

The latest version of this document is here: [www.keil.com/appnotes/docs/apnt\\_295.asp](http://www.keil.com/appnotes/docs/apnt_295.asp)

For the lab using the LPCXpresso 54114: [www.keil.com/appnotes/docs/apnt\\_294.asp](http://www.keil.com/appnotes/docs/apnt_294.asp)

## Introduction:

The purpose of this lab is to introduce you to the NXP Cortex™-M4/M0+ processor family using the ARM Keil MDK toolkit featuring the µVision® IDE. At the end of this tutorial, you will work confidently with NXP processors and Keil MDK™.

Keil MDK has an evaluation version that limits code and data size to 32 Kbytes. Most Keil examples are under this 32K. The addition of a license number will turn it into a full, unrestricted version. Contact Keil sales for a temporary full version license to evaluate MDK with larger programs or Keil Middleware. MDK includes a full version of Keil RTX™ RTOS with all source code included. See [www.keil.com/NXP](http://www.keil.com/NXP) for information concerning Keil support of NXP products.

## Why Use Keil MDK ?

MDK provides these features particularly suited for NXP Cortex-M0, M0+, M3, M4, M7, M23 and Cortex-M33 processor users:

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is turn-key "out-of-the-box".
2. ARM Compiler 5 and Compiler 6 (LLVM) are included. GCC is supported. <https://launchpad.net/gcc-arm-embedded>
3. Dynamic Syntax checking on C/C++ source lines.
4. **Keil Middleware:** Network, USB, Flash File and Graphics.
5. **NEW! Event Recorder for Keil Middleware.** See Page 24.
6. MISRA C/C++ support using PC-Lint. [www.gimpel.com](http://www.gimpel.com)
7. **Compiler Safety Certification Kit:** [www.keil.com/safety/](http://www.keil.com/safety/)
8. TÜV certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
9. CMSIS-RTOS RTX is included. RTX has a BSD or Apache 2.0 license with source code. [www.keil.com/RTX](http://www.keil.com/RTX).
10. CoreSight™ Serial Wire Viewer and ETM trace capability on appropriately equipped NXP processors.
11. Choice of adapters: LPC-Link2 (CMSIS-DAP), ULINK™2, ULINK-ME, ULINKpro and Segger J-Link.
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.

## Significant features shown in this document:

1. DualCore debugging control for the Cortex-M0+ and the Cortex-M4 cores on the NXP LPC5411x processors.
2. Serial Wire Viewer (SWV) with the ULINK2, ULINK-ME, ULINKpro and J-Link.
3. Real-time Read and Write memory accesses for Watch, Memory, Peripheral and RTX Tasks windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
4. RTX and its two kernel awareness windows. These windows update while the program is running.
5. A DSP example using ARM CMSIS-DSP libraries.
6. Create your own projects with and without RTX. Create a project using NXP MCUXpresso.

**Serial Wire Viewer (SWV):** Use a ULINK2, ULINKpro or a J-Link for this debugging feature.

**Serial Wire Viewer (SWV)** displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and a timestamp. SWV is non-intrusive and no code stubs are needed in your source code.

**Embedded Trace Macrocell (ETM):** Use a ULINKpro for this debugging feature.

The LPC5411x family does not implement ETM instruction trace. Many other NXP family members do have ETM. ETM provides program flow debugging, Performance Analysis and Code Coverage. Coverage can be exported in .gcov format.



<b>General Information, Hardware and Software:</b>	<b>3</b>
1. NXP Evaluation Boards, 5 Steps, Keil Software Installation:	3
2. Software Pack Installation Process:	4
3. Copy the Examples to your PC using Pack Installer:	5
4. Software Pack Version Selection and Manage Run-Time Environment:	6
5. CoreSight Definitions:	8
6. Debug Adapter Types:	9
7. Debug Adapter Configuration:	10
<b>RTX_Blinky Example Program using LPC54114:</b>	<b>11</b>
1. Open, Compile and RUN Blinky:	11
2. Hardware Breakpoints:	12
3. Call Stack + Locals Window:	13
4. Watch and Memory Windows and how to use them:	14
5. System Viewer (Peripherals):	15
6. Watchpoints: Conditional Breakpoints:	16
7. Configuring Serial Wire Viewer (SWV) with ULINK2, ULINK <i>pro</i> or J-Link:	17
8. Using the Logic Analyzer (LA) with ULINK2, ULINK <i>pro</i> or J-Link:	18
9. RTX Kernel Awareness Windows:	19
10. Debug (printf) Viewer:	20
<b>Dual Core Example:</b>	<b>21</b>
1. Notes on the DualCore Example:	21
2. Loading and Configuring $\mu$ Vision for the Dual CPU LPC4300 series:	21
3. Compiling the Programs:	21
4. Programming the Flash as a Test:	21
5. Programming the Flash and Running the two Blinky Programs:	22
6. Setting Hardware Breakpoints:	22
7. Watch window with DualCore:	23
<b>USB Mass Storage Example:</b>	<b>24</b>
1. Installing and Running the USB Mass Storage Example:	24
2. Selecting Keil Middleware:	24
3. <b>NEW!</b> Event Recorder for Keil Middleware:	25
4. Using Serial Wire Viewer:	26
5. Event Viewer with RTX:	27
6. Using Event Viewer to view Exceptions and Interrupts with a ULINK <i>pro</i> :	27
<b>DSP SINE example using ARM CMSIS-DSP Libraries:</b>	<b>28</b>
1. Running the DSP SINE example:	28
<b>Creating your Own MDK 5 Project:</b>	<b>29</b>
1. Create a New Project:	29
2. Create Source Files and Configure Flash Programming:	30
3. Running Your Program and Determining CPU Frequency:	31
4. Adding RTX RTOS:	32
5. Adding a Thread:	33
6. View RTX Thread Timing:	34
7. Using MCUXpresso:	35
<b>General Information:</b>	<b>36</b>
1. Serial Wire Viewer Summary:	36
2. Document Resources:	37
3. Keil Products and Contact Information:	38

## 1) NXP ARM Cortex Processor Evaluation Boards:

This tutorial is for the Keil MCB54110 evaluation board. Keil supports and makes boards with other NXP processors. See [www.keil.com/NXP](http://www.keil.com/NXP) for more information.

All NXP processors implement SWV with the exception of Cortex-M0 and M0+, which have neither SWV nor ETM. The Cortex-M0+ (LPC800, LPC81x, LPC82x and LPC83x) has a MTB Micro Trace Buffer. All have non-intrusive read/write to memory locations (for Watch, Memory and Peripheral windows), hardware breakpoints and Watchpoints. Many NXP Cortex-M3, M4 and M7 have ETM instruction trace which also provides Code Coverage and Performance Analysis.

Keil produces the MCB54110 full feature board shown below. See [www.keil.com/NXP](http://www.keil.com/NXP) for more information.

### Keil MCB54110 Evaluation Board:

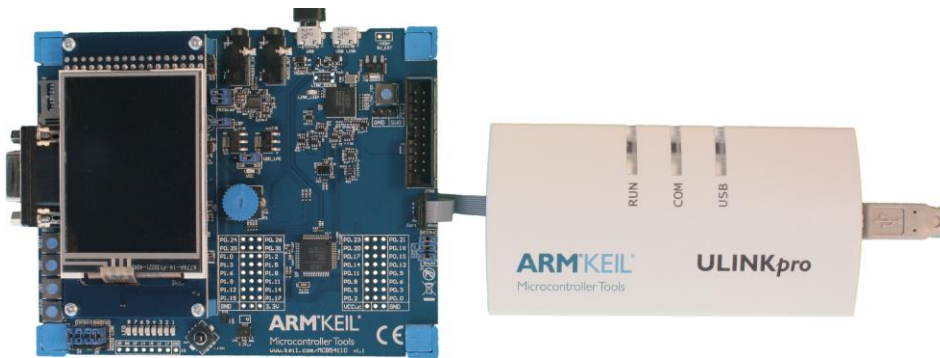
Contact Keil sales or your local distributor for pricing.

For more information:

[www.keil.com/boards2/keil/mcb54110](http://www.keil.com/boards2/keil/mcb54110) and [www.keil.com/mcb54110/mcb54110v1-1-schematics.pdf](http://www.keil.com/mcb54110/mcb54110v1-1-schematics.pdf)

The Software Pack including USB, File System, RTX and emWin GUI examples is available for download.

This board contains an on-board LPC-Link2. It can be used in either CMSIS-DAP mode or J-Link mode. It comes preprogrammed with the Keil version of CMSIS-DAP. The NXP LPCScript contains a slightly different version.




### Summary: Five Easy Steps to Get Connected and Configured:

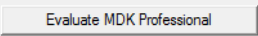
1. Obtain and install Keil MDK Core (evaluation version) on your PC. Use the default directory C:\Keil\_v5\.
2. Download the Software Pack for LPC54000\_DFP and the examples for the MCB54110 board.
3. Configure  $\mu$ Vision to use a ULINK2, ULINKpro, LPC-Link2 or a J-Link to communicate with the SWD port.
4. If desired, configure the Serial Wire Viewer with a ULINK2, ULINKpro or a J-Link as appropriate. LPC-Link2 in CMSIS-DAP mode does not currently support SWV. LPC-Link2 in J-Link mode does. A ULINK-ME is electrically equivalent to a ULINK2.
5. If used, connect your debug adapter to the Keil board. Power both of these appropriately with USB cables.

These steps are detailed in the next few pages.

### Keil MDK Core Software Download and Installation: This tutorial used MDK 5.23.

1. Download MDK Core from the Keil website. [www.keil.com/mdk5/install](http://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.
4. If you install MDK or the examples into different folders, you will have to adjust for the folder location differences.

### Licensing:



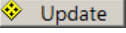
1. You can use the evaluation version (MDK-Lite) for this lab. No license is needed. However, the USB example requires a MDK Professional license.
2. To try the USB example and other Middleware examples, you can obtain a one-time free 7 day MDK Pro license in File/License Management. If you are eligible,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

## 2) Software Pack Installation Process:

A Software Pack is a zip file with a .pack file extension. It can contain header files, Flash programming algorithms, examples, documentation and more. The contents of a Pack are described by a .pdsc file contained inside the Pack. A Pack is installed using the  $\mu$ Vision Pack Installer utility or installed manually.

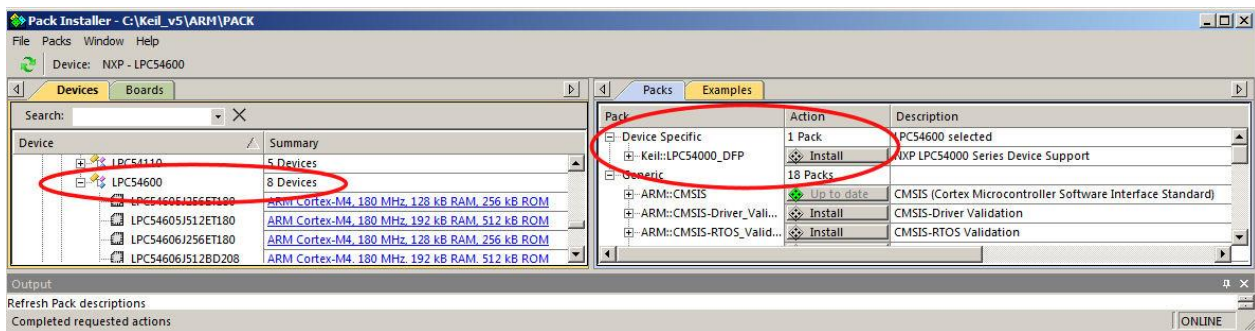
### 1) Start $\mu$ Vision and open Pack Installer:


When the first MDK install is complete and if you are connected to the Internet,  $\mu$ Vision and Software Packs will automatically start. Otherwise, follow Steps 1 and 2 below.

1. Connect your computer to the Internet. This is normally needed to download the Software Packs unless you have a standalone Pack. In this case, import it with Pack Installer or double-click on it to manually install it.
2. Start  $\mu$ Vision by clicking on its desktop icon.  The Pack descriptions will download on the initial  $\mu$ Vision 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. 
5. The window below opens up: Select the Devices tab. Scroll down and expand NXP. Select LPC54000 as shown below. You could select a specific NXP processor but in this case one Pack supports all of them at this time.
6. Alternatively, you can click on the Boards tab and select MCB54110 to filter the Packs or Examples tabs.

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

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 are 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 NXP LPC54000 Device Family Pack (Keil::LPC54000\_DFP):

1. Click on the Packs tab.
2. Select the Install icon  beside **Keil::LPC54000\_DFP** as shown above.
3. The latest Pack will download and install to C:\Keil\_v5\ARM\PACK\NXP\ by default. This download can take two to four minutes depending on your Internet connection speed.
4. The Pack's status will then be indicated by the “Up to date” icon: 
5. Leave Pack Installer open to copy the 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  $\mu$ Vision. 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 and an SDK. This is a way to distribute confidential material.

For CMSIS-Pack documentation: [www.keil.com/pack/doc/CMSIS/Pack/html/](http://www.keil.com/pack/doc/CMSIS/Pack/html/)

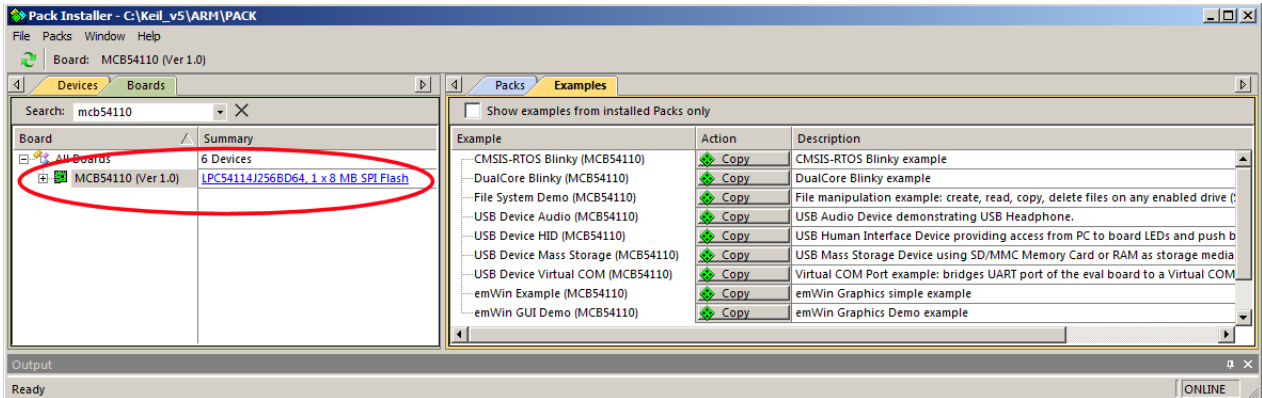
For complete CMSIS documentation: [www.keil.com/CMSIS/](http://www.keil.com/CMSIS/)

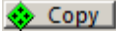


For CMSIS 5 on GitHub: [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)

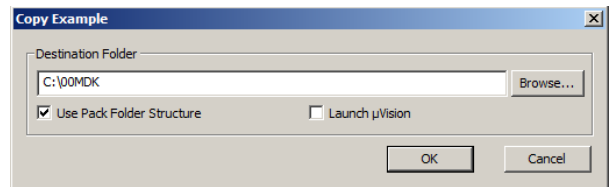
### 3) Copy the Examples to your PC using Pack Installer:

#### 1) Copy RTX\_Blinky, CMSIS-RTOS DualCore Blinky, USB Device HID and USB Mass Storage:

1. Select the Boards tab. In the Search box enter MCB54110. Highlight MCB54110 as shown below on the left:
2. Select the Examples tab: Note there are nine examples for the MCB54110. We will use four of them.




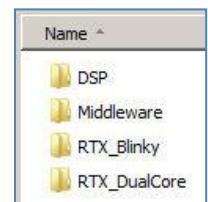
3. Beside CMSIS-RTOS Blinky as shown above: Select Copy: 
4. The Copy Example window below opens up: Select Use Pack Folder Structure. Unselect Launch  $\mu$  Vision:
5. Type in C:\00MDK as shown here: 
6. Click OK to copy this example file into C:\00MDK\Boards\Keil\MCB54110\
7. Pack Installer creates the appropriate subfolders.
8. Repeat for CMSIS-RTOS DualCore Blinky, USB Device HID and USB Mass Storage. You can copy all the example files if you prefer. You need an MDK Pro license to compile the Network, USB and emWin graphics examples.
9. Close Pack Installer. Open it any time by clicking: 
10. If a dialog box 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](http://www.keil.com):

1. Obtain the DSP example zip file from [www.keil.com/appnotes/docs/apnt\\_295.asp](http://www.keil.com/appnotes/docs/apnt_295.asp).
2. This folder has now been created: C:\00MDK\Boards\Keil\MCB54110\
3. Extract the DSP example zip into this folder. You will now have these folders present: 



**Super TIP:**  $\mu$  Vision icon meanings are found here: [www.keil.com/support/man/docs/uv4/uv4\\_ca\\_filegrp\\_att.htm](http://www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm)

**At this point, you have everything loaded and installed to run some examples and start developing your own  $\mu$  Vision projects.**



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


These three sections are provided only for reference. These three utilities are useful for managing and using Software Packs.

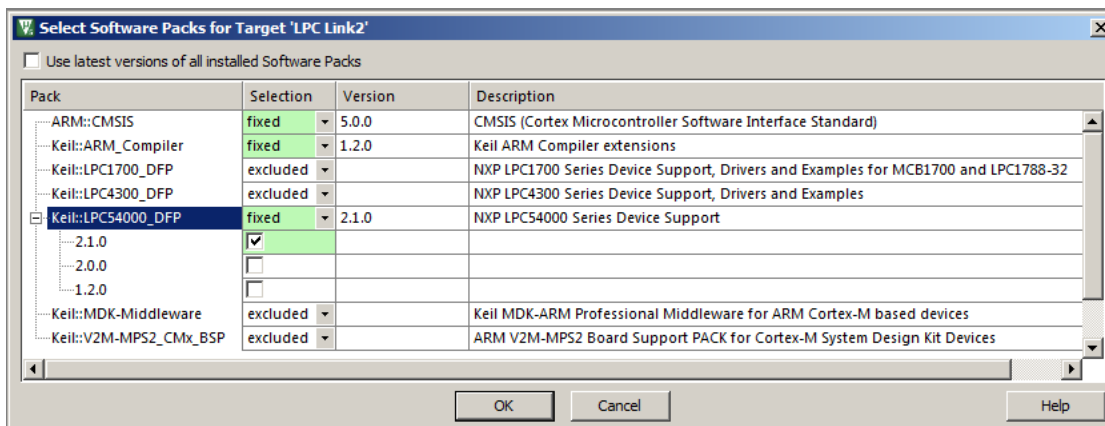
**This section contains three parts on this page and the next one:**

- A) Select Software Pack Version:
- B) Manage Run-Time Environment:
- C) Updating Source Files:

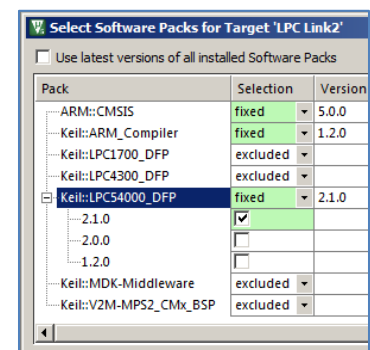
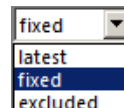
### A) Select Software Pack Version: This section is provided for reference:

This  $\mu$ Vision 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  $\mu$ Vision running and any project open for the following exercises:

1. Select Project/Open Project and navigate to C:\00MDK\Boards\Keil\MCB54110\RTX\_Blinky.
2. Open Blinky.uvprojx. Blinky will load into  $\mu$ Vision.
3. Open Select Software Packs by clicking on its icon: 
4. The window below opens: Note **Use latest versions ...** is selected.
5. Unselect this setting and the window changes as shown similar to the one below right:




6. Expand the header Keil::LPC54000\_DFP as shown here:
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 a different version, you would select fixed and then select the check box opposite the version you wanted to use in your project. These settings are stored with your project. This is a good way to freeze software components.

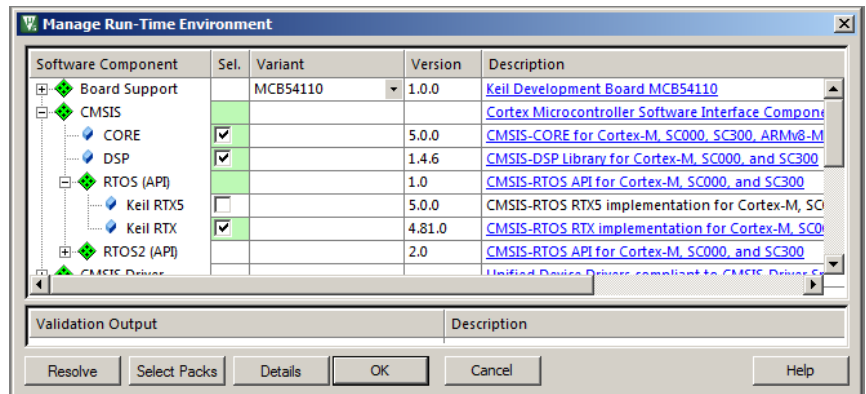


10. Re-select Use latest versions... **Do not make any changes at this time.**
11. Click OK or Cancel to close this window.

## B) Manage Run-Time Environment: This section is provided for reference:

The Manage Run-Time Environment utility selects components from various Software Packs and inserts them into your Project list. From here, you can access these files in the usual manner.

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 the appropriate files listed in the  $\mu$  Vision Project window.
4. **Do not make any changes.**  
Click Cancel to close this window.



### Select column colour meanings:

**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

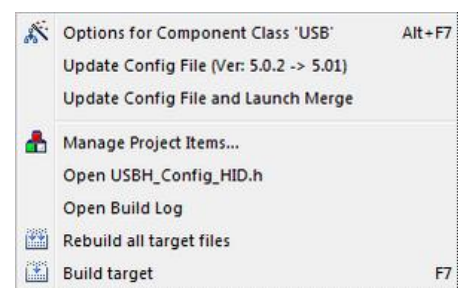
If you update a Software Pack, some of these files might need to be updated from the new Pack. These files will show new icons as shown below and described here: [www.keil.com/support/man/docs/uv4/uv4\\_ca\\_updswwmpfiles.htm](http://www.keil.com/support/man/docs/uv4/uv4_ca_updswwmpfiles.htm)

**Updating Source Files:** **Note:** Any changes you made to a file will be lost when it is replaced with the new one.

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. Remember, 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](http://www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm)

Icons for Software Components (SWC). Refer to Update Software Component Files for SWC containing modifications.

-  SWC is available for the selected microcontroller.
-  SWC is not available for the selected microcontroller. The SWC is part of another project target using another microcontroller.
-  SWC has been selected previously, but the Software Pack containing this SWC has been uninstalled.
-  Contains files with fully backward compatible corrections.
-  Contains files with fully backward compatible extensions or new features.
-  Contains files with incompatible modifications requiring modifications in the application code.
-  File with fully backward compatible corrections. A file update is not required.
-  File with fully backward compatible extensions or new features. A file update is recommended.
-  File with incompatible modifications. A file update is required.




## 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  $\mu$ Vision 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.  $\mu$ Vision 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.  $\mu$ Vision 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  $\mu$ Vision, 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.  $\mu$ Vision 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 user 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  $\mu$ Vision editor.
6. It is a good idea to make the file read-only when you are finished modifications.

**Super TIP:**  $\mu$ Vision icon meanings are found here: [www.keil.com/support/man/docs/uv4/uv4\\_ca\\_filegrp\\_att.htm](http://www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm)



## 6) Debug Adapter Types:

### A) LPC-Link2 (This has both CMSIS-DAP and J-Link Lite modes):

A small processor located on the target board acts as the debug adapter. On the MCB45110 board, this extra processor is IC1. Connect a USB cable to connector USB LINK to use LPC-LINK2. CMSIS-DAP provides run control debugging, Flash and RAM programming and Watchpoints. Hardware breakpoints can be set/unset while the program runs. Reads and writes in the Watch, Memory and Peripheral windows are updated in real-time as well as the RTX System kernel awareness viewer. At this time, the LPC-Link2 version of CMSIS-DAP does not support Serial Wire Viewer (SWV). LPC-Link2 also has a J-Link Lite mode. Both can be installed with LPCScript. The MCB45110 is preprogrammed with a Keil version of CMSIS-DAP. The NXP version using LPCScript will also work.

**You must unselect Verify Code Download.** See instruction 6 on page 10. This only needs to be done with the LPC-Link2.

You are able to incorporate a CMSIS-DAP design on your own custom target boards. For documents and software go to [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5) Click on the CMSIS folder and then on DAP/Firmware.

This document was prepared using MDK 5.23. CMSIS-DAP, ULINK2 and ULINKpro work well with dual core operations.

### To Install/Update the LPC-Link2 firmware on the Keil MCB54110 board: (Keil has its own version)

1. Download and Install NXP LPCScript on your PC: [www.nxp.com/lpcscript](http://www.nxp.com/lpcscript)
2. Open the LPCscript folder under Windows Start/All Programs/LPCScript.
3. Run the “Program LPC-Link2 with CMSIS-DAP” utility **OR** “Program LPC-Link2 with Segger J-Link”.
4. With the board not powered, press and hold LINK\_ISP button, apply power to USB connector USB LINK.
5. Release LINK\_ISP when the USB dual tone sounds and follow the instructions provided in the terminal window.
6. When finished, unplug/re-plug the board to RESET the CMSIS-DAP processor. Connect using  $\mu$ Vision.

**Trouble ?** Try a different USB port on your PC and/or cycle the power to the board and try programming again.

### B) Keil ULINK2:


This is a hardware JTAG/SWD debugger. It connects to various connectors found on boards populated with ARM processors. With NXP Cortex-M3, M4 and M7 processors, ULINK2 supports Serial Wire Viewer (SWV). ULINK-ME is equivalent to a ULINK2.

### C) Keil ULINKpro:

ULINKpro is Keil’s most advanced debug adapter. ULINKpro provides Serial Wire Viewer (SWV) and ETM Instruction Trace support for NXP Cortex-M3, Cortex-M4 and Cortex-M7 processors. Code Coverage, Performance Analysis and Execution Profiling are then provided using ETM. ULINKpro programs the Flash memories very fast. ARM DS-MDK can use a ULINKpro or a ULINKpro D to debug NXP i.MX Cortex-A processors.

### D) Segger J-Link:

$\mu$ Vision supports J-link and J-Link Ultra (black case only) Version 6.0 or later.

1. Select the Target Options icon .
2. Click on the Debug tab. Select J-Link/J-Trace in the Use: box:
3. For SWV operation: Enter JLINK\_SWO.ini in the Initialization File: box. Copy this file from the DSP example.



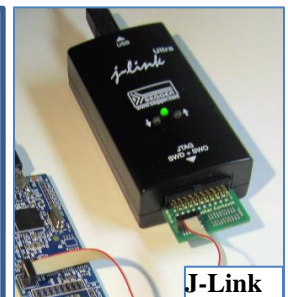
### Debug Adapter USB Connections: **Important !**

#### LPC-Link2:

1. Use USB Link USB connector.

#### ULINK2, ULINKpro or a J-Link:

2. Use USB connector to supply board power.
3. Connect a Debug Adapter to the 10 pin Cortex Debug or the JTAG 20 pin legacy connector.
4. Do **not** use USB Link as a conflict will arise between external and internal debug hardware.



## 7) Debug Adapter Configuration:

Once you know which debug adapter you will be using, you can quickly select and test it in  $\mu$ Vision.




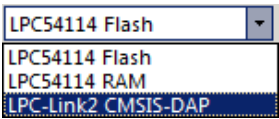

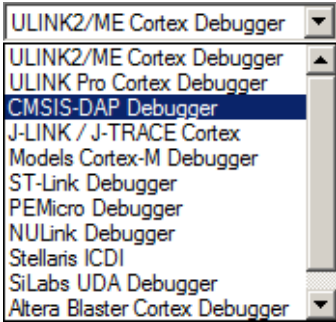
It is easy to configure  $\mu$ Vision for a variety of supported Debug Adapters.

### Prerequisites:

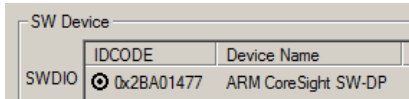
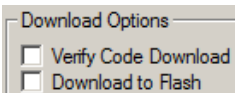

$\mu$ Vision must be running and in Edit mode (not Debug mode). Your project must be loaded.

We will use the Blinky example and add LPC-Link2 CMSIS-DAP support. You can choose your own debug adapter by using these instructions.


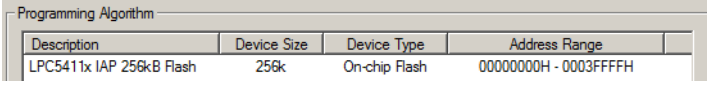

### Create a new Target Selection:

1. Select a Target Option LPC54114 Flash 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 **LPC-Link2 CMSIS-DAP** and press Enter. Click OK to close this window.
5. In the Target Selector menu, select the **LPC-Link2 CMSIS-DAP** selection you just created: 
6. Select Options for Target  or ALT-F7. Click on the Debug tab.
7. Select CMSIS-DAP Debugger... as shown here: 
8. If you are using a different debug adapter, choose it here now.

### Test the Debug Adapter:

1. Connect your debug adapter as described on the previous page.
2. Power the board appropriately.
3. In this case, for CMSIS-DAP, connect USB to the USB Link connector.
4. Click on Settings: Confirm SW is selected in the Port: box and not JTAG.
5. A valid SW Device **must** be displayed as shown here. 
6. **Trouble ?** If nothing or an error is displayed, check the LPCXpresso firmware programming and USB connections. See the previous page for instructions. This **must** be solved before you can continue.
7. If using LPC-Link2, unselect Verify Code Download as shown here: 
8. If this selected, an error may occur during the Flash programming process.
9. When completed, click OK twice to close the Options for Target windows.
10. Select File/Save All or click: 

**Verify the Flash Program Algorithm: This step is optional. It is useful if Flash programming fails.**


1. Select Options for Target  or ALT-F7. 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. This is the correct algorithm for the LPC5411x: It might also state 512 kB. 
5. If it is not visible, select Add to add it.
6. Click OK twice to return to the main  $\mu$ Vision window.
7. Select File/Save All or click .

*The new Debug Adapter is now ready to use.*


## 1) *Blinky* example program using MCB54110:

Now we will run the Blinky example on the Keil MDK development system using the MCB54110 evaluation board.

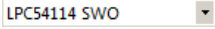
### Open the RTX\_Blinky example program:

1. Start  $\mu$ Vision by clicking on its desktop icon. 
2. Select Project/Open Project from the main menu.
3. Open the file C:\00MDK\Boards\Keil\MCB54110\Blinky\Blink.uvprojx.
4. This project uses either a LPC-Link2 in CMSIS-DAP mode or a Keil ULINK2 by default. You can add other debug adapters. See page 9 for instructions.


### A) If using the on-board LPC-Link2 in CMSIS-DAP mode:

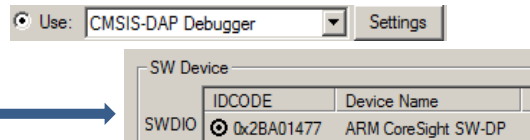
1. Select LPC54114 Flash in the Target Selection box: This is configured for LPC-LINK2 in CMSIS-DAP mode.  

2. The MCB54110 board is preprogrammed with CMSIS-DAP.
3. Connect your PC USB cable to the USB Link connector. You will hear the standard USB dual tone sound.
4. Go to Confirm Target processor Connection below:

### B) If using an external Keil ULINK2 or ULINK-ME:

1. Connect a ULINK2 or ULINK-ME to either the 20 pin JTAG legacy or 10 pin Cortex Debug connector.
2. Select LPC54114 SWO in the Target Selection box: This is configured for a Keil ULINK2. 
3. Power the MCB board with a USB cable to the USB connector. Do not use USB Link.

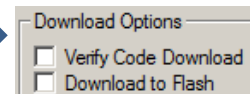
### Confirm Target Processor Connection:

1. Select Target Options  or ALT-F7. Select the Debug tab:
2. Select Settings on the right where your debug adapter is displayed:
3. Confirm SW is selected in the Port: box and not JTAG.
4. A valid SW Device *must* be displayed as shown here.







**Trouble ?** If nothing or an error is displayed, check the MCB54110 firmware programming and USB connections. See the previous page for instructions. This *must* be solved before you can continue.

5. Unselect Verify Code Download. This only needs to be done with the LPC-Link2: If this selected, an error will occur at the end of the Flash programming process.
6. When completed, click OK twice to close the Options for Target windows.



7. Select File/Save All or click: 


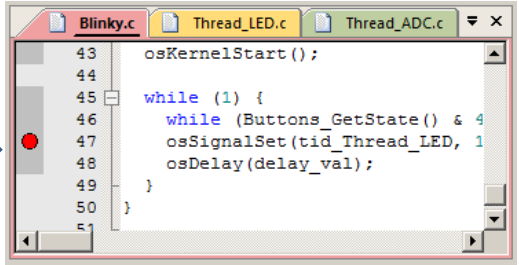

### Compile, Load and Run the Project:

1. Compile the source files by clicking on the Rebuild icon.  Progress is indicated in the Build Output window.
2. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears. Flash programming progress is indicated in bottom left corner.
3. Click on the RUN icon  to start the Blinky program. If the LEDs blink, *Blinky is now running !*
4. Stop the program with the STOP icon.  The program will probably stop in the os\_idle\_demon function.

***The eight red LEDs will blink in sequence ! Pot P1 will vary the step rate.***  
**Now you know how to compile a program, load it into the LPC54114 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:


1. Click on the RUN icon .
2. Bring Blinky.c in focus by clicking on its tab. If it is not visible, double-click on it in the Project window.
3. There are 3 threads in this example plus the idle\_demon. main() in Blinky.c is one of them. The other two are Thread\_LED and Thread\_ADC found in Thread\_LED.c and Thread\_ADC.c.
4. In Blinky.c, scroll down to inside the while (1) loop in Thread 1 found starting near line 46 as shown here: 
5. Note on the left of the line numbers are darker grey blocks. This indicates there is assembly code present and you can set a hardware breakpoint on these lines. You can also see these blocks in the Disassembly window.
6. While the program is still running, click to the left of a suitable line in the while loop and a red circle will be created. This is a hardware breakpoint. Cortex-M3, M4, M7 have six and Cortex-M0+ has two hardware breakpoints you can use. µVision will warn you if you exceed this limit.
7. The program will soon stop at the line where you set the breakpoint on as shown below. The yellow arrow is the current program counter position. This will be the next instruction executed. The cyan arrow is a placeholder you can use to explore the relationship between a source window and the Disassembly window.
8. Add another breakpoint in while (1) or in a suitable place in one of the other threads.
9. Each time you click on RUN , the program will cycle to the next breakpoint.

**TIP:** You can set/unset hardware breakpoints with µVision while the program is running or stopped. This is an important feature.


**TIP:** You can identify an RTX thread by it having a first line such as this example:

```
void Thread_LED (void const *argument) {
```

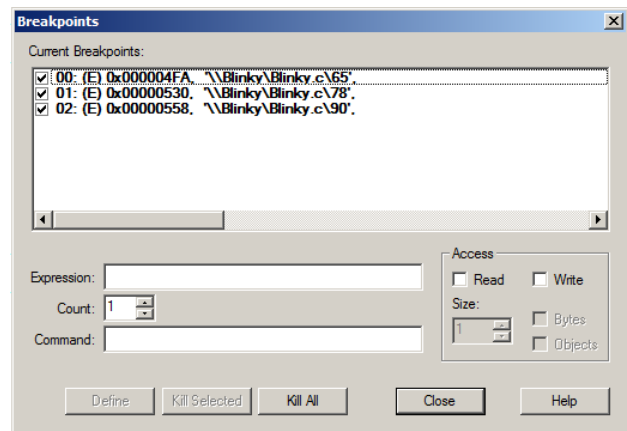
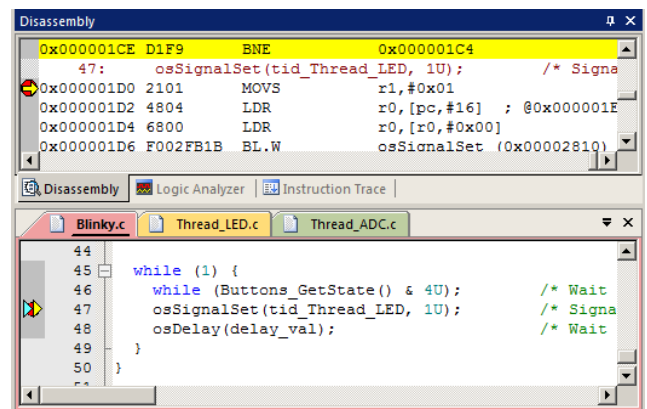
## Managing Breakpoints:

1. Select Debug/Breakpoints or Ctrl-B. This window opens:
2. You can temporarily unselect, delete or create breakpoints in this window. It is easier to create a breakpoint by clicking in a source file or the Disassembly window.
3. Watchpoints are also created in this window. This is discussed in a few pages.
4. **Select Kill All to remove all breakpoints.**
5. Click Close.
6. Click RUN  for the next exercise.

**TIP:** If you get multiple cyan arrows or can't understand the relationship between the C source and assembly, try lowering the compiler optimization to Level 0 and rebuilding your project. Compilers can do strange things while optimizing code.

The optimization level is set in Options for Target  under the C/C++ tab when not in Debug mode.

See MDK-ARM Compiler Optimizations: Appnote 202: [www.keil.com/appnotes/files/apnt202.pdf](http://www.keil.com/appnotes/files/apnt202.pdf)




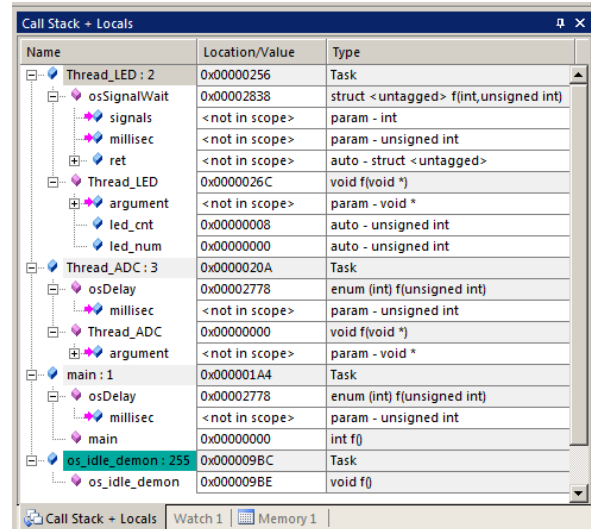
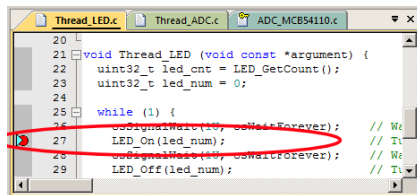
### 3) Call Stack + Locals Window:



#### Local Variables:

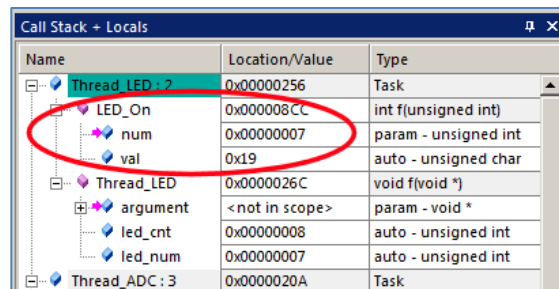
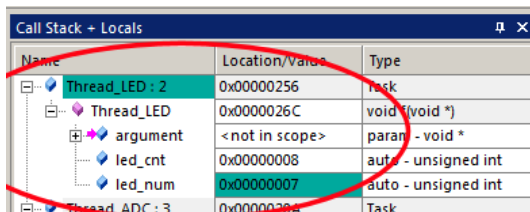
The Call Stack and Local windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function.

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

1. Stop the program with the STOP icon. The program will probably stop in the RTX os\_idle\_demon function.
2. Click on the Call Stack + Locals tab in  bottom right corner of µVision.
3. Inspect the various entries in the Call Stack + Locals window as shown below. Local variables are displayed only when they are in scope.
4. Note the os\_idle\_demon is highlighted with green. This program spends most of its time here at a branch to itself. You can easily adjust RTX time ratios in your code.
5. Double click on Thread\_LED.c in the Project window to open it.
6. In Thread\_LED.c, set a breakpoint on the call to the function LED\_On near line 27 as shown below:




7. Click on RUN  to run to the breakpoint. The Call Stack window below left shows Thread\_LED is active.
8. Click on Step  or F11 until you enter the function LED\_On as shown below right screen:
9. Shown below is the function Switch\_On active. Its local variable led has a value of 0x01 in this case.



10. Click on the Step Out icon  to exit the function LED\_On. This is indicated in the Call Stack window.

#### 11. Remove any hardware breakpoints by clicking on its **red circle** or Ctrl-B, then select Kill All and then Close !

**TIP:** This is standard “Stop and Go” debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables and structures updated in real-time while the program is running. No additions or changes to your code are required. They must be converted to global or static variables so they always remain in scope.

**TIP: Single Step:**  (F11): If you click inside a source window to bring it into focus, the program will step one source line at a time. If you click inside the Disassembly window to bring it into focus, it will then step one instruction at a time.

#### Call Stack Uses:

The list of stacked functions is displayed when the program is stopped. This is when you need to know which functions have been called and are stored on the stack. This is useful to find program flow problems such as crashes.

Normal functions are displayed only when they are on the Stack. They are removed and added as appropriate. When RTX is used, the threads are always displayed, even if not running. The active thread name and its number are highlighted in green.




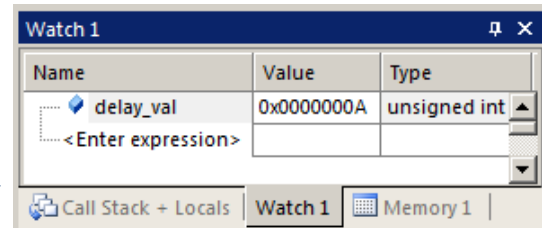
## 4) Watch and Memory Windows and how to use them:

The Watch and Memory windows will display updated variable values in real-time. It does this using ARM CoreSight DAP debugging technology. It is also possible to “put” or insert values into these memory locations in real-time using the Memory window while the program is running.

There is a global variable `delay_val` located in `Blinky.c` near line 26 we can use in the Watch and Memory windows.

### Watch window:

1. Click on the RUN icon  to start the Blinky program.
2. Right click on `delay_val` in `Blinky.c` near line 26 and select Add 'delay\_val' to... and select Watch 1.
3. `delay_val` will be displayed in Watch 1 as shown here:
4. As you vary the potentiometer P1, new values of `delay_val` are displayed in real-time.

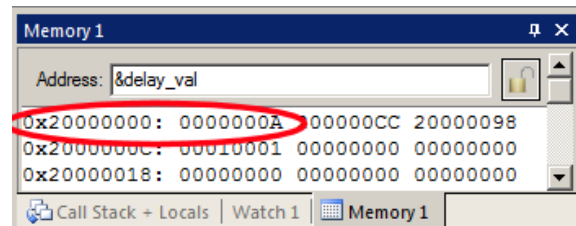


**TIP:** You do not need to stop the program to enter variables, raw addresses or structures in a Watch or Memory window.



5. You can modify the value in a Watch window when the program is stopped or changing slowly. You can modify a variable in a Memory window anytime. Step 5 below illustrates this technique.

### Memory window:

1. Right click on `delay_val` and select Add 'delay\_val' to... and select Memory 1.
2. Note the changing value of `delay_val` is displaying its address in Memory 1 as if it is a pointer as you vary the pot P1. This is useful to see what address a pointer is pointing to: but this not what we want to see at this time.
3. Right click in Memory 1 and select Unsigned Long to see the data field as 32-bit numbers.
4. Add an ampersand "&" in front of the variable name `delay_val` and press Enter. Now the physical address is shown (0x2000\_0000) in this case. This physical address could change with different compilation optimizations.
5. The data contents of `delay_val` is displayed as shown here:
6. Right click on the memory data value and select Modify Memory. Enter a value and this will be pushed into `delay_val`. Since `delay_val` is updated often, you will not see a new value displayed. But the update did occur.



**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 can change a value in a Memory window on-the-fly.

7. The global variable `delay_val` is updated in real-time. This is ARM CoreSight technology working.
8. Stop the CPU  and exit Debug mode  for the next step.

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

### 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. The Cortex-M3, M4 and M7 are Harvard architectures. This means they have 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 to memory without stealing any CPU cycles.

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

Remember you are not able to view local variables while the program is running. They are visible only when the program is stopped in their respective functions. You must change them to a different type of variable such as global to see them update.



## 5) System Viewer (SV):

System Viewer provides the ability to view certain registers in peripherals and the CPU core. In many cases, these windows are updated in real-time while your program is running. They are available only while in Debug mode. Select Peripherals/System Viewer to open the peripheral windows. Select CPU core registers by opening Peripherals/Core Peripherals. Both of these are illustrated here:

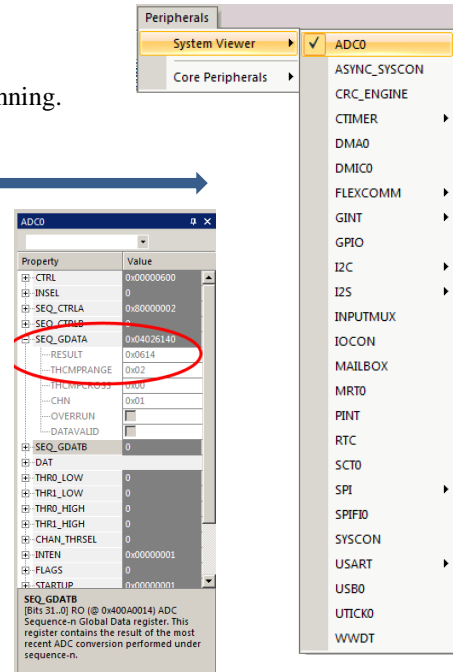
In our Blinky example, the ADC controls the speed the LEDs flash.

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



### ADC Port A:

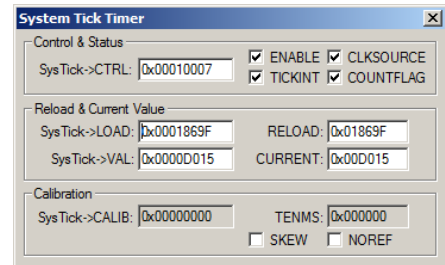
2. Select Peripherals/System Viewer and then ADC0 as shown here: 
3. The ADC0 window opens up: 
4. Expand SEQ\_GDATA as shown here:
5. As the pot P1 is rotated, the value in RESULT will change.
6. You can change the values in the System Viewer while the program is running or stopped. It will be difficult to see this as these values in this case are updated so often that your changes will be overwritten.
7. This window is updated periodically using the same CoreSight DAP technology as the Watch and Memory windows. For fast changing data, some values might be skipped.
8. 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 here for SEQ\_GDATB.



**SysTick Timer:** The Blinky example program uses the Cortex SysTick timer as the tick timer for RTX.

1. Select Peripherals/Core Peripherals and then select SysTick Timer. Run the program.
2. The SysTick Timer 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 value is set in the configuration of RTX in RTX\_Conf\_CM.c.
5. Note that it is set to 0x0001\_869F = dec 99,999. This is created by 100 MHz/1000-1 = 99,999. 1000 is specified as the timer tick value RTX\_Conf\_CM.c.. 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 0x5000 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 0x0001\_869F. A CPU RESET  will also accomplish this easier.
9. When you are done, Stop the program  and close all the System Viewer windows that are open.





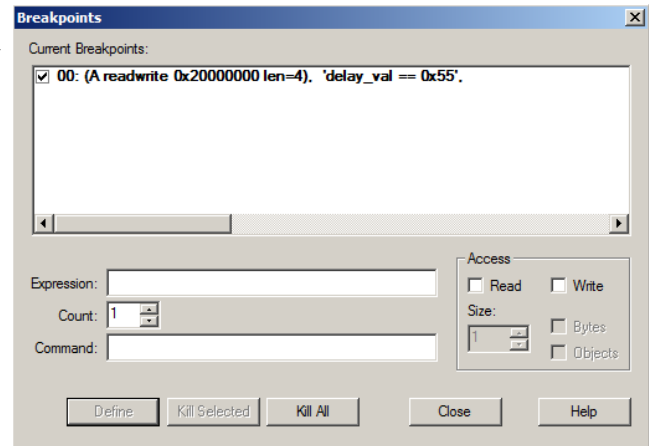
**TIP:** It is true: you can modify values in the SV while the program is running. This is very useful for making slight timing value changes instead of the usual modify, compile, program, run cycle.

You must make sure a given peripheral register allows 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.

## 6) Watchpoints: Conditional Breakpoints

Most NXP Cortex-M3, M4 and M7 processors have four data comparators. Since each Watchpoint uses two comparators, you can configure two complete Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same comparators in its operations. This means in  $\mu$ Vision you must have two variables free in the Logic Analyzer to use Watchpoints. The Cortex-M0 might have one Watchpoint. The Cortex-M0+ might have two of them. Currently,  $\mu$ Vision supports only one Watchpoint.

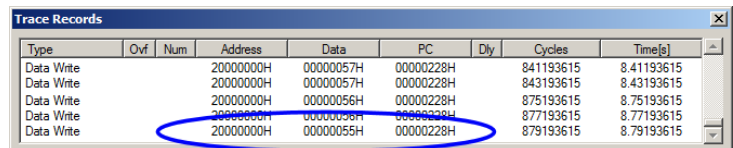
1. Using the example from the previous page, stay in Debug mode. Click on RUN  if the program is not running.
2. Enter the global variable `delay_val` into the Watch 1 window if it is not already there.
3. Click on Debug and select Breakpoints or press Ctrl-B. The Breakpoints window opens up.
4. Serial Wire Viewer (SWV) data trace does not need to be configured to use Watchpoints.
5. Select both the Read and Write Access.
6. Enter in Expression: "`delay_val == 0x55`" without the quotes.
7. Click on Define and it will be accepted as shown here:  
(the Expression: box will go blank) 
8. Click on Close. The program will still be running.
9. Rotate the pot P1 to produce 0x55 in Watch1
10. When `delay_val` equals 0x55, the program will stop. This is how a Watchpoint works.
11. You will see `delay_val` displayed as 0x55 in the Watch window.



### If You Have Serial Wire Viewer Running:

**Trace Records:** You would be able to see the data writes of `delay_val` displayed in the Trace Records window shown below: right:

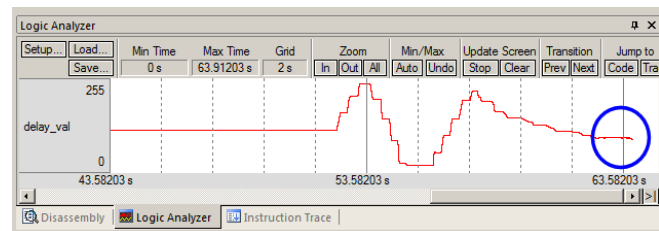
Here is the Trace Records display showing the data writes around 0x55: The address the data written to and the PC of the write instruction is displayed as well as the timestamps: A ULINKpro and J-Link trace windows are slightly different from ULINK2.





Type	Opf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	00000057H	00000228H		841193615	8.41193615
Data Write			20000000H	00000057H	00000228H		843193615	8.43193615
Data Write			20000000H	00000056H	00000228H		875193615	8.75193615
Data Write			20000000H	00000056H	00000228H		877193615	8.77193615
Data Write			20000000H	00000055H	00000228H		879193615	8.79193615

**Logic Analyzer:** This is the Logic Analyzer (LA) window displaying `delay_val`. The blue circle is where `delay_val` equaled 0x55 in the Watch1 window. You can have up to four displays in the LA. Watchpoints will use up one or two.

This is nicer than watching a variable changing in Watch 1.



12. When finished, click on STOP .
13. **Delete the Watchpoint** by selecting Debug and select Breakpoints and select Kill All.
14. Select Close.
15. Leave Debug mode. .

**TIP:** You can set Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

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

**TIP:** The checkbox beside the expression in Current Breakpoints as shown above allows you to temporarily unselect or disable a Watchpoint without deleting it.

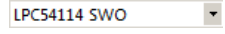





**TIP:** You can create a Watchpoint with a raw address and no variable value. This is useful for detecting stack overruns. Physical addresses can be entered as `*((unsigned long *)0x20000000)`.

## 7) Configuring Serial Wire Viewer (SWV): **Need a ULINK2, ULINKpro or a J-Link:**



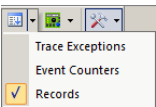
Serial Wire Viewer provides program information in real-time and is extremely useful in debugging programs.

SWV is available with any ULINK2, ULINK-ME, ULINKpro or a J-Link (external or LPC-Link2 in J-Link mode).

### Configure SWV:

1.  $\mu$  Vision must be stopped and in Edit mode (not Debug mode).
2. **If using the on-board LPC-Link2 in J-Link mode:** Load LPCScript as described on page 9 and install the J-Link firmware. Connect a USB cable from your PC to connector USB Link to connect LPC-Link2 and power the board.
3. **If using an external adapter such as a ULINK2, ULINKpro or J-Link:** Connect to the 10 pin CoreSight Debug or 20 pin JTAG connector. Power the board with a cable to the USB connector. **Do not** connect to USB Link.
4. Select LPC54114 SWO in the Target box. ULINK2 is preconfigured in this Target. 
5. Select Target Options  or ALT-F7 and select the Debug tab.
6. Confirm the name of your debug adapter is correct: in this case ULINK2/ME Cortex Debugger. 
7. If you are using a ULINKpro or a J-Link, select it here now.
8. J-Link needs an .ini file. See page 9 under j-Link.
9. Select Settings beside the name of your adapter. 
10. In the area **SW Device** must display a valid ARM CoreSight core.
11. In the Port: box, SW **must be selected**. SWV will not work with JTAG. 
12. Click on the Trace tab. The Target Driver Setup window below is displayed:
13. In Core Clock: enter 100 and select Trace Enable. This value is crucial for a ULINK2 or J-Link as a UART is used.
14. If using a ULINKpro, unselect Autodetect and enter 2 in SWO Clock Prescaler box to give a 50 MHz SWO Clock.
15. Confirm EXCTRC is selected.
16. Click on OK twice to return to the main  $\mu$  Vision menu. SWV is now configured.
17. Select File/Save All or click .

### To Display Trace Records:

1. Enter Debug mode.  Click on the RUN icon. .
2. Open Trace Records window by clicking on the small arrow beside the Trace icon shown here: (ULINK2)
3. The Trace Records window will open and display Exception 15 as shown below: (ULINK2 version) 
4. Double-click in the window to clear it. It will fill up again.

**TIP:** If you are using a ULINKpro or a J-Link, you must stop the program to display the trace data frames.

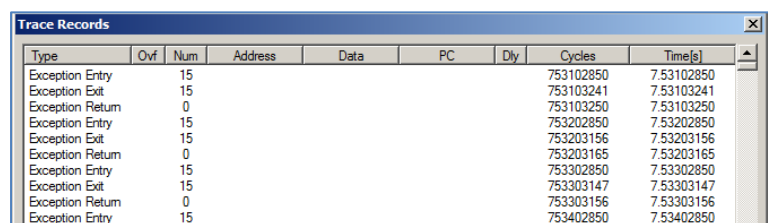
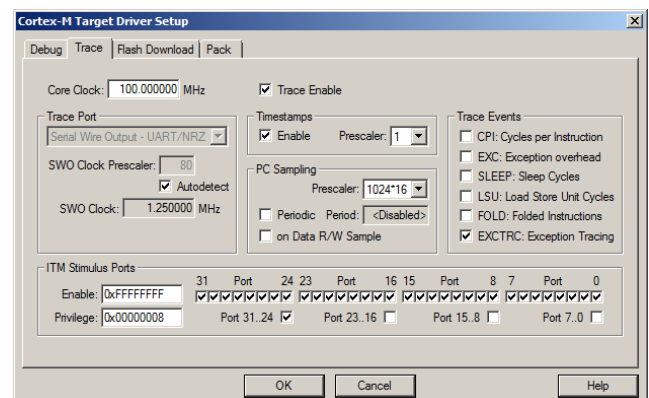
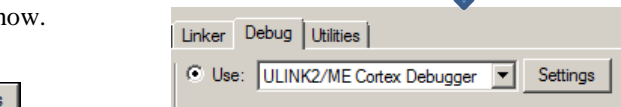
5. Displayed is Exception 15 (SYSTICK timer) with Entry, Exit and Return points.
6. Close the Trace Records window when you are done.

**Entry:** when the exception or interrupt is entered.

**Exit:** when the exception or interrupt exits.

**Return:** when all exceptions or interrupts exit. This indicates that no Cortex tail chaining is occurring.

**Trouble ?:** If you do not see Exceptions as shown and instead see either nothing or frames with strange data: the trace is not configured correctly. The most probable cause is the Core Clock: frequency is wrong. ITM frames 31 and 0 are the only valid ones. **Any other ITM frames are bogus and usually indicate a wrong Core Clock value.**



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry	15						753102850	7.53102850
Exception Exit	15						753103241	7.53103241
Exception Return	0						753103250	7.53103250
Exception Entry	15						753202850	7.53202850
Exception Exit	15						753203156	7.53203156
Exception Return	0						753203165	7.53203165
Exception Entry	15						753302850	7.53302850
Exception Exit	15						753303147	7.53303147
Exception Return	0						753303156	7.53303156
Exception Entry	15						753402850	7.53402850


## 8) Using the Logic Analyzer (LA): A ULINK2, ULINKpro or J-Link is needed for SWV:

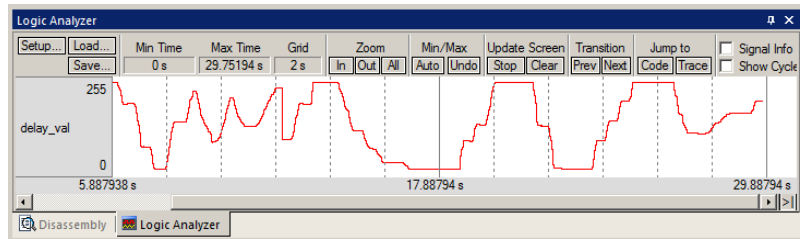
µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using Serial Wire Viewer. Logic Analyzer data points occur when the events are active with each data write. If SWV overload occurs and turning off unneeded SWV elements is not enough, you can sample the data writes with another variable and display it.

The LA shares the comparators in CoreSight with the Watchpoints. They are mutually exclusive.



1. The project Blinky.uvprojx should still be open and is still in Debug mode and either running or stopped.
2. Right click on delay\_val in Blinky.c and select Add 'delay\_val' to ... and select Logic Analyzer. The LA will open.

**TIP:** An error message saying delay\_val cannot be added usually means SWV is not configured or delay\_val is not in focus.

3. In the LA, click on Setup and set Max: in Display Range to 0xFF. Click on Close. The LA is now configured.
4. Select Debug/Debug Settings and select the Trace tab.
5. Select on **Data R/W Sample**.
6. Click OK twice to exit this menu.
7. Run the program. 
8. delay\_val will still be visible in Watch 1.
9. Adjust the Zoom OUT or the All icon in the LA to provide a suitable scale of about 2 seconds as shown above:




10. Rotate Pot P1 and delay\_val will display in LA as shown above and in Watch 1:

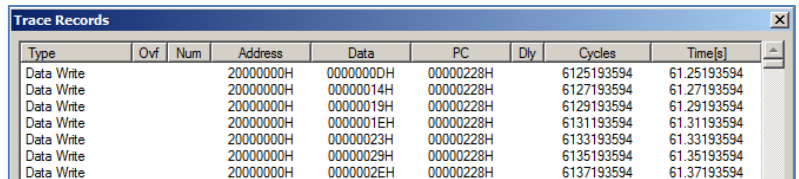
**TIP:** If no waveform is displayed in the LA, exit and reenter Debug mode   to refresh the comparators.

11. Enable Amplitude, Signal Info and Cursor in the LA window.
12. Select Stop in the Update Screen to stop the LA from collecting data. The program keeps running.
13. Click on a spot in the LA and position the mouse to demonstrate the method used to measure timing values.

**TIP:** When you enter a variable in the Logic Analyzer window, it is also displayed in the Trace Records (or Trace Data for ULINKpro) window. A ULINK2 or ULINKpro displays these Data Writes. A J-Link does not.

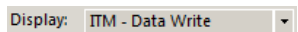
### View Trace Records Window (ULINK2):

1. Open the Trace Records window.
2. Right click inside Trace Records and unselect Exceptions and ITM Events.
3. Clear it by double clicking in it.
4. The window similar to here displays: 
5. Note the times might be the same. This is from SWV overload as the Data Write frames are created at every event. This example program creates many events. A ULINKpro works better as it can process SWO frames faster than a ULINK2 or J-Link.
6. The first line in the Trace Records window above states: The instruction at 0x0000\_0228 caused a write of data 0x0D to RAM memory address 0x2000\_0000 at the listed time in CPU Cycles or accumulated Time in seconds.
7. The PC column was activated when you selected on Data R/W Sample in Step 7 and is optional to lessen overloads.
8. An X in the DLY or Ovf column means there is a SWO overrun. Solutions are to eliminate any SWV selections not needed or use a ULINKpro which is faster. Candidates are Exceptions, ITM 31, ITM 0, PC Samples and entries in the LA. Sometimes you can turn off Timestamps but this can cause SWV to stop displaying if these are used.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	0000000DH	00000228H		6125193594	61.25193594
Data Write			20000000H	00000014H	00000228H		6127193594	61.27193594
Data Write			20000000H	00000019H	00000228H		6129193594	61.29193594
Data Write			20000000H	0000001EH	00000228H		6131193594	61.31193594
Data Write			20000000H	00000023H	00000228H		6133193594	61.33193594
Data Write			20000000H	00000029H	00000228H		6135193594	61.35193594
Data Write			20000000H	0000002EH	00000228H		6137193594	61.37193594

### View Trace Records Window (ULINKpro):

1. You must stop the program with a ULINKpro to view the frames in the Trace Data window.
2. In the Display: pull-down menu, select ITM – Data Write to filter the frames displayed. 
3. Double-click on a data write frame (or any frame) to highlight it in the source and Disassembly windows.


**TIP:** The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: make them static or global. To see peripheral registers, enter their physical addresses into the Logic Analyzer and read or write to them. Physical addresses can be entered as \*((unsigned long \*)0x20000000).

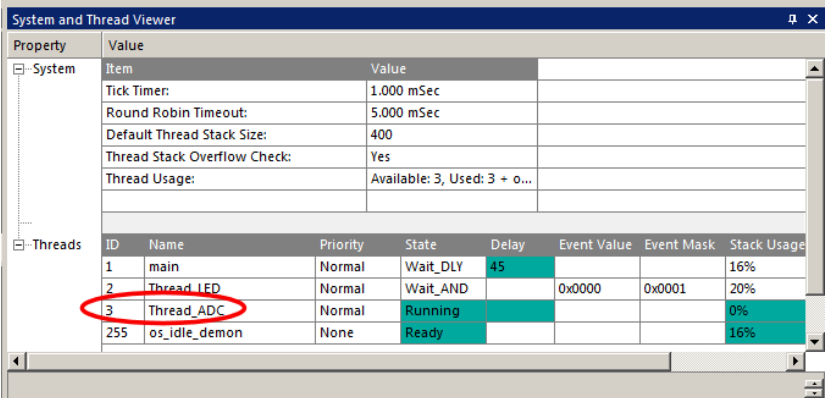


## 9) RTX Kernel Awareness Windows:

Users often want to know the current operating thread number and the status of the other threads. Keil provides two RTX Task Aware windows. Keil uses "threads" instead of "tasks" for consistency. RTX\_Blinky is running RTX\_Blinky with three threads. It is assumed an appropriate debug adapter is connected to your board as used on the previous page.

### A) System and Thread Viewer: All adapters including LPC-Link2 in CMSIS-DAP mode work.

1. Use the RTX\_Blinky example from the previous page.
2. Click on the Run icon. 
3. Open Debug/OS Support and select System and Thread Viewer and the window below opens. You might have to grab the window and move it into the center of the screen for viewing convenience. These values are updated in real-time using the same technology as used in the Watch and Memory windows.
4. Set a breakpoint in the thread Thread\_ADC in file Thread\_ADC.c. Line 29 is a good place.
5. The program will halt while Thread\_ADC is active and this is indicated as "Running" in the State column.
6. Statistics such as Stack Usage and timings are displayed.
7. os\_idle\_demon is the only thread ready to run at this time.
8. Remove the breakpoint.
9. Download the free Getting Started Guide: [www.keil.com/gsg](http://www.keil.com/gsg)






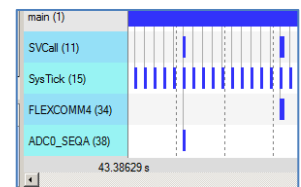
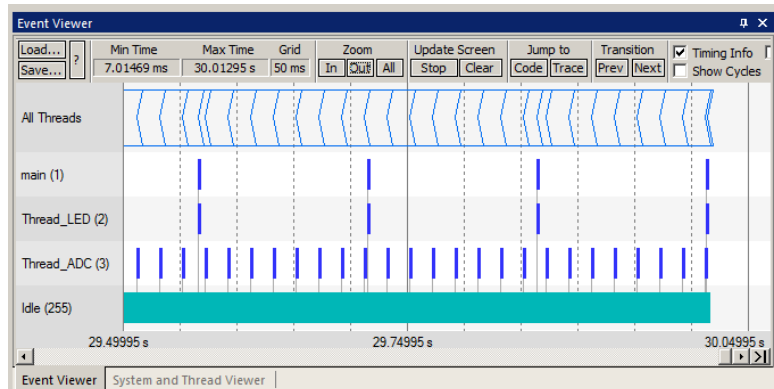
Property	Value
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	400
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 3, Used: 3 + 0...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	main	Normal	Wait_DLY	45			16%
2	Thread_LED	Normal	Wait_AND		0x0000	0x0001	20%
3	Thread_ADC	Normal	Running				0%
255	os_idle_demon	None	Ready				16%

### B) Event Viewer: Need ULINK2, ULINKpro, J-Link (LPC-Link2 or external) with SWV configured:

1. Open Debug/OS Support and select Event Viewer. The window below opens. Use Zoom In & Out to show threads.
2. If no threads are visible, exit and enter Debug mode to refresh all the CoreSight registers. 
3. Note the threads are listed on the left side. A green block indicates an active thread.
4. This program spends most of its time in idle\_demon. You can change this.
5. Click on Stop in Update Screen to halt data collection. The program keeps running.
6. Select Timing Info and Cursor.
7. If you hover the mouse over an active thread green block, the block turns yellow and timings are displayed as shown below:
8. Click on a spot and move the mouse to another spot to display timing information.
9. If you are using a ULINKpro: Exceptions will be displayed below the threads. Shown here are four interrupts. You can easily determine how long they are active and when they occur. This feature is very useful for optimizing interrupt handlers for peripherals such as DMA, USB, I<sup>2</sup>C and more.
10. If you hover your mouse on a event, in this example FLEXCOMM is on for 2.22 usec and ADC0 for 0.4 usec. Also see the Trace Exceptions window.
11. Stop the program.  Exit Debug mode. 



**TIP:** If the Event Viewer does not work, the most common cause is a wrong Core Clock: value or SWV overloaded. Reduce the active selections in trace config to the minimum. Unselect EXCTRC in Trace Exceptions window. Clear the LA.

**TIP:** If your program gets stuck in an ITM function see [www.keil.com/support/docs/3861.htm](http://www.keil.com/support/docs/3861.htm) SWV is not configured.



## 10) Debug (printf) Viewer: **Need ULINK2, ULINKpro, J-Link (LPC-Link2 or external)**

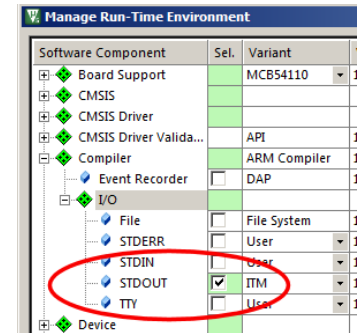
ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into  $\mu$ Vision for display in the Debug (printf) Viewer window. It is possible to send ITM data to a file: [www.keil.com/apnotes/docs/apnt\\_240.asp](http://www.keil.com/apnotes/docs/apnt_240.asp).

This works only with the Cortex-M3, M4 and M7. It will not work with the Cortex-M0+ because it does not have SWV.

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 will add 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 Thread\_LED.c:

1. In Thread\_LED.c, near line 4, add this line: `#include <stdio.h>`
2. In Thread\_LED.c, in the thread Thread\_LED (near line 21), near line 27, just before the call to LED\_On, enter this line: `printf("led_num = %d\n", led_num);`

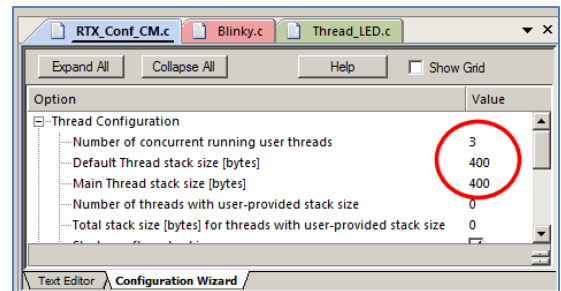
### Configure Serial Wire Viewer:

1. Serial Wire has already been configured. ITM 0 must be enabled in the Trace configuration.








### Increase the RTX stack size (If the program fails to run, perform – increase the stack size):

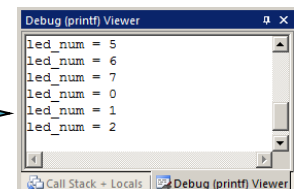
When a *printf* statement is added to a thread, you might have to increase the RTX stack size. This example has enough stack. These instructions are for reference to show how to manage the configuration of RTX. You can skip these steps in this case.

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.
4. Click RUN .
5. In the Debug (printf) Viewer you will see the *printf* statements appear. .
6. Right click on the Debug (printf) Viewer and select Mixed Hex ASCII mode.
7. When you are finished, stop the program  and exit Debug mode .



### 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.CM3.h.

This is documented here: [www.keil.com/pack/doc/CMSIS/Core/html/group\\_ITM\\_Debug\\_gr.html](http://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 ULINKpro works best.

**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](http://www.keil.com/support/docs/2735.htm) for instructions on how to add this feature to your own source code.

## Dual Core Example: All adapters will work. SWV is not used in this section.

This example uses on-board CMSIS-DAP mode by default. It is possible to use a ULINK2, ULINKpro or a J-Link. To use one of these debug adapters, see page 9 for configuration instructions. The CMSIS-DAP that comes with the MCB54110 board is different than the one distributed by NXP LPCScript. A ULINK2 or ULINKpro is best to use.



### 1) Notes on the DualCore Example:

1. MDK controls both processors by running two instances of  $\mu$ Vision: one controlling the Cortex-M4 and the other the Cortex-M0+. A dual monitor on your PC is a useful setup: use one for each instance of  $\mu$ Vision.
2. There are two  $\mu$ Vision projects: one for the Cortex-M4 and another for the Cortex-M0+. They are located in the \CM4 and \CM0plus folders respectively. They are configured into a  $\mu$ Vision Multi-Project: DualCore.uvmpw
3. Each processor is running its own instance of RTX (Keil RTOS).
4. The Cortex-M4 sends LED status to the M0+ via an RTX mailbox. The Cortex-M0+ changes the LEDs accordingly.
5. In each of the instance  $\mu$ Vision you need to select your debug adapter if not using CMSIS-DAP.


### 2) Loading and Configuring $\mu$ Vision for the Dual CPU project:

1. Connect a USB cable to USB LINK connector on the MCB54110 board for CMSIS-DAP mode. See page 9 for connecting other debug adapters. Currently, Keil CMSIS-DAP, a ULINK2 or a ULINKpro are most stable.

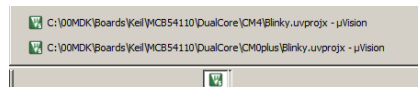
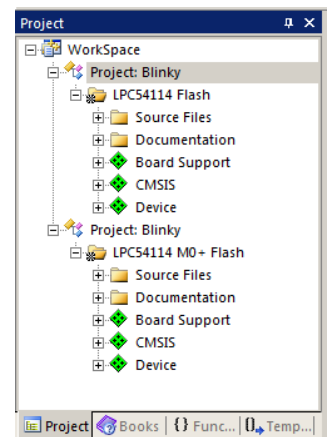
#### Cortex-M4 Instance of $\mu$ Vision:

2. Start  $\mu$ Vision by clicking on its desktop icon:  or use a running instance. This will be the Cortex-M4 instance.
3. Select Project/Open Project and open C:\00MDK\Boards\Keil\MCB54110\DualCore\DualCore.uvmpw.
4. Note the Project window shows two projects: both are called Blinky. 
5. The top one is for the Cortex-M4. The bottom one is for the Cortex-M0+.
6. The Cortex-M4 is active as evidenced by the grey block.
7. If it is not, right click on the upper Project Blinky and select Set as Active Project:



#### Cortex-M0+ Instance of $\mu$ Vision:

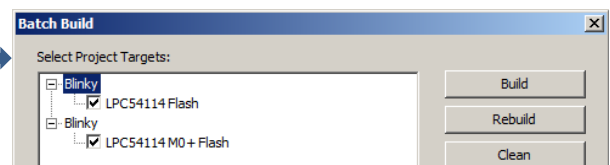
1. Start a second instance of  $\mu$ Vision by clicking on its desktop icon again. 
2. The DualCore uvmpw project should open automatically.
3. In the Project window, right click on the lower Project: Blinky and select Set as Active:
4. You will now have two instances of  $\mu$ Vision running as shown below: The top header of each instance of  $\mu$ Vision will indicate the instance by CM4 or CM0plus in the filename.

Set as Active Project



### 3) Compiling the Programs:


1. Select the Cortex-M4 instance of  $\mu$ Vision.
2. Click on the Batch Build icon:  This window opens: 
3. Click Rebuild to compile both projects.
4. There will be no errors or warnings in the Build Output window. It will display "2 succeeded".



**TIP:** If you have created other Options Targets for other debug adapters, they will be visible here. Select what you need.

1. You have now successfully compiled both the Cortex-M4 and M0+ source files in the Blinky example programs.
2. On the next page, you will program the Flash and run the program.



### 4) Programming the Flash (as an optional test):

1. From the CM4 instance, click the Load icon.  The Flash programming progress is indicated in Build Output.
2. The eight LEDs will now blink in sequence. If not, something is wrong and you must fix this to continue.
3. **Note:** the pot P1 will not vary the LED sequence rate in this project.



## 5) Programming the Flash and Running the two Blinky programs:

At this point, the two Blinky programs are running on both cores from Flash memory without any intervention by  $\mu$ Vision. If you disconnect the ULINK2 and reset the board, these two programs will now run stand-alone from the Flash. Now, we will connect  $\mu$ Vision to each in turn to demonstrate two core debugging techniques.

### CM4 Blinky:

- 1) Bring the CM4 instance of  $\mu$ Vision into focus by clicking on or selecting it.
- 2) Enter Debug mode for M4 Blinky . The Flash will be programmed for both the M4 and M0+.
- 3) The LEDs will stop blinking. Click on RUN  and they will start again. The M0+ was started by the M4 and is writing to the LEDs. The  $\mu$ Vision debugger is not connected to the Cortex-M0+ core at this time.

### CM0plus Blinky:



- 1) Bring the Cortex-M0+ instance of  $\mu$ Vision into focus by clicking on it or selecting it.
- 2) Enter Debug mode for M0 Blinky . Click on RUN  and the Cortex-M0+ program will run.

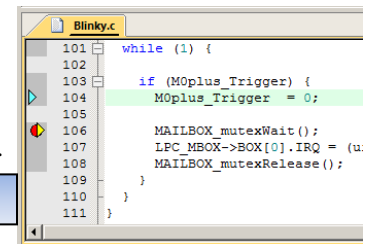
**Troubles ?** If you encounter trouble, repower the board and/or an external debug adapter and try again. With LPC-Link2 CMSIS-DAP, wait for the USB dual tone to sound indicating it has enumerated and connected before entering Debug mode.

**TIP:** You can see a source file from one instance of  $\mu$ Vision in the other instance. *Be careful to modify the correct one.* To determine which one is active: look for the grey boxes on the left indicating there is assembly instructions present as shown below. No grey boxes mean this is not the correct Blinky.c for the active instance of  $\mu$ Vision.



## 6) Setting Hardware Breakpoints:

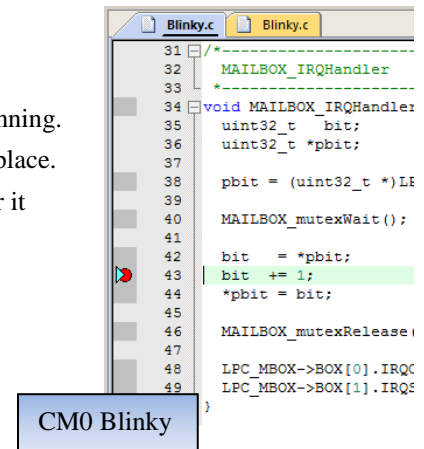
### CM4 Blinky:

- 1) Bring the CM4 instance of  $\mu$ Vision into focus. Select the correct Blinky.c.
- 2) Set a breakpoint in Blinky.c in the while (1) loop near line 106 as shown here:
- 3) This is where the Cortex-M4 tells the Cortex-M0+ to switch LEDs via the mailbox.
- 4) The CM4 program will soon stop on the breakpoint.
- 5) Each time you click on RUN , the LEDs will change to the next position.
- 6) Remove the breakpoint. Click on RUN .



### CM0plus Blinky:

- 1) Bring the CM0+ instance of  $\mu$ Vision into focus by clicking on it. It will be running.
- 2) Set a breakpoint in Blinky.c in the mailbox\_IRQ\_Handler. Line 43 is a good place.
- 3) This is where the Cortex-M0+ signals the main thread to modify the LED after it receives a message from the Cortex-M4 through the mailbox to do so.
- 4) The CM0+ program will soon stop on the breakpoint.
- 5) Each time you click on RUN, the LEDs will change to the next position.
- 6) **When finished, remove all breakpoints from both instances of  $\mu$ Vision.**
- 7) When you are finished, Stop the program  and exit Debug mode  in both instances of  $\mu$ Vision to continue.




**TIP:** You can see that you can easily control both processors using two instances of  $\mu$ Vision. A standard MDK license allows this operation. You do not need to purchase two licenses to operate the two Cortex cores as demonstrated here.

## 7) Watch Window with DualCore:


Create two global variables to increment in each processor.

### CM4 Blinky:


- 1) Bring the CM4 instance of  $\mu$ Vision into focus by clicking on it.
- 2) Create a global variable near line 23 in Blinky.c: **unsigned int CM4 = 0;**
- 3) In Blinky.c in the main() function near line 108, add this line just before MAILBOX\_mutexRelease();: **CM4++;**
- 4) Select File/Save All or click: 

**TIP:** Make sure you are modifying the correct Blinky.c ! Select Blinky.c in the Project window to make sure.




### CM0plus Blinky:

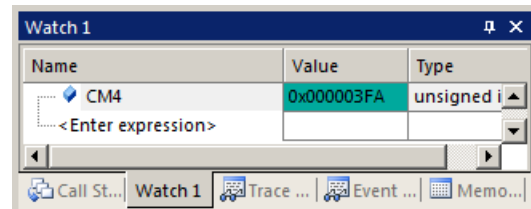
- 1) Bring the CM0 instance of  $\mu$ Vision into focus by clicking on it. If asked to reload Blinky.c message, enter YES.
- 2) Create a global variable near line 16 in Blinky.c: **unsigned int CM0 = 0;**
- 3) In Blinky.c, in the function MAILBOX\_IRQHandler, near line 44 and just after bit +=1:, add this line: **CM0++;**
- 4) Select File/Save All or click: 

### Compile:





- 1) Select the Cortex-M4 instance of  $\mu$ Vision.
- 2) Click on the Batch Build icon as done previously:  The Batch Build window opens.
- 3) Click Rebuild to compile both projects. A successful build (2) is displayed in the Build Output window.
- 4) There will be no errors or warnings. If there are, please fix them before continuing.

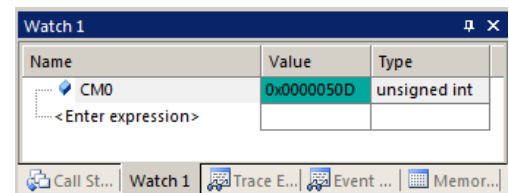
### Run CM4 Blinky:

- 1) Enter Debug mode for M4 Blinky . The Flash will be programmed for both the Cortex-M4 and Cortex-M0+.
- 2) The LEDs will stop blinking.
- 3) Click on RUN  and it will start.
- 4) Right click on the CM4 variable in Blinky.c and select Add 'CM4' to... and then select Watch 1.
- 5) Watch 1 will open and display CM4 incrementing from within the CM4 Blinky project as shown here: 



### Run CM0plus Blinky:

- 1) Select the Cortex-M0+ instance of  $\mu$ Vision.
- 2) Enter Debug mode for M0plus Blinky . The Flash is already programmed by the M4 project.
- 3) Click RUN .
- 4) Right click on the CM0 variable in Blinky.c and select Add 'CM0' to... and then select Watch 1.
- 5) Watch 1 will open and display CM0 incrementing from the CM0+ project.
- 6) When you are finished, Stop the program  and exit Debug mode  in both instances of  $\mu$ Vision.

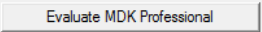


**TIP:** With this system, the core you are developing can be controlled with  $\mu$ Vision while the other runs by itself. You must always build the projects from the CM4 instance as it creates files for the CM0plus instance.



## USB Mass Storage Example: All adapters will work. SWV is not used in this section.




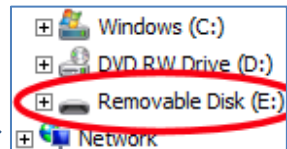
There are many examples for the Keil MCB54110 board. They are components from Keil Middleware USB (Device and Host classes), emWin graphics and a File System. These are loaded from a Software Pack with Pack Installer.

1. MDK Professional is needed to compile components of Keil Middleware. You can obtain a one-time 7 day license in File/License Management. If you are eligible, this button is visible: 
2. This gives you unlimited code size compilation and access to Keil Middleware. Contact Keil Sales to extend this license for evaluation purposes.


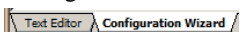
### Notes:

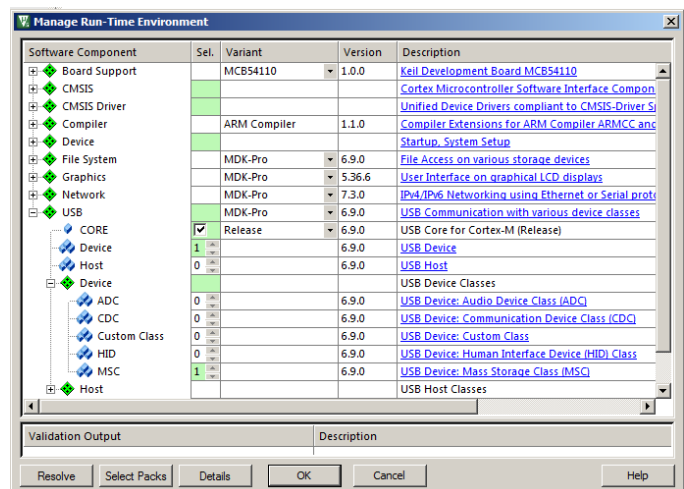
1. You can use any of the available debug adapters including LPC Link in either CMSIS-DAP or J-Link mode.
2. If you use a ULINK2, ULINK $pro$  or a J-Link, you can use the Serial Wire Viewer. A ULINK $pro$  works best.
3. The USB Mass Storage connector will be the USB connector. If using an external debug adapter, use this also for power. If using the on-board LPC Link 2, connect a USB cable to both the USB and USB Link connectors.
4. Insert a micro SD card into the socket SD1. You will be able to read and write this device with this example.

## 1) Installing the USB Mass Storage Example:

1. Connect the MCB54110 board with an external debug adapter or the on-board LPC-Link2.
2. Select Project/Open Project from the main  $\mu$ Vision menu.
3. Load C:\00MDK\Boards\Keil\MCB54110\Middleware\USB\Device\MassStorage\MassStorage.uvprojx.
4. Compile the source files by clicking on the Rebuild icon. 
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears. Flash programming progress is indicated in bottom left corner. You might hear the USB double down tone indicating a USB device has been disconnected.
6. Click RUN .
7. You will hear the usual USB dual up tone as USB is initialized.
8. Open Windows Explorer and open the Removable Disk as shown here: 
9. You can copy or read any files that are on the micro SD card with Microsoft Explorer.

## 2) Keil USB Middleware:

1. Open the Manage Run Time Environment window: 
2. Expand various USB headers that are shaded in green to see the potential settings that were made to create the Mass Storage device.
3. Selecting these components added the appropriate files to the project.
4. Additional configuration is made with the Configuration Wizard on some of these files.  

5. Other Middleware components such as Network, File System and Graphics are selected and configured in the same way. It is clear to see this is a very efficient way to distribute and use Keil Middleware products. Keil provides comprehensive options for its Middleware.
6. Close the MRTE window. Do not make any changes.







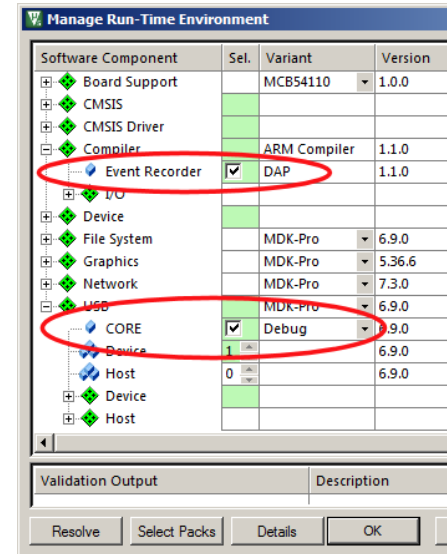
### 3) Event Recorder: All adapters will work. SWV is not used in this section.

A new feature provides listing events that occur in the Keil Middleware. We will activate this feature. It uses CoreSight DAP to output data from the target. This means any debug adapter including LPC-Link2 in CMSIS-DAP mode can be used.







For more information on Event Recorder: [www.keil.com/support/man/docs/uv4/uv4\\_db\\_dbg\\_evr.htm](http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr.htm)

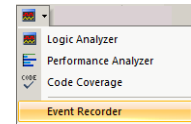
#### Configure for Event Recorder Operation:




1. Stop the USB Mass Storage program  and exit Debug mode 
2. Open the Manage Run Time Environment window: 
3. Expand Compiler and Select Event Recorder as shown here:
4. Expand USB and beside CORE, change from Release to Debug.
5. Click on OK to close this window.
6. Various files are now added to the Project window.
7. In MassStorage.c near line 11 enter this line:  
`#include "EventRecorder.h"`
8. Near line 13, enter these two lines at the start of main():  
`EventRecorderInitialize(EventRecordAll, 1);`  
`EventRecorderStart ();`
9. Select File/Save All or click: 

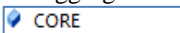



#### Compile and RUN the Program:

1. Compile the source files by clicking on the Rebuild icon. 
2. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears. Flash programming progress is indicated in bottom left corner.
3. Select the small arrow beside Analysis icon and select Event Recorder which will open:
4. Click on the RUN icon  to start the Blinky program.
5. Events will start to fill in the Event Recorder window as shown here:
6. You can stop the recording by unselecting Enable Recorder:
7. When you re-enable it, the events not shown will be displayed as they were collected in the background.
8. Stop the USB Mass Storage program 
9. Hover your mouse over Event Property entries to gain more information: 
10. Double clicking on an entry in the Event Property will bring up Help on that subject while the program is halted.
11. You can modify the information displayed with the Funnel icon: 







Event Recorder				
Enable Recorder: <input checked="" type="checkbox"/>    Mark: <input type="text"/> All Operations <input type="text"/> Recording				
Event	Time (sec)	Component	Event Property	Value
0	0.00725633		Init Event	Restart Count=0x00000001
1	0.00725956		Start Event	
2	0.00726350	USBD_Core	Initialize	device=0
3	0.00734070	USBD_Core	OnInitialize	n=0
4	0.00734523	USBD_MSC	Initialize	instance=0
5	0.00735093	USBD_MSC	OnInitialize	n=0
6	0.00745109	USBD_MSC	OnGetCacheInfo	n=0, size=512
7	0.00749323	USBD_Driver	Initialize	device=0
8	0.00750697	USBD_Driver	PowerControl	device=0, state=ARM_POWER_OFF
9	0.00751122	USBD_Core	Connect	device=0
10	0.00752172	USBD_Driver	DeviceConnect	device=0
11	0.01150405	USBD_Driver	OnSignalDeviceEvent	device=0, event=ARM_USBD_EVENT_SUSPEND

**TIP:** When you are finished with debugging with the Event Recorder, you might want to return to the Release setting under USB::CORE instead of Debug.  

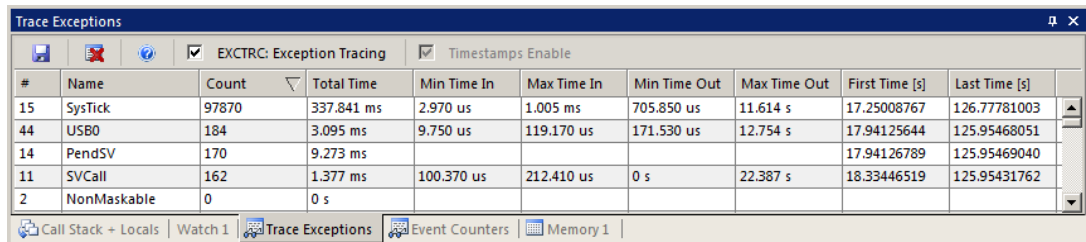
## 6) Using Serial Wire Viewer (SWV): Needs a ULINK2, ULINKpro or a J-Link.

The USB Mass Storage example uses RTX. We can use SWV to examine parts of RTX running but also measure the times of interrupt routines (with ULINKpro) and view variables and peripheral registers as shown in previous exercises in this lab.

1. Stop the program  and exit Debug mode .
2. To configure SWV for your debug adapter, refer to Configuring Serial Wire Viewer (SWV): on page 16.
3. Enter Debug mode . Click on RUN .

### Trace Exceptions (and Interrupts):

1. The Trace Exceptions window will be active at the bottom right of  $\mu$ Vision. Click on its tab: You can toggle its view in View/Trace/Trace Exceptions or by using the Trace icon.



The screenshot shows the 'Trace Exceptions' window in  $\mu$ Vision. It has a toolbar with icons for file operations and checkboxes for 'EXCTRC: Exception Tracing' and 'Timestamps Enable'. Below is a table with columns: #, Name, Count, Total Time, Min Time In, Max Time In, Min Time Out, Max Time Out, First Time [s], and Last Time [s]. The table lists several exceptions: SysTick (97870 counts), USB0 (184 counts), PendSV (170 counts), SVCALL (162 counts), and NonMaskable (0 counts). At the bottom, there are tabs for 'Call Stack + Locals', 'Watch 1', 'Trace Exceptions' (which is selected), 'Event Counters', and 'Memory 1'.

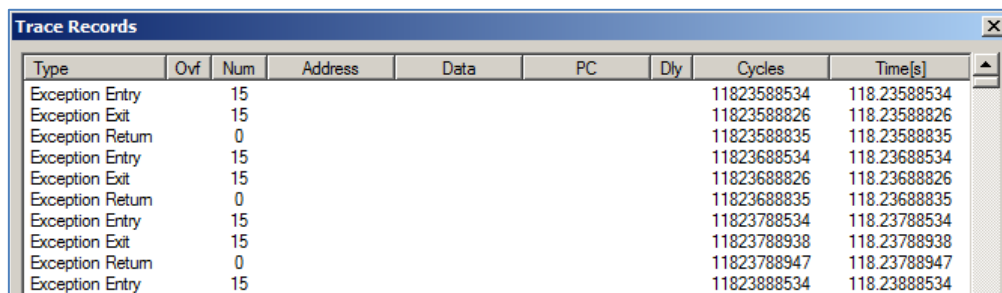
#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
15	SysTick	97870	337.841 ms	2.970 us	1.005 ms	705.850 us	11.614 s	17.25008767	126.77781003
44	USB0	184	3.095 ms	9.750 us	119.170 us	171.530 us	12.754 s	17.94125644	125.95468051
14	PendSV	170	9.273 ms					17.94126789	125.95469040
11	SVCALL	162	1.377 ms	100.370 us	212.410 us	0 s	22.387 s	18.33446519	125.95431762
2	NonMaskable	0	0 s						

2. Click on the Count column name to bring any active exceptions at the top.
3. Note exceptions visible – especially the USB0 one. Exceptions can be turned on/off with the EXCTRC box. This has the same effect as selecting/unselecting the EXCTRC box in the Trace configuration window.
4. Open the Trace Records window. Note this window will be different for ULINKpro and J-Link and the program must be halted to view the trace frames.

### Three ways to view Exceptions:

1. Trace Exceptions window.
2. Trace Records or Instruction Trace (ULINKpro).
3. Event Viewer (with RTX and ULINKpro).

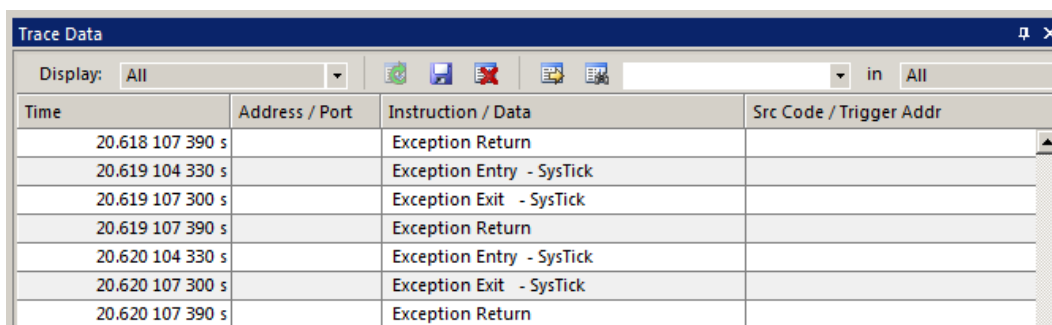
### Trace Records window with ULINK2:



The screenshot shows the 'Trace Records' window. It has a table with columns: Type, Ovf, Num, Address, Data, PC, Dly, Cycles, and Time[s]. The table lists several 'Exception Entry' and 'Exception Exit' records with their corresponding addresses and times.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					11823588534	118.23588534
Exception Exit		15					11823588826	118.23588826
Exception Return		0					11823588835	118.23588835
Exception Entry		15					11823688534	118.23688534
Exception Exit		15					11823688826	118.23688826
Exception Return		0					11823688835	118.23688835
Exception Entry		15					11823788534	118.23788534
Exception Exit		15					11823788938	118.23788938
Exception Return		0					11823788947	118.23788947
Exception Entry		15					11823888534	118.23888534

### Trace Data window with ULINKpro:



The screenshot shows the 'Trace Data' window. It has a toolbar with icons for file operations and a 'Display' dropdown menu set to 'All'. Below is a table with columns: Time, Address / Port, Instruction / Data, and Src Code / Trigger Addr. The table lists several 'Exception Return' and 'Exception Entry - SysTick' records with their corresponding times and addresses.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
20.618 107 390 s		Exception Return	
20.619 104 330 s		Exception Entry - SysTick	
20.619 107 300 s		Exception Exit - SysTick	
20.619 107 390 s		Exception Return	
20.620 104 330 s		Exception Entry - SysTick	
20.620 107 300 s		Exception Exit - SysTick	
20.620 107 390 s		Exception Return	

## Event Viewer with RTX: A ULINK2, ULINKpro or a J-Link is needed for SWV operation.

All Keil Middleware uses RTX by default. RTX has either a BSD or Apache 2.0 license depending on the version. See [www.keil.com/rtx](http://www.keil.com/rtx). Source code is included. RTX can be selected using the Manage Run Time Environment utility.

Event Viewer displays timing of RTX Threads and, with a ULINKpro, exception handler timings.

1. If using a ULINK2, unselect EXCTRC Exception Tracing in the Trace Exception window. ☐ EXCTRC: Exception Tracing
2. Open Debug/OS Support and select Event Viewer. The window below opens up:
3. Adjust ZOOM In and Out to about 1 ms or so. Scroll to the beginning of the Event Viewer window.

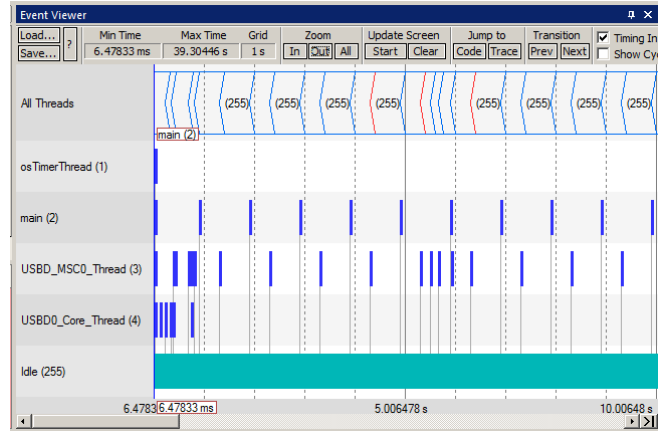
**TIP:** If Event Viewer is still blank, exit and re-enter Debug mode to reset the CoreSight components.  

### Main Thread:

1. Select Stop in the Update Screen. Scroll to the beginning of the Event Viewer.
2. You can see which threads ran in which order. The main thread is the main() function in MassStorage.c. It runs some RTX initialization code at the beginning.

**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. Use a ULINKpro.

3. The 4 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 on Zoom In so several threads are displayed.
5. Select Cursor and Timing Information. Position the cursor over one set of bars and click once. A red line is set.
6. Move your cursor to the next set and total time and difference are displayed.
7. Since you enabled Show Cycles, the total cycles and difference is also shown when you hover your mouse.



## 6) Using a Keil ULINKpro to view Interrupt Handler execution times:

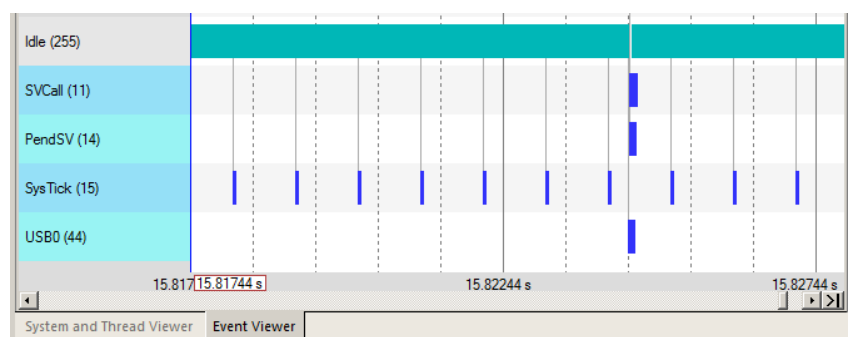
**SWV Throughput:** ULINKpro is much better handling SWO bandwidth issues than a ULINK2 or a J-Link.

ULINKpro can also use the 4 bit Trace Port for even faster operation for SWV. The LPC54114 does not have ETM but many other NXP processors do. Trace Port use is mandatory for ETM trace. A ULINKpro in ETM mode provides program flow debugging, Code Coverage and Performance Analysis. ULINKpro also supports ETB (Embedded Trace Buffer) as found in some NXP processors.

**Exceptions:** A ULINKpro displays exceptions at the bottom of the Event Viewer. Shown here are the SysTick, PendSV, SVCcall and USB0 exceptions. Any other exception events such as DMA will also be displayed here.

You can easily measure the duration of the time spent a handler runs with the Event Viewer techniques you have learned.

In this window, USB0 (interrupt 44) handler runs for approximately 10 msec.



## 1) DSP Example using ARM CMSIS-DSP Libraries:

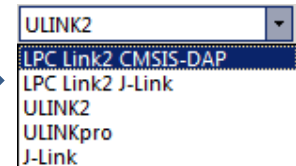
ARM CMSIS-DSP libraries with sources are provided for Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7. DSP libraries plus all sources are provided in MDK in C:\Keil\_v5\ARM\Pack\ARM\CMSIS\ and are selected with MRTE. See [www.keil.com/cmsis](http://www.keil.com/cmsis) and [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_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 or an Apache 2.0 license. All source code is provided.

This program will run with CMSIS-DAP but to see the interesting and useful SWV features you need any ULINK, J-Link or LPC-Link2 in J-Link mode. See page 9 for instructions to implement these modes.

1. Open the project file C:\00MDK\Boards\Keil\MCB54110\DSP\sine.uvprojx with  $\mu$ Vision.
2. **To use LPC-Link:** connect a USB cable to the USB connector.
3. **For any ULINK or J-Link:** Connect it to J5 Target Power and connect the ULINK or J-Link to the 10 pin Cortex Debug or 20 pin legacy JTAG connector.
4. Select your debug adapter from the pull-down menu as shown here:
5. Compile the source files by clicking on the Rebuild icon.
6. Enter Debug mode by clicking on the Debug icon. The Flash will be programmed.

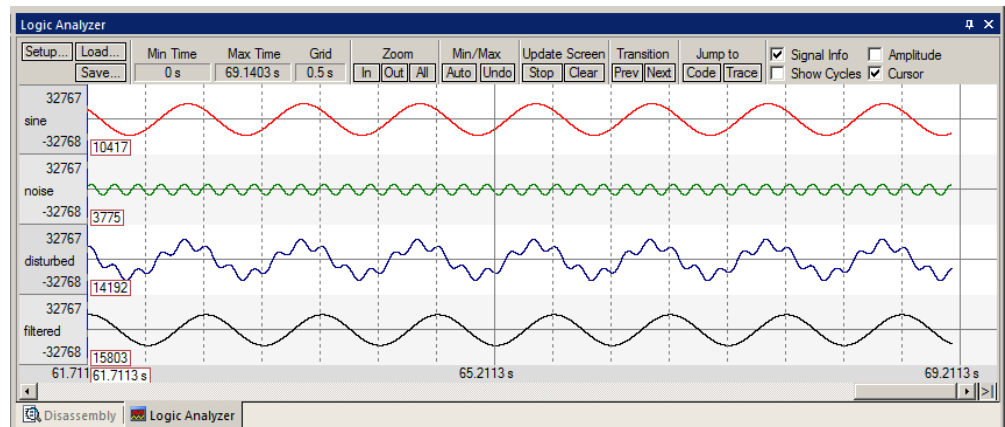


**TIP:** The default Core Clock: is 100 MHz.

7. Click on the RUN icon. Open the Logic Analyzer window and the Watch 1 window: View/Watch/
8. This project has Serial Wire Viewer configured and the Logic Analyzer and Watch 1 loaded with the four variables.
9. Four waveforms are displayed in the Logic Analyzer using Serial Wire Viewer as shown below. Adjust Zoom for an appropriate display. Displayed are the 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. ULINKpro handles SWV data faster than a ULINK2 or J-Link can. Make sure the Core Clock is set to 100. If the variables in Watch 1 are changing, the program is running correctly.

10. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below:
11. 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 ULINKpro you must stop the program to display the data Trace Data window. J-Link does not display any data read or write operations. LPC-Link2 CMSIS-DAP has no SWV support at this time. It does in CMSIS 5.

12. Select View/Serial Windows/Debug (printf) Viewer. ASCII data is displayed from the printf statements in DirtyFilter.c. This features uses SWV so it is not available with CMSIS-DAP.
13. You can view other windows such as the two for RTX and the Trace Records window.

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

14. Stop the program and exit Debug mode.



## 1) Creating your own MDK 5 project: with no RTOS:

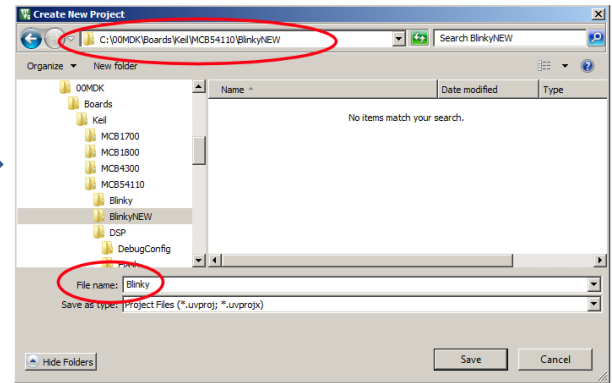
All examples provided by Keil or MCUXpresso are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will create this example project from the beginning to illustrate how easy this process is. Once you have the new project configured; 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. Then, we will add the Keil RTX RTOS.

### Install the Software Pack for your processor:

1. Start  $\mu$ Vision 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::LPC54000\_DFP Software Pack must be installed. This was done on page 4.

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

1. In the main  $\mu$ Vision menu, select Project/New  $\mu$ Vision Project... A Create New Project window opens:
2. In this window, shown here, navigate to the folder C:\00MDK\Boards\Keil\MCB54110\
3. Right click in this window and select New (or click New Folder) and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj. (MDK 4 format)
7. The Select Device for Target... automatically opens:

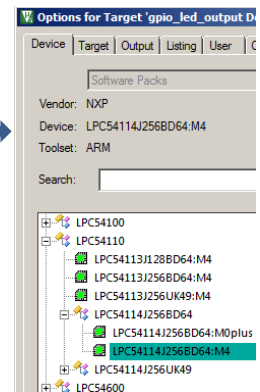


### Select the Device you are using:

1. Expand NXP and LPC54110 and select LPC54114J256BD64:M4 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.

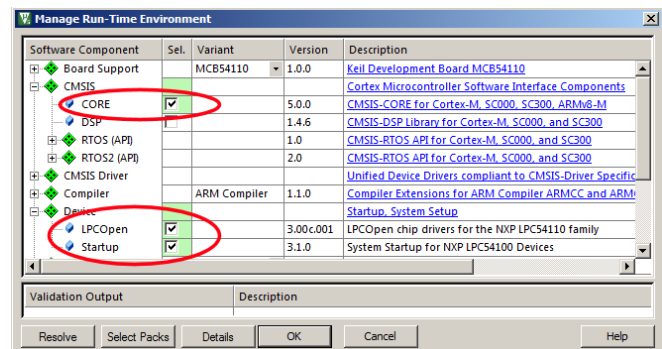
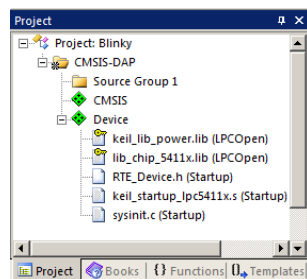


### Select the CMSIS components you want:

1. Expand CMSIS and Device. Select CORE, LPCOpen 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 are 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 **CMSIS-DAP** and press Enter. The Select Target name will also change. You can enter any name you want.

### What has happened to this point:

You have created a blank  $\mu$ Vision 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.




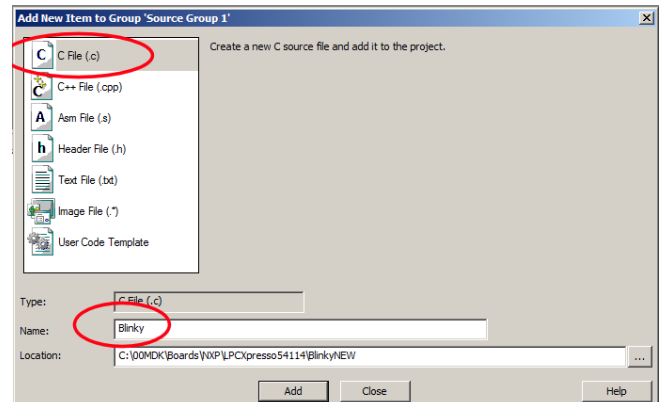
## 2) Create Source Files and Confirm Flash Programming:

### Create a blank C Source File:



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

Add New Item to Group 'Source Files'...

2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter Blinky.
5. Click 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 chip.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. Fix them if there are any.

```
#include "chip.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.

### Confirm the Target Flash Configuration: µVision configures Flash settings based on the Software Pack.

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:  Use: CMSIS-DAP Debugger 

5. Connect your Keil MCB54110 board to your PC USB port. See page 9 for instructions.

6. Select the Settings: button beside Use: CMSIS-DAP Debugger as shown above.


7. Set SW as shown here:  If your 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 MCB54110 board:


Programming Algorithm			
Description	Device Size	Device Type	Address Range
LPC5411x IAP 256kB Flash	256k	On-chip Flash	00000000H - 0003FFFFH

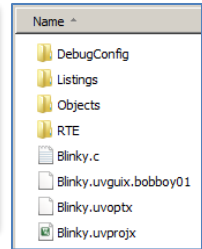
9. Click on OK twice to return to the main menu.

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 !

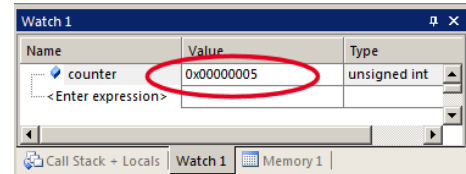
### What we have so far ?

1. A project has been created in C:\00MDK\Boards\Keil\MCB54110\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.



## 3) 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.









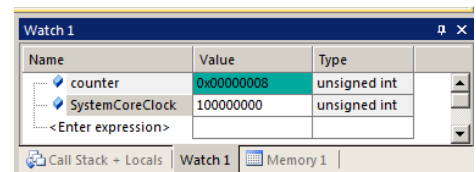
**TIP:** Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip some sequential values that you know must exist.

### Determining the CPU Clock:

This project is running at the default of 100 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 chip\_5411x.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 100 MHz as shown in Watch 1.
10. Stop the CPU.  and exit Debug mode. 



**TIP:** To change the CPU frequency, use the Clocks Tool in MCUXpresso Config Tools.





### What else can we do ?

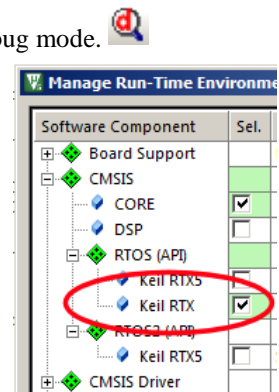
5. You can create new source files using the Add New Item window. See the top of the previous page.
6. You can add existing source files by right clicking on a Group name and selecting Add Existing Files.
7. You can easily add NXP example files to your project. You can use MCUXpresso to help you.
8. 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 !

### 3) 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.

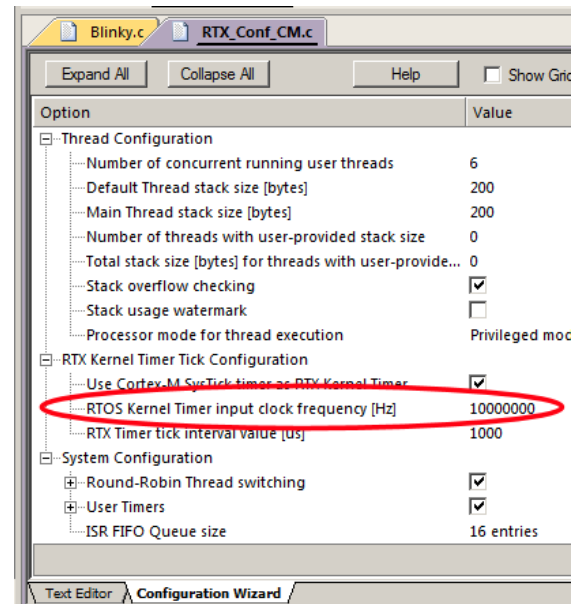
Configuring RTX is easy in MDK 5. These steps use the preceding Blinky example you constructed.

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 the elements as shown here: 
4. Select Keil RTX as shown and click OK. Do not select RTX5.
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". It 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 Wizard is displayed here: 
5. Select Use Cortex-M SysTick Timer as RTX Kernel Timer.
6. Set Timer clock: to 100000000 as shown: (100 MHz)
7. Use the defaults for the other settings.

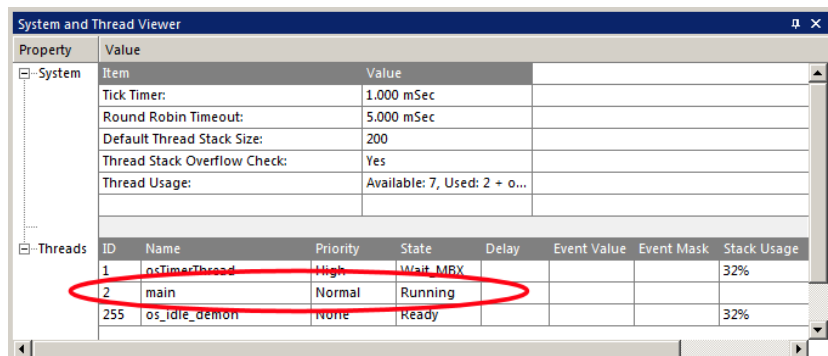


#### Build and Run Your RTX Program:

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 be automatically 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/](http://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, ULINKpro or a J-Link. Event Viewer uses SWV and is not yet supported by LPC-Link2 CMSIS-DAP. You can use LPC-Link2 in J-Link mode to use SWV. See page 9 for setup information.



**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](http://www.keil.com/support/docs/2735.htm) for instructions on how to add this feature to your own source code.

## 4) Adding a Thread:

We will create and activate a thread. We will add an additional global variable counter2 that will be incremented by this new thread we will call job1.

1. In Blinky.c, add this line near line 5:

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

### Create the Thread job1:

1. In Blinky.c, add this code before main():  
This will be the new thread named job1.


```
7 void job1 (void const *argument) {  
8     for (;;) {  
9         counter2++;  
10        if (counter2 > 0x0F) counter2 =  
11        0;  
12        osDelay(1500);  
13    }  
14 }
```

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

1. Add this line just after the `if (counter > . .` statement near line 25:

```
25 osDelay(1500);
```




### Define and Create the Thread:

1. Add this line near line 15 just before main():
2. Create the thread job1 near line 18 just after main() and before the while(1) loop:
3. Click on File/Save All or 

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

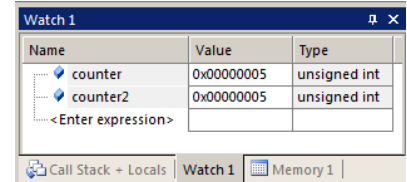
```
18 osThreadCreate(osThread(job1), NULL);
```

### Build and RUN the Program:

1. Define and Create the Thread Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.
2. Enter Debug mode:  Click on the RUN icon. 
3. Right click on counter2 in Blinky.c and select Add counter2 to ... and select Watch 1.
4. Both counter and counter2 will increment but slower than before:  
The two `osDelay(1500)` function calls each slow the program down by 1500 msec. This makes it easier to watch these two global variables increment.

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

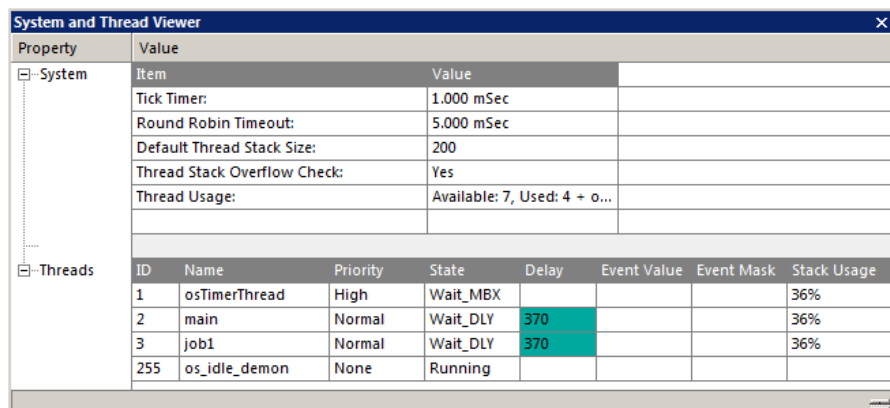
5. Open the System and Thread Viewer by selecting Debug/OS Support.
6. Note that job1 has now been added as a thread as shown below:
7. Note `os_idle_demon` is always labelled as Running. This is because the program spends most of its time here.
8. Set a breakpoint in thread job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
9. Set another breakpoint in the while(1) loop in main(). Each time you click RUN, the program will change threads.
10. 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`.
11. **Remove any breakpoints you have created.**



Name	Value	Type
counter	0x00000005	unsigned int
counter2	0x00000005	unsigned int
<Enter expression>		

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

RTX is part of the new CMSIS 5 See [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)



Property	Value
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 4 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				36%
2	main	Normal	Wait_DLY	370			36%
3	job1	Normal	Wait_DLY	370			36%
255	os_idle_demon	None	Running				







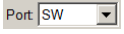

## 5) View RTX Thread Timing: Requires a Keil ULINK2/ME, ULINKpro or a Segger J-Link.

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). For this exercise, you must use any Keil ULINK or a J-Link. LPC-Link2 in CMSIS-DAP mode does not support SWV. LPC-Link2 in J-Link mode does. A ULINKpro is best for SWV.

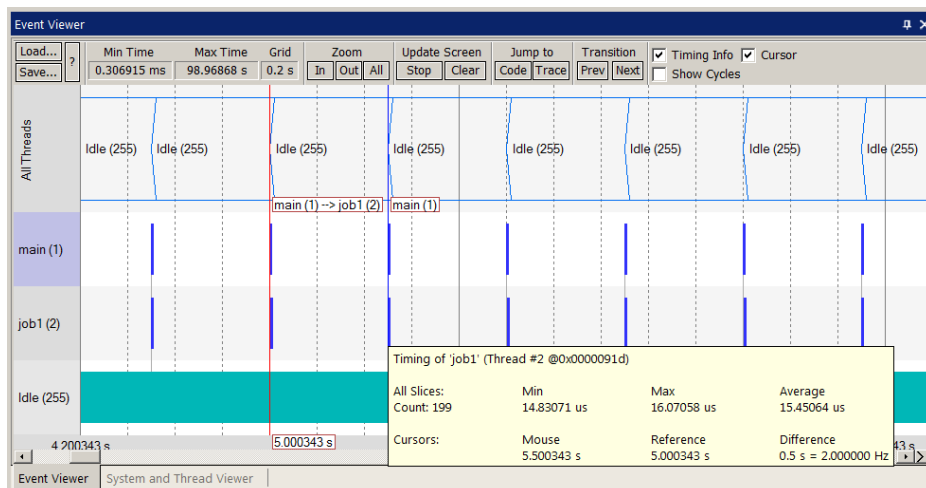
This page assumes you are running your Blinky with any ULINK or a J-Link connected to your Keil MCB54110 board.

### 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: .
4. If using a J-Link, enter JLINK\_SWO.ini in the Initialization File: box. Copy this file from the DSP example.
5. Click on Settings on the right side of the target Options window. The Cortex-M Target Driver Setup window opens.
6. Set SW as shown here:  JTAG does not support SWV. If your 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.**
7. Select the Trace tab. Select Core Clock: to 100 MHz. Select Trace Enable.
8. If using a ULINKpro, unselect Autodetect and enter 2 in SWO Clock Prescaler. 100 MHz is too fast for this SWO.
9. Unselect EXCTRC to minimize potential SWO overflows. With a ULINKpro you can leave EXCTRC set.
10. Click OK twice to close the Target configuration windows.
11. 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 will 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 available and correctly configured, the threads will be displayed.
5. Use the In and/or Out buttons to select an appropriate horizontal scale.



6. 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 example  $\mu$ Vision projects with MCUXpresso.**


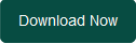
## 6) Using MCUXpresso : <http://mcuxpresso.nxp.com>

MCUXpresso is a utility provided by NXP to simplify creating and configuring NXP devices. A set of MDK 5 projects and other files are created for download. They can be loaded into  $\mu$ Vision, compiled and run or added to your own projects.

### Starting and Configuring MCUXpresso:

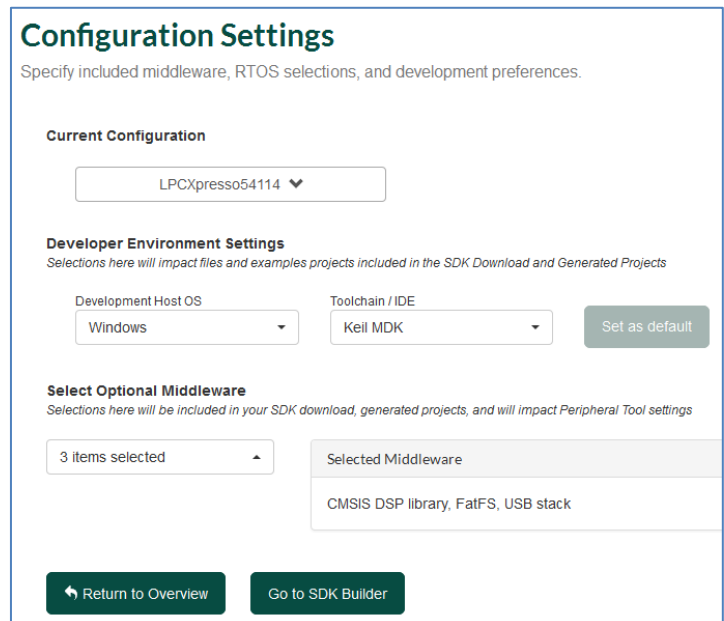
1. The URL for MCUXpresso is <http://mcuxpresso.nxp.com>
2. You will need to create an account and sign in:
3. Select the Config Settings:
4. The page below will open. Click Current Configuration and Select New Configuration:
5. Select the processor or board you are using: Name this configuration and click Select configuration:
6. Select Keil MDK and Windows as shown here:
7. Choose an RTOS or no RTOS as you require. You can add RTX later if you prefer.

### Building and Downloading the SDK:

1. Select Build an SDK to access SDK Builder.  

2. Select Download.  

3. Your custom SDK will be built and stored for you to download. An email will be sent to you.
4. In the menu bar at the top of the screen, select Manage to download your SDK: see this menu below:
5. Extract the SDK file into a suitable folder: I used C:\00MDK\Boards\Keil\MCB54110\SDK
6. You can explore and see the many other examples and useful code is present in the SDK.
7. You should also explore the Pins Tool and Clocks Tool.

### Notes:

1. Ensure View/Periodic Update is enabled. If not, your windows will update only when the processor is stopped.



**This is the End of the exercises !**

## 1) Serial Wire Viewer Summary:

We have several basic debug systems as implemented in NXP Cortex-M3, Cortex-M4 and Cortex-M7 processors:

1. SWV and ITM data output on the SWO pin located on the JTAG/SWD debug connector.
2. ITM is a printf type viewer. ASCII characters are displayed in the Debug printf Viewer in  $\mu$ Vision.
3. Memory Reads and Writes in/out the DAP SWD port. The Memory, Watch and Peripheral views use this.
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.

### What kind of data can the Serial Wire Viewer display ?

1. Global, Static variables, Structures and physical addresses.
2. Can see Peripheral registers – just read or write to them. Use Peripheral Viewer as it uses DAP reads and writes.
3. Can see some executed instructions. SWV only samples them.
4. CPU counters. Folded instructions, sleep, 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 CPU register operations. PC Samples records some of the instructions but not the data values.

SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU.

### Overloading the SWO pin:

The Serial Wire Output pin (SWO) is just that – a single bit high speed line that is expected to handle all the data sent to it. A Cortex-M3 can read and write data much faster than the SWO can process.

Overloads have an X in the Ovf or Dly (overflow or Delay) column in the Trace Records window: but not always. ITM frames with IDs other than 0 or 31 are a sure sign of either overloading or an incorrect Core Clock setting.

### Here are some hints on reducing traffic to minimize SWO overruns:

1. Reduce the number of variables in the Logic Analyzer.
2. Reduce the number of items selected in the Trace Configuration window: These include:
  - a. PC Sampling. Turn this off or increase the Prescaler to reduce the sampling rate.
  - b. On Data R/W Sample.
  - c. EXCTRC: Exception Tracing
  - d. Any Trace Events. (the Counters)
  - e. ITM Stimulus Ports 31 and 0. (Event Viewer and ITM printf respectively)
3. Set the Timestamp Prescaler to 1.
4. Turn the timestamps off. Sometimes this will turn the trace off though. But it is worth a try.
5. Lower the rate of reads, writes and exception calls in your software.
6. Turn off the SysTick timer in your software if you are not using it.
7. The General Rule is: activate only those SWV features you need.
8. Use a ULINK*pro*. It uses the SWO pin in Manchester mode which is faster than the UART mode used in ULINK2 or J-Link. It can also access SWV via the 4 bit ETM Trace Port (if equipped) and this is even faster.

**TIP:**  $\mu$ Vision recovers easily and painlessly from data overruns. You are made aware when frames are lost.

### Top Five Reasons why you can't get SWV working:

Symptoms can range from an inability to enter a known variable in the Logic Analyzer, corrupted or no trace frames.

1. Core Clock is wrong. This is the number 1 reason. Normally, the SWO speed is derived from the Core Clock speed. ULINK2 and J-Link need this set accurately. ULINK*pro* uses Core Clock: for timing display calculations.
2. View/Periodic Update is not enabled. Your windows update only when the processor is stopped.
3. Trace Enable is not checked. No needed initialization file is present. A few processors need an initialization file.
4. Using JTAG instead of SWD (SW in  $\mu$ Vision).  $\mu$ Vision will complain about this when you enable Trace.
5. You have selected too many SWV items and it is seriously overloaded.

## 2) Document Resources:

See [www.keil.com/NXP](http://www.keil.com/NXP)

### Books:

1. **NEW!** Getting Started with MDK 5: Obtain this free book here: [www.keil.com/gsg/](http://www.keil.com/gsg/)
2. There is a good selection of books available on ARM: [www.arm.com/support/resources/arm-books/index.php](http://www.arm.com/support/resources/arm-books/index.php)
3.  $\mu$ Vision 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](http://www.arm.com/products/processors/cortex-m/index.php)
5. Or search for the Cortex-M processor you want on [www.arm.com](http://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](http://www.keil.com/safety)
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions [www.keil.com/appnotes/files/apnt209.pdf](http://www.keil.com/appnotes/files/apnt209.pdf)
3. CAN Primer: [www.keil.com/appnotes/files/apnt\\_247.pdf](http://www.keil.com/appnotes/files/apnt_247.pdf)
4. Segger emWin GUIBuilder with  $\mu$ Vision™ [www.keil.com/appnotes/files/apnt\\_234.pdf](http://www.keil.com/appnotes/files/apnt_234.pdf)
5. Porting mbed Project to Keil MDK™ [www.keil.com/appnotes/docs/apnt\\_207.asp](http://www.keil.com/appnotes/docs/apnt_207.asp)
6. MDK-ARM™ Compiler Optimizations [www.keil.com/appnotes/docs/apnt\\_202.asp](http://www.keil.com/appnotes/docs/apnt_202.asp)
7. GNU tools (GCC) for use with  $\mu$ Vision <https://launchpad.net/gcc-arm-embedded>
8. RTX CMSIS-RTOS Download [www.keil.com/demo/eval/rtx.htm](http://www.keil.com/demo/eval/rtx.htm)
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](http://www.arm.com) and search for DAI0298A
11. Cortex Debug Connectors: [www.keil.com/coresight/coresight-connectors](http://www.keil.com/coresight/coresight-connectors)
12. Sending ITM printf to external Windows applications: [http://www.keil.com/appnotes/docs/apnt\\_240.asp](http://www.keil.com/appnotes/docs/apnt_240.asp)
13. FlexMemory configuration using MDK [www.keil.com/appnotes/files/apnt220.pdf](http://www.keil.com/appnotes/files/apnt220.pdf)
14. Sending ITM printf to external Windows applications: [www.keil.com/appnotes/docs/apnt\\_240.asp](http://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](http://www.keil.com/appnotes/docs/apnt_270.asp)
16. **NEW!** ARMv8-M Architecture Technical Overview <https://community.arm.com/docs/DOC-10896>
17. **NEW!** Using TrustZone on ARMv8-M with Keil MDK: [www.keil.com/appnotes/docs/apnt\\_291.asp](http://www.keil.com/appnotes/docs/apnt_291.asp)
18. **NEW!** Determining Cortex-M CPU Frequency using SWV [www.keil.com/appnotes/docs/apnt\\_297.asp](http://www.keil.com/appnotes/docs/apnt_297.asp)

### Useful ARM Websites:

1. CMSIS: [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5) [www.arm.com/cmsis/](http://www.arm.com/cmsis/) [www.keil.com/cmsis](http://www.keil.com/cmsis)
2. ARM and Keil Community Forums: [www.keil.com/forum](http://www.keil.com/forum) and <http://community.arm.com/groups/tools/content>
3. ARM Developer: <https://developer.arm.com/>
4. ARM University Program: [www.arm.com/university](http://www.arm.com/university). Email: [university@arm.com](mailto:university@arm.com)
5. mbed™: <http://mbed.org>

**Sales In Americas:** [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. **Europe/Asia:** [sales.intl@keil.com](mailto:sales.intl@keil.com) +49 89/456040-20  
**Global Inside Sales Contact Point:** [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com) **Keil Distributors:** [www.keil.com/distis/](http://www.keil.com/distis/)  
**Keil Technical Support in USA:** [support.us@keil.com](mailto:support.us@keil.com) or 800-348-8051. Outside the US: [support.intl@keil.com](mailto:support.intl@keil.com)

### 3) Keil Products and Contact Information:

See [www.keil.com/NXP](http://www.keil.com/NXP)

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

- **MDK-Lite™** (Evaluation version) 32K Code and Data Limit - \$0
- **New MDK-ARM-Essential™** 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](http://www.keil.com/mdk5/version520).

For the latest MDK details see: [www.keil.com/mdk5/selector/](http://www.keil.com/mdk5/selector/)

Keil Middleware includes Network, USB, Graphics and File System. [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/)

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

- **ULINK2** -(ULINK2 and ME - SWV only – no ETM) **ULINK-ME** – Electrically equal to ULINK2.
- **New ULINKplus-** Cortex-Mx High performance SWV & power measurement.
- **ULINKpro** - Cortex-Mx SWV & ETM trace.
- **ULINKpro D** – The same as a ULINKpro without ETM support. It can be used with µVision or DS-5.

For Serial Wire Viewer (SWV), a ULINK2 or a J-Link is needed. For ETM support, a ULINKpro is needed.

All ULINK products support MTB (Micro Trace Buffer) with NXP Cortex-M0+.

The Keil RTX RTOS is provided with a BSD or Apache 2.0 license. This makes it free. All source code is provided.

See [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5) and [www.keil.com/gsg](http://www.keil.com/gsg).  
RTX documentation is free. [www.keil.com/pack/doc/CMSIS/RTOS/html/](http://www.keil.com/pack/doc/CMSIS/RTOS/html/)  
For the feature list see: [www.keil.com/RTX](http://www.keil.com/RTX)

Keil provides free DSP libraries for the Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7.

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

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

Call Keil Sales for special university pricing. Go to [www.arm.com/university](http://www.arm.com/university)

Keil supports many other NXP processors including ARM7™ and ARM9™ series processors. See the Keil Device Database® on [www.keil.com/dd](http://www.keil.com/dd) for the complete list of NXP support. Also see [www.keil.com/NXP](http://www.keil.com/NXP)

NXP i.MX processors are supported by ARM DS-MDK™. [www.keil.com/ds-mdk](http://www.keil.com/ds-mdk) or DS-5: [www.arm.com/ds5](http://www.arm.com/ds5)



#### For more information:

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

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

Global Inside Sales Contact Point: [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)

Keil Distributors: [www.keil.com/distis/](http://www.keil.com/distis/)

Forums: [www.keil.com/forum](http://www.keil.com/forum) <http://community.arm.com/groups/tools/content> <https://developer.arm.com/>

