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

LPC1768: Keil MCB1700 evaluation board

Fall 2009 Version 1.3 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 Keil MDK-ARM™ Microcontroller Development Kit featuring µVision®. MDK contains an excellent NXP simulator. We will not use the simulator in favor of Serial Wire Viewer (SWV) on the LPC1768. At the end of this tutorial, you will be able to confidently work with these packages and try the examples. The Keil MDK™ you will be using supports all NXP ARM processors including ETM support. Check the Keil Device Database® on www.keil.com/dd for NXP processor support.

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* or Signum JtagJetTrace.

We will be using µVision4 but you can use µVision3 if you prefer. The example project files are initially in µVision3 format and will be converted to µVision4 with the originals backed up. The two versions use different formats for the project files. There have been no major changes the way µVision works so there is no steep learning curve to endure.

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. Contact Keil sales for a temporary full version license if you require this to evaluate Keil MDK toolset. Keil also provides RL-ARM. This package includes the source files for RTX RTOS, TCP/IP stack, CAN drivers, Flash file system and USB drivers.

Software Installation:

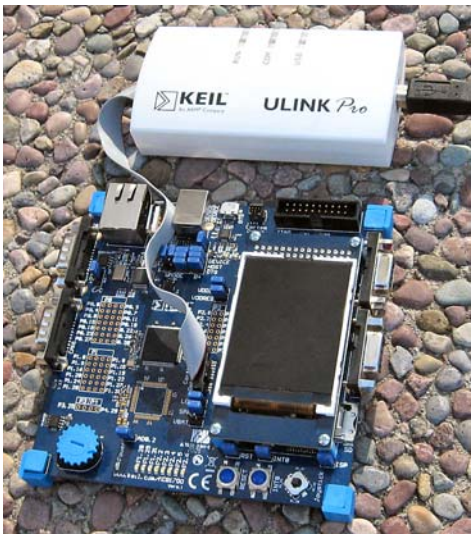
This document was written for Keil MDK 3.8 with the µVision4 add-on available from the Keil website. You can use either µVision 3 or µVision 4 but some of the menu locations have changed. In October 2009, MDK 4.0 will be released. This major update contains only µVision4. Do not confuse µVision4 with 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. Your previous µVision settings and files will then be preserved. For a clean install, erase your project directories.

The example files will compile without a license for MDK as they are all well within the 32K limit.

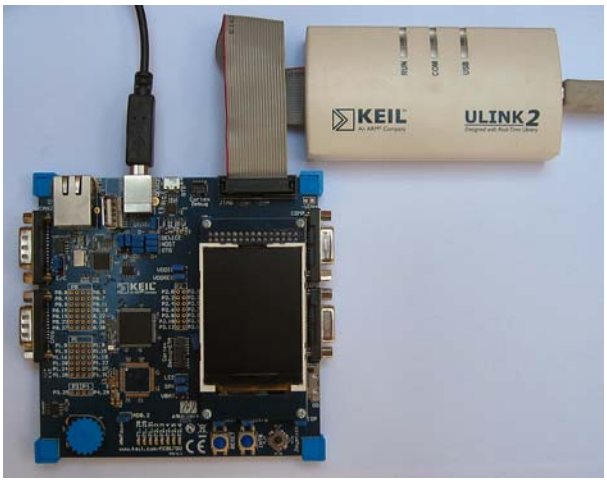
If you are using a Segger J-Link, you do not need to install any additional files. J-Link Version 6 and later support Serial Wire Viewer with MDK. Keil also supports the Segger J-Trace for ETM trace capabilities.

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.



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.Uv2.
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:
   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.

In this simple example the LEDs on the MCB1700 will now blink in succession.

8. When you have observed the program running click on Stop and exit debug mode for the next step.

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

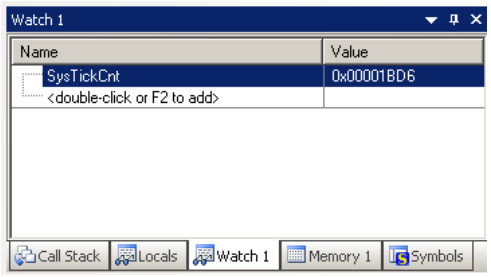
2) Watch and Memory Windows and how to use them:

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. Enter the Debug mode by clicking on the Debug icon. Click on the RUN icon.
2. In the source file Blinky.c is the global variable **`SysTickCnt`** near line 16.
3. Double click it 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 as shown below and using Ctrl-C and Ctrl-V 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.

4. **`SysTickCnt`** will be entered and immediately will be displayed and updated in real-time as shown here in this screen shot:



You can insert a number in a Watch or Memory window in real-time:

5. Double-click on the value field for SysTickCnt in the Watch window.
6. When it is highlighted, enter 0x0 or just 0 and press Enter.
7. SysTickCnt will be set to zero or any other number you enter.

How It Works:

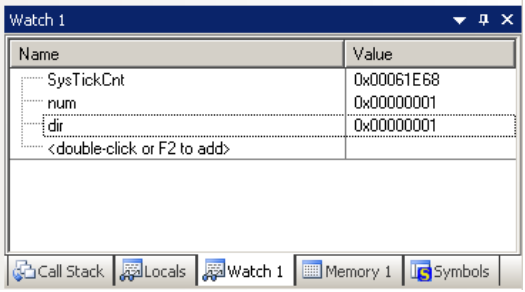
µVision uses CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. Remember the Cortex-M3 is a Harvard architecture. This means separate instruction and data buses. While the CPU is 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.

The only time this will be intrusive is in the unlikely event the program running on the CPU and µVision reads or writes to the same memory location at exactly the same. Then the CPU will be stalled for only one clock cycle. In practice, this cycle stealing never happens.

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

8. Stop the program by clicking on the Stop icon.
9. Enter the local variables **num** and **dir** from main() in Blinky.c to Watch 1.
10. Both are entered as "not in scope" because the PC is probably not in main().
11. Start the program by clicking on the Run icon.
12. **num** and **dir** change to ????????. Set a breakpoint on the second **Delay(500);** in main() around line 51 or 52. The program will soon stop on this hardware breakpoint.

    TIP: Remember you can set breakpoints on-the-fly in the Cortex-M3 !
13. **num** and **dir** display values as shown in the second Watch 1 window as shown here:
14. Each time you click RUN, these values are updated.



How to view these variables updated in real-time:

All you need to do is to make **num** and **dir** static !

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

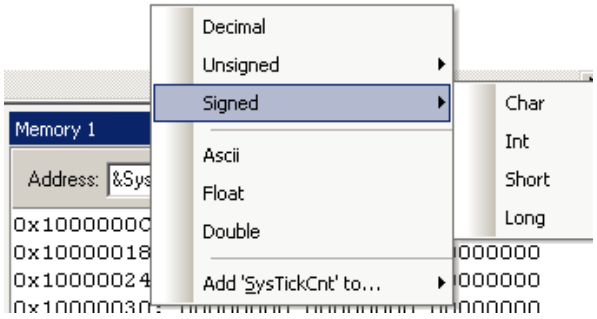
```
int main (void) {
        static int num = -1;
        static int dir =  1;
```

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 the variables in the Watch 1 window. Drag 'n Drop them in is the fastest way. (this has been fixed in the next release to make µVision remember the settings)
6. Remove the breakpoint you previously set and click on RUN. Can use Debug/Kill All Breakpoints to do this.
7. All three variables will now update in real-time. **TIP:** You can now drag these over while the program is running.

Memory window:

1. Drag 'n Drop **SysTickCnt** into the Memory window or enter in manually.
2. Note the value of **SysTickCnt** 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 (0x1000000C) and its data contents. Right click in the memory window to see the options as shown below.
4. Stop the CPU and exit debug mode for the next step.  and 



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

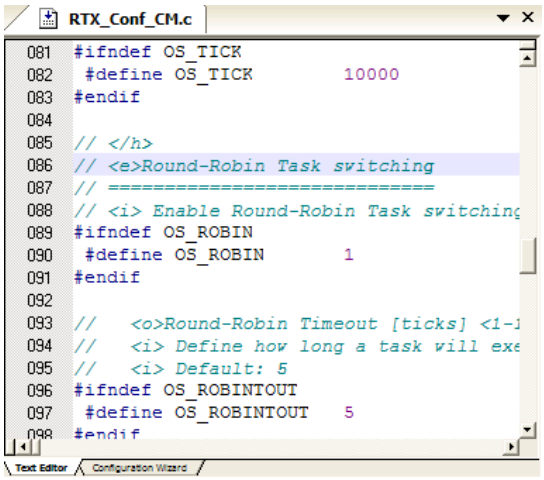
Keil provides RTX, a full feature RTOS. RTX is included for no charge as part of the Keil MDK full tool suite. It can have up to 255 tasks and no royalty payments are required. If source code is required, this is included in the Keil RL-ARM™ Real-Time Library which also includes USB, CAN, TCP/IP networking 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.

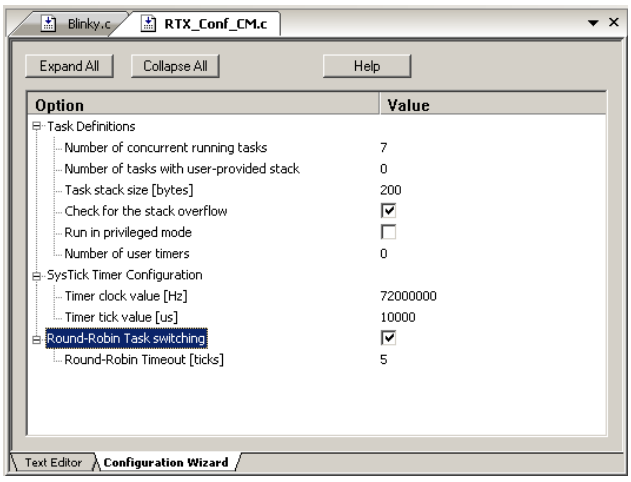
1. Start µVision4 by clicking on its icon on your Desktop if it is not already running.
2. Select Project/Open Project.
3. Open the file C:\Keil\ARM\Boards\Keil\MCB1700\RTX_Blinky\Blinky.Uv2.
4. Make sure "MCB1700" is selected in the Target window and not Simulator. MCB1700
   TIP: This is just the name 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.
8. Click on the RUN icon.
9. The LEDs will blink indicating the waveforms of a stepper motor driver. This will also be displayed on the LCD screen. Click on STOP .

Revisiting the Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left.
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

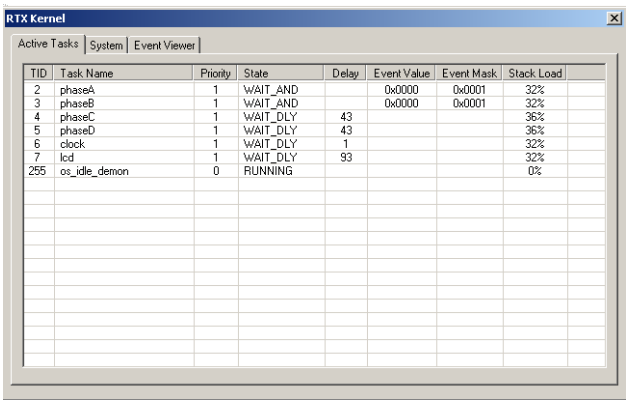


Configuration Wizard

TIP: µVision4 windows can be floated anywhere. You can restore them by selecting 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.

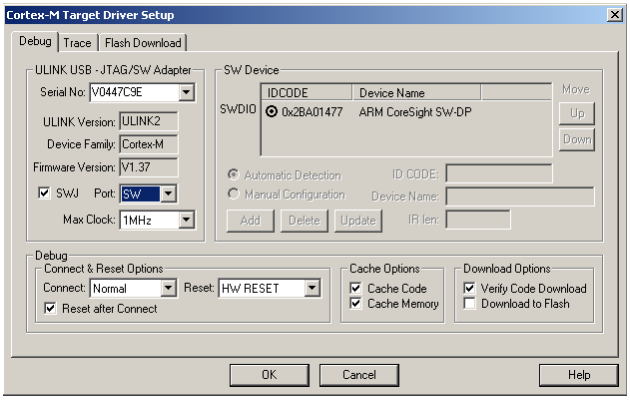


1. Run RTX_Blinky by clicking on the Run icon. You must be in Debug mode to do this.
2. Open Debug/OS Support and select RTX Kernel. Note these values are updated in real-time using the CoreSight technology as used in the Watch and Memory windows.
3. Click on the Event Viewer tab. There is probably no data.

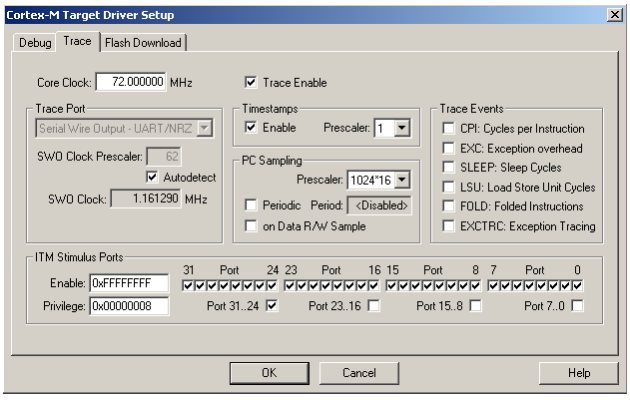
Event Viewer: Configuring the Serial Wire Viewer:

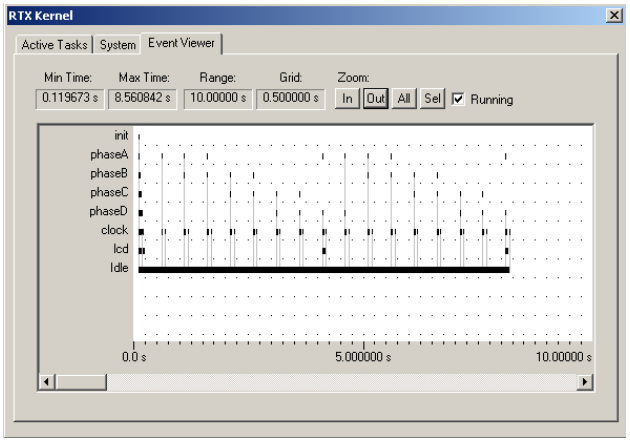
In order to get this working we have to activate the Serial Wire Viewer section of µVision.

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 72 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. Open Debug/OS Support and select RTX Kernel again.
11. Note the values are updated with the program running.
12. Click on the Event Viewer tab. The next window opens up.
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 Out or In button.





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. 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  *----------------------------
```

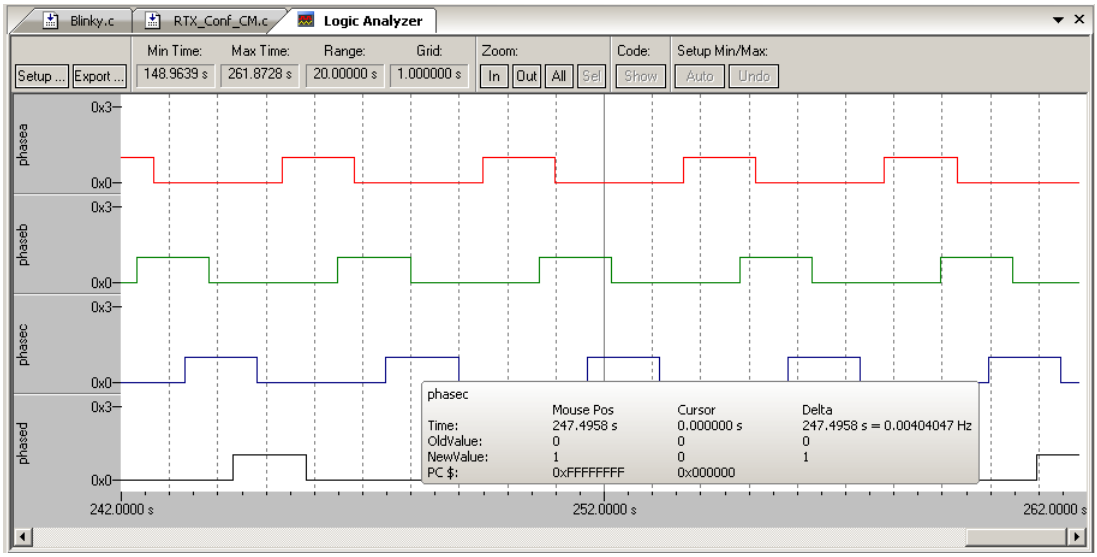
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them.

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.

```
075 /*----------------------------------------
076  *        Task 1 'phaseA': Phase A output
077  *----------------------------------------
078 __task void phaseA (void) {
079   for (;;) {
080     os_evt_wait_and (0x0001, 0xffff);     /*
081     LED_On (LED_A);
082     phasea = 1;
083     signal_func (t_phaseB);               /*
084     LED_Off(LED_A);
085     phasea = 0;
086   }
087 }
088
```

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 the Max. in the Display Range: area 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:

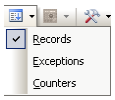
Data Reads and Writes:

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

Now we will examine some of the features available to you. SWV works with uVision3 or 4 and a ULINK2, ULINK-ME or a Segger J-Link V 6 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.

1. Use RTX_Blinky from the last exercise.
2. Enter Debug mode and run the program.
3. Select View/Trace/Records or click on the Trace icon and select Records.



4. The Trace Records window will open up as shown here:
5. 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, Port 0, EXCTRC and Periodic if set. **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 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.

Trace Records

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

What Is This ?

1. When variables are entered in the Logic Analyzer (remember phasea through phased ?), any reads and 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.

Trace Records

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

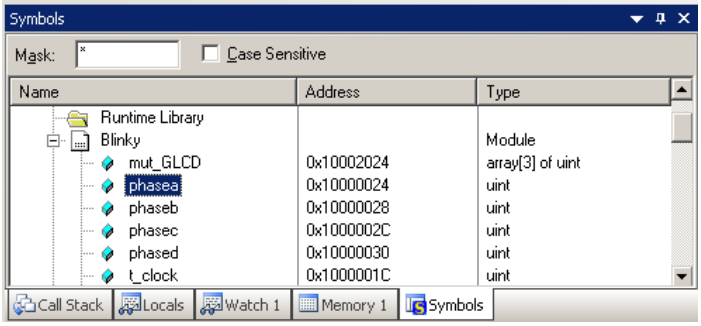
TIP: You can have up to four variables in the Logic Analyzer and subsequently displayed in the Trace Records window.

TIP: If you read from a variable – this will also be displayed.

TIP: If you select View/Symbol Window you can see where the addresses of the variables. This window is shown here displaying the addresses in Blinky.c.

TIP: The next version of uVision will display the source and assembly code in a new trace window.

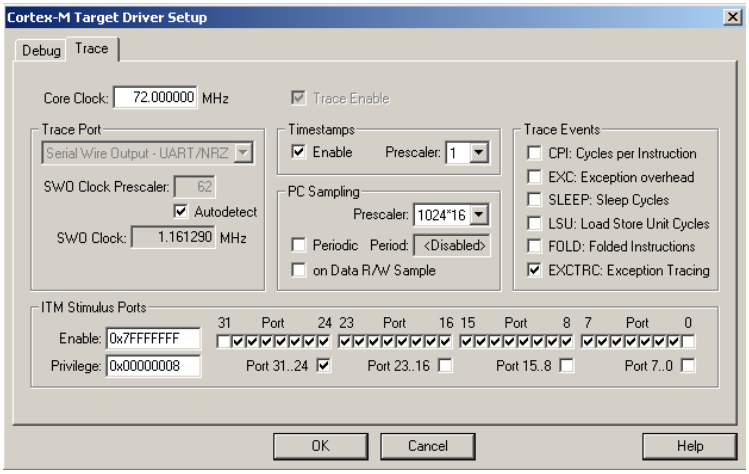
Note: You must have Browser Information selected in the Options for Target/Output tab for the Symbol Browser to function.



Exceptions and Interrupts:

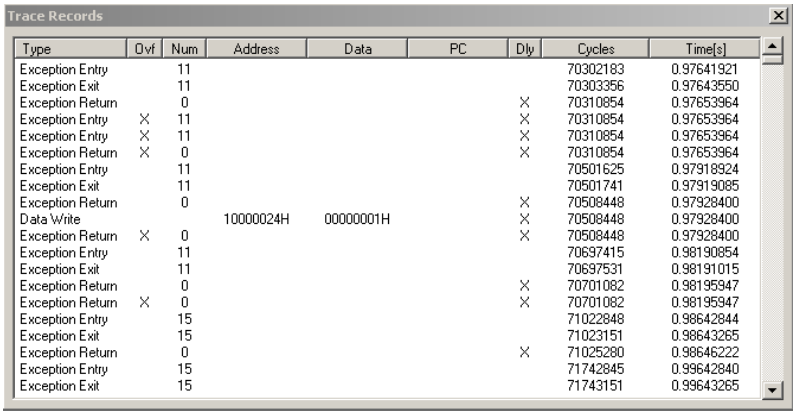
The LPC1700 family has many interrupts and it can be difficult to determine when they are being activated. SWV on the LPC1700 makes this easy.

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.



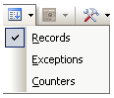
What Is This ?

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.

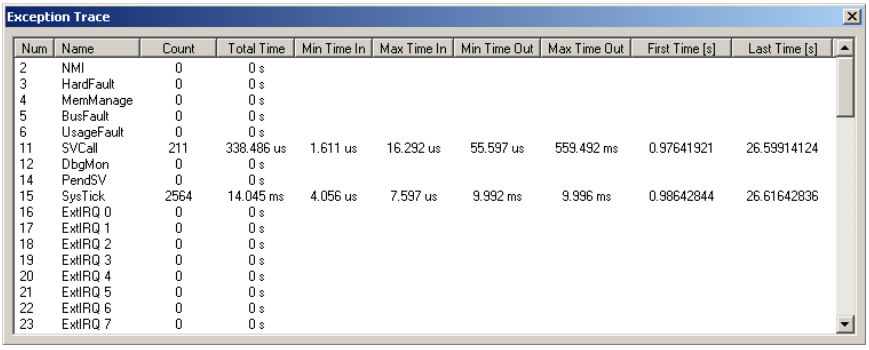


TIP: The SWO pin is one pin that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate.

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 listed as shown.
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 also clear this trace window by double-clicking on it.
6. All this information is displayed in real-time and without stealing CPU cycles !



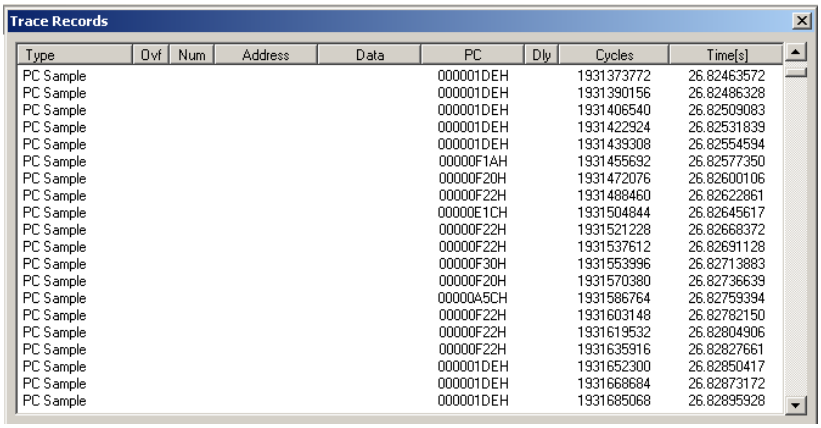
TIP: Num is the exception number: RESET is 1. External interrupts start at Num 16. For LPC1768, 41 is CAN IRQ. This is found in the LPC17xx Users Manual. Num 41 is also known as 41-16 = External IRQ 25.

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?".

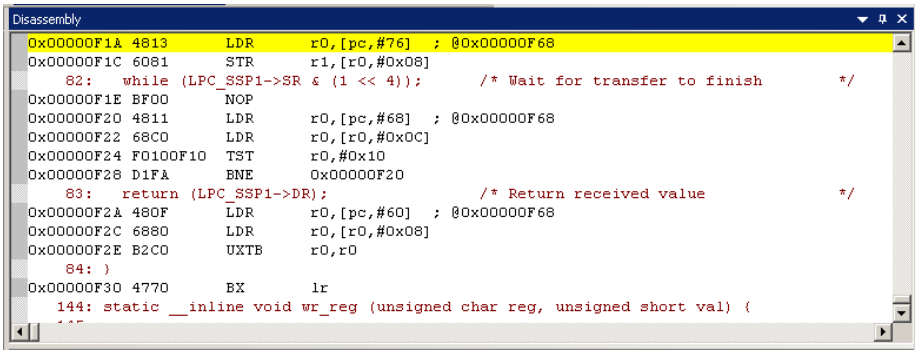
SWV can display at best every 64th instruction. It is better to keep this number as low as possible to avoid overloading the Serial Wire Output (SWO) pin.

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 01DE which is a branch to itself and is part of the RTOS.
7. Scroll down to see some other code as I did here: Clearly, the CPU spends most of its time in this tight loop.
8. Note some instructions near 0F1A. I opened the disassembly window to show that not all the PCs were captured. Between F1A and F20 in Trace Records, there is a space of 16,384 CPU cycles that are missed. Not all instructions execute in one cycle, but clearly many were missed.
9. Still, PC Samples can give you some idea of where your program is; especially if it is caught in a loop (like at 0x1DE).



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

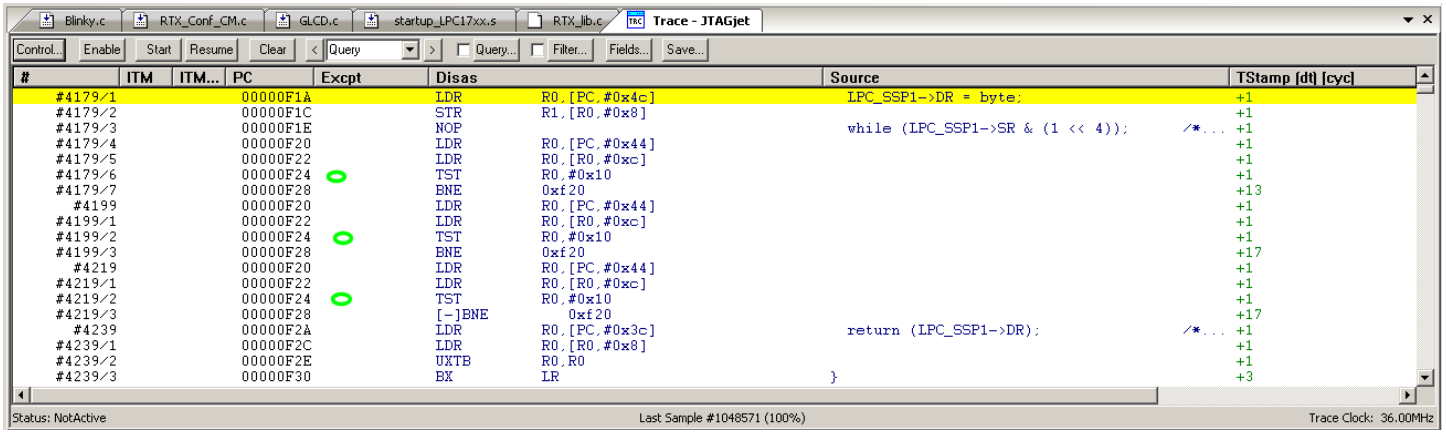
ETM provides all the program counter values and is especially useful for timing issues or where how a problem was caused disappears. ETM is a recording of all PC values.



In the screen below all the program counters starting at 0xF1A and ending at 0xF30 are recorded with ETM. Compare this to the disassembly listing above. Clearly the program flow is apparent: there is a green circle opposite every 0xF24.

Note the timestamp indicating the number of CPU cycles for each instruction. Note the disassembled instructions as well as the source lines.

You need a ULINK*Pro*, Segger J-Link or a Signum JtagJetTrace to view ETM signals. This window is from Signum.



8) ITM (Instruction Trace Macrocell)

Recall in Section 4) RTX Kernel Awareness on page 5 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.

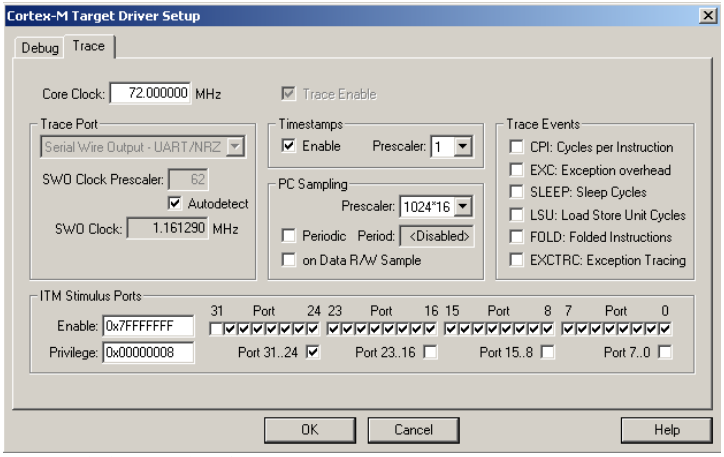
1. Add this code to Blinky.c. A good place is right after the place where you declared the four phase*x* variables.

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

2. In the task phaseA near line 90 enter these three lines:

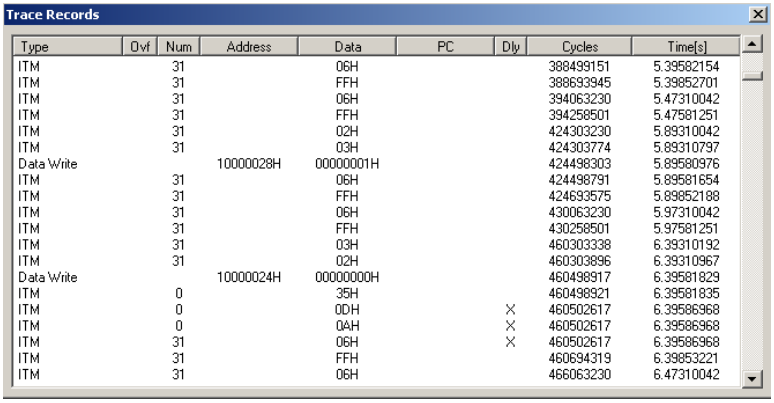
```
ITM_Port8(0) = 0x35;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```

3. If necessary stop the program execution, exit debug mode and rebuild the source files.
4. Program the Flash memory and enter debug mode.
5. Open Debug/Debug Settings and select the Trace tab. Select ITM Port 0, Unselect ITM Poprt 31, EXTRC and Periodic as shown:
6. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
7. In the printf viewer you will see the value "5" appear every few seconds.



Trace Records

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample and EXCTRC.
3. Select ITM 31.
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. You may have to scroll to see any ITM 0 frames.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		06H			388499151	5.39582154
ITM		31		FFH			388693945	5.39852701
ITM		31		06H			394063230	5.47310042
ITM		31		FFH			394258501	5.47581251
ITM		31		02H			424303230	5.89310042
ITM		31		03H			424303774	5.89310797
Data Write			10000028H	00000001H			424498303	5.89580976
ITM		31		06H			424498791	5.89581654
ITM		31		FFH			424693575	5.89852188
ITM		31		06H			430063230	5.97310042
ITM		31		FFH			430258501	5.97581251
ITM		31		03H			460303338	6.39310192
ITM		31		02H			460303896	6.39310967
Data Write			10000024H	00000000H			460498917	6.39581829
ITM		0		35H			460498921	6.39581835
ITM		0		0DH		X	460502617	6.39586968
ITM		0		0AH		X	460502617	6.39586968
ITM		31		06H		X	460502617	6.39586968
ITM		31		FFH			460694319	6.39853221
ITM		31		06H			466063230	6.47310042

What Is This ?

ITM 31 frames are from the RTX Kernel Awareness window.

Data Write frames are the writes to phasea through phased. These are here because they are entered 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.

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. 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.

9) Watchpoints:

LPC1700 processors have 8 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. Usually the debugger 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.

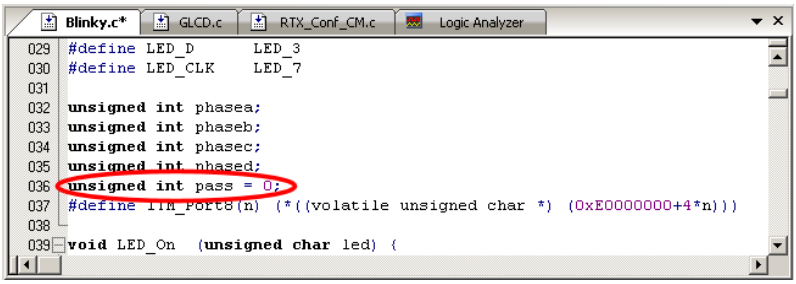
1. Stop the program and leave Debug mode.
2. Add this line in Blinky.c in the area where you declared phasea. This means we want this to be a global variable.

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

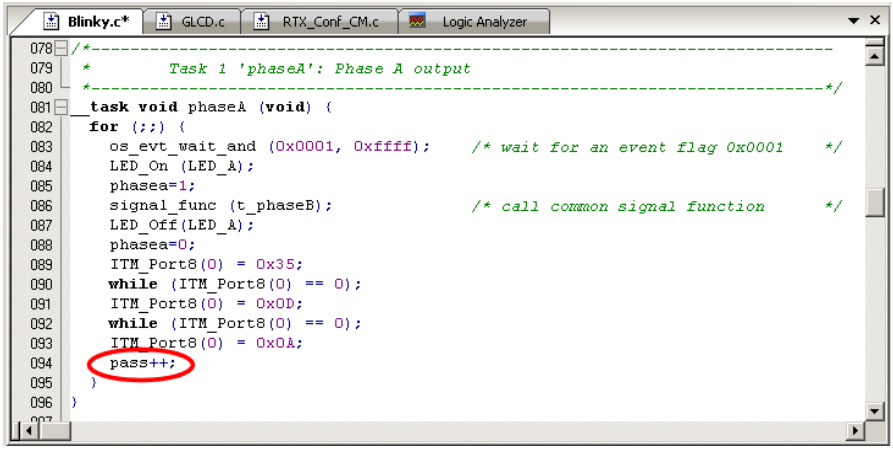
3. In task1 near where you entered the ITM write code, enter this line:

```
pass++;
```

4. Your result should look similar to the two segments displayed below.
5. Compile the project and program the Flash.
6. Enter Debug mode.
7. Remove the variables in the Logic Analyzer window by clicking on "Setup" and selecting the "Kill All" button.
8. Click on Close to return.
9. Select the Debug tab and select Breakpoints or press Ctrl-B.



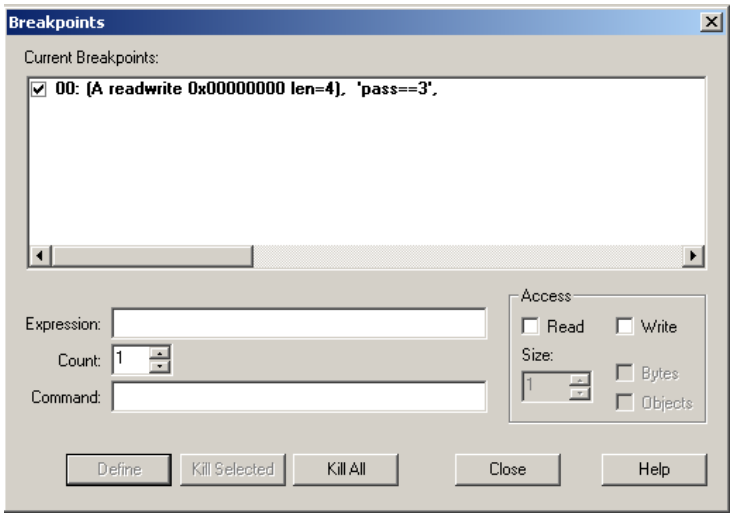
10. In the Expression box enter: pass==3. Select both Read and Write boxes.
11. Click on Define and your expression will be accepted as shown below:
12. Click on Close.
13. Enter the variable pass to the Watch window by dragging and dropping it or enter manually.
14. Click on RUN.
15. When pass equals 3, the program will stop.
16. That is how a Watchpoint works.
17. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints box.



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



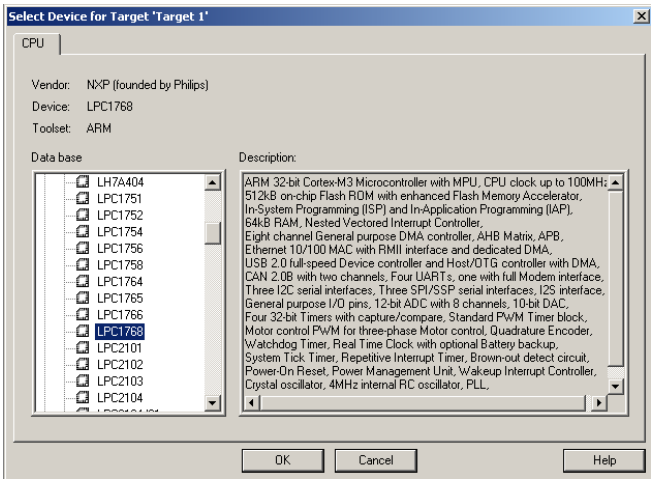
10) Creating a new project: Using the Blinky source files: *optional*

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.

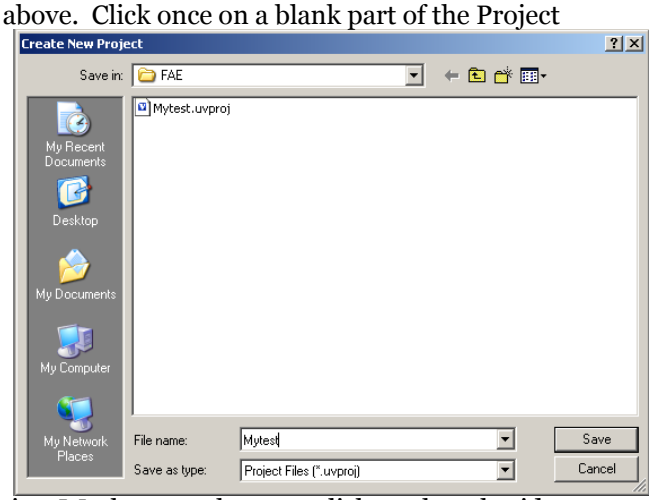
Create a new project called Mytest:

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.

11) CAN: Controller Area Network

For exercises using CAN for the LPC2300 and LPC1700 series please obtain the CAN Primer:
http://www.standardics.nxp.com/support/documents/microcontrollers/?search=CAN&type=article

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:

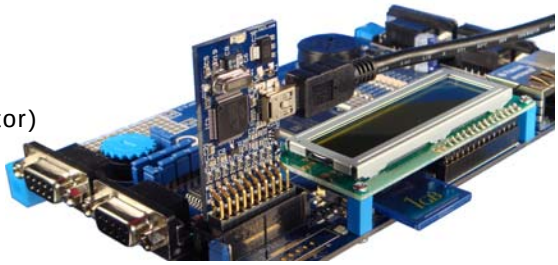
- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened.*
- Trace can often find nasty problems very quickly.
- Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time 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-M3 SWV & ETM trace (November 2009)

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

14) Keil Supports these NXP Processors:

Please visit www.keil.com/dd for the latest list. For not yet released NXP products, contact Keil Technical Support or Sales.

ARM7 / ARM9 / Cortex Family
LH75400, LH75401, LH75410, LH75411, LH79520, LH79524, LH79525, LH7A400, LH7A404, LPC1751, LPC1752, LPC1754, LPC1756, LPC1758, LPC1764, LPC1765, LPC1766, LPC1768, LPC2101, LPC2102, LPC2103, LPC2104, LPC2104/01, LPC2105, LPC2105/01, LPC2106, LPC2106/01, LPC2109, LPC2109/01, LPC2114, LPC2114/01, LPC2119, LPC2119/01, LPC2124, LPC2124/01, LPC2129, LPC2129/01, LPC2131, LPC2131/01, LPC2132, LPC2132/01, LPC2134, LPC2134/01, LPC2136, LPC2136/01, LPC2138, LPC2138/01, LPC2141, LPC2142, LPC2144, LPC2146, LPC2148, LPC2194, LPC2194/01, LPC2210, LPC2210/01, LPC2212, LPC2212/01, LPC2214, LPC2214/01, LPC2220, LPC2290, LPC2290/01, LPC2292, LPC2292/01, LPC2294, LPC2294/01, LPC2364, LPC2365, LPC2366, LPC2367, LPC2368, LPC2377, LPC2378, LPC2387, LPC2388, LPC2420, LPC2458, LPC2460, LPC2468, LPC2470, LPC2478, LPC2880, LPC2888, LPC2917, LPC2917/01, LPC2919, LPC2919/01, LPC2921, LPC2923, LPC2925, LPC2927, LPC2929, LPC2930, LPC2939, LPC3130, LPC3131, LPC3180, LPC3220, LPC3230, LPC3240, LPC3250, SJA2010

Smart Card Family
MIFARE PRO X, SmartMX, WE Family

8051 Family
80/87C51, 80/87C52, 80C31, 80C31X2, 80C32, 80C32X2, 80C451, 80C51FA, 80C51RA+, 80C528, 80C550, 80C552, 80C554, 80C575, 80C652, 83/87C451, 83/87C524, 83/87C528, 83/87C550, 83/87C552, 83/87C554, 83/87C575, 83/87C652, 83/87C654, 83/87C750, 83/87C751, 83/87C752, 8XC51FA/8xL51FA, 8XC51FB/8xL51FB, 8xC51FC/8xL51FC, 8xC51MA2, 8xC51MB2, 8xC51MB2/02, 8xC51MC2, 8xC51MC2/02, 8xC51RA+, 8xC51RB+, 8xC51RC+, 8xC51RD+, 8XC52, 8XC54, 8XC58, P80/P87C51X2, P80/P87C52X2, P80/P87C54X2, P80/P87C58X2, P80C557E4, P80C557E6, P80C557E8, P80C562, P80C591, P80C592, P80CE558, P80CE560, P80CE598, P80CL31, P80CL410, P80CL51, P80CL580, P83/87C654X2, P83/87C660X2, P83/87C661X2, P83/P87C557E8, P83/P87CE560, P83/P89C557E4, P83/P89CE558, P83C557E6, P83C562, P83C591, P83C592, P83CE598, P83CL410, P83CL580, P87C51RA2, P87C51RB2, P87C51RC2, P87C51RD2, P87C591, P87CL52X2, P87CL54X2, P87CL888, P87LPC759, P87LPC760, P87LPC761, P87LPC762, P87LPC764, P87LPC767, P87LPC768, P87LPC769, P87LPC778, P89C51RA2xx, P89C51RB2Hxx, P89C51RB2xx, P89C51RC2Hxx, P89C51RC2xx, P89C51RD2Hxx, P89C51RD2xx, P89C51X2, P89C52X2, P89C54X2, P89C58X2, P89C60X2, P89C61X2, P89C660, P89C662, P89C664, P89C668, P89C669, P89C738, P89C739, P89LPC901, P89LPC902, P89LPC903, P89LPC904, P89LPC906, P89LPC907, P89LPC908, P89LPC9102, P89LPC9103, P89LPC9107, P89LPC912, P89LPC913, P89LPC914, P89LPC915, P89LPC916, P89LPC917, P89LPC920, P89LPC9201, P89LPC921, P89LPC9211, P89LPC922, P89LPC9221, P89LPC922A1, P89LPC924, P89LPC9241, P89LPC925, P89LPC9251, P89LPC930, P89LPC9301, P89LPC931, P89LPC9311, P89LPC931A1, P89LPC932, P89LPC9321, P89LPC932A1, P89LPC933, P89LPC9331, P89LPC934, P89LPC9341, P89LPC935, P89LPC9351, P89LPC936, P89LPC9361, P89LPC938, P89LPC9401, P89LPC9402, P89LPC9408, P89LPC952, P89LPC954, P89LV51RB2, P89LV51RC2, P89LV51RD2, P89V51RB2, P89V51RC2, P89V51RD2, P89V52X2, P89V660, P89V662, P89V664, PCD6001, PCD6002, SAA5645HL, SAA5647HL, SAA5665HL, SAA5667HL, TDA8006, TDA8008, TDA8028, TDA8029

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.