Infineon XMC4000: Cortex™-M4 Lab

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

For the Hitex XMC-HiLight board with ULINK-ME™ Version 1.0 Robert Boys bob.boys@arm.com

Introduction:

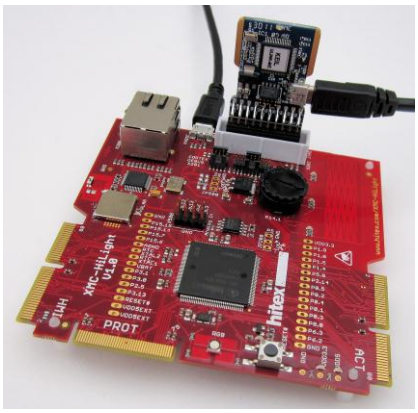
For the latest version of this document: www.keil.com/appnotes/docs/apnt_231.asp

The purpose of this lab is to introduce you to the Infineon Cortex™-M4 processor family using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) and ETM trace on the XMC-HiLight evaluation board from Hitex. This board comes with a Keil ULINK-ME debug adapter. At the end of this tutorial, you will be able to confidently work with Cortex-M4 processors and MDK. We provide a DSP example.

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 a commercial 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-M3 and Cortex-M4 users:



1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is a turn-key product with included examples and is easy and fast to get running.
2. Serial Wire Viewer and ETM trace capability is included. A full feature Keil RTOS called RTX is included with MDK and includes source code. It has 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.
4. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™ or Segger J-Link (black case, version 6 or later). µVision supports CMSIS-DAP.
5. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
6. Keil also supports Infineon 8051 and C166 processors. See www.keil.com/dd for the complete list.

This document details these features:

1. Serial Wire Viewer (SWV) with ULINK2, ULINK-ME and ULINK*pro*. Plus, ETM Trace using ULINK*pro*.
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.
5. A DSP example with variables displayed using Serial Wire Viewer.

Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (includes 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 or the Trace Port on X201.

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 the ULINK*pro*.

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 requires a special debugger adapter such as 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. The Hitex PowerScale uses ETM trace to determine power consumption in relation to instructions with a Keil ULINK*pro*: www.hitex.com/powerscale

Copyright © 2012 ARM Ltd. All rights reserved

1. **Infineon boards, MDK Install, Useful Definitions** 3

Part A: Connecting and Configuring to the target board:

1. **Connecting ULINK2, ULINK-ME or ULINK*pro* to the HiLight board:** 4
2. **ULINK2, ULINK-ME or ULINK*pro* Configuration:** 5

Part B: Blinky Example Programs using a ULINK2 or ULINK-ME:

1. **Blinky Example Program using the XMC4000 and ULINK2 or ULINK-ME:** 6
2. **Hardware Breakpoints:** 6
3. **Call Stack + Locals Window** 7
4. **Watch and Memory Windows:** 8
5. **Configuring the Serial Wire Viewer (SWV):** 9
    a. For ULINK2 or ULINK-ME: 9
    b. For ULINK*pro*: 10
6. **Using the Logic Analyzer (LA) with ULINK2 or ULINK-ME:** 11
7. **Another use of the Logic Analyzer:** 12
8. **Watchpoints: Conditional Breakpoints** 13
9. **RTX_Blinky example program with Keil RTX RTOS:** 14
10. **RTX Kernel Awareness using Serial Wire Viewer (SWV):** 15
11. **Logic Analyzer Window: Viewing Variables in real-time in a graphical format:** 16
12. **Serial Wire Viewer (SWV) and how to use it: (with ULINK2 or ULINK-ME)** 17
    a. Data Reads and Writes: 17
    b. Exceptions and Interrupts: 18
    c. PC Samples: 19
13. **ITM (Instruction Trace Macrocell) a printf feature:** 20

Part C: DSP Example

1. **Running the DSP example:** 21
2. **Signal Timings in Logic Analyzer:** 22
3. **RTX Tasks & System:** 22
4. **Event Viewer:** 23
5. **Event Viewer Timing & Systick Timer:** 24

Part D: Using the ULINK*pro* with ETM Trace

1. **Target Selector Box:** 25
2. **Blinky Example with ETM trace:** 26
3. **Code Coverage:** 27
4. **Performance Analysis:** 28
5. **Execution Profiling:** 29
6. **In-the-weeds Example:** 30

Part E: Additional Information:

1. **Cache Flash and Configuring Memory Areas:** 31
2. **Creating your own project from scratch:** 32
3. **Serial Wire Viewer and ETM Summary:** 34
4. **Useful Documents:** 34
5. **Keil Products and contact information:** 35

Copyright © 2012 ARM Ltd. All rights reserved

Five Steps to Get Connected and Configured:

1. Physically connect a ULINK to the HiLight or other target board. Power both of these appropriately with USB.
2. Configure µVision to use a ULINK2, ULINK-ME, ULINK*pro* or a J-Link to communicate with the SWD port.
3. Configure the µVision Flash programmer to program the XMC4000 internal flash memory.
4. If desired, configure the Serial Wire Viewer.
5. If desired, configure the ETM trace with the ULINK*pro*.

XMC4000 processors need a simple ascii .ini file that configures the ETM trace. If you do not intend to use ETM you do not need this file. It is entered in the Options for Target window under the Debug tab. This file is provided in the DSP example.

Software Installation:

Keil MDK Toolchain:

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

Install MDK into the default directory of C:\ for the purposes of this lab. The evaluation version of MDK will compile all the examples used here. The addition of a license number converts the evaluation into a commercial copy of MDK.

This document was written using Keil MDK 4.54 which contains µVision 4. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence.

Debug Adapters:

The ULINK-ME is used in this lab. You can use a ULINK2, ULINK-ME, ULINK*pro* or J-Link (black case, V6 or higher) for this lab. A debugger must be configured in two windows under Options for Target: (1) Debug tab (for JTAG or SWD connection) and (2) Utilities tab (for Flash programming). Instructions are provided later.

The ULINK*pro* adds Cortex-M ETM trace support. It also has faster programming time and better SWV trace performance due to its use of Manchester encoding instead of UART for the SWO port. The XMC4000 has both SWV and ETM trace.

Example Programs:

MDK 4.54 contains two example programs: Blinky and RTX_Blinky. A new example, DSP must be added.

These files can be downloaded from www.keil.com/appnotes/docs/apnt_231.asp. The latest version of this document is also available at this location. Put these two directories in file C:\Keil\ARM\Boards\Hitex\XMC-HiLight\ to create \DSP.

The directory \RL consists of middleware examples. Such middleware is a component of MDK Professional. To run these examples a full license is needed. Please contact Keil sales for a temporary license if you want to evaluate Keil middleware and for the list of supported processors.

Board Support Packages:

MDK provides a BSP for both the XMC-HiLight board and the Infineon Hexagon board kit. These can be found in the C:\Keil\ARM\Boards\Hitex and C:\Keil\ARM\Boards\Infineon directories respectively.

This lab can easily be adapted for use with the Hexagon board.

More information can be found at: www.hitex.com/hexagon and for HiLight: www.hitex.com/XMC-HiLight

More Information and Keil Contacts:

For more information about Keil support for Infineon products please visit www.keil.com/infineon

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

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-M 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.

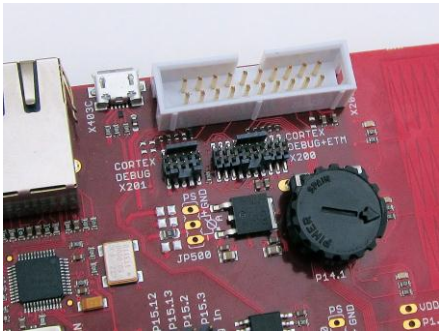
Trace Port: A 4 bit port that ULINK*pro* uses to output ETM frames and optionally SWV (rather than the SWO pin).

ETM: Embedded Trace Macrocell: Provides all the program counter values. Only the ULINK*pro* works with ETM.

Copyright © 2012 ARM Ltd. All rights reserved

Part A)

1) Connecting ULINK2, ULINK-ME or ULINK*pro* to the HiLight board:



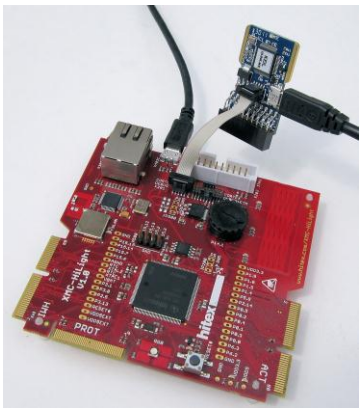
The HiLight is equipped with the new ARM Hi-Density connectors called CoreSight20 and CoreSight10 as well as the legacy 20 pin JTAG connector. All three provide JTAG, SWD and SWO access. These are pictured here:

JTAG connector: Legacy JTAG (X202)

CoreSight10: Cortex Debug (X201)

CoreSight20: Cortex Debug+ETM (X200). (adds ETM)

There is no practical difference in using CoreSight10 or the legacy JTAG connector. Use whichever is most convenient. No ETM signals are available on either of these two connectors. The legacy JTAG connector is being replaced by the CoreSight hi-density connectors to save board real estate.



The CoreSight 20 pin connector X200 provides JTAG, SWD, SWO and adds 4 bit ETM support and connects to the ULINK*pro*.

The Samtec parts are FTSH-105-01-L-DV-K (10 pin) and FTSH-110-01-L-DV-K (20 p).

The XMC4000 processor needs an initialization script to activate ETM.

TIP: Any of the Keil ULINK adapters can connect to any of the three connectors.

Connecting a ULINK2 or ULINK-ME:

Legacy 20 Pin JTAG Connector:

A ULINKME plugged to the HiLight board is pictured on page 1 of this document.

The ULINK-ME is pictured here connected to the 10 pin Hi-Density connector.

20 Pin Connector: Keil does have a 10 pin to 20 pin adapter cable available to connect to this connector. The first 10 pins on the CoreSight20 replicate the CoreSight10 pin.

The second 10 pins on the 20 pin contain the five ETM signals: Four data and one clock.

Note: some call these connectors 9 pin and 19 pin respectively because of one pin often removed for a key.

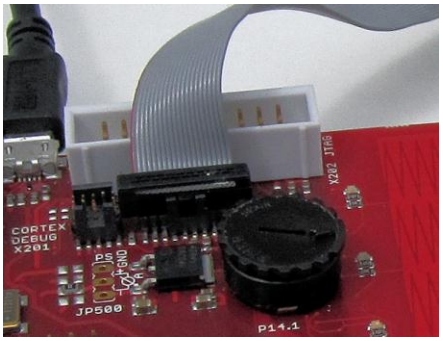
Connecting a ULINK*pro*:

The ULINK*pro* connects to the HiLight board with its Cortex Debug+ETM 20 pin connector or the standard JTAG connector with the supplied adapter.

In order to use ETM trace you must connect the ULINK*pro* to the 20 pin Cortex Debug+ETM (X200) connector as shown directly below:

If you use the legacy 20 pin JTAG connector (X202) you can use JTAG, SWD and SWV but not ETM.

Pictured to the right is a ULINK*pro* with the HiLight.



Copyright © 2012 ARM Ltd. All rights reserved

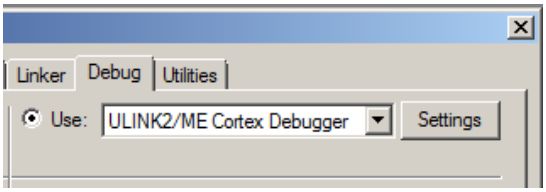
2) ULINK2, ULINK-ME or ULINK*pro* Configuration:

It is easy to select a USB debugging adapter in µVision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are each selected using the Debug and Utilities tabs.

This document will use a ULINK2 or ULINK-ME as described. You can substitute a ULINK*pro* with suitable adjustments.

Serial Wire Viewer (SWV) is completely supported by ULINK2 and ULINLK-ME. They are essentially the same devices electrically and any reference to ULINK2 in this document includes the ME. The ULINK*pro*, which is a Cortex-Mx ETM trace adapter, has all the features of a ULINK2 with the advantages of faster programming time, adds ETM trace support, an enhanced Trace Data window and much faster SWV trace speed.

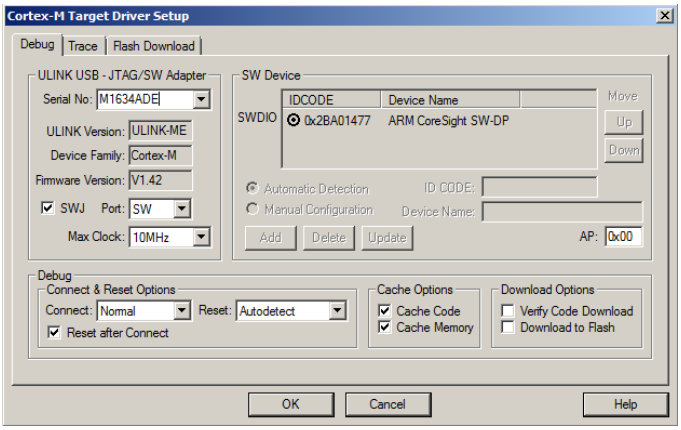
Step 1) Select the debug connection to the target:



1. Assume the ULINK2 is connected to a powered up Hitex target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK-ME is shown connected to the XMC4000 board on page 1.
2. Select Options for Target or ALT-F7 and select the Debug tab. Select ULINK2/ME as shown above:
3. Select Settings and the next window below opens up. This is the control panel for the ULINK2, ULINK-ME (they are the same) and ULINK*pro*. The J-Link has a slightly different configuration window..
4. In **Port:** select SWJ and SW. Serial Wire Viewer (SWV) will not work with JTAG selected. JTAG is not available with the XMC4000 combination as dedicated debug pins TCK and TMS are the only ones pinned out at RESET.
5. 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.

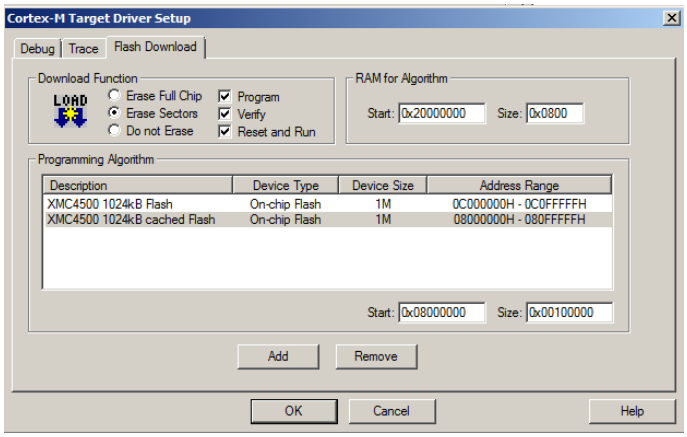
TIP: Max clock is default to 10 MHz. If the Flash programming is not stable or causes an error, select a lower speed such as 1 or 2 MHz.

TIP: You can do regular debugging using SWD. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode. You must activate some GPIO pins to get JTAG operable.



Step 2) Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ULINK2/ME similar to Step 2 above.
8. Click Settings to view and/or select or change the Flash programming algorithm.
9. Select XMC4000 Flash as shown below:
10. Click on OK once.



TIP: To program the Flash every time you enter Debug mode, check Update target before Debugging.

11. Click on OK to return to the µVision main screen. Select File/Save All.
12. You have successfully connected to the XMC4000 processor and configured the Flash programmer.

TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV) and ETM trace. This is described later.

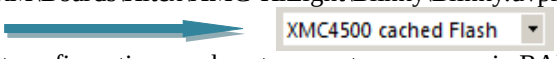
TIP: If you select ULINK or ULINK*pro*, and have the opposite ULINK physically connected to your PC; the error message will say "**No ULINK device found**". This message actually means that µVision found the wrong Keil adapter connected. Select the actual ULINK that is connected and attempt the connection again.

Copyright © 2012 ARM Ltd. All rights reserved

Part B)

1) *Blinky* Example Program using the XMC4000 and a ULINK2 or ULINK-ME:

We will connect a Keil MDK development system using the XMC4000 eval board and a ULINK2 or ULINK-ME. It is possible to use the ULINK*pro* for this example but you must configure it as in ULINK*pro* and µVision Configuration: on page 10. A J-Link (black case V6 or higher) can also be used. This project is pre-configured to use ULINK2 or ULINK-ME.

1. Connect the ULINK-ME as pictured on the first page to the JTAG connector X202 or to Cortex Debug X201.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Hitex\XMC-HiLight\Blinky\Blinky.uvproj.
4. Select "XMC4500 cached Flash" : 

This is where you can create and select different target configurations such as to execute a program in RAM, Flash or Cache Flash. If you want to run in RAM, select XMC4500 PSRAM. See page 31 for info on Cache Flash. You then omit the Load step number 7 and directly enter Debug mode.
5. If you want to use a ULINK*pro* or J-Link, now is the time to configure it. See page 10.
6. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
7. Program the XMC4000 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
8. Enter 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 for RAM operation or the simulator.
9. Click on the RUN icon. Note: you stop the program with the STOP icon.

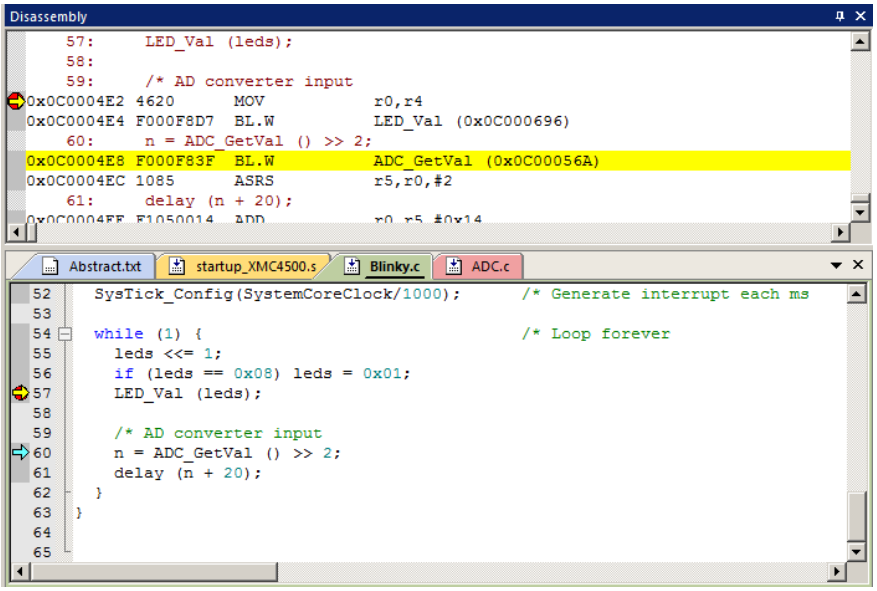
The LEDs on the Hitex board will now blink at a rate determined by the setting of the pot.

Rotate the potentiometer. The rate the three LEDs blink will change.

Now you know how to compile a program, load it into the XMC4000 processor Flash, run it and stop it.

2) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block somewhere appropriate between Lines 55 through 61 in the source file Blinky.c in the main() function as shown below:
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 technology does this.
6. The XMC4000 has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to. This is a very important feature for effective debugging.



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.

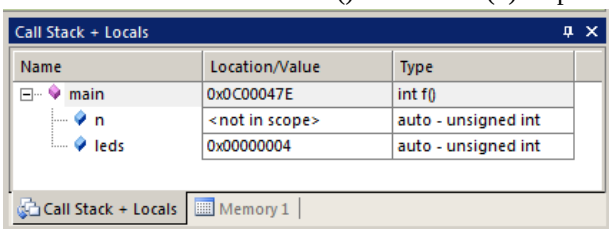
Copyright © 2012 ARM Ltd. All rights reserved

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.

1. Shown is the Locals window for the main function with the hardware breakpoint active from the previous page.
2. The contents of the local variables n and leds are displayed. This is from the function main() in the while(1) loop.
3. With the breakpoint set as in the previous page, as you click on RUN, these locals will update as appropriate.



Hint: To get n to display, set the breakpoint in the last line in the while(1) loop. Specifically on the line: `delay (n + 20);`

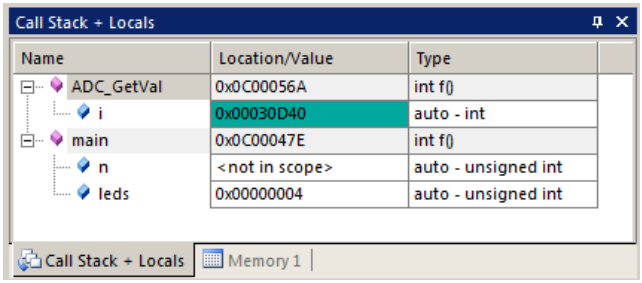
TIP: The contents of the local variables are displayed as well as names of active functions. Each function name will be displayed as it is called from the function before it or from an interrupt or exception. Exactly which local will be visible or not depends on precisely where you stop the program.

When a function exits, it is removed from the list.

The first called function is at the bottom of this table.

This table is active only when the program is stopped.

4. Remove all the breakpoints you have set. Click individually or Ctrl-B and Kill All or select Debug/Breakpoints and Kill All.
5. Set a new breakpoint on the line `n = ADC_GetVal () >> 2;` in Blinky.c. (near line 60).
6. Click on RUN and the program will stop on this line.
7. Click on the Step In icon or F11:
8. Note the function ADC_GetVal is displayed. See here:



TIP: If you get trapped in the Delay function, use Step Out or Ctrl-F11 to exit it faster.

9. Click numerous times on Step In and see other functions.
10. Click on the StepOut icon to exit all functions to return to main().
11. **Remove all breakpoints when you are done.** You can click on them individually or Ctrl-B and select Kill All.

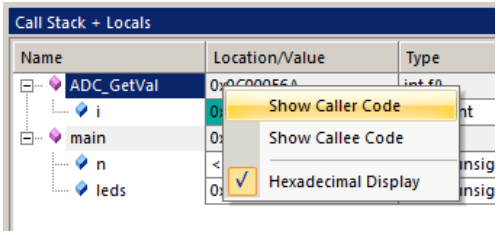
TIP: You can modify a variable value in the Call Stack & Locals window when the program is stopped.

Call Stack:

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

A window such as this one shows two functions (ADC_GetVal and main) and their local variables.

12. Right click on a function name and try the Show Callee Code and Show Caller Code options as shown here:
    The appropriate code will be shown in the source and/or disassembly windows.



Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

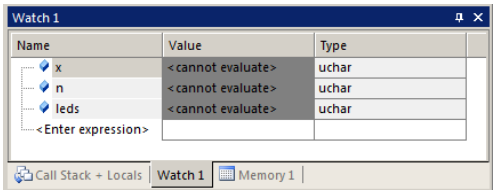
4) Watch and Memory Windows and how to use them:

µVision can display variables and update them while your program is running. You can also modify these values while the program is running. No code stubs are needed. No CPU cycles are stolen so this is non-intrusive to your program.

µVision can display global and static variables, structures and peripheral ports as well as physical memory addresses while the program is running. It cannot display local variables which are moving in and out of scope as functions are executed.

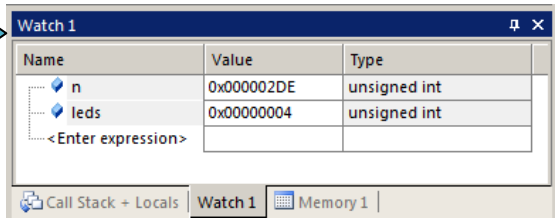
Watch window:

1. Click on the Run icon to start the program. You can add variables to the Watch and Memory windows when the program is running. You are not able to delete them. You must stop the program to delete a variable.
2. Find the declaration of local variables **n** and **leds** in Blinky.c. These are near line 47 in the main function.
3. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
4. Right click on **n** and **leds** in turn and select Add **n** to… and select Watch 1. Repeat for **leds**.
5. You can also block the variable and drag and drop it into the Watch window.
6. This window will display showing **n** and **leds**. **x** is not used in this exercise.
7. Note the values are displayed as <cannot evaluate> as these are local variables and uVision cannot access them. If you stop the program with a breakpoint on the line `delay (n + 20);`, **n** and **leds** will display.



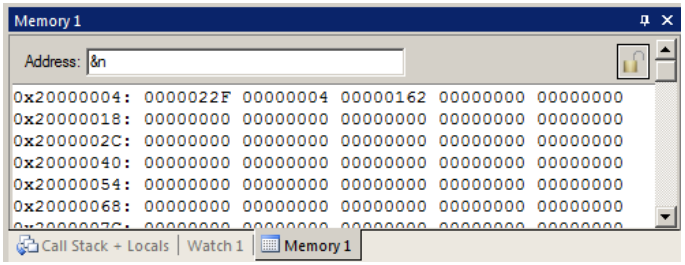
Make n and leds static:

8. Add the static declaration as shown here to the declarations of n and leds: `static uint32_t n, leds = 0x01;`
9. Stop the program, delete **x** (is optional step) and exit Debug mode.
10. Rebuild the project. Program the Flash (Load) and enter debug mode. Click on RUN.
11. **n** and **leds** will now be updated in real time as shown here:
12. Rotate the pot to see n update in real-time.
6. Double click on the value for n 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 this program so you will probably not see this happen.



Memory window:

1. Right click on **n** and select Add **n** to … and select Memory 1. Rotate the pot and watch the window change.
2. Note the value of n 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.
3. Add an ampersand "&" in front of the variable name and press Enter. Now the physical address is shown (0x2000_00004) with n = 0x22F.
4. Right click in the memory window and select Unsigned/Int.



1. Right-click on the data field and select Modify Memory. This is how you can modify a memory location.
2. Stop the CPU for the next step. Stay in Debug mode.

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 and M4 are 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 to memory 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.

Copyright © 2012 ARM Ltd. All rights reserved

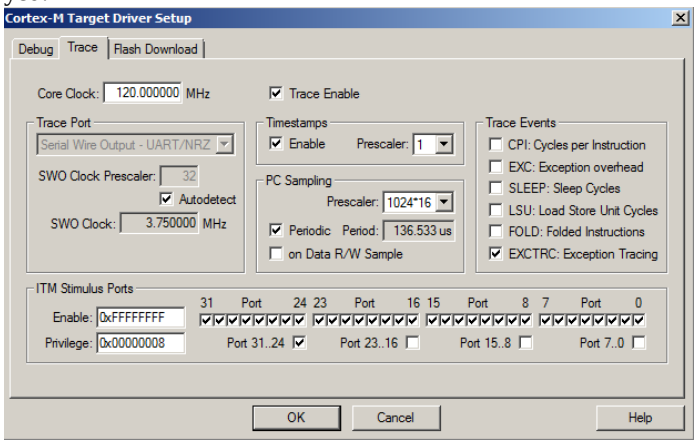
5) Configuring the Serial Wire Viewer (SWV):

Serial Wire Viewer provides program information in real-time.

A) SWV for ULINK2 or ULINK-ME: (ULINK*pro* instructions are on the next page)

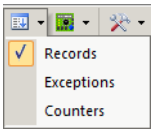
Configure SWV:

1. µVision 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/ME 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** 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 120 and select the Trace Enable box as shown here:.
8. Select Periodic and leave everything else at default. Periodic activates PC Samples.
9. Click on OK twice to return to the main µVision menu. SWV is now configured.
10. Select File/Save All.



To Display Trace Records:

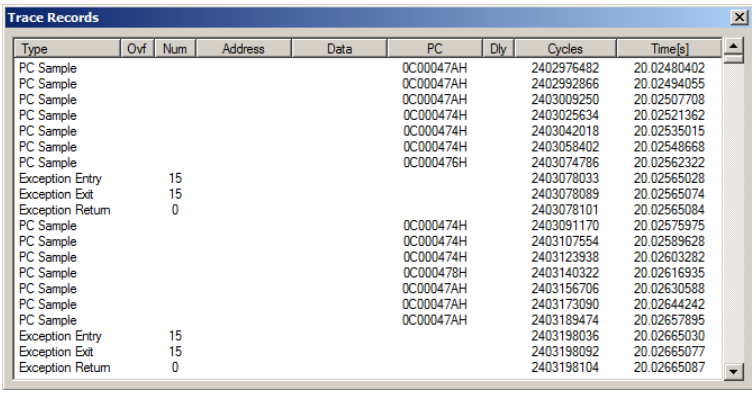
1. Enter Debug mode.
2. Click on the RUN icon.
3. Open Trace Records window by clicking on the small arrow beside the Trace icon:



4. The Trace Records window will open and display PC Samples and Exception 15 as shown below:

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.

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



5. Double-click this window to clear it.
6. If you right click inside this window you can see how to filter various types of frames out. Unselect PC Samples and you will see only exception frames displayed.
7. Select the Exceptions window as in Step 3.
8. Exceptions will be listed with various timing information. Double-click to clear.
9. When you are done, select Debug/Debug Settings.. and select the Trace tab. Unselect PC Samples. Generally you do not want this elected as it can overload the SWO pin and cause erroneous data trace due to dropped frames or delayed timestamps..
10. Click on OK twice. This procedure will automatically stop the processor.

Did you know Exception 15 was being activated ? Now you do. This is a very useful tool for displaying how many times an exception is firing and when. Many performance issues are created by exceptions firing too often.

TIP: If you are using a ULINK*pro* or J-Link, you must stop the program to see the trace frames.

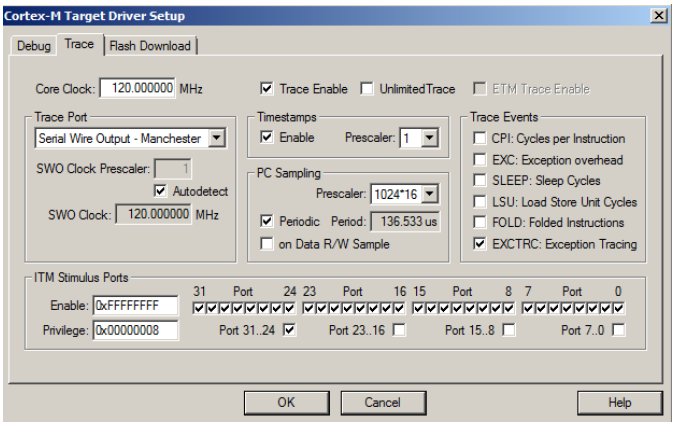
TIP: SWV is easily overloaded as indicated by an "x" in the OVF or Dly column. Select only that information needed to reduce overloading. There are more useful features of Serial Wire Viewer as we shall soon discover.

Copyright © 2012 ARM Ltd. All rights reserved

B) SWV for ULINK*pro*: *for reference: Follow these instructions if you have a ULINK*pro*:*

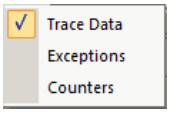
Configure SWV: This uses the SWO output pin rather than the 4 bit Trace Port that is normally used with the ULINK*pro*.

1. µVision must be stopped and in edit mode and with a ULINK*pro* connected to any of the three debug connectors.
2. Select Options for Target or ALT-F7 and select the Debug tab.
3. ULINK*pro* must be configured for both debugging and Flash programming. (the Debug and Utilities tabs)
4. In the Use: box select ULINK Pro Cortex debugger. In the Utilities tab, select ULINK Pro Cortex debugger.
5. Select Settings: select Add and then select XMC4000 Flash and then Add. Click on OK once. Select Debug tab.
6. Click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side.
7. Make sure SWJ and SW are selected. JTAG is not available with default configuration of the HiLight board.
8. Click on the Trace tab. The window below is displayed. This almost the same as for the ULINK2/ME.
9. Core Clock: ULINK*pro* uses this only for calculating timing values. Enter 120 MHz. Select the Trace Enable box.
10. In the Trace Port select Serial Wire Output – Manchester. Selecting UART/NRZ will cause an error.
11. Select Periodic and leave everything else at default. Selecting Periodic activates PC Samples.
12. Click on OK twice to return to the main µVision menu. SWV is now configured for the ULINK*pro*.
13. Rebuild this project and program the Flash with the Load icon. Select File/Save All.



Display Trace Records:

1. Enter Debug mode.
2. Click on the RUN icon.
3. Open the Trace selection window by clicking on the small arrow beside the Trace icon:



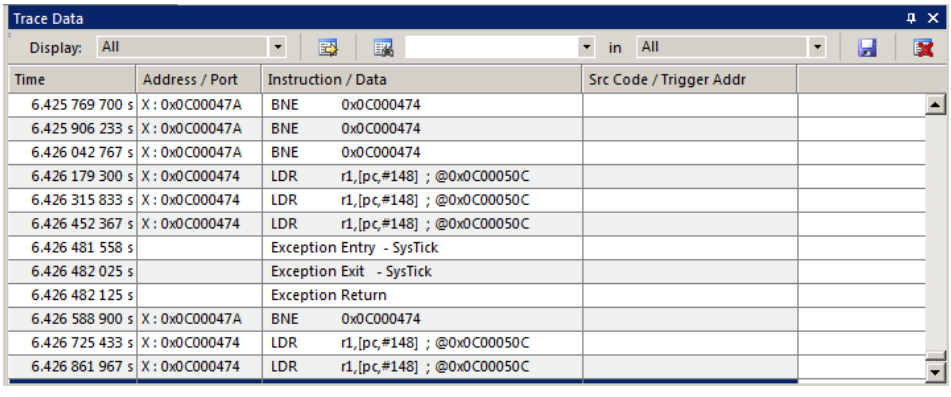
4. Select Trace Data and the Trace Data window shown below will open.
5. Stop the processor and frames are displayed as shown below:
6. Select various types of frames with the Display: box to filter out various types of frames. The default is ALL.
7. Double-click on a PC Sample and this instruction will be highlighted in the Disassembly window.

TIP: The Trace Data window is different than the Trace Records window provided with the ULINK2. Trace Records display only SWV frames and Trace Data can display SWV OR ETM instruction frames. Note the disassembled instructions are displayed and if available, the source code is also displayed. If you want to see all the program counter values, use ETM trace. A Ulink*pro* using ETM trace also provides Code Coverage, Performance Analysis and Execution Profiling.

Clear and Save Trace Records:

You can clear the Trace Data window by clicking on the Clear icon.

You can also save the contents by clicking on the Save icon.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
6.425 769 700 s	X : 0x0C00047A	BNE 0x0C000474	
6.425 906 233 s	X : 0x0C00047A	BNE 0x0C000474	
6.426 042 767 s	X : 0x0C00047A	BNE 0x0C000474	
6.426 179 300 s	X : 0x0C000474	LDR r1,[pc,#148] ; @0x0C00050C	
6.426 315 833 s	X : 0x0C000474	LDR r1,[pc,#148] ; @0x0C00050C	
6.426 452 367 s	X : 0x0C000474	LDR r1,[pc,#148] ; @0x0C00050C	
6.426 481 558 s		Exception Entry - SysTick	
6.426 482 025 s		Exception Exit - SysTick	
6.426 482 125 s		Exception Return	
6.426 588 900 s	X : 0x0C00047A	BNE 0x0C000474	
6.426 725 433 s	X : 0x0C000474	LDR r1,[pc,#148] ; @0x0C00050C	
6.426 861 967 s	X : 0x0C000474	LDR r1,[pc,#148] ; @0x0C00050C	

1. When you are done, select Debug/Debug Settings.. and select the Trace tab. Unselect PC Samples. Generally you do not want this elected as it can overload the SWO pin and cause erroneous data trace due to dropped frames or delayed timestamps..
2. Click on OK twice. This procedure will automatically stop the processor.

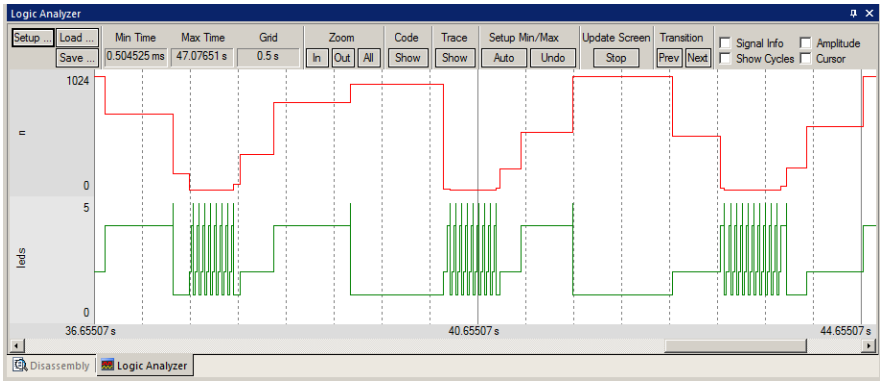
Copyright © 2012 ARM Ltd. All rights reserved

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

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. Even so, SWV is extremely useful for displaying various information and events.

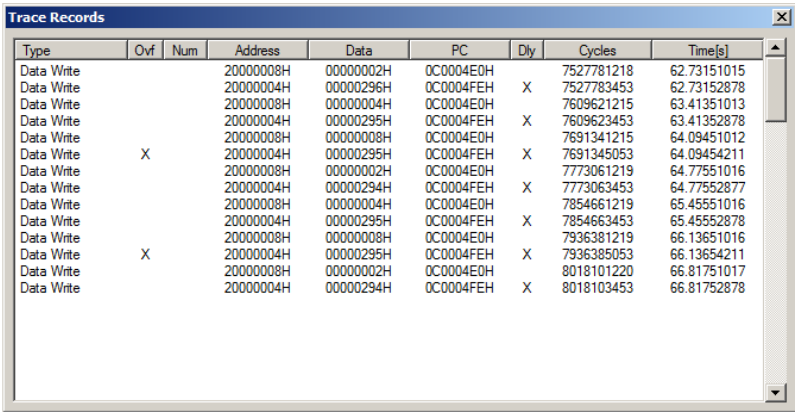
This example will display variables n and leds from the Blinky example. Please connect a ULINK2 or ULKINK-ME to your board and configure it for SWV trace. You probably have already done this. If you want to use a ULINK*pro* you will have to configure it as on the previous page. ULINK*pro* provides a better SWV support with its Manchester or Trace Port support.

1. The project Blinky should still be open and is probably still in Debug mode. Click on STOP if running..
2. Recall we have previously created two static variables: n and leds. We will graph these in the Logic Analyzer.
3. SWV must be configured. See the two previous pages for instruction to implement SWV for your debug adapter.
4. Enter Debug mode if not already in this mode.
5. Select Debug/Debug Settings and select the Trace tab.
6. Unselect Periodic and EXCTRC. This is to prevent overload on the SWO pin. Click OK twice.
7. Run the program. . **Note:** You can configure the LA while the program is running or stopped.
8. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.
9. Locate variables n and leds from Blinky.c. Right click on each and select Insert var_name into… to Logic Analyzer.
10. Or block each variable and drag it into the LA window and release it.
11. Click Setup and set Max: in Display Range for n to 0x400 and leds to 0x5. Click on Close. The LA is configured.
12. Adjust the Zoom OUT or the All icon in the LA window to provide a suitable scale of about 0.5 second.
13. Rotate the pot to obtain an interesting waveform as shown here:



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 them into the Logic Analyzer and read or write to them.

1. Select Debug/Debug Settings and select the Trace tab.
2. Select On Data R/W Sample. Click OK twice.
3. Run the program. .
4. Open the Trace Records window and clear it by double clicking in it.
5. The window similar below opens up:
6. The first line below says:
   The instruction at 0x0C00 04E0 caused a write of data 0x02 to address 0x2000_0008 (is the address of leds) at the listed time in CPU Cycles or accumulated Time in seconds.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000008H	00000002H	0C0004E0H		7527781218	62.73151015
Data Write			20000004H	00000296H	0C0004FEH	X	7527783453	62.73152878
Data Write			20000008H	00000004H	0C0004E0H		7609621215	63.41351013
Data Write			20000004H	00000295H	0C0004FEH	X	7609623453	63.41352878
Data Write			20000008H	00000008H	0C0004E0H		7691341215	64.09451012
Data Write	X		20000004H	00000295H	0C0004FEH	X	7691345053	64.09454211
Data Write			20000008H	00000002H	0C0004E0H		7773061219	64.77551016
Data Write			20000004H	00000294H	0C0004FEH	X	7773063453	64.77552877
Data Write			20000008H	00000004H	0C0004E0H		7854661219	65.45551016
Data Write			20000004H	00000295H	0C0004FEH	X	7854663453	65.45552878
Data Write			20000008H	00000008H	0C0004E0H		7936381219	66.13651016
Data Write	X		20000004H	00000295H	0C0004FEH	X	7936385053	66.13654211
Data Write			20000008H	00000002H	0C0004E0H		8018101220	66.81751017
Data Write			20000004H	00000294H	0C0004FEH	X	8018103453	66.81752878

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.

TIP: You can enter raw addresses into the LA as such: *((unsigned long *)0x20000000).

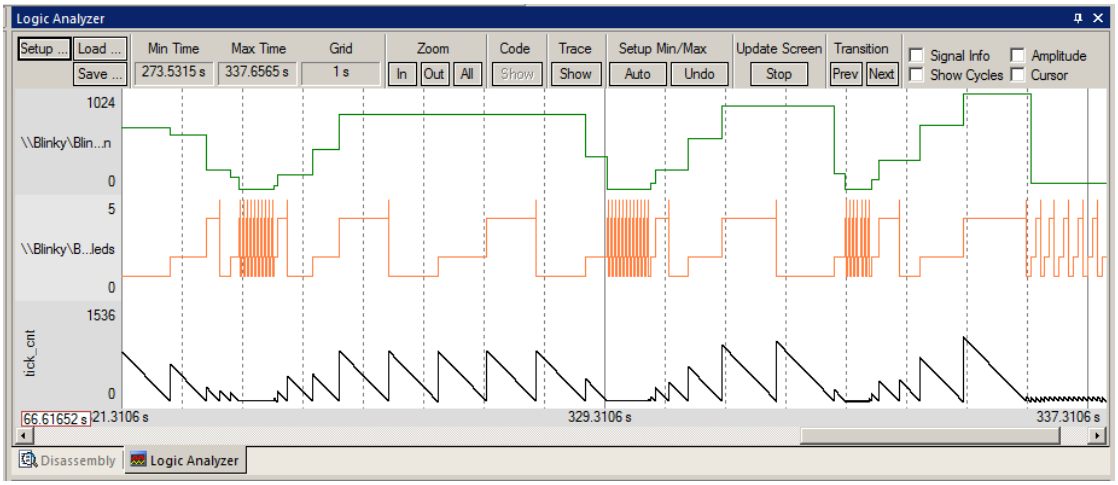
Erratta: The XMC4000 resets the CoreSight only on power up. If the trace acts erratic, cycle the Hitex board power.

Copyright © 2012 ARM Ltd. All rights reserved

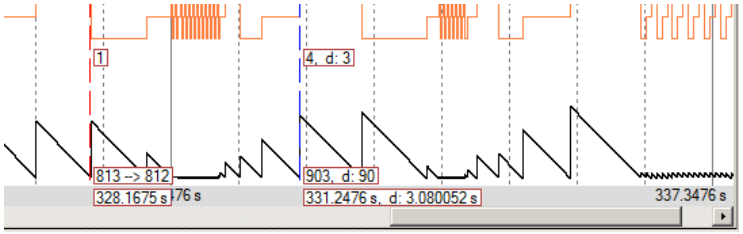
7) Another Use for the Logic Analyzer:

In Blinky.c, there is a global variable called tick_cnt. It is decremented by the SysTick timer and used in the delay function.. A question might arise: what is the duty cycle of this variable and how does it vary, if at all, with the setting of the pot ? You could examine the code to determine this or instrumentate your code to send it out to a GPIO port to see with a scope…or merely put the variable in the Logic Analyzer.

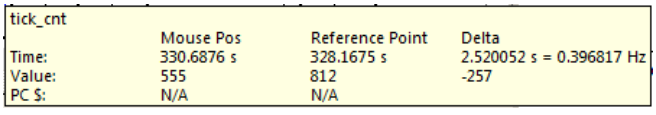
1. Locate the global variable tick_cnt in Blinky.c around Line 20 along with n and leds.
2. Enter this into the Logic Analyzer: either manually, a right click or by dragging and dropping. You can do this while the program is running.
3. Open Setup and either set the Display Range: Max to 0x600. Click on Close.
4. Adjust the Zoom with IN, Out or All for a suitable display as shown below: Rotate the pot. You can easily see the duty cycle and waveform of the variable tick_cnt and how it behaves relative to the pot position.
   It would be tricky to find this out without trace and would take much longer than we have demonstrated here.



5. Stop the program. Select the Cursor checkbox.
6. Click on a tick_cnt rising edge and note a fixed red line is created.
7. A blue line follows the mouse and times are displayed as shown below:
8. Select Signal Info.
9. Hover the cursor for a few seconds and timing information is displayed as shown below in the yellow box:
10. Click on Setup… and remove tick_cnt from the LA by highlighting it and selecting the delete icon.



TIP: Remember you can display a variable that does not exist in the outside world for debugging purposes. These can include variables that are an internal part of a mathematical formula and impossible to measure with instruments such as oscilloscopes or logic analyzers.



TIP: The data writes to the variables listed in the LA will be visible in the Trace Records window. The variable tick_cnt is written to at a high rate. Make sure no extra SWV events are selected that can cause problems. These may manifest themselves as variables stop nor will never changing or become erroneous. The solution is to reduce the SWV events or use a ULINK*pro*.

TIP: **In the LA above, t**he variables n and leds are Fully Qualified to help µVision find them. This is sometimes necessary for static local variables. To enter variables Fully Qualified, right click on the variable in the Symbol window or enter the path manually. To display the Symbol window, select View/Symbol Window.

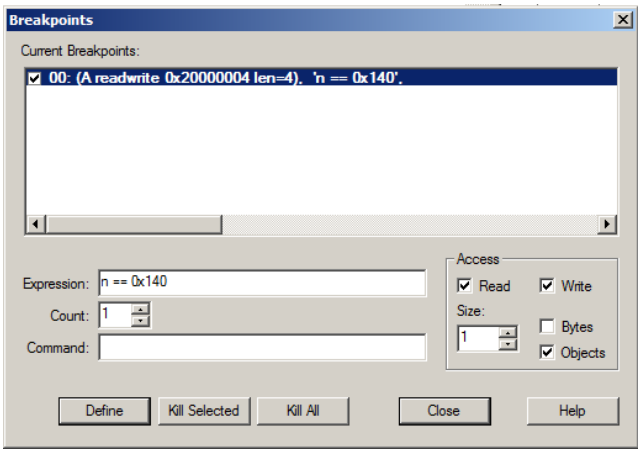
If you still have problems, convert them to global variables.

Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

8) Watchpoints: Conditional Breakpoints

The XMC4000 Cortex-M4 processors have 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. This is why you deleted tick_cnt on the previous page. There was only one comparator left.

1. Using the example from the previous page, stop the program. Stay in Debug mode.
2. Click on Debug and select Breakpoints or press Ctrl-B.
3. The SWV Trace does not need to be configured to use Watchpoints. However, we will use it in this exercise.
4. In the Expression box enter: "**n** == 0x120" without the quotes. Select both the Read and Write Access.
5. Click on Define and it will be accepted as shown here:
6. Click on Close.
7. If you get an error, make sure you deleted tick_cnt.
8. Double-click in the Trace Records window to clear it.
9. Click on RUN.
10. Turn the pot to get n to equal 0x120.
11. When **n** equals 0x120, the program will stop. This is how a Watchpoint works.
12. You will see **n** displayed as 0x120 in the Logic Analyzer, the Watch and Trace Records windows.
13. Note the data write of 0x120 in the Trace Records window shown below in the Data column. The address the data written to and the PC of the write instruction is displayed as well as the timestamps. This is with a ULINK2 or ULINK-ME. The ULINK*pro* will display a different window.
14. There are other types of expressions you can enter and they are detailed in the Help button in the Breakpoints window.
15. To repeat this exercise, click on RUN. If the program stops immediately, enter a different value in n in Watch 1 window and try again.
16. **When finished, click on STOP and delete this Watchpoint by selecting Debug and select Breakpoints and select Kill All. You can also access this window with a Ctrl-B.**
17. 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.

TIP: To see the PC column, you must have on R/W Samples enabled in the Trace Configuration window. You can unselect this if the SWV is overloaded as indicated by many "X" displayed. That is not the case here. There are no lost frames or delayed timestamps.

Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000004H	00000112H	0C0004FEH		1425544361	11.87953634
Data Write			20000004H	00000113H	0C0004FEH		1460824338	12.17353615
Data Write			20000004H	00000113H	0C0004FEH		1496224326	12.46853605
Data Write			20000004H	0000011BH	0C0004FEH		1531624356	12.76353630
Data Write			20000004H	0000011CH	0C0004FEH		1567984336	13.06653613
Data Write			20000004H	0000011AH	0C0004FEH		1604464327	13.37053606
Data Write			20000004H	0000011CH	0C0004FEH		1640704360	13.67253633
Data Write			20000004H	00000121H	0C0004FEH		1677184340	13.97653617
Data Write			20000004H	00000126H	0C0004FEH		1714264334	14.28553612
Data Write			20000004H	0000012BH	0C0004FEH		1751944364	14.59953637
Data Write			20000004H	0000012CH	0C0004FEH		1790224335	14.91853612
Data Write			20000004H	0000012CH	0C0004FEH		1828624329	15.23853608
Data Write			20000004H	0000012BH	0C0004FEH		1867024359	15.55853633
Data Write			20000004H	0000012BH	0C0004FEH		1905304339	15.87753616
Data Write			20000004H	00000123H	0C0004FEH		1943584327	16.19653606
Data Write			20000004H	00000123H	0C0004FEH		1980904360	16.50753633
Data Write			20000004H	00000122H	0C0004FEH		2018224334	16.81853612
Data Write			20000004H	0000011EH	0C0004FEH		2055424331	17.12853609
Data Write			20000004H	0000011FH	0C0004FEH		2092144364	17.43453637
Data Write			20000004H	00000120H	0C0004FEH		2128984335	17.74153613

Copyright © 2012 ARM Ltd. All rights reserved

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

Keil provides RTX, a full feature RTOS. RTX is included as part of the Keil MDK full tool suite. It can have up to 255 tasks and no royalty payments are required. RTX has a BSD type license. Source code is provided with all versions of MDK. See www.arm.com/cmsis. This example explores the RTOS project. Keil will work with any RTOS. A RTOS is just a set of C functions that gets compiled with your project. A real-time awareness viewer for RTX is provided inside µVision.

1. Start µVision by clicking on its icon.
2. Select Project/Open Project and open C:\Keil\ARM\Boards\Hitex\XMC-HiLight\RTX_Blinky\Blinky.uvproj.
3. Select XMC4000 Flash in the Target selector:

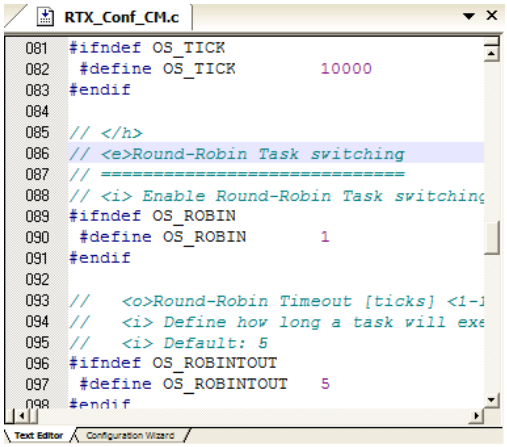
4. RTX_Blinky uses the ULINK2 as default: if you are using a ULINK*pro*, please configure it as described on page 5. You only have to do this once for each project – it will be saved in the project file by selecting File/Save All.
5. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
6. To program the Flash manually, click on the Load icon. A progress bar will be at the bottom left.
7. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
8. The LEDs will blink indicating the three of four waveforms of a stepper motor driver. (board has only three LEDs)
9. Click on STOP.

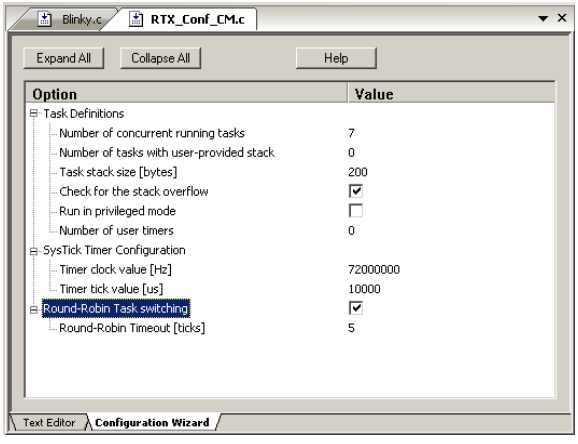
The Configuration Wizard for RTX:

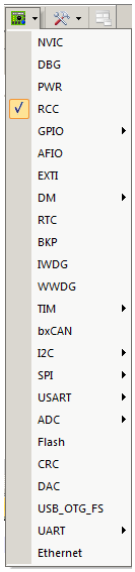
1. Click on the RTX_Conf_CM.c source file tab as shown below on the left below. You can open it with File/Open.
2. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. This is a great feature as it is much easier changing items here than in the source code.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>.
8. The µVision4 System Viewer windows are created in a similar fashion. Select View/System Viewer or click on the icon. The window similar to the one on the far right opens.

TIP: If you don't see any System Viewer entries, either the System Viewer is not available for your processor or you are using an older example project and it needs to be "refreshed" by the following instructions:

Exit Debug mode. Click on the Target Options icon and select the Device tab. Note which processor is currently selected. Select a different one, reselect the original processor and click on OK. System Viewer is now activated. Close this window and select File/Save All.





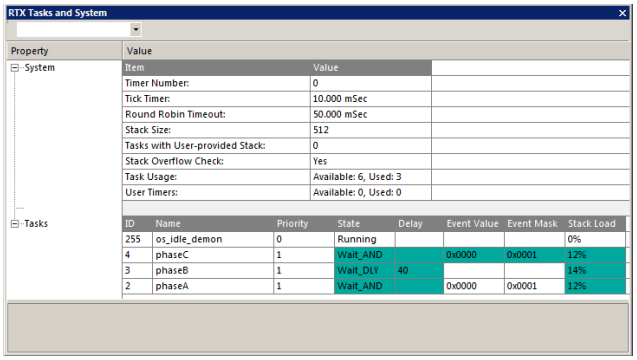


Text Editor: Source Code **Configuration Wizard** **System Viewer**

Copyright © 2012 ARM Ltd. All rights reserved

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

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



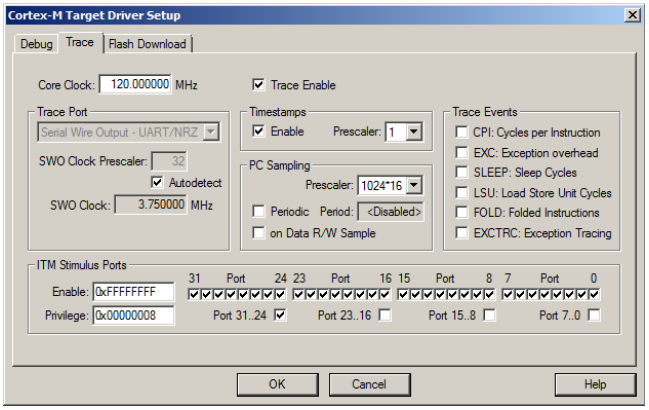
1. Run RTX_Blinky by clicking on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. These values are updated in real-time using the same read write technology as used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

Important TIP: View/Periodic Window Update must be selected ! Otherwise will update only when program is stopped.

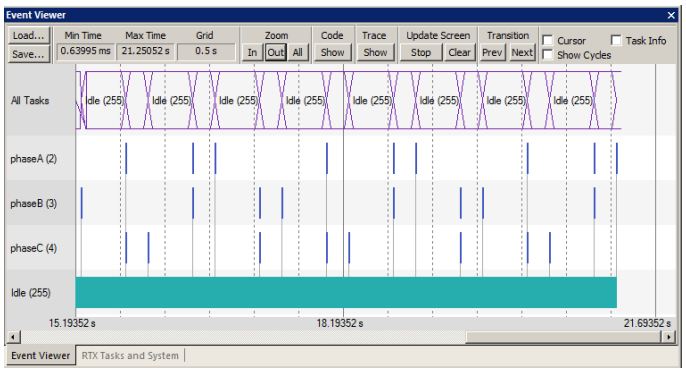
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured which the Event Viewer uses.

RTX Viewer: Configuring Serial Wire Viewer (SWV):

We must activate Serial Wire Viewer to get the Event Viewer working.



1. Stop the CPU and exit debug mode.
2. Click on the Target Options icon.
3. Click on the Debug tab.
4. Click the Settings box next to ULINK/ME Cortex Debugger.
5. In the Debug window, make sure SWJ is checked and Port: is set to SW and not JTAG.
6. Click on the Trace tab.
7. Set Core Clock: to 120 MHz and select Trace Enable.
8. Unselect the Periodic and EXCTRC boxes as shown above: Leave everything else at default.
9. 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.
10. Click on OK twice to return to µVision. ***The Serial Wire Viewer is now configured in µVision.***
11. Enter Debug mode and click on RUN to start the program.
12. Click on the Event Viewer tab.
13. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 0.5 second by clicking on the Zoom ALL and then In and Out.



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

Cortex-M3 Alert: µVision 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 31 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.

Copyright © 2012 ARM Ltd. All rights reserved

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

µVision has a graphical Logic Analyzer window. Up to four variables can be displayed in real-time using the Serial Wire Viewer in the XMC4000. RTX_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.

1. Close the RTX Viewer windows. Stop the program and exit debug mode.
2. Add 3 global variables **`unsigned int phasea`** through **`unsigned int phasec`** to Blinky.c as shown below:
3. Add 2 lines to each of the four tasks Task1 through Task3 in Blinky.c as shown below: **phasea=1;** and **phasea=0;** :the first two lines are shown added at lines 38 and 41 (just after LED_On and LED_Off function calls). For each of the three tasks, add the corresponding variable assignment statements phasea, phaseb, and phasec.

```
17  OS_TID t_phaseC;
18  unsigned int phasea = 0;
19  unsigned int phaseb = 0;
20  unsigned int phasec = 0;
21
```

4. We do this because in this simple program there are not enough suitable variables to connect to the Logic Analyzer.

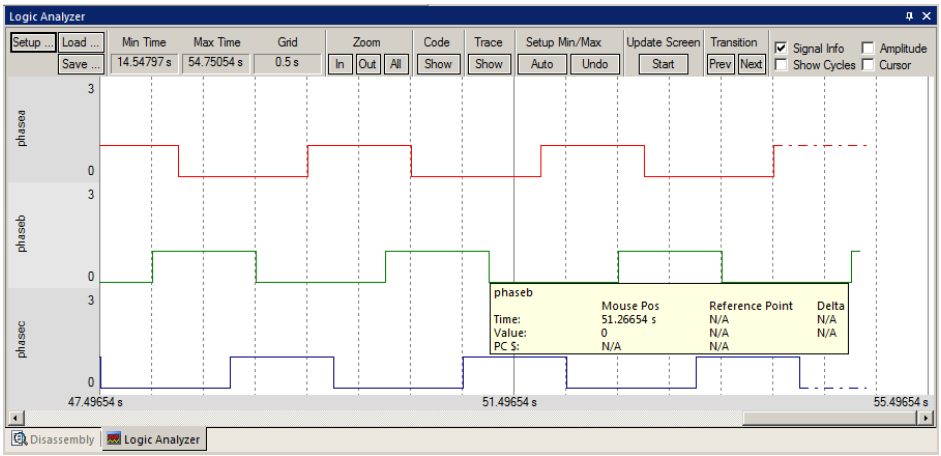
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

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

```
34  __task void phaseA (void) {
35    for (;;) {
36      os_evt_wait_and (0x0001, 0x
37      LED_On (0);
38      phasea = 1;
39      signal_func (t_phaseB);
40      LED_Off (0);
41      phasea = 0;
42    }
43  }
```

Enter the Variables into the Logic Analyzer:

9. Click on the Blinky.c tab. Block **phasea**, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
10. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release. You can also right-click on the variable name and select Add "phasea" to … and select Logic Analyzer.



11. Repeat for **phaseb and phasec.** These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.
12. Click on the Setup icon and click on each of the three variables and set Max. in the Display Range: to 0x3.
13. Click on Close to go back to the LA window.
14. Using the All, OUT and In buttons set the range to 1second or so. Move the scrolling bar to the far right if needed.
15. 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:
16. Select Cursor and click on a waveform. Move the cursor to see the timings.

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

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

Copyright © 2012 ARM Ltd. All rights reserved

12) Serial Wire Viewer (SWV) and how to use it: (with ULINK2 or ME)

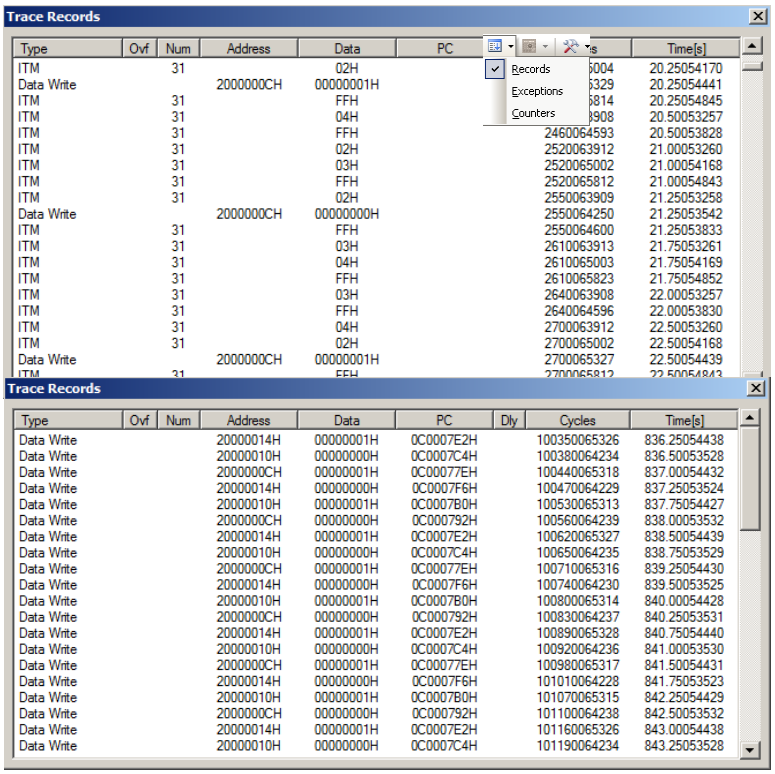
a) Data Reads and Writes: (Note: Data Reads but not Writes are disabled in the current version of µVision**).**

You have configured Serial Wire Viewer (SWV) two pages back in Page 15 under **Configuring the Serial Wire Viewer:**

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

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

1. Use RTX_Blinky from the last exercise. Enter Debug mode and run the program.
2. Select View/Trace/Records or click on the Trace icon and select Records.
3. The Trace Records window will open up:
4. The ITM frames 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 ITM Stim. Port 31. Or right click in the trace records window and unselect ITM frames to filter them.



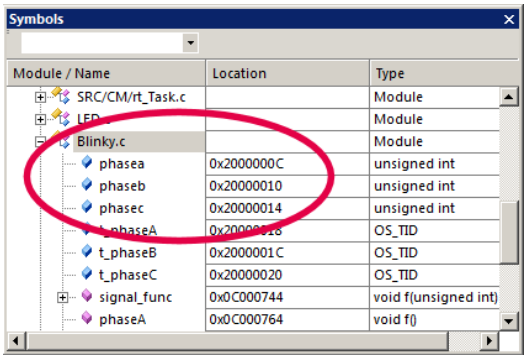
TIP: Port 0 is used for Debug `printf` Viewer.

5. Select Debug/Debug Settings…
6. Select the Trace tab.
7. Unselect EXCTRC, Periodic and ITM 31.
8. Select On Data R/W Sample.
9. Click on OK twice to return.
10. Click on the RUN icon.
11. Double-click anywhere in the Trace records window to clear it.
12. Only Data Writes will appear now.

TIP: You could have also right clicked on the Trace Records window to filter the ITM frames out but this will not reduce any potential SWO pin overloads. The general rule is: select only those SWV events you really need.

What is happening here ?

1. When variables are entered in the Logic Analyzer (remember phasea through phasec ?), the writes will appear in Trace Records.
2. The Location column shows where the four variables address.
3. The Data column displays the data values written to phasea through phasec.
4. PC is the address of the instruction causing the writes. You activated it by selecting On Data R/W Sample in the Trace configuration window.



5. The Cycles and Time(s) columns show when these events happened.

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

TIP: If you select View/Symbol Window you can see where the addresses of the variables are as shown above:

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

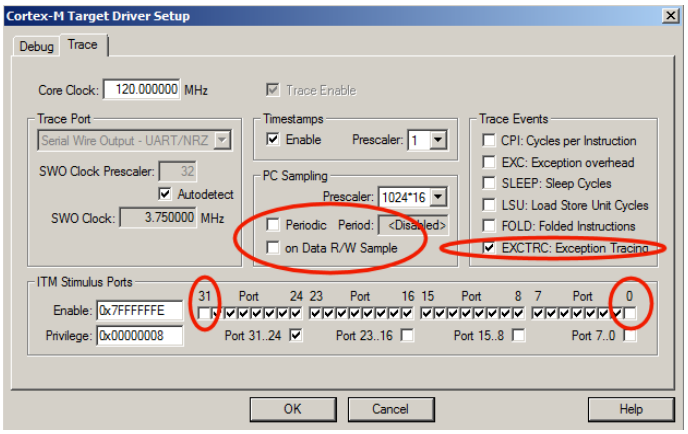
TIP: ULINK*pro* and the Segger J-Link adapters display the trace frames when the program is stopped..

Copyright © 2012 ARM Ltd. All rights reserved

b) Exceptions and Interrupts:

The XMC4000 family using Cortex-M4 processors has many interrupts and it can be difficult to determine when they are being activated and when. Serial Wire Viewer (SWV) makes the display of exceptions and interrupts 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. (this is to minimize overloading the SWO port)
3. Select EXCTRC as shown here:
4. Click OK twice.
5. The Trace Records should still be open. Double click on it to clear it.
6. Click RUN to start the program.
7. You will see a window similar to the one below with Exceptions frames displayed.

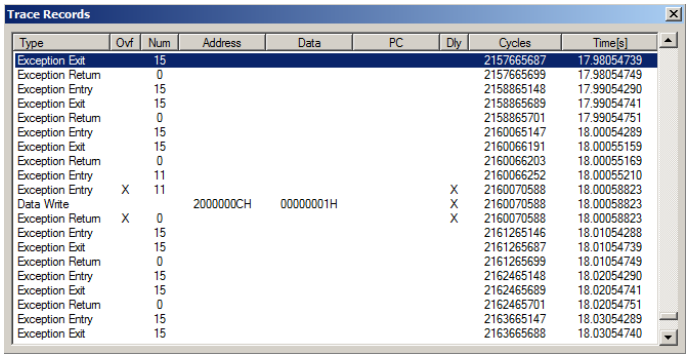


What Is Happening ?

1. You can see two exceptions (11 & 15) happening.

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned. This is useful to detect tail-chaining.

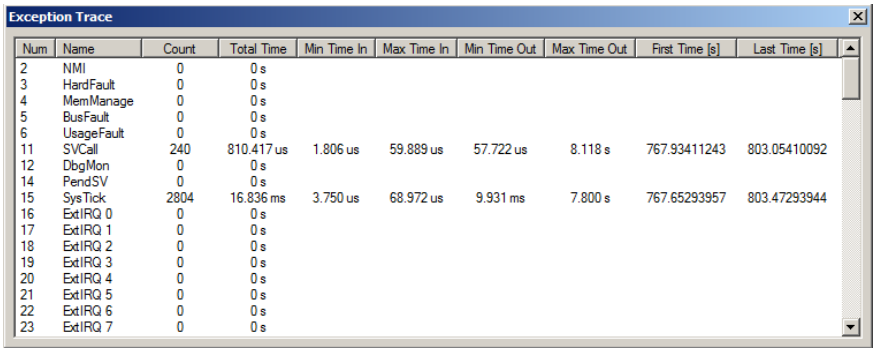
2. Num 11 is SVCall from the RTX calls.
3. Num 15 is the Systick timer.
4. In my example you can see one data write from the Logic Analyzer.
5. Note everything is timestamped.
6. The "X" in Ovf is an overflow and some data was lost. The "X" in Dly means the timestamps are delayed because too much information is being fed out the SWO pin.



TIP: The SWO pin is one pin on the Cortex-M4 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. The exception is the ULINK*pro* which can also send SWV out the 4 bit Trace Port. These exceptions are happening at a very fast rate. Overloads are gracefully handled.

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 is displayed as shown.
3. Note the number of times these have happened under Count. This is very useful information in case interrupts come at rates different from what you expect.
4. ExtIRQ are the peripheral interrupts.
5. You can clear this trace window by double-clicking on it.
6. All this information is displayed in real-time and without stealing CPU cycles !



TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

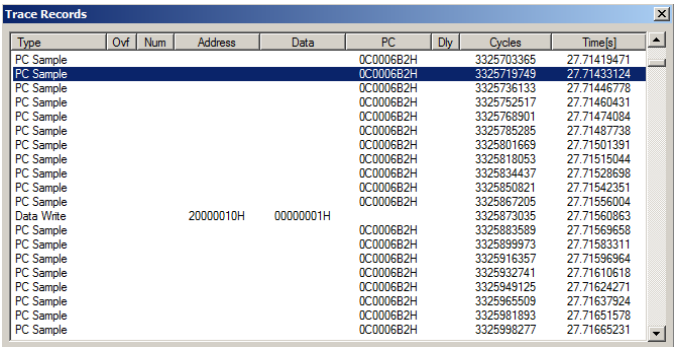
Copyright © 2012 ARM Ltd. All rights reserved

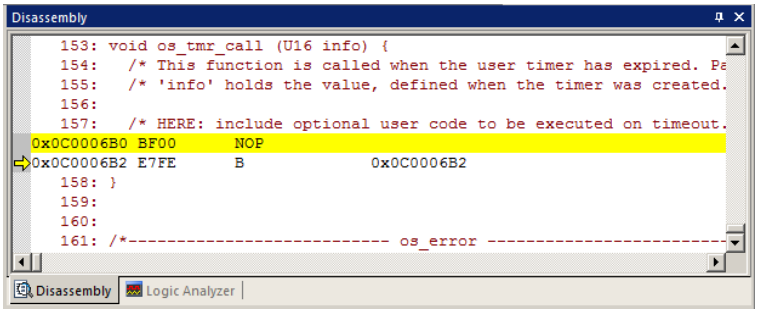
c) PC Samples:

Serial Wire Viewer can display a sampling of the program counter. If you need to see all the PC values, use the ETM trace with a Keil ULINK*pro*. ETM trace also provides Code Coverage, Execution Profiling and Performance Analysis.

SWV can display at best every 64th instruction but usually every 16,384 is more common. It is best to keep this number as high as possible to avoid overloading the Serial Wire Output (SWO) pin. This is easily set in the Trace configuration.

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear it.
5. Click on RUN and this window opens:
6. Most of the PC Samples are 0x0C00_06B2 which is a branch to itself in a loop forever routine.
7. Stop the program and the Disassembly window will show this Branch:
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 tight loop like in this case.
9. **Note:** you can get different PC values depending on the optimization level set in µVision.
10. Set a breakpoint in one of the tasks in Blinky.c.
11. Run the program and when the breakpoint is hit, the program and trace collection is stopped.





12. Scroll to the bottom of the Trace Records window and you might see the correct PC value displayed. Usually, it will be a different PC depending on when the sampling took place. The processor spends a great deal of time executing this one instruction.
13. Remove the breakpoint for the next step.
14. Open Debug/Debug Settings and select the Trace tab.
15. Unselect PC Samples.

TIP: In order to see all the program Counter values, use ETM trace with the ULINK*pro*. XMC4000 processors have ETM.

ETM is much superior for program flow debugging than PC Samples.

µVision with a ULINK*pro* uses ETM to provide Code Coverage, Execution Profiling and Performance Analysis.

TIP: PC Samples is very useful to determine if the Core Clock is set correctly. If you see PC Samples with nearly always the same PC value – the Core Clock is correctly set (or is very close).

If you see strange frames such as ITM not 0 or 31 – or interrupts or data frames that do not exist – the Core Clock is not set correctly.

Copyright © 2012 ARM Ltd. All rights reserved

13) ITM (Instruction Trace Macrocell) the `printf` feature:

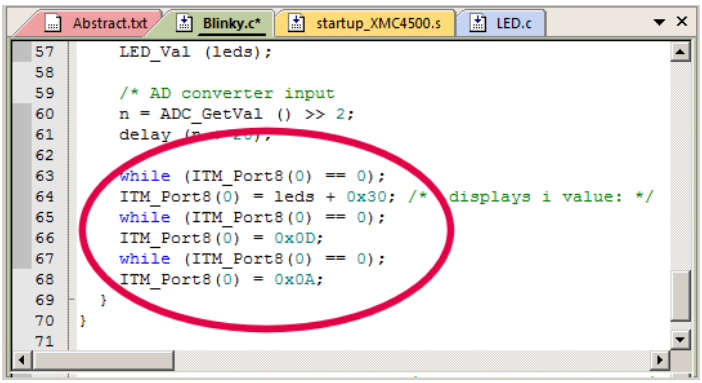
Recall that we showed you can display information about the RTOS in real-time using the RTX Viewer. This is done through ITM Stimulus Port 31. ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in the Debug (printf) Viewer window.

1. Open the project Blinky.uvproj (not RTX Blinky).
2. Add this #define to Blinky.c. A good place is near line 34, just after the declaration of **`char text[40];`**.

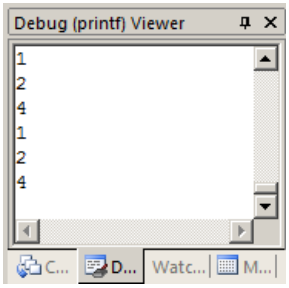
```
#define ITM_Port8(N)    (*((volatile unsigned char *)(0xE0000000+4*N)))
```

3. In the function LED_Out in Blinky.c, enter these lines starting at near line 128:

```
while (ITM_Port8(0) == 0);
ITM_Port8(0) = leds + 0x30; /*
displays leds value: */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```



4. Rebuild the source files, program the Flash memory and enter debug mode.
5. Select File/Save All.
6. Open Debug/Debug Settings and select the Trace tab.
7. Unselect On Data R/W Sample, Periodic and ITM Port 31. (this is to help not overload the SWO port)
8. Select EXCTRC and ITM Port 0. ITM Stimulus Port "0" enables the Debug (prinftf) Viewer.
9. Click OK twice.
10. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
11. In the Debug (printf) Viewer you will see the ASCII value of variable **`leds`** appear.



TIP: This code seems to interfere with using static variables in the Logic Analyzer.

Trace Records

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

What Is This ?

1. ITM 0 frames (Num column) are our ASCII characters from **`num`** with carriage return (0D) and line feed (0A) as displayed the Data column.
2. All these are timestamped in both CPU cycles and time in seconds.
3. Note the "X" in the Dly column. This means the timestamps might/are not be correct due to SWO pin overload.

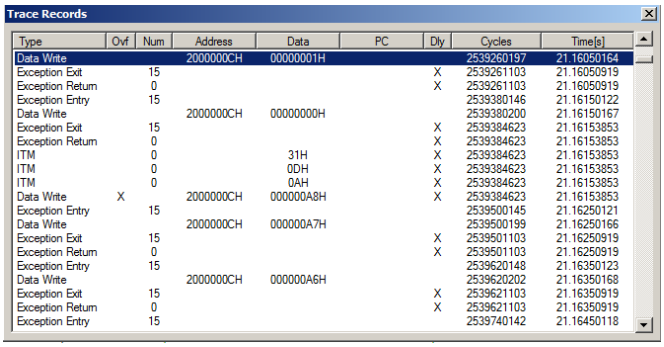
ITM Conclusion

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 XMX4000 processor via the Serial Wire Output pin.

This is much faster than using a UART and none of your peripherals are used.

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

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



Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

Part C)

1) DSP SINE example using ARM CMSIS-DSP Libraries:

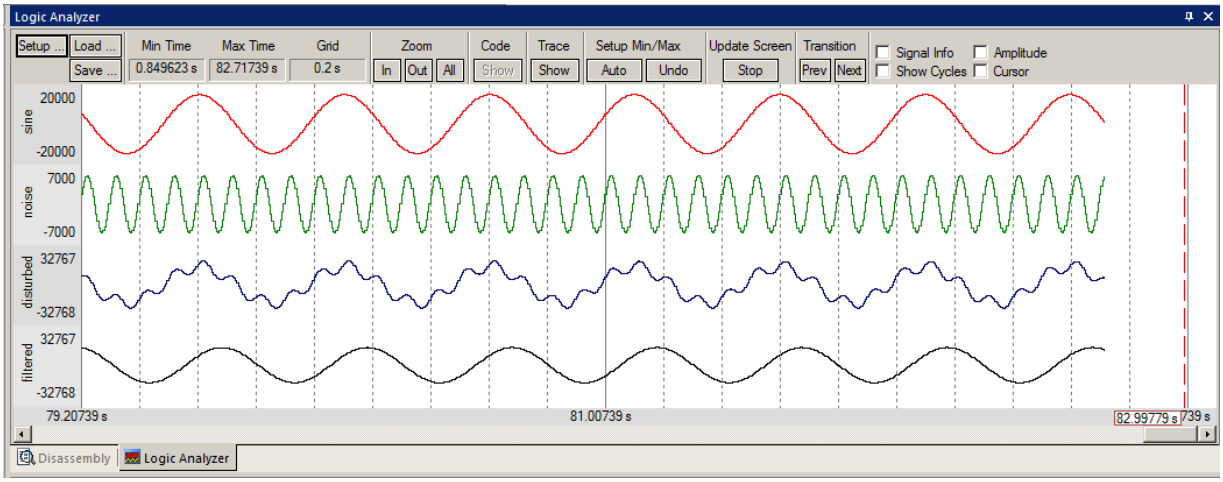
ARM CMSIS-DSP libraries are offered for ARM Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil\ARM\CMSIS. README.txt describes the location of various CMSIS components. See www.arm.com/cmsis and www.onarm.com/cmsis/download/ for more information. CMSIS is an acronym for Cortex Microcontroller Software Interface Standard. CMSIS was created and is maintained by ARM.

This example creates a sine wave with noise added, and then the noise is filtered out. The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer.

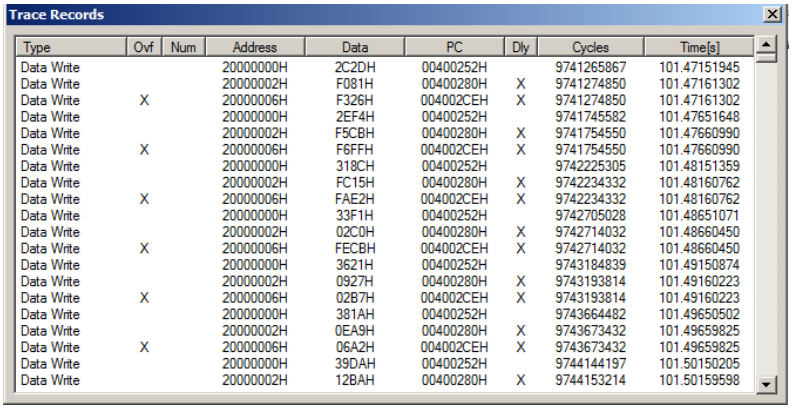
This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. Source code is provided.

To obtain this example, go to **www.keil.com/appnotes/docs/apnt_231.asp** Put it in the ...\Hitex\XMC-HiLight\ directory.

1. Open the project file sine: C:\Keil\ARM\Boards\Hitex\XMC-HiLight\DSP\sine.uvproj
2. Build the files. There will be no errors or warnings.
3. Program the XMC4000 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
4. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
5. Click on the RUN icon. Open the Logic Analyzer window.
6. Open Watch 1 window by selecting View/Watch/Watch 1 if it is not already open.
7. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom Out for an appropriate display. Displayed are 4 global variables: sine, noise, disturbed and filtered.
8. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.



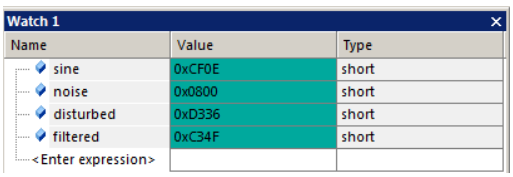
9. Open the Trace Records window and the Data Writes to the four variables are listed as shown here:
10. Leave the program running.
11. Close the Trace Records window.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	2C2DH	00400252H		9741265867	101.47151945
Data Write			20000002H	F081H	00400280H	X	9741274850	101.47161302
Data Write	X		20000006H	F326H	004002CEH	X	9741274850	101.47161302
Data Write			20000000H	2EF4H	00400252H		9741745582	101.47651648
Data Write			20000002H	F5CBH	00400280H	X	9741754550	101.47660990
Data Write	X		20000006H	F6FFH	004002CEH	X	9741754550	101.47660990
Data Write			20000000H	318CH	00400252H		9742225305	101.48151359
Data Write			20000002H	FC15H	00400280H	X	9742234332	101.48160762
Data Write	X		20000006H	FAE2H	004002CEH	X	9742234332	101.48160762
Data Write			20000000H	33F1H	00400252H		9742705028	101.48651071
Data Write			20000002H	02C0H	00400280H	X	9742714032	101.48660450
Data Write	X		20000006H	FECBH	004002CEH	X	9742714032	101.48660450
Data Write			20000000H	3621H	00400252H		9743184839	101.49150874
Data Write			20000002H	0927H	00400280H	X	9743193814	101.49160223
Data Write	X		20000006H	02B7H	004002CEH	X	9743193814	101.49160223
Data Write			20000000H	381AH	00400252H		9743664482	101.49650502
Data Write			20000002H	0EA9H	00400280H	X	9743673432	101.49659825
Data Write	X		20000006H	06A2H	004002CEH	X	9743673432	101.49659825
Data Write			20000000H	39DAH	00400252H		9744144197	101.50150205
Data Write			20000002H	12BAH	00400280H	X	9744153214	101.50159598

TIP: The ULINK*pro* display will be different and the program must be stopped to update it.

The Watch 1 window will display the four variables updating in real time as shown below:

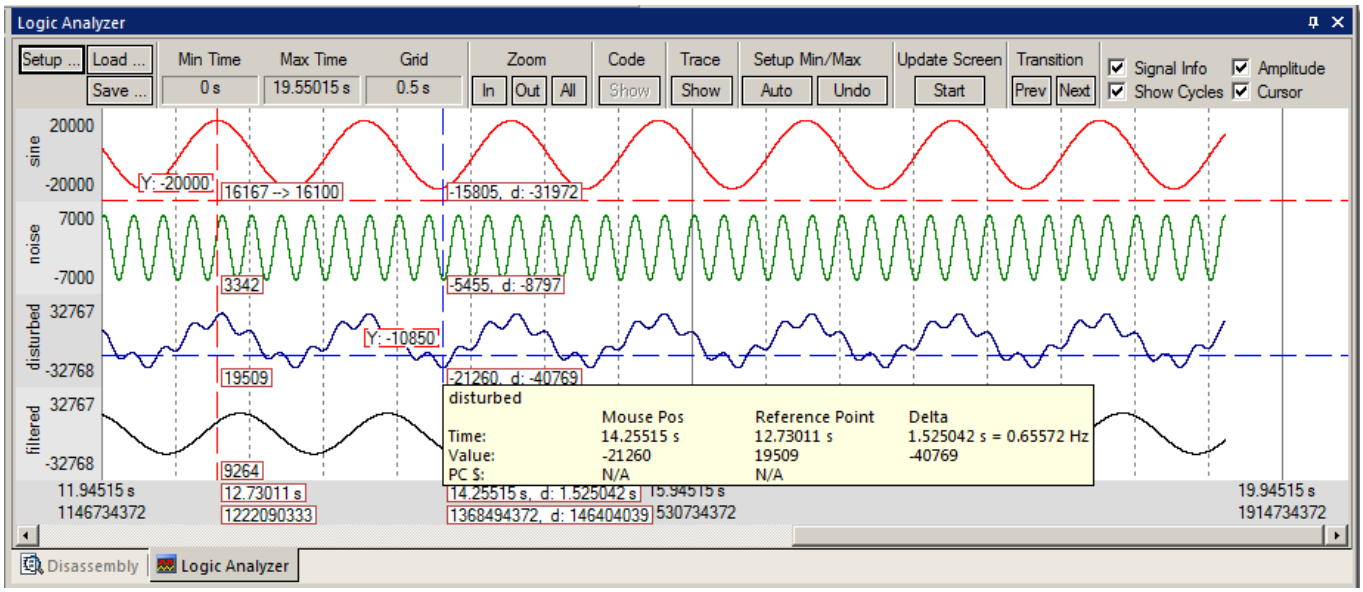


Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short
<Enter expression>		

Copyright © 2012 ARM Ltd. All rights reserved

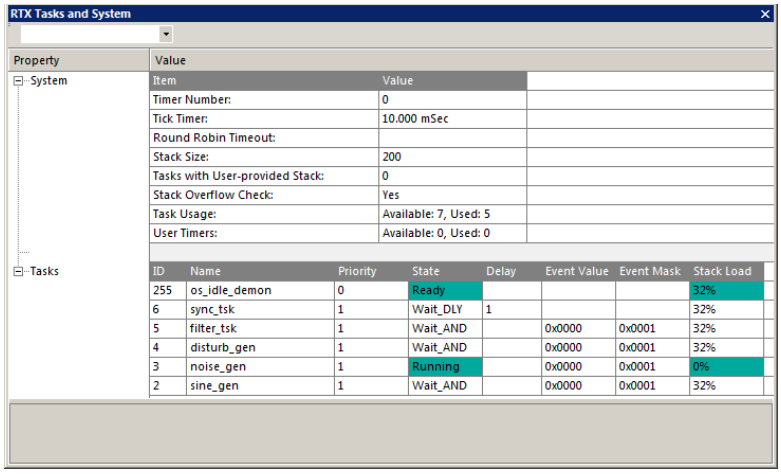
Signal Timings in Logic Analyzer (LA):

1. In the LA window, select Signal Info, Show Cycles, Amplitude and Cursor.
2. Click on STOP in the Update Screen box. You could also stop the program but leave it running in this case.
3. Click somewhere in the LA to set a reference cursor line.
4. Note as you move the cursor various timing information is displayed as shown below:



RTX Tasks and System:

5. Click on Start in the Update Screen box to resume the collection of data.
6. Open Debug/OS Support and select RTX Tasks and System. A window similar to below opens up. You probably have to click on its header and drag it into the middle of the screen.
7. Note this window does not update: nearly all the processor time is spent in the idle daemon: it shows it is Running. The processor spends relatively little time in other tasks. You will see this illustrated clearly on the next page.
8. Set a breakpoint in one of the tasks in DirtyFilter.c by clicking in the left margin on a grey area.
9. Click on Run and the program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the noise_gen task:
10. Clearly you can see that noise_gen was running when the breakpoint was activated.
11. Remove the breakpoint.



TIP: You can set/unset breakpoints while the program is running with any ULINK or J-Link.

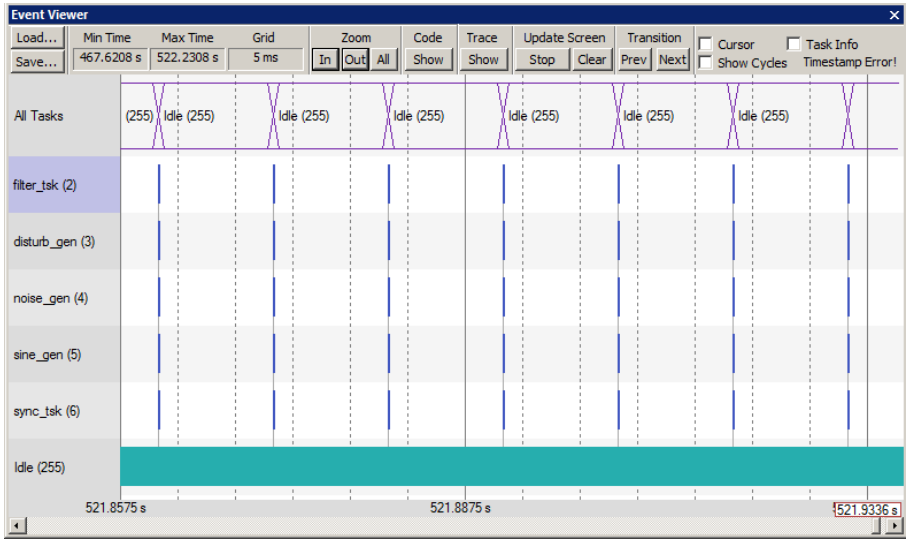
TIP: Recall this window uses the CoreSight DAP read and write technology to update this window. Serial Wire Viewer is not used and is not required to be activated for this window to display and be updated.

The Event Viewer does use SWV and this is demonstrated on the next page.

Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

Event Viewer:

1. Click on RUN.
2. Open Debug/OS Support and select Event Viewer. The window here opens up:
3. Note there is no Task 1 listed. Task 1 is main_tsk and is found in DirtyFilter.c near line 208. It runs some RTX initialization code at the beginning and then deletes itself with os_tsk_delete_self(); found near line 195.



TIP: If the window is blank, exit and re-enter debug mode to refresh the screens. Click on RUN. Try cycling the power to the board.

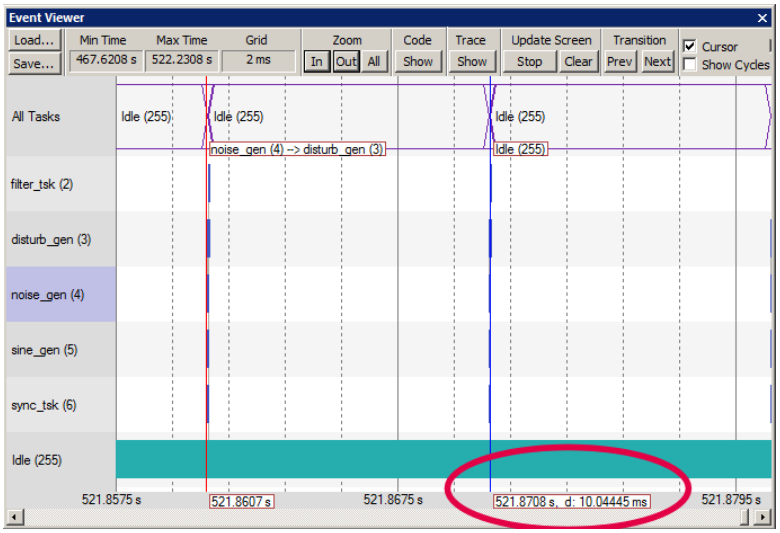
TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some bandwidth.

ULINK*pro* is much better with SWV bandwidth issues. It has been able to display both the Event and LA windows better. ULINK*pro* uses the faster Manchester format than the slower UART mode that ULINK2 or ULINK-ME uses.

1. Note on the Y axis each of the 5 running tasks plus the idle daemon. Each bar is an active task and shows you what task is running, when and for how long.
2. Click Stop in the Update Screen box.
3. Click on Zoom In so three or four tasks are displayed.
4. Select Cursor. Position the cursor over one set of bars and click once. A red line is set there:
5. Move your cursor to the right over the next set and total time and difference are displayed.
6. Note, since you enabled Show Cycles, the total cycles and difference is also shown.

The 10 msec shown is the SysTick timer value. This value is set in RTX_Conf_CM.c . The next page describes how to change this.

TIP: ITM Port 31enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port if you are not using the Event Viewer. This is normally a good idea.

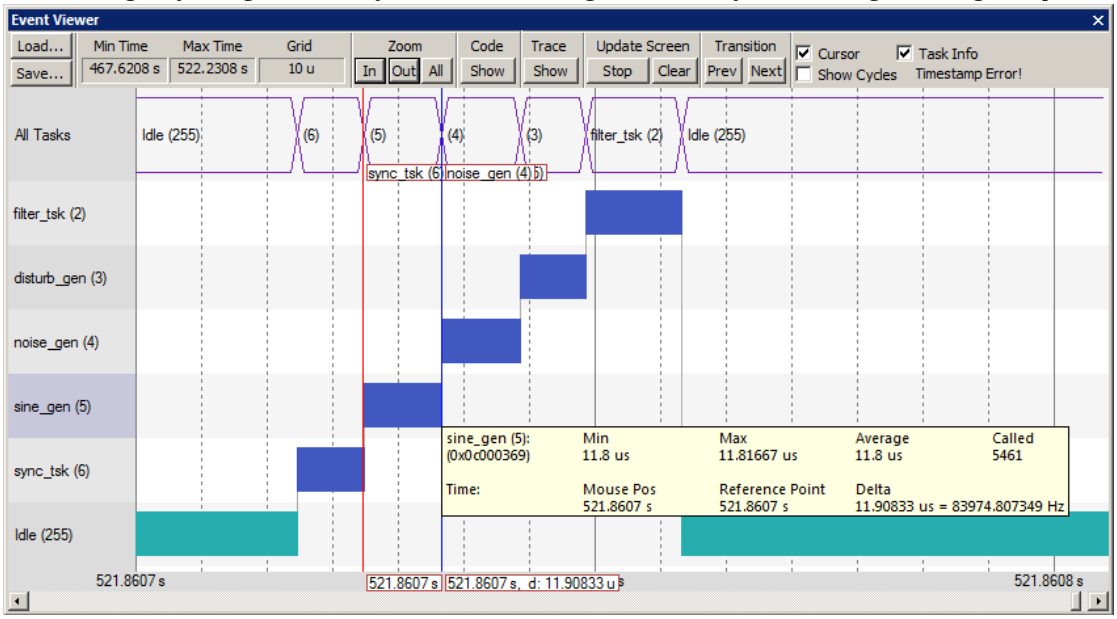


Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

Event Viewer Timing:

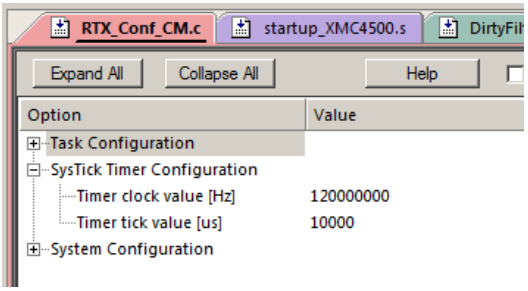
1. Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might/will occur. Using a ULINK*pro* will alleviate this problem as it processes SWV data faster.
2. Click on Zoom In until one set of tasks is visible as shown below:
3. Enable Task Info (as well as Cursor and Show Cycles from the previous exercise).
4. Note one entire sequence is shown. This screen is taken with ULINK-ME.
5. Click on a task to set the cursor and move it to the end of the task (or anywhere if you like). The time difference is noted and a Task Info box is displayed as shown below:

The Event Viewer can give you a good idea if your RTOS is configured correctly and running in the right sequence.



Changing the SysTick Timer:

1. Stop the processor and exit debug mode.
2. Open the file RTX_Conf_CM.c from the Project window. You can also select File/Open in C:\Keil\ARM\Boards\Hitex\XMC-HiLight\DSP.
3. Select the Configuration Wizard tab at the bottom of the window. See page 14 for an explanation on how the Wizard works.
4. This window opens up. Expand SysTick Timer Configuration.
5. Note the Timer tick value is 10,000 usec or 10 msec.
6. Change this to 20,000.



TIP: The 120,000,000 is the CPU clock speed.

7. Rebuild the source files and program the Flash.
8. Enter debug mode and click on RUN.
9. When you check the timing of the tasks in the Event Viewer window as you did on the previous page, they will now be spaced at 20 msec.

TIP: The SysTick is a dedicated timer on Cortex-M processors that is used to switch tasks in an RTOS. It does this by generating an exception 15. You can view these exceptions in the Trace Records window by enabling EXCTRC in the Trace Configuration window.

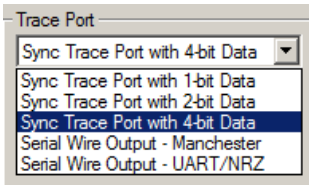
1. Set the SysTick timer back to 10,000. You will need to recompile the source files and reprogram the Flash.

Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

Part D) Using the ULINK*pro* with ETM Trace:

The examples previously shown with the ULINK2 will also work with the ULINK*pro*. There are two major differences:

1) The window containing the trace frames is now called Trace Data. More complex and useful filtering is available.
2) The SWV (Serial Wire Viewer) data is sent out the SWO pin to the ULINK2 using UART encoding. The ULINK*pro* can send SWV data either out the SWO pin using Manchester encoding ***or*** out the 4 bit Trace Port. This is configured in the Trace configuration window as shown below.
   ETM data is always sent out the Trace Port and if ETM is being used, SWV frames are also sent out this port.

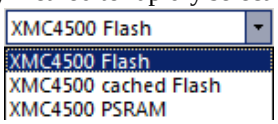


ULINK*pro*:

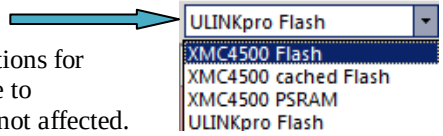
1) Offers faster Flash programming than the ULINK2.
2) Has all Serial Wire Viewer features as the ULINK2 with much higher throughput.
3) Adds ETM trace which provides records of all Program Counter values.
4) **Code Coverage**: were all the assembly isntructions executed ? Untested code can be dangerous. Code Coverage is automatic with ETM.
5) **Performance Analysis**: where did the processor spent its time ? This is displayed in a graphical format.
6) **Execution Profiling**: How long instructions, ranges of instructions, functions or C source code took in both time and and number of times these were executed.

1) Target Selector Box: Creating this is optional: you can make changes directly to an existing target:

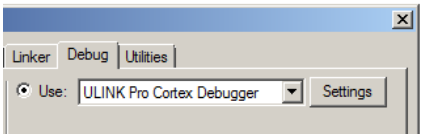
Beside the Target Options icon is the Target Selector drop down menu as shown here. The entries shown select between different settings in the Target Options menus and source files associations. This is a handy method to rapidly select different configurations. **Note:** Cached Flash offers the fastest Flash program execution.



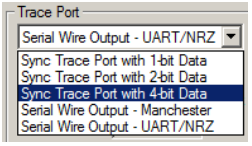
To create your own Target Option:

1. Open C:\Keil\ARM\Boards\Hitex\XMC-HiLight\Blinky\Blinky.uvproj.
2. Select the target you want the next one to copy from. In this case, choose XMC4500 Flash.
3. Select Project/Manage and select Components,….
4. Select the New (Insert) icon and enter ULINKpro Flash and press Enter. You can choose any name.
5. Click on OK to close this window. There is a new entry in the Target box: 

6. Select ULINKpro Flash in the Target box. Now you have a copy of the Options for Target configuration in the ULINKpro target name. Any changes you make to ULINKpro Flash settings will be saved for later recall. XMC4500 Flash is not affected.
7. **To configure your new Target Option:**
8. Select Options for Target and select the Debug tab.
9. Select ULINK Pro Cortex Debugger as shown here:



10. Click on the Settings box. Select SWJ and SW.
    a. Click on the Trace tab. Set Core Clock: to 120 MHz and select Trace Enable. To send SWV data out the 1 bit SWO pin: select *Serial Wire Output – Manchester.*
    b. To send SWV **and/or** ETM trace data out the 4 bit trace port: select *Sync Trace Port with 4 bit data*. Then select the ETM Trace Enable box.



11. Click on OK and if you selected the 4 bit Trace Port, add the file XMC4500_TP.ini to the Initialization File box. You can use the Browse icon. This file is provided with the DSP example code.
12. Click on the Utilities tab to access the Flash programmer.
13. Select ULINK Pro Debugger and select Settings: If you need to add a Flash algorithm: select Add.
14. Select the Flash programming algorithm: XMC4500 1024kB Flash and 1024kB cached Flash. Then select Add.
15. Select OK twice and ULINK*pro* is now configured for use with the trace: either/and SWV and ETM.
16. Select XMC4500 Flash target and you are switched back using the ULINK2-ME. You can create many targets.

Copyright © 2012 ARM Ltd. All rights reserved

2) Blinky Example with ETM: (available only with ULINK*pro* and ETM Trace or Keil Simulator)

1. Connect the ULINK*pro* to the XMC4000 board using the 20 pin Cortex Debug+ETM connector X200.
2. Configure your target as describe on the previous page. Blinky.uvproj will still be the active project.
3. Select ULINKpro Flash in the target window.
4. Add these two lines to the file system_XMC4500.c found : in C:\Keil\ARM\Boards\Hitex\XMC-HiLight\Blinky. You can easily edit this file in µVision. This is to configure the ETM Trace Port pins for high speed operation.

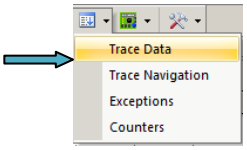
```
PORT6->PDR0 = 0x20022200; /*set P2 and P6 to Strong Driver, Sharp Edge for efficient ETM */
PORT2->PDR1 = 0x22220222;
```

5. A good place is just after the instruction to Disable Branch Prediction near Line 150:

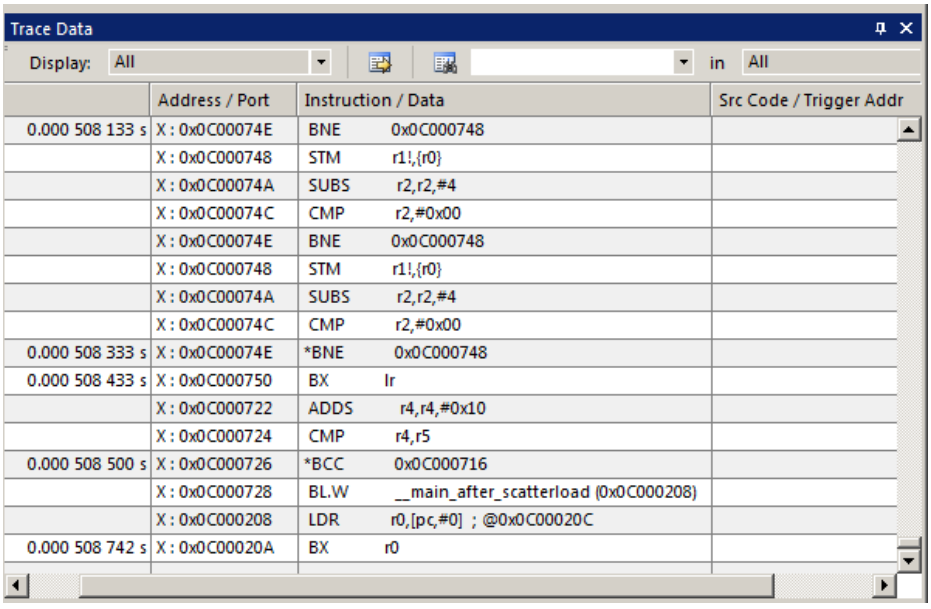
```
/* Disable branch prediction - PCON.PBS = 1 */
PREF->PCON |= (PREF_PCON_PBS_Msk);
```

6. Select File/Save All.
7. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
8. Program the XMC4000 Flash by clicking on the Load icon:

Progress will be indicated in the Output Window.
9. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
10. DO NOT CLICK ON RUN YET !!!
11. Open the Data Trace window by clicking on the small arrow beside the Trace Windows icon.



12. Examine the Instruction Trace window as shown below: This is a complete record of all the program flow since RESET until µVision halted the program at the start of main() since Run To main is selected.



	Address / Port	Instruction / Data	Src Code / Trigger Addr
0.000 508 133 s	X : 0x0C00074E	BNE 0x0C000748	
	X : 0x0C000748	STM r1!,{r0}	
	X : 0x0C00074A	SUBS r2,r2,#4	
	X : 0x0C00074C	CMP r2,#0x00	
	X : 0x0C00074E	BNE 0x0C000748	
	X : 0x0C000748	STM r1!,{r0}	
	X : 0x0C00074A	SUBS r2,r2,#4	
	X : 0x0C00074C	CMP r2,#0x00	
0.000 508 333 s	X : 0x0C00074E	*BNE 0x0C000748	
0.000 508 433 s	X : 0x0C000750	BX lr	
	X : 0x0C000722	ADDS r4,r4,#0x10	
	X : 0x0C000724	CMP r4,r5	
0.000 508 500 s	X : 0x0C000726	*BCC 0x0C000716	
	X : 0x0C000728	BL.W __main_after_scatterload (0x0C000208)	
	X : 0x0C000208	LDR r0,[pc,#0] ; @0x0C00020C	
0.000 508 742 s	X : 0x0C00020A	BX r0	

In this case, the last instruction to be executed. (BX r0). In the Register window the PC will display the value of the next instruction to be executed (BL.W at 0x0C00_049A in my case). This also shows in the Disassembly window. Without the trace, it is difficult to ascertain the last instructions executed. Can you see how easy this is to use ?

13. Click on Single Step once or a few times.

The instruction BL.W will display with its source code: ADC_Init();

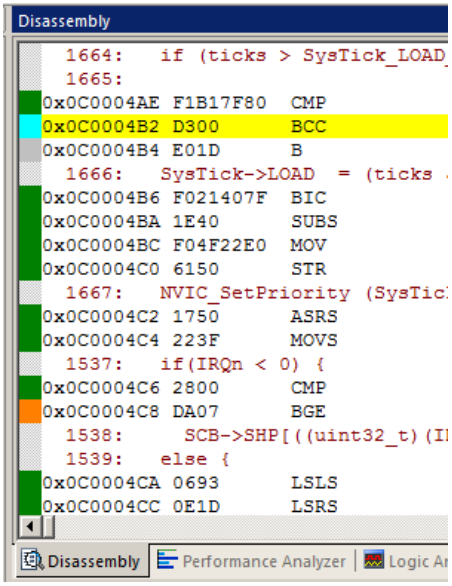
14. Scroll to the top of the Trace Data window. This is the very first instruction executed after RESET. In my case it is 0x0C00 0214 LDR. Double click on this line to show it in the Disassembly and Source windows.

Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

3) Code Coverage: (available only with ULINK*pro* and ETM Trace or Keil Simulator)

15. Click on the RUN icon. After a second or so stop the program with the STOP icon.
16. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin:
17. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not.

Colour blocks indicate which assembly instructions have been executed.



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

In the window on the right you can easily see examples of each type of Code Coverage block and if they were executed or not and if branches were taken (or not).

Why was the branch BCC always taken resulting in 0x0C00 04B4 ***never*** being executed ? Or why the branch BGE at 0x0C00 04C8 was never taken ? You should devise tests to execute these instructions. Untested code can be very dangerous.

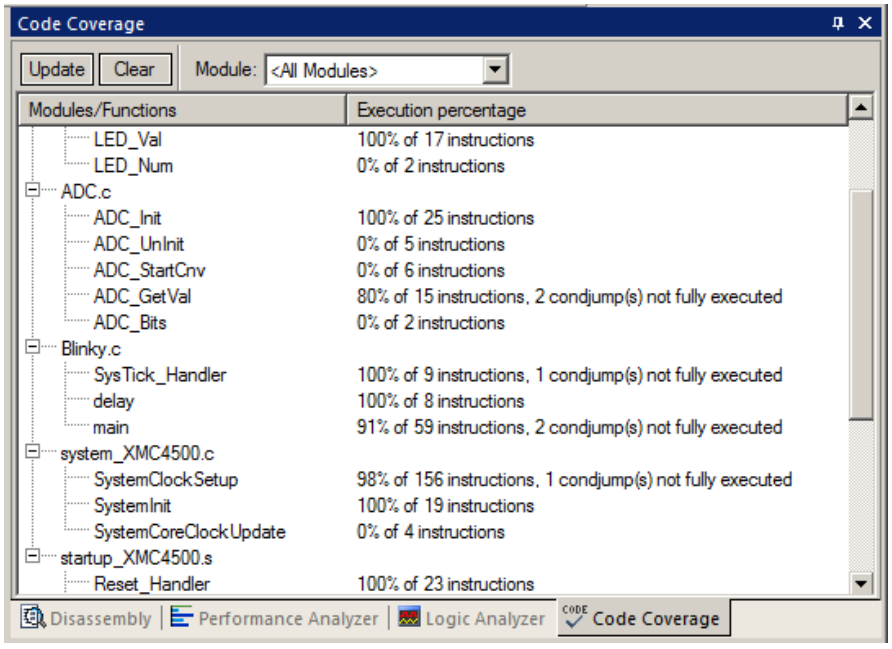
Code Coverage tells what assembly instructions were executed. It is important to ensure all assembly code produced by the compiler is executed and tested. You do not want a bug or an unplanned circumstance to cause a sequence of untested instructions to be executed. The result could be catastrophic as unexecuted instructions are be tested. Some agencies such as the US FDA require Code Coverage for certification.

Good programming practice requires that these unexecuted instructions be identified and tested.

Code Coverage is captured by the ETM. Code Coverage is also available in the Keil Simulator.

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage. Note your display may look different due to different compiler options.

In a future release of MDK, it will be possible to save this data to a file. Contact Keil tech support.



Copyright © 2012 ARM Ltd. All rights reserved

4) Performance Analysis (PA): (available only with ULINK*pro* and ETM Trace or Keil Simulator)

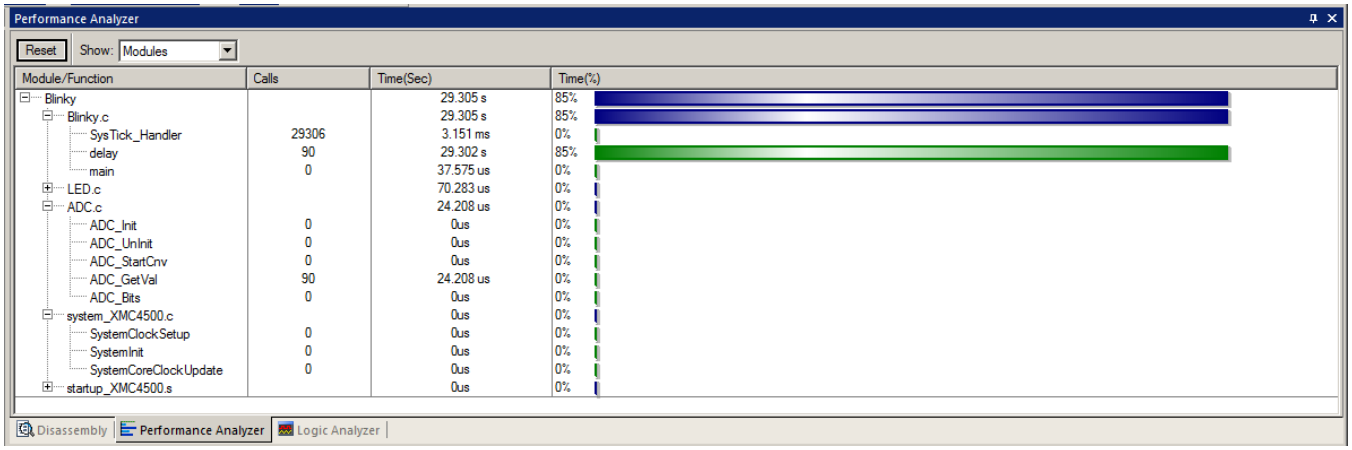
Performance Analysis tells you how much time was spent in each function. The data can be provided by either the SWV PC Samples or the ETM. If provided by the SWV, the results will be statistical and more accuracy is improved only with longer runs. Small loops will likely be entirely missed. ETM provides complete Performance Analysis. Keil provides only ETM PA for completeness and accuracy.

Keil provides Performance Analysis with the µVision simulator or with ETM and the ULINK*pro*. The number of total calls made as well as the total time spent in each function is displayed. A graphical display is generated for a quick reference. If you are optimizing for speed, obviously work first on those functions taking the longest time to execute. PA tells you which ones very quickly.

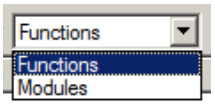
TIP: You can also obtain timings from ETM trace. ETM frames can be saved to a ASCII file. Also see Execution Profiling on the next page.

TIP: For ARM compiler optimizations, see www.keil.com/appnotes/files/apnt202.pdf.

1. Use the same setup as used with Code Coverage. Click on RUN if necessary.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Select the Reset icon to clear the PA window. The PA will start to fill with data in real-time.
4. Expand some of the module names as shown below.
5. Note the execution information that has been collected in this initial short run. Both times and number of calls is displayed.



6. We can tell that most of the time at this point in the program has been spent in the delay routine.
7. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the information. No code stubs are needed in your source files.
8. Select Functions from the pull down box as shown here and notice the difference.
9. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.
10. When you are done, exit Debug mode.



Remember, you also get Code Coverage, Performance Analysis, Execution Profiling and Trace with the Keil Simulator.

Super TIP: You can also click on the RESET icon but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

When you click on the RESET icon, the Initialization File XMC4500_TP.ini ***will no longer be in effect*** and this will cause SWV and/or ETM to stop working through the trace port. Exiting and re-entering Debug mode executes the .ini script again.

This ini file is not needed to operate SWV through *Serial Wire Output – Manchester* **. It is only needed for the 4 bit Trace Port operation and therefore only with ULINK*pro*.**

TIP: For more detailed information down to each assembly instruction, investigate Execution Profiling on the next page.

Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

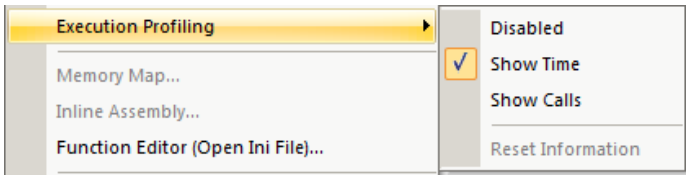
5) Execution Profiling: (available only with ULINK*pro* and ETM Trace or Keil Simulator)

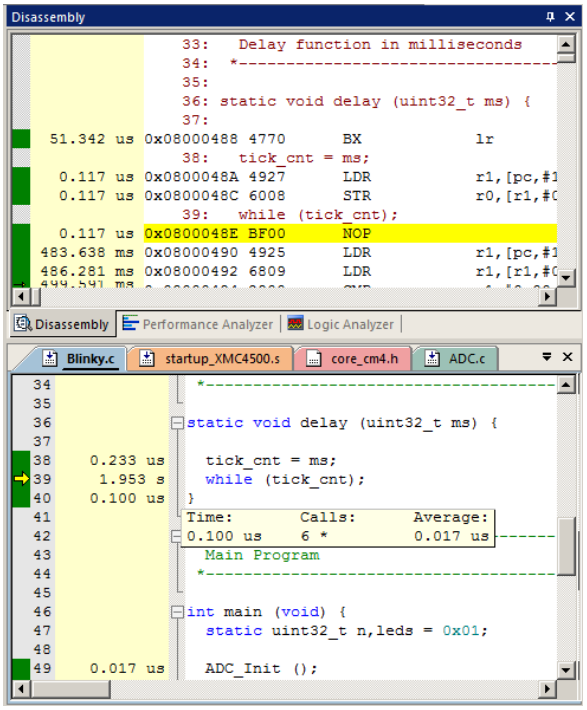
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace. It is possible to group source lines (called collapse) to get combined times and number of calls. This is called Outlining. The µVision Simulator also provides Execution Profiling.

1. Enter Debug mode.
2. Select Debug/Execution Profiling/Show Time.
3. In the left margin of the disassembly and C source windows will display various time values.
4. Click on RUN.
5. The times will start to fill up as shown below right:
6. Click inside the yellow margin of Blinky.c to refresh it.
7. This is done in real-time and without stealing CPU cycles.
8. Hover the cursor over a time and ands more information appears as in the yellow box here:

Time:	Calls:	Average:
19.599 s	139910257 *	0.140 µs

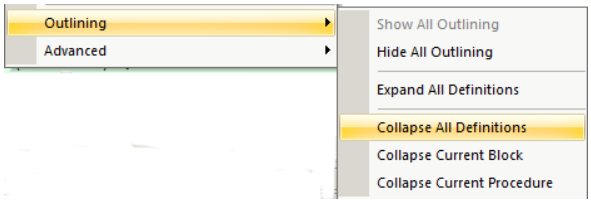
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the margin.





Outlining:

1) Outlining is used to aggregate various functions so their times are summed.
2) Right click an area in a ource file and select Outlining and then Collapse Current Block as shown below:



3) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
4) Click on the + to expand it.
5) Stop the program and exit Debug mode.

Copyright © 2012 ARM Ltd. All rights reserved
www.keil.com

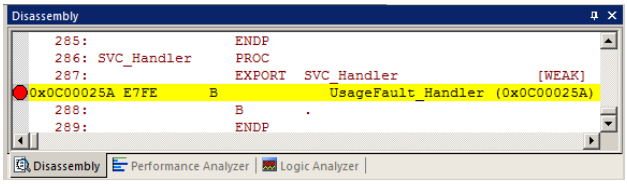
6) In-the-Weeds Example: (available only with ULINK*pro* and ETM Trace or Keil Simulator)

Some of the hardest problems to solve are those when a crash has occurred and you have no clue what caused this. You only know that it happened and the stack is corrupted or provides no useful clues. Modern programs tend to be asynchronous with interrupts and RTOS task switching plus unexpected and spurious events. Having a recording of the program flow is useful especially when a problem occurs and the consequences are not immediately visible. Another problem is detecting race conditions and determining how to fix them. ETM trace handles these problems and others easily and is not hard to use.

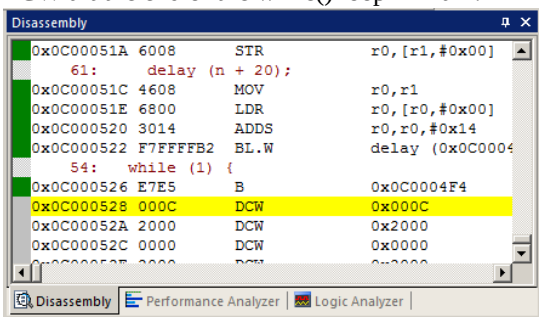
If a fault occurs, the CPU will end up at the address specified in the fault vectors located starting at 0x0C00 0252 and are created in startup_XMC4500.s. These vectors are normally a branch to itself and this Branch instruction will run forever. The trace buffer will save millions of the same branch instructions. This is not useful. We need to stop the CPU at this point.

In our example, we will use the Usage Fault vector located at0x0C00 025A. Open startup_XMC4500.s and set a breakpoint on it as shown here: If a Usage Fault occurs, the program will be stopped here ***before the Branch is executed***.

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



1. Use the Blinky example as on the previous page. Stop the program and stay in Debug mode.
2. Select the XMC4500.s tab in the source file windows.
3. Locate the Usage Fault vector near line 287 in the disassembly window or in startup_XMC4500.s near line 284.
4. Set a breakpoint at this point. A red circle will appear as shown above.
5. Run the Blinky example for a few seconds and click on STOP.
6. In the Disassembly window, scroll down until you find a series of DCW aft the end of the while() loop in main:
7. Right click on the DCW shown here at 0x0C00 0528 and select Set Program Counter. This DCW will be the next instruction executed, but it not an instruction.
8. Click on RUN and almost immediately the program will stop on the Usage Fault exception branch instruction.
9. Examine the Data Trace window and you find this DCW plus several more that execution was attempted as shown here:
10. Note the Branch at the Hard Fault does not show in the trace window because a hardware breakpoint does execute the instruction it is set to therefore it is not recorded in the trace buffer.
11. Click on single step and you will see this B executed and recorded.



The frames above the DCW are a record of all previous instructions executed and tells you the complete program flow.

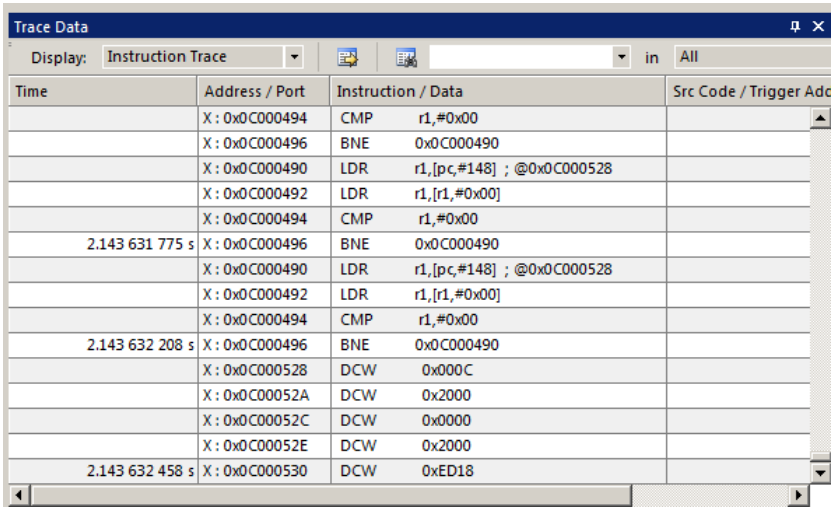
This is a very simple example but this clearly shows the advantages of using a quality trace.

1. To repeat this demonstration, exit Debug mode and re-enter and repeat the steps. This resets the trace.

TIP: Processors do not like executing memory locations that are not instructions. Another useful way to crash a system (for demo purposes) is to execute a POP instruction out of order.

To do this: use Step Out to exit back to main(). You will need to set a breakpoint on the Hard Fault vector.

TIP: Keil ETM trace is undergoing significant improvements at this time (summer 2012). Please check with later versions of MDK (after 4.54) for progress.



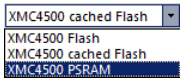
Copyright © 2012 ARM Ltd. All rights reserved

Part E: Additional Information:

1) Cache Flash and Configuring Memory Areas:

XMC4000 Cache Flash is easily handled by µVision. The XMC4000 has a 4 KB cache to accelerate Flash accesses.

The Blinky example has several targets as shown here:



They are:

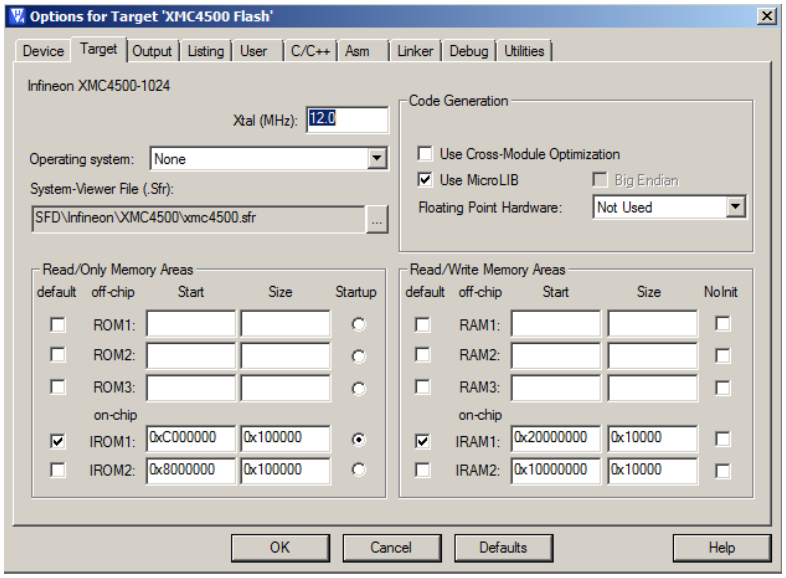
1. XMC4500 Flash: executable code is placed in uncached Flash at 0x0C000 0000
2. XMC4500 cached Flash: executable code is placed in cached Flash at 0x08000 0000. 4 KB cache is enabled.
3. XMC4500 PSRAM: executable code is placed at code RAM (PSRAM) at 0x0C000 0000.

As each target is selected, its set of Options for Target is selected. They can be accessed in Edit mode by clicking on .

Among other things, this is how memory usage is configured.

1. Select XMC4500 Flash and click on the Options for Target icon. Select the Target tab. This window opens:

Flash Memory: cached and uncached:



2. Note the address area on the left side: Read/Only Memory Areas. This is where your executable is stored: either Flash or RAM.
3. In this window, IROM1 0x0C00 0000 is selected for your startup files and is available to all modules in your sources.
4. This uncached Flash area will be programmed with your executable.
5. If you select default and Startup for the IROM2 address 0x0800 0000, your executable will be programmed to this cached Flash area. The 4 KB Instruction cache will be automatically enabled and your program will run faster. ***This is a good default to use.***
6. If you click on close, and select the target XMC45000 cached Flash and open the Options for target window, you will see that IROM2 is now selected.
7. Select the Utilities tab. This is where the Flash programming algorithms are selected. Select Settings: and note there are two algorithms: one for cached and uncached Flash. µVision will choose the correct one to program.
8. See the XMC4000 datasheet from Infineon for more information on Cache Flash.

RAM Memory for program execution:

9. On the right, Read/Write Memory Areas is where your storage is, usually RAM. If you want to run your program in RAM, you can divide up the RAM into the left and right sides.
10. Exit this window by clicking on OK. Select the XMC PSRAM target selection and click on the Options for Target icon. Select the Target tab.
11. Note your program will now be executed at RAM address 0x0100 0000 and program RAM at 0x0200 0000.
12. You do not need to use the Load icon: programming RAM is done automatically by a .ini file. Click on the Debug tab to access this in the Initialization File: box.

Scatterfile:

ARM processors use a scatterfile to determine where in memory various parts of your program are located. uVision creates a scatterfile for you by using the address entries in the Target tab. To enable this, click on the Linker tab and select "Use Memory Layout from Target Dialog". You can also use your own scatterfile if you prefer.

TIP: Select MicroLIB in the Target tab to minimize the size of your executables.

Copyright © 2012 ARM Ltd. All rights reserved

2) Creating your own project from scratch: Using the Blinky source files:

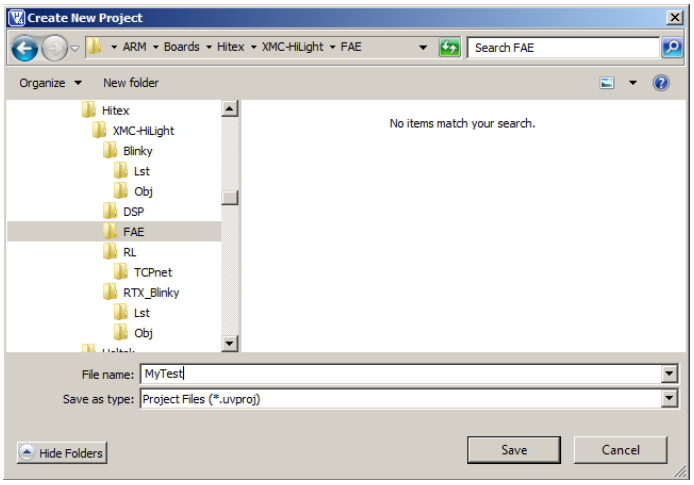
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run a bare Blinky example. It has an empty main() function so it does not do much. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using an RTOS. These instructions are for a XMC4500 processor.

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 that opens, go to the folder C:\Keil\ARM\Boards\Hitex\XMC-HiLight.

Create a new folder and name your project:

3. Right click inside this window 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. It will be empty.
5. Name your project in the File name: box. I called mine Mytest. You can choose your own name but you will have to keep track of it. This window is shown here:
6. Click on Save.



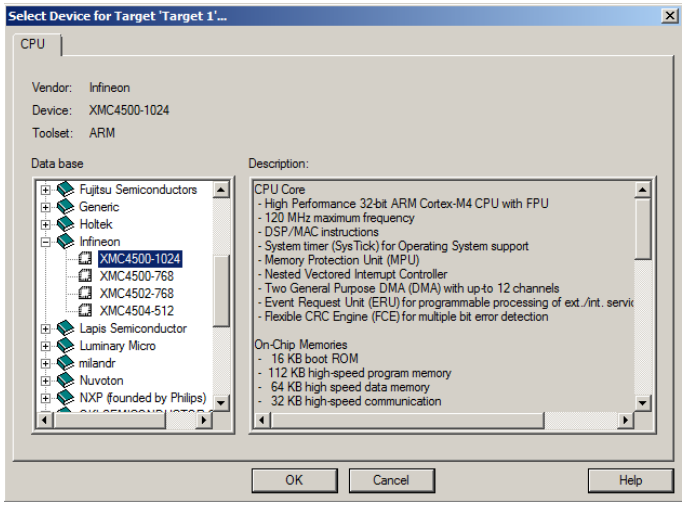
Select your processor:

7. The Select Device for "Target 1" opens up as shown at the bottom of this page:
8. This is the Keil Device Database® which lists all the devices Keil supports. You can create your own if desired for processors not released yet.
9. Locate the Infineon directory, open it and select a XMC4500-1024 (or the device you are using). Note the device features are displayed.
10. Select OK.
    µVision will configure itself to this device.

Select the startup file:

11. A window opens up asking if you want to insert the default XMC4500 startup file to your project. Click on "Yes". This will save you some time.
12. In the Project Workspace in the upper left hand of µVision, expand the folders Target 1 and Source Group 1 by clicking on the "+" beside each folder.
13. We have now created a project called Mytest with the target hardware called Target 1 with one source assembly file startup_XMC4500.s and using the Infineon processor you chose.

TIP: You can create more target hardware configurations and easily select them. This can include multiple Options settings, simulation and RAM operations. See Projects/Manage/Components



Rename the Project names for convenience:

14. Click once on the name "Target 1" (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose XMC4500 Flash. Press Enter to accept this change. Note the Target selector in the main µVision window also changes to XMC4500 Flash. This is your first target.
15. Similarly, change Source Group 1 to Startup. This will add some consistency to your project with the Keil examples. You can name these or organize them differently to suit yourself.
16. Select File/Save All.

Continued on the next page…

Copyright © 2012 ARM Ltd. All rights reserved

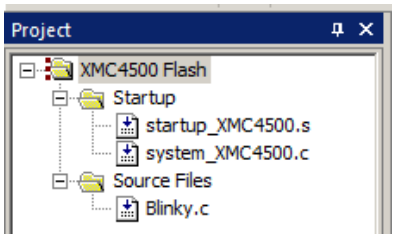
www.keil.com

Select the source files and debug adapter:

1. Using MS Explore (right click on Windows Start icon), copy blinky.c and system_XMC4500.c from C:\Keil\ARM\Boards\Hitex\XMC-HiLight\Blinky to your C:\Keil\ARM\Boards\Hitex\XMC-HiLight\FAE folder.

Source Files:

2. In the Project Workspace in the upper left hand of µVision, right-click on "XMC4500 Flash" and select "Add Group". Name this new group "Source Files" and press Enter. You can name it anything. Keil has no restrictions.
3. Right-click on "Source Files" and select **Add files to Group "Source Files**".
4. Select the file Blinky.c and click on Add (once!) and then Close.

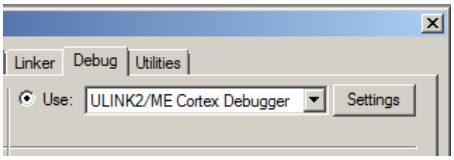


System File:

5. Right-click on "Startup" and select **Add files to Group "Source Files**".
6. Select system_XMC4500.c and click on Add (once!) and then Close.
7. Your Project window will look similar to the one shown here:

Select your Debug Adapter:

1. By default the simulator is selected when you create a new µVision project. You probably need to change this to a Debug adapter such as a ULINK2, ULINK*pro* or Segger J-Link.
2. Select Options for Target or ALT-F7 and select the Debug tab. Select ULINK/ME Cortex Debugger as shown below: If you are using another adapter, select the appropriate adapter from the pull-down list.
3. Select this debug adapter (as opposed to selecting the Simulator) by checking the circle just to the left of the word "Use:" as shown in the window to the right:
4. Select Run to Main (unless you do not want this)
5. If you have a debug adapter connected, select Settings and select SWJ and SW. JTAG does not work with the XMC4000. Click on OK once.
6. Select the Utilities tab and select the appropriate debug adapter and the proper Flash algorithm for your processor. Click OK once.
7. Click on the Target tab and select Use MicroLIB for smaller programs. See www.keil.com/appnotes/files/apnt202.pdf for details.
8. Click on OK.



Modify Blinky.c

1. Double-click the file Blinky.c in the Project window to open it in the editing window or click on its tab if open.
2. Delete everything in Blinky.c except the main () function to provide a basic platform to start with:

```
#include <XMC4500.h>            /* XMC4500 Definitions                */

/*----------------------------------------------------------------------
  Main Program
 *----------------------------------------------------------------------*/

int main (void) {

   while (1) {                                /* Loop forever                */
     }
}
```

3. Select File/Save All.

Compile and run the program:

1. Compile the source files by clicking on the Rebuild icon. You can also use the Build icon beside it.
2. Program the XMC4500 Flash by clicking on the Load icon: Progress will be indicated in the Output Window.
3. Enter Debug mode by clicking on the Debug icon.
4. Click on the RUN icon. Note: you stop the program with the STOP icon.
5. The program will run but since while(1) is empty – it does not do much. It consists of a NOP and Branch to itself. You can set a breakpoint on any assembly or source lines that have a darker grey box signifying assembly code.
6. You are able to add your own source code to create a meaningful project.

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

You can also configure a new RTX project from scratch using RTX_Blinky project.

Copyright © 2012 ARM Ltd. All rights reserved

3) Serial Wire Viewer and ETM Summary:

Serial Wire Viewer can see:

- Global and Static variables and 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 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:

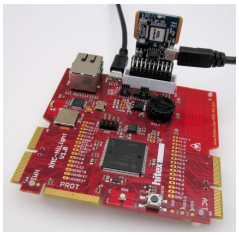
- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), corrupted stack. ***How did I get here ?***
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What 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 especially oif the stack is corrupted.
- **ETM trace with the ULINK*pro* is best for solving program flow problems.**
- Communication protocol and timing issues. System timing problems.

4) Useful Documents:

1. **The Definitive Guide to the ARM Cortex-M3** by Joseph Yiu. (he also has one for the Cortex-M0)
2. **MDK-ARM Compiler Optimizations: Appnote 202:** www.keil.com/appnotes/files/apnt202.pdf
3. **Lazy Stacking and Context Switching Appnote 298:** www.arm.com and search for Lazy Stacking.
4. **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.
5. **ARM Infocenter: http://infocenter.arm.com** Look for Application Notes and Cortex-M4 listings.

Copyright © 2012 ARM Ltd. All rights reserved

5) Keil Products:



Keil Microcontroller Development Kit (MDK-ARM™)

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

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.
  MDK also supports Segger J-Link Debug adapters and CMSIS-DAP.
- ***For special promotional pricing and offers, please contact Keil Sales.***

The Keil RTX RTOS is now provided under a Berkeley BSD type license. This makes it free.

All versions, including MDK-Lite, includes Keil RTX RTOS *with source code !*

Keil provides free DSP libraries for the Cortex-M0, Cortex-M3 and Cortex-M4.

Call Keil Sales for details on current pricing, specials and quantity discounts. Sales can also provide advice about the various tools options available to you. They will help you find various labs and appnotes that are useful.

All products are available from stock.

All products include Technical Support for 1 year. This is easily renewed.

Call Keil Sales for special university pricing. Go to www.arm.com and search for university to view various programs and resources.

Keil supports Infineon Cortex-M4 processors. Keil supports many other Infineon processors including 8051and C166 series. See the Keil Device Database® on www.keil.com/dd for the complete list of Infineon support. This database 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.

For more information:

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

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

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

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

---

Copyright © 2012 ARM Ltd. All rights reserved