8008 at the listed time in Cycles or Time.

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

**TIP:** The ULINK<sub>pro</sub> will give a more sophisticated Instruction Trace window.

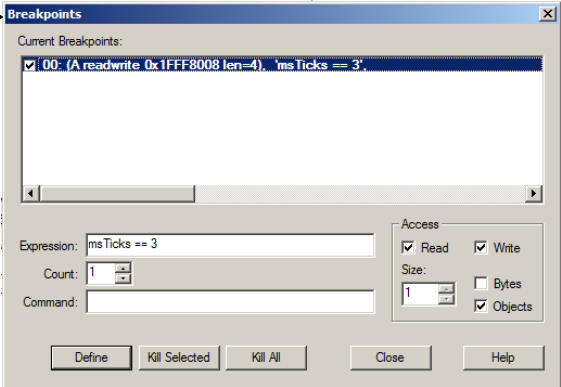
Watchpoints are described on the next page.

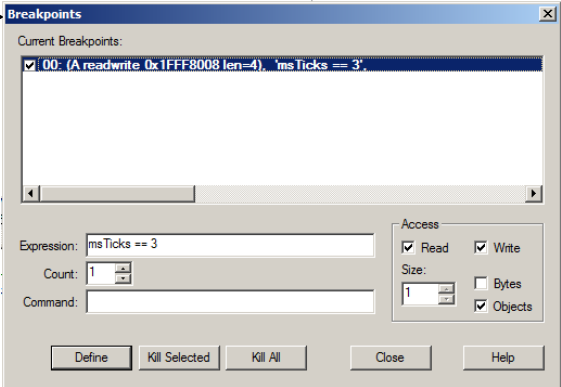


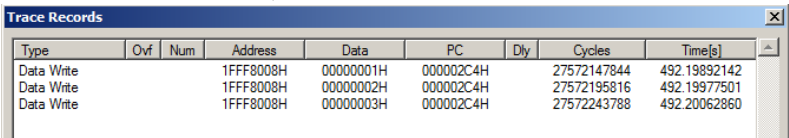
Type	Ovr	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			1FFF8008H	0000F07DH	000002C4H		5734717783	103.63255022
Data Write			1FFF8008H	0000F07EH	000002C4H		5734765755	103.63340382
Data Write			1FFF8008H	0000F07FH	000002C4H		5734813727	103.63425741
Data Write			1FFF8008H	0000F080H	000002C4H		5734861699	103.63511101
Data Write			1FFF8008H	0000F081H	000002C4H		5734909671	103.63596460
Data Write			1FFF8008H	0000F082H	000002C4H		5734957643	103.63681820
Data Write			1FFF8008H	0000F083H	000002C4H		5735005615	103.63767179
Data Write			1FFF8008H	0000F084H	000002C4H		5735053587	103.63852538
Data Write			1FFF8008H	0000F085H	000002C4H		5735101559	103.63937898
Data Write			1FFF8008H	0000F086H	000002C4H		5735149531	103.64023257
Data Write			1FFF8008H	0000F087H	000002C4H		5735197503	103.64108617
Data Write			1FFF8008H	0000F088H	000002C4H		5735245475	103.64193976
Data Write			1FFF8008H	0000F089H	000002C4H		5735293447	103.64279336
Data Write			1FFF8008H	0000F08AH	000002C4H		5735341419	103.64364695
Data Write			1FFF8008H	0000F08BH	000002C4H		5735389391	103.64450054
Data Write			1FFF8008H	0000F08CH	000002C4H		5735437363	103.64535414
Data Write			1FFF8008H	0000F08DH	000002C4H		5735485335	103.64620773
Data Write			1FFF8008H	0000F08EH	000002C4H		5735531307	103.64706133
Data Write			1FFF8008H	0000F08FH	000002C4H		5735579279	103.64791492
Data Write			1FFF8008H	0000F090H	000002C4H		5735627251	103.64876852

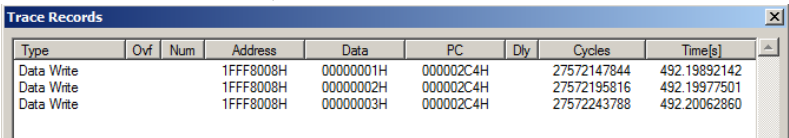
## 6) Watchpoints: Conditional Breakpoints

The Kinetis Cortex-M4 processors have four Watchpoint comparators. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses 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. You can use any ULINK can be used. Each Watchpoint uses two comparators. MDK 4.20 and earlier does not warn you if you have set too many Watchpoints. MDK 4.21 and later does indicate when you have used up all the comparators.

1. Using the example from the previous page, stop the program.
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 Watch 1 window to zero.
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.



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<sub>pro</sub> will display a different window. 
13. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.



Type	Ovf	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 0 and select RUN.
15. When finished, open the Breakpoints window and either use Kill All to delete the watchpoints or deselect them by unchecking them. Having undeleted Watchpoints activate unexpectedly can be rather confusing.
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.

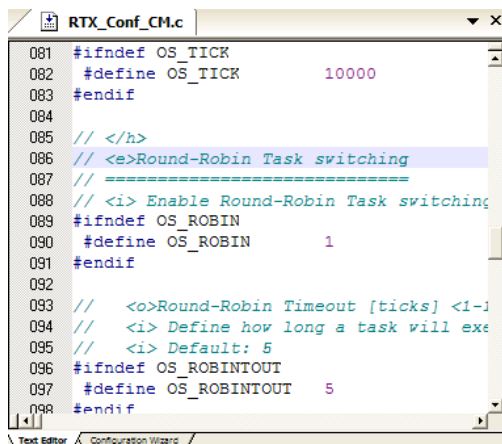
## 7) RTX\_Blinky Example Program with Keil RTX RTOS: A Stepper Motor example

Keil provides RTX, a full feature RTOS is a component of MDK. RTX is included with source code and is not crippled or limited. It can have up to 255 tasks and no royalty payments are required. This example explores the RTX\_Blinky project. Keil will work with any RTOS. An RTOS is just a set of C functions that gets compiled with your project.

1. Start  $\mu$ Vision by clicking on its icon on your Desktop if it is not already running. 
2. Select Project/Open Project.
3. Open the file C:\Keil\ARM\Boards\Freescale\TWR-K60N512\RTX\_Blinky\Blinky.uvproj.
4. RTX\_Blinky uses a ULINK<sub>pro</sub> as default: if you are using a ULINK2 please configure it as described on page 4 under 2) ULINK2 or ULINK-ME and  $\mu$ Vision Configuration: You do not need to configure the trace yet. After you have configured your adapter, save the settings by selecting File/Save All. Closing  $\mu$ Vision also effectively performs a Save All. You can also make a new target configuration to select between different configurations.. In the Flash programming in the Utilities tab, make sure RAM for Algorithm is set to 0x8000 and not 0x800.
5. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
6. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
7. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
8. The LEDs will blink indicating the four waveforms of a stepper motor driver changing. Click on STOP .

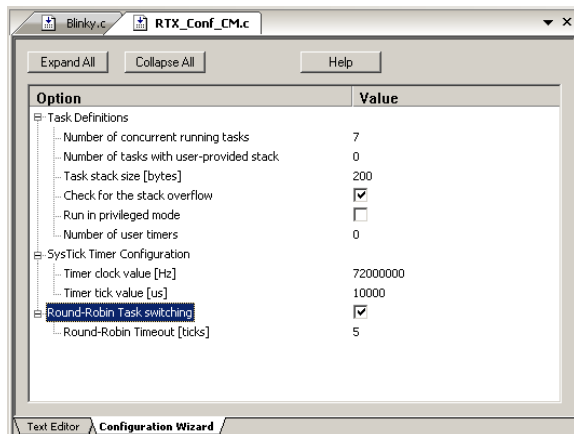
### The Configuration Wizard for RTX:

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.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. This is a great feature as it is much easier changing settings here than in the source code.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a `</h>` or `<i>`.
8. The new  $\mu$ Vision4 System Viewer windows are created in a similar fashion. They use XML.



```
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 exe
095 // <i> Default: 5
096 #ifndef OS_ROBINTOUT
097 #define OS_ROBINTOUT  5
098 #endif
```

Text Editor: Source Code



Configuration Wizard

## 8) 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 a Task Aware window for RTX. Other RTOS companies also provide awareness plug-ins for  $\mu$ 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.

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 ULINK 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.
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  $\mu$ Vision main menu. **The Serial Wire Viewer is now configured in  $\mu$ 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 5 seconds by clicking on the ALL and then the + and – icons.

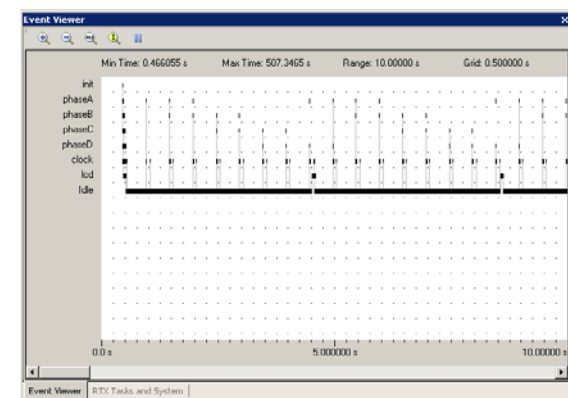
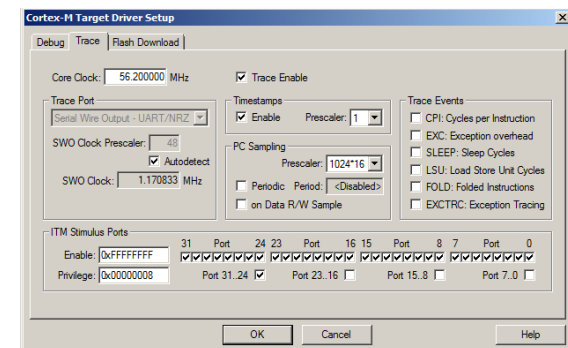
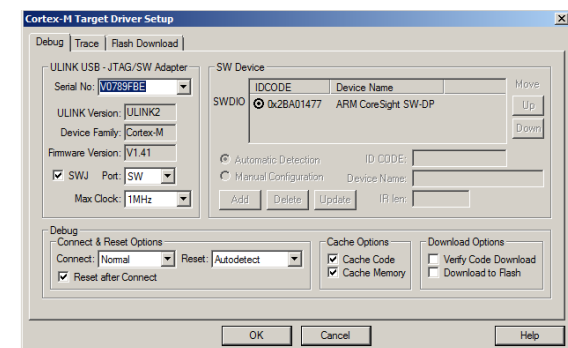
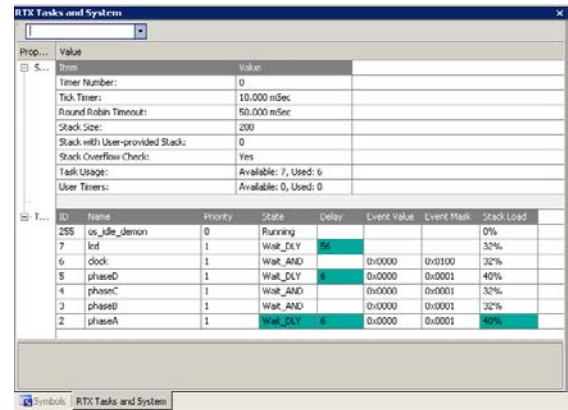
**TIP:** View/Periodic Window Update must be selected !

**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 good ITM frames is the Core Clock: is not set correctly

**Cortex-M4 Alert:**  $\mu$ 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 will not have to stop the program to view this data. No CPU cycles are used. Your program runs at nearly full speed. No instrumentation code need be inserted into your source. You will find this feature very useful !

**TIP:** You can use a ULINK2, ULINK-ME, ULINKpro or J-Link for these RTX Kernel Awareness windows.



## 9) 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. 

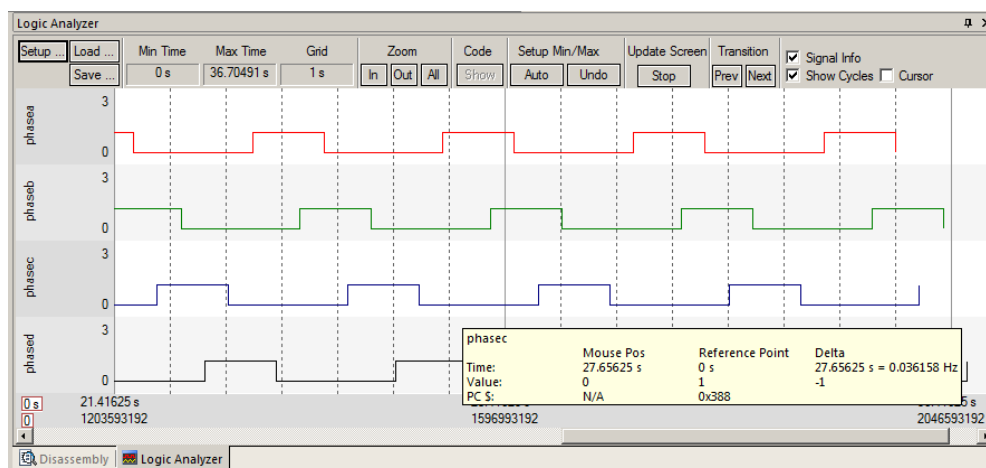
```

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. Click on the Blinky.c tab. Block `phasea`, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
10. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release.
11. Repeat for `phaseb`, `phasec` and `phased`. These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.
12. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
13. Click on Close to go back to the LA window.
14. Using the OUT and In buttons set the range to 1 or 2 seconds. Move the scrolling bar to the far right if needed.
15. You will see the following waveforms appear. 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 `phasec`:



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

## 10) 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  $\mu$ Vision).

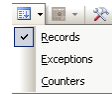
You have configured Serial Wire Viewer (SWV) in Section 6 under **RTX Viewer: Configuring the Serial Wire Viewer:**

Now we will examine some of the features available to you. SWV works with  $\mu$ Vision and a ULINK2, ULINK-ME, ULINKpro or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment 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 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. 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.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000028H	0000001H			107630769	1.68173077
ITM		31		06H			107631243	1.68173817
ITM		31		FFH			107644897	1.68195152
ITM		31		06H			112736793	1.76151239
ITM		31		FFH			112750297	1.76172339
ITM		31		03H			139616915	2.18151430
ITM		31		02H			139617499	2.18152342
Data Write			20000024H	00000000H			139630790	2.18173109
ITM		31		06H			139630965	2.18173383
ITM		31		FFH			139644633	2.18194739
ITM		31		06H			144736793	2.26151239
ITM		31		FFH			144750297	2.26172339
ITM		31		03H			171616793	2.68151239
ITM		31		04H			171617361	2.68152127
Data Write			2000002CH	00000001H			171630771	2.68173080
ITM		31		06H			171831245	2.68173820
ITM		31		FFH			171844899	2.68195155
ITM		31		06H			176736793	2.76151239
ITM		31		FFH			176750297	2.76172339
ITM		31		04H			203616915	3.18151430

5. Unselect EXCTRC and Periodic.

6. Select On Data R/W Sample.

7. Click on OK to return.

8. Click on the RUN icon.

9. Double-click anywhere in the Trace records window to clear it.

10. Only Data Writes will appear now.


**TIP:** You could have right clicked on the Trace Records window to filter the ITM frames out.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000028H	00000000H	00400794H	×	1483631864	23.18174787
Data Write			20000030H	00000001H	004007E4H	×	1515630769	23.68173077
Data Write			2000002CH	00000000H	004007C9H	×	1547631844	24.18174756
Data Write			20000024H	00000001H	0040074EH	×	1579630779	24.68173082
Data Write			20000030H	00000000H	00400794H	×	1611630786	25.18173103
Data Write			20000028H	00000001H	00400780H	×	1643630769	25.68173077
Data Write			20000024H	00000000H	00400762H	×	1675631859	26.18174780
Data Write			2000002CH	00000001H	004007B2H	×	1707630771	26.68173080
Data Write			20000028H	00000000H	00400794H	×	1739631839	27.18174748
Data Write			20000030H	00000001H	004007E4H	×	1771630769	27.68173077
Data Write			2000002CH	00000000H	004007C9H	×	1803630786	28.18173103
Data Write			20000024H	00000001H	0040074EH	×	1835630776	28.68173088
Data Write			20000030H	00000000H	00400794H	×	1867630783	29.18173088

### What is happening here ?

- When variables are entered in the Logic Analyzer (remember phasea through phased ?), the reads and/or writes will also appear in Trace Records.
- The Address column shows where the four variables are located.
- The Data column displays the data values written to phasea through phased.
- 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.
- 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. 

Name	Address	Type
Peripheral Registers		
Blinky		Application
Runtime Library		
Blinky		Module
mut_SLED	0x20000200	array[3] of uint
phasea	0x20000024	uint
phaseb	0x20000028	uint
phasec	0x2000002C	uint
phased	0x20000030	uint
clock	0x2000001C	uint
t_lcd	0x20000020	uint

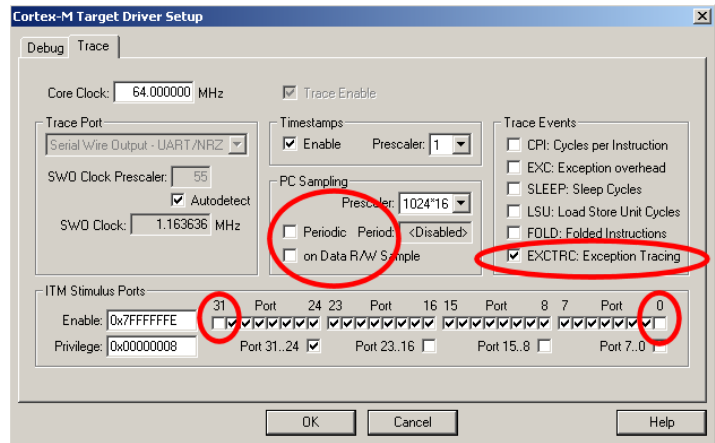
**Note:** You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

**TIP:** ULINKpro, and J-Link adapters display the trace frames in a different style trace window.

## 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.



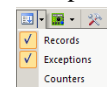
### What Is Happening ?

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 Return	X	0				X	1163664507	18.18225792
Exception Entry		11					1163674947	18.18242105
Exception Exit		11					1163675072	18.18242300
Exception Return		0				X	1163680402	18.18250628
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 Entry		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.

1. Select View/Trace/Exceptions or click on the Trace icon and select Exceptions.
2. The next window opens up and more information about the exceptions is displayed as shown.
3. Note the number of times these have happened. This is very useful information in case interrupts come too fast or slow.
4. ExtIRQ are the peripheral interrupts.
5. You can clear this trace window by double-clicking on it.
6. All this information is displayed in real-time and without stealing CPU cycles !



Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCcall	211	338.486 us	1.611 us	16.292 us	55.597 us	559.492 ms	0.97641921	26.59914124
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	2564	14.045 ms	4.056 us	7.597 us	9.932 ms	9.996 ms	0.98642844	26.61642836
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	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 64<sup>th</sup> 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:

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

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:

```

154: /* This function is called when the user timer has expired. Parameter */
155: /* 'info' holds the value, defined when the timer was created. */
156:
157: /* HERE: include optional user code to be executed on timeout. */
0x004005E0 BF00 NOP
->0x004005E2 E7FE B 0x004005E2
158: }
159:

```

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 µVision.

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.

The screenshot shows the IDE with the Disassembly window open to a function named `task void phaseD (void)`. The assembly code includes instructions like `MOVVS r0, #0x00`, `LDR r1, [pc, #480] ; @0x004009A8`, and `STR r0, [r1, #0x00]`. The Trace Records window is overlaid on top, showing a list of PC samples. The last entry in the trace records shows a different PC value: `00400F72H`, which corresponds to the `STR` instruction in the disassembly window.

## 11) ITM (Instruction Trace Macrocell)


Recall that we showed 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  $\mu$ Vision for display in the Debug (*printf*) Viewer window.

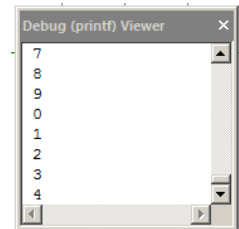
1. Open the project Blinky.uvproj (not RTX Blinky).
2. Add this code to Blinky.c. A good place is near line 16.

```
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. Remove any variables in the Logic Analyzer. Select Setup and Kill All.
10. Click on View/Serial Windows and select Debug (*printf*) Viewer and click on RUN.
11. Right click on the Debug window and deselect Settings/Add CR to LF. Note other settings.
12. In the Debug (*printf*) Viewer you will see the ASCII value of count appear. 



### Trace Records

1. Open the Trace Records if not already open. Double click on it to clear it.
2. You will see a window such as the one below with 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. Now you will see only the ITM writes.

**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. It is found in the header `core.CM3.h`.

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

## Part C) Using the ULINKpro with ETM Trace:

Contact Keil technical support for more information on ETM support with Kinetis processors.

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 Instruction Trace. More complete filtering is available.
- 2) The SWV (Serial Wire Viewer) data is sent out the SWO pin with the ULINK2 using UART encoding. The ULINKpro can send SWV data either out the 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 hence no Trace Port. The Trace Port is found on the 20 pin Hi-density connector and is configured in the Trace configuration window. ETM data is always sent out the Trace Port and if ETM is being used, SWV data is also 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 trace which provides records of all Program Counter values. ULINK2 provides only PC Samples and is not nearly as useful.
- 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 the ULINKpro ETM Trace:

### Background:

The ULINKpro was configured for SWV operation using the SWO pin and Manchester encoding on page 12. 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. One version is printed here: This used to configure the ETM and also the I/O port pins. This file is provided with MDK 4.21 and later.

```
/*
/*****
/* TracePort.ini: Initialization Script for Kinetis TracePort
/*****
FUNC void SetupTrace (void) {
    // SIM_SCGC5: enable PORT A clock
    _wDWORD(0x40048038, (_RDWORD(0x40048038) | 0x00000200));

    _wDWORD(0x40049018, 0x00000740); // PORTA_PCR6 -> (ALT7 + DSE) TRACE_CLKOUT
    _wDWORD(0x4004901C, 0x00000740); // PORTA_PCR7 -> (ALT7 + DSE) TRACE_D3
    _wDWORD(0x40049020, 0x00000740); // PORTA_PCR8 -> (ALT7 + DSE) TRACE_D2
    _wDWORD(0x40049024, 0x00000740); // PORTA_PCR9 -> (ALT7 + DSE) TRACE_D1
    _wDWORD(0x40049028, 0x00000740); // PORTA_PCR10 -> (ALT7 + DSE) TRACE_D0
}
SetupTrace();
*/
```




This file will be entered in the Initialization File: box in Target Options and with the Debug tab selected.

This file will be included as part of the MDK toolkit. It will have a ini file extension. Its complete name will be something like TracePort.ini. This file is specific to the Kinetis processors as the GPIO ports are configured.

This ini file will be executed every time you enter Debug mode. A RESET negates it. You must cycle through Debug mode.

The next page describes how to configure ETM.

## Configuring SWV and ETM Trace:

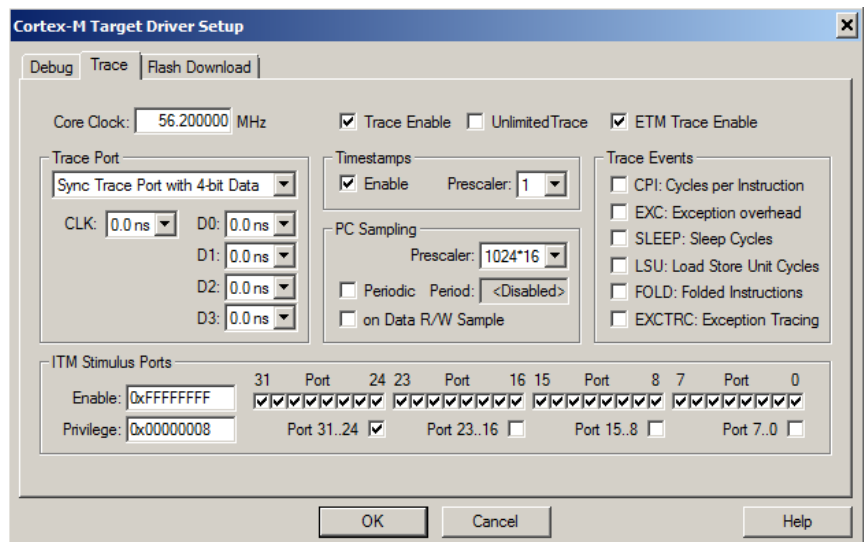
- 1) Connect the ULINK<sub>pro</sub> to the K40 or K60 Kinetis target.
- 2) Select the Kinetis project at C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky\Blinky.proj
- 3) Configure ULINK<sub>pro</sub> for the Kinetis processor as described on page 7: **4) Configuring ULINK<sub>pro</sub> and  $\mu$ Vision:** Do not forget to configure the Flash programmer as well.
- 4)  $\mu$ Vision must be stopped and in edit mode (not debug mode).
- 5) Select Options for Target  or ALT-F7 and select the Debug tab.
- 6) In the box Initialization File: an ini file should be visible. If not, you must enter an appropriate file. It will be in the Keil example files and an example is on the previous page. This file is called TracePort.ini and configures the GPIO port for ETM operation as well as the CoreSight ETM registers. You can use the Browse icon to select this file.
- 7) Click on the Edit box.  The specified ini file will open. You can examine it. Do not change it.
- 8) Click OK.
- 9) Select Options for Target  or ALT-F7 and select the Debug tab (again).
- 10) Click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side of the window.
- 11) Click on the Trace tab. The window below is displayed.
- 12) Core Clock: ULINK<sub>pro</sub> calculates this from the Trace Port.  $\mu$ Vision uses this to display timing values so it is a good idea to insert a valid clock speed.
- 13) In Trace Port select Sync Trace Port with 4 bit data. It is best to use the widest size.
- 14) Select CLK: 0.0 ns clock delay. D0 through D3 should be set to zero also.
- 15) Select Trace Enable and ETM Trace Enable. Unselect Periodic and EXCTRC and leave everything else at default as shown below.
- 16) Click on OK twice to return to the main  $\mu$ Vision menu. Both ETM and SWV are now configured.
- 17) 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<sub>pro</sub> 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<sub>pro</sub> 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.

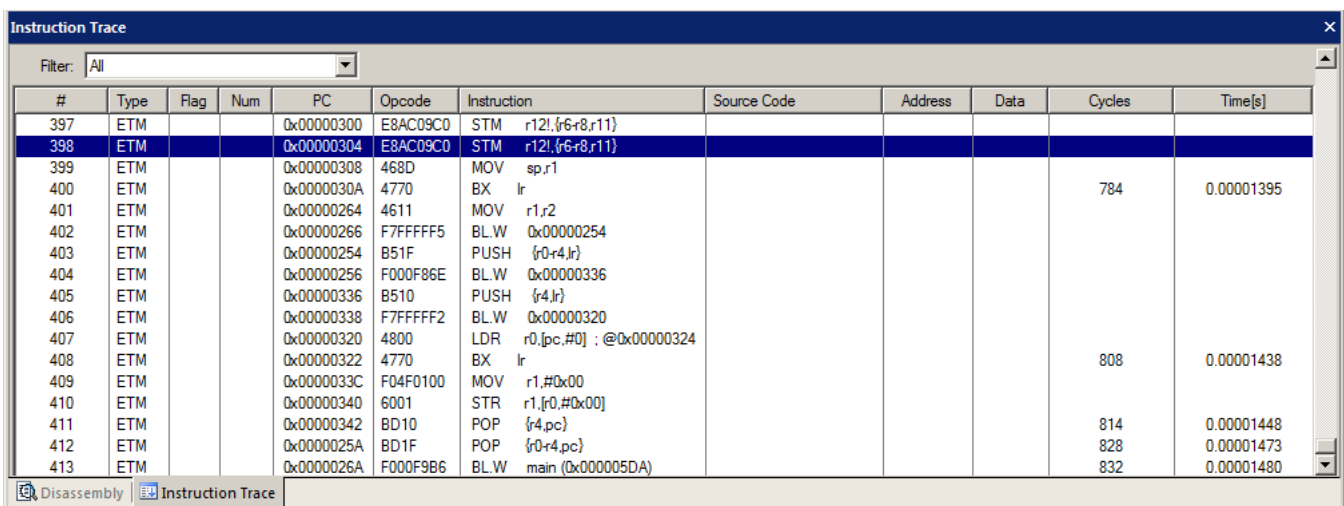


## 2) Blinky Example: ETM Frames from RESET and beyond:

The project in C:\Keil\ARM\Boards\Freescale\TWR-K60N512\Blinky (or for the K40 board) has now been modified on the previous page to provide ETM Trace and all the features it provides using a ULINKpro.

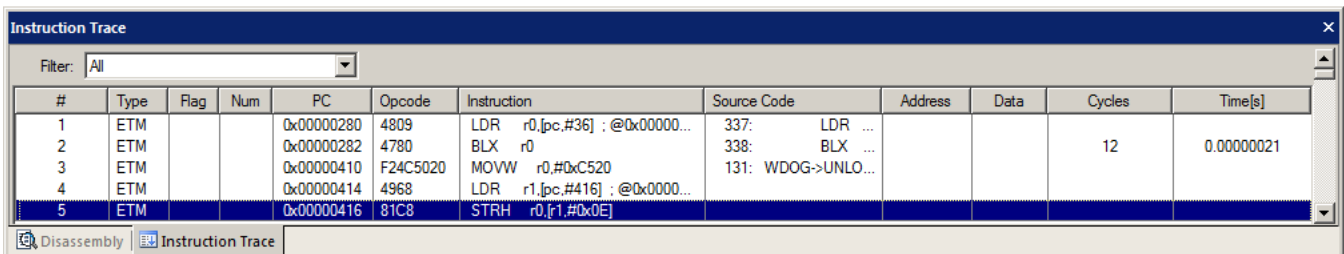
**Note:** The example on this page will work only with MDK 4.21 and later. ETM works in 4.20 but does not capture the first few hundred instructions for the Kinetis 1<sup>st</sup> silicon. 4.21 works perfectly.

1. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
2. Program 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 !!!
5. Examine the Instruction Trace window as shown below: This is a complete record of all the program flow since RESET until  $\mu$ Vision halted the program at the start of main() since Run To main is selected in  $\mu$ Vision.
6. In this case, # 413 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 (0x0800\_05DA in my case). Click on Single Step once. 



#	Type	Flag	Num	PC	Opcod	Instruction	Source Code	Address	Data	Cycles	Time[s]
397	ETM			0x00000300	E8AC09C0	STM r12, {r6-r8, r11}					
398	ETM			0x00000304	E8AC09C0	STM r12, {r6-r8, r11}					
399	ETM			0x00000308	468D	MOV sp, r1					
400	ETM			0x0000030A	4770	BX lr				784	0.00001395
401	ETM			0x00000264	4611	MOV r1, r2					
402	ETM			0x00000266	F7FFFFFF5	BL.W 0x00000254					
403	ETM			0x00000254	B51F	PUSH {r0-r4, lr}					
404	ETM			0x00000256	F000F86E	BL.W 0x00000336					
405	ETM			0x00000336	B510	PUSH {r4, lr}					
406	ETM			0x00000338	F7FFFFFF2	BL.W 0x00000320					
407	ETM			0x00000320	4800	LDR r0, [pc, #0] : @0x00000324					
408	ETM			0x00000322	4770	BX lr				808	0.00001438
409	ETM			0x0000033C	F04F0100	MOV r1, #0x00					
410	ETM			0x00000340	6001	STR r1, [r0, #0x00]					
411	ETM			0x00000342	BD10	POP {r4, pc}				814	0.00001448
412	ETM			0x0000025A	BD1F	POP {r0+r4, pc}				828	0.00001473
413	ETM			0x0000026A	F000F9B6	BL.W main (0x000005DA)				832	0.00001480



7. The instruction MOV will display: `0x000005DA | F04F34FF | MOV r4, #0xFFFFFFFF | 76: int num = -1;`
8. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.
9. 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\_0280 + 1 (+1 says it is a Thumb instruction). This is shown below:










#	Type	Flag	Num	PC	Opcod	Instruction	Source Code	Address	Data	Cycles	Time[s]
1	ETM			0x00000280	4809	LDR r0, [pc, #36] : @0x0000...	337: LDR ...				
2	ETM			0x00000282	4780	BLX r0	338: BLX ...			12	0.00000021
3	ETM			0x00000410	F24C5020	MOVW r0, #0xC520	131: WDOG->UNLO...				
4	ETM			0x00000414	4968	LDR r1, [pc, #416] : @0x0000...					
5	ETM			0x00000416	81C8	STRH r0, [r1, #0x0E]					

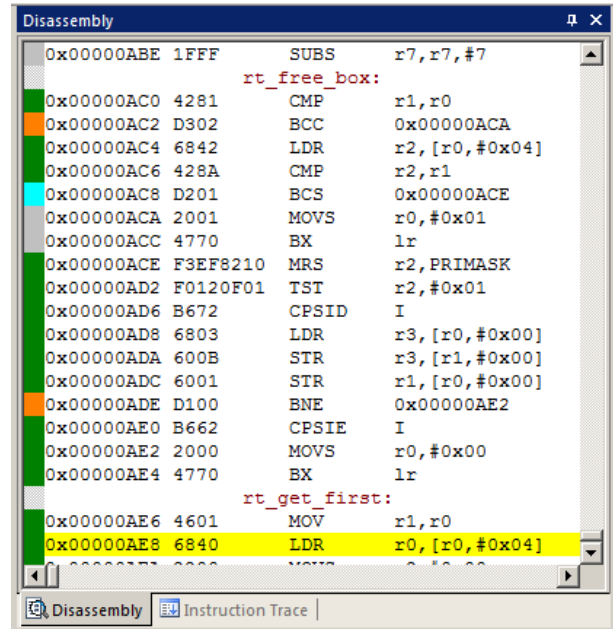
ETM trace provides a powerful tool for finding nasty bugs not easily found any other way. See page 29 for problems ETM and Serial Wire Viewer can help solve.

### 3) Code Coverage:

10. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
11. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin:
12. 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.

- |   |   |
|---|---|
| <br><br><br><br><br><br> | <ol style="list-style-type: none"> <li>1. Green: this assembly instruction was executed.</li> <li>2. Gray: this assembly instruction was not executed.</li> <li>3. Orange: a Branch is always not taken.</li> <li>4. Cyan: a Branch is always taken.</li> <li>5. Light Gray: there is no assembly instruction at this point.</li> <li>6. RED: Breakpoint is set here.</li> <li>7. Next instruction to be executed.</li> </ol> |
|---|---|



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.

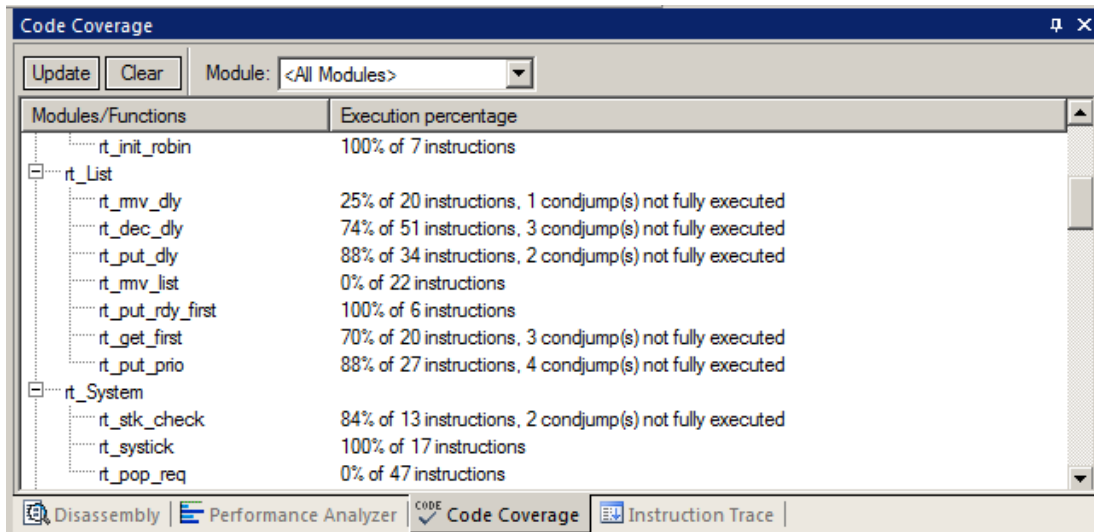
Why was 0x0000\_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

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 cannot be 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.

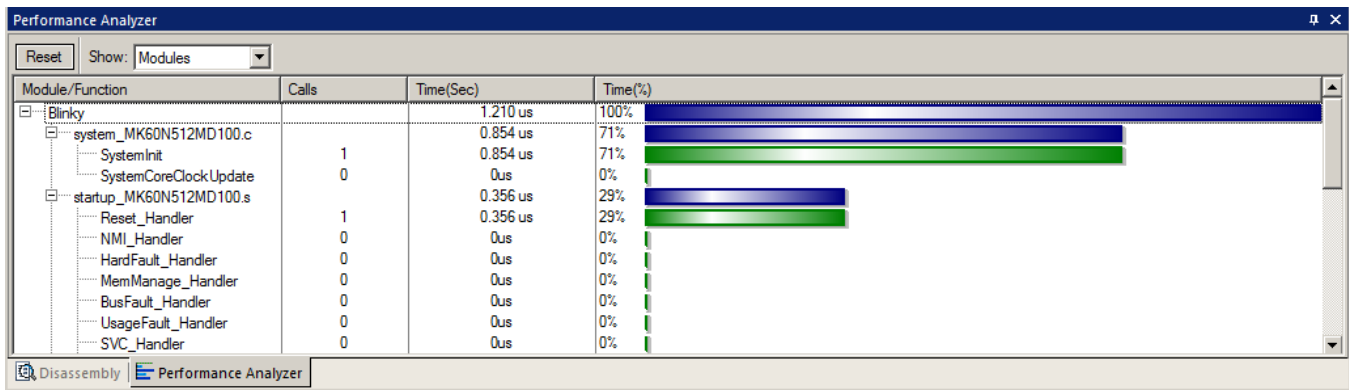



#### 4) Performance Analysis (PA):

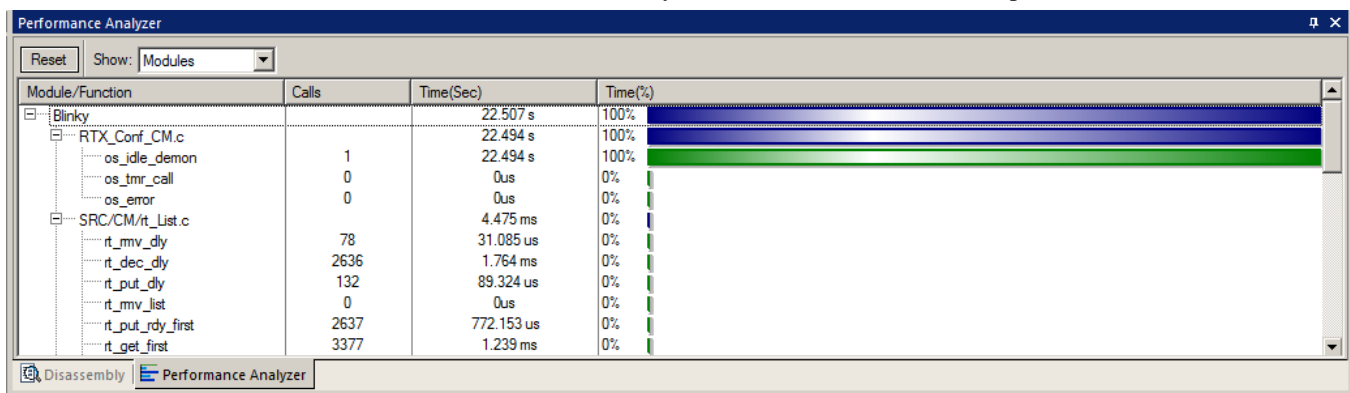
Performance Analysis tells you how much time was spent in each function. The data can be provided by either the SWV PC Samples or the ETM. SWV does not provide complete coverage results. With PC Samples, accuracy is not very good. More accuracy is improved with longer runs. Small loops could be entirely missed. ETM provides complete Performance Analysis. Keil provides only ETM PA.

Keil provides Performance Analysis with the  $\mu$ Vision simulator or with ETM and the ULINK $pro$ . SWV PA is not offered. 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.

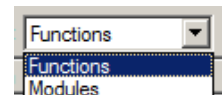
1. Use the same setup as used with Code Coverage. RTX\_Blinky.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it.  This clears the PA window and resets the Kinetis processor and reruns it to main() as before. Note: you need MDK 4.21 to display this initial window.
4. Expand some of the module names as shown below.
5. Execution information that has been collected in this initial short run. Times and number of calls is displayed.
6. We can tell that most of the time at this point in the program has been spent in the system and initialization routines.




7. Click on the RUN icon. 
8. 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 os\_idle\_demon.



9. Select Functions from the pull down box as shown here and notice the difference.
10. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.
11. When you are done, exit Debug mode.



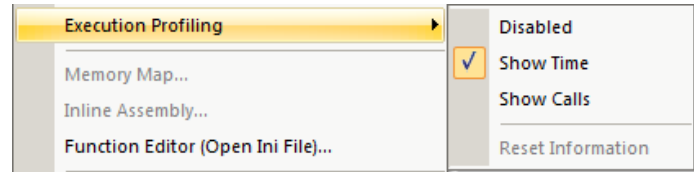
**TIP:** You can also click on the RESET icon  but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

When you click on RESET, the Initialization File .ini will no longer be in effect and this might cause SWV and/or ITM to stop working. Exiting and re-entering Debug mode executes the .ini script again.

## 5) 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. It is possible to group source lines (called collapse) to get combined times and number of calls. This is called Outlining. The  $\mu$ 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 more information appears as in the yellow box here:

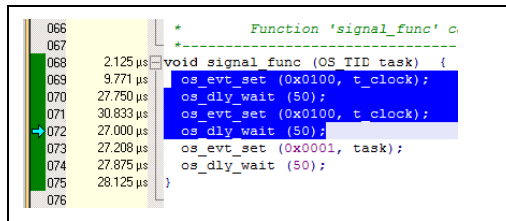


Time:	Calls:	Average:
19.599 s	139910257 *	0.140 $\mu$ s

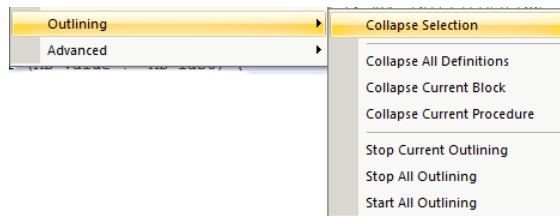
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.

## Outlining:

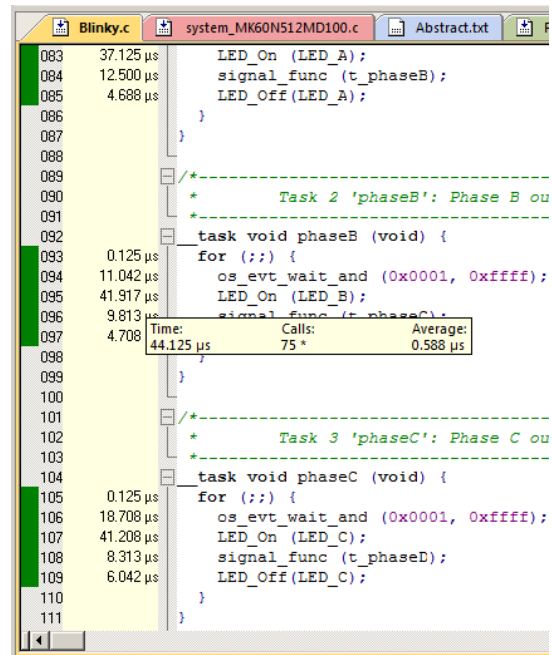
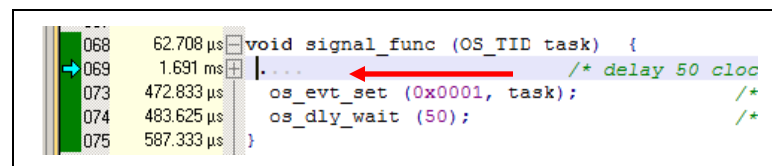
- 1) Block a section of source as similar to this:



- 2) Right click on the blue block and select Outlining and then Collapse Section as shown below:



- 3) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
- 4) Click on the + to expand it.
- 5) Stop the program and exit Debug mode.



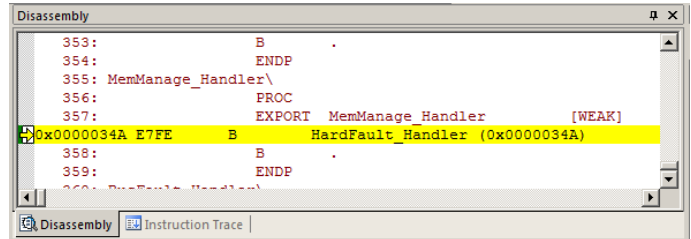
**6) In-the-Weeds Example:** Note: This example works best with MDK 4.21 or later.

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

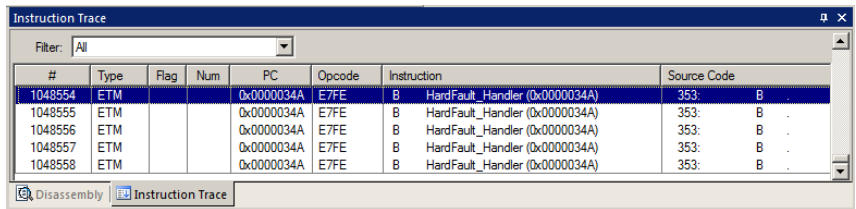
If a Bus Fault occurs in our example, the CPU will end up at 0x800\_043A as shown in the disassembly window below. This is the Bus Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. The Instruction Trace window below shows this branch forever. This is not useful.

This exception vector is found in the file startup\_MK60N512MD100.s. If we set a breakpoint by double-clicking on the Hard Fault handler and run the program: at the next Bus Fault event the CPU will again jump to the Hard Fault 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 of the crash.



- Using the Blinky example from the previous exercise, exit and re-enter Debug mode to clear the trace.
- Locate the Hard fault vector near line 353 in the disassembly window or in startup\_MK60N512MD100.s.
- Set a breakpoint at this point. A red block will appear.
- Run the Blinky example for a few seconds and click on STOP.
- In the Disassembly window, scroll down until you find a POP instruction. I found one at 0x0000\_03FA.
- Right click on the POP instruction and select Set Program Counter. This will be the next instruction executed.
- Click on RUN and immediately the program will stop on the Hard Fault exception branch instruction.
- Examine the Instruction Trace window and you find this POP plus everything else that was previously executed.



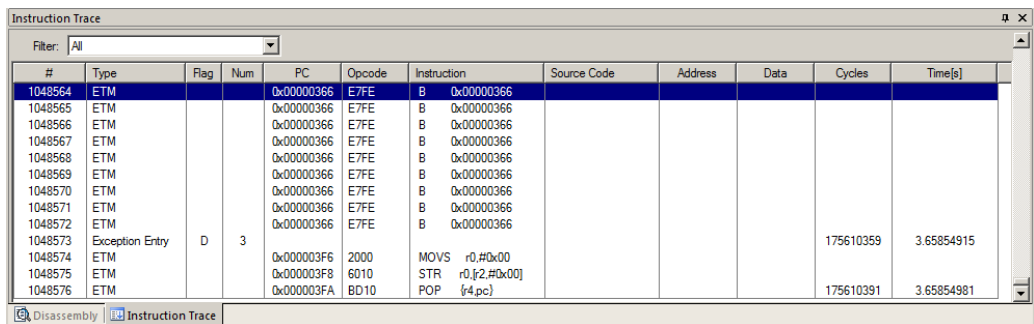
**How this was done:**

To create the hard fault, a POP instruction was executed out of order. A breakpoint was set on the Hard Fault Handler location 0x000\_034A. The program was run and stopped. A POP was located by scrolling down through the disassembly window. The PC was set to the POP at this location by right clicking on it and selecting Set PC. Click on RUN and the CPU immediately goes to the Hard Fault Handler and stops because the stack had a non-valid return PC address to be popped.

**TIP:** You might have to do a step-out-of to clear out all other running interrupt routines running otherwise you will just return from an interrupt rather than crash. See the µVision Call Stack window for information on interrupts running.

The frames above the POP are a record of all previous instructions executed and tells you the complete program flow.

The Exception Entry is the Hard Fault vector number 3.



It shows at a different frame number because this frame comes from the Serial Wire Viewer and not the ETM program trace. You would expect his frame to occur just after the POP instruction. It is normal for the SWV and ETM to not always be in synchronization.

## 7) Serial Wire Viewer Summary:

### Serial Wire Viewer can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers – DMA bypasses CPU and SWV by definition.

### Serial Wire Viewer displays in various ways:

- PC Samples.
- Data reads and writes.
- Exception and interrupt events.
- CPU counters.
- Timestamps for these.

### Trace is good for:

- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened*.
- Trace can often find nasty problems very quickly.
- Weeks or months can be replaced by minutes.
- Especially 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).
- Plus - don't have to stop the program. Crucial to some.

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

- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), corrupted stack.  
*How did I get here ?*
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. **ETM trace with the ULINK<sup>pro</sup> is best for this.**
- Communication protocol and timing issues. System timing problems.

For complete information on CoreSight for the Cortex-M4: Search for **DDI0314F\_coresight\_component\_trm.pdf** on [www.arm.com](http://www.arm.com).

**8) Keil Products:** See [www.keil.com/freescale](http://www.keil.com/freescale)

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

- MDK-Freescale™ For all Kinetis Cortex-M4, 1 year term license. RTX included. - \$745
- MDK-Standard™ (with included RTX RTOS with source code) - \$4,895
- MDK-Basic™ (256K compiler limit, No debug limit) RTX is not included - \$2,695
- MDK-Lite™ (Evaluation version) 32K Code and Data Limit - \$0

A Keil ULINK must be purchased or you can use the OS-JTAG on the Kinetis Tower board. For Serial Wire Viewer or ETM support, a ULINK2 or ULINK<sub>pro</sub> is needed. OS-JTAG does not support this debug technology.

**Note: USA prices. Contact [sales.intl@keil.com](mailto:sales.intl@keil.com)** for pricing in other countries.

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

Call Keil Sales for special university pricing.

For the ARM University program: go to [www.arm.com](http://www.arm.com) and search for university. Email: [university@arm.com](mailto: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. Is the full version of RTX – it is not restricted or crippled.
- No royalties are required. Very easy to use.
- RTX source code is included. Full version – is not restricted.
- Kernel Awareness visibility integral to  $\mu$ Vision. Updated using Serial Wire Viewer therefore steals no CPU cycles.

**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.
- ULINK<sub>pro</sub> - \$1,395 – Cortex-Mx SWV & ETM trace



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

For the entire Keil catalog see [www.keil.com](http://www.keil.com) or contact Keil or your local distributor.

For Linux, Android and bare metal (no OS) support on Freescale ARM9™ and Cortex-A series processors, please see the ARM DS-5™ toolkit at [www.arm.com/ds5/](http://www.arm.com/ds5/)

---

**For more information:**

**Keil Sales** In North and South America: [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. Outside the US: [sales.intl@keil.com](mailto:sales.intl@keil.com)

**Keil Technical Support** in USA: [support.us@keil.com](mailto:support.us@keil.com) or 800-348-8051. Outside the US: [support.intl@keil.com](mailto:support.intl@keil.com).

For comments or corrections please email [bob.boys@arm.com](mailto:bob.boys@arm.com).

For the latest version of this document, contact the author, Keil Technical support or [www.keil.com/freescale](http://www.keil.com/freescale).

