NXP Kinetis: TWR-K64F120M Cortex™-M4 Lab

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

Fall 2016 Version 1.0 by Robert Boys, bob.boys@arm.com

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_288.asp

A similar lab for the FRDM-K64F: www.keil.com/appnotes/docs/apnt_287.asp

Introduction:

The purpose of this lab is to introduce you to the NXP Cortex™-M4 processor by using the ARM Keil MDK toolkit featuring µVision® IDE. We will use the Serial Wire Viewer (SWV) and ETM™ trace on the Kinetis processor. www.keil.com/NXP. Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. The addition of a license number will turn it into a full commercial version. Contact Keil sales for pricing, options and current special offers. Contact info is on last page.

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

Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is turn-key "out-of-the-box".
2. Keil Middleware is offered: www.keil.com/mdk5/middleware/
3. **Compiler Safety Certification Kit:** www.keil.com/safety/
4. **TÜV** certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
5. ARM Compiler 5 and Compiler 6 (LLVM) included. GCC can be used.
6. MISRA C/C++ support using PC-Lint. www.gimpel.com
7. Keil RTX is included. This full feature RTOS has a BSD license and sources are provided. This makes it free. www.keil.com/RTX.
8. **MQX:** An MQX port for MDK is available including Kernel Awareness windows. See www.keil.com/NXP.
9. CoreSight™ Serial Wire Viewer and ETM trace capability.
10. Choice of debug adapters: ULINK2™, ULINK-ME™, ULINK*pro*™, OpenSDA (P&E) and Segger J-Link.
11. **Kinetis Expert** compatible: A MDK 5 project is created that you can load directly into µVision. See kex.nxp.com/.
12. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
13. Affordable perpetual and term licensing. Contact Keil sales for pricing, options and current special offers.

ULINK*pro* connected to a TWR-K64

This document details these features *and more*:

1. Serial Wire Viewer (SWV) data trace including exceptions, display variables and ETM Instruction Trace.
2. Real-time Read and Write to memory locations for Watch, Memory, System Viewer and RTX Tasks windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is needed.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and one Watchpoint (also called Access Breaks).
4. A DSP example using ARM CMIS-DSP libraries which are included with MDK. The DSP source code is included.
5. Create a µVision project from scratch. Optionally add RTX. An example using Kinetis Expert is also provided.

Serial Wire Viewer (SWV): With a Keil ULINK2, ULINK*pro* or J-Link (Not with OpenSDA):

Serial Wire Viewer (SWV) displays Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters, PC Samples and a timestamp. RTX Event Viewer uses SWV. This information comes from the ARM CoreSight™ debug module integrated into the Cortex-M4. SWV is output on the 1 bit Serial Wire Output (SWO) or the 4 bit Trace Port.

SWV does not steal any CPU cycles and is completely non-intrusive (except for the ITM Debug printf Viewer).

Embedded Trace Macrocell™ (ETM): ETM (and ETB) is available only with a Keil ULINK*pro*.

ETM records all executed instructions in addition to the features provided by SWV. ETM provides advanced features including Program Flow instruction debugging in assembly and C, Code Coverage and Performance Analysis.

General Information and Introduction to Keil MDK: **Page**

1) General Information and introduction to MDK:

Keil Software: MDK 5 **This document used MDK 5.21 and Software Pack 1.4.0. You can use later versions.**

MDK 5 uses Software Packs to distribute processor specific software, examples, documentation and middleware. MDK 5 Core is first installed and you then download the Software Packs required for your processor(s) from the web. A Pack can also be imported manually. You no longer need to wait for the next version of MDK or install patches to get the latest files. Software Packs are an ARM CMSIS standard. See www.keil.com/cmsis and https://github.com/ARM-software/CMSIS_5

Kinetis Expert provides software projects in MDK 5 format consistent with Software Packs.

Summary of the Keil software installation: This is a three step process:

A. Download and install MDK Core. This is done in Step 2 below.

B. Download and install the appropriate Software Pack for the processor you are using. This is done on the next page.

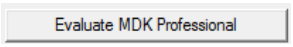
C. In addition, you need to download and install the examples used in this tutorial. See page 5.

Keil MDK Core Software Download and Installation:

1. Download MDK Core from the Keil website. www.keil.com/mdk5/install 

2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5
3. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples. The name Freescale is still used in MDK software but this will be soon changed to NXP.
4. If you install MDK or the examples into different folders, you will have to adjust for the folder location differences.

Licensing:

1. You can use the evaluation version (MDK-Lite) for this lab. No license is needed.
2. You can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible:
3. This gives you access to the Keil Middleware as well as unlimited code size compilation. Contact Keil Sales to extend this license for evaluation purposes. 

Debug Adapters:

1. You do not need any debug adapters: just the TWR board, a USB cable, MDK 5 and a Pack installed on your PC.
2. If you want to use Serial Wire Viewer (recommended) use any Keil ULINK or a Segger J-Link. You will also need a 10 to 20 pin CoreSight cable to connect to K64 JTAG EZPORT. This cable is normally provided with a ULINK.
3. A ULINK*pro* connects directly into the 20 pin K64 JTAG EZPORT connector. ETM trace is then provided.

Keil manufactures several adapters. These are listed below with a brief description. See page 10 for photos.

1. **ULINK2 and ULINK-ME:** ULINK2 is pictured on page 10. ULINK-ME is offered only as part of certain evaluation board packages. ULINK2 can be purchased separately. They are electrically the same and both support Serial Wire Viewer (SWV), run-time memory reads and writes for the Watch, Memory and SVD Peripheral windows, Watchpoints and hardware breakpoints can be set/unset on-the-fly while the program is running..
2. **ULINK*pro*:** This is pictured on page 1. ULINK*pro* supports all SWV features and adds ETM Trace support. ETM records all executed instructions. ETM provides complete instruction flow debugging, Code Coverage, Execution Profiling and Performance Analysis. ULINK*pro* also provides the fastest Flash programming times.

Keil supports more adapters:

1. **OpenSDAV2:** An extra processor on your board becomes a debug adapter compliant to CMSIS-DAP. The NXP Freedom boards incorporate CMSIS-DAP. µVision communicates via USB to the CMIS-DAP processor which controls the target processor. This is selected like any adapter in the Target Options menu under the Debug tab.
2. **P&E:** µVision running on your PC directly connects to the Kinetis board via a USB connection without any debugging hardware. See P&E for configuration details.
3. **Segger J-Link and J-Trace:** J-Link Version 6 (black case) or later supports Serial Wire Viewer. J-Trace provides ETM but has not been tested in this document. Data reads and writes are not currently supported with a J-Link.

Getting Started MDK 5: Obtain this free, useful book here: www.keil.com/gsg/.

2) Software Pack Install Process:

1) Start µVision and open Pack Installer:

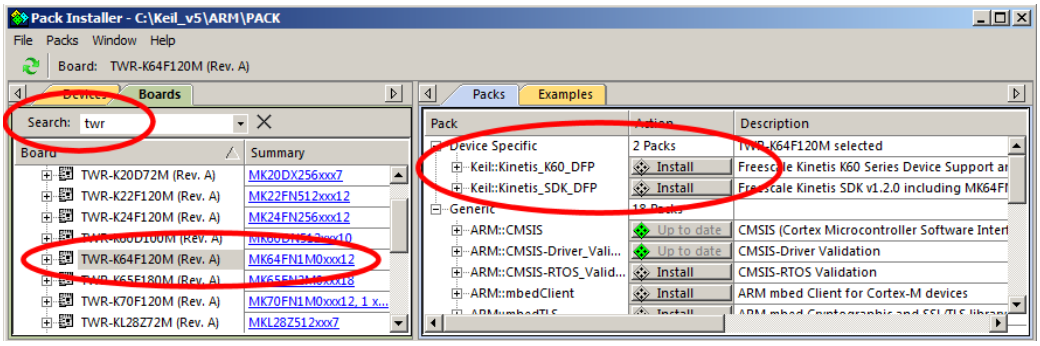
When the first MDK install is complete and if you are connected to the Internet, µVision and Software Packs will automatically start. Otherwise, follow Steps 1 and 2 below. Initially, the Pack master list must be downloaded from the web.

1. Connect your computer to the Internet. This is normally needed to download the Software Packs.
2. Start µVision by clicking on its desktop icon. The Pack descriptions will download on the initial µVision run.
3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
4. If there are any Updates available, you can download them now if they are applicable. Update
5. The window below opens up: Select the Boards tab. Type **twr** in the Search box as shown to filter the Packs and Example tabs on the right side of the Pack Installer window as shown below:

TIP: The Devices and Boards tabs are used to filter the items displayed on the right side in the Packs tabs.

6. Select TWR-K64F120M as shown below left: You can also select individual processors under the Devices tab.
7. Note: "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet.

TIP: 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. This is not done automatically.



2) Install the Kinetis K60 Device Family Pack (K60_DFP):

1. Click on the Packs tab. Initially, the Software Pack ARM::CMSIS is installed by default.
2. Click on Install beside Keil::Kinetis_K60_DFP as shown above and. The latest Pack will download and install to C:\Keil_v5\ARM\Pack\Keil\Kinetis_K60_DFP\ by default. This download can take two to four minutes.
3. Its status will then be indicated by the "Up to date" icon: Up to date

3) Install the Kinetis K60 Software Development Kit (SDK_DFP):

1. Click on Install beside Keil::Kinetis_SDK_DFP to install the Kinetis SDK Pack.
2. Its status will then be indicated by the "Up to date" icon: Up to date
3. Do not close Pack Installer yet as it is needed to download the Blinky and RTX_Blinky examples on the next page.

TIP: You can also install a Pack manually. A Pack has a file extension .pack. It is an ordinary zip file with the extension changed so it is recognized by µVision. You can download the Pack from the web or transfer the file in any other way. Double click on this file and it will automatically be recognized (.pack) and installed by Pack Installer.

TIP: You can create your own Pack to distribute a DFP, BSP, SDK. This is a good way to distribute confidential material.

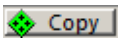
For CMSIS-Pack documentation: www.keil.com/pack/doc/CMSIS/Pack/html/

For complete CMSIS documentation: www.keil.com/CMSIS/

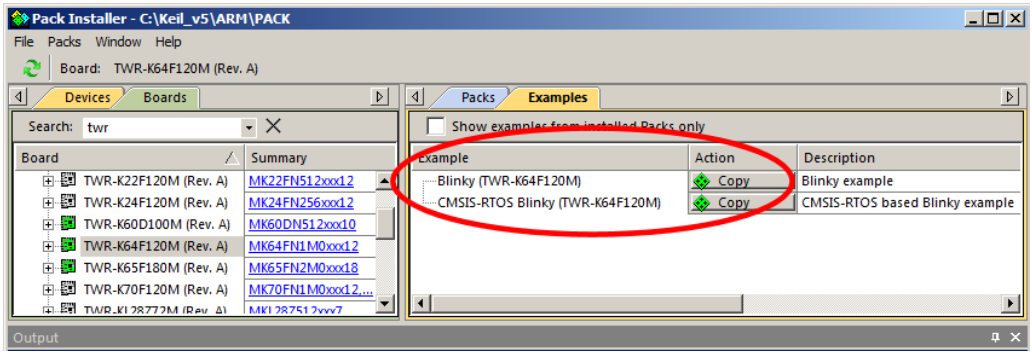
For CMSIS 5 on GitHub: https://github.com/ARM-software/CMSIS_5

3) Copy the Blinky, RTX_Blinky and DSP Examples:

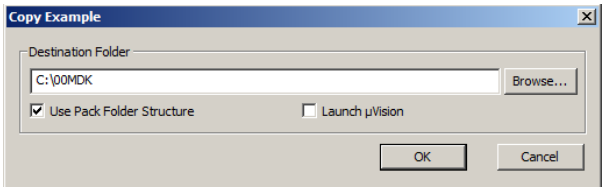
1) Copy Blinky and RTX_Blinky using Pack Installer Utility:

1. Select the Examples tab:
2. Beside Blinky as shown below: Select Copy : 

3. The Copy Example window below opens up: Select Use Pack Folder Structure. Unselect Launch µVision:



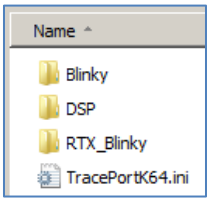
4. Type in **C:\00MDK** as shown below right:
5. Click OK to copy the example file into C:\00MDK\Boards\Freescale\TWR-K64F120M\.
6. Pack Installer creates the appropriate subfolders in C:\00MDK\.
7. Repeat for CMSIS-RTOS Blinky. Select the second one as it is the latest version (2.20 in this case)
8. Close Pack Installer. Open it any time by clicking: 
9. If a window opens stating the Software Packs folder has been modified, select Yes to "Reload Packs ?"



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 folder of C:\00MDK\ in this tutorial. You can use any folder of your choosing.

2) Copy the DSP Example from www.keil.com:

1. Obtain the example software zip file from www.keil.com/appnotes/docs/apnt_287.asp.
2. This folder will have been created: C:\00MDK\Boards\Freescale\TWR-K64F120M\
3. Extract the example zip into this folder. These folders have been created in \TWR-K64F120M\



TIP: The file TracePortK64.ini will be used in the ETM exercises on page 26.

Super TIP: µVision icon meanings are found here: **www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm**

What you have accomplished so far:

1. Keil MDK Core is installed on your computer.
2. The K60 Device Family Pack (DFP) K60 is installed to C:\Keil_v5\ARM\Pack\Keil\Kinetis_K60_DFP.
3. The Kinetis SDK Device Family Pack (DFP) is installed to C:\Keil_v5\ARM\Pack\Keil\Kinetis_SDK_DFP.
4. The example RTX_Blinky is copied to C:\00MDK\Boards\Freescale\TWR-K64F120M\.
5. The DSP example is copied from www.keil.com to C:\00MDK\Boards\Freescale\TWR-K64F120M\.

Next ? We will examine two µVision utilities for managing Software Packs and Pack Version selection.

4) Software Pack Version Selection and Manage Run-Time Environment:

This section contains three parts on this and the next page:

A) Select Software Pack Version:

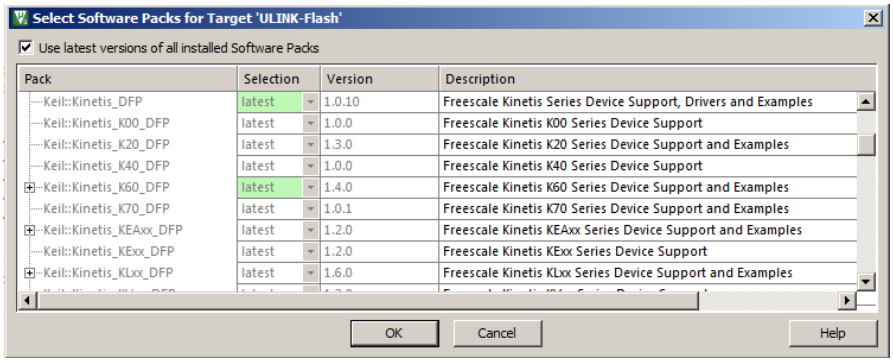
B) Manage Run-Time Environment:

C) Updating Source Files:

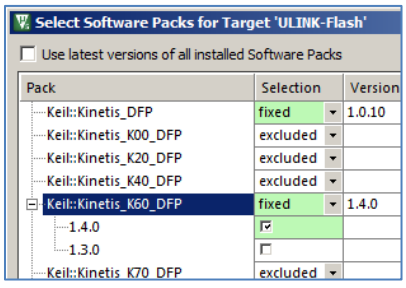
A) Select Software Pack Version: *this section is provided for reference:*

This µVision utility provides the ability to choose various Software Pack versions installed in your computer. You can select the versions you want to use. You must have µVision running and any project open for the following exercises:

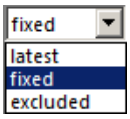
1. Select Project/Open Project and navigate to C:\00MDK\Boards\Freescale\TWR-K64F120M\Blinky.
2. Open Blinky.uvprojx. Blinky will load into µVision.
3. Open Select Software Packs by clicking on its icon: 
4. This window opens up. Note **Use latest versions …** is selected. The latest version of the Pack will be used.
5. Unselect this setting and the window changes as shown similar to the one below right:



6. Expand the header Keil::Kinetis_K60_DFP.
7. You will see only one version – the one you installed. If you had installed others, you would see them listed like this:



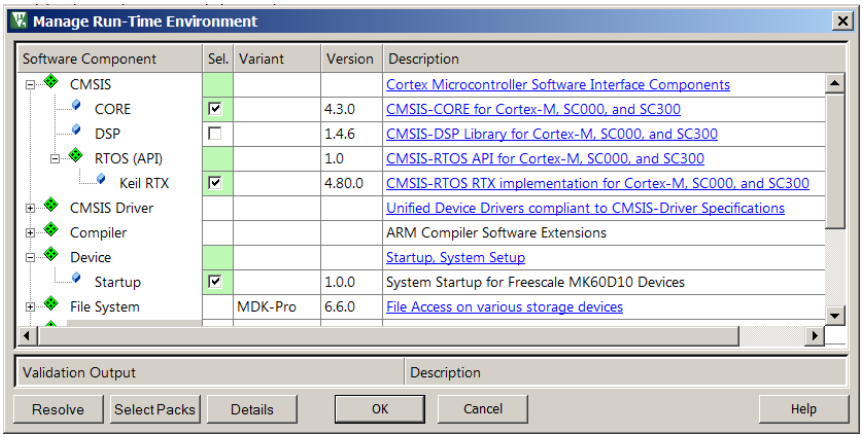
8. Select the fixed pull-down menu and see the three options as shown below:
9. If you wanted to use V 1.3.0, you would select fixed and then select the check box opposite 1.3.0.



10. Re-select Use latest versions… **Do not make any changes at this time.**
11. Click OK or Cancel to close this window.

B) Manage Run-Time Environment: *this section is provided for reference:*

1. Click on the Manage RTE icon:  The window below opens: This includes Keil Middleware, selected open source software, RTX and CMSIS drivers for various peripherals. Not all Packs offer all options but more are being added.
2. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project.
3. Note CMSIS/Core (system.c), Keil RTX and Device/Startup (startup.s) files are selected. You can see these files in the µVision Project window.
4. **Do not make any changes.**
   Click Cancel to close this window.



TIP: Different colors represent messages:

 Green: all required files are located.

 Yellow: some files need to be added. Click the Resolve button to add them automatically or add them manually.

 Red: some required files could not be found. Obtain these files or contact Keil technical support for assistance.

The Validation Output area at the bottom of this window gives some information about needed files and current status.

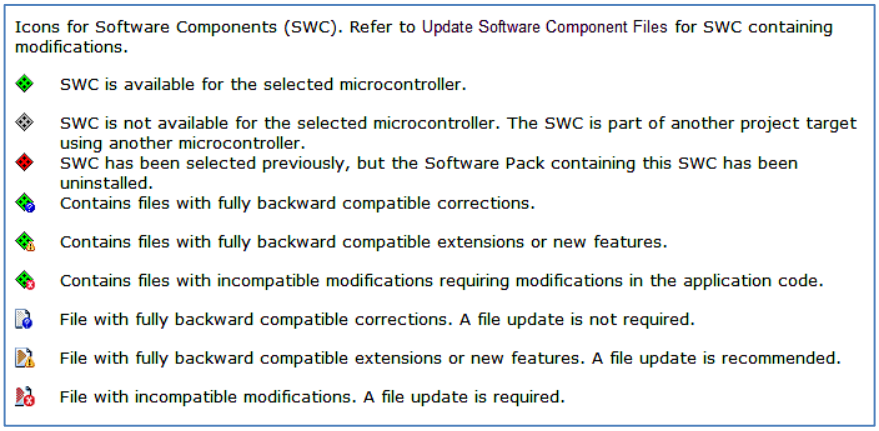
C) Updating Source Files: *this section is provided for reference:*

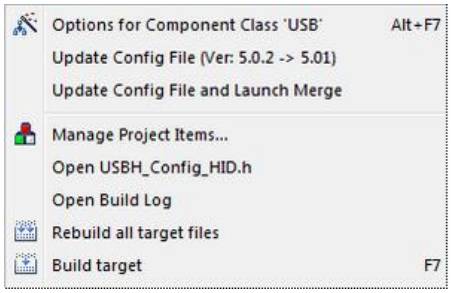
Some of the files provided by a Software Pack are stored in your project in ..\RTE

If you update a Software Pack, some of these files might need to be updated from the new Pack. These files will show new icons as shown below and described here: www.keil.com/support/man/docs/uv4/uv4_ca_updswcmpfiles.htm

Updating Source Files:

1. Right click on a file you want to update. A window similar to the one below right opens:
2. Select the appropriate Update selection. Any changes you made to the original file are lost.
3. This procedure is described here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm





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

Cortex-M0 and Cortex-M0+ may have only features 2) and 4) plus 11), 12) and 13) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 can have all features listed implemented. MTB is normally found on Cortex-M0+. It is possible some processors have all features except ETM Instruction trace and the trace port. Consult your specific datasheet.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except Boundary Scan is not possible. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup.
   The SWJ box must be selected in ULINK2/ME or ULINK*pro*. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDO shares the same pin as SWO. The SWV data normally comes out the SWO pin or Trace Port.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **DAP:** Debug Access Port. This is 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, Peripheral and RTOS kernel awareness windows 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. You do not need to configure or activate DAP. µVision configures DAP when you select a function that uses it. Do not confuse this with CMSIS_DAP which is an ARM on-board debug adapter standard.
5. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
6. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
7. **Trace Port:** A 4 bit port that ULINK*pro* uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by µVision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK*pro* provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis. ETM is output on the Trace Port or accessible in the ETB (ETB has no Code Coverage or Performance Analysis).
10. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK*pro*. ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. Cortex-M3/M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 usually have 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. The CPU is halted before the instruction is executed.
13. **WatchPoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can 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. There also referred to as Access Breaks in Keil documentation.

Read-Only Source Files:

Some source files in the Project window will have a yellow key on them:  This means they are read-only. This is to help unintentional changes to these files. This can cause difficult to solve problems. These files normally need no modification.

If you need to modify one, you can use Windows Explorer to modify its permission.

1. In the Projects window, double click on the file to open it in the Sources window.
2. Right click on its source tab and select Open Containing folder.
3. Explorer will open with the file selected.
4. Right click on the file and select Properties.
5. Unselect Read-only and click OK. You are now able to change the file in the µVision editor.
6. It is a good idea to make the file read-only when you are finished modifications.

Super TIP: µVision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

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

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

If you are using any Keil ULINK or J-Link as your debug adapter: you can skip this page:
µVision supports P&E OpenSDA. This allows debugging the Kinetis Tower with a USB cable. No external adapter is required. MDK contains the P&E drivers located in C:\Keil_v5\ARM\PEMicro. You usually need to install them initially.

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

Install the P&E drivers:

1. Plug a USB cable from your PC to the TWR-K64 board to J2, the K20 Debug connector.
2. Using Microsoft Explorer, navigate to C:\Keil_v5\ARM\PEMicro\Drivers and execute PEDrivers_install.exe.
3. The P&E drivers will install to C:\PEMicro\.
4. The red power LED and green OpenSDA K20 LED will illuminate.

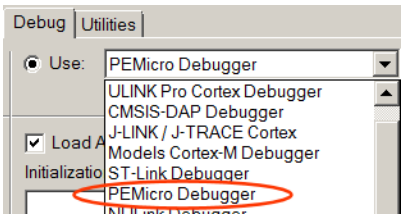
Start µVision and select the Blinky Project:

1. Start µVision by clicking on its desktop icon. 
2. Select Project/Open Project.
3. Select the Blinky.uvprojx project in C:\00MDK\Boards\Freescale\TWR-K64F120M\Blinky\.

Create a new Target Selection for P&E OpenSDA:

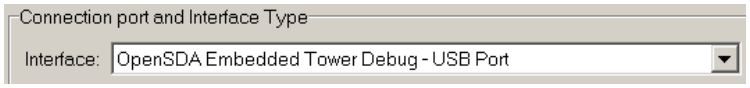
1. Select Project/Manage/Project Items… or select: 
2. In the Project Targets area, select NEW  or press your keyboard INSERT key.
3. Enter **PE Flash** and press Enter. Click OK to close this window.
4. In the Target Selector menu, select the PE Flash selection you just made: 

5. Select Options for Target  or ALT-F7. Click on the Debug tab to select a debug adapter.
6. Select PEMicro Debugger… as shown here: <an important step> 

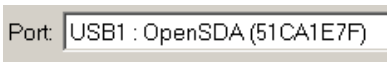


Configure the P&E Connection Manager: (the K64 board must be connected)

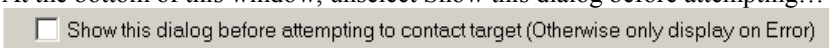
1. Click on Settings: and the P&E Connection Manager window opens.
2. In the Interface box, select OpenSDA Embedded Tower Debug: USB Port: as shown here:



3. Click the Select New Device box, and select your exact processor. In this case it is K64FN1M0M12 as shown here: This step is very important.
4. Click on the Refresh List and you will get a valid Port: box:



5. This means µVision is connected to the target K64 using P&E OSJTAG.
6. At the bottom of this window, unselect Show this dialog before attempting… as shown below:



7. Click on OK to close this window.
8. If you see **Undetected** in Port:, this means µVision is not connected to the target. Problems can be the K64 board is not connected to USB, or the wrong device is selected. Fix the problem and click Refresh List to try again.
9. Click the Utilities tab. The next step prevents the Flash being programmed twice when entering Debug mode.
10. Unselect Update target before Debugging in the Utility tab. Click OK to return to the main µVision menu.
11. Select File/Save All or click  . P&E OpenSDA is now completely configured including Flash programming.
12. You can go to page 12 to compile and run Blinky !

2) Connecting a ULINK2/ME, ULINK*pro* or J-Link to the NXP TWR board:

NXP provides the ARM standard 20 pin CoreSight Debug connector for JTAG/SWD on the TWR board as shown below:

This is J5 and is labelled K64 JTAG EZPORT on the board.

Pins 1 through 10 provide JTAG, SWD and SWO signals. Pins 11 through 20 add 4 bit ETM trace plus a clock.

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

20 Pin Cortex Debug Connector

Connecting ULINK2: See page 1 for a photo.

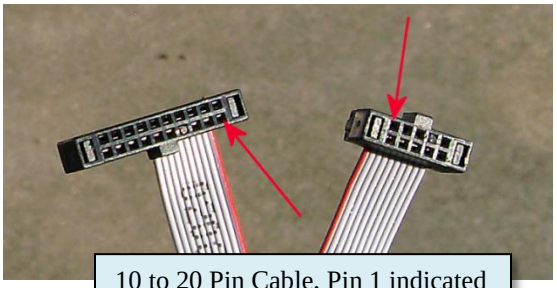
A ULINK2 and ULINK-ME come with a 10 to 20 pin cable. You will need to take the case off the ULINK2 and install this cable.

Connecting a ULINK*pro*: See page 1 for a photo:

The ULINK*pro* connects directly to the Kinetis TWR board with a supplied 20 pin cable.

You might need to remove a key pin to connect to the Kinetis target as described above.

Power: Power the board with a USB cable to the K64_USB connector as shown below. Using the K20 Debug connector might create a onflict as the K20 is the OpenSD debug adapter. The red LED D8 POWER will illuminate.


10 to 20 Pin Cable, Pin 1 indicated

TIP: You can use the TWR-K64 board without the elevator boards.

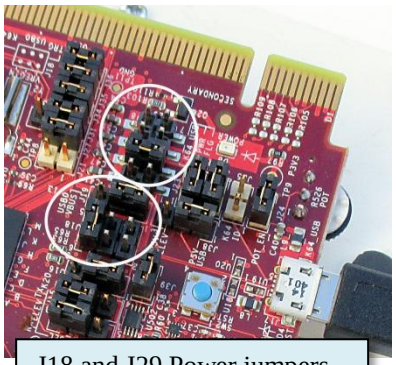
20 to 10 Pin Cable connected to J9

TWR-K64 Power Jumpers:

To obtain board power from the K64_USB connector J18 and J29 must be changed.

By default, power is obtained from the K20 Debug USB connector.

J18 and J29 are shown on the photo to the right:


J18 and J29 Power jumpers

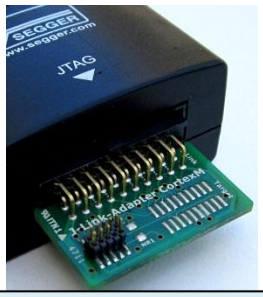
J18: change from 1-2 (for P&E OpenSDA) to 5-6 (for ULINK2, ULINK*pro*).

J29: change from 1-2 (for P&E OpenSDA) to 3-4 (for ULINK2, ULINK*pro*)..

You can also set these jumpers to obtain power from the elevator boards if used.

See the TWR-K64F120M Tower Module User's Guide for complete jumper information.

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



Segger CoreSight Adapter

Samtec Connector Part Numbers: The 20 pin male connector as shown on the TWR K64 is Samtec part number: FTSH-110-01.
You may want to add appropriate suffixes for suitable guide options.

The 10 pin Samtec cable is FTSH-105-01. This is pictured on this J-Link adapter.

3) Configuring Debug Adapters in µVision: This section is for reference:

It is easy to configure for a variety of Debug Adapters. Blinky and RTX_Blinky examples are preconfigured for a Keil ULINK2. You can add a configuration for a ULINK*pro*, a J-Link or P&E OpenSDA easily.

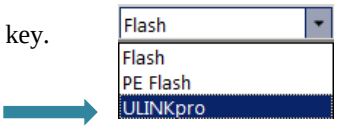
This will explain how to add a ULINK*pro* to the Blinky example. You can add a J-Link in a similar fashion.

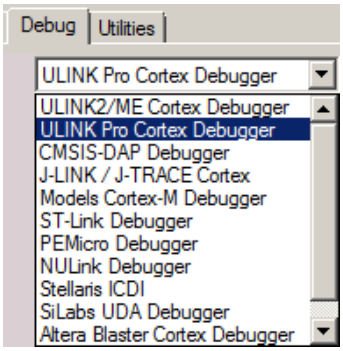
Prerequisites:

µVision must be running and in Edit mode (not Debug mode). Your project must be loaded. We will use Blinky.

Create a new Target Selection:

1. Select ULINK2 to use as the template adapter: 

2. Select Project/Manage/Project Items… or select: 
3. In the Project Targets area, select NEW  or press your keyboard INSERT key.
4. Enter **ULINKpro** and press Enter. Click OK to close this window.
5. In the Target Selector menu, select the ULINKpro selection you just made: 

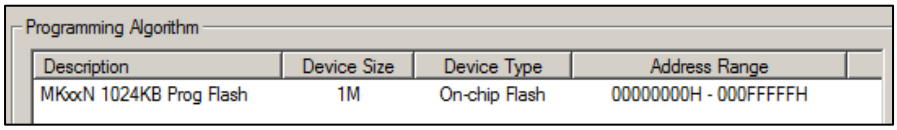
6. Select Options for Target  or ALT-F7. Click on the Debug tab to select a debug adapter.
7. Select ULINK Pro Cortex Debugger… as shown here: 

8. If you select Settings:, you can test the connection to the target board if you have the appropriate Debug Adapter connected.
9. If you are using a J-:Link, you can select it instead.

Verify the Flash Program Algorithm:

1. Select the Utilities tab.
2. Select Settings:
3. The window that opens will display the correct algorithm. This is selected automatically according to the processor selected in the Device tab.
4. Below is the correct algorithm for the K64:
5. Click OK to return to the main µVision window.
6. Select File/Save All or click  .

The new Debug Adapter is now ready to use.



TIP: You have now created a new target Option called ULINKpro. You can make any changes in the Options Target windows and save it. These are easily recalled by selecting the appropriate Options Target in the drop down menu as described above.

Part B: Keil Example Projects

1) *Blinky* example program:

We will run the example program Blinky on the TWR-K64F board using the P&E OpenSDA.

If you want to use a Keil ULINK2, it is preconfigured in the Target Selector. If you are using a ULINK*pro*, you must select it in the Target Options window under the Debug tab. See the previous page for instructions.

If using P&E OpenSDA (no external debug adapter):

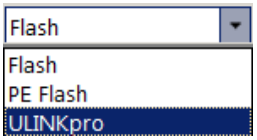
1. Power your TWR board with a USB cable connected to J2 K20 DEBUG USB connector and to your PC.

If using a Keil ULINK or a J-Link:

1. Connect your debug adapter as described on the page 10 to J5 the K64 JTAG EZPORT connector.
2. Change jumpers J18 and J29 as described on page 10 to power the board from J5.
3. Power your TWR board with a USB cable connected to J2 K20 DEBUG USB connector and to your PC.

Open and compile the Blinky Program:

1. Start µVision by clicking on its desktop icon.
2. Select Project/Open Project.
3. Open the file C:\00MDK\Boards\Freescale\TWR-K64F120M\Blinky\Blinky.uvprojx.
4. Select your debug adapter in the Target Options pull down menu box as shown here:



5. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.

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

6. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears. The Kinetis FLASH memory will be programmed. Progress will be indicated in the Output Window.
7. Click on the RUN icon. Note: you stop the program with the STOP icon.

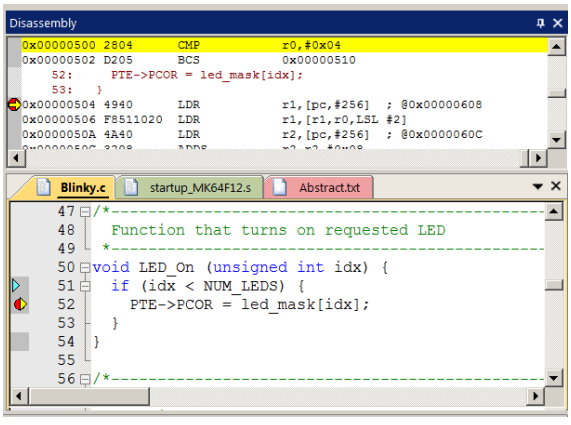
The four LEDs D5, D6, D7 and D9 will now blink in sequence.

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

TIP: The board will run Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

2) Hardware Breakpoints: (with OpenSDA, the program must be stopped to set/unset breakpoints)

1. With Blinky running, click once in the margin in the source file Blinky.c on a darker gray block in the LED_On function near line 52 as shown here:
2. A red circle is created and soon the program will stop here:



TIP: You can set/unset breakpoints on-the-fly with any ULINK or a J-Link. The program must be stopped with P&E OpenSDA.

3. The yellow arrow is where the program counter is pointing to in both the Disassembly and appropriate source window.
4. Click RUN and the program will run and then break again.
5. The Kinetis K64 has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set on. This is a very important feature.
6. If you set too may breakpoints, µVision will warn you.
7. Remove any breakpoints by clicking again on the red circle.

TIP: You can view the Breakpoint list by selecting Debug/Breakpoints or Ctrl-B. Watchpoints are also displayed here.

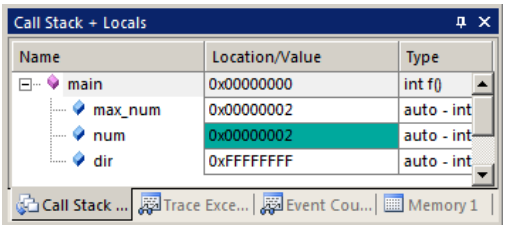
3) Call Stack + Locals Window: (This page works with all debug adapters)

Local Variables:

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

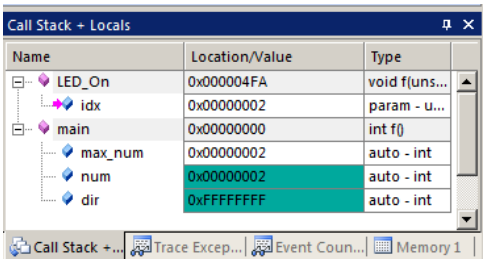
If possible, the values of the local variables will be displayed and if not the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack window.

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

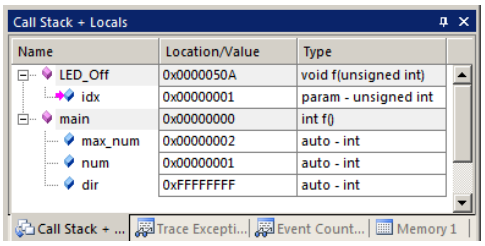


- The contents of local variables are displayed as well as names of active functions. Each function name will be displayed as it is called by another function or interrupt. We see here in main since the program spends most of its time here in this simple program.
- When a function exits, it is removed from the list. When using RTX, all threads are always displayed.
- The first called functions are at the bottom of this table.
- This table is active only when the program is stopped.

3. In Blinky.c, there is the function LED_On near line 48.
4. Set a breakpoint in this function.
5. Click on RUN The program will stop at this breakpoint.
6. The Call Stack window will now display the function LED_On as called by main(). This is shown below:



7. Set another breakpoint in Led_Off near line 59.
8. Click on RUN again. The program will stop here and this is indicated in the Call Stack and Locals window.
9. As you click RUN, the functions will alternate when the program is halted.
10. Click on Step Out or F11 and the program will return to main().

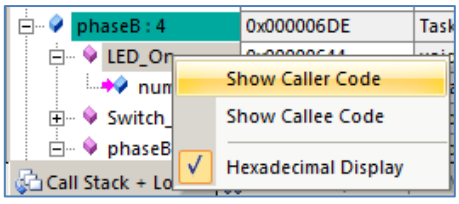


This is a very simple program to illustrate this feature. More complicated programs including using an RTOS such as RTX will display all functions. See the program example program RTX_Blinky for more complicated example with threads and functions.

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

Callee and Caller Code:

1. Right click on a function (I used LED_On) and select either Show Caller Code or Caller code as shown here:



2. The source window and dissambly windows will display the proper code.
3. **Remove all breakpoints before continuing.** Click on them or enter Ctrl_B in your keyboard and select Kill All in the Breakpoints window that opens. You can also temporarily deselect them.

4) Watch and Memory Windows: (The program must be stopped to updates with OpenSDA)

The Watch and Memory windows will display updated variables (not local or automatic variables) in real-time. It does this using ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to "put" or insert values into memory locations using the Memory and Peripheral windows in real-time or Watch if the data changes slowly enough..

a) Create a Global Variable counter and increment it:

1. Stop the program and exit Debug mode.
2. In Blinky.c near line 18, declare this global variable: **unsigned int counter = 0;**
3. In the function LED_On which is located near line 49, add this code at the end of this function near line 53:

   counter++;

   if (counter > 0x0F) counter = 0;

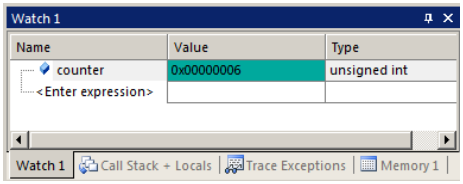
4. Compile the source files by clicking on the Rebuild icon. Enter Debug mode Click RUN

b) Enter counter in a Watch window:

1. In Blinky.c, right click on any instance of **counter** and select Add counter to … and select Watch 1.

TIP: You can add a variable to a Watch or Memory window while the program runs only with any Keil ULINK or a J-Link.

2. This action will open the Watch window if it is not already open. **counter** is displayed incrementing as shown here:

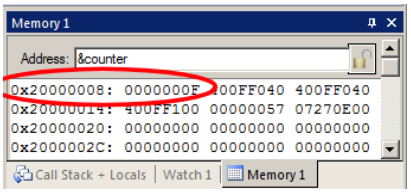


These three methods of entering variables also work with the Memory and Logic Analyzer windows. Variables can also be selected from the Symbol table. This also has the benefit of fully qualifying them (specifying where they are located).

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

c) Enter counter in a Memory window:

1. In Blinky.c, right click on **counter** and this time select Add counter to … and select Memory 1.
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 is not what we want to see at this time.
3. Add an ampersand "&" in front of the variable name. Now the physical address is shown (0x2000 0008). The address where **counter** is located might be slightly different in your program but it will be displayed the same way.
4. Right click in the Memory window and select Unsigned/Int.
5. The data contents of **counter** is displayed as shown here:
6. Both the Watch and Memory windows are updated in real-time.
7. Right-click on a memory location and select Modify Memory. You can change the value of this location while the program is running. The Watch window can be modified when stopped or data not changing fast.



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

TIP: You are not able to view local variables while the program is running. Convert them to static or global variables.

How It Works:

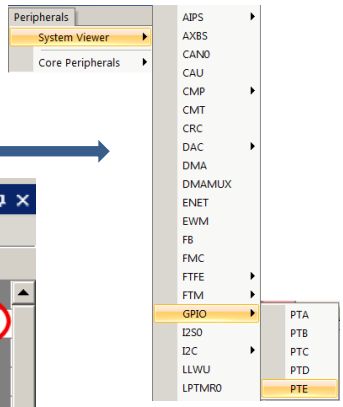
µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. Remember the Cortex-M4 is a Harvard architecture. This means it has separate instruction and data buses. While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

This can be slightly intrusive in the unlikely event the CPU and µVision reads or writes to the same memory location at exactly the same time. Then the CPU will be stalled for one clock cycle. In practice, this cycle stealing rarely happens.

5) System Viewer (SV): (Updates while program runs only with ULINK2, ULINK*pro* and J-Link.)

System Viewer provides the ability to view certain registers in the CPU core and in peripherals. In many 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:

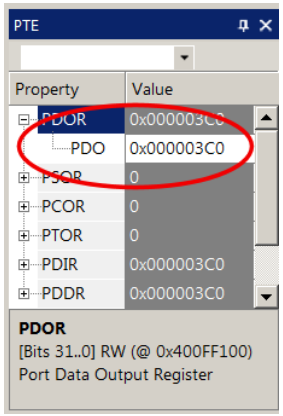
In our Blinky example, the four LEDs blinking are connected to GPIO Port E PTE6 through 9.



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

GPIO Port A:

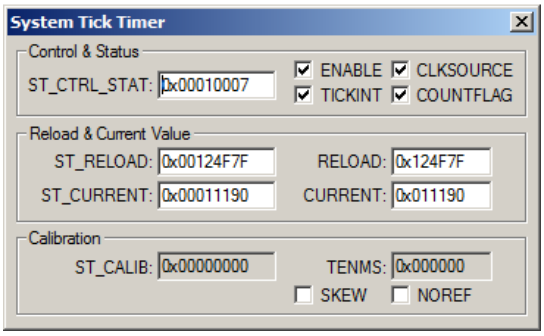
2. Select Peripherals/System Viewer, GPIO and then PTE as shown here:
3. The PTE window opens up:
4. You can now see PDOR update as the red and blue LEDs blink in succession:
5. You can change the values in the System Viewer while the program is running or stopped. It will be difficult to see this as these values are updated so often that your changes will be overwritten.
6. Stop the program and enter 0x0. All the LEDS will come on.
7. Click RUN to continue.
8. This window is updated using the same CoreSight DAP technology as the Watch and Memory windows.
9. Look at other Peripherals contained in other System Viewer windows to see what else is available.



TIP: If you click on a register in the properties column, a description about this register will appear at the bottom of the window as shown above for register PDOR.

SysTick Timer: This program uses the Cortex SysTick timer as the tick timer for RTX RTOS.

1. Select Peripherals/Core Peripherals and then select SysTick Timer. Run the program.
2. The SysTick window shown below opens:
3. Note it also updates in real-time while your program runs using CoreSight DAP technology.
4. Note the ST_RELOAD and RELOAD register contents. This is the reload register value. This is set in the call to SysTick_Config function in core_cm4.h by SysTick_Config(SystemCoreClock/100); near line 95 in Blinky.c.
5. Note that it is set to 0x0012 4F7F = 1,199,999. This is created by 120 MHz/100-1 = 41,939. 100 is specified as the timer tick value. Changing the reload value changes how often the SysTick interrupt 15 occurs.
6. In the RELOAD register in the SysTick window, *while the program is running*, type in 0xF000 and click inside ST_RELOAD ! (or the other way around)
7. The blinking LEDs will speed up. This will convince you of the power of ARM CoreSight debugging.
8. Replace RELOAD with 0x0012 4F7F. A CPU RESET  will also accomplish this.
9. When you are done, stop the program  and close all the System Viewer windows that are open.



TIP: You can also do this exercise with the SysTick window in the System Viewer.

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 for and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.

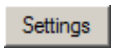
6a) Configuring Serial Wire Viewer (SWV) only with ULINK2, ULINK-ME or J-Link:

These instructions are not for ULINK*pro*: they are on the next page.

Serial Wire Viewer provides data trace information including interrupts in real-time without any code stubs in your sources. These instructions are for a ULINK2, ULINK-ME or a J-Link. **They are not for ULINK*pro*: those are on the next page.**

Configure SWV:

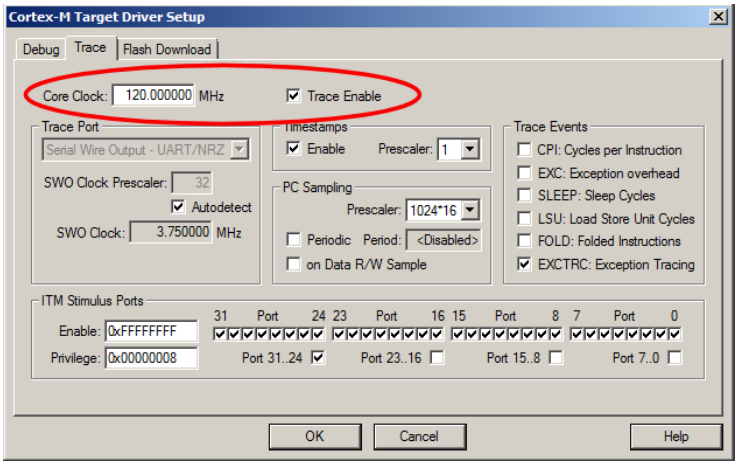
1. Connect a ULINK2 or J-Link and change the power jumpers as described on page 10.
2. µVision must be stopped and in Edit mode (not Debug mode). Blinky.uvprojx must be loaded.
3. For ULINK2/ME: select Flash: 

ULINK2 is preconfigured in this project.
4. Select Options for Target  or ALT-F7 and select the Debug tab. Your debugger must be displayed beside Use:.
5. Select Settings: 

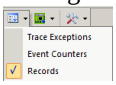
on the right side of this window.
6. Confirm Port: is set to SW and SWJ box is enabled for SWD operation. SWV will not work with JTAG.
7. Click on the Trace tab. The window below is displayed.
8. In Core Clock: enter 120 MHz. Select Trace Enable. This value ***must*** be set correctly to your CPU speed.

TIP: To find Core Clock frequency: Enter the global variable SystemCoreClock in a Watch window and run the program.

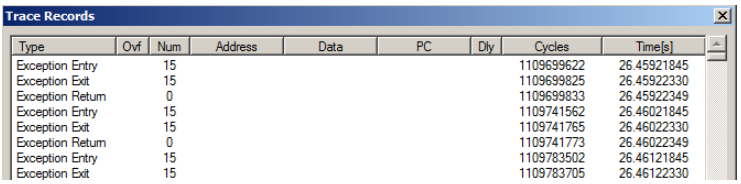
9. Click on OK twice to return to the main µVision menu. SWV is now configured and ready to use.



Display Trace Records:

1. Select File/Save All or click  .
2. Rebuild the source files. 
3. Enter Debug mode. 
4. Click on the RUN icon. 
5. Open Trace Records window by clicking on the small arrow beside the Trace icon  and select Records: 

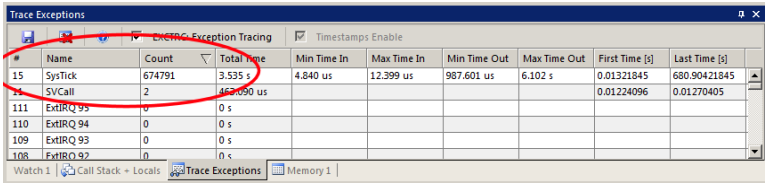
6. The Trace Records window will open:
7. If you see Exceptions as shown, SWV is working correctly. If not, the most probably cause is a wrong Core Clock:.
8. Double-click inside Trace Records to clear any spurious first frames.
9. Exception 15 is the SYSTICK timer. It is the timer provided for RTOS use.
10. All frames have a timestamp displayed.
11. Exception Return means all exceptions have returned. This can be used to detect Cortex exception tail-chaining.



Display Trace Exceptions:

1. Select the Trace Exception tab (located beside the Watch and Memory windows) or select it in Step 5 above.
2. Click in the Count column heading to display SysTick.
3. Unselect EXCTRC: 

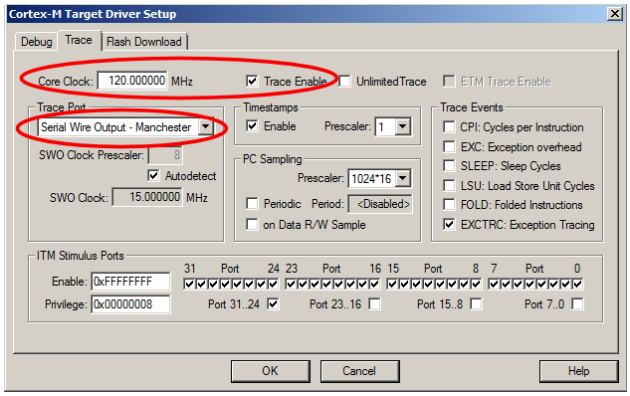
4. This is a quick way to disable Trace Exceptions if you have SWO overload. Exception frames are not captured and the bus load is less on the single bit SWO pin.
5. Reselect EXCTRC:



6b) Using Serial Wire Viewer (SWV) with ULINK*pro*: (using the 1 bit SWO Port)

1) Configure SWV: (You can also use the 4 bit Trace Port with the ULINK*pro*.)

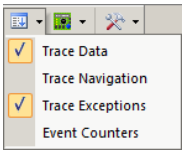
1. µVision must be stopped and in Edit mode (not Debug mode). Blinky must be loaded.
2. Connect a ULINK*pro* to J5 K64 JTAG and change the power jumpers as described on page 10.
2. Create a new target option for ULINK*pro* as described on page 11. Select ULINKpro: in the pulldown menu.
3. Select Options for Target or ALT-F7 and select the Debug tab. ULINKpro must be visible in the dialog box.
4. Select Settings: Settings on the right side of this window.
5. Click on the Trace tab. The window below is displayed. Confirm these settings are correct.
6. Set Core Clock: to 120 MHz. ULINK*pro* uses this only to calculate timings displayed in some windows.
7. Select the Trace Enable box.
8. Unselect ETM Trace Enable (will look at this later).
9. In Trace Port, select Serial Wire Output - Manchester.
10. Select EXTRC to display exceptions and interrupts.
11. Click on OK twice to return to the main µVision menu. SWV is now configured and ready to use.
12. In this configuration, SWV data will be output on the 1 bit SWO pin.



TIP: If Sync Trace Port with 4-bit Data is chosen in Trace Port: box, SWV data is sent out the 4 bit Trace Port pins. This has much higher data throughput that the 1 bit SWO pin.

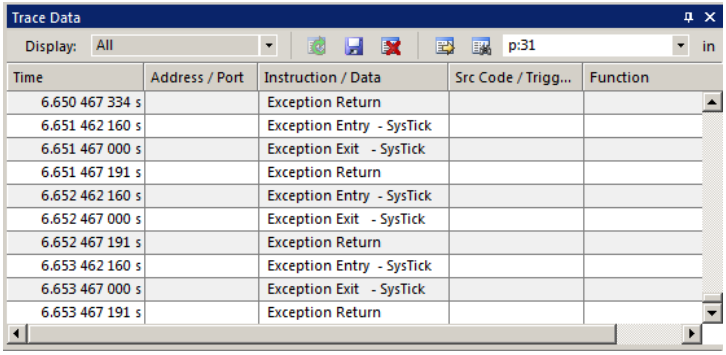
2) Display the Trace Data window:

1. Select File/Save All or click . Click on the Rebuild icon to build the source files. .
2. Enter Debug mode. Click on the RUN icon. .
3. Open the Trace Data window by clicking on the small arrow beside the Trace icon:
4. The Trace Data window shown below will open.
5. STOP the program to display the Exceptions as shown below:



TIPS:

1. The Trace Data window is different than the Trace Records window provided with ULINK2.
2. Clear the Trace Data window by clicking
3. The contents of the Trace Data window can be saved to a file.
4. ULINK*pro* does not update the Trace Data window while the program is running.
5. The Trace Exceptions window does update in real-time. Select the Trace Exceptions window to see the SysTick updated while running Blinky. Double click in the Count column heading to bring SysTick to the top of the list.
6. The Trace Port outputs SWV data faster than the 1 bit SWO with UART (ULINK2) or Manchester with ULINK*pro*. The 1 bit SWO port can still be useful for very high CPU speeds that ETM is unable to handle. (> 100 MHz)



7) Using the Logic Analyzer (LA) with ULINK2, ULINK-ME, ULINK*pro* or J-Link:

This example will use a ULINK2, ULINK*pro* or a J-Link with the Blinky example. Please connect a your debug adapter to your Kinetis board and configure it for Serial Wire Viewer (SWV) trace as described on the previous two pages.

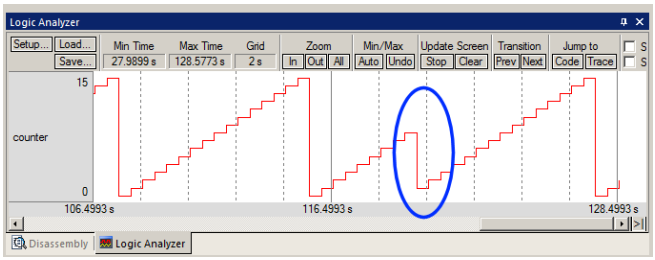
µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using the Serial Wire Viewer as implemented in the Kinetis. LA uses the same comparators as Watchpoints so all can't be used at same time.

1. SWV must be configured as found on the two previous two pages for the debug adapter you are using.
2. µVision must be in Debug mode.
3. Run the program. **TIP:** Recall you can configure the LA while the program is running or stopped.
4. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.
5. Locate the global variable **counter** in Blinky.c. You declared it near line 18.
6. Right click on **counter** and select Add counter to… and select Logic Analyzer.

TIP: If an error results when adding counter to the LA, the most probable cause is SWV is not configured correctly.

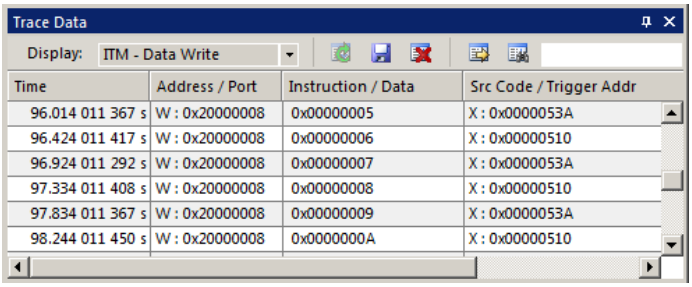
7. In the LA, click on Setup and set Max: in Display Range to 0x0F. Click on Close.
8. The LA is now configured to display counter in a graphical format.
9. **counter** should still be in the Watch and Memory windows. It will be incrementing if the program is running.
10. Adjust the Zoom OUT icon in the LA window to about 1 sec or so to get a nice ramp as shown below.
11. In the Memory 1 window, right click on the **counter** data field.
12. In Modify Memory at 0x2000 0008, at an interesting **counter** value, enter 0 and press Enter.
13. This modified value will be displayed in the LA window as shown below inside the blue circle:

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



1. Select Debug/Debug Settings. Select the Trace tab.
2. Select On Data R/W Sample. Unselect EXCTRC. This is to lessen the load on SWO pin.
3. Click OK twice. This adds addresses to the Src Code/Trigger Addr column.
4. Clear the Trace Data or Trace Records window.

   Double click for ULINK2 or for ULINK*pro*:
5. RUN the program. STOP the program.
6. Open the Trace Data or Trace Records window.
7. The window similar to this opens up: This one is for ULINK*pro*. ULINK2 is different.
8. In the Display box, select ITM Data Write:
9. The first line in ***this*** Trace Data window means:



   The instruction at 0x0000 053A caused a write of data 0x05 to address 0x2000 0008 at the listed time in seconds.
10. If using a ULINK*pro*, in the Trace Data window, double click on a data write frame and the instruction causing this write will be highlighted in the Disassembly and the appropriate source windows.

TIP: The Src Code/Trigger Addr column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin if you are not interested in it. With a ULINK2, this column is called PC.

TIP: The ULINK2 gives a different Trace window. It is the same Trace Records as shown elsewhere in this document.

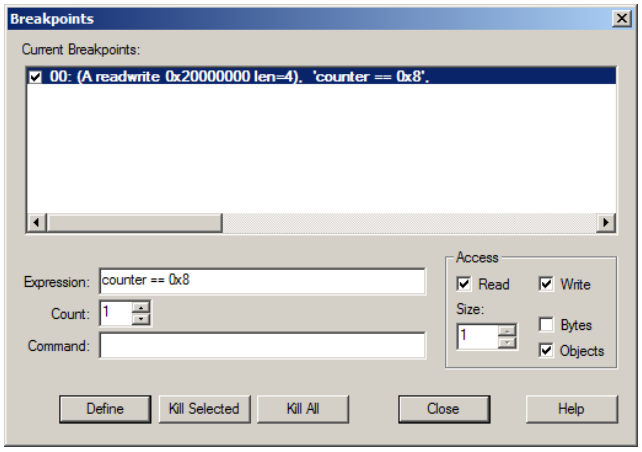
TIP: Raw addresses can also be entered into the Logic Analyzer. An example is: *((unsigned long *)0x20000000)

8) Watchpoints: Conditional Breakpoints: With P&E OpenSDA, ULINK2, ULINKpro or J-Link.

Kinetis Cortex-M4 processors have two Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same CoreSight components as Watchpoints in its operations. A Watchpoint requires two of the four comparators. Keil documentation often refers to Watchpoints as Access Breaks. µVision uses only one at this time.

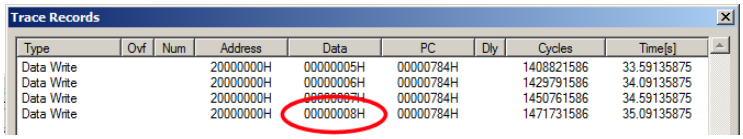
µVision warns you if you attempt to set more than one Watchpoint. SWV or ETM do not need to be configured and any ULINK or J-Link can be used for Watchpoints.

1. Use the Blinky example from the previous page. A Watchpoint can be set while the program is running.
2. While in Debug mode, click on Debug and select Breakpoints or press Ctrl-B.
3. The SWV Trace does not need to be configured for Watchpoints. However, we will use it in this exercise.
4. OpenSDA does not currently support SWV and therefore does not support the LA which uses SWV.
5. In the Expression box enter: `counter == 0x8`. Select both the Read and Write Access for convenience.
6. Click on Define and it will be accepted as shown here:
7. If the program is running and when counter = 0x8, the program will stop as the Watchpoint is immediately set.
8. Click on Close.
9. With ULINK2, double-click in the Trace Records window to clear it or with ULINK*pro* click  to clear Trace Data for convenience.
10. Set **counter** in the Watch 1 or Memory 1 window to 1. This is to allow the program to run for a bit.
11. Click on RUN if the program is stopped. 
12. When **counter** equals 0x8, the program will stop. This is how a Watchpoint works.



TIP: P&E OpenSDA will trigger on every data read or write to counter with no regard to the data value entered.

13. You will see **counter** had incremented in the Logic Analyzer as well as in the Watch window.
14. Note the four data writes in the Trace Records window shown below. The last one is when counter = 0x08. The data writes to **counter** are shown plus the address the data written to and the PC of the write instruction. This is with a ULINK2 or ULINK-ME. The ULINK*pro* will display a different window and the program must be stopped to display it. This is also true for the J-Link. OpenSDA does not support SWV at this time.



TIP: Data writes are displayed in the SWV Trace window ***only*** when a variable is displayed in the Logic Analyzer.

15. The only type of expressions you can currently enter is the equal compare (= =) or address only compare.
16. To repeat this exercise, select RUN twice to get **counter** past 8.
17. When finished, open the Breakpoints window and either use Kill All to delete the Watchpoint or deselect it by unchecking it. Having an undeleted Watchpoint activate unexpectedly can be rather confusing while debugging.
18. Leave Debug mode  for the next exercise.

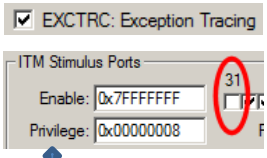
TIP: To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Unselect the Watchpoint Current Breakpoints window. Modify the Watchpoint. Click on Define to create another Watchpoint. You probably should delete the old one by highlighting it and click Kill Selected.

J-Link does not currently display Data reads or writes in its trace window and the LA does not display with Watchpoints.

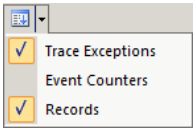
9) Exceptions and Interrupts using SWV: **Only ULINK2, ULINK*pro* or J-Link. No OPenSDA.**

The Kinetis family using the Cortex-M4 processor has many useful interrupts and SWV makes it easy to determine when and how often they are being activated. Interrupts are a subset of Exceptions. This page assumes you are using a ULINK2.

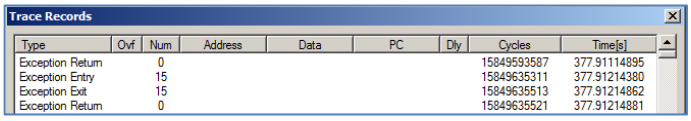
Exceptions are enabled in SWV by EXCTRC in the Trace Config or Trace Exception windows:



1. Click Options for Target:  If using a ULINK*pro*, you can skip steps 2 through 4.
2. Select the Debug tab and then click the Settings box. Select the Trace tab.
3. Unselect ITM 31 to lessen the output on the SWO pin to lessen trace data overflows: This turns the Event Viewer off and none of its data is output on the SWO pin.
4. Click on OK twice to return to the main µVision menu.
5. Open the Trace Records window. 
6. If you are using a ULINK*pro* or J-Link, this name is Trace Data.
7. Double click in the Trace Records window to clear it or use  with ULINK*pro*.
8. Click RUN to start the program.
9. You will see a window similar to the one here with Exceptions frames displayed:



TIP: With a ULINK*pro* or a J-Link, stop the program to see these exceptions. They are displayed differently.



What Is Happening ?

- You can see SysTick Exception 15 occurrences with timestamps.
- **Entry:** When the exception enters.
- **Exit:** When the exception exits or returns.
- **Return:** When all the exceptions have returned. This is useful to detect Cortex interrupt tail-chaining.

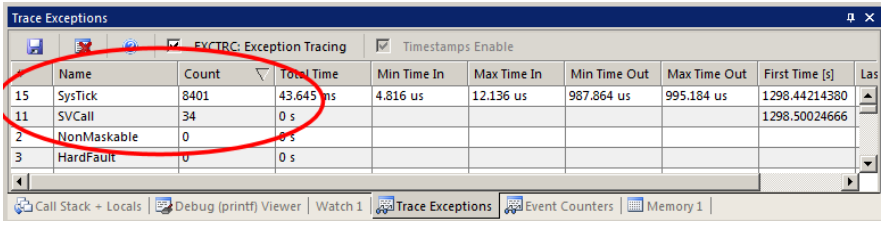
TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

10. Right click in the Trace Records window and unselect Exceptions. Only the data writes from the LA are displayed.

TIP: The SWO pin is one pin on the Cortex-M4 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. ULINK*pro* has the option of sending this data out the 4 bit Trace Port with much greater throughput tan the SWO pin. ULINK*pro* handles SWV data faster than a ULINK2 or J-Link.

Trace Exceptions window:

1. Select the Trace Exceptions tab and the window below opens:
2. Click in the Count column heading to bring exceptions that have occurred to the top. Exceptions 15 and 11 will be displayed and are updated in real time without intrusions to your program using SWV.
3. Note the number of times these have happened. This is useful information in case interrupts come too fast or slow.
4. Scroll down in this window and note the other exceptions are listed by name. This comes from the SVD file.
5. You can clear this trace window by clicking on the clear icon.
6. All this information is displayed in real-time and without stealing CPU cycles !



ULINK*pro*: The Trace Exceptions update while the program runs. Stop the program to update the Trace Data window.

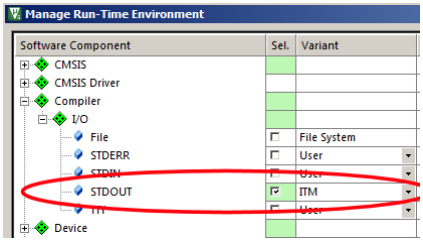
If you are using a ULINK*pro* and RTX, you can view interrupts with processing times in the Event Viewer.

OpenSDA does not support anything on this page. **J-Link** supports everything except no displaying Data read and writes.

10) *printf* using ITM 0 (Instrumentation Trace Macrocell) SWV is required:

ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in the Debug (printf) Viewer window. It is possible to send ITM data to a file: www.keil.com/appnotes/docs/apnt_240.asp.

1. Stop the program and exit Debug mode.



Add STDOUT File (retarget_io.c):

1. Open the Manage Run-Time Environment window (MRTE).
2. Expand Compiler and I/O as shown here:
3. Select STDOUT and ITM. This adds the file retarget_io.c to the project.
4. Ensure all blocks are green and click OK to close the MRTE.

Add printf and #include <stdio.h> to Blinky.c:

1. In Blinky.c near line 14, add this line: #include <stdio.h>
2. Inside the function LED_On found near line 50, add this line: printf("LED On\n");
3. Inside the function LED_Off found near line 62, add this line: printf("LED Off\n");

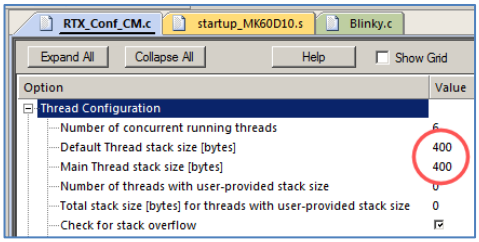
Enable Microlib and Configure Serial Wire Viewer:

1. Select Options for Target or ALT-F7. Select the Target tab.
2. Select Use MicroLIB. If you don't want to use MicroLIB add 200 bytes to the Heap in startup_MK60D10.s.
3. Select the Debug tab. Select Settings and then the Trace tab.
4. Unselect On Data R/W Sample and ITM Port 31. (this is to help not overload the SWO pin so this step is optional)
5. Select ITM Port 0. ITM Stimulus Port "0" enables the Debug (prinftf) Viewer. All ports 1 through 30 are unused.
6. Click OK twice to return to the main µVision menu.

Increasing RTX stack size: This step is *only* if you are using RTX: You are not using RTX so skip these three steps !

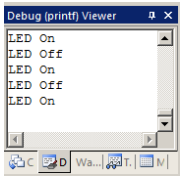
When a printf statement is added to a thread, you must increase the RTX stack size as follows:

1. Select the RTX_Conf_CM.C tab. Click on the Configuration Wizard tab at the bottom of this window.
2. Set the Default and Main stack size to 400 bytes as shown here:
3. In general, if you experience trouble with RTX operation, try increasing the number of threads and size of the RTX stack. This stack is not the same as the CPU system stack as referenced by the Stack Pointer but it is still in RAM.



Compile and Run the Project:

1. Select File/Save All or click.
2. Rebuild the source files and enter Debug mode.
3. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
4. In the Debug (printf) Viewer you will see the printf statements appear.
5. Right click on the Debug window and select Mixed Hex ASCII mode. Note other settings.



Obtaining a character typed into the Debug printf Viewer window from your keyboard:

It is possible for your program to input characters from a keyboard with the function ITM_ReceiveChar in core.CM4.h.

This is documented here: www.keil.com/pack/doc/CMSIS/Core/html/group__ITM__debug__gr.html.

A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer.

TIP: ITM_SendChar is a useful function you can use to send characters out ITM. It is found in core.CM4.h.

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enable only those SWV features that you need. If you need high performance SWV, a ULINK*pro* using 4 bit Trace Port is fastest.

11) Trace Configuration Fields and General Trace Information:

For ULINK2 see www.keil.com/support/man/docs/ulink2/ulink2_ctx_trace.htm

For ULINK*pro* see www.keil.com/support/man/docs/ulinkpro/ulinkpro_ctx_trace.htm

For Segger J-Link see http://www.keil.com/support/man/docs/jlink/jLink_cortexTrace.htm

SWO Overload:

Serial Wire Viewer data (frames) are output on the 1 pin SWO with a ULINk2 or J-Link. This pin is located on 13 on the 20 pin legacy connector (not on Tower boards). It is also available on the 10 or 20 pin CoreSight Debug connectors. SWO is on pin 6 on the 10 pin and pin 14 on the 20 pin. SWO is multiplexed with JTAG TDO pin. This means SWD (Serial Wire Debug) must be used and not JTAG mode. This is easily set in µVision. SWD = SW in µVision.

SWO is one pin and it can be challenging to send a large amount of SWV data through it. A ULINK*pro* using Manchester mode on the SWO pin is more efficient. For even more throughput, ULINK*pro* can output SWV on the 4 bit Trace Port. This port is available on most NXP Cortex-M3, M4 and M7. Cortex-M0 does not have SWV nor the Trace Port. It does have DAP read and writes to RAM and Peripheral addresses.

It is important to ensure the Serial Wire Output (SWO) pin is not overloaded. µVision will alert you when an overflow occurs with an "X" in the Trace Records window or with a "D" or a "O" in the ULINK*pro* Trace Data window. µVision easily recovers from these overflows and immediately continues displaying the next available trace frame. Dropped frames are somewhat the normal situation especially with many data reads and/or writes.

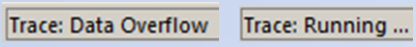
Variables entered in the LA with the resulting Data writes to the Trace Records window plus exceptions and interrupts create a trace frame each time there is an event. You may have to sample a rapidly changing variable.

ULINK*pro* can process SWV information much faster than the ULINK2 or ULINK-ME can. This results in fewer dropped frames especially with higher data transfer rates out the SWO pin. ULINK*pro* has the option of collecting information from the 4 bit Trace Port instead of the 1 bit SWO pin. Data overruns are often associated with a fast stream of data reads and writes which are created in the Logic Analyzer. Minimize these issues by displaying only the information you really need or use a ULINK*pro* with UART Manchester or better the 4 bit Trace Port.

Since LA is event driven, you can reduce overload by sampling a variable in your program and sending this to the LA.

T **race Overload Display:**

An indication of Trace overflows or not is displayed at the bottom of µVision.
It is common to see Data Overflow displayed and this is not always a need for high concern.

Here are examples of these notices: 

Display CPU Frequency:

The global variable SystemCoreClock contains the speed of the CPU. It is declared and set in system_MK64F12.c.

1. Select View/Watch Windows/Watch 1 if Watch 1 is not already open.
2. Double click on <Enter expression> and enter SystemCoreClock and press the Enter key.
3. A value will appear. Right click on the variable name and unselect Hexadecimal Display.
4. The CPU frequency will now display in Hertz.
5. When Debug mode is first entered and the program counter is still at the start of main(), the default CPU frequency is usually lower.
6. After SystemClock_Config() is executed, it will higher according to system clock initialization if any.

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

1) DSP SINE example:

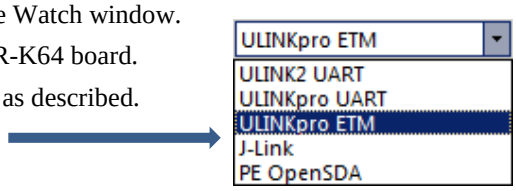
ARM CMSIS-DSP libraries are offered for ARM Cortex-M0, Cortex-M3, Cortex-M4 and Cortex-M7 processors. DSP libraries plus all sources are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS\.
See www.keil.com/cmsis and https://github.com/ARM-software/CMSIS_5.

This example creates a ***sine*** wave, then creates a second to act as ***noise***, which are then added together (***disturbed***), and then the noise is filtered out (***filtered***). The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer.

This example incorporates the Keil RTOS RTX. RTX has a BSD (soon Apache 2.0) license. All source code is provided.

This program will run with OpenSDA but to see the interesting and useful SWV features you need any ULINK or a J-Link.

1. This program was copied to C:\00MDK\Boards\Freescale\TWR-K64F120M on page 5.
2. Open the project file C:\00MDK\Boards\Freescale\TWR-K64F120M\sine.uvprojx with µVision.
3. **To use P&E OpenSDA:** connect a USB cable to J2 K20 DEBUG USB connector. No SWV including the LA will function with OpenSDA. The program must be stopped to update the Watch window.
4. **For any ULINK or J-Link:** Connect it to J5 K64 JTAG on the TWR-K64 board.
5. Adjust the power jumpers as described on page 10. Power the board as described.
6. Select your debug adapter from the pull-down menu as shown here:



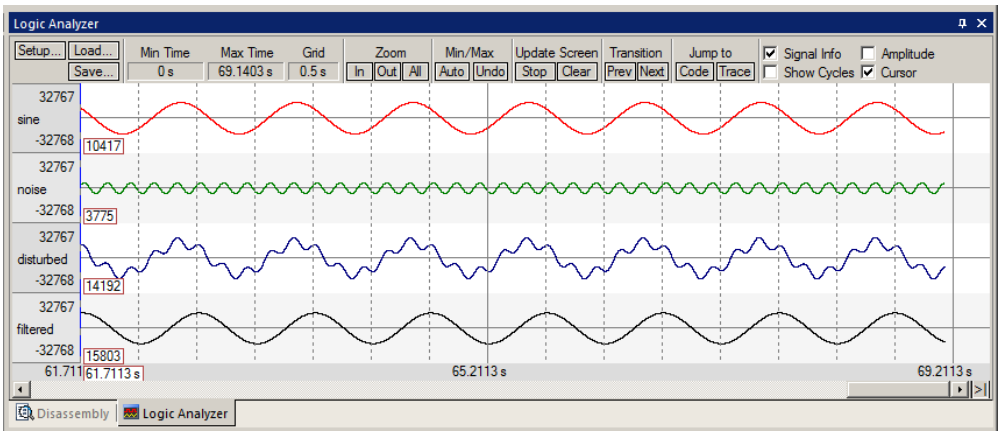
7. Compile the source files by clicking on the Rebuild icon.  .
8. Enter Debug mode by clicking on the Debug icon.  The Flash will be programmed.

TIP: The default Core Clock: is 120 MHz for use by the Trace configuration window under the Trace tab.

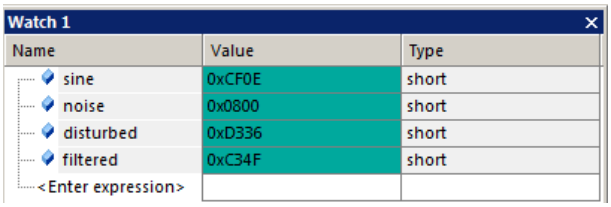
9. Click on the RUN icon.  Open the Logic Analyzer window  and the Watch 1 window: View/Watch/
10. This project has Serial Wire Viewer configured and the Logic Analyzer and Watch 1 loaded with the four variables.
11. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom for an appropriate display. Displayed are 4 global variables: **`sine`**, **`noise`**, **`disturbed`** and **`filtered`**.

Trouble: If one or two variables display no waveform, disable ITM Stimulus Port 31 in the Trace Config window. The SWO pin is probably overloaded if you are using a ULINK2. ULINK*pro* handles SWV data faster than a ULINK2 or J-Link can. Make sure the Core Clock is set to 120. If the variables in Watch 1 are changing, the program is running correctly.

12. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below:

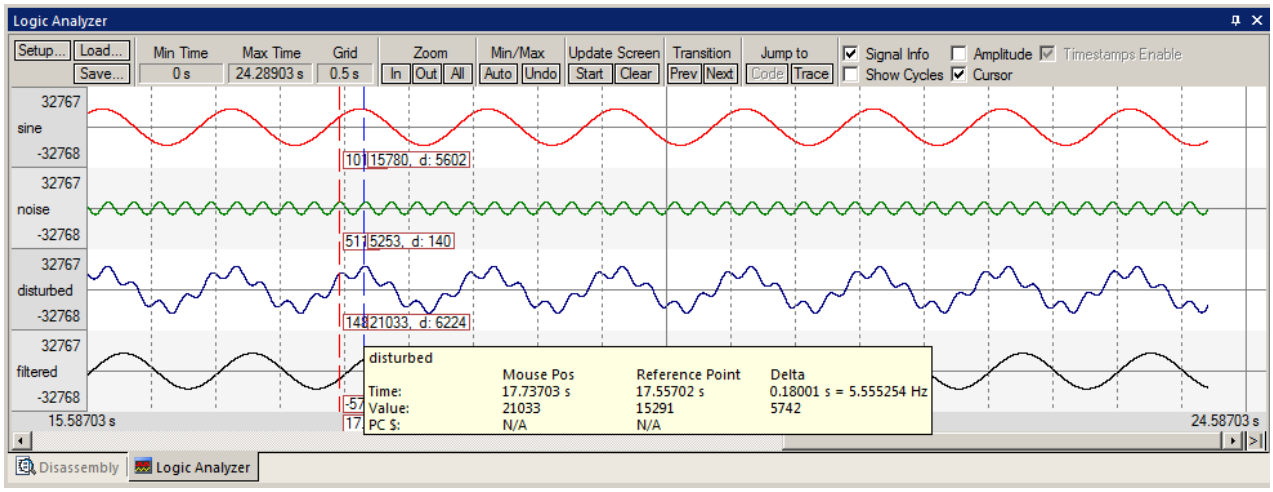


13. Open the Trace Records window and the Data Writes to the four variables are displayed using Serial Wire Viewer. When you enter a variable in the LA, its data write is also displayed in the Trace window. With ULINK*pro* you must stop the program to display the data Trace Data window. J-Link does not display any data read or write operations. OpenSDA has no SWV support.
14. Select View/Serial Windows/Debug (printf) Viewer. ASCII data is displayed from the printf statements in DirtyFilter.c. Not with OpenSDA.
15. Leave the program running.
16. Close the Trace Records window.



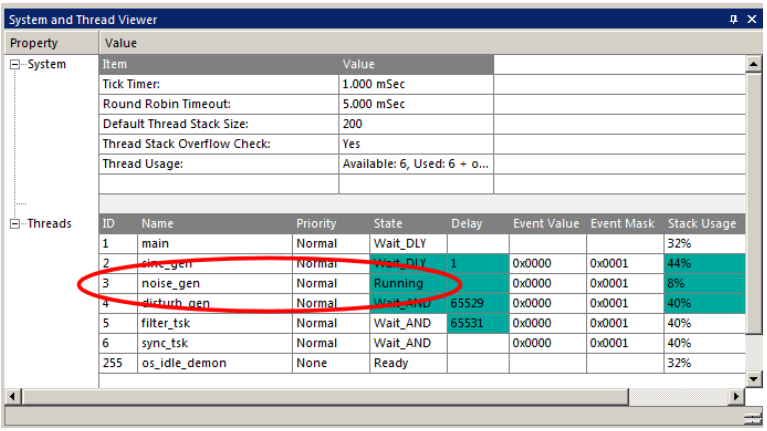
2) Signal Timings in Logic Analyzer (LA):

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



3) RTX System and Threads Viewer: **This works with OpenSDA, ULINK and J-Link.**

1. Click on Start in the Update Screen box to resume the collection of data. The program must be running.
2. Open Debug/OS Support and select RTX System and Thread Viewer. A window similar to below opens up. You may have to click on its header and drag it into the middle of the screen to comfortably view it.
3. As the various threads switch state this is displayed. Note most of the CPU time is spent in the idle daemon: it shows as Running. The processor spends relatively little time in other tasks. You will see this illustrated clearly on the next page. It is possible to adjust these timings to give more CPU time to various threads as needed.
   NOTE: With P&E OpenSDA, the program must be stopped to update this window.
4. Set a breakpoint in each of the four tasks in DirtyFilter.c by clicking in the left margin on a grey area. Do not select while(1) as this will not stop the program.
5. Click on Run and the program will stop at a thread and the System and Threads Viewer will be updated accordingly. In the screen below, the program stopped in the noise_gen task:
6. Clearly you can see that noise_gen was Running when the breakpoint was activated.
7. Each time you click on RUN, the next task will display as Running.
8. Remove all the breakpoints by clicking on each one. You can use Ctrl-B and select Kill All.
9. Stay in Debug mode for the next page.



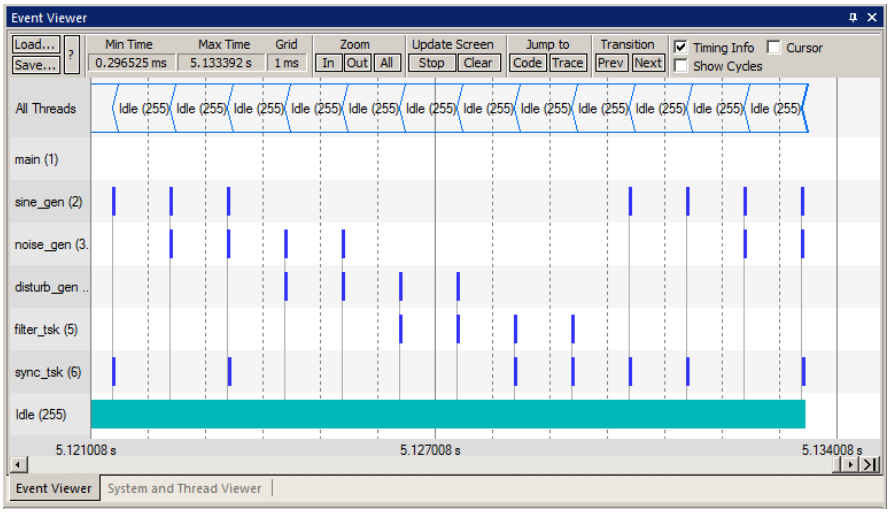
TIP: You can set/unset hardware breakpoints while the program is running.

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

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

4) RTX Event Viewer (EV): ULINK2, ULINK*pro* and J-Link: Not with OpenSDA.

1. ***If you are using a* ULINK*pro*, *skip this step unless you want to see SWV overload.:*** Stop the program. Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables and select Close. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might/will occur.
2. Select Debug/Debug Settings.
3. Click on the Trace tab.
4. Enable ITM Stimulus Port 31. Event Viewer uses this port to collect its information.
5. Click OK.
6. Click on RUN  .
7. Open Debug/OS Support and select Event Viewer. The window here opens up:

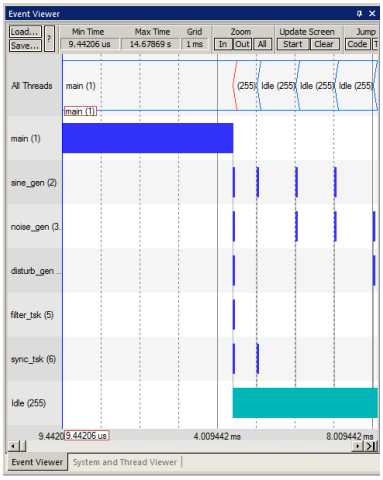


TIP: If Event Viewer is still blank, exit and re-enter Debug mode.  

Main Thread:

1. Select Stop in the Update Screen. Scroll to the beginning of the Event Viewer.
2. The first thread in this program was main() as depicted in the Event Viewer. The main thread is the main() function in DirtyFilter.c It runs some RTX initialization code at the beginning and is stopped with osDelay(osWaitForever);.

TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some SWO or Trace Port bandwidth. Try turning off the exceptions with EXTRC.



3. The 5 running threads plus the idle daemon are displayed on the Y axis. Event Viewer shows which thread is running, when and for how long.
4. Click Stop in the Update Screen box.
5. Click on Zoom In so three or four threads are displayed as shown here:
6. Select Cursor. Position the cursor over one set of bars and click once. A red line is set here:
7. Move your cursor to the next set and total time and difference are displayed.
8. Since you enabled Show Cycles, the total cycles and difference is also shown.

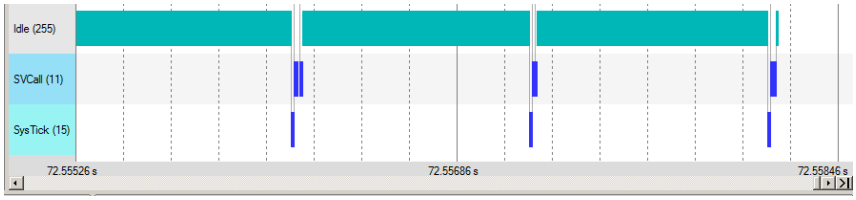
The 1 msec shown is the SysTick timer value which is set in RTX_Conf_CM.c in the OS_CLOCK and OS_TICK variables.

Using a Keil ULINK*pro* to view Interrupt Handler execution times:

SWV Throughput: ULINK*pro* is much better with SWO bandwidth issues. It has been able to display both the EV and LA windows. The ULINK*pro* ETM and ULINK*pro* UART modes are preconfigured Options for Target.

ULINK*pro* can also use the 4 bit Trace Port for even faster operation for SWV. Trace Port use is mandatory for ETM trace. A ULINK*pro* in ETM mode provides program flow debugging, Code Coverage and Performance Analysis. ULINK*pro* also supports ETB (Embedded Trace Buffer) as found in many Kinetis processors.

Exceptions: A ULINK*pro* displays exceptions at the bottom of the Event Viewer. Shown here are the SysTick and SVCall exceptions. You can easily measure the duration of the time spent in the handlers. Any other exception events such as DMA will also be displayed here.



You will be able to easily measure the time a handler runs with the Event Viewer techniques you have learned.

Part D) ETM Trace: ETM is available only with Keil ULINK*pro*:

1) Introduction:

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

1) The window containing the trace frames is now called Trace Data. More complete filtering is available.
2) The SWV (Serial Wire Viewer) data is sent out the 1 bit SWO pin with the ULINK2 using UART encoding. The ULINK*pro* can send SWV data ***either*** out this same SWO pin using Manchester encoding or through the 4 bit Trace Port. SWV throughput is greater than ULINK2 UART and is the fastest using the 4 bit Trace Port. The Trace Port is found on the 20 pin Cortex connector (J5 K64 JTAG) and is configured in the Trace Configuration window. ETM frames are always sent out the Trace Port and if this is the case, SWV data is also sent out this port.
   ETB: ULINK*pro* can access the Embedded Trace Buffer (ETB) which is an on-chip instruction trace buffer. See the bottom of the next page for more information.

ULINK*pro* offers:

1) Faster Flash programming than the ULINK2.
2) Serial Wire Viewer frames are provided at a much faster data throughput than a ULINK2 or J-Link.
3) ETM Instruction Trace is added which provides a record of all executed instructions. The Trace Data window has Trace start and stop, filtering and ability to save records to a file.
4) **Code Coverage**: were all the assembly instructions executed ? This data can be saved as a report.
5) **Performance Analysis**: where the processor spent its time in graphical and numerical formats.
6) **Execution Profiling**: How long instructions, ranges of instructions, functions or C source code took in both time and CPU cycles as well as number of times these were executed.
7) **Exception handler timing in the RTX RTX System and Thread Viewer:**

General ETM with ULINK*pro* information:

If you are a power user of Serial Wire Viewer and have problems with SWV signal overloads, a ULINK*pro* is an excellent investment. A ULINK*pro* is worth the small cost and is easy to operate and analyze the results to obtain useful information.

Initialization File: This is enabled on the next page:
A script must be executed upon entering Debug mode to configure the K64 ETM registers and GPIO ports. This script is provided as TracePortK64.ini. This file is located in C:\00MDK\Boards\Freescale\TWR-K64F120M\DSP\. This is an ASCII file. This file is specific to Kinetis processors as their proprietary GPIO ports must be configured. Note other Kinetis processors might need a different .ini file depending on the pins the Trace Port is located.

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

This ini file might be optional in future Packs in the Debug Description files.

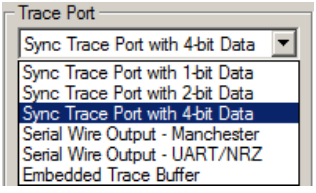
TIP: General information on ETM trace is found here: www.keil.com/support/man/docs/ulinkpro/ulinkpro_ctx_trace.htm

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

ULINK*pro* can send the SWV signals out the 4 bit Trace Port which does not share any pins with JTAG. SWO shares the JTAG signal TDO hence the conflict.

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

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



Trace Port options for use with ULINK*pro*:

Serial Wire Output – Manchester: output SWV out SWO pin. No ETM.

Sync Trace Port with 4 bit data: ETM out Trace Port pins. Up to 4 bits.

Embedded Trace Buffer: (ETB) : a small RAM inside the Kinetis used as a trace memory. Limited frame capture capability. ETB runs at full CPU speed.

2) Configuring ULINK*pro* ETM Trace: The RTX_Blinky example is used here:

Connecting and configuring the physical connection to the TWR-K64:

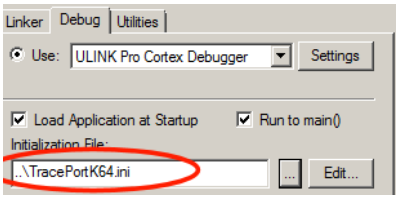
1. Connect a ULINK*pro* to the J5 K64 JTAG CoreSight ETM connector.
2. Set power jumpers as described on page 10. Power the board and ULINK*pro*.
3. The RTX_Blinky example is configured for a ULINK2 and not the ULINK*pro*. We will add this configuration here.

Create a new Target Selection for ULINK*pro*:

1. Select Project/Manage/Project Items… or select: 
2. In the Project Targets area, select NEW  or press your keyboard INSERT key.
3. Enter **ULINKpro ETM** and press Enter. Click OK to close this window.
4. In the Target Selector menu, select the ULINKpro ETM selection you just made:

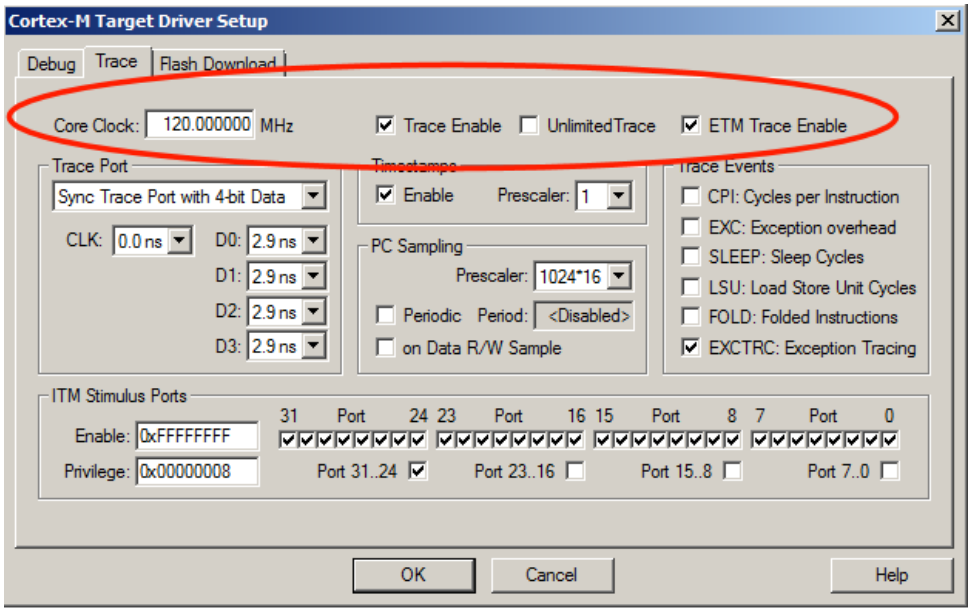
Select ULINK*pro* and enter the TracePortK64.ini file:

1. Select Options for Target  or ALT-F7. Click on the Debug tab to select a debug adapter.
2. Select ULINK Pro Cortex Debugger as shown below:
3. Click on the browse icon  and navigate to C:\00MDK\Boards\Freescale\TWR-K64F120M\DSP\.
4. Click on TracePortK64.ini. It will be entered in the Initialization box:
5. If you click on the Edit icon, TracePortK64.ini will be opened with the other source files. You can then view it.



Configuring ETM Trace *(and SWV using the 4 bit Trace port)*:

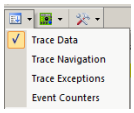
1. In the Options for Target window under the Debug tab, click on Settings:
2. Click on the Trace tab. The window below is displayed.
3. Select Trace Enable. This selects SWV.
4. Set Core Clock: to 120 MHz. ULINK*pro* uses this to calculate timings in various windows.
5. In Trace Port, select Sync Trace Port with 4-bit data. It is best to use the widest size which is 4 bits.
6. Set Trace Port CLK: to 0 ns clock delay. D0 through D3 should be set to 2.9 ns as shown below: **Note:** not all Kinetis processors need these delay times. If ETM doesn't work try 0 or 4.9 ns.
7. Select ETM Trace Enable. This activates ETM Trace.
8. Select EXCTRC and leave everything else at default as shown below. Only ITM 31 and 0 need to be selected.
9. Click on OK twice to return to the main µVision menu.
10. ETM and SWV are now configured through the Trace Port.
11. Select File/Save All. 



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

The RTX_Blinky project has now been configured on the previous page to provide ETM Trace.

1. Compile the Blinky source files by clicking on the Rebuild icon.  . You can also use the Build icon beside it.
2. Enter Debug mode.  Select OK if the Evaluation Mode box appears.
3. DO NOT CLICK ON RUN YET !!! If you did, simply exit and re-enter Debug mode.
4. Open the Trace Data window by clicking on the small arrow beside the Trace icon as shown here:



5. Right click anywhere in the Trace Data window and select Show Functions. Size this window.
6. Examine the Trace Data window as shown below: This is a complete record of all the program flow since RESET until µVision halted the program at the start of main() since Run To main is selected in µVision.
7. In this case, Time 284 500 shows the last instruction to be executed. (BX lr). This is the jump to main().
8. The last two entries are an Exception entry (SVCall) and a Exception Return. These are Serial Wire Viewer frames. The red D indicates a trace overflow.

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
0.000 284 300 s	X : 0x0000198A	BX lr	}	rt_stk_check
	X : 0x0000059C	POP {r2-r3}	POP {R2,R3}	SVC_Handler
	X : 0x0000059E	STR r2,[r3,#0x00]	STR R2,[R3] ; os_tsk.run = os_tsk.new	SVC_Handler
	X : 0x000005A0	LDR r12,[r2,#0x28]	LDR R12,[R2,#TCB_TSTACK] ; os_tsk.new->tsk_stack	SVC_Handler
	X : 0x000005A4	LDM r12!,{r4-r11}	LDMIA R12!,{R4-R11} ; Restore New Context	SVC_Handler
	X : 0x000005A8	LDRB r0,[r2,#0x25]	LDRB R0,[R2,#TCB_STACKF] ; Stack Frame	SVC_Handler
	X : 0x000005AC	CMP r0,#0x00	CMP R0,#0 ; Basic/Extended Stack Frame	SVC_Handler
	X : 0x000005AE	ITEE EQ	MVNEQ LR,#:NOT:0xFFFFFFFD ; set EXC_RETURN value	SVC_Handler
	X : 0x000005B0	MVNEQ lr,#0x02		SVC_Handler
0.000 284 467 s	X : 0x000005B4	*MVNNE lr,#0x12	MVNNE LR,#:NOT:0xFFFFFFED	SVC_Handler
0.000 284 483 s	X : 0x000005B8	*VLDMIANE r12!, {s16-s31}	VLDMIANE R12!,{S16-S31} ; restore VFP hi-registers	SVC_Handler
	X : 0x000005BC	MSR PSP,r12	MSR PSP,R12 ; Write PSP	SVC_Handler
0.000 284 500 s	X : 0x000005C0	BX lr	BX LR	SVC_Handler
0.000 284 558 s		Exception Exit - SVCall		
D 0.000 284 63...		Exception Return		

9. In the Register window the PC will display the value of the next instruction to be executed (0x000 07FE in my case). Click on Single Step once. 

	TRACE RUN			
0.000 284 625 s	X : 0x000007FE	BL.W SystemCoreClockUpdate...	SystemCoreClockUpdate();	main
D 0.000 284 66...		Exception Return		

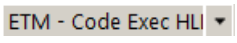
10. The instruction BL.W will display at 0x0000 07FE:
11. Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is a LDR instruction in the RESET_Handler function as shown below:
12. If you use the Memory window to look at location 0x4, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0000 0093A + 1 = 0x093B (+1 says it is a Thumb instruction). These first instructions after RESET are shown below: Note the source information that is displayed including which function and instruction belongs to. The first occurrence in a function is highlighted in orange to make the start of functions easier to find.
13. If you double-click on any line, this will be highlighted in both the Disassembly and relevant source windows.

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x0000093A	LDR r0,[pc,#44] ; @0x00...	LDR R0, =SystemInit	Reset_Handler
0.000 000 100 s	X : 0x0000093C	BLX r0	BLX R0	Reset_Handler
	X : 0x00000980	LDR r0,[pc,#552] ; @0x0...	SCB->CPACR |= ((3UL << 10*2) | (3UL << 11*2)); /* set CP...	SystemInit
	X : 0x00000982	LDR r0,[r0,#0x00]		SystemInit
	X : 0x00000984	ORR r0,r0,#0xF00000		SystemInit

TIP: If you unselect Run to main() in the Debug tab, no instructions will be executed when you enter Debug mode. The PC will equal 0x08B4. You can run or single-step from that point and this will be recorded in the Trace Data window.

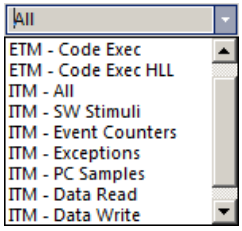
TIP: If you see mostly the same B instructions, these are in the RTX Idle daemon code. To filter these out, select ETM – Code Exec HLL: Only instructions with C source will be displayed. 

4) Finding the Trace Frames you are looking for:

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

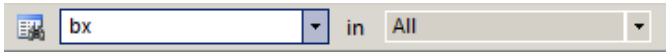
Trace Filters:

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



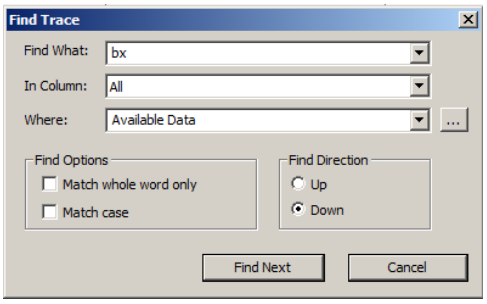
Find a Trace Record:

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



You can select properties where you want to search in the "in" box. "All" is shown in the screen above:

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



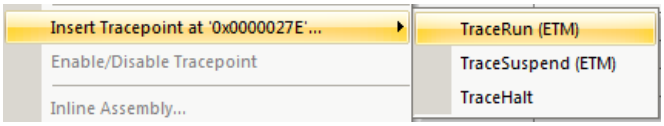
TIP: Or you can press Enter to go to the next occurrence of the search term.

Trace Triggering:

µVision has three trace triggers currently implemented:

1. **TraceRun:** Starts ETM trace collection when encountered.
2. **TraceSuspend:** Stops ETM trace collection when encountered. TraceRun has to have first been set and encountered to start the trace collection for this trigger to have an effect.
3. **TraceHalt:** Stops ETM trace, SWV and ITM. Trace collection can be resumed only with a STOP/RUN sequence. TraceStart will not restart it.

They are selected from the menu shown here:
This menu is accesses by right-clicking on a valid assembly instruction in the Disassemble window.



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

How it works:

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

Sometimes there is some skid past the trigger point which is normal.

Trace Commands:

There are a series of Trace Commands you can enter in the Command window while in Debug mode.

See www.keil.com/support/man/docs/uv4/uv4_debug_commands.htm

TL - Trace List: list all tracepoints.

TK - Trace Kill: tk* kills all tracepoints or tk *number* only a specified one i.e. tk 2.

TIP: TK* is very useful for deleting tracepoints when you can't find them in the source or Disassembly windows.

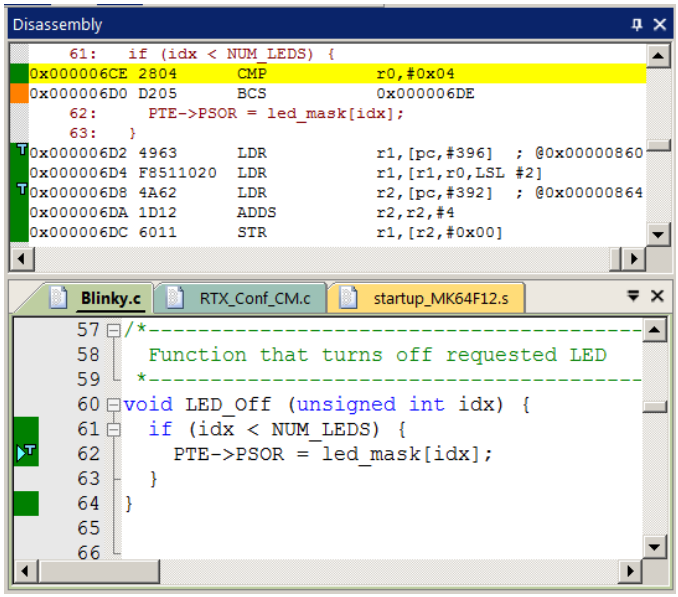
TL is useful for finding any tracepoints set. Results are displayed in the Command window.

5) Setting Trace Triggers:

1. Stop the program and stay in Debug mode.
2. In Blinky.c, click inside the function LED_Off near line 61 or 62 to highlight the code in the Disassembly window.
3. Right-click on LDR at 0x6D2 in the Disassembly window (just after line 63). (your addresses might be different)
4. Select Insert Tracepoint at 0x06D2 and select **TraceRun (ETM).** A cyan T will appear as shown below:
5. Right-click on the third LDR in the left margin at 0x06D8 as shown below in the Disassembly window:
6. Select Insert Tracepoint at 0x06D8 and select **TraceSuspend (ETM).** A cyan T will appear.
7. Clear the Trace Data window for convenience. This is an optional step.
8. Click RUN and after a few seconds click STOP.
9. Filter exceptions out by selecting ETM – Code Exec in the Display in the Trace Data window:
   Display: ETM - Code Exec
10. Examine the Trace Data window as shown below:

You can see below where the trace started on 0x6D2 and stopped on 0x6D8:
All other frames are discarded.

11. In the Command window, enter TL and press Enter. The two Trace points are displayed.
12. **Enter TK* and press Enter to delete all Tracepoints.**



Trace Skid:

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

You might have to adjust the trigger point location to minimize this effect.

This is because of the nature of the comparators in the CoreSight module and it is normal behavior.

Trace Data

Display: ETM - Code Exec | bx | in All

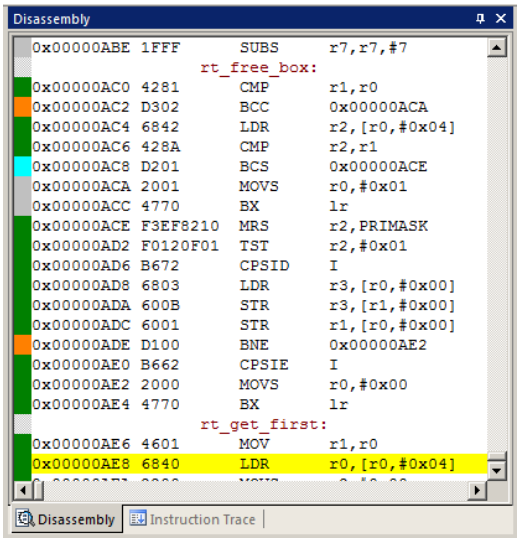
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x000006D2	LDR r1,[pc,#396] ; @0x00000860	PTE->PSOR = led_mask[idx];	LED_Off
	X : 0x000006D4	LDR r1,[r1,r0,LSL #2]		LED_Off
3.582 017 783 s	X : 0x000006D8	LDR r2,[pc,#392] ; @0x00000864		LED_Off
	TRACE RUN			
	X : 0x000006D2	LDR r1,[pc,#396] ; @0x00000860	PTE->PSOR = led_mask[idx];	LED_Off
	X : 0x000006D4	LDR r1,[r1,r0,LSL #2]		LED_Off
4.082 017 783 s	X : 0x000006D8	LDR r2,[pc,#392] ; @0x00000864		LED_Off
	TRACE RUN			
	X : 0x000006D2	LDR r1,[pc,#396] ; @0x00000860	PTE->PSOR = led_mask[idx];	LED_Off
	X : 0x000006D4	LDR r1,[r1,r0,LSL #2]		LED_Off
4.502 026 183 s	X : 0x000006D8	LDR r2,[pc,#392] ; @0x00000864		LED_Off
	TRACE RUN			
	X : 0x000006D2	LDR r1,[pc,#396] ; @0x00000860	PTE->PSOR = led_mask[idx];	LED_Off
	X : 0x000006D4	LDR r1,[r1,r0,LSL #2]		LED_Off
4.582 017 783 s	X : 0x000006D8	LDR r2,[pc,#392] ; @0x00000864		LED_Off

6) Code Coverage:

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

Colour blocks indicate which assembly instructions have been executed.

1. Green: this assembly instruction was executed.
2. Gray: this assembly instruction was not executed.
3. Orange: a Branch is never taken.
4. Cyan: a Branch is always taken.
5. Light Gray: there is no assembly instruction here.
6. RED: Breakpoint is set here.
7. The next instruction to be executed.



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

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

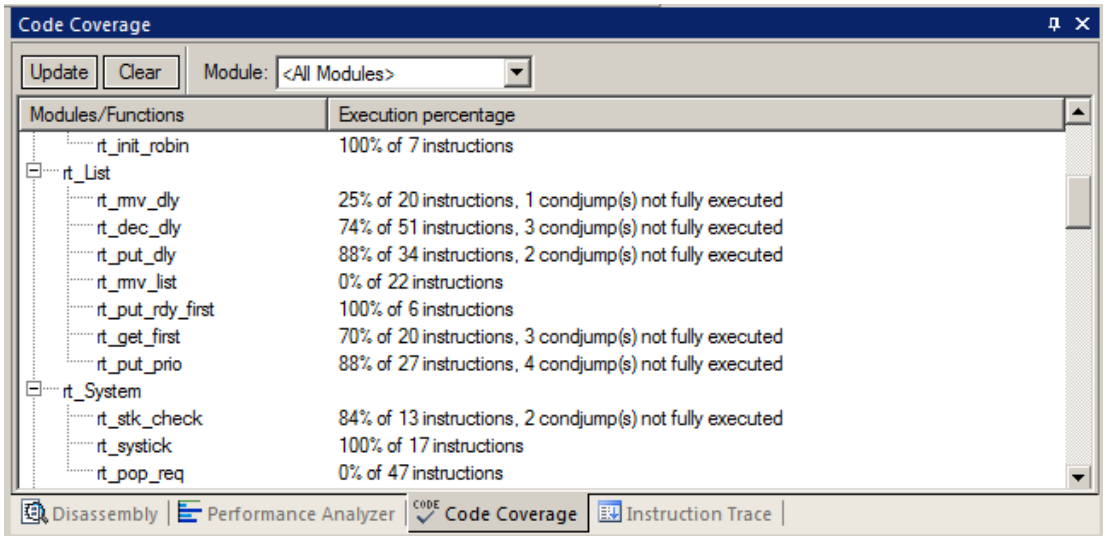
In the window above, why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

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

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

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

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



7) Saving Code Coverage:

Code Coverage information is temporarily saved during a run and is displayed in various windows as already shown.

It is possible to save this information in an ASCII file for use in other programs.

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

You can Save Code Coverage in two formats:

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

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

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

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

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

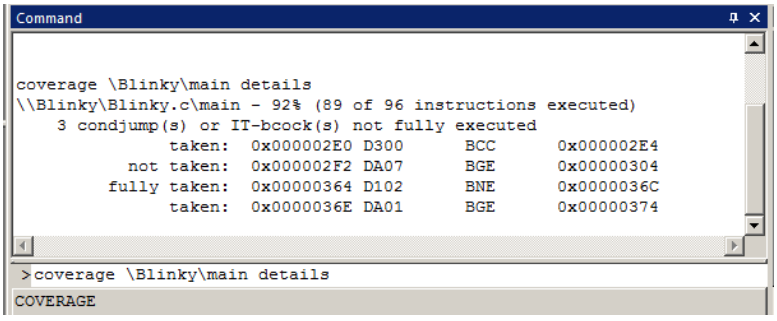
The first column above describes the execution as follows:

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

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

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

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

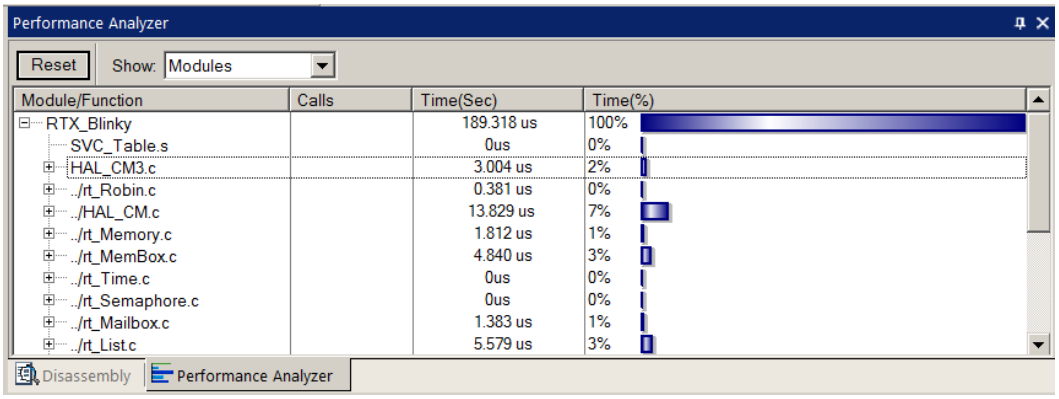


8) Performance Analysis (PA):

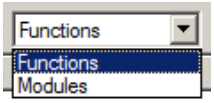
Performance Analysis tells you how much time was spent in each function. It is useful to optimize your code for speed.

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

1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and re-enter it.  This clears the PA window and resets the Kinetis processor and reruns to main(). You might have to cycle the power to the ULINK*pro* and the TWR board to get this exact window below. This will remove any artifacts left over. Close and restart µVision if necessary.
4. Do ***not*** click on RUN yet Expand some of the module names as shown below.

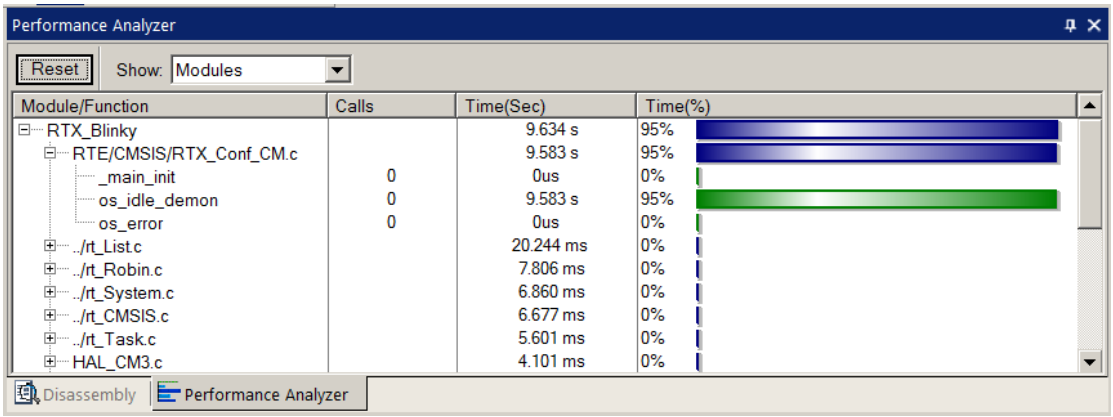


5. Shown is the number of calls and percentage of total time in this short run from RESET to the beginning of main().
6. Click on the RUN icon.  See the PA window below:
7. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the data. No code stubs are needed in your source files. Most time is spent in the RTX Delay function.
8. Select Functions from the pull down box as shown here and notice the difference.
9. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.



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

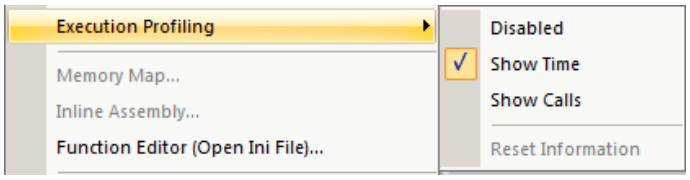
10. Click on the PA RESET icon.  Watch as new data is displayed in the PA window.
11. When you are done, STOP  and exit Debug mode  .



TIP: The Performance Analyzer uses ETM to collect its raw data. A ULINK*pro* is needed.

9) Execution Profiling:

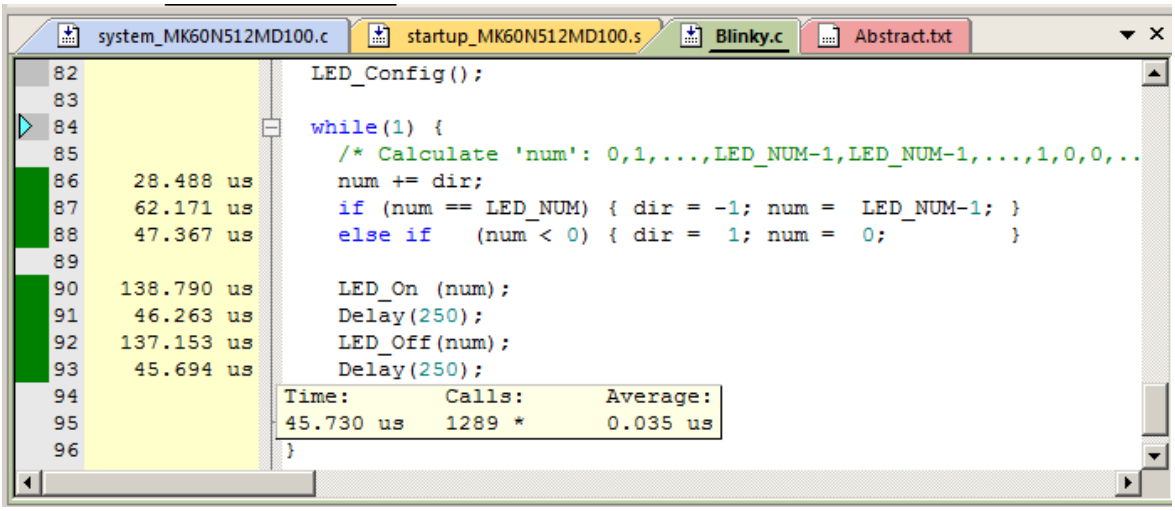
Execution profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.



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

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

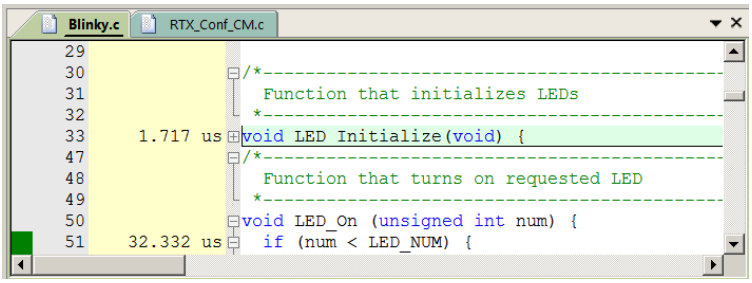
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



Outlining:

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

1) Click in the square near the while(1) loop near line 33 as shown here:
2) The C source in the function LED_Initialize is now collapsed into one line.
3) The times are added together to 1.717 usec in this case:
4) This can be useful to hide sections of code to simplify the window you are reading.
5) Click on the + to expand it.
6) Stop the program
7) Exit Debug mode .



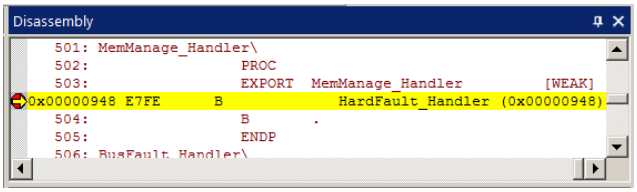
10) In-the-Weeds Example: (your addresses might not be the same as shown here)

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

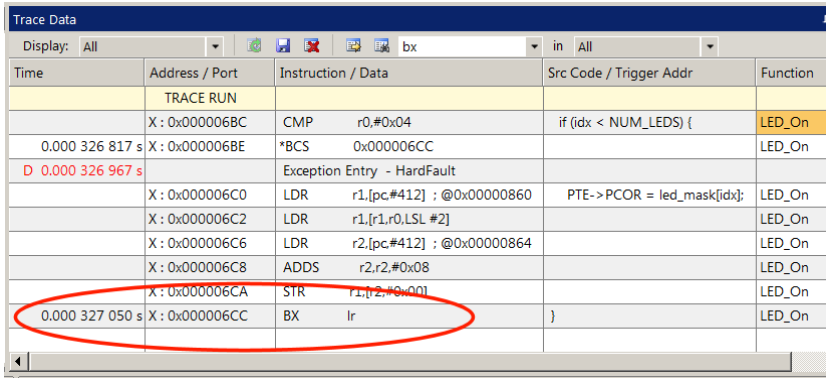
If a Bus Fault occurs in our example, the CPU will end up at 0x08BE as shown in the Disassembly window below. This is the Bus Fault handler. This is a branch to itself and will run this instruction forever. The trace buffer will save millions of the same branch instructions. This is not very useful.

The Hard Fault handler exception vector is found in the file startup_MK64F12.s. If we set a breakpoint by clicking on the Hard Fault Handler and run the program: at the next Bus Fault event the CPU will again jump to the HardFault_Handler.

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



1. Use the RTX_Blinky example from the previous exercise, enter Debug mode.
2. Locate the Hard Fault near address 0x0948 in the Disassembly window or near line 499 in startup_MK60D10.s.
3. Set a breakpoint at this point. A red circle will appear. You can do this in Edit or Debug mode in the .s file.
4. In the Command window enter: **g, LED_On** and press ENTER. This will put the PC at the start of this function. LED_On returns with a BX lr; near address 0x66C which we will use to create a Hard Fault by using lr = 0.
5. The assembly and sources in the Disassembly window do not always match up and this is caused by anomalies in ELF/DWARF specification. In general, scroll downwards in this window to provide the best match.
6. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
7. In the Register window, double-click on R14 (LR) register and set it to zero. This is guaranteed to cause a Hard Fault when the processor tries to execute an instruction at 0x2000 10E8. 0x0 contains the initial SP address.
8. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
9. In the Trace Data window you will find the LED_On function with the BX LR at the end. This is what caused the Hard Fault since you set lr = 0. When the function tried to return, the bogus value of lr caused a Hard Fault.
10. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.
11. Click on Single Step. You will now see the Hard Fault branch as shown in the bottom screen:



This example clearly shows how quickly ETM trace can help debug program flow bugs.

Exception Entry: Note at the top the Hard fault exception is listed. How can this happen before the Hard Fault happens ? This entry is part of SWV – the timestamps are different for SWV and ETM instructions so they can be out of sync. Use Display: to filter out such events.

TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 489… and select TraceHalt. When Hard Fault is reached is reached, trace collection will halt but the program will keep executing.

12. Exit Debug mode.

11) Serial Wire Viewer and ETM Trace Summary:

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

1. SWV and ITM data output on the SWO pin located on the JTAG/SWD 10 pin CoreSight debug connector. The 20 pin connector adds ETM trace.
2. ITM is a printf type viewer. ASCII characters are displayed in the Debug printf Viewer in µVision.
3. Non-intrusive Memory Reads and Writes in/out the JTAG/SWD ports (DAP).
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.
5. ETM provides a record of all instructions executed. ETM also provides Code Coverage and Performance Analysis.

These features are completely controlled through µVision via a ULINK2 or ULINK*pro*.

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

SWV Trace adds significant power to debugging efforts. Problems which may take hours, days or even weeks in big projects can often be found in a fraction of these times with a trace. Especially useful is where the bug occurs a long time before the consequences are seen or where the state of the system disappears with a change in scope(s) or RTOS thread switches.

Usually, these techniques allow finding bugs without stopping the program. These are the types of problems that can be found with a quality trace: Some of these items need ETM trace.

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes). How I did I get to this Fault vector ?
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs). ***How did I get here ?***
4) A corrupted stack.
5) Out of bounds data. Uninitialized variables and arrays.
6) Stack overflows. What causes the stack to grow bigger than it should ?
7) **Runaway programs:** your program has gone off into the weeds and you need to know what instruction caused this. ***This is probably the most important use of trace.***
8) Communication protocol and timing issues. System timing problems.
9) Trace adds significant power to debugging efforts. Tells you where the program has been.
10) Weeks or months can be replaced by minutes.
11) Especially where the bug occurs a long time before any consequences are seen.
12) Or where the state of the system disappears with a change in scope(s).
13) Plus - don't have to stop the program to test conditions. Crucial to some applications.
14) A recorded history of the program execution ***in the order it happened.*** Source and Disassembly is as it was written.
15) Trace can often find nasty problems very quickly.
16) Profile Analysis and Code Coverage is provided. Available only with ETM trace.

What kind of data can the Serial Wire Viewer display ?

1. Global variables.
2. Static variables.
3. Structures.
4. Can see Peripheral registers – just read or write to them. The same is true for memory locations.
5. Can see executed instructions. SWV only samples them. Use ETM to capture all instructions executed.
6. CPU counters. Folded instructions, extra cycles and interrupt overhead.

What Kind of Data the Serial Wire Viewer can't display…

1. Can't see local variables. (just make them global or static).
2. Can't see register operations. PC Samples records some of the instructions but not the data values.
3. SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU. SWV and ETM can only see CPU actions. If using a ULINK*pro* and RTX, DMA exceptions will display in the Event Viewer.

Part E:

1) Creating your own MDK 5 project from scratch: no RTOS:

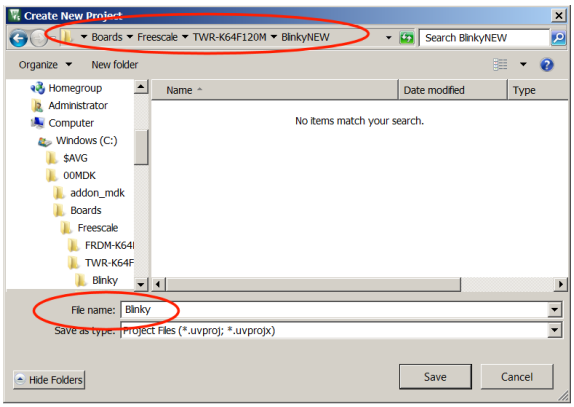
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run a bare metal (no OS) Blinky example. It will have a simple incrementing counter to monitor. 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 Software Pack for your processor:

1. Start µVision and leave it in Edit mode. Do not enter Debug mode. A project must be loaded. Any project at all.
2. **Pack Installer:** The Keil::Kinetis_K60_DFP Software Pack must be installed. This was done on page 4.

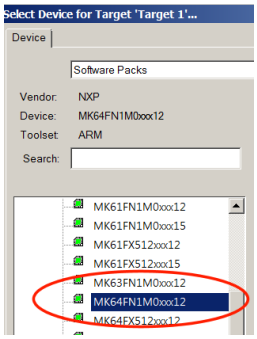
Create a new Directory and a New Project:

3. In the main µVision menu, select Project/New µVision Project… Create New Project window opens:
4. In this window, shown here, navigate to the folder C:\00MDK\Boards\Freescale\TWR-K64F120M\
5. Right click in this window and select New (or click New Folder) and create a new folder. I called it BlinkyNEW.
6. Double click on BlinkyNew to open it or highlight it and select Open.
7. In the File name: box, enter Blinky. Click on Save.
8. This creates the project Blinky.uvproj. (MDK 4 format)
9. The Select Device for Target…opens:



Select the Device you are using:

1. Expand NXP and K60. Select MK64FN1M0xx12 as shown here:



TIP: Make sure you select the deepest layer device or this will not work correctly.

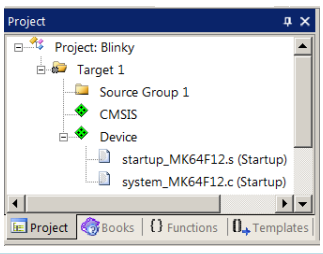
2. Click OK and the Manage Run Time Environment window shown below right opens.

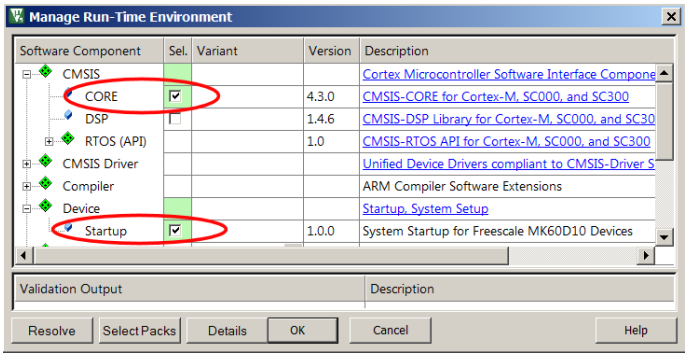
Select the CMSIS components you want:

1. Expand CMSIS and Device. Select CORE and Startup as shown below. They will be highlighted in Green indicating there are no other files needed. Click OK to close.
2. Click on File/Save All or select the Save All icon:
3. The project Blinky.uvproj will now be changed to Blinky.uvprojx. (MDK 4 ➡ MDK 5 format)
4. You now have a new project list as shown on the bottom left below: The CMSIS files you selected have been automatically entered and configured into your project for your selected processor.
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 **ULINK2 Flash** and press Enter. The Select Target name will also change.

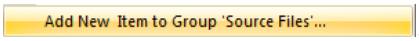
What has happened to this point:

You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files and select your debug adapter. The Software Pack has pre-configured many settings for your convenience.

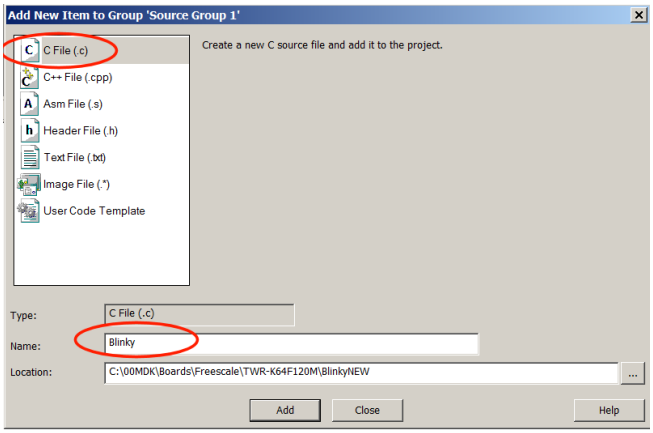




Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select 

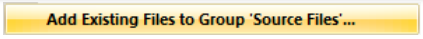
.
2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter Blinky.
5. Click Add to close this window.
6. Click on File/Save All or 
7. Expand Source Group 1 in the Project window and Blinky.c will now display. It is a blank file.



Add Some Code to Blinky.c:

1. Right click in Blinky.c and select Insert '#include file'.
2. Select MK64F12.h. This will be added to Blinky.c
3. In the nearly blank Blinky.c, add the C code below:
4. Click on File/Save All or 
5. Build the files.  There will be no errors or warnings if all was entered correctly.

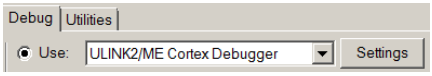
```
#include "MK60D10.h"    //Device Header
unsigned int counter = 0;
/*-----------------------------------------------
  MAIN function
 *-----------------------------------------------*/
int main (void) {

  while(1) {
     counter++;
        if (counter > 0x0F) counter = 0;
}
}         //make sure you add a CR Carriage Return or Enter after the last parentheses.
```

TIP: You could also add existing source files: 

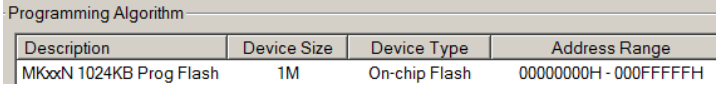
But not at this time.

Configure the Target Flash: ***Please complete these instructions carefully to prevent unusual problems…***

1. Select the Target Options icon  . Select the **Target** tab. Note the Flash and RAM addresses are already entered.
2. Select Use MicroLIB to optimize for smaller code size. An error will be generated if you cannot use this feature.
3. Select the **Linker** tab. Select Use Memory Layout from Target Dialog. The addresses in the Target tab will be used. If unselected, you can add your own custom scatter file.
4. Click on the **Debug** tab. Select the debugger you are using in the Use: box: You can use a P&E, ULINK2, ULINK*pro* or a J-Link.



5. Connect your NXP TWR to your PC USB port. See page 10 to connect adapters and to set power jumpers.
6. Select Settings: box beside Use ULINK2/ME Debugger as shown above.
7. Set SW and SWJ as shown here:  If your TWR board is connected to your PC, you should now see a valid IDCODE and Device Name in the SW Device box. If you do not, you **must** correct this before continuing.
8. Click on the **Flash Download** tab. Confirm the correct Flash algorithms are present: Shown here is the correct one for the TWR-K64F120M board:
9. Click on OK twice to return to the main menu.
10. Click on File/Save All or 

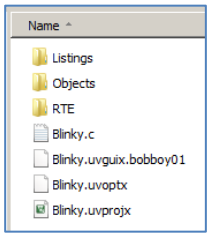


11. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

The Next Step ? First we will do a summary of what we have done so far and then …. you will run the program !

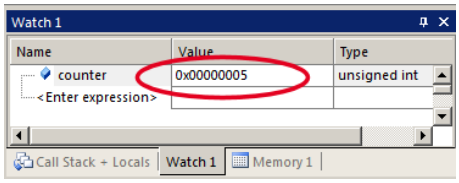
What we have so far ?

1. A project has been created in C:\00MDK\Boards\Freescale\TWR-K64F120M\BlinkyNEW\
2. The folders have been created as shown here:
3. RTE contains the CMSIS-Core startup and system files.
4. The Software Pack has pre-configured many items in this new project for your convenience.



Running Your Program:

1. Enter Debug mode by clicking on the Debug icon . The Flash will be programmed.
2. Click on the RUN icon. Note: you stop the program with the STOP icon.
3. Right click on counter in Blinky.c and select Add 'counter' to … and select Watch 1.
4. counter should be updating as shown here:
5. 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.
6. You are now able to add your own source code to create a meaningful project. You can select software components in the Manage Run-time Environment window. You can experiment with this later.



TIP: 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 some sequential values that you know must exist.

Configuring the CPU Clock:

The file system_MK64F12.c contains the CPU clock setup code. This project is running at the default of 20.97 MHz.

1. STOP the program. Exit Debug mode .
2. In Blinky.c, near line 8, just after int main(void) {, add this line:

```
8   SystemCoreClockUpdate();
```

3. Click on File/Save All or
4. Build the files. There will be no errors or warnings.
5. Enter Debug mode. The Flash will be programmed.
6. Click on the RUN icon.
7. In Watch 1, double click on <Enter expression> and enter SystemCoreClock and press Enter. You can also right-click on SystemCoreClock as found in system_MK64F12.c and enter it in the usual way.
8. Right click on SystemCoreClock in Watch1 and unselect Hexadecimal Display.
9. The CPU speed is displayed as 20.971520 MHz as shown in the global variable SystemCoreClock in Watch 1.
10. Stop the CPU. and exit Debug mode.

TIP: To change the CPU frequency, modify line 61 in system_MK64F12.c: 4 equals 120 MHz and 0 equals 20.97 MHz.

```
61  #define CLOCK_SETUP     4
```

What else can we do ?

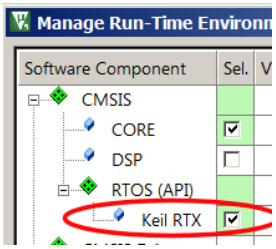
5. You can create new source files using the Add New Item window. See the top of the previous page.
6. You can add existing source files by right clicking on a Group name and selecting Add Existing Files.
7. You can easily add NXP example files to your project. You can use Kinetis Expert to help you.
8. If you use RTX or Keil Middleware, source and template files are provided in the Add New window.
9. Now, we will add RTX to your new project !

2) Adding RTX to your MDK 5 project:

Software Packs contain all the code needed to add RTX to your project. RTX is CMSIS-RTOS compliant.

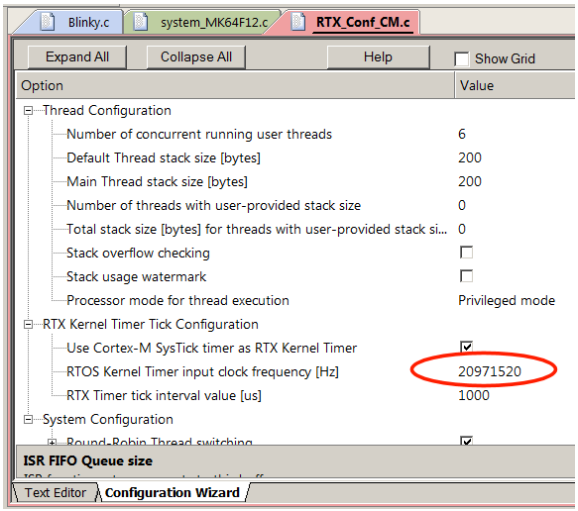
Configuring RTX is easy in MDK 5. These steps use the preceding Blinky example you constructed.

1. Using the same example from the preceding pages, Stop the program and Exit Debug mode.
2. Open the Manage Run-Time Environment window:
3. Expand all the elements as shown here:
4. Select Keil RTX as shown and click OK.
5. Appropriate RTX files will be added to your project. See the Project window.
6. In Blinky.c, on the first line, right click and select Insert '# include file'. Select "cmsis_os.h". This will be added as the first line in Blinky.c.



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 at the bottom of this window: Select Expand All.
4. The window is displayed here:
5. Select Use Cortex-M SysTick Timer as RTX Kernel Timer.
6. Set Timer clock value: to 20971520 as shown: (20.97 MHz)
7. Use the defaults for the other settings.

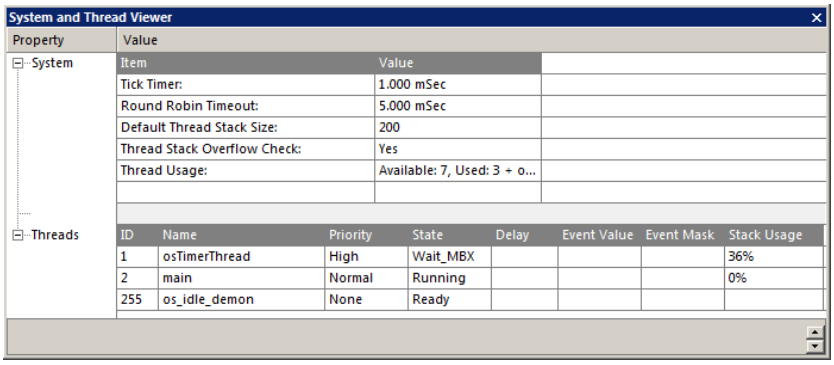


Build and Run Your RTX Program:

1. Click on File/Save All or
2. Build the files. There will be no errors or warnings.
3. Enter Debug mode: Click on the RUN icon.
4. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
5. You can see three 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.
6. Stop the program and Exit Debug mode.

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.
2. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/gsg/. It has information on implementing RTX as well as other subjects.
3. You can use the Event Viewer to examine your threads graphically if you have a ULINK2, ULINK*pro* or a J-Link. Event Viewer uses SWV and is not yet supported by P&E.



TIP: The Configuration Wizard is a scripting language as shown in the Text Editor as comments starting such as a </h> or <i>. See www.keil.com/support/docs/2735.htm for instructions on how to add this feature to your own source code.

3) Adding a Thread:

We will create and activate a thread. We will add an additional variable counter2 that will be incremented by this new thread.

1. In Blinky.c, add this line near line 5: unsigned int counter2 = 0;

```
5   unsigned int counter2 = 0;
```

Create the Thread job1:

2. Add this code before main():
   This will be the new thread named job1.

   osDelay(500) delays the program by 500 clock ticks to slow it down so we can easier see the values of counter and counter2 increment by 1.

```
7    void job1 (void  const *argument) {
8    for (;;) {
9            counter2++;
10             if (counter2 > 0x0F) counter2 = 0;
11            osDelay(500);
12   }
13  }
```

Add another osDelay to main(): (main() is a thread !)

3. Add this line just after the if statement near line 21:

```
21   osDelay(500);
```

Define and Create the Thread:

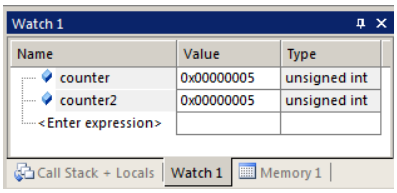
4. Add this line near line 15 just before main():

```
15   osThreadDef(job1, osPriorityNormal, 1, 0);
```

5. Create the thread job1 near line 18 just after main() and before the while(1) loop:

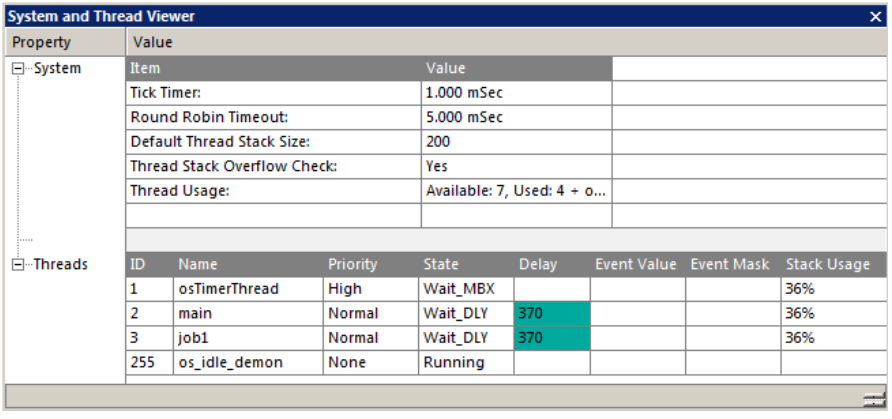
```
18   osThreadCreate(osThread(job1), NULL);
```

6. Click on File/Save All or 
7. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.
8. Enter Debug mode:  Click on the RUN icon. 
9. Right click on counter2 in Blinky.c and select Add counter2 to … and select Watch 1.
10. Both counter and counter2 will increment but slower than before:
    The two osDelay(500) function calls each slow the program down by 500 msec. This makes it easier to watch these two global variables increment.



TIP: osDelay() is a function provided by RTX and is triggered by the SysTick timer.

11. Open the System and Thread Viewer by selecting Debug/OS Support.
12. Note that job1 has now been added as a thread as shown below:
13. Note os_idle_demon is always labelled as Running. This is because the program spends most of its time here.
14. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
15. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
16. There are many attributes of RTX you can add. RTX help files are located here depending on your CMSIS version: C:/Keil_v5/ARM/Pack/ARM/CMSIS/*x.x.x*/CMSIS/Documentation/RTX/html/index.html.
17. **Remove any breakpoints you have created.**



On the next page, we will demonstrate the Event Viewer. For this you must use a Keil ULINK2, ULINK*pro* or a Segger J-Link.

RTX will be part of the new CMSIS 5 See https://github.com/ARM-software/CMSIS_5

.

4) View RTX Thread Timing: Requires a Keil ULINK2/ME, ULINK*pro* or a Segger J-Link.

The Event Viewer displays threads running in a graphical format. It is possible to make timing measurements.

The Event Viewer uses Serial Wire Viewer (SWV). The P&E debug adapter does not support SWV. In order to do the exercise on this page, you must use any Keil ULINK or a J-Link. A ULINK2 is better and a ULINK*pro* is best for SWV.

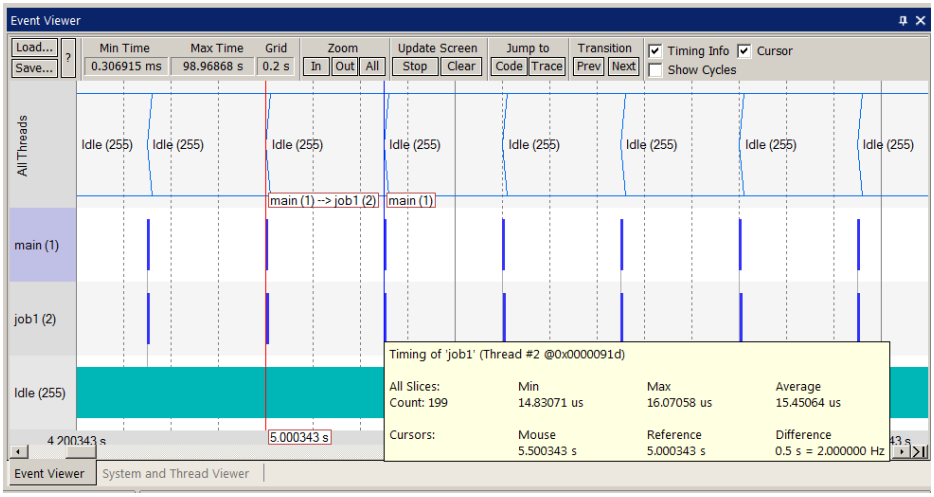
This page assumes you are running this Blinky with any ULINK or a J-Link connected to your TWR-K64F120M board.

Configure SWV:

1. Stop the program and Exit Debug mode.
2. Select the Target Options icon .
3. Click on the Debug tab. Select the debugger you are using in the Use: box: This is for ULINK2: ULINK2/ME Cortex Debugger
4. Click on Settings on the right side of the target Options window. The Cortex-M Target Driver Setup window opens.
5. Set SW as shown here: Port SW JTAG does not support SWV. If your TWR board is connected to your PC, you should now see a valid IDCODE and Device Name in the SW Device box.
   If you do not, you **must** correct this before continuing.
6. Select the Trace tab. Select Core Clock: to 20.97 (20.97 MHz). Select Trace Enable.
7. Unselect EXCTRC to minimize potential SWO overflows. This might not be needed. You can experiment later.
8. Click OK twice to close the Target configuration windows.
9. Click on File/Save All or SWV is now configured.

Run Blinky and Open Event Viewer:

1. Enter Debug mode: Click on the RUN icon.
2. The program should be running as shown in the System and Thread Viewer as shown on the previous page. Viewing this window gives a good indication RTX is configured and running properly.
3. Select Debug/OS Support and select Event Viewer.
4. This window will open and if SWV is correctly configured, the threads will be visible.
5. Use the In and/or Out buttons to select an appropriate horizontal scale.



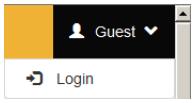
6. The space between the same thread is 500 msec or 0.5 sec. This is 500 times the tick rate as set in the RTX config file. The tick rate is 1 msec.
7. Note the program spends most of its time in Idle (os demon). This can be modified in your program.

The next section describes how to create a µVision project from NXP Kinetis Expert.

5) Using Kinetis Expert to create a MDK 5 Project:

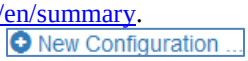
Kinetis Expert is a utility provided by NXP to simplify creating and configuring Kinetis devices. A MDK 5 .uvprojx is created that can be directly loaded into µVision, compiled and run. You can then add your own source code.

Starting and Configuring Kinetis Expert:

1. The URL for Kinetis Expert is http://kex.nxp.com
2. You will need to create an account and sign in: 

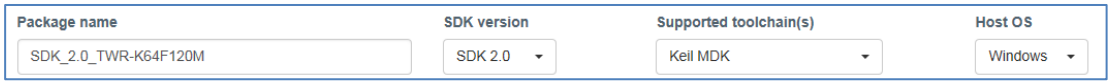
3. In the System Configuration Tool (http://kex.nxp.com/en/welcome): select Build an SDK: 

4. Select the Configurations: button: 

This is at http://kex.nxp.com/en/summary.
5. The page (http://kex.nxp.com/en/configs) will open. Select New Configuration: 

6. Select the processor or board you are using: Name this configuration and click Select configuration: 

7. In the Kinetis SDK Summary page your board or processor will be displayed.
8. Select SDK 2.0 (or the latest), Keil MDK and Windows as shown here:



9. Unselect any RTOSs selected. We do not need or want an RTOS at this time for simplicity.

Building and Downloading Your Kinetis MDK Project:

1. Select Build SDK package: You will be prompted with the progress. 

2. When the Build is completed this message will display: 

3. The Software Vault icon will indicate the number of files in it: 

4. Click the Software Vault icon and download the project to your PC. It will be a .zip file.
5. Extract this file into a suitable folder: I used C:\00MDK\Boards\Freescale\TWR-K64F120M \KEX\ It will be in a sub-folder with the name of the SDK: I simplified the folder to SDK_2.0 to shorten the filenames.
6. You can explore and see the many other examples and useful code is present.
7. You should also explore the Pins Tool in Kinetis Expert: 

Importing an Example into uVision:

1. We will use the example Bubble which uses the accelerometer to change the display of the LEDS.
2. In µVision, select Project/Open Project and navigate to:

   C:\00MDK\Boards\Freescale\TWR-K64F120M\KEX\SDK_2.0\twrk60d100m\demo_apps\bubble\

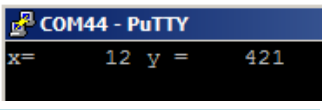
3. In the MDK folder is the project bubble.uvprojx. Highlight this and select Open. Bubble will open in µVision.

Build and Run Bubble: To use P&E: OpenSDA See page 9 for instructions to configure P&E OpenSDA.

1. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.
2. Connect the K60 board to your PC with a USB cable. The P&E debug adapter is used by default.
3. Enter Debug mode:  Click on the RUN icon. 
4. The yellow and green LEDs will vary brightness depending on how you move the board. Try moving it slowly.

Display X and Y angles using the P&E CDC COM Port:

1. Determine from Windows Device manager the COM port used by OSBDM/OSJTAG CDC Serial Port.
2. Enter this into your favourite terminal program (such as PuTTY) and set the speed to 115200 Baud.
3. The angle the board is positioned is displayed in degrees for both the X and Y axis:
4. The next page describes how to view these variables using Serial Wire Viewer: You will need a ULINK2, ULINKpro or a J-Link.



6) Displaying Serial Wire Viewer Information on a Kinetis Expert Project:

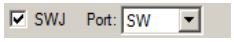
A ULINK2, ULINK*pro* or a J-Link are needed to display Serial Wire information. P&E does not currently support SWV. We will display the Xangle and Yangle variables in the Logic Analyzer window. First, we need to make them global. This tutorial uses a ULINK2. To use a ULINKpro or a J-Link you must configure µVision differently. Recall, with a ULINK*pro* you must stop the program to update the Trace Records window. ULINK2 updates this window while the program runs.

Connection to target board:

1. If the program is running: stop it  and Exit Debug mode. 
2. Connect a ULINK2 (or ULINK*pro* or a J-Link) to the TWR-K64F120M board and power both with USB cables.
3. Refer to page 10 for directions for various debug adapters and setting the two power jumpers.

Configure SWV:

1. Select the Target Options icon  . Select the Debug tab.
2. Select the debugger you are using in the Use: box: This is for ULINK2: 

3. In the Initialization File field, remove any text that is present. If using Trace Port with a ULINK*pro* a file is needed.
4. Click on Settings on the right side of the target Options window. The Cortex-M Target Driver Setup window opens.
5. Set SWJ and SW as shown here: 

JTAG does not support SWV.
6. If everything is working correctly, you should now see a valid IDCODE and Device Name in the SW Device box. **If you do not**, you **must** correct this before continuing.
7. Select the Trace tab. Select Core Clock: to 120 (120 MHz). Select Trace Enable.

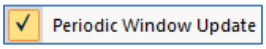
TIP: To find Core Clock frequency: Enter the global variable SystemCoreClock in a Watch window and run the program.

8. Unselect EXCTRC to minimize potential SWO overflows. This might not be needed. You can experiment later .
9. Click OK twice to close the Target configuration windows.

Add Two Global Variables to display in Logic Analyzer and Watch windows:

1. In bubble.c, there are two interesting variables xAngle and yAngle that we want to display. These are local variables and hence cannot be displayed in the Logic Analyzer or Watch windows.
2. Declare two global variables near line 68 and 69. **unsigned int XANGLE=0;** and **unsigned int YANGLE=0;**
3. At the bottom of bubble.c near line 229 add these two assignment lines:

```
229         XANGLE = xAngle;
230         YANGLE = yAngle;
```

4. Click on File/Save All or 
5. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.
6. Enter Debug mode: 
7. In the main µVision Debug menu, select View and enable Periodic Window Update. 

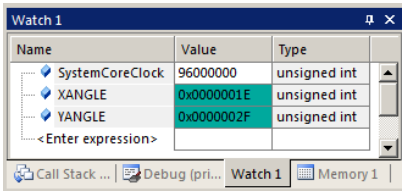
8. Click on RUN.  The program will be running with the red and blue LEDs changing when the board is moved.

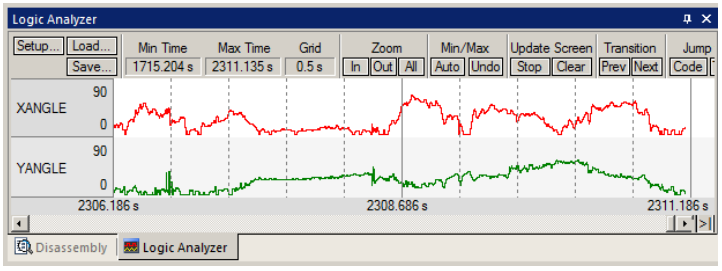
Add the XANGLE and YANGLE global variables to LA and Watch 1:

1. Right click on **XANGLE** and select Add XANGLE to… and select Logic Analyzer. Repeat for Watch 1.
2. Right click on **YANGLE** and select Add YANGLE to… and select Logic Analyzer. Repeat for Watch 1.

TIP: If an error results when adding counter to the LA, the most probable cause is SWV is not configured correctly.

3. In the LA, click on Setup and set Max: in Display Range to 90 or 0x5A for both variables. Click on Close.
4. In Zoom, set scaling for about 0.5 seconds.
5. The variables change as the board is rotated:





The End of the exercises !

1) Document Resources: See www.keil.com/NXP

Books:

1. ***NEW!*** **Getting Started with MDK 5:** Obtain this free book here: www.keil.com/gsg/
2. There is a good selection of books available on ARM: www.arm.com/support/resources/arm-books/index.php
3. µVision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab. Or select the Cortex-M processor you want in the Processor panel on the left.
5. Or search for the Cortex-M processor you want on www.arm.com.
6. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
7. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
8. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
9. MOOC: Massive Open Online Class: University of Texas: http://users.ece.utexas.edu/~valvano/

Application Notes:

1. ***NEW!*** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ 4 www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with µVision https://launchpad.net/gcc-arm-embedded
8. RTX CMSIS-RTOS Download https://github.com/ARM-software/CMSIS_5
9. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
10. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
11. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
12. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
13. FlexMemory configuration using MDK www.keil.com/appnotes/files/apnt220.pdf
14. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
15. ***NEW!*** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
16. ***NEW!*** ARMv8-M Architecture Technical Overview https://community.arm.com/docs/DOC-10896

Useful ARM Websites:

1. ***NEW!*** CMSIS Standards: https://github.com/ARM-software/CMSIS_5 and www.keil.com/cmsis/
2. ARM and Keil Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content
3. ARM University Program**:** www.arm.com/university. Email: university@arm.com
4. mbed™: http://mbed.org

Sales In Americas: sales.us@keil.com or 800-348-8051. **Europe/Asia:** sales.intl@keil.com +49 89/456040-20
Keil Distributors: See www.keil.com/distis/ **DS-5 Direct Sales Worldwide:** orders@arm.com
Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.
For comments, additions or corrections please email bob.boys@arm.com

2) Keil Products and contact information: See **www.keil.com/NXP**

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

- **MDK-Lite™** (Evaluation version) 32K Code and Data Limit - $0
- *New* **MDK-ARM-CM™** For all Cortex-M series processors – unlimited code limit
- *New* **MDK-Plus™** MiddleWare Level 1. ARM7™, ARM9™, Cortex-M, SecureCore®.
- *New* **MDK-Professional™** MiddleWare Level 2. For details: www.keil.com/mdk5/version520.

Keil Middleware includes Network, USB, Graphics and File System. www.keil.com/mdk5/middleware/

USB-JTAG/SWD Debug Adapter (for Flash programming too)

- ULINK2 - *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM: is equivalent to a ULINK2.
- ULINK*pro* - Cortex-M*x* SWV & ETM trace
- ULINK*pro* D - Cortex-M*x* SWV no ETM trace ULINK*pro* also works with ARM DS-5

You can use OS-JTAG on the Kinetis Tower board. For Serial Wire Viewer (SWV), a ULINK2, ULINK-ME or a J-Link is needed. For ETM support, a ULINK*pro* is needed. OS-JTAG or OpenSDA do not support either SWV or ETM debug technology.

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

For the ARM University program: go to www.arm.com/university Email: university@arm.com

All software products include Technical Support and Updates for 1 year. This can easily be renewed.

Keil RTX™ Real Time Operating System

- RTX is provided free as part of Keil MDK. It is the full version of RTX – it is not restricted or crippled.
- No royalties are required and is very easy to use. It has a BSD license.
- RTX source code is included with all versions of MDK.
- Kernel Awareness visibility windows are integral to µVision.

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor. For NXP support: www.keil.com/NXP

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

For more information:

Sales In Americas: sales.us@keil.com or 800-348-8051. **Europe/Asia:** sales.intl@keil.com +49 89/456040-20

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

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™ Intelligent Processors by ARM®

ARM® KEIL® Microcontroller Tools