NXP lab: Cortex-M3 Training *with Serial Wire Viewer*

LPC1768/65: Keil MCB1700 evaluation board

Summer 2010 Version 2.0 by Robert Boys, bob.boys@arm.com

Introduction:

The purpose of this lab is to introduce you to the NXP Cortex™-M3 processor using the ARM® Keil™ MDK toolkit featuring µVision®. We will use the Serial Wire Viewer (SWV) on the LPC1768 or LPC1765. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. Keil MDK supports all NXP ARM processors including ETM support. Check the Keil Device Database® on www.keil.com/dd for the complete list of NXP support. A similar lab is available for the STMicroelectronics STM32 and the Actel SmartFusion.

SWV allows real-time (no CPU cycles stolen) display of memory and variables, data reads and writes, exception events and program counter sampling plus some CPU event counters. ETM adds all the program counter values and is controlled with triggers and filters. SWV is supported by the Keil ULINK2, ULINK-ME and Segger J-Link adapters. ETM Trace is supported with either the ULINK*pro*, the Signum JtagJetTrace, or Segger J-Trace (under development).

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 limit. The addition of a license number will turn it into the full, unrestricted version. Contact Keil sales for a temporary full version license if you need to evaluate at greater than 32K. Keil also provides RL-ARM. This package includes the source files for the RTX RTOS, a TCP/IP stack, CAN drivers, a Flash file system and USB drivers.

Why Use Keil MDK ?

MDK provides these features particularly suited forCortex-M3 users: ARM® Keil™ MDK toolkit

1. µVision IDE with Integrated Debugger, Flash programmer and the RealView ARM compiler.
2. A full feature RTOS is included with MDK: RTX is a Keil product.
3. Serial Wire Viewer trace capability is included.
4. ETM Trace support with ULINK*pro.*
5. RTX Kernel Awareness window. It is updated in real-time
6. Choice of USB adapters: ULINK2, ULINK-ME, ULINK*pro*.
7. Kernel Awareness for Keil RTX, CMX, Quadros and Micrium. All RTOSs will compile with MDK.
8. Keil Technical Support is included for one year. This helps you get your project completed faster.

This document details these features:

1. Serial Wire Viewer (SWV) and ETM trace.
2. Real-time Read and Write to memory locations for Watch, Memory and RTX Tasks windows. Non-intrusive.
3. Breakpoints and Watchpoints (also called Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTOS – RTX.

Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and a timestamp. This information comes from the ARM CoreSight™ debug module integrated into the Cortex-M3. SWV does not steal any CPU cycles and is completely non-intrusive.

ETM Trace:

ETM (Embedded Trace Macrocell) adds all the program counter (PC) values to SWV. ETM is especially useful in finding PC related bugs such as classic “in the weeds”, spurious writes and stack problems. AN ETM emulator such as the Keil UKLINK*pro* (pictured above) also provides SWV and a very fast Flash programming time. UKLINK*pro* also provides an additional trace to source/assembly link in the µVision source windows.

Software Installation:

This document was written for Keil MDK 4.12 which contains µVision 4. MDK 4.12 is available on the Keil website. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence

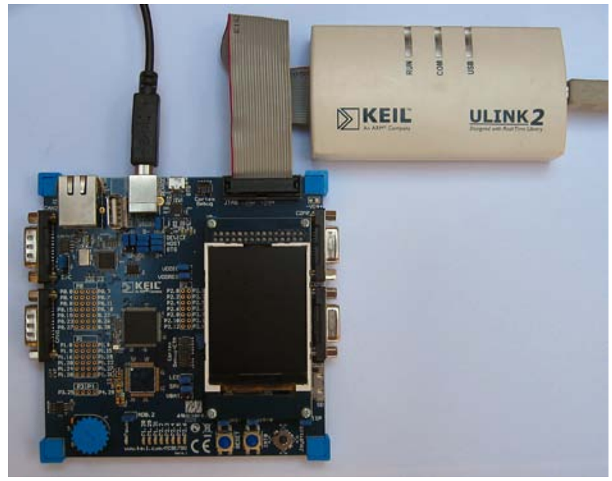
If you have a previous version of MDK, do not uninstall it; just install the new version on top. For a clean install, erase your project directories as well as those in C:\Keil\ARM\Boards. This is where the examples are stored.

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

Index:

1) *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.
   LPC1768 Flash This is where you select the Simulator or to execute a program in RAM or Flash.
4. Compile the source files by clicking on the Build icon.
5. Program the LPC1700 flash by clicking on the Load icon:
   LOAD Progress will be indicated in the Output Window.
6. 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.
7. Click on the RUN icon. Note: you stop the program with the STOP icon.

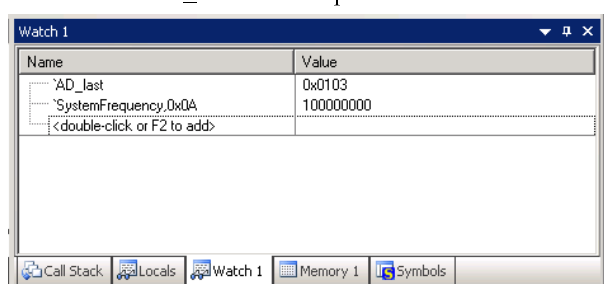
The LEDs on the MCB1700 will now blink at a speed according to the setting of the blue pot P7.

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

2) 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 NXP Cortex-M3 processors. It is also possible to "put" or insert values into these memory locations in real-time. It is possible to "drag and drop" variables into windows or enter them manually.

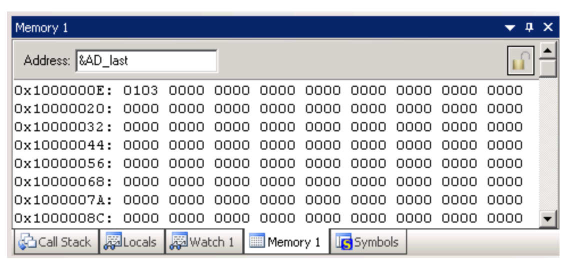
1. In the source file IRQ.c is the global variable **`AD_last`** near line 19. Select File/Open to access IRQ.c if needed.
2. Open the Watch window by clicking on the Watch 1 tab as shown below. AD_last will be updated in real-time.
3. If AD_last is not entered, double click it in IRQ.c to block it and drag 'n drop to the Watch 1 window. You can also enter the variable manually by double-clicking or pressing F2 and using copy and paste or typing the variable.



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. Drag 'n Drop **`AD_last`** into the Memory window or enter in manually.
2. 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.
3. Add an ampersand "&" in front of the variable name and press Enter. Now the address is shown (0x1000000E).
4. Right click in the memory window and select Unsigned/Short.
5. The data contents of **`AD_last`** is displayed as shown here:
6. 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. See the next page for an example.

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

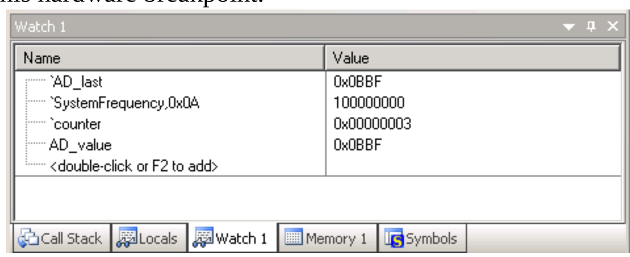
4. Stop the CPU and exit debug mode.
5. In the source file IRQ.c add a global variable counter near line 21 like this: **`unsigned int counter = 0;`**
6. In the function SysTick_Handler add the line **`counter++;`** just before **`ticks = 0;`** near line 35.
8. Compile the source files by clicking on the Build icon. .
9. Program the flash by clicking on the Load icon: and enter Debug mode. Click on the RUN icon.
7. Enter the variable **`counter`** in the Watch 1 window by your preferred method. Note it increments every second.
8. Double-click on the value field for **`counter`** in the Watch window.
9. When it is highlighted, enter 0x0 or just 0 and press Enter.
10. **`counter`** will be set to zero or to any other number you entered. You can also do this in the memory window.

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

11. Stop the program. Enter the local variable **`AD_value`** from main() in Blinky.c near line 95 to the Watch 1 window.
12. **`AD_value`** 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> will be displayed.
13. Start the program by clicking on the Run icon.
14. **`AD_value`** changes to ????????. Set a breakpoint by double-clicking in the margin beside the line **`clock_1s = 0;`** in main() around line 111. The program will soon stop on this hardware breakpoint.

TIP: You can set breakpoints on-the-fly in the Cortex-M3 !

15. **`AD_value`** displays the value as shown here:
16. 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_value`** static !

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

```
int main (void) {
        static short AD_value, AD_print;
```

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. 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 Options For Target , select the Utilities tab and select the “Update Target before Debugging” box.

5. Enter Debug mode. You will have to re-enter **`AD_value`** in the Watch 1 window because it isn’t the same variable anymore – it is a static variable now instead of a local. Drag ‘n Drop it in is the fastest way.
6. Remove the breakpoint you previously set and click on RUN. You can use Debug/Kill All Breakpoints to do this.
7. **`AD_value`** is now updated in real-time.
7. Stop the CPU and exit debug mode for the next step. and

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 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.

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

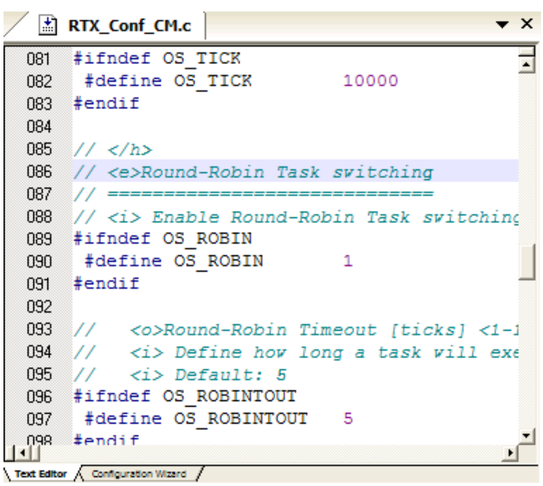
Keil provides RTX, a full feature RTOS. RTX is included for no charge as part of the Keil MDK full tool suite. It can have up to 255 tasks and no royalty payments are required. If source code is required, this is included in the Keil RL-ARM™ Real-Time Library which also includes USB, CAN, TCP/IP networking and a Flash File system. This example explores the RTOS project. Keil will work with any RTOS. An RTOS is merely another program that gets compiled.

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

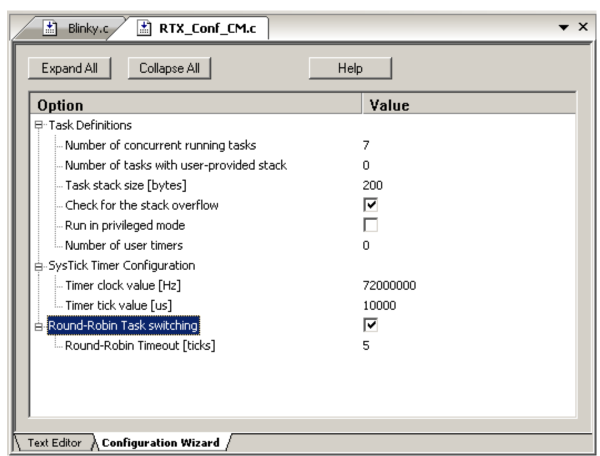
1. Start µVision4 by clicking on its icon on your Desktop if it is not already running.
2. Select Project/Open Project.
3. Open the file C:\Keil\ARM\Boards\Keil\MCB1700\RTX_Blinky\Blinky.uvproj.
4. Make sure “MCB1700” is selected in the Target window and not Simulator. MCB1700
   TIP: This is just the name of the target project. Any name can be used to distinguish different setups.
5. Compile the source files by clicking on the Build icon. . They will compile with no errors or warnings.
6. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
7. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
8. 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.
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. The new µVision4 System Viewer windows are created in a similar fashion.



Text Editor

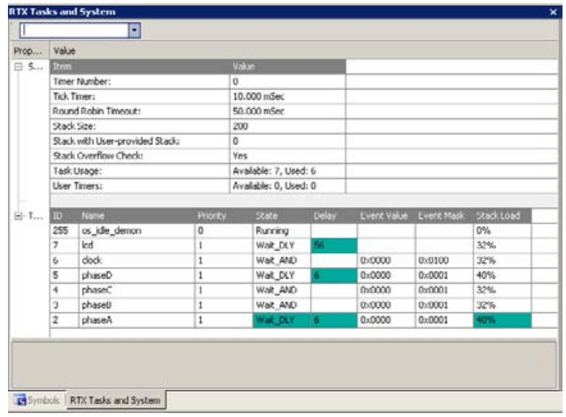


Configuration Wizard

TIP: µVision windows can be floated anywhere. You can restore them by setting Window/Reset Views to default.

4) RTX Kernel Awareness using Serial Wire Viewer

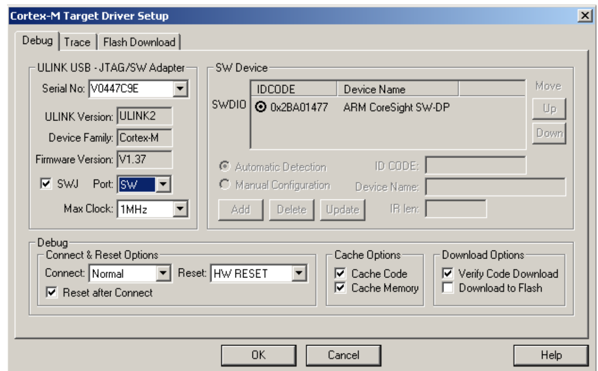
Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides a Task Aware window for RTX. Other RTOS companies also provide awareness for µVision.



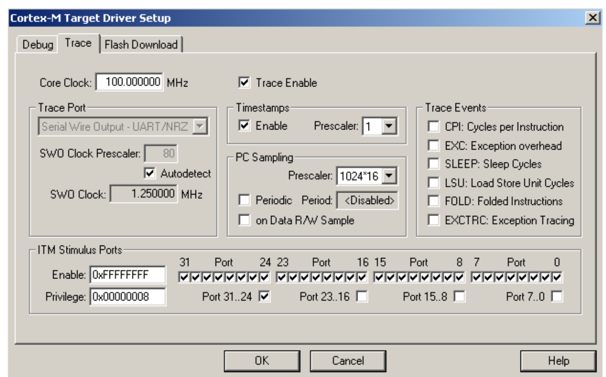
1. Run RTX_Blinky again by clicking on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. Note these values are updated in real-time using the same technology as used in the Watch and Memory windows.
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.

RTX Viewer: Configuring Serial Wire Viewer (SWV):

In order to get this working we have to activate the Serial Wire Viewer section of µVision.

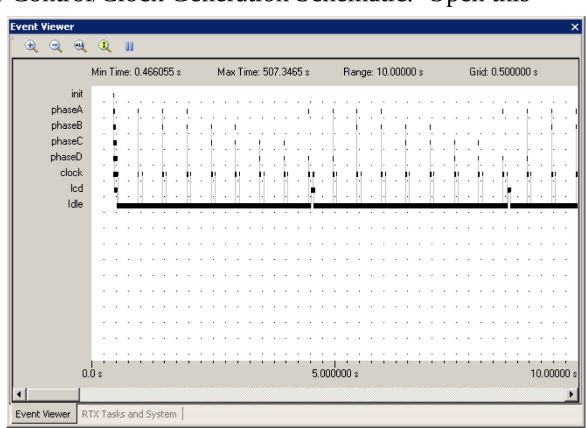


1. Stop the CPU and exit debug mode.
2. Click on the Options icon next to the target box.
3. Select the Debug tab and then click the Settings box next to ULINK Cortex Debugger dialog.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.
5. Click on the Trace 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.
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 to start the program.
10. Select Tasks and System and note the display is updated.
11. Note the values are updated with the program running.
12. Click on the RTX Tasks and System tab.
13. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 5 seconds by clicking on the AL and then the + and – icons.



TIP: View/Periodic Window Update must be selected !

TIP: To find the Core frequency select Peripherals/Clocking and Power Control/Clock Generation Schematic. Open this window now to see it. This is a very useful window. If you open this after RESET and before run you can see the basic frequency. This window can track changes in the PLL.



Cortex-M3 Alert: The LPC1700 will update all RTX information in real-time on a target board due to its Serial Wire Viewer and read/write capabilities as already described. You will not have to stop the program to view this data. No CPU cycles are used. Your program runs at full speed. You will find feature very useful !

TIP: Cortex-M0 processors do not have Serial Wire Viewer or ETM facilities. It is possible to use a LPC1700 to emulate a Cortex-M0. M0 executable code will run on a M3 without modification.

This technique will provide you with advanced debugging power including ETM trace to find those difficult bugs.

5) Logic Analyzer Window: view 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.

1. Close the RTX Viewer windows. 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:
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 082 and 085 (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 simple program there are not enough variables to connect to the Logic Analyzer. The program is too simple.

```
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  *----------------------------
```

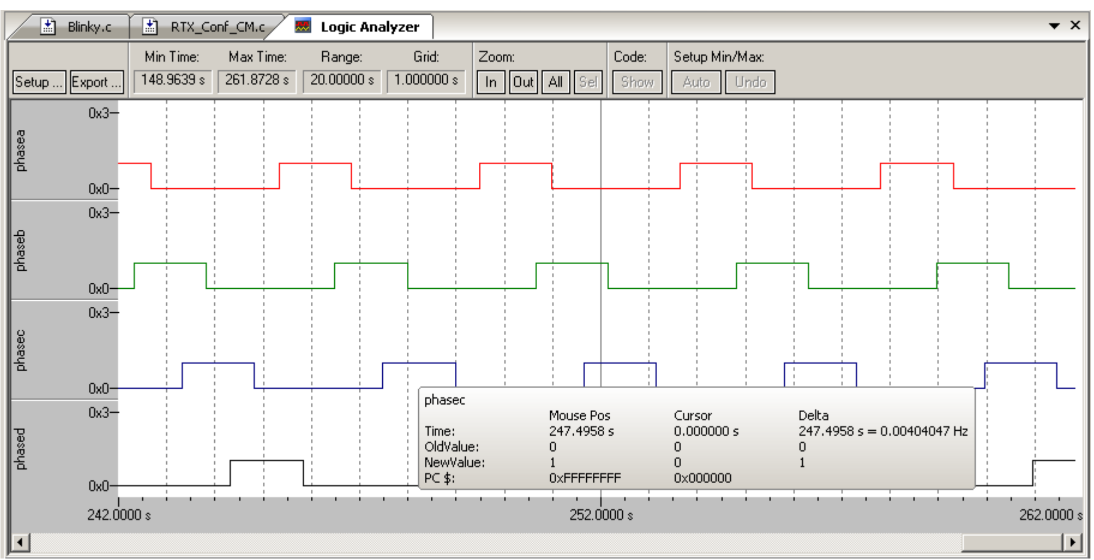
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

5. Build the project. Program the Flash and enter debug mode.
6. You can run the program at this point.
7. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.

```
073 /*------------------------------------------
074  *        Task 1 'phaseA': Phase A output
075  *------------------------------------------
076 __task void phaseA (void) {
077   for (;;) {
078     os_evt_wait_and (0x0001, 0xffff);    /*
079     LED_On (LED_A);
080     phasea=0;
081     signal_func (t_phaseB);              /*
082     LED_Off(LED_A);
083     phasea=1;
084   }
085 }
```

Enter the Variables into the Logic Analyzer:

8. Click on the Blinky.c tab. Block **`phasea`**, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
9. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release.
10. 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.
11. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
12. Click on Close to go back to the LA window.
13. Using the OUT and In buttons set the range to 20 seconds. Move the scrolling bar to the far right if needed.
14. You will see the following waveforms appear. Click to mark a place See 252 s below. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phasec:



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

6) Serial Wire Viewer (SWV) and how to use it:

Data Reads and Writes: (Note: Data Reads are disabled in the current version of µVision)

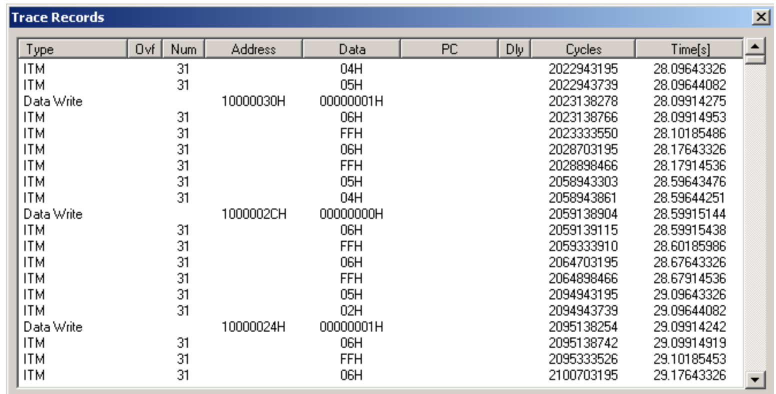
You have configured Serial Wire Viewer (SWV) in Section 4 under **RTX Viewer: Configuring the Serial Wire Viewer:**

Now we will examine some of the features available to you. SWV works with µVision and a ULINK2, ULINK-ME, ULINK*Pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. A user program runs at full speed and needs no code stubs or instrumentation software added to your programs.

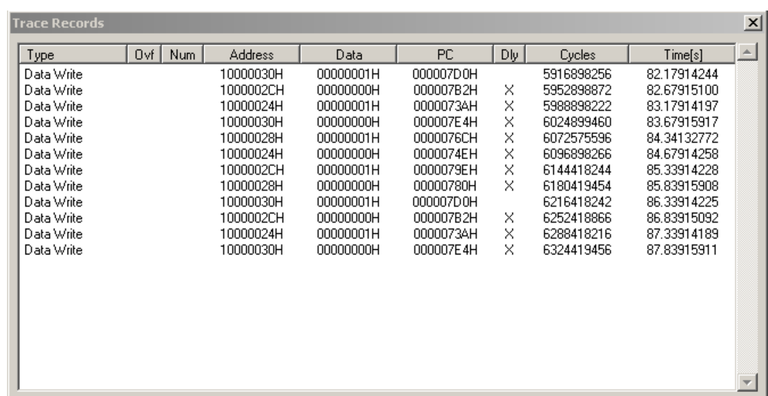
1. Use RTX_Blinky from the last exercise. Enter Debug mode and run the program if not already running.
2. Select View/Trace/Records or click on the Trace icon  and select Records.

3. The Trace Records window will open up as shown here:



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		04H			2022943195	28.09643326
ITM		31		05H			2022943739	28.09644082
Data Write			10000030H	00000001H			2023138278	28.09914275
ITM		31		06H			2023138766	28.09914953
ITM		31		FFH			2023333550	28.10185486
ITM		31		06H			2028703195	28.17643326
ITM		31		FFH			2028898466	28.17914536
ITM		31		05H			2058943303	28.59643476
ITM		31		04H			2058943861	28.59644251
Data Write			1000002CH	00000000H			2059138904	28.59915144
ITM		31		06H			2059139115	28.59915438
ITM		31		FFH			2059333910	28.60185986
ITM		31		06H			2064703195	28.67643326
ITM		31		FFH			2064898466	28.67914536
ITM		31		05H			2094943195	29.09643326
ITM		31		02H			2094943739	29.09644082
Data Write			10000024H	00000001H			2095138254	29.09914242
ITM		31		06H			2095138742	29.09914919
ITM		31		FFH			2095333526	29.10185453
ITM		31		06H			2100703195	29.17643326

4. The ITM entries are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. To turn this off select Debug/Debug Settings and click on the Trace tab. Unselect the ITM Stimulus Port 31.
5. Port 0, EXCTRC and Periodic can also be unselected.
   TIP: Port 0 is used for a printf.
6. Select On Data R/W Sample.
7. Click on OK to return.
8. Click on the RUN icon.
9. Double-click anywhere on the Trace records window to clear it.
10. Only Data Writes will appear now.
    TIP: You could have right clicked on the Trace Records window to filter these frames out another way.



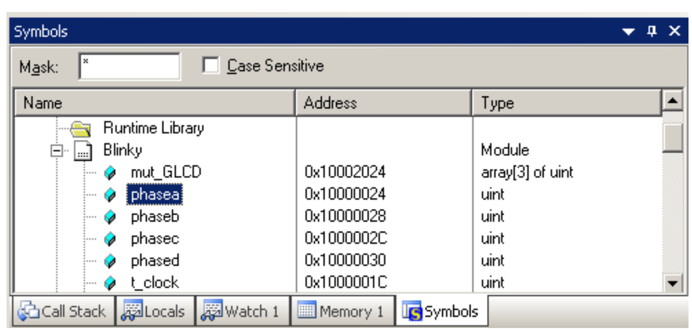
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000030H	00000001H	000007D0H		5916898256	82.17914244
Data Write			1000002CH	00000000H	000007B2H	X	5952898872	82.67915100
Data Write			10000024H	00000001H	0000073AH	X	5988898222	83.17914197
Data Write			10000030H	00000000H	000007E4H	X	6024899460	83.67915917
Data Write			10000028H	00000001H	0000076CH	X	6072575596	84.34132772
Data Write			10000024H	00000000H	0000074EH	X	6096898266	84.67914258
Data Write			1000002CH	00000001H	0000079EH	X	6144418244	85.33914228
Data Write			10000028H	00000000H	00000780H	X	6180419454	85.83915908
Data Write			10000030H	00000001H	000007D0H		6216418242	86.33914225
Data Write			1000002CH	00000000H	000007B2H	X	6252418866	86.83915092
Data Write			10000024H	00000001H	0000073AH	X	6288418216	87.33914189
Data Write			10000030H	00000000H	000007E4H	X	6324419456	87.83915911

What is happening here ?

1. When variables are entered in the Logic Analyzer (remember phasea through phased ?), the reads and/or writes will appear in Trace Records.
2. The Address column shows where the four variables are located.
3. The Data column are the data values written to phasea through phased.
4. PC is the address of the instruction causing the writes. You activated it by selecting On Data R/W Sample.
5. The Cycles and Time(s) columns are when these events happened.

TIP: You can have up to four variables in the Logic Analyzer and subsequently displayed in the Trace Records window.

TIP: If you select View/Symbol Window you can see where the addresses of the variables.



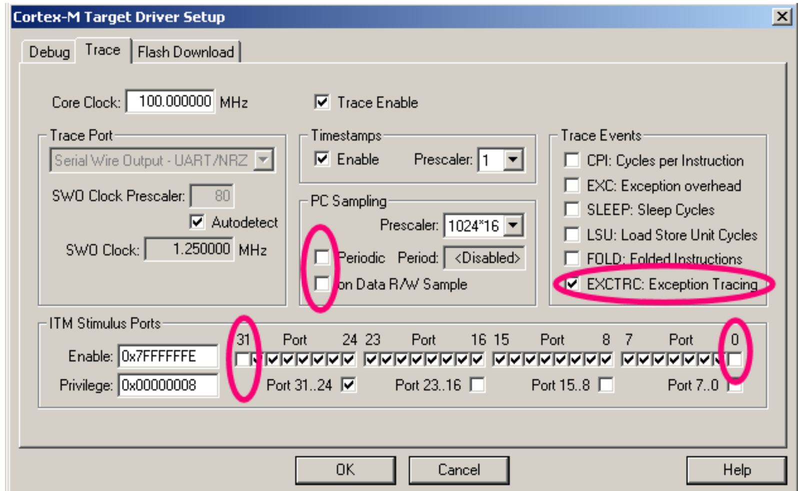
TIP: The ULINK*pro* displays the source and assembly code in a different style trace window. Double-clicking on a source line will take you to the appropriate place in the source and disassembly windows.

Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

Exceptions and Interrupts:

The LPC1700 family has many interrupts and it can be difficult to determine when they are being activated. SWV on the LPC1700 family makes this easy.

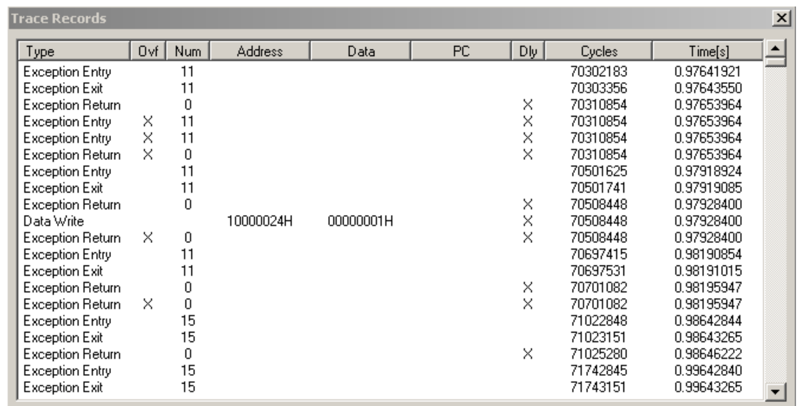
1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample and ITM Ports 31 and 0.
3. Select EXCTRC as shown here:



4. Click OK twice.
5. The Trace Records should still be open. 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 Exceptions frames.

What Is Happening ?

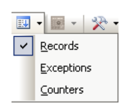
1. You can see two exceptions happening.
   - **Entry:** when the exception enters.
   - **Exit:** When it exits or returns.
   - **Return:** When all the exceptions have returned including any tail-chaining.
2. Num 11 is SVCall from the RTX calls.
3. Num 15 is the Systick timer.
4. In my example you can see one data write from the Logic Analyzer.
5. Note everything is timestamped.
6. The "X" in Ovf is an overflow and some data was lost. The "X" in Dly means the timestamps are delayed because too much information is being fed out the SWO pin.


Trace Records

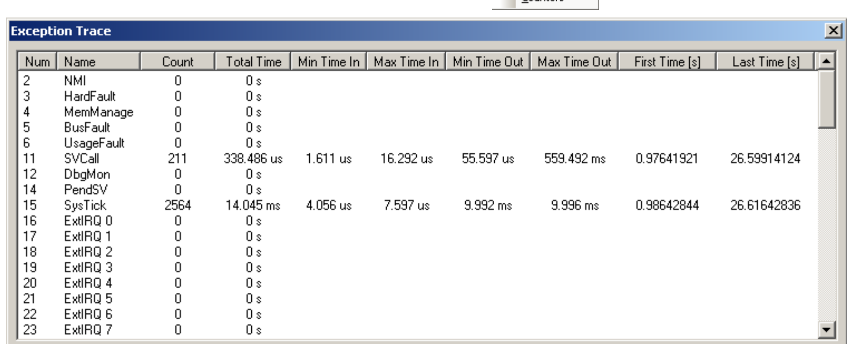
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		11					70302183	0.97641921
Exception Exit		11					70303356	0.97643550
Exception Return		0				X	70310854	0.97653964
Exception Entry	X	11				X	70310854	0.97653964
Exception Entry	X	11				X	70310854	0.97653964
Exception Return	X	0				X	70310854	0.97653964
Exception Entry		11					70501625	0.97918924
Exception Exit		11					70501741	0.97919085
Exception Return		0				X	70508448	0.97928400
Data Write			10000024H	00000001H		X	70508448	0.97928400
Exception Return	X	0				X	70508448	0.97928400
Exception Entry		11					70697415	0.98190854
Exception Exit		11					70697531	0.98191015
Exception Return		0				X	70701082	0.98195947
Exception Return	X	0				X	70701082	0.98195947
Exception Entry		15					71022848	0.98642844
Exception Exit		15					71023151	0.98643265
Exception Return		0				X	71025280	0.98646222
Exception Entry		15					71742845	0.99642840
Exception Exit		15					71743151	0.99643265

TIP: The SWO pin is one pin on the LPC1700 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate.

1. Select View/Trace/Exceptions or click on the Trace icon and select Exceptions.



2. The next window opens up and more information about the exceptions are dispalyed as shown.


Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	211	338.486 us	1.611 us	16.292 us	55.597 us	559.492 ms	0.97641921	26.59914124
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	2564	14.045 ms	4.056 us	7.597 us	9.992 ms	9.996 ms	0.98642844	26.61642836
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						

3. Note the number of times these have happened under Count. This is very useful information in case interrupts come too fast or slow.
4. ExtIRQ are the peripheral interrupts.
5. You can clear this trace window by double-clicking on it.
6. All this information is displayed in real-time and without stealing CPU cycles !

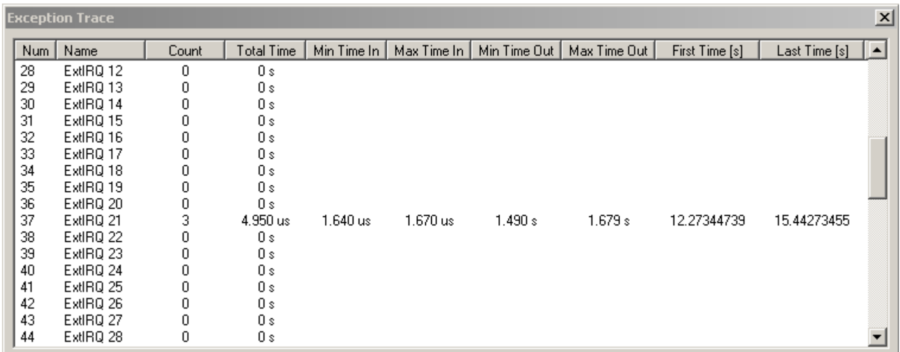
TIP: Num is the exception number: RESET is 1. External interrupts start at Num 16. For LPC1768, 41 is CAN IRQ. This is found in the LPC17xx Users Manual. Num 41 is also known as 41-16 = External IRQ 25.

External Interrupt Example: EXTI

Serial Wire Viewer can help debug many tricky interrupt issues. A special project, EXTI, is available to demonstrate these powerful SWV features. EXTI is an example program in µVision. Serial Wire Viewer is already configured in EXTI.

The button INT0 is connected to a GPIO port and each time it is pressed an interrupt is generated.

1. Open the project C:\Keil\ARM\Boards\Keil\MCB1700\EXTI\Exti.uvproj.
2. Build, program the Flash and enter Debug mode. Run the program.
3. The Trace Records and Exception Trace windows should be open. Open them if they are not.
4. Clear them by double-clicking anywhere inside them.
5. Press the INT0 button and ExtIRQ 21 (Number 37) will display in the Exception Trace window as shown below.
6. The LEDs on the board will also advance.
7. You may have to scroll down to see this. Press INT0 multiple times and ExtIRQ will count the number each time.



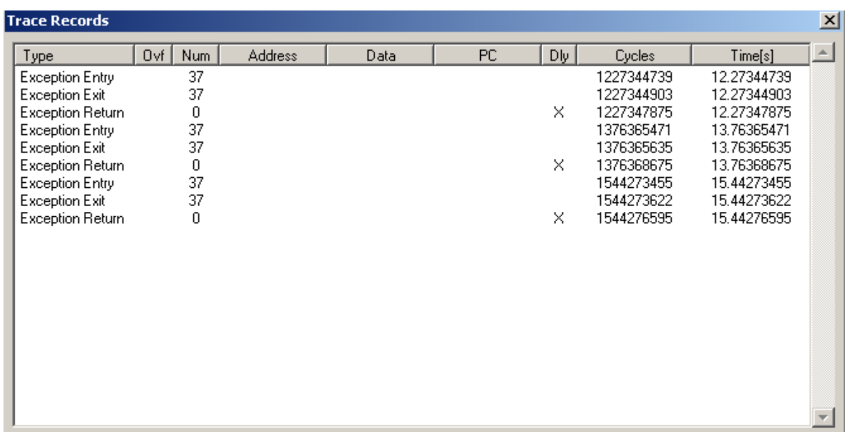
Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
28	ExtIRQ 12	0	0 s						
29	ExtIRQ 13	0	0 s						
30	ExtIRQ 14	0	0 s						
31	ExtIRQ 15	0	0 s						
32	ExtIRQ 16	0	0 s						
33	ExtIRQ 17	0	0 s						
34	ExtIRQ 18	0	0 s						
35	ExtIRQ 19	0	0 s						
36	ExtIRQ 20	0	0 s						
37	ExtIRQ 21	3	4.950 us	1.640 us	1.670 us	1.490 s	1.679 s	12.27344739	15.44273455
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						
43	ExtIRQ 27	0	0 s						
44	ExtIRQ 28	0	0 s						

8. The Trace Records window displays more information concerning the exceptions.

Exception Entry 37: Exception 37 is entered.

Exception Exit 37: Exception 37 exits.

Exception Return: All exceptions are finished. This will show any Cortex-M3 tail-chaining.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		37					1227344739	12.27344739
Exception Exit		37					1227344903	12.27344903
Exception Return		0				X	1227347875	12.27347875
Exception Entry		37					1376365471	13.76365471
Exception Exit		37					1376365635	13.76365635
Exception Return		0				X	1376368675	13.76368675
Exception Entry		37					1544273455	15.44273455
Exception Exit		37					1544273622	15.44273622
Exception Return		0				X	1544276595	15.44276595

TIP: This is a very simple example showing the usefulness of Serial Wire Viewer in displaying interrupts.

The Entry, Exit and Return (of all IRQs from any tail-chaining) is shown correctly and timestamped.

TIP: Exceptions:

Interrupts are a subset of Exceptions in ARM terminology.

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned including any Cortex-M3 tail-chaining.

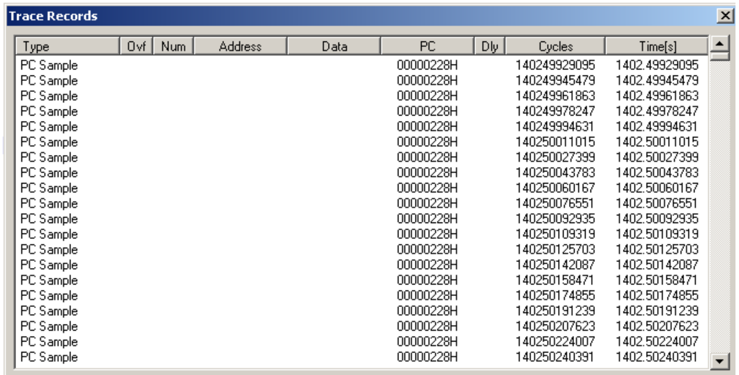
If the Exception Return 0 was missing: this means the CPU went directly from one IRQ to the next IRQ without the regular push-pop sequence. This is Cortex-M3 tail chaining. Tail chaining saves a great deal of CPU time.

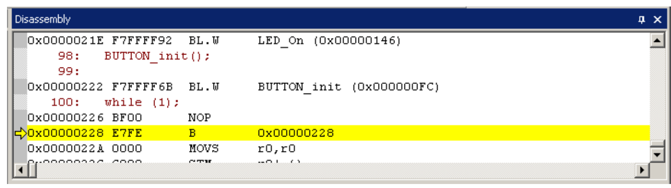
PC Samples:

Serial Wire Viewer can only display a sampling of the program counter. To capture all of the PCs use the ETM trace. ETM is perfect to find problems associated with program flow such as “I went into the weeds and how did I get here?”.

SWV can display at best every 64th instruction. It is better to keep this number as high as possible to avoid overloading the Serial Wire Output (SWO) pin. This is easily set in the Trace configuration window.

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear it.
5. Click on RUN and this window opens:
6. Most of the PC Samples are 228 which is a branch to itself in a loop forever routine.
7. Stop the program and the Disassembly window will show this Branch. The only time the program goes elsewhere is when you press the INT0 button.
8. Not all the PCs will be captured. Still, PC Samples can give you some idea of where your program is; especially if it is caught in a loop (like at 0x228).
9. **Note:** you can get different PC values depending on the optimization level of µVision.
10. Set a breakpoint in the IRQ handler.
11. Run the program and press the INT0 button and the program will stop at somewhere other than 228.
12. Scroll to the bottom of the Trace Records window and notice that this instruction (usually 0x332) was missed.





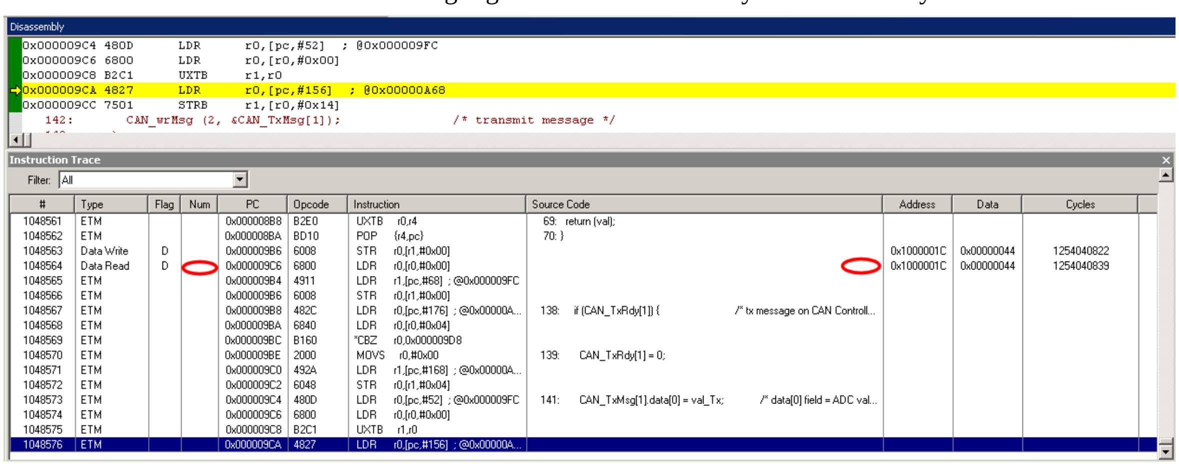
TIP: If you need to see these instructions, use ETM trace. ETM records all the instructions.

7) ETM Trace: (You need a ULINK*Pro* or Signum Systems JtagJetTrace for this step)

ETM captures all the program counters and is especially useful for timing issues or where problems or their causes disappear with time. The Serial Wire Viewer information is also contained in the ETM Trace window as shown below.

This screen is from the CAN example at the bottom of page 13. ETM trace shows the data read 0f 0x44 at the red circles but also all the instructions that were executed following the Watchpoint activation. This is known as “skid” and is normal for data breakpoints.

Note the source lines, disassembled instructions and data read and write values in the Instruction Trace window. I double-clicked on the last line and this instruction was highlighted in the Disassembly window at the yellow line.



8) ITM (Instruction Trace Macrocell)

Recall in Section 4) RTX Kernel Awareness on page 6 that we showed you can display information about the RTOS in real-time. This is done through the ITM Stimulus Port 31. ITM 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 in the Debug (printf) Viewer window.

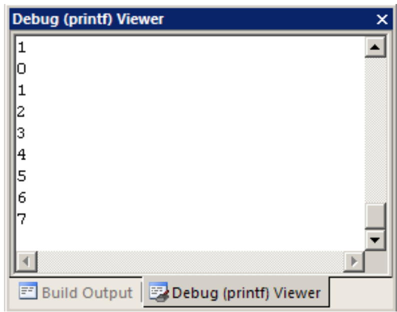
1. If necessary stop the program execution of Exti.c and exit debug mode.
2. Add this code to Exti.c. A good place is near line 22, just after the two #include directives..

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

3. In the function EINT3_IRQHandler() enter these five lines near line 82 just after the line **`idxOld = idxCur;`** :

```
ITM_Port8(0) = idxCur + 0x30;     /*  displays LED position: +0x30 converts to ASCII */
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. The default trace settings can be used. The checkbox ITM Stimulus Ports "0" enables the Debug (prinftf) Viewer.
6. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
7. In the Debug (printf) Viewer you will see the Led position number appear every time you press the EXT0 button. This is shown in the Debug (printf) Viewer window shown here. The global variable **`idxCur`** value is displayed.
8. Open View/Watch/Watch1. Enter variable **`idxCur`** if necessary. This will increment as you press the INT0 button.

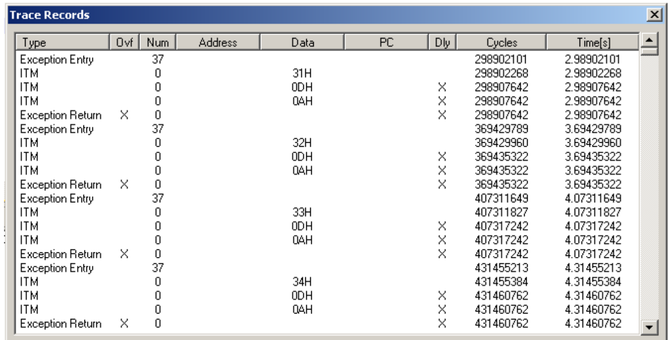


Trace Records

1. Open the Trace Records if not already open. Double click on it to clear it.
2. You will see a window such as the one below with ITM Exception frames.

What Is This ?

1. You can see Exception 37 Enter, the three ITM writes and Exception 37 Exit.
2. ITM 0 frames (Num column) are our ASCII characters from **`idxCur`** with carriage return (0D) and line feed (0A) as displayed the Data column.
3. All these are timestamped in both CPU cycles and time in seconds.
4. Note the "X" in the OVF column. This means some frame were lost due to SWO pin overload.
5. Click on Debug/Debug Setting and select the Trace tab.
6. Unselect Timestamps. Click on OK and then RUN.
7. On the Trace Records shown below, note no frames are lost and the Exception Return 0 now shows.



Trace Records

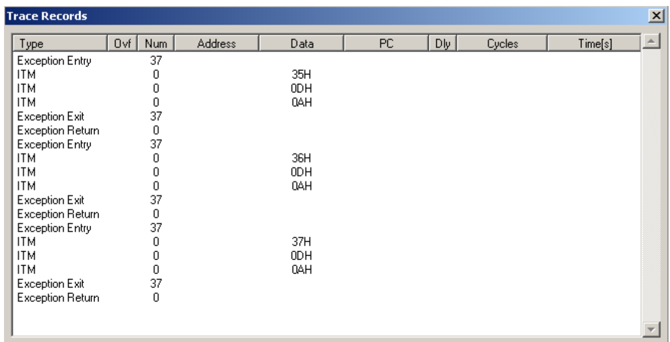
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		37					298902101	2.98902101
ITM		0		31H			298902268	2.98902268
ITM		0		0DH		X	298907642	2.98907642
ITM		0		0AH		X	298907642	2.98907642
Exception Return	X	0				X	298907642	2.98907642
Exception Entry		37					369429789	3.69429789
ITM		0		32H			369429960	3.69429960
ITM		0		0DH		X	369435322	3.69435322
ITM		0		0AH		X	369435322	3.69435322
Exception Return	X	0				X	369435322	3.69435322
Exception Entry		37					407311649	4.07311649
ITM		0		33H			407311827	4.07311827
ITM		0		0DH		X	407317242	4.07317242
ITM		0		0AH		X	407317242	4.07317242
Exception Return	X	0				X	407317242	4.07317242
Exception Entry		37					431455213	4.31455213
ITM		0		34H			431455384	4.31455384
ITM		0		0DH		X	431460762	4.31460762
ITM		0		0AH		X	431460762	4.31460762
Exception Return	X	0				X	431460762	4.31460762

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.

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enter only those features that you really need.

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



Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		37						
ITM		0		35H				
ITM		0		0DH				
ITM		0		0AH				
Exception Exit		37						
Exception Return		0						
Exception Entry		37						
ITM		0		36H				
ITM		0		0DH				
ITM		0		0AH				
Exception Exit		37						
Exception Return		0						
Exception Entry		37						
ITM		0		37H				
ITM		0		0DH				
ITM		0		0AH				
Exception Exit		37						
Exception Return		0						

9) Watchpoints: *Conditional Breakpoints*

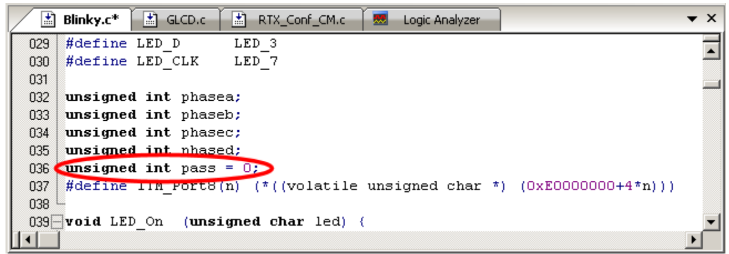
LPC1700 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. Often µVision will take one and perhaps two breakpoints for its operations. The LPC1700 also has four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses watchpoints in its operations. This means in µVision you must have two variables free in the Logic Analyzer to use Watchpoints.

1. Open the project RTX_Blinky that you used before.
2. Add this line in Blinky.c in the area where you declared phasea. This means we want this to be a global variable.

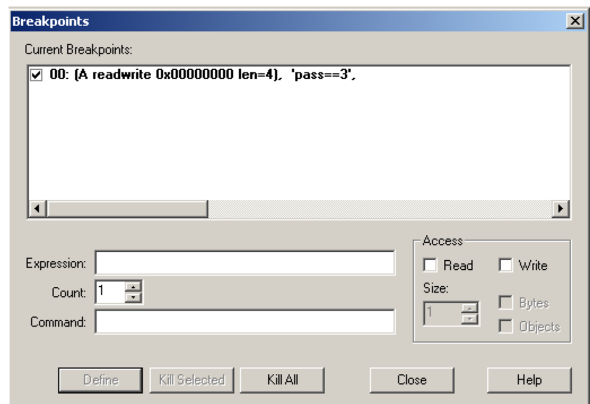
   ```
   unsigned int pass = 0;
   ```

3. Your result should look similar to the screen displayed below for the declaration of **`pass`**.
4. In task1 near phasea=0; enter this line:

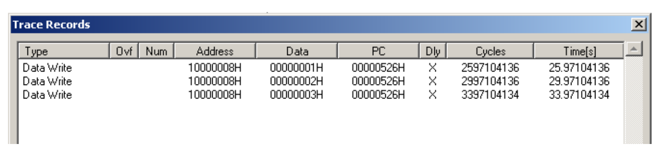
   `pass++;` **`Note:`** It is not important exactly where in Task 1 this is entered.



5. Compile the project and program the Flash.
6. Enter Debug mode.
7. Remove all variables in the Logic Analyzer window if there any by clicking on "Setup" and selecting the "Kill All" button.
8. Add variable **`pass`** to the Logic Analyzer. Set the Display Range to Min 0x0 and Max 0x5.
9. Click on Close to return. Set Range to 20 seconds by using the Zoom: Out button in the Logic Analyzer window.



10. Select the Debug tab and select Breakpoints or press Ctrl-B.
11. In the Expression box enter: pass = = 3. Select both the Read and Write Access.
12. Click on Define and it will be accepted as shown on the right here:
13. Click on Close.
14. Enter the variable pass to the Watch 1 window by dragging and dropping it or enter manually.
15. Open Debug/Debug Settings and select the trace tab. Check "on Data R/W sample" and uncheck both EXTRC and ITM 31.
16. Click on OK. Open the Trace Records window. Click on RUN.
17. When **`pass`** equals 3, the program will stop. This is how a Watchpoint works.
18. You will see **`pass`** incremented in the Logic Analyzer as well as in the Watch window.
19. Note the three data writes in the Trace Records window shown here. 1, 2 and 3 in the Data column. Plus the address written to and the PC of the write instruction.



20. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.
21. To repeat this exercise, click on the RESET icon and then RUN. You can also set the value in the watch window to 0.

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

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

TIP: The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it.

10) CAN: Controller Area Network

CAN is a network that is easy to implement with minimal software needed. 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 LPC2300 and LPC1700 series obtain the CAN Primer from www.keil.com. Exercises are also available for the STM32.

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.
2. Start µVision4 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 Build icon. . They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
7. The LCD 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.

An external CAN analyzer displays the CAN frames transmitted as shown here: → → →

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

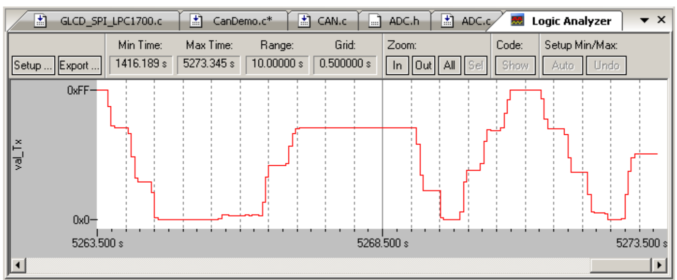
The CAN Identifier is 21 (ID column) and the data values displayed. There is once data byte per frame in this case. It is possible to have from 0 to 8 data bytes per frame.

TIP: If only Tx: changes, either the loopback cable isn't connected or you are using a ULINK-ME to power the board. This board must be powered by a USB cable for CAN to work properly.

Logic Analyzer Window:

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

1. Stop the program end exit Debug mode.
2. Open the Options for Target, Select the Debug tab, Settings and then the Trace tab. Ensure the Trace window is set to 100 mHz, Trace is enabled. Uncheck Periodic. Select Data R/W Sample. Select EXCTRC. Click Close twice.
3. Enter Debug mode.
4. Insert the global variable **`val_Tx`** that is declared in CanDemo.c into the Logic Analyzer window with a range from min 0 to max 0xFF.
5. Insert **`val_Tx`** into the Watch window in the usual way.
6. Open the Trace Records window. RUN the program.
7. 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.
8. The trace records window will show both the CAN interrupt (EXTIRQ 41) Entry and Exit and the data write to the variable **`val_Tx`**. All are time stamped.



Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			1000001CH	00000081H			69256483	0.69256483
Data Read			1000001CH	00000081H		X	69258051	0.69258051
Exception Entry		41					69267853	0.69267853
Exception Exit		41					69268220	0.69268220
Exception Return		0				X	69272451	0.69272451
Data Read			1000001CH	00000081H			69297151	0.69297151
Data Write			1000001CH	00000081H			75142131	0.75142131
Data Read			1000001CH	00000081H		X	75143731	0.75143731
Exception Entry		41					75153473	0.75153473
Exception Exit		41					75153840	0.75153840
Exception Return		0				X	75158131	0.75158131
Data Read			1000001CH	00000081H			75182799	0.75182799
Data Write			1000001CH	00000081H			81027779	0.81027779
Data Read			1000001CH	00000081H		X	81029331	0.81029331
Exception Entry		41					81039093	0.81039093
Exception Exit		41					81039460	0.81039460
Exception Return		0				X	81043731	0.81043731
Data Read			1000001CH	00000081H			81068447	0.81068447
Data Write			1000001CH	00000081H			86913427	0.86913427
Data Read			1000001CH	00000081H		X	86915011	0.86915011

9. You can successfully use Serial Wire Viewer to trace many CAN problems faster and easier than with traditional Stop 'n Go debugging.
10. For this example using ULINKPro and ETM Trace please see section **7) ETM Trace:** on page 10.

Using Watchpoints and Serial Wire Viewer with CAN

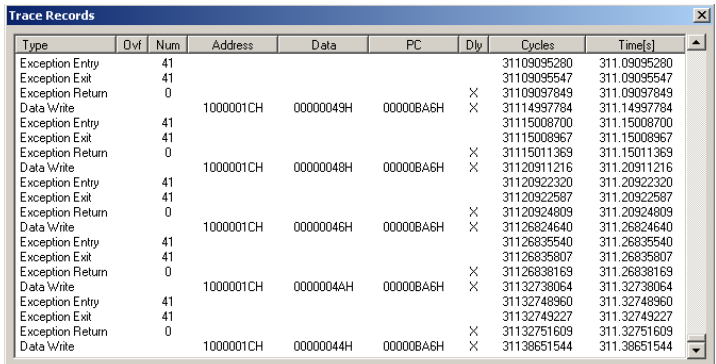
1. Stop the program if still running.
2. Open Debug/Breakpoints and enter in the dialog box: **val_Tx == 0x44** Select Read and click on Define and then Close.
3. Double-click in the Trace Records box to clear it and run the program by clicking on GO.
4. Adjust the pot to indicate 0x44. The first time this value is written to val_Tx, the program will stop.
5. Note the value in the Watch window will equal 0x44 ! The LCD may or may not have been updated yet.
6. Scroll to the bottom of the Trace Records and the value of 0x44 will be visible on the last line.

What is happening: Note the last frame is a data write of 0x44, to memory 0x10000001C by the instruction located at 0xBa6. This is the Watchpoint trigger point.

The instruction doing the write operation is made visible when you selected on Data R/W sample in the Trace configuration menu.

You can also see the CAN EXTIRQ 41 occurring. 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 are great with trace.

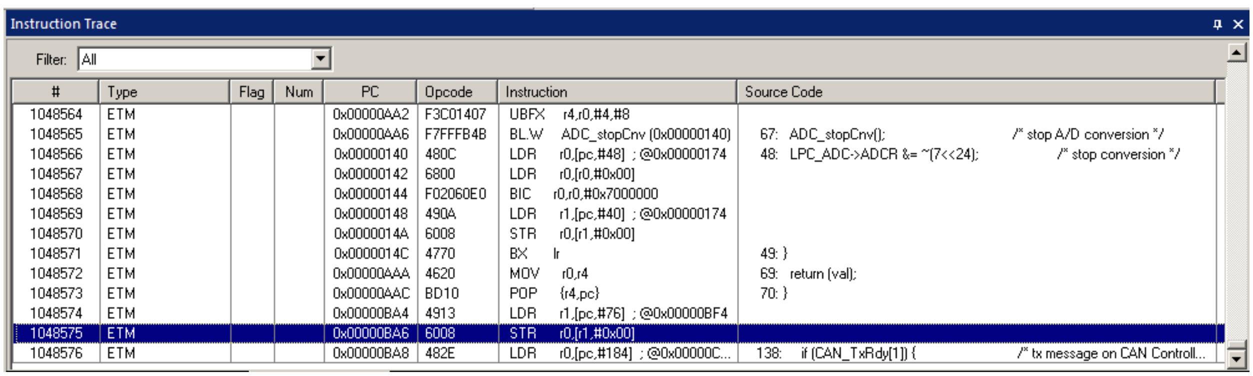


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		41					31109095280	311.09095280
Exception Exit		41					31109095547	311.09095547
Exception Return		0				X	31109097849	311.09097849
Data Write			1000001CH	00000049H	00000BA6H	X	31114997784	311.14997784
Exception Entry		41					31115008700	311.15008700
Exception Exit		41					31115008967	311.15008967
Exception Return		0				X	31115011369	311.15011369
Data Write			1000001CH	00000048H	00000BA6H	X	31120911216	311.20911216
Exception Entry		41					31120922320	311.20922320
Exception Exit		41					31120922587	311.20922587
Exception Return		0				X	31120924809	311.20924809
Data Write			1000001CH	00000046H	00000BA6H	X	31126824640	311.26824640
Exception Entry		41					31126835540	311.26835540
Exception Exit		41					31126835807	311.26835807
Exception Return		0				X	31126838169	311.26838169
Data Write			1000001CH	0000004AH	00000BA6H	X	31132738064	311.32738064
Exception Entry		41					31132748960	311.32748960
Exception Exit		41					31132749227	311.32749227
Exception Return		0				X	31132751609	311.32751609
Data Write			1000001CH	00000044H	00000BA6H	X	31138651544	311.38651544

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

TIP: The ULINK*Pro* has the source code and disassembly instructions in the new Trace window. Here is an example: The instruction causing the write is highlighted. Note there is a skid of one instruction and is normal for a Watchpoint.
If you click on one of these lines, you will be taken to the appropriate source and disassembly line.

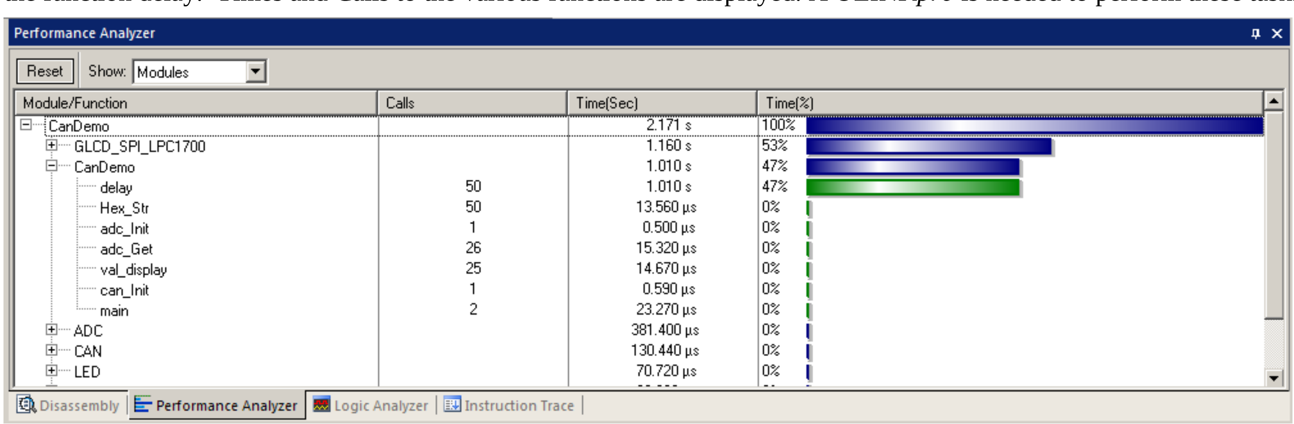


Instruction Trace

Filter: All

#	Type	Flag	Num	PC	Opcode	Instruction	Source Code
1048564	ETM			0x00000AA2	F3C01407	UBFX r4,r0,#4,#8	
1048565	ETM			0x00000AA6	F7FFFB4B	BL.W ADC_stopCnv (0x00000140)	67: ADC_stopCnv(); /* stop A/D conversion */
1048566	ETM			0x00000140	480C	LDR r0,[pc,#48] ; @0x00000174	48: LPC_ADC->ADCR &= ~(7<<24); /* stop conversion */
1048567	ETM			0x00000142	6800	LDR r0,[r0,#0x00]	
1048568	ETM			0x00000144	F02060E0	BIC r0,r0,#0x7000000	
1048569	ETM			0x00000148	490A	LDR r1,[pc,#40] ; @0x00000174	
1048570	ETM			0x0000014A	6008	STR r0,[r1,#0x00]	
1048571	ETM			0x0000014C	4770	BX lr	49: }
1048572	ETM			0x00000AAA	4620	MOV r0,r4	69: return (val);
1048573	ETM			0x00000AAC	BD10	POP {r4,pc}	70: }
1048574	ETM			0x00000BA4	4913	LDR r1,[pc,#76] ; @0x00000BF4	
1048575	ETM			0x00000BA6	6008	STR r0,[r1,#0x00]	
1048576	ETM			0x00000BA8	482E	LDR r0,[pc,#184] ; @0x00000C...	138: if (CAN_TxRdy[1]) { /* tx message on CAN Controll...

ULINK*pro* also provides Code Coverage Performance Analysis by using the ETM trace. In the performance Analysis window below, it shows 53% of its time writing to the LCD and 47% in CanDemo. Most of the time in CanDemo is spent in the function delay. Times and Calls to the various functions are displayed. A ULINK*pro* is needed to perform these tasks.



Performance Analyzer

Reset Show: Modules

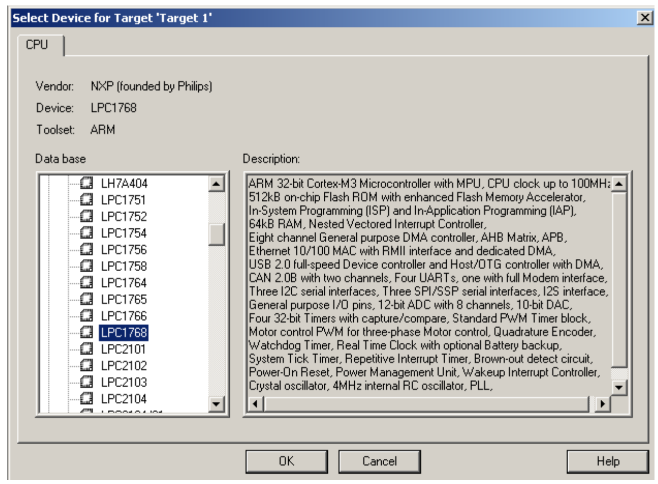
Module/Function	Calls	Time(Sec)	Time(%)
CanDemo		2.171 s	100%
GLCD_SPI_LPC1700		1.160 s	53%
CanDemo		1.010 s	47%
delay	50	1.010 s	47%
Hex_Str	50	13.560 µs	0%
adc_Init	1	0.500 µs	0%
adc_Get	26	15.320 µs	0%
val_display	25	14.670 µs	0%
can_Init	1	0.590 µs	0%
main	2	23.270 µs	0%
ADC		381.400 µs	0%
CAN		130.440 µs	0%
LED		70.720 µs	0%

11) Creating a new project: Using the Blinky source files: *optional exercise*

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.

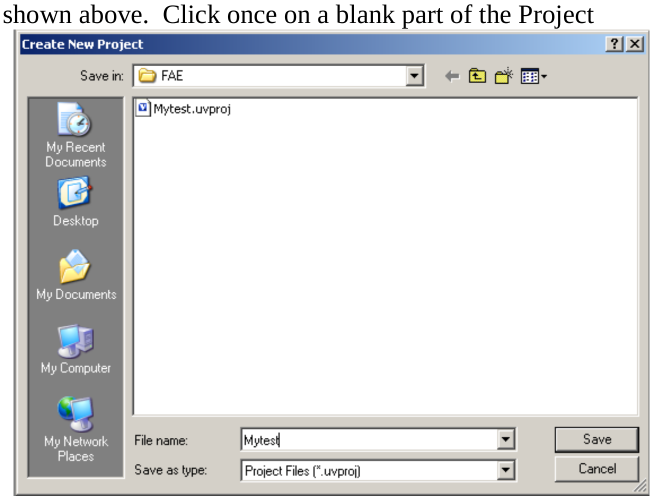
Create a new project called Mytest:

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 Options For Target 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.

12) 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:

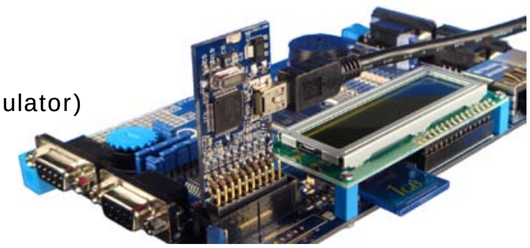
- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened.*
- Trace can often find nasty problems very quickly.
- Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time 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.

13) Keil Products:



Keil Microcontroller Development Kit (MDK-ARM™)

- MDK with included RTX RTOS – $4,895 (MDK has a great simulator)
- MDK-ARM-B: 256K code limit, no RTOS – $2,895

Keil Real Time Library (RL-ARM™)

- RTX sources, Flash File, TCP/IP, CAN, USB driver libraries - $4,195

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 - $1,395 – Cortex-M*x* SWV & ETM trace

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

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor.

For more information:

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

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

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

For the latest version of this document, contact the author, Keil Technical support or www.keil.com.

For Signum Systems: www.signum.com and Segger: www.segger.com.

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

KEIL™
Tools by ARM