

Introduction: *For the latest version of this document:* www.keil.com/apnotes/docs/apnt_229.asp

The purpose of this lab is to introduce you to the Atmel Cortex™-M3 processor using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) on the ATSAM3X processor. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. See www.keil.com/atmel for more labs.

Keil MDK supports and has examples for most Atmel ARM processors. Check the Keil Device Database® on www.keil.com/dd for the complete list which is also included in MDK: in µVision select Project/Select Device for target...

Linux: SAM9 processors running Linux, Android and bare metal are supported by ARM DS-5™. www.arm.com/ds5.

Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into the full, unrestricted version. Contact Keil sales for a temporary full version license if you need to evaluate MDK with programs greater than 32K or with Keil middleware.

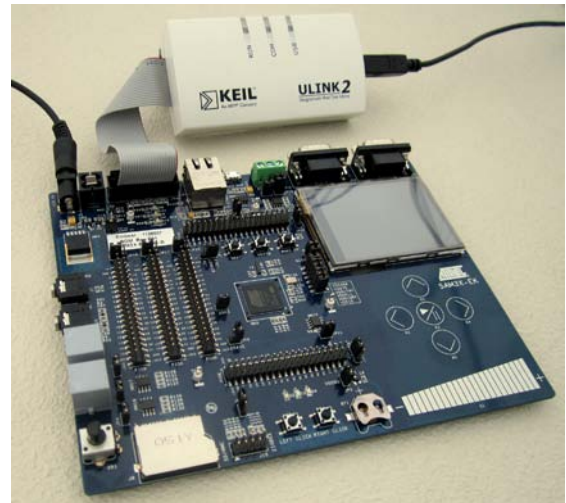
Middleware: MDK Professional contains middleware libraries including TCP/IP stack, CAN drivers, a Flash file system and USB drivers. Contact Keil sales for information regarding middleware for your processor. <http://www.keil.com/arm/mdk.asp>.

RTX RTOS: All variants of MDK contain the full version of RTX with Source Code. See www.keil.com/rl-arm/kernel.asp.

Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M users:

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is a turn-key product.
2. A full feature Keil RTOS called RTX is included with MDK. RTX comes with a BSD type license. Source code is provided.
3. Serial Wire Viewer trace capability is included.
4. RTX Kernel Awareness window. It is updated in real-time.
5. Choice of adapters: ULINK™ 2, ULINK-ME, ULINK_{pro}, Segger J-Link and SAM-ICE (version 6 or later for SWV).
6. Kernel Awareness is available for Keil RTX, CMX, Quadros, Micrium and FreeRTOS. All RTOSs will compile with MDK.
7. Keil Technical Support is included for one year and is easily renewable. This helps you get your project completed faster.
8. MDK includes support for Atmel ARM7 and ARM9 processors. Keil also supports many Atmel 8051 processors.



This document details these features:

SAM3X-EK with Keil ULINK2

1. Serial Wire Viewer (SWV). Real-time tracing updated while the program is running.
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 known as Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS.

Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and a timestamp. This information comes from the ARM CoreSight™ debug module integrated into the SAM3 and SAM4. SWV does not steal any CPU cycles and is completely non-intrusive. (except for the ITM Debug printf Viewer).

CoreSight displays memory contents and variable values in real-time and these can be modified on-the-fly. SWV is supported by the Keil ULINK2, ULINK-ME, ULINK_{pro}, Segger J-Link (Version 6 or later) and Atmel SAM-ICE Version 6 or later.

No special equipment or software is needed beyond MDK and one of these USB adapters. There is nothing extra to purchase. SWV provides you with the ability to do advanced software development beyond regular stop and go debugging.

Index:

1. Atmel Evaluation Boards & Keil Evaluation Software:	3
2. Software Installation:	3
3. CoreSight Definitions:	3
4. CMSIS: Cortex Microcontroller Software Interface Standard	3
5. Using Various USB adapters: J-Link, SAM-ICE, Keil ULINK:	4
1) Configuring a Segger J-Link and SAM-ICE:	4
2) Configuring a Keil ULINK2 or ULINK-ME:	5
3) Configuring a Keil ULINK <i>pro</i> :	6
6. Serial Wire Viewer (SWV) Configuration:	7
7. <i>Blinky</i> example using the Atmel SAM3X-EK:	8
8. Hardware Breakpoints:	8
9. Call Stack & Locals window:	9
10. Watch and Memory windows and how to use them:	10
11. How to view Local Variables in Watch and Memory windows:	11
12. View Variables Graphically with the Logic Analyzer (LA):	12
13. Watchpoints: Conditional Breakpoints:	13
14. RTX_Blinky: Keil RTX RTOS example:	14
15. RTX Kernel Awareness using RTX Viewer:	15
16. Logic Analyzer: View variables real-time in a graphical format:	16
17. Serial Wire Viewer (SWV) and how to use it:	17
1) Data Reads and Writes	17
2) Exceptions and Interrupts	18
3) PC Samples (program counter samples)	19
18. Segger J-Link and SAM-ICE Trace Windows:	20
19. Keil ULINK<i>pro</i> Trace Windows:	21
20. ITM (Instruction Trace Macrocell):	23
21. DSP SINE example using ARM CMSIS-DSP Libraries:	24
22. Creating your own project from scratch:	28
23. Serial Wire Viewer summary:	30
24. Useful Documents:	30
25. Keil Products and contact information:	31

Using this document:

1. The latest version of this document and the necessary example source files are available here:
www.keil.com/apnotes/docs/apnt_229.asp
2. Configuring various debug adapters starts on page 4.
3. ULINK2 is used by default.
4. The first exercise starts on page 8. You can go directly there if using a ULINK2 or a ULINK-ME.

1) Atmel Evaluation Boards & Keil Evaluation Software:

Keil currently provides board support for four SAM3 (Cortex-M3) and one SAM4 (Cortex-M4) boards as listed here:

SAM3N-EK, SAM3S-EK: SAM3U-EK, SAM3X-EK and SAM4S. SAM7 and SAM9 boards are also supported.

Example Programs: Keil provides various example programs. See C:\Keil\ARM\Boards\Atmel\ for the project files.

Most programs can be compiled with MDK-Lite: the free evaluation version. LCD_Blinky and \RL (the middleware) are exceptions. The Executable LCD_Blinky.axf is provided precompiled in the \Flash directory. Attempting to compile the LCD_Blinky project will erase the .axf. It is a good idea to back it up first.

\RL consists of Flash File 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.

Blinky: blinks a LED or series of LEDs.

RTX_Blinky uses the RTX RTOS in a stepper motor driver example. It has the Event Viewer: RTX kernel awareness.

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

2) Software Installation:

This document was written for Keil MDK 4.23 or later which contains μ Vision 4. The evaluation copy of MDK (MDK-Lite) is available free on the Keil website. Do not confuse μ Vision4 with MDK 4.0. The number “4” is a coincidence.

To obtain a copy of MDK go to www.keil.com/arm and select the “Download” icon located on the right side.

You can use the evaluation version of MDK-Lite and a ULINK2, ULINK-ME, ULINK*pro*, J-Link or SAM-ICE for this lab.

The ULINK*pro* adds Cortex-M ETM trace support. ETM is not currently implemented on the SAM3 processor family. The ULINK*pro* can be used with the SAM3 for SWV support and it also provides the fastest Flash programming and SWV speeds available. If you need fast Serial Wire Viewer data, ULINK*pro* is the best choice since it uses Manchester encoding.

3) CoreSight Definitions: It is useful to have a basic understanding of these terms:

- **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
- **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except for no Boundary Scan. SWD is referenced as SW in the μ Vision Cortex-M Target Driver Setup. See page 4, 2nd picture. The SWJ box must be selected. SWV must use SWD because of the TDIO conflict described in **SWO** below.
- **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
DAP: Debug Access Port. A component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. μ Vision uses the DAP to update memory, watch and RTOS kernel awareness windows in real-time while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed in your sources. You do not need to configure or activate DAP. μ Vision does this automatically when you select the function.
- **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 collect ETM frames and optionally SWV (rather than SWO pin).
- **ETM:** Embedded Trace Macrocell: Provides all the program counter values. Only the ULINK*pro* provides ETM.

Note: Current SAM3 and SAM4 Atmel processors do not have a Trace Port or ETM trace. They do fully support all other ARM CoreSight features. SAM-ICE and Segger J-Link V 6 or later supports Serial Wire Viewer.

4) CMSIS: Cortex Microcontroller Software Interface Standard

ARM CMSIS-DSP libraries are offered for all Cortex-M3 and Cortex-M4 processors.

CMSIS-RTOS provides standard APIs for RTOSs. CMSIS_DSP provides DSP libraries.

Atmel example software is CMSIS hardware abstraction layer compliant.

See www.arm.com/cmsis and www.onarm.com/cmsis/download/ for more information.

5) Using Various USB adapters: J-Link, SAM-ICE, Keil ULINK series:

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

This document will use a ULINK2. You can substitute a J-Link, SAM-ICE or ULINK pro with suitable adjustments.

1) Configuring a Segger J-Link and SAM-ICE: (the Atmel SAM-ICE is pictured on page 6)



Serial Wire Viewer is supported by hardware Version 6 or later only and only those with the black case. The main difference between V 6 through 8 is speed, so the later the version the better. The version number is printed on the Segger adapters.

The trace window with J-Link or SAM-ICE is different than that used with the Keil ULINK2 or ME. J-Link uses the same Instruction Trace window as used by ULINK pro . This window has more advanced features than the one presented by the ULINK2. The SAM-ICE and J-Link Trace Data window requires the program be stopped to be updated. All other display windows are the same as the ULINK2. Currently, SWV data writes or reads are not supported with J-Link or SAM-ICE.

USB Drivers: The Segger USB drivers are located in C:\Keil\ARM\Segger\USBDriver should you need them.


1. Assume the SAM-ICE or J-Link is connected to a powered up Atmel target board, μ Vision is running in Edit mode (as when first started – not in Debug mode) and you have selected a valid project. See the exercise on page 8.

Select the debug connection to the target:

2. Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select the J-Link as shown here: 
3. Select Settings and the next window below opens up. This is the control panel for the J-Link.
4. In Port: select SW. Serial Wire Viewer (SWV) will not work with JTAG. If you do not plan to use SWV, you can use JTAG.
5. Clicking on Auto Clk will select the highest JTAG/SWD speed possible. If debugging or Flash programming operation is unstable, select a lower speed. The Flash programming speed is affected the most by this setting.
6. 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 J-Link and the board. No number in the SN: box means μ Vision is unable to connect to the J-Link or SAM-ICE via USB.

TIP: To refresh this screen, change the option in Port: or click OK once to leave this screen and then re-enter it.

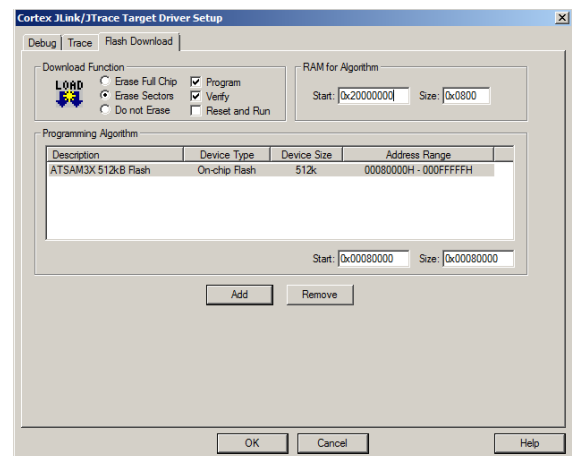
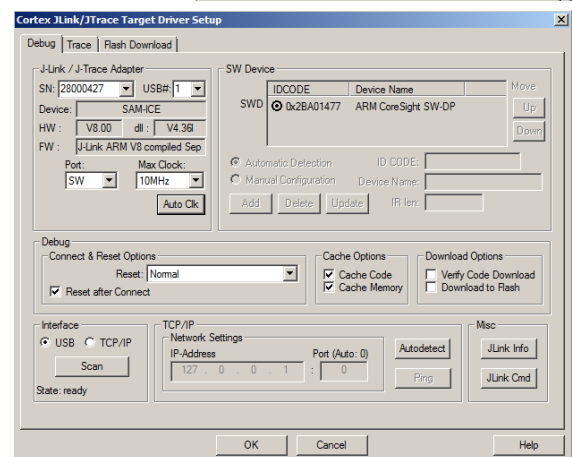
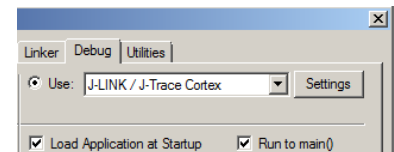
Configure the Keil Flash Programmer:

7. Click on OK once and select the Utilities tab.
8. Select the J-Link similar to Step 2 above.
9. Click Settings to select the programming algorithm.
10. Select Add and select the appropriate SAM3 Flash if necessary as shown here: 
11. ATSAM3X512kB Flash is for the SAM3X-EK board.
12. Click on OK once.

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

13. Click on OK to return to the μ Vision main screen.


TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this on pages 7 and 12.

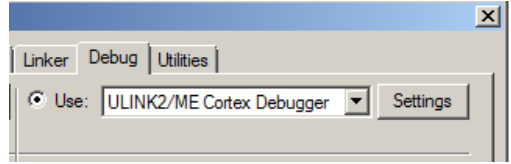


2) **Configuring a Keil ULINK2 or ULINK-ME:** Keil example programs are configured for ULINK2 by default. Serial Wire Viewer is completely supported by these two adapters. They are essentially the same devices electrically and functionally. Any reference to ULINK2 here includes the ME. The ULINK_{pro}, which is a Cortex-M ETM trace adapter, can be used like a ULINK2 or ULINK-ME. The ETM program trace frames will not display as SAM3 processors do not have ETM trace. A ULINK2 is pictured on page 1.

Assume a ULINK2 is connected to a powered up Atmel target board, μ Vision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project as described in the exercises.

Select the debug connection to the target:

1. Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select the ULINK as shown here:
2. Select Settings and the next window below opens up. This is the control panel for the ULINK 2 and ULINK-ME (they are the same).
3. In **Port:** select SWJ and SW. SWV will not work with JTAG selected.
4. 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 is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board.

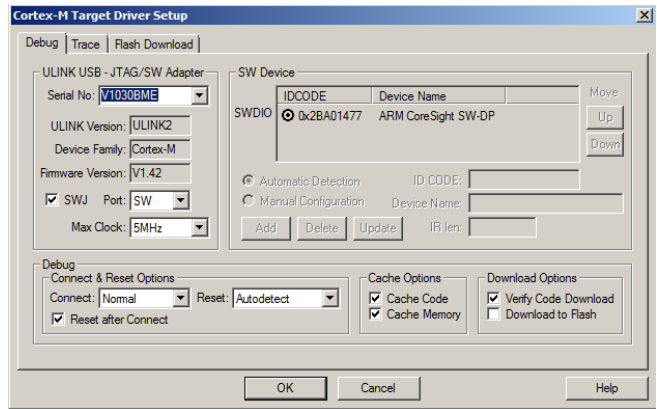


TIP: To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.

TIP: You can use JTAG if you do not intend to use SWV. SWD and JTAG operate at approximately the same speed.

Configure the Keil Flash Programmer:

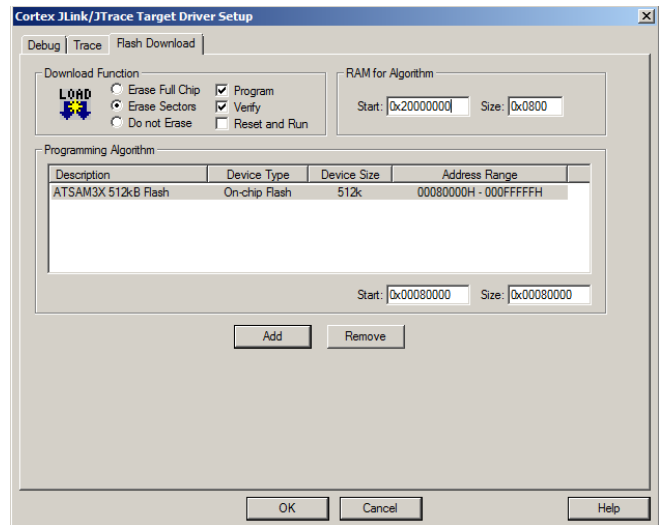
5. Click on OK once and select the Utilities tab.
6. Select the ULINK similar to Step 2 above.
7. Click Settings to select the programming algorithm.
8. Select Add and select the appropriate SAM3 Flash if necessary as shown below on the right:
9. ATSAM3X512kB Flash is for the SAM3X-EK board.
10. Click on OK once.



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

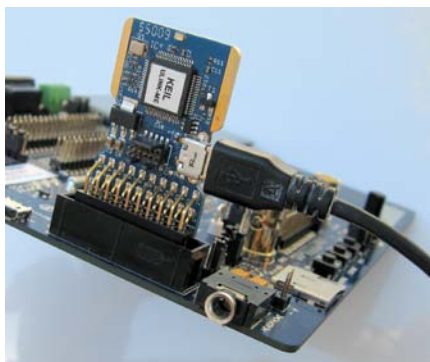
11. Click on OK to return to the μ Vision main screen.
12. You have successfully connected to the SAM3 target.

TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this on pages 7 and 13.



Keil ULINK-ME

ULINK-ME is available only as part of a board kit from Keil or another OEM.



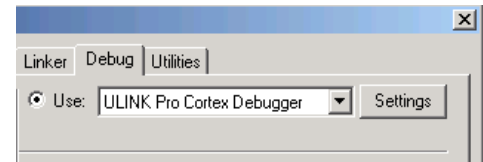
TIP: If you select ULINK or ULINK_{pro}, and have the opposite ULINK actually connected to your PC; the error message will say “No ULINK device found”. This message actually means that μ Vision found the wrong Keil adapter connected, and not that no ULINK was attached. Select the correct ULINK.

- 3) **Configuring a Keil ULINK_{pro}:** This is configured the same way as a ULINK2 except for the two selection entries. One is in the Debug tab (shown below) and the other in the Utilities tab.
1. In the Options for target in the Debug tab, select ULINK Pro Cortex Debugger as shown below.
 2. In Settings, it is configured the same as a ULINK as described on the previous page.
 3. Select the Utilities tab and select the ULINK_{pro} and select the programming algorithm as done with the ULINK2. Refer to the previous page for instructions.

TIP: If you select ULINK or ULINK_{pro}, and have the opposite ULINK actually connected; the error message will say “No ULINK device found”. This message actually means that μ Vision found the wrong Keil adapter connected.

TIP: A ULINK_{pro} will act very similar to a ULINK2. The trace window (Instruction Trace) will be quite different from the ULINK2 Trace Records.

TIP: A ULINK_{pro} must use “Serial Wire Output Manchester” as found in the Trace tab when the trace is enabled. If “Serial Wire Output UART/NRZ” mode is selected, an error will be generated.



A special adapter is provided with a ULINK_{pro} to connect to the target standard 20 pin JTAG connector. This small adapter is pictured below plugged into the SAM3X-EK JTAG connector.

TIP: A ULINK_{pro} can be used to debug an ARM7 or an ARM9 but ETM trace will not be visible. ARM7 or ARM9 processors normally do not have SWV. Contact Keil technical support for SAM9 ETM trace support information.

TIP: μ Vision windows can be floated anywhere. You can restore them by setting Window/Reset Views to default. μ Vision supports two display screens.

Keil ULINK_{pro}

An ETM trace adapter that can be used as a ULINK2. It has very fast Flash programming and an enhanced Instruction Trace window that connects the trace frames to your source code.

Instruction trace requires an ETM Trace Port (not available on current SAM3 or SAM4 devices).

ULINK_{pro} supports Serial Wire Viewer (SWV) with all Atmel Cortex-M3 and Cortex-M4 devices. ULINK_{pro} offers the best SWV support due to its use of Manchester encoding for the SWO pin. Most other adapters, including ULINK2 use UART encoding which is much slower.

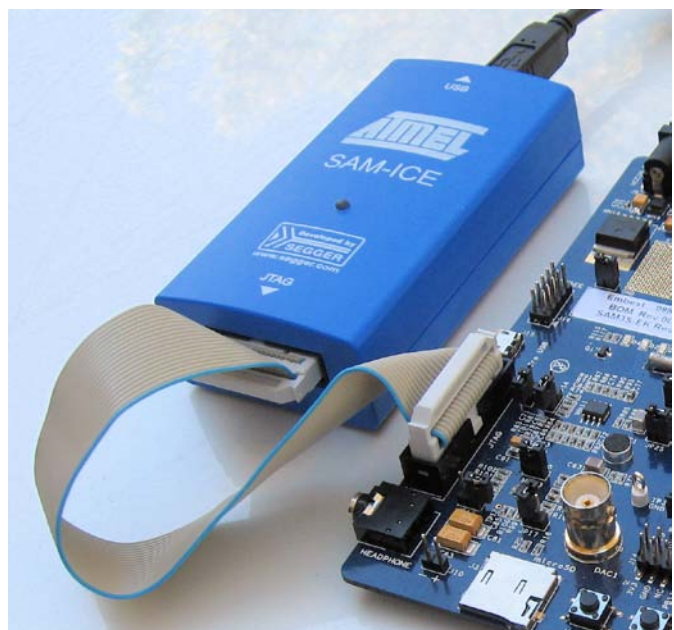


Segger SAM-ICE

Equivalent to a Segger J-Link. (black case)


Serial Wire Viewer is supported in hardware Version 6 or later.


Segger also has a new, faster J-Link Ultra. It is configured the same way in μ Vision as the J-Link is.



6) Serial Wire Viewer (SWV) Configuration:

The essential place to configure the trace is in the Trace tab as shown below. You cannot set SWV globally for μ Vision. You must configure SWV for every project and additionally for every target settings within a project you want to use SWV. This configuration information will be saved in the project. There are two ways to access this menu:

A. **In Edit mode:** Select Options for Target  or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window and then the Trace tab. Edit mode is selected by default when you start μ Vision.

B. **In Debug mode:** Select Debug/Debug Settings and then select the Trace tab. Debug mode is selected with .

1) **Core Clock:** The CPU clock speed for SWV. The CPU speed can be found in your startup code or in Abstract.txt. It is usually called SYSCLK or Main Clock. It is 84 MHz for SAM3X-EK.

2) **Trace Enable:** Enables SWV and ITM. It can only be changed in Edit mode. This does not affect the Watch and Memory window display updates.

3) **Trace Port:** This is preset for ULINK2.

4) **Timestamps:** Enables timestamps and selects the Prescaler. 1 is the default.

5) **PC Sampling:** Samples the program counter.

a. **Prescaler** 1024*16 (the default) means every 16,384th PC is displayed. The rest are not collected.

b. **Periodic:** Enables PC Sampling.

c. **On Data R/W Sample:** Displays the address of the instruction that caused a data read or write of a variable listed in the Logic Analyzer. This is not connected with PC Sampling but rather with data tracing.

6) **ITM Stimulus Ports:** Enables the thirty-two 32 bit registers used to output data in a *printf* type statement to μ Vision. Port 31 (a) is used for the Keil RTX Viewer which is a real-time kernel awareness window. Port 0 (b) is used for the Debug (*printf*) Viewer. The rest are currently unused in μ Vision.

- **Enable:** Displays a 32 bit hex number indicating which ports are enabled.

- **Privilege:** Privilege is used by an RTOS to specify which ITM ports can be used by a user program.

7) **Trace Events:** Enables various CPU counters. All except EXCTRC are 8 bit counters. Each counter is cumulative and an event is created when this counter overflows every 256 cycles. These values are displayed in the Counter window. The event created when a counter wraps around is displayed in the Instruction Trace window.

a. **CPI: Cycles per Instruction:** The cumulative number of extra cycles used by each instruction beyond the first, one including any instruction fetch stalls.

b. **Fold:** Cumulative number of folded instructions. These results from a predicted branch instruction where unused instructions are removed (flushed) from the pipeline giving a zero cycle execution time.

c. **Sleep:** Cumulative number of cycles the CPU is in sleep mode. Uses FCLK for timing.

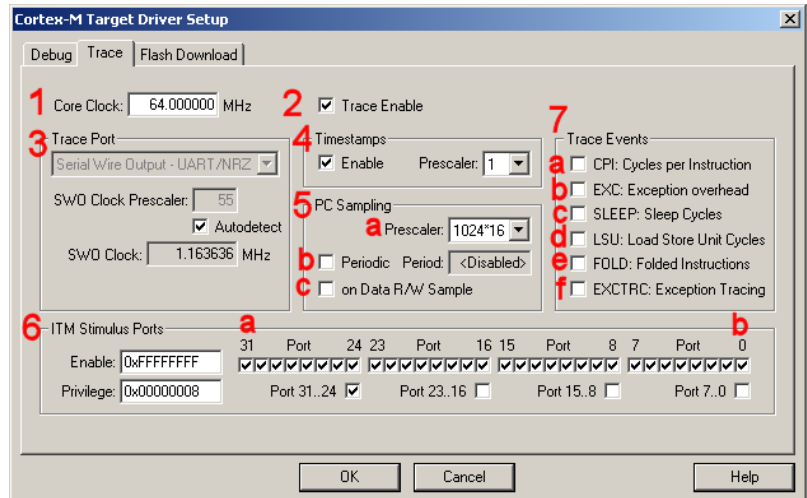
d. **EXC:** Cumulative cycles CPU spent in exception overhead not including total time spent processing the exception code. Includes stack operations and returns.

e. **LSU:** Cumulative number of cycles spent in load/store operations beyond the first cycle.

f. **EXCTRC:** Exception Trace. This is different than the other items in this section. This enables the display of exceptions in the Instruction Trace and Exception windows. It is not a counter. This is a very useful feature to display exception events and is often used in debugging.


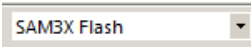





TIP: Counters will increment while single stepping. This can provide some very useful information.

TIP: If you have any lockup problems with a ULINK2 or ULINK-ME when using SWV, and these problems disappear when SWV (Trace) is not enabled, your laptop might have some USB port speed issues. Desktop computers are not affected. You can add an external USB PCMCIA card to a laptop, use a ULINK*pro*, J-Link or SAM-ICE to solve this problem.



7) Blinky example program using the Atmel SAM3X-EK and ULINK2:

Now we will connect a Keil MDK development system using real target hardware and a ULINK2 or ULINK-ME.

1. Connect the equipment as pictured on the first page.
2. Start μ Vision by clicking on its desktop icon. 
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Atmel\ATSAM3X-EK\Blinky\Blinky.uvproj.
4. Make sure "SAM3X Flash" is selected.  This is where you create and select different target configurations such as to execute a program in RAM or Flash.
5. Configure your USB-JTAG adapter at this point if you are not using a ULINK2. ULINK2 is selected by default. See pages 4 through 6. **Make sure SW is selected and not JTAG in the Port: box.** This is an important step later for Serial Wire Viewer (SWV) operation. If you do not use SWV, you can select JTAG.
6. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
7. Program the SAM3X 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 if it is chosen.
9. Click on the RUN icon.  Note: you stop the program with the STOP icon. 

The LEDs on the SAM3X-EK will now blink.

1. Rotating the potentiometer VR1 will change the speed the LEDs blink. The pot is connected to an A/D convertor.
2. The A/D value will also be displayed on the LCD as shown below: This is the local variable `ad_val` in `main()`.
3. Pressing the Left Click (BP5) and Right Click (BP4) buttons will enhance the Buttons icon on the LCD.



Now you know how to compile a program, program it into the ATSAM3X processor Flash, run it and stop it !

This program will now run stand-alone on the SAM3X-EK if you remove the debug adapter and RESET the board. Blinky is permanently programmed in the device flash memory.

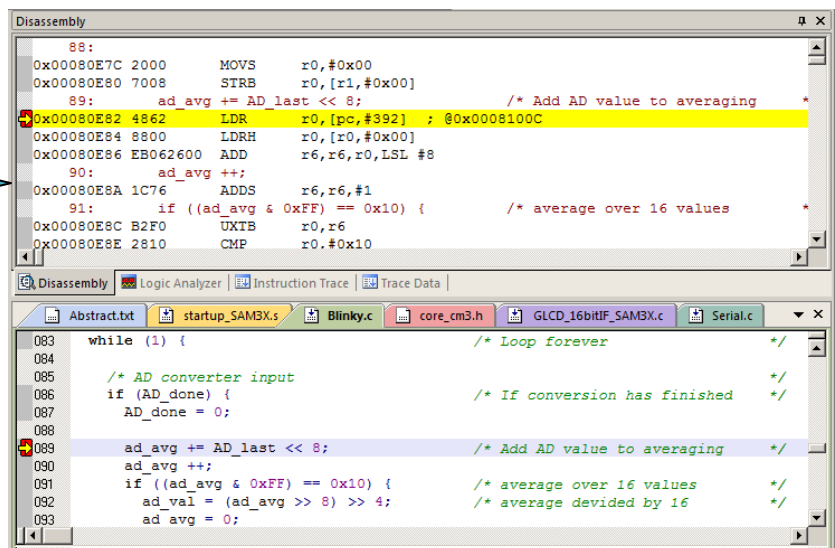


8) Hardware Breakpoints:

The SAM3X has six hardware breakpoints that can be set or unset on the fly while the program is running.

1. With Blinky running, in the Blinky.c window, click on a darker block in the left margin in either the disassembly or an appropriate source window.
2. A red circle will appear and the program will stop.
3. Note the breakpoint is displayed in both the disassembly and source windows as shown here: 
4. Every time you click on the RUN icon  the program will run until the breakpoint is again encountered.
5. Remove the breakpoint by clicking on it.

TIP: A hardware breakpoint does not execute the instruction it is set to. ARM CoreSight breakpoints are no-skid. This is a rather important feature. Earlier versions of μ Vision required a double-click to set a breakpoint.




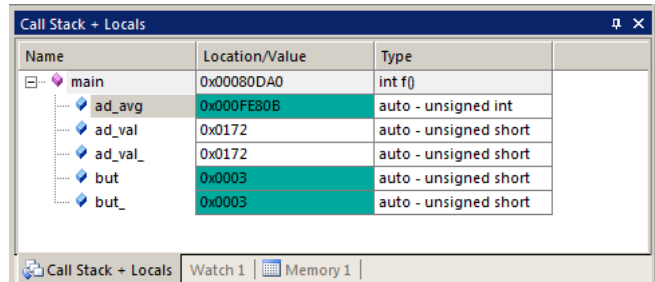
9) Call Stack + Locals Window:



Local Variables:

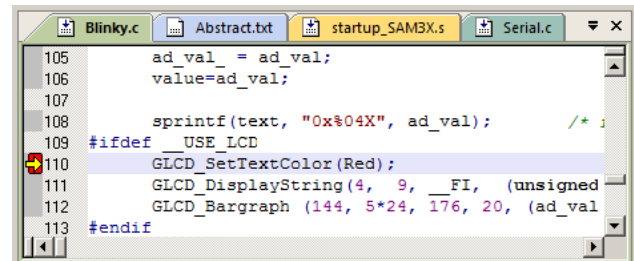
Starting with MDK 4.22, the Call Stack and Local windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function.

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

1. Run and Stop Blinky. Click on the Call Stack + Locals tab.
2. Shown is the Call Stack + Locals window.
3. The contents of the local variables are displayed as well as function names.
4. In this example, the LCD displayed 0x0172 as does the local `ad_val` as shown in the window here: 
5. Two buttons Left and Right are depressed when the program was run and variable `but` = 0x0003.
6. As you click on RUN and STOP, with the pot turned and different switches pressed these variables will update as appropriate.
7. Set a breakpoint in `Blinky.c` near line 109 (`GLCD_SetTextColor(Red)`); as shown below.



8. Click on RUN. The program will soon stop at the breakpoint. Click on the Step In icon or F11: 
9. Note the function `GLCD_SetTextColor` is displayed.
10. Click numerous times on Step In and see other functions.
11. Click on the StepOut icon  to exit all functions to return to `main()`.



12. When you ready to continue, remove the hardware breakpoint by clicking on its red circle !

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

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

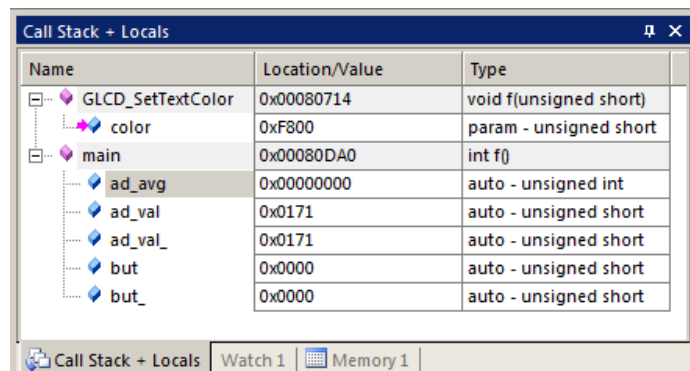
Call Stack:

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

TIP: You can modify a variable value when the program is stopped.

TIP: You can access the Hardware Breakpoint table by clicking on Debug/Breakpoints or Ctrl-B. This is also where Watchpoints (also called Access Points) are configured. You can temporarily disable entries in this table.

Selecting Debug/Kill All Breakpoints depletes Breakpoints but not Watchpoints.






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

The Watch and memory windows will display updated variable values in real-time. It does this through the ARM CoreSight debugging technology that is part of Cortex-M3 processors. It is also possible to “put” or insert values into these memory locations in real-time. It is possible to “drag and drop” variable names into windows or enter them manually.

Watch window:

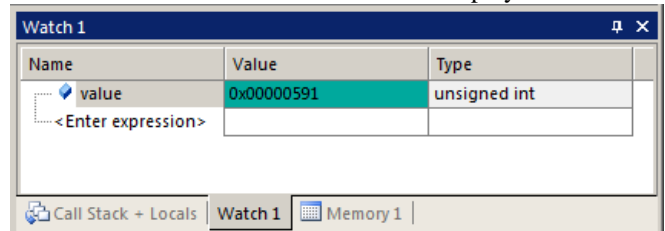
Add a global variable: Recall the Watch and Memory windows can’t see local variables unless stopped in their function.

1. Stop the processor  and exit debug mode. 
2. Declare a global variable (I called it value) near line 30 in Blinky.c like this: **unsigned int value = 0;**
3. Add the statement **value = ad_val;** as shown here near line 106
4. Click on Rebuild icon and program the Flash with the Load icon.
5. Enter Debug mode . Click on RUN if desired. You can set a Watch window while the program is running. You can do this with a Memory window too.
6. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
7. In Blinky.c, block **value**, click and hold and drag it into Watch 1. Release it and **value** will be displayed as shown here:
8. Rotate the pot and value will change in real-time.

```

103
104     if (ad_val ^ ad_val_) {
105         ad_val_ = ad_val;
106         value = ad_val;
107

```



TIP: Make sure View/Periodic Window Update is selected.

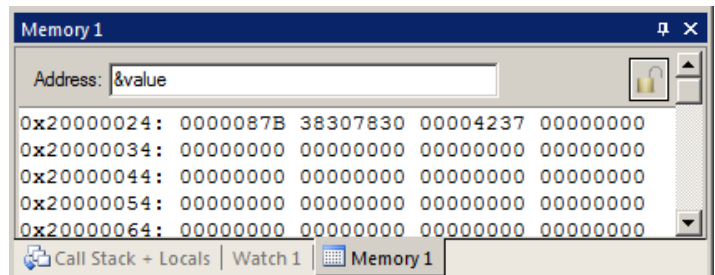
9. You can also enter a variable manually by double-clicking under Name or pressing F2 and using copy and paste or typing the variable.
10. Another way is to right click on the variable and select Add *var name* to ...

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

6. Double click on the value for **value** in the Watch window. Enter the value 0 and press Enter. 0 will be inserted into memory in real-time. You will probably not see the change as this value is constantly updated by the program.

Memory window:

1. Drag ‘n Drop **value** into the Memory 1 window or enter it manually. Select View/Memory Windows if necessary.
2. Note the value of **value** 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. The physical address is shown (0x2000_0024).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **value** is displayed as shown here:
6. Both the Watch and Memory windows are updated in real-time.
7. You can modify value in the Memory window with a right-click with the mouse cursor over the data field and select Modify Memory.



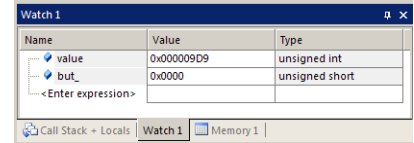
TIP: No CPU cycles are used to perform these operations. See the next page for an explanation how this works.

TIP: To view variables and where they are located use the Symbol window. Select View/Symbol Window while in Debug mode.



Serial Wire Viewer does not need to be configured in order for the Memory and Watch windows to operate as shown. This mechanism uses a different feature of CoreSight than SWV. These Read and Write accesses are handled by the Serial Wire Debug (SWD) or JTAG connection via the CoreSight Debug Access Port (DAP), which provides on-the-fly memory accesses.

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

1. Make sure Blinky.c is running. We will use the local variable `but_` (has underscore after it)
2. Locate where the local variable `but_` is declared in Blinky.c near line 59, at the start of the main function.
3. Drag and Drop `but_` into Watch 1 window. Note it says “not in scope” because μ Vision cannot access the CPU registers while running which is where value is located.
4. Stop the program and a value of 0x0 will probably appear.
5. Start the program, hold down the left or right click button and then stop the program. A value of 1, 2 or 3 will display. **Make sure you do not hit the RESET button instead !**
6. Note that sometimes the correct value does not show depending on where the program stops and when `but_` is sampled.
7. Start the program and set a breakpoint in the `SER_PutChar` function in `Serial.c` near line 104. The program will stop and <not in scope > appears. You can open `Serial.c` in the Project window or by selecting File/Open.



TIP: Remember: you can set and unset hardware breakpoints on-the-fly in the Cortex-M3 while the program is running !

8. μ Vision is unable to determine the value of `but_` when the program is running because it exists only when main is running. It disappears in functions and handlers outside of main. `But_` is also a local or automatic variable and this means it is probably stored in a CPU register which μ Vision is unable to access during run time.
9. Remove the breakpoint and make sure the program is not running . Exit Debug mode. 


How to view local variables updated in real-time:


All you need to do is to make `but_` static !





1. In the declaration for `but_` add `static` like this: you will have to separate the declaration of `but` from `but_`.

```
int main (void) {
    uint16_t but = 0;
    static uint16_t but_ = 0xFFFF;
```

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

2. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
3. To program the Flash, click on the Load icon. . A progress bar will be displayed at the bottom left.

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

4. Enter Debug mode.  You might have to re-enter `but_` in the Watch 1 window if it is no longer displayed because it isn't the same variable anymore – it is now a static variable instead of a local. Drag 'n Drop is the fastest way. With later versions of μ Vision, you do not have to reenter it because they are now fully qualified when entered.
5. Click on RUN. 
6. `but_` is now updated in real-time. Press the Left and Right click buttons separately and also together. 1, 2 and 3 will display appropriately and in real-time. This is ARM CoreSight technology working.
7. Stop the CPU and exit debug mode for the next step.  and 

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

How It Works:

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

This can be slightly intrusive in the unlikely event the CPU and μ Vision 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.

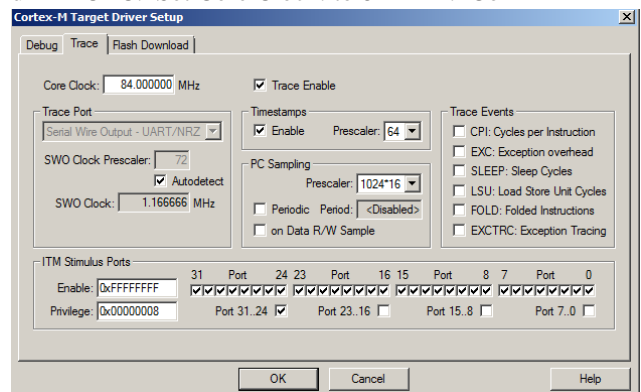
12) View Variables Graphically with the Logic Analyzer (LA):

We have seen the program Blinky display the variable `ad_val` on the LCD. We will display this in the Logic Analyzer. Recall we created the global variable value that tracks `ad_val`. We will plot value since `ad_val` is a local.

1. Stop the processor and exit Debug mode.

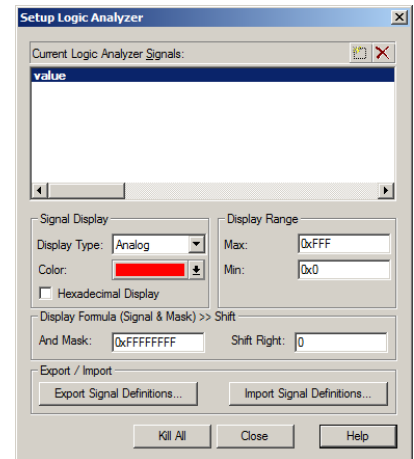
Configure Serial Wire Viewer (SWV): The LA uses SWV to collect the data:

2. Select Options for Target or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window. Confirm SWJ and SW is selected (J-Link has no SWJ). SW selection is mandatory for SWV. Select the Trace tab.
3. In the Trace tab, select Trace Enable. Unselect Periodic and EXTCRC. Set Core Clock: to 84 MHz. Confirm everything else is set as in this window:
4. Click OK twice to exit.
5. Enter debug mode.



Configure Logic Analyzer:

1. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.
2. Click on the Blinky.c tab. Block `value`, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
3. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release.
4. Click on the Select box and the LA Setup window appears:
5. With `value` selected, set Display Range Max: to `0xFFFF` as shown here:
6. Click on Close. Click on Run.

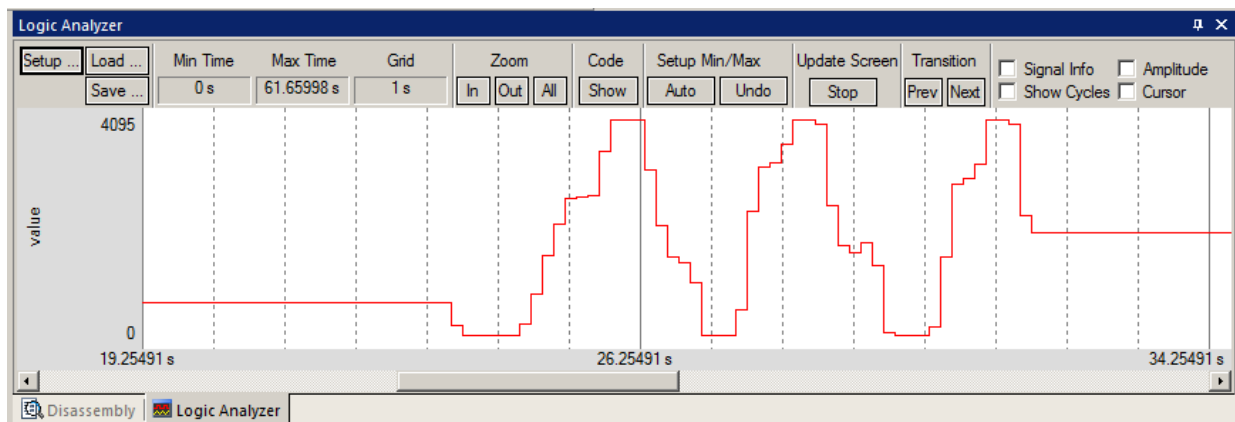


Run Program: Note: The LA can be configured while the program is running.

- 1) Click on Run. Click on Zoom Out until Grid is about 1 second.
- 2) Rotate the pot and see the window below.

TIP: You can show up to 4 variables in the Logic Analyzer. These variables must be global, static or raw addresses such as `*((unsigned long *)0x20000000)`.


- 3) Recall we made `but_` a static variable. Enter this into the LA and set the Display Range Max: to `0x4`. Press the buttons and see the voltages.
- 4) Select Signal Info, Show Cycles, Amplitude and Cursor to see effects. Stop the CPU and stay in debug mode for the next exercise.

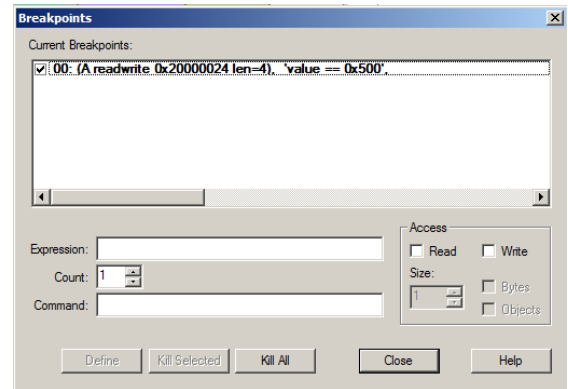


13) Watchpoints: Conditional Breakpoints

SAM3 and SAM4 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. The SAM3X also has four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same comparators as Watchpoints in its operations. This means in μ Vision you must have two variables free in the Logic Analyzer to use Watchpoints. Watchpoints are also referred to as Access Breaks.

1. Use the same Blinky configuration as the previous page. Stop the program if necessary. Stay in debug mode.
2. We will use the global variable `value` you created in `Blinky.c` to explore Watchpoints.
3. The Trace does not need to be configured for Watchpoints. However, we will use it in this exercise.
4. The variable `value` should be in the to the Logic Analyzer from the last exercise on the previous page.
5. Select Debug in the main μ Vision window and select Breakpoints or press Ctrl-B.
6. In the Expression box enter: "`value == 0x500`" without the quotes. Select both the Read and Write Access.

7. Click on Define and it will be accepted as shown here: 
8. Click on Close.
9. Enter the variable `value` to the Watch 1 window by dragging and dropping it if it is not already there.



10. Open Debug/Debug Settings and select the trace tab. Check "on Data R/W sample", uncheck EXTRC and ITM 31 and 0.
11. Click on OK twice. Open the Trace Records window.
12. Click on RUN. Rotate the pot so 0x500 is displayed on the LCD. You might have to adjust the pot very carefully.
13. You will see `value` change in the Logic Analyzer as well as in the Watch window.

14. When `value` equals 0x500, the Watchpoint will stop the program. The LCD might not display exactly 0x500 due to sample timing.
15. Note the data writes in the Trace Records window shown below. 0x500 is in the last Data column. Plus the address the data written to and the PC of the write instruction. This is with a ULINK2 or ULINK-ME.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000024H	000004EFH	00080EC4H	X	7095573065	84.47110792
Data Write			20000024H	00000514H	00080EC4H	X	7122334602	84.78969764
Data Write			20000024H	00000520H	00080EC4H	X	7135774667	84.94969842
Data Write			20000024H	00000575H	00080EC4H	X	7149214604	85.10969767
Data Write			20000024H	00000524H	00080EC4H	X	7162654605	85.26969768
Data Write			20000024H	00000545H	00080EC4H	X	7176094670	85.42969845
Data Write			20000024H	000005C7H	00080EC4H	X	7189534671	85.58969846
Data Write			20000024H	00000500H	00080EC4H	X	7202974672	85.74969848

16. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.

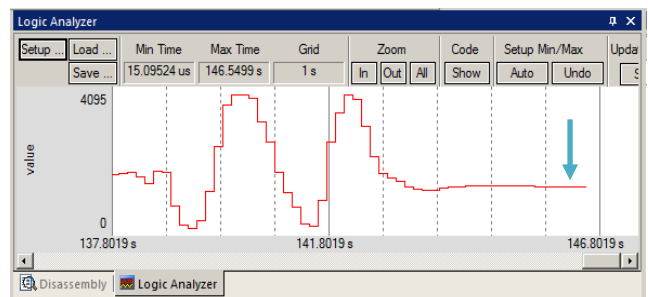
17. To repeat this exercise, click on RUN and rotate the pot again.
18. When finished, stop the program, click on Debug and select Breakpoints (or Ctrl-B) and Kill the Watchpoint.
19. Leave Debug mode.
20. Note the J-Link or SAM-ICE will not display the writes in the Trace Records window. Watchpoints do work.

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

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

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






TIP: Raw addresses can also be entered into the Logic Analyzer. An example is: `*((unsigned long *)0x20000000)`
Shown here is the Logic Analyzer window displaying the variable `value` trigger point of 0x500.



The cyan arrows point to when the Watchpoint trigger `value` equals 0x500.

14) 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 Keil MDK including source. It can have up to 255 tasks and no royalty payments are required. This example explores the RTX RTOS project. MDK will work with any RTOS. An RTOS is just a set of C functions that gets compiled with your project.

1. With μ Vision in Edit mode (not in debug mode): Select Project/Open Project.
2. Open the file C:\Keil\ARM\Boards\Atmel\SAM3X-EK\RTX_Blinky\Blinky.uvproj.
3. If you are not using a ULINK2 or ULINK-ME, you now have to configure μ Vision for the adapter you are using. You only have to do this once – it will be saved in the project file. You can also make a new target configuration.
4. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
7. The LCD display will indicate the four waveforms of a stepper motor driver changing. Click on STOP .



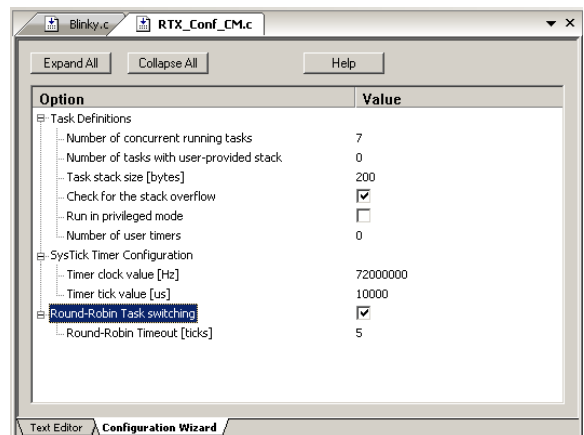
The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open if needed.
2. Click on the Configuration Wizard tab 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. Changing an attribute in one tab changes it in the other automatically. You should save a modified window.
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 μ Vision System Viewer windows are created in a similar fashion. Select View/System Viewer.

```

081 #ifndef OS_TICK
082 #define OS_TICK      10000
083 #endif
084
085 // </h>
086 // <e>Round-Robin Task switching
087 // =====
088 // <i> Enable Round-Robin Task switching
089 #ifndef OS_ROBIN
090 #define OS_ROBIN      1
091 #endif
092
093 // <o>Round-Robin Timeout [ticks] <1-1
094 // <i> Define how long a task will exe
095 // <i> Default: 5
096 #ifndef OS_ROBINTOUT
097 #define OS_ROBINTOUT  5
098 #endif
  
```

Text Editor: Source Code



Configuration Wizard

15) RTX Kernel Awareness using RTX Viewer

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

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. Note these values are updated in real-time using the same technology as used in the Watch and Memory windows.
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.

RTX Viewer: Configuring Serial Wire Viewer (SWV):

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

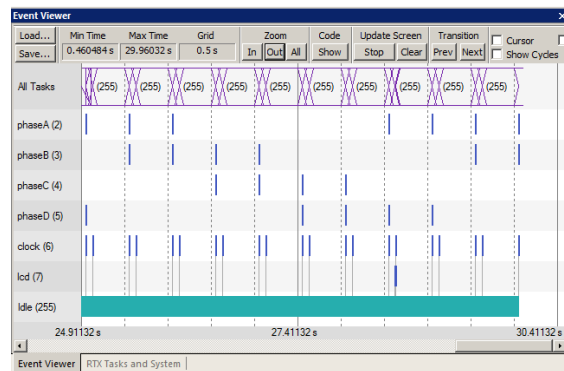
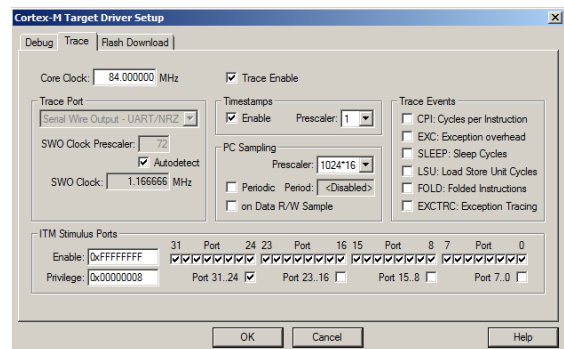
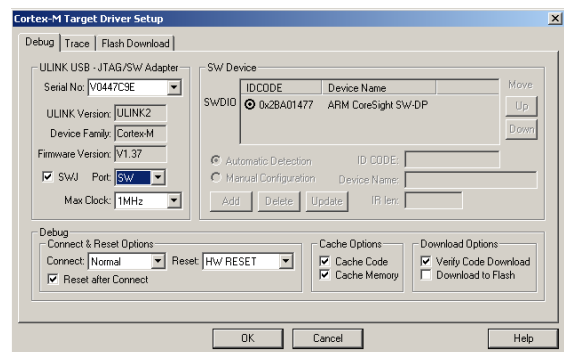
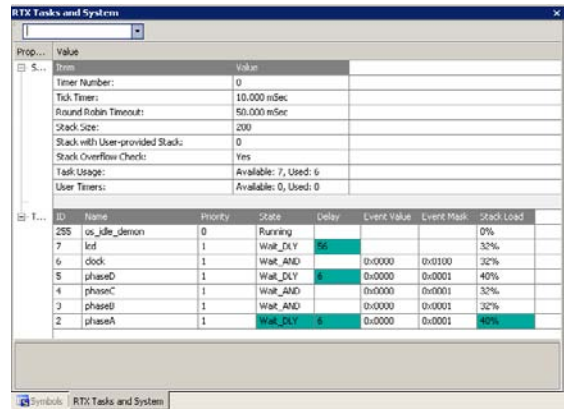
1. Stop the CPU and exit debug mode.
2. Click on the Options icon next to the target box.
3. Select the Debug tab and then click the Settings box next to ULINK2/ME Cortex Debugger dialog.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 84 MHz and select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown here:
8. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer.
9. Click on OK twice to return to μ Vision.
The Serial Wire Viewer is now configured in μ Vision.
10. Enter Debug mode and click on RUN to start the program.
11. Select "RTX Tasks and System" tab: the display is updated.
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 1 second by clicking on the ALL, In and Out icons.
14. Close these windows when you are done for the next exercise.

TIP: View/Periodic Window Update must be selected !

TIP: To find the Core frequency, open the file System_SAM3S.s and install the global variable SystemFrequency in the Watch window.

Cortex-M3 Alert: The ATSAM3 will update all RTX information in real-time on a target board due to its Serial Wire Viewer and read/write capabilities as already described.

You will not have to stop the program to view this data. No CPU cycles are used. Your program runs at full speed. No instrumentation code needs to be inserted into your source. The Event Viewer uses the ITM Stimulus Ports which is slightly intrusive. You will find this feature very useful. You can use a ULINK2, ULINK-ME, ULINKpro, J-Link or SAM-ICE for RTX Kernel Awareness windows.








16) 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 ATSAM3. 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. The SWV must be configured.
2. Add 4 global variables `unsigned int phasea` through `unsigned int phased` to Blinky.c as shown here:
3. Add 2 lines to each of the four tasks Task1 through Task4 in Blinky.c as shown below: `phasea=1;` and `phasea=0;`; the first two lines are shown added at lines 084 and 087 (just after LED_On and LED_Off function calls). For each of the four tasks, add the corresponding variable assignment statements `phasea`, `phaseb`, `phasesc` and `phased`.
4. We do this because in this simple program there are not enough suitable global variables to connect to the Logic Analyzer.

```
030 #define LED_D      3
031 #define LED_CLK    7
032
033 unsigned int phasea = 0;
034 unsigned int phaseb = 0;
035 unsigned int phasesc = 0;
036 unsigned int phased = 0;
037
```

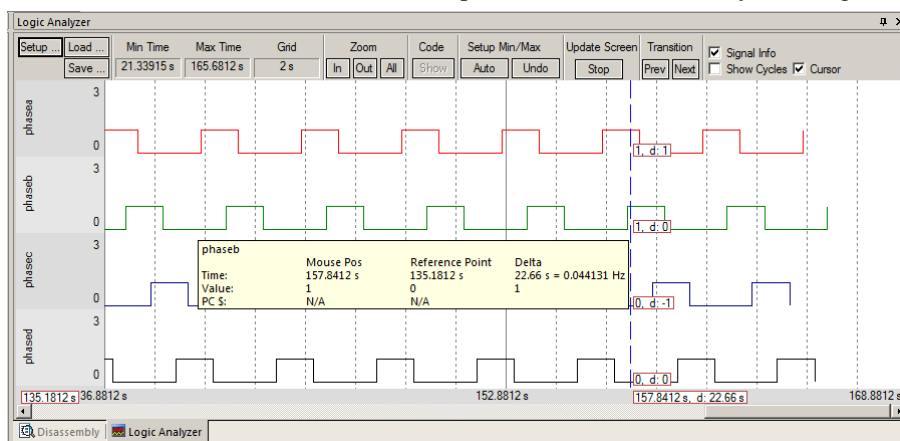
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

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. .

```
077 /*-----Task 1 'phaseA': Phase A out
078 *
079 *-----
080 _task void phaseA (void) {
081     for (;;) {
082         os_evt_wait_and (0x0001, 0xffff);
083         LED_on (LED_A);
084         phasea = 1;
085         signal_func (t_phaseB);
086         LED_off(LED_A);
087         phasea = 0;
088     }
089 }
```

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.
11. Repeat for `phaseb`, `phasesc` and `phased`. 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 four 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 1 or 2 seconds. Move the scrolling bar to the far right if needed.
15. You will see the following waveforms appear. Click on Stop in the Update Screen box. Select the Show Info and Cursor boxes. Click to mark a place See 152 s below. Place the cursor on one of the waveforms to obtain timing and other information as shown in the inserted box labeled `phaseb` as shown below by hovering over a location:



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

TIP: Raw addresses can also be entered into the Logic Analyzer. An example is: `*((unsigned long *)0x20000000)`

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

1) Data Reads and Writes: (Note: Data Writes but not Reads are enabled in the current version of μ Vision).


Currently, Summer 2012, the display of data read, write and ITM trace frames are not implemented in the J-Link or SAM-ICE.

You have already configured Serial Wire Viewer (SWV) on page 13 under **RTX Viewer: Configuring the Serial Wire Viewer:**

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

Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. Your program runs at full speed and needs no code stubs or instrumentation software added to your source code. Screens are shown using a ULINK2.

Use RTX_Blinky from the last exercise. Enter Debug mode and run the program if not already running.

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

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

3. 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 Stimulus Port 31. **TIP:** Port 0 is used for Debug `printf` Viewer.

4. Unselect EXCTRC and Periodic.

5. Select On Data R/W Sample.

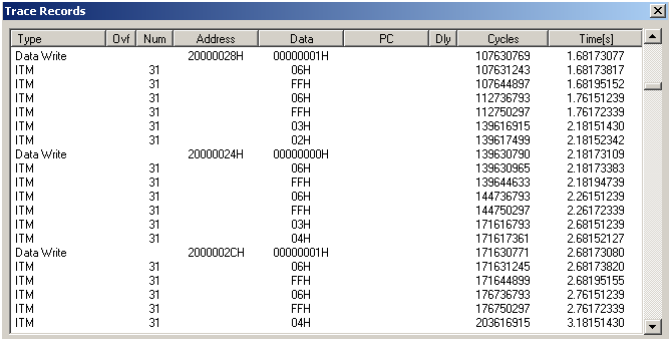
6. Click on OK twice to return.

7. Click on the RUN icon.

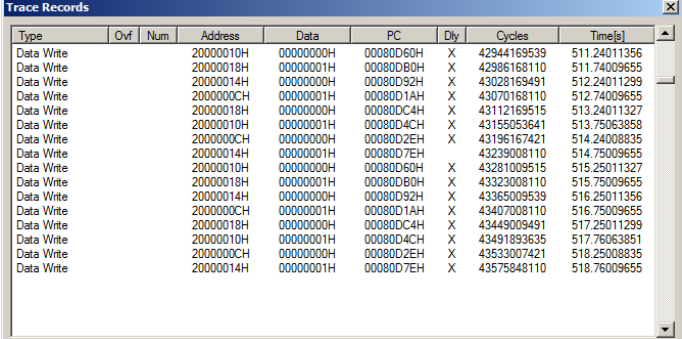
8. Double-click anywhere in the Trace records window to clear it.

9. Only Data Writes will appear now.

TIP: You could have right clicked on the Trace Records window to filter the ITM frames out. Unselecting a feature is better as it reduces SWO pin traffic and therefore trace overflows.




Type	Ovrfl	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write		31	20000028H	0000001H			107630769	1.68173077
ITM		31		08H			107631243	1.68173617
ITM		31		FFH			107844897	1.68195152
ITM		31		06H			112736793	1.76151239
ITM		31		FFH			112750297	1.76172339
ITM		31		03H			139616915	2.18151430
ITM		31		02H			139617499	2.18152342
Data Write		31	20000024H	00000000H			139630790	2.18173109
ITM		31		06H			139630965	2.18173383
ITM		31		FFH			139844633	2.18194739
ITM		31		06H			144736793	2.26151239
ITM		31		FFH			144750297	2.26172339
ITM		31		03H			171616793	2.68151239
ITM		31		04H			171617361	2.68152127
Data Write		31	2000002CH	00000001H			171630771	2.68173080
ITM		31		06H			171631245	2.68173820
ITM		31		FFH			171644899	2.68195155
ITM		31		06H			176736793	2.76151239
ITM		31		FFH			176750297	2.76172339
ITM		31		04H			203616915	3.18151430



Type	Ovrfl	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000010H	0000000H	00080D60H	X	42944169539	511.24011356
Data Write			20000018H	00000001H	00080D60H	X	42985168110	511.74009655
Data Write			20000014H	00000000H	00080D92H	X	43028169491	512.24011299
Data Write			2000000CH	00000001H	00080D1AH	X	43070168110	512.74009655
Data Write			20000018H	00000000H	00080DC4H	X	43112169515	513.24011327
Data Write			20000010H	00000001H	00080D4CH	X	43155053641	513.75068858
Data Write			2000000CH	00000000H	00080D2EH	X	43196167421	514.24008835
Data Write			20000014H	00000001H	00080D7EH	X	43239008110	514.75009655
Data Write			20000010H	00000000H	00080D60H	X	43281009515	515.25011327
Data Write			20000018H	00000001H	00080DB0H	X	43323008110	515.75009655
Data Write			20000014H	00000000H	00080D92H	X	43365009539	516.25011356
Data Write			2000000CH	00000001H	00080D1AH	X	43407008110	516.75009655
Data Write			20000018H	00000000H	00080DC4H	X	43449009491	517.25011299
Data Write			20000010H	00000001H	00080D4CH	X	43491893635	517.76063851
Data Write			2000000CH	00000000H	00080D2EH	X	43533007421	518.25008835
Data Write			20000014H	00000001H	00080D7EH	X	43575848110	518.76009655

What is happening here ?

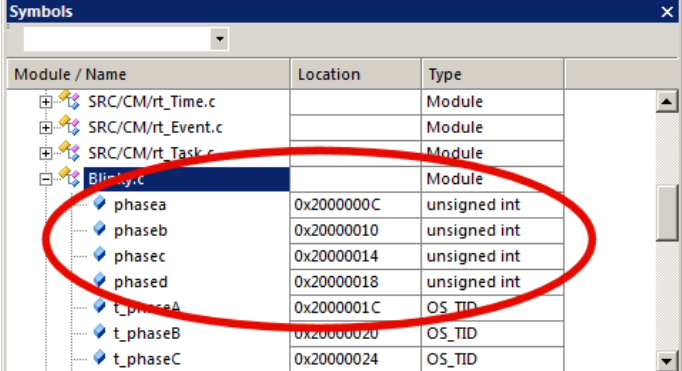
- When variables are entered in the Logic Analyzer (remember phasea through phased ?), the reads and/or writes will appear in Trace Records.
- The Address column shows where the four variables are located. 
- The Data column displays the data values written to phasea through phased.
- PC is the address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

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

TIP: If you select View/Symbol Window you can see where the addresses of the variables are. Yours might be different.

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

TIP: ULINK pro , J-Link and SAM-ICE adapters display the trace frames in a slightly different style trace window.

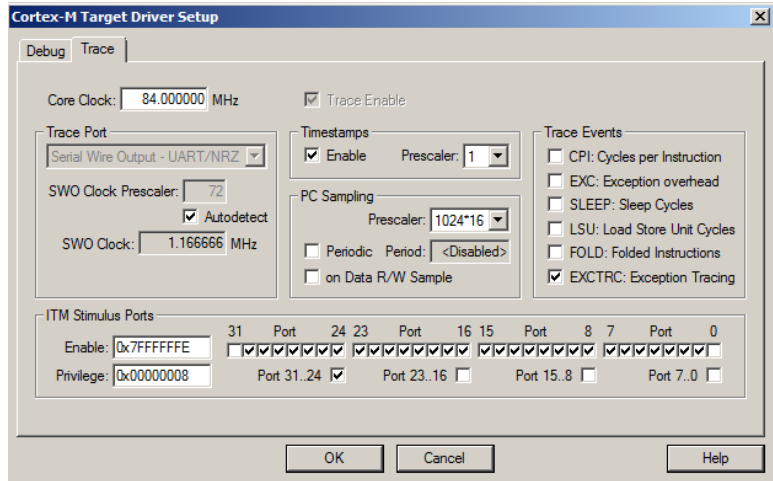


Module / Name	Location	Type
SRC/CM/rt_Time.c		Module
SRC/CM/rt_Event.c		Module
SRC/CM/rt_Task.c		Module
Blinky.c		Module
phasea	0x2000000C	unsigned int
phaseb	0x20000010	unsigned int
phasec	0x20000014	unsigned int
phased	0x20000018	unsigned int
t_phaseA	0x2000001C	OS_TID
t_phaseB	0x20000020	OS_TID
t_phaseC	0x20000024	OS_TID

2) Exceptions and Interrupts:

The ATSAM3 family using the Cortex-M3 processor has many interrupts and it can be difficult to determine when they are being activated and how often. Serial Wire Viewer (SWV) on the ATSAM3 family makes this task easy.

1. Use the RTX_Blinky example program. Be in Debug mode. 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.



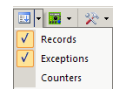
What Is Happening ?

1. You can see two exceptions happening.
 - **Entry:** when the exception enters.
 - **Exit:** When it exits or returns.
 - **Return:** When all the exceptions have returned to the main program. This is useful to detect tail-chaining.
2. Num 11 is SVCcall 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. Always enable the fewest SWO features to only those you really need.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					46985391834	559.34990279
Exception Exit		15					46985392117	559.34990615
Exception Return		0			X		46985394720	559.34993714
Exception Entry		15					46986231834	559.35990279
Exception Exit		15					46986232520	559.35991095
Exception Return		0			X		46986236832	559.35996229
Exception Return	X	0			X		46986236832	559.35996229
Exception Entry		11					46986248092	559.36009633
Exception Exit		11					46986248226	559.36009793
Data Write			20000010H	00000000H		X	46986254112	559.36016800
Exception Return	X	0			X		46986254112	559.36016800
Exception Entry		11					46986263533	559.36028015
Exception Exit		11					46986263667	559.36028175
Exception Return	X	0			X		46986267792	559.36033086
Exception Entry		15					46987071834	559.36990279
Exception Exit		15					46987072122	559.36990621
Exception Return		0			X		46987074696	559.36993686
Exception Entry		15					46987911834	559.37990279
Exception Exit		15					46987912117	559.37990615
Exception Return		0			X		46987914720	559.37993714

TIP: The SWO pin is one pin on the Cortex-M3 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate. μ Vision easily recovers gracefully from these overflows. Overflows are shown when they happen.

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



Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCcall	311	2.344 ms	1.595 us	57.238 us	181.905 us	430.365 ms	518.84008663	559.86028219
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	4111	18.116 ms	3.369 us	8.214 us	9.992 ms	9.997 ms	518.79990279	559.89990279
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						

TIP: 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.

3) PC Samples:

Serial Wire Viewer can display a sampling of the program counter.

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.
5. Click on RUN and this window opens:
6. Most of the PC Samples in the example shown are 0x0040_05E2 which is a branch to itself in a loop forever routine.

Note: the exact address you get depends on the source code and compiler settings.

7. Stop the program and the Disassembly window will show this Branch as shown below:

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					004005E2H		86035838109	1344.30997045
PC Sample					004005E2H		86035854493	1344.31022645
PC Sample					004005E2H		86035870877	1344.31048245
PC Sample					004005E2H		86035887261	1344.31073845
PC Sample					004005E2H		86035903645	1344.31099445
PC Sample					004005E2H		86035920029	1344.31125045
PC Sample					004005E2H		86035936413	1344.31150645
PC Sample					004005E2H		86035952797	1344.31176245
PC Sample					004005E2H		86035969181	1344.31201845
PC Sample					004005E2H		86035985565	1344.31227445
PC Sample					004005E2H		86036001949	1344.31253045
PC Sample					004005E2H		86036018333	1344.31278645
PC Sample					004005E2H		86036034717	1344.31304245
PC Sample					004005E2H		86036051101	1344.31329845
PC Sample					004005E2H		86036067485	1344.31355445
PC Sample					004005E2H		86036083869	1344.31381045
PC Sample					004005E2H		86036100253	1344.31406645
PC Sample					004005E2H		86036116637	1344.31432245
PC Sample					004005E2H		86036133021	1344.31457845
PC Sample					004005E2H		86036149405	1344.31483445

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. Set a breakpoint in one of the tasks.

10. Run the program and when the breakpoint is hit, you might see another address at the bottom of the Trace Records window. See the screen below:

```

154: /* This function is called when the user timer has expired. Parameter */
155: /* 'info' holds the value, defined when the timer was created. */
156:
157: /* HERE: include optional user code to be executed on timeout. */
0x004005E0 BFD0 NOP
->0x004005E2 E7FE B 0x004005E2
158: }
159:

```

11. 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.

12. Remove the breakpoint for the next step.

The screenshot shows the IDE interface with the Disassembly window and Trace Records window open. The Disassembly window shows assembly code for a task, with the instruction `0x004007C6 6008 STR r0, [r1, #0x00]` highlighted in yellow. The Trace Records window shows a list of PC samples, with the last entry at `0040072H`.

18) Segger J-Link and SAM-ICE Trace Windows: *for reference*

µVision provides three basic trace windows with a ULINK2: Trace Records, Exception Trace and Event Counters.

A Segger J-Link or SAM-ICE provides a slightly different trace window. The main difference is the Trace Records window. It is called Instruction Trace on SAM-ICE or J-Link. Exception Trace and Event Counters are the same as the ULINK2. J-Link Ultra provides faster operation and is compatible with µVision. A J-Trace can be used but was not tested at this time.

The program must be halted in order to update information in the Instruction Trace window.

Instruction Trace: Currently displays PC Samples and Exceptions. No data read or write frames are provided.

Displayed here are PC Samples. The columns contain the address, the assembly instruction opcode, the disassembled instruction and any source code relevant.

Note the source code is displayed. If you double click on a frame line, you will be taken to that line in the source and /or disassembly windows.

Exception frames are part of this window but you need to scroll to them or select the filtering box.

Nr.	Ovf	Address	Opcode	INT	Instruction	Source
237		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
238		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
239		0x00080792	D3FB		BCC 0x0008078C	129: LCD_DAT16 = dat;
240		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
241		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
242		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
243		0x00080790	4290		CMP r0,r2	129: LCD_DAT16 = dat;
244		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
245		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
246		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
247		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
248		0x0008078C	804C		STRH r4,[r1,#0x02]	129: LCD_DAT16 = dat;
249		0x000817EC	F7FFBDA		BL.W rt_dispatch(0x000...	129: LCD_DAT16 = dat;

Interleaved with the PC Samples will be the exception frames. You can filter these out as shown here:

Two exceptions are listed here: SysTick and SVC call.

Note the exception entry and exit points are shown. “Entry” can mean either a entry or exit point. Return 00 ISR Return is the return from all exceptions. The absence of this line can mean tail-chaining is happening as the processor goes from one exception directly to the next without popping and pushing everything on the stack. This saves many CPU cycles and is automatically done in Cortex-M series processors.

Nr.	Ovf	Address	Opcode	INT	Instruction	Source
3696			Entry	15	SysTick_Handler	
3697			Return	00	ISR Return	
3698			Return	00	ISR Return	
3699	X		Entry	11	SVC_Handler	
3700	X		Return	00	ISR Return	
3702			Entry	11	SVC_Handler	
3703			Return	00	ISR Return	
3704			Return	00	ISR Return	
3705	X		Return	00	ISR Return	
3707			Entry	11	SVC_Handler	
3708			Return	00	ISR Return	
3709	X		Return	00	ISR Return	
3711			Return	00	ISR Return	

The “x” in the Ovf (overflow) column indicates an overload problem. Some frames may be missing or distorted. µVision is able to recover painlessly from such overloads and indicates that such an event took place. Reduce traffic as much as possible.

The PC Samples displayed here are all the opcode E7FE which is a Branch to itself. This is the RTX idle daemon. Most of the processor time is spent executing this branch instruction.

Remember, PC Samples does not capture every instruction: only some of them.

TIP: These windows will be improved in subsequent releases of MDK.

Nr.	Ovf	Address	Opcode	INT	Instruction	Source
539		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
540		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
541		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
542		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
543		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
544		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
545		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
546		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
547		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
548		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
549		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
550		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
551		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {
552		0x00080400	E7FE		B os_idle_demon(0x00080400)	145: for(;;) {

19) Keil ULINK_{pro} Trace Windows: *for reference*

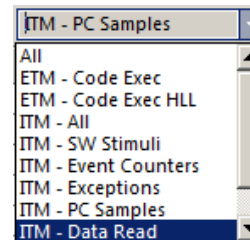
ULINK_{pro} provides the most advanced trace windows. New features are being added as development continues.

Trace Data window: This new window displays both ETM trace frames (if selected and the processor is so equipped) and Serial Wire Viewer (SWV) data frames. Note all SWV frames are called ITM (Instrumentation Trace Macrocell).

Frames are collected by ULINK_{pro} either by selecting them in the Trace Configuration window as shown on page 7 or in the case of data reads and/or writes, by adding a variable name or raw memory address to the Logic Analyzer.

Frames are displayed in the Trace Data window by selecting various filters in the Display: drop down menu shown here:

Frames are also known as Records in Keil documentation. They are the same.



Example Trace Data windows:

1) Exceptions and Data Writes: Shown below results from selecting Exceptions and the four variables phasea through phased in the RTX_Blinky example. ITM 0 and 31 are disabled. Note the “ALL” filter is selected.

Frames where a “Delay” has occurred are listed in RED with a “D” in the Time column. An “O” indicates an overflow condition. In both cases, you should be careful with the timestamps and frames: some frames could be missing. Reducing the features selected will help reduce overflows.

In this window you can see:

1. SVCalls by the RTOS. These events will also be displayed in the Exception Trace window.

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.


Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
D - 0.000 175 488 s		Exception Return		
0.000 000 000 s		Exception Entry - SVCall		
+ 0.000 001 595 s		Exception Exit - SVCall		
+ 0.000 001 702 s		Exception Return		
D + 0.000 004 048 s	W: 0x2000000C	0x00000001	X: 0x00080D1A	
D + 0.000 004 048 s		Exception Entry - SVCall		
+ 0.000 005 393 s		Exception Exit - SVCall		
D + 0.000 006 619 s		Exception Return		


2. The cyan row is where I set the timestamp to 0. You can see the Entry to Exit of the SVCall took 1.595 μsec and 1.702 μsec from Entry to Return. To set the time to zero, place the cursor on a line and right click. Select Set Time Reference.
3. At Time 4.048 μSec, is a Write to address 0x2000 0000C of data 0x1 by the instruction located at 0x0008 0D1A. 0x2000 0000C contains the data for the global variable phasea which you created in the RTX exercise.
4. The “D” shows a delay issue with the timestamp. There are no overflow frames displayed here (would be a “O”).

2) Data Writes: Data Read/Write is selected in the Display: filter box.

5. Shown below are only the data writes. Everything else has been filtered out by Display: box.
6. They show the data writes to phasea through phased with the data value, the address written to and the address of the instruction that caused this write.
7. Note one frame has had its timestamp set to zero. You can quickly determine a data write occurs with 0.5 seconds spacing.

If you double-click on a frame, the write instruction will be highlighted in the source and/or assembly windows.

Find: Click on the Find icon  to open a selection window to search for particular frames. You can select which type of frames to search in by selecting the In drop down menu.

Save: Click on the Save icon  to save the trace frames in a CSV format.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
D - 2.000 000 000 s	W: 0x20000010	0x00000000	X: 0x00080D60	
D - 1.500 001 774 s	W: 0x20000018	0x00000001	X: 0x00080D80	
D - 0.999 999 964 s	W: 0x20000014	0x00000000	X: 0x00080D92	
D - 0.500 001 643 s	W: 0x2000000C	0x00000001	X: 0x00080D1A	
D 0.000 000 000 s	W: 0x20000018	0x00000000	X: 0x00080DC4	
D + 0.499 998 226 s	W: 0x20000010	0x00000001	X: 0x00080D4C	
D + 0.999 999 988 s	W: 0x2000000C	0x00000000	X: 0x00080D2E	
D + 1.499 998 226 s	W: 0x20000014	0x00000001	X: 0x00080D7E	

3) ITM: SW Stimuli is selected in the Display: filter box.

Shown are the data writes out ITM port 31.

The Port: 31, data and timestamp are shown:

Port 0 is for the Debug (printf) Viewer which is also known as ITM. Port 31 is to send RTX data to the RTX Event Viewer. The other 30 ports are not currently used by μ Vision.

There is an ITM printf exercise on the next page. You will be able to see Port 0 frames in this window or the Trace Records window if you are using a ULINK2.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
-0.008 177 857 s	P: 31	0x0105		
-0.008 169 595 s	P: 31	0x0080DC9		
-0.008 169 429 s	P: 31	0x0106		
-0.008 160 798 s	P: 31	0x0080DEF		
-0.008 160 631 s	P: 31	0x0107		
D -0.008 153 012 s	P: 31	0x0080E93		
D -0.008 153 012 s	P: 31	0x0001		
D -0.008 152 250 s	P: 31	0x02		
-0.007 965 881 s	P: 31	0x03		
-0.007 962 321 s	P: 31	0x04		

4) PC Samples:

PC Samples are selected in the Trace Configuration window and displayed in this window:

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
-0.000 390 095 s	X: 0x000807E0	BL.W wr_reg (0x00080B46)		GLCD_SetWindow
-0.000 195 048 s	X: 0x00080B44	BX lr		wr_dat_only
0.000 000 000 s	X: 0x0008096E	ADDS r4,r4,#1		GLCD_DrawChar
+0.000 195 048 s	X: 0x0008095C	LSR r1,r8,r4		GLCD_DrawChar
+0.000 390 095 s	X: 0x0008096E	ADDS r4,r4,#1		GLCD_DrawChar
+0.000 585 143 s	X: 0x00080972	BCC 0x0008095C		GLCD_DrawChar
+0.000 780 190 s	X: 0x0008095C	LSR r1,r8,r4		GLCD_DrawChar
+0.000 975 238 s	X: 0x00080964	LDR r2,[pc,#500] ; @0x00080B5C		GLCD_DrawChar
+0.001 170 286 s	X: 0x0008096A	BL.W wr_dat_only (0x00080B3E)		GLCD_DrawChar
+0.001 365 333 s	X: 0x00080B48	MOV r5,r0		wr_reg
+0.001 560 381 s	X: 0x00080970	CMP r4,r6		GLCD_DrawChar
+0.001 755 429 s	X: 0x00080972	BCC 0x0008095C		GLCD_DrawChar
+0.001 784 976 s		Exception Entry - SysTick		
+0.001 788 405 s		Exception Exit - SysTick		

Each timestamped frame contains the address the instruction is located, its disassembled mnemonic and the source function it is located in. If you double-click on a frame, the write instruction will be highlighted in the source and/or assembly windows.

Note a SysTick timer event is displayed at the bottom of the window.

TIP: Filtering out trace frames with the Display: box does not reduce the load on the SWO pin. This is post filtering. To lower the load, deselect any features you do not need in the Trace Configuration window.

4) Counters:

Counters are described under **6) Serial Wire Viewer (SWV) Configuration with ULINK2 or ULINK-ME on page 7:**

Most counters are 8 bit registers and when they roll over, an event is displayed in the Trace Data window as shown here:

A timestamp can be set to zero to measure the timing of the events.

These events are also displayed in the Counters window where the number of rollovers is displayed in real time. Each counter can be reset to zero.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
0.062 478 524 s		Counter Event LSU		
0.187 321 679 s		Counter Event LSU		
0.187 333 821 s		Counter Event LSU		
0.437 007 369 s		Counter Event LSU		
0.437 018 548 s		Counter Event LSU		
0.459 887 298 s		Counter Event LSU		

20) ITM (Instruction Trace Macrocell) This example used a ULINK2.

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 μ Vision for display in its Debug (printf) Viewer window. Note: the global variable value from page xyz 10: **10) Watch and Memory Windows ...** must be entered and compiled in Blinky.c in order for this exercise to work.

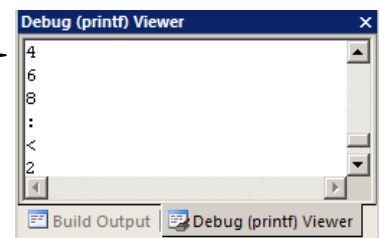
1. Stop the program if it is running and exit Debug mode.
2. Open the project ...\Atmel\SAM3X-EK\Blinky\Blinky.uvproj (do not use RTX_Blinky).
3. Configure the Serial Wire Viewer as described on page 13xyz. Use 84 MHz for the Core Clock. Select EXCTRC.
4. Add this code to Blinky.c. A good place is near line 27, just after the declaration of `value`.

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

5. In the main function in Blinky.c right after `Clock_Is=0`; near line 140, enter these lines:

```
ITM_Port8(0) = (value >>8)+0x30;    /* displays 3rd hex digit of value in ASCII */  
while (ITM_Port8(0) == 0);  
ITM_Port8(0) = 0x0D;  
while (ITM_Port8(0) == 0);  
ITM_Port8(0) = 0x0A;
```

6. Rebuild the source files, program the Flash memory and enter debug mode.
7. Open Debug/Debug Settings and select the Trace tab.
8. Unselect On Data R/W Sample and PC Sample. (this is to help not overload the SWO port)
9. Select EXCTRC and ITM Port 0. ITM Stimulus Port "0" enables the Debug (printf) Viewer.
10. Click OK twice.
11. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
12. In the Debug (printf) Viewer you will see the ASCII of value appear.
13. You can rotate the pot to change the values displayed.



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.

What Is This ?

1. You can see Exception 15 Entry, Exit, Return and the three ITM writes. You probably have to scroll down.
2. ITM 0 frames (Num column) are our ASCII characters from `value` with carriage return (0D) and line feed (0A) as displayed the Data column.
3. All these are timestamped in both CPU cycles and time in seconds.

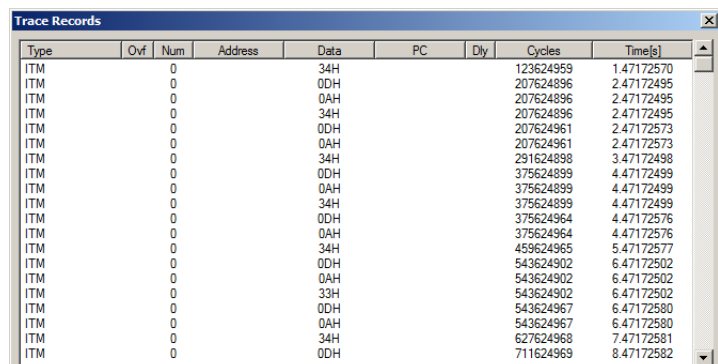
Note: J-Link and SAM-ICE will not display ITM frames. You can stop the program to see the exceptions.

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 SAM3 processor and to your PC via the Serial Wire Output pin.

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

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



Type	Ofv	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM	0	0	34H	34H			123624969	1.47172570
ITM	0	0	0DH	0DH			207624896	2.47172495
ITM	0	0	0AH	0AH			207624896	2.47172495
ITM	0	0	34H	34H			207624896	2.47172495
ITM	0	0	0DH	0DH			207624961	2.47172573
ITM	0	0	0AH	0AH			207624961	2.47172573
ITM	0	0	34H	34H			291624898	3.47172498
ITM	0	0	0DH	0DH			375624899	4.47172499
ITM	0	0	0AH	0AH			375624899	4.47172499
ITM	0	0	34H	34H			375624899	4.47172499
ITM	0	0	0DH	0DH			375624964	4.47172576
ITM	0	0	0AH	0AH			375624964	4.47172576
ITM	0	0	34H	34H			459624965	5.47172577
ITM	0	0	0DH	0DH			543624902	6.47172502
ITM	0	0	0AH	0AH			543624902	6.47172502
ITM	0	0	33H	33H			543624902	6.47172502
ITM	0	0	0DH	0DH			543624967	6.47172500
ITM	0	0	0AH	0AH			543624967	6.47172500
ITM	0	0	34H	34H			627624968	7.47172581
ITM	0	0	0DH	0DH			711624969	8.47172582

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






ARM CMSIS-DSP libraries are offered for 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. The lab for the SAM4 Cortex-M4 is located here: www.keil.com/appnotes/docs/apnt_228.asp

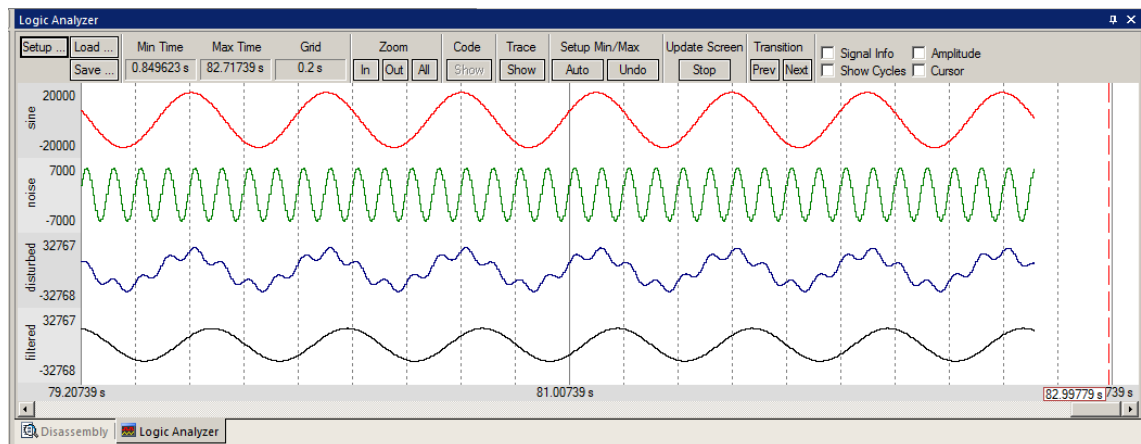
This example creates a sine wave with noise added, and then the noise is filtered out. The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer.

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

To obtain this example file, go to www.keil.com/appnotes/docs/apnt_229.asp MDK 4.54 does not contain it.

Copy these files into C:\Keil\ARM\Boards\Atmel\SAM4S-EK so you have a \DSP directory.

1. Open the project file sine: C:\Keil\ARM\Boards\Atmel\SAM3X-EK\DSP\sine.uvproj
2. Build the files.  There will be no errors or warnings.
3. Program the SAM3X 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. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust the Grid for an appropriate display. Displayed are 4 global variables: sine, noise, disturbed and filtered.
7. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.



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

TIP: The ULINK_{pro} display will be different and the program must be stopped to update it. J-Link or SAM-ICE will not display the data writes.

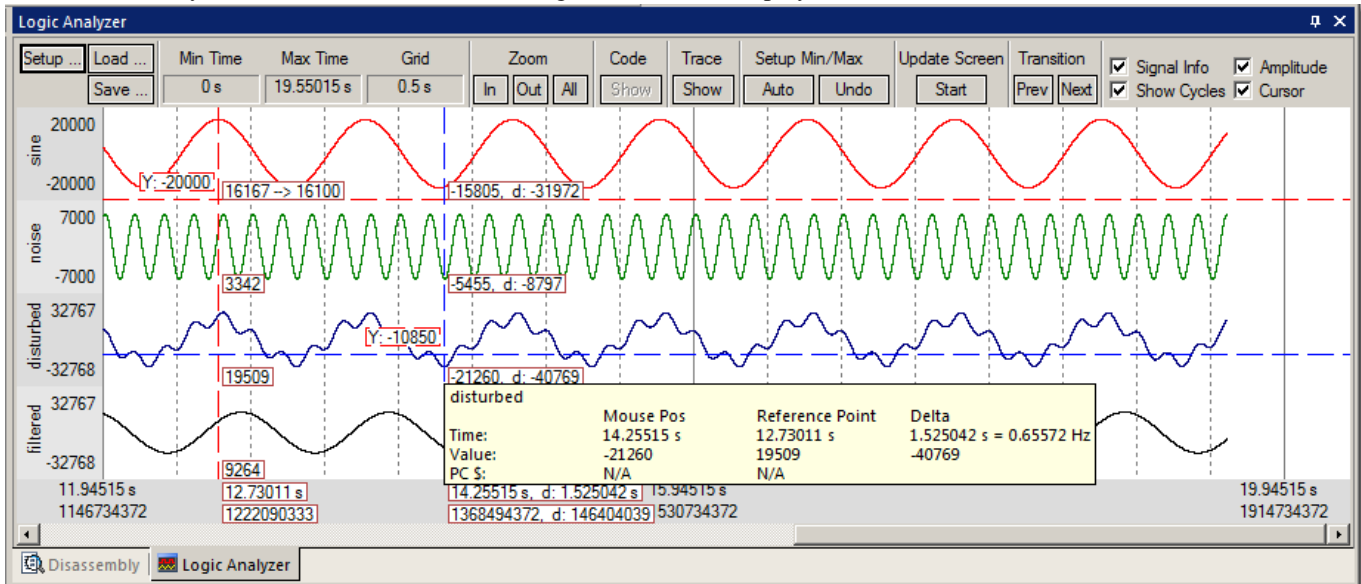
The Watch 1 window will display the four variables updating in real time as shown below:

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

Type	Of	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

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 to resume the collection of data.
6. Open Debug/OS Support and select RTX Tasks and System. A window similar to below opens up.
7. Note this window does not update: nearly all the processor time is spent in the idle daemon. The processor spends relatively little time in each task. 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. 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: 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.

The screenshot shows the RTX Tasks and System window. The System properties section is as follows:

Property	Value
Timer Number:	0
Tick Timer:	10.000 mSec
Round Robin Timeout:	
Stack Size:	200
Tasks with User-provided Stack:	0
Stack Overflow Check:	Yes
Task Usage:	Available: 7, Used: 5
User Timers:	Available: 0, Used: 0

The Tasks section is as follows:

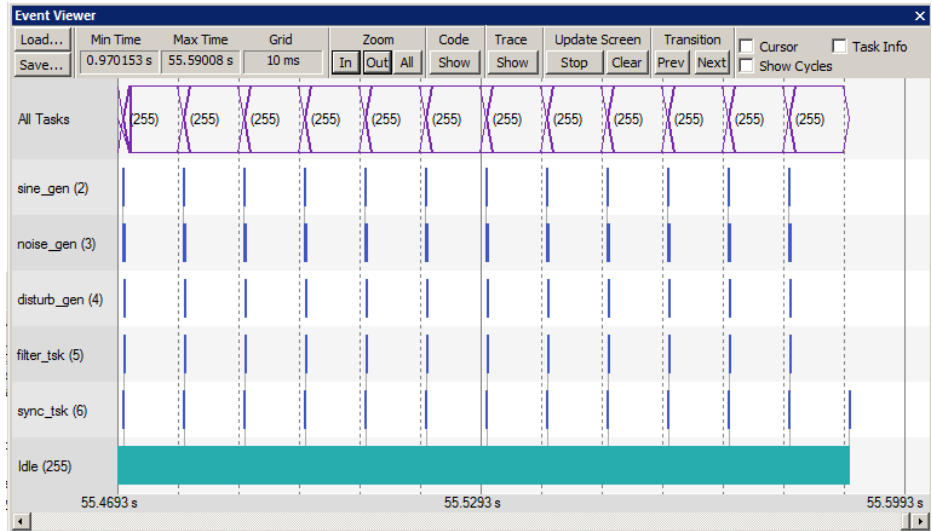
ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Ready				32%
6	sync_tsk	1	Wait_DLY	1			32%
5	filter_tsk	1	Wait_AND		0x0000	0x0001	32%
4	disturb_gen	1	Wait_AND		0x0000	0x0001	32%
3	noise_gen	1	Running		0x0000	0x0001	0%
2	sine_gen	1	Wait_AND		0x0000	0x0001	32%

Event Viewer:

1. Select Debug/Debug Settings and select the Trace tab. Select ITM Port 31. Click on OK twice.
2. Click on RUN.
3. Open Debug/OS Support and select Event Viewer. The window below opens up:
4. Note there is no Task 1 listed. Task 1 is main_tsk and is found in DirtyFilter.c near line 168. It runs some RTX initialization code at the beginning and then deletes itself with os_tsk_delete_self(); found near line 186.

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

TIP: If Event Viewer is still 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.



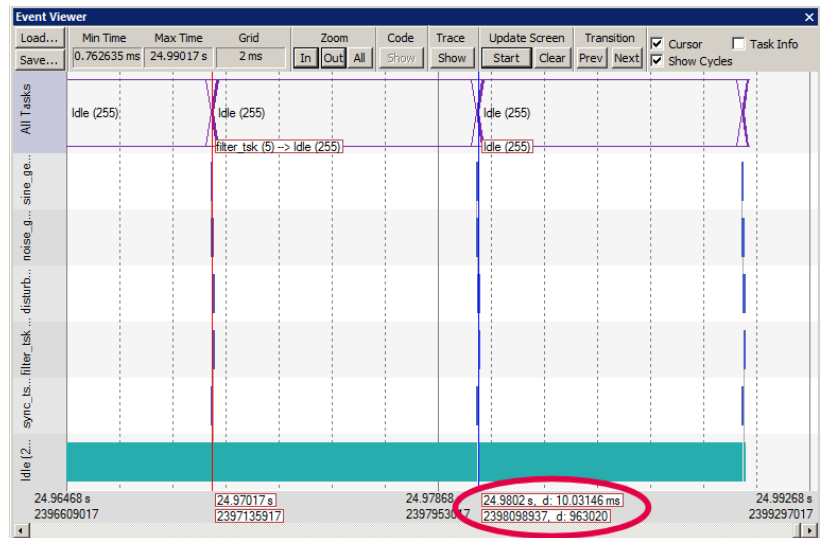
Solutions are to delete some or all of the variables in the Logic Analyzer to free up some bandwidth.

ULINK pro , J-Link and SAM-ICE are much better with SWO bandwidth issues. These have been able to display both the Event and LA windows. ULINK pro uses the faster Manchester format than the slower UART mode that ULINK2 uses.

5. 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.
6. Click Stop in the Update Screen box.
7. Click on Zoom In so three or four tasks are displayed.
8. Select Cursor. Position the cursor over one set of bars and click once. A red line is set here:
9. Move your cursor to the right over the next set and total time and difference are displayed.
10. 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 31 enables 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.

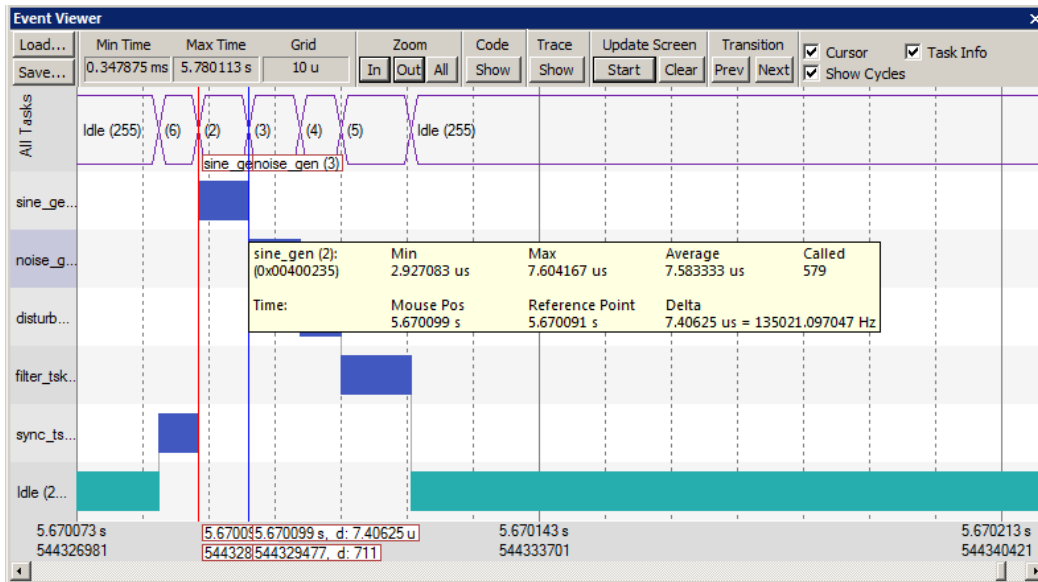


Event Viewer Timing:



1. Click on Zoom In until one set of tasks is visible as shown below:
2. Enable Task Info (as well as Cursor and Show Cycles from the previous exercise).
3. Note one entire sequence is shown. This screen is taken with ULINK pro . Other adapters such as SAM-ICE and ULINK2 might have different values since ULINK pro has the best ability to collect SWV frames without overloads.
4. Click on a task to set the cursor and move it to the end. The time difference is noted. The Task Info box will appear.

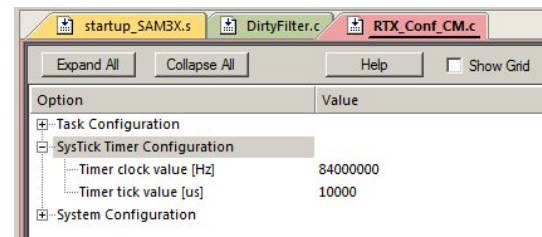
TIP: If the Event Viewer displays no tasks, exit and re-enter debug mode.

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





SysTick Timer Changing:

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\Atmel\SAM3X-EK\DSP\src.
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 84,000,000 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.

This ends the exercises. Thank you.


22) Creating a new project: Using the Blinky source files:

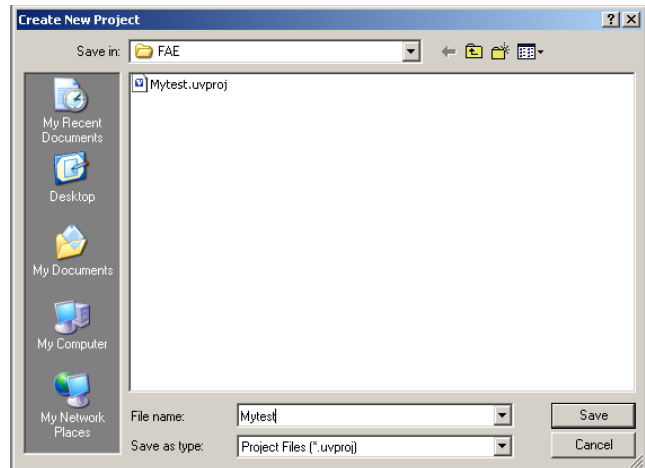
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will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

Create a new project called Mytest:

1. With μ Vision running and not in debug mode, select Project/New μ Vision Project...
2. In the window Create New Project that opens, go to the folder C:\Keil\ARM\Boards\Atmel\SAM3X-EK.

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.



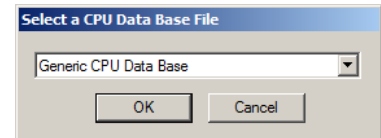
Select your processor:

7. "Select a CPU Data Base File" shown below opens up.
8. Click on OK and the Select Device for "Target 1" opens up as shown below.
9. 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.
10. Locate the Atmel directory, open it and select SAM3X8H (or the device you are using). Note the device features are displayed.
11. Select OK.

μ Vision will configure itself to this device.

Select the startup file:

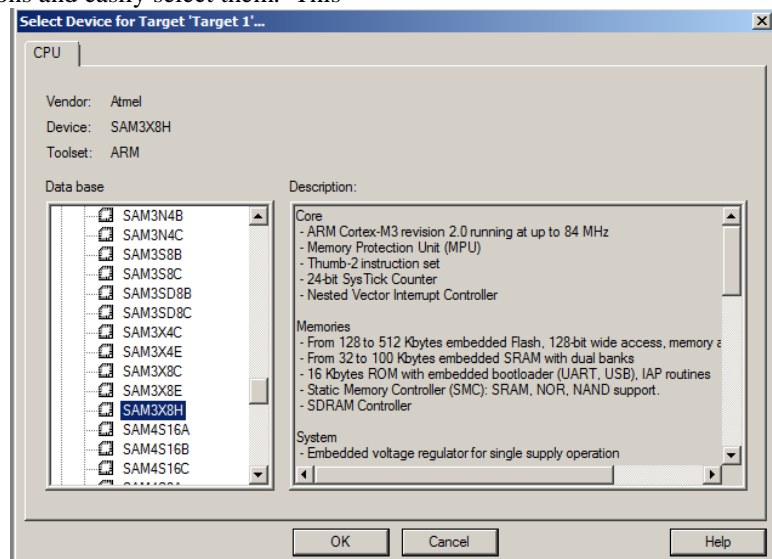
12. A window opens up asking if you want to insert the default SAM3X startup file to your project. Click on "Yes". This will save you some time.
13. In the Project Workspace in the upper left hand of μ Vision, open up the folders Target 1 and Source Group 1 by clicking on the "+" beside each folder.
14. We have now created a project called Mytest with the target hardware called Target 1 with one source assembly file startup_SAM3X.s and using the SAM3X8H processor.



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:

15. 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 SAM3X Flash. Press Enter to accept this change. Note the Target selector in the main μ Vision window also changes to SAM3X Flash.
16. 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.
17. Select File/Save All.




Select the source files and debug adapter:

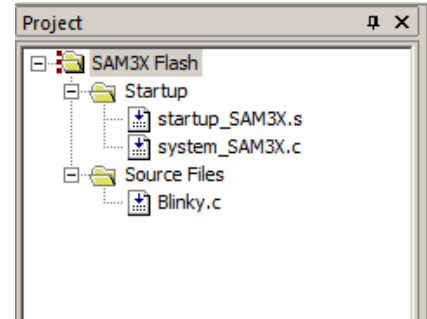
1. Using MS Explore (right click on Windows Start icon), copy blinky.c and system_SAM3X.c from C:\Keil\ARM\Boards\Atmel\SAM3X-EK\Blinky to the ..\SAM3X-EK\FAE folder you created.

Source Files:


2. In the Project Workspace in the upper left hand of μ Vision, right-click on “SAM3X Flash” and select “Add Group”. Name this new group “Source Files” and press Enter. You can name it anything. There are no restrictions from Keil.
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.

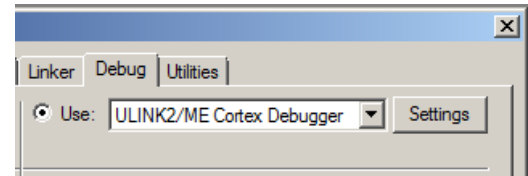
System File:

5. Right-click on “Startup” and select **Add files to Group “Source Files”**.
6. Select the file system_SAM3X.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:

8. By default the simulator is selected when you create a new μ Vision project. You probably need to change this to a USB adapter such as a ULINK2.
9. 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 such as SAM-ICE, J-Link or ULINK*pro*, select the appropriate adapter from the pull-down list.
10. Select JTAG/SWD debugging (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:
11. Select the Utilities tab and select the appropriate debug adapter and the proper Flash algorithm for your processor. Refer to Using Various USB adapters: starting on page 4 for more information.
12. Click on the Target tab and select MicroLIB for smaller programs. See www.keil.com/appnotes/files/apnt202.pdf for details.



Modify Blinky.c

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

```
#include <stdio.h>
#include "SAM3X.h"          /* SAM3X definitions          */

/*-----
   Main Program
   *-----*/
int main (void) {






    while (1) {             /* Loop forever          */

    }

}
```

15. Select File/Save All

Compile and run the program:

16. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
17. Program the SAM3X flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
18. Enter Debug mode by clicking on the Debug icon. 
19. Click on the RUN icon.  Note: you stop the program with the STOP icon. 
20. The program will run but since while(1) is empty – it does not do much. You can set a breakpoint.
21. You should be 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.

23) Serial Wire Viewer Summary:

Serial Wire Viewer can see:

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

Serial Wire Viewer displays in various ways:

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

Trace is good for:

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

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), a 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. ETM trace is best for this.
- Communication protocol and timing issues. System timing problems.

24) Useful Documents:

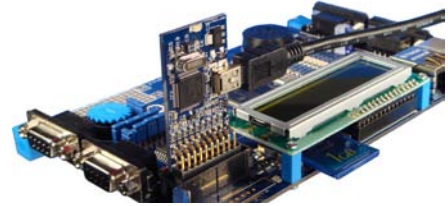
1. **The Definitive Guide to the ARM Cortex-M3** by Joseph Yiu. (he also has one for the Cortex-M0) Search the web.
2. **MDK-ARM Compiler Optimizations: Appnote 202:** www.keil.com/appnotes/files/apnt202.pdf
3. **A list of resources is located at:** <http://www.arm.com/products/processors/cortex-m/index.php>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

25) 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

Note: All versions of MDK now include RTX RTOS with source code.



USB-JTAG adapter (for Flash programming too)

- ULINK2 - \$395 (*ULINK2 and ME - SWV only – no ETM*)
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINKpro - \$1,395 – Cortex-Mx SWV & ETM trace.
MDK also supports SAM-ICE and Segger J-Link Debug adapters.
- **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-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 many other Atmel processors including 8051, ARM7™ and ARM9™ processors. See the Keil Device Database® on www.keil.com/dd for the complete list of Atmel support. This information is also included in MDK.

MDK supports Atmel SAM3, SAM4, SAM7 and SAM9 processors. Check www.keil.com/dd for the complete list.

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 Linux, Android and bare metal (no OS) support on Atmel SAM9 processors, please see DS-5 www.arm.com/ds5.

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

