# Freescale Kinetis L Series: Cortex-M0+ Training
# Using the Freedom KL25Z   *featuring MTB: Micro Trace Buffer*
**ARM Keil MDK 5.10 Toolkit** Winter 2014  V 2.1  Robert Boys  bob.boys@arm.com

**ARM** KEIL
Microcontroller Tools

## Introduction:

The latest version of this document is here:  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 processer including Micro Trace Buffer (MTB).  At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. We recommend you obtain the *new* **Getting Started MDK 5:** from here:  www.keil.com/mdk5/.

Keil MDK supports and has examples for most Freescale ARM processors.  Check the Keil Device Database® on 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.

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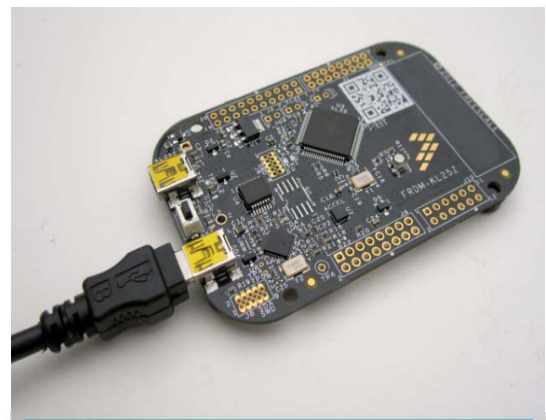
**Middleware:**  MDK Professional contains middleware libraries including TCP/IP stack, CAN drivers, a Flash file system and USB drivers.  Contact Keil sales for information regarding middleware for your processor. http://www.keil.com/arm/mdk.asp.

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

## 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/assembler/linker 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.

3. All applicable ARM debugging technology is supported.

4. RTX Kernel Awareness window.  It is updated in real-time.

5. Available Debug Adapters:  OpenSDA (CMSIS-DAP), ULINK™2, ULINK-ME and ULINK*pro*.

6. Kernel Awareness is available for Keil RTX and Freescale MQX.  RTX Kernel Awareness is updated in real-time.  Many other RTOSs are compatible with MDK.

7. **MQX:**  An MQX port for MDK is available including Kernel Awareness windows.  See www2.keil.com/freescale/mqx.



The Freedom KL25Z board connected to run OpenSDA (CMSIS-DAP) with Keil MDK.

8. **Processor Expert** compatible.  For more information see www.keil.com/appnotes/files/apnt_235_pe2uv.pdf.

9. Keil Technical Support is included for one year and is easily renewable.  This helps your project get completed faster.

10. 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: the executed instructions.

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*pro*.  MTB provides instruction trace which is essential for solving program flow and other related problems.  How to use MTB is described in this document.

---

**1**

**General Information:**

---

www.keil.com

## 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 the i.MX series see www.arm.com/ds5

On the last page of this document is an extensive list of resources that will help you successfully create your projects. This list includes application notes, books and labs and tutorials for other Freescale boards.

We recommend you obtain the latest Getting Started Guide for MDK5: It is available on www.keil.com/mdk5/.

**Community Forums:** www.keil.com/forum and http://community.arm.com/groups/tools/content

## 2) MDK 4.73 vs MDK 5.10 Keil Software: *This document uses MDK 5.10 or later.*

There are two versions of this document: one for MDK 4.7x and the other for MDK 5.10 and later.

**MDK 4.7x: Legacy MDK.** Does not contain Software Packs. Projects created with 4.7x are compatible with MDK 5.10 and up. MDK 4.73 is the current official Keil release. It is available at www.keil.com/demo/eval/armv4.htm

**Example Project Files:** MDK 4.73 provides Blinky and RTX_Blinky in C:\Keil\ARM\Boards\Freescale\. The DSP example is available where this document is stored: www.keil.com/appnotes/docs/apnt_232.asp

**MDK 5.10:** This is the new MDK containing Software Packs. This is a new method of providing software including middleware, CMSIS header and configuration files, RTX and middleware from a webserver. This document uses MDK 5.10. These Packs are downloaded with the Packs Installer and configured with the Run Time Environment (RTE).

MDK 4 projects will run on MDK 5.10. You normally have to add MDK v4 Legacy Support: www2.keil.com/mdk5/legacy

**Example Project Files:** This document uses the RTX_Blinky example project contained in 5.10. Blinky5 and DSP5 are available where this document is stored: www.keil.com/appnotes/docs/apnt_232.asp. Lab projects from appnotes and based on MDK 5 Software Packs are appended with a "5" as in Blinky5. This is not true for example projects provided with MDK.

To import 4.7x examples into MDK 5.10: www2.keil.com/mdk5/legacy. You do not need to install these files for this tutorial.

**MDK 4.7x and 5.10 both have:**

- Complete OpenSDA (CMSIS-DAP) support.
- MTB Trace works with OpenSDA, ULINK2, ULINK-ME and ULINK*pro*.
- Example files for the Freedom and Tower KL25Z boards and many others. MDK 5.10 has a collection of CMSIS-Pack compliant example files including for Kinetis processors. More are to be added.

**RTX:** MDK 4.7x and MDK 5.10 RTX are slightly different. 4.7x RTX is not compliant to CMSIS-RTOS. MDK 5.10 RTX is compliant. The main difference is the different names of functions called. They both work essentially the same.

Install MDK into the default directory. You can install MDK into any directory, but references in this lab refer to C:\Keil_v5.

**OpenSDA:** OpenSDA is Freescale's name for ARM's CMSIS-DAP. **T**his is an ARM standard that specifies an on-board debug adapter. The Freedom board incorporates CMSIS-DAP. You are able to incorporate CMSIS-DAP debugger on your own board. See www.arm.com/cmsis. You do not need an external debugger such as a ULINK2 to do this lab.

If you construct your own board, in order to debug, you either have to add a CMSIS-DAP processor or connect a ULINK.

## 3) Keil Software Download and Installation:

1. Download MDK 5.10 or later from the Keil website. www.keil.com/mdk5/install

   Download MDK-Core Version 5

2. Install MDK into the default directory. You can install into any directory, but this lab uses the default C:\Keil_v5

3. We recommend you use the default directories for this tutorial. We will use C:\MDK\ for the examples.

4. If you install MDK into a different directory, you will have to adjust for the directory differences.

5. The examples Blinky5 and DSP5 are available on the web where you got this document. Please note there are two versions of this lab and its example projects: one for MDK 4.7x and the other for MDK 5.10 and later. See www.keil.com/appnotes/docs/apnt_232.asp

1. You do not need any debug adapters: just the Freedom board, a USB cable and MDK installed on your PC.

## 4) μVision Software Packs Download and Install Process:

### 1) Start μVision and open Pack Installer:

1. Connect your computer to the internet. This is needed to download the Software Packs.

2. Start μVision by clicking on its desktop icon.

3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
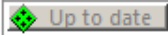
4. This window opens up: Select the Packs tab.

5. Note "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet before continuing.

6. If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or to refresh once you have connected to the Internet.
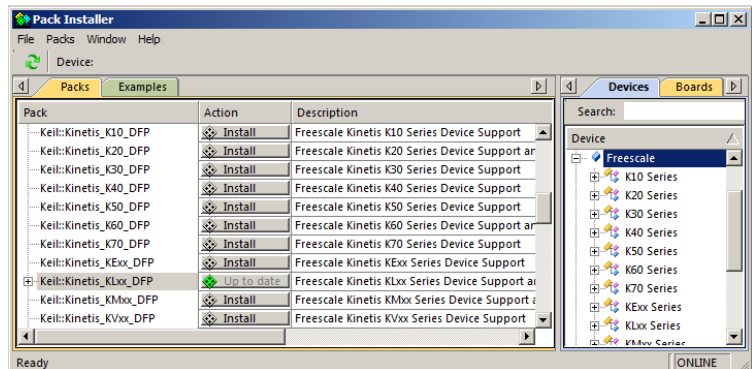
### 2) Install The KL25 Software Pack:

1. Initially, the Software Pack ARM::CMSIS is installed by default. Now, you will install Keil::Kinetis_KLxx_DFP. This is for the KL25 Freedom board.

2. Select Keil::Kinetis_KLxx_DFP and click on Install. This Software Pack will download and install to C:\Keil_v5\ARM\Pack\Keil\Kinetis_KLxx_DFP\1.0.0\ by default. This download can take two to four minutes.

3. Its status is indicated by the "Up to date" icon:

**TIP:** If you click on the Devices tab, you can select the processor you are using and this will be displayed in the Packs tab.
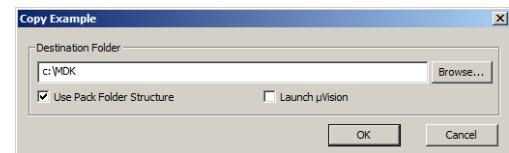
### 3) Install the RTX_Blinky MDK 5.10 Example:

1. Select the Examples tab. Select "Show examples from installed Packs only" as shown:

2. Select CMSIS-RTOS Blinky (FRDM-KL25Z):

3. Select Copy opposite CMSIS-RTOS Blinky (FRDM-KL25Z): as shown:

4. This window opens up: Select Use Pack Folder Structure:

5. Type in C:\MDK. Click OK to copy the RTX_Blinky project.

6. The RTX_Blinky example will now copy to C:\MDK\Boards\Freescale\FRDM-KL25Z\RTX_Blinky\.
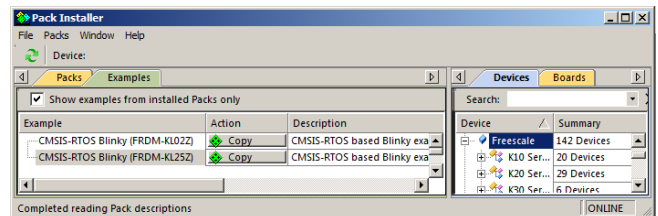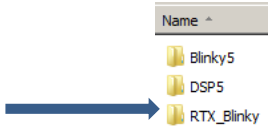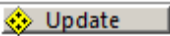
**TIP:** The default directory for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default directory of C:\MDK\ in this tutorial. You can use any directory you prefer.

7. Close the Packs Installer. You can open it any time by clicking on its icon.

### 4) Install the Blinky5 and DSP5 Examples from Keil.com:

1. Obtain the software zip file from www.keil.com/appnotes/docs/apnt_232.asp.

2. Unzip this into the directory C:\MDK\Boards\Freescale\FRDM-KL25Z\.

3. Blinky5 and DSP5 folders will be created with the RTX_Blinky folder as shown here:

**TIP:** An Update icon means there is an updated Software Pack available for download.

**TIP:** If you look in the directory C:\Keil_v5\ARM\Pack\Keil\Kinetis_KLxx_DFP\1.0.0\Boards\Freescale\FRDM-KL25Z\, you will find another RTX_Blinky. This is a read-only version. Use only the projects you copied over from the Examples tab. If you ever need to modify a read only file, you must change its permissions in the usual manner.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit            www.keil.com

## 5)  Programming the KL25Z with OpenSDA: an on-board Debug Adapter:

This document will use OpenSDA as a SWD Debug Adapter.  Target connection by µVision will be via a standard USB cable connected to SDA J7.  The on-board Kinetis K20 acts as the debug adapter.  Micro Trace Buffer frames can be displayed.

### This Step MUST be done !  at least once...

**Program the K20 with the CMSIS-DAP application file CMSIS-DAP.S19:**

**1)  Locate the file CMSIS-DAP.S19:**

1. CMSIS-DAP.S19 is located in the OpenSDA directory in MDK.  Using Windows Explorer navigate to C:\MDK\Boards\Freescale\FRDM-KL25Z\RTX_Blinky\OpenSDA. CMSIS-DAP.S19 is located here.  The other projects also contain this file.  You will copy this file into the Freedom board USB device as described below.

**2)  Put the Freedom Board into Bootloader: Mode:**

2. Hold RESET button SW1 on the Freedom board down and connect a USB cable to J7 SDA as shown here:

3.  When you hear the USB dual-tone, release RESET.

4. The green led D4 will blink about once per second.  The Freedom is now ready to be programmed with the CMSIS-DAP application.

5. The Freedom will act as a USB mass storage device called BOOTLOADER connected to your PC.  Open this USB device with Windows Explorer.
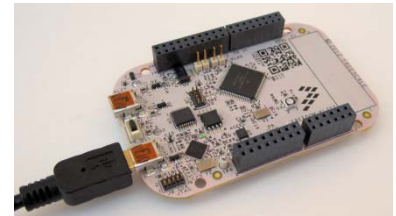
**3)  Copy CMSIS-DAP.S19 into the Freedom Board:**

6. Copy and paste or drag and drop CMSIS-DAP.S19 into this Bootloader USB device.
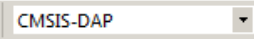
**4)  Exit Bootloader Mode:**

7. Cycle the power to the Freedom board while **not** holding RESET button down.  The green led will blink once and then stay off.

8. The Freedom board is now ready to connect to the µVision debugger and Flash programmer.

**TIP:**  The green led will indicate when µVision is in Debug mode and connected to the OpenSDA debug port SWD.  Remember, JTAG is not used.  The Kinetis Cortex-M0+ has only the SWD port.  You can do everything with the SWD port as you can with a JTAG port.  SWD is referenced as SW in the µVision configuration menu.

**TIP:**  This application will remain in the U6 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 the Motorola S record format that loads and runs on the K20 OpenSDA processor.

**TIP:**  If you must later re-program CMSIS-DAP and it still does not work:  check that Port: is set to SW and not JTAG.

## 6)  Testing The OpenSDA Connection:  (Optional Exercise)

1. Start µVision ![icon] if it is not already running.  Select Project/Open Project.

2. Select the Blinky project C:\MDK\Boards\Freescale\FRDM-KL25Z\Blinky5\Blinky.uvprojx.

3. Select "CMSIS-DAP" in the Select Target menu.

4. Select Target Options ![icon] or ALT-F7 and select the Debug tab:

5. Click on Settings: and the window below opens up:  If an ICODE and Device name is displayed, OpenSDA is working.  You can continue with the tutorial.  Click on OK twice to return to the µVision main menu.

6. If nothing or an error is displayed in this SW Device box, this **must** be corrected before you can continue.

**TIP:**  To refresh the SW Device box, in the Port: box select JTAG and then select SW again.  You can also exit then re-enter this window.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit
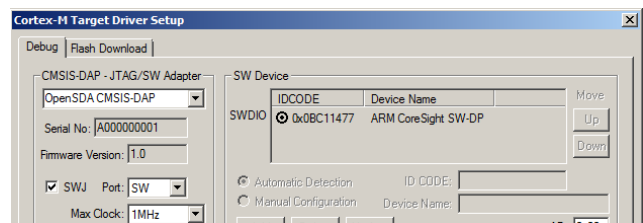
## 7)  *Blinky* example program using the Freescale Freedom KL25Z and OpenSDA:

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 the previous page.

2.  Connect a USB cable between your PC and Freedom USB J7 as shown here:

3.  Start µVision by clicking on its desktop icon.

4.  Select Project/Open Project.

5.  Open the file:
    C:\MDK\Boards\Freescale\FRDM-KL25Z\Blinky5\Blinky.uvprojx.

6.  Select "CMSIS-DAP" 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.
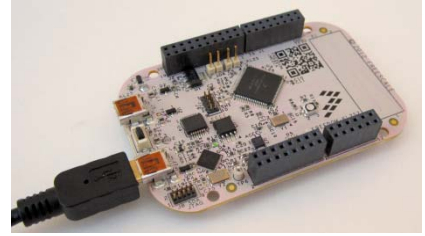
**TIP:**  If you get an error this probably means the Freedom board is not programmed with CMSIS-DAP.S19. Refer to the previous page for instructions.

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 enabled.

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 grey block on the left on a suitable part of the source code.  This means assembly instructions are present at these points.  Inside the while loop inside the main() function will work as shown below:  You can also click in the Disassembly window to set a breakpoint.

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.  Set a second breakpoint in the while() loop as before.

5.  Every time you click on the RUN icon    the program will run until the breakpoint is again encountered.

6.  **Remove the breakpoints by clicking on them.**

7.  Clicking in the source window will indicate the appropriate code line in the Disassembly window and vice versa.  This is relationship indicated by the cyan arrow and the yellow highlight:
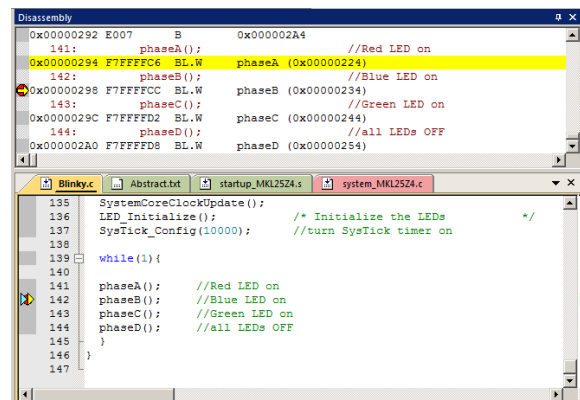
**TIP:**  If you set too many breakpoints, uVison will warn you. Note the µVision debugger sometimes uses one breakpoint for certain operations such as run to main().  You can only use one.
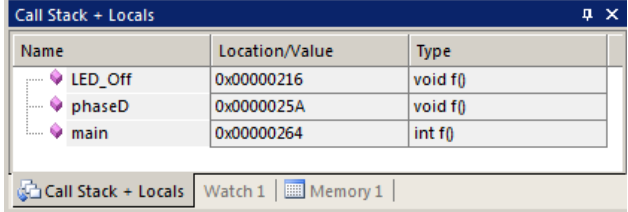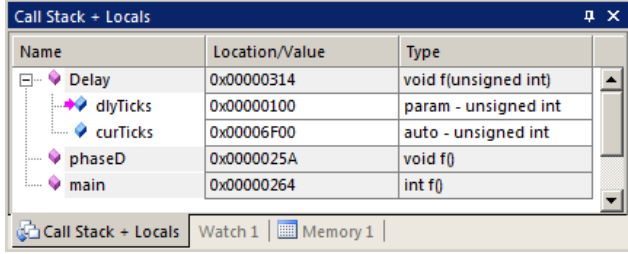
**TIP:**  ARM hardware breakpoints do **not** execute the instruction they are set to and land on.  CoreSight hardware breakpoints are no-skid.  This is a rather important feature for effective debugging.

---

**6**

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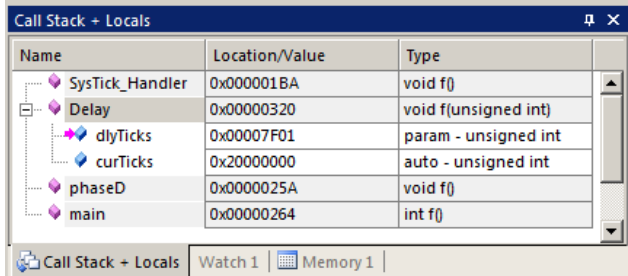
If possible, the values of the local variables will be displayed and if not the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

1. Click on RUN.

2. Set a breakpoint in the LED_Off function near line 92 in Blinky.c. It will soon stop on this breakpoint.

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

4. Shown is this Call Stack + Locals window:

5. The functions as they were called are displayed. If these functions had local variables, they would be displayed.

6. Click on the Step In icon or F11 a few times:

7. Continue until the program enters the Delay function. The Call Stack + Locals window will now show Delay with its two local variables and their values:

8. **Click on the StepOut icon or CTRL-F11 a few times in order 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, select Kill All and then Close.

**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 and static variables or structures, all updated in real-time in the Watch or Memory windows 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 this type of variable is usually stored in a CPU register.

### 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 near line 47 and this event is clearly shown at the top of this window:

As you click on the StepOut icon or Ctrl-F11 each function will be removed as it comes off the stack until you are left with only main(). These are added as the program is run and stopped.

**TIP:** You can set two hardware breakpoints with the Freescale Cortex-M0+ processor. If you set more than two, or you have two set and the debugger needs one for an operation, µVision will warn you to delete the excessive breakpoints. The Kinetis Cortex-M4 family of processors has 6 hardware breakpoints available.

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

**TIP:** To locate the definition of a variable, structure, function or define, right click on it and select Go To Definition…
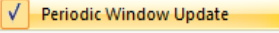
**TIP:** You can access the Hardware Breakpoint table by clicking on Debug/Breakpoints or Ctrl-B. This is also where Watchpoints (also called Access Points) are configured. You can temporarily disable entries in this table by unchecking them.
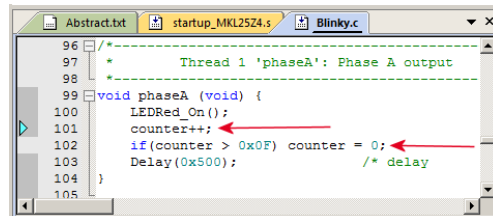
## 10) Watch and Memory Windows and how to use them:

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.
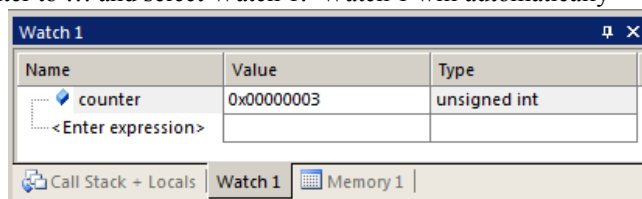
### Watch window:

**Add a global variable:** Call Stack, 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 15 in Blinky.c:

    **unsigned int counter = 0;**

3.  Add the statements near Line 101 just after LEDred_On();

    **counter++;**

    **if (counter > 0xF) counter = 0;**

4.  Select File/Save All.

5.  Click on Rebuild 🗓 and program the Flash with Load 🔽.

6.  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.

7.  Select View and select Periodic Window Update if necessary: ✓ Periodic Window Update

8.  In Blinky.c, right click on **counter** and select Add counter to … and select Watch 1. Watch 1 will automatically open. **counter** will be displayed as shown here:

9.  **counter** will update in real time.
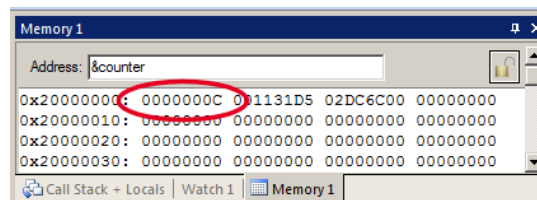
**TIP:** You can also block a variable name drag and drop it into Watch or Memory windows. You can also enter a variable manually by double-clicking <Enter expression> or press F2 and use copy and paste or typing the variable name.

**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.  Right click on **counter** and select Add counter to … and select the Memory 1 window.

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_0000).

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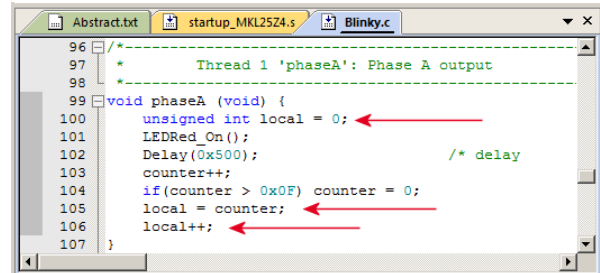
**TIP:** No CPU cycles are used to perform these operations. See the next page for an explanation how this works.

**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.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com

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

1. Stop the processor ⊗ and exit Debug mode. 🔍

2. Create a local variable in the Thread 1 function phaseA where you added the variable counter.

3. I called it local: **unsigned int local=0;** as shown at line 100 below:

4. Add the lines **local = counter;** and **local++;** as shown at line 105 and 106 below:

5. Select File/Save All. Compile the source files 📋. Program the Flash 📥.

6. Enter Debug mode. 🔍 Click on RUN. 📲

7. Enter **local** into Watch 1 window by right clicking on it and selecting Add local to.... Watch 1.

8. Note it says <cannot evaluate> or "not in scope". Stop the program ⊗ and the value of local will still not display.

9. Start the program. 📲



```
96 □/*------------------------------------
97  *     Thread 1 'phaseA': Phase A output
98  *------------------------------------
99 □void phaseA (void) {
100     unsigned int local = 0; ←
101     LEDRed_On();
102     Delay(0x500);                    /* delay
103     counter++;
104     if(counter > 0x0F) counter = 0;
105     local = counter; ←
106     local++; ←
107 }
```

10. Set a breakpoint in phaseA on Delay(500);. The program will stop and a value for **local** will now be displayed. The only time a local variable value will be displayed is if the program happens to stop while it is in scope.

11. µVision is unable to determine the value of **local** when the program is running because it exists only when the function phaseA is running. It disappears in functions and handlers outside of phaseA. **local** is a local or automatic variable and this means it is probably stored in a CPU register which µVision is unable to access during run time.

12. **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 **local** static where it is declared in Blinky.c !
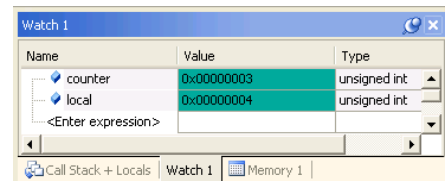
1. In the declaration for **local,** add the **static** keyword like this:

```
void phaseA (void) {
        static unsigned int local = 0;
```

**TIP:** You can also make a variable global or have it as part of a structure so it will update in real-time.

2. Compile the source files by clicking on the Rebuild icon 📋. Select File/Save All or 💾.

3. To program the Flash, click on the Load icon. 📥. Enter Debug mode. 🔍 Click on RUN. 📲

4. Note local is still not updated in real-time. You must first show local to µVision by stopping the program in phaseA.

5. Set a breakpoint in the function phaseA. The program will stop and a value for local will now be displayed. **Remove the breakpoint**. Click on RUN and local will now update in real-time.

**TIP:** You must fully qualify a variable in order for it to update without initially stopping the program while it is in scope. To do this, you can open the View/Symbols window and enter the variable from there. This automatically fully qualifies the variable. In this case, **local** fully qualified is \\Blinky\Blinky.c\phaseA\local. You also can enter this directly into the Watch or Memory windows.



6. You can also enter a variable into a Memory window. Remember to prefix it with an &.

**TIP:** Recall you can modify variables or memory locations in the Memory window when the program is running.

7. Stop the CPU ⊗ for the next step. Select File/Save All.

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

**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. This is set by default in the two Blinky projects. This means you can skip using the LOAD icon. 📥 This will be automatically done when you enter Debug mode. 🔍.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com

## 12) System Viewer (SV):

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

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

**GPIO Port B:**

2.  Select Peripherals/System Viewer and then GPIO and select PTB.  This Port B.

3.  This window opens up.  Expand PDOR:

4.  You can now see PTB update:

5.  You can also open Port D as one LED is connected to this port.

**TIP:**  Notice Port B stops changing for a time.  This is because Port D is now changing.  Two LEDs are on Port B and the other (blue) is connected to Port D.

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

**SysTick Timer:**  This program uses the SysTick timer as part of the Delay() function.

1.  Select Peripherals/System Viewer and then select SysTick.

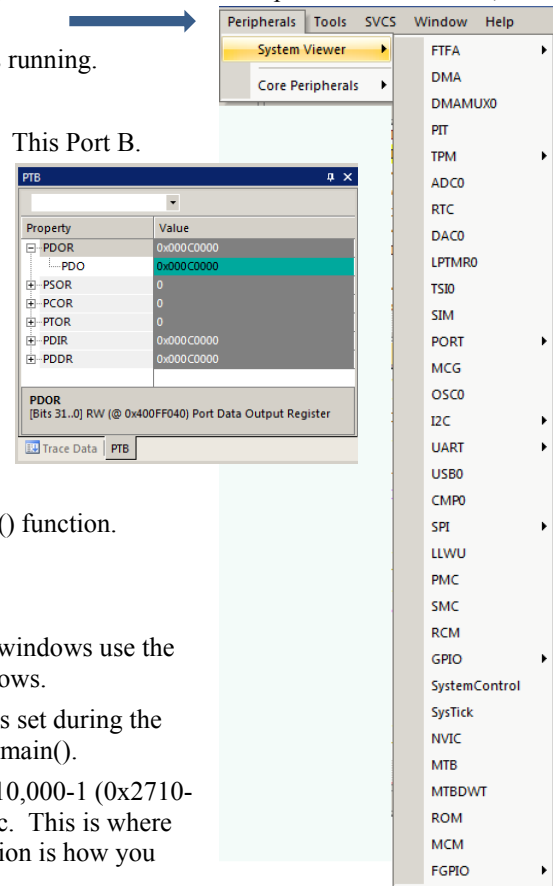2.  The SysTick window shown below opens:

3.  Note it also updates in real-time while your program runs.  These windows use the same CoreSight DAP technology as the Watch and Memory windows.

4.  Expand the RVR register.  This is the reload register value.  This is set during the SysTick configuration by the SysTick_Config(10000) function in main().

5.  Note that it is set to 0x270F.  This is the same value hex value of 10,000-1 (0x2710-1) that is programmed into SysTick_Config() in main() in Blinky.c.  This is where this value comes from.  Changing the variable passed to this function is how you change how often the SysTick timer creates its interrupt 15.

6.  In the RELOAD register in the SysTick window, *while the program is running,* type in 0x5000 and press Enter !

7.  The blinking LEDs will slow down.  This will convince you of the power of ARM CoreSight debugging.

8.  Replace RELOAD with 0x270F.  A CPU RESET  will also do this.

9.  You can look at other Peripherals contained in the System View windows.

10. When you are done, stop the program  and close all the System Viewer windows that are open.

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

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

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit
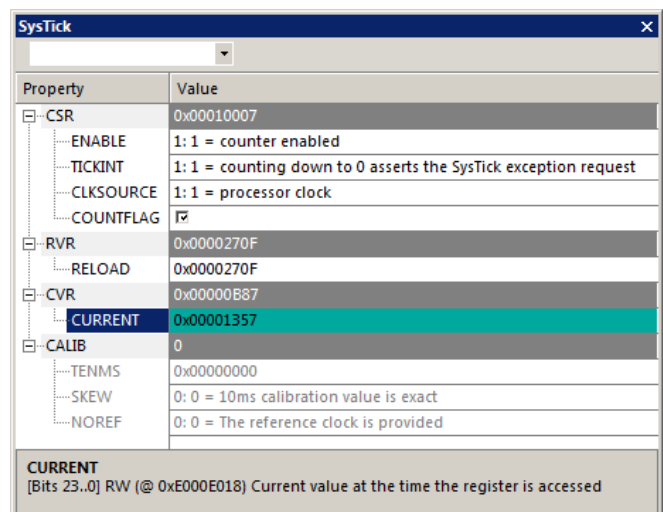
# 13) Watchpoints: *Conditional Breakpoints*

The KL25 Cortex-M0+ processor has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil documents. Cortex-M0+ Watchpoints are slightly intrusive. When the Watchpoint is hit, µVision must test the memory location. Cortex-M3/M4 Watchpoints are not intrusive for equality test.

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 µVision window and then select Breakpoints or press Ctrl-B.

4. Select Access to Read.

5. In the Expression box enter: "**counter == 0x5**" without the quotes. This window will display:

6. Click on Define or press Enter and the expression will be accepted as shown below in the bottom Breakpoints window:

7. Click on Close.

8. Enter the variable **counter** in Watch 1 if it is not already there.

9. Click on RUN. ⬇.
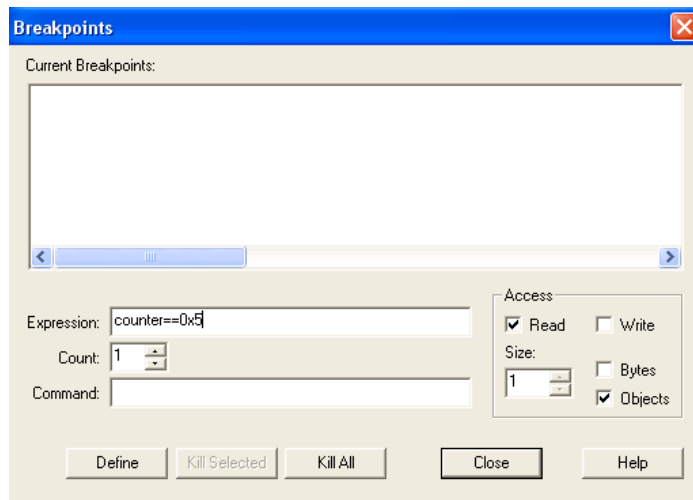
10. **counter** might not update in the Watch window depending on how fast the variable is changed. This feature is turned off in µVision for speed considerations. You will also notice the program slows down. This is because µVision must test the condition when the write or read occurs to `counter`. Minimize this by selecting only Read or Write Access.

11. When **counter** equals 0x5, the Watchpoint will stop the program. See Watch 1 shown below:

12. Watch expressions you can enter are detailed in the Help button in the Breakpoints window. Triggering on a data read or write is most common. You can leave out the value and trigger on just a Read and/or Write as you select.

13. To repeat this exercise, change **counter** to something other than 0x05 in the Watch window and click on RUN.

14. Stop the CPU. ⊗

15. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.

16. Exit Debug mode. 🄳
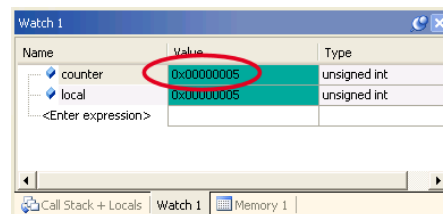
**TIP:** You cannot set Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

**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 be used with a Watchpoint. An example is: *((unsigned long *)0x20000004)

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com
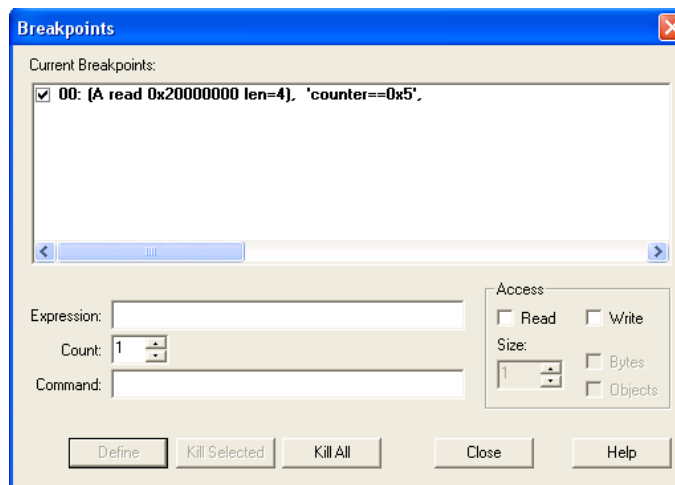
# 14)  MTB:  Micro Trace Buffer:

The Kinetis KL25 processor contains an instruction trace called MTB. The trace buffer is an area in the KL25 internal RAM which your program must not use.  The trace frames are stored here.  The size of this buffer is set in the file DBG_MTB.ini.

Instruction trace is very valuable in finding program flow bugs and other issues such as, but not limited to: race conditions and program crashes.  Trace is useful to determine the actual program flow such as when branches occurred (or not) and interrupt calls.  Trace displays the order the instructions were executed as opposed as the way they were written and displayed in the source files.  The Trace Data window shows the last instruction executed.  The Disassembly and source windows point to the next instruction or source line to be executed.

**Basic Instruction Trace Operation:**
The project Blinky.uvprojx is pre-configured to use MTB.  This exercise demonstrates the use of MTB.

1. Select CMSIS-DAP-MTB in the Target selector:

2. Compile the source files by clicking on the Rebuild icon .
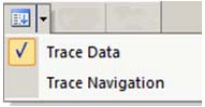
3. Program the Flash. .  Enter Debug mode.  The program runs to main().

4. Open the Trace Data window by clicking on the small arrow beside this icon:

5. You can also select View/Trace/Trace Data.

6. A window similar to this will be visible:  Size accordingly.  This is a record of the last number of instructions executed by the processor limited by the size of the microcontroller's internal RAM allocated to MTB.

7. Right click on any frame and select Show Functions.  The name of the function a frame belongs to will be displayed.

8. Note the last instruction executed was a BL.W to main() at  frame 1,670 (in this example).

9. In the Disassembly window shown below, the next instruction to be executed is a BL.W at 0x0242 to SystemCoreClockUpdate() which is the first function call in main().

10. Scroll up near Index 1,280 and you should see some C source lines.

**Tracking between Trace Data and Disassembly and Source Windows:**
1. In the example above, the last instruction executed was a BL.W to 0x0242.  This calls the beginning of main().
2. In the Registers window, the PC (R15) indeed points to 0x0242.
3. Cyan  is a marker from a source window to the Disassembly where it becomes a yellow highlight block.
4. The Yellow arrow  and Yellow triangle  is where the PC is pointing to.
5. Double-click in the Trace Data window on a frame and it is indicated in the Disassembly and source windows.

**Single-Stepping:**
1. Put the Disassembly window in focus by clicking inside it.
2. Click on Step (F11). 
3. The BL.W at 0x242 will be executed and added to the bottom of the Trace Data window.
   **Note:**  You might see slightly different addresses depending on your compiler and other settings
4. Examine the Disassembly and Blinky.c windows and see how the trace position is tracked in these windows.
5. The next instruction to be executed is a PUSH at memory 0x048A.  You can confirm the PC is equal to 0x048A in the Registers window.
6. Click on RUN and Step to see the effects of the executed instructions.  Pay particular attention to branch and stack operations such as POP and PUSH as they are faithfully recorded.  Scroll down to see the last frames.
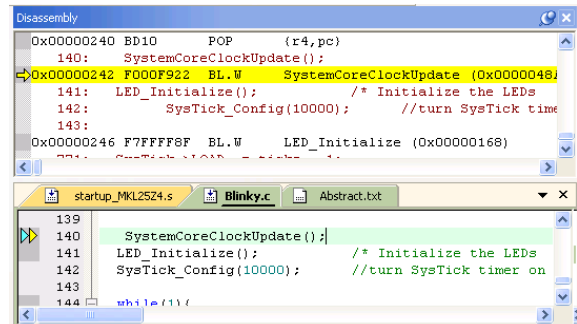
**TIP:** The trace frames can be saved to a file  and the Trace Data window can be cleared. .

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit                                    www.keil.com

## 15) Exploring the MTB Instruction Trace:

MTB provides a record of all instructions executed.  No CPU cycles are stolen to accomplish this.

Make sure all breakpoints and Watchpoints are removed.  Enter Ctrl-B or Debug/Breakpoints and select Kill All.

**Note:**  You might see different addresses as shown here depending on compiler settings.  These are using MDK 5.10.

1. Stop the CPU. ❌ Click on RESET. 🔧 Clear the Trace ▣. This makes it easier to see the frames we want.
2. In the Command window, enter "**g, main**" without the quotes and press Enter.  The program will run to main().
3. The Data Trace window as seen on the previous page will display.

**Note these items:**

1. The last instruction executed was a branch: BL.W main which is located at memory 0x0152.
2. Double-click on the POP at 1,669.  This is just above the BL.W.
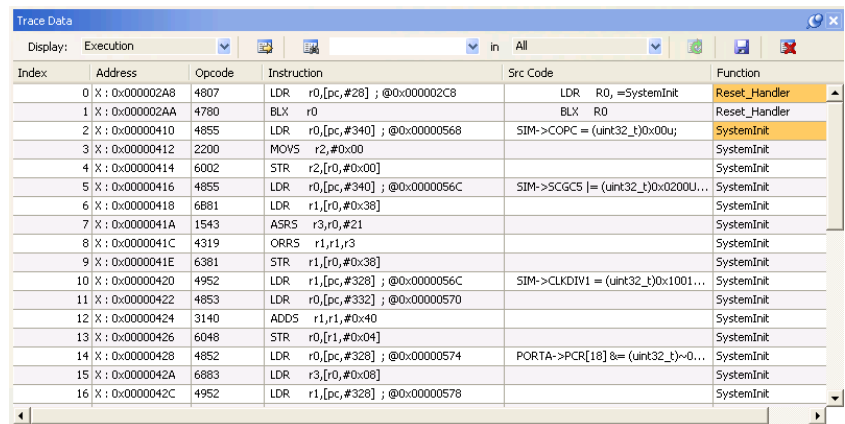3. Examine this instruction in the Disassembly window.  Note there is no clue this POP would result in the next instruction being BL.W.  The MTB has faithfully recorded that it was.  This is one of the powers of instruction trace.

 **TIP:** It is possible to have determined the next instruction after the POP by carefully examining the stack.  But using the trace, as you can see, is much easier and faster.  If the stack is destroyed by a program crash, the trace will still be available.

**Examine the Start of the Blinky Program:**
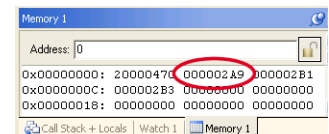
4. Scroll to the top of Trace Data.  We want to look at the very first instruction executed but it has been overrun.
5. Double-click on one of the instructions close to the top of the Trace Data window.
6. Set a breakpoint at this instruction in the Disassembly window.  It will be highlighted in yellow and with a red circle.
7. Exit and re-enter Debug mode. 🔍 🔍 The processor will stop at this breakpoint.  Remove this breakpoint.
8. Scroll to the top of the Trace Data window.  Note the Function column shows the functions the instructions belong to.
9. The first occurrence is blocked in orange to help you easily locate these.

10. Note the first instruction is at 0x2A8 and is an LDR instruction.  This is the very first instruction executed.

11. Open Memory 1 window and enter address 0x0.  Right click and select Unsigned Long.  This is the beginning of Flash memory.  See the window below:

12. In the window below, 0x00 is the Initial Stack Pointer (0x2000_0470).

13. 0x4 is the initial PC and is 0x2A9. Bit 0 indicates Thumb$^®$2 instruction so subtract 1 and you get 0x2A8. This is the address of the first instruction in the Trace which is the first instruction executed after RESET.



14. Click once somewhere in the Disassembly window.
15. Click on Step 🔁 (F11) and see the Blinky program step by each instruction executed and recorded in the Trace Data window.



16. Click once somewhere in the source window that is in focus.
17. Click on Step 🔁 (F11) and see program step by C source line and recorded in the Trace.  Multiple assembly instructions will/might be recorded depending on how many are associated with each particular C source line.
18. Exit Debug mode. 🔍

**TIP:** If **Run to main**() is not set in the Target Config window under the Debug tab, 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.

---

**13**

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit       www.keil.com

## 16) Trace Buffer Control:

Freescale's Cortex-M0+ has two methods of controlling the trace buffer and/or the program. They are:

**a) Stop the trace collection** when the trace buffer is full and **b) Stop the program execution** when the trace buffer is full.

**DBG_MTB.ini file and MTB Configuration:**

The MTB trace is activated and controlled by the ASCII file DBG_MTB.ini. It is executed when Debug mode is entered. To activate MTB to any project, just add this file as shown here. You must allocate the size of the internal RAM between your program and the trace buffer. Use the file from Blinky and not the RTX_Blinky project. A RESET function is added.

1. Select the Target Options icon. Select the Debug tab.
2. Note DBG_MTB.ini is entered in the Initialization File: box as shown:
3. Select Edit… to open it in μVision along with the other source files.
4. Click on OK to return to the main μVision menu.
5. At the window bottom: click on the Configuration Wizard tab.
6. An asterisk in the \DBG_MTB.ini tab indicates this file is not saved. It must be saved by selecting File/Save All or.
7. Click on Expand All. This is where MTB is configured.
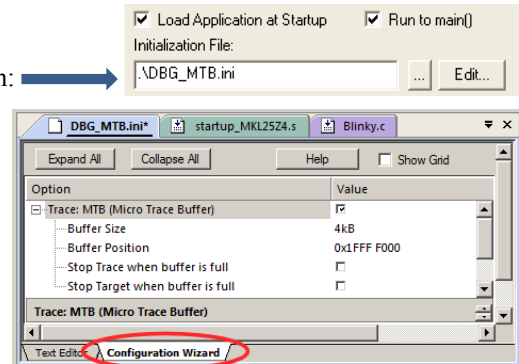
**a) Stop Trace when buffer is full:**

1. In DBG_MTB.ini, Select "Stop Trace when buffer is full". Select File/Save All or.
2. Select the Target Options icon. Select the Debug tab.
3. Unselect Run to main(). Click OK. After RESET, the program will not run. It will be at the initial PC address.
4. Enter Debug mode. Click on RUN. After a second or so, click on STOP.
5. The Trace Data window does not show the BL.W to main(). A Trace Gap frame is displayed: the trace was stopped.
6. Scroll to the top of the Trace Data and confirm the first instruction (LDR) SystemInit at 0x02A8 is recorded.

**b) Stop Target when buffer is full:**

1. Exit Debug mode.
2. In DGB_MTB.ini, Unselect "Stop Trace when buffer is full". Select "Stop Target when buffer is full".
3. Select File/Save All or. Enter Debug mode. Click on RUN.
4. The program will stop when the trace is full. The trace contains all the executed instructions from the first (0x2A8) at Index 0 to the last which is a LSLS 0x0458 at Index 1,559. You may get slightly different numbers depending on your compiler optimizations.

**When Finished:**

1. STOP the program. Exit Debug mode.
2. Select the Target Options icon. Select the Debug tab. Select Run to main(). Click OK.
3. In DGB_MTB.ini, Unselect "Stop Trace when buffer is full" and unselect "Stop Target when buffer is full".
4. Select File/Save All or.

**What are these features useful for ?**

The MTB trace will be over written as your program runs. It is possible, even probable, that the trace frames you want to examine will disappear. Using one of these two features can help you keep instructions in your field of interest.

**TIP:** Set a breakpoint on an exception vector (i.e. the Hard Fault if you end up here). If a fault occurs in your program, this will stop the program and also the trace collection. Otherwise the trace buffer will be full with only the Hard Fault instruction.

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com

## 17)  Trace Search:

With all the trace frames collected it can be difficult to find the frames you want.  The Trace Data window includes two useful search tools.

**Pull-Down Menu:**

1. Enter Debug mode. The Trace Data window will display some trace frames.
2. Click in the Trace Search window and enter a term such as push as shown here:
3. Press Enter and the PUSH instruction or any other occurrence of the word push will be highlighted.
4. Press Enter and each time this will advance to the next occurrence and be highlighted.
5. F3 will advance to the next highest occurrence and Shift-F3 advances to the preceding one.

**TIP:**  If "The text as specified below was not found:" is displayed, try clicking on any trace frame to bring them into focus.

**Find Trace:**

6. Click on the Find a trace record icon:  This window opens up:  You can enter search terms in the usual manner.

**TIP:**  You can also select the Find Trace window by clicking on one of the trace frames and press Ctrl-F.  Make sure the Find Trace window opens.
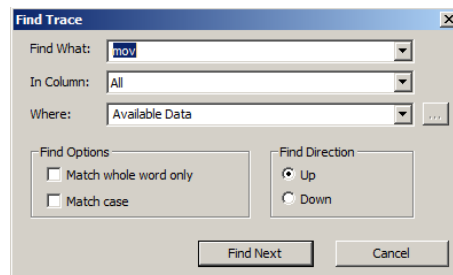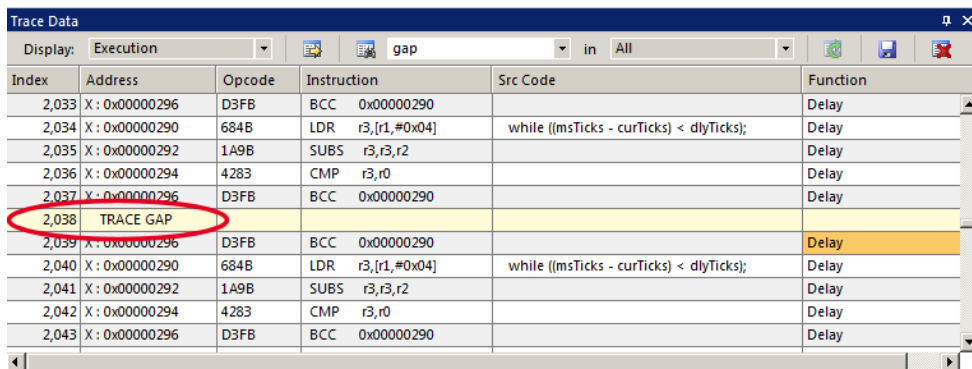
## 18)  Trace Data Wrap Around:

The MTB trace buffer is limited in size.  In our case, it is set to 4 KB.  We have seen so far approximately 1,500 to 2,000 trace frames that can be stored.  The actual number depends on the instruction size and other factors.  The trace frames coming from CoreSight are highly compressed and μVision reconstructs the trace data as you see it displayed.

As your program runs, old trace frames are over written and discarded by new ones.  When you stop the program, any trace frames present are saved/appended to a file.  The next run of trace frames is collected and when the program is stopped again, they are appended to the older frames and they are all displayed in the Trace Data window.  We will demonstrate this:

1. Clear the Trace Data window.  Click on RUN .  After a second or so, click on STOP .
2. There will be approximately 2,000 trace frames displayed in the Trace Data window.  Remember this number.
3. Click on RUN .  After a second or so, click on STOP .  Compare this Index to your last Index number.
4. Now there will be more trace frames:  more than there is processor internal RAM to store them…maybe 4,000.
5. Search the trace buffer for "gap": without the quotes using a method described above.
6. After the first "end of trace" (in my case it was 2,037) there will be a Trace Gap note as shown here:
7. Repeat a few runs and see that the number of trace frames increases.  Search for gap and note this appending method.
8. Each new set of trace frames is appended to the older set of preceding frames and are all displayed in the Trace Data.

**TIP:**  You must always remember that a Trace Gap represents an undeterminable number of instructions that are not recorded and hence lost.  You are not able to assume any executed instructions are linked over any Trace Gap.

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit

www.keil.com

## 19) More MTB Exploration:

Now we will look at some more interesting features of the MTB.
1. In the function LEDRed_On, set a breakpoint near line 67: `FPTD->PSOR    = led_mask[LED_GREEN];`

2. Clear the Trace [icon]. Click on RUN [icon].

3. The program will run to the breakpoint.

4. You will get a Trace Data window as the one below:

```
65  void LEDRed_On (void) {
66      FPTB->PSOR    = led_mask[LED_BLUE];
67      FPTD->PSOR    = led_mask[LED_GREEN];
68      FPTB->PCOR    = led_mask[LED_RED];
69  }
```

**Function Column:**
1. The executed instructions are displayed. The functions they are located in are noted by the Function column.

2. The beginning of a function run is highlighted in orange.

**Interrupt Subroutine Calls:**
1. The SysTick timer creates Interrupt 15. This is handled in the SysTick_Handler in Blinky.c.

2. Note the instructions executed a result of this interrupt are displayed at Index 2,013 through 2,017.

3. You can see these instructions reside from 0x196 to 0x19E. Double-click on a line and will show in source windows.

**Delay Function Interrupted:**
1. You can see the Delay function was interrupted by the SysTick interrupt 15 at Index 2,013 and it continued at 2,018.

2. This can be very useful information in locating tricky bugs.

**Continuation of a Function:**
1. Located at Index 2,023 is the POP instruction that is the end of the phaseD function. Double-click this line to highlight it in the Disassembly window. Where is the rest of this function in the trace buffer ?

2. If you search for phaseD, you will not find any other occurrence. It was not recorded. What happened is the Delay function, which is called by phaseD near line 131 in Blinky.c, swamps the trace buffer. The other instructions in phaseD were over written. You can confirm this by looking backwards in the trace until the next Trace Gap. If you need to record phaseD, you will have to set a breakpoint at a thoughtful spot in your program.

3. If you set a breakpoint on the first instruction in the line Delay(): ➡️
```
131:        Delay(0x100);
0x00000238 20FF        MOVS    r0,#0xFF
```

4. You will see all executed instructions of phaseD except for the Delay function and the POP at the very end of phaseD at address 0x240.

| 2,025 | X : 0x00000232 | B510 | PUSH | {r4,lr} | void phaseD (void) { | phaseD |
| 2,026 | X : 0x00000234 | F7FFFFD5 | BL.W | LED_Off (0x000001E2) | LED_Off(); | phaseD |

5. **Remove all breakpoints.** Click on them or you can enter Ctrl-B and select Kill All and then Close.

**Trace Data**

Display: Execution | gap | in | All

| Index | Address | Opcode | Instruction | Src Code | Function |
|---|---|---|---|---|---|
| 2,008 | X : 0x00000282 | D3FB | BCC    0x0000027C | | Delay |
| 2,009 | X : 0x0000027C | 684B | LDR    r3,[r1,#0x04] | while ((msTicks - curTicks) < dl... | Delay |
| 2,010 | X : 0x0000027E | 1A9B | SUBS   r3,r3,r2 | | Delay |
| 2,011 | X : 0x00000280 | 4283 | CMP    r3,r0 | | Delay |
| 2,012 | X : 0x00000282 | D3FB | BCC    0x0000027C | | Delay |
| 2,013 | X : 0x00000196 | 4840 | LDR    r0,[pc,#256] ; @0x00000298 | msTicks++;          /* in... | SysTick_Handler |
| 2,014 | X : 0x00000198 | 6841 | LDR    r1,[r0,#0x04] | | SysTick_Handler |
| 2,015 | X : 0x0000019A | 1C49 | ADDS   r1,r1,#1 | | SysTick_Handler |
| 2,016 | X : 0x0000019C | 6041 | STR    r1,[r0,#0x04] | | SysTick_Handler |
| 2,017 | X : 0x0000019E | 4770 | BX     lr | } | SysTick_Handler |
| 2,018 | X : 0x0000027C | 684B | LDR    r3,[r1,#0x04] | while ((msTicks - curTicks) < dl... | Delay |
| 2,019 | X : 0x0000027E | 1A9B | SUBS   r3,r3,r2 | | Delay |
| 2,020 | X : 0x00000280 | 4283 | CMP    r3,r0 | | Delay |
| 2,021 | X : 0x00000282 | D3FB | BCC    0x0000027C | | Delay |
| 2,022 | X : 0x00000284 | 4770 | BX     lr | } | Delay |
| 2,023 | X : 0x00000240 | BD10 | POP    {r4,pc} | } | phaseD |
| 2,024 | X : 0x00000276 | E7F6 | B      0x00000266 | while(1){ | main |
| 2,025 | X : 0x00000266 | F7FFFFC4 | BL.W    phaseA (0x000001F2) | phaseA();        //Re... | main |
| 2,026 | X : 0x000001F2 | B510 | PUSH   {r4,lr} | void phaseA (void) { | phaseA |
| 2,027 | X : 0x000001F4 | F7FFFFD4 | BL.W    LEDRed_On (0x000001A0) | LEDRed_On(); | phaseA |
| 2,028 | X : 0x000001A0 | 2101 | MOVS   r1,#0x01 | FPTB->PSOR  = led_mask[LED... | LEDRed_On |
| 2,029 | X : 0x000001A2 | 483C | LDR    r0,[pc,#240] ; @0x00000294 | | LEDRed_On |
| 2,030 | X : 0x000001A4 | 04C9 | LSLS   r1,r1,#19 | | LEDRed_On |
| 2,031 | X : 0x000001A6 | 6041 | STR    r1,[r0,#0x04] | | LEDRed_On |

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit        www.keil.com

## 20) 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 instructions executed prior to this usually catastrophic error is recorded to the limits of the trace buffer size.

A good way to crash your program is to make register LR = 0. When a BX lr is executed, the program will surely crash. There is a suitable section of code in the Blinky.c Delay function:  `while ((msTicks - curTicks) < dlyTicks); .`

1.  Enter Debug mode.   Click on RUN .  Click on STOP .  Delay should be visible in the Call Stack window.

2.  The program will probably be in the Blinky.c line (near 57): `while ((msTicks - curTicks) < dlyTicks);`

3.  Note in the Disassembly window near memory location 0x284, the last instruction of the Delay function is BX lr.

4.  In the Register window, change R14 (LR)  to 0x0 as shown here:

    ```
    R14 (LR)     0x00000000
    R15 (PC)     0x00000342
    ```

5.  Click on RUN .  The LEDS will not blink indicating a problem.  Click on STOP .  The program will be at the Hard Fault vector as shown here:

6.  This happened when the function Delay tried to return.

7.  The Trace Buffer will be mostly full of B instructions.

    ```
    266:                  EXPORT  SVC_Handler           [W.
    0x000002B2 E7FE       B       HardFault_Handler (0x000002B2)
    267:                  B       .
    ```

8.  *We want to stop the program when a hard fault occurs. Otherwise, a HardFault_handler which is by default a branch to itself, will fill up the trace buffer.*

9.  The PC in Blinky.c will be on the Hard Fault Handler.  It is usually around address 0x02B2 as shown above.

10. Set a breakpoint on this B instruction as shown above.

11. Exit and re-enter Debug mode to reset everything. 

12. Click on RUN .  Then, click on STOP .  Clear the trace buffer.   This is to make things look clearer.

13. Set R14 (LR) in the Registers window to zero as before.

14. Click on RUN .

15. The program will immediately go to the Hard Fault state and the breakpoint will stop execution at the Branch.

16. The Call Stack window will not show much useful information as the Stack was destroyed on purpose in this case.

17. The Trace Data now shows the last number of instructions executed plus the BX instruction.  Usually the last one listed is the one that caused the crash. But not always.  Clearly, in this case, you can see the sequence of instructions that caused the fault.

18. Note you can see the execution of the SysTick Handler.  This type of event is very hard to determine without trace.

19. Click on Step (F11) a few times and the B at the HardFault_Handler will be executed and displayed as shown below

20. **Remove the breakpoint.**

21. Exit Debug mode. 



| Index | Address | Opcode | Instruction | Src Code | Function |
|---|---|---|---|---|---|
| 2,023 X : 0x00000280 | 4283 | CMP r3,r0 | | | Delay |
| 2,024 X : 0x00000282 | D3FB | BCC 0x0000027C | | | Delay |
| 2,025 X : 0x00000196 | 4840 | LDR r0,[pc,#256] ; @0x00000298 | msTicks++; | /* in... | SysTick_Handler |
| 2,026 X : 0x00000198 | 6841 | LDR r1,[r0,#0x04] | | | SysTick_Handler |
| 2,027 X : 0x0000019A | 1C49 | ADDS r1,r1,#1 | | | SysTick_Handler |
| 2,028 X : 0x0000019C | 6041 | STR r1,[r0,#0x04] | | | SysTick_Handler |
| 2,029 X : 0x0000019E | 4770 | BX lr | } | | SysTick_Handler |
| 2,030 X : 0x0000027C | 6840 | LDR r3,[r1,#0x04] | while ((msTicks - curTicks) < di... | | Delay |
| 2,031 X : 0x0000027E | 1A9B | SUBS r3,r3,r2 | | | Delay |
| 2,032 X : 0x00000280 | 4283 | CMP r3,r0 | | | Delay |
| 2,033 X : 0x00000282 | D3FB | BCC 0x0000027C | | | Delay |
| 2,034 X : 0x00000284 | 4770 | BX lr | } | | Delay |

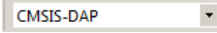| 2,034 | X : 0x00000284 | 4770 | BX lr | } | Delay |
|---|---|---|---|---|---|
| 2,035 | X : 0x000002B2 | E7FE | B HardFault_Handler (0x000002B2) | B . | HardFault_Handler |
| 2,036 | X : 0x000002B2 | E7FE | B HardFault_Handler (0x000002B2) | B . | HardFault_Handler |
| 2,037 | X : 0x000002B2 | E7FE | B HardFault_Handler (0x000002B2) | B . | HardFault_Handler |

**TIP:**  Remember, a CoreSight hardware breakpoint does not execute the instruction it is set to.

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

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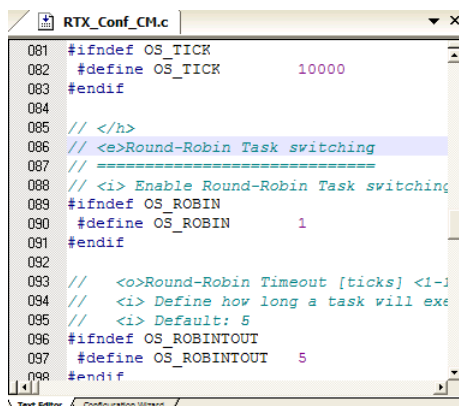
Keil provides RTX, a full feature RTOS.  RTX is included as part of Keil MDK including source.  It can have up to 255 tasks and no royalty payments are required.  This example explores the RTX RTOS project.  MDK will work with any RTOS.  An RTOS is just a set of C functions that gets compiled with your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

This tutorial uses the version of RTX that is CMSIS-RTOS compliant.  It is included in MDK 5.10 and later.

We highly recommend you get the  *NEW!*  **Getting Started MDK 5:**    www.keil.com/mdk5/.

1.  With µVision in Edit mode (not in debug mode):  Select Project/Open Project.

2.  Open the file C:\MDK\Boards\Freescale\FRDM-KL25Z\RTX_Blinky\Blinky.uvprojx.

3.  Select the debug adapter you are using.  This exercise uses CMSIS-DAP.   `CMSIS-DAP`

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 the four tasks or threads representing a stepper motor driver.  Because there are only three LEDs and four threads, the blue LED comes on twice in the sequence.

8.  Click on STOP  .

### The Configuration Wizard for RTX:

1.  Double-click on RTX_Conf_CM.c in the Project window to open it.  Note the source code to configure RTX.

2.  Click on the Configuration Wizard tab at the bottom and your view will change to the Configuration Wizard.

3.  Click on Expand All to 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.  *Do not change anything at this time !*

6.  Changing an attribute in one tab changes it in the other automatically.  Save changes with File/Save All.

7.  You can create Configuration Wizards in any source file with the scripting language as seen in the Text Editor.

8.  This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>.
    See www.keil.com/support/docs/2735.htm for instructions to add this to your own source code.

<div align="center">

**Text Editor: Source Code**          **Configuration Wizard**

</div>

---

**18**

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com
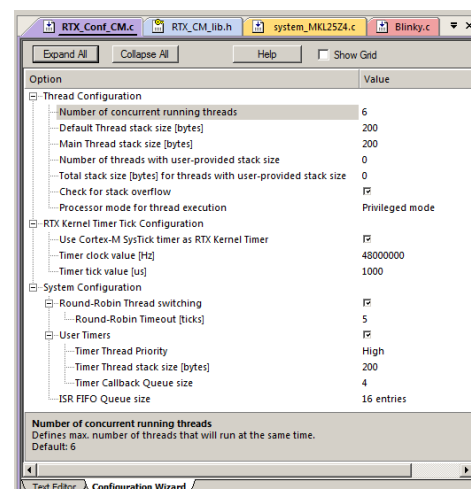
## 22)  RTX Kernel Awareness using RTX System and Thread Viewer

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.  Keil provides two such windows with the Kinetis Cortex-M4.  Other RTOS companies also provide awareness plug-ins for µVision.

µVision has extensive kernel awareness windows for MQX.  See www2.keil.com/freescale/mqx.

1.  Run RTX_Blinky by clicking on the Run icon.

2.  Open Debug/OS Support and select System and Thread Viewer.           ✓  System and Thread Viewer

3.  The window below opens up.  You 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 CoreSight technology as used in the Watch and Memory windows.

4.  Select View and select Periodic Window Update if these values do not change:           ✓  Periodic Window Update

1.  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.  Most of the time the CPU is executing the os_idle_demon.  The processor spends relatively little time in each task.  You can change this to suit your needs.

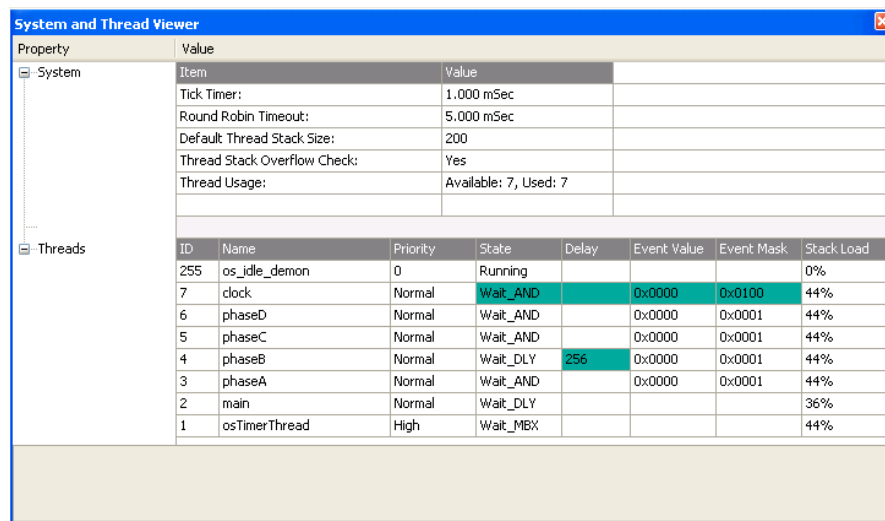| System and Thread Viewer | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Property** | **Value** | | | | | | |
| ⊟ System | **Item** | | **Value** | | | | |
| | Tick Timer: | | 1.000 mSec | | | | |
| | Round Robin Timeout: | | 5.000 mSec | | | | |
| | Default Thread Stack Size: | | 200 | | | | |
| | Thread Stack Overflow Check: | | Yes | | | | |
| | Thread Usage: | | Available: 7, Used: 7 | | | | |
| ⊟ Threads | **ID** | **Name** | **Priority** | **State** | **Delay** | **Event Value** | **Event Mask** | **Stack Load** |
| | 255 | os_idle_demon | 0 | Running | | | | 0% |
| | 7 | clock | Normal | Wait_AND | | 0x0000 | 0x0100 | 44% |
| | 6 | phaseD | Normal | Wait_AND | | 0x0000 | 0x0001 | 44% |
| | 5 | phaseC | Normal | Wait_AND | | 0x0000 | 0x0001 | 44% |
| | 4 | phaseB | Normal | Wait_DLY | 256 | 0x0000 | 0x0001 | 44% |
| | 3 | phaseA | Normal | Wait_AND | | 0x0000 | 0x0001 | 44% |
| | 2 | main | Normal | Wait_DLY | | | | 36% |
| | 1 | osTimerThread | High | Wait_MBX | | | | 44% |

**Demonstrating States:  (note:** Tasks and Threads are used interchangeable in Keil MDK**)**

Blinky.c contains four threads that blink the LEDs.  Thread 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 Thread 1 as shown:  (but not on the for (;;;) line)

3.  Set a breakpoint in one other thread.

4.  Click on RUN.

```
92  /*-----------------------------------------------------------------
93  *       Thread 1 'phaseA': Phase A output
94  *-----------------------------------------------------------------*/
95  void phaseA (void const *argument) {
96    for (;;) {
97      osSignalWait(0x0001, osWaitForever);    /* wait for an event flag 0x0001 */
98      LEDRed_On();
99      osDelay(1000);                          /* delay 1000ms                  */
100     signal_func(tid_phaseB);                /* call common signal function   */
101   }
102 }
```

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.  The window above shows the program stopped and phaseA is running.  The states of the other tasks are displayed as well as other useful information.

6.  Click on RUN.  The other thread will show as "Running".  Each time you click RUN, the next thread will run.

7.  Remove the breakpoints and close the RTX Tasks window.

### More Information of using RTX:

It is very beneficial to use an RTOS.  RTX is a good choice.  It is small, efficient and easy to use yet it is full featured.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit                www.keil.com
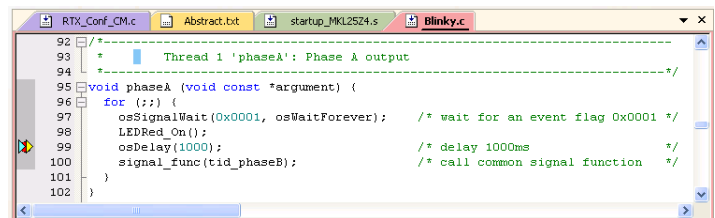
## 23) Blinky Project File System Explanation:

You can use the Blinky project as a template for your own projects.
For more information regarding creating your own projects:
See the **Getting Started: Create Applications with MDK Version 5** available on www.keil.com/mdk5

The Project window displays the files found in a given project as shown here:

1. If you double click on a file, it will be opened in the source section of µVision.

2. The Target setting is displayed at the top: It is CMSIS-DAP in this case: This is selected by the Select Target box in the main µVision menu. You can create your own Target Setting by selecting Project/Manage/Components,…

3. Files are arranged by Groups: such as Source Files, CMSIS and Device etc.

**Adding Files:**

1. To add an existing source file to a Group, right click on a Group name (i.e. Source Files) and select: `Add Existing Files to Group 'Source Files'...` Select an existing source file. This file will be added to the Group.

4. To create and add a new file to a Group, right click on a Group name and select: `Add New Item to Group 'Source Files'...`. The window below right opens up. Select the type of file you want created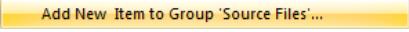. Enter a name for it as shown (I chose DSP) and select Add. A blank file will be created and added to the project. This shown below on the bottom left:
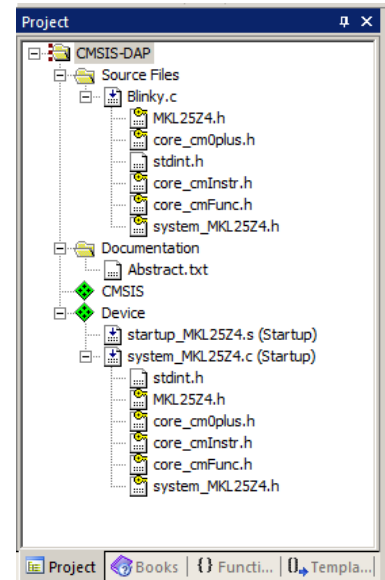
**File Description by Groups:**

**Source Files:**

5. **Blinky.c:** The file containing the main() function. You can change the name of this file.

**Documentation:**

6. **Abstract.txt:** a description of the program detailing its attributes.

**CMSIS:**

**Empty.** RTX and other files can go here.

**Device:** *These are CMSIS files and are compliant with CMSIS 2.0.*

7. **startup_MKL25Z4.s:** Creates initial SP and PC and exception vector table plus more.

8. **system_MKL25Z4.c:** Configures PLL and clock systems.

Header files used by the source files are not described. To view them, just double click on them and they will be opened in the source window.
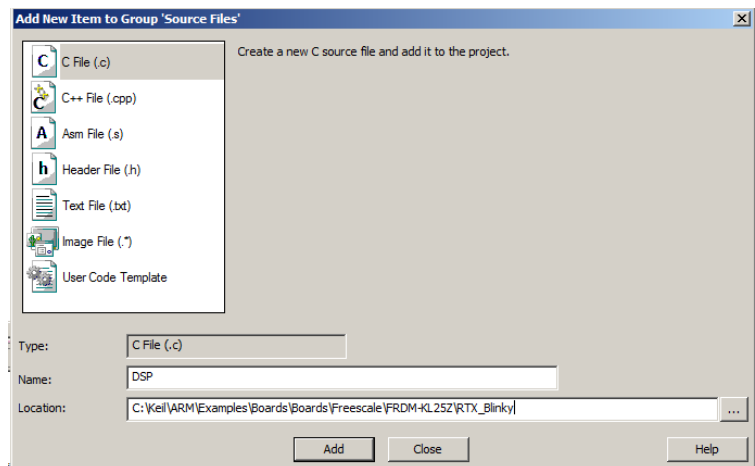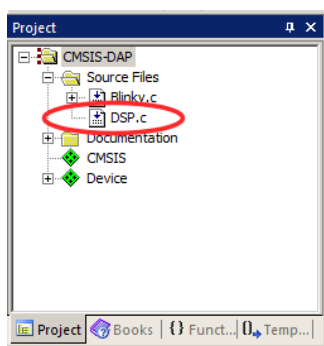
**TIP:** The icons with the yellow key means this file is read-only. If you need to modify such a file, you will need to find it and modify its permissions. It might be in such a directory: C:\Keil_v5\ARM\Pack\Keil\Kinetis_KLxx_DFP\1.0.0\Device

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit

## 24) RTX_Blinky Project File System Explanation:

You can use the RTX_Blinky project as a template for your own projects.

For more information regarding RTX:
See the **Getting Started: Create Applications with MDK Version 5** available on www.keil.com/mdk5

The Project window displays the files found in a given project as shown here:

1. If you double click on a file, it will be opened in the source section of µVision.

2. The Target setting is displayed at the top: It is CMSIS-DAP in this case: This is selected by the Select Target box in the main µVision menu. You can create your own Target Setting by selecting Project/Manage/Components,…

3. Files are arranged by Groups: such as Source Files, CMSIS and Device etc.

**Adding Files:**

2. To add an existing source file to a Group, right click on a Group name (i.e. Source Files) and select:  Add Existing Files to Group 'Source Files'…  Select an existing source file. This file will be added to the Group.

4. To create and add a new file to a Group, right click on a Group name and select: Add New Item to Group 'Source Files'… . The window below right opens up. Select the type of file you want created. Enter a name for it as shown (I chose DSP) and select Add. A blank file will be created and added to the project. This shown below on the bottom left:

**File Description by Groups:**

**Source Files:**

5. **Blinky.c:** The file containing the main() function.

**Documentation:**

6. **Abstract.txt:** a description of the program detailing its attributes.

**CMSIS:** *These are CMSIS files and are compliant with CMSIS 2.0.*
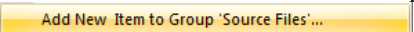
7. **RTX_Conf_CM.c:** Contains configuration item for RTX.

8. **cmsis_os.h:** header file needed for RTX RTOS.

**Device:**

9. **startup_MKL25Z4.s:** Creates initial SP and PC and exception vector table plus more.

10. **system_MKL25Z4.c:** Configures PLL and clock systems.

Header files used by the source files are not described. To view them, just double click on them and they will be opened in the source window.

**TIP:** The icons with the yellow key  means this file is read-only. If you need to modify such a file, you will need to find it and modify its permissions.

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com
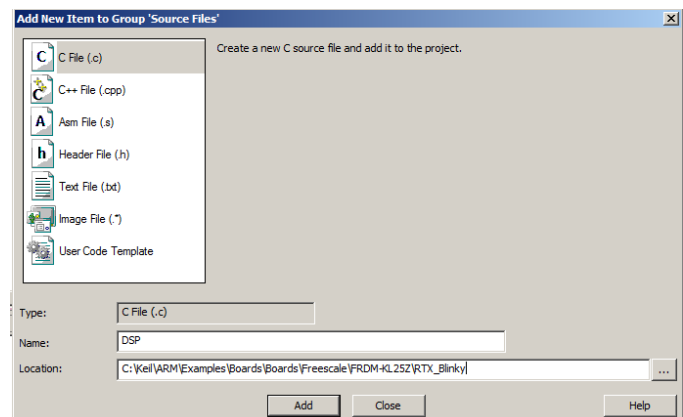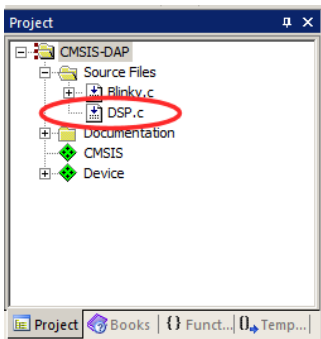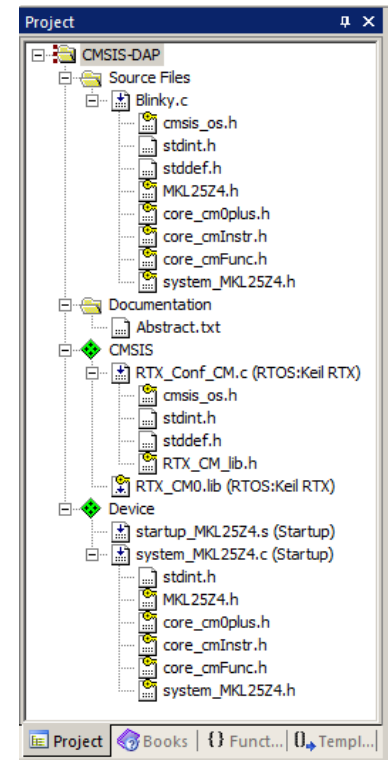
## 25) DSP SINE example using ARM CMSIS-DSP Libraries:

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 and www.onarm.com/cmsis/download/ .
CMSIS is an acronym for Cortex Microcontroller Software Interface Standard.

This example creates a sine wave to which noise is added, and then the noise is filtered out leaving the original sine wave.

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** and copy the MDK 5 version into C:\MDK\Boards\Freescale\FRDM-KL25Z. A \DSP5 directory will be created. You should have already done this.

1. Stop the program and exit Debug mode if necessary.

2. Open the project file sine: C:\MDK\Boards\Freescale\FRDM-KL25Z\DSP5\sine.uvprojx.

3. Select CMSIS-DAP in the Target Selector:

4. Build the files. There will be no errors or warnings.

5. Program the KL25Z flash by clicking on the Load icon: Progress will be indicated in the Output Window.

6. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.

7. Click on the RUN icon.

8. Open Watch 1 by selecting View/Watch/Watch 1 if necessary.

9. Four global variables will be displayed in Watch 1 as shown here: If these variables are changing the program is likely working properly.

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

**Serial Wire Viewer (SWV) and ETM Instruction Trace:** (only for reference)

The Kinetis Cortex-M4 processors have Serial Wire Viewer (SWV). The four waveforms of the global variables shown above 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.

If you use a Kinetis Cortex-M4 and with any Keil ULINK or a J-Link, you can use this Logic Analyzer windows plus many other Serial Wire Viewer (SWV) features. Kinetis Coretx-M4 processors also have ETM trace.

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

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com
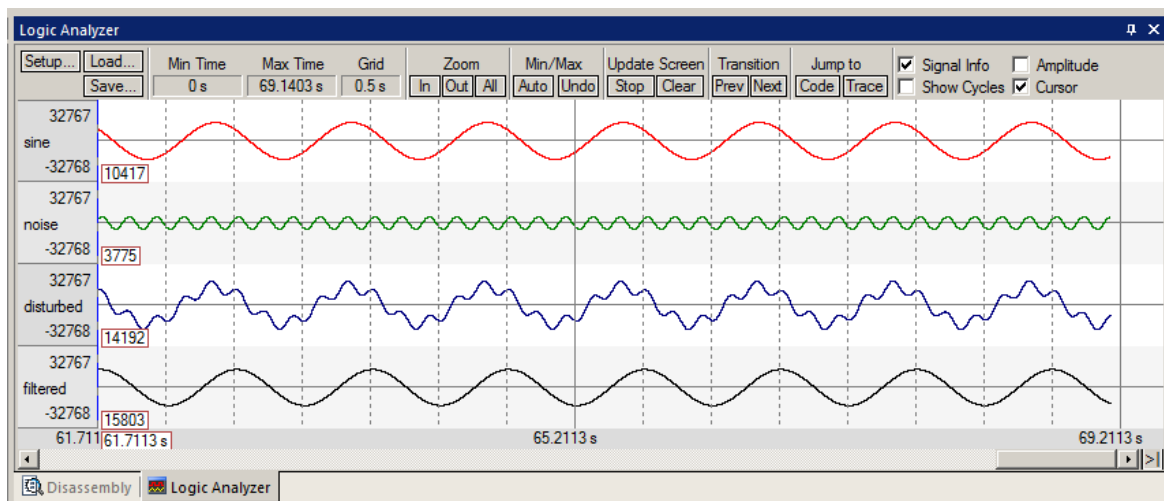
**RTX Tasks and System:**

2. Click on the RUN icon.

3. Open Debug/OS Support and select System and Thread Viewer.  A window similar to below opens up.

4. This window updates in real-time.  Note nearly all the processor time is spent in the idle daemon os_idle_demon. The processor spends relatively little time in each thread.  You can change this if you need to.

5. Set a breakpoint in one of the threads in DirtyFilter.c by clicking in the left margin on a grey area.

6. Here are the four threads with their starting line numbers:

   **1)** sine_gen (87)  **2)** noise_gen (105)  **3)** disturb_gen (125)  **4)** filter_tsk (145)

7. The program will stop here and the Task window will be updated accordingly.  Here, I set a breakpoint in the noise_gen task:  You can see that noise_gen was running when the breakpoint was activated.

8. Os_idle_demon is Ready to run when noise_gen is finished and no other task is Ready.

**System and Thread Viewer**

| Property | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| System | Item | | Value | | | | |
| | Tick Timer: | | 1.000 mSec | | | | |
| | Round Robin Timeout: | | 5.000 mSec | | | | |
| | Default Thread Stack Size: | | 200 | | | | |
| | Thread Stack Overflow Check: | | Yes | | | | |
| | Thread Usage: | | Available: 6, Used: 6 | | | | |
| | | | | | | | |
| Threads | ID | Name | Priority | State | Delay | Event Value | Event Mask | Stack Load |
| | 255 | os_idle_demon | 0 | Ready | | | | 32% |
| | 6 | sync_tsk | Normal | Wait_AND | | 0x0000 | 0x0001 | 44% |
| | 5 | filter_tsk | Normal | Wait_AND | 65384 | 0x0000 | 0x0001 | 44% |
| | 4 | disturb_gen | Normal | Wait_AND | 65284 | 0x0000 | 0x0001 | 44% |
| | 3 | noise_gen | Normal | Running | 65532 | 0x0000 | 0x0001 | 8% |
| | 2 | sine_gen | Normal | Wait_DLY | 50 | 0x0000 | 0x0001 | 48% |
| | 1 | main | Normal | Wait_DLY | | | | 36% |

**TIP:**  os_idle_demon has a Priority of 0 which is the lowest priority possible.  Every other task has a higher priority.

9. Set another breakpoint in a different task.  Click on the RUN icon.

10. Each time you click on RUN, the program will stop at one of the two tasks and this is indicated by the Running state.

**TIP:**   Recall this window uses the CoreSight DAP read and write technology to update this window in real-time.

**TIP:**   You might have noticed the Event Viewer in Debug/OS Support.  This uses Serial Wire Viewer to get its information and this feature is available on the Kinetis Cortex-M4 processors with any Keil ULINK debug adapter.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit

## Call Stack and Locals:

1. Click on the Call Stack + Locals tab. This window opens up:

2. Each time you click on RUN the information is updated depending on which thread is running.

3. Right click on an element and select Callee or Caller Code to go there:

> Show Caller Code
> **Show Callee Code**
> ☑ Hexadecimal Display
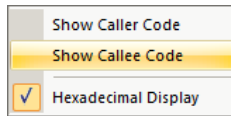
**TIP:** Recall the Call Stack and Locals window updates only when the program is stopped by one of the two breakpoints that were set on the previous page.

## MTB Instruction Trace:

1. Exit Debug mode.
2. Select CMSIS-DAP MTB in the Target Selector:

   CMSIS-DAP-MTB ▼

3. Enter Debug mode. The breakpoints will be preserved. μVision will complain because it wants to use one breakpoint in its run to main() operation. Ignore this warming.

4. Click on RUN. It will take a few seconds before the breakpoint is hit.

5. Open the Trace Data window: click on the small arrow to open and select Trace Data as shown:

> ☑ Trace Data
> Trace Navigation

6. Right click inside Trace Data and select Show Functions.

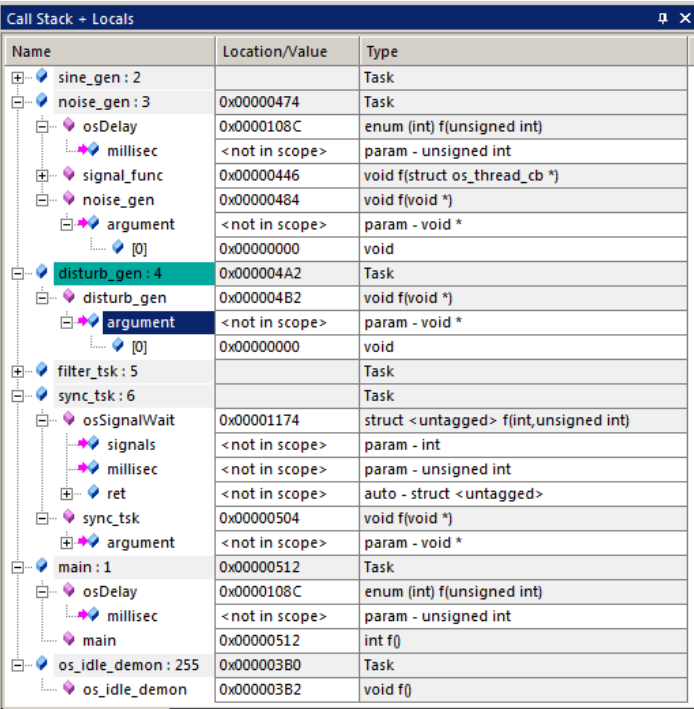7. This window was when noise_gen was Running as shown in the System and Thread Viewer window.

8. Each time you click on RUN the trace will be updated. Makes sure you scroll to the bottom each time.

9. Remove the breakpoints when you have finished this page.

**TIP:** Search for "gap" to find when the trace frame collection was stopped as discussed before.

**Call Stack + Locals**

| Name | Location/Value | Type |
|---|---|---|
| ⊞ sine_gen : 2 | | Task |
| ⊟ noise_gen : 3 | 0x00000474 | Task |
| osDelay | 0x0000108C | enum (int) f(unsigned int) |
| millisec | <not in scope> | param - unsigned int |
| ⊞ signal_func | 0x00000446 | void f(struct os_thread_cb *) |
| ⊟ noise_gen | 0x00000484 | void f(void *) |
| ⊟ argument | <not in scope> | param - void * |
| [0] | 0x00000000 | void |
| ⊟ disturb_gen : 4 | 0x000004A2 | Task |
| ⊟ disturb_gen | 0x000004B2 | void f(void *) |
| ⊟ argument | <not in scope> | param - void * |
| [0] | 0x00000000 | void |
| ⊞ filter_tsk : 5 | | Task |
| ⊟ sync_tsk : 6 | | Task |
| ⊟ osSignalWait | 0x00001174 | struct <untagged> f(int,unsigned int) |
| signals | <not in scope> | param - int |
| millisec | <not in scope> | param - unsigned int |
| ⊞ ret | <not in scope> | auto - struct <untagged> |
| ⊟ sync_tsk | 0x00000504 | void f(void *) |
| ⊞ argument | <not in scope> | param - void * |
| ⊟ main : 1 | 0x00000512 | Task |
| ⊟ osDelay | 0x0000108C | enum (int) f(unsigned int) |
| millisec | <not in scope> | param - unsigned int |
| main | 0x00000512 | int f() |
| ⊟ os_idle_demon : 255 | 0x000003B0 | Task |
| os_idle_demon | 0x000003B2 | void f() |

☐ Call Stack + Locals | Watch 1 | ☐ Memory 1

**Trace Data**

Display: Execution ▼      in All ▼

| Index | Address | Opcode | Instruction | Src Code | Function |
|---|---|---|---|---|---|
| 4,687 | X : 0x00001954 | D002 | BEQ 0x0000195C | | rt_stk_check |
| 4,688 | X : 0x0000195C | BD10 | POP {r4,pc} | } | rt_stk_check |
| 4,689 | X : 0x0000019A | BC0C | POP {r2-r3} | POP {R2,R3} | SVC_Handler |
| 4,690 | X : 0x0000019C | 601A | STR r2,[r3,#0x00] | STR R2,[R3] ; os_tsk.run = os_tsk.new | SVC_Handler |
| 4,691 | X : 0x0000019E | 6A90 | LDR r0,[r2,#0x28] | LDR R0,[R2,#TCB_TSTACK] ; os_tsk.new->tsk_stack | SVC_Handler |
| 4,692 | X : 0x000001A0 | 3010 | ADDS r0,r0,#0x10 | ADDS R0,R0,#16 ; Adjust Start Address | SVC_Handler |
| 4,693 | X : 0x000001A2 | C8F0 | LDM r0!,{r4-r7} | LDMIA R0!,{R4-R7} ; Restore new Context (R8-R11) | SVC_Handler |
| 4,694 | X : 0x000001A4 | 46A0 | MOV r8,r4 | MOV R8,R4 | SVC_Handler |
| 4,695 | X : 0x000001A6 | 46A9 | MOV r9,r5 | MOV R9,R5 | SVC_Handler |
| 4,696 | X : 0x000001A8 | 46B2 | MOV r10,r6 | MOV R10,R6 | SVC_Handler |
| 4,697 | X : 0x000001AA | 46BB | MOV r11,r7 | MOV R11,R7 | SVC_Handler |
| 4,698 | X : 0x000001AC | F3808809 | MSR PSP,r0 | MSR PSP,R0 ; Write PSP | SVC_Handler |
| 4,699 | X : 0x000001B0 | 3820 | SUBS r0,r0,#0x20 | SUBS R0,R0,#32 ; Adjust Start Address | SVC_Handler |
| 4,700 | X : 0x000001B2 | C8F0 | LDM r0!,{r4-r7} | LDMIA R0!,{R4-R7} ; Restore new Context (R4-R7) | SVC_Handler |
| 4,701 | X : 0x000001B4 | 2002 | MOVS r0,#0x02 | MOVS R0,#:NOT:0xFFFFFFFD ; Set EXC_RETURN value | SVC_Handler |
| 4,702 | X : 0x000001B6 | 43C0 | MVNS r0,r0 | MVNS R0,R0 | SVC_Handler |
| 4,703 | X : 0x000001B8 | 4700 | BX r0 | BX R0 ; RETI to Thread Mode, use PSP | SVC_Handler |
| 4,704 | X : 0x00001174 | BD80 | POP {r7,pc} | } | osSignalWait |
| 4,705 | X : 0x00000480 | 466B | MOV r3,sp | | noise_gen |
| 4,706 | X : 0x00000482 | C307 | STM r3!,{r0-r2} | | noise_gen |

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit      www.keil.com

# 26) Creating your own MDK 5 project from scratch:

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. Once you have the new project configured; you can build, load and run a bare Blinky example. It will have 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.

**Install the Kinetis Software Pack for your processor:**

1. Start µVision and leave in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Pack for the KL25Z processor must be installed. This has already been done on page 4.
3. You do not need to copy any examples over.

**Create a new Directory and a New Project:**

1. In the main µVision menu, click on Project/New µVision Project…
2. In the window that opens, shown below, go to the folder C:\MDK\Boards\Freescale\FRDM-KL25Z\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj.
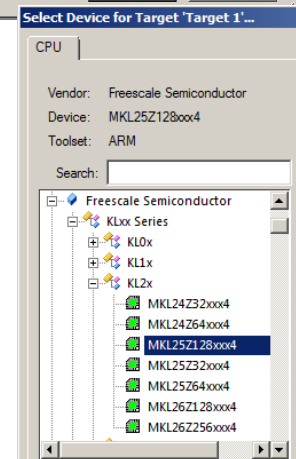7. As soon as you click on Save, the next window opens:
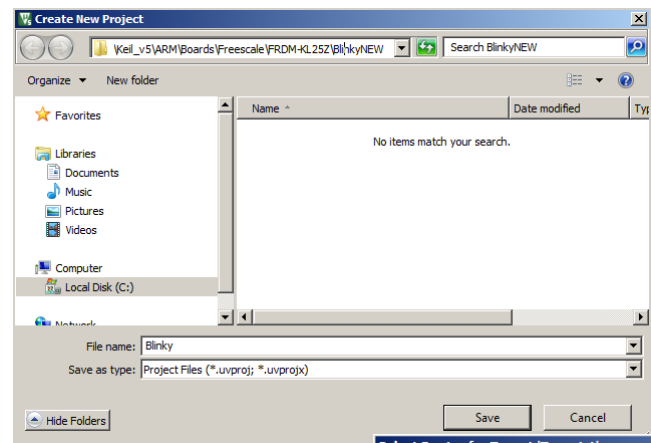
**Select the Device you are using:**

1. Expand Freescale Semiconductor and then KLxx Series, then KL2x and then select MKL25Z128xxx4 as shown here:

   **TIP:** Chip icons in colour are from the Software Packs. Any grey icons are from MDK 4.7x.

2. Click OK and the Manage Run Time window shown below bottom right opens.

**Select the CMSIS components you want:**

1. Expand all the items and select CORE and Startup as shown below. They will be highlighted in Green indicating there are no other files needed. Click OK.
2. Click on File/Save All or select the Save All icon:
3. The project Blinky.uvproj. will now be changed to Blinky.uvprojx.
4. You now have a new project list as shown on the bottom left below: The appropriate CMSIS files you selected have been automatically entered and configured.
5. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
6. Click once on it and change its name to CMSIS-DAP and press Enter. The Target selector name will also change.

**What has happened to this point:**

You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files.

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit

www.keil.com
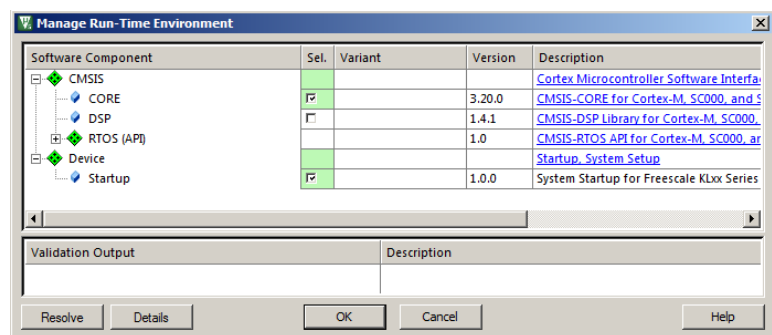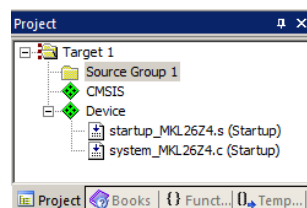
**Create a blank C Source File:**

1. Right click on Source Group 1 in the Project window and select [Add New Item to Group 'Source Files'...].
2. This window opens up:
3. Highlight the upper left icon:  C file (.c):
4. In the Name: field, enter Blinky.
5. Click on Add to close this window.

6. Click on File/Save All or

7. Expand  Source Group 1 in the Project window and Blinky.c will now display.

8. It will also open in the Source window.
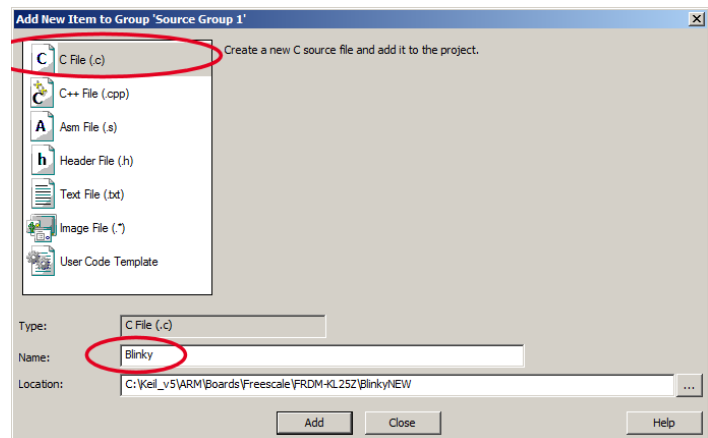
**Add Some Code to Blinky.c:**

9. In the blank Blinky.c, add the C code below:

10. Click on File/Save All or

11. Build the files.  There will be no errors or warnings if all was entered correctly.

```
#include <MKL25Z4.h>

unsigned int counter = 0;

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

  while(1) {
       counter++;
       if (counter > 0x0F) counter = 0;
       }
}
```
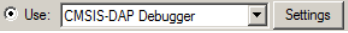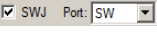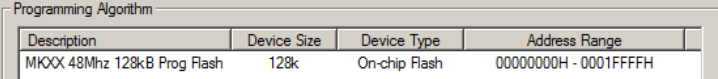
**TIP:**  You can also add existing source files: [Add Existing Files to Group 'Source Files'...]

**Configure the Target CMSIS-DAP:**  *Please complete these instructions carefully to prevent unusual problems…*

1. Select the Target Options icon .  Select the **Target** tab.

2. Enter 8 in Xtal (MHz).  This is used for timing calculations.  Select Use MicroLIB to optimize for smaller code size.

3. Select the **Output** tab.  Click on Select Folder for Objects…: [Select Folder for Objects...]

4. In the Browse for Folder window that opens: right click and create a new folder called Flash.

5. Double click on Flash to enter this folder and click OK.  Compilation files will now be stored in this Flash folder.

6. Click on the **Listings** tab.  Click on Select Folder for Objects…:  Double click on Flash and click OK to close.

7. Click on the **Debug** tab.  Select CMSIS-DAP Debugger in the Use: box: [Use: CMSIS-DAP Debugger  Settings]

8. Select Settings: icon beside Use:  CMSIS-DAP.

9. Select SWJ and SW as shown here: [☑ SWJ  Port: SW]  A JTAG selection here will return an RDDI error.  If your KL25Z is connected to your PC, you should now see a valid IDCODE and Device Name in the SW Device box.

10. Click on OK *once* to go back to the Target Configuration window.

11. Click on the **Utilities** tab.  Select Settings and confirm the correct Flash algorithm is present:  Shown is the correct one for the Freedom KL25Z board:

| Description | Device Size | Device Type | Address Range |
|---|---|---|---|
| MKXX 48Mhz 128kB Prog Flash | 128k | On-chip Flash | 00000000H - 0001FFFFH |

12. Click on OK twice to return to the main menu.

13. Click on File/Save All or

14. Build the files.  There will be no errors or warnings if all was entered correctly.  If there are, please fix them !

**The Next Step ?  Let us run your program and see what happens !  Please turn the page….**

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com

**Running Your Program:**

1. Program the KL25 Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.

2. Enter Debug mode by clicking on the Debug icon .

3. Click on the RUN icon.  Note: you stop the program with the STOP icon.

4. Right click on counter in Blinky.c and select Add counter to … and select Watch 1.

5. counter should be updating as shown here:

6. You can also set a breakpoint in Blinky.c and the program should stop at this point if it is running properly. If you do this, remove the breakpoint.

7. You should now be able to add your own source code to create a meaningful project.
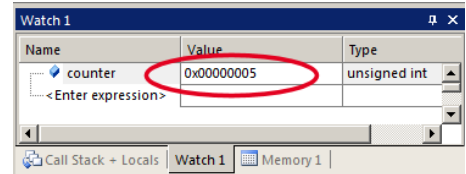
**TIP:** The Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip sequential values you know must exist.

**Creating a New Target Setting:** This Target Configuration will activat the MTB Instruction Trace feature:

1. Stop the program.  Remove any breakpoints (Ctrl-B, Kill All, Close). Exit Debug mode.
2. Select Project/Manage/Components, Environment, Books… and this window opens up:
3. Select the Insert icon  or press Insert key on the PC.
4. Enter CMSIS-DAP MTB and press Enter. Click OK to exit.
5. Open the Target Selector and you will now find CMSIS-DAP MTB visible:
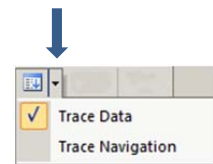6. Select CMSIS-DAP MTB.
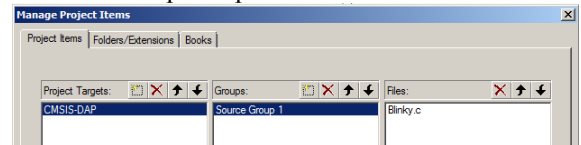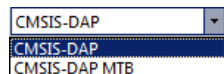
**What this means:** We now have two targets: CMSIS-DAP and CMSIS-DAP MTB. Each one points to its own Target Options configuration. At this point they are the same. We will modify CMSIS-DAP MTB to activate the MTB trace.

7. Select the Target Options icon . Select the **Debug** tab. Note the Initialization File: box is empty.
8. Using the Browse icon, go to the directory C:\MDK\Boards\Freescale\FRDM-KL25Z\Blinky5 and insert DBG_MTB.ini as shown here:
9. Click OK to exit this window. MTB is now activated with the defaults.

**TIP:** In real life you are probably better to copy this file into your own project.

10. Click on File/Save All or  to save all your work so far.

11. Enter Debug mode.  The program will Run to main() automatically.
12. Open the Trace Data window: View/Trace/Trace Data or the small arrow beside the icon:
13. The Trace Data window will be full of trace frames. Right click inside it and select Show Functions.

**What this means:** A collection of Target Options is saved in the Target Setting CMSIS-DAP. Another one, with the MTB activated, is saved with CMSIS-DAP MTB. You can modify any Target Options window and it will be saved.

**Cleaning up your Project:** (you only need to do this once: this is not a critical step)

We modified the folder where the output and listings files are stored. This was in Steps 3 through 7 on the preceding page. If you did a Build before this was done, there will be files in your project root directory. Now we want them only in .\Flash.

1. Exit μVision. Otherwise, you can't delete files that it still has open.

2. Open Microsoft Explorer and navigate to:
   C:\MDK\Boards\Freescale\FRDM-KL25Z\BlinkyNEW\.

3. Delete all files and folders except these: (you can delete Flash – a Build will recreate it.)

4. You can also leave any backup or μVision files that identify your computer to retain your settings.

5. Restart μVision. Having all compilation files stored in the .\Flash folder makes it cleaner.

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com
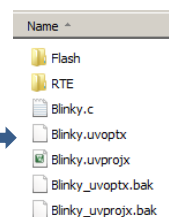
## 27) Creating your own RTX MDK 5 project from scratch:

The MDK Software Packs makes it easy to configure an RTX project.  There are two versions of RTX:  The first comes with MDK 4.7x and earlier.  The second comes with MDK 5.10 and later.  This second one is CMSIS-RTOS compliant.

Configuring RTX is easy in MDK 5.10 and later.  These steps use the same configuration as in the preceding Blinky example.

1. Using the same example from the preceding pages, Stop the program ⊗ and Exit Debug mode. 
2. Select CMSIS-DAP:  `CMSIS-DAP`
3. In Blinky.c, at the top,  add this line:  #include "cmsis_os.h"
4. Open the Manage Run-Time Environment window: 
5. Expand all the elements as shown here:
6. Select Keil RTX as shown and click OK.
7. Appropriate RTX files will be added to your project.  See the Project window.
8. Click on File/Save All or 
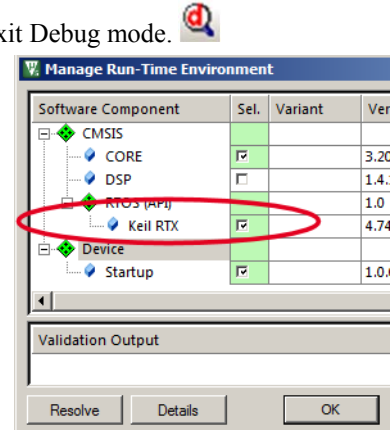
**Configure RTX:**
1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.

3. Select the Configuration Wizard tab:  Select Expand All.
4. The window is displayed here: 
5. Set Timer clock value: to  48000000 as shown:  (48 MHz)
6. Unselect User Timers.  Use defaults for the other settings.
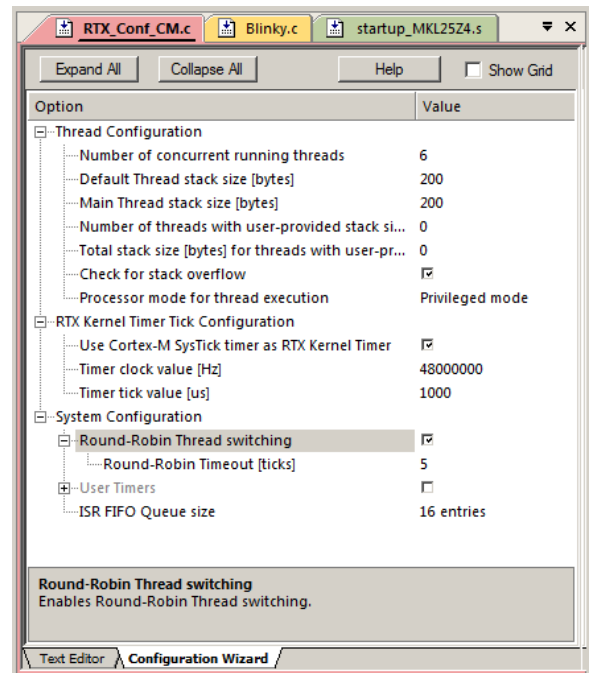
**Build and Run Your RTX Program:**

1. Build the files.  Program the Flash: .

2. Enter Debug mode:   Click on the RUN icon. 

3. Select Debug/OS Support/System and Thread Viewer.  The window below opens up.

4. You can see two threads: the main thread is the only one running.  As you add more threads to create a real RTX program, these will automatically be added to this window.

**What you have to do now:**
1. You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs.
2. See the DSP5 and RTX_Blinky examples to use as templates and hints.
3. If you copy Blinky.c from the RTX_Blinky project, it will blink the LEDs as it has the RTX code incorporated into it.

4. **Getting Started MDK 5:**  Obtain this useful book here: www.keil.com/mdk5/.  It has very useful information on implementing RTX.

This completes the exercise of creating your own RTX project from scratch.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com

## 28) Some Interesting Bits & Pieces:

**Processor Clock Speed:**

The clock speed is determined in the file system_MKL25Z4.c. This is where the processor PLL and other clock attributes are configured.

**SystemCoreClock:** system_MKL25Z4.c contains a global variable SystemCoreClock near line 96. You can view this in the Watch window to determine the processor clock frequency. Its value is determined from DEFAULT_SYSTEM_CLOCK which is defined in lines 78, 83 or 88. Which one is used depends on #define CLOCK_SETUP 0 near line 58. By setting CLOCK_SETUP to 0, 1 or 2 changes the clock frequency to 41.94, 48 or 100 MHz respectively. This is easily seen in lines 74 through 89. This is not actually measuring the CPU speed, rather just indicating which of the choices is selected.

**Display SystemCoreClock:**

1. With µVision in Debug mode and running, enter SystemCoreClock into Watch 1.

2. Right click on the Value field for SystemCoreClock and unselect Display Hexadecimal to get a base 10 value.

3. Note the clock is about 41.9 MHz. We thought it was 48 MHz !

4. In system_MKL25Z4.c, change the #define CLOCK_SETUP 0 to 1. This is near line 58.

5. Stop the program.  Enter Debug mode: 

6. Build the files.  Program the Flash: .

7. Enter Debug mode:  Click on the RUN icon. 
8. SystemCoreClock will now display 48 MHz.

| Watch 1 | | |
|---|---|---|
| Name | Value | Type |
| counter | 0x0000000B | unsigned int |
| SystemCoreClock | 41943040 | unsigned int |
| <Enter expression> | | |

Call Stack + Locals | Watch 1 | Memory 1

---

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit          www.keil.com
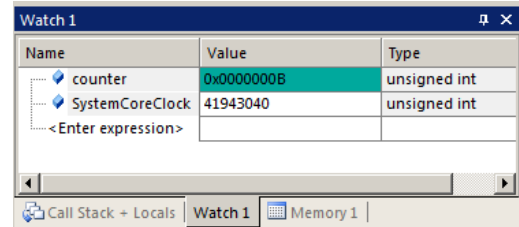
## 29) Kinetis KL25 Cortex-M0+ Trace Summary:

**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.

## 30) 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 features except MTB.  To use SWV, any Keil ULINK or Segger J-ink debug adapter is needed and to use ETM trace, a ULINK*pro* is needed.

- **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 µVision Cortex-M Target Driver Setup.

- The SWJ box must be selected.  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.  µVision 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.  µVision 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 output ETM frames and optionally SWV (rather than the 1 bit port SWO pin).  ULINK*pro* normally uses the Trace Port.

- **ETM:**  Embedded Trace Macrocell: Lists the executed instructions.  Cortex-M4 with a ULINK*pro* provide ETM.

- **MTB:**  Micro Trace Buffer.  A portion of the device internal RAM is used for an instruction trace buffer.  Only on Cortex-M0+ processors.  Kinetis Cortex-M4 processors provide ETM trace instead.

- **Hardware Breakpoints:**  The Kinetis Cortex-M0+ has 2 breakpoints. The Kinetis Cortex-M4 has 6.  These can be set/unset on-the-fly without stopping the processor.  They are no skid:  they do not execute the instruction they are set on when a match occurs.

- **WatchPoints:**  Both the Freescale Cortex-M0+ and Cortex-M4 have 2 Watchpoints.  These are conditional breakpoints.  They stop the program when a specified value is read and/or written to a specified address or variable.

## 31) Debug Adapter Summary for Keil MDK with µVision 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 file needs to be programmed into the K20 Flash.  This file is CMSIS-DAP.S19 and is provided in each project in its OpenSDA directory.  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 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 µVision, 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 µVision supports P&E OpenSDA.  OSJTAG drivers V1.06 or later need to be installed in µVision.  See www.keil.com/download/docs/408.asp for the drivers.  P&E was not tested for this document.  Use OpenSDA for this tutorial.

**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.

---

## 32)  Configure µVision for OpenSDA (CMSIS-DAP):

7.  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.

8.  Start µVision if not already running.  Select the project you want to use for your board.

9.  Select Target Options or ALT-F7 and select the Debug tab.  In the drop-down menu box select CMSIS-DAP Debugger as shown here:

9.  Select Settings and the next window below opens up.  This is the control panel for debug control.

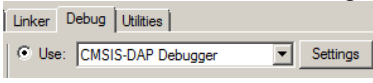10. In Port: select SW.  Select the SWJ box.  Note:  The KL25 has only SWD (SW).

11. 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.
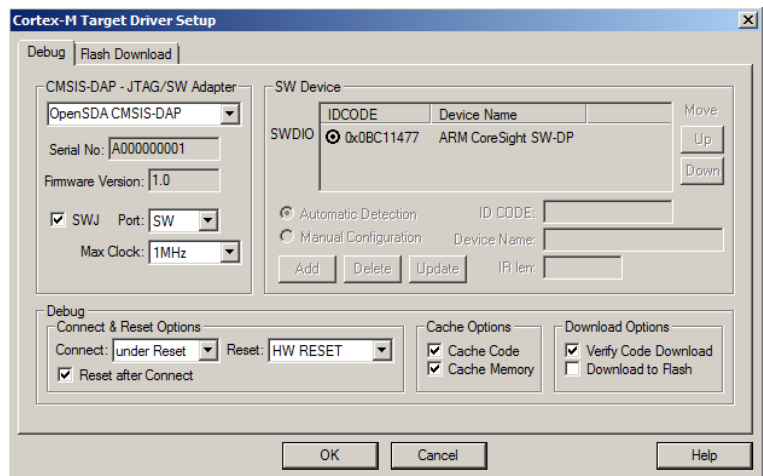
**TIP:**  To refresh this screen, change the option in Port: to JTAG and then back to SW or click OK once to leave this screen and then Settings: to re-enter it.

**Configure the Keil Flash Programmer:**

1.  Click on OK once and select the Utilities tab.

2.  Select these two selections as shown here:

3.  This will use the adapter selected in Step 9.

4.  Click Settings to confirm the programming algorithm.

5.  Confirm MKXX 48 MHz 128kB Prog Flash is selected as shown below:

6.  Click on OK once.

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

7.  Click on OK once more to return to the µVision main screen.
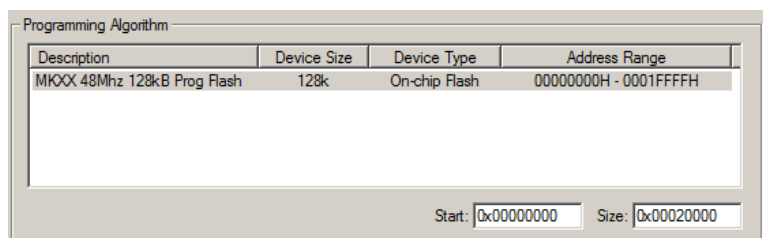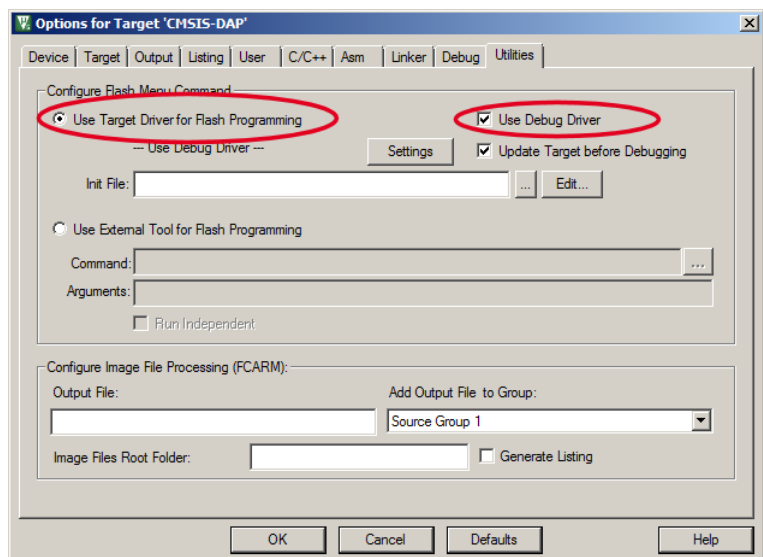
To use a ULINK2. ULINK-ME or ULINK*pro*, please see the next two pages.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit           www.keil.com

## 33) 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*pro*, 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 with the Kinetis Coretx-M0+ processors. Additional features are provided with the Kinetis Cortex-M4 processors. See www.keil.com/freescale for more information.

Assume a ULINK is connected to a powered up Kinetis target board as shown below. Add a USB cable to J7 to power the board. The next page has a close-up photo. µVision 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 to JTAG and then back to SW or click OK once to leave and then click on Settings again to re-enter.

**Configure the Keil Flash Programmer:**

5. Click on OK once and select the Utilities tab.

6. Select the two default settings as shown below:

7. Click Settings to confirm the programming algorithm.

8. The MKXX 48 MHz 128kB Prog Flash is the correct one to use with the Freedom board as shown:

9. Click on OK once.

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

10. Click on OK to return to the µVision main screen.

11. You have successfully connected to the KL25 target.



**Keil ULINK-ME**

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.
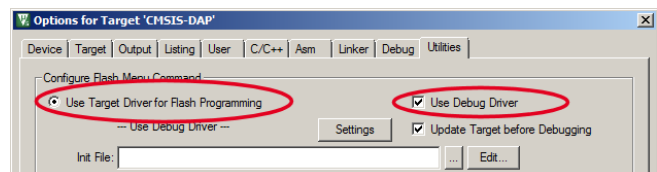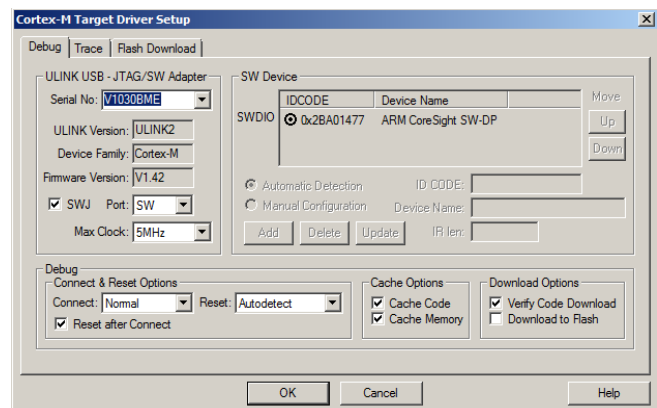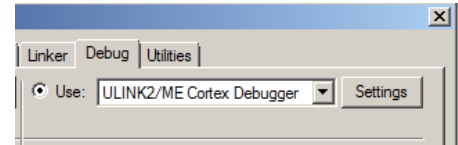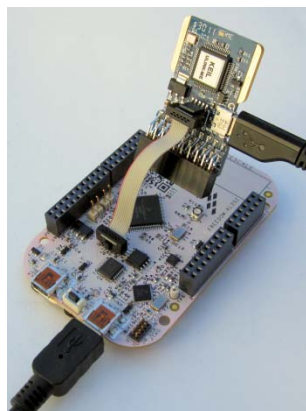


ULINK-ME is available only as part of a board kit from
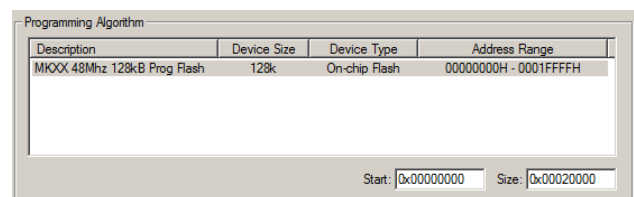


**TIP:** If you select ULINK or ULINK*pro*, and have the opposite ULINK actually connected to your PC; the error message will say "No ULINK device found". This message actually means that µVision found the wrong Keil adapter connected, and not that no ULINK was attached. Select the correct ULINK and try again.

Freescale Cortex-M0+ Lab with ARM® Keil™ MDK 5 toolkit                www.keil.com

## 34) Configuring a Keil ULINK*pro*:

This is configured the same way as a ULINK2 except for the selection entry in the Debug tab (shown below).

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 ULINK*pro*.

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 ULINK*pro* and the board.

6. Select the Utilities tab and select the ULINK*pro* and confirm 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 two items as shown here:

3. Click Settings to confirm the programming algorithm.

4. MKXX 48 MHz 128kB Prog Flash is the correct one to use with the Freedom board as shown here:
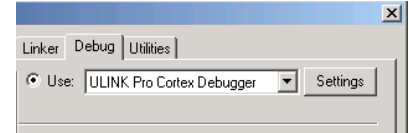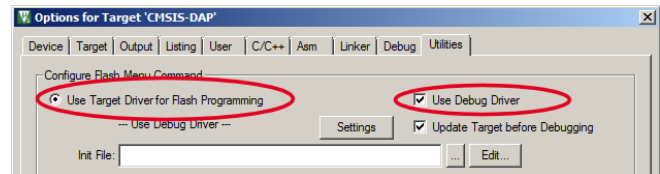
5. Click on OK twice.

**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 ULINK*pro*. If the RAM is erased the ULINK*pro* must be sent back to Keil for reprogramming. ULINK2 has no such battery.

**TIP:** If you select ULINK or ULINK*pro*, 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.
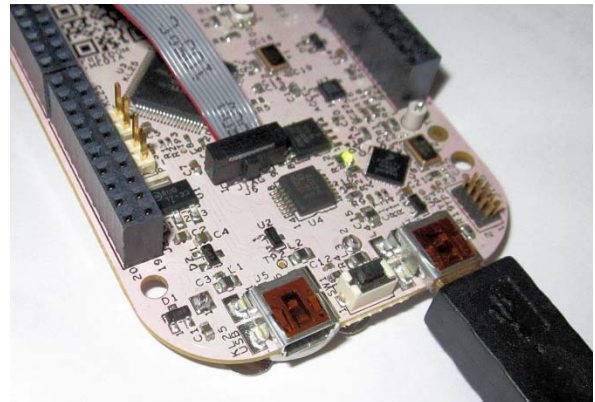
## Keil ULINK*pro* information:

ULINK*pro* is an ETM trace adapter that can be used as a ULINK2. ULINK*pro* 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.

ULINK*pro* features are best exploited with Kinetis Cortex-M4 processors. ULINK2 provides SWV but not ETM on Kinetis Cortex-M4 processors. See the labs on www.keil.com/freescale.

ULINK*pro* 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.
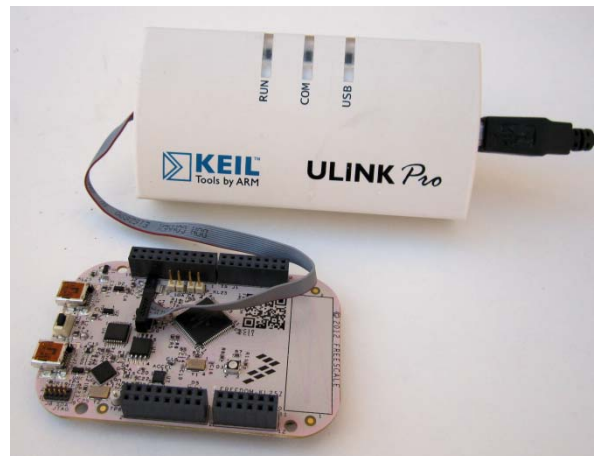
## 35) Document Resources:    See www.keil.com/freescale

## Books:

1. *NEW!* **Getting Started MDK 5:**    Obtain this free book here: www.keil.com/mdk5/.

2. There is a good selection of books available on ARM processors. A good list of books on ARM processors is found at www.arm.com/university by selecting "Teaching Resources".  You can also select ARM Related Books but make sure to also select the "Books suited for Academia" tab to see the full selection.

3. µVision contains a window titled Books.  Many documents including data sheets are located there.

4. **Keil manuals and documents:**    www.keil.com/arm/man/arm.htm    **Videos:** www.keil.com/videos

5. **A list of resources is located at:**    www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab.  Or search for "Cortex-M3" on www.arm.com and click on the Resources tab.

6. **The Definitive Guide to the ARM Cortex-M0/M0+** by Joseph Yiu.    Search the web.

7. **The Definitive Guide to the ARM Cortex-M3/M4** by Joseph Yiu.    Search the web.

8. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano

## Application Notes:

9. Using Cortex-M3 and Cortex-M4 Fault Exceptions    www.keil.com/appnotes/files/apnt209.pdf

10. Segger emWin GUIBuilder with µVision™    www.keil.com/appnotes/files/apnt_234.pdf

11. Porting mbed Project to Keil MDK™    www.keil.com/appnotes/docs/apnt_207.asp

12. MDK-ARM™ Compiler Optimizations    www.keil.com/appnotes/docs/apnt_202.asp

13. Using µVision with CodeSourcery GNU    www.keil.com/appnotes/docs/apnt_199.asp

14. RTX CMSIS-RTOS Download    www.keil.com/demo/eval/rtx.htm

15. Barrier Instructions    http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html

16. Lazy Stacking on the Cortex-M4:    www.arm.com and search for DAI0298A

17. Cortex Debug Connectors:    www.arm.com and search for cortex_debug_connectors.pdf

18. Sending ITM printf to external Windows applications:    http://www.keil.com/appnotes/docs/apnt_240.asp

19. FlexMemory configuration using MDK    www.keil.com/appnotes/files/apnt220.pdf

20. Export Processor Expert Projects to µVision™    www.keil.com/appnotes/docs/apnt_235.asp

21. Using MQX with Keil µVision Kernel Awareness    www.keil.com/freescale/mqx.asp

22. Keil Videos:    www.keil.com/videos

## Keil Tutorials for Freescale Boards:    www.keil.com/freescale

23. KL25Z Freedom (the full version of this document)    www.keil.com/appnotes/docs/apnt_232.asp

24. K20D50M Freedom Board    www.keil.com/appnotes/docs/apnt_243.asp

25. Kinetis K60N512 Tower    www.keil.com/appnotes/docs/apnt_239.asp

26. Kinetis K60D100M Tower    www.keil.com/appnotes/docs/apnt_249.asp

**Community Forums:**  www.keil.com/forum  and  http://community.arm.com/groups/tools/content

**ARM University program:**  www.arm.com/university.   Email: university@arm.com

**ARM Accredited Engineer Program:**  www.arm.com/aae

**mbed:**  http://mbed.org

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

For more information on the ARM CMSIS standard:  www.arm.com/cmsis,

## 36) Keil Products and Contact Information:  See www.keil.com/freescale

## 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
- *NEW !!*  MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit)  - $3,840
- MDK-Standard (unlimited compile and debug code and data size) **-** $4,895
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries)  $9, 500
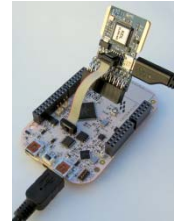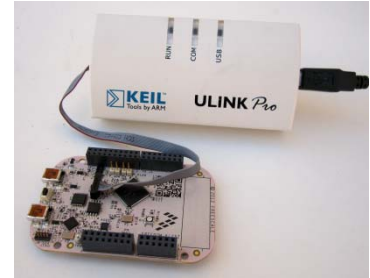
## USB-JTAG adapter  (for Flash programming too)

- ULINK2 - $395  *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINK*pro* - $1,250 – Cortex-M*x* SWV & ETM trace.

    *All ULINK products support MTB.*

- *For special promotional or quantity pricing and offers, please contact Keil Sales.*

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

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

Keil provides free DSP libraries for the Cortex-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 and search for university to view various programs and resources.

Keil supports many other Freescale processors including ARM9™ and Cortex-R series processors.  See the Keil Device Database® on 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 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.

---

## For more information:

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

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

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

For the latest version of this document, go to www.keil.com/freescale and for more Freescale specific information.

CMSIS Version 3:  www.onarm.com/downloads or www.arm.com/cmsis

---