NXP LPC1700 Lab *for the Keil MCB1768/58*

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

For the Keil MCB1700 Evaluation Board Version 4.7

by Robert Boys bob.boys@arm.com

For the Keil MCB1700 Evaluation Board

Introduction:

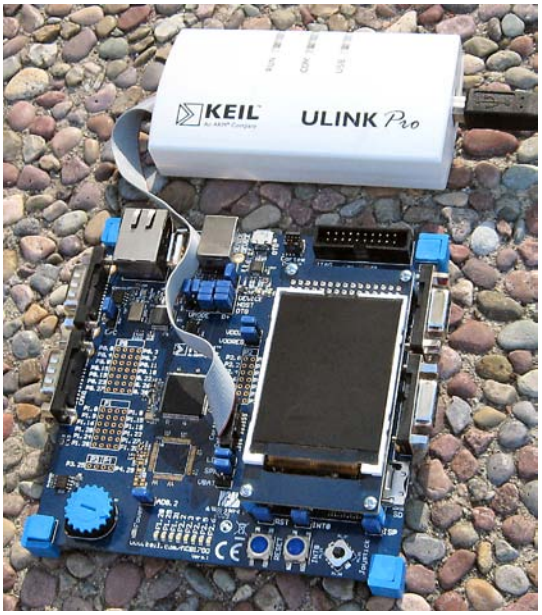
The purpose of this lab is to introduce you to the NXP Cortex™-M3 processor using the Keil MDK-ARM™ Microcontroller Development Kit featuring µVision®. MDK also contains a simulator. We will use the Serial Wire Viewer (SWV) on the LPC1768 or LPC1765 rather than the simulator in this lab. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. The Keil MDK you will be using supports all NXP ARM processors including Serial Wire Viewer and ETM trace. Check the Keil Device Database® on www.keil.com/dd for the complete list of NXP support.

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

Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler, Assembler and Linker. MDK is a turn-key product with included examples and is easy to get running. Keil supports Eclipse and GCC.
2. Serial Wire Viewer and ETM trace capability is included. A full feature Keil RTOS called RTX is included with MDK and includes source code with all versions of MDK. RTX now comes free with a BSD type license. Source code is provided.
3. A RTX Kernel Awareness window is updated in real-time. Kernel Awareness exists for Keil RTX, CMX, Quadros and Micrium. MDK can compile all RTOSs written in C.
4. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™
5. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.



This document details these features (plus more):

1. Serial Wire Viewer (SWV) with ULINK2, ULINK-ME and ULINK*pro*. ETM Trace using ULIN*Kpro*.
2. Real-time Read and Write to memory locations for Watch, Memory and RTX Tasks windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also called Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS that updates while the program is running.

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 most Cortex processors. SWV is output on the Serial Wire Output (SWO) pin found on the JTAG/SWD adapter connector.

SWV does not steal any CPU cycles and is completely non-intrusive except for ITM Debug printf Viewer. SWV is provided by the Keil ULINK2, ULINK-ME, ULINK*pro* and the Segger J-Link and J-Link Ultra. Best results are with a ULINK family adapter. ULINK*pro* provides faster Flash programming performance and ETM trace as described below.

Embedded Trace Macrocell (ETM):

ETM adds all the program counter values to the data provided by SWV. This allows advanced debugging features including timing of areas of code (Execution Profiling), Code Coverage, Performance Analysis and program flow debugging and analysis. ETM support requires the ULINK*pro*. This document uses a ULINK*pro* for ETM. A ULINK2 or ULINK-ME is used for the Serial Wire Viewer exercises in this lab. A ULINK*pro* or a Segger J-Link can easily be substituted.

Software Installation:

This document was written for Keil MDK 4.53 which contains µVision4. MDK is available on the Keil website and the specific example files are included with this document. Example files are subject to improvement and can change. Use the files specified for this document. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence.

If you have a previous version of MDK 4, do not uninstall it; just install the new version on top. For a clean install, erase your project directories.

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

If you are using a Segger J-Link, you do not need to install any additional files. You will need to configure µVision to use the J-Link to run programs and program the Flash memory. This is easy to do. J-Link Version 6 and later supports Serial Wire Viewer with Keil µVision.

JTAG and SWD Definitions: It is useful to have an understanding of these terms:

JTAG: JTAG provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.

SWD: Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except no Boundary Scan. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup.

SWV: Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.

SWO: Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.

Trace Port: A 4 bit port that ULINK*pro* uses to output ETM frames and optionally SWV (rather than the SWO pin).

ETM: Embedded Trace Macrocell: Provides all the program counter values. Only the ULINK*pro* works with ETM. The trace frames come out the 4 bit Trace Port pins as found on the compact 20 pin connector.

MCB1700 debug adapter connectors:

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

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

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

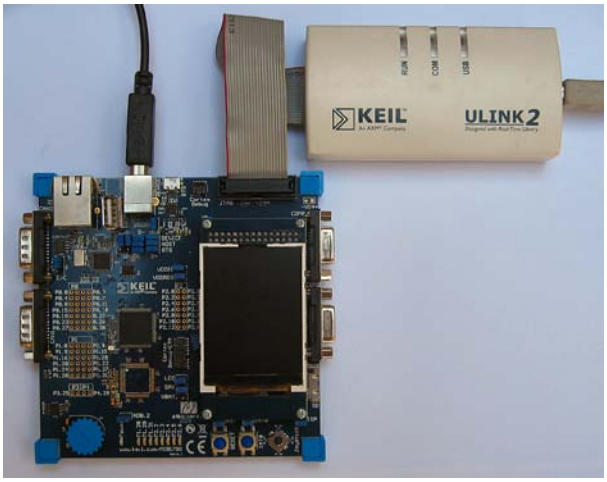
You can use any of these three connectors for the examples in his document.

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.Uv2.
3. Make sure "LPC1768 Flash" is selected.

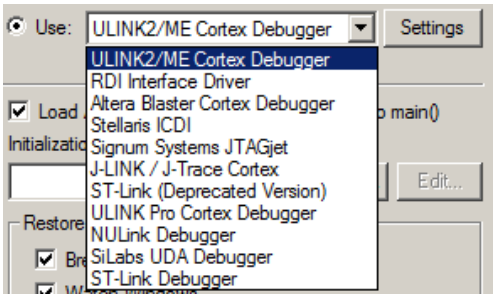
This is where you select the Simulator or to execute a program in RAM or Flash.

4. Select Options For Target icon . The USB adapter is selected in this window as shown here: ULINK2/ME will be pre-selected in this example project.

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

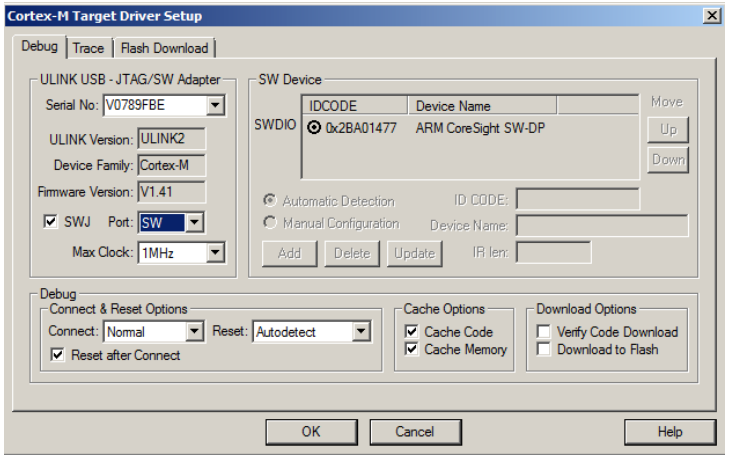
5. Select the Debug tab. The bottom window below opens up.
6. In **Port:** select SWJ and SW. You can use JTAG if you are not going to use Serial Wire Viewer (SWV).
7. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is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

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



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

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



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

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

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

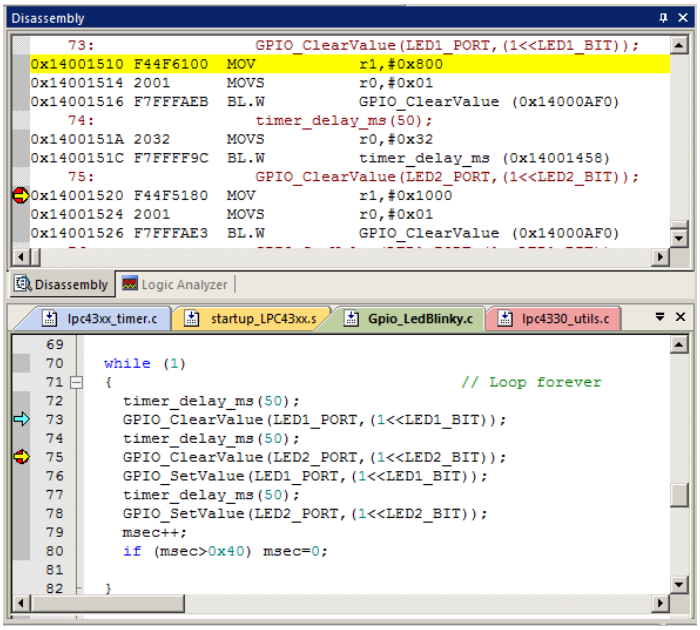
2) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block somewhere appropriate in the while(1) loop between Lines 062 through 088 in the source file Blinky.c as shown below: Click on the Blinky.c tab if not visible.
2. A red circle is created and soon the program will stop at this point as shown below.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows. This instruction has not been executed yet.
4. The cyan arrow is a mouse selected pointer and is associated with the yellow band in the disassembly window. Click on a line in one window and this place will be indicated in the other window.
5. Note you can set and unset hardware breakpoints while the program is running. ARM CoreSight debugging technology does this. There is no need to stop the program for many other CoreSight features.
6. The LPC1700 family has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to.

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

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

TIP: Earlier versions of µVision use a double-click to set/unset breakpoints and create a red square box.



3) Call Stack + Locals Window:

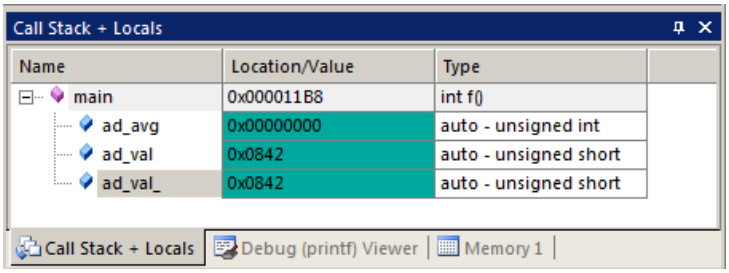
Local Variables:

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

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

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

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



Call Stack:

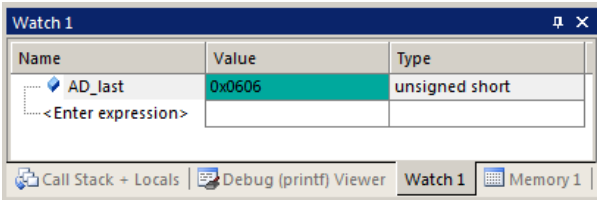
The list of stacked functions is displayed when the program is stopped. This is when you need to know which functions have been called and are stored on the stack.

6. **Remove all hardware breakpoints by clicking on its red circle !** There are various options in the Debug selection when in debug mode to manage breakpoints such as Kill All Breakpoints.

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

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

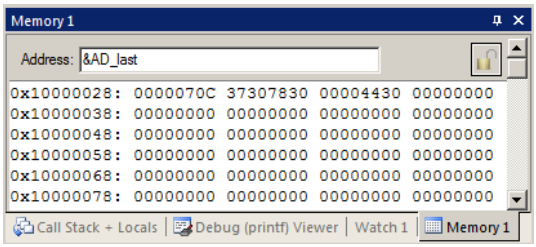
1. In the source file ADC.c is the global variable **`AD_last`** near line 21. Select File/Open to access ADC.c if needed.
2. Enter debug mode.  The program can be running or stopped for the following steps.
3. Open a Watch window by selecting View/Watch Windows/Watch 1 in the main µVision window.
4. Block AD_last and drag and drop into Watch 1.
5. You can also right-click on it and select Add AD_last to.. into Watch 1. You can click on F2 to type it in manually.
6. AD_last will display as shown here:
7. Click on RUN if necessary.
8. Rotate the pot and **AD_last** is updated in real-time.



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

Memory window:

1. Drag 'n Drop **`AD_last`** into the Memory window or enter it manually. You can also right-click on AD_last and select Add AD_last to … Rotate the pot and watch the memory window.
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 of **`AD_last`** is shown (0x10000028 in this case).
4. Right click in the memory window and select Unsigned/Long.
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 !

7. Stop the CPU  and exit debug mode. 
8. In the source file Blinky.c add a global variable counter near line 30 like this: **`unsigned int counter = 0;`**
9. In the main function add the line **`counter++;`** just after the printf statement near line 89.
10. Compile the source files by clicking on the Build icon.  .
11. Program the flash by clicking on the Load icon:  and enter Debug mode.  Click on the RUN icon. 
12. Enter the variable **`counter`** in the Watch 1 window by your preferred method. Note it increments every second.
13. Double-click on the value field for **`counter`** in the Watch window.
14. When it is highlighted, enter 0x0 or just 0 and press Enter.
15. **`counter`** will be set to zero or to any other number you entered. You can also do this in the memory window by right-clicking on the memory data and select Modify Memory at 0x….

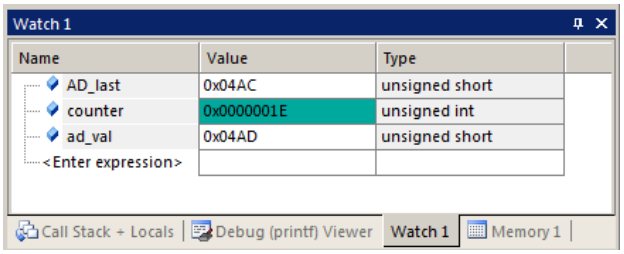
Note: In Keil MDK 4.53 modifying a Watch window can be difficult under certain circumstances. This is to be fixed. Modifying the memory window works as described.

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

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

10. Stop the program. Enter the local variable **ad_value** from main() in Blinky.c near line 37 to the Watch 1 window.
11. It will probably have a value displayed as the program spends nearly all its time in main() so it is in scope. If the PC is outside of main(), <out of scope> or <cannot evaluate> will be displayed.
12. Start the program by clicking on the Run icon.
13. Set a breakpoint by double-clicking in the margin beside the line **clock_1s = 0;** in main() around line 86 The program will soon stop on this hardware breakpoint.

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



14. **ad_value** displays the value as shown here:
15. 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) {
        uint32_t ad_avg = 0;
        static uint16_t ad_val = 0, ad_val_ = 0xFFFF;
```

2. Exit debug mode.

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

3. Compile the source files by clicking on the Build icon or press F7. Hopefully they compile with no errors or warnings.
4. To program the Flash click on the Load icon. 

. A progress bar will be at the bottom left.

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

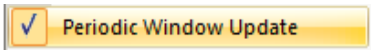
5. Enter Debug mode. 
6. Remove the breakpoint you previously set and click on RUN. You can use Debug/Kill All Breakpoints to do this.
7. **ad_value** is now updated in real-time.
8. Stop the CPU and exit debug mode for the next step. Click on  and then 

How It Works:

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

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

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

In the main menu select View/Periodic Window Update: 

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

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

Users often want to know the current operating task number and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides two Task Aware windows for RTX by accessing this information. Other RTOS companies also provide awareness plug-ins for µVision. Any RTOS ported to a Cortex-M or R processor will compile with MDK. See www.keil.com/rl-arm/kernel.asp for complete RTX details.

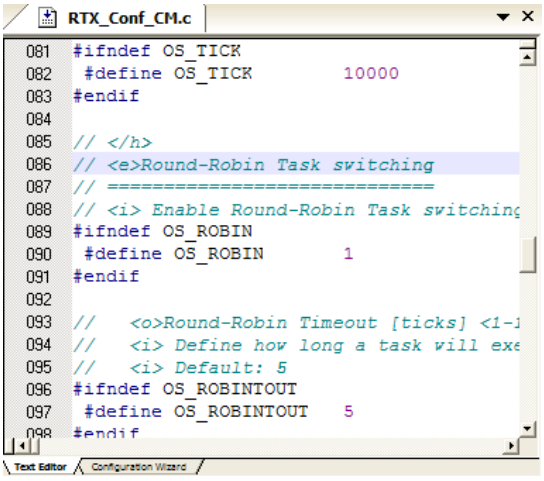
RTX is a Keil produced RTOS that is provided with MDK. Source code is provided with all versions of MDK.

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

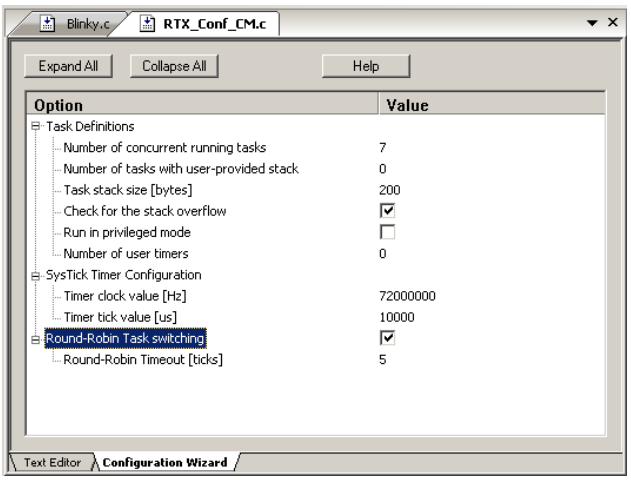
1. Start µVision4 by clicking on its icon on your Desktop if it is not already running.
2. Select Project/Open Project and open C:\Keil\ARM\Boards\Keil\MCB1700\RTX_Blinky\Blinky.Uv2.
3. Compile the source files by clicking on the Build icon. . They will compile with no errors or warnings.
4. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
5. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
6. The LEDs will blink indicating the waveforms of a stepper motor driver. This will also be displayed on the LCD screen. Click on STOP .

The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open or double-click on it in the Project window if you are not in Debug mode.
2. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. This is a great feature as it is much easier changing items here than in the source code.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>.
8. The µVision4 System Viewer windows used to display the peripherals are created in a similar fashion.



Text Editor

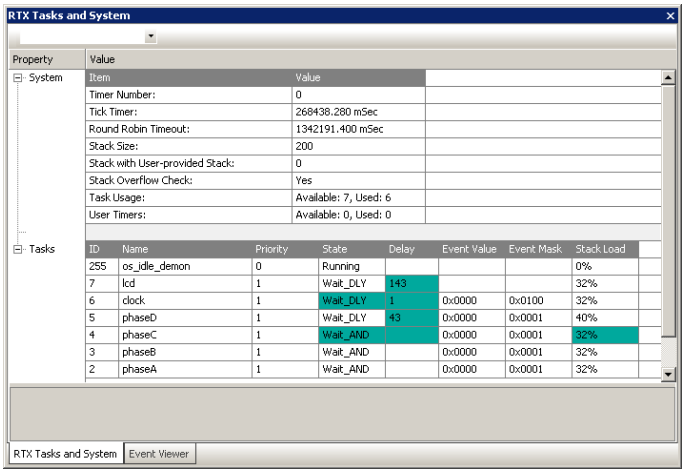


Configuration Wizard

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

7) 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 Aware window for RTX. Other RTOS companies also provide awareness for µVision.



1. Click on the RUN icon to run RTX_Blinky.
2. Open Debug/OS Support and select RTX Tasks and the window on the right opens up. RTOS visibility is updated in real-time using CoreSight technology as used in the Watch and Memory windows.

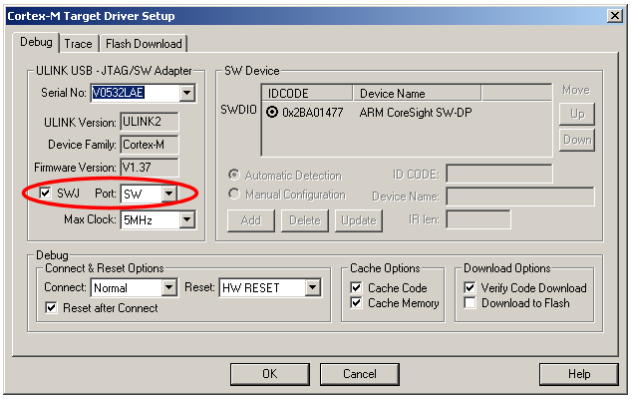
TIP: View/Periodic Window Update must be selected for the RTX Task to be updated. The Serial Wire Viewer must be configured for the Event Viewer to be updated.

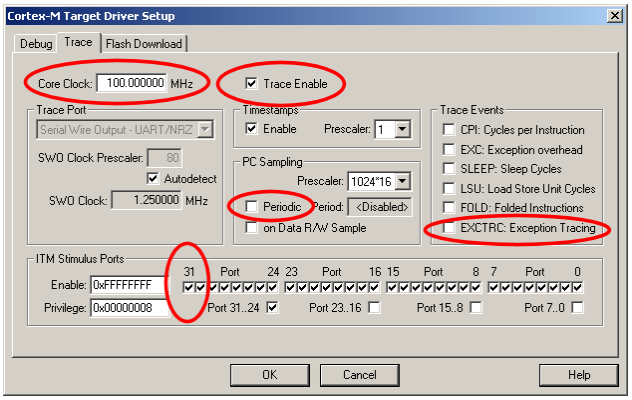
3. Open Debug/OS Support and click on Event Viewer. There is probably no data visible because... SWV is not configured yet.

Configuring the Serial Wire Viewer (SWV):

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

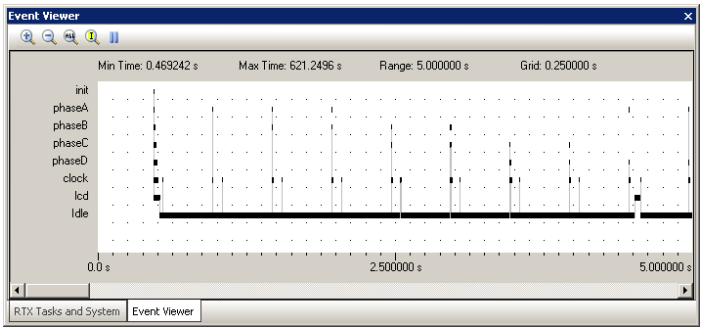
1. Stop the CPU and exit debug mode.
2. Click on the Options icon.
3. Select the Debug tab and then click the Settings box next to ULINK Cortex Debugger dialog.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW. Max Clock can be 1 or 5 MHz.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 100 MHz and select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown here. ITM Stimulus Port 31 must be checked.
8. Click on OK twice to return to µVision. The Serial Wire Viewer is now configured in µVision.
9. Enter Debug mode and click on Run to start the program.
10. Open Debug/OS Support and select RTX Tasks and System.
11. Note the values are updated with the program running.
12. 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 or the ALL icon.





TIP: To find the CPU 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 base frequency.

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



8) 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.

Add the eight source lines to the four tasks:

1. Stop the program and exit debug mode.
2. Add 4 global variables **`unsigned int phasea`** through **`unsigned int phased`** to Blinky.c as shown here:
3. Add 2 lines to each of the four tasks Task1 through Task4 in Blinky.c as shown below: **phasea=1;** and **phasea=0;** :the first two lines are shown added at lines 081 and 084 (just after LED_On and LED_Off function calls. For each task, add the corresponding variable assignment statements phasea, phaseb, phasec and phased.
4. We do this because in this example program there are not enough global or static variables to connect to the Logic Analyzer.

```
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: make them static. To see peripheral registers values, read or write to them.

5. Rebuild the project.  Program the Flash  and enter debug mode  .
6. 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. 

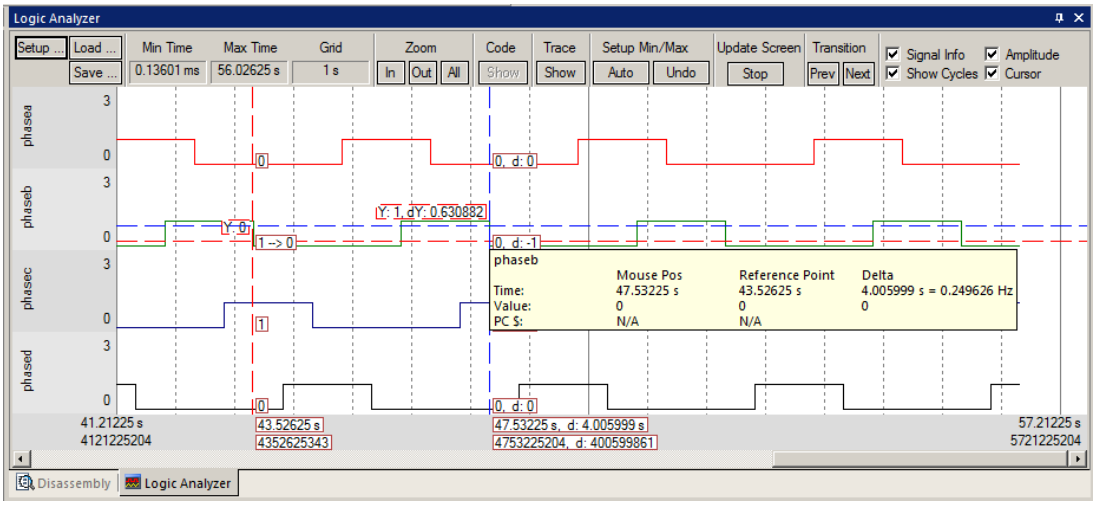
```
074 /*-----------------------------------------
075  *         Task 1 'phaseA': Phase A output
076  *-----------------------------------------
077 __task void phaseA (void) {
078   for (;;) {
079     os_evt_wait_and (0x0001, 0xffff);     /*
080     LED_On (LED_A);
081     phasea=1;
082     signal_func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

Enter the Variables into the Logic Analyzer:

8. Click on the Blinky.c tab. Block **phasea**, click, hold and drag 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.

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

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 1 second or so. 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:
15. Select Signal Info, Amplitude, Show Cycles and Cursor. Move the cursor to see the information displayed.



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

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

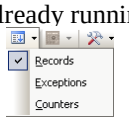
Data Reads and Writes: **Note:** the current version of MDK (4.53) displays only data writes and no data reads.

You need to have configured Serial Wire Viewer (SWV) in xyzxyz under **Event Viewer: Configuring the Serial Wire Viewer:**

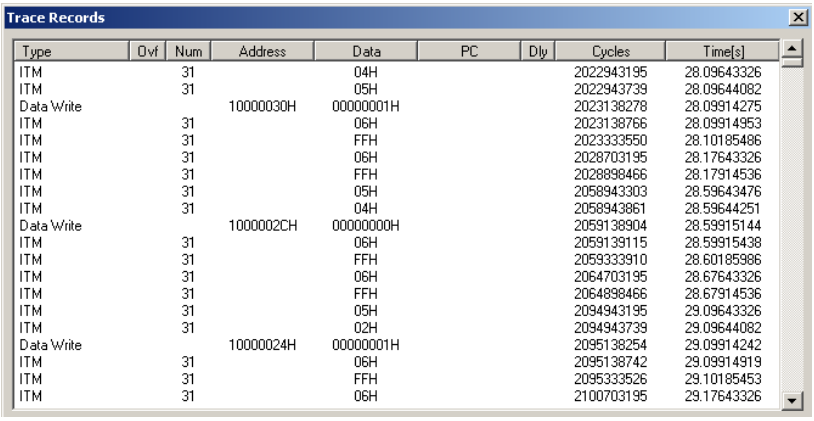
Wwe will examine some of the features available to you. SWV works with uVision and a ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no additional equipment or software need be purchased.

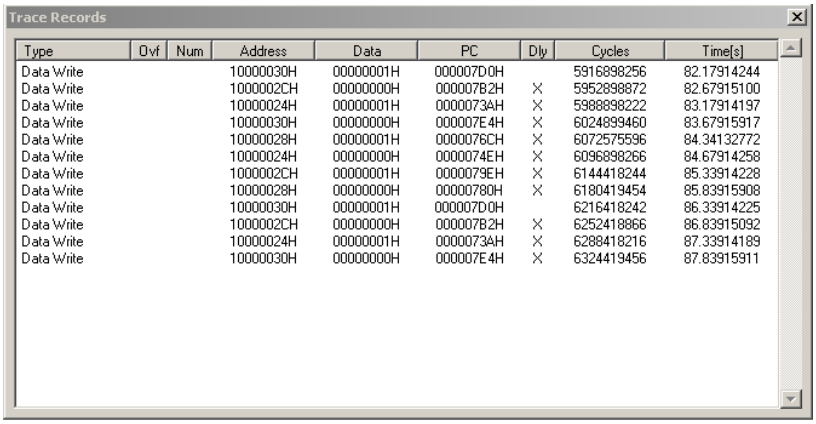
Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. A user program runs at full speed and needs no code stubs or instrumentation software added to your programs.

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.



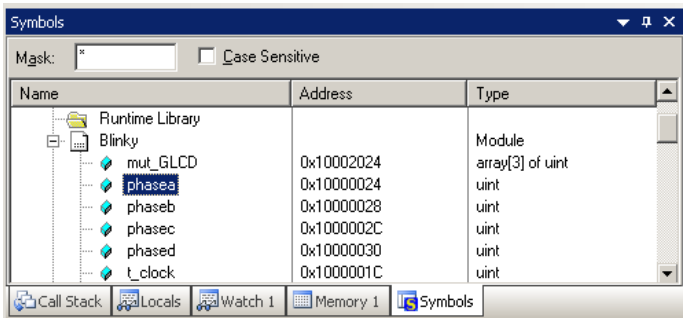
3. The Trace Records window will open up as shown here:
4. The ITM entries are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. To turn this off select Debug/Debug Settings and click on the Trace tab. Unselect the ITM Stimulus Port 31.
5. Port 0, EXCTRC and Periodic can also be unselected.
   TIP: Port 0 is used for the Debug Viewer.
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.





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 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.

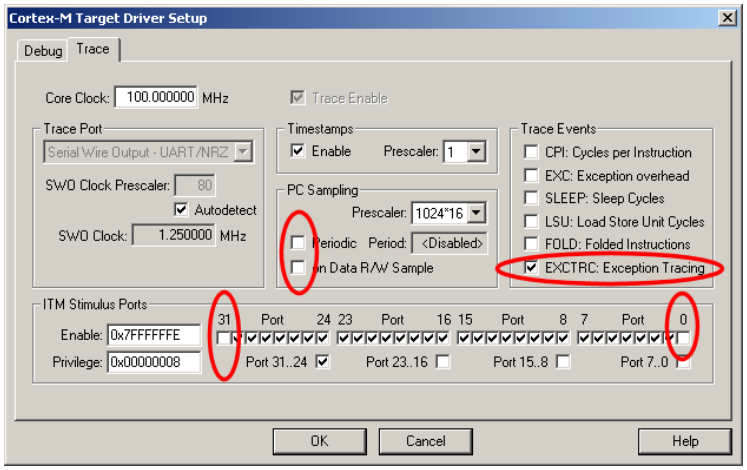
TIP: The next version of uVision will display the source and assembly code in a new trace window.

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

Exceptions and Interrupts:

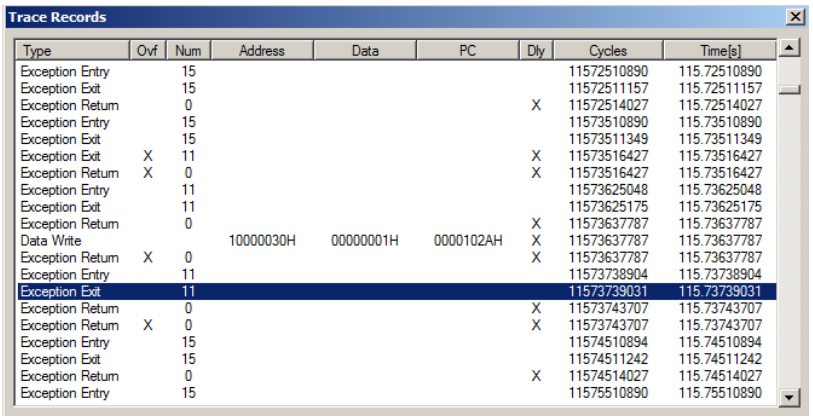
The LPC1700 family has many interrupts and it can be difficult to determine when or how many times they are activated. Serial Wire Viewer on the LPC1700 family makes this easy.

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 window 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. You can scroll for the other three data writes.
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.

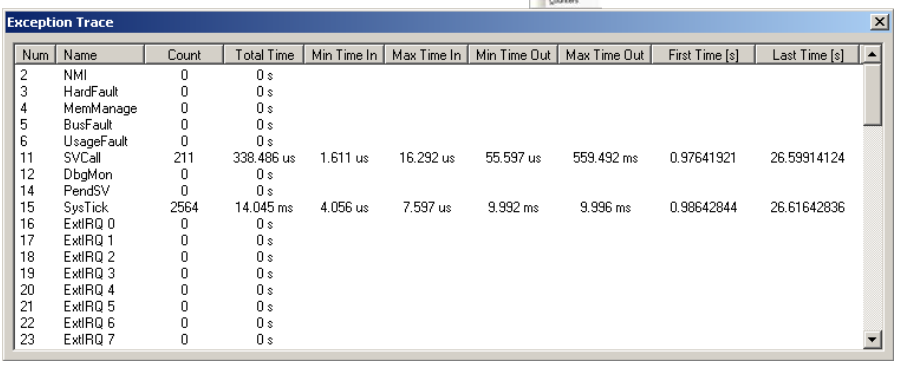


Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					11572510890	115.72510890
Exception Exit		15					11572511157	115.72511157
Exception Return		0				X	11572514027	115.72514027
Exception Entry		15					11573510890	115.73510890
Exception Exit		15					11573511349	115.73511349
Exception Exit	X	11				X	11573516427	115.73516427
Exception Return	X	0				X	11573516427	115.73516427
Exception Entry		11					11573625048	115.73625048
Exception Exit		11					11573625175	115.73625175
Exception Return		0				X	11573637787	115.73637787
Data Write			10000030H	00000001H	0000102AH	X	11573637787	115.73637787
Exception Return	X	0				X	11573637787	115.73637787
Exception Entry		11					11573738904	115.73738904
Exception Exit		11					11573739031	115.73739031
Exception Return		0				X	11573743707	115.73743707
Exception Return	X	0				X	11573743707	115.73743707
Exception Entry		15					11574510894	115.74510894
Exception Exit		15					11574511242	115.74511242
Exception Return		0				X	11574514027	115.74514027
Exception Entry		15					11575510890	115.75510890

TIP: The SWO pin is a one pin output 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 displayed 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.



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						

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 This example uses a ULINK2/ME. You can configure a ULINK*pro* or J-Link.

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

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

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

Configure Serial Wire Viewer trace:

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

Build, Load and RUN EXTI:

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

View the Trace and create exception EXTI:

6. The Trace Records and Exception Trace windows should still be open. Open them if they are not.
7. Press the INT0 button and EXTIrq 21 (Number 37) will display. You may have to scroll down to see this.
8. When you press the INT0 button, Exception 37 events are displayed in both windows.
9. The interrupt handler function **`EINT3_IRQHandler()`** is executed each time you press INT0.

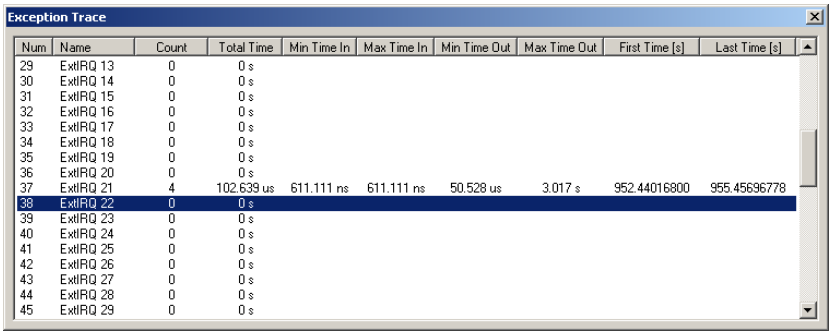
Trace Records Exceptions Type:

Entry: when the exception is entered.

Exit: when the exception or interrupt exits.

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

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



Switch Bounce:

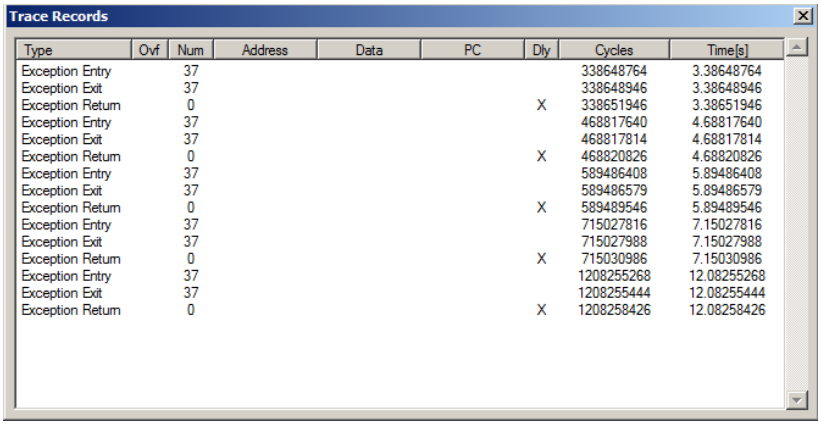
You might notice as you press the INT0 button that sometimes the sequence of switching LEDs jumps. This is caused by switch bounce. You can correct this by adding this C code to the beginning of the interrupt handler EINT3_IRQHandler():

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

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

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

Please leave this program running for the next exercise on PC Samples.

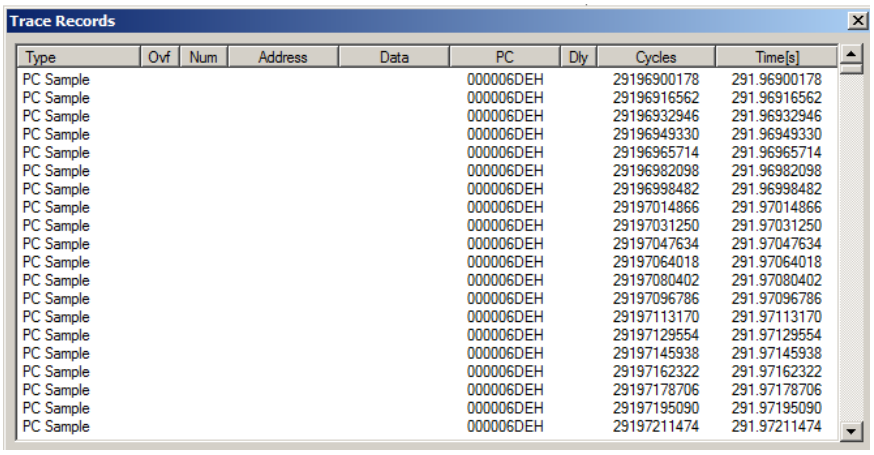


PC Samples:

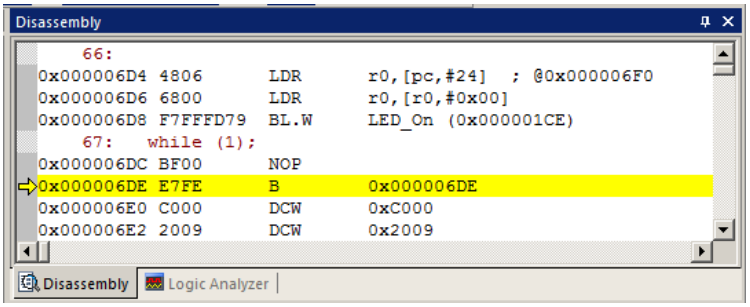
Serial Wire Viewer can only display a sampling of the program counter. To capture all of the PCs use the ETM trace. ETM is perfect to find problems associated with program flow such as "I went into the weeds and how did I get here?".

SWV can display at best every 64th instruction (with ULINK*pro*). It is better to keep the sample rate 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.
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 6DE 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 which then executes the interrupt service routine.



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 0x6DE).
9. **Note:** you can get different PC values depending on the compiler optimization level.
10. Since this program spends almost all of its time executing the Branch to itself, it will be difficult to see any other PC value with this program.



TIP: If you need to see these instructions, use ETM trace. ETM records all the instructions. Most NXP Cortex-M3 and M4 processors support ETM.

10) 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.

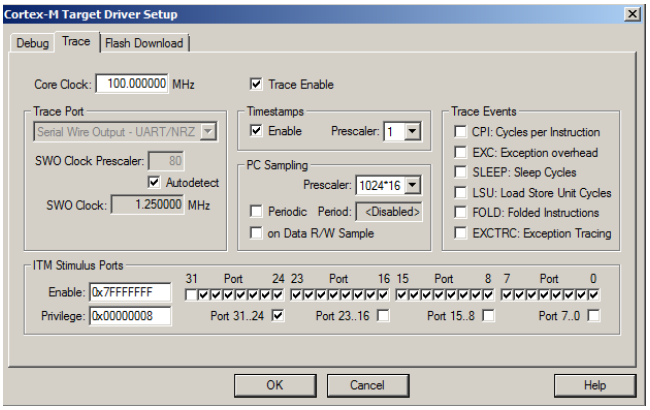
1. Open the RTX_Blinky project you used before. You can select it at the bottom of Project menu in the recent files list.
2. Add this code to Blinky.c. A good place is right after the place where you declared the four phase*x* variables.

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

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

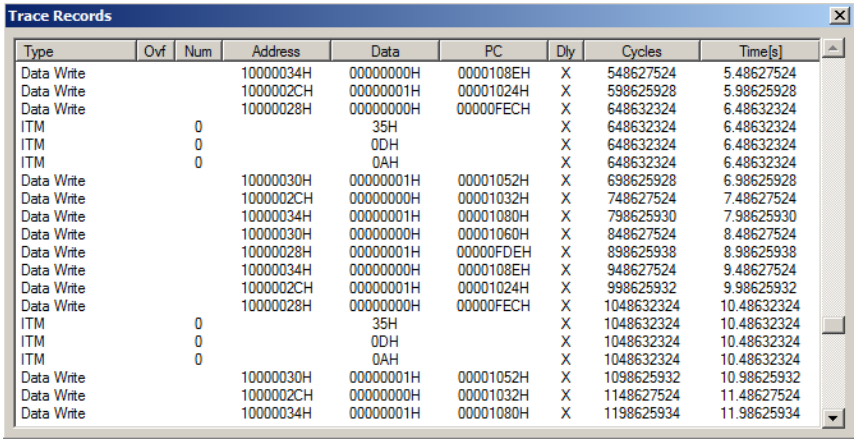
```
ITM_Port8(0) = 0x35;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```

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



To see the Trace Records

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



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

Explanation:

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

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

ITM Conclusion

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

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

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

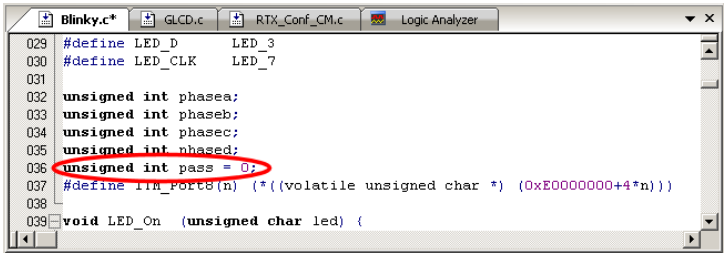
TIP: ITM_SendChar is a useful function you can use to send characters. It is found in the header core.CM3.h.

11) Watchpoints: *Conditional Breakpoints*

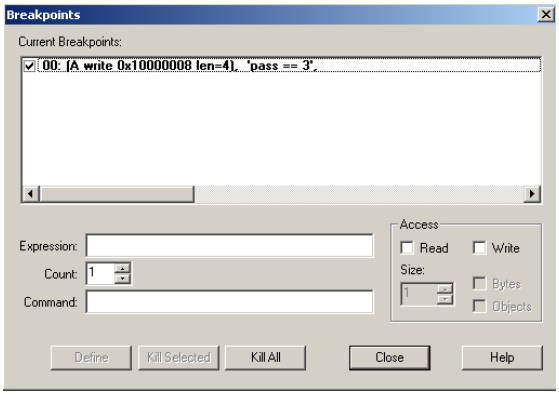
LPC1700 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. Usually the debugger will take one and perhaps two breakpoints for its operations. The LPC1700 also has four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic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

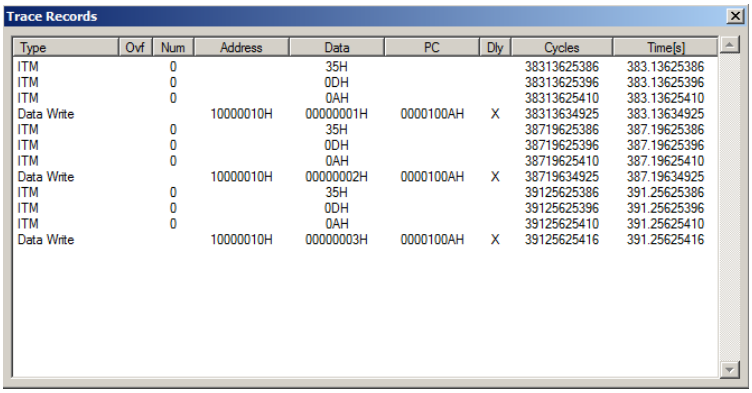
1. Stop the program and leave Debug mode. Click on  and then 
2. Add this line in Blinky.c in the area where you declared phasea. This means we want this to be a global variable.

   ```
   unsigned int pass = 0;
   ```
3. In task1(phasea) near where you entered the ITM write code, enter this line: **`pass++;`**
4. Your result should look similar to the screen displayed below for the declaration of **`pass`**.
5. Rebuild the project, Load and Enter Debug mode. 
6. Remove all variables in the Logic Analyzer window if there any by clicking on "Setup" and selecting the "Kill All" button.
7. Add variable **`pass`** to the Logic Analyzer. Set the Display Range to 0x0 and 0xF.
8. Click on Close to return. Set Range to 5 seconds by using the Zoom: buttons.



9. Select the Debug menu and select Breakpoints or press Ctrl-B.
10. In the Expression box enter: pass == 3. Select the Write box.
11. Click on Define and it will be accepted as on the right here:
12. Click on Close.
13. Enter the variable pass to the Watch window by dragging and dropping it or enter manually.
14. Open Debug/Debug Settings and select the trace tab. Check "on Data R/W sample" and uncheck EXTRC if checked.
15. Click on OK twice. Open the Trace Records window.
16. Click on RUN.
17. When **`pass`** equals 3, the program will stop. This is how a Watchpoint works. There are more options.
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. Also note the data writes 0x35, 0x0D and 0x0A from your previous ITM exercise.
21. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints box.





22. To repeat this, click on the RESET icon and then RUN.

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.

12) CAN: Controller Area Network

CAN is a network that is easy to implement. It is a peer-to-peer network and adding nodes is very easy. For more detailed information on the CAN bus and complete exercises using CAN for the LPC2300 and LPC1700 series obtain the CAN Primer from www.keil.com.

1. **Connectors:** The MCB1700 board has two DB-9 connectors labeled CAN1 and CAN2. These are the two CAN controllers. You must connect pin 2 of each connector to the other and also pin 7 to the other. Do not cross them. You can use two DB-9 connectors or jumper wires. Make sure the connections are reasonably sturdy. See the first TIP below for an explanation.
2. Start µVision by clicking on its icon on your Desktop if it is not already running.
3. Select Project/Open Project and open the project file C:\Keil\ARM\Boards\Keil\MCB1700\CAN\CAN.Uv2.
4. Compile the source files by clicking on the Rebuild icon.  . They will compile with no errors or warnings.
5. Click on the Load icon to program the Flash memory.  . A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
7. The LCD screen will display a value of both Tx: and Rx: and will vary when you rotate the potentiometer P2.

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

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

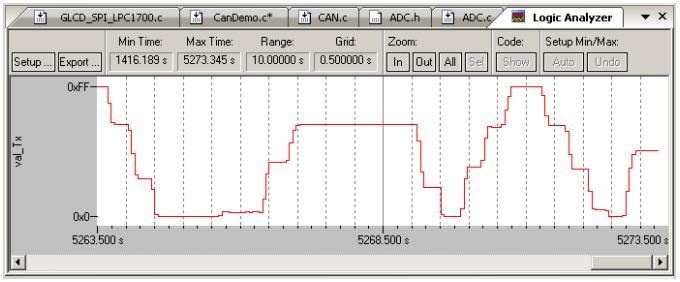
Time / 10 mSec	Identifier	Format	Flags	Data
00:00:39.85	21	Std		40
00:00:39.91	21	Std		45
00:00:39.97	21	Std		4B
00:00:40.03	21	Std		53
00:00:40.09	21	Std		5C
00:00:40.15	21	Std		69
00:00:40.21	21	Std		74
00:00:40.27	21	Std		87
00:00:40.33	21	Std		98
00:00:40.38	21	Std		A2
00:00:40.44	21	Std		B1
00:00:40.50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

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

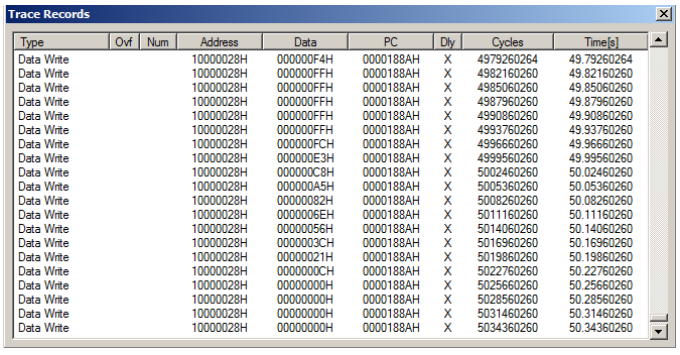
TIP: If only Tx: changes, either the loopback cable isn't connected or you are using only a ULINK-ME to power the board. Connect a USB cable from your computer to the MCB1700 board to provide 5 volts to the CAN transceiver in this case.

Logic Analyzer Window:

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



1. Stop the program and leave debug mode.
2. Open 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 and EXCTRC. Select on Data R/W Sample. Click Close twice.
3. Enter debug mode. 
4. Insert the global variable **`val_Tx`** in CanDemo.c into the Logic Analyzer window with a range 0 to 0xFF.
5. Click on Zoom icons to set Grid to 2 seconds.
6. Insert **`val_Tx`** into the Watch window.
7. Open the Trace Records window. RUN the program.
8. You will see the data change as you rotate the pot in both the LA window shown here and in the Watch window in real time stealing no CPU cycles.
9. The trace records window will show the CAN data write to the variable **`val_Tx`**. All are timestamped.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000028H	000000F4H	0000188AH	X	4979260264	49.79260264
Data Write			10000028H	000000FFH	0000188AH	X	4982160260	49.82160260
Data Write			10000028H	000000FFH	0000188AH	X	4985060260	49.85060260
Data Write			10000028H	000000FFH	0000188AH	X	4987960260	49.87960260
Data Write			10000028H	000000FFH	0000188AH	X	4990860260	49.90860260
Data Write			10000028H	000000FFH	0000188AH	X	4993760260	49.93760260
Data Write			10000028H	000000FCH	0000188AH	X	4996660260	49.96660260
Data Write			10000028H	000000E3H	0000188AH	X	4999560260	49.99560260
Data Write			10000028H	000000C8H	0000188AH	X	5002460260	50.02460260
Data Write			10000028H	000000A5H	0000188AH	X	5005360260	50.05360260
Data Write			10000028H	00000082H	0000188AH	X	5008260260	50.08260260
Data Write			10000028H	0000006EH	0000188AH	X	5011160260	50.11160260
Data Write			10000028H	00000056H	0000188AH	X	5014060260	50.14060260
Data Write			10000028H	0000003CH	0000188AH	X	5016960260	50.16960260
Data Write			10000028H	00000021H	0000188AH	X	5019860260	50.19860260
Data Write			10000028H	0000000CH	0000188AH	X	5022760260	50.22760260
Data Write			10000028H	00000000H	0000188AH	X	5025660260	50.25660260
Data Write			10000028H	00000000H	0000188AH	X	5028560260	50.28560260
Data Write			10000028H	00000000H	0000188AH	X	5031460260	50.31460260
Data Write			10000028H	00000000H	0000188AH	X	5034360260	50.34360260

For more detail on creating your own CAN network, obtain the CAN Primer from www.keil.com.

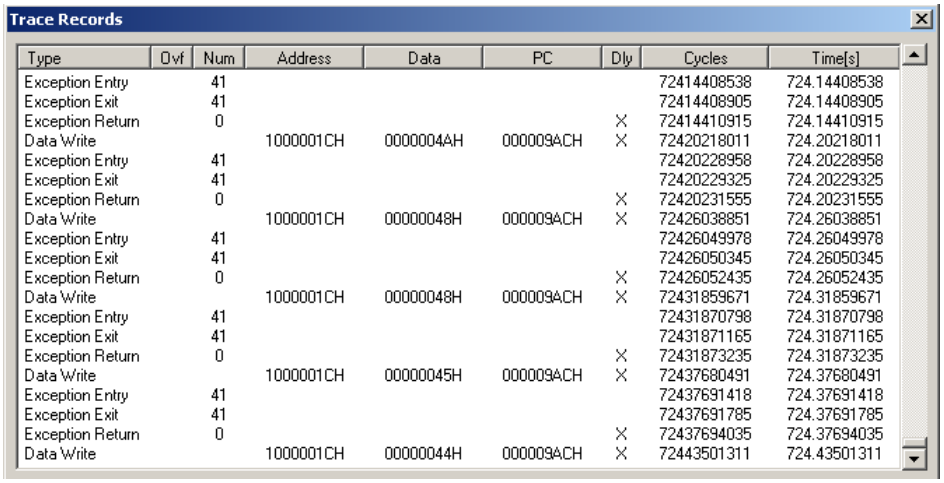
Using Watchpoints and Serial Wire Viewer with CAN

1. Stop the program if still running. µVision must be in debug mode to access the watchpoints.
2. Double-click on trace Records to clear it. (this step is not strictly necessary: new trace frames are appended to the end).
3. Open Debug/Breakpoints and enter in the dialog box: **val_Tx == 0x44** Select Read and click on Define and then Close.
4. Double-click in the Trace Records box to clear it and run the program by clicking on RUN. Or open it if it is not.
5. Adjust the pot to indicate 0x44. The first time this value is written to val_Tx, the program will stop.
6. Note the value in the Watch window will equal 0x44 ! The LCD may or may not have been updated yet.
7. Scroll to the bottom of the Trace Records and the value of 0x44 will be visible on the last line as shown below.
8. There will be a read of 0x44 at the end of the trace plus the address of the instruction that caused the trigger !
9. In this case, the last frame says a Data Read 0f 0x44 occurred to address 0x1000001C by the instruction located at 0x09AC.

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

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.

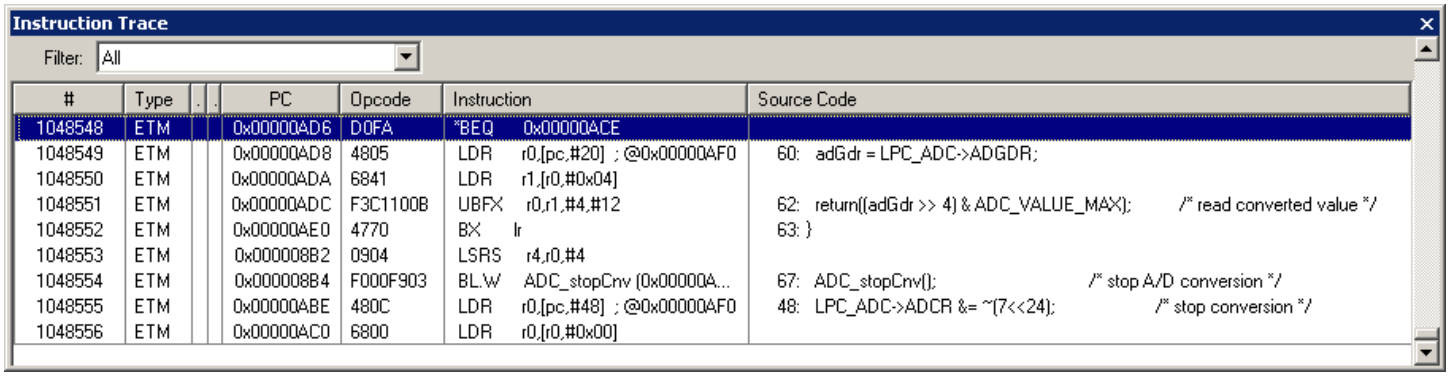
This is one of the powers of trace: you can see what happened to your program and how. If a bad value was written to one of your variables; you can tell when it happened and what instruction made this write. The possibilities of advanced debugging are great with trace.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		41					72414408538	724.14408538
Exception Exit		41					72414408905	724.14408905
Exception Return		0				X	72414410915	724.14410915
Data Write			1000001CH	0000004AH	000009ACH	X	72420218011	724.20218011
Exception Entry		41					72420228958	724.20228958
Exception Exit		41					72420229325	724.20229325
Exception Return		0				X	72420231555	724.20231555
Data Write			1000001CH	00000048H	000009ACH	X	72426038851	724.26038851
Exception Entry		41					72426049978	724.26049978
Exception Exit		41					72426050345	724.26050345
Exception Return		0				X	72426052435	724.26052435
Data Write			1000001CH	00000048H	000009ACH	X	72431859671	724.31859671
Exception Entry		41					72431870798	724.31870798
Exception Exit		41					72431871165	724.31871165
Exception Return		0				X	72431873235	724.31873235
Data Write			1000001CH	00000045H	000009ACH	X	72437680491	724.37680491
Exception Entry		41					72437691418	724.37691418
Exception Exit		41					72437691785	724.37691785
Exception Return		0				X	72437694035	724.37694035
Data Write			1000001CH	00000044H	000009ACH	X	72443501311	724.43501311

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

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



#	Type	PC	Opcode	Instruction	Source Code
1048548	ETM	0x00000AD6	D0FA	*BEQ 0x00000ACE	
1048549	ETM	0x00000AD8	4805	LDR r0,[pc,#20] ; @0x00000AF0	60: adGdr = LPC_ADC->ADGDR;
1048550	ETM	0x00000ADA	6841	LDR r1,[r0,#0x04]	
1048551	ETM	0x00000ADC	F3C1100B	UBFX r0,r1,#4,#12	62: return((adGdr >> 4) & ADC_VALUE_MAX); /* read converted value */
1048552	ETM	0x00000AE0	4770	BX lr	63: }
1048553	ETM	0x000008B2	0904	LSRS r4,r0,#4	
1048554	ETM	0x000008B4	F000F903	BL.W ADC_stopCnv (0x00000A...	67: ADC_stopCnv(); /* stop A/D conversion */
1048555	ETM	0x00000ABE	480C	LDR r0,[pc,#48] ; @0x00000AF0	48: LPC_ADC->ADCR &= ~(7<<24); /* stop conversion */
1048556	ETM	0x00000AC0	6800	LDR r0,[r0,#0x00]	

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

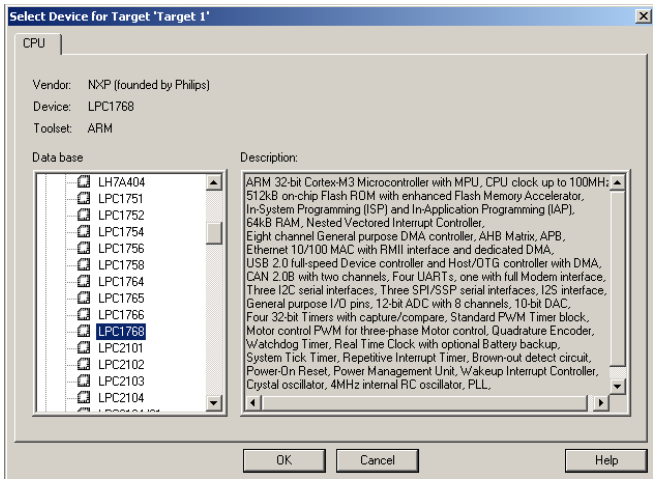
Note: The current version of Keil MDK (4.53) only displays data writes and not reads. This is to prevent data overruns in the Trace Records window. Future versions may include data reads.

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

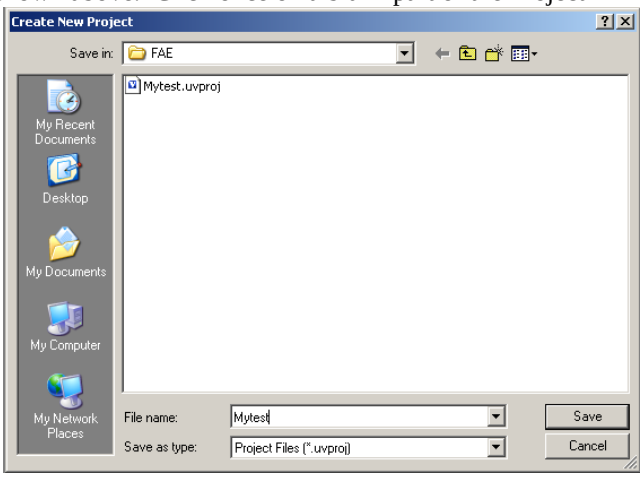
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will 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.

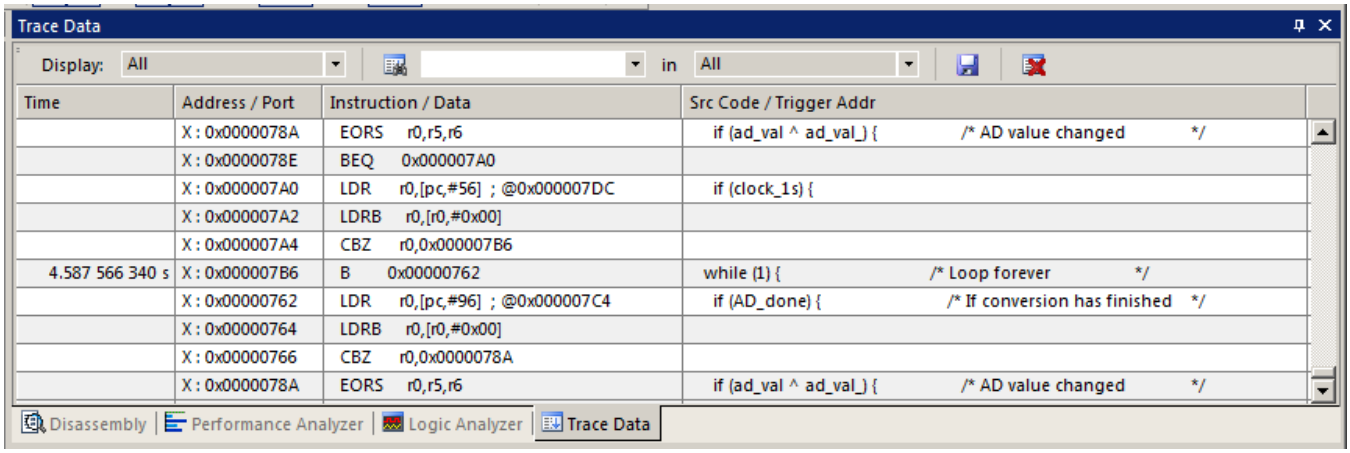
This completes the exercise of creating your own project from scratch.

14) What does a ULINK*pro* offer you ?

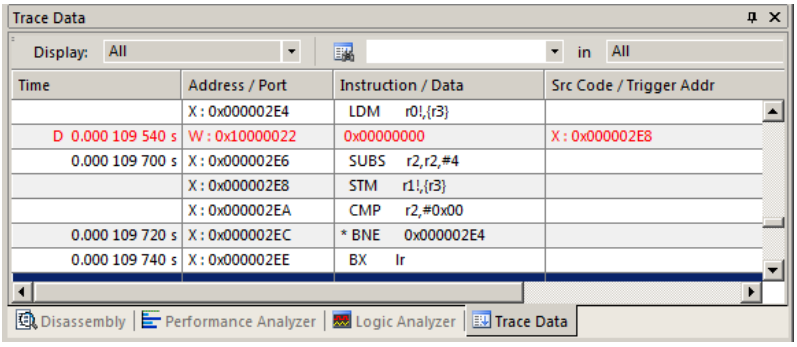
We have seen what features the Serial Wire Viewer provides with the LPC4330. Many NXP Cortex-M3 and M4 processors, also have Embedded Trace Macrocell (ETM). ETM provides all the program counter (PC) values.

Once we have all the PC values, we can easily determine the following four functions and display them in µVision:

1. **Instruction Trace:**
   This enables program flow debugging such as the infamous "in the weeds" problem since a complete record of the program flow is recorded for later examination. The PC values are tied to the appropriate C source and assembly instructions as shown in the first window shown below:
   Problems that normally would take extensive debugging time can be found very quickly with ETM trace. Double click on a line in this window and you will be taken to that location in the Disassembly and Source windows.

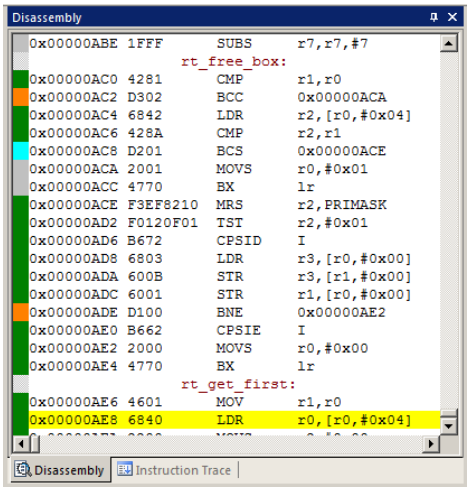


Serial Wire Viewer frames are stored along with ETM frames. Shown here in red is a data write interleaved with assembly instructions.
Filtering displays only those types you want to see.

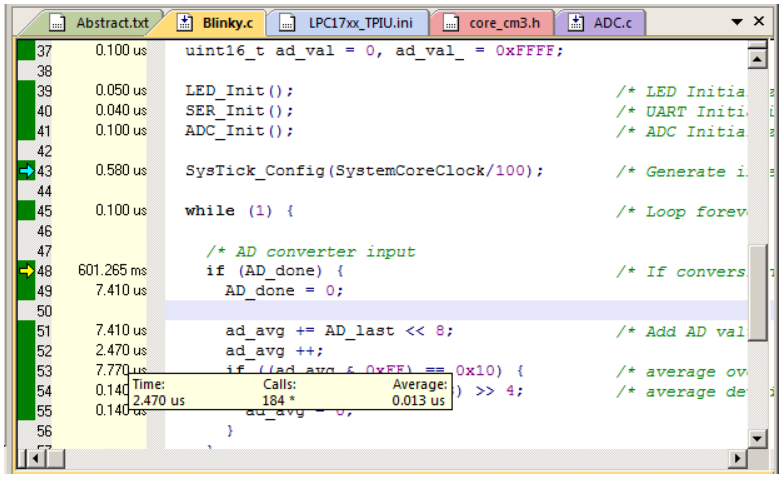


2. **Code Coverage:** Were all the instructions in your program executed hence tested ? Unexecuted instructions are a hazard. Code Coverage is often required for certain certifications such as the US FDA. Each instruction is colour coded as shown here and a report can be created. This colour coding is also displayed in the C/C++ source windows.

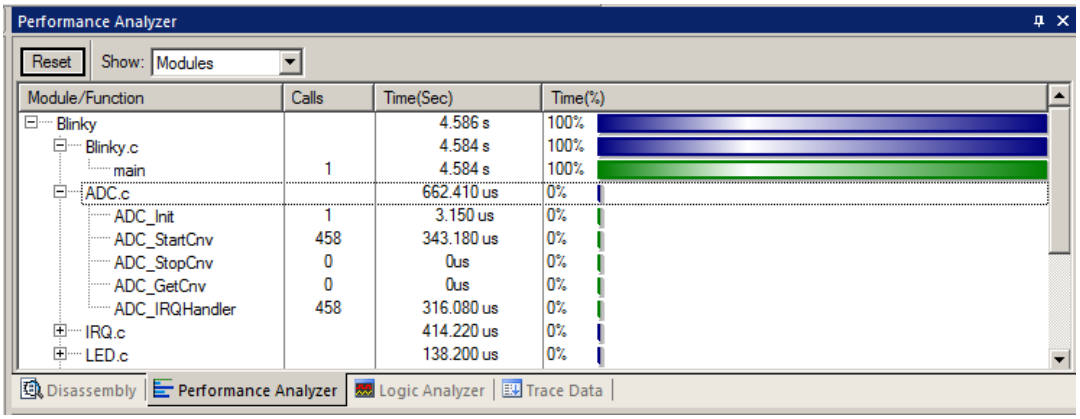
1. Green: this assembly instruction was executed.
2. Gray: this assembly instruction was not executed.
3. Orange: a Branch is always not taken.
4. Cyan: a Branch is always taken.
5. Light Gray: there is no assembly instruction here.
6. RED: Breakpoint is set here.
7. Yellow arrow: Next instruction to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



3. **Execution Profiling:** How long did it take for this function or set of assembly instructions to execute ? How many times did they execute ? With a configurable resolution from many C source lines down to individual instructions, Instruction profiling gives you an accurate indication of program timings.
How many times an instruction or a section of code was executed is also available.
This screen shows how much time each source line has executed. Number of times a line or section of code was executed can also be shown instead.
Hover your mouse over an instruction or section and the statistics are displayed as shown in this window:



4. **Performance Analysis (PA):** Where is my program spending all of its time ? PA tells you in a graphical format how long it takes for each function to execute. You can compare this to how long you expected them to run and to find areas where the program is outside of your expectations and design. This information and more is presented in the Performance Analysis window shown below:



If you have a ULINK*pro*, you can try the MCB1700 example BlinkyUlp: C:\Keil\ARM\Boards\Keil\MCB1700\Blinky_ULp

15) Serial Wire Viewer Summary:

Serial Wire Viewer can see:

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

Serial Wire Viewer displays in various ways:

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

Trace is good for:

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

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

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

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

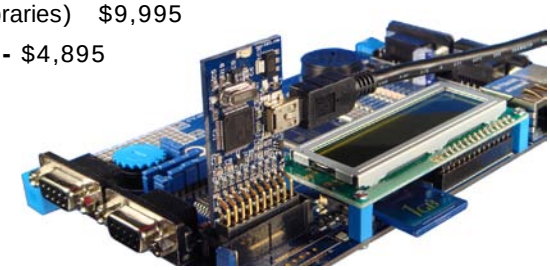
Other Useful Documents:

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

16) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) $9,995
- MDK-Standard (unlimited compile and debug code and data size) - $4,895
- MDK-Basic (256K Compiler Limit, No debug Limit) - $2,695
- MDK-Lite (Evaluation version) $0



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

- ***For special promotional pricing and offers, please contact Keil Sales for details.***

The Keil RTX RTOS is now provided under a Berkeley BSD type license. This makes it free.
Keil also provides free DSP libraries for the Cortex-M3 and Cortex-M4.

All versions, including MDK-Lite, includes Keil RTX RTOS *with source code !*
Call Keil Sales for more details on current pricing. All products are available.
All products include Technical Support for 1 year. This can easily be renewed.
Call Keil Sales for special university pricing.
For the ARM University program: go to www.arm.com and search for university.
Keil supports many other NXP processors including 8051, ARM7™ and ARM9™ processors. See the Keil Device Database® on www.keil.com/dd for the complete list of NXP support. This information is also included in MDK.

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.
Prices are for reference only and are subject to change without notice.
For the entire Keil catalog see www.keil.com or contact Keil or your local distributor.
See www.keil.com/nxp for more NXP specific information.

For more information:

Keil products can be purchased directly from ARM or through various distributors.
Keil Distributors: See www.keil.com/distis/ or www.embeddedsoftwarestore.com
Keil Direct Sales In USA: sales.us@keil.com or 800-348-8051. **Outside the US:** sales.intl@keil.com
Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.
For comments or corrections please email bob.boys@arm.com.
For the latest version of this document, see www.keil.com/nxp