NXP LPC1700 Lab *for the Keil MCB1768/58™*

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

For the Keil MCB1700™ Evaluation Board Version 4.94 by Robert Boys bob.boys@arm.com

For the Keil MCB1700 Evaluation Board

Introduction:

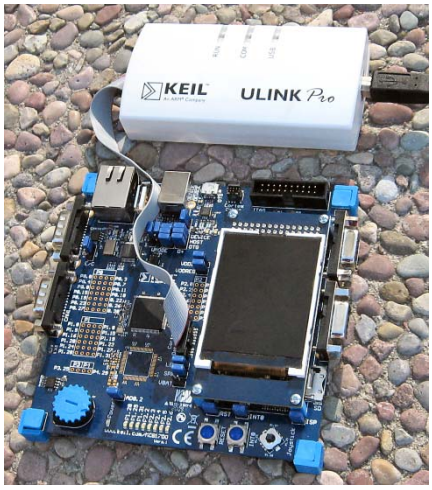
The latest version of this document is here: www.keil.com/appnotes/docs/apnt_246.asp

The purpose of this lab is to introduce you to the NXP Cortex®-M3 processor using the Keil MDK-ARM™ Microcontroller Development Kit featuring µVision®. MDK also contains a simulator. We will use the Serial Wire Viewer (SWV) on the LPC1768 or LPC1765 rather than the simulator in this lab. Keil MDK supports all NXP ARM processors including Serial Wire Viewer and ETM trace. Check the Keil Device Database® for NXP on www.keil.com/dd/chips/nxp/arm.htm.

Keil MDK comes in an evaluation version that limits code and data size to 32 Kbytes. Nearly all Keil examples will compile within this 32K boundary. The addition of a license number will turn it into the full, unrestricted version. Keil also provides middleware in MDK-Professional™. This package includes a TCP/IP stack, CAN drivers, a Flash file system and USB drivers. See the last page of this document for details on MDK and ARM contact information.

For more information: www.keil.com/nxp, forums.arm.com and www.arm.com/cmsis

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, Assembler and Linker 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. A full feature Keil RTOS called RTX is included with MDK and includes source code with all versions of MDK. RTX now comes free with a BSD type license. See www.keil.com/demo/eval/rtx.htm.
3. Two RTX Kernel Awareness windows are updated in real-time. Kernel Awareness exists for Keil RTX, CMX, Quadros and Micrium. MDK can compile all RTOSs written in C.
4. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™
5. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.

This document details these features (plus more):

1. Serial Wire Viewer (SWV) with ULINK2, ULINK-ME and ULINK*pro*. ETM Trace using ULIN*Kpro*.
2. Real-time Read and Write to memory locations 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.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also called Access Breaks).
4. RTX Viewer: Two kernel awareness windows for the Keil RTX RTOS that update while the program is running.
5. A DSP example using the ARM DSP libraries for Cortex-M0, Cortex-M0+, Cortex-M3 and Cortex-M4.

Serial Wire Viewer (SWV): (Data Trace) *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.

SWV does not steal any CPU cycles and is completely non-intrusive except for ITM Debug printf Viewer. SWV is provided by the Keil ULINK2, ULINK-ME, ULINK*pro* and the Segger J-Link. A ULINK2 is used for the Serial Wire Viewer exercises in this lab. An LPC-Link 2 can be used with this document but it currently does not support SWV.

Embedded Trace Macrocell™ (ETM): (Instruction Trace) *Use a ULINKpro for ETM.*

ETM displays all the executed instructions. As well as 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 support requires the ULINK*pro*. This document uses a ULINK*pro* for ETM.

Index:

1. Software Installation, JTAG and SWD Definitions, debug adapter connectors: 3
2. Debug Adapter Summary: ULINK2/ME, ULINK*pro*, J-Link, LPC-Link 2: 4
3. ***Blinky* example using the Keil MCB1700 board and ULINK2:** **5**
4. Hardware Breakpoints: 6
5. Call Stack + Locals Window: 6
6. Watch and Memory Windows and how to use them 7
7. How to view Local Variables in the Watch or Memory windows: 8
8. Configuring the Serial Wire Viewer (SWV) with the ULINK2: 9
9. Logic Analyzer: graphical data display using Serial Wire Viewer 10
10. Watchpoints: Conditional Breakpoints 11
11. **RTX_Blinky Example Program with Keil RTX RTOS: A Stepper Motor:** **12**
12. RTX Kernel Awareness example using Serial Wire Viewer 13
13. Logic Analyzer Window: view RTX variables real-time in a graphical format: 14
14. **External Interrupt Example: EXTI using SWV:** **15**
15. ITM (Instruction Trace Macrocell) 16
16. printf Statements using ITM: 17
17. **CAN (Controller Area Network)** **18**
18. Using Watchpoints and Serial Wire Viewer with CAN 19
19. **DSP SINE Example using ARM CMSIS-DSP Libraries:** **20**
    1) Running the DSP SINE example: 20
    2) Signal Timings in the Logic Analyzer (LA): 21
    3) RTX Tasks and System Awareness window: 21
    4) RTX Event Viewer (EV): 22
    5) Event Viewer Timing: 23
    6) Changing the SysTick Timer: 23
20. **Using a ULINK*pro*: Instruction Trace and more…** **24**
21. Using a ULINK*pro*: 24
22. Using the Blinky_Ulp example: 25
23. Finding the Trace Frames you are looking for: 28
24. Trace Triggering: 28
25. Setting Trace Triggers: 29
26. In-the-Weeds Example: 30
27. **Program Analysis** **31**
28. Code Coverage: 31
29. Saving Code Coverage information: 32
30. Performance Analysis (PA): 33
31. Execution Profiling: 34
32. **Creating a new project: Using the Blinky source files:** **35**
33. Serial Wire Viewer summary 36
34. **Keil Products and contact information** **37**

1) Software Installation:

This document was written with Keil MDK 4.72a which contains µVision4. MDK is available on the Keil website and the specific example files are included with this document. Example files are subject to improvement and can change. Use the files specified for this document. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence.

MDK 5 will be released late October 2013. See www.keil.com/mdk5.

If you have a previous version of MDK 4, do not uninstall it; just install the new version on top. If you want to refresh the examples, delete them in C:\Keil\ARM\boards\ and they will be replaced at install time.

You can use the evaluation version of MDK and a ULINK2, ULINK-ME, ULINK*pro*, Segger J-Link or a LPC-Link 2.

If you are using a Segger J-Link, you do not need to install any additional files. You will need to configure µVision to use the J-Link to run programs and program the Flash memory. This is easy to do. A J-Link (black case) Version 6 and later supports Serial Wire Viewer with Keil µVision. For LPC-Link 2, select CMSIS-DAP mode.

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.

CoreSight™, 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.

MCB1700 debug adapter connectors:

JTAG: standard 20 pin header for JTAG, SWD and SWV connections.

Cortex-Debug: compact ARM connector for JTAG, SWD and SWV connections.

Cortex-Debug+ETM: 20 pin compact JTAG, SWD, SWV and ETM connections.

http://infocenter.arm.com/help/topic/com.arm.doc.faqs/attached/13634/cortex_debug_connectors.pdf

www.keil.com/coresight/coresight-connectors/

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.

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.

ULINK*pro*

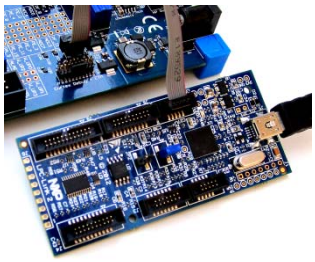
CMSIS-DAP:

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.

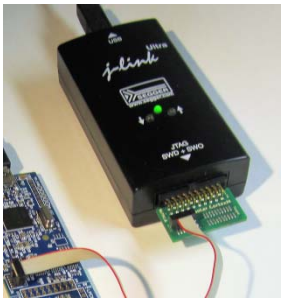
CMSIS-DAP provides run control debugging, Flash and RAM programming, Watchpoints and hardware breakpoints. DAP reads and writes in the Watch and Memory windows are updated in real-time as well as the RTX System kernel awareness.

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

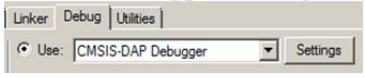
Segger J-Link:

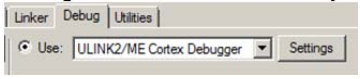
The fifth picture. µVision supports J-link and J-Link Ultra (black case only) Version 6.0 or later. The J-Link family supports all CoreSight components except MTB. This will be supported in the future. J-Link is configured similar to the ULINK2/ME. Select J-Link/J-Trace Cortex in the Debug and Utilities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


J-Link Ultra

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

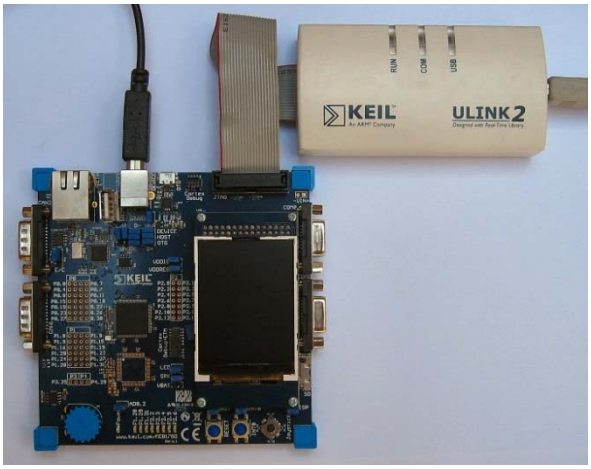
Starting with MDK 4.71, you can use either of these as a CMSIS-DAP compliant debugger or in standard ULINK2/ME mode. With a LPC800, CMSIS-DAP mode adds trace search, save and clear window options. This is the same window as with the ULINK*pro*. ULINK2 CMSIS-DAP can start and stop the CPU and set/unset 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 the standard ULINK2/ME, ULINKpro or J-Link (black case) V6 or higher.

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) *Blinky* example program using the Keil MCB1700 and ULINK2 or ULINK-ME:

Now we will connect up a Keil MDK development system using real target hardware and a ULINK2 or ULINK-ME. These examples will also run on the MCB1750 which uses a LPC1758 processor.



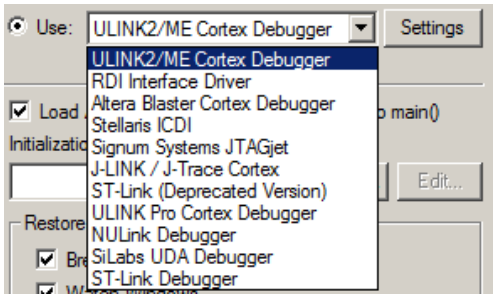
1. Connect the equipment as pictured here:
2. Start µVision4 by clicking on its desktop icon.

1. Select Project/Open Project.
2. Open the file C:\Keil\ARM\Boards\Keil\MCB1700\Blinky\Blinky.uvproj.
3. Make sure "LPC1768 Flash" is selected. This is where you select the Simulator or to execute a program in RAM or Flash. You can easily create your own Target Options.
4. Select Target Options icon and select the Debug tab. The USB adapter is selected in this window as shown below: ULINK2/ME will be pre-selected in this example project.

TIP: The Flash programming algorithm is selected in the Utilities tab. The ULINK2/ME is pre-selected for this project. You do not need to change this unless you are using a different adapter.

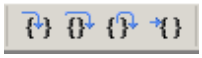
5. Select the Settings: box under the Debug tab. The window below opens up.
6. In **Port:** select SWJ and SW. You can use JTAG if you are not going to use Serial Wire Viewer (SWV).
7. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or it is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board.

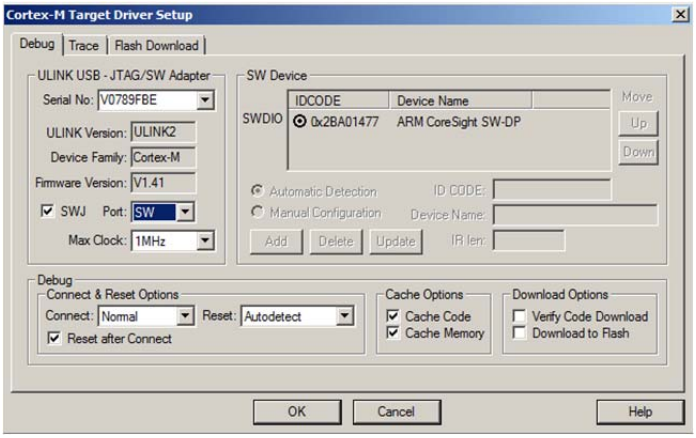
TIP: To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.



TIP: You can do regular debugging using JTAG. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode.

8. Click on OK twice to return to the main µVision screen.
9. Compile the source files by clicking on the Rebuild icon.
10. Program the LPC1700 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
11. Enter the Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
    Note: You only need to use the Load icon to download to FLASH and not to program the simulator or RAM.
12. Click on the RUN icon. Note: you stop the program with the STOP icon.
13. You can single-step with these icons: Hover your mouse over each icon to identify its function:





The LEDs on the MCB1700 will now blink at a rate according to the setting of the pot P7.

Now you know how to compile a program, load it into the LPC1700 Flash, run it and stop it.

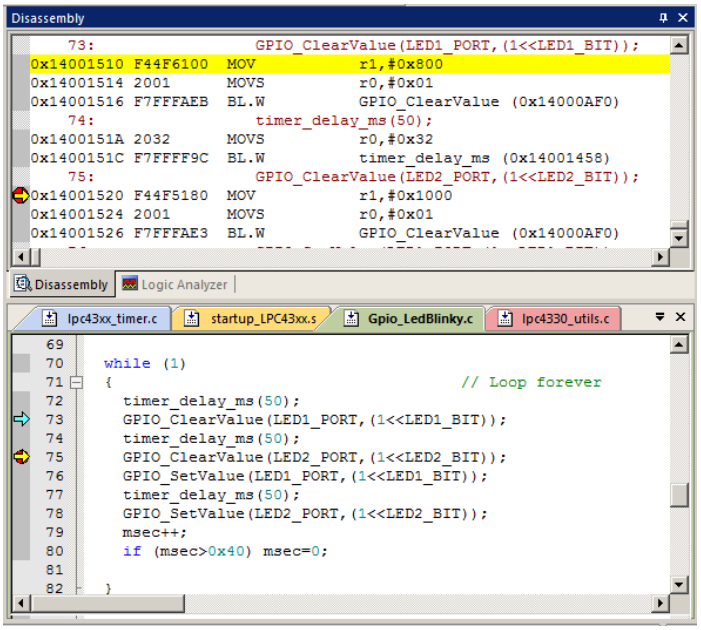
Note: This program is now running on the Cortex-M3 processor in the LPC1700. If you remove the ULINK2/ME, this program will run standalone as you have programmed it in the Flash. This is the complete development cycle.

4) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block somewhere appropriate in the while(1) loop between Lines 062 through 088 in the source file Blinky.c as shown below: Click on the Blinky.c tab if not visible.
2. A red circle is created and soon the program will stop at this point as shown below.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows. This instruction has not been executed yet.
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 indicated 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 other CoreSight features.
6. The LPC1700 family has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to.
7. Remove the breakpoint to continue.

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 Target Options  under the C/C++ tab when not in Debug mode.



5) Call Stack + Locals Window:

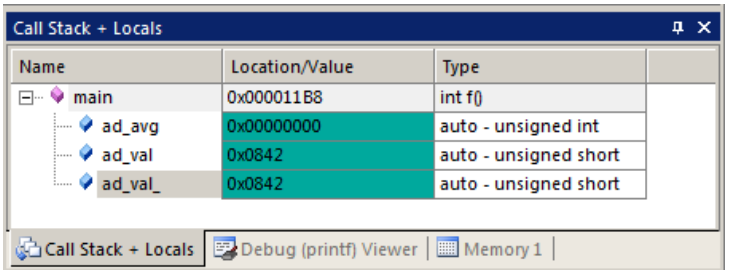
Local Variables:

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> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack window.

1. Click on the Call Stack + Locals tab. Stop the program if necessary.
2. Shown below is the Locals window in the Blinky program.
3. The contents of three local variables in the main() function are displayed as well as the function name(s).
4. Using RUN, Step, Step Over and Step Out to enter and exit various functions, this window will update.
5. Set a breakpoint at an appropriate place in GLCD_SPI_LPC1700.c source file and select RUN to see other Locals.

TIP: This is standard "Stop and Go" debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables updated in real-time while the program is running. No additions or changes to your code are required. Update while the program is running is not possible with local variables. They must be converted to global or static variables or be part of a structure or array so they always remain in scope.



Call Stack:

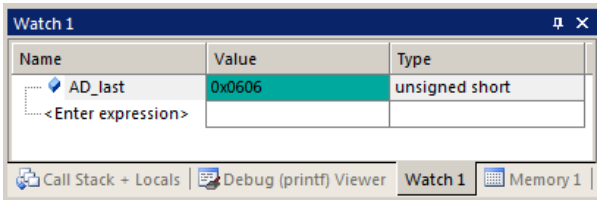
The list of stacked functions is displayed when the program is stopped. This is when you need to know which functions have been called and are stored on the stack.

6. **Remove all hardware breakpoints by clicking on its red circle !** Select Debug/Breakpoints (or Ctrl-B) to see how you can manage breakpoints such as using Kill All Breakpoints or temporarily unselect them.

6) 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 through the ARM CoreSight debugging technology that is part of most NXP Cortex-M processors. It is also possible to "put" or insert values into these memory locations in real-time. You can "drag and drop" variables into windows or enter them manually to configure them.

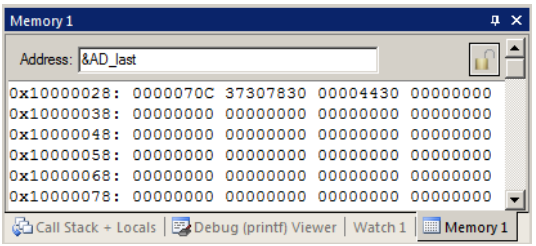
1. In the source file ADC.c is the global variable **`AD_last`** near line 21. Select File/Open to access ADC.c if needed.
2. Enter debug mode if not there already.  The program can be running or stopped for the following steps.
3. Right-click on AD_last and select Add 'AD_last' to … and select Watch 1. Watch 1 will open if necessary.
4. You can also highlight the variable and drag it into the Watch window. You can also select on <Enter expression> in Watch 1 to type it in manually.
5. You can always open a Watch window by selecting View/Watch Windows/Watch 1 in the main µVision window.
6. AD_last will display as shown here:
7. Click on RUN if necessary.
8. Rotate the pot and **AD_last** is updated in real-time.



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

Memory window:

1. Right-click on AD_last and select Add AD_last to … and then select Memory 1.
2. Rotate the pot and watch the memory window.
3. Note the value of **`AD_last`** is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing at but this not what we want to see at this time.
4. Add an ampersand "&" in front of the variable name and press Enter. Now the address of **`AD_last`** is shown (0x10000028 in this case).
5. Right click in the memory window and select Unsigned/Long.
6. The data contents of **`AD_last`** is displayed as shown here:
7. Both the Watch and Memory windows are updated in real-time.



TIP: You are able to configure the Watch and Memory windows and change their values while the program is still running in real-time without stealing any CPU cycles.

You can insert a number in a Memory window in real-time: No CPU cycles are stolen !

8. Stop the CPU  and exit debug mode. 
9. In the source file Blinky.c add a global variable counter near line 30 like this: **`unsigned int counter = 0;`**
10. In the main function add the following two lines just after the printf statement near line 88:

```
counter++;
if (counter > 0x0F) counter = 0;
```

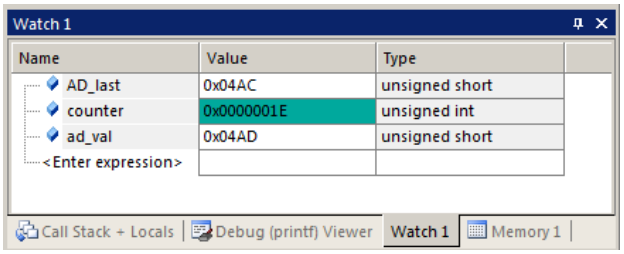
11. Compile the source files by clicking on the Build icon.  .
12. Program the flash by clicking on the Load icon:  and enter Debug mode.  Click on the RUN icon. 
13. Enter the variable **`counter`** in the Memory 1window by your preferred method. Add the & in front of **`counter.`** Note it increments every second or so.
14. Right-click on the memory word for **`counter`** in the Memory 1 window.
15. Select Modify Memory at 0x*address* and enter 0x0 or just 0 and press Enter.
16. **`counter`** will be set to zero while the program still runs or to any other value you entered. You can also do this in the Watch window when the program is halted.

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

Watch, Memory windows and many functions of the Serial Wire Viewer can view variables located in physical memory. This includes static and global variables plus arrays and structures. Local variables, usually held in CPU registers, are not visible. To view locals (also called automatics), simply convert them to static or global variables.

1. Stop the program  . Enter the local variable **ad_val** from main() in Blinky.c near line 37 to Watch 1. If you are unable to enter **ad_val,** make sure that the program is stopped in main() and not some function.
2. It will probably have a value displayed as the program spends nearly all its time in main() so it is in scope. If the PC is outside of main(), <out of scope> or <cannot evaluate> will be displayed.
3. Start the program by clicking on the Run icon  .
4. Set a breakpoint by clicking in the margin beside the line **clock_1s = 0;** in main() around line 86 The program will soon stop on this hardware breakpoint.

TIP: You can set breakpoints on-the-fly with Cortex-M processors !



5. **ad_val** is displayed as shown here:
6. Each time you click RUN, these values are updated. You might have to rotate the pot to see a difference.

How to view these variables updated in real-time:

All you need to do is to make **ad_val** static !

1. In the declaration for **ad_val** add **static** like this and recompile:

```
int main (void) {
        uint32_t ad_avg = 0;
        static uint16_t ad_val = 0, ad_val_ = 0xFFFF;
```

2. Exit debug mode.

TIP: You can edit files in edit or debug mode, but can compile them only in edit mode.

3. Compile the source files by clicking on the Build icon or press F7. Hopefully they compile with no errors or warnings.
4. To program the Flash click on the Load icon  . A progress bar will be at the bottom left.

TIP: To program the Flash automatically when you enter Debug mode select Target Options  , select the Utilities tab and select the "Update Target before Debugging" box.

5. Enter Debug mode. 
6. Remove the breakpoint you previously set and click on RUN. You can use Debug/Kill All Breakpoints to do this.
7. **ad_val** is now updated in real-time.
8. Stop the CPU  and exit debug mode  for the next step.

How It Works:

µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. Remember the Cortex-M3 is a Harvard architecture. This means it has separate instruction and data buses. While the CPU is fetching instructions at full speed, there is plenty of time for the CoreSight debug module to read or write values without stealing any CPU cycles.

This can be slightly intrusive in the unlikely event the CPU and 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.

TIP: If various windows update only when the programs stops, make sure the Update is enabled:

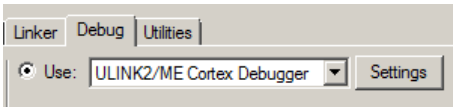
In the main menu select View/Periodic Window Update: 

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

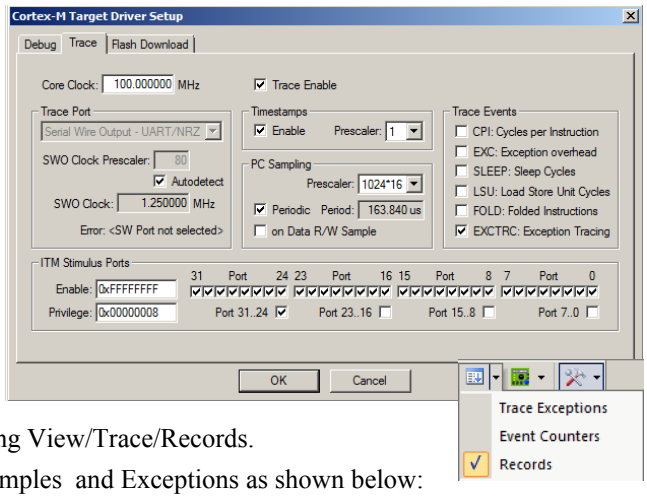
Serial Wire Viewer provides program information in real-time and is extremely useful in debugging programs.

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: this is 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 100 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. Click on the RUN icon.
2. 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.
3. The Trace Records window will open and display PC Samples and Exceptions 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.

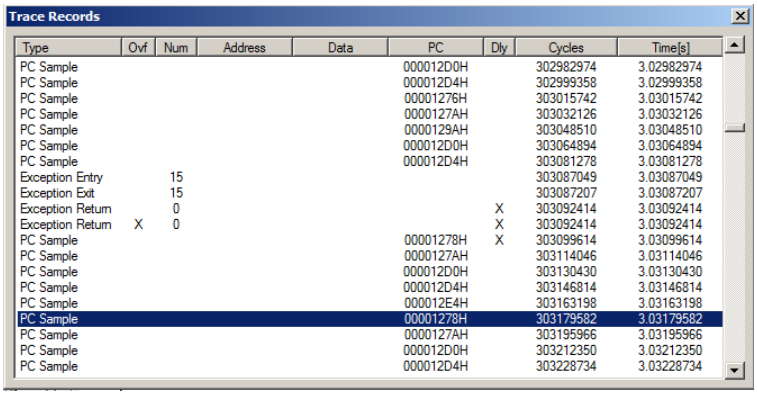
All frames have a timestamp displayed in CPU cycles and accumulated time.

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					000012D0H		302982974	3.02982974
PC Sample					000012D4H		302999358	3.02999358
PC Sample					00001276H		303015742	3.03015742
PC Sample					0000127AH		303032126	3.03032126
PC Sample					0000129AH		303048510	3.03048510
PC Sample					000012D0H		303064894	3.03064894
PC Sample					000012D4H		303081278	3.03081278
Exception Entry		15					303087049	3.03087049
Exception Exit		15					303087207	3.03087207
Exception Return		0				X	303092414	3.03092414
Exception Return	X	0				X	303092414	3.03092414
PC Sample					00001278H	X	303099614	3.03099614
PC Sample					0000127AH		303114046	3.03114046
PC Sample					000012D0H		303130430	3.03130430
PC Sample					000012D4H		303146814	3.03146814
PC Sample					000012E4H		303163198	3.03163198
PC Sample					00001278H		303179582	3.03179582
PC Sample					0000127AH		303195966	3.03195966
PC Sample					000012D0H		303212350	3.03212350
PC Sample					000012D4H		303228734	3.03228734

TIP: SWV is easily overloaded as indicated by an "x" in the OVF or Dly column. In this case, the extra Exception Return 0 marked with the X is spurious. Select only that information needed to reduce overloading. In this case, removing the PC Samples will solve this issue. There are more useful features of Serial Wire Viewer as we shall soon discover.

TIP: Num is the exception number: RESET is 1. External interrupts start at Num 16. For LPC1768, 41 is CAN IRQ. This is found in the LPC17xx Users Manual. Num 41 is also known as 41-16 = External IRQ 25.

9) Logic Analyzer: graphical data display using Serial Wire Viewer and 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.

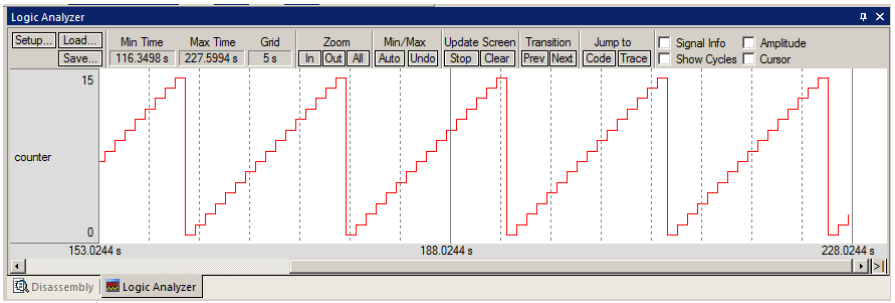
µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using the Serial Wire Viewer. The Serial Wire Output pin is easily overloaded with many data reads and/or writes and data can be lost. The LA shares the address comparators in CoreSight with the Watch 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 **`counter`** you previously created in Blinky.c.
7. Right click on counter and select Add 'counter' 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\Blinky.c\counter.

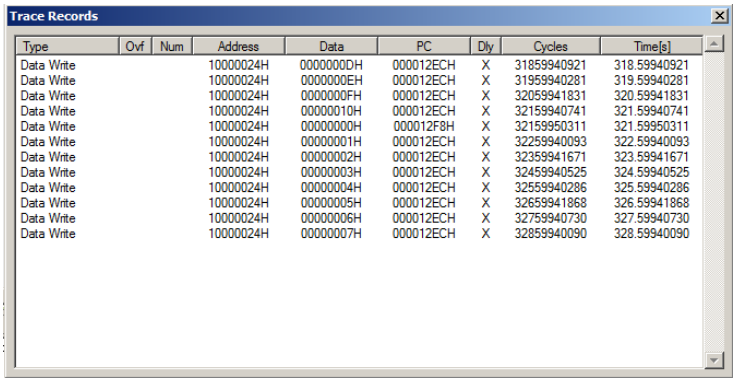
8. Click on Setup and set Max: in Display Range to 0x0F. Click on Close. The LA is completely configured now.
9. counter should still be visible in Watch 1. If not, enter it into the Watch 1 window.
10. Adjust the Zoom OUT or the All icon in the LA to provide a suitable scale of about 5 s as shown here:
11. Would you have guess counter is a sawtooth wave from looking at its value changing in the Watch 1 window ? Select Amplitude and use the cursor to see when counter = 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 0x0000_12EC caused a write of data 0x00 to RAM memory address 0x1000_0024 at the listed time in CPU Cycles or accumulated Time in seconds.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000024H	0000000DH	000012ECH	X	31859940921	318.59940921
Data Write			10000024H	0000000EH	000012ECH	X	31959940281	319.59940281
Data Write			10000024H	0000000FH	000012ECH	X	32059941831	320.59941831
Data Write			10000024H	00000010H	000012ECH	X	32159940741	321.59940741
Data Write			10000024H	00000000H	000012F8H	X	32159950311	321.59950311
Data Write			10000024H	00000001H	000012ECH	X	32259940093	322.59940093
Data Write			10000024H	00000002H	000012ECH	X	32359941671	323.59941671
Data Write			10000024H	00000003H	000012ECH	X	32459940525	324.59940525
Data Write			10000024H	00000004H	000012ECH	X	32559940286	325.59940286
Data Write			10000024H	00000005H	000012ECH	X	32659941868	326.59941868
Data Write			10000024H	00000006H	000012ECH	X	32759940730	327.59940730
Data Write			10000024H	00000007H	000012ECH	X	32859940090	328.59940090

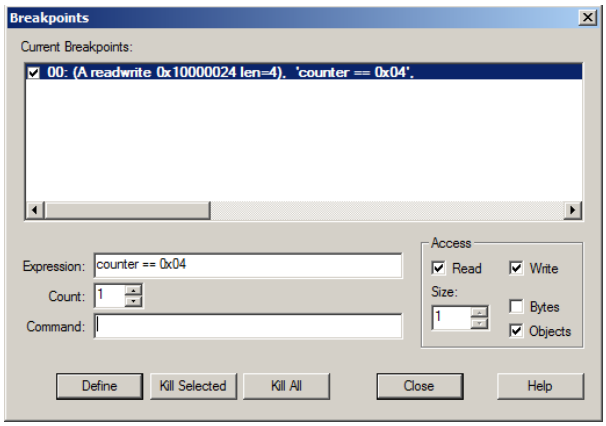
TIP: The PC column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin if there are too many overruns. µVision and CoreSight recover gracefully from trace overruns.

TIP: ULINK*pro* can process SWV data faster than a ULINK2 or a J-Link as it uses either the Manchester format or outputs the frames on the 4 bit trace port instead of the single bit SWO pin. ULINK2/ME and J-Link use the slower UART mode. If you are trying to output SWV trace at high rates, consider using a ULINK*pro*. It also programs Flash memory much faster.

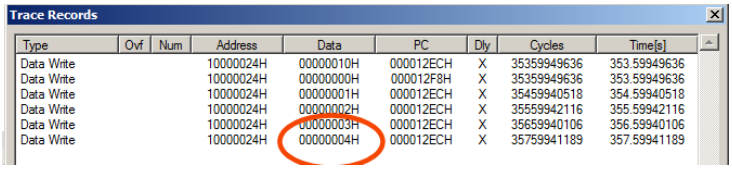
10) Watchpoints: Conditional Breakpoints

Most NXP Cortex-M3 and M4 processors have four data comparators. Since a 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 you must have two variables free in the Logic Analyzer to use Watchpoints.

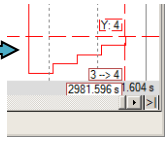
1. Using the example from the previous page, stop the program. Stay in Debug mode.
2. Enter the global variable counter into the Watch 1 window if it is not already there.
3. Click on Debug and select Breakpoints or press Ctrl-B.
4. The SWV Trace does not need to be configured to use Watchpoints. However, we will use it in this exercise.
5. Enter in Expression: "counter == `0x4`" without the quotes. Select both the Read and Write Access.
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. counter should still be entered in the Logic Analyzer window from the previous exercises.
10. Click on RUN.
11. When **`counter`** equals 0x4, the program will stop. This is how a Watchpoint works.
12. You will see **`counter`** 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:



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: This can be very useful to determine what the value of a variables really is at a given point in time.



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.

11) *RTX_Blinky* Program with Keil RTX RTOS: A Stepper Motor example

Keil provides RTX, a full feature RTOS. RTX now comes with a BSD type license. This means it is free and no licensing or product fees or royalties are payable with RTX. RTX is easy to implement full feature RTOS with up to 255 tasks.

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. Other RTOS companies also provide awareness plug-ins for µVision. Any RTOS ported to a Cortex-M or R processor will compile with MDK. See www.keil.com/rl-arm/kernel.asp for complete RTX details.

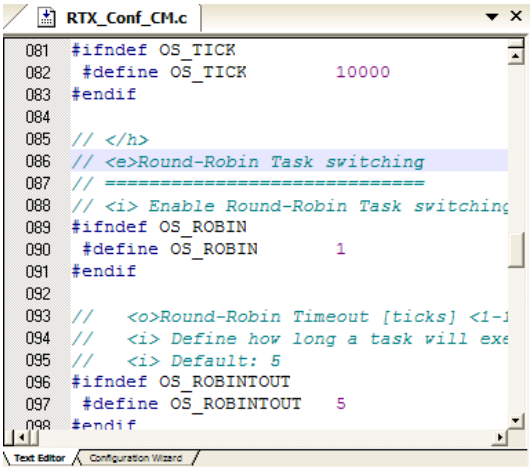
RTX is a Keil produced RTOS that is provided with MDK. Source code is provided with all versions of MDK.

TIP: You can also run this program with the simulator.

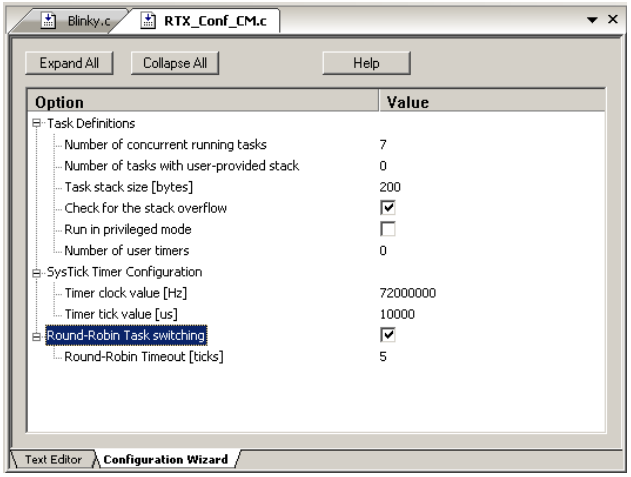
1. Start µVision4 by clicking on its icon on your Desktop if it is not already running.
2. Select Project/Open Project and open C:\Keil\ARM\Boards\Keil\MCB1700\RTX_Blinky\Blinky.uvproj.
3. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
4. To program the Flash manually, click on the Load icon. A progress bar will be at the bottom left.
5. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
6. The LEDs will blink indicating the waveforms of a stepper motor driver. This will also be displayed on the LCD screen. 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 or double-click on it in the Project window if you are not in Debug mode.
2. Click on Configuration Wizard 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. This is a great feature as it is much easier changing items here than in the source code.
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 µVision4 System Viewer windows used to display the peripherals are created in a similar fashion.



Text Editor

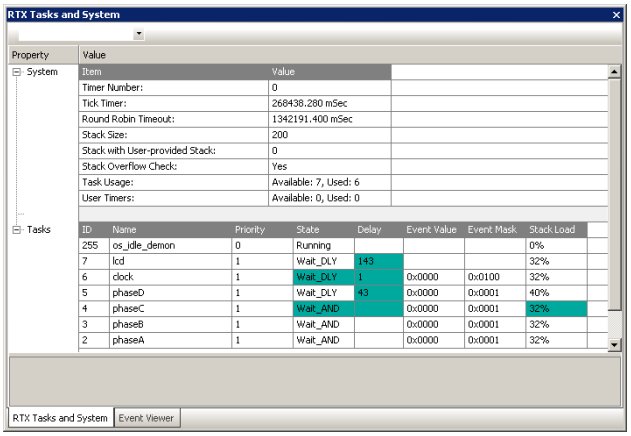


Configuration Wizard

TIP: µVision windows can be floated anywhere. You can restore them by selecting Window/Reset Views to default. µVision supports dual monitors.

12) RTX Kernel Awareness using Serial Wire Viewer

Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides two kernel aware windows for RTX. Other RTOS companies also provide awareness for µVision.



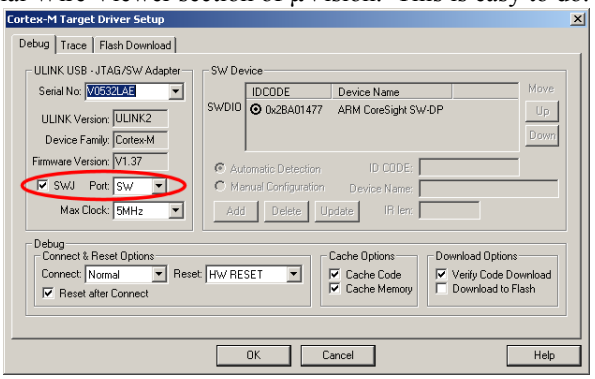
1. Click on the RUN icon to run RTX_Blinky.
2. Open Debug/OS Support and select RTX Tasks and the window on the right opens up. You probably have to drag it in the middle of your screen to view it.
3. RTOS visibility is updated in real-time using CoreSight technology as used in the Watch and Memory windows.

TIP: View/Periodic Window Update must be selected for the RTX Task and System window to be updated. The Serial Wire Viewer must be configured for the Event Viewer to be updated.

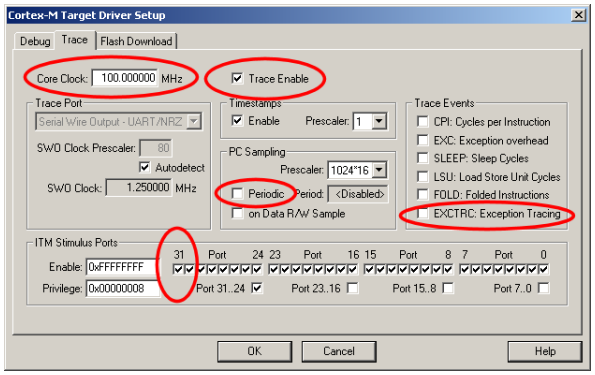
4. Open Debug/OS Support and click on Event Viewer. There is probably no data visible because... SWV is not configured yet. If it is working, jump to step

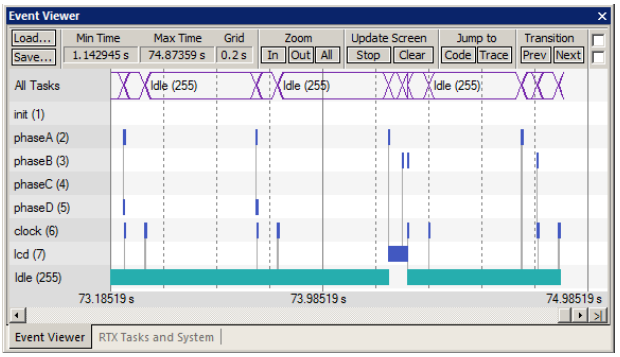
Configuring the Serial Wire Viewer (SWV):

In order to get the Event Viewer working we have to configure the Serial Wire Viewer section of µVision. This is easy to do.



1. Stop the CPU and exit debug mode.
2. Click on the Target Options icon .
3. Select the Debug tab and then click the Settings box next to ULINK Cortex Debugger dialog.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW. Max Clock can be 10 MHz.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 100 MHz and select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown here. ITM Stimulus Port 31 must be checked.
8. Click on OK twice to return to µVision. The Serial Wire Viewer is now configured in µVision.
9. Enter Debug mode and click on Run .
10. Note the values in the Event Viewer and tasks and System are updated with the program running.
11. This window displays task events in a graphical format as shown in the RTX Event Viewer window below. You probably have to change the Range to about 0.2 seconds by clicking on the Out or In button or the ALL icon.





TIP: To find the CPU Core frequency select Peripherals/Clocking and select the Power Control/Clock Generation Schematic. Do this now to see it. This is a very useful window. If you open this after RESET and before run, you can see the base frequency.

TIP: Cortex-M0 and Cortex-M0+ processors do not have Serial Wire Viewer or ETM facilities. Cortex-M0+ can have MTB Instruction Trace. They do have hardware breakpoints and read/write memory capabilities. See your specific processor datasheet for details.

13) Logic Analyzer Window: view RTX variables real-time in a graphical format:

µVision has a graphical Logic Analyzer window. Variables will be displayed in real-time using the Serial Wire Viewer in the LPC1700. RTX_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.

Add the eight source lines to the four tasks:

1. Stop the program and exit debug mode.
2. Add 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 081 and 084 (just after LED_On and LED_Off function calls. For each task, add the corresponding variable assignment statements phasea, phaseb, phasec and phased.
4. We do this because in this example program there are not enough global or static 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: make them static. To see peripheral registers values, read or write to them.

5. Rebuild the project. Program the Flash and enter debug mode .
6. Run the program at this point.

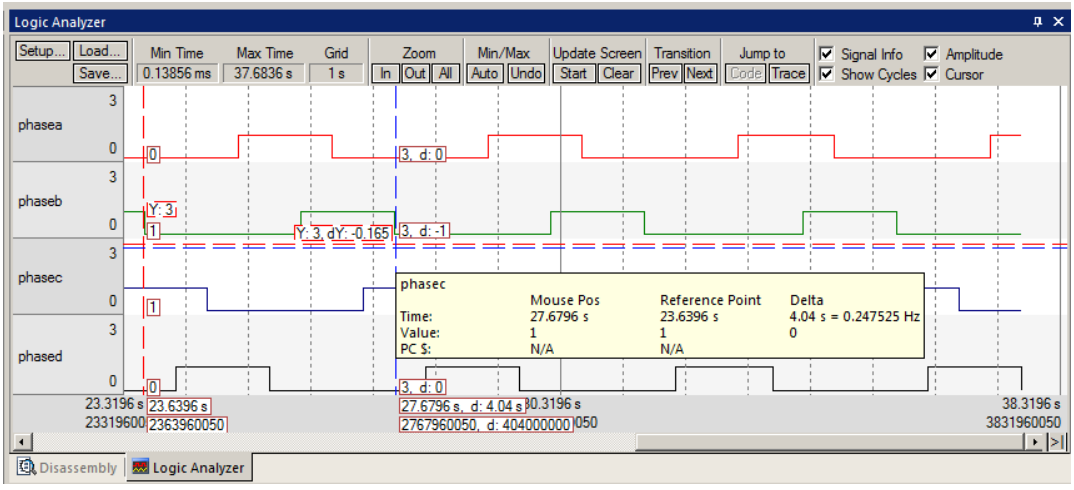
```
074 /*----------------------------------------
075  *        Task 1 'phaseA': Phase A output
076  *----------------------------------------
077 __task void phaseA (void) {
078   for (;;) {
079     os_evt_wait_and (0x0001, 0xffff);     /*
080     LED_On (LED_A);
081     phasea=1;
082     signal_func (t_phaseB);               /*
083     LED_Off(LED_A);
084     phasea=0;
085   }
086 }
```

Enter the Variables into the Logic Analyzer:

7. Click on the Blinky.c tab. Right-click on **phasea** and select Add 'counter' to … and select Logic Analyzer **Repeat** for **phaseb, phasec and phased.** These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.

TIP: If you can't enter a variable, make sure the Serial Wire Viewer is configured as detailed on the previous page.

8. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
9. Click on Close to go back to the LA window.
10. Using the OUT and In buttons set the range to 1 second or so. Move the scrolling bar to the far right if needed.
11. Click on Stop in the Update Screen box. Note the program continues running.
12. You will see the following waveforms appear. Click to mark a place See 27 s below. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phasec:
13. Select Signal Info, Amplitude, Show Cycles and Cursor. Move the cursor to see the information displayed.
14. When you are finished, click on Stop and exit Debug mode .



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

14) External Interrupt Example: EXTI using Serial Wire Viewer (SWV):

This example uses a ULINK2/ME and SWV. You can also use a ULINK*pro* or J-Link with proper configuration.

Serial Wire Viewer can help debug many tricky interrupt issues. The project EXTI is available to demonstrate these powerful SWV features. The Serial Wire Viewer will be configured for this example to work.

In this program the button INT0 is connected to a GPIO port (p2.10) and each time it is pressed an interrupt is generated.

1. Open the project C:\Keil\ARM\Boards\Keil\MCB1700\EXTI\EXTI.uvproj.

Configure Serial Wire Viewer trace:

2. Select Target Options and select the Debug tab. Confirm the SJ box is checked and SW is selected.
3. Select the Trace tab.
4. Set Core Clock: to 100 MHz. Select Trace Enable.
5. Select EXCTRC, unselect Periodic and on Data R/W Sample. Click on OK twice to return to the main menu.

Build, Load and RUN EXTI:

15. Build the source files, load the Flash and enter Debug mode. Run the program.

View the Trace and create exception EXTI:

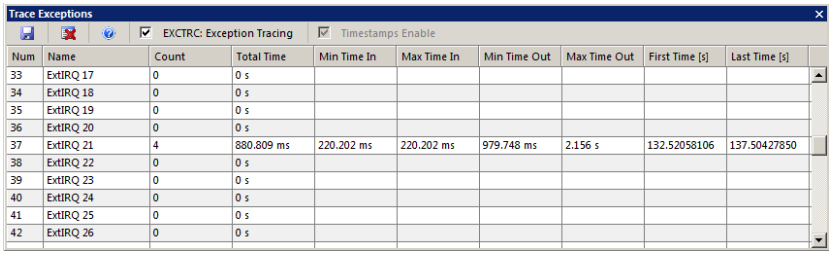
6. Open the Trace Records and Exception Trace windows:
7. Click on the Trace Exception window tab and move it into the middle of your screen.
8. Press the INT0 button and EXTIrq 21 (Number 37) will display in both windows and cause a Led will advance.
9. Make sure you do not press the RESET button by accident !
10. The interrupt handler function **`EINT3_IRQHandler()`** in Exti.c is executed each time you press INT0.

Trace Records Exceptions Type:

Entry: when the exception is entered.

Exit: when the exception or interrupt exits.

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



Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
33	ExtIRQ 17	0	0 s						
34	ExtIRQ 18	0	0 s						
35	ExtIRQ 19	0	0 s						
36	ExtIRQ 20	0	0 s						
37	ExtIRQ 21	4	880.809 ms	220.202 ms	220.202 ms	979.748 ms	2.156 s	132.52058106	137.50427850
38	ExtIRQ 22	0	0 s						
39	ExtIRQ 23	0	0 s						
40	ExtIRQ 24	0	0 s						
41	ExtIRQ 25	0	0 s						
42	ExtIRQ 26	0	0 s						

TIP: If you do not see PC Samples and Exceptions as shown and instead either nothing or frames with strange data, the trace is not configured correctly. The most probable cause is the Core Clock: frequency is wrong.

Switch Bounce:

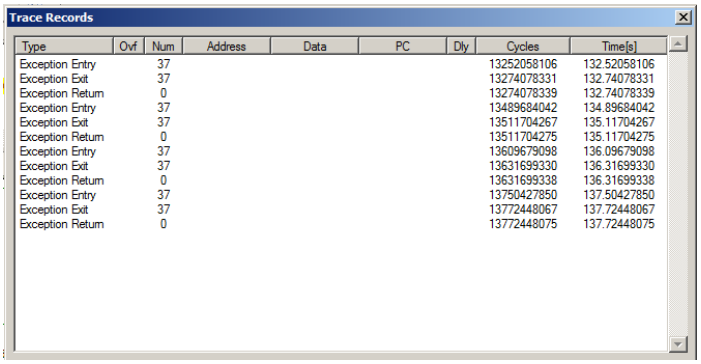
You might notice as you press the INT0 button that sometimes the sequence of switching LEDs jumps. This is caused by switch bounce. You can correct this by adding this C code to the beginning of the interrupt handler EINT3_IRQHandler():

```
unsigned int i =0;
for (i = 0; i < 0x60000;  i++)
```

1. Exit debug mode and enter the two C lines to the beginning of the interrupt handler found in EXTI.c.
2. Rebuild the project.
3. Program the Flash.
4. Enter debug mode.
5. Run the program and press INT0.
6. You will see the issue is resolved.

Using SWV to debug exceptions is very useful and is completely non-intrusive to your program.

7. Stop the program and exit Debug mode.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		37					13252058106	132.52058106
Exception Exit		37					13274078331	132.74078331
Exception Return		0					13274078339	132.74078339
Exception Entry		37					13489684042	134.89684042
Exception Exit		37					13511704267	135.11704267
Exception Return		0					13511704275	135.11704275
Exception Entry		37					13609679098	136.09679098
Exception Exit		37					13631699330	136.31699330
Exception Return		0					13631699338	136.31699338
Exception Entry		37					13750427850	137.50427850
Exception Exit		37					13772448067	137.72448067
Exception Return		0					13772448075	137.72448075

15) ITM (Instruction Trace Macrocell)

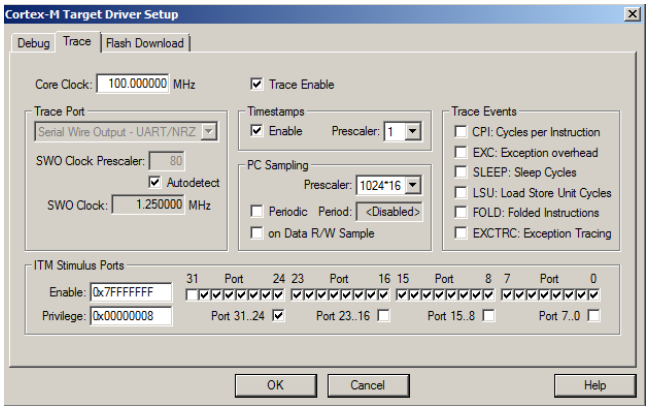
Recall in the Section RTX Kernel Awareness on page 13 that we showed you can display information about the RTOS in real-time. This is done through the ITM Stimulus Port 31. Port 0 is available for a *printf* type of instrumentation that requires minimal use code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display.

1. Open the RTX_Blinky project you used before. You can select it at the bottom of Project menu in the recent files list.
2. Add this code to Blinky.c. A good place is right after the place where you declared the four phase*x* variables.

```
#define ITM_Port8(n)   (*((volatile unsigned char *)(0xE0000000+4*n)))
```

3. In the task phaseA near line 85 enter these three lines:

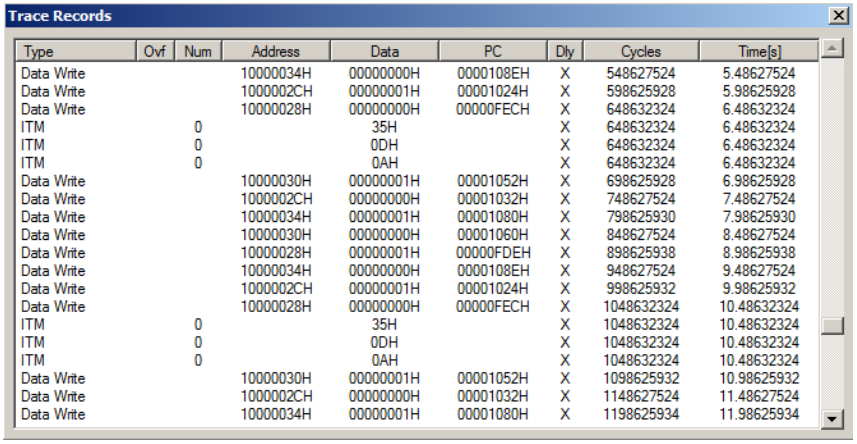
```
ITM_Port8(0) = 0x35;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```



4. Rebuild the source files, program the Flash memory and enter Debug mode.
5. Select on View/Serial Windows and select Debug (printf) Viewer and click on RUN. Make sure Periodic Update is selected.
6. In the Debug (printf) Viewer you will see the value "5" appear every few seconds.

To see the Trace Records

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample, ITM Port 31 and EXCTRC.
3. Select ITM Port 0.
4. Click OK twice.
5. The Trace Records should still be open. Open it if not. Double click on it to clear it.
6. Click RUN to start the program.
7. You will see a window similar to the one below with ITM and data write frames.
8. Stop and exit Debug mode when you are finished.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000034H	00000000H	0000108EH	X	548627524	5.48627524
Data Write			1000002CH	00000001H	00001024H	X	598625928	5.98625928
Data Write			10000028H	00000000H	00000FECH	X	648632324	6.48632324
ITM		0		35H		X	648632324	6.48632324
ITM		0		0DH		X	648632324	6.48632324
ITM		0		0AH		X	648632324	6.48632324
Data Write			10000030H	00000001H	00001052H	X	698625928	6.98625928
Data Write			1000002CH	00000000H	00001032H	X	748627524	7.48627524
Data Write			10000034H	00000001H	00001080H	X	798625930	7.98625930
Data Write			10000030H	00000000H	00001060H	X	848627524	8.48627524
Data Write			10000028H	00000001H	00000FDEH	X	898625938	8.98625938
Data Write			10000034H	00000000H	0000108EH	X	948627524	9.48627524
Data Write			1000002CH	00000001H	00001024H	X	998625932	9.98625932
Data Write			10000028H	00000000H	00000FECH	X	1048632324	10.48632324
ITM		0		35H		X	1048632324	10.48632324
ITM		0		0DH		X	1048632324	10.48632324
ITM		0		0AH		X	1048632324	10.48632324
Data Write			10000030H	00000001H	00001052H	X	1098625932	10.98625932
Data Write			1000002CH	00000000H	00001032H	X	1148627524	11.48627524
Data Write			10000034H	00000001H	00001080H	X	1198625934	11.98625934

Explanation:

The Data Write frames are the writes to phasea through phased. These are here because you previously entered them in the Logic Analyzer window.

ITM 0 frames are our ASCII characters "5" and carriage return and line feed. You can see these values in the Data column.

ITM Conclusion

The writes to ITM Stimulus Port 0 are intrusive and are usually one cycle. It takes no CPU cycles to get the data out the LPC1700 processor via the Serial Wire Output pin to µVision to be displayed.

Note the X in the Dly column. The three writes are too fast for the SWO and you can see the timing as shown in the Cycles column are all the same but the data values are correct. As mentioned before, this is a limitation of SWV. But SWV is intensely useful for debugging.

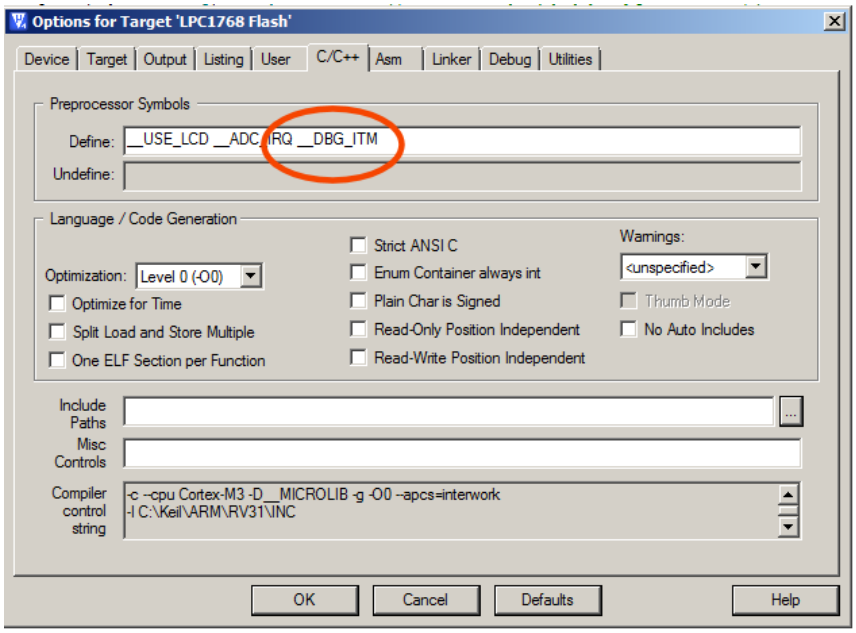
Examination with an ETM Trace shows the total time to display the digit is 25 CPU cycles including the while wait time.

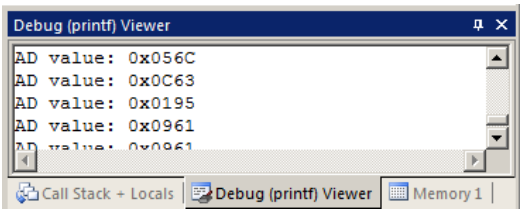
TIP: ITM_SendChar is a useful function you can use to send characters. It is found in the header core.CM3.h.

16) printf Statements using ITM:

It is possible to retarget printf statements to either the Debug (printf) Viewer or to the LPC1700 UART.
The Serial Wire Viewer will need to have been configured previously. Otherwise, you will need to repeat these steps.

1. Open the Blinky project C:\Keil\ARM\Boards\Keil\MCB1700\Blinky\Blinky.uvproj.
2. Select the Target Options icon and select the C/C++ tab. This window opens:
3. In the Define box, enter _DBG_ITM as shown below:
4. Select the Debug tab and then click the Settings box next to ULINK Cortex Debugger dialog.
5. In the Debug window, make sure SWJ is checked and Port: is set to SW.
6. Click on the Trace tab to open the Trace window.
7. Set Core Clock: to 100 MHz and select Trace Enable.
8. Unselect the Periodic and EXCTRC boxes. ITM Stimulus Port 0 ***must*** be checked.
9. Click on OK twice to return.
10. Rebuild the project.
11. Program the Flash.
12. Enter debug mode.
13. Run the program
14. Open View/Serial Windows and select Debug (printf) Viewer.
15. This window will now display the variable AD value.
16. This happens on the printf near line 90 in Blinky.c:
    printf("AD value: %s\r\n", text);





TIP: Examine the files Retarget.c and Serial.c for the functions used to accomplish this.

UART Operation:

1. In the C/C++ tab, in the Define: box enter __UART0 instead of _DBG_ITM.
2. Rebuild and program the Flash.
3. Remove the ISP and RST jumpers.
4. Connect a serial port to COM1. 115200 baud, 8 data bits, no parity, 1 stop bit
5. Enter Debug mode and Run the program and serial data will be displayed on your favourite serial program.

17) CAN: Controller Area Network

CAN is a network that is easy to implement. It is a peer-to-peer network and adding nodes is very easy. For more detailed information on the CAN bus and complete exercises using CAN for the LPC1700 series obtain the CAN Primer from: www.keil.com/appnotes/docs/apnt_247.asp.

1. **Connectors:** The MCB1700 board has two DB-9 connectors labeled CAN1 and CAN2. These are the two CAN controllers. You must connect pin 2 of each connector to the other and also pin 7 to the other. Do not cross them. You can use two DB-9 connectors or jumper wires. Make sure the connections are reasonably sturdy. See the first TIP below for an explanation.
2. Start µVision by clicking on its icon on your Desktop if it is not already running.
3. Select Project/Open Project and open the project file C:\Keil\ARM\Boards\Keil\MCB1700\CAN\CAN.uvproj.
4. Compile the source files by clicking on the Rebuild icon.  . They will compile with no errors or warnings.
5. Click on the Load icon to program the Flash memory.  . 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 screen will display a value of both Tx: and Rx: and will vary when you rotate the potentiometer P2.

What is happening: The LPC1758 or 68 contains two CAN controllers and we have connected them together to form a two node network. CAN2 is sending messages to CAN1 and they are displayed on the LCD as TX: and RX: respectively. You need at least two CAN nodes to have a working CAN network. See the Keil CAN Primer for more information.

I connected a CAN analyzer to the CAN bus and it displays the CAN frames transmitted as shown here: CAN analyzers are a good investment.

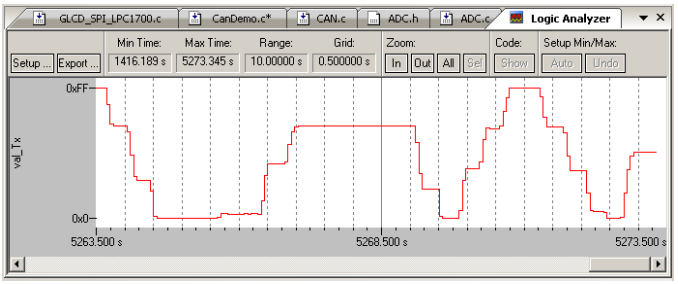
Time / 10 mSec	Identifier	Format	Flags	Data
00:00:39.85	21	Std		40
00:00:39.91	21	Std		45
00:00:39.97	21	Std		4B
00:00:40.03	21	Std		53
00:00:40.09	21	Std		5C
00:00:40.15	21	Std		69
00:00:40.21	21	Std		74
00:00:40.27	21	Std		87
00:00:40.33	21	Std		98
00:00:40.38	21	Std		A2
00:00:40.44	21	Std		B1
00:00:40.50	21	Std		BC
00:00:40.56	21	Std		CB
00:00:40.62	21	Std		DA
00:00:40.68	21	Std		EB

The CAN Identifier is 21 (ID column) and the data values displayed. There is one data byte per frame in this case. It is possible to have from 0 to 8 data bytes per frame.

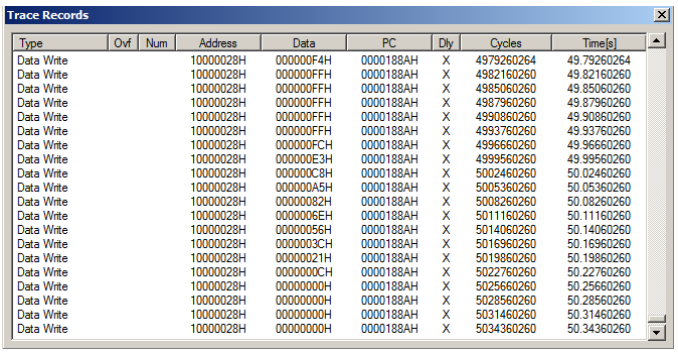
TIP: If only Tx: changes, either the loopback cable isn't connected or you are using only an early ULINK-ME to power the board. Connect a USB cable from your computer to the MCB1700 board to provide 5 volts to the CAN transceiver in this case.

Logic Analyzer Window:

We can display the CAN data as a graph updated in real-time with Serial Wire Viewer.



1. Stop the program and leave Debug mode.
2. Open Target Options  , Select the Debug tab and then Settings. Ensure SWJ and SW are selected so SWV will be operational.
   Select the Trace tab. Set the Core Clock: to 100 MHz and select Trace Enable.
   Uncheck Periodic and EXCTRC. Select on Data R/W Sample. Click OK twice to return.
3. Select File/Save All. Enter debug mode. 
4. Insert the global variable **`val_Tx`** in CanDemo.c into the Logic Analyzer window with a range 0 to 0xFF.
5. Click on Zoom icons to set Grid to 2 seconds.
6. Insert **`val_Tx`** into the Watch window.
7. Open the Trace Records window. RUN the program.
8. You will see the data change as you rotate the pot in both the LA window shown here and in the Watch window in real time stealing no CPU cycles.
9. The trace records window will show the CAN data write to the variable **`val_Tx`**. All are timestamped.



18) Using Watchpoints and Serial Wire Viewer with CAN

1. Stop the program if still running. µVision must be in debug mode to access the Watchpoints.
2. Open Target Options  , Select the Debug tab and then Settings. Select the Trace tab. Select EXCTRC. Click OK twice to return.
3. Open Debug/Breakpoints and enter in the dialog box: **val_Tx == 0x44** Select Read and Write.
4. Click on Define to create the Watchpoint and then Close.
5. Double-click in the Trace Records box to clear it and run the program by clicking on RUN. Or open it if it is not.
6. Adjust the pot to indicate 0x44. The first time this value is written to val_Tx, the program will stop.
7. Note the value in the Watch window will equal 0x44 ! The LCD may or may not have been updated yet.
8. Scroll to the bottom of the Trace Records and the value of 0x44 will be visible on the last line as shown below.
9. There will be a read of 0x44 at the end of the trace plus the address of the instruction that caused the trigger !
10. In this case, the last frame says a Data Write of 0x44 occurred to address 0x10000028 by the instruction located at 0x189C.

What is happening: Note the last frame has the data value of 0x44. Recall you set the Watchpoint to a READ of 0x44.

You are not able to see the CAN EXTIRQ 41 occurring because it is swamped out by the Systick timer which is being used in this example as a general purpose timer. If you turn Systick off and use a software delay, you will see only the CAN exceptions.

Recall the Exception Return of Num 0 means all the exceptions have returned and there is no tail-chaining.

This is one of the powers of trace: you can see what happened to your program and how. If a bad value was written to one of your variables; you can tell when it happened and what instruction made this write. The possibilities of advanced debugging are great with trace.

Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Return		0				X	10345949730	103.45949730
Exception Entry		15					10346047359	103.46047359
Exception Exit		15					10346047369	103.46047369
Exception Return		0				X	10346049730	103.46049730
Exception Entry		15					10346147359	103.46147359
Exception Exit		15					10346147369	103.46147369
Exception Return		0				X	10346149730	103.46149730
Exception Entry		15					10346247359	103.46247359
Exception Exit		15					10346247369	103.46247369
Exception Return		0				X	10346249730	103.46249730
Exception Entry		15					10346347359	103.46347359
Exception Exit		15					10346347369	103.46347369
Exception Return		0				X	10346349730	103.46349730
Exception Entry		15					10346447359	103.46447359
Exception Exit		15					10346447369	103.46447369
Exception Return		0				X	10346449730	103.46449730
Exception Entry		15					10346547359	103.46547359
Exception Exit		15					10346547369	103.46547369
Exception Return		0				X	10346548868	103.46548868
Data Write			10000028H	00000044H	0000189CH	X	10346548868	103.46548868

1. Open the Trace Exception window. 
2. Grab it by its tab and bring it in the middle of the screen for convenient viewing.
3. The CAN Exceptions (41) are visible with various timings displayed as well as the number of times it occurred.

Trace Exceptions

EXCTRC: Exception Tracing Timestamps Enable

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
38	ExtIRQ 22	0	0 s						
39	ExtIRQ 23	0	0 s						
40	ExtIRQ 24	0	0 s						
41	ExtIRQ 25	1533	650.930 us	2.930 us	72.000 us	30.000 ms	44.280 s	56.57560110	103.43564930
42	ExtIRQ 26	0	0 s						
43	ExtIRQ 27	0	0 s						
44	ExtIRQ 28	0	0 s						

TIP: Recall that you can right click in the Trace Records window to filter out various Types of frames.

TIP: The ULINK*pro* displays the source code and disassembly instructions in the new Trace window.

ULINK*pro* also provides Code Coverage, Performance Analysis and Execution Profiling by using the ETM trace.

Note: The current version of Keil MDK (4.72a) only displays data writes and not reads. This is to prevent data overruns in the Trace Records window. Future versions may include data reads.

19) 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-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.

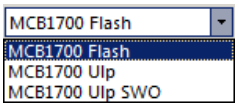
You can use this example with other Cortex-M boards with possible changes to the startup and system files.

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, download it from: www.keil.com/appnotes/docs/apnt_246.asp

1. Extract DSP.zip to C:\Keil\ARM\Boards\Keil\MCB1700\ to create the folder \DSP.
1. Open the project file sine.uvproj with µVision. Connect a ULINK2 or ULINK*pro* to the MCB1700 board.
2. If you are using a ULINK2 or ME, select **MCB1700 Flash** from the target drop menu: If using a ULINK*pro*, select **MCB1700 Ulp** to send SWV out the Trace Port. Select **MCB1700 Ulp SWO** to send SWV out the 1 bit SWO pin.



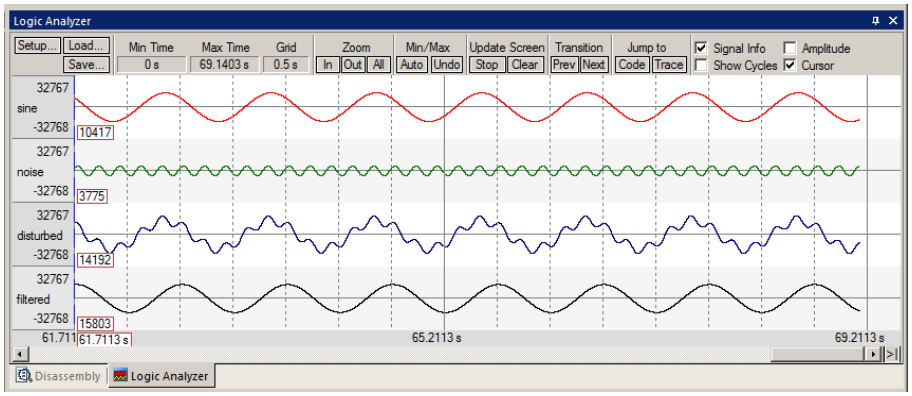
3. Compile the source files by clicking on the Rebuild icon.  .
4. Program the MCB1700 flash by clicking on the Load icon: 
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.

TIP: The default Core Clock: is 100 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.

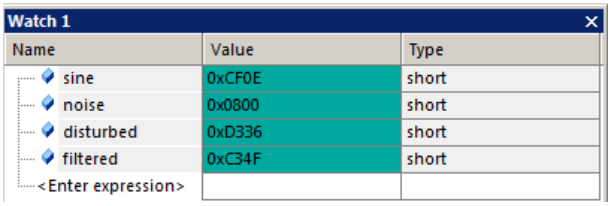
8. This project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.
9. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below: They are pre-configured in this project.



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 at initialization time.

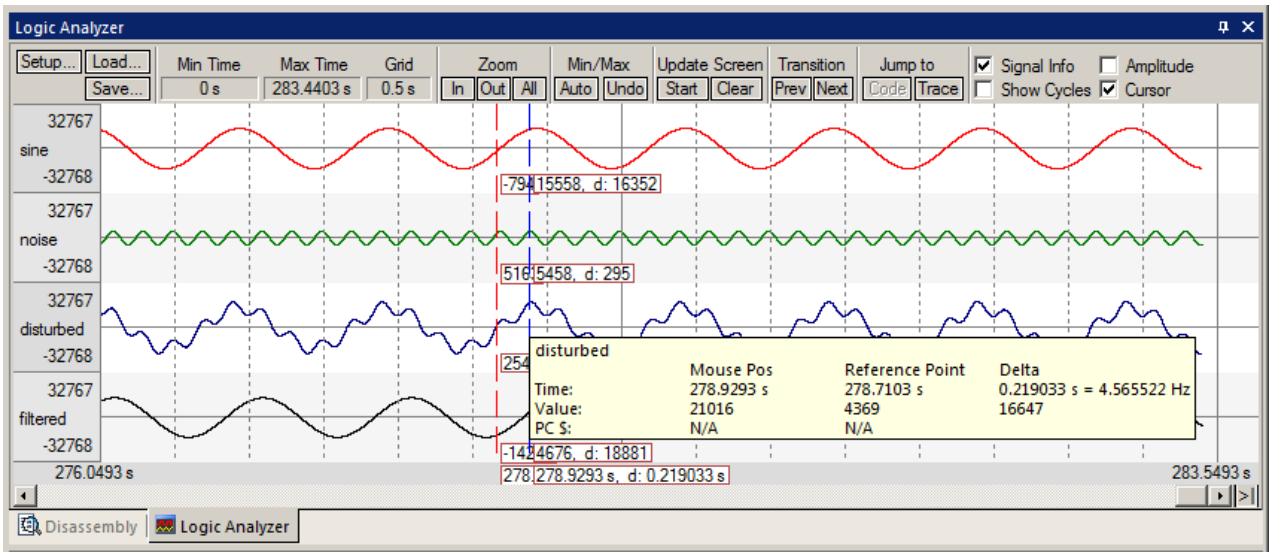
TIP: The ULINK*pro* trace display is different and the program must currently be stopped to update it. The LA will still be updated in real-time.

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



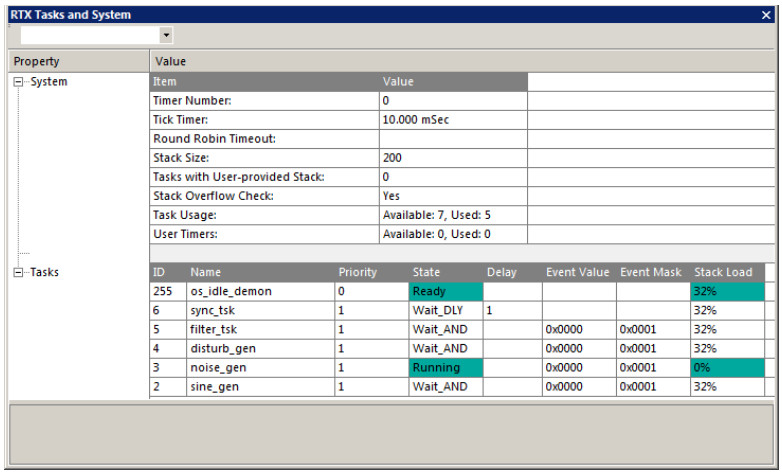
2) Signal Timings in the Logic Analyzer (LA):

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.



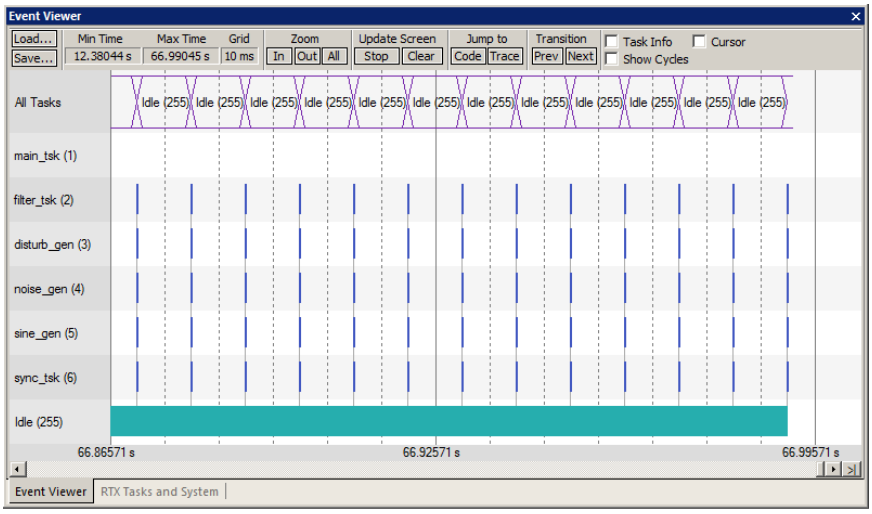
TIP: You can set hardware breakpoints while the program is running.

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 Task 1 (main_tsk) has no events displayed. Tis means it is not running. Task 1 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.



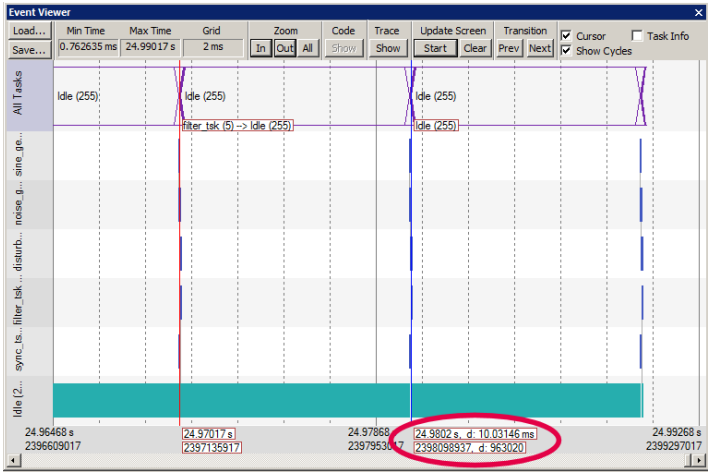
TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some SWO or Trace Port bandwidth.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

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.

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.

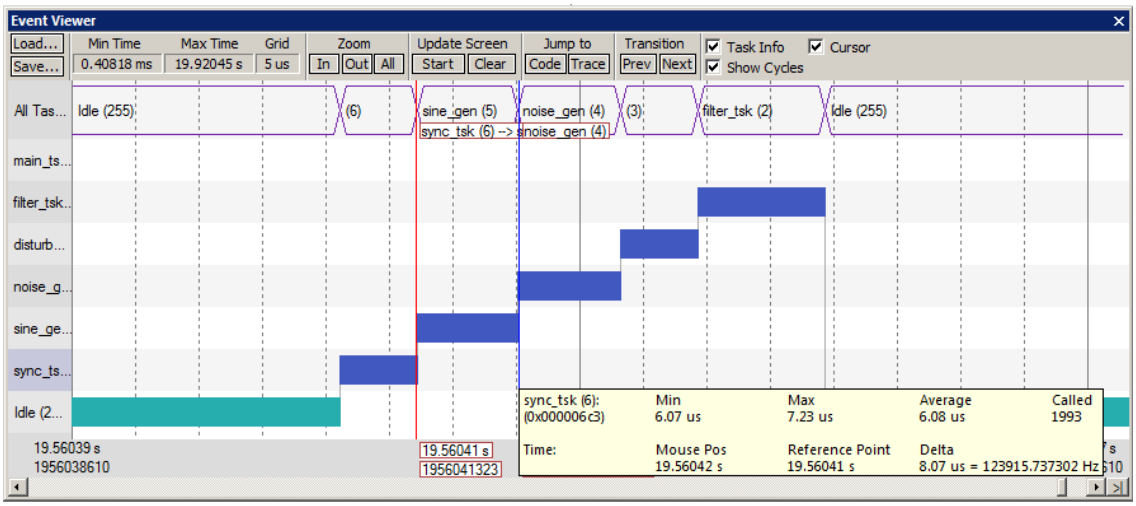
If the Event Viewer is closed, the data is still being sent out the SWO pin or the Trace Port and contributes to overloading.

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.

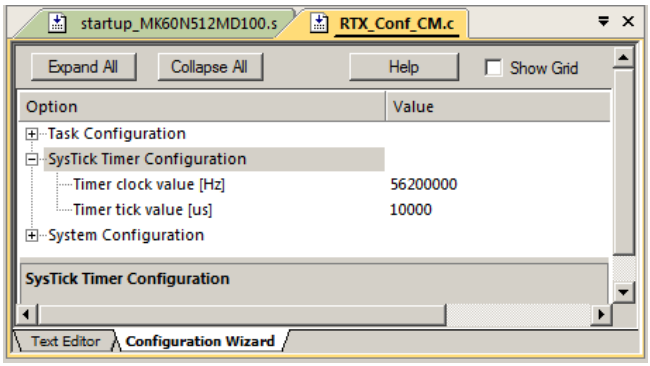
TIP: If the Event Viewer does not display correctly, the display of the variables in the Logic Analyzer window might be overloading the SWO pin. In this case, stop the program and delete all LA variables (Kill All) and click on Run.

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\MCB1700\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 100,000,000 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.

20) Using a ULINK*pro*:

We have seen what features the Serial Wire Viewer provides with many NXP Cortex-M3 and M4 processors. Most of these processors also have Embedded Trace Macrocell (ETM). ETM provides a record of all executed instructions and is invaluable for program flow analysis of many kinds.

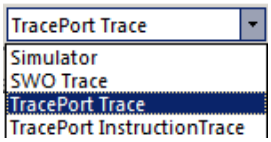
For more information regarding ETM instruction trace obtain the MCB4300 lab: www.keil.com/appnotes/docs/apnt_241.asp

Blinky_Ulp example:

This project is a version of the standard Blink project but is pre-configured to use with a ULINK*pro*. This project is located here: C:\Keil\ARM\Boards\Keil\MCB1700\Blinky_Ulp\Blinky.uvproj.

Trace Display:

The main trace window (known as Trace Records with a ULINK2) is called Trace Data. It has added features such as the ability to save the frames and extra filter modes. Data Trace (SWV) and Instruction trace can be displayed in this window but not at the same time. The program must be stopped in order for this window to be updated. The Trace Data window features are being constantly improved with new features added.



You can right-click in this window and set these elements: (and more)

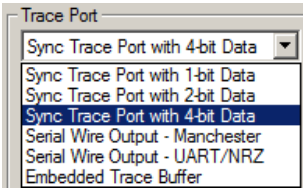
1. Set a timestamp to zero. All other timestamps are now relative to this one.
2. Add the Functions column. Displays the function an instruction belongs to.
3. Show Record Information: displays additional information about a function.

Target Configurations for trace configuration:

There are three Target Configurations as shown here: This menu is available in µVision main screen while in Edit mode. You can easily create your own targets: They are described on the next page.

1. **Simulator:** Selects the Keil simulator. No target hardware is needed.
2. **SWO Trace:** Serial Wire Viewer (SWV) is activated and is sent out the one bit SWO pin using Manchester mode. Used by ULINK*pro* only. ULINK2/ME and J-Link use UART mode.
3. **Trace Port Trace:** Serial Wire Viewer (SWV) is sent out the 4 bit Trace Port for faster operation.
4. **Trace Port Instruction Trace:** ETM trace is sent out the 4 bit Trace Port. SWV information is not always displayed in the Trace Data window in this mode. Shorts bursts of runtime they are captured. In long runtimes, they are deleted to save space.

Trace Port Configuration Box:



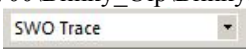
The Debug/Settings/Trace tab  in the Target Configuration window selects how the trace information (Data or Instruction) is physically output form the processor:

This is not to be confused with the 4 bit Trace Port mentioned above which is a subset of this box. These selections are configured in each Target Configuration as listed above.

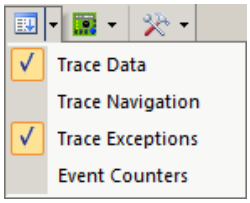
1. **Sync Trace Port with 4-bit Data:** SWV frames are sent out the 4 bit (or 1 or 2) parallel port with a clock. Only ULINK*pro* can use this selection. A script, LPC17xx_TPIU.ini, is needed to configure GPIO pins.
2. **Serial Wire Output – Manchester:** SWV out the SWO pin on the debug connector using Manchester encoding. ULINK*pro* only.
3. **Serial Wire Output – UART/NRZ:** SWV out the SWO pin on the debug connector using UART/NRZ encoding. ULINK2, ULINK-ME and J-Link only.
4. **Embedded Trace Buffer:** ETB: An internal memory in the processor used as a instruction trace buffer. Currently available with ULINK*pro* only. Not available on the LPC1700 processors. Works at the top processor speed.

22) Using the Blinky_Ulp example:

1) SWO Trace: Serial Wire Viewer (SWV) is activated and is sent out the one bit SWO pin using Manchester mode. No ETM instruction trace frames are available with this selection.

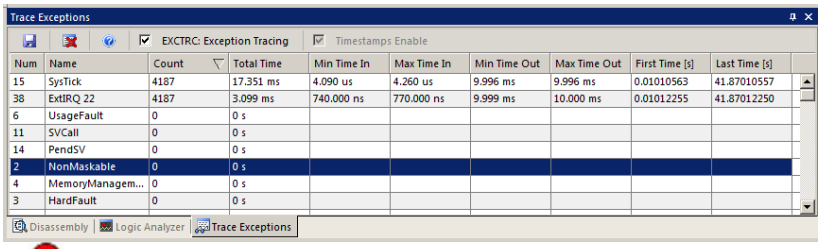
1. Open the project C:\Keil\ARM\Boards\Keil\MCB1700\Blinky_Ulp\Blinky.uvproj.
2. In the Target Selection window, select SWO Trace. 

3. Compile the source files by clicking on the Rebuild icon.  . They will compile with no errors or warnings.
4. Click on the Load icon to program the Flash memory.  . A progress bar will be at the bottom left.
5. Enter the Debug mode by clicking and click on the RUN icon. 



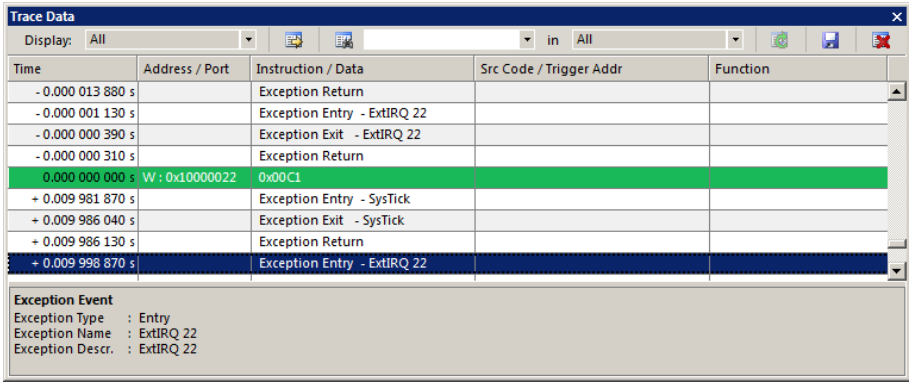
TIP: When switching Target Selections, it is always a good idea to rebuild your project.

6. Click on the small arrow beside the Trace icon and open the Trace Data and Trace Exception windows a shown here:
7. Drag them to an appropriate position and size on your screen.
8. Examine the Trace Exception window shown below. It will be updating in real-time and displays the Systick timer (15) and ADC interrupt (ExtIRQ 22):
9. Click on the top of the Count column to bring both exceptions together. The default is Num selected.



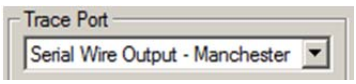
10. Stop the processor  .
11. The Trace Data window will now display these same exceptions in a different format as shown below:

TIP: If there were any other SWV frames such as Data Writes or PC Samples selected, they would also be displayed here. You only need to select them as described in preceding examples in this document.



TIP: In the Trace Data window above, I right-clicked inside it and selected Show Functions and Show Record Description. I set the timestamp to a data write. This timestamp is now zero and all others are displayed relative to it.

12. Exit Debug mode. 
13. Select the Target Options windows. 
14. Select the Debug tab and then the Settings: box. Select the Trace tab.
15. Note the Trace Port selection **Serial Wire Output – Manchester:** is selected. The trace frames are sent out the 1 bit SWO pin on a debug connector using Manchester encoding.



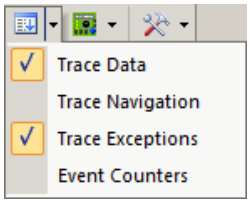
TIP: Only the ULINKpro can use this selection. It is unable to use UART/NRZ such as the ULINK2/ME or J-Link does. An error will result if you attempt this.

2) TracePort Trace: Serial Wire Viewer (SWV) is activated and is sent out the 4 bit (or 1 or 2) parallel port with a clock. No ETM instruction trace frames are available with this selection. This mode has the greatest SWV output available.

.

1. µVision must be not in Debug mode. This will be Edit mode.
2. In the Target Selection window, select Trace Port Trace. 

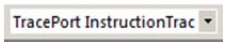
3. Compile the source files by clicking on the Rebuild icon.  . They will compile with no errors or warnings.
4. Click on the Load icon to program the Flash memory.  . A progress bar will be at the bottom left.
5. Enter the Debug mode by clicking and click on the RUN icon. 
6. Click on the small arrow beside the Trace icon and open the Trace Data and Trace Exception windows a shown here if they are not already open:
7. You will see all the SWV functions that work on the previous page also function here.



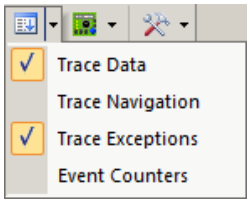
The difference between this selection and SWO Trace is here the SWV frames come out the 4 bit Trace Port as found on the Cortex Debug+ETM connector.
The SWO Trace comes out the 1 bit SWO pin found on all the debug connectors and shared with the JTAG TDO pin. This conflict is why SWD and not JTAG must be used with SWV. Because TracePort Trace comes out the 4 bit Trace Port, no conflict arises and you can use JTAG with SWV. Neither JTAG nor SWD have any significant speed advantages.

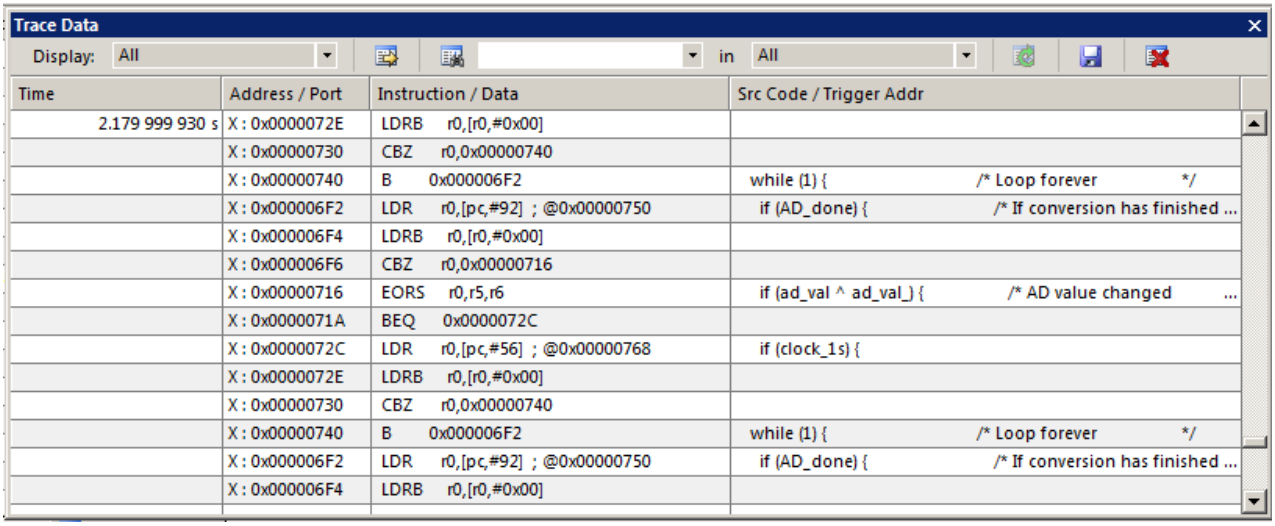
3) TracePort InstructionTrace: ETM trace is sent out the 4 bit Trace Port. SWV information is not displayed in the Trace Data window in this mode. The Exception Trace, Logic Analyzer and Debug (printf) Viewer windows will still function.

This mode has a high SWV output.

1. µVision must be not in Debug mode. This will be Edit mode.
2. In the Target Selection window, select TracePort InstructionTrace. 

3. Compile the source files by clicking on the Rebuild icon.  . They will compile with no errors or warnings.
4. Click on the Load icon to program the Flash memory.  . A progress bar will be at the bottom left.
5. Enter the Debug mode by clicking and click on the RUN icon. 
6. Click on the small arrow beside the Trace icon and open the Trace Data and Trace Exception windows a shown here if they are not already open:
7. You will see all the SWV functions that work on the previous page also function here.
8. Except SWV frames do not appear in the Trace Data window.
9. Stop the processor  .
10. The Trace Data window now displays all the instructions executed with Source if available:
11. Double-click on a trace frame and this instruction will be highlighted in the Disassembly and source windows.





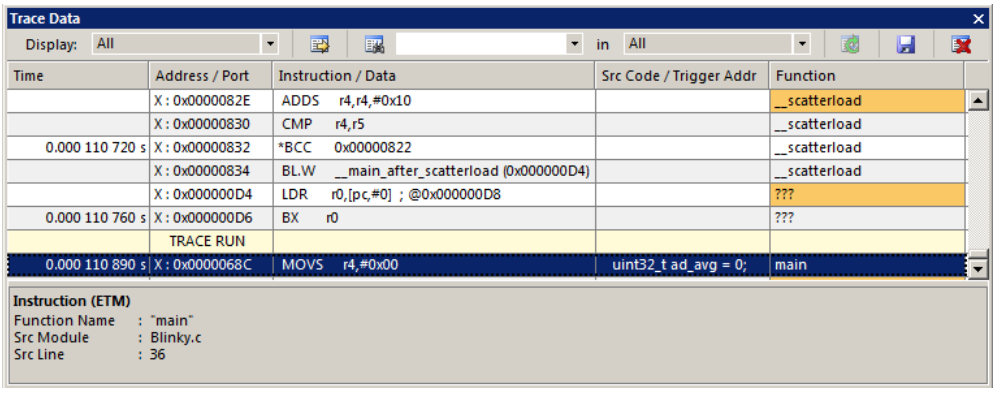
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
2.179 999 930 s	X : 0x0000072E	LDRB r0,[r0,#0x00]	
	X : 0x00000730	CBZ r0,0x00000740	
	X : 0x00000740	B 0x000006F2	while (1) { /* Loop forever */
	X : 0x000006F2	LDR r0,[pc,#92] ; @0x00000750	if (AD_done) { /* If conversion has finished ...
	X : 0x000006F4	LDRB r0,[r0,#0x00]	
	X : 0x000006F6	CBZ r0,0x00000716	
	X : 0x00000716	EORS r0,r5,r6	if (ad_val ^ ad_val_) { /* AD value changed ...
	X : 0x0000071A	BEQ 0x0000072C	
	X : 0x0000072C	LDR r0,[pc,#56] ; @0x00000768	if (clock_1s) {
	X : 0x0000072E	LDRB r0,[r0,#0x00]	
	X : 0x00000730	CBZ r0,0x00000740	
	X : 0x00000740	B 0x000006F2	while (1) { /* Loop forever */
	X : 0x000006F2	LDR r0,[pc,#92] ; @0x00000750	if (AD_done) { /* If conversion has finished ...
	X : 0x000006F4	LDRB r0,[r0,#0x00]	

More Information on TracePort InstructionTrace:

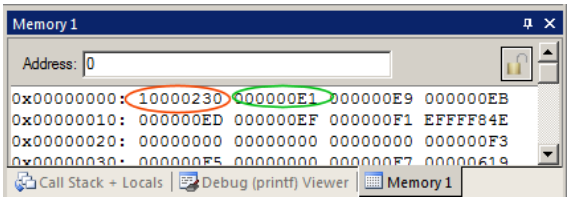
1. Select the Target Options windows.  and select the select the Debug tab.
2. Note there is an entry in the Initialization file: LPC17xx_TPIU.ini
3. This configures the GPIO port to allow Trace Port operation.
4. Select Edit… and this file will be displayed.
5. Click on OK to close this window.

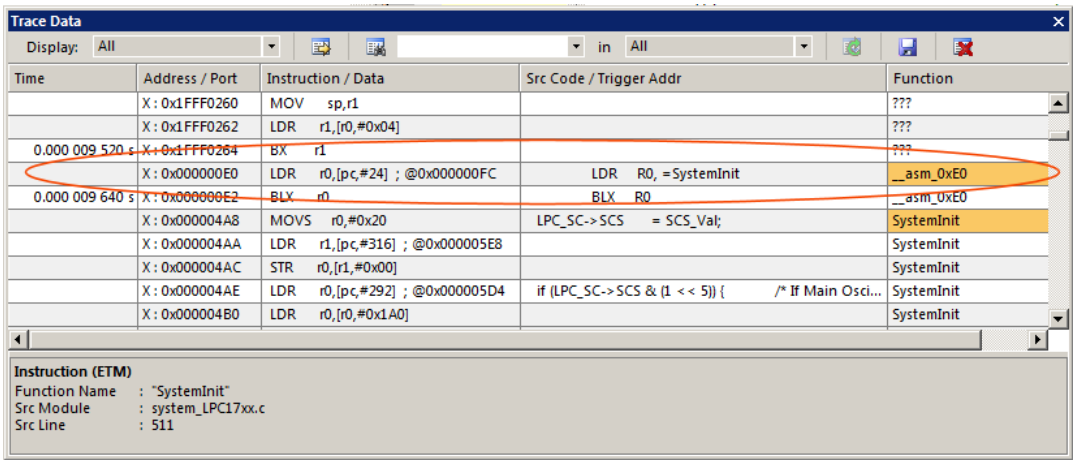
Looking how the ETM Instruction Trace works:

1. Exit Debug mode.  Immediately re-enter Debug mode  . This has the effect of resetting everything.
2. Do NOT click on RUN. The program runs (but does not execute) to the beginning of main().
3. The last instruction executed was probably a BX r0 at 0x00D6 as shown below at time 0.000 111 400 s.
4. Click on STEP  or press F11 to single-step once.
5. The first instruction in main() will be displayed as shown below: It will probably be a MOVS r4,#0x00
6. Scroll to the top of the Trace Data window. You will notice many instructions with an address with 0x1FFF 00xx.
7. This is part of the NXP internal initialization sequence.



8. Scroll down to where the addresses start with 0x00E0 as shown below: This is the start of the user code.
9. Use a memory window to view location 0x00. Address 0 contains the initial SP. Here it is 0x1000 0230:
10. Address 0x04 contains the initial PC. Here is it 0xE1. 0xE1-1 = 0xE0 will correspond to the first instruction as shown in the trace data window circled below. It is a LDR instruction.
11. It is apparent that ETM instruction trace is a power tool for debugging and providing information.



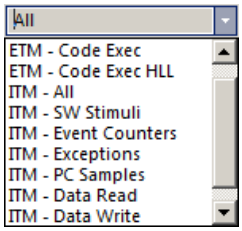


23) Finding the Trace Frames you are looking for:

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:

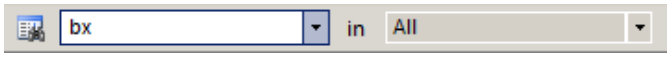
In the Trace Data window you can select various types of frames to be displayed. Open the Display: box and you can see the various options available as shown here: These filters are post collection. Future enhancements to µVision will allow more precise filters to be selected.



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

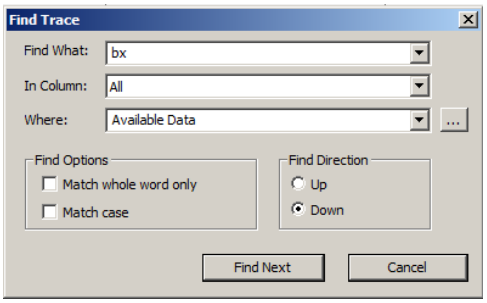
Find a Trace Record:

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



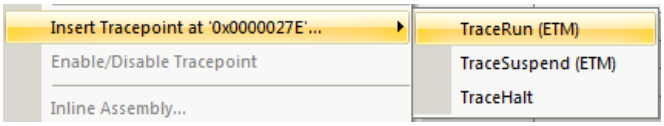
Note you can select properties where you want to search in the "in" box. All is shown in the screen above

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.



24) 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. A TraceStart will not restart the collection of trace. Can be resumed only with a STOP/RUN sequence.

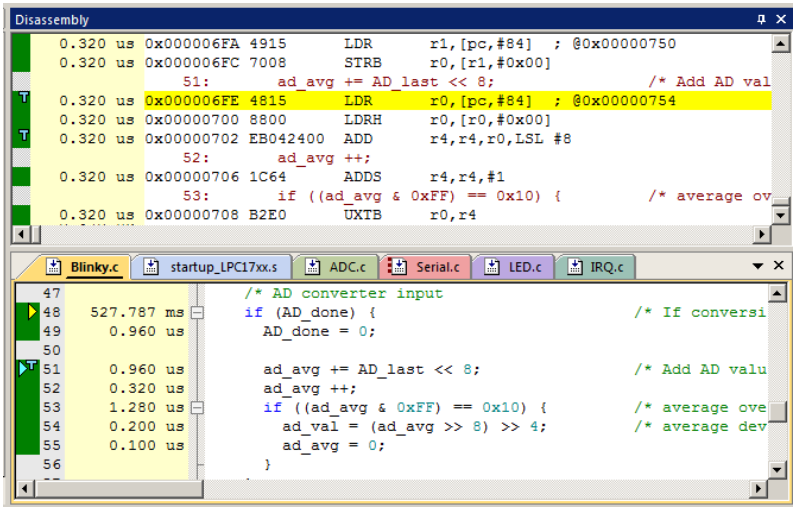


How it works:
When you set a TraceRun point in assembly language point, ULINK*pro* will start collecting trace records. When you set a TraceSuspend point, trace records collection will stop there. EVERYTHING in between these two times will be collected. This includes all instructions through any branches,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

The next page has a real example of Trace Triggers.

25) Setting Trace Triggers:

1. With Blinky in Debug mode, click on the C source line AD_avg += AD_Last << 8; near line 51 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 0x0000_1064 through 0x0000_1068 are displayed as belonging to this C source line.
3. Right-click on the instruction MOV and select Insert Tracepoint at 0x0000_1064 and select **TraceRun.** A cyan **T** will appear as shown:
4. Right-click on the instruction ADD and select Insert Tracepoint at 0x0000_1068 and select **TraceSuspend.** Two cyan **T** icons will appear as shown:
5. Clear the Trace Data window.
6. RUN the program and after a few seconds STOP it. Examine the first Trace Data window as shown below:



TIP: If you see any other frames than instructions, select ETM – Code Exec in the Display: window as shown below:.

You can see where the trace started on 0x0000_0700 and stopped on 0x0000_07106 multiple times shown below in the first Trace Data window.
In the first frame below it is clear the first instruction 0x6FE LDR was not recorded. There is also a skid of one instruction.
TIP: The ETM trace will collect everything between the TraceRun and TraceSuspend points. This will include any branches and exception calls.

1. Right click on 0x06FE and delete the existing trigger.
2. Right-click on the instruction STRS at 0x06FC and select Insert Tracepoint at 0x0000_06FC and select **TraceRun.**
3. RUN the program again and the second Trace Data window below will probably display depending on your settings.
4. When you are done, right-click on each trigger and select Remove Tracepoint at '0x*address*'

TIP: Once started, the trace will collect frames until it encounters a TraceSuspend or a TraceHalt point.

Trace recording with LDR at 0x06FE not recorded and one skid 0x0706 ADDS added.

Trace Data

Display: ETM - Code Exec in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x00000700	LDRH r0,[r0,#0x00]		main
	X : 0x00000702	ADD r4,r4,r0,LSL #8		main
0.000 000 000 s	X : 0x00000706	ADDS r4,r4,#1	ad_avg ++;	main
	TRACE RUN			
	X : 0x00000700	LDRH r0,[r0,#0x00]		main
	X : 0x00000702	ADD r4,r4,r0,LSL #8		main
+ 0.010 000 040 s	X : 0x00000706	ADDS r4,r4,#1	ad_avg ++;	main
	TRACE RUN			

Trace recording with LDR at 0x06FE recorded and one skid 0x0706 ADDS added.

Trace Data

Display: ETM - Code Exec in All

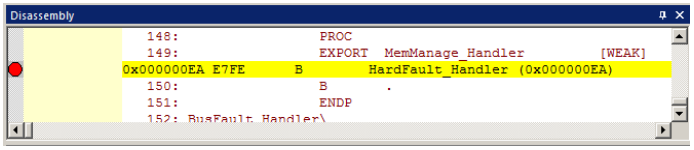
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x000006FE	LDR r0,[pc,#84] ; @0x00000754	ad_avg += AD_last << 8; /* Add AD value ...	main
	X : 0x00000700	LDRH r0,[r0,#0x00]		main
	X : 0x00000702	ADD r4,r4,r0,LSL #8		main
3.272 297 830 s	X : 0x00000706	ADDS r4,r4,#1	ad_avg ++;	main
	TRACE RUN			
	X : 0x000006FE	LDR r0,[pc,#84] ; @0x00000754	ad_avg += AD_last << 8; /* Add AD value ...	main
	X : 0x00000700	LDRH r0,[r0,#0x00]		main
	X : 0x00000702	ADD r4,r4,r0,LSL #8		main
3.282 300 550 s	X : 0x00000706	ADDS r4,r4,#1	ad_avg ++;	main

26) 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.

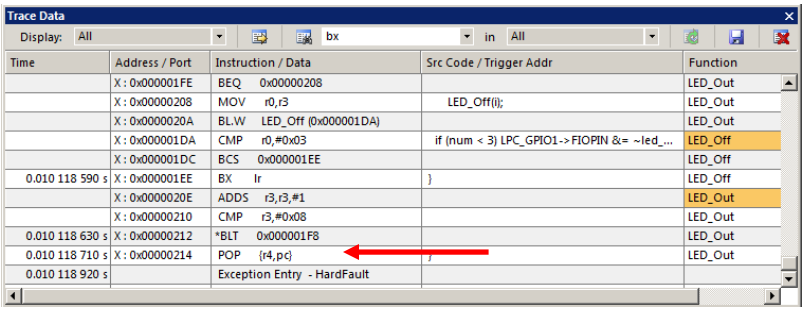
If a Hard Fault occurs in our example, the CPU will end up at 0x0000_00EA as shown in the disassembly window below. This is the Hard Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. The Trace Data 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.



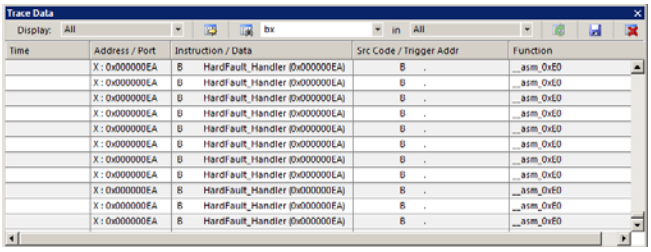
The difference this time is the breakpoint will stop the CPU and also the trace collection. The trace buffer will be visible and extremely useful to investigate and determine the cause of the crash.

1. Use the Blinky example from the previous exercise. Enter Debug mode.
2. Locate the Hard fault vector near address 0x0000_00EA in the Disassembly window or near line 145 in the file startup_LPC43xx.s.
3. Set a breakpoint at this point. A red circle will appear.
4. Find the function LED_Out in the file LED.c near line 55. Set a breakpoint on the start of LED_Out.
5. Click on RUN. The program will soon stop here.
6. Clear the Trace data window by clicking on the Clear Trace icon:  This is to help clarify what is happening.
7. In the Registers 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 0x0. (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. The Exception Entry frame is actually from SWV. EXCTRC is set. Note Display: is set to All.



12. To repeat click on RESET  .
13. Go to step 5.
14. Remove the breakpoint and click on RUN and then STOP.
15. You will now see all the Hard Fault branches as shown in the bottom screen:

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



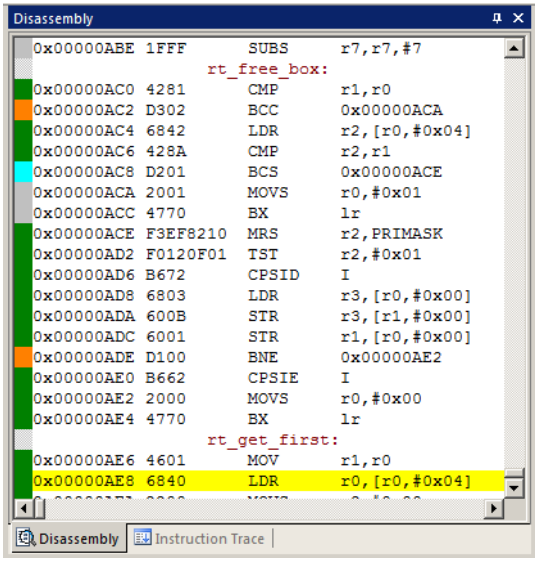
TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 145… and select TraceHalt. When Hard Fault is reached is reached, trace collection will halt but the program will keep executing the B instructions forever.

27) 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.

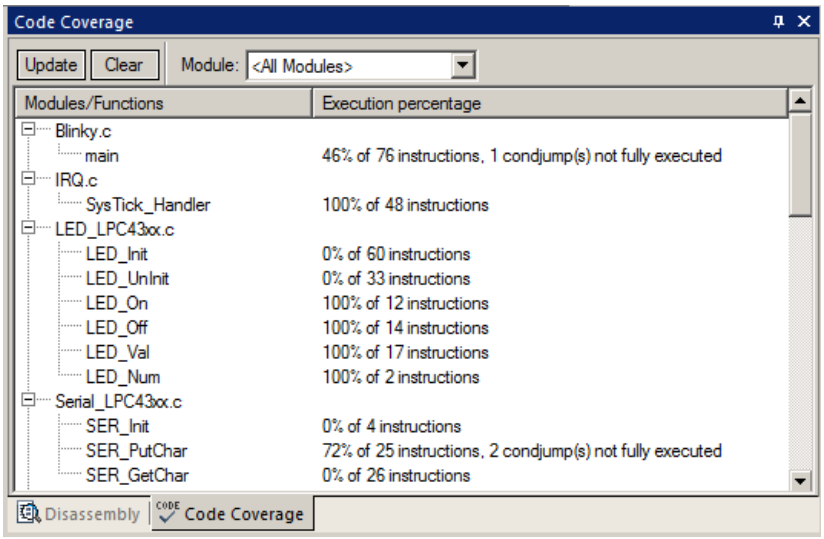
Why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

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.



28) 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
```

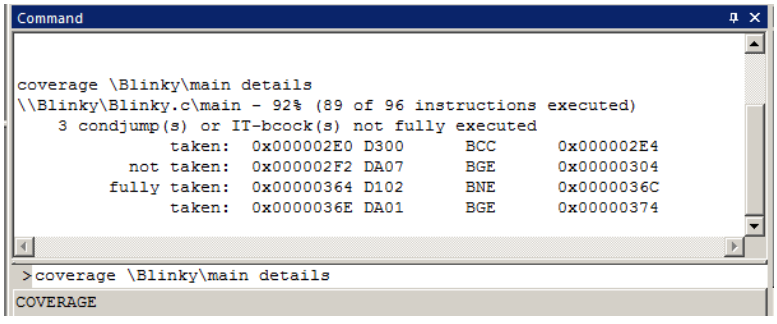
2. The command **`coverage asm`** produces this listing (partial is shown):

```
\\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.

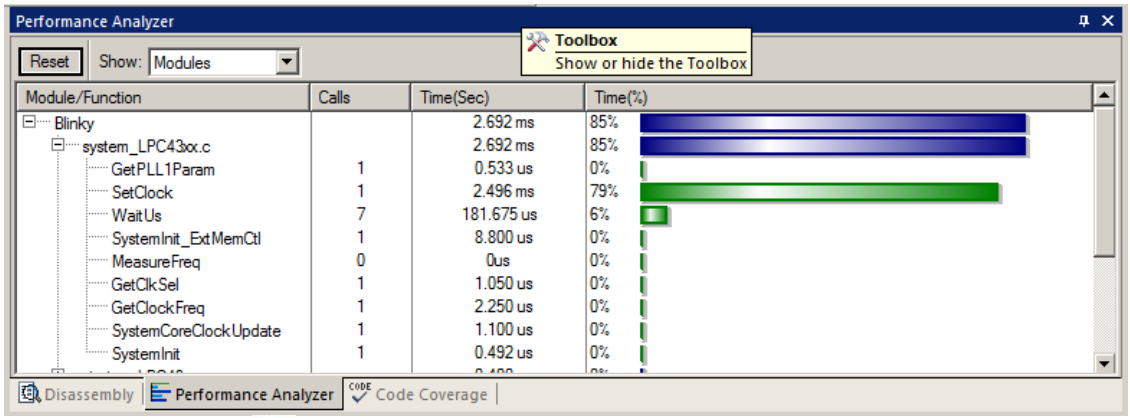
You can enter the command **`coverage`** with various options to see what is displayed in the Command window and saved.

29) Performance Analysis (PA):

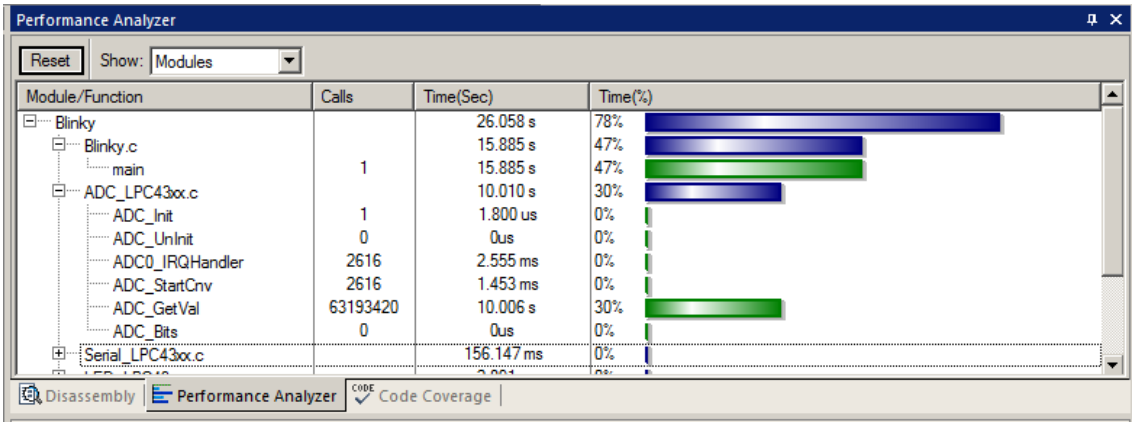
Performance Analysis tells you how much time was spent in each function. It is useful to help optimize your code for speed.

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.

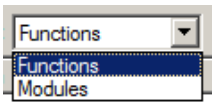
1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it.  This clears the PA window and resets the LPC4300 processor and reruns it to main(). You can also click 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.

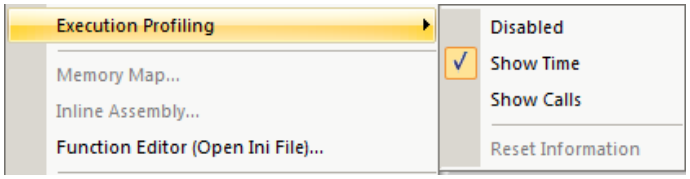


9. Click on the PA RESET icon.  Watch as new data is displayed in the PA window.
10. When you are done, exit Debug mode.

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

30) Execution Profiling:

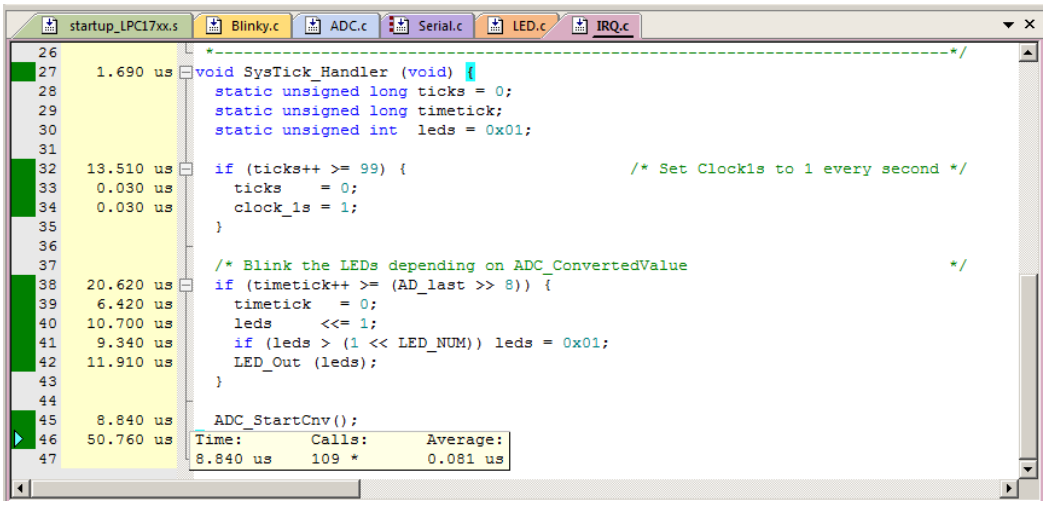
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.



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

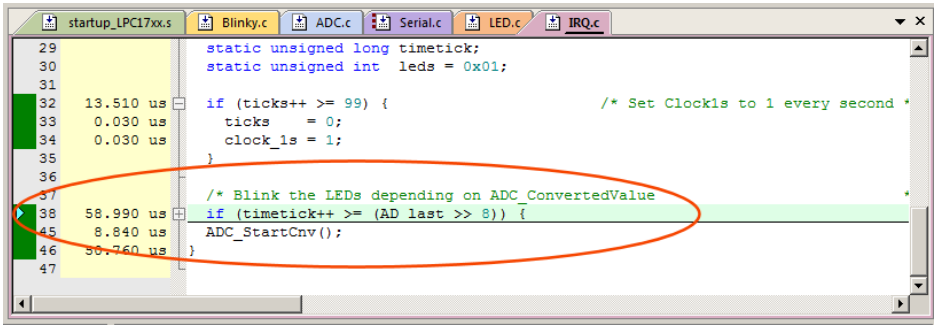
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



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.

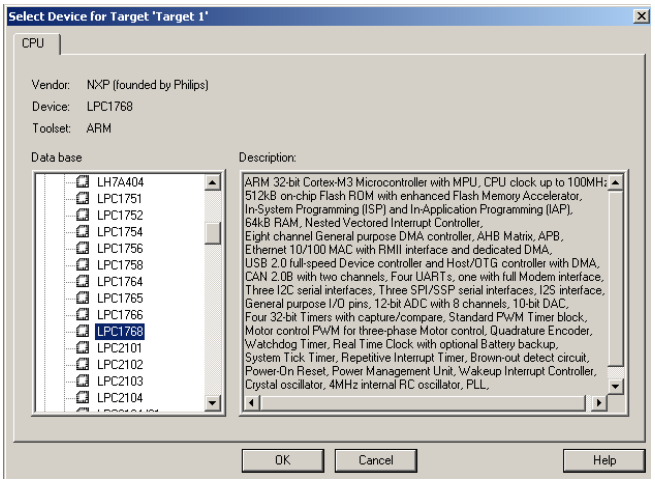


31) Creating a new project: Using the Blinky source files:

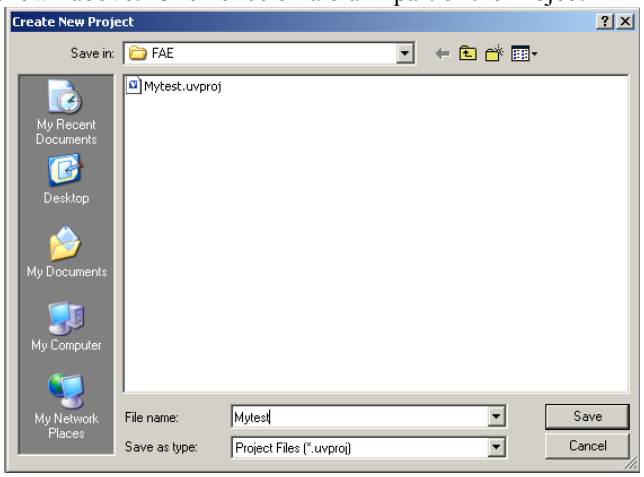
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run the Blinky example as usual. You can use this process to create any new project from your own source files created with µVision's editor or any other editor.

Create a new project called Mytest:

1. With µVision running and not in debug mode, select Project/New µVision Project.
2. In the window Create New Project go to the folder C:\Keil\ARM\Boards\Keil\MCB1700.
3. Right click and create a new folder by selecting New/Folder. I named this new folder FAE.
4. Double-click on the newly created folder "FAE" to enter this folder as is shown below.
5. Name your project. I called mine Mytest. You can choose your own name but you will have to keep track of it.
6. Click on Save.



7. "Select Device for Target 1" shown here opens up.
8. This is the Keil Device Database® which lists all the devices Keil supports (plus some secret ones).
9. Locate the NXP directory, open it and select LPC1768. Note the device features are displayed
10. Click on OK.
11. A window opens up asking if you want to insert the default LPC17xx startup file to your project. Click on "Yes". This will save you a great deal of time.
12. In the Project Workspace in the upper left hand of µVision, open up the folders by clicking on the "+" beside each folder.
13. We have now created a project called Mytest and the target hardware called Target 1 with one source file startup_LPC17xx.s.
14. Click once (carefully) on the name "Target 1" (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose LPC1700 as shown above. Click once on a blank part of the Project Workspace to accept this. Note the Target selector also changes. Click on the + to open up the directory structure. You can create many target hardware configurations including a simulator and easily select them.



Select the source files:

1. Using MS Explore (right click on Windows Start icon), copy blinky.c, core_cm3.c and system_LPC17xx.c from C:\Keil\ARM\Boards\Keil\MCB1700\Blinky to the Keil\MCB1700\FAE folder.
2. In the Project Workspace in the upper left hand of µVision, right-click on "LPC1700" and select "Add Group". Name this new group "Source Files" and press Enter.
3. Right-click on "Source Files" and select **Add files to Group "Source Files**".
4. Select the file Blinky.c, core_cm3.c and system_LPC17xx.c and click on Add and then Close. These will show up in the Project Workspace when you click on the + beside Source Files..
5. Select Target Options and select the Debug tab. Make sure ULINK Cortex Debugger is selected. Select this by checking the circle just to the left of the word "Use:".
6. At this point you could build this project and run it on your MCB1700 board.

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

32) Serial Wire Viewer Summary:

Serial Wire Viewer can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers – DMA bypasses CPU and SWV by definition.

Serial Wire Viewer displays in various ways:

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

Trace is good for:

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

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

- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), corrupted stack. *How did I get here ?*
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace.
- Communication protocol and timing issues. System timing problems.
- Profile Analyzer. Where is the CPU spending its time ?
- Code Coverage. Is a certification requirement. *Was this instruction executed ?*

For complete information on CoreSight for the Cortex-M3: Search for **DDI0314F_coresight_component_trm.pdf** on www.arm.com. You do not need to know the information in this document to use Serial Wire Viewer or the ETM trace.

Other Useful Documents:

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

33) 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 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 and search for university to view various programs and resources.

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.

See www.keil.com/nxp for more NXP specific information.

For more information:

Keil products can be purchased directly from ARM or through various distributors.

Keil Distributors: See www.keil.com/distis/

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

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

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

For the latest version of this document, see www.keil.com/nxp or www.keil.com/appnotes/docs/apnt_246.asp

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

KEIL™
Tools by ARM