Atmel Lab: Cortex™-M3 Lab *featuring Serial Wire Viewer*

SAM3S-EK evaluation board using ARM® Keil™ MDK Toolkit

Spring 2012 Version 4.5

by Robert Boys, bob.boys@arm.com

Introduction: *For the latest version of this document please visit www.keil.com/atmel*

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®. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. We also have labs for SAM3S and SAM4S. See www.keil.com/atmel for more information.

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 the 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.

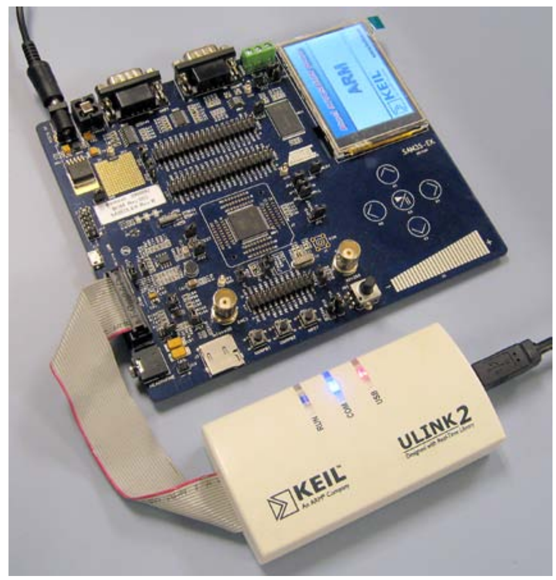
Middleware: MDK Professional contains middleware libraries including TCP/IP stack, CAN drivers, a Flash file system and USB drivers. Contact Keil sales for information regarding middleware for your processor. http://www.keil.com/arm/mdk.asp.

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

Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M3 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.
3. Serial Wire Viewer trace capability is included.
4. RTX Kernel Awareness window. It is updated in real-time.
5. Choice of adapters: ULINK2, ULINK-ME, ULINK*pro*, Segger J-Link and SAM-ICE (version 6 or later for SWV).
6. Kernel Awareness for Keil RTX, CMX, Quadros and Micrium. All RTOSs will compile with MDK.
7. Keil Technical Support is included for one year and is 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:

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 called Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS.
5. USB HID and memory examples.

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:

Using this document:

1. Configuring various debug adapters starts on page 4.
2. ULINK2 is used by default.
3. The first exercise starts on page 8. You can directly go there if using a ULINK2 or a ULINKL-ME.

1) Atmel Evaluation Boards & Keil Evaluation Software:

Keil 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 USB examples. Such middleware is part 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 µVision 4. The evaluation copy of MDK (MDK-Lite) is available free on the Keil website. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence.
Note: MDK 4.23 Blinky examples do not use the LCD. MDK 4.24 has updated examples.

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 speed available.

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 µVision 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. µVision 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. µVision 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.

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 µVision. 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..

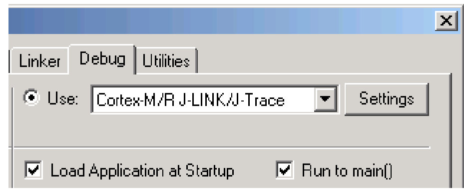
1) Configuring a Segger J-Link and SAM-ICE: (Atmel SAM-ICE is pictured on page 6)

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.

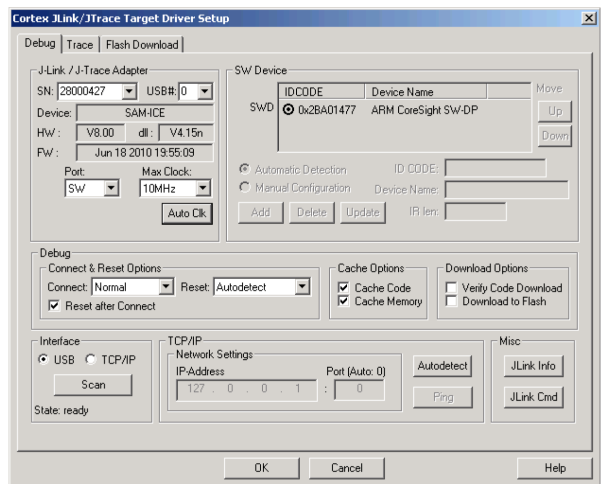
USB Drivers: The Segger USB drivers are located in C:\Keil\ARM\Segger\USBDriver should you need them.

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



Select the debug connection to the target:

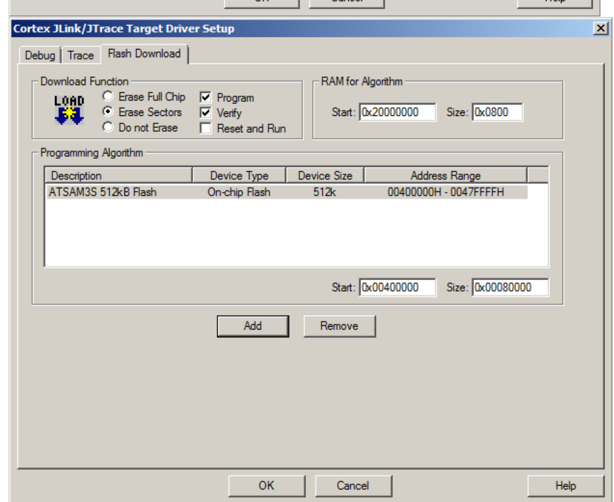
2. Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select the J-Link as shown here:
3. Select Settings and the next window below opens up. This is the control panel for the J-Link.
4. In Port: select SW. SWV will not work with JTAG. If you will not use SWV, you can use JTAG.
5. Clicking on Auto Clk will select the highest JTAG 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 µVision is unable to connect to the J-Link via USB.



TIP: To refresh this screen select Port: or click OK once in order to leave and then reenter this screen.

Configure the Keil Flash Programmer:

7. Click on OK once and select the Utilities tab.
8. Select the J-Link in the drop down box as done before.
9. Click Settings to select the programming algorithm.
10. Select Add and select ATSAM3S Flash if necessary: For this board it is: ATSAM3S 512kB Flash.
11. If using a SAM3U device, select ATSAM3U Flash.
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 µVision 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 14.

2) Configuring a Keil ULINK2 or ULINK-ME: Keil examples are configured for ULINK2 by default.

Serial Wire Viewer is completely supported by these two adapters. They are essentially the same devices electrically and 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. The ETM trace will not display as current SAM3 processors do not have ETM trace capabilities. Ulink2 is pictured on page 1 and a ULINK-ME on this page. ULINLK-ME is only available as part of a board package and not separately.

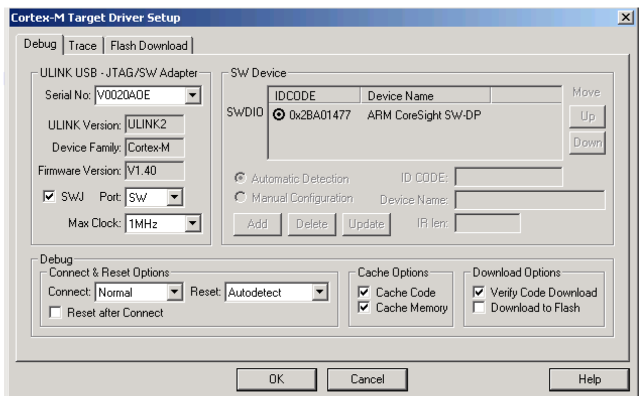
1. Assume the ULINK2 is connected to a powered up Atmel target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project as described in the exercises.

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 ULINK as shown here:
3. 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).
4. In **Port:** select SWJ and SW. SWV will not work with JTAG selected.
5. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board.

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

TIP: You can do regular debugging using JTAG. SWD and JTAG operate at approximately the same speed.



Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the J-Link in the drop down box as done before.
8. Click Settings to select the programming algorithm.
9. Select Add and select ATSAM3S Flash as shown: For this board it is: ATSAM3S 512kB Flash.

14. If using a SAM3U select ATSAM3U Flash.

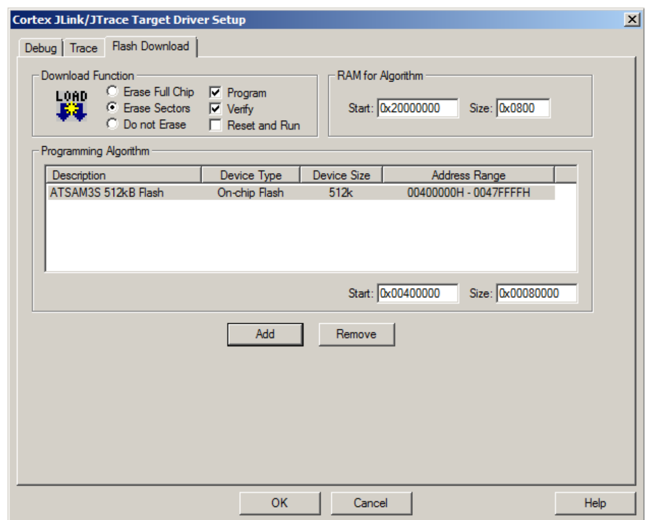
10. Click on OK once.

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

Click on OK to return to the µVision main screen.

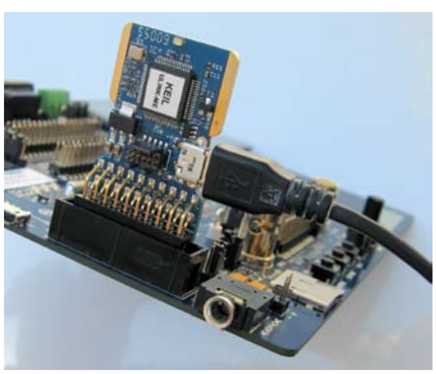
11. You have successfully connected to the SAM3 target

TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this later.



Keil ULINK-ME

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



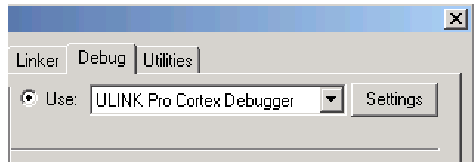
ULINK2 is pictured on page 1. They are essentially electrically and functionally the same.

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 µVision found the wrong Keil adapter connected.

3) Configuring a Keil ULINK*pro*: This is configured the same way as a ULINKK2 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 ULINK2.
3. Select the Utilities tab and select the ULINK*pro* as done before and select the programming algorithm. 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 µVision 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 if trace is enabled or an error will be generated. A special adapter is provided with a ULINK*pro* to connect to the target standard JTAG connector. This small adapter is pictured below plugged into the SAM3S-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 do not have SWV. Contact Keil technical support for SAM9 ETM trace support information.

TIP: µVision windows can be floated anywhere. You can restore them by setting Window/Reset Views to default.

Keil ULINK*pro*

An ETM trace adapter that

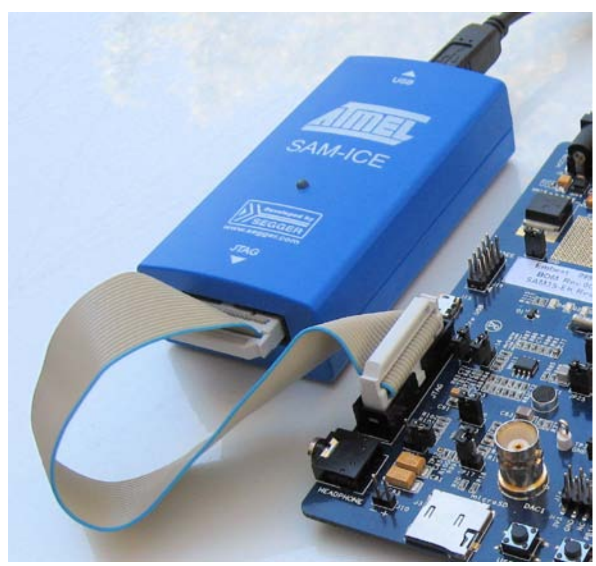
can be used as a ULINK2.
It has very fast Flash programming and an enhanced Instruction Trace window that connects the trace frames to your source code.

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 J-Link Ultra.
It is configured the same way as the J-Link.

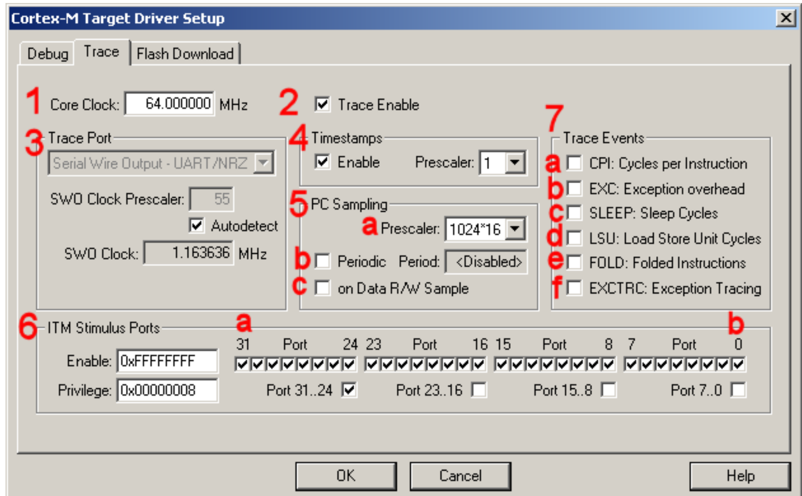


6) Serial Wire Viewer (SWV) Configuration with ULINK2:

The essential place to configure the trace is in the Trace tab as shown below. You cannot set SWV globally for µVision. 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 µVision.

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. It is usually called SYSCLK.
2) **Trace Enable:** Enables SWV and ITM. This does not affect the Watch and Memory window display updates. It can only be changed in Edit mode.
3) **Trace Port:** This is preset.
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 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 µVision. 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 µVision.
   - **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 plus any instruction fetch stalls.
   b. **Fold:** Cumulative number of folded instructions**.** This will result from a predicted branch instruction removed and 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 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 SAM3S-EK and ULINK2:

Now we will connect a Keil MDK development system using real target hardware and a ULINK2 or ULINK-ME.
Note: MDK 4.23 Blinky examples do not use the LCD. MDK 4.24 has updated examples.

1. Connect the equipment as pictured on the first page.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\Blinky\Blinky.uvproj.
4. Make sure "SAM3S Flash" is selected. SAM3S Flash
   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 3 through 5. **Make sure SW is selected and not JTAG in the Port: box.** This is an important step for later.
6. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
7. Program the SAM3S 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.
9. Click on the RUN icon. Note: you stop the program with the STOP icon.

The 2 LEDs on the SAM3S-EK will now blink.

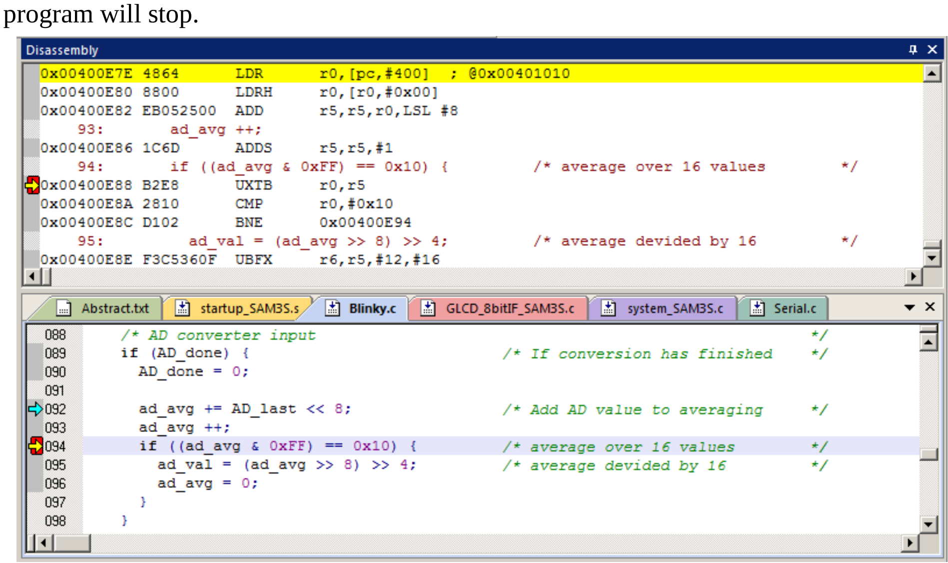
1. Rotating the potentiometer VR1 will change the speed the LEDs blink. The pot is connected to an A/D convertor.
2. The A/D value will also be displayed on the LCD as shown below: This is the local variable ad_val in main().
3. Pressing the Left Click (BP5) and Right Click (BP4) buttons will enhance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

Now you know how to compile a program, program it into the ATSAM3S processor Flash, run it and stop it.

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, double-click on a darker block in the left margin in either the disassembly or an appropriate source window.
2. A red block 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. Remover the breakpoint by double-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.

SAM3 processors have 6 breakpoints.

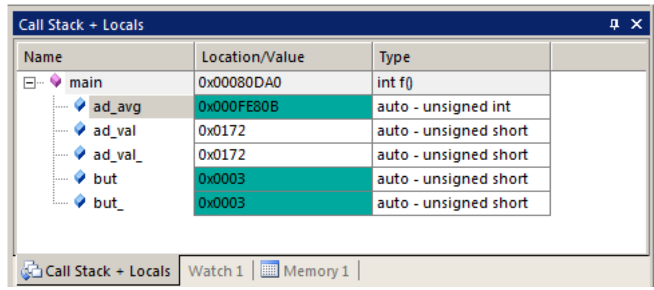
9) Call Stack + Locals Window:

Local Variables:

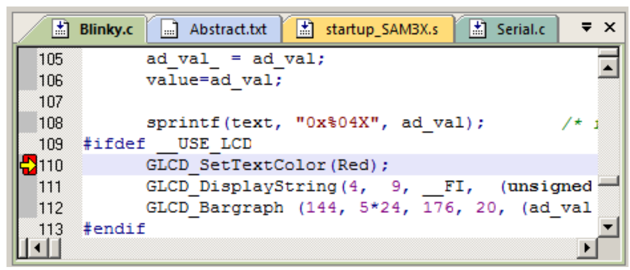
Starting with MDK 4.22, the Call Stack and Local windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function.

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 are 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 110 (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 double-clicking on its red block !**

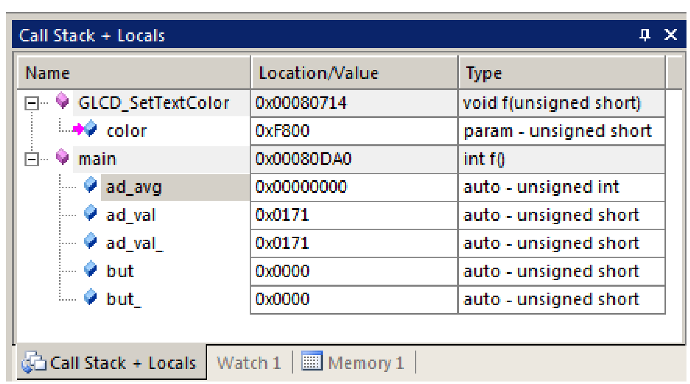


TIP: You can modify a variable value in the Call Stack & Locals window when the program is stopped.

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.

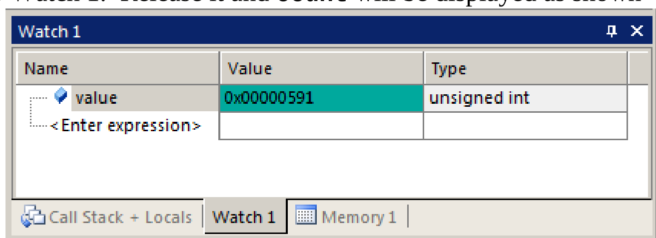
Watch window:

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 109:
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.

```
107      if (ad_val ^ ad_val_) {
108        ad_val_ = ad_val;
109        value = ad_val;
110
```

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 **`count`** will be displayed as shown here:



8. Rotate the pot and value will change in real-time.

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

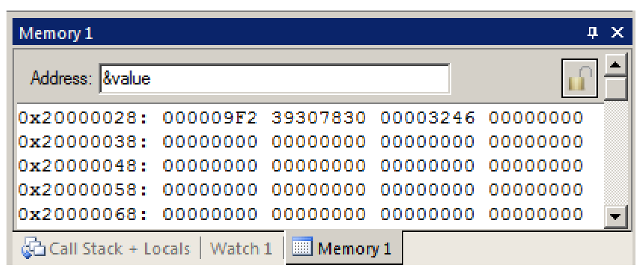
9. You can also enter a variable manually by double-clicking under Name or pressing F2 and using copy and paste or typing the variable.

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.

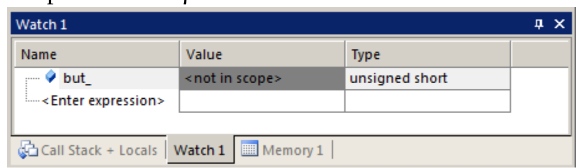
TIP: No CPU cycles are used to perform these operations. See the next page for an explanation how this works.

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 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 µVision 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.

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

8. µVision 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 µVision 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 ! Look near Line

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. LOAD. 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 µVision, 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:

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

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

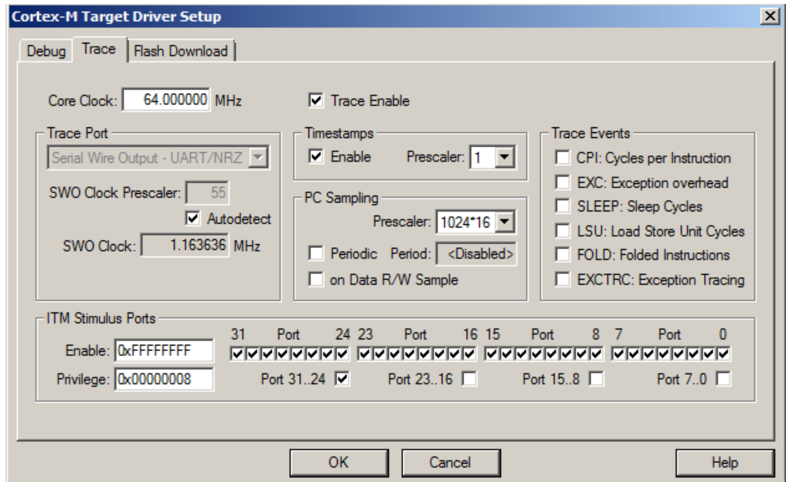
12) View Variables Graphically with the Logic Analyzer (LA):

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.

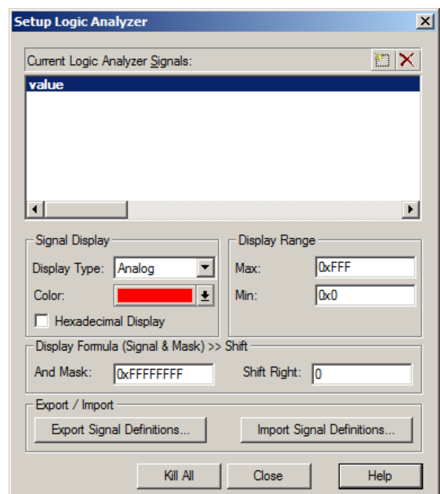
Configure Serial Wire Viewer (SWV): The LA uses SWV to collect the data:

2. Select Options for Target or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window. Confirm SW is selected (J-Link has no SW. This selection is mandatory for SWV. Select the Trace tab.
3. In the Trace tab, unselect Periodic and EXTRC. Set Core Clock: to 64 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 Setup… box and the LA Setup window appears:
5. With value selected, set Display Range Max: to 0xFFF as shown here:
6. Click on Close. Click on Run.

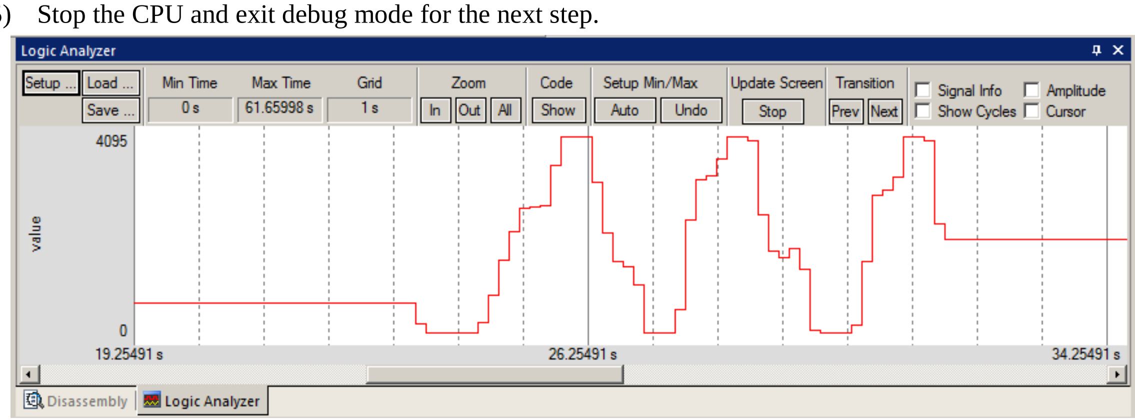


Run Program: Note: The LA can be configured while the program is running.

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.
5) Stop the CPU and exit debug mode for the next step.

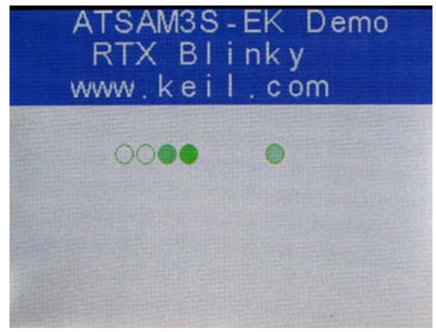


13) *RTX_Blinky* Example Program with Keil RTX RTOS: A Stepper Motor example

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.

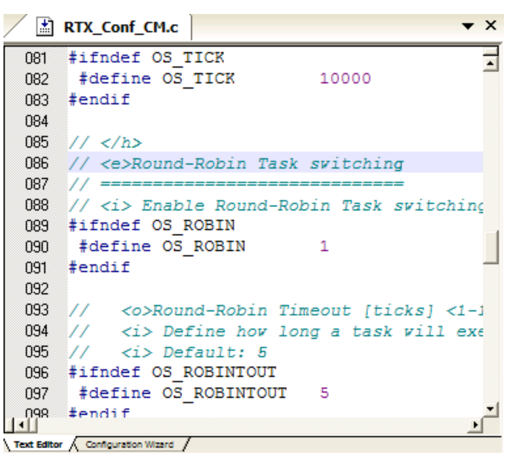
Start µVision by clicking on its icon on your Desktop if it is not already running.

1. Select Project/Open Project.
2. Open the file C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\RTX_Blinky\Blinky.uvproj.
3. If you are not using a ULINK2 or ULINK-ME, you now have to configure µVision 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 .

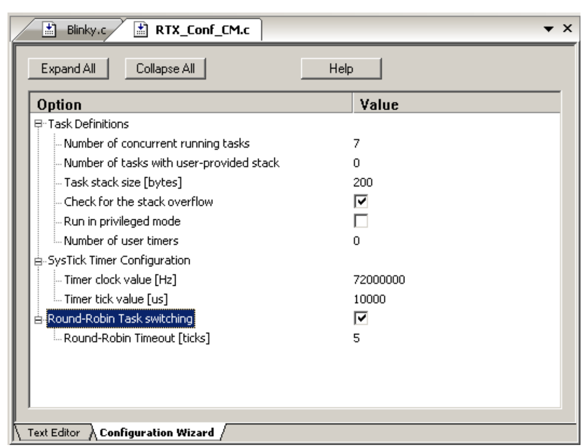


The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open.
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. This is a great feature as it is much easier changing items here than in the source code.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>.
8. The µVision System Viewer windows are created in a similar fashion. Select View/System Viewer.



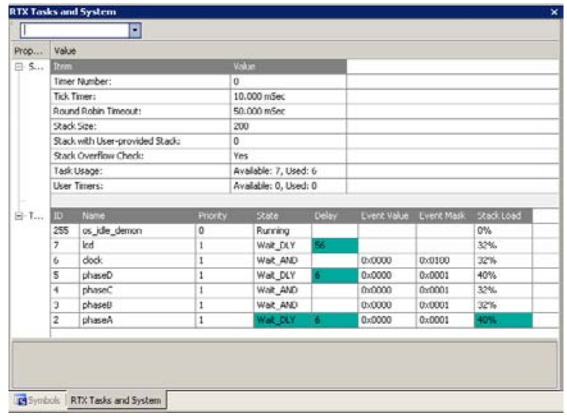
Text Editor: Source Code



Configuration Wizard

14) RTX Kernel Awareness using Serial Wire Viewer

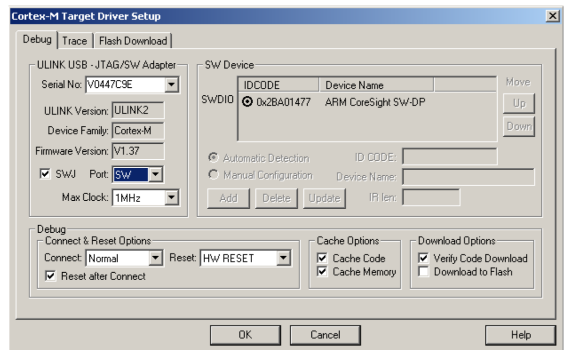
Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides a Task Aware window for RTX. Other RTOS companies also provide awareness plug-ins for µVision.



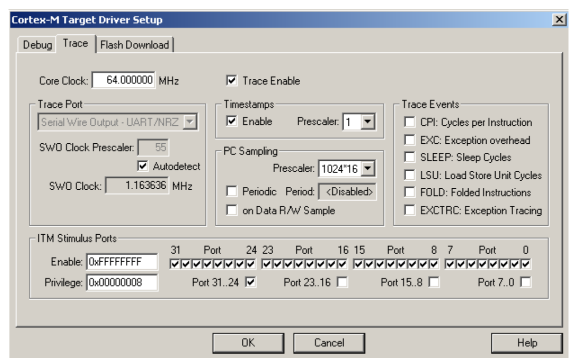
1. Run RTX_Blinky again by clicking on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. Note these values are updated in real-time using the same technology as used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



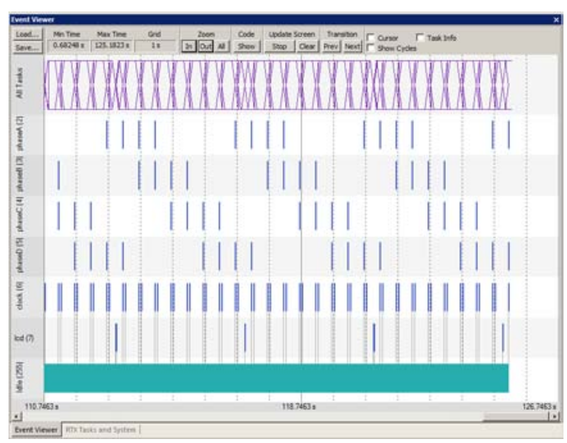
1. Stop the CPU and exit debug mode.
2. Click on the Options icon next to the target box.
3. Select the Debug tab and then click the Settings box next to ULINK Cortex Debugger dialog.
4. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 64 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 µVision.
   The Serial Wire Viewer is now configured in µVision.
10. Enter Debug mode and click on RUN to start the program.
11. Select "Tasks and System" tab: note 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.



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 need be inserted into your source. The Event Viewer uses the ITM Stimulus Ports which is slightly intrusive. You will find feature very useful ! You can use a ULINK2, ULINK-ME, ULINK*pro*, J-Link or SAM-ICE for RTX Kernel Awareness windows.

15) 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. 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 081 and 084 (just after LED_On and LED_Off function calls). For each of the four tasks, add the corresponding variable assignment statements phasea, phaseb, phasec and phased.
4. We do this because in this simple program there are not enough suitable global variables to connect to the Logic Analyzer.

```
028 #define LED_D       0
029 #define LED_CLK     LED_1
030
031 unsigned int phasea;
032 unsigned int phaseb;
033 unsigned int phasec;
034 unsigned int phased;
035
036 /*----------------------------
037  *        Function 'signal_fu
038  *----------------------------
```

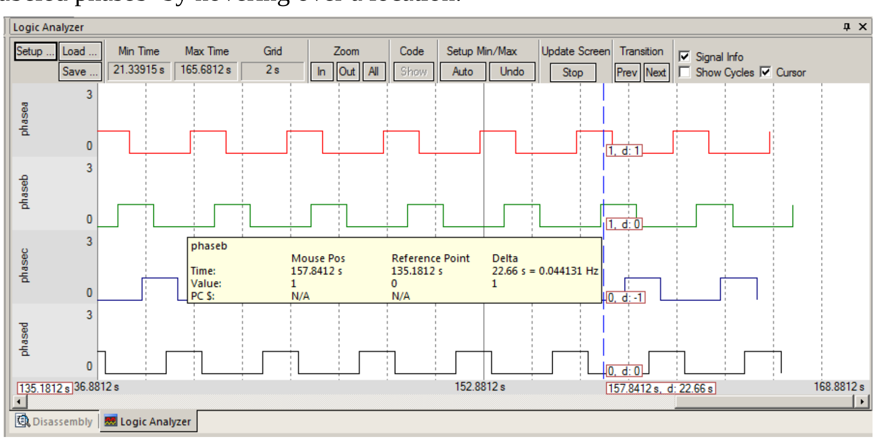
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

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

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.

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, phasec and phased.** These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.
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 2 seconds or so. Move the scrolling bar to the far right if needed.
15. You will see the following waveforms appear. Click on Stop. Select Show Info and Cursor. Click to mark a place See 152 s below. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phaseb 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)

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

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

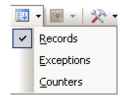
Currently, March 2012, the display of data read, write and ITM trace frames are not implemented in the J-Link or SAM-ICE.

You have configured Serial Wire Viewer (SWV) in Section 14 under **RTX Viewer: Configuring the Serial Wire Viewer:** Page 14

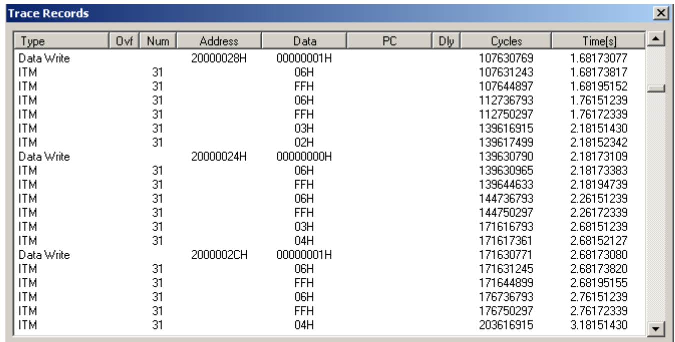
Now we will examine some of the features available to you. SWV works with µVision and a ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

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.

1. Use RTX_Blinky from the last exercise. Enter Debug mode and run the program if not already running.
2. Select View/Trace/Records or click on the Trace icon  and select Records.

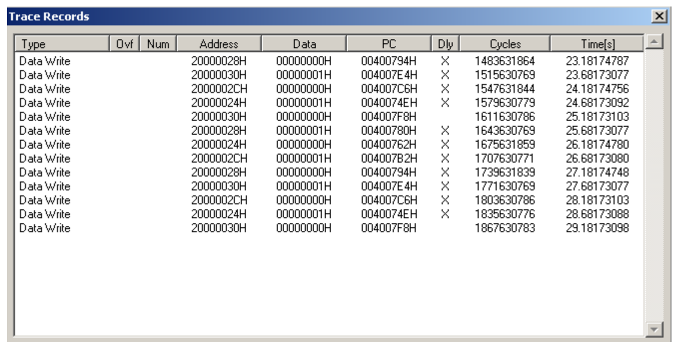


3. The Trace Records window will open up as shown here:



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000028H	00000001H			107630769	1.68173077
ITM		31		06H			107631243	1.68173817
ITM		31		FFH			107644897	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			20000024H	00000000H			139630790	2.18173109
ITM		31		06H			139630965	2.18173383
ITM		31		FFH			139644633	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			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

4. The ITM frames are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. To turn this off select Debug/Debug Settings and click on the Trace tab. Unselect ITM Stimulus Port 31. **TIP:** Port 0 is used for Debug `printf` Viewer.
5. Unselect EXCTRC and Periodic.
6. Select On Data R/W Sample.
7. Click on OK twice to return.
8. Click on the RUN icon.
9. Double-click anywhere in the Trace records window to clear it.
10. Only Data Writes will appear now.
    TIP: You could have right clicked on the Trace Records window to filter the ITM frames out. Unselecting a feature is better as it reduces SWO pin traffic and therefore trace overflows.



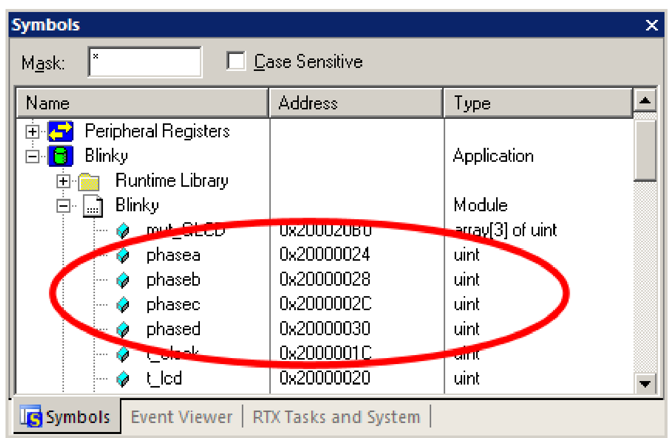
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000028H	00000000H	00400794H	X	1483631864	23.18174787
Data Write			20000030H	00000001H	004007E4H	X	1515630769	23.68173077
Data Write			2000002CH	00000000H	004007C6H	X	1547631844	24.18174756
Data Write			20000024H	00000001H	0040074EH	X	1579630779	24.68173092
Data Write			20000030H	00000000H	004007F8H		1611630786	25.18173103
Data Write			20000028H	00000001H	00400780H	X	1643630769	25.68173077
Data Write			20000024H	00000000H	00400762H	X	1675631859	26.18174780
Data Write			2000002CH	00000001H	004007B2H	X	1707630771	26.68173080
Data Write			20000028H	00000000H	00400794H	X	1739631839	27.18174748
Data Write			20000030H	00000001H	004007E4H	X	1771630769	27.68173077
Data Write			2000002CH	00000000H	004007C6H	X	1803630786	28.18173103
Data Write			20000024H	00000001H	0040074EH	X	1835630776	28.68173088
Data Write			20000030H	00000000H	004007F8H		1867630783	29.18173098

11. What is happening here ?

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

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

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



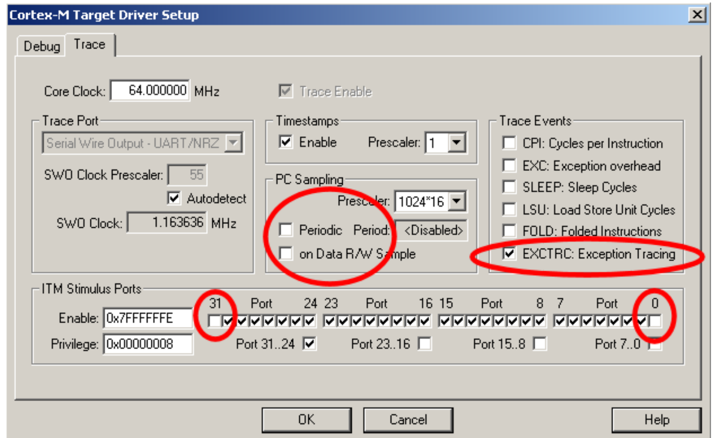
Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

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

2) Exceptions and Interrupts:

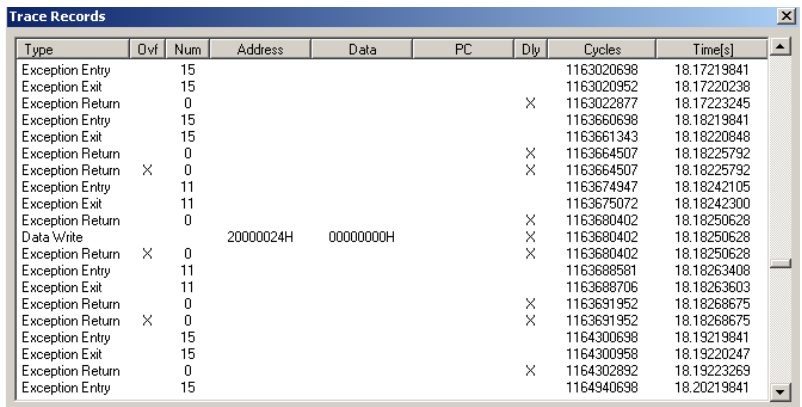
The ATSAM3 family using the Cortex-M3 processor has many interrupts and it can be difficult to determine when they are being activated. Serial Wire Viewer (SWV) on the ATSAM3 family makes this task easy.

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

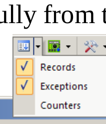


Trace Records

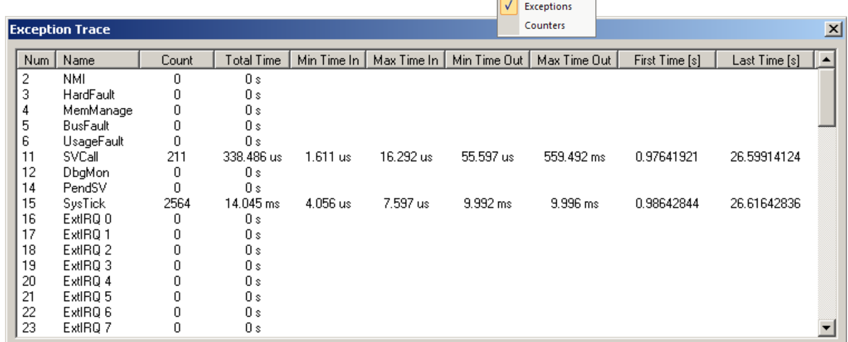
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					1163020698	18.17219841
Exception Exit		15					1163020952	18.17220238
Exception Return		0				X	1163022877	18.17223245
Exception Entry		15					1163660698	18.18219841
Exception Exit		15					1163661343	18.18220848
Exception Return		0				X	1163664507	18.18225792
Exception Return	X	0				X	1163664507	18.18225792
Exception Entry		11					1163674947	18.18242105
Exception Exit		11					1163675072	18.18242300
Exception Return		0				X	1163680402	18.18250628
Data Write			20000024H	00000000H		X	1163680402	18.18250628
Exception Return	X	0				X	1163680402	18.18250628
Exception Entry		11					1163688581	18.18263408
Exception Exit		11					1163688706	18.18263603
Exception Return		0				X	1163691952	18.18268675
Exception Return	X	0				X	1163691952	18.18268675
Exception Entry		15					1164300698	18.19219841
Exception Exit		15					1164300958	18.19220247
Exception Return		0				X	1164302892	18.19223269
Exception Entry		15					1164940698	18.20219841

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. µVision easily recovers gracefully from these overflows.

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



2. The next window opens up and more information about the exceptions is displayed as shown.
3. Note the number of times these have happened under Count. This is very useful information in case interrupts come too fast or slow.
4. ExtIRQ are the processor peripheral interrupts.
5. You can clear this trace window by double-clicking on it.
6. All this information is displayed in real-time and without stealing CPU cycles !



Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	211	338.486 us	1.611 us	16.292 us	55.597 us	559.492 ms	0.97641921	26.59914124
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	2564	14.045 ms	4.056 us	7.597 us	9.992 ms	9.996 ms	0.98642844	26.61642836
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						

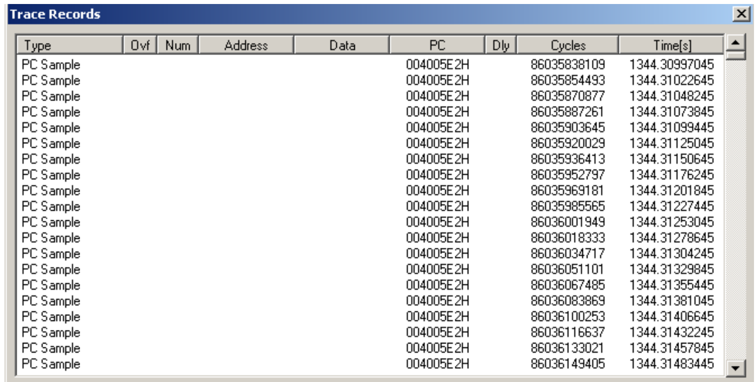
TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

3) PC Samples:

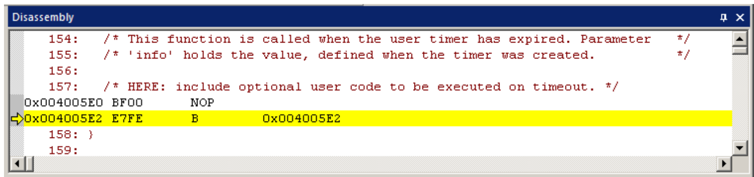
Serial Wire Viewer can display a sampling of the program counter.

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.

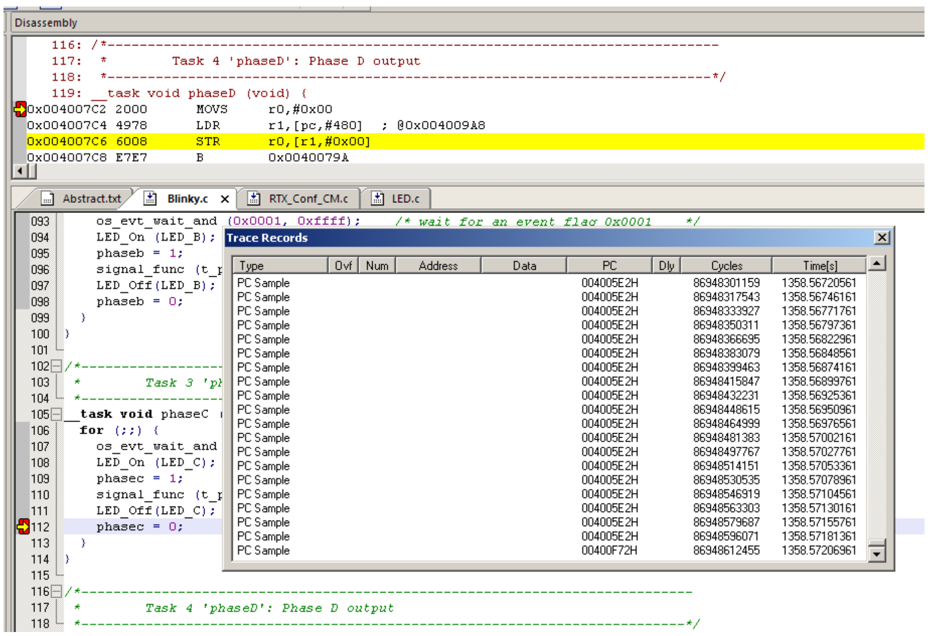
1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear.
5. Click on RUN and this window opens:



6. Most of the PC Samples 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 probably show this Branch:
8. Not all the PCs will be captured. Still, PC Samples can give you some idea of where your program is; especially if it is caught in a tight loop like in this case.



9. **Note:** you can get different PC values depending on the optimization level set in µVision.
10. Set a breakpoint in one of the tasks.
11. 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: You might have to try a few times to see this effect.
12. Scroll to the bottom of the Trace Records window and you might see the correct PC value displayed. Usually, it will be a different PC depending on when the sampling took place.
13. Remove the breakpoint for the next step.



17) 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.

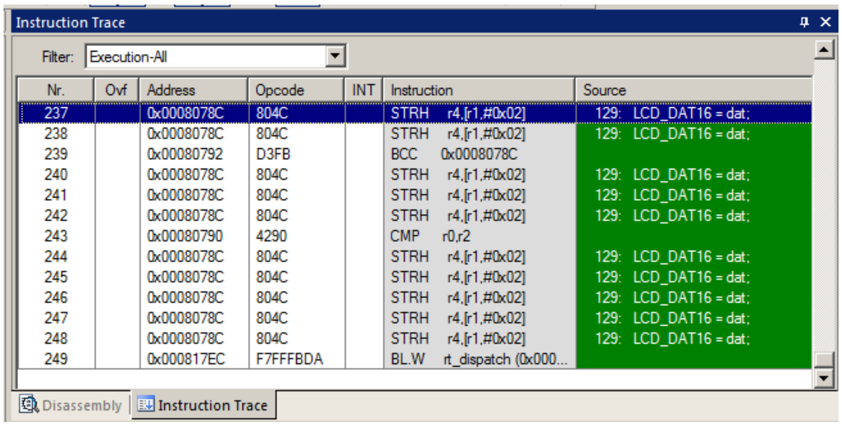
The program must be halted in order to update information in the Instruction Trace window.

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.



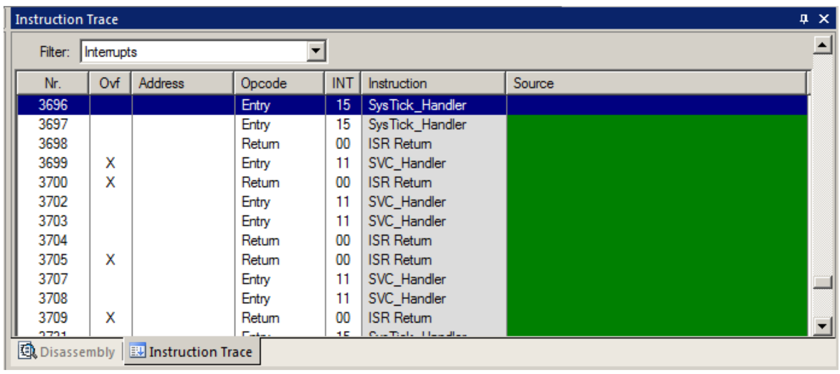
Instruction Trace

Filter: Execution-All

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	
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	
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	F7FFFBDA		BL.W rt_dispatch (0x000...	

Disassembly | Instruction Trace

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



Instruction Trace

Filter: Interrupts

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

Disassembly | Instruction Trace

Two exceptions are listed here: SysTick and SVC call.

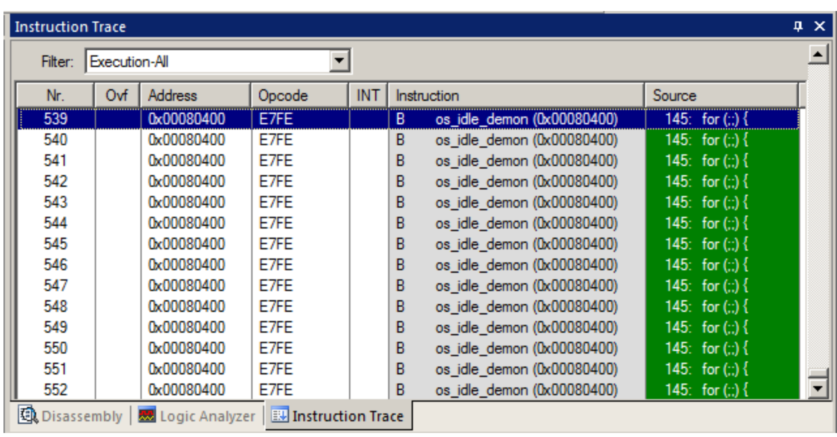
Note the exception entry and exit points are shown. "Entry" can mean either a entry or exit point. Return 00 ISR Return is the return from all exceptions. The absence of this line can mean tail-chaining is happening as the processor goes from one exception directly to the next without popping and pushing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

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.



Instruction Trace

Filter: Execution-All

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 (;;) {

Disassembly | Logic Analyzer | Instruction Trace

18) Keil ULINK*pro* Trace Windows: *for reference*

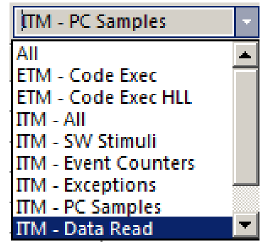
ULINK*pro* provides the most advanced trace windows. New features are being added as development continues.

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 called Records in Keil documents. They are the same thing.

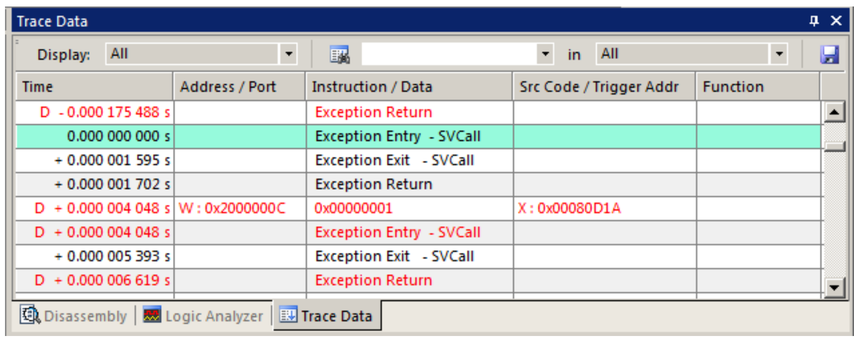


Example Trace Data windows:

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:



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		

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

   Entry: when the exception enters.

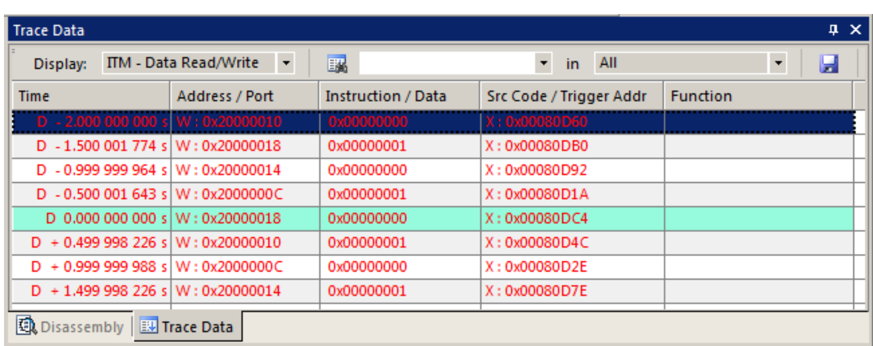
   Exit: When it exits or returns.

   Return: When all the exceptions have returned. This is useful to detect tail-chaining.
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.



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 : 0x00080DB0	
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	

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.

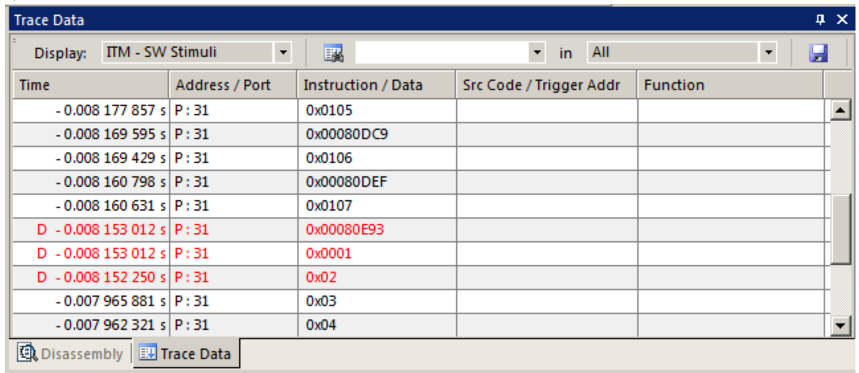
3) ITM: SW Stimuli is selected in the Display: filter box.

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 µVision.

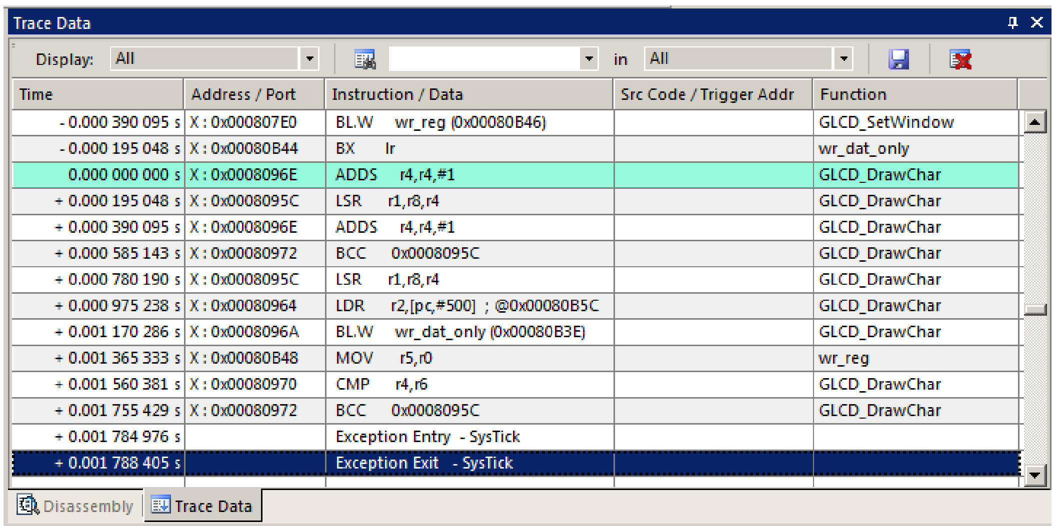
There is an ITM printf exercise on the next page. You will be able to see Port 0 frames in this window or the Trace Records window if you are using a ULINK2.



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	0x00080DC9		
- 0.008 169 429 s	P : 31	0x0106		
- 0.008 160 798 s	P : 31	0x00080DEF		
- 0.008 160 631 s	P : 31	0x0107		
D - 0.008 153 012 s	P : 31	0x00080E93		
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.

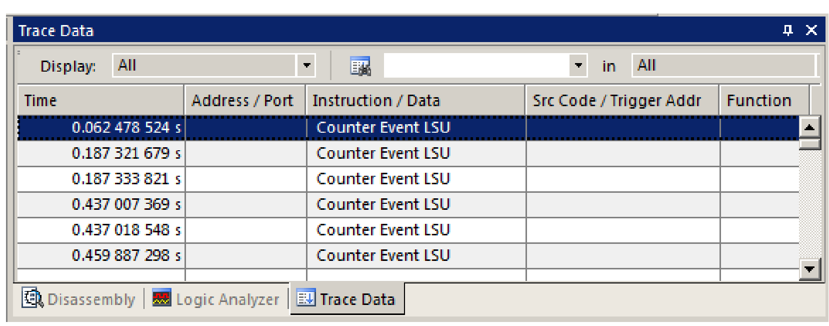
3) Counters:

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.

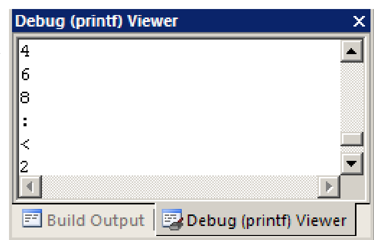


Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
0.062 478 524 s		Counter Event LSU		
0.187 321 679 s		Counter Event LSU		
0.187 333 821 s		Counter Event LSU		
0.437 007 369 s		Counter Event LSU		
0.437 018 548 s		Counter Event LSU		
0.459 887 298 s		Counter Event LSU		

19) ITM (Instruction Trace Macrocell)

Recall that we showed you can display information about the RTOS in real-time using the RTX Viewer. This is done through ITM Stimulus Port 31. ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in its Debug (printf) Viewer window. Note the global variable value must have been created in a previous exercise on page 10.

1. Exit Debug mode and open the project …\Atmel\SAM3S-EK\Blinky\Blinky.uvproj (do not use RTX_Blinky).
2. Configure the Serial Wire Viewer as described in Steps 1 through 9 on page 12. 64 MHz. Leave EXCTRC selected.
3. Add this #define to Blinky.c. A good place is near line 16, just after the declaration of **`value`**.
```
#define ITM_Port8(n)   (*((volatile unsigned char *)(0xE0000000+4*n)))
```
4. In the main function in Blinky.c right after the clock_1s = 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;
```
5. Rebuild the source files, program the Flash memory and enter debug mode.
6. Open Debug/Debug Settings and select the Trace tab.
7. Unselect On Data R/W Sample, PC Sample and ITM Port 31. (this is to help not overload the SWO port)
8. Select EXCTRC and ITM Port 0. ITM Stimulus Port "0" enables the Debug (prinftf) Viewer.
9. Click OK twice.
10. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
11. In the Debug (printf) Viewer you will see the ASCII value of count appear.

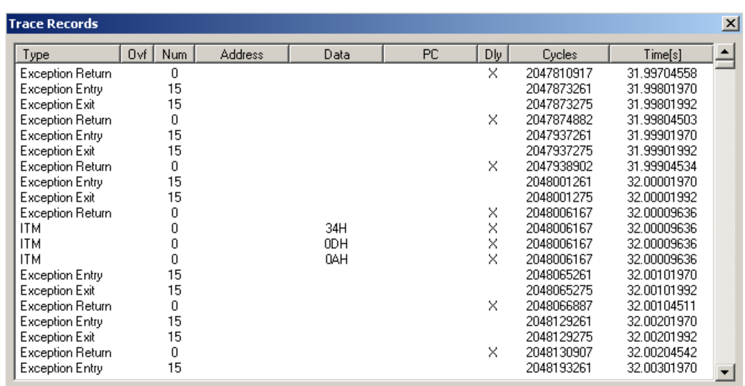


Trace Records

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 **`count`** 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.
4. Note the "X" in the Dly column. This means the timestamps might not be correct due to SWO pin overload.
5. Right click in the Trace Records window and deselect Exceptions. Now you will see only the ITM writes.


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Return		0				X	2047810917	31.99704558
Exception Entry		15					2047873261	31.99801970
Exception Exit		15					2047873275	31.99801992
Exception Return		0				X	2047874882	31.99804503
Exception Entry		15					2047937261	31.99901970
Exception Exit		15					2047937275	31.99901992
Exception Return		0				X	2047938902	31.99904534
Exception Entry		15					2048001261	32.00001970
Exception Exit		15					2048001275	32.00001992
Exception Return		0				X	2048006167	32.00009636
ITM		0		34H		X	2048006167	32.00009636
ITM		0		0DH		X	2048006167	32.00009636
ITM		0		0AH		X	2048006167	32.00009636
Exception Entry		15					2048065261	32.00101970
Exception Exit		15					2048065275	32.00101992
Exception Return		0				X	2048066887	32.00104511
Exception Entry		15					2048129261	32.00201970
Exception Exit		15					2048129275	32.00201992
Exception Return		0				X	2048130907	32.00204542
Exception Entry		15					2048193261	32.00301970

Note: J-Link and SAM-ICE will not display ITM frames. 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 ATSAM3 processor via the Serial Wire Output pin.

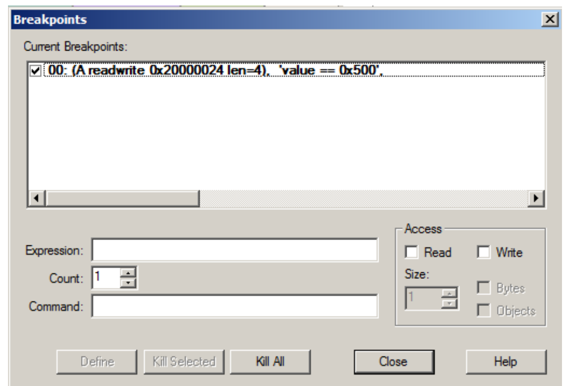
TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enter only those features that you really need.

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

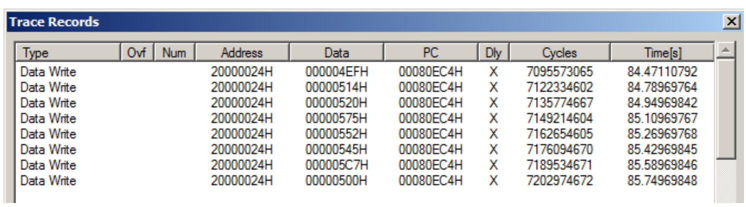
20) **Watchpoints:** *Conditional Breakpoints*

SAM3 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 µVision you must have two variables free in the Logic Analyzer to use Watchpoints.

1. Open the project Blinky that you used before if not already open. Exit Debug mode if necessary.
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. Compile the project and program the Flash. Enter Debug mode.
5. Add variable **`value`** to the Logic Analyzer. Click on Setup and set the Display Range to Min: 0x0 and Max: 0xFFF.
6. Click on Close to return. Set Grid to 1 second by using the Zoom: Out button in the Logic Analyzer window.
7. Select Debug in the main µVision window and select Breakpoints or press Ctrl-B.
8. In the Expression box enter: "**`value`** == 0x500" without the quotes. Select both the Read and Write Access.
9. Click on Define and it will be accepted as shown here:



10. Click on Close.
11. Enter the variable **`value`** to the Watch 1 window by dragging and dropping it if it is not already there.
12. Open Debug/Debug Settings and select the trace tab. Check "on Data R/W sample", uncheck EXTRC and ITM 31 and 0.
13. Click on OK twice. Open the Trace Records window.
14. Click on RUN. Rotate the pot so 0x500 is displayed on the LCD. You might have to adjust the pot very carefully.
15. You will see **`value`** change in the Logic Analyzer as well as in the Watch window.
16. When **`value`** equals 0x500, the Watchpoint will stop the program. The LCD might not display exactly 0x500 due to sample timing.
17. 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.


Trace Records

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	00000552H	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

18. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.
19. To repeat this exercise, click on RUN and rotate the pot again.
20. When finished leave Debug mode.
21. 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.

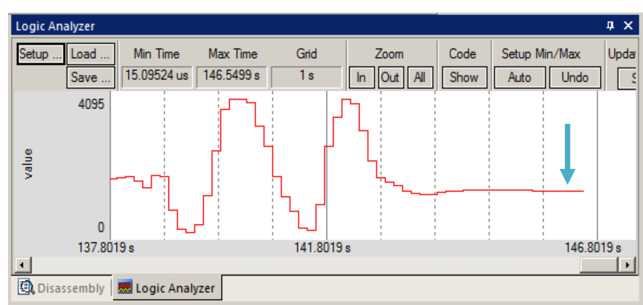
TIP: To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint. You should delete the old one by highlighting it and click on Kill Selected or try the next TIP:

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`**.

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



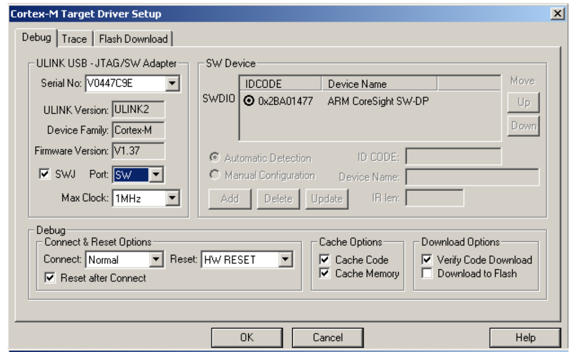
21) USB HID (Human Interface Device):

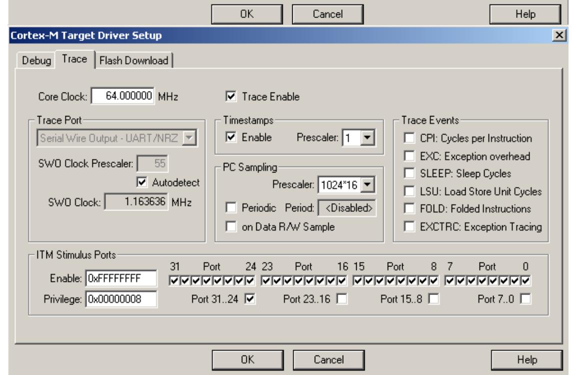
This USB example provided with MDK demonstrates two way communications over USB using the HID interface. A HID client program is provided to run on your PC and communicate with the SAM3S-EK. Pressing the USRPB1 and USRPB2 will activates corresponding boxes in the HID Client window. Selecting output boxes result in control over the LEDs.

1. Open the project C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\USBHID\HID.uvproj
2. Select the USB JTAG adapter at this point if not using a ULINK2.
3. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
4. Program the SAM3S flash by clicking on the Load icon: Progress will be indicated in the Output Window.

Configure the Serial Wire Viewer (SWV):

Serial Wire Viewer is not needed to run the USB example but we will use it to display some interesting things.





1. Click on the Options icon next to the target box.
2. Select the Debug tab and then click the Settings box next to ULINK Cortex Debugger dialog.
3. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.
4. Click on the Trace tab to open the Trace window.
5. Set Core Clock: to 64 MHz and select Trace Enable.
6. Unselect the Periodic and EXCTRC boxes as shown here.
7. Click on OK twice to return to µVision.
   The Serial Wire Viewer is now configured in µVision.

Connect the USB:

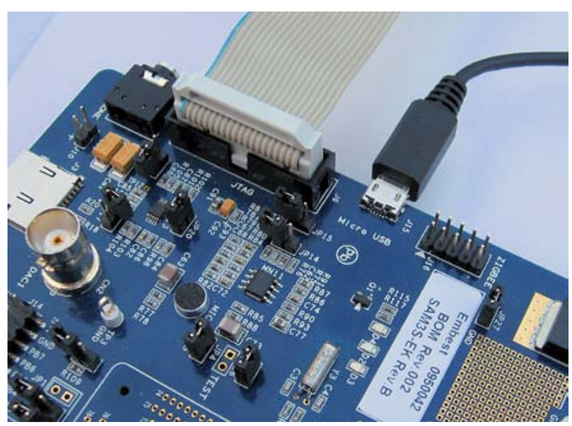
8. Connect a Micro USB cable to the SAM3S-EK and a PC as shown below to the right: Drivers will install.

Start the HID Demo program and start the HID Client:

9. Enter Debug mode.
10. Click Run to start the program.
11. Start the HID Client. You need not install it to Windows. Just double-click on its filename. **HIDClient.exe** is found in C:\Keil\ARM\Utilities\HID_Client\Release.
12. USB will enumerate and a device will appear in HID Client as shown below. Select this device. The boxes will not be grayed out which would mean loss of the USB connection.

    As you select Outputs 0 and 1, the LEDs will toggle. RED is normally ON, the others toggle. Add these lines near Line 77 to toggle the RED led. Rebuild, Flash and Run the program.

```
if (OutReport & 0x04)
PIOC->PIO_CODR = (1<<20);
else
PIOC->PIO_SODR = (1<<20);
```

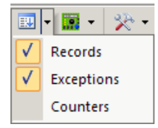


13. Pressing USRPB1 and USRPB2 (or both) will cause an effect in the Inputs (Buttons) boxes. Try not to press the RESET button !
14. If you stop the CPU, the USB connection will stop. To reactivate the HID Client, reselect the Device option.
15. Turn the page for more details of this USB exercise including using the Serial Wire Viewer.

TIP: If the HID Client will not display the ATSAM3 board, close and restart the HID Client.

Viewing the USB Interrupt using Serial Wire Viewer:

1. The Serial Wire Viewer must be enabled for these examples to work.
2. Leave the program running or stopped but still in Debug mode:
3. Open Debug/Debug Settings and click on the Trace tab.
4. Select EXCTRC to collect and display the exceptions. Click on OK twice.
5. Click on RUN to start the program again.
6. Open the Trace Records window and the Exception Trace window.
   You can select View/Trace or open this window up:



7. The bottom two windows open up:
8. Exception 50 is displayed in the Trace Records window and is updated in real-time. You can see the Entry, Exit and Return frames quite easily and the times they occurred.
9. The Exception Trace window displays each exception in a different format. The number of occurrences with various timing information is displayed.

TIP: If these windows do not update and do so only when you stop the processor: make sure View/Periodic Window Update is selected.

10. Exception 50 is also known as ExtIRQ 34. (50 - 16 = 34). ExtIRQ 34 is listed in the ATSAM3 datasheet as the exception for the UDP (USB Device Port). See Table 37-3 below from that document.

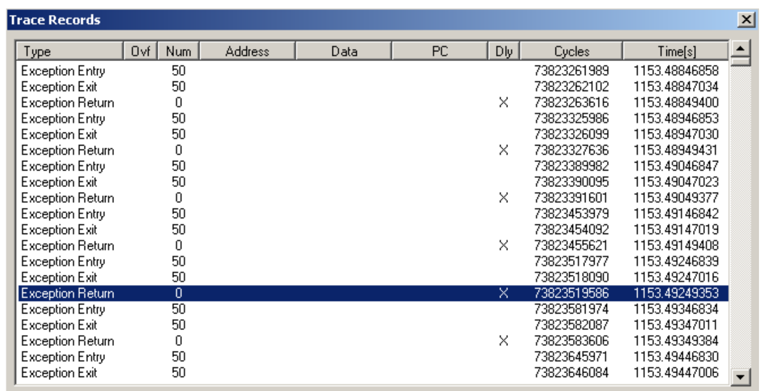
Recall: Exceptions:

Interrupts are a subset of Exceptions in ARM terminology.

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned to the main program including any Cortex-M3 tail-chaining.

TIP: You can double-click inside either of these windows to clear them.

Note: In the Instruction Trace window when using the J-Link or SAM-ICE, you must stop the program to see the exception frames displayed. The ULINK2 or ULINK-ME updates these automatically while the program is running.

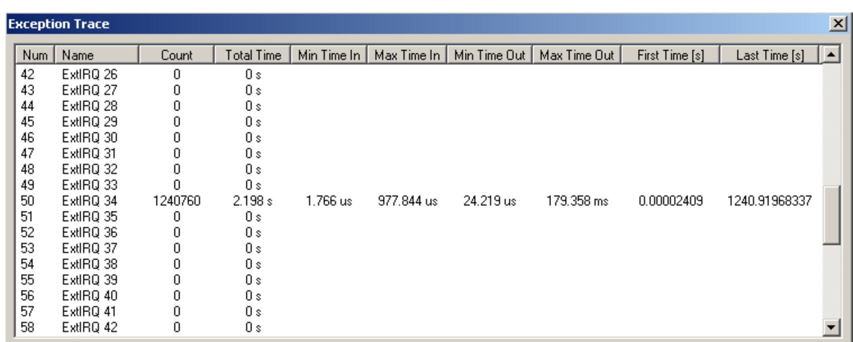


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		50					73823261989	1153.48846858
Exception Exit		50					73823262102	1153.48847034
Exception Return		0				X	73823263616	1153.48849400
Exception Entry		50					73823325986	1153.48946853
Exception Exit		50					73823326099	1153.48947030
Exception Return		0				X	73823327636	1153.48949431
Exception Entry		50					73823389982	1153.49046847
Exception Exit		50					73823390095	1153.49047023
Exception Return		0				X	73823391601	1153.49049377
Exception Entry		50					73823453979	1153.49146842
Exception Exit		50					73823454092	1153.49147019
Exception Return		0				X	73823455621	1153.49149408
Exception Entry		50					73823517977	1153.49246839
Exception Exit		50					73823518090	1153.49247016
Exception Return		0				X	73823519586	1153.49249353
Exception Entry		50					73823581974	1153.49346834
Exception Exit		50					73823582087	1153.49347011
Exception Return		0				X	73823583606	1153.49349384
Exception Entry		50					73823645971	1153.49446830
Exception Exit		50					73823646084	1153.49447006

Table 37-3. Peripheral IDs

Instance	ID
UDP	34



Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
42	ExtIRQ 26	0	0 s						
43	ExtIRQ 27	0	0 s						
44	ExtIRQ 28	0	0 s						
45	ExtIRQ 29	0	0 s						
46	ExtIRQ 30	0	0 s						
47	ExtIRQ 31	0	0 s						
48	ExtIRQ 32	0	0 s						
49	ExtIRQ 33	0	0 s						
50	ExtIRQ 34	1240760	2.198 s	1.766 us	977.844 us	24.219 us	179.358 ms	0.00002409	1240.91968337
51	ExtIRQ 35	0	0 s						
52	ExtIRQ 36	0	0 s						
53	ExtIRQ 37	0	0 s						
54	ExtIRQ 38	0	0 s						
55	ExtIRQ 39	0	0 s						
56	ExtIRQ 40	0	0 s						
57	ExtIRQ 41	0	0 s						
58	ExtIRQ 42	0	0 s						

SAM3S USB support is provided with MDK-Professional™ in C:\Keil\ARM\Boards\Atmel\SAM3S-EK\RL\USB\Device. You need a full MDK license to compile the files in \RL. No license is needed to compile the examples in this document. No royalty payments or per-project fees are required with RTX or any Keil Middleware. Contact Keil sales for more details.

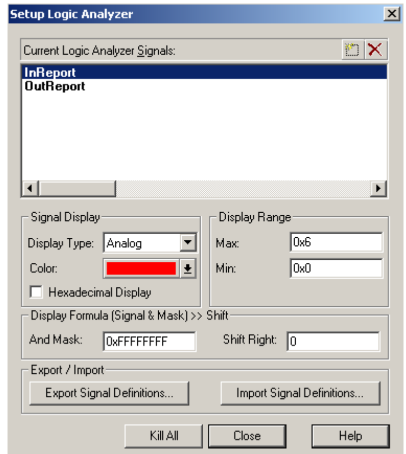
Viewing Global Variables as they change using Serial Wire Viewer:

Demo.c has two interesting global variables we can view. They are of type **`unsigned char`**:

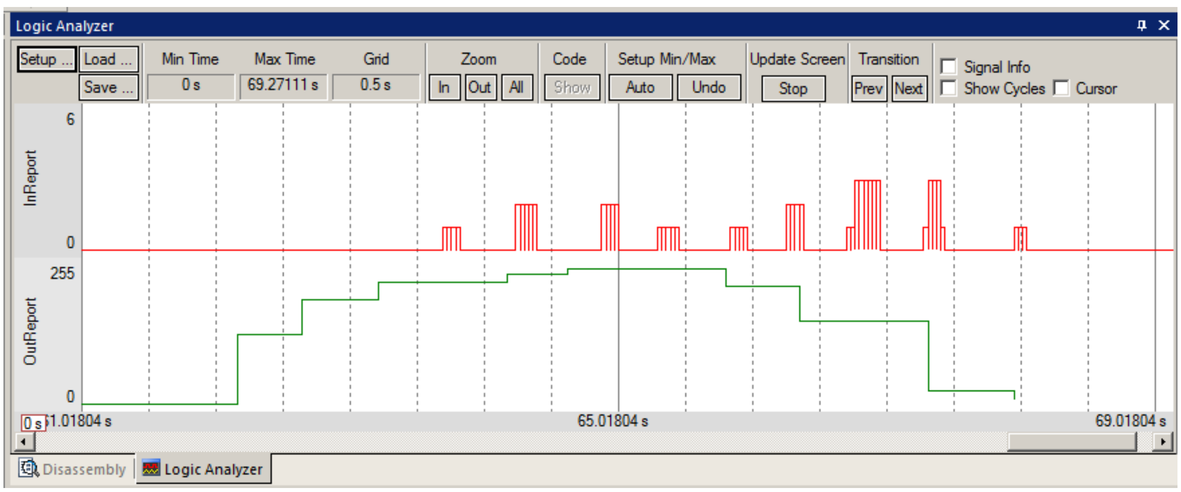
InReport – two bits contain the button status.

OutReport – three bits contain which LEDs are to be activated.

1. In Debug mode: Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC and Periodic if set. Select "on Data R/W Sample". Click on OK twice to close this window.
3. Close the Exception Trace window and click on RUN to start the program.
4. Open the Logic Analyzer window and enter both the InReport and OutReport variables.
5. Click on Setup and set InReport to max 0x6 and OutReport to 0xFF (the default in this case) as shown here:



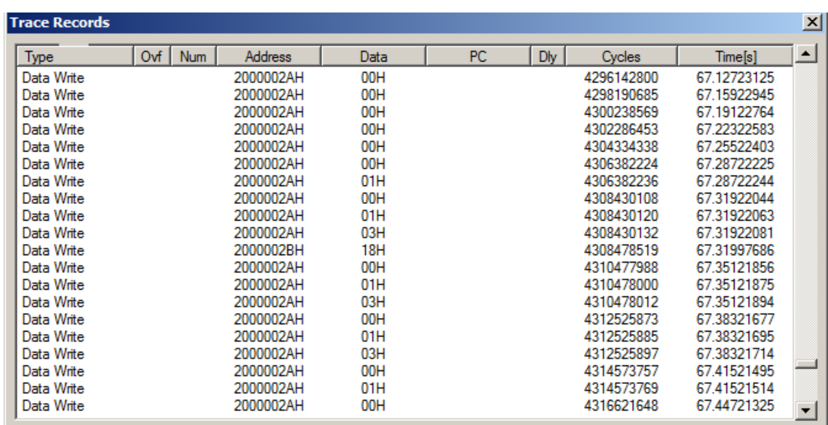
6. Click on Close.
7. Using the Zoom In and Out buttons set Range to 1 second or so.
8. RUN the program and by pressing the two buttons and clicking the boxes in the HID Client you can see these values in the LA as shown:
9. The RED trace shows when the buttons were pushed and have the value from 0 to 3.
10. The Green trace represents the Outputs checkboxes in the HID Client and have a value from 0 to 0xFF.
11. The Trace Records window will display the data write frames to these two variables as shown in the bottom screen:
12. You probably have to scroll to see some interesting numbers.



In the Trace Records window you can see the writes to 0x2000_002A (InReport) and 0x2000_002B (OutReport), the data values written and the time of the write both in CPU cycles and in seconds.

You can see the times representing the short pulses of about 0.19 µsec and the length of time the USRPB1 button was pressed.

TIP: If you enable EXCTRC, you can have the USB interrupts display. You must be aware that enabling too many SWV features can lead to missing frames. These will be indicated by a "X" in Trace Records.



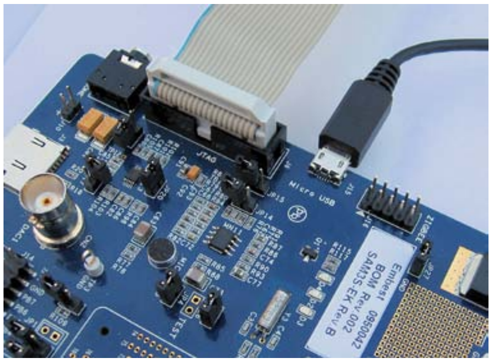
Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			2000002AH	00H			4296142800	67.12723125
Data Write			2000002AH	00H			4298190685	67.15922945
Data Write			2000002AH	00H			4300238569	67.19122764
Data Write			2000002AH	00H			4302286453	67.22322583
Data Write			2000002AH	00H			4304334338	67.25522403
Data Write			2000002AH	00H			4306382224	67.28722225
Data Write			2000002AH	01H			4306382236	67.28722244
Data Write			2000002AH	00H			4308430108	67.31922044
Data Write			2000002AH	01H			4308430120	67.31922063
Data Write			2000002AH	03H			4308430132	67.31922081
Data Write			2000002BH	18H			4308478519	67.31997686
Data Write			2000002AH	00H			4310477988	67.35121856
Data Write			2000002AH	01H			4310478000	67.35121875
Data Write			2000002AH	03H			4310478012	67.35121894
Data Write			2000002AH	00H			4312525873	67.38321677
Data Write			2000002AH	01H			4312525885	67.38321695
Data Write			2000002AH	03H			4312525897	67.38321714
Data Write			2000002AH	00H			4314573757	67.41521495
Data Write			2000002AH	01H			4314573769	67.41521514
Data Write			2000002AH	00H			4316621648	67.44721325

22) USB Memory:

The USB Memory example demonstrates configuring the SAM3S-EK as a USB memory stick of size 8 Kbytes.

1. Stop the program and exit debug mode.
2. Open the project C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\USBMem\Memory.uvproj.
3. Configure the USB JTAG adapter you are using if not the ULINK2 or ULINK-ME.
4. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
5. Program the SAM3S flash by clicking on the Load icon: Progress will be indicated in the Output Window.
6. Connect a USB cable to the Micro USB connector on the SAM3S-EK board to your PC as shown here:
7. Enter Debug mode. Click on Run to start the program.
8. A Windows Explorer window similar to the one below will open up. It will show a file readme.txt in the USB stick.
9. **Note:** You might have to unplug and reinsert the USB cable.
10. Right click in the area underneath README.TXT and select Properties. The Properties window will open up.
11. You can move files in and out of the USB stick as long as you stay within the memory limits.
12. This could be made larger with external RAM.

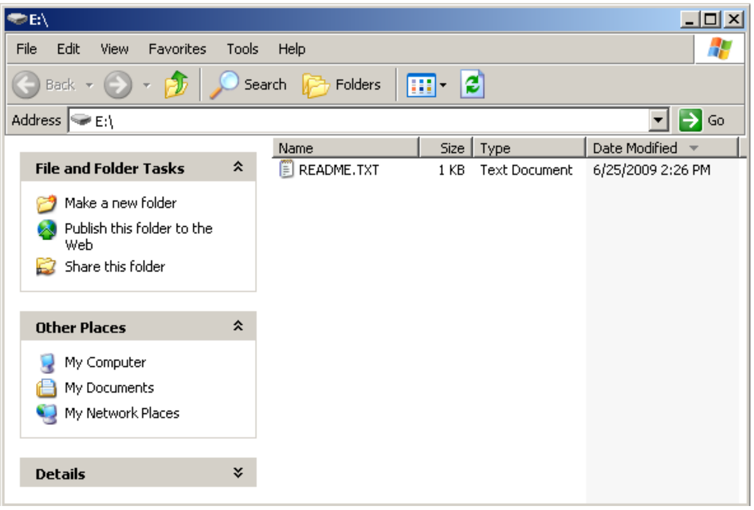


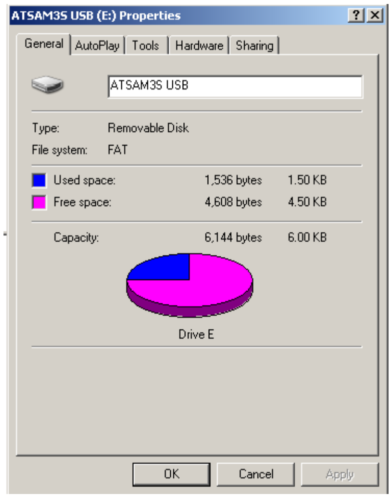
For information on Keil RL-ARM Real-Time Library please visit: www.keil.com/arm/rl-arm/

Remember that all versions of Keil MDK includes RTX with source code. Even the evaluation version MDK Lite.

Keil USB Features: A component with MDK-Professional from Keil.

- RL-USB provides device interfaces for common USB device classes. These interfaces have default support in Windows 2000, XP, Vista, and Windows 7 which reduces the work required to connect your embedded applications to host computers.
- USB Hardware layer and event handler (hardware specific)
- Generic USB core supporting USB 1.1 and 2.0
  - o Low Speed (1.5Mbit/s), Full Speed (12Mbit/s), & High-Speed (480Mbit/s)
- Common USB Device Class support
  - o Human Interface Devices (HID), Mass Storage Class (MSC),
  - o Audio Device (ADC), Communication Device (CDC), & Composite Device





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

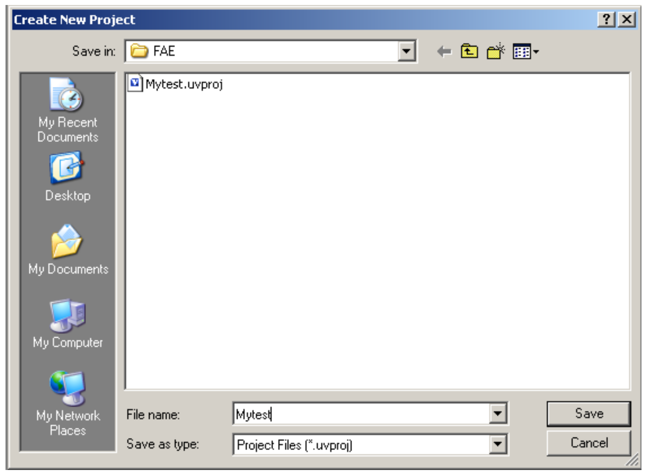
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will 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. This example uses the SAM3X processor – make appropriate changes to filenames and header files to match the device you are using.

Create a new project called Mytest:

1. With µVision running and not in debug mode, select Project/New µVision Project…
2. In the window Create New Project that opens, go to the folder C:\Keil\ARM\Boards\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:



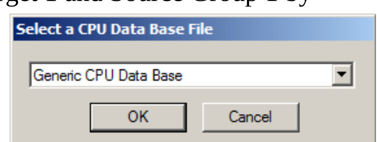
6. Click on Save.

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.
    µVision will configure itself to this device.

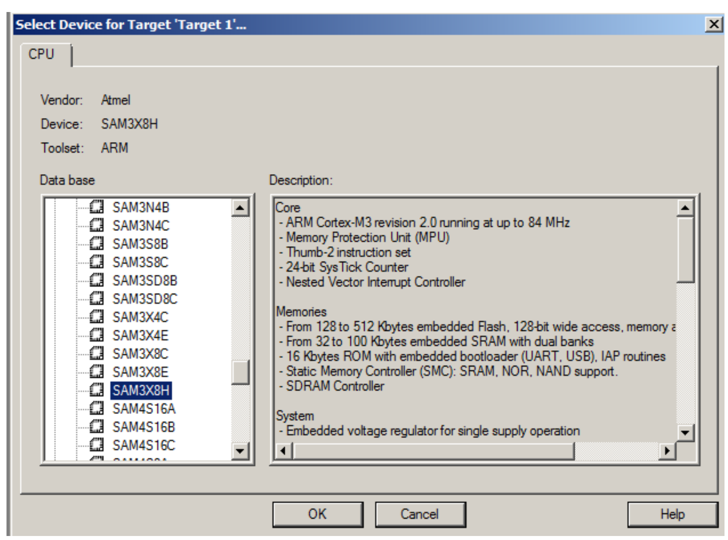
Select the startup file:

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 µVision, 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:

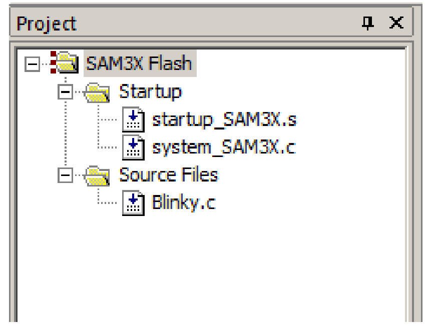
15. Click once on the name “Target 1” (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose SAM3X Flash. Press Enter to accept this change. Note the Target selector in the main µVision window also changes to SAM3X Flash.
16. Similarly, change Source 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 µVision, 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.

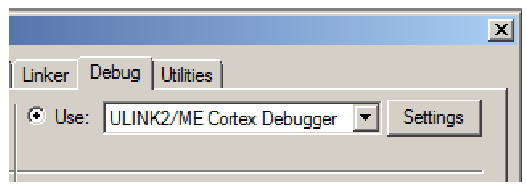


System File:

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 µVision 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.

24) 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), 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.

25) 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. CoreSight for the Cortex-M3: Search for **DDI0314F_coresight_component_trm.pdf** on www.arm.com.

26) Keil Products:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Professional™ (Includes Keil Middleware Libraries). (ULINK*pro* included for limited time) - $9,995
- MDK-Standard™ (unlimited compile limit) (ULINK2 included for a limited time) **-** $4,895
- MDK-Basic™ (256K Compiler Limit, No debug Limit) - $2,695
- MDK-Lite™ (Evaluation version) 32K Code and Data Limit - $0

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

Keil Middleware Libraries (formerly RL-ARM™) Available only as a component of MDK-Professional.

- Flash File, TCP/IP, CAN, USB driver libraries and a GUI library. http://www.keil.com/arm/mdk.asp
- Contact Keil Sales for support information for your ARM processor.
- See http://www.keil.com/rl-arm/rl_license.asp for licensing details.

USB-JTAG adapter (for Flash programming too)

- ULINK2 - $395 *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM
- ULINK*pro* - $1,395 – Cortex-M family SWV & ETM trace

Note: USA prices. Contact sales.intl@keil.com for international pricing.

Prices are for reference only and are subject to change without notice. Call Keil Sales for details on current pricing and product development.

All products are available.

All products include Technical Support for 1 year. This can be renewed.

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor.

ARM has an extensive university program. Call Keil Sales or the ARM University Program for special university pricing.

For the ARM University program: go to www.arm.com and search for university. Email: university@arm.com.

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

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

For more information:

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

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

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

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

Segger: www.segger.com.