

For the NGX Evaluation Board with the LPC4330 Processor & ULINK-ME

Introduction

The purpose of this lab is to introduce you to the NXP Cortex™-M4 processor family using the ARM® Keil™ MDK toolkit featuring the IDE μ Vision®. We will use the Serial Wire Viewer (SWV) to display various processor events real-time.. At the end of this tutorial, you will be able to confidently work with NXP processors and MDK. For the latest version of this document please visit www.keil.com/nxp

Keil MDK comes in an evaluation version that limits code and data size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a license number will turn it into the full, unrestricted version. Contact Keil sales for a temporary full version license if you need to evaluate MDK with programs greater than 32K. MDK includes a full version of Keil RTX™ RTOS. No royalty payments are required. RTX source code is now included with all versions of Keil MDK™.

Why Use Keil MDK ?

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

1. μ Vision 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 is free with a BSD type license.
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.
4. Choice of adapters: ULINK2™, ULINK-ME™, ULINKpro™ or Segger J-Link (version 6 or later).
5. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.



Other significant features shown in this document:

1. Serial Wire Viewer (SWV) with the ULINK2. ETM trace examples with the ULINKpro are shown starting on p 20.
2. Dual Core debugging control for the Cortex-M0 and the Cortex-M4 cores on the NXP LPC4350 processor.
3. Real-time Read and Write 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.
4. Eight Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also called Access Breaks).
5. RTX Viewer: a kernel awareness program for the Keil RTX RTOS that updates while the program is running.

Serial Wire Viewer (SWV): Use any ULINK for this debugging feature.

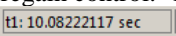
Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and a timestamp. This information comes from the ARM CoreSight™ debug module integrated into the Cortex-M4. SWV is output on the Serial Wire Output (SWO) pin found on the JTAG/SWD adapter connector. SWV is not available for the Cortex-M0 processor because of its tiny size. SWV is a standard feature on Cortex-M3 and Cortex-M4.

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

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

1) NGX Evaluation Boards, MDK and Examples Install	3
2) GPIO Example with Blinky:	4
1. GPIO_Blinky Example Program using a ULINK2:	4
2. Hardware Breakpoints:	5
3. Call Stack + Locals Window:	5
4. Watch and Memory Windows and how to use them:	6
5. Configuring the Serial Wire Viewer (SWV):	7
6. Using the Logic Analyzer (LA) with ULINK2:	8
7. Watchpoints: Conditional Breakpoints:	9
3) RTX RTOS Example RTX_Blinky:	10
1. RTX Kernel Awareness using Serial Wire Viewer (SWV):	10
2. RTX Viewer: Configuring Serial Wire Viewer (SWV):	11
3. Logic Analyzer Window: View RTOS variables:	12
4. Trace Records and Exception Windows:	13
4) USB and Interrupt Example:	14
5) Dual Core MBX Example:	16
1. Open and Compile Cortex-M0	16
2. Open and Compile Cortex-M04	16
3. Configuring the ULINK2/ME:	17
4. Running the Program:	18
5. Breakpoints in main():	19
6. Watch windows:	19
6) What does a ULINK_{pro} and ETM Trace provide you ?	20
1. Instruction Trace:	20
2. Code Coverage:	21
3. Execution Profiling:	21
4. Performance Analysis:	21
7) Keil Products and Contact Information:	22

USB laptop bug:

A few laptops exhibit difficulty processing the large amount of data from Serial Wire Viewer (SWV). It causes μ Vision to freeze and disconnecting the ULINK2/ME is the only way to regain control. This problem does not happen if you do not have SWV enabled. The T1 counter will stop incrementing.  Desktops and Windows 7 are not affected: only XP. SOLUTIONS: Add a USB PCMCIA card, turn off SWV, or use a different computer.

1) NGX ARM Processor Evaluation Boards:

LPC-4330-Xplorer: NXP dual core Cortex-M4 and Cortex-M0 processor. This board connects to a Keil ULINK with a compact 10 pin JTAG/SWD connector. This processor does not have internal flash. The board has external SPIFI flash.

LPC-1830-Xplorer: NXP Cortex-M3 processor. The instructions in this document should generally work with this board.

Labs similar to this one are available for the NXP Cortex-M3 (LPC1700) with a version with CAN (Controller Area Network). See www.keil.com/nxp for more information. Keil makes several boards with NXP processors for ARM7TDMI, ARM9, Cortex-M0 and Cortex-M3 and the 8051. See www.keil.com/nxp

Software Installation:

This document was written for Keil MDK 4.50 or later which contains μ Vision 4. The evaluation copy of MDK (MDK-Lite) is available free on the Keil website. Do not confuse μ Vision4 with MDK 4.0. The number "4" is a coincidence.

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



Please install MDK into the default location of C:\Keil. You can use the evaluation version of MDK and a ULINK2, ULINK-ME, ULINKpro or J-Link for this lab. No license number is needed to compile up to 32 Kb.

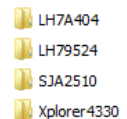
The addition of a license number converts MDK-Lite, the evaluation version, into a full, unrestricted copy of MDK.

The ULINKpro adds Cortex-Mx ETM trace support. It also adds faster programming time and better trace display. Most NXP Cortex-M3/M4 parts are equipped with ETM. All have SWV. The Cortex-M0 has neither SWV, Watchpoints or ETM. It does have non-intrusive read/write to memory locations (for Watch and Memory windows) and hardware breakpoints.

The NGX board does not have the 20 pin compact connector required for ETM operation with a ULINKpro.

2. NGX Example Programs:

This tutorial uses the NGX V 1.2 example files available from <http://shop.ngxtechnologies.com>. Click on the picture of the LPC4330 board. The filename is LPC4330_Xplorer_Examples_V1.2.zip. Please unzip these to the directory C:\Keil\ARM\Boards\NXP as shown here:



1. **Flash Programming Algorithms:** μ Vision uses files with an .FLM extension to program Flash memories. These files must be located in C:\Keil\ARM\Flash from where you select the correct one. For the NGX examples, they are initially located in:
C:\Keil\ARM\Boards\NXP\Xplorer4330\CMSISv2p10_LPC43xx_DriverLib\Tools\Flash\Keil_Binaries.
2. These need to be copied to C:\Keil\ARM\Flash. The correct Flash file to use with the NGX LPC4330 board is: SPIFI_LPC18xx-43xx_140.FLM (Flash starts at 0x1400 0000).

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

- **JTAG:** Provides access to the CoreSight debugging module located in 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 μ Vision Cortex-M Target Driver Setup. See page 15, first picture. The SWJ box must be selected. SWV must use SWD with ULINK2 because of the TDIO conflict described in **SWO** below.
- **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. Therefore, to use SWV, you must use SWD instead of JTAG. ULINKpro does not have this limitation.
- **Trace Port:** A 4 bit port ULINKpro uses to output ETM frames and optionally SWV (rather than SWO pin).
- **ETM:** Embedded Trace Macrocell: Provides all the program counter values. Only the ULINKpro provides ETM. The NGX boards currently do not have the required 20 pin compact connector needed for ETM operation.

NGX Debug Connectors:

JTAG connector: A compact 10 pin JTAG/SWD connector (J2) is provided. J2 provides JTAG, SWD and SWO access.

Memory: The NGX board contains a serial SPIFI flash memory U6. You can also run programs in internal RAM. μ Vision is able to program both the flash and RAM.

2) Gpio_Blinky Example Program:

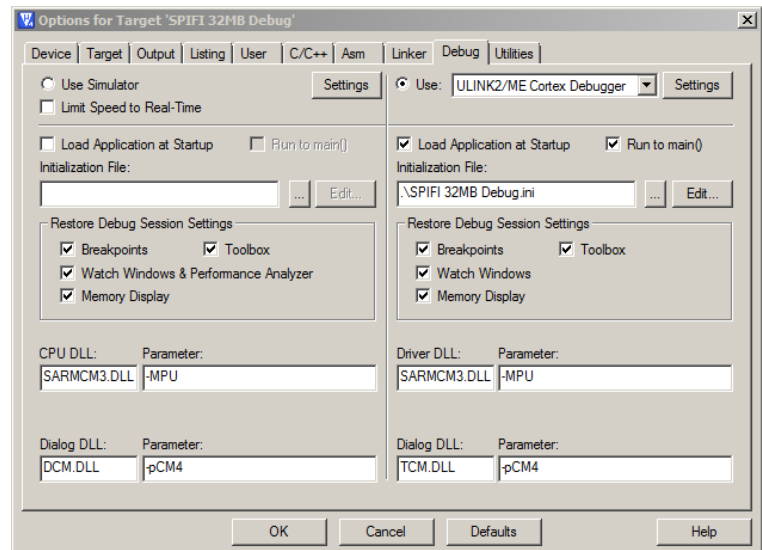
We will connect a Keil MDK development system to the NGX LPC4330 board using a ULINK-ME. We will be using the SPIFI flash for this part of the demonstration. A RAM example is available.

1. Connect the ULINK-ME as pictured here to the J2 JTAG connector. Connect the ULINK-ME to your PC with a USB cable.
2. Power the NGX board with a USB cable to either USB0 or USB1 and to your PC. USB1 is used in the USB example later on page 14.



3. Start μ Vision by clicking on its desktop icon.
4. Click Project/Open Project in the main μ Vision menu and select: LPC4330_Xplorer_Blinky.uvproj in the directory:
C:\Keil\ARM\Boards\NXP\Xplorer4330\LPC4330_Xplorer_Blinky\Keil
5. Select SPIFI Debug as shown here: This is where you could select RAM operation...but not right now.
6. Select Options for Target or ALT-F7 and select the Debug tab and the window below opens up:
7. This project is configured by default to use the ULINK2/ME debug adapter.
8. Delete the .ini file if present in the Initialization File: box. This is not needed by this program.
9. Click on OK to close this window.

10. Compile the source files by clicking on the Rebuild icon. Progress is indicated in the Build Output window.
11. Select File/Save All.
12. Program the SPIFI flash by clicking on the Load icon: Progress is indicated in the Build Output window.
13. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
14. Click on the RUN icon.



The yellow and green LEDs (D2 and D3) beside the LPC4330 processor will now blink:

Super TIP: If you get an error message when attempting to program the Flash or when entering debug mode, press the RESET button on the board *twice* and try again. Sometimes you have to press it only once or maybe three times. You may have to cycle the power to the board and/or ULINK.

1. You stop the program with the STOP icon.
2. You can single-step the program by clicking on its icon: or F11. Bring the disassembly window into focus if single step does not start.
3. Use Step Out and then Step to single step in other functions. Step Over is used to bypass a function.


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

Note: This program is running on the Cortex-M4 processor in the LPC4330. If you remove the ULINK2/ME, this program will run standalone in the LPC4330 as you have programmed it in the SPIFI 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 between Lines 072 through 080 in the source file Gpio_LedBlinky.c as shown below: Click on its tab if not visible.
2. A red circle is created and soon the program will stop at this point.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows.
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 LPC4350 has 8 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 μ Vision 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.

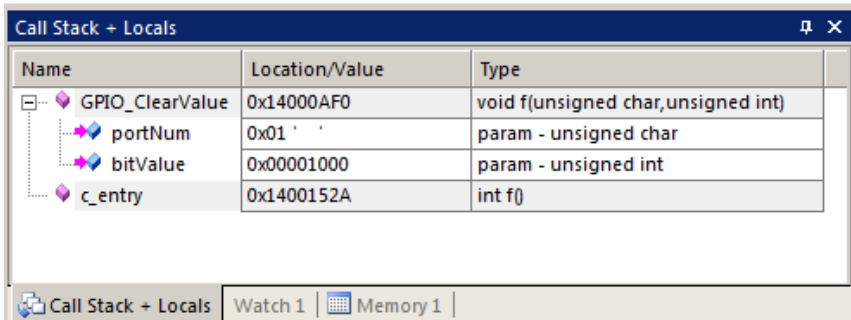
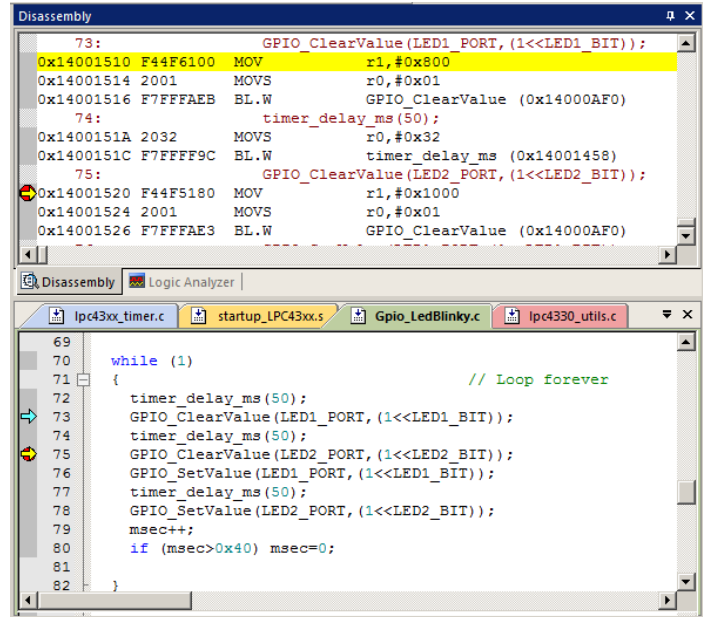
1. Click on the Call Stack + Locals tab.
2. Shown is the Locals window obtained after Step from the hardware breakpoint active from the previous page.
3. The contents of the local variables portNum and bitValue are displayed as well as the function name(s).
4. Using RUN, Step, Step Over and Step Out to enter and exit various functions, this window will update as appropriate.

TIP: This is standard “Stop and Go” debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables 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.

5. Remove the hardware breakpoint by clicking on its red circle !




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. It is also possible to “put” or insert values into these memory locations in real-time. It is possible to “drag and drop” variables or enter physical addresses or variables manually into windows while the program is running.


Watch window:


1. We will display in real time the global variable msec in the Watch window.
2. The global variable msec is in Gpio_LedBlinky.c. msec has been optimized out by the compiler as it is not used.

We will give it something to do. Stop the processor and exit debug mode .

3. In GPIO_Blinky.c, just after the last line (78) in the while loop, enter these two lines:

```
msec++;  
if (msec > 0x40) msec=0;
```

4. Compile the source files by clicking on the Rebuild icon. . Progress is indicated in the Build Output window.

5. Program the SPIFI flash by clicking on the Load icon: . Progress is indicated in the Build Output window.

6. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.

7. Start the processor. You can select Watch and Memory window variables while the program is running.

8. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.

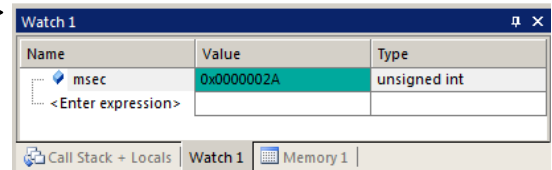
9. In Gpio_LedBlinky.c, block msec, click and hold and drag it into Watch 1.

Release it and it will be displayed updating as shown here: 

10. Msec will update in real-time.

11. You can also enter a variable manually by double-clicking and using copy and paste or typing the address or name manually.

You can also right click on msec and select its destination.



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.

6. Block the value for msec in the Watch window. Enter the value 0 and press Enter. 0 will be inserted into memory in real-time. It will quickly change as the variable is updated often by the program so you probably will not see this happen. You can also do this in the Memory window with a right-click and select Modify Memory.


Note: This will be optimized so values are easier to enter in a future release of μ Vision.

Memory window:

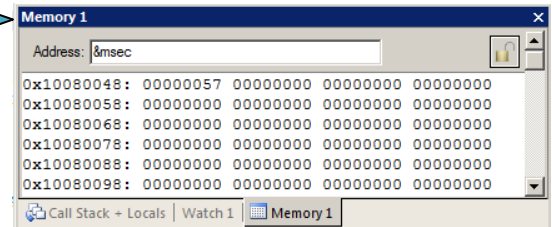
1. Drag ‘n Drop msec into the Memory 1 window or enter it manually while the program is running.
2. Note the changing value of msec is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to: but this not what we want to see at this time.

Note: right click in Memory 1 and select Unsigned Long to see the addresses as 32-bit numbers.

3. Add an ampersand “&” in front of the variable name and press Enter. Now the physical address is shown (0x1008_0048) in this case. This physical address could change with different compilation optimizations.



4. The data contents of msec is displayed as shown here: 

5. Right click on the memory value and select Modify Memory. Enter a value and this will be pushed into msec.



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

1. The global variable msec is now updated in real-time. This is ARM CoreSight technology working.


2. Stop the CPU and exit debug mode for the next step.  and 

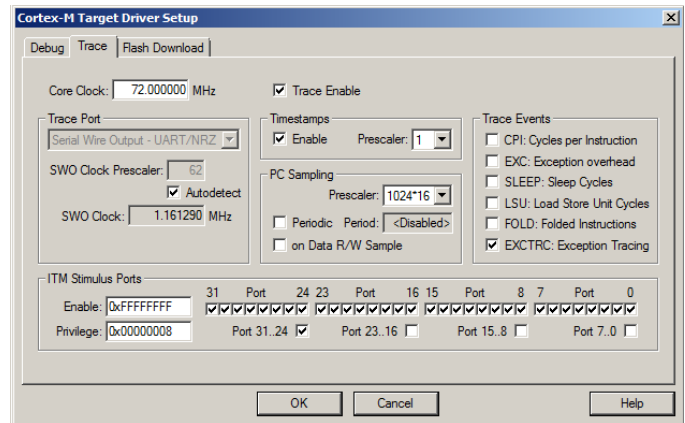
TIP: View/Periodic Window Update must be selected. Otherwise variables update only when the program is stopped.

5) Configuring the Serial Wire Viewer (SWV) with the ULINK2 or ULINK-ME:




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

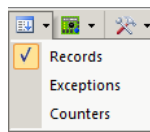
Configure SWV:

1. μ Vision must be stopped and in edit mode (not debug mode).
2. Select Options for Target  or ALT-F7 and select the Debug tab.
3. Click on Settings: beside the name of your adapter (ULINK Cortex Debugger) on the right side of the window.
4. Select the SWJ box and select SW in the Port: pulldown menu.
5. In the area **SW Device** this must be displayed: ARM CoreSight SW-DP. SWV will not work with JTAG.
6. Click on the Trace tab. The window below is displayed.
7. In Core Clock: enter 72 and select the Trace Enable box.
8. Select Periodic and leave everything else at default. Periodic activates PC Samples.
9. Click on OK twice to return to the main μ Vision menu. SWV is now configured.
10. Select File/Save All.



To Display Trace Records:

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



TIP: If you do not see PC Samples 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. ITM frames 0 and 31 are the only valid ones. Type 1 through 30 are not used in μ Vision and if appear are spurious and not valid. All frames have a timestamp displayed in CPU cycles and accumulated time.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					1400113CH		421355122	5.85215447
PC Sample					14001140H		421371506	5.85238203
PC Sample					14001138H		421387890	5.85260958
PC Sample					14001132H		421404274	5.85283714
PC Sample					1400112EH		421420658	5.85306469
PC Sample					1400112CH		421437042	5.85329225
PC Sample					1400149CH		421453426	5.85351981
PC Sample					14001498H		421469810	5.85374736
PC Sample					1400113CH		421486194	5.85397492
PC Sample					1400113CH		421502578	5.85420247
PC Sample					1400113EH		421518962	5.85443003
PC Sample					14001136H		421535346	5.85465758
PC Sample					1400112EH		421551730	5.85488514
PC Sample					1400112EH		421568114	5.85511269
PC Sample					1400112AH		421584498	5.85534025
PC Sample					1400149AH		421600882	5.85556781
PC Sample					140014A2H		421617266	5.85579536
PC Sample					1400113CH		421633650	5.85602292
PC Sample					14001140H		421650034	5.85625047
PC Sample					14001138H		421666418	5.85647803

Double-click this window to clear it.

If you right-click inside this window you can see how to filter various types of frames out. Unselect PC Samples and you will see nothing displayed as there are no other frames.



This is an excellent way to see various events such as interrupts are occurring and when. This is a very useful tool for displaying how many times an exception or other events are firing and when. We shall see more on this shortly.

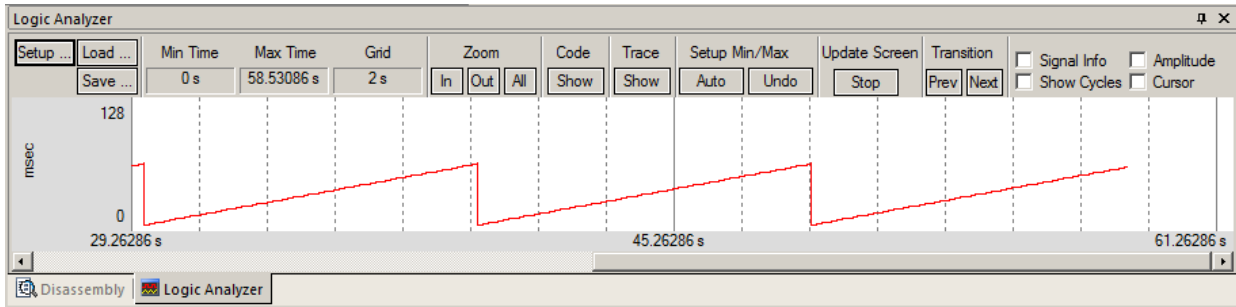
TIP: SWV is easily overloaded as indicated by an "x" in the OVF or Dly column. Select only that information needed to reduce overloading. There are more useful features of Serial Wire Viewer as we shall soon discover.

6) Using the Logic Analyzer (LA) with the ULINK2 or ULINK-ME:

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

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


1. The project Gpio_LedBlinky should still be open and is still in Debug mode and running.
2. **Note:** You can configure the LA while the program is running or stopped.
3. Select Debug/Debug Settings and select the Trace tab.
4. Unselect Periodic and EXCTRC. This is to prevent overload on the SWO pin. Click OK twice.
5. Click on RUN  to start the program again.
6. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 
7. Locate the variable msec in Gpio_LedBlinky.c.
8. Block msec and drag it into the LA window and release it. Or click on Setup in the LA and enter it manually. You can also right-click on msec and select Add 'msec' and then select Logic Analyzer.
9. Click on Setup and set Max: in Display Range to 0x80. Click on Close. The LA is completely configured now.
10. msec should still be visible in Watch 1. If not, enter it into the Watch 1 window.



11. Adjust the Zoom OUT or the All icon in the LA window to provide a suitable scale of about 2 sec:
12. Would you have guessed msec is a sawtooth wave from looking at its value changing in the watch window ?

TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: make them static or global. To see peripheral registers, enter their physical addresses into the Logic Analyzer and read or write to them. Physical addresses can be entered as: *((unsigned long *)0x2000000).

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

1. Select Debug/Debug Settings and select the Trace tab.
2. Select on Data R/W Sample. Click OK twice. Run the program. .
3. Open the Trace Records window and clear it by double-clicking in it.
4. The window similar below opens up:
5. The first line says:
The instruction at 0x1400_0E7C caused a write of data 0x001A to address 0x1008_0048 at the listed time in CPU Cycles or accumulated Time in seconds.

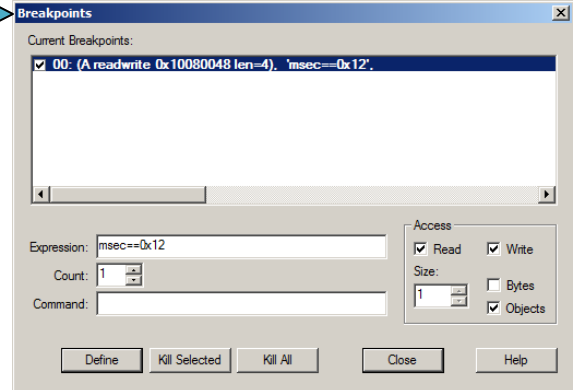
TIP: The PC column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin if there are too many overruns. µVision and CoreSight recover gracefully from trace overruns.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10080048H	0000001AH	14000E7CH		20891367778	290.15788581
Data Write			10080048H	00000019H	14000E7CH		20891439778	290.15888581
Data Write			10080048H	00000018H	14000E7CH		20891511778	290.15988581
Data Write			10080048H	00000017H	14000E7CH		20891583778	290.16088581
Data Write			10080048H	00000016H	14000E7CH		20891655778	290.16188581
Data Write			10080048H	00000015H	14000E7CH		20891727778	290.16288581
Data Write			10080048H	00000014H	14000E7CH		20891799778	290.16388581
Data Write			10080048H	00000013H	14000E7CH		20891871778	290.16488581
Data Write			10080048H	00000012H	14000E7CH		20891943778	290.16588581
Data Write			10080048H	00000011H	14000E7CH		20892015778	290.16688581
Data Write			10080048H	00000010H	14000E7CH		20892087778	290.16788581
Data Write			10080048H	0000000FH	14000E7CH		20892159778	290.16888581
Data Write			10080048H	0000000EH	14000E7CH		20892231778	290.16988581
Data Write			10080048H	0000000DH	14000E7CH		20892303778	290.17088581
Data Write			10080048H	0000000CH	14000E7CH		20892375778	290.17188581
Data Write			10080048H	0000000BH	14000E7CH		20892447778	290.17288581
Data Write			10080048H	0000000AH	14000E7CH		20892519778	290.17388581
Data Write			10080048H	00000009H	14000E7CH		20892591778	290.17488581
Data Write			10080048H	00000008H	14000E7CH		20892663778	290.17588581
Data Write			10080048H	00000007H	14000E7CH		20892735778	290.17688581

7) Watchpoints: Conditional Breakpoints

Most NXP Cortex-M3 and M4 processors have four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses watchpoints in its operations. This means in μ Vision you must have two variables free in the Logic Analyzer to use Watchpoints. The Cortex-M0 does not have any watchpoints because of its small size.

- Using the example from the previous page, stop the program. Stay in Debug mode.
- Enter the global variable msec into the Watch 1 window if it is not already there.
- Click on Debug and select Breakpoints or press Ctrl-B.
- The SWV Trace does not need to be configured to use Watchpoints. However, we will use it in this exercise.
- In the Expression box enter: “msec == 0x12” without the quotes. Select both the Read and Write Access.
- Click on Define and it will be accepted as shown here:
- Click on Close.
- Double-click in the Trace Records window to clear it.
- Msec should still be entered in the Logic Analyzer window.
- Click on RUN.



- When msec equals 0x12, the program will stop. This is how a Watchpoint works.
- You will see msec displayed as 0x12 in the Logic Analyzer as well as in the Watch window.
- Note the data write of 0x12 in the Trace Records window shown below in the Data column. The address the data written to and the PC of the write instruction are displayed as well as the timestamps.
- Click on RUN (maybe a couple of times) to start this again. You can modify the value of msec in the Watch or Memory window to get to the trigger value of 0x12 faster.
- There are other types of expressions you can enter and they are detailed in the Help button in the Breakpoints window.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10080048H	0000025H	14000E7CH		680202895	9.44726243
Data Write			10080048H	0000024H	14000E7CH		680274895	9.44826243
Data Write			10080048H	0000023H	14000E7CH		680346895	9.44926243
Data Write			10080048H	0000022H	14000E7CH		680418895	9.45026243
Data Write			10080048H	0000021H	14000E7CH		680490895	9.45126243
Data Write			10080048H	0000020H	14000E7CH		680562895	9.45226243
Data Write			10080048H	000001FH	14000E7CH		680634895	9.45326243
Data Write			10080048H	000001EH	14000E7CH		680706895	9.45426243
Data Write			10080048H	000001DH	14000E7CH		680778895	9.45526243
Data Write			10080048H	000001CH	14000E7CH		680850895	9.45626243
Data Write			10080048H	000001BH	14000E7CH		680922895	9.45726243
Data Write			10080048H	000001AH	14000E7CH		680994895	9.45826243
Data Write			10080048H	0000019H	14000E7CH		681066895	9.45926243
Data Write			10080048H	0000018H	14000E7CH		681138895	9.46026243
Data Write			10080048H	0000017H	14000E7CH		681210895	9.46126243
Data Write			10080048H	0000016H	14000E7CH		681282895	9.46226243
Data Write			10080048H	0000015H	14000E7CH		681354895	9.46326243
Data Write			10080048H	0000014H	14000E7CH		681426895	9.46426243
Data Write			10080048H	0000013H	14000E7CH		681498895	9.46526243
Data Write			10080048H	0000012H	14000E7CH		681570895	9.46626243

- When finished, click on STOP if the program is running and delete this Watchpoint by selecting Debug and select Breakpoints and select Kill All. Select Close.
Note: Selecting Debug and Kill all Breakpoints will not delete Watchpoints.

- Leave Debug mode.

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

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

TIP: The checkbox beside the expression in Current Breakpoints as shown above allows you to temporarily unselect or disable a Watchpoint without deleting it.

3) RTX RTOS Example RTX_Blinky:

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

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

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


RTX now comes with a BSD type license. There are no licensing or product fees or royalties payable with RTX. RTX is easy to implement and is a full feature RTOS. It is not crippled or limited in any way. Source code is provided with RTX. See www.keil.com/rl-arm/kernel.asp.

Software:

1. Obtain the RTX example program files for Blinky.uvproj created for the NGX board. Check www.keil.com/nxp
2. Copy it to or look for it in C:\Keil\ARM\Boards\NXP\Xplorer4330\RTX_Blinky

Compile and RUN Blinky:

3. Open the project C:\Keil\ARM\Boards\NXP\Xplorer4330\RTX_Blinky\Blinky.uvproj.

4. Compile the source files: Click on the Rebuild icon. 

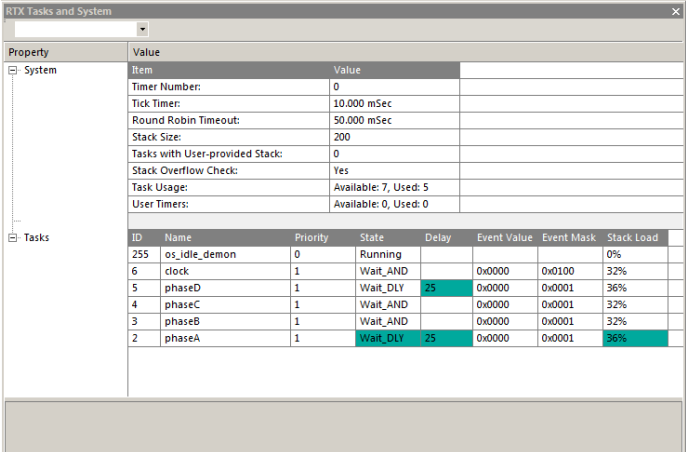
5. Program the Flash by clicking on the Load icon.  Progress will be indicated at the lower left of your screen.

6. Enter Debug mode. 

7. Click on the Run icon. 

8. The blue LED blinks with the clock task (ID 6) and the yellow when phaseA task (ID 2) runs.

9. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. These values are updated in real-time using the same read write technology as used in the Watch and Memory windows.



Property	Value
System	
Item	Value
Timer Number:	0
Tick Timer:	10,000 mSec
Round Robin Timeout:	50,000 mSec
Stack Size:	200
Tasks with User-provided Stack:	0
Stack Overflow Check:	Yes
Task Usage:	Available: 7, Used: 5
User Timers:	Available: 0, Used: 0

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
6	clock	1	Wait_AND		0x0000	0x0100	32%
5	phaseD	1	Wait_DLY	25	0x0000	0x0001	36%
4	phaseC	1	Wait_AND		0x0000	0x0001	32%
3	phaseB	1	Wait_AND		0x0000	0x0001	32%
2	phaseA	1	Wait_DLY	25	0x0000	0x0001	36%

Note Blinky.c has four tasks phaseA through phaseD plus the clock task as shown in the screen. These tasks are visible in Blinky.c tab. Task 1 phaseA starts near line 64.

Each task toggles its own global variable. These are named phasea through phased and they are declared starting near line 29. We will enter these into the Watch 1 and Logic Analyzer.

Important TIP: View/Periodic Window Update must be selected for real-time updates, otherwise data values will not be changed until the program is stopped !

10. Open Debug/OS Support and select Event Viewer. There is probably no data displayed in this window because SWV is not yet configured. The Event Viewer needs SWV operational. We will do this on the next page.

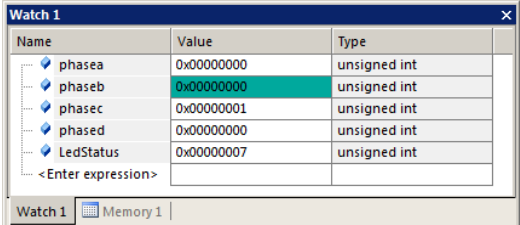
Using the Watch Window to display global variables phasea through phased:

11. With Blinky running, enter the four variables phasea through phased plus LedStatus into Watch 1.

12. These will start to update in real time as shown here:

Note that it appears that LedStatus is always the value of 7 but examining the source code will show that this is not so: the numbers 1, 2, 3 and 4 are also displayed (see #define LED_A through LED_D at line 21 in Blinky.c). LedStatus is changed in the LED_on and LED_off functions.




You will learn how to find out how to determine the timing relations of the values of LedStatus later using the Logic Analyzer.



Name	Value	Type
phasea	0x00000000	unsigned int
phaseb	0x00000000	unsigned int
phasec	0x00000001	unsigned int
phased	0x00000000	unsigned int
LedStatus	0x00000007	unsigned int

4) RTX Viewer: Configuring Serial Wire Viewer (SWV):

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

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

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

5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 120 MHz and select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown here:
8. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer. It is slightly intrusive.
9. Click on OK twice to return to μ Vision.
10. Select File/Save All.

The Serial Wire Viewer is now configured.

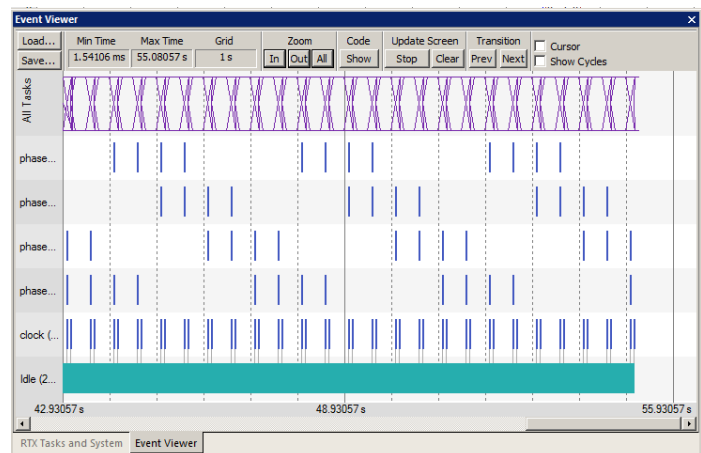
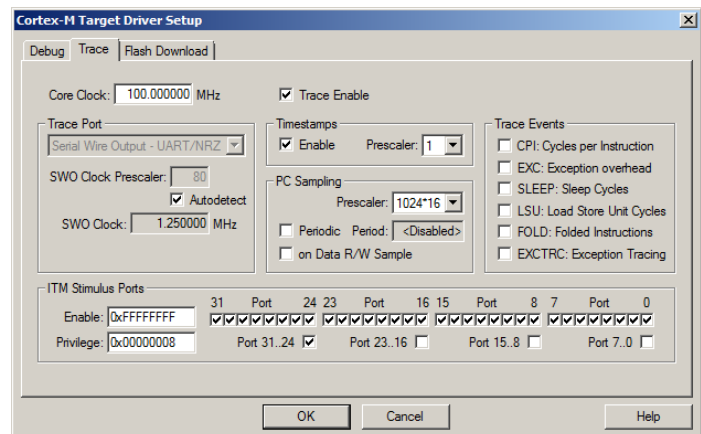
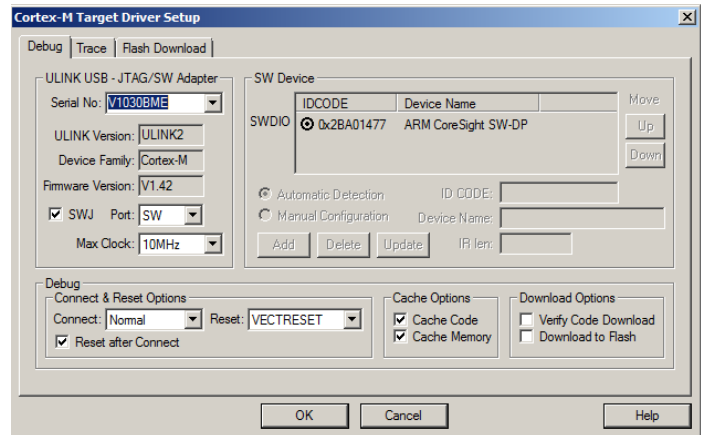
11. Enter Debug mode and click on RUN.
12. Select “RTX Tasks and System” tab: note the display is being updated.
13. Click on the Event Viewer tab.
14. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 0.2 second by clicking on the ALL, and then the In and Out icons.

TIP: If Event Viewer doesn't work, open up the Trace Records window and confirm there are good ITM 31 frames present. Is the Core Clock frequency correct ?

TIP: There is an enormous amount of SWV information coming out the SWO pin. If you have trouble getting the Event Viewer to work and the Trace Records window contains ITM frames other than 31, this could be from overloading. Remove all or some of the variables in the LA window and/or unneeded features in the Trace config window to lessen the pressure on the SWO pin.


Cortex-Mx Alert: μ Vision will update all RTX information in real-time on a target board due to its read/write capabilities as already described. The Event Viewer uses ITM and is slightly intrusive.

The data is updated while the program is running. No instrumentation code needs to be inserted into your source. You will find this feature very useful ! Remember, RTX with source code is included with all versions of MDK.



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

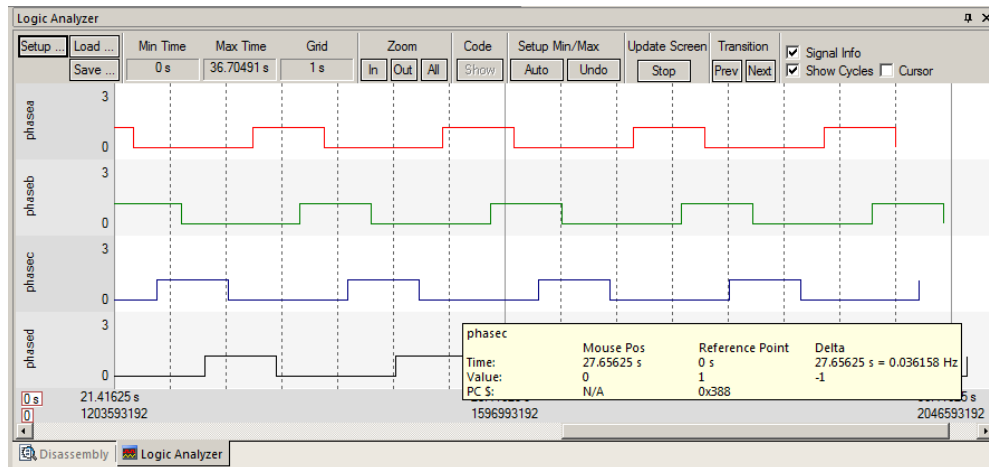
μ Vision has a graphical Logic Analyzer window. Up to four variables can be displayed in real-time using the Serial Wire Viewer. RTX_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.

1. The RTX_Blinky program can be running for these steps.
2. Open the Logic Analyzer (LA) window if not already open. 
3. If there are any variables present in the LA, click on Setup and use Kill All to remove them.

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 their addresses and enter them into the Logic Analyzer. You can enter physical addresses into the LA. Here is an example: *((unsigned long *)0x20000000)

Enter the Variables into the Logic Analyzer:

4. Click on the Blinky.c tab. Block **phasea**, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
5. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release.
6. Repeat for **phaseb**, **phasec** and **phased**. These variables will be listed on the left side of the LA window as shown. Note, you could have right clicked on a variable and add it to the LA. Now we have to adjust the scaling.
7. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
8. Click on Close to go back to the LA window.
9. Using the All, OUT and In buttons set the range to 1second or so. Move the scrolling bar to the far right if needed.
10. Select Signal Info and Show Cycles. Click to mark a place move the cursor to get timings. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phasec:
It is easier to do this when the program is stopped.

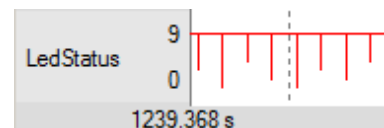


TIP: You can view signals that exist mathematically in a variable and not available for measuring in the outside world.

LedStatus:

LedStatus is a global variable declared around line 27. Its value changes in the functions LED_on and LED_off according to which task calls these two functions. We will also show this in the Logic Analyzer.

1. Click on Setup in the LA and delete phased to make room for the variable LedStatus. Note: you might not have to delete phasea, LPC4300 might be able to handle five variables in the LA.
2. Enter the variable LedStatus and set its Max to 0x9.
3. Click on Close.
4. RUN the program and adjust Zoom to display LedStatus as shown here:
5. Note most of the time LedStatus equals 7 and the spikes represent the other values of 1, 2, 3 and 4. This is why it does not change in Watch 1.

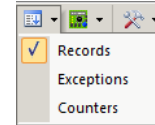


This demonstrates the ability of CoreSight to deliver very useful debugging tools.

4) Trace Records and Exception Windows:

µVision offers real-time viewing of various data tracing including listing of interrupts.

1. With Blinky still running from the previous page: select Debug/Debug Settings and select the Trace tab. Select EXCTRC: Exception Tracing and select OK twice.
2. Start the program again by clicking on RUN.
3. Open the Trace Records window:
4. The following window opens:
5. Displayed are ITM, Data Write and Exception frames.
6. Exception 15 is the SYSTICK timer and 11 is SVCcall exception.
7. **Entry:** when the exception or interrupt is entered.
8. **Exit:** when the exception or interrupt exits.
9. **Return:** when all exceptions or interrupts exit. This indicates no tail chaining is occurring.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry	15						307356631237	1707.53684021
Exception Exit	15						307356650171	1707.53694539
Exception Return	0						307356650809	1707.53694894
Exception Entry	15						307357631179	1707.54239544
Exception Entry	15						307357650113	1707.54250063
Exception Return	0						307357650751	1707.54250417
Exception Entry	15						307358631175	1707.54795097
ITM	31			06H			307358654267	1707.54807926
Exception Exit	15						307358661012	1707.54811673
Exception Return	0						307358661541	1707.54811967
Data Write			10000008H	00000007H			307358663360	1707.54812978
Exception Entry	11						307358667251	1707.54815139
ITM	31			FFH			307358679699	1707.54822055
Exception Exit	11						307358687427	1707.54826348
Exception Return	0						307358688672	1707.54827040
Exception Entry	15						307359631968	1707.55351093
Exception Entry	15						307359652141	1707.55362301
Exception Return	0						307359652779	1707.55362655
Exception Entry	15						307360631172	1707.55906207
Exception Exit	15						307360654117	1707.55918954

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.

10. Right-click in the Trace Records and unselect Exceptions. This is to filter out selections you don't want to see. Double-click anywhere in the window to clear it.
11. Unselect ITM frames in the same fashion.
12. Open Debug/Debug Settings, unselect EXCTRC and select On Data R/W Samples (to turn on instruction address).
13. Click on OK twice and click on RUN to start the program.
14. The Data Write shows the address the data value was written to by the (optional) instruction's address. Data Write frames are placed here when they are entered in the Logic Analyzer.

TIP: This data comes out a one pin port (SWO) and is easily overloaded. It is better to select only those frames you want in the Trace Config window than filter them out afterwards in the Trace Records window.

15. Open the Exceptions window: The window below opens:
16. Each exception is listed with an occurrence count and various timings.
17. When you are finished, please stop the program and exit debug mode.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			10000008H	00000007H	14000C46H		1300755522	7.22641957
Data Write			10000008H	00000007H	14000C4CH		1308663665	7.27035369
Data Write			10000008H	00000001H	14000C4CH		1350709855	7.50394364
Data Write			10000000CH	00000000H	14000C2H	X	1350712724	7.50395958
Data Write			10000008H	00000007H	14000C46H		1350736900	7.50409389
Data Write			10000008H	00000007H	14000C4CH		1358663662	7.54813146
Data Write			10000008H	00000003H	14000C46H		1400708807	7.78171559
Data Write			10000014H	00000001H	14000D12H	X	1400711684	7.78173158
Data Write			10000008H	00000007H	14000C46H		1400754819	7.78197122
Data Write			10000008H	00000007H	14000C4CH		1408663676	7.82590931
Data Write			10000008H	00000002H	14000C4CH		1450709889	8.05949938
Data Write			10000008H	00000007H	14000C46H		1450737756	8.05965420
Data Write			10000008H	00000007H	14000C4CH		1458663658	8.10368699
Data Write			10000008H	00000004H	14000C46H		1500708792	8.33727107
Data Write			10000008H	00000007H	14000C46H		1500755515	8.33753064
Data Write			10000008H	00000007H	14000C4CH		1508663667	8.38146482
Data Write			10000008H	00000003H	14000C4CH		1550709848	8.61505471
Data Write			10000014H	00000000H	14000D26H	X	1550712692	8.61507051
Data Write			10000008H	00000007H	14000C46H		1550737456	8.61520809
Data Write			10000008H	00000007H	14000C4CH		1558663685	8.65924269




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	SVCcall	34066	3.549 s	17.061 us	401.989 us	9.850 us	233.225 ms	0.00331873	1719.77037344
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	309564	34.702 s	105.144 us	182.739 us	5.377 ms	5.454 ms	0.00906652	1719.80350658
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						

TIP: Data Read frames are currently disabled in µVision. Data Write frames are available.

4) USB and Interrupt Example: LPC4330_Xplorer_ExtInt.uvproj

This example demonstrates USB and interrupts. Pressing the button SW2 create an interrupt at one of the general purpose inputs (GPIO0 pin 7) and sends a message out the USB 1 port. This message can be displayed on a serial terminal program on your PC. The firing of the interrupts will be displayed using SWV.

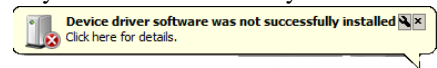
Configure μ Vision and prepare the program:

1. Start μ Vision if not already running. Connect the board as shown. Use J3 USB1 to power the board.
2. Open the project LPC4330_Xplorer_ExtInt.uvproj found here:
C:\Keil\ARM\Boards\NXP\Xplorer4330\LPC4330_Xplorer_ExtInt\Keil
3. Click on the Target Options icon  and select the Debug tab. Delete the .ini file as shown: Click on OK.
4. Compile the source files: Click on the Rebuild icon. 
5. Program the Flash by clicking on the Load icon.  Progress will be indicated at the lower left of your screen.
6. At this time, Windows will detect the LPC4330 USB port and if the driver had not previously been installed, it will try to install a USB driver on your PC. A standard USB double tone will probably sound.



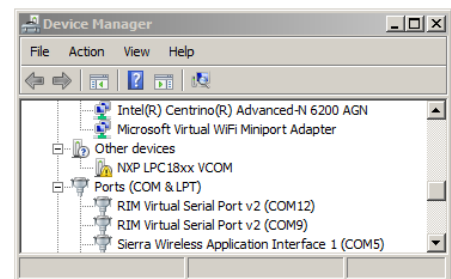
Configure USB on your PC:

7. If the USB driver install is not automatically done, you must do it manually. This will depend on the version of Windows you are using on your PC. If the USB drivers have been installed in a previous run of this exercise, jump to the next page.
8. If a notification appears that the USB driver was successfully installed, jump to the next page.
9. If a notification says the Device driver software was not successfully installed: you must now manually install it.

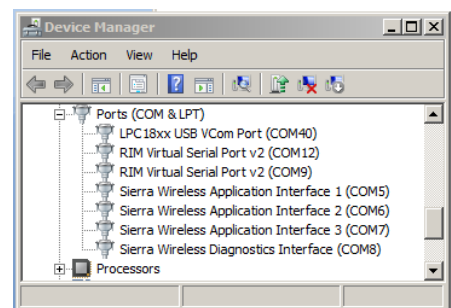
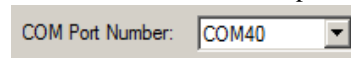


Install USB on your PC: Locate hardware in Windows Device Manager:

1. Open Windows Control Panel appropriate for your version of Windows.
2. Select Device Manager. The exact steps will depend on your version of Windows.
3. When the NXP hardware has been detected but the USB driver is not installed, it will show under Other Devices:
4. If the driver is installed, it will appear under Ports (COM & LPT)
5. Right click on NXP LPC18xx VCOM and select Update Driver Software...
6. Select Browse My Computer... and select the Browse icon and locate:
C:\Keil\ARM\Boards\NXP\Xplorer4330\LPC4330_Xplorer_UsbVcomLib
7. Select Next: to complete this task.
8. If a window opens and says Windows cannot verify this driver, select install it anyway.
9. When the driver has successfully been installed, this fact will be displayed. Click on Close.
10. The LPC4330 will now be displayed under Ports (COM & LPT) as shown here: Note the COM port number – you will need to input this for your terminal program. Close this window.





TIP: Some terminal programs are unable to handle high COM port numbers. To change this, right click on the NXP driver in Device Manager and select Properties. Select Port Settings and then the Advanced icon. In the COM port select a new COM port that is not used.



The program is actually running on the LPC4330. The blue and red led should be on. Press SW2 and the blue led will cycle. This shows everything is normal. Press RESETSW1 once or twice to get to this state.

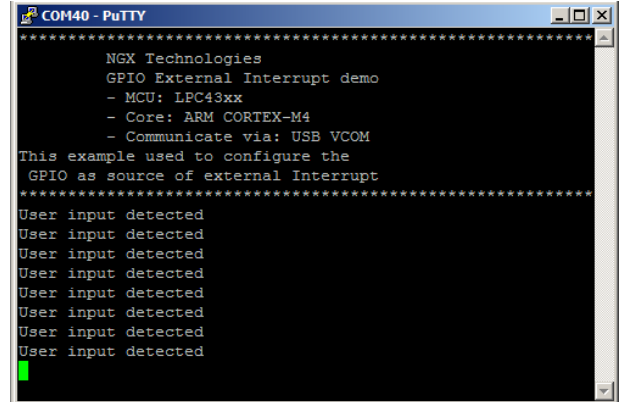
Cycle the power to the NGX board for the next steps.

Run the program and configure a Terminal program on your PC:

1. If you have a terminal program running for this tutorial – please exit it now to ensure a clean start.
2. Enter Debug mode.  This stops the program running on the NGX board.
3. Start the program by clicking on the Run icon.  The USB double tone will sound indicating a connection.
TIP: USB is only active when the program is running. At this point, the LPC4330 is ready to communicate with your PC over USB1. The blue and red LEDs will be on.
4. Open your terminal program: speed is 115200 baud using the COM port for your particular setup. I used PuTTY but any common terminal program will work. See Step 10 on the previous page if you do not know the COM port.
5. Press SW2 once and the blue LED will go out and some text will appear on the screen as shown below.
6. Press it several more times and the blue LED will cycle and more text will appear as shown here:

7. **Trouble ?:** Cycle the power to the NGX board and ULINK and start over. Remember to exit the terminal every time you RESET the board or exit/enter debug mode in µVision. Terminal programs are usually not able to track such events. Getting a Connect Error when starting the terminal ? Cycle the NGX board power. The blue LED must be on after a repower.

Try running the program without the ULINK attached. Everything will still work as the program is programmed in Flash and is configured to start from RESET.








```

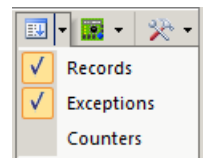
*****
NGX Technologies
GPIO External Interrupt demo
- MCU: LPC43xx
- Core: ARM CORTEX-M4
- Communicate via: USB VCOM
This example used to configure the
GPIO as source of external Interrupt
*****
User input detected
User input detected
User input detected
User input detected
User input detected
User input detected
User input detected
User input detected
User input detected
User input detected

```

What is happening: When you press SW2 which is connected to GPIO Port 0 pin 7, an interrupt is generated. A message is sent out the USB1 port to the PC where it is displayed on a terminal.

Serial Wire Viewer:

1. STOP the program . Exit the terminal. Exit Debug mode. 
2. Click on the Target Options icon  and select the Debug tab. Select Settings: and confirm Port: is set to SW.
3. Select the Trace tab and set Core Clock: to 60 MHz and select the Trace Enable box.
4. Unselect Periodic and select EXCTRC. Click OK twice to return to the main menu. The SWV is now configured.
5. Cycle the power to the NGX board and enter Debug mode. 
6. Open both the Trace Records and Exceptions windows. Double-click in each to clear them.
7. Click on RUN . Open your terminal program.
8. Each time you press SW2, two interrupt sequences occur and are displayed in the trace windows as shown below:



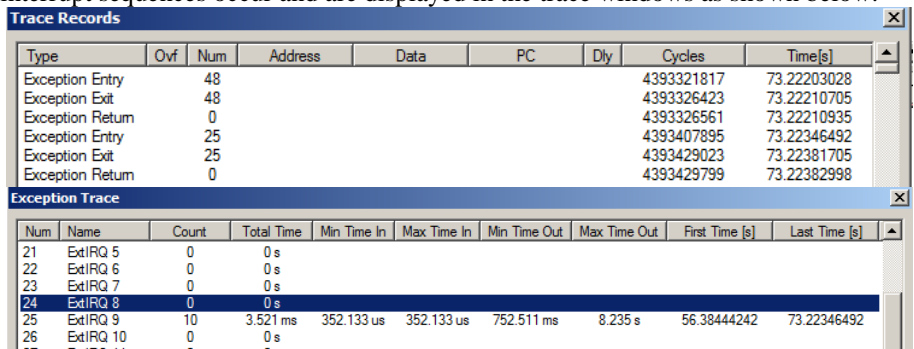
ExtIRQ 32(48) is the GPIO0 interrupt and is created when you press SW2.

ExtIRQ 9(25) is USB1 interrupt and occurs in the USB write functions.

Entry: when the exception or interrupt is entered.

Exit: when the exception or interrupt exits.

Return: when all exceptions or interrupts exit. This indicates no tail chaining is occurring. SWV makes it easy to see when interrupts are happening and at what times.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry	48						4393321817	73.22203028
Exception Exit	48						4393326423	73.22210705
Exception Return	0						4393326561	73.22210935
Exception Entry	25						4393407895	73.22346492
Exception Exit	25						4393429023	73.22381705
Exception Return	0						4393429799	73.22382998

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						
24	ExtIRQ 8	0	0 s						
25	ExtIRQ 9	10	3.521 ms	352.133 us	352.133 us	752.511 ms	8.235 s	56.38444242	73.22346492
26	ExtIRQ 10	0	0 s						
27	ExtIRQ 11	0	0 s						

5) Dual Core MBX Example:

Running and controlling the Cortex-M4 and Cortex-M0 cores in LPC4300:



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

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



MDK controls both processors by using two instances of μ Vision running: one controlling the M4 and the other the M0. Either the ULINK2 or the ULINK pro can be used. This exercise will use the ULINK2 or ULINK-ME. Serial Wire Viewer will not work because we must use JTAG and not SWD in order to access each core separately. A ULINK pro can send the Cortex-M4 SWV data out the 4 bit trace port. SWV frames will then be available for the Cortex-M4 but not the Cortex-M0.

This exercise will store the programs in the processor RAM. We will not use the SPIFI Flash this time.

1) Open and compile the Cortex-M0 project: (you must compile the Cortex-M0 project first)

- 1) Connect ULINK2 or ULINK-ME to the NGX board and start μ Vision if it is not already running. 
- 2) Select the project by opening LPC4330_Xplorer_CM0.uvproj found in:
C:\Keil\ARM\Boards\NXP\Xplorer4330\LPC4330_Xplorer_DualCore\LPC4330_Xplorer_CM0
- 3) Click on the Rebuild icon.  It will compile with no errors or warnings.
This project provides the file CM0_image.c to the M4 project.

2) Open and compile the Cortex-M4 project:

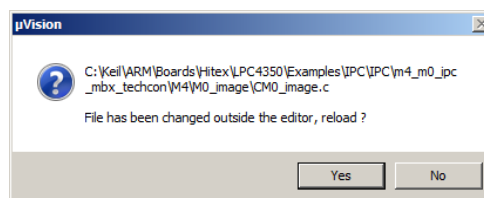
- 1) Open a second instance of μ Vision.  This instance will operate the Cortex-M4 and the first one the Cortex-M0.
- 2) Select the project by opening LPC4330_Xplorer_CM4.uvproj found in:
C:\Keil\ARM\Boards\NXP\Xplorer4330\LPC4330_Xplorer_DualCore\LPC4330_Xplorer_CM4
- 3) Click on the Rebuild icon. 
- 3) You have successfully compiled both the M4 and M0 source files.
- 4) Position the two instances of μ Vision conveniently on your screen. If you have double screen setup: μ Vision allows you to have one instance on each screen.

TIP: Remember when you make changes to your dual core projects, you must compile the Cortex-M0 project first. This is because the output of the M0 project is used as an input to the M4 project.

TIP: If you have the file CM0_image.c or its tab visible in the editing window when you compile the M0 source files, you will get this notice:


μ Vision has updated this file and it will not allow an old version to be displayed without warning you.

Please click on the Yes box if you see this window



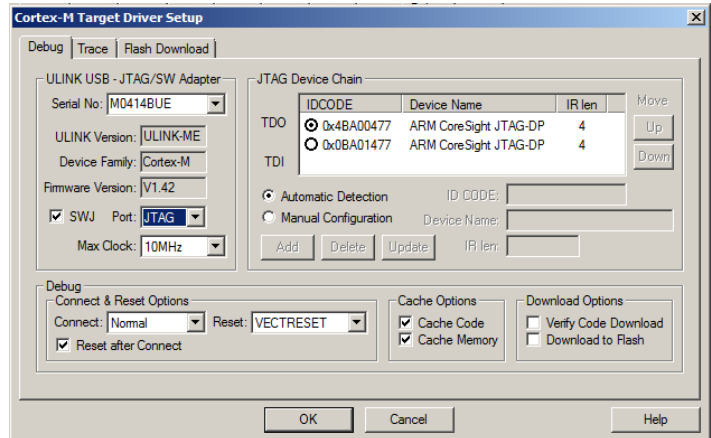
3) Configuring the ULINK2/ME:

Cortex-M4 core:

- 1) Select the M4 instance of μ Vision by bringing it into focus by clicking inside it and therefore making it active.
- 2) Select Options for Target  or ALT-F7 and select the Debug tab and this window opens up:
- 3) Select ULINK2/ME Cortex Debugger:
- 4) Click on the Settings: box on the right side. This window opens up:
- 5) In the RESET: box, select VECTRESET.


TIP: This VECTRESET setting is very important. If the program ends up in the hard fault vector when you run it, check that this box is set correctly.

- 6) Select SWJ.
- 7) In the Port: box, select JTAG.
- 8) Note two entries in the JTAG Device Chain box:
0x4BA00477 is the Cortex-M4
0x0BA01477 is the Cortex-M0
- 9) Select the Cortex-M4 as shown.
- 10) Click on OK twice to return to the main menu.
- 11) Select File/Save All.



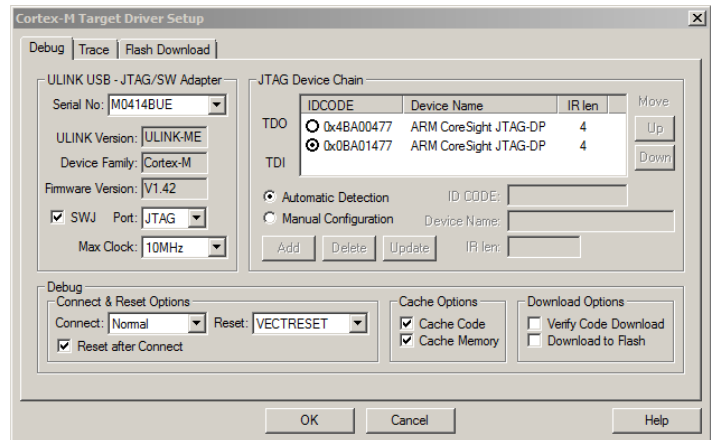
At this point, you have successfully connected to the Cortex-M4 and compiled the source files for both cores.

Cortex-M0 core:

- 1) Select the M0 instance of μ Vision by bringing it into focus by clicking inside it.
- 2) Select Options for Target  or ALT-F7 and select the Debug tab and this window opens up:
- 3) Select ULINK2/ME Cortex Debugger:
- 4) Click on the Settings: box on the right side. This window opens up:
- 5) In the RESET: box, select VECTRESET.

TIP: This VECTRESET setting is very important. If the program ends up in the hard fault vector when you run it, check that this box is set correctly.

- 6) Select SWJ.
- 7) In the Port: box select JTAG.
- 8) Note two entries in the JTAG Device Chain box.
0x4BA00477 is the Cortex-M4
0x0BA01477 is the Cortex-M0
- 9) Select the Cortex-M0 as shown.
- 10) Click on OK twice to return to the main menu.
- 11) Select File/Save All.




Next you will run the program.






4) Running the Program:

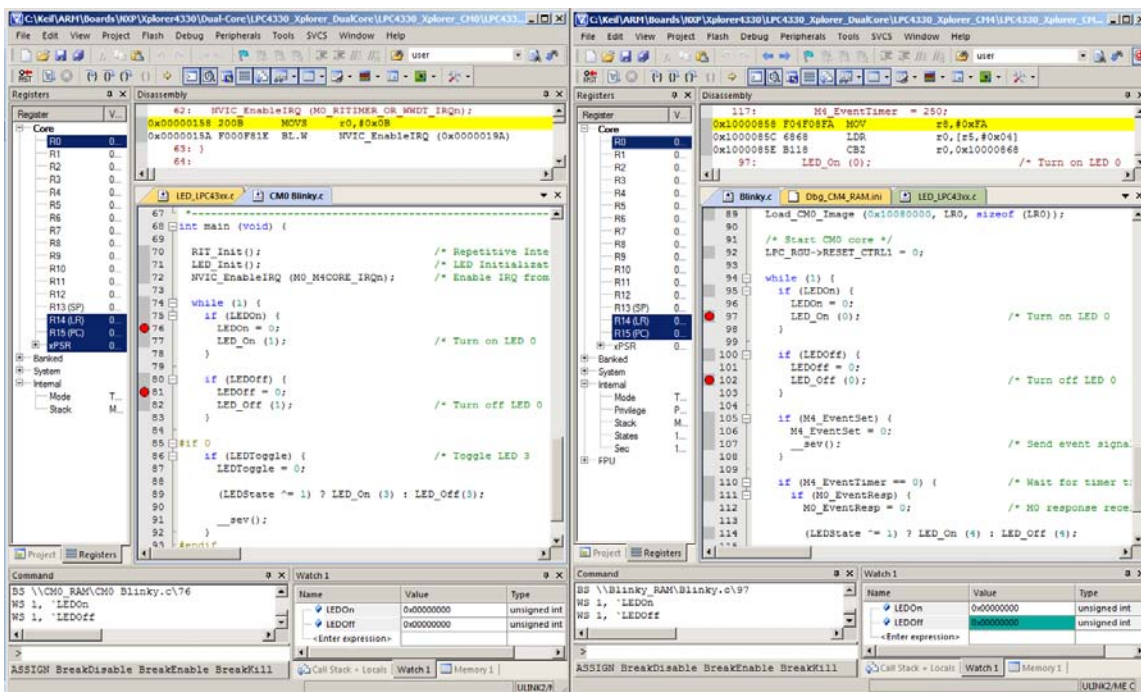
The Cortex-M4 blinks the blue LED and the Cortex-M0 blinks the yellow LED.

Cortex-M4 core:

- 1) In the M4 instance, enter Debug mode by clicking on the debug icon .

TIP: You do not program the RAM the same way as with the Flash. The file Dbg_CM4_RAM.ini initializes the initial stack and PC and some other items and loads the program and runs to main(). You do not use the Load icon for RAM.

- 2) Click on the RUN icon. 
- 3) Note two LEDs will now be blinking.
The yellow LED is controlled by the M0 and the blue LED by the M4.
- 4) Click on STOP .
- 5) Note the blue LED stops blinking. The yellow one of the right continues to blink.
This means the M4 is stopped but the M0 is still running. When you started the M4, you also started the M0 but you are unable to stop the M0 with the M4 instance of μ Vision. The M0 instance of μ Vision is not doing anything at this time because to do so it must be in debug mode and it is not.
- 6) Click on the RUN icon. 
- 7) In the M0 instance, enter Debug mode by clicking on the debug icon .
- 8) Note the yellow LED stops blinking. This is because the M0 instance of μ Vision is now in debug mode and has halted the Cortex-M0 processor.
- 9) Click on the RUN icon.  Note both LEDs are blinking.
- 10) Click on STOP on the M0 instance and the yellow LED stops blinking.
- 11) Click on STOP on the M4 instance and the blue LED stops blinking.
- 12) Click on RUN on the M0 instance and the yellow LED starts blinking.
- 13) Try various combinations of the two instances and note you are now able to independently control both processors.
- 14) This screen is how I arranged the two instances of μ Vision. The M0 is on the left. Two screens would be much more convenient. μ Vision does have a dual screen capability.




5) Breakpoints in main():

- 1) Start both processors by clicking on RUN in each instance. Confirm both LEDs are blinking.
- 2) In the M4 instance in the file Blinky.c, click in the left margin where there is a gray block (signifies there is assembly language at this C source line) to set a hardware breakpoint. Select lines LEDOn = 0; (line 96) and LEDOff = 0; (line 101). A red circle will appear and presently the program will stop. This is shown in the screen on the previous page in the M4 instance.
- 3) Note the M4 LED has stopped blinking while the M0 is still blinking.
- 4) In the M0 instance, in the the file CM0 Blinky.c, similarly create a hardware breakpoint at the these source lines. Select lines LEDOn = 0; (line 76) and LEDOff = 0; (line 81). A red circle will appear and presently the program will stop. This is shown in the screen on the previous page in the M4 instance.

TIP: It is possible your line numbers will be different due to differing compilation settings.


TIP: Remember, you can set/unset hardware breakpoints in μ Vision while the program is running. A breakpoint does not execute the instruction it is set to. These are “no-skid” hardware breakpoints.

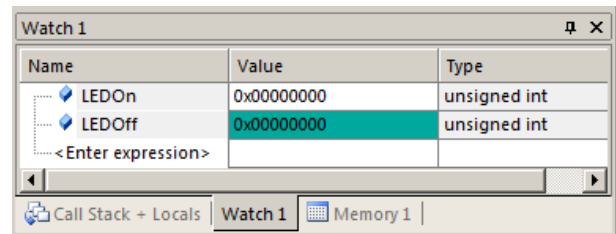
- 5) Start the program with the RUN icon in each instance and see you can enter each breakpoint in order. 
Depending on where you are in the program, LEDs will be on or off appropriately.

6) Watch Window:

- 1) In each instance of μ Vision, open a Watch window by selecting View/Watch Windows/Watch 1. Click on <Enter Expression> and enter the global variables LEDOff and LEDOn. These variables are displayed and are updated in real time as shown below.

You can also enter a variable by finding each variable in the source window Blinky.c and right-click it and select “Add “LEDOff” to and then select Watch 1 in each instance. Repeat for LEDOn.

- 2) Each instance should have this window: 
- 3) Select View/Periodic Window Update and make sure this is enabled.
- 4) As you run each program in each instance, you will see LEDOff and LEDOn updated with different numbers. You will be able to confirm, with different number patterns, that indeed there are two cores running independently in this example as they cycle through the breakpoints.
- 5) You can also enter these variables into a memory window.
- 6) You can also set a Watchpoint on it.
- 7) You cannot use the Logic Analyzer. See (SWV) below.
- 8) When you are done, stop both instances of μ Vision, leave debug mode and close the M0 instance of μ Vision.



TIP: Make sure View/Periodic Window Update is selected. Otherwise variables will be updated only when the program is stopped.

TIP: If you have trouble entering counter into the Watch window while the program is running, try selecting it from View/Symbol Table. Sometimes static variables must be fully qualified.

6) What does a ULINKpro offer you ?

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

Note the Xplorer board does not have the required 20 pin compact connector for ETM. Many other boards do.



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

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.








Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x0000078A	EORS r0,r5,r6	if (ad_val ^ ad_val) { /* AD value changed */ }
	X: 0x0000078E	BEQ 0x000007A0	
	X: 0x000007A0	LDR r0,[pc,#56] ; @0x000007DC	if (clock_1s) {
	X: 0x000007A2	LDRB r0,[r0,#0x00]	
	X: 0x000007A4	CBZ r0,0x000007B6	
4.587 566 340 s	X: 0x000007B6	B 0x00000762	while (1) { /* Loop forever */ }
	X: 0x00000762	LDR r0,[pc,#96] ; @0x000007C4	if (AD_done) { /* If conversion has finished */ }
	X: 0x00000764	LDRB r0,[r0,#0x00]	
	X: 0x00000766	CBZ r0,0x0000078A	
	X: 0x0000078A	EORS r0,r5,r6	if (ad_val ^ ad_val) { /* AD value changed */ }

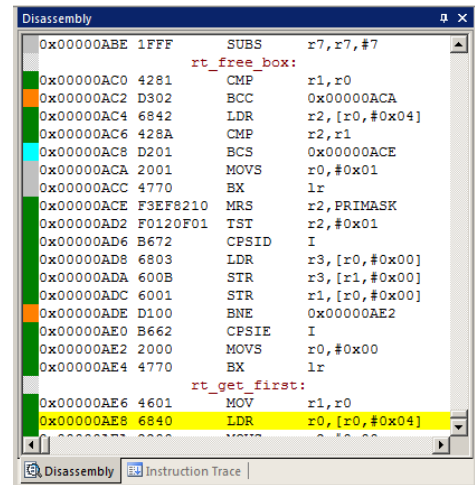
Serial Wire Viewer frames are stored along with ETM frames. Shown here in red is a data write interleaved with assembly instructions.

Filtering displays only those types you want to see.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x000002E4	LDM r0!,{r3}	
D 0.000 109 540 s	W: 0x10000022	0x00000000	X: 0x000002E8
0.000 109 700 s	X: 0x000002E6	SUBS r2,r2,#4	
	X: 0x000002E8	STM r1!,{r3}	
	X: 0x000002EA	CMP r2,#0x00	
0.000 109 720 s	X: 0x000002EC	* BNE 0x000002E4	
0.000 109 740 s	X: 0x000002EE	BX lr	

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.

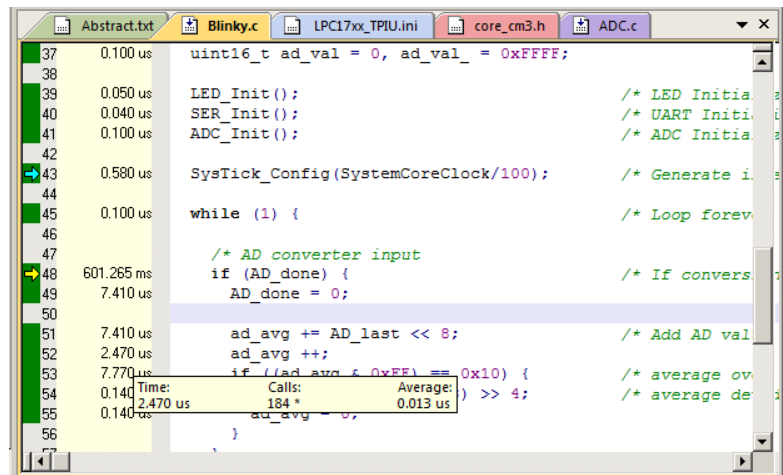
-  1. Green: this assembly instruction was executed.
-  2. Gray: this assembly instruction was not executed.
-  3. Orange: a Branch is always not taken.
-  4. Cyan: a Branch is always taken.
-  5. Light Gray: there is no assembly instruction here.
-  6. RED: Breakpoint is set here.
-  7. Yellow arrow: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



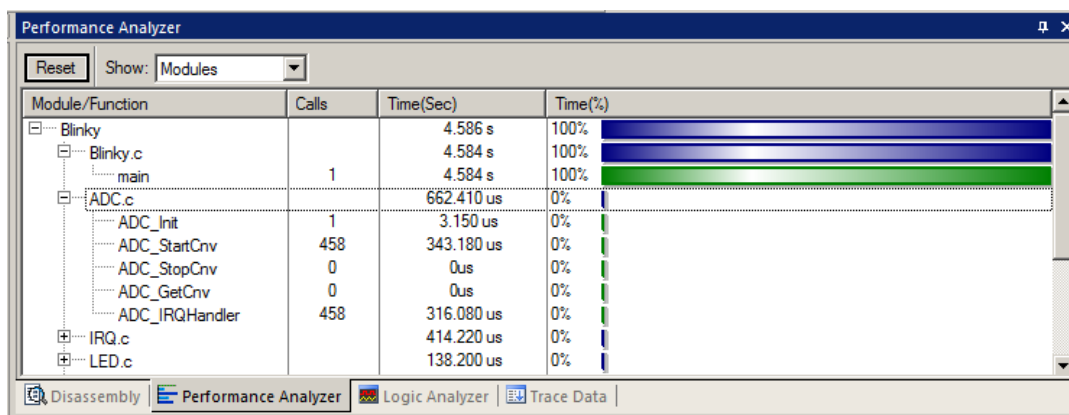
3. **Execution Profiling:** How long did it take for this function or set of assembly instructions to execute? How many times did they execute? With a configurable resolution from many C source lines down to individual instructions, Instruction profiling gives you an accurate indication of program timings.

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:



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:



7) Keil Products:

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

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.

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.
- ULINKpro - \$1,395 – Cortex-Mx SWV & ETM trace

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.



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

