

**Introduction:** *For the latest version of this document and files:* [www.keil.com/appnotes/docs/apnt\\_232.asp](http://www.keil.com/appnotes/docs/apnt_232.asp)

The purpose of this lab is to introduce you to the Freescale Kinetis Cortex™-M0+ processor using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will demonstrate all debugging features available on this processor including Micro Trace Buffer (MTB). At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. See [www.keil.com/freescale](http://www.keil.com/freescale) for more labs, appnotes and more information including MQX support in MDK.

Keil MDK supports and has examples for most Freescale ARM processors. Check the Keil Device Database® on [www.keil.com/dd](http://www.keil.com/dd) for the complete list which is also included in MDK: In µVision, select Project/Select Device for target...

**Linux:** ARM processors running Linux, Android and bare metal are supported by ARM DS-5™. [www.arm.com/ds5](http://www.arm.com/ds5).

Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into the unrestricted commercial version. MDK-Freescale is a one year renewable license for Kinetis processors and costs US\$ 745 including technical support.

**Middleware:** MDK Professional contains middleware libraries including TCP/IP stack, CAN drivers, a Flash file system and USB drivers. Contact Keil sales for information regarding middleware for your processor. <http://www.keil.com/arm/mdk.asp>.

**RTX RTOS:** All variants of MDK contain the full version of RTX with Source Code. See [www.keil.com/rl-arm/kernel.asp](http://www.keil.com/rl-arm/kernel.asp).

## Why Use Keil MDK ?

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

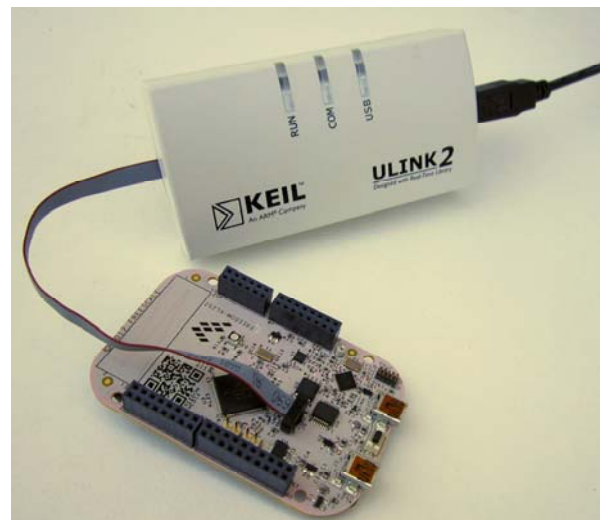
1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler and assembler toolchain. MDK is a complete turn-key tool solution.
2. A full feature Keil RTOS called RTX is included with MDK. RTX comes with a BSD type license. Source code is provided. See [www.arm.com/cmsis](http://www.arm.com/cmsis).
3. All applicable ARM debugging technology is supported.
4. RTX Kernel Awareness window. It is updated in real-time.
5. Available Debug Adapters: ULINK™2, ULINK-ME, ULINK<sup>pro</sup> and OpenSDA (it is CMSIS-DAP compliant).
6. Kernel Awareness is available for Keil RTX and Freescale MQX. Many other RTOSs are compatible with MDK.
7. Keil Technical Support is included for one year and is easily renewable. This helps your project get completed faster.
8. MDK includes board support for Kinetis Cortex-M0+ and Cortex-M4 processors on Tower and Freedom boards.

**This document includes details on these features plus more:**

1. Micro Trace Buffer (MTB). Instruction trace. A history where your program has been.
2. Real-time Read and Write to memory locations for Watch, Memory. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
3. Two Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
4. RTX and RTX Tasks window: a kernel awareness program for RTX that updates while your program is running.
5. A DSP example program using ARM CMSIS-DSP libraries.

## Micro Trace Buffer (MTB):

MDK supports MTB with OpenSDA (CMSIS-DAP) or ULINK2/ME and ULINK<sup>pro</sup>. MTB provides instruction trace which is essential for solving program flow and other related problems. How to use MTB is described in this document.



### Keil ULINK2 connected to Freedom J6 JTAG.

The USB cable provides power to the Freedom.  
Keil also works with OpenSDA. (no ULINK is needed)

## Index:

1. Freescale Evaluation Boards & Keil Evaluation Software:	3
2. Keil Software Installation:	3
3. CoreSight Definitions:	4
4. CMSIS: Cortex Microcontroller Software Interface Standard	4
5. Debug Adapter Summary for Keil MDK with $\mu$ Vision IDE:	4

### Using the OpenSDA CMSIS-DAP Debug Adapter:

6. Configuring OpenSDA for $\mu$ Vision:	5
7. <i>Blinky</i> example using the Freedom KL25Z and OpenSDA:	7
8. Hardware Breakpoints:	7
9. Call Stack & Locals window:	8
10. Watch and Memory windows and how to use them:	9
11. How to view Local Variables in Watch and Memory windows:	10
12. Watchpoints: Conditional Breakpoints:	11
13. MTB: Micro Trace Buffer:	12
14. Exploring the MTB:	13
15. Trace “In the Weeds” Example:	14
16. RTX_Blinky: Keil RTX RTOS example:	15
17. RTX Kernel Awareness using RTX Viewer:	16
18. Configuring the MTB Trace:	17
19. DSP Sine Example using ARM CMSIS-DSP Libraries	18
20. Creating your own project from scratch:	20
21. KL25 Cortex-M0+ Trace Summary:	22
22. Useful Documents:	22

### Using Keil ULINK Debug Adapters:

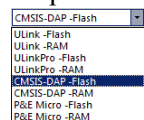
23. Configuring a Keil ULINK2 or ULINK-ME:	23
24. Configuring a Keil ULINK <i>pro</i> :	24
25. Keil Products and contact information:	25

### Using this document:

1. The latest version of this document and the necessary example source files are available here:  
[www.keil.com/appnotes/docs/apnt\\_232.asp](http://www.keil.com/appnotes/docs/apnt_232.asp)
2. You do not need any debug adapters: just the Freedom board, a USB cable and MDK installed on your PC. Configuring OpenSDA starts on page 5. If using OpenSDA, you **must** complete Step 1 on page 5 to program the OpenSDA processor U6 with CMSIS-DAP.S19 or maybe later the hex file DAP32.SDA.
3. OpenSDA (CMSIS-DAP) is used by default in this document. A ULINK2, ULINK-ME or a ULINK*pro* can be used. Most examples provided have Target Options for each of these in RAM and Flash operation.

**The first exercise starts on page 7.** You can go directly there if you are using OpenSDA (CMSIS-DAP).

For any ULINK you must select the correct adapter in the Target selector box as shown here: It is easy to add these targets (shown here): See step 2 on page 5 for instructions on creating your own target configurations.



## 1) Freescale Evaluation Boards & Keil Evaluation Software:

Keil provides board support for Kinetis Cortex-M0+ and Cortex-M4 processors. They include KwikStik, Tower K20, K40, K53, K60, K70 and **KL25Z (both Tower and Freedom boards)**. For Vybrid and i.MX series see ARM DS-5.

**Example Programs:** Keil provides various example programs. See C:\Keil\ARM\Boards\Freescale\ for the board support files. FRDM-KL25Z is for the Freedom KL25 board and XTWR-KL25Z48M is for the Tower version of the KL25.

\RL consists of Flash File examples. Such middleware is a component of MDK Professional. To run these examples a full license is needed. Please contact Keil sales for a temporary license if you want to evaluate Keil middleware.

**MDK** supports Freescale MQX and includes awareness windows. See [www.keil.com/freescale](http://www.keil.com/freescale) for an informative video.

**JTAG Connector J6:** The production version of the Freedom board might not include all hardware features such as various connectors. To use a ULINK adapter you will need to obtain a Samtec 2x5 connector FTSH-105-01-F-D (add -K for shroud). It is easily soldered on J6. Any ULINK will now connect and operate. Otherwise use OpenSDA with a USB cable.

## 2) Keil Software Installation:

This document was written for Keil MDK 4.60 or later which contains  $\mu$ Vision 4. The evaluation copy of MDK (MDK-Lite) is available free on the Keil website. Do not confuse  $\mu$ Vision4 with MDK 4.0. The number “4” is a coincidence.

MDK 4.60 is the current official Keil release.

To obtain a copy of MDK go to [www.keil.com/arm](http://www.keil.com/arm) and select the “Download” icon located on the right side.

Install MDK into the default directory. You can install MDK into any directory, but references in this lab are made to C:\.

### MDK 4.60 has:

- Complete OpenSDA (CMSIS-DAP) support for MTB trace.
- MTB Trace works with OpenSDA, ULINK2, ULINK-ME and ULINK<sub>pro</sub>.
- Example files (but not the DSP example) for the Freedom and Tower KL25Z boards.
- You will still need to get CMSIS-DAP.S19 or DAP32.SDA and the DSP example file from the web: [www.keil.com/apnotes/docs/apnt\\_232.asp](http://www.keil.com/apnotes/docs/apnt_232.asp)

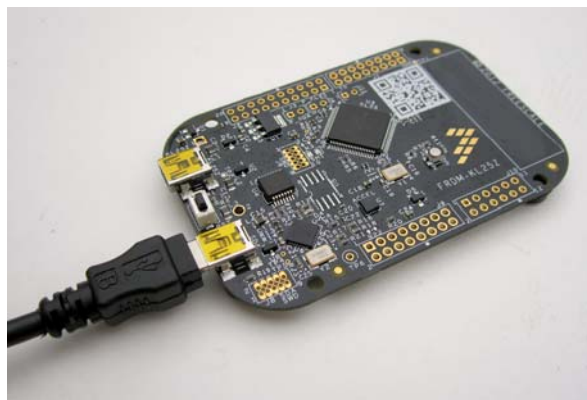
You can use the evaluation version of MDK-Lite and OpenSDA (CMSIS-DAP) or a ULINK2, ULINK-ME, ULINK<sub>pro</sub> for this lab. Keil supports P&E OSJTAG but this was not tested at this time. OpenSDA is a good choice to use.

**OpenSDA:** OpenSDA is Freescale’s name for ARM’s CMSIS-DAP. Essentially this is an ARM standard that specifies an on-board debug adapter. The Freedom board incorporates CMSIS-DAP.

You do not need an external debugger such as a ULINK2 to do the examples in this lab. Just the Freedom or Tower board and a USB cable is needed as pictured below:

The Freescale Vybrid board (Cortex-A5 + Cortex-M4) also has CMSIS-DAP to use with ARM DS-5 toolchain. Vybrid and i.MX (including i.MX6) support is available today. See [www.arm.com/ds5](http://www.arm.com/ds5)

The Freedom production board connected to run OpenSDA with Keil  $\mu$ Vision:  
No external debug adapter is needed for this lab.



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

**Note:** The KL25Z Cortex-M0+ options are highlighted in red below: Kinetis Cortex-M4 processors have all except MTB.

- JTAG: Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
- **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except for no Boundary Scan. SWD is referenced as SW in the  $\mu$ Vision Cortex-M Target Driver Setup. See page 6, 2<sup>nd</sup> screen. The SWJ box must be selected. The KL25 processors use SWD exclusively. There is no JTAG on the KL25.
- SWV: Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf. SWV must use SWD because of the TDIO conflict described in **SWO** below.
- **DAP:** Debug Access Port. A component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention.  $\mu$ Vision uses the DAP to update memory, watch and RTOS kernel awareness windows in real-time while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed in your sources. You do not need to configure or activate DAP.  $\mu$ Vision does this automatically when you select the function.
- SWO: Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
- Trace Port: A 4 bit port that ULINK $pro$  uses to collect ETM frames and optionally SWV (rather than the SWO pin).
- ETM: Embedded Trace Macrocell: Provides all the program counter values. Only ULINK $pro$  provides ETM.
- **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. Only on KL25 Cortex-M0+ processors. Kinetis Cortex-M4 processors provide ETM trace.

### 4) CMSIS: Cortex Microcontroller Software Interface Standard

ARM CMSIS-DSP libraries are currently offered for all Cortex-M0, Cortex-M3 and Cortex-M4 processors.

CMSIS-RTOS provides standard APIs for RTOSs. RTX is a free RTOS available from ARM as part of CMSIS Version 3.0.

Freescale example software is CMSIS hardware abstraction layer compliant.

CMSIS-DAP is a new ARM standard for on-board debugging adapters. An external adapter such as a ULINK2 is not needed for most debugging applications. The Freescale Kinetis KL25Z Freedom board supports CMSIS-DAP. The ARM/Keil component of Freescale OpenSDA is CMSIS-DAP compliant. SDA is an acronym for “Serial Debug Adapter”.

See [www.arm.com/cmsis](http://www.arm.com/cmsis) and [www.onarm.com/cmsis/download/](http://www.onarm.com/cmsis/download/) for more information on these standards.

### 5) Debug Adapter Summary for Keil MDK with $\mu$ Vision IDE:

**CMSIS-DAP:** Freescale’s implementation is called OpenSDA. The debug adapter is a K20 ARM Cortex-M4 processor incorporated on the Freedom and Tower KL25Z boards. It connects to your PC with a standard USB cable. No external hardware is needed. A hex file needs to be programmed into the K20 Flash. This file is CMSIS-DAP.S19 or DAP32.SDA. Instructions are provided on the next page.

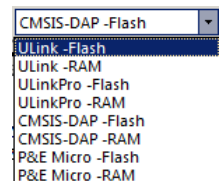
Provides run control debugging, Flash and RAM programming, Watchpoints, hardware breakpoints, and DAP reads and writes in Watch and memory windows updated in real-time as well as MTB trace. RTX System kernel awareness for the ARM RTX RTOS is provided. All ULINK devices provide these features plus more depending on the processor involved.

**ULINK2:** Pictured on page 1. This is a hardware JTAG/SWD debugger. It connects to various connectors found on boards populated with ARM processors. Provides all the features of CMSIS-DAP. With Kinetis Cortex-M4 processors, ULINK2 adds Serial Wire Viewer (SWV). See the Kinetis lab on [www.keil.com/freescale](http://www.keil.com/freescale) for examples using SWV and ETM.

**ULINK-ME:** Pictured on page 23. ULINK-ME is provided only combined with a board package from Keil or an OEM. Electrically and functionally it is very similar to a ULINK2. With Keil  $\mu$ Vision, they are used as equivalents.

**ULINK $pro$ :** Pictured on Page 24. ULINK $pro$  is Keil’s most advanced debug adapter. With Freescale Kinetis Cortex-M4 processors, ULINK $pro$  provides Serial Wire Viewer (SWV) and adds ETM instruction trace. Code Coverage, Performance Analysis and Execution Profiling are then provided using ETM. ULINK $pro$  provides the fastest Flash programming speeds.

**P&E Micro:** Keil  $\mu$ Vision supports P&E OpenSDA. OSJTAG drivers V1.06 or later need to be installed in  $\mu$ Vision. See [www.keil.com/download/docs/408.asp](http://www.keil.com/download/docs/408.asp) for the file [FslKinetisDriversV106.exe](#). P&E was not tested for this document but will be done in a later version. The current P&E SDA file for the K20 Flash is DEBUG-APP-V004.SDA.





## 6) Using a CMSIS-DAP Debug adapter: Configuring OpenSDA for µVision:

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

The JTAG connector J6 is used to program and control U3 KL25. The other JTAG connector J8 labelled SDA (Serial Debug Adapter) is used to program the on-board K20. You will not need J8 for this lab.

This document will use OpenSDA. Target connection by µVision will be via a standard USB cable connected to J7. The on-board Kinetis K20 acts as the debug adapter. J6 JTAG is not used. Micro Trace Buffer (MTB) frames will be displayed.

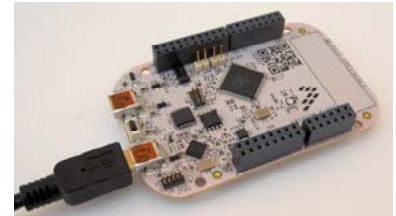
You can also use a ULINK2, ULINK-ME or ULINK<sub>pro</sub> with suitable adjustments. See the Page 23 for instructions.

### **Step 1 MUST be done ! at least once...**

These instructions tell how to connect µVision to the target KL25 processor CoreSight debugging mechanism via USB. The on-board K20 U6 will now be programmed to act as a debug adapter for SWD.

#### **Step 1) Program the K20 with the CMSIS-DAP app CMSIS-DAP.S19:**

1. Hold RESET button SW1 down and connect a USB cable to J7 as shown here:
2. When you hear the USB dual-tone, release RESET.
3. The green led D4 will blink about once per second. The K20 is now ready to be programmed with the application.
4. The K20 will act as a USB mass storage device connected to your PC called BOOTLOADER. It contains an empty file READY.TXT and maybe a few more.
5. Locate the file CMSIS-DAP.S19 using Windows Explorer. CMSIS-DAP.S19 is not located in MDK 4.60. It is part of the software that comes with this document. This must be copied into the K20 Flash with the next step.
6. Copy and paste or drag and drop CMSIS-DAP.S19 into the K20 device directory and visible as shown:
7. Cycle the power to the Freedom board. (this step might get deprecated). The green led will blink once and then stay on.



FSL\_WEB.HTM  
TOOLS.HTM

LASTSTAT.TXT  
CMSIS-DAP.S19


**Note:** The file CMSIS-DAP.S19 might get replaced by DAP32.SDA. Refer to [www.keil.com/appnotes/docs/apnt\\_232.asp](http://www.keil.com/appnotes/docs/apnt_232.asp)

**TIP:** The green led, after accessed once by the µVision Flash programming tool or by entering Debug mode once, will indicate that µVision is in Debug mode and connected to the KL25 debug port SWD. Remember, JTAG is not used.



**TIP:** This application will remain in the K20 Flash each time the board power is cycled with RESET off. The next time board is powered with RESET held on, it will be erased. CMSIS-DAP.S19 is the CMSIS application in Motorola S record format that loads and runs on the K20 OpenSDA. DAP32.SDA is a hex file that does the same thing.

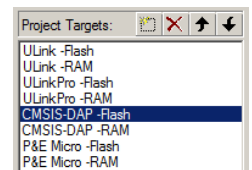
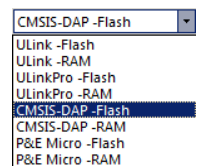
#### **Step 2) (Optional) Create a new Target configuration:**

Note: Steps 2 through 4 are already configured in the Keil examples provided. This is for reference should you want to create new target options for your own custom projects.



Selecting Target Options  opens up a set of extensive configuration items for µVision. It is possible to have multiple copies of Target Options with different settings. Each is selected with the Target Options pulldown menu. For instance, each, as in this example, can point to different debug adapter selections.

To setup a new debug adapter, you can modify an existing Target or create a new one:

1. Select a Target Option you want to be the template for your new entry.
2. Select Projects/Manage/Components... The Target Option you selected will be highlighted in Project Targets.
3. In the Project Targets box, select Insert . Enter the name of the target you want to create and press Enter and OK.
4. Select the Target Option you just created in the drop down window.
5. Select Target Options  and modify the contents as desired as described on the next page.
6. These will be saved in your Target Options configuration.
7. When completed, select File/Save All.



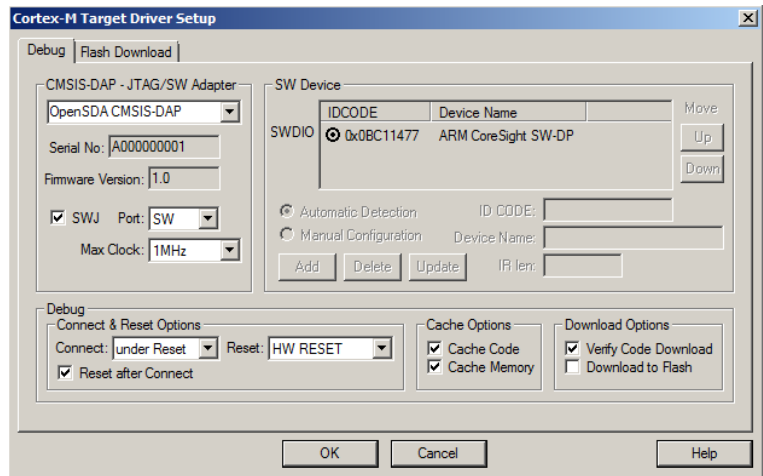
### Step 3) Configure µVision for OpenSDA and CMSIS-DAP:

1. Most of the examples are pre-configured to use various debug adapters including OpenSDA (CMSIS-DAP). These are accessible in the Target Options menu shown below in Step 3. These steps show you how to manage these.
2. Start µVision  if not already running. Select the project you want to use for your board.
3. Select Target Options  or ALT-F7 and select the Debug tab. In the drop-down menu box select CMSIS-DAP



Debugger as shown here:

8. Select Settings and the next window below opens up. This is the control panel for debug control.
9. In Port: select SW. Select the SWJ box.  
Note: The KL25 has only SWD (SW).
10. 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 it is blank: this **must** be fixed before you can continue. Check the target power supply. Reprogram the K20 with CMSIS-DAP.S19. No number in the Serial No: box means µVision is unable to connect to the K20. This is dreadful news and you must fix it before you can continue.



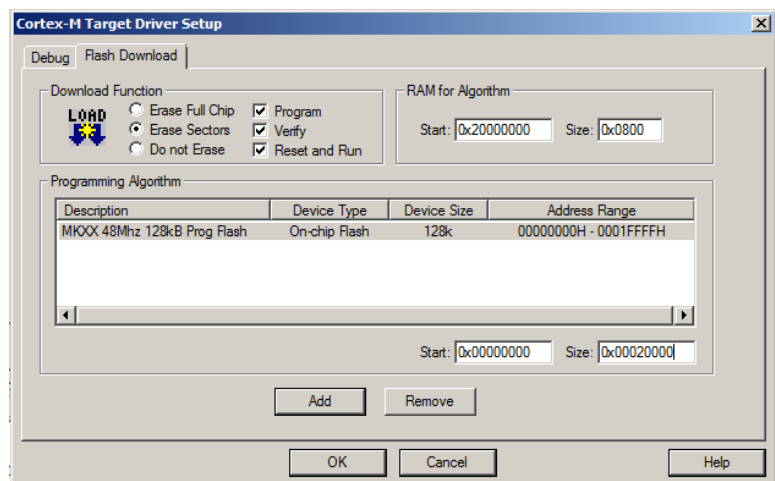
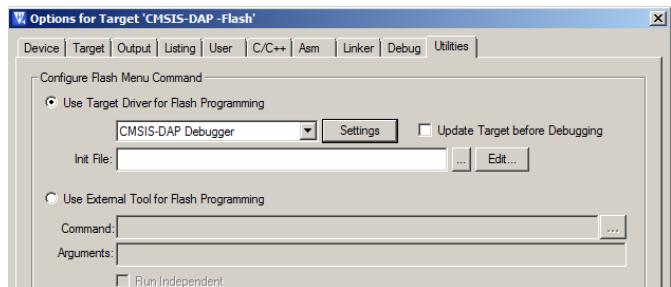
**TIP:** To refresh this screen, change the option in Port: to JTAG and then back to SW or click OK once to leave this screen and then re-enter it.

### Step 4) Configure the Keil Flash Programmer:

1. Click on OK once and then select the Utilities tab.
2. Select CMSIS-DAP Debugger as shown here:
3. Click Settings to select the programming algorithm.
4. Select Add and select the appropriate Kinetis Flash if necessary as shown below:
5. MKXX 48 MHz 128kB Prog Flash is the correct one to use with the Freedom board.
6. Click on OK once.


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

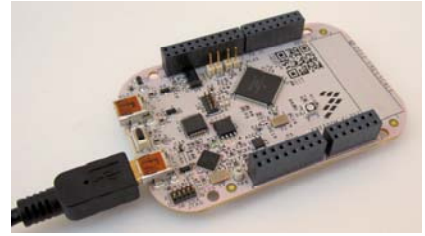
7. Click on OK once more to return to the µVision main screen.

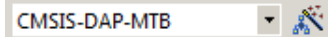


To use a ULINK2, ULINK-ME or ULINKpro, please see pages 23 and 24.






## 7) Blinky example program using the Freescale Freedom KL25Z and OpenSDA:

1. Now we will connect a Keil MDK development system using the Freedom board and OpenSDA as the debug adapter. OpenSDA is Freescale's implementation of CMSIS-DAP. Your board **must** have the application CMSIS-DAP.S19 programmed into the OpenSDA processor before you can continue. See Step 1 on page 5.
2. Connect a USB cable between your PC and Freedom USB J7 as shown here:
3. Start  $\mu$ Vision by clicking on its desktop icon. 
4. Select Project/Open Project.
5. Open the file:  
C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z\Blinky\_MTB\Blinky.uvproj.
6. Select "CMSIS-DAP-MTB" in the Select Target menu.





This is where you create and select different target configurations such as to execute a program in RAM or Flash. This Target selection is pre-configured to use OpenSDA which is ARM CMSIS-DAP compliant.

7. Compile the source files by clicking on the Rebuild icon.  You can also use the Build icon beside it.
8. Program the KL25Z flash by clicking on the Load icon:  Progress will be indicated in the Output Window.  
**TIP:** If you get an error this probably means the Freedom board is not programmed with CMSIS-DAP.S19. Refer to page 4 Step 1) for instructions.
9. 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 if it is chosen.
10. Click on the RUN icon.  Note: you stop the program with the STOP icon. 


### ***The three colour LED D3 on the Freedom board will now blink in sequence.***

Now you know how to compile a program, program it into the KL25Z processor Flash, run it and stop it !

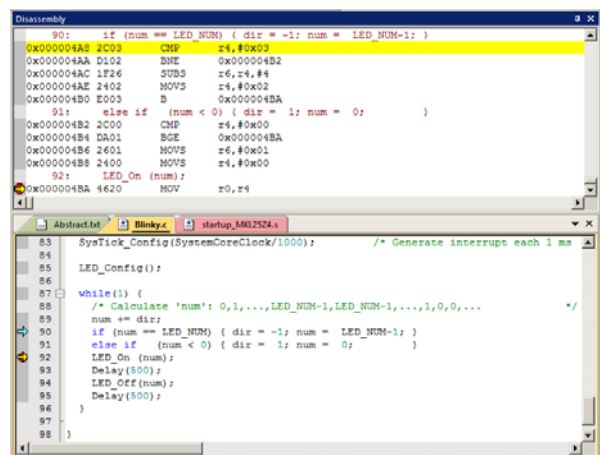
**Note:** The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

## 8) Hardware Breakpoints:

The KL25Z has two hardware breakpoints that can be set or unset on the fly while the program is running.

1. With Blinky running, in the Blinky.c window, click on a darker block in the left margin in either the disassembly or an appropriate source window. Inside the while loop inside the main() function is best near lines 89 to 95.
2. A red circle will appear and the program will presently stop.
3. Note the breakpoint is displayed in both the disassembly and source windows as shown here:
4. Every time you click on the RUN icon  the program will run until the breakpoint is again encountered.
5. Remove the breakpoint by clicking on it.
6. Clicking in the source window will indicate the appropriate code line is the Disassembly window and vice versa. This is relationship indicated by the cyan arrow as shown at Line 90:

**TIP:** A hardware breakpoint does **not** execute the instruction it is lands on. ARM CoreSight hardware breakpoints are no-skip. This is a rather important feature.



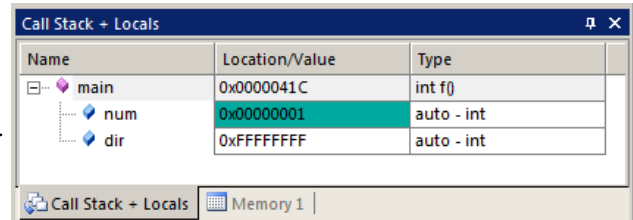
## 9) Call Stack + Locals Window:

### Local Variables:



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

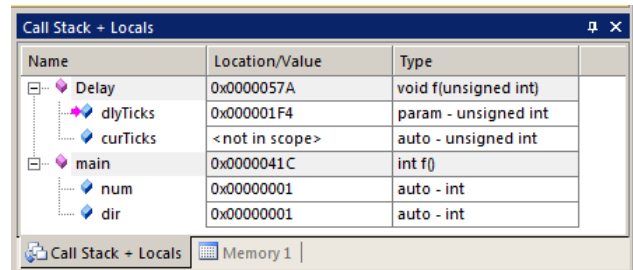
If possible, the values of the local variables will be displayed and if not the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

1. Set a breakpoint in the while loop in main(). Run Blinky. It will soon stop on this breakpoint.
2. Click on the Call Stack + Locals tab if necessary to open it.
3. Shown is the Call Stack + Locals window.
4. The contents of the local variables are displayed as well as function names.
5. In this example, two local variables num and dir are displayed in the window here with their values: →



Name	Location/Value	Type
main	0x0000041C	int f()
num	0x00000001	auto - int
dir	0xFFFFFFFF	auto - int

6. Click on the Step In icon or F11: 
7. Continue until the program enters the Delay function. The Call Stack + Locals window will now show this event:
8. Click on the StepOut icon  or CTRL-F11 to exit all function(s) to return to main().
9. **When you ready to continue, remove the hardware breakpoint by clicking on its red circle ! You can also type Ctrl-B and select Kill All.**




Name	Location/Value	Type
Delay	0x0000057A	void f(unsigned int)
dlyTicks	0x000001F4	param - unsigned int
curTicks	<not in scope>	auto - unsigned int
main	0x0000041C	int f()
num	0x00000001	auto - int
dir	0x00000001	auto - int

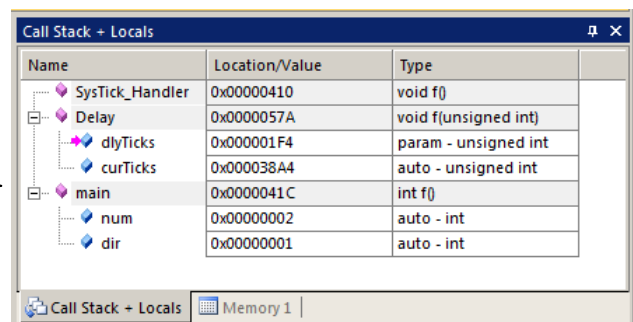
**TIP:** You can modify a variable value in the Call Stack & Locals window when the program is stopped.

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

### Call Stack:

The list of stacked functions is displayed when the program is stopped as you have seen. This is useful when you need to know which functions have been called and are stored on the stack. A breakpoint was set in the SysTick\_Handler function and this event is clearly shown at the top of this window: →

As you click on the StepOut icon  each function will be removed as it comes off the stack until you are left with only main().



Name	Location/Value	Type
SysTick_Handler	0x00000410	void f()
Delay	0x0000057A	void f(unsigned int)
dlyTicks	0x000001F4	param - unsigned int
curTicks	0x000038A4	auto - unsigned int
main	0x0000041C	int f()
num	0x00000002	auto - int
dir	0x00000001	auto - int

**Do not forget to remove the hardware breakpoints before continuing.**

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





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

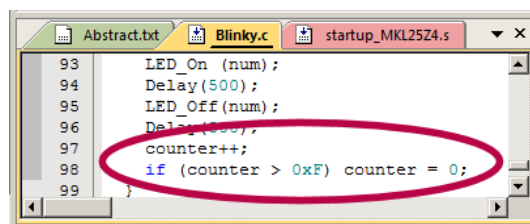
The Watch and Memory windows will display updated variable values in real-time. It does this using the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to “put” or insert values into the Memory window in real-time. It is possible to “drag and drop” variable names into windows or enter them manually. You can also right click on a variable and select Add *varname* to.. and select the appropriate window.





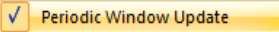

### Watch window:

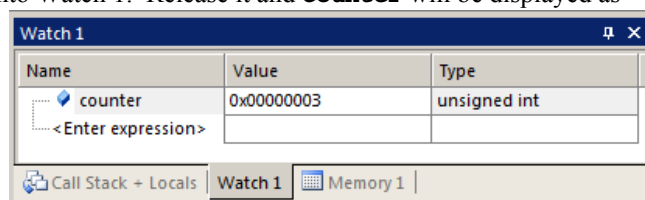
**Add a global variable:** Recall the Watch and Memory windows can’t see local variables unless stopped in their function.

1. Stop the processor  and exit debug mode. 
2. Declare a global variable (I called it counter) near line 18 in Blinky.c like this: **unsigned int counter = 0;**
3. Add the statements near Line 95 just after the second Delay(500);

```
counter++;  
if (counter > 0xF) counter = 0;
```



4. Click on Rebuild  and program the Flash with Load .
5. Enter Debug mode.  Click on RUN . You can configure a Watch window while the program is running. You can also do this with a Memory window.
6. Select View and select Periodic Window Update if necessary: 
7. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
8. In Blinky.c, block **counter**, click and hold and drag it into Watch 1. Release it and **counter** will be displayed as shown here: 
9. counter will update in real time.

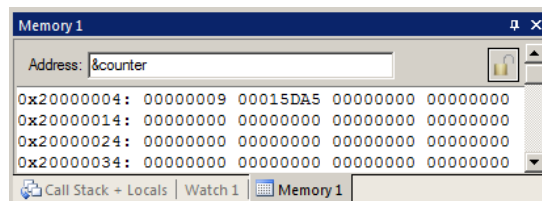


**TIP:** You can also right click on the variable name and select Add value to ... and select Watch 1. You can also enter a variable manually by double-clicking <Enter expression> or press F2 and use copy and paste or typing the variable name.

**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.

### Memory window:

1. Drag ‘n Drop **counter** into the Memory 1 window. Select View/Memory Windows if necessary.
2. Note the value of **counter** 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 not what we want to see at this time.
3. Add an ampersand “&” in front of the variable name and press Enter. The physical address is shown (0x2000\_0004).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **counter** is displayed as shown here:
6. Both the Watch and Memory windows are updated in real-time.
7. Right-click with the mouse cursor over the desired data field and select Modify Memory. You can change a memory or variable on-the-fly while the program is still running.




**TIP:** No CPU cycles are used to perform these operations. See the next page for an explanation how this works.



**TIP:** To view variables and their location use the Symbol window. Select View/Symbol Window while in Debug mode.

These Read and Write accesses are handled by the Serial Wire Debug (SWD) connection via the CoreSight Debug Access Port (DAP), which provides on-the-fly memory accesses.

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

1. Make sure Blinky.c is running. We will use the local variable **num**
2. Locate where the local variable **num** is declared in Blinky.c near line 79, at the start of the main() function.
3. Enter **num** into Watch 1 window by right clicking on it and selecting Add num to.... Note it says “not in scope” because  $\mu$ Vision cannot access the CPU registers while running which is where value is located. Stop the program and “not in scope” will probably still be displayed. (this depends on where your program stops)
4. Start the program  and set a breakpoint in the while loop in main(). The program will stop and a value for num will be displayed. The only time a value will be displayed is if the program happens to stop in the while loop in main() where num is in scope. Most of the time the program is executing the Delay function.

**TIP:** Remember: you can set and unset hardware breakpoints on-the-fly in the Cortex-M0+ while the program is running !

5.  $\mu$ Vision is unable to determine the value of **num** when the program is running because it exists only when main is running. It disappears in functions and handlers outside of main. num is a local or automatic variable and this means it is probably stored in a CPU register which  $\mu$ Vision is unable to access during run time.
6. Remove the breakpoint and make sure the program is not running . Exit Debug mode. 


### How to view local variables updated in real-time:


All you need to do is to make **num** static where it is declared in Blinky.c !





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

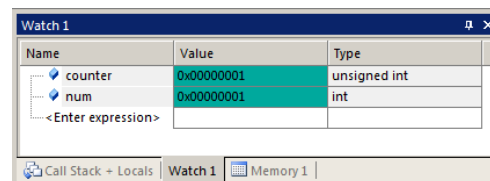
```
int main (void) {  
    static int num = -1;  
    int dir = 1;  
}
```

**TIP:** You can edit files in edit or debug mode. However, you can compile them only in edit mode.

2. Compile the source files by clicking on the Rebuild icon . They will compile with no errors or warnings.
3. 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 Target Options , select the Utilities tab and select the “Update Target before Debugging” box.

4. Enter Debug mode. 
5. Click on RUN. 
6. **num** is now updated in real-time. This is ARM CoreSight technology working.
7. **Recall you can modify num in the Memory window when the program is running.**
8. Stop the CPU and exit debug mode for the next step.  and 



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


### How It Works:

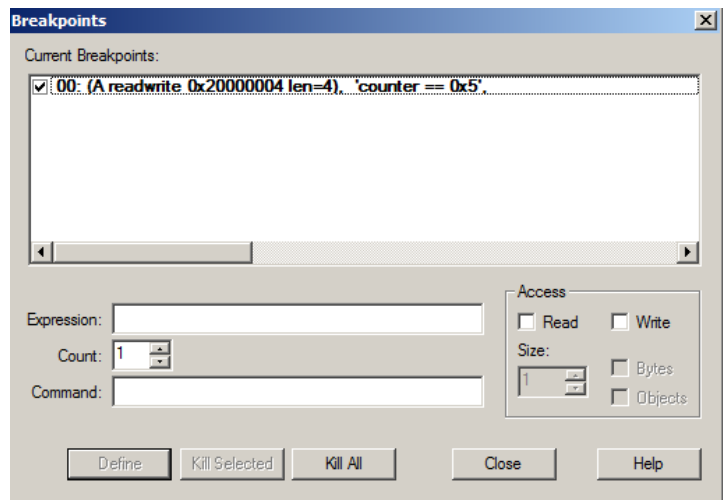
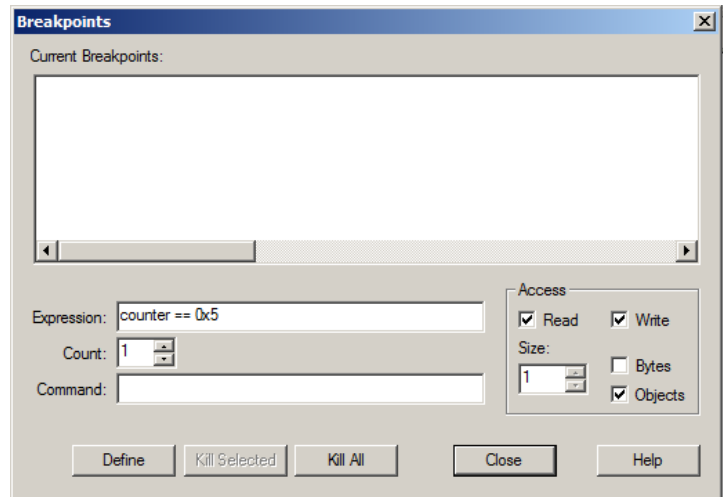
$\mu$ Vision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. Remember the Cortex-M3 is a Harvard architecture. This means it has separate instruction and data buses. 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  $\mu$ Vision 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.

## 12) Watchpoints: Conditional Breakpoints

The KL25 has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil literature.

1. Use the same Blinky configuration as the previous page. Stop the program if necessary. Stay in debug mode.
2. We will use the global variable `counter` you created in Blinky.c to explore Watchpoints.
3. Select Debug in the main  $\mu$ Vision window and select Breakpoints or press Ctrl-B.
4. In the Expression box enter: "`counter == 0x5`" without the quotes. Select both the Read and Write Access.
5. Click on Define and it will be accepted as shown below:
6. Click on Close.
7. Enter the variable `counter` to the Watch 1 window by dragging and dropping it if it is not already there.
8. Click on RUN. .
9. You will see `counter` change in the Watch window.
10. When `value` equals 0x5, the Watchpoint will stop the program.
11. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.
12. To repeat this exercise, change counter to something other than 0x05 in the Watch window and click on RUN.
13. Stay in Debug mode for the MTB trace example on the next page.
14. Leave the Watchpoint defined.



**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.

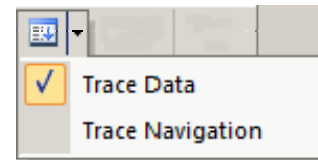
**TIP:** Raw addresses can also be entered into the Watchpoint. An example is: \*((unsigned long \*)0x20000004)

### 13) MTB: Micro Trace Buffer:


The Kinetis KL25 processor contains an instruction trace called MTB. The trace buffer is an area in the KL25 internal RAM. The trace frames are stored here. The size of this buffer is adjustable in the file DBG\_MTB.ini.

The project Blinky.uvproj found in the directory \Blinky\_MTB is pre-configured to use MTB. This exercise demonstrates the use of MTB.

1. Use the same program as used for the Watchpoint on the preceding page.
2. Open the Trace Data window by selecting View/Trace/Trace Data or using this icon:



3. A window similar to this will be visible: Size accordingly. Note the instructions with source displayed. This is a record of the last number of instructions executed by the processor.

4. Click on RUN  and allow the Watchpoint to stop the program at counter == 0x5 as before on the previous page.

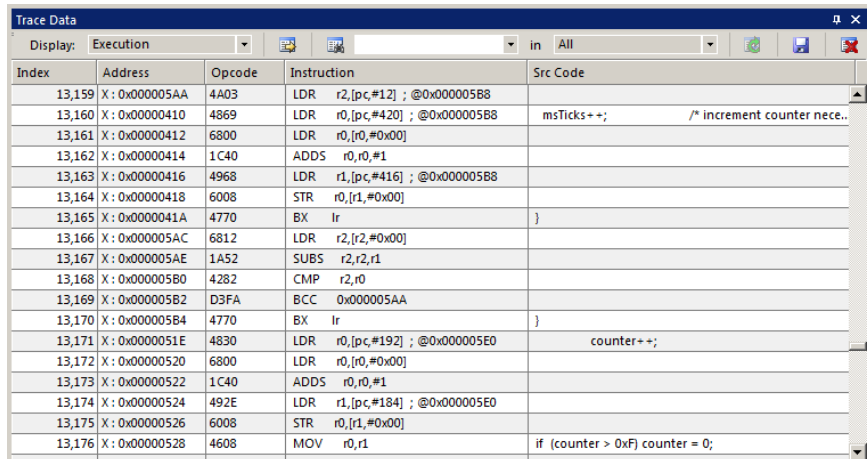
**TIP:** The trace data can be saved to a file.

5. The next Trace Data window will appear: Note the last instruction to be executed is a MOV r0,r1:
6. Double-click on this line to go to this spot in the Blinky.c and

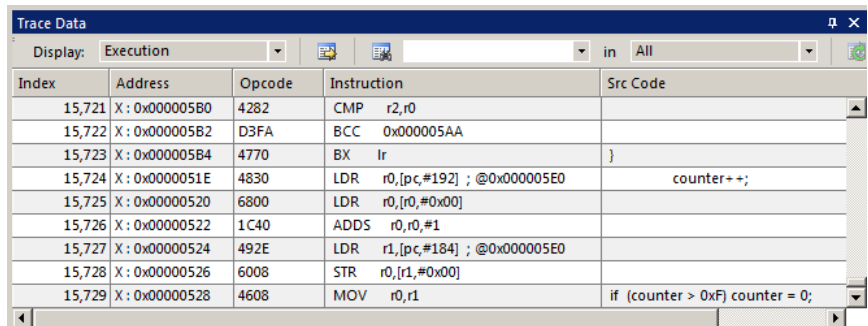
Disassembly window as shown:

**Information is as follows:**

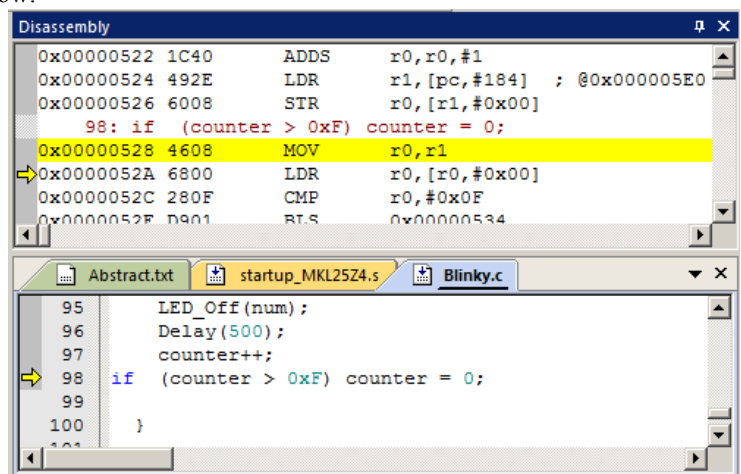
1. The yellow arrow is the next instruction to be executed is LDR at 0x52A part of C statement at Line 98. This is also the Program Counter value.
2. The yellow highlight is the last instruction executed (MOV r0, r1). This was selected by double-clicking on an instruction in the Trace data window.
3. The second to last instruction executed in STR r0, [r1, #0x00]. This is visible in both the Trace Data and Disassembly windows.
4. This STR is the write that caused the Watchpoint to stop the program. Note there is a small skid associated with a Watchpoint.
5. This is a simple example. If a branch, an asynchronous RTOS context switch or interrupt occurred, the Disassembly window would not provide obvious clues to such program flow changes. The MTB trace is clearly able to provide this type of valuable debugging information.
6. Remove the Watchpoint for the next exercise.



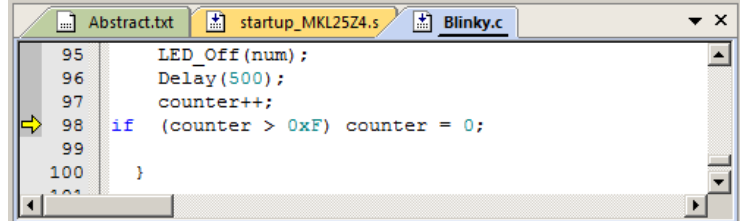
Index	Address	Opcode	Instruction	Src Code
13,159	X: 0x000005AA	4A03	LDR r2,[pc,#12] ; @0x000005B8	
13,160	X: 0x00000410	4869	LDR r0,[pc,#420] ; @0x000005B8	msTicks++; /* increment counter nece..
13,161	X: 0x00000412	6800	LDR r0,[r0,#0x00]	
13,162	X: 0x00000414	1C40	ADDS r0,r0,#1	
13,163	X: 0x00000416	4968	LDR r1,[pc,#416] ; @0x000005B8	
13,164	X: 0x00000418	6008	STR r0,[r1,#0x00]	
13,165	X: 0x0000041A	4770	BX lr	}
13,166	X: 0x000005AC	6812	LDR r2,[r2,#0x00]	
13,167	X: 0x000005AE	1A52	SUBS r2,r2,r1	
13,168	X: 0x000005B0	4282	CMP r2,r0	
13,169	X: 0x000005B2	D3FA	BCC 0x000005AA	
13,170	X: 0x000005B4	4770	BX lr	}
13,171	X: 0x0000051E	4830	LDR r0,[pc,#192] ; @0x000005E0	counter++;
13,172	X: 0x00000520	6800	LDR r0,[r0,#0x00]	
13,173	X: 0x00000522	1C40	ADDS r0,r0,#1	
13,174	X: 0x00000524	492E	LDR r1,[pc,#184] ; @0x000005E0	
13,175	X: 0x00000526	6008	STR r0,[r1,#0x00]	
13,176	X: 0x00000528	4608	MOV r0,r1	if (counter > 0xF) counter = 0;



Index	Address	Opcode	Instruction	Src Code
15,721	X: 0x000005B0	4282	CMP r2,r0	
15,722	X: 0x000005B2	D3FA	BCC 0x000005AA	
15,723	X: 0x000005B4	4770	BX lr	}
15,724	X: 0x0000051E	4830	LDR r0,[pc,#192] ; @0x000005E0	counter++;
15,725	X: 0x00000520	6800	LDR r0,[r0,#0x00]	
15,726	X: 0x00000522	1C40	ADDS r0,r0,#1	
15,727	X: 0x00000524	492E	LDR r1,[pc,#184] ; @0x000005E0	
15,728	X: 0x00000526	6008	STR r0,[r1,#0x00]	
15,729	X: 0x00000528	4608	MOV r0,r1	if (counter > 0xF) counter = 0;








Address	Opcode	Instruction
0x00000522	1C40	ADDS r0,r0,#1
0x00000524	492E	LDR r1,[pc,#184] ; @0x000005E0
0x00000526	6008	STR r0,[r1,#0x00]
98: 0x00000528	4608	MOV r0,r1
0x0000052A	6800	LDR r0,[r0,#0x00]
0x0000052C	280F	CMP r0,#0x0F
0x0000052E	D901	BTS 0x00000534

```
95 LED_Off(num);
96 Delay(500);
97 counter++;
98 if (counter > 0xF) counter = 0;
99
100 }
```

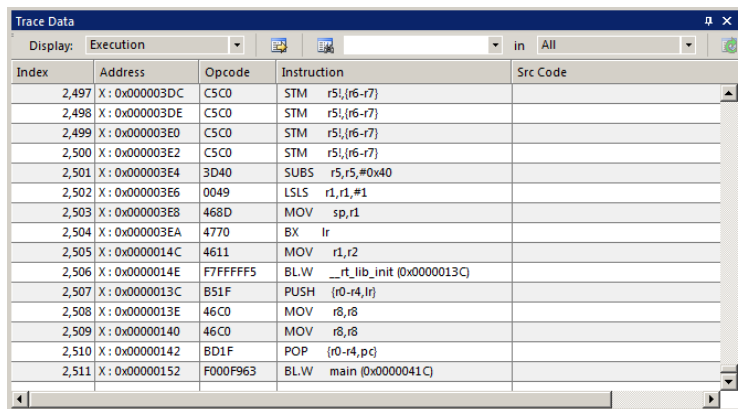
## 14) Exploring the MTB:

The MTB provides a record of instructions executed. No CPU cycles are stolen to accomplish this. MTB is part of CoreSight debugging technology.

1. Make sure all breakpoints and watchpoints are removed. Enter Ctrl-B or Debug/Breakpoints and select Kill All.
2. Exit Debug mode.  Select the Target Options icon .
3. Select the Debug tab and ensure Run to main() is selected. Click on OK.  
4. Enter Debug mode.  This has the effect of restarting your entire program and the processor from a fresh start. The program will run to main() as directed. The following Data Trace window will display:

### Note these items:

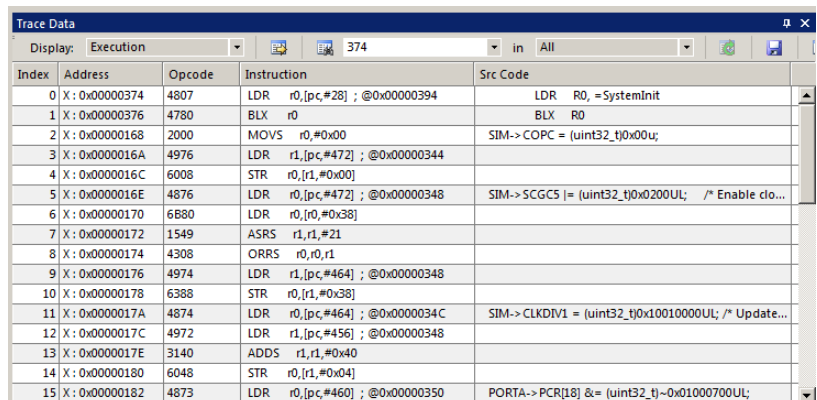
1. The last instruction executed was a branch: BL.W main(0x41C)
2. There are 2,511+ 1 instructions displayed in this Micro Trace Buffer.
3. The PC equals 0x41C which was set by the BL.W instruction.
4. Double-click on the POP at 2,510.
5. Examine this instruction in the Disassembly window. Note there is no clue this POP would result in the next instruction being BL.W. The MTB has faithfully recorded that it was.



Index	Address	Opcode	Instruction	Src Code
2,497	X: 0x000003DC	C5C0	STM r5!,r6-r7	
2,498	X: 0x000003DE	C5C0	STM r5!,r6-r7	
2,499	X: 0x000003E0	C5C0	STM r5!,r6-r7	
2,500	X: 0x000003E2	C5C0	STM r5!,r6-r7	
2,501	X: 0x000003E4	3D40	SUBS r5,r5,#0x40	
2,502	X: 0x000003E6	0049	LSLS r1,r1,#1	
2,503	X: 0x000003E8	468D	MOV sp,r1	
2,504	X: 0x000003EA	4770	BX lr	
2,505	X: 0x0000014C	4611	MOV r1,r2	
2,506	X: 0x0000014E	F7FFFFF5	BL.W __rt_lib_init (0x0000013C)	
2,507	X: 0x0000013C	B51F	PUSH {r0-r4,lr}	
2,508	X: 0x0000013E	46C0	MOV r8,r8	
2,509	X: 0x00000140	46C0	MOV r8,r8	
2,510	X: 0x00000142	BD1F	POP {r0-r4,pc}	
2,511	X: 0x00000152	F000F963	BL.W main (0x0000041C)	

### Examine the Start of Program:

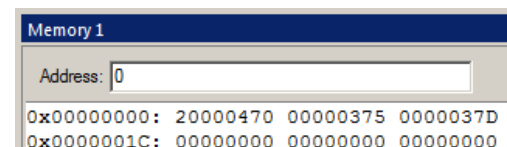
6. Scroll to the top of Trace Data. We want to look at the very first instruction executed but the MTB buffer has not captured this.
7. Double-click on one of the instructions near the top of the Trace Data window.
8. Set a breakpoint at this instruction in the Disassembly window.
9. Exit and reenter Debug mode. The processor will stop at this breakpoint. Scroll to the top of the Trace Data window.
10. Note the first instruction is at 0x374 and is a LDR instruction. This is the very first instruction executed.
11. Open a Memory window and enter address 0x0. Right click and select Unsigned Long. See the window below:
12. Note 0X0 is the Initial Stack Pointer (0x2000\_0470).
13. 0x4 is the initial PC and is 0x375. Bit 0 indicates Thumb2 instruction so subtract 1 and you get 0x374. This is the address of the first instruction in the Trace.
14. Click on Step (F11) and see these instructions executed and recorded in the Trace Data window.
15. Remove any breakpoints. Enter Ctrl-B and select Kill All.



Index	Address	Opcode	Instruction	Src Code
0	X: 0x00000374	4807	LDR r0,[pc,#28] ; @0x00000394	LDR R0,=Systeminit
1	X: 0x00000376	4780	BLX r0	BLX R0
2	X: 0x00000168	2000	MOVS r0,#0x00	SIM->COPC = (uint32_t)0x00u;
3	X: 0x0000016A	4976	LDR r1,[pc,#472] ; @0x00000344	
4	X: 0x0000016C	6008	STR r0,[r1,#0x00]	
5	X: 0x0000016E	4876	LDR r0,[pc,#472] ; @0x00000348	SIM->SCGCS  = (uint32_t)0x0200UL; /* Enable clo...
6	X: 0x00000170	6B80	LDR r0,[r0,#0x38]	
7	X: 0x00000172	1549	ASRS r1,r1,#21	
8	X: 0x00000174	4308	ORRS r0,r0,r1	
9	X: 0x00000176	4974	LDR r1,[pc,#464] ; @0x00000348	
10	X: 0x00000178	6388	STR r0,[r1,#0x38]	
11	X: 0x0000017A	4874	LDR r0,[pc,#464] ; @0x0000034C	SIM->CLKDIV1 = (uint32_t)0x10010000UL; /* Update...
12	X: 0x0000017C	4972	LDR r1,[pc,#456] ; @0x00000348	
13	X: 0x0000017E	3140	ADDS r1,r1,#0x40	
14	X: 0x00000180	6048	STR r0,[r1,#0x04]	
15	X: 0x00000182	4873	LDR r0,[pc,#460] ; @0x00000350	PORTA->PCR[18] &= (uint32_t)~0x1000700UL;

**TIP:** If Run to main() is not set, no instructions will be executed when Debug mode is entered. The PC will be at the first instruction. You can Step (F11) or RUN from this point and the Trace Data window will update as the instructions are executed.

**TIP:** These addresses will probably be different with different compiler options. The addresses shown were obtained with MDK 4.60 defaults.



Address	0
0x00000000:	20000470 00000375 0000037D
0x0000001C:	00000000 00000000 00000000

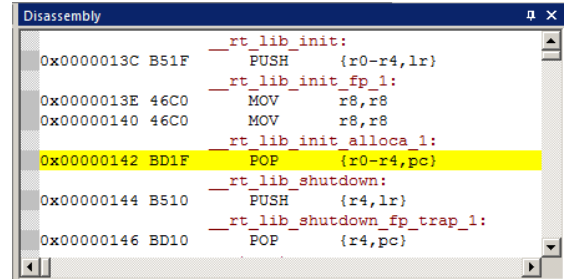


## 15) Trace “In the Weeds” Example





Perhaps the most useful use of trace is to show how your program got to an unexpected place. This can happen when normal program flow is disturbed by some error and it goes “into the weeds”. Finding these errors can be extremely challenging. A record of all the instructions executed prior to this usually catastrophic error are recorded to the limits of the trace buffer size.

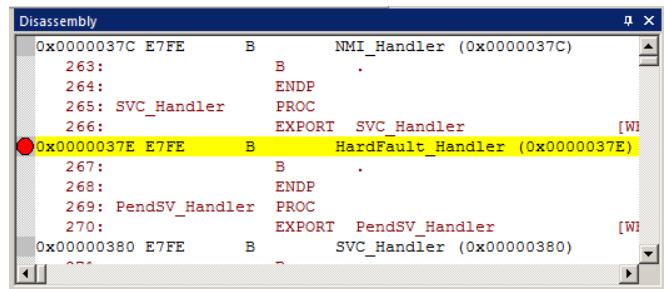
Remember the POP instruction shown in the trace on the last page? A good way to crash your program is to execute a POP out of order. This is because the stack does not contain a valid return address. Here is how to do this:

1. Run the Blinky\_MTB program and confirm the leds blink. This indicates Blinky is running normally.
2. Stop the program.
3. In the Disassembly window, right click anywhere and select Show Disassembly at Address... Enter 0x13C. This is the location of the PUSH instruction. This window displays:
4. Note the two MOV and POP instructions following it:
5. It will be more interesting in the trace to show these two instructions if you have them available or any other that will not change program flow. The preceding PUSH is not good for us to use at this time as it changes the program flow.
6. Right click on the POP, or better the first MOV and select Set Program Counter. What will happen is these two MOV instructions will be executed first and then the POP. Since the POP is executed without a valid PUSH, a hard fault error will result.
7. *We want to stop the program when a hard fault occurs. Otherwise a Hard fault handler which is normally a branch to itself will fill up the trace buffer.*
8. In the Disassembly window, find the Hard fault Handler. It is usually around address 0x37E as shown here. Set a breakpoint at this instruction as shown:

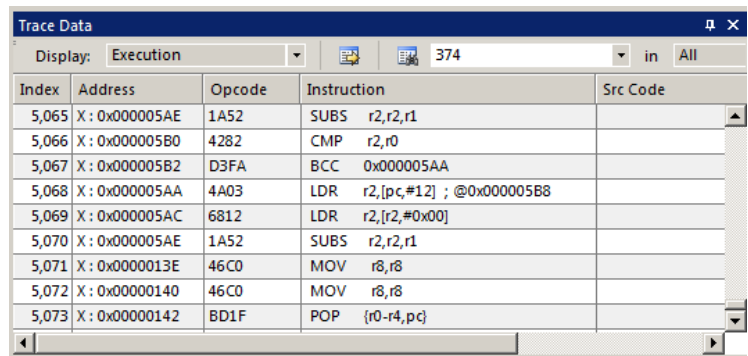


```
Disassembly
0x0000013C B51F  _rt_lib_init:
                PUSH    {r0-r4,lr}
0x0000013E 46C0  _rt_lib_init_fp_1:
                MOV     r8,r8
0x00000140 46C0  _rt_lib_init_fp_1:
                MOV     r8,r8
0x00000142 BD1F  _rt_lib_init_alloca_1:
                POP     {r0-r4,pc}
0x00000144 B510  _rt_lib_shutdown:
                PUSH    {r4,lr}
0x00000146 BD10  _rt_lib_shutdown_fp_trap_1:
                POP     {r4,pc}
```

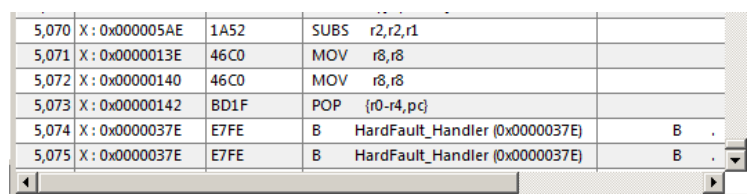
9. Clear the trace buffer. 
10. Click on RUN .
11. The program will immediately go to the Hard fault state and the breakpoint will stop execution at the B.
12. The Call Stack window will correctly show the program is in the Hard Fault Handler. See the bottom window below:
13. The Trace data now shows the last number of instructions executed plus the two MOV and the POP instructions. Clearly you can see the sequence of instructions that caused the fault. The trace was not cleared in these windows.
14. Click on Step (F11) a few times and the B at the HardFault\_Handler will be executed and displayed as shown below: Remember, a CoreSight breakpoint does not execute the instruction it is set to.
15. Remove the breakpoint. 
16. Exit Debug mode. 



```
Disassembly
0x0000037C E7FE  B      NMI_Handler (0x0000037C)
263:          B          .
264:          ENDP
265: SVC_Handler PROC
266:          EXPORT SVC_Handler
0x0000037E E7FE  B      HardFault_Handler (0x0000037E)
267:          B          .
268:          ENDP
269: PendSV_Handler PROC
270:          EXPORT PendSV_Handler
0x00000380 E7FE  B      SVC_Handler (0x00000380)
```

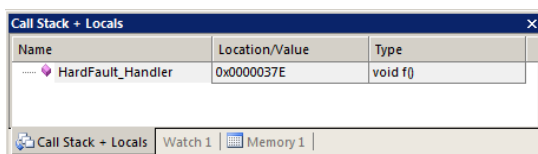


Index	Address	Opcode	Instruction	Src Code
5,065	X: 0x000005AE	1A52	SUBS r2,r2,r1	
5,066	X: 0x000005B0	4282	CMP r2,r0	
5,067	X: 0x000005B2	D3FA	BCC 0x000005AA	
5,068	X: 0x000005AA	4A03	LDR r2,[pc,#12] ; @0x000005B8	
5,069	X: 0x000005AC	6812	LDR r2,[r2,#0x00]	
5,070	X: 0x000005AE	1A52	SUBS r2,r2,r1	
5,071	X: 0x0000013E	46C0	MOV r8,r8	
5,072	X: 0x00000140	46C0	MOV r8,r8	
5,073	X: 0x00000142	BD1F	POP {r0-r4,pc}	



Name	Location/Value	Type
HardFault_Handler	0x0000037E	void f()







**TIP:** MTB can be used to solve many program flow problems that often require much effort to solve.



Name	Location/Value	Type
HardFault_Handler	0x0000037E	void f()

## 16) RTX\_Blinky Example Program with Keil RTX RTOS:

Keil provides RTX, a full feature RTOS. RTX is included as part of Keil MDK including source. It can have up to 255 tasks and no royalty payments are required. This example explores the RTX RTOS project. MDK will work with any RTOS. An RTOS is just a set of C functions that gets compiled with your project. RTX comes with a BSD type license and source code.

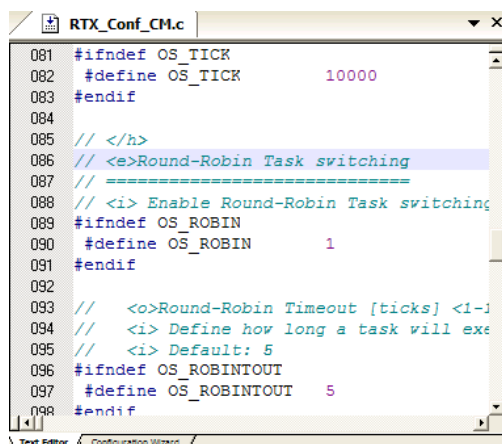
1. With  $\mu$ Vision in Edit mode (not in debug mode): Select Project/Open Project.
2. Open the file C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z\RTX\_Blinky\Blinky.uvproj.
3. Select the debug adapter you are using. This exercise uses CMSIS-DAP - Flash. 
4. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
7. The three leds will blink in accordance with three of the four tasks running representing a stepper motor driver.
8. Click on STOP .

### To Make The Leds blink more realistically:

1. In Task 3 in Blinky.c, near line 119, change from LEDRed\_On(); to LEDGreen\_On();
2. In Task 5 in Blinky.c, near line 143, comment out the line LEDGreen\_On();
3. Exit Debug mode, rebuild, program Flash, enter Debug mode and click on RUN.

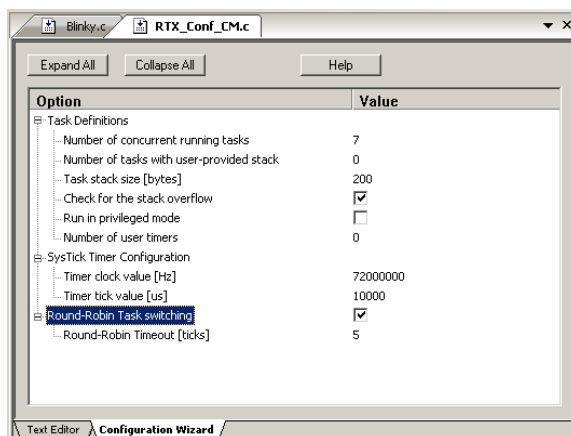
### The Configuration Wizard for RTX:

1. Click on the RTX\_Conf\_CM.c source file tab as shown below on the left. You can open it with File/Open if needed.
2. Click on the Configuration Wizard tab 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. Changing an attribute in one tab changes it in the other automatically. You should save a modified window.
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  $\mu$ Vision System Viewer windows are created in a similar fashion. Select View/System Viewer.



```
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] <i>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
```


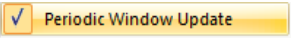
Text Editor: Source Code



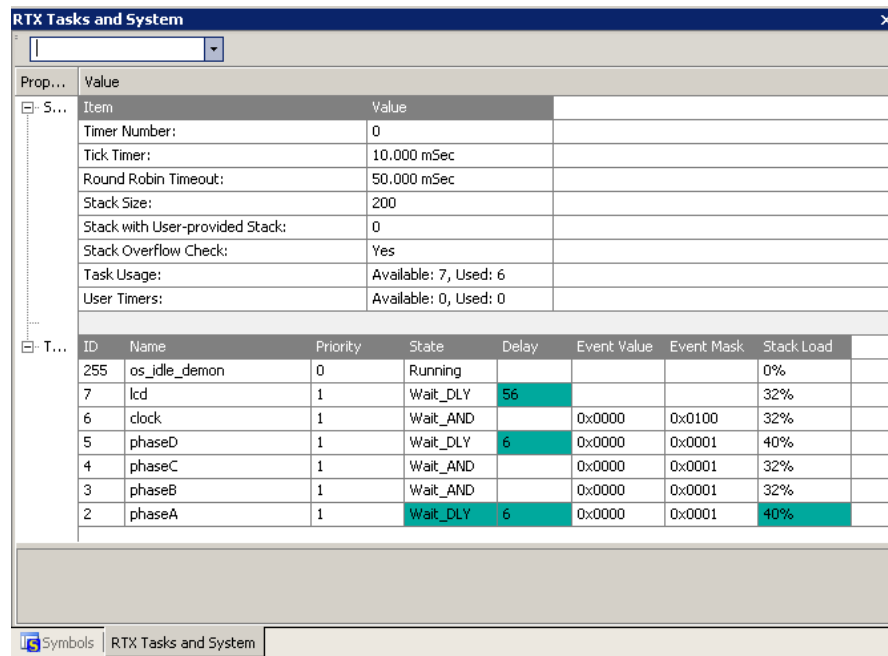
Configuration Wizard

## 18) RTX Kernel Awareness using RTX 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 µVision. µVision has an extensive kernel awareness windows for MQX. See [www.keil.com/freescale/mqx.asp](http://www.keil.com/freescale/mqx.asp) for a video on this feature.

1. Run RTX\_Blinky by clicking on the Run icon. 
2. Open Debug/OS Support and select RTX Tasks and System and the window below opens up. You might have to grab the window and move it into the center of the screen. Note these values are updating in real-time using the same technology as used in the Watch and Memory windows.
3. Select View and select Periodic Window Update if these values do not change: 

You will not have to stop the program to view this data. No CPU cycles are used. Your program runs at full speed. No instrumentation code needs to be inserted into your source.




Prop...	Value
Item	Value
Timer Number:	0
Tick Timer:	10.000 mSec
Round Robin Timeout:	50.000 mSec
Stack Size:	200
Stack with User-provided Stack:	0
Stack Overflow Check:	Yes
Task Usage:	Available: 7, Used: 6
User Timers:	Available: 0, Used: 0

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
7	lcd	1	Wait_DLY	56			32%
6	clock	1	Wait_AND		0x0000	0x0100	32%
5	phaseD	1	Wait_DLY	6	0x0000	0x0001	40%
4	phaseC	1	Wait_AND		0x0000	0x0001	32%
3	phaseB	1	Wait_AND		0x0000	0x0001	32%
2	phaseA	1	Wait_DLY	6	0x0000	0x0001	40%

### Demonstrating States:


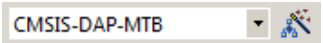
Blinky.c contains four tasks. Task 1 (phaseA) is shown below:

1. The gray areas opposite the line numbers indicate there is valid assembly code located here.
2. Set a breakpoint on one of these in Task 1 as shown: (but not on the for (;;) line)
3. Set a breakpoint in one of the other tasks at a similar location.
4. Click on RUN .
5. When the program stops, this information will be updated in the RTX Tasks window. The Task running when the program stopped will be indicated with a "Running" state. Most of the time the CPU is executing the os\_idle\_demon.
6. Remove the breakpoints and close the RTX Tasks window.



**Next:** How to configure the MTB trace.

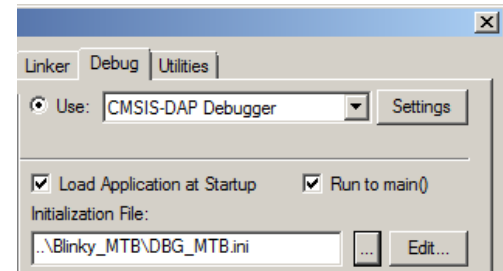
## 18) Configuring the MTB Trace:

It is easy to configure the MTB trace.  $\mu$ Vision automatically detects an MTB equipped processor. It then sets much of the configuration registers from this information. You can use the provided defaults or select several options.


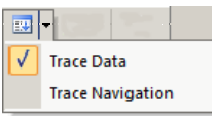


1. Using the RTX\_Blinky example from the previous page, exit Debug mode .
2. Select the Debug adapter you want to use. MTB works with any ULINK and OpenSDA (CMSIS-DAP). P&E was not tested for this document. 

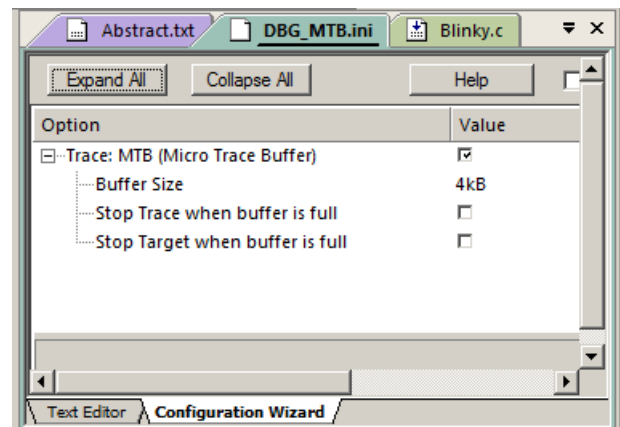
### Select the Initialization File:

3. Click on Target Options  and select the Debug tab.
4. The file DBG\_MTB.ini needs to be inserted in the Initialization box as shown here:
5. This file is located in C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z\Blinky\_MTB.
6. You can use the Browse icon to locate it. 
7. When entered, click on Edit... to open it up. Click on OK to return.
8. DBG\_MTB.ini will now be displayed with the other source files.
9. It is a small function that writes to several registers to set up MTB.
10. Click on the Configuration Wizard tab at the bottom. The next screen opens:
11. You can enable/disable and change the size of the trace buffer here.  
The last two entries are not yet implemented in MDK 4.56. Leave all entries at the default.



### Run The Program:

1. Enter Debug mode .
2. Open the Trace Data window using the icon: or by selecting View/Trace/Trace Data. 
3. The Trace Data window will be visible: Size accordingly.
4. Click on RUN .
5. When you stop  the program, the Trace Data window will be updated.



Most of the trace frames will not have source lines associated with them as most of these instructions are part of the RTX libraries.

Trace Data					
Display: Execution					
Index	Address	Opcode	Instruction	Src Code	
5,511	X: 0x00000246	6018	STR r0,[r3,#0x00]	< Source File Not Available >	
5,512	X: 0x00000248	2302	MOVS r3,#0x02	< Source File Not Available >	
5,513	X: 0x0000024A	43DB	MVNS r3,r3	< Source File Not Available >	
5,514	X: 0x0000024C	4718	BX r3	< Source File Not Available >	
5,515	X: 0x000006F0	BDB0	POP {r4-r5,r7,pc}	}	
5,516	X: 0x00000738	E7EE	B 0x00000718		
5,517	X: 0x00000718	2201	MOVS r2,#0x01	os_evt_wait_and (0x0001, 0xffff); ...	
5,518	X: 0x0000071A	4949	LDR r1,[pc,#292] ; @0x00000840		
5,519	X: 0x0000071C	4610	MOV r0,r2		
5,520	X: 0x0000071E	4F49	LDR r7,[pc,#292] ; @0x00000844		
5,521	X: 0x00000720	46BC	MOV r12,r7		
5,522	X: 0x00000722	DF00	SVC 0x00		
5,523	X: 0x000001D8	F3EF8009	MRS r0,PSP	< Source File Not Available >	

## 19) DSP SINE example using ARM CMSIS-DSP Libraries:





ARM CMSIS-DSP libraries are offered for Cortex-M0, Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil\ARM\CMSIS. README.txt describes the location of various CMSIS components. for more information see [www.arm.com/cmsis](http://www.arm.com/cmsis) and [www.onarm.com/cmsis/download/](http://www.onarm.com/cmsis/download/).

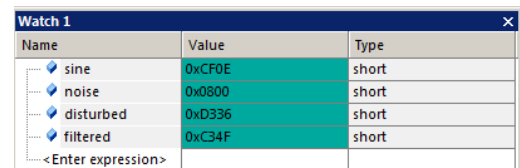
CMSIS is an acronym for Cortex Microcontroller Software Interface Standard.

This example creates a sine wave with noise added, and then the noise is filtered out.

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

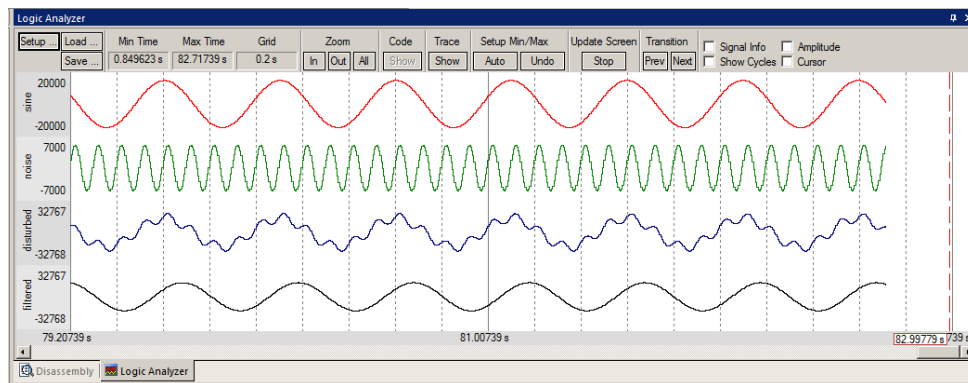
To obtain this example file, go to [www.keil.com/appnotes/docs/apnt\\_232.asp](http://www.keil.com/appnotes/docs/apnt_232.asp) and copy it into C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z. A \DSP directory will be created.

1. Stop the program and exit Debug mode if necessary.
2. Open the project file sine: C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z\DSP\sine.uvproj
3. Build the files.  There will be no errors or warnings.
4. Program the KL25Z flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
6. Click on the RUN icon. 
7. Open Watch 1 by selecting View/Watch/Watch 1 if necessary.
8. Four global variables will be displayed in Watch 1 as shown here:

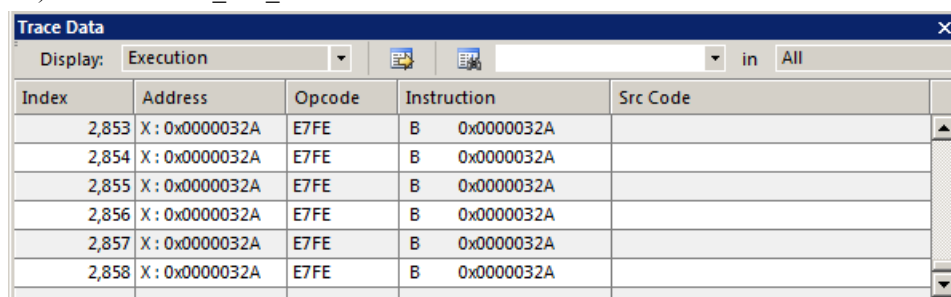


Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short
<Enter expression>		

9. The Kinetis Cortex-M4 processors have Serial Wire Viewer (SWV). The four waveforms will be displayed in the Logic Analyzer as shown below. The Kinetis KL25Z does not have SWV so this screen is shown for reference only. You can see the four global variables graphed:



10. Open the Trace Data window and the executed instructions will be displayed. Most will be the branch instruction to itself (B 0x32A) which is the os\_idle\_demon as shown below:

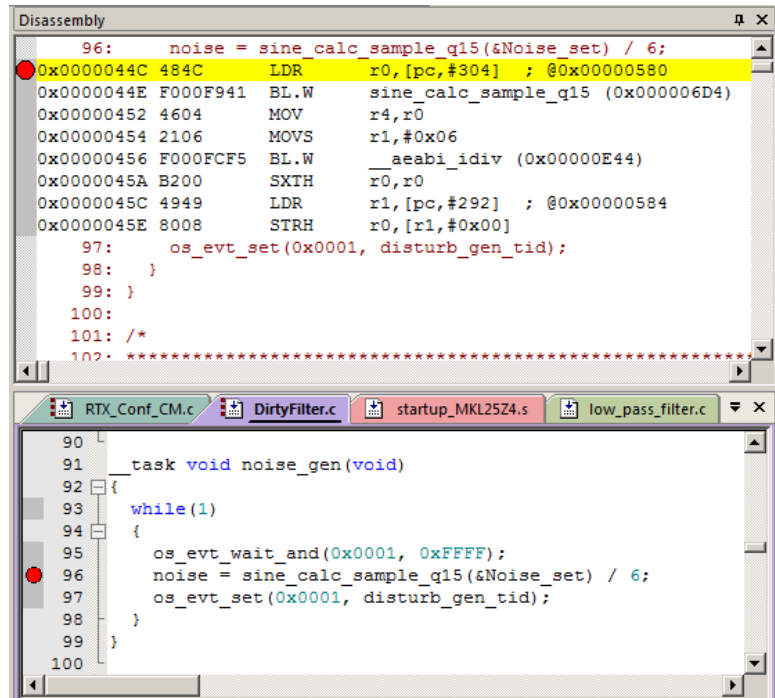


Index	Address	Opcode	Instruction	Src Code
2,853	X : 0x0000032A	E7FE	B 0x0000032A	
2,854	X : 0x0000032A	E7FE	B 0x0000032A	
2,855	X : 0x0000032A	E7FE	B 0x0000032A	
2,856	X : 0x0000032A	E7FE	B 0x0000032A	
2,857	X : 0x0000032A	E7FE	B 0x0000032A	
2,858	X : 0x0000032A	E7FE	B 0x0000032A	



## RTX Tasks and System:

1. Open Debug/OS Support and select RTX Tasks and System. A window similar to below opens up.
2. Note this window does not update: nearly all the processor time is spent in the idle daemon. The processor spends relatively little time in each task.
3. Set a breakpoint in one of the tasks in DirtyFilter.c by clicking in the left margin on a grey area.
4. The program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the noise\_gen task:



5. Clearly, in the RTX Tasks and System window below, you can see that noise\_gen was running when the breakpoint was activated. Os\_idle\_demon is Ready to run when noise\_gen is finished and no other task is Ready.

**TIP:** os\_idle\_demon has a Priority of 0 which is the lowest priority possible. Everything other task has a higher priority.

6. Remove the breakpoint and click on RUN.

**TIP:** Recall this window uses the CoreSight DAP read and write technology to update this window and does not steal CPU cycles.

## This is the end of the exercises.

Next is how to create your own projects.

Then:

How to configure a ULINK2 or ULINKpro.

A review of what trace is good for.

Keil product and contact information.

RTX Tasks and System

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


## 20) Creating your own project from scratch: Using the Blinky source files:

All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run a bare Blinky example. It has an empty main() function so it does not do much. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using an RTOS.

### Create a new project called Mytest:

1. With  $\mu$ Vision running and not in debug mode, select Project/New  $\mu$ Vision Project...
2. In the window Create New Project that opens, go to the folder C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z\.

### Create a new folder and name your project:

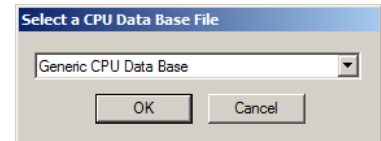
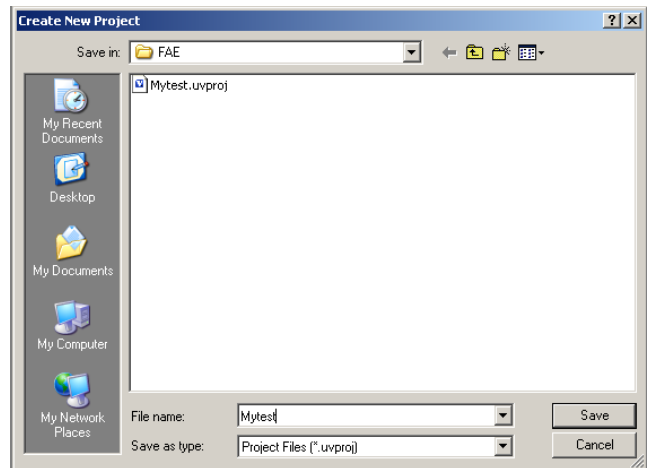
3. Right click inside this window and create a new folder by selecting New/Folder. I named this new folder FAE.
4. Double-click on the newly created folder “FAE” to enter this folder. It will be empty.
5. Name your project in the File name: box. I called mine Mytest. You can choose your own name but you will have to keep track of it. This window is shown here: 
6. Click on Save.

### Select your processor:

7. “Select a CPU Data Base File” shown below opens up.
  8. Click on OK and the Select Device for “Target 1” opens up as shown below.
  9. This is the Keil Device Database<sup>®</sup> which lists all the devices Keil supports. You can create your own if desired for processors not released yet.
  10. Locate the Freescale directory, open it and select MKL25Z128xxx4 (or the device you are using). Note the device features are displayed.
  11. Select OK.
- $\mu$ Vision will configure itself to this device.

### Select the startup file:

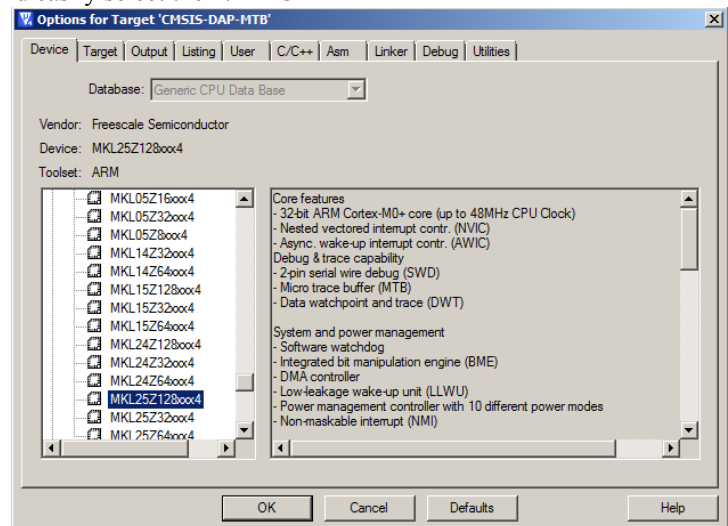
12. A window opens up asking if you want to insert the default MKL25Z startup file to your project. Click on “Yes”. This will save you some time.
13. In the Project Workspace in the upper left hand of  $\mu$ Vision, open up the folders Target 1 and Source Group 1 by clicking on the “+” beside each folder.
14. We have now created a project called Mytest with the target hardware called Target 1 with one source assembly file startup\_MKL25Z4.s and using the MKL25Z processor.



**TIP:** You can create more target hardware configurations and easily select them. This can include multiple Target settings, simulation and RAM operations. See Projects/Manage/Components

### Rename the Project names for convenience:

15. Click once on the name “Target 1” (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose KL25Z Flash. Press Enter to accept this change. Note the Target selector in the main  $\mu$ Vision window also changes to KL25Z Flash.
16. Similarly, change Source Group 1 to Startup. This will add some consistency to your project with the Keil examples. You can name these or organize them differently to suit yourself.
17. Select File/Save All.




### Select the source files and debug adapter:

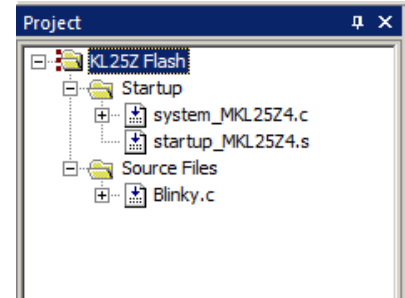
1. Using MS Explore (right click on Windows Start icon), copy blinky.c and system\_MKL25Z4.c from C:\Keil\ARM\Boards\Freescale\FRDM-KL25Z\Blinky to the ..\FRDM-KL25Z\FAE folder you created.

### Source Files:


2. In the Project Workspace in the upper left hand of  $\mu$ Vision, right-click on “SAM3X Flash” and select “Add Group”. Name this new group “Source Files” and press Enter. You can name it anything. There are no restrictions from Keil.
3. Right-click on “Source Files” and select **Add files to Group “Source Files”**.
4. Select the file Blinky.c and click on Add (once!) and then Close.

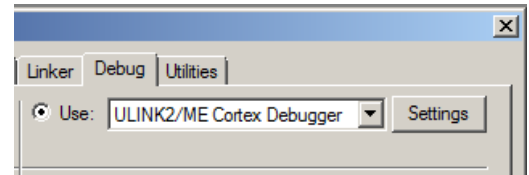
### System File:

5. Right-click on “Startup” and select **Add files to Group “Source Files”**.
6. Select the file system\_MKL25Z4.c and click on Add (once!) and then Close.
7. Your Project window will look similar to the one shown here: 



### Select your Debug Adapter:

8. By default the simulator is selected when you create a new  $\mu$ Vision project. You probably need to change this to a USB adapter such as a ULINK2.
9. Select Target Options  or ALT-F7 and select the Debug tab. Select ULINK/ME Cortex Debugger as shown below: If you are using another adapter such as CMSIS-DAP or ULINKpro, select the appropriate adapter from the pull-down list.
10. Select JTAG/SWD debugging (as opposed to selecting the Simulator) by checking the circle just to the left of the word “Use:” as shown in the window to the right:
11. Select the Utilities tab and select the appropriate debug adapter and the proper Flash algorithm for your processor. Refer to Using Various USB adapters: starting on pages 4 and 23 for more information.
12. Click on the Target tab and select MicroLIB for smaller programs. See [www.keil.com/appnotes/files/apnt202.pdf](http://www.keil.com/appnotes/files/apnt202.pdf) for details.



### Modify Blinky.c

13. Double-click the file Blinky.c in the Project window to open it in the editing window or click on its tab if it is already open.
14. Delete everything in Blinky.c except the main () function to provide a basic platform to start with:

```
#include <MKL25Z4.H>

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






    while(1) {

    }

}
```

15. Select File/Save All

### Compile and run the program:

16. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
17. Program the KL25 Flash by clicking on the Load icon: . Progress will be indicated in the Output Window.
18. Enter Debug mode by clicking on the Debug icon. 
19. Click on the RUN icon. . Note: you stop the program with the STOP icon. 
20. The program will run but since while(1) is empty – it does not do much. You can set a breakpoint.
21. You should be able to add your own source code to create a meaningful project.

**This completes the exercise of creating your own project from scratch.**  
**You can also configure a new RTX project from scratch using RTX\_Blinky project.**

## 21) Kinetis KL25 Cortex-M0+ Trace Summary:

### Watch and Memory windows 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 CoreSight and CPU by definition.

### Serial Wire Viewer displays in various ways: : (Cortex-M0+ does not have SWV. Kinetis Cortex-M4 does)

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

### Instruction Trace (MTB) 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).

### 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), a 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 is best for this.
- Communication protocol and timing issues. System timing problems.

## 22) Useful Documents: See [www.keil.com/freescale](http://www.keil.com/freescale)


1. **The Definitive Guide to the ARM Cortex-M3** by Joseph Yiu. (he also has one for the Cortex-M0) Search the web.
2. **MDK-ARM Compiler Optimizations: Appnote 202:** [www.keil.com/appnotes/files/apnt202.pdf](http://www.keil.com/appnotes/files/apnt202.pdf)
3. **Kinetis FlexMemory:** [http://www.keil.com/appnotes/docs/apnt\\_220.asp](http://www.keil.com/appnotes/docs/apnt_220.asp)
4. **A list of resources is located at:** <http://www.arm.com/products/processors/cortex-m/index.php>  
Click on the Resources tab. Or search for "Cortex-M3" on [www.arm.com](http://www.arm.com) and click on the Resources tab.

## 23) Configuring a Keil ULINK2 or ULINK-ME:

The ULINK2 and ULINK-ME are essentially the same devices electrically and functionally. Any reference to ULINK2 in this document includes the ME. The ULINK<sub>pro</sub>, which is a Cortex-M ETM trace adapter, can be used like a ULINK2 or ULINK-ME. All three provide the same functions as CMSIS-DAP.

Assume a ULINK2 is connected to a powered up Kinetis target board as shown on page 1. Add a USB cable to J7 to power the board. The next page has a close-up photo.  $\mu$ Vision must be in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid Freedom project as described.

### Select the debug connection to the target:

1. Select Target Options  or ALT-F7 and select the Debug tab. In the drop-down menu box select the ULINK2/ME as shown here:
2. 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).
3. In **Port**: select SWJ and SW. The KL25 processor does not have JTAG.
4. 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.

### Configure the Keil Flash Programmer:

5. Click on OK once and select the Utilities tab.
6. Select the ULINK similar to Step 2 above.
7. Click Settings to select the programming algorithm.
8. Select Add and select the appropriate Kinetis Flash if necessary as shown below:
9. MKXX 48 MHz 128kB Prog Flash is the correct one to use with the Freedom board.
10. Click on OK once.

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

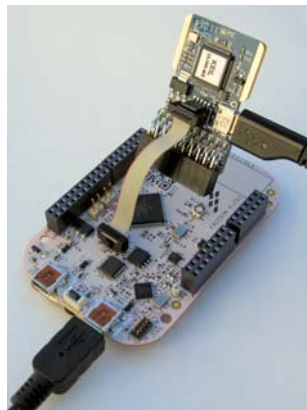
11. Click on OK to return to the  $\mu$ Vision main screen.
12. You have successfully connected to the KL25 target.

### Keil ULINK-ME

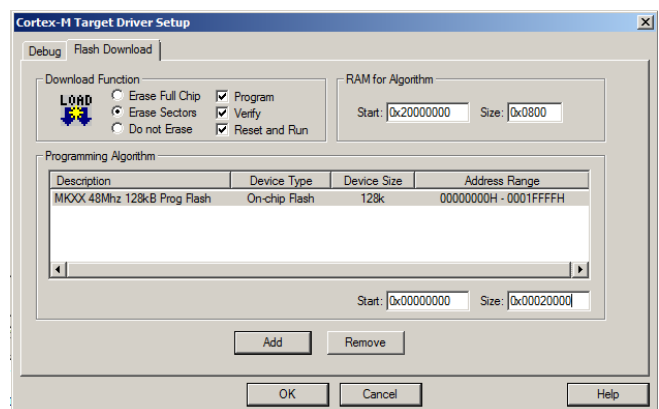
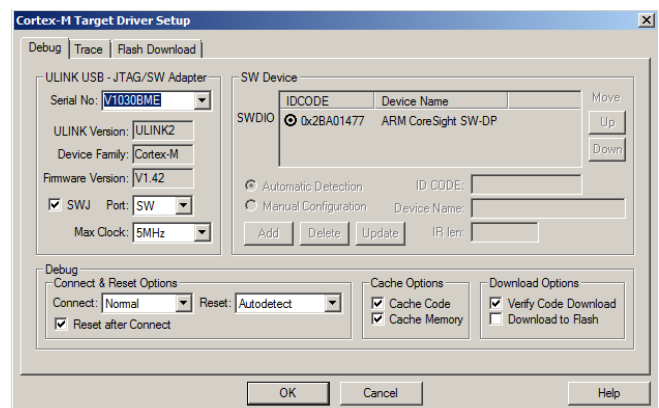
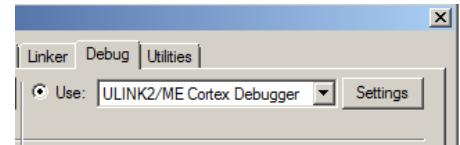
ULINK-ME is available only as part of a board kit from Keil or another OEM.

It is shown here connected to J6 on the KL25 Freedom board.

J6 is the SWD connector. All ULINK models connect to J6 for CoreSight debug control access.



**TIP:** If you select ULINK or ULINK<sub>pro</sub>, and have the opposite ULINK actually 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, and not that no ULINK was attached. Select the correct ULINK.

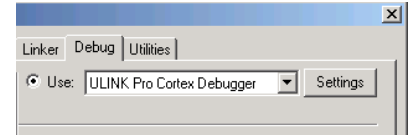




## 24) Configuring a Keil ULINKpro:

This is configured the same way as a ULINK2 except for the two selection entries. One is in the Debug tab (shown below) and the other in the Utilities tab for Flash programming.

1. The first photo shows how the ULINK CoreSight 10 pin cable is connected to J6 (JTAG) on the Freedom board. Make sure Pin 1 (red wire) is orientated correctly and that all pins are connected if the connector is offset. Some cables might have a blank in Pin 7 which you will have to remove.
2. In the Debug tab in Options for target window, select ULINK Pro Cortex Debugger as shown below.
3. Select Settings and the next window below opens up. This is the control panel for the ULINKpro.
4. In **Port**: select SWJ and SW. The KL25 processor does not have JTAG.
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 ULINKpro and the board.
6. Select the Utilities tab and select the ULINKpro and select the programming algorithm as done with the ULINK2. Refer to the previous page for instructions.

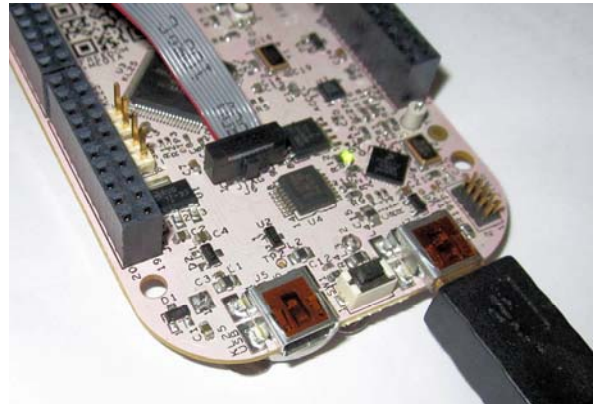


### Configure the Keil Flash Programmer:

1. Click on OK once and select the Utilities tab.
2. Select the ULINK similar to Step 2 above.
3. Click Settings to select the programming algorithm.
4. Select Add and select the appropriate Kinetis Flash if necessary as shown on the previous page.
5. MKXX 48 MHz 128kB Prog Flash is the correct one to use with the Freedom board.
6. Click on OK twice.

**TIP:** With a ULINK2 and ULINKpro, the case must be removed to change the cable. Make sure you do not disturb the battery in the ULINKpro. If the RAM is erased the ULINKpro must be sent back to Keil for reprogramming. ULINK2 has no such battery.

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



### Keil ULINKpro information:

ULINKpro is an ETM trace adapter that can be used as a ULINK2. ULINKpro has very fast Flash programming and an enhanced Instruction Trace window that connects the trace frames to your source code.

Instruction trace on Kinetis Cortex-M4 processors requires ETM.

ULINKpro features are best exploited with Kinetis Cortex-M4 processors. ULINK2 provides SWV but not ETM on Kinetis Cortex-M4 processors. See the lab on [www.keil.com/freescale](http://www.keil.com/freescale).

ULINKpro supports Serial Wire Viewer (SWV) and ETM trace with all Cortex-M3 and Cortex-M4 devices.

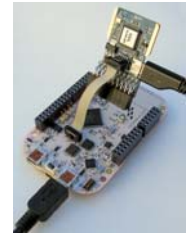
The KL25 Cortex-M0+ processors support MTB trace and DAP read/write memory cycles. These are described in this document. They do not support SWV or ETM trace.



## 25) Keil Products and Contact Information:

### Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite (Evaluation version) \$0
- MDK-Freescale™ For all Kinetis Cortex-M0+, Cortex-M4. 1 year term license - \$745
- MDK-Basic (256K Compiler Limit, No debug Limit) - \$2,695
- MDK-Standard (unlimited compile and debug code and data size) - \$4,895
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) \$9,995



### USB-JTAG adapter (for Flash programming too)

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



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

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

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

Call Keil Sales for details on current pricing, specials and quantity discounts. Sales can also provide advice about the various tools options available to you. They will help you find various labs and appnotes that are useful.

All products are available from stock.

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

Call Keil Sales for special university pricing. Go to [www.arm.com](http://www.arm.com) and search for university to view various programs and resources.

Keil supports many other Freescale processors including ARM9™ and Cortex-A series processors. See the Keil Device Database® on [www.keil.com/dd](http://www.keil.com/dd) for the complete list of Freescale support. This information is also included in MDK.

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

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

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



### For more information:

**Keil Sales** In USA: [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, go to [www.keil.com/freescale](http://www.keil.com/freescale) and for more Freescale specific information.

CMSIS Version 3: [www.onarm.com/downloads](http://www.onarm.com/downloads) or [www.arm.com/cmsis](http://www.arm.com/cmsis)

