NXP LPC4300: Cortex®-M4 / M0 Hands-On Lab

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

For the Keil MCB4357™ EVAL board Version 1.0

Robert Boys bob.boys@arm.com

For the Keil MCB4300™ Evaluation Board with the LPC4357 or LPC4350

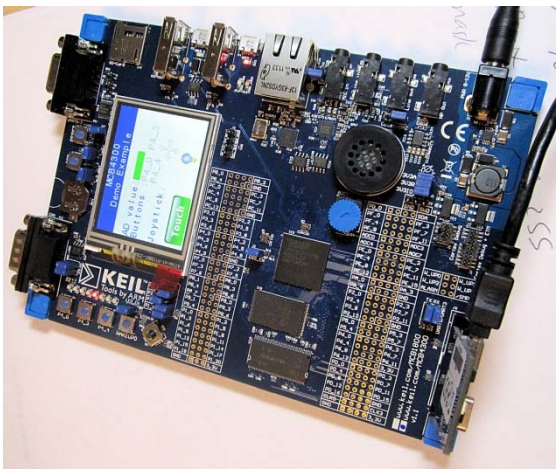
Introduction:

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_241.asp

The purpose of this lab is to introduce you to the NXP Cortex™-M4 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 MDK™.

Keil MDK comes in an evaluation version that limits code and data size to 32 Kbytes. Most Keil examples are under 32K. The addition of a license number will turn it into the full, unrestricted version. Contact Keil sales for a temporary full version license if you need to evaluate MDK with programs greater than 32K. MDK includes a full version of Keil RTX™ RTOS. RTX source code is now included with all versions of Keil MDK™ and now has a BSD license. See www.keil.com/nxp for more useful information concerning Keil support of NXP microprocessors.

Why Use Keil MDK ?



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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolkit. MDK is a turn-key product with included examples and is easy to get running.
2. Serial Wire Viewer and ETM trace capability is included.
3. A full feature Keil RTOS called RTX is included with MDK and includes source code with all versions of MDK.
4. RTX Kernel Awareness windows are updated in real-time.
5. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™, Segger J-Link and LPC-Link 2 (CMSIS-DAP).
6. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.

Other significant features shown in this document:

1. Serial Wire Viewer (SWV) with the ULINK2. ETM trace examples with the ULINK*pro* are shown starting on p 25.
2. Dual Core debugging control for the Cortex-M0 and the Cortex-M4 cores on the NXP LPC4350 processor.
3. Real-time Read and Write memory accesses for Watch, Memory and RTX Tasks windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
4. Up to six Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also called Access Breaks).
5. RTX and its two kernel awareness windows. These windows update while the program is running.
6. A DSP example using ARM DSP libraries. See www.arm.com/cmsis, www.keil.com/forum/ and forums.arm.com.

Serial Wire Viewer (SWV): *Use a ULINK2 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. This information comes from the ARM CoreSight™ debug module integrated into the Cortex-M4. SWV is output on the Serial Wire Output (SWO) pin found on the JTAG/SWD adapter connector. SWV is not available for the Cortex-M0 processor. SWV is a standard feature on NXP Cortex-M3 and Cortex-M4 processors.

SWV does not steal any CPU cycles and is completely non-intrusive except for the ITM Debug printf Viewer. SWV is provided by the Keil ULINK2, ULINK-ME, ULINK*pro* and Segger J-Link. Best results are with a ULINK.

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

ETM captures all executed instructions in addition to the data provided by SWV. This allows advanced debugging features including timing of areas of code (Execution Profiling), Code Coverage, Performance Analysis and program flow debugging and analysis. ETM requires a special debug adapter such as the ULINK*pro*.

1) NXP ARM Cortex Processor Evaluation Boards:

Keil supports and makes boards with NXP processors for ARM7TDMI, ARM9, Cortex-M0, Cortex-M0+, Cortex-M3 and Cortex-M4/M0. See www.keil.com/nxp for more information. MDK supports many other manufacturers of NXP boards.

Examples are provided with Keil MDK including Keil middleware (in RL directory). Nearly all examples do not require a licensed version of MDK but the \RL middleware requires a MDK Pro license. Contact Keil sales for a temporary license.

Labs similar to this one are available for the NXP Cortex-M3 (LPC1700), Cortex-M0+ (LPC800) and a version for CAN (Controller Area Network) using the MCB1700. See www.keil.com/nxp for more information. MCB1800 lab is planned.

The ULINK*pro* adds Cortex-M ETM trace support. It also adds faster programming time and better trace display. Most NXP Cortex-M3/M4 parts are equipped with ETM. All have SWV. The Cortex-M0 and M0+ have neither SWV nor ETM. The Cortex-M0+ (LPC800) has a MTB Micro Trace Buffer. It also has non-intrusive read/write to memory locations (for Watch and Memory windows), hardware breakpoints and Watchpoints.

Note: When debugging the Cortex-M0 and Cortex-M4 processors at the same time in the LPC4300 series, you must use JTAG rather than SWD (Serial Wire Debug). This means Serial Wire Viewer is not available for the Cortex-M4 processor. The SWO pins shares a pin with JTAG TDIO. A ULINK*pro* can avoid this conflict by sending SWV frames out the 4-bit Trace Port instead of the SWO pin.

The generic MCB4300 board series has currently one member:

MCB4357 which is populated with a LPC4357 processor. The MCB4357 has both internal and external flash memory.

MCB4350 contained a LPC4350 and this processor had no internal flash. It has been replaced with the MCB4357 board. You can use this lab with either board. The LPC4350 has more internal RAM than the LPC4357.

The MCB1800 is a close relative to the MCB4300 but with a Cortex-M3 processor from the LPC1800 family. You can use most of the examples in this lab, except for the dual CPU, with the MCB1800 with suitable modifications. See www.keil.com/nxp for a future MCB1800 lab.

Five Easy Steps to Get Connected and Configured:

1. Physically connect your debug adapter to the Keil board. Power both of these appropriately with USB cables.
2. Obtain and install Keil MDK Lite (evaluation version) on your PC. Use the default directory C:\Keil\.
3. Configure µVision to use a ULINK2, ULINK-ME or ULINK*pro* to communicate with the JTAG or SWD port.
4. Configure the Flash programmer inside µVision to program the internal or external flash or SPIFI memory.
5. If desired, configure the Serial Wire Viewer and/or ETM trace with a ULINK2 or a ULINK*pro* as appropriate.

Keil Software Installation:

This document was written using Keil MDK 4.17a which contains µVision 4. The evaluation copy of MDK (MDK-Lite) is available free on the Keil website. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence.

MDK 5.0 will be available late Summer 2013. See www.keil.com/mdk5 for information.

1. **MDK:** To obtain a copy of MDK go to www.keil.com/nxp and select the Download icon: 

   Please install MDK into the default location of C:\Keil. You can use the evaluation version of MDK and a ULINK2, ULINK-ME, ULINK*pro* or J-Link for this lab. You can use a LPC-Link 2 but the Serial Wire Viewer (SWV) is not currently supported. ETM is supported only with a ULINK*pro* which has the hardware necessary to connect to the 4 bit trace port.
   The addition of a valid Keil license number converts the evaluation into a full commercial copy of MDK.

2. **Example Programs:** MDK 4.71a and later contains all example programs except for the DSP example. This can be found with this document at www.keil.com/appnotes/docs/apnt_241.asp. Please extract the DSP zip file to C:\Keil\ARM\Boards\Keil\MCB4300\. This will create a directory \DSP containing the example DSP project. Version 5.0 of the file system_LPC43xx.c is provided in the zip directory \system. This provides 120 MHz operation instead of the usual 180 MHz as used in the other Keil examples. This version is more stable with MCB4300 external memory than the one found in the Blinky_Ulp project in MDK 4.71a. MDK 4.72 will include this later version.

2) Debug Adapter Summary for use with µVision IDE:

ULINK2

ULINK2:

This is a hardware JTAG/SWD debugger. It connects to various connectors found on boards populated with ARM processors. With NXP Cortex-M3 and M4 processors, ULINK2 supports Serial Wire Viewer (SWV) and MTB (Micro Trace Buffer) with Cortex-M0+.

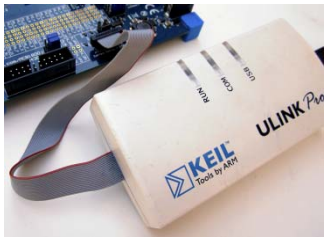
ULINK-ME:

ULINK-ME is provided combined with a board package from Keil or a OEM. Electrically and functionally it is very similar to a ULINK2. With Keil µVision, they are used as equivalents. ULINK-ME has the both a legacy 20 pin and the new 10 Cortex Debug connector.

ULINK-ME

ULINK*pro*:

ULINK*pro* is Keil's most advanced debug adapter. With NXP Cortex-M3 and Cortex-M4 processors, ULINK*pro* provides Serial Wire Viewer (SWV) and adds ETM Instruction Trace. Code Coverage, Performance Analysis and Execution Profiling are then provided using ETM. ULINK*pro* programs the Flash memories very fast.



ULINK*pro*

LPC-Link 2:

A CMSIS-DAP compliant debug adapter. This provides either a J-Link Lite or a CMSIS-DAP connection. µVision supports both. LPC-Link 2 supports MTB trace on the LPC800. Serial Wire Viewer (SWV) will be supported late 2013.

The LPC-Link 2 can start and stop the CPU and set/unset breakpoints. Watchpoints function. The Watch and Memory windows update in real-time.

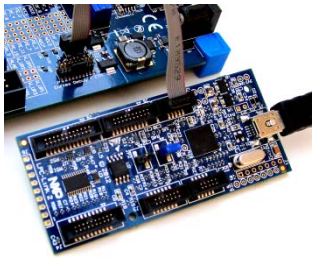
CMSIS-DAP:

This is a new ARM standard where a small processor located on the target board acts as the Debug Adapter. It connects to your PC with a standard USB cable.

The processor can also be external as in the NXP LPC-Link 2.

CMSIS-DAP provides run control debugging, Flash and RAM programming, Watchpoints and hardware breakpoints. DAP reads and writes in the Watch and Memory windows are updated in real-time as well as the RTX System kernel awareness.

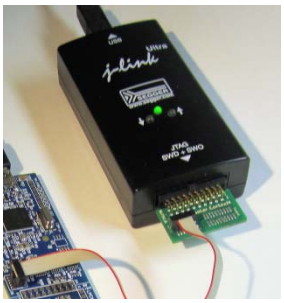
MTB is supported with the LPC800 Cortex-M0+. You are able to easily incorporate a CMSIS-DAP design on your own custom target boards. For documents go to silver.arm.com/browse/CMSISDAP and download CMSIS-DAP Beta 0.01. Also see www.arm.com/cmsis and forums.arm.com.



LPC-Link 2

Segger J-Link:

µVision supports J-link and J-Link Ultra (black case only) Version 6.0 or later. The J-Link family supports all CoreSight components except MTB. This will be supported in the future. J-Link is configured similar to the ULINK2/ME. Select J-Link/J-Trace Cortex in the Debug and Utilities tab. The Trace windows are slightly different from the ULINK2/ME.

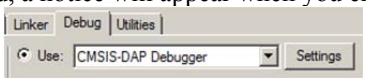


J-Link Ultra

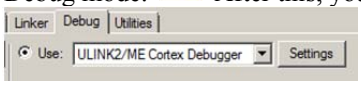
NEW ! ULINK2 and ULINK-ME with CMSIS-DAP Support:

Starting with MDK 4.71, you can use either of these as a CMSIS-DAP compliant debugger or in standard ULINK2/ME mode. With a LPC800, CMSIS-DAP mode adds trace search, save and clear window options. This is the same window as with the ULINK*pro*. ULINK2 CMSIS-DAP can start and stop the CPU and set/unset breakpoints. Watchpoints function. The Watch and Memory windows update in real-time. SWV does not function in CMSIS-DAP mode at this time. If you want to use SWV, choose a standard ULINK2/ME, ULINKpro or J-Link.

If the ULINK2/ME firmware needs to be updated, a notice will appear when you enter Debug mode. After this, you can select either mode in the Target Options menus:



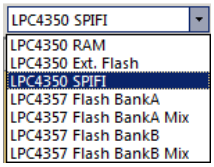
or



3) MCB4300 Flash Memory Programming:

µVision contains projects to run your programs in internal RAM, external Flash (EMC), SPIFI Flash and internal Flash.

These are selected with the target Options Selector as shown here:

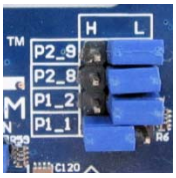


The various projects contain the configuration elements necessary to program the various memories. You can easily create new Target Options to your specific needs and target boards.

Flash Programming Algorithms: MDK contains all Flash programming algorithms for external and internal Flash memory: The LPC4357 has 136KB and the MCB4350 has 264KB of internal RAM respectively. µVision has examples to run your program in any of the Flash memories plus the internal RAM.

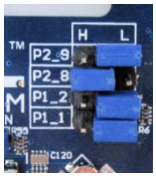
Boot Addresses: Four jumpers P1_1, P1_2, P2_8 and P2_9 are used to determine from where the user program is run from. We will use external Flash (EMC) in our examples. The settings of these jumpers are shown below:

SPIFI setting:



EMC setting:

EMC = External Memory Controller



Start Address: The LPC4357 will start executing from its internal Flash if a viable program exists in it. To run from a different Flash device, either a) erase the internal Flash by selecting the correct Target Selector and then select Flash/Erase from the main µVision menu or b) configure the jumpers above and simultaneously press the MCB4300 RESET and ISR buttons and first release RESET first and then ISR. This forces the jumper settings to override any internal Flash program.

Target Options for Various Memory Devices:

There are several places you might need to enter configuration items. Check the examples provided with MDK to help you with specific configurations. These are found in Target Options under various tabs. Select the appropriate memory you are trying to use in the Target Selector box as shown above. You can use these examples as templates for your own targets.

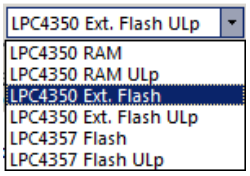
.ini Initialization files: There are two places these might be needed: the Debug tab and the Utilities tab. In the Debug tab, this ASCII file is executed when entering Debug mode. You can examine this file by clicking on the associated Edit icon. Usually these files are used to configure hardware items like GPIO ports for external memory or for the Trace Port.

Memory Addresses: Located in the Target tab. The addresses of the Flash and RAM are entered here. The left side is where the program runs (in RAM or ROM) and the right side is RAM addresses for variables and other volatile storage.

Flash programming Algorithms: Located in the Utilities tab and Settings selection. The algorithms, with source if available, are located in C:\Keil\ARM\Flash.

4) Configuring µVision for various debug adapters:

Most MCB4300 examples are pre-configured to use the ULINK2 or ULINK-ME. A few include the ULINK*pro*. These are selected in the Target Selector box as shown here: The Ulp suffix indicates support for ULINK*pro*. The rest are for the ULINK2/ME. This is also where you select the various memory options you can use. It is easy to add options as described here:



Making Your Own Target Selection:

1) Select the Target Selection in the boxes above that you want to be the template for your new version.
2) Select Project/Manage and then choose Components, Environments, Books…
3) In the left pane, select the New icon or press your PC Insert key.
4) Enter the name you choose for your new target option and click OK. Your new entry will now be visible in the Target Selector box. Select this entry. Now, you will modify your new target selection entry.
5) Select Target Options or ALT-F7. Make your modifications in the windows that open.
6) When completed, click OK to close the Target Options. Select File/Save All.

Configuring Various Debug Adapters starting on page 37 includes instructions for configuring LPC-Link 2, ULINK2, ULINK-ME and ULINK*pro*.

5) JTAG and SWD Definitions:

It is useful to have an understanding of these terms:

Note: NXP Cortex-M3 and Cortex-M4 have most features except MTB. Consult your device datasheet for specifics.

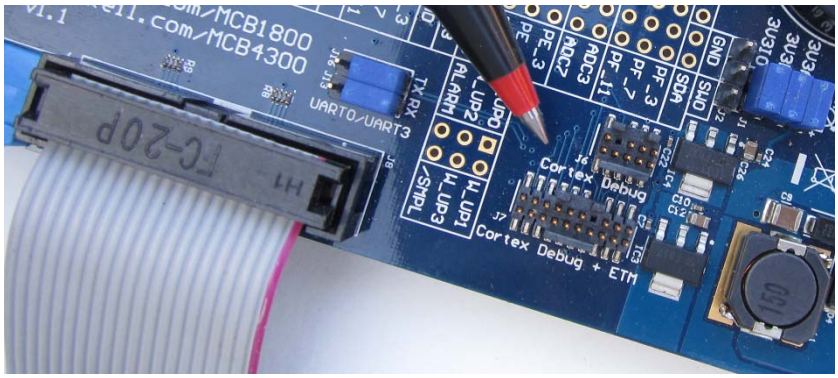
- **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
- **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except for no Boundary Scan. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup. The SWJ box must be selected. LPC800 Cortex-M0+ processors use SWD exclusively. There is no JTAG on the LPC800.
- **SWV:** Serial Wire Viewer: A data trace capability providing display of reads, writes, exceptions, PC Samples, ITM printf and Counters. SWV must use SWD because of the TDO conflict described in **SWO** below.
- **DAP:** Debug Access Port. A component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. µVision uses the DAP to update Memory, Watch and a RTOS kernel awareness window in real-time while the processor is running. You can also modify variable values on the fly with the Memory window. No CPU cycles are used, the program can be running and no code stubs are needed. CMSIS-DAP uses this port. The LPC-Link 2 debug adapter is CMSIS-DAP compliant. You do not need to configure or activate DAP. µVision does this automatically when you select the function.
- **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
- **Trace Port:** A 4 bit port that ULINK*pro* uses to collect ETM frames and optionally SWV (rather than SWO pin).
- **ETM:** Embedded Trace Macrocell: Captures all the executed instructions. Only ULINK*pro* provides ETM.
- **ETB:** Embedded Trace Buffer: a small dedicated on chip RAM (4 to 8 Kbytes) accessible with a debug adapter. Currently, only the ULINK*pro* can access the ETB. It is useful for instruction trace at the highest CPU speeds.
- **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used as an instruction trace buffer. MTB is only on LPC800 Cortex-M0+ processors. Most NXP Cortex-M3 and Cortex-M4 processors provide ETM trace.

6) Debug Connectors:

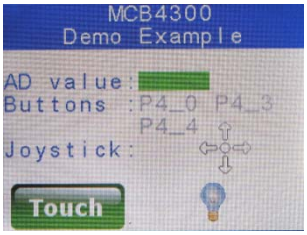
All three connectors listed can be used for JTAG and SWD debugging. All include SWV using the SWO pin. These pins have the same JTAG/SWD connections. The 20 pin Cortex Debug + ETM provides access to the 4 bit Trace Port.

1. **JTAG connector:** J8: The legacy 20 pin JTAG connector is provided. This provides JTAG, SWD and SWO access. It is the large connector labeled JTAG: The ULINK2 is connected to this connector in the photo below:
2. **Cortex Debug:** J6: A 10 pin connector providing JTAG, SWD and SWO signals. The connector J6 mounted on MCB4350 is Samtec FTSH-105-01-L-DV-K.
3. **Cortex Debug + ETM:** J7: this 20 pin connector is for use with the Keil ULINK*pro* ETM trace adapter. It provides JTAG, SWD, SWO and ETM signals. J7 part number is Samtec FTSH-110-01-L-DV-K.

TIP: Note pin 7 on both of the Cortex connectors are blank. This is part of the ARM standard that provides polarization. The Samtec connectors used on Keil boards also provide this effect with a polarizing key on the connector shroud.



Demo Example Program:

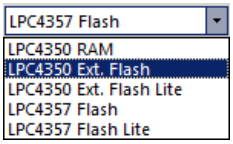


The MCB4300 boards come pre-programmed from the factory with the Demo example program. It starts upon powering the MCB4300 with a USB cable to P2 or P4. It will run stand-alone. No debug adapter is needed. You can exercise various features of Demo by following the instructions on the LCD screen.
The Demo project, with source code, is included with MDK:
C:\Keil\ARM\Boards\Keil\MCB4300\Demo\Demo.uvproj.

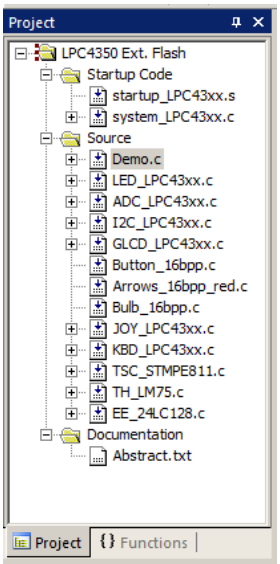
Notes concerning Demo.uvproj:

1) Demo compiles over the 32 KB code and data limit restriction of MDK-Lite. You are not able to compile Demo.uvproj with MDK Lite (eval version). You can program the provided Demo.axf (the executable) to Flash memory and run it. A copy of the executable, Demo.axf is included for the various options described below. **Note:** If you attempt to compile Demo.uvproj with MDK Lite, Demo.axf will be deleted as part of the normal build process. If you have a successful build using a license, a new Demo.axf will be created. The exception is if you are compiling the special project for Flash Lite. This uses a smaller LCD bit-map to come under the 32K limit. All of the other examples compile less than 32 KB. **TIP:** Save Demo.axf somewhere in the event it is erased. To restore it otherwise, you will need to re-install MDK.

2) There are five types of memory that Demo can be loaded into. These are selected in the Target Options Select box as shown here: Abstract.txt also describes these options.



   a) **LPC4350 RAM:** runs Demo in internal RAM. Only on LPC4350 – not on LPC4357.
   b) **LPC 4350 Ext. Flash:** runs Demo in the LPC4350 and LPC4357 external flash devices.
   c) **LPC 4350 Ext. Flash Lite:** A smaller LCD bitmap to compile less than 32KB limit in the external flash devices.
   d) **LPC 4357 Flash:** runs Demo in the LPC4357 internal flash. The LPC4350 does not have any internal Flash.
   e) **LPC 4357 Flash Lite:** A smaller LCD bitmap file to compile less than 32KB limit in the internal flash memory is provided. The Touch button look is different. This enables MDK Lite to be used to compile Demo.uvproj.



TIP: These are sets of all the settings under the Target Options.

Examining the Demo Source Code with µVision:

1. Start µVision by clicking on its icon:
2. Click Project/Open Project and select:
   C:\Keil\ARM\Boards\Keil\MCB4300\Demo\Demo.uvproj.
3. In the Project tab in the Project window you can see the various source files. Double-click on any of them to display them in the Source window and then select their tabs:
4. Startup_LPC43xx.s contains initialization code. system_LPC43xx.c contains code to configure the processor. Both are CMSIS compliant. Demo.c contains main().

Running the Demo Example with uVision: Demo is pre-configured only for a ULINK2:

1. Connect the ULINK2 to the JTAG connector. If using a different probe, configure it.
5. Connect the ULINK2 to your PC with a standard USB cable and power the board with a USB cable to your PC.
6. Select the appropriate Target Selector box above. Select the memory jumpers on page 6 if using external memory.
7. Program the selected memory by clicking on the Load icon: The pre-compiled version of Demo.axf is used.
8. Click on Debug mode .
9. Click on the RUN icon. If there are no errors, Demo will now run on the Keil board.
10. You can stop the program with the STOP icon.

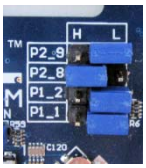
Demo.uvproj contains many useful programming techniques you can use in your own program. In succeeding examples you will learn how to use various ARM CoreSight debugging tools to examine the system your program is running in.

TIP: If there is an error when entering Debug mode, the probable cause is that Demo.axf has inadvertently been erased.

Blinky Example Program using a ULINK2 or ULINK-ME:

We will connect a Keil MDK development system using the Keil MCB4300 board and a ULINK2. We will be using the external flash for this part of the demonstration.

1. Connect the ULINK2 as pictured here: Connect the ULINK2 to your PC with a standard USB cable. Power the MCB4300 with a USB cable from P2 or P4 to your PC.
2. If you are using a debug adapter other than a ULINK2, you will have to configure it. You can use a ULINK2/ME, ULINK*pro*, LPC-Link 2 or a J-Link. See configuration instructions starting on page 37.
3. Confirm the jumpers are in EMC mode as shown here: This is the default position that the board comes in.



EMC means **E**xternal **M**emory **C**ontroller.

4. Start µVision by clicking on its icon. 
5. Click Project/Open Project and select:

   C:\Keil\ARM\Boards\Keil\MCB4300\Blinky\Blinky.uvproj.

6. Make sure "LPC4350 Ext. Flash" is selected as shown here: 

   This is where you can create and select different target configurations such as to execute a program in RAM or Flash. If using RAM, you then omit the Load step in number 9. For now, just use external Flash as indicated.
7. This project is configured by default to use the ULINK2.
8. Compile the source files by clicking on the Rebuild icon.  . Progress is indicated in the Build Output window.
9. Program the external flash by clicking on the Load icon: 

   Progress is indicated in the Build Output window.
10. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
11. Click on the RUN icon. 

The eight LEDs located beneath the Keil logo on the MCB4357 board will now blink in sequence:

There is no display on the LCD with this Blinky. RTX_Blinky uses the LCD.

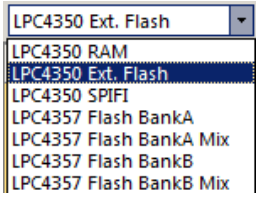
1. You stop the program with the STOP icon. 
2. You can single-step the program by clicking on the Step icon:  or F11.

TIP: 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 assembly instruction at a time.

3. Use Step Out  and then Step to single step in other functions. This simple example does not have many functions to test.
4. Click on the RUN icon  for the breakpoint example on the next page.

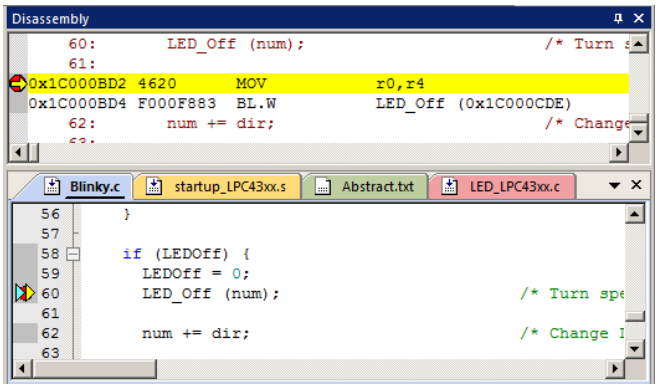
Now you know how to compile a program, program it into the external flash and start it, stop it and single-step it !

TIP: You can run Blinky in other memories by selecting it and re-building your project.
You might have to change the memory jumpers between SPIFI and Ext. Flash. For now, go to the next page to experiment with hardware breakpoints.



1) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block on line 55 LED_On (num) or line 60 LED_Off (num) in Blinky.c as shown below: Click on the Blinky.c tab if not visible. Gray means assembly is located there.
2. A red circle is created and soon the program will stop at this point.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows. This is the next instruction to execute.
4. The cyan arrow is a mouse selected pointer and is associated with the yellow band in the disassembly window. Click on a line in one window and this place will be highlighted in the other window.
5. Note you can set and unset hardware breakpoints while the program is running. ARM CoreSight debugging technology does this. There is no need to stop the program for many CoreSight functions including breakpoints.



6. Leave this breakpoint for the Call Stack example below.

TIP: The LPC4350 has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to.

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.

The 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

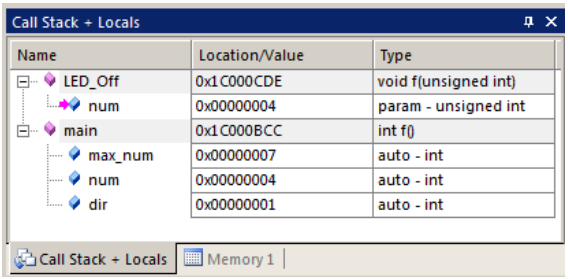
2) Call Stack + Locals Window:

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. Click on the Call Stack + Locals tab.
2. Click on Step or F11 until you enter the function LED_Off (or LED_On as you chose) as shown below:
3. Shown is the Call Stack + Locals window with the hardware breakpoint active on the function LED_Off (num).
4. The contents of the local variables max_num, num and dir are displayed as well as the function names.
5. Click on the Step Out icon to return to main().
6. With the breakpoint set as above in the breakpoint example, as you click on RUN, these locals will update as appropriate.



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. Update while the program is running is not possible with local variables. They must be converted to global or static variables so they always remain in scope and not deallocated.

Call Stack:

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.

7. **Remove the hardware breakpoint by clicking on its red circle or Ctrl-B, then select Kill All and then Close !**

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

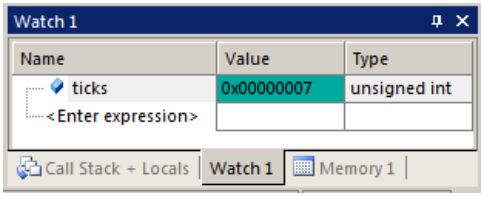
The Watch and Memory windows will display updated variable values in real-time. It does this using ARM CoreSight debugging technology. It is also possible to "put" or insert values into these memory locations in real-time using the Memory window. It is possible to enter variables or physical addresses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

Watch window:

1. We will display in real time the static variable ticks in the Watch window.
2. Start the processor if stopped. You can select Watch and Memory window variables while the program is running.
3. Find the static variable ticks in Blinky.c. This is declared in the function SysTick_Handler near line 28.
4. Right click on ticks and select Add 'ticks' to… and select Watch 1. ticks will be displayed as <cannot evaluate>. This is because ticks is a static variable and must be entered as fully qualified. In this case it would be \Blinky\Systick_Handler\ticks. You can also select it from the Symbol table(View/Symbols) or when it is in scope.
5. Set a breakpoint in SysTick_Handler: The program will stop and ticks will now display a valid value a shown here:

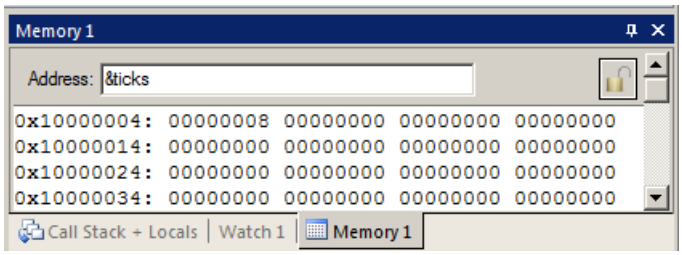
TIP: You do not need to stop the program or enter them qualified for global variables, raw addresses or structures.

6. Remove the breakpoint and click on RUN. ticks will now update.
7. You can modify the value in a Watch window when the program is stopped. You can modify a variable in a Memory window while the program is running. Step 5 below illustrates this technique.



Memory window:

1. Right click on ticks and select Add 'ticks' to… and select Memory 1. You can do this while the program is running.
2. Note the changing value of ticks is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to: but this not what we want to see at this time.
   Note: right click in Memory 1 and select Unsigned Long to see the addresses as 32-bit numbers.
3. Add an ampersand "&" in front of the variable name ticks and press Enter. Now the physical address is shown (0x1000_0004) in this case. This physical address could change with different compilation optimizations.
4. The data contents of ticks is displayed as shown here:
5. Right click on the memory value and select Modify Memory. Enter a value and this will be pushed into ticks. This variable is updated often so you might not be able to see this occurrence.



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.

1. The static variable ticks is updated in real-time. This is ARM CoreSight technology working.
2. Stop the CPU and exit debug mode for the next step. and

TIP: If you do not see memory locations in the Memory window, stop in the SysTick_Handler as before or fully qualify it.

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 and M4 are both 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.

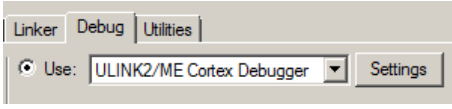
Remember you are not able to view local variables while the program is running. They are visible only when the program is stopped in their respective functions. You must change them to a different type of variable such as global to see them update.

4) Configuring the Serial Wire Viewer (SWV) with the ULINK2:

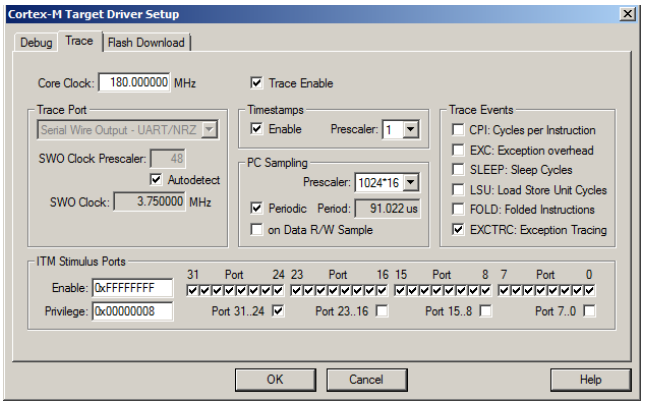
Serial Wire Viewer provides program information in real-time and is extremely useful in debugging programs.

SWV is available with any ULINK2, ULINK-ME, ULINK*pro* or a J-Link. LPC-Link 2 does not provide SWV at this time.

Configure SWV:

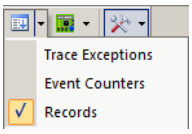


1. µVision must be stopped and in edit mode (not debug mode).
2. Select Target Options or ALT-F7 and select the Debug tab.
3. Click on Settings: beside the name of your adapter (in this case ULINK2/ME Cortex Debugger).
4. Select the SWJ box and select SW in the Port: pulldown menu.
5. In the area **SW Device** must be displayed: ARM CoreSight SW-DP. SWV will not work with JTAG.
6. Click on the Trace tab. The window here is displayed:
7. In Core Clock: enter 180 and select Trace Enable.
8. Select Periodic and leave everything else at default. Periodic activates PC Samples.
9. Click on OK twice to return to the main µVision menu. SWV is now configured.
10. Select File/Save All.



To Display Trace Records:

1. Enter Debug mode.
2. Click on the RUN icon.
3. Open Trace Records window by clicking on the small arrow beside the Trace icon shown here: You can also open the Trace Records window by selecting View/Trace/Records.
4. The Trace Records window will open and display PC Samples as shown below:



Displayed are PC samples and Exception 15 (SYSTICK timer) Entry, Exit and Return points.

Entry: when the exception or interrupt is entered.

Exit: when the exception or interrupt exits.

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

TIP: If you do not see PC Samples and 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 numbers are bogus and usually indicate a wrong Core Clock value.

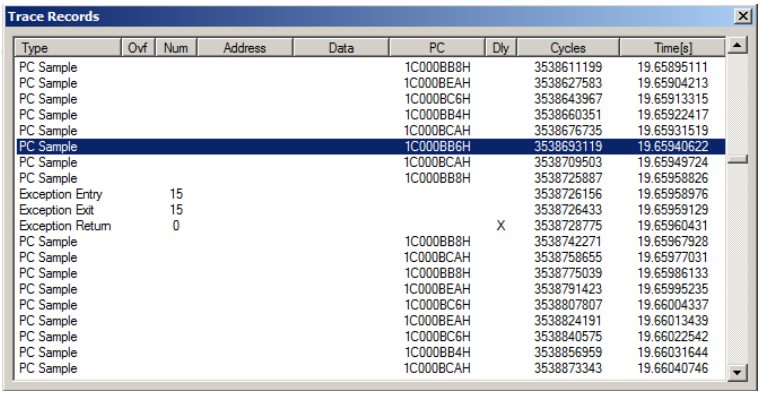
All frames have a timestamp displayed in CPU cycles and accumulated time.

Double-click this window to clear it.

If you right click inside this window you can see how to filter various types of frames out. Unselect PC Samples and you will see only exception frames.

Did you know Exception 15 is being activated ? Now you do. This is a very useful tool for displaying how many times an exception is firing and when. You can open up the Exceptions window the same way you opened the Trace Records window. Exceptions are listed with various timings. Both windows are updated in real-time.

Close both Trace windows when done with them.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					1C000BB8H		3538611199	19.65895111
PC Sample					1C000BEAH		3538627583	19.65904213
PC Sample					1C000BC6H		3538643967	19.65913315
PC Sample					1C000BB4H		3538660351	19.65922417
PC Sample					1C000BCAH		3538676735	19.65931519
PC Sample					1C000BB6H		3538693119	19.65940622
PC Sample					1C000BCAH		3538709503	19.65949724
PC Sample					1C000BB8H		3538725887	19.65958826
Exception Entry		15					3538726156	19.65958976
Exception Exit		15					3538726433	19.65959129
Exception Return		0				X	3538728775	19.65960431
PC Sample					1C000BB8H		3538742271	19.65967928
PC Sample					1C000BCAH		3538758655	19.65977031
PC Sample					1C000BB8H		3538775039	19.65986133
PC Sample					1C000BEAH		3538791423	19.65995235
PC Sample					1C000BC6H		3538807807	19.66004337
PC Sample					1C000BEAH		3538824191	19.66013439
PC Sample					1C000BC6H		3538840575	19.66022542
PC Sample					1C000BB4H		3538856959	19.66031644
PC Sample					1C000BCAH		3538873343	19.66040746

TIP: SWV is easily overloaded as indicated by an "x" in the OVF or Dly column. Select only that information needed to reduce overloading. There are more useful features of Serial Wire Viewer as we shall soon discover.

5) Using the Logic Analyzer (LA) with the ULINK2:

This example will use the ULINK2 with the Blinky example. It is assumed a ULINK2 is connected to your Keil board and configured for SWV trace as described on the previous page.

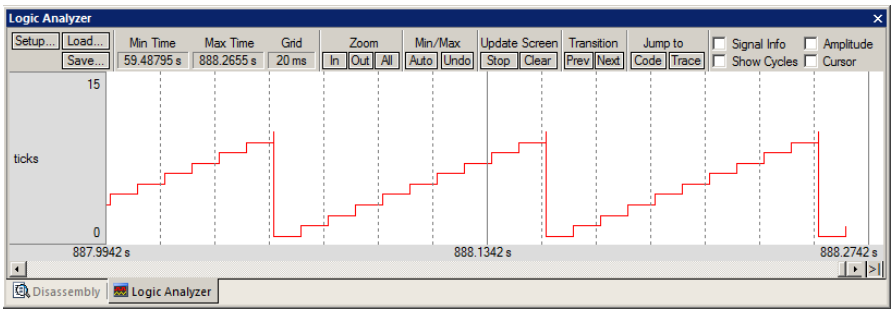
µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using the Serial Wire Viewer. The Serial Wire Output pin is easily overloaded with many data reads and/or writes and data can be lost. The LA shares the address comparators in CoreSight with the Watch windows. They are mutually exclusive.

1. The project Blinky.uvproj should still be open and is still in Debug mode and running.
2. **Note:** You can configure the LA while the program is running or stopped.
3. Select Debug/Debug Settings and select the Trace tab.
4. Unselect Periodic and EXCTRC. This is to prevent overload on the SWO pin. Click OK twice.
5. Click on RUN  to start the program again.
6. Locate the variable ticks in Blinky.c. Set a breakpoint at this point to stop the processor at this point. When it stops, remove the breakpoint. This allows you to select a variable that ordinarily needs to be fully qualified.
7. Right click on ticks and select Add 'ticks' to … and select Logic Analyzer. This will open the LA window.

TIP: An error message saying ticks cannot be added usually means SWV is not configured or ticks is not in focus.

TIP: You can also open the LA and select Setup and then select the New icon and enter \Blinky\SysTick_handler\ticks.

8. Click on Setup and set Max: in Display Range to 0xF. Click on Close. The LA is completely configured now.
9. ticks should still be visible in Watch 1. If not, enter it into the Watch 1 window.
10. Adjust the Zoom OUT or the All icon in the LA to provide a suitable scale of about 20 ms as shown here:

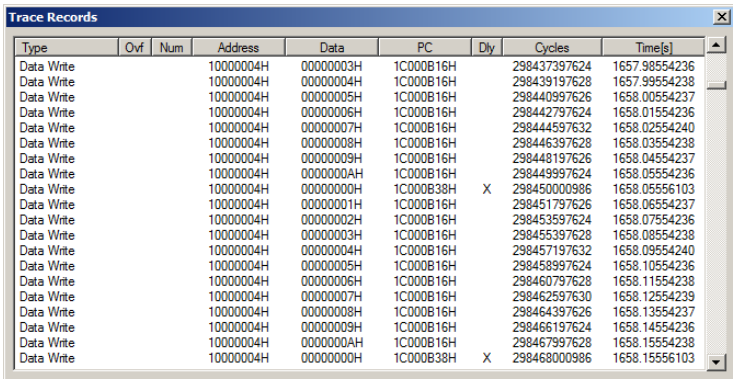


11. Would you have guess ticks is a sawtooth wave from looking at its value changing in the Watch window ? Select Amplitude and use the cursor to see when ticks = 0xA. Select Stop in Update Screen to stop the LA from collecting data.

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).

When you enter a variable in the Logic Analyzer window, it will also be displayed in the Trace Records window.

1. Select Debug/Debug Settings and select the Trace tab.
2. Select on Data R/W Sample. Click OK twice.
3. Run the program.  .
4. Open the Trace Records window and clear it by double clicking in it.
5. The window similar below opens up:
6. The first line says:
   The instruction at 0x1C00_0B16 caused a write of data 0x03 to RAM memory address 0x1000_0004 at the listed time in CPU Cycles or accumulated Time in seconds.



Trace Records

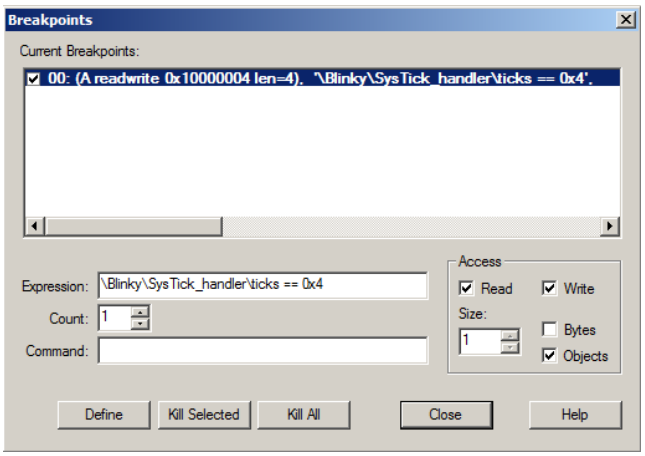
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000004H	00000003H	1C000B16H		298437397624	1657.98554236
Data Write			10000004H	00000004H	1C000B16H		298439197628	1657.99554238
Data Write			10000004H	00000005H	1C000B16H		298440997626	1658.00554237
Data Write			10000004H	00000006H	1C000B16H		298442797624	1658.01554236
Data Write			10000004H	00000007H	1C000B16H		298444597632	1658.02554240
Data Write			10000004H	00000008H	1C000B16H		298446397628	1658.03554238
Data Write			10000004H	00000009H	1C000B16H		298448197626	1658.04554237
Data Write			10000004H	0000000AH	1C000B16H		298449997624	1658.05554236
Data Write			10000004H	00000000H	1C000B38H	X	298450000986	1658.05556103
Data Write			10000004H	00000001H	1C000B16H		298451797626	1658.06554237
Data Write			10000004H	00000002H	1C000B16H		298453597624	1658.07554236
Data Write			10000004H	00000003H	1C000B16H		298455397628	1658.08554238
Data Write			10000004H	00000004H	1C000B16H		298457197632	1658.09554240
Data Write			10000004H	00000005H	1C000B16H		298458997624	1658.10554236
Data Write			10000004H	00000006H	1C000B16H		298460797628	1658.11554238
Data Write			10000004H	00000007H	1C000B16H		298462597630	1658.12554239
Data Write			10000004H	00000008H	1C000B16H		298464397626	1658.13554237
Data Write			10000004H	00000009H	1C000B16H		298466197624	1658.14554236
Data Write			10000004H	0000000AH	1C000B16H		298467997628	1658.15554238
Data Write			10000004H	00000000H	1C000B38H	X	298468000986	1658.15556103

TIP: The PC column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin if there are too many overruns. µVision and CoreSight recover gracefully from trace overruns.

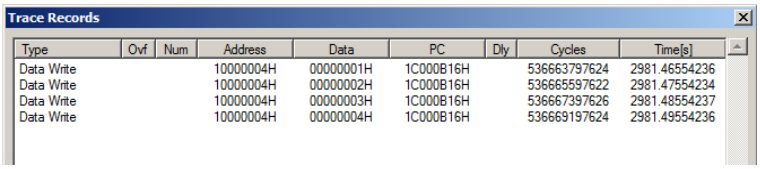
6) Watchpoints: Conditional Breakpoints

Most NXP Cortex-M3 and M4 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 watchpoints in its operations. This means in µVision you must have two variables free in the Logic Analyzer to use Watchpoints. The Cortex-M0 does not have any watchpoints. The LPC800 Cortex-M0+ has two of them.

1. Using the example from the previous page, stop the program. Stay in Debug mode.
2. Enter the global variable ticks into the Watch 1 window if it is not already there.
3. Click on Debug and select Breakpoints or press Ctrl-B.
4. The SWV Trace does not need to be configured to use Watchpoints. However, we will use it in this exercise.
5. Enter in Expression: "`\Blinky\SysTick_handler\ticks == 0x4`" without the quotes. Select both the Read and Write Access. If the program is stopped in SysTick_handler or ticks is a global variable: "enter ticks == 0x4".
6. Click on Define and it will be accepted as shown here: (the Expression: box will actually go blank)

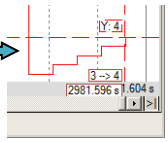


7. Click on Close.
8. Double-click in the Trace Records window to clear it.
9. ticks should still be entered in the Logic Analyzer window from the previous exercises.
10. Click on RUN.
11. When **`ticks`** equals 0x4, the program will stop. This is how a Watchpoint works.
12. You will see **`ticks`** displayed as 0x4 in the Logic Analyzer as well as in the Watch window.
13. Note the data write of 0x4 in the Trace Records window shown below in the Data column. The address the data written to and the PC of the write instruction is displayed as well as the timestamps:



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000004H	00000001H	1C000B16H		536663797624	2981.46554236
Data Write			10000004H	00000002H	1C000B16H		536665597622	2981.47554234
Data Write			10000004H	00000003H	1C000B16H		536667397626	2981.48554237
Data Write			10000004H	00000004H	1C000B16H		536669197624	2981.49554236

14. There are other types of expressions you can enter and they are detailed in the Help button in the Breakpoints window.
15. To repeat this exercise, enter a different value for ticks in Watch 1 and click on RUN. The trace will be updated.
16. With the program stopped (or the LA Update Screen is Stopped) you can measure value of ticks:



17. **When finished, click on STOP if the program is running and delete this Watchpoint by selecting Debug and select Breakpoints and select Kill All. Select Close.**

    Note: Selecting Debug and the Kill all Breakpoints will not delete Watchpoints.

18. Leave Debug mode.

TIP: You cannot set Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

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

Dual Core MBX Example:

This example uses a ULINK2 by default. It is possible to use a ULINK*pro*, J-Link or a LPC-Link 2. SWV is not used in this example. Since the two CPUs must be controller with the JTAG chain and not SWD, Serial Wire Viewer (SWV) is not available except by using a ULINK*pro* and the 4 bit Trace Port instead of the SWO pin. The LPC4300 CPU speed must then be limited to 120 MHz because the LPC4300 Trace Port lacks sufficient drive at higher frequencies. It is important to note that by using SWD and only the Cortex-M4 core, SWV is available at full speed. Recall the Cortex-M0 does not have SWV.

LPC4300 IPC for the Cortex-M4 and Cortex-M0 cores:

NXP provides two methods of Inter Processor Communication (IPC) between the two cores of the LPC4300 series. The Cortex-M4 is the "master" and the Cortx-M0 the "slave". Example projects are provided for both techniques, but we will examine only the mailbox system in this document.

1) **Message Queue:** Two areas of shared memory are defined. The Command buffer is used exclusively by the master (M4) to send commands to the slave (M0). The Message buffer is used exclusively by the slave to send data to the master. An interrupt mechanism is used to signal to the core a message or command is available.
2) **Mailbox:** An area in RAM is used by the sending processor to place a message for the receiving processor. The master uses an interrupt to signal to the slave that data has been placed in the mailbox(s).

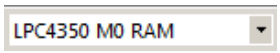
MDK controls both processors by running two instances of µVision running: one controlling the M4 and the other the M0. A ULINK2, ULINK*pro* or LPC-Link 2 can be used. This exercise will use the ULINK2. JTAG and not SWD is used.

1) Starting and Configuring µVision for the Dual CPU LPC4300 series:

1) Connect a ULINK2 to the Keil MCB4300 board and power both in the usual way.
2) Start µVision by clicking on its desktop icon if it is not already running.  µVision must be in Edit mode.
3) Open the Project C:\Keil\ARM\Boards\Keil\MCB4300\DualCore\CM4\CM4 Blinky.uvproj
4) Start a second instance of µVision by clicking on its desktop icon. You can use a second monitor.
5) Open the Project C:\Keil\ARM\Boards\Keil\MCB4300\DualCore\CM0\CM0 Blinky.uvproj

2) Compiling the Programs:

TIP: The CM0 project must be compiled first. It provides the file CM0_image.c that is necessary to the M4 project.

1) Make sure the instance of µVision loaded with CM0 Blinky.uvproj is in focus.
2) Select LPC4350 M0 RAM in the target selection box: 

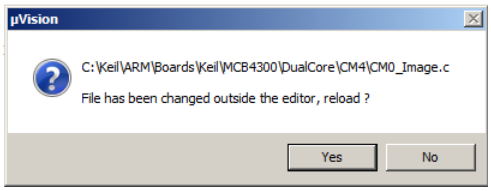
3) Click on the Rebuild icon to compile the Cortex-M0 source files.  There will be no errors or warnings.
4) The file CM0_Image.c is now created which is used in the CM0 Blinky.uvproj compilation. See the **TIP:** below.
5) Bring the instance µVision loaded with CM4 Blinky.uvproj loaded into focus by clicking inside it.
6) Select the MCB4300 external flash in the Target Selector box: 

TIP: You can use other Flash devices on the MCB4300 board but you might have to modify the jumpers shown on page 5.

7) Click on the Rebuild icon to compile the Cortex-M4 source files.  There will be no errors or warnings.
8) You have now successfully compiled both the M4 and M0 source files in the Blinky example programs.
9) On the next page you will program the Flash and run the program.

TIP: If you have the file CM0_Image.c or its tab visible in the editing window in the M4 instance after you compile the M0 source files, you will get this notice:

µVision has updated this file and it will not allow an old version to be used without warning you. Please click on the Yes if you see this message:



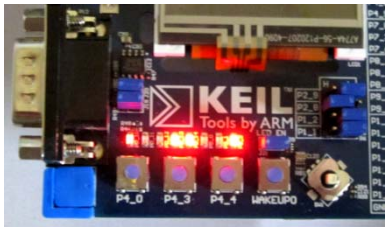
3) Programming the Flash:

1) In Steps 6 and 7 on the preceding page you selected the MCB4350 external Flash and compiled to this selection.
2) Bring the CM4 Blinky.uvproj instance of µVision in focus by clicking on it.
3) Program the Flash by clicking on the Load icon. Progress will be indicated at the lower left of your screen.
4) You will receive a Verify OK in the Build Output window.
5) At this point in time, if four LEDs are now blinking on the board, then both CPUs are running. This is a good sign.

TIP: The LPC4357 will run by default any valid program in internal flash. To bypass this: press the RESET and ISP buttons simultaneously and release RESET first or erase the program residing in the LPC4357 internal Flash. See page 5 for details.

4) Describing the LEDs:

There are 8 LEDs on the MCB4300 board and this example will use 4 of them as described below:



6) Right: P2.9 LED 0: The Cortex-M0 blinks this LED when its is running.
7) Left: P0.10 LED7: The Cortex-M4 blinks this LED when it is running.
8) Middle: P0.13 LED3 and P0.14 LED4: Alternate as the cores communicate.

5) Running the LPC4300 processors with µVision:

At this point, the Blinky programs are running on both cores without any intervention by µVision. If you disconnect the ULINK2 and reset the board, these two programs will now run stand-alone from the Flash. Now, we will connect µVision to the two cores.

The Cortex-M0 CPU:

1) Bring the CM0 Blinky.uvproj instance of µVision into focus by clicking on it.
2) Enter Debug mode for M0 Blinky . The right LED will stop blinking. Click on RUN and it will start again.

The Cortex-M4 CPU:

3) Bring the CM4 Blinky.uvproj instance of µVision into focus by clicking on it.
4) Enter Debug mode for M4 Blinky . The left LED will stop blinking. Click on RUN and it will start again.
5) The middle LEDs will start to alternate indicating communication between the cores.
6) You can start and stop each core to see the effect. Make both CPUs are running for the next step.

Troubles ?: If you encounter trouble, repower the board and/or the ULINK2 and try again. See the **TIP:** above.

6) Setting Hardware Breakpoints:

CM4 Blinky:

1) Bring the CM4 Blinky.uvproj instance of µVision into focus by clicking on it.
2) Set two breakpoints in Blinky.c in CM4 near lines 97 and 102 as shown here: This is where where LED_On(7) and LED_Off are called.
3) The CM4 program will soon stop on one of the breakpoints.
4) Each time you click on RUN, the left LED 7 will toggle.

```
94  while (1) {
95    if (LEDOn) {
96      LEDOn = 0;
97      LED_On (7);
98    }
99
100   if (LEDOff) {
101     LEDOff = 0;
102     LED_Off (7);
103   }
104
105   if (M4_EventSet) {
```

CM4 Blinky

CM0 Blinky:

5) Bring the CM0 Blinky.uvproj instance of µVision into focus by clicking on it.
6) Set two breakpoints in Blinky.c in CM0 in a similar position near lines 78 and 83 where the functions LED_On(0) and LEDOff(0) are called.
7) The CM0 program will soon stop on one of the breakpoints.
8) Each time you click on RUN, the right LED 0 will toggle.
9) **When finished, remove all 4 breakpoints from both instances of µVision.**

```
75  while (1) {
76    if (LEDOn) {
77      LEDOn = 0;
78      LED_On (0);
79    }
80
81    if (LEDOff) {
82      LEDOff = 0;
83      LED_Off (0);
84    }
```

CM0 Blinky

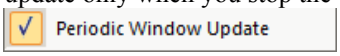
TIP: You can see that you can easily control both processors using two instances of µVision. A standard MDK license allows this operation. You do not need to purchase two licenses to operate the two Cortex cores as demonstrated here.

7) Watch Window:

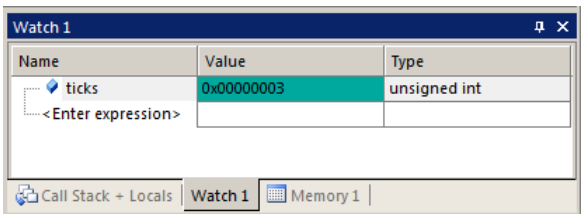
1) Start both processors by clicking on RUN in each instance of µVision.

CM0 Blinky Watch window:

2) Bring the CM0 Blinky.uvproj instance of µVision into focus by clicking on it.
3) Select View and ensure Periodic Window Update is enabled. If not, windows will update only when you stop the program. We want to see the ticks update in real-time as the program is running.



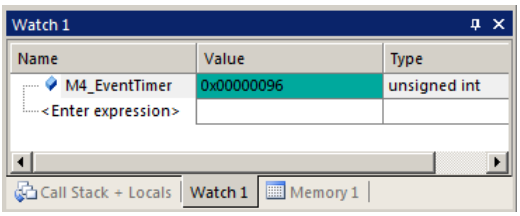
4) In Blinky.c, set a breakpoint somewhere in the function M0_RIT_OR_WWDT_IRQHandler near line 24 in Blinky.c. The program will stop here. We want to bring ticks into scope.
5) Right click on the static variable ticks and select Add 'ticks' to… and select Watch 1.
6) Watch 1 will be opened and ticks is displayed. You do not have to stop the program for global variables.
7) Remove the breakpoint and click on RUN. ticks will start to update in the Watch 1 window as shown here:



TIP: ticks is a static local variable. While its value is stored in RANM and not temporarily on the stack or in a CPU register, its scope does not cover the entire run of the program like a global variable does. This is why you must either specify ticks while it is in scope (as you just did in step 4), fully qualify it or select it from the Symbol table in µVision (select this in View/Symbols Window).

CM4 Blinky Watch window:

1) Bring the CM4 Blinky.uvproj instance of µVision into focus by clicking on it.
2) Select View and ensure Periodic Window Update is enabled as described in step 2 above.
3) If CM4 Blinky is not running, start it by clicking on RUN.
4) In Blinky.c, find the global variable M4_EventTimer near line 24 and right click on it and select Add 'M4_EventTimer' to … and select Watch 1. Watch 1 will be opened if necessary and M4_EventTimer is displayed and updated in real-time as the program is running.



TIP: M4_EventTimer is a global variable. By definition it is in scope during the entire run of the program. It can be entered in µVision's various windows without stopping the program or specifying it fully qualified.

TIP: If the Watch window does not update, this can happen when the Periodic Update is not enabled or the CM0 Blinky program is not running.

TIP: You can also open a Memory window in the same fashion. In a Memory window, recall you can modify a memory location while the program is running.

8) Serial Wire Viewer (SWV) with dual CPU operation:

SWV needs Serial Wire Debug to be enabled (SWD or SW) and not JTAG. The reason is the SWO pin is shared with the JTAG TDO pin on the JTAG connector and this conflict precludes SWV operation.

However, if you use a ULINK*pro*, you can configure it to send the SWV trace frames out the 4-bit Trace Port rather than the SWO pin and this eliminates the conflict. SWV works perfectly in this case with the added advantage of ETM trace frames.

The LPC4300 series must be limited in speed to 80 MHz in order for ETM and SWV trace to work properly. See the LPC4300 data sheet for more information. This is due to a lack of sufficient Trace Port drive on the LPC4300 processors. Experiments with the MCB4357 Keil board and a ULINK*pro* has determined up to 120 MHz operation is possible. For instruction trace at full speed, use Embedded Trace Buffer (ETB) which is supported by ULINK*pro*.

This problem arises only when you must use JTAG as with dual CPU debugging. SWV works well using the SWO port. If you can use SWD, you can use Serial Wire Viewer data trace at very high CPU speeds. See page 25.

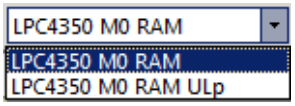
17) Notes on Dual CPU operation using µVision:

Pre-configured Target Settings:

The Dual CPU projects have various preset target selections for various memory configurations and debug adapters.

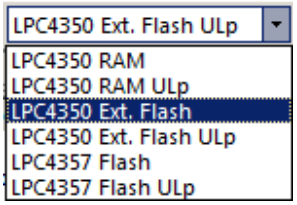
CM0 Blinky project:

The CM0 source files, once compiled, runs in RAM. The CM4 program loads the CM0 executable into RAM and then starts the Cortex-M0 which in turn runs its executable. There are only two possible selections provided. One is for the ULINK2 or ULINK-ME and another for the ULINK*pro*.



CM4 Blinky project:

There are more selections for the Cortex-M4 in the LPC4300 series. As shown, there are targets pre-configured for RAM, external Flash, internal Flash (LPC4357 only) and SPIFI memory. The LPC4350 settings work with the LPC4357 but be aware the RAM on the LPC4357 is about half half of the LPC4350. Some examples for the LPC4350 RAM such as Demo may not work.
The "Ulp" selections are configured for the ULINK*pro* and the rest are for the ULINK2/ME.



Adding Target Selections:

It is easy to create your own target selection. Every setting in the Target Options windows will be saved in your new target. This can include different debug adapters such as the LPC-Link 2.

Select Target Options or ALT-F7 to browse the possible attributes you can modify. Close this window when done.

Making Your Own Target Selection:

7) Select the Target Selection in the boxes above that you want to be the template for your new version.
8) Select Project/Manage and then choose Components, Environments, Books…
9) In the left pane, select the New icon or press your PC Insert key.
10) Enter the name you choose for your new target option and click OK. Your new entry will now be visible in the Target Selector box. Select this entry. Now, you will modify your new target selection entry.
11) Select Target Options or ALT-F7. Make your modifications in the windows that open.
12) When completed, click OK to close the Target Options. Select File/Save All.

.ini Files:

There are various initialization files that are used to configure specific Target Options settings. These are found in the Initialization box in the Debug tab and the Init box under Utilities. Look for example ini files you can use. An ini file is not always necessary. Look at various pre-configurations for examples to use as templates for your own projects.

Processor name and Memory Settings:

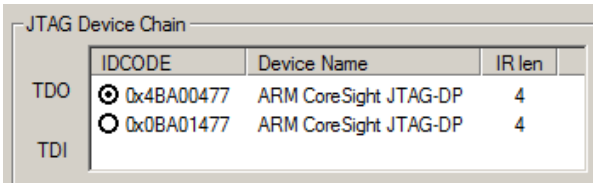
In the Target Options under the Device tab is where you select your particular processor.
Under the Target tab is where you enter the memory maps. µVision uses these settings to create a scatter file for you.

RESET:

In the Debug tab: select Settings: and there is a RESET box. It is set to VECTRESET. This setting is very important. If the program ends up in the Hard Fault vector when you run it, check that this box is set correctly.

Selecting the CPU:

µVision selects the Cortex-M4 or Cortex-M0 CPU by using the JTAG chain. In the Debug/Settings: The CPU you want to connect to is listed in the JTAG Device Chain box as shown here:



Note two entries in the JTAG Device Chain box.
0x**4**BA00477 is the Cortex-M4 JTAG device.
0x**0**BA01477 is the Cortex-M0 JTAG device.

Select the appropriate CPU as needed. The Cortex-M4 is selected in this case.

RTX_Blinky Example Program with Keil RTX: A Stepper Motor example

Keil provides RTX, a full feature RTOS. RTX is included as part of Keil MDK including source. It can have up to 255 tasks and no royalty payments are required. This example explores the RTX RTOS project. MDK will work with any RTOS. RTX comes with a BSD license and source code. See www.keil.com/demo/eval/rtx.htm and www.arm.com/cmsis.

For detailed information about RTX see www.keil.com/rl-arm/kernel.asp.

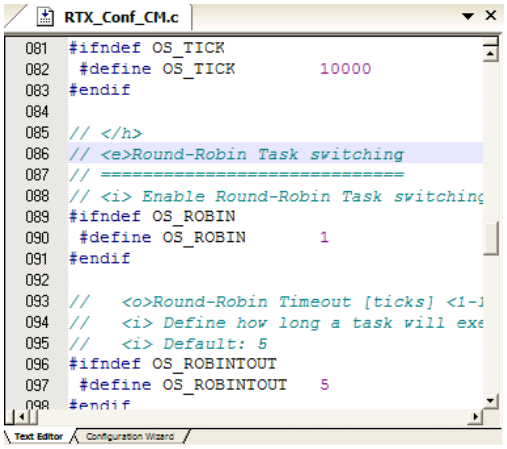
1) Running RTX Blinky:

1. With µVision in Edit mode (not in debug mode): Select Project/Open Project.
2. Open the file C:\Keil\ARM\Boards\Keil\MCB4300\RTX_Blinky\Blinky.uvproj.
3. If you are not using a ULINK2 or ULINK-ME, you now have to configure µVision for the adapter you are using. You only have to do this once – it will be saved in the project file. You can also make a new target configuration.
4. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
7. The LCD display and four LEDs will indicate the four waveforms of a stepper motor driver changing. The LCD screen is shown here:

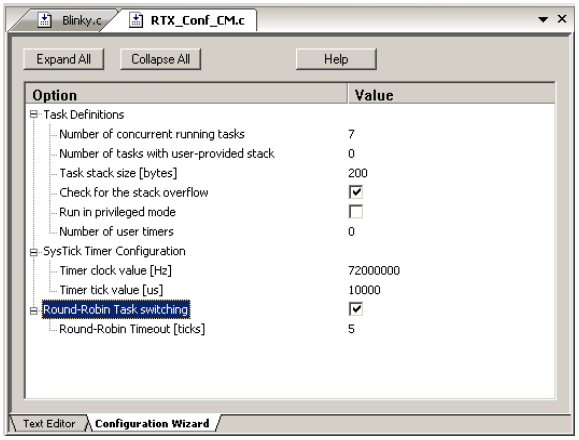
8. Click on STOP.

The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open if needed from the µVision main menu.
2. Click on the Configuration Wizard tab at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. Changing an attribute in one tab changes it in the other automatically. You should save a modified window.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>.
8. See www.keil.com/support/docs/2735.htm for instructions on using this feature in your own source code.
9. The µVision System Viewer windows are created in a similar fashion. Select View/System Viewer.



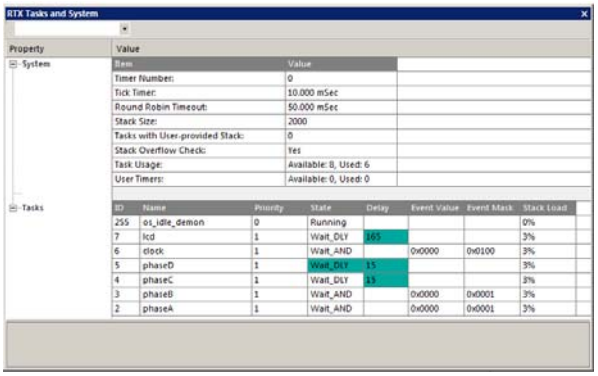
Text Editor: Source Code



Configuration Wizard

2) RTX Kernel Awareness using Serial Wire Viewer (SWV):

Users often want to know the current operating task number and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides two Task Aware windows for RTX by accessing this information.

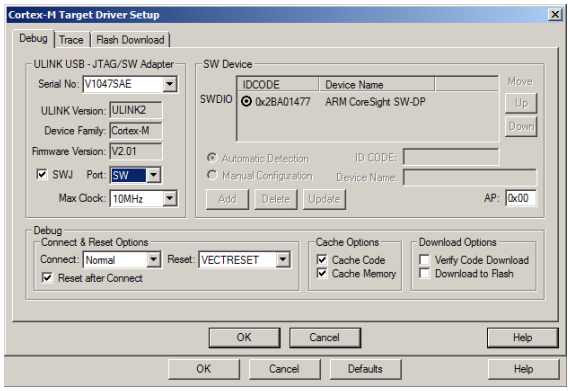


1. Use the RTX_Blinky example from the previous page.

   Click on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen.
   These values are updated in real-time using the same technology as used in the Watch and Memory windows.
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed in this window because SWV is not yet configured. The Event Viewer needs SWV operational. We will do this on the next page.

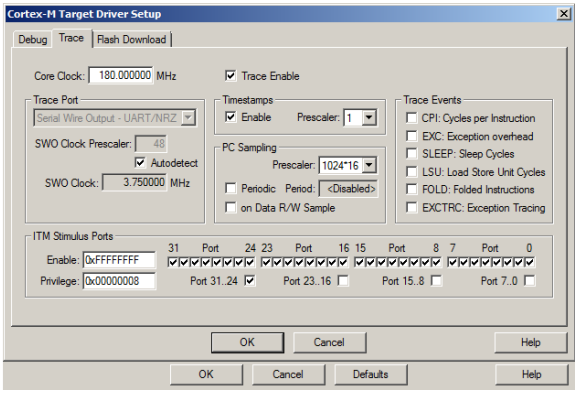
RTX Viewer: Configuring Serial Wire Viewer (SWV):

We must activate Serial Wire Viewer to get the Event Viewer working (and also the Logic Analyzer)..

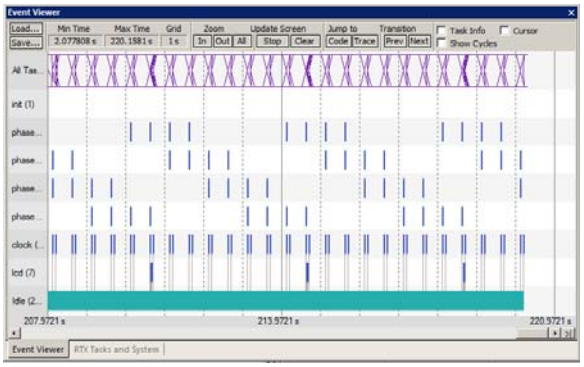


1. Stop the CPU and exit debug mode.
2. Click on the Target Options icon next to the target box.
3. Select the Debug tab. Click the Settings box next to ULINK2/ME Cortex Debugger.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.

TIP: Make sure the RESET: box is set to VECTRESET as shown. If you experience strange problems, check this setting.



5. Click on the Trace tab to open the Trace configuration window.
6. Set Core Clock: to 180 MHz and select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown here:
8. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer. It is slightly intrusive.
9. Click on OK twice to return to µVision.
10. Select File/Save All.
    The Serial Wire Viewer is now configured.
11. Enter Debug mode and click on RUN.
12. Select "Tasks and System" tab: note the display is updated.
13. Click on the Event Viewer tab.
14. This window displays task events in a graphical format as shown in the RTX Kernel window here. You probably have to change the Range to about 1 seconds by clicking on the ALL and then the In and Out icons.



TIP: If the Event Viewer doesn't work, the most common cause is a wrong Core Clock: value or too many trace items set.

3) Logic Analyzer Window (LA): View variables real-time in a graphical format:

µVision has a graphical Logic Analyzer window. Up to four variables can be displayed in real-time using the Serial Wire Viewer. RTX_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.

1. Close the RTX Viewer windows. Stop the program and exit Debug mode.
2. Declare 4 global variables **`unsigned int phasea`** through **`unsigned int phased`** to Blinky.c as shown here:

```
028 #define LED_D       0
029 #define LED_CLK     LED_1
030
031 unsigned int phasea;
032 unsigned int phaseb;
033 unsigned int phasec;
034 unsigned int phased;
035
036 /*----------------------------
037  *        Function 'signal_fu
038  *----------------------------
```

3. Add 2 lines to each of the four tasks Task1 through Task4 in Blinky.c as shown below: **phasea=1;** and **phasea=0;** :the first two lines are shown added at lines 84 and 87 (just after the Switch_On and Switch_Off function calls). For each of the four tasks, add the corresponding variable assignment statements phasea, phaseb, phasec and phased.
4. We do this because in this simple program there are not enough suitable variables to connect to the Logic Analyzer.

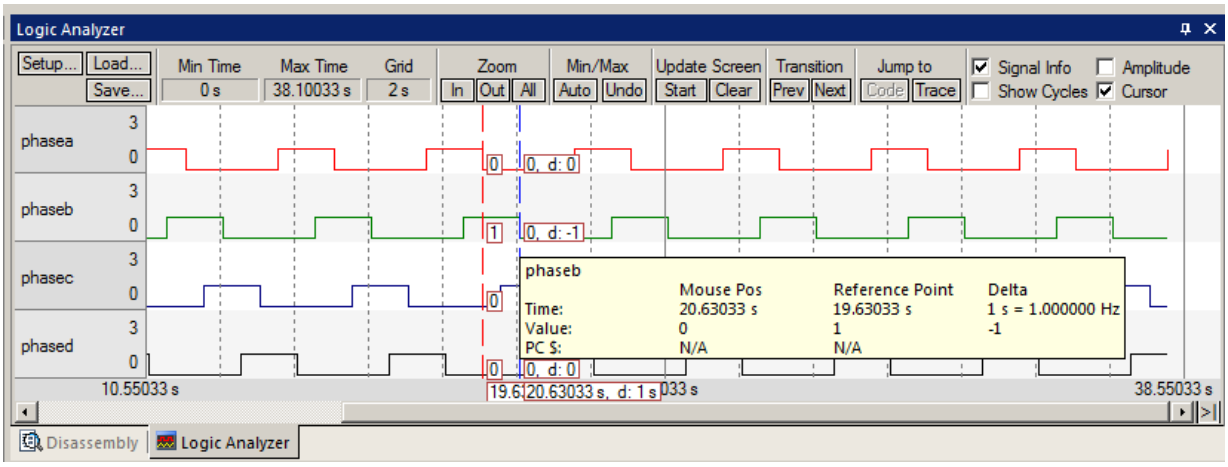
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers, merely read or write to them and enter them into the Logic Analyzer.

```
77 /*----------------------------------------
78  *        Task 1 'phaseA': Phase A output
79  *----------------------------------------
80 __task void phaseA (void) {
81   for (;;) {
82     os_evt_wait_and (0x0001, 0xffff);     /*
83     Switch_On (LED_A);
84     phasea=1;
85     signal_func (t_phaseB);               /*
86     Switch_Off(LED_A);
87     phasea=0;
88   }
89 }
```

5. Rebuild the project. Program the Flash.
6. Enter debug mode.
7. You can run the program at this point.

Enter the Variables into the Logic Analyzer:

8. Right-click on the variable **phasea** and select Add 'phasea' to… and then select Logic Analyzer. The LA will open.
9. Repeat for **phaseb, phasec and phased.** These variables will be listed on the left side of the LA window as shown. You might have to adjust the LA margins. Now we have to adjust the scaling.
10. Click on the Setup icon and click on each of the four variables in turn and set Max. in the Display Range: to 0x3.
11. Click on Close to go back to the LA window.
12. Using the OUT and In buttons set the range to 1 or 2 seconds. Move the scrolling bar to the far right if needed.
13. You will see the following waveforms appear. Click on STOP in the Update Screen box. The program will keep running. Select Signal Info and Cursor.
14. Click to mark a place. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phaseb: Note each LED is on for about 1.5 seconds.
15. Stop the program and exit Debug mode when you are finished.



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

TIP: You can view signals that exist mathematically in a variable and not available for measuring in the outside world. Serial Wire Viewer can assist you in debugging problems that defy conventional tools and techniques.

DSP SINE Example using ARM CMSIS-DSP Libraries:

1) Running the DSP SINE example:

ARM CMSIS-DSP libraries are offered for ARM Cortex-M0, Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil\ARM\CMSIS. README.txt describes the location of various CMSIS components. See www.arm.com/cmsis and forums.arm.com for more information.

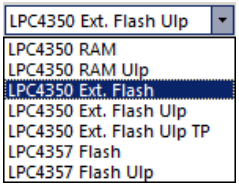
You can use this example with other Cortex-M boards with possible changes to the startup and system files.

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 is available free with a BSD type license. Source code is provided.

To obtain this DSP example project, go to **www.keil.com/appnotes/docs/apnt_241.asp**

1. Extract DSP.zip to C:\Keil\ARM\Boards\Keil\MCB4300\ to create the folder \DSP.
1. Open the project file sine.uvproj with µVision. Connect a ULINK2 or ULINK*pro* to the MCB4300 board.
2. If you are using a ULINK2 or ME, select LPC4350 Ext Flash from the target drop menu: If using a ULINK*pro*, select LPC4350 Ext Flash Ulp to send SWV data out the SWO pin.



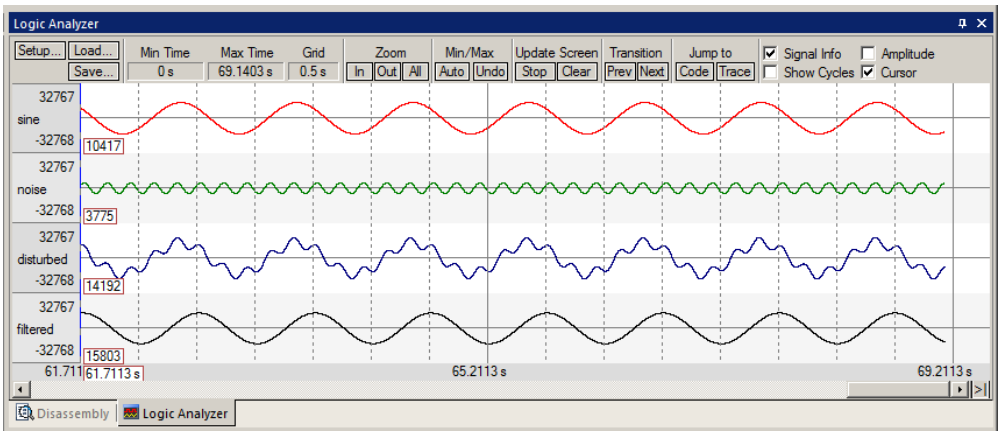
3. Compile the source files by clicking on the Rebuild icon.  .
4. Program the MCB4300 flash by clicking on the Load icon: 
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.

TIP: The default Core Clock: is 180 MHz for use by the Serial Wire Viewer configuration window in the Trace tab.

6. Click on the RUN icon.  Open the Logic Analyzer window if necessary. 
7. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom for an appropriate display. Displayed are 4 global variables: `sine`, `noise`, `disturbed` and `filtered`.

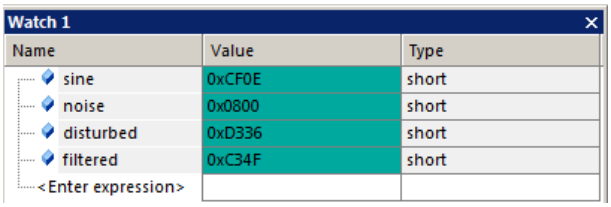
TIP: If one or two variables shows no waveform, disable the ITM Stimulus Port 31 in the Trace Config window. The SWO pin is probably overloaded if you are using a ULINK2. ULINK*pro* handles SWV data faster than a ULINK2 or J-Link can.

8. This project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.
9. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below: They are pre-configured in this project.



10. 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.
11. Open View/Serial Windows/Debug (printf) Viewer. printf data is displayed from printf statements in DirtyFilter.c near lines 174 through 192 using the ITM.

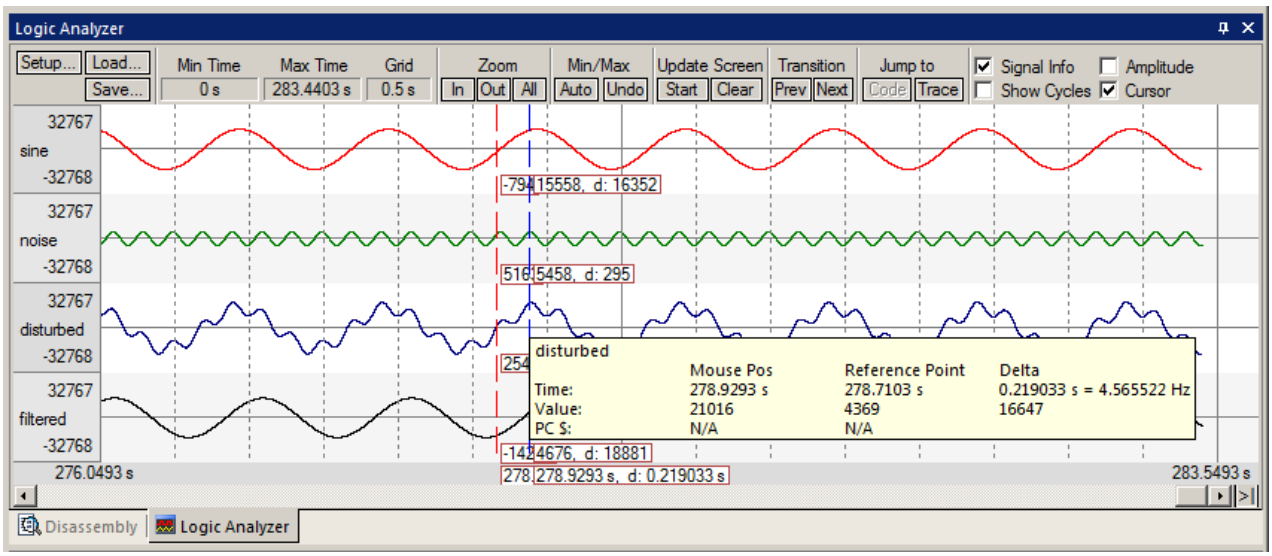
TIP: The ULINK*pro* trace display is different and the program must currently be stopped to update it. The LA will be updated in real-time.



12. Leave the program running.
13. Close the Trace Records window if it is open.

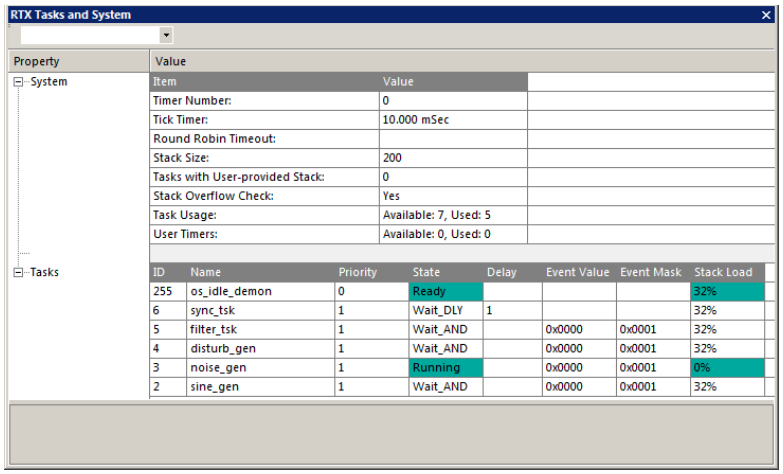
2) Signal Timings in Logic Analyzer (LA):

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



3) RTX Tasks and System Awareness window:

6. Click on Start in the Update Screen box to resume the collection of data.
7. Open Debug/OS Support and select RTX Tasks and System. A window similar to below opens up. You probably have to click on its header and drag it into the middle of the screen.
8. Note this window does not change much: most of the processor time is spent in the idle daemon: which shows as Running. The processor spends relatively little time in other tasks. You will see this illustrated on the next page.
9. Set a breakpoint in each of the four tasks in DirtyFilter.c by clicking in the left margin on a grey box near lines 77, 95, 115 and 135. Do not select the actual line while(1) as this will not stop the program.
10. If the program is not running, click on Run and the program will stop at one of the breakpoints and the Task window will be updated accordingly. In the screen below, the program stopped in the noise_gen task:
11. Clearly you can see that noise_gen was running when the breakpoint was activated.
12. Each time you click on RUN, the next task will display as Running.
13. Remove all the breakpoints. You can use Ctrl-B and select Kill All, then Close.



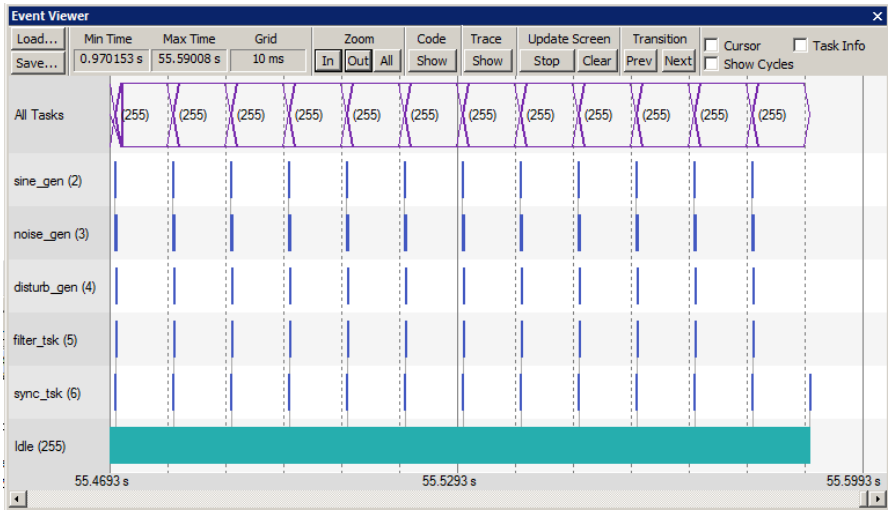
TIP: You can set hardware breakpoints while the program is running.

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

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

4) RTX Event Viewer (EV):

1. Select Debug/Debug Settings. Click on the Trace tab.
2. Enable ITM Stimulus Port 31. Event Viewer uses this to collect its information.
3. Click OK twice.
4. Exit and re-enter Debug mode  to refresh the Trace Configuration.
5. Click on RUN.
6. Open Debug/OS Support and select Event Viewer. The window here opens up:
7. Note there is no Task 1 listed. Task 1 is main_tsk and is found in DirtyFilter.c near line 169. It runs some RTX initialization code at the beginning and then deletes itself with os_tsk_delete_self(); found near line 195.



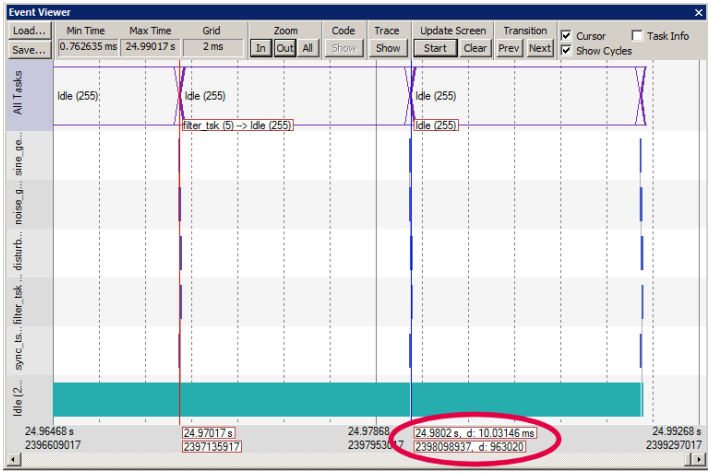
TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some SWO or Trace Port bandwidth. It depends on how much trace data is sent to the ports.

How to unload the SWO if Event Viewer does not work: Stop the program. Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables and select Close. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might occur. You can also leave the LA loaded with the four variables to see what the Event Viewer will look like. Or you can remove just one or two. Later, delete them to see the effect on the EV.

ULINK*pro* is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows. ULINK*pro* uses the faster Manchester format than the slower UART mode that ST-Link, ULINK2 and J-Link uses. If you expect to use SWV extensively and at high data rates, please consider purchasing a ULINK*pro*.

ULINK*pro* can also use the 4 bit Trace Port for even faster operation for SWV. Trace Port use is mandatory for ETM trace. With the LPC4300 series, you must slow the CPU clock down to 120 MHz or less to get the Trace Port to operate corectly.

8. Note on the Y axis each of the 5 running tasks plus the idle daemon. Each bar is an active task and shows you what task is running, when and for how long.
9. Click Stop in the Update Screen box.
10. Click on Zoom In so three or four tasks are displayed as shown below:
11. Select Cursor. Position the cursor over one set of bars and click once. A red line is set here:
12. Move your cursor to the right over the next set and total time and difference are displayed. D = ~ 10 ms.
13. Note, since you enabled Show Cycles, the total cycles and difference is also shown.



The 10 ms shown is the SysTick timer value. This value is set in RTX_Conf_CM.c. The next page describes how to change this.

TIP: ITM Port 31enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port even if you are not using the Event Viewer and this is a good idea if you are running RTX with high SWO use.

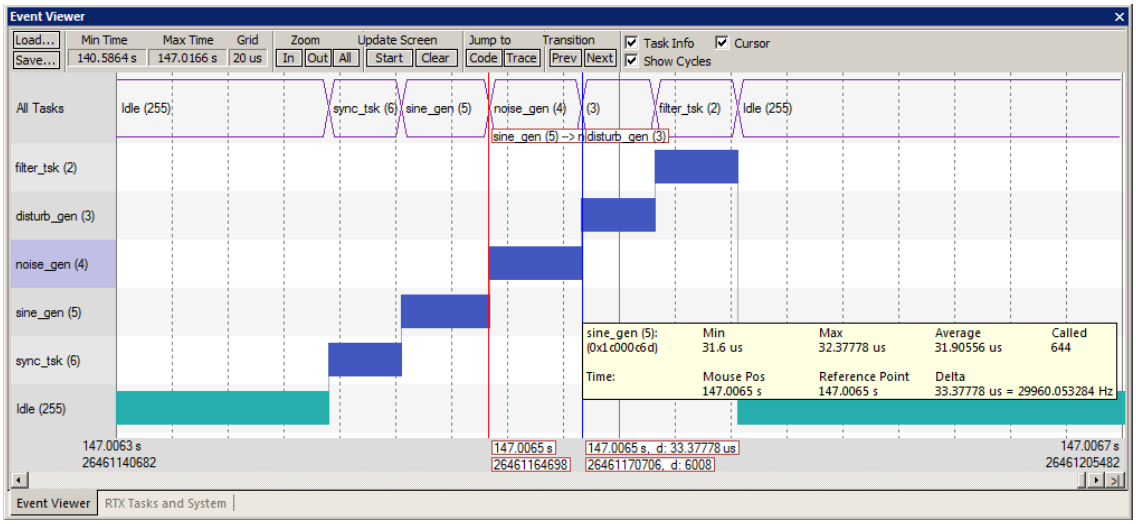
If the Event Viewer is closed, the data is still being sent out the SWO pin or the Trace Port and contributes to overloading.

5) Event Viewer Timing:

1. Click on In under Zoom until one set of tasks is visible as shown below:
2. Enable Task Info (as well as Cursor and Show Cycles from the previous exercise).
3. Note one entire sequence is shown. This screen was taken with a ULINK2 with LA cleared of all variables.
4. Click on a task to set the cursor and move it to its end. The time difference is noted. The Task Info box will appear.

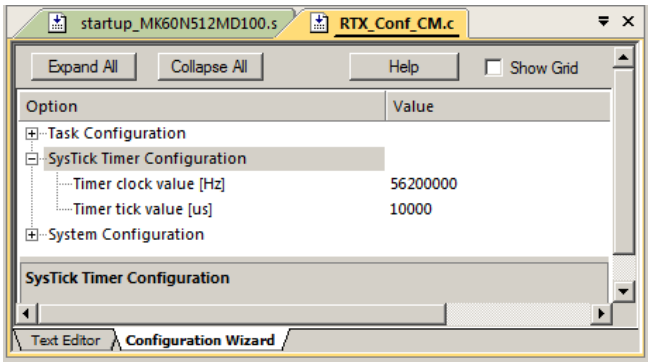
TIP: If the Event Viewer does not display correctly, the display of the variables in the Logic Analyzer window might be overloading the SWO pin. In this case, stop the program and delete all LA variables (Kill All) and click on Run.

The Event Viewer can give you a good idea if your RTOS is configured correctly and running in the right sequence.



6) Changing the SysTick Timer:

1. Stop the processor and exit debug mode.
2. Open the file RTX_Conf_CM.c from the Project window. You can also select File/Open and select it in C:\Keil\ARM\Boards\Keil\MCB4300\DSP\.
3. Select the Configuration Wizard tab at the bottom of the window. This scripting language is shown in the Text Editor as comments starting such as a </h> or <i>. See www.keil.com/support/docs/2735.htm for instructions.
4. This window opens up. Expand the SysTick Timer Configuration as shown here:
5. Note the Timer tick value is 10,000 µs or 10 ms.
6. Change this value to 20,000.



TIP: The 180,000,000 is the CPU speed and is correct for this DSP example.

7. Rebuild the source files and program the Flash.
8. Enter debug mode and click on RUN.
9. When you check the timing of the tasks in the Event Viewer window as you did on the previous page, they will now be spaced at 20 msec.

TIP: The SysTick is a dedicated timer on Cortex-M processors that is used to switch tasks in an RTOS. It does this by generating an Exception 15 periodically every 10 µs or to what you set it to. You can view these exceptions in the Trace Records window by enabling EXCTRC in the Trace Configuration window. You can use SysTick for other purposes.

10. Set the SysTick timer back to 10,000. Click on File/Save All.
11. Stop the processor and exit Debug mode and re- compile the source files to the RTOS settings back to original.

Using a ULINK*pro* Trace Port with a LPC4300 series Processors:

We have seen what features offered with a ULINK2 including Serial Wire Viewer (SWV) which is Data Trace. Having a ULINK*pro* adds the capability of capturing ETM Instruction Trace. ETM further brings Code Coverage, Performance Analysis and Execution Profiling as well as program flow debugging and analysis.

SWV Data Trace can be brought out:

a) The SWO pin on the JTAG/SWD connectors with any debug adapter listed except LPC-Link 2.

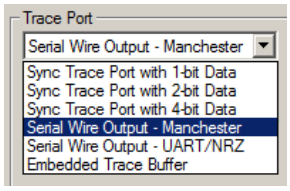
b) OR the 4 bit Trace Port on the 20 pin Cortex Debug + ETM connector.

ETM Instruction Trace can be brought out: (using a ULINK*pro*)

a) The 4 bit Trace Port. In this case, the SWV information also must come out here and not the SWO pin.

b) From the ETB (Embedded Trace Buffer). Currently, only the ULINK*pro* can access this trace memory.

How these options are selected:



The ULINK*pro* can be set to any option shown here except for the UART/NRZ mode.

Sync Trace Port: Used for Trace Port. 4 bit is most useful. On LPC4300 the maximum frequency is NXP specified at 80 MHz. On MCB4300 it will work to 120 MHz.

Serial Wire Output – Manchester: No ETM trace. Only SWV. Uses the SWO pin in Manchester encoding mode. The SWO pin works to the full CPU speed with a ULINK2/ME or ULINK*pro*.

Serial Wire Output – UART/NRZ: SWO output used only by ULINK2/ME and J-Link. Is preset. No ETM trace

Embedded Trace Buffer: (ETB) – a small on-chip trace memory to provide instruction trace. ETB will operate at full CPU speed. This is useful when using ETM at a lower CPU speed is not acceptable. SWV is not available in this mode.

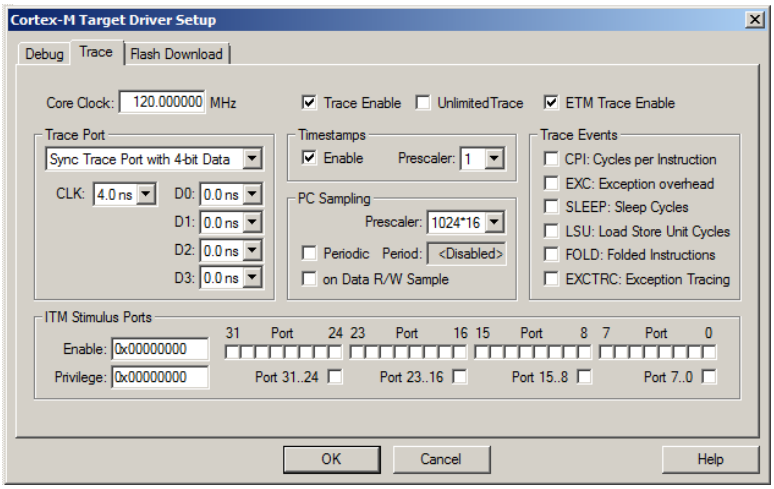
Other Issues:

CPU Speed: The CPU clock speed must be lowered to 120 MHz in order for the LPC4300 Trace Port to operate correctly because if its port drive specification. This is easily configured on the file system_LPC43xx.c. Several values must be changed: PLL!_MSEL and PLL1_DIRECT. **`LPC_EMC->`**STATICWAITPAG0 might need to be changed for external Flash to 2. Re-build after changing any of these values. These are detailed in the comments of these files. Version 5.0 of system_LPC43xx.c that runs at 120 MHz is available as part of the DSP files and is provided here: www.keil.com/appnotes/docs/apnt_241.asp.

Recall that the SWO pin can operate to at least 180 MHz with either the ULINK2/ME or ULINK*pro*.

Trace Configuration window (shown below):

The window below is obtained by opening Blinky_Ulp and selecting the Sync Trace Port with 4 bit data. Select Target Options  and select the Debug tab. Choose Settings: and then the Trace tab.



CLK: This must be set to 4.0 ns as shown below when the Sync Trace Port is selected. This compensates for clock slew on the LPC4300 Trace Port pins.

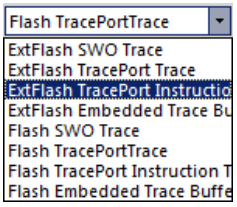
Core Clock: ULINK*pro* determines the trace clock speed itself. The Core Clock: value is used to display µVision timing values. ULINK2/ME must have the correct value inserted or the SWV will not operate.

ITM Port 31 is used for the RTX Event Viewer. Port 0 is used by the Debug printf Viewer. The others are not used by µVision.

Help: Clicking on Help will display more information about the other settings.

Initialization File (.ini): A script is needed to configure the LPC4300 GPIO to Trace Port configuration. Look for function TraceSetup in the file C:\Keil\ARM\Boards\Keil\MCB4300\Blinky_Ulp\LPC43xx_TP.ini. Duplicate this function in an existing .ini file or a new one. Put it in the Initialization box as shown here:

1) Blinky_Ulp ETM Trace Example: ULINK*pro* is needed for these steps:



The Blinky_Ulp example is pre-configured only for the ULINK*pro*. It runs at 120 MHz. All the other LPC4300 example projects run at 180 MHz.

There are various Target Options available. You can use these as templates for your own Target Selections. Two sets are provided for both external and internal Flash.

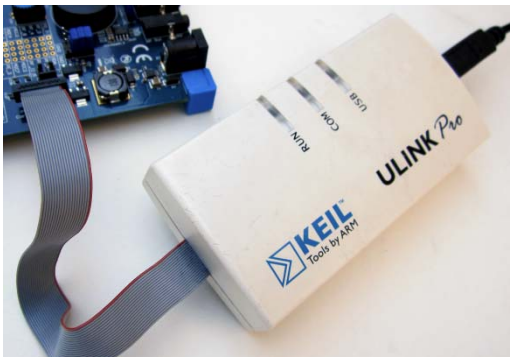
Flash: Operates only with LPC4357 and its internal Flash. This is preferable to use than the external Flash options as it is easier to use.

ExtFlash: Operates with LPC4350 and LPC4357 running on the MCB4300 external Flash. If you experience program instability, in system_LPC43xx.c near line 664, change **`LPC_EMC->STATICWAITPAG0 = 1;`** to a 2. Re-build and Flash.

Version 5.0 of system_LPC43xx.c incorporates this change. It is available as part of the DSP example files zip file. This will be part of the future MDK 4.72.

Abstract.txt describes the various options indicated in the target selector above.

Running Blinky_Ulp:



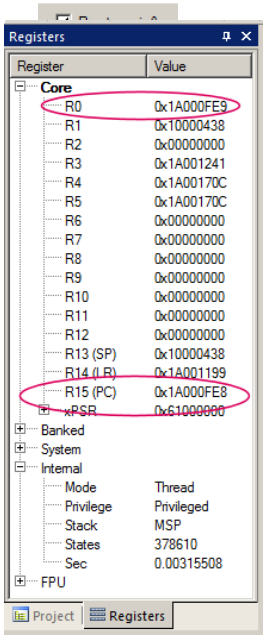
1. Connect a ULINK*pro* to the target board Cortex Debug + ETM connector as shown here:
2. Connect both with USB to your PC.
3. Start µVision.
4. Select Flash TracePort Instruction if using a LPC4357. If using a LPC4350, select Ext. Flash TracePort Instruction.

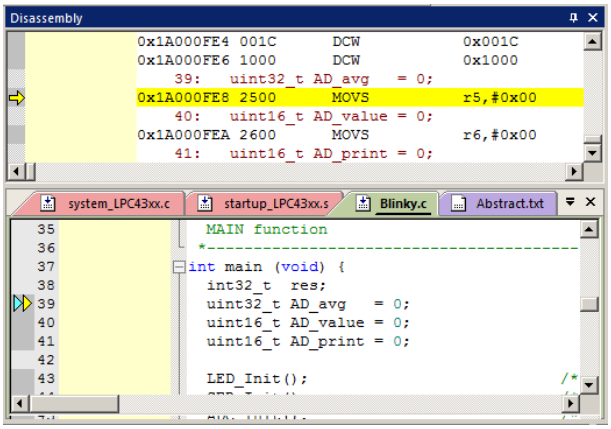
5. Compile the source files by clicking on the Rebuild icon.  . Progress is indicated in the Build Output window.
6. Program the Flash by clicking on the Load icon:  Progress is indicated in the Build Output window.
7. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
8. DO NOT click on the RUN icon. 
9. We will examine the Trace Data window on the next page.

What Has Happened:

µVision has loaded the Blinky executable in the Flash (internal or external as you chose it) and executed the program and stopped at main(). It stopped here because this is selected in the Target Options under the Debug tab:

1. The ETM Instruction Trace has recorded every instruction executed since RESET.
2. The next instruction to be executed is 0x1A00_0FE8 as shown here in the R15 (PC):
3. The Disassembly window below displays the next instruction to be executed (MOVS).
4. In the Blinky.c source window shows the source line the MOVS was compiled from.
5. Clicking on a line in one of these windows highlights the connection in the other window.





TIP: The MOVS instruction is not executed because a hardware breakpoint was set as Blinky was executed and removed when it was encountered. This is how the g,main command works. ARM breakpoints do not execute the instructions they are set to. They are non-skid. This is a very important feature.

On the next page, we will look at the Trace Data window.

2) Trace Data Window using ULINK*pro* and ETM Trace:

The source code details how a program was written in C, C++ or Assembly. Various elements of programming are laid out in a fashion that depicts a logical arrangement of how and what this program will do.

Instruction Trace describes how the program was executed and in what order by collecting the executed assembly instructions in a sequential flow. For instance, all loops are displayed unrolled. Program flow debugging is easy do with ETM.

How did the program get to main() ?

On the previous page, it was shown that the next instruction to be executed is MOVS at 0x1A00_0FE8. A more interesting question might be: how did the PC get to this point ? This could happen with a sequential PC, a branch or exception activity. In our case it cannot be sequential because the memory address before the MOVS in the Disassembly window is a DCW.

This question is not very easy to discover – unless you have ETM trace. One glance at the last record in the Trace Data window discloses the last instruction executed was a BX r0. A partial Trace Data window is shown here:

If you double-click on the BX frame: this instruction will be highlighted in the Disassembly window.

0.006 313 742 s	X : 0x1A001192	*BCC 0x1A001182		__scatterload
	X : 0x1A001194	BL.W __main_after_scatterload (0x1A000308)		__scatterload
	X : 0x1A000308	LDR r0,[pc,#0] ; @0x1A00030C		???
0.006 313 817 s	X : 0x1A00030A	BX r0		???

If you click on Step (or F11) the MOVS instruction will be executed and displayed as shown here:

Note the indication that the program has entered the main() function in the Function column.

	X : 0x1A001194	BL.W __main_after_scatterload (0x1A000308)		__scatterload
	X : 0x1A000308	LDR r0,[pc,#0] ; @0x1A00030C		???
0.006 313 817 s	X : 0x1A00030A	BX r0		???
	TRACE RUN			
0.006 313 833 s	X : 0x1A000FE8	MOVS r5,#0x00	uint32_t AD_avg = 0;	main

As you click on Step (or F11), you will get a series of executed instructions displayed in execution order as shown here:

Note that C source code has now appeared in the Trace Data window:

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x1A00118E	ADDS r4,r4,#0x10		__scatterload
	X : 0x1A001190	CMP r4,r5		__scatterload
0.006 313 742 s	X : 0x1A001192	*BCC 0x1A001182		__scatterload
	X : 0x1A001194	BL.W __main_after_scatterload (0x1A000308)		__scatterload
	X : 0x1A000308	LDR r0,[pc,#0] ; @0x1A00030C		???
0.006 313 817 s	X : 0x1A00030A	BX r0		???
	TRACE RUN			
0.006 313 833 s	X : 0x1A000FE8	MOVS r5,#0x00	uint32_t AD_avg = 0;	main
	TRACE RUN			
0.006 313 842 s	X : 0x1A000FEA	MOVS r6,#0x00	uint16_t AD_value = 0;	main
	TRACE RUN			
0.006 313 850 s	X : 0x1A000FEC	MOVS r7,#0x00	uint16_t AD_print = 0;	main
	TRACE RUN			
0.006 313 858 s	X : 0x1A000FEE	BL.W LED_Init (0x1A000E30)	LED_Init(); /* LED...	main
	TRACE RUN			
0.006 313 867 s	X : 0x1A000E30	MOVS r0,#0x03	LPC_CCU1->CLK_M4_GPIO_CFG ...	LED_Init
	TRACE RUN			
0.006 313 917 s	X : 0x1A000E32	LDR r1,[pc,#288] ; @0x1A000F54		LED_Init
	TRACE RUN			
0.006 313 925 s	X : 0x1A000E34	STR r0,[r1,#0x00]		LED_Init
	TRACE RUN			
0.006 313 933 s	X : 0x1A000E36	NOP	while (!(LPC_CCU1->CLK_M4_GP...	LED_Init

Now you have seen how the ETM Instruction Trace collects, displays and saves executed instructions.

TIP: Right-click in the Trace Data window to enable the Function column. You can also set a timestamp to zero.

3) Trace Frames from the Beginning: RESET:

In the Trace Data window, scroll up to the very top: You will see a screen like this:

1. Note the very first record is a LDR at 0x1A00_0314. This is the very first instruction executed after RESET.

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x1A000314	LDR r0,[pc,#24] ; @0x1A000330	LDR R0, =SystemInit	__asm_0x1A000314
0.000 000 100 s	X : 0x1A000316	BLX r0	BLX R0	__asm_0x1A000314
	X : 0x1A000C8E	PUSH {r4,lr}	void SystemInit (void) {	SystemInit
	X : 0x1A000C90	MOV r0,#0xE000E000	SysTick->CTRL &= ~(SysTick_CTRL_TICKI...	SystemInit
	X : 0x1A000C94	LDR r0,[r0,#0x10]		SystemInit
	X : 0x1A000C96	BIC r0,r0,#0x03		SystemInit
	X : 0x1A000C9A	MOV r1,#0xE000E000		SystemInit
	X : 0x1A000C9E	STR r0,[r1,#0x10]		SystemInit
	X : 0x1A000CA0	LDR r0,[pc,#36] ; @0x1A000CC8	SCB->VTOR = ((uint32_t)(&__Vectors)) &...	SystemInit
	X : 0x1A000CA2	BFC r0,#0,#20		SystemInit
	X : 0x1A000CA6	LDR r1,[pc,#36] ; @0x1A000CCC		SystemInit
	X : 0x1A000CA8	STR r0,[r1,#0x00]		SystemInit
	X : 0x1A000CAA	BL.W SetClock (0x1A000338)	SetClock();	SystemInit
	X : 0x1A000338	PUSH {r4-r5,lr}	void SetClock (void) {	SetClock
	X : 0x1A00033A	MOVS r4,#0x1F	LPC_EMC->STATICWAITRD0 = 0x1F;	SetClock
	X : 0x1A00033C	LDR r5,[pc,#992] ; @0x1A000720		SetClock
0.000 000 400 s	X : 0x1A00033E	STR r4,[r5,#0x00]		SetClock

2. In Memory 1, enter the address 0x0 or 0x1A00_0000. Right click on the data field and select unsigned long.
3. Address 0 is the initial Stack Pointer: 0x1000_0438:
4. Address 4 is the initial PC: 0x1A00_0315 – 1.

TIP: LSBit of address 4 indicates a Thumb2 instruction. So subtract 1 to get the actual address = 0x1A00_0314.

5. Note this matches the address of the first address in the Trace data window.
6. You can scroll down to see every instruction executed.
7. You can search for data in a record or frame, save to a file and clear the window.

Memory 1

Address: 0

```
0x00000000: 10000438 1A000315 1A00031D 1A00031F
0x00000010: 1A000321 1A000323 1A000325 53FFE90E
0x00000020: 00000000 00000000 00000000 1A000327
0x00000030: 1A000329 00000000 1A00032B 1A000F6D
```

Call Stack + Locals | Memory 1

Single-Step from RESET:

1. Exit Debug mode by clicking on the Debug icon.
2. Select Target Options and select the Debug tab.
3. Unselect Run to Main. Run to main()
4. Enter Debug mode by clicking on the Debug icon. The Trace Data window is blank because no instructions have been executed.
5. Click on Step (or F11) and the first instruction, a LDR is now displayed:
6. Click on Step multiple times and watch the program progress.

Trace Data

Display: All ubfx in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
0.000 000 017 s	X : 0x1A000314	LDR r0,[pc,#24] ; @0x1A000330	LDR R0, =SystemInit	__asm_0x1A000314

7. To get to main() quickly, type g,main in the Command window and press Enter:

```
>g,main
<break address>
```

TIP: At this point, you can also click on the RUN icon and the program will run forever.

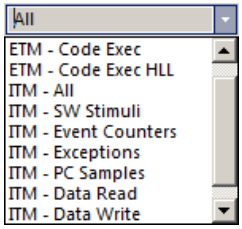
1. Select Target Options and select the Debug tab. Select Run to Main. Run to main() and then OK.

4) Finding the Trace Frames you are looking for:

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

Trace Filters:

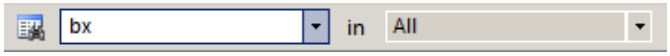
In the Trace Data window you can select various types of frames to be displayed. Open the Display: box and you can see the various options available as shown here: These filters are post collection. Future enhancements to µVision will allow more precise filters to be selected.



TIP: The ITM prefix signifies all SWV frames in this perspective.

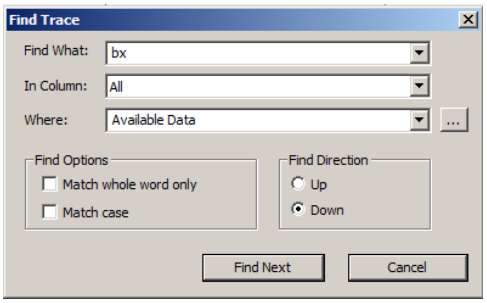
Find a Trace Record:

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



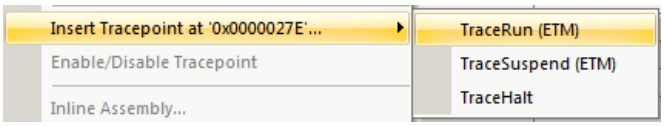
Note you can select properties where you want to search in the "in" box. All is shown in the screen above

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



5) Trace Triggering:

µVision has three trace triggers currently implemented:
TraceRun: Starts ETM trace collection when encountered.
TraceSuspend: Stops ETM trace collection when encountered. TraceRun has to have been encountered for this to have an effect.
These two commands have no effect on SWV or ITM. TraceRUN starts the ETM trace and TraceSuspend stops it.
TraceHalt: Stops ETM trace, SWV and ITM. Can be resumed only with a STOP/RUN sequence. TraceStart will not restart this.



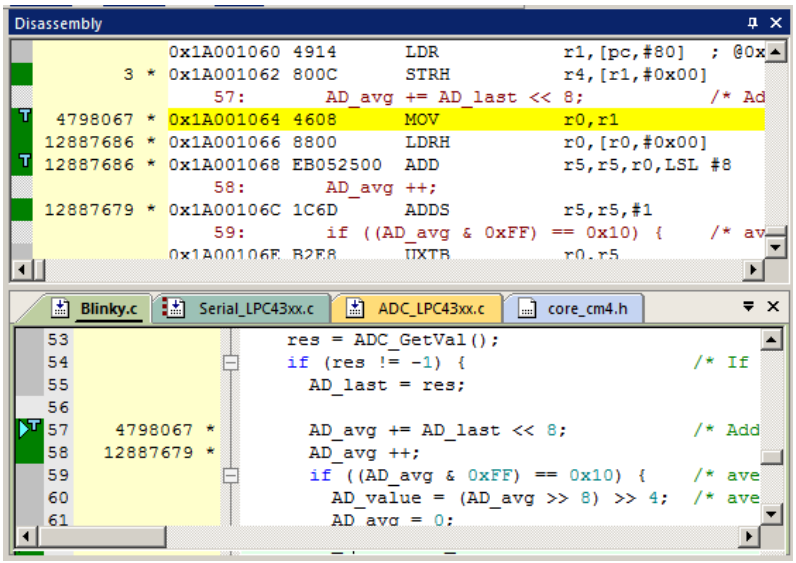
How it works:

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

The next page has a real example of Trace Triggers.

6) Setting Trace Triggers:

1. With Blinky in Debug mode, click on the C source line AD_avg += AD_Last << 8; near line 57 in Blinky.c. This is shown in the Blinky.c window below:
2. This source line will then be highlighted in the Disassembly window as shown. Instructions at 0x1A00_1064 through 0x1A00_1068 are displayed as belonging to this C source line.
3. Right-click on the instruction MOV and select Insert Tracepoint at 0x1A001064 and select **TraceRun.** A cyan **T** will appear as shown:
4. Right-click on the instruction ADD and select Insert Tracepoint at 0x1A001068 and select **TraceSuspend.** A cyan **T** will appear as shown:
5. Clear the Trace Data window. 
6. RUN the program and after a few seconds STOP it. Examine the Trace Data window as shown below:

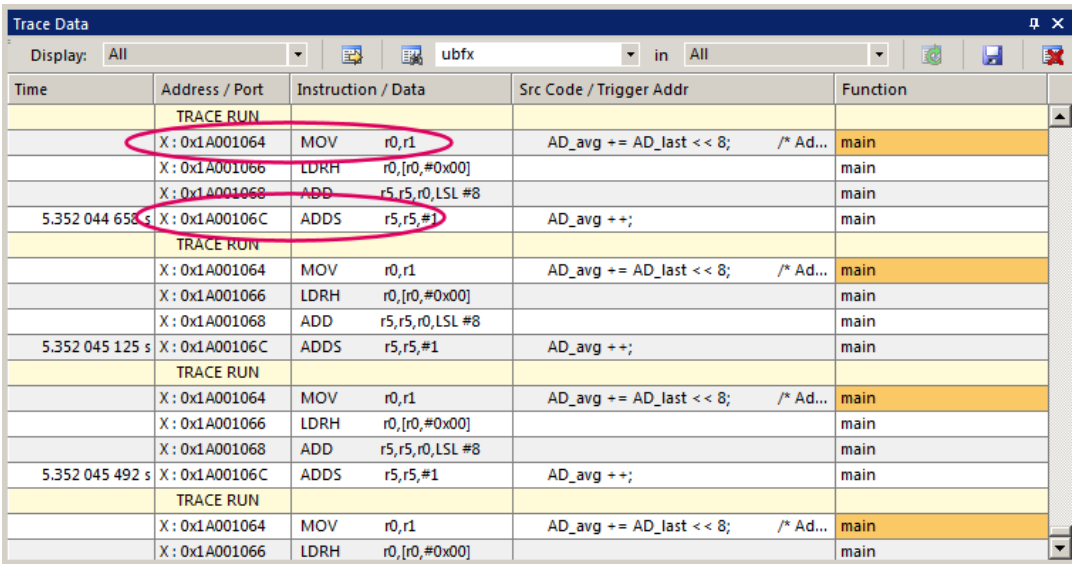


You can see where the trace started on 0x1A00_1064 and stopped on 0x1A00_106C multiple times shown with the red arrows.

There is a skid of one instruction. You might be able to get it exactly if you move the TracePoints around…or accept this. It is still useful.

TIP: The ETM trace will collect everything between the TraceRun and TraceSuspend points. This will include any branches and exception calls.
Once started, the trace will collect frames until it encounters a TraceSuspend or a TraceHalt point.

When you are done, right-click on each trigger and select Remove Tracepoint at '0x*address*'

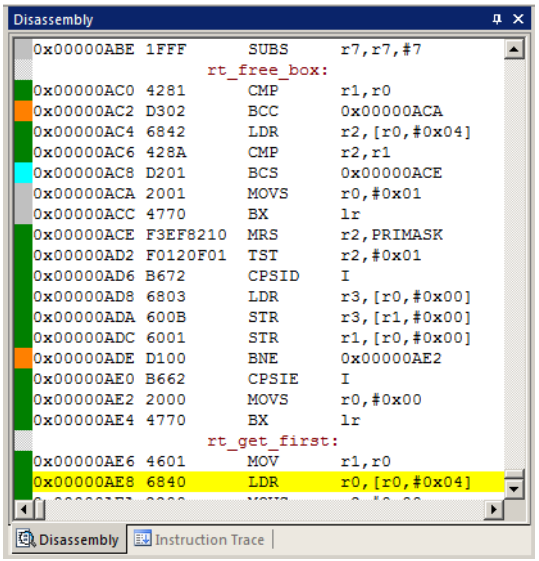


7) Code Coverage:

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

Colour blocks indicate which assembly instructions have been executed.

1. Green: this assembly instruction was executed.
2. Gray: this assembly instruction was not executed.
3. Orange: a Branch is always not taken.
4. Cyan: a Branch is always taken.
5. Light Gray: there is no assembly instruction here.
6. RED: Breakpoint is set here. (is actually a circle)
7. Next instruction to be executed.



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

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

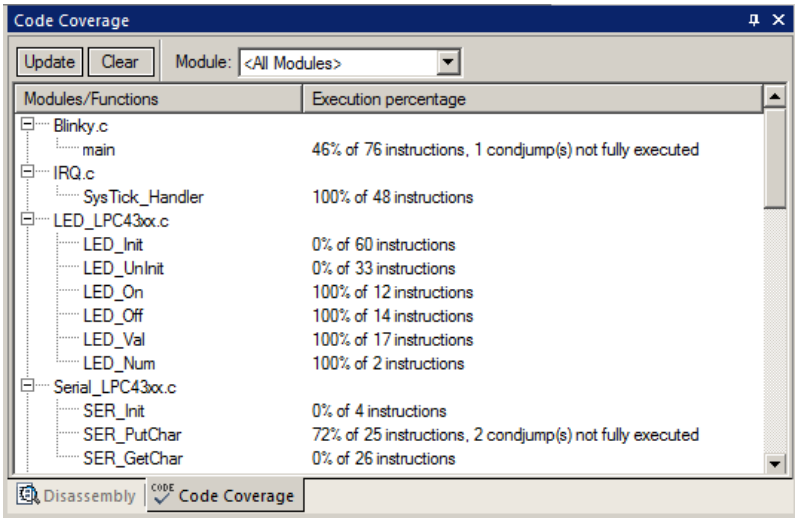
Why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

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

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

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

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



8) Saving Code Coverage information:

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

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

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

You can Save Code Coverage in two formats:

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

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

```
\\Blinky\Blinky.c\main - 46% (35 of 76 instructions executed)
    1 condjump(s) or IT-bcock(s) not fully executed
 NE |  0x1A000FE8 main:
 NE |  0x1A000FE8 2500      MOVS          r5,#0x00
 EX |  0x1A001056 F7FFFE8A  BL.W          ADC_GetVal (0x1A000D6E)
 NT |  0x1A00105E D00D      BEQ           0x1A00107C
 EX |  0x1A001070 2810      CMP           r0,#0x10
 FT |  0x1A001072 D103      BNE           0x1A00107C
 \\Blinky\LED_LPC43xx.c\LED_UnInit - 0% (0 of 33 instructions executed)
 NE |  0x1A000EAA LED_UnInit:
 NE |  0x1A000EAA 482D      LDR           r0,[pc,#180]  ; @0x1A000F60
\\Blinky\Serial_LPC43xx.c\SER_PutChar - 72% (18 of 25 instructions executed)
    2 condjump(s) or IT-bcock(s) not fully executed
 EX |  0x1A000DB0 SER_PutChar:
 EX |  0x1A000DB0 B510      PUSH          {r4,lr}
 EX |  0x1A000DB4 4B1D      LDR           r3,[pc,#116]  ; @0x1A000E2C
```

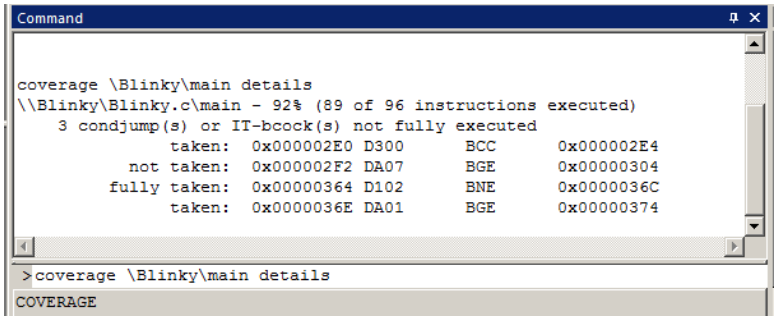
2. The command **`coverage asm`** produces this listing (partial is shown):

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

The first column above describes the execution as follows:

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

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



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

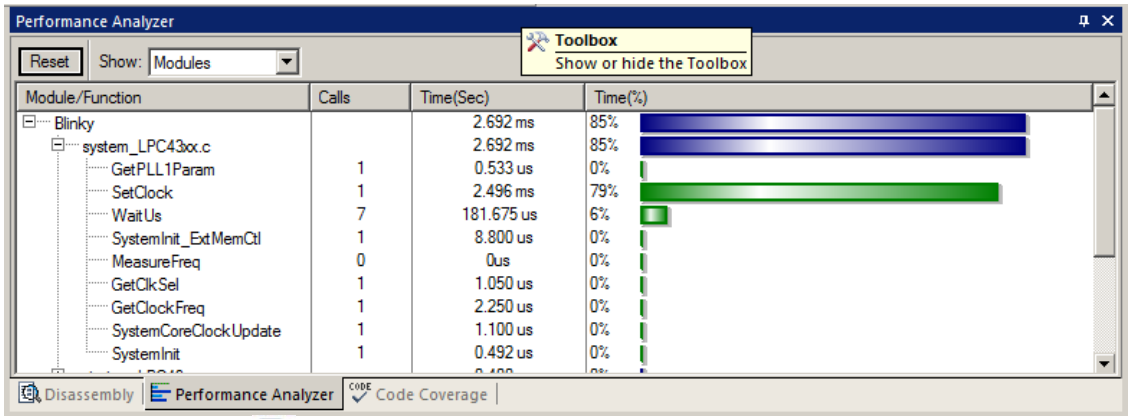
You can enter the command **`coverage`** with various options to see what is displayed in the Command window and saved.

9) Performance Analysis (PA):

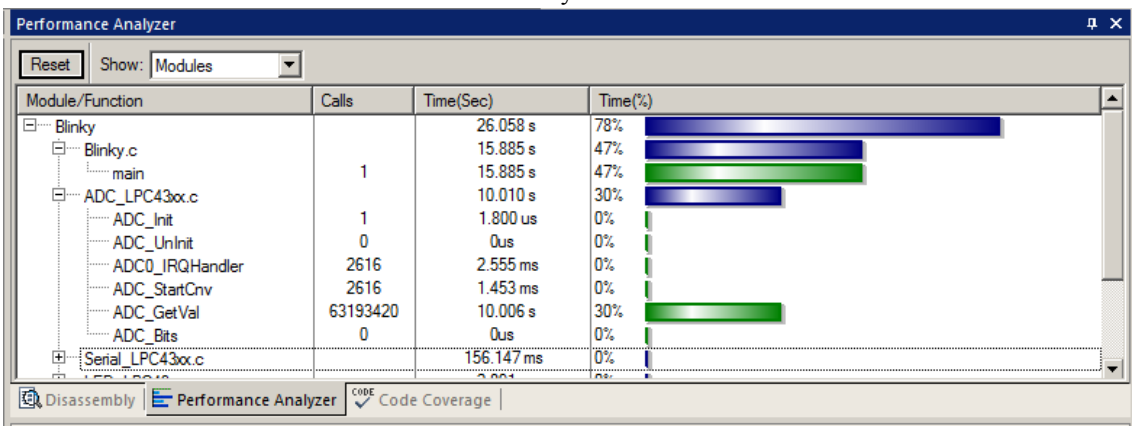
Performance Analysis tells you how much time was spent in each function. It is useful to help optimize your code for speed.

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

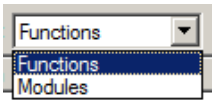
1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it. This clears the PA window and resets the LPC4300 processor and reruns it to main(). You can also click on the RESET icon and enter g,main in the Command window.
4. Do ***not*** click on RUN yet Expand some of the module names as shown below.
5. Times and number of calls has been collected in this short run from RESET to main().



6. Click on the RUN icon.
7. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the information. No code stubs are needed in your source files.



8. Select Functions from the pull down box as shown here and notice the difference.

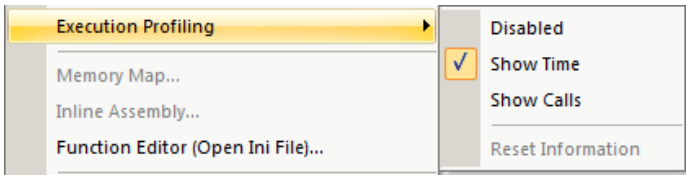


9. Click on the PA RESET icon. Reset Watch as new data is displayed in the PA window.
10. When you are done, exit Debug mode.

TIP: The Performance Analyzer uses ETM to collect its raw data.

10) Execution Profiling:

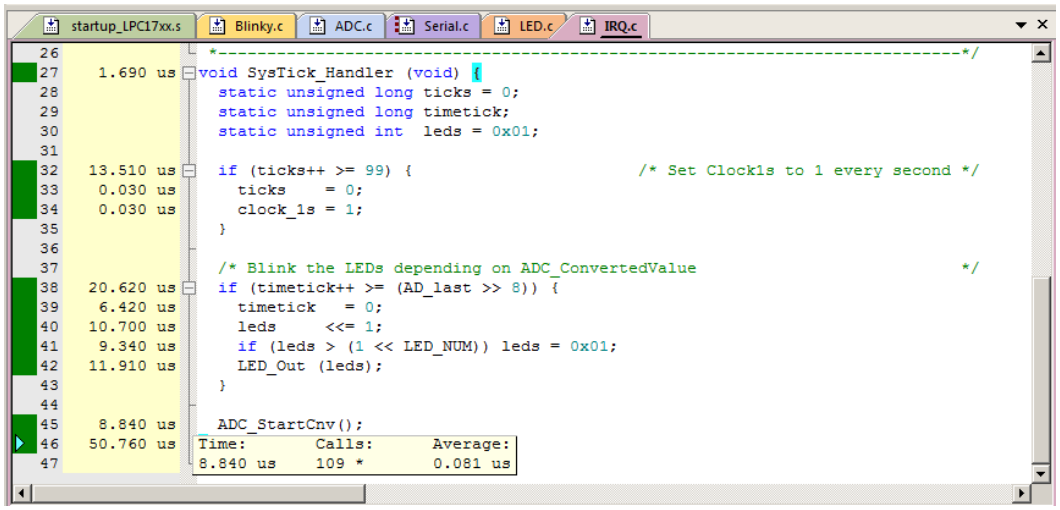
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.



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

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

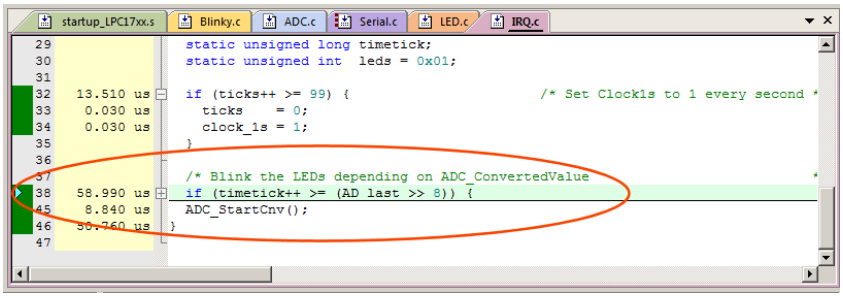
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



Outlining:

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

1) Click in the square near the while(1) loop near line 38 as shown here:
2) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
3) Click on the + to expand it.
4) Stop the program and exit Debug mode.

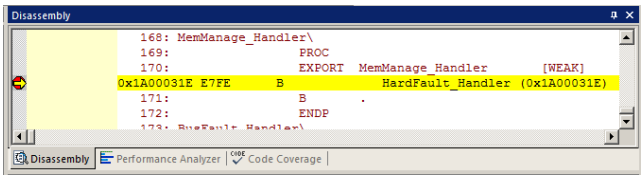


11) In-the-Weeds Example:

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

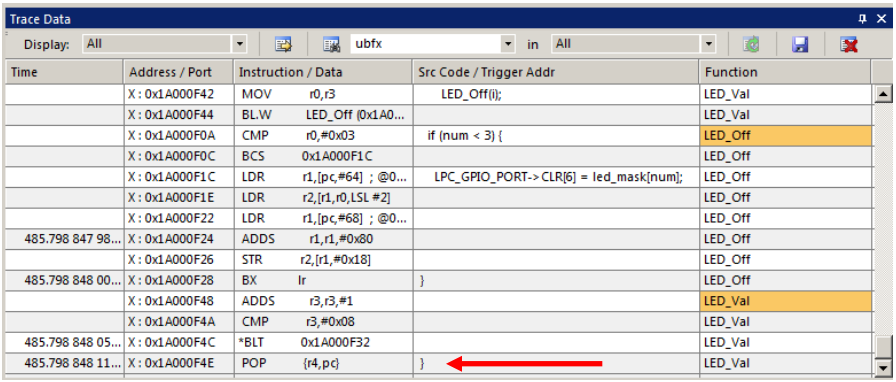
If a Hard Fault occurs in our example, the CPU will end up at 0x1A00_031E as shown in the disassembly window below. This is the Hard Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. The Trace Data window at the bottom shows this branch forever. This is not very useful or practical.

This exception vector is found in the file startup_LPC43xx.s. If we set a breakpoint by clicking on the Hard Fault handler and run the program: at the next Bus Fault event the CPU will again jump to its handler.



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

1. Use the Blinky example from the previous exercise. Enter Debug mode.
2. Locate the Hard fault vector near address 0x1A00_031E in the Disassembly window or near line 166 in the file startup_LPC43xx.s.
3. Set a breakpoint at this point. A red circle will appear.
4. Find the function LED_Val in the file LED_LPC43xx near line 114. Set a breakpoint on the start of LED_Val.
5. Click on RUN. The program will soon stop here. Remove the breakpoint.
6. Clear the Trace data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
7. In the register window, double-click on R14 (LR) register and set it to zero. LR contains the return address from a subroutine. This is guaranteed to cause a Hard Fault when the processor tries to execute an instruction at 0x1FFF 0468. (initial SP). This will cause a real and serious problem.
8. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
9. At the end of the Trace Data window you will find the offending instruction (a POP) at the end.
10. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.
11. Remove the breakpoint and click on RUN and then STOP.
12. You will now see all the Hard Fault branches as shown in the bottom screen:

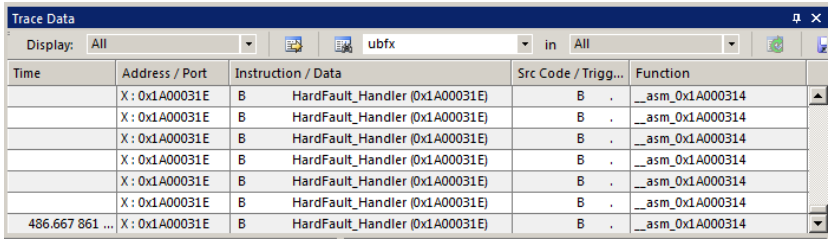


Time	Address / Port	Instruction / Data		Src Code / Trigger Addr	Function
	X : 0x1A000F42	MOV	r0,r3	LED_Off(i);	LED_Val
	X : 0x1A000F44	BL.W	LED_Off (0x1A0...		LED_Val
	X : 0x1A000F0A	CMP	r0,#0x03	if (num < 3) {	LED_Off
	X : 0x1A000F0C	BCS	0x1A000F1C		LED_Off
	X : 0x1A000F1C	LDR	r1,[pc,#64] ; @0...	LPC_GPIO_PORT-> CLR[6] = led_mask[num];	LED_Off
	X : 0x1A000F1E	LDR	r2,[r1,r0,LSL #2]		LED_Off
	X : 0x1A000F22	LDR	r1,[pc,#68] ; @0...		LED_Off
485.798 847 98...	X : 0x1A000F24	ADDS	r1,r1,#0x80		LED_Off
	X : 0x1A000F26	STR	r2,[r1,#0x18]		LED_Off
485.798 848 00...	X : 0x1A000F28	BX	lr	}	LED_Off
	X : 0x1A000F48	ADDS	r3,r3,#1		LED_Val
	X : 0x1A000F4A	CMP	r3,#0x08		LED_Val
485.798 848 05...	X : 0x1A000F4C	*BLT	0x1A000F32		LED_Val
485.798 848 11...	X : 0x1A000F4E	POP	{r4,pc}	}	LED_Val

This is admittedly a contrived example but it clearly shows how quickly and easily ETM trace can help you solve tricky program flow problems.

.

TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 166… and select TraceHalt. When Hard Fault is reached is reached, trace collection will halt but the program will keep executing the B instructions forever.

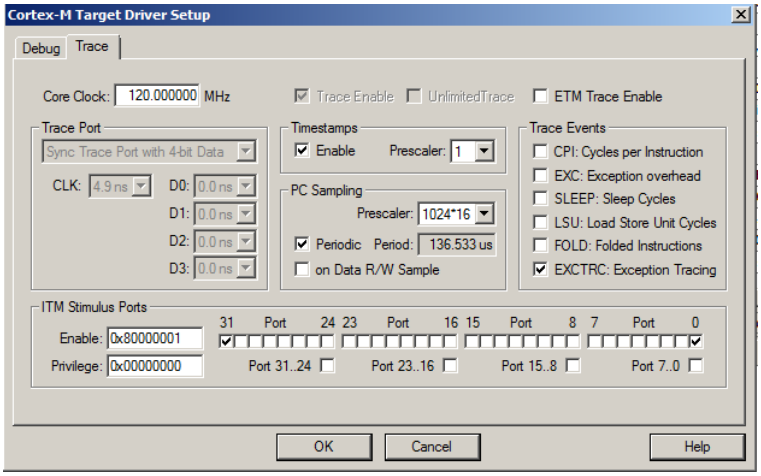


Time	Address / Port	Instruction / Data		Src Code / Trigg...	Function
	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314
	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314
	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314
	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314
	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314
	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314
486.667 861 ...	X : 0x1A00031E	B	HardFault_Handler (0x1A00031E)	B .	__asm_0x1A000314

12) Serial Wire Viewer (SWV) with ULINK*pro* and the Trace Port:

Currently it is possible to view Instruction trace or SWV trace but not both at the same time in the Trace Data window. We have seen Instruction Trace in the last few pages. Now, we will test out the SWV capabilities.

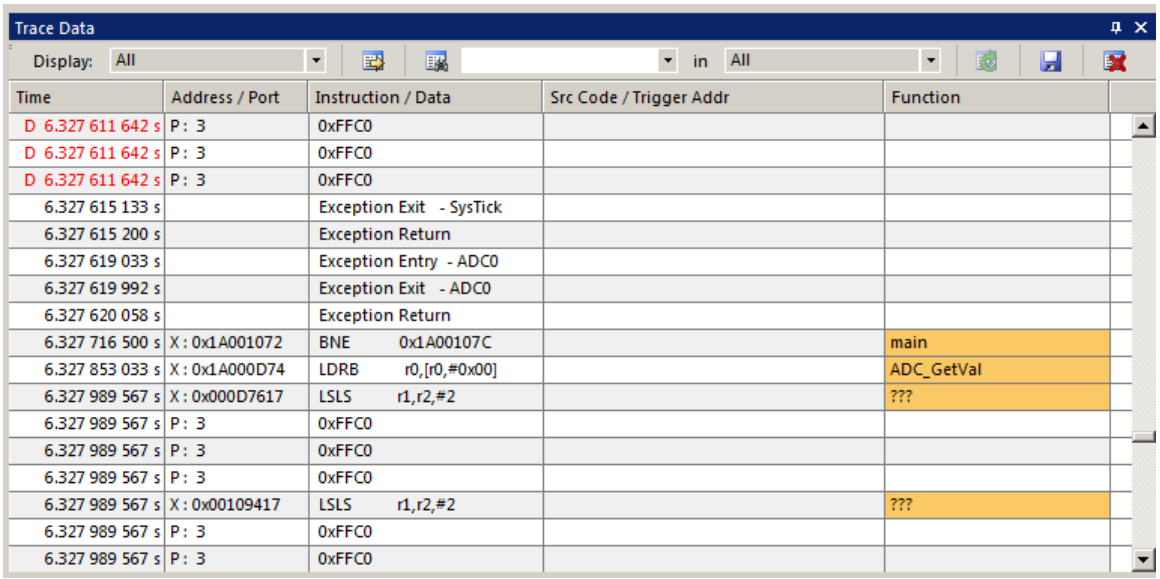
1. Use the Blinky_Ulp project as used in previous pages. Stop the program and stay in Debug mode.
2. From the main menu, select Debug and then Debug Settings… Click on the Trace tab to open this window:
3. Unselect ETM Trace Enable.
4. Select Periodic, EXTRC and ITM Port 31 and 0 as shown here:
5. Click on OK twice. Click on RUN.
6. Click on STOP.
7. The Trace Data window will display frames as shown below:



Three types of frames displayed:

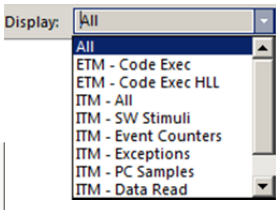
1. P: 3 – ITM Port 31 output to Debug printf Viewer. To see this output select View/Serial Windows and then choose Debug printf Viewer.
2. Exception for SysTick and ADC0.
3. X: PC Samples. This is the Periodic selection. Double-click on a PC Sample to highlight this instruction in the Disassembly and source windows.

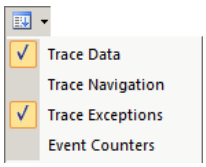
TIP: The red D notation indicates the timestamps may not be accurate. Notice in this case all three are the same value. This is the type of problem that occurs from SWV overload. Often these are not very serious and do not materially affect debugging. Reducing the number of SWV features chosen usually helps. Sometimes sampling a variable helps.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
D 6.327 611 642 s	P: 3	0xFFC0		
D 6.327 611 642 s	P: 3	0xFFC0		
D 6.327 611 642 s	P: 3	0xFFC0		
6.327 615 133 s		Exception Exit - SysTick		
6.327 615 200 s		Exception Return		
6.327 619 033 s		Exception Entry - ADC0		
6.327 619 992 s		Exception Exit - ADC0		
6.327 620 058 s		Exception Return		
6.327 716 500 s	X: 0x1A001072	BNE 0x1A00107C		main
6.327 853 033 s	X: 0x1A000D74	LDRB r0,[r0,#0x00]		ADC_GetVal
6.327 989 567 s	X: 0x000D7617	LSLS r1,r2,#2		???
6.327 989 567 s	P: 3	0xFFC0		
6.327 989 567 s	P: 3	0xFFC0		
6.327 989 567 s	P: 3	0xFFC0		
6.327 989 567 s	X: 0x00109417	LSLS r1,r2,#2		???
6.327 989 567 s	P: 3	0xFFC0		
6.327 989 567 s	P: 3	0xFFC0		

4. In the Display: pull-down menu select various items to see how the filters work. Not shown here are Data Write and Data Read/Write selections.
5. Click on the small arrow beside the Trace icon and select Exceptions:
6. The Trace Exceptions window will open and display exceptions. This will update while the program is running. Click on RUN to see this window operate.





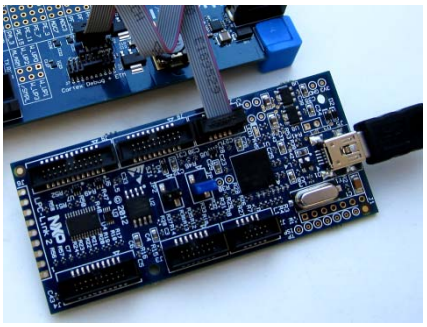
1) Configuring the LPC-Link 2 as a CMSIS-DAP Compliant debug adapter:

Note: If you are not using the LPC-Link 2 debug adapter you can skip this page.

LPC-Link 2 provides either a Segger J-Link Lite or CMSIS-DAP mode. µVision supports both. CMSIS-DAP mode provides MTB support on LPC800. SWV and ETM are not supported. SWV support is scheduled for late 2014.
All files needed to configure LPC-Link 2 are here: http://www.lpcware.com/lpclink2-config-tool.

There are three steps to configure LPC-Link 2 with µVision:

1. Install the NXP USB drivers. The filename currently is lpc18xx_43xx_winusb_drivers_v1.00.zip. Extract these into a temp directory and plug the LPC-Link 2 into the PC USB and install the USB drivers in the usual way.
2. **.NET:** Your PC must have Microsoft .NET Framework 4 or later. When you run the LPC configuration tool in Step 3 and you do not have the correct version of .NET, there will be an error. If you do not receive an error, then your .NET version is correct. You can get .NET here: www.microsoft.com/net/download.
3. Program the LPC-Link to either J-Link or CMSIS-DAP mode. A program called LPC-Link 2 Configuration Tool is used. Obtain it from the website listed above: LPC-Link 2 Config tool.zip Extract it into a suitable directory on your PC. Execute it by double-clicking on it or in the fashion you prefer.



Configuring LPC-Link 2 for CMSIS-DAP:

4. Remove jumper JP1. Whether JP1 installed or not will be sensed by the processor during its startup routines.
5. Connect LPC-Link 2 to your PC's USB port. The USB drivers in Step 1 above must be installed and must connect.
6. Start LPC-Link 2 Config.exe by double-clicking on it. This file runs alone, it does not install like most Windows programs do. The window below opens up. Follow the steps – it is fairly simple and reliable.

TIP: JP1 must be either installed or not when the power is applied or cycled. In Step 1 below it is assumed the LPC-Link 2 is not connected to the USB port when you are instructed to remove JP1. If it is powered, just cycle the power with JP1 off.

TIP: If the instructions in Step 2 below are greyed out, this usually means the LPC-Link 2 is not connected to the PC USB drivers. You will have to fix this before you can continue. Solutions include: check the USB port or re-install the drivers.

TIP: Step 3 assumes power will be applied via the USB cable. After replacing JP1, the USB cable must be plugged in.

Configuring µVision for LPC-Link 2:

In µVision, Select Target Options  and select the Debug tab. In the drop-down menu box select CMSIS-DAP Debugger as shown at the bottom of this page: You will also have to select the Flash programming algorithm in the Utilities tab.

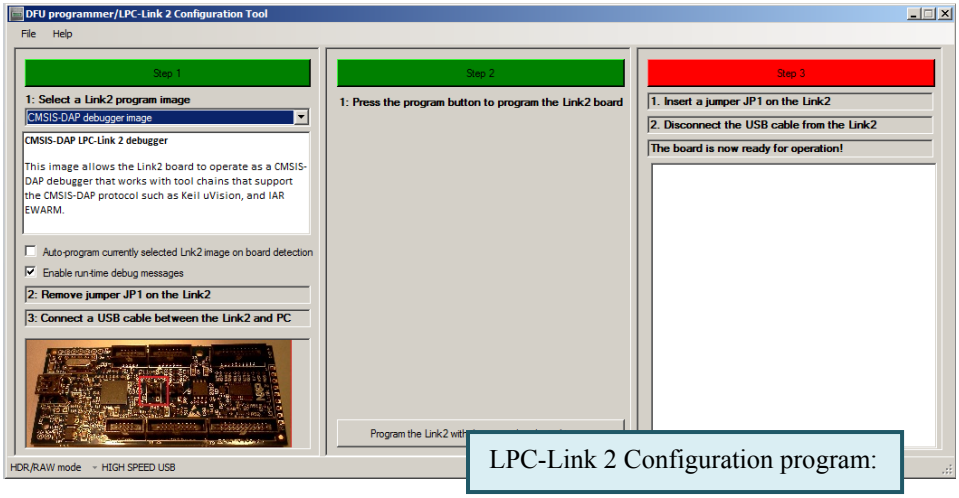
You only need to program the LPC-Link 2 once. The firmware will stay programmed in Flash..

Flash Programming:

Complete section **2) Configure the Keil Flash Programmer:** on the next page. Select CMSIS-DAP instead of ULINK2/ME in step 7.

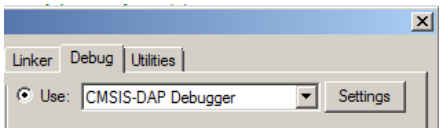
Your LPC-Link 2 is now configured to work with µVision.

TIP: JP2 supplies power to your target board if installed. You can leave JP2 on or off for all the programming steps above.



LPC-Link 2 Configuration program:

Selecting CMSIS-DAP:



2) Configuring ULINK2 or ULINK-ME and µVision:

If you are not using a ULINK2 or a ULINK-ME. you can skip this page:

The examples are pre-configured to use the ULINK2 or ULINK-ME by default. This is for reference.

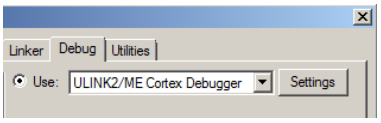
It is easy to select a USB debugging adapter in µVision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are selected using the Debug and Utilities tabs.

This document will use a ULINK2 or ULINK-ME as described. You can substitute a ULINK*pro* with suitable adjustments.

Serial Wire Viewer is supported by all three adapters. ULINK2 and ULINK-ME are essentially the same devices electrically and any reference to ULINK2 here includes the ME. The ULINK*pro*, which is a Cortex-M ETM trace adapter, performs like a ULINK2 or ULINK-ME with the advantages of faster programming time and an enhanced instruction trace window.

1) Select the debug connection to the target:

1. Assume the ULINK2 is connected to a powered up Keil target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK2 is shown connected to the Keil MCB4300 board on page 4.



2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select ULINK2/ME Cortex Debugger as shown here:
3. Select Settings and the next window below opens up. This is the control panel for the ULINK 2 and ULINK-ME (they are the same).
4. In **Port:** select SWJ and SW. Serial Wire Viewer (SWV) will not work with JTAG selected.

TIP: You can do regular debugging using either JTAG or SWD. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will operate only in SW (SWD) mode. To use JTAG and SWV, use a ULINK*pro*.

5. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board and try again.

TIP: To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.

2) Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ULINK similar to Step 2 above.

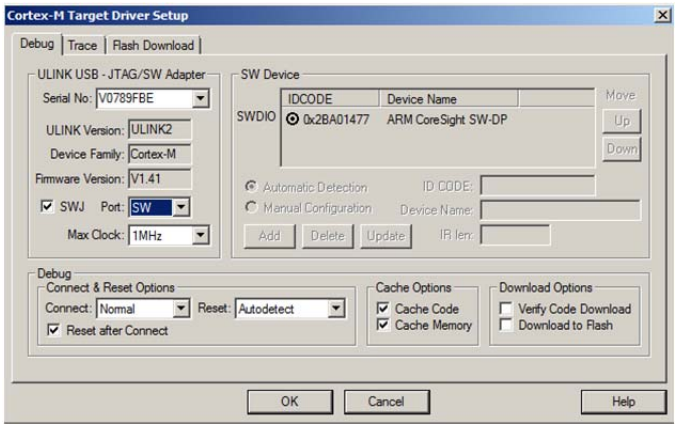
TIP: You might have to add an .ini file. See the Flash programming instructions on page 5. This depends on which memory use: internal, external or SPIFI flash.

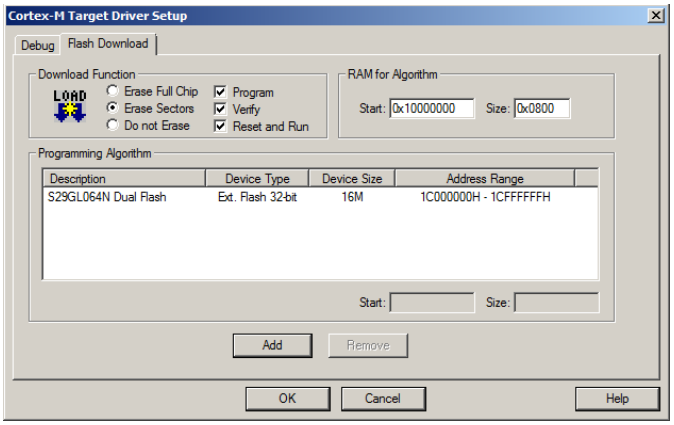
8. Click Settings to select the programming algorithm.
9. If an algorithm is not already selected, select Add and select S29GL064N Dual Flash as shown below or the one for your processor: The example shown is for the MCB4300 external Flash.
10. Click on OK once when the algorithm is entered.

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

11. Click on OK to return to the main screen.
12. You have successfully connected to the MCB4300.
13. At this point you can compile source code, program it into Flash, enter Debug mode, start and stop your program and set/unset breakpoints plus much more.

TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV).





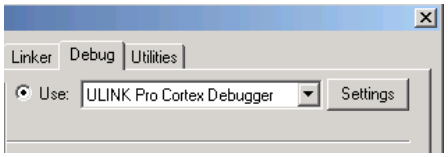
3) Configuring ULINK*pro* and µVision:

Many Keil examples are configured for* ULINK*pro* *by default. These projects have a Ulp suffix.

1. Assume a ULINK*pro* is connected to a powered up MCB4300 target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK*pro* is shown connected to the Keil MCB4300 board on page 4.

1) Select the debug connection to the target:

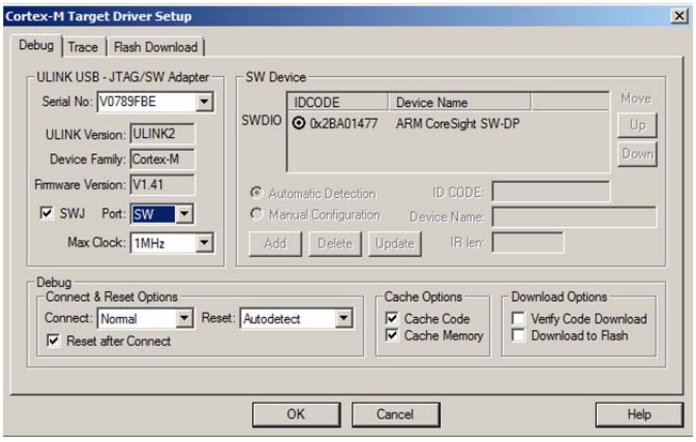
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select the ULINK Pro Cortex as shown here:
3. Select Settings and Target Driver window below opens up:
4. In **Port:** select SWJ and SW. SWV will not work with JTAG selected unless you are using the 4 bit Trace Port. See the second **TIP:** below:



5. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board and try again.

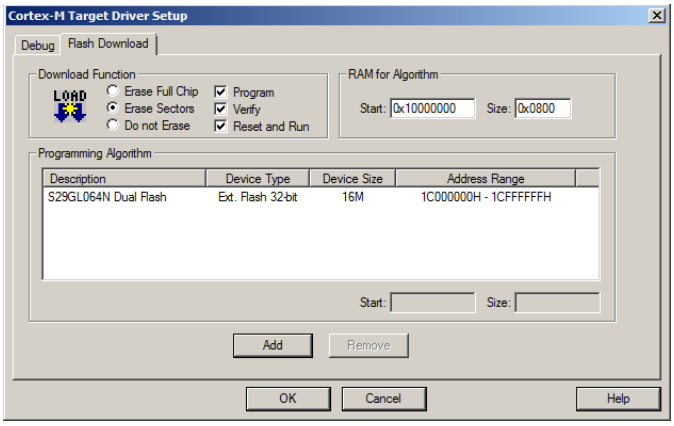
TIP: To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.

TIP: You can do regular debugging using JTAG or SWD. SWD (also called SW) and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode unless the ULINK*pro* is using the Trace Port with ULINK*pro* to output the trace frames rather than the 1 bit SWO pin. This is selected in the Trace tab and is discussed on page 24.



2) Configure the Keil Flash Programmer:

1. Click on OK once and select the Utilities tab.
2. Select the ULINK*pro* similar to Step 2 above.
3. Click Settings to select the programming algorithm.
14. If one is not visible click Add and select S29GL064N Dual Flash as shown below or the appropriate one for your processor: The example shown is for the MCB4300 external Flash.
4. Click on Add to select your chosen algorithm.
5. Click on OK once.



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

6. Click on OK to return to the µVision main screen.
7. You have successfully connected to the Keil target processor.
8. At this point you can compile source code, program then into Flash, enter Debug mode, start and stop these programs and set/unset breakpoints and much more.

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

TIP: A ULINK*pro* will act very similar to a ULINK2. The trace window (Trace Data) is quite different from the ULINK2 Trace Records as it offers additional features.

TIP: µVision windows can be floated anywhere. You can restore them by setting Window/Reset Views to default. Using two monitors is also supported.

4) Serial Wire and ETM Trace Summary:

We have several basic debug systems as implemented in NXP Cortex-M3 and Cortex-M4 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 µVision.
3. Memory Reads and Writes in/out the JTAG/SWD ports. The Memory and Watch windows use this technology.
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.
5. ETM provides a record of all instructions executed.

These are all completely controlled through µVision via a ULINK.

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

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

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

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes). How I did I get to this Fault vector ?
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs). ***How did I get here ?***
4) A corrupted stack.
5) Out of bounds data. Uninitialized variables and arrays.
6) Stack overflows. What causes the stack to grow bigger than it should ?
7) **Runaway programs:** your program has gone off into the weeds and you need to know what instruction caused this. ***This is probably the most important use of trace. Needs ETM to be most useful.***
8) Communication protocol and timing issues. System timing problems.

- Trace adds significant power to debugging efforts. Tells you where the program has been.
- Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time before any consequences are seen.
- Or where the state of the system disappears with a change in scope(s).
- Plus - don't have to stop the program. Crucial to some. Not for ETM.
- A recorded history of the program execution ***in the order it happened.***
- Trace can often find nasty problems very quickly.
- Profile Analysis and Code Coverage is provided. Available only with ETM trace.

What kind of data can the Serial Wire Viewer display ?

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

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

1. Can't see local variables. (just make them global or static).
2. Can't see register operations. PC Samples records some of the instructions but not the data values.

3. SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU. SWV and ETM can only see CPU actions.

5) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite (Evaluation version) $0
- ***NEW !!*** MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit) - $3,200
- MDK-Standard (unlimited compile and debug code and data size) - $4,895
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) $9,500

For special promotional or quantity pricing and offers, contact Keil Sales or your favourite distributor.

USB-JTAG adapter (for Flash programming too)

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

All ULINK products support MTB with LPC800 Cortex-M0+.

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

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

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

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

All products are available from stock.

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

Call Keil Sales for special university pricing. Go to www.arm.com/university

Keil supports many other NXP processors including ARM7™and ARM9™ series processors. See the Keil Device Database® on www.keil.com/dd for the complete list of NXP support. This information is also included in MDK.

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.

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

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

For more information:

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

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

International Distributors: www.keil.com/distis/ See www.embeddedsoftwarestore.com

For more NXP specific information: visit www.keil.com/NXP

CMSIS: www.arm.com/cmsis **ARM forums:** forums.arm.com **Keil Forums:** www.keil.com/forum/