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

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

Spring 2011 Version 4.0

by Robert Boys, bob.boys@arm.com

Introduction:

The purpose of this lab is to introduce you to the Atmel Cortex™-M3 processor using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) on the ATSAM3S processor. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. Keil MDK supports most Atmel ARM processors including ETM support. Check the Keil Device Database® on www.keil.com/dd for the complete list of Atmel support. A copy of the Device Database is included in MDK: in µVision select Project/Select Device for target…

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

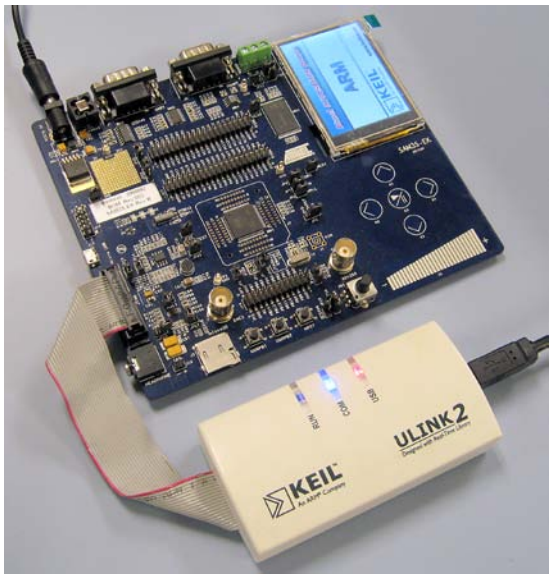
Serial Wire Viewer (SWV) allows real-time (no CPU cycles stolen) display of memory and variables, data reads and writes, exception events and program counter sampling plus some CPU event counters. ETM is currently available on certain Atmel ARM9™ processors. SWV is supported by the Keil ULINK™2, ULINK-ME, ULINK*pro*, Segger J-Link (Version 6 or later), Atmel SAM-ICE Version 6 or later and the new J-Link Ultra.

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. Keil also provides RL-ARM™. RL-ARM includes source files for the RTX™ RTOS, a TCP/IP stack, CAN drivers, a Flash file system and USB drivers.

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).
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.

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 Cortex-M3. SWV does not steal any CPU cycles and is completely non-intrusive. (except for the ITM Debug printf Viewer).

Serial Wire Viewer is accessible with a standard USB adapter such as the Keil ULINK series, Segger J-Link and Atmel SAM-ICE. No special equipment or software beyond the standard copy of MDK and the USB adapters mentioned above.

Atmel Evaluation Boards:

Keil provides examples for four SAM3 boards as listed below.

SAM3S-EK: This tutorial uses the SAM3S-EK Rev B board. You can use a Rev A board but the blue and green LEDs are on Port C. Rev B has the blue and green LEDs on Port A. You can easily change the port in the software. USB and LCD examples are provided. A pre-compiled example is provided for LCD_Blinky. It will be deleted if you attempt to build it with MDK-Lite. It is a good idea to save the file Blinky.axf. You can load, run and debug this program with MDK-Lite.

SAM3U-EK: You can use this document with the SAM3U-EK board and others below. You might need to make certain allowances with the slightly different hardware. The LEDs connect to Port B. USB and LCD examples are provided.

SAM3N-EK: Keil provides Blinky and RTX_Blinky examples.

SAM3X-EK: Keil provides Blinky and RTX_Blinky examples.

Software Installation:

This document was written for Keil MDK 4.20 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.

To obtain a copy of MDK go to www.keil.com/arm and select "Download MDK-Lite" from the left column.

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-M3 ETM trace support. ETM is not currently implemented on the SAM3 processor family. The ULINK*pro* can be used with the ATSAM3 for SWV support and it provides the fastest Flash programming speed available.

Index:

1) Using Various USB adapters: J-Link, SAM-ICE, Keil ULINKs

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) Segger J-Link and SAM-ICE: (SAM-ICE is pictured two pages down)

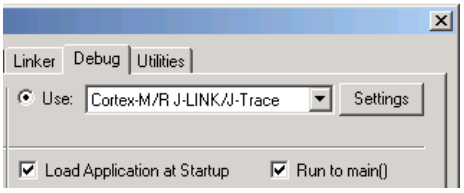
Serial Wire Viewer is supported by hardware Version 6 or later only. The main difference between V 6 through 8 is apparently speed so the later the version the better. The version number is printed on the Segger adapters.

Due to the high speed requirements of SWV, the Keil ULINK family currently performs the best. Segger products at this time have some challenges providing data fast enough for the Logic Analyzer window. I have not yet tested the J-Link Ultra which reportedly solves these issues. You can exercise the experiments in this document but the Logic Analyzer has some distortion. 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*. All other display windows are the same as the ULINK2.

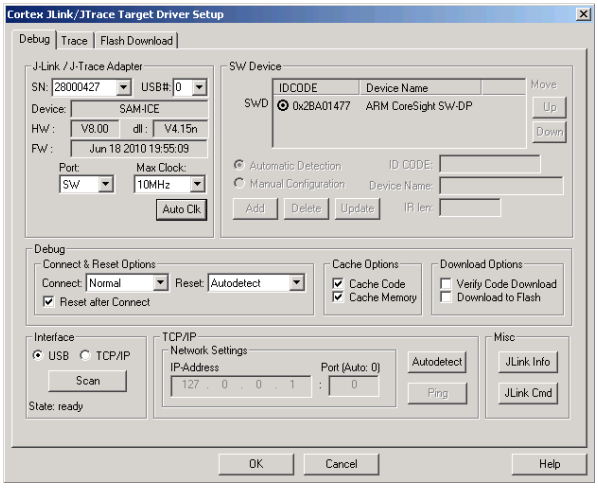
USB Drivers: The Segger USB drivers are located in C:\Keil\ARM\Segger\USBDriver.

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:

Select the debug connection to the target:



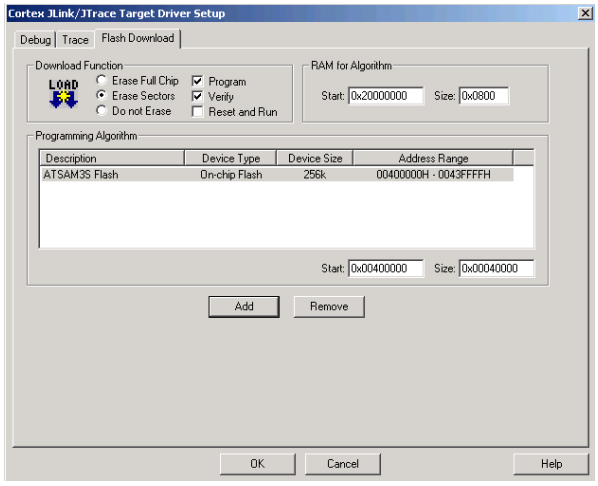
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select the J-Link as shown here:
3. Select Settings and the next window below opens up. This is the control panel for the J-Link.



4. In Port: select SW. SWV will not work with JTAG.
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 similar to Step 2 above.
9. Click Settings to select the programming algorithm.
10. Select Add and select ATSAM3S Flash if necessary:
11. If using a SAM3U device select ATSAM3U Flash.
12. Confirm Start = 0x2000_0000 and not 0x0020_0000
13. Click on OK once.

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

14. Click on OK to return to the µVision main screen.

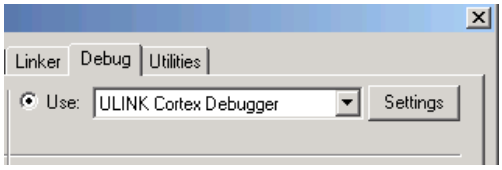
TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this on pages 6 and 10.

2) Keil ULINK2 or ULINK-ME:

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-M3 ETM trace adapter, can be used like a ULINK2 or ULINK-ME. The ETM trace will not display as current SAM3 processors do not have ETM trace capabilities.

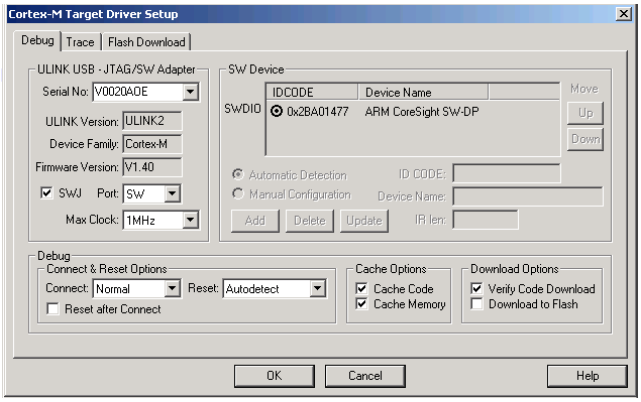
1. Assume the ULINK2 is connected to a powered up Atmel target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project.

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.

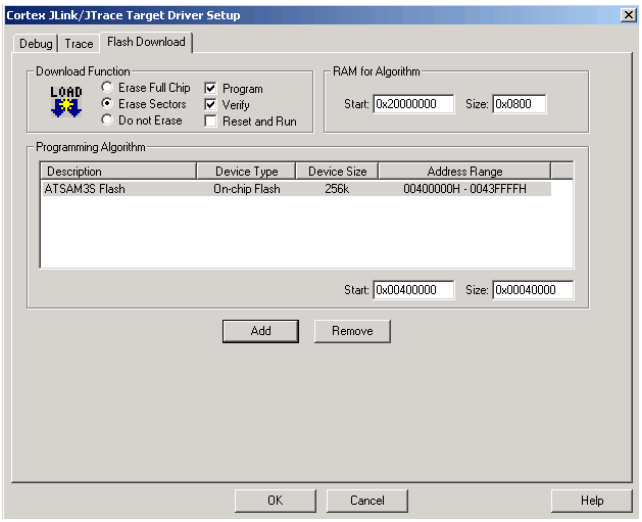


TIP: You can do regular debugging using JTAG. SWD and JTAG operate at approximately the same speed.

Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ULINK similar to Step 2 above.
8. Click Settings to select the programming algorithm.
9. Select Add and select ATSAM3S Flash as shown:
10. If using a SAM3U select ATSAM3U Flash.
11. **Confirm**: Start = 0x2000_0000 and not 0x0020_0000
12. Click on OK once.

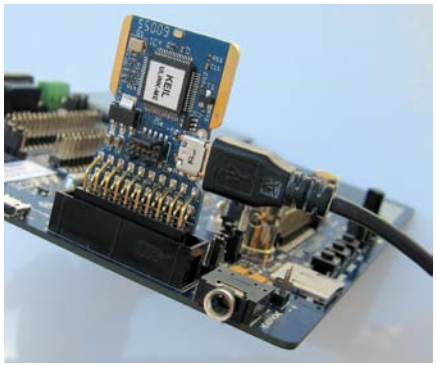
TIP: To program the Flash every time you enter Debug mode, check Update target before Debugging.



Click on OK to return to the µVision main screen.

13. 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

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

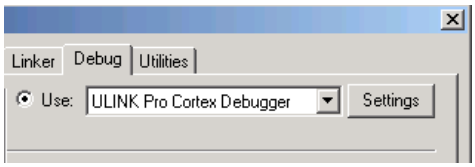
TIP: If you select ULINK or ULINK*pro*, and have the opposite ULINK actually connected to your PC; the error message will say "No ULINK device found". This message actually means that µVision found the wrong Keil adapter connected.

3) Keil ULINK*pro*:

1. In the Options for target in the Debug tab, select ULINK Pro Cortex Debugger as shown below.
2. In Settings, it is configured the same as a ULINK.
3. Select the Utilities tab and select the ULINK*pro* as done before and select the programming algorithm.
4. Confirm Start = 0x2000_0000 and not 0x0020_0000. Do not include the underscore.

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. Keil uses a Signum Systems JtagJetTrace for ARM9 ETM support.

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

Keil ULINK*pro*

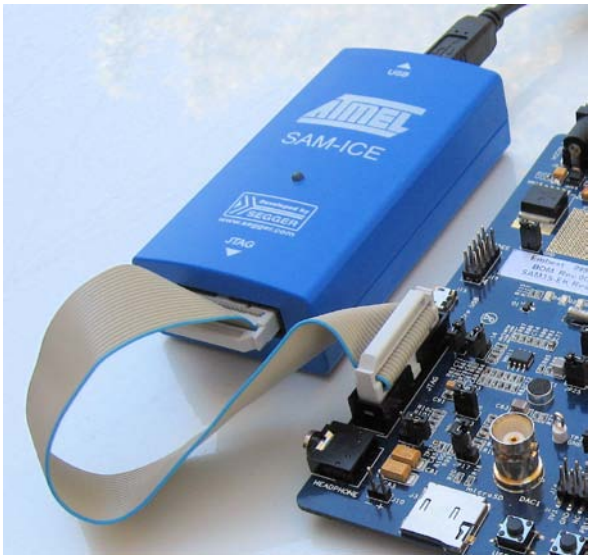
An ETM trace adapter that can be used as a ULINK2. It has very fast Flash programming and an enhanced Instruction Trace window that connects the trace frames to your source code.

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.

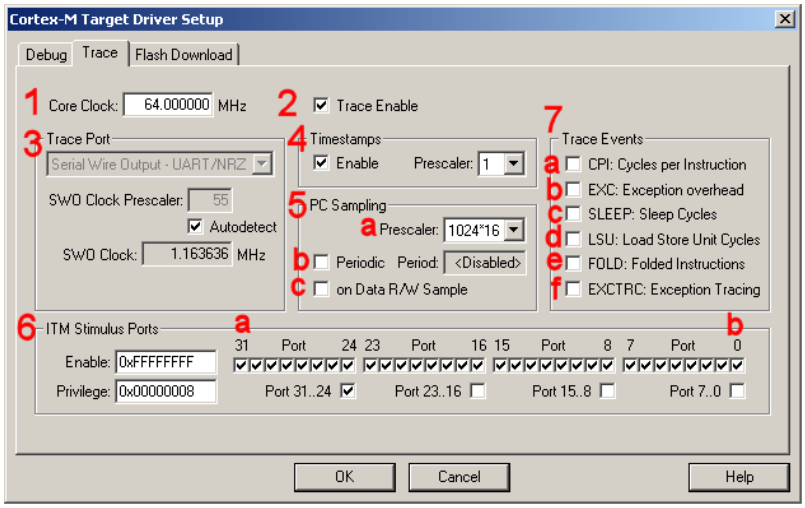


2) Serial Wire Viewer (SWV) Configuration with ULINK2 or ULINK-ME:

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 the 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 or you can add an external USB PCMCIA card to a laptop.

3) *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.

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**BlinkyB**\Blinky.uvproj.
4. Make sure you selected Blinky**B** and not Blinky. Make sure "SAM3S Flash" is selected. 

This is where you create and select different target configurations such as to execute a program in RAM or Flash.
5. Configure your USB-JTAG adapter at this point if you are not using a ULINK2. ULINK2 is selected by default. See pages 3 through 5. **Make sure SW is selected and not JTAG in the Port: box.** This is an important step.
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 LEDs on the SAM3S-EK will now blink.

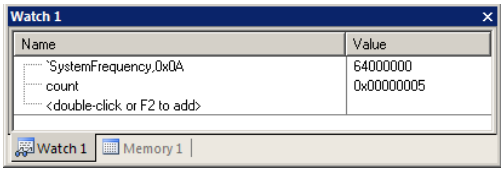
Now you know how to compile a program, load it into the ATSAM3S processor Flash, run it and stop it.

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

The Watch and memory windows will display updated variable values in real-time. It does this through the ARM CoreSight debugging technology that is part of 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" variables into windows or enter them manually.

Watch window:

1. Stay in Debug mode. You can do the following steps while the CPU is running. Click on RUN if needed.
2. In the file Blinky.c locate the global variable **count**. It will be near line 15.
3. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
4. In Blinky.c, block **count**, click and hold and drag it into Watch 1. Release it and **count** will be displayed updating as shown here:



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

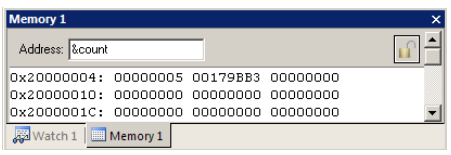
5. You can also enter a variable manually by double-clicking or pressing F2 and using copy and paste or typing the variable.

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 **count** in the Watch window. Enter the value 0 and press Enter. 0 will be inserted into memory in real-time. You can do this in the Memory window with a right-click and select Modify Memory.

Memory window:

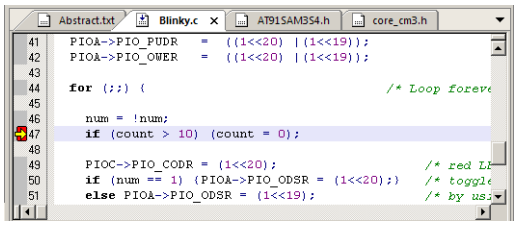
1. Drag 'n Drop **count** into the Memory 1 window or enter it manually. Select View/Memory Windows if necessary.
2. Note the value of **count** is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to but this not what we want to see at this time.
3. Add an ampersand "&" in front of the variable name and press Enter. The physical address is shown (0x2000_0004).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **count** is displayed as shown here:
6. Both the Watch and Memory windows are updated in real-time.



TIP: No CPU cycles are used to perform these operations. See the next page.

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

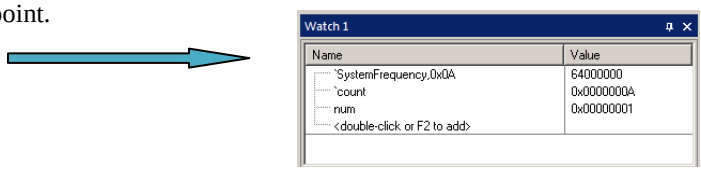
1. Make sure Blinky.c is running.
2. Locate where the local variable **num** is declared in Blinky.c near line 29, at the start of the main function.
3. Drag and Drop **num** into Watch 1 window. Note you are unable to accomplish this.
4. Double-click in the left margin, somewhere past the for (;;) loop in main to set a hardware breakpoint such as at line 47 shown here:



5. The program will stop and you can now enter **num** into Watch 1. Remove the breakpoint by double-clicking on it to remove the red block.

TIP: You can set hardware breakpoints on-the-fly in the Cortex-M3 !

6. Run the program and note **num** value displays as ?????????.
7. Stop the program and it will probably stop in the Delay function. **num** now displays as <out of scope> or ????????.
8. These effects are because µVision is unable to determine the value of **num** when the program is running because it exists only when main is running. It disappears in functions and handlers outside of main.
9. Start the program by clicking on the Run icon.
10. **num** changes to ????????. Set a breakpoint by double-clicking in the margin for (;;;) loop as before in Step 4 above. The program will soon stop on this hardware breakpoint.
11. **num** correctly displays the value as shown here:



12. Each time you click RUN, these values are updated.

How to view local variables updated in real-time:

All you need to do is to make **num** static !

1. In the declaration for **num** add **static** like this:

```
int main (void) {
        static int num = 1;
```

2. Exit debug mode. **TIP:** You can edit files in edit or debug mode, but can compile them only in edit mode.
3. Compile the source files by clicking on the Rebuild icon. Hopefully they compile with no errors or warnings.
4. To program the Flash, click on the Load icon. 

. A progress bar will be displayed at the bottom left.

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

5. Enter Debug mode.  You will have to re-enter **num** in the Watch 1 window because it isn't the same variable anymore – it is a static variable now instead of a local. Drag 'n Drop is the fastest way.
6. Remove the breakpoint you previously set and click on RUN. You can use Debug/Kill All Breakpoints to do this.
7. **num** is now updated in real-time. This is ARM CoreSight technology working.
8. 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:

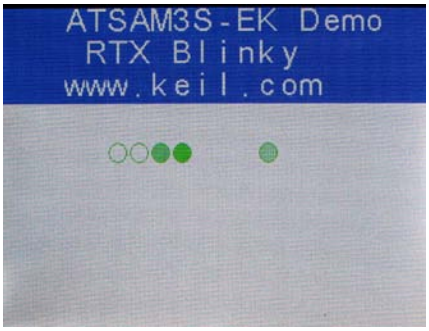
µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. Remember the Cortex-M3 is a Harvard architecture. This means it has separate instruction and data buses. While 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.

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

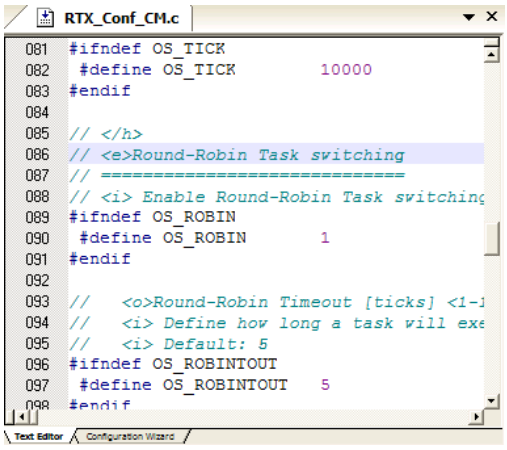
Keil provides RTX, a full feature RTOS. RTX is included as part of the Keil MDK-Professional and MDK-Standard. It can have up to 255 tasks and no royalty payments are required. If source code is required, this is included in the Keil RL-ARM™ Real-Time Library which also includes USB, CAN, TCP/IP networking and a Flash File system. This example explores the RTOS project. Keil will work with any RTOS. An RTOS is just a set of C functions that gets compiled with your project.

1. Start µVision by clicking on its icon on your Desktop if it is not already running.
2. Select Project/Open Project.
3. Open the file C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\RTX_Blinky\Blinky.uvproj.
4. 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.
5. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
6. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
7. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
8. The LCD display will indicate the four waveforms of a stepper motor driver changing. Click on STOP .

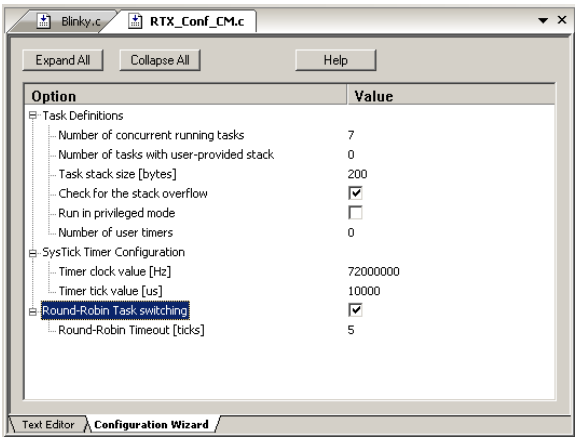


The Configuration Wizard for RTX:

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 Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. This is a great feature as it is much easier changing items here than in the source code.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>.
8. The new µVision4 System Viewer windows are created in a similar fashion. Available Q4 2010.



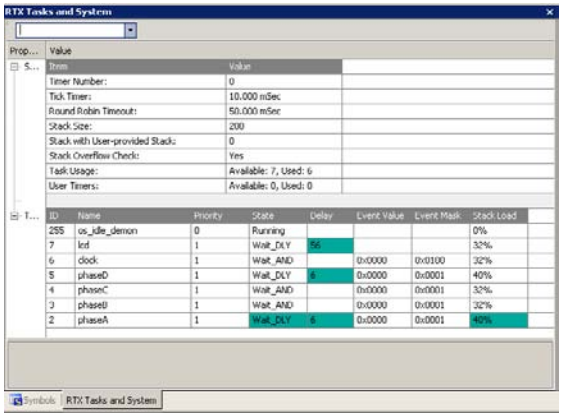
Text Editor: Source Code



Configuration Wizard

6) RTX Kernel Awareness using Serial Wire Viewer

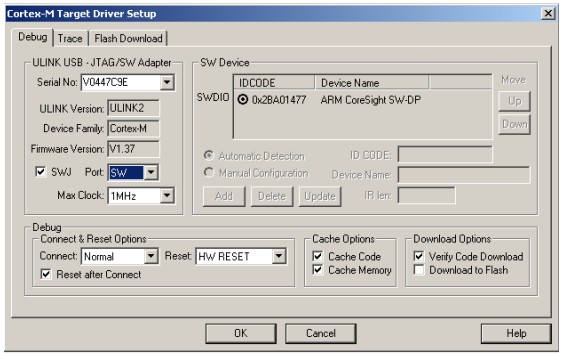
Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides a Task Aware window for RTX. Other RTOS companies also provide awareness plug-ins for µVision.



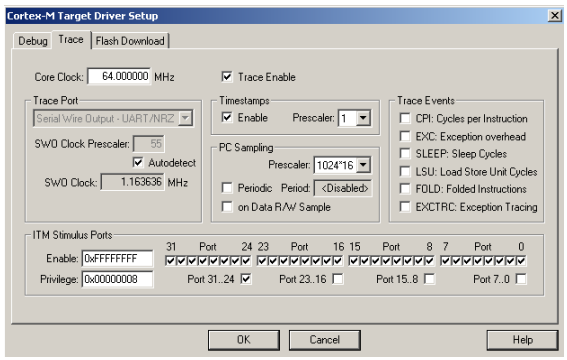
1. Run RTX_Blinky again by clicking on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. Note these values are updated in real-time using the same technology as used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

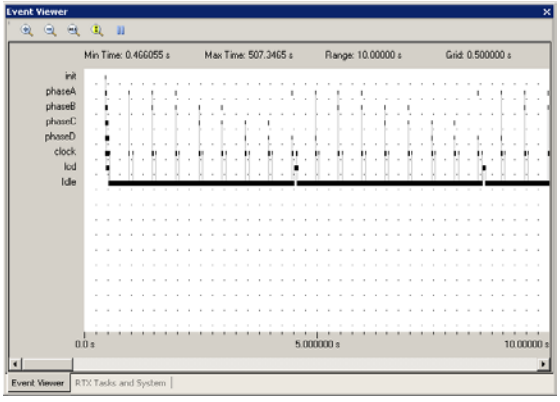
RTX Viewer: Configuring Serial Wire Viewer (SWV):

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 5 seconds by clicking on the ALL and then the + and – 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(near line 313) 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.

7) 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.
2. Add 4 global variables **`unsigned int phasea`** through **`unsigned int phased`** to Blinky.c as shown here:

```
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  *----------------------------
```

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.

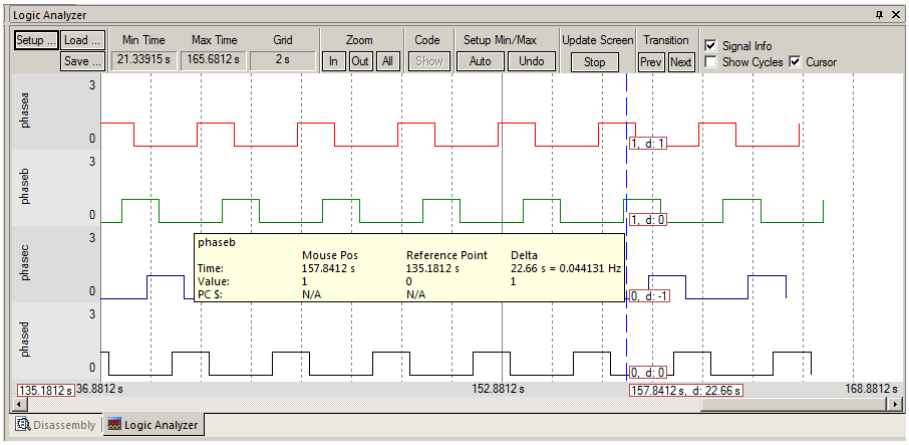
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

```
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. 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: The J-Link and SAM-ICE are challenged by the high SWO data rate. Some blocks are slightly shorter than others.

8) 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**).**

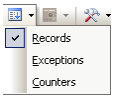
Currently, March 2011, the display of data read, write and ITM trace frames are not implemented in the J-Link or SAM-ICE.

You have configured Serial Wire Viewer (SWV) in Section 6 under **RTX Viewer: Configuring the Serial Wire Viewer:**

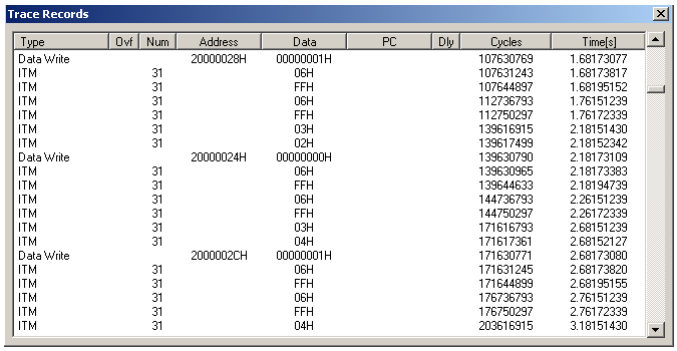
Now we will examine some of the features available to you. SWV works with µVision and a ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

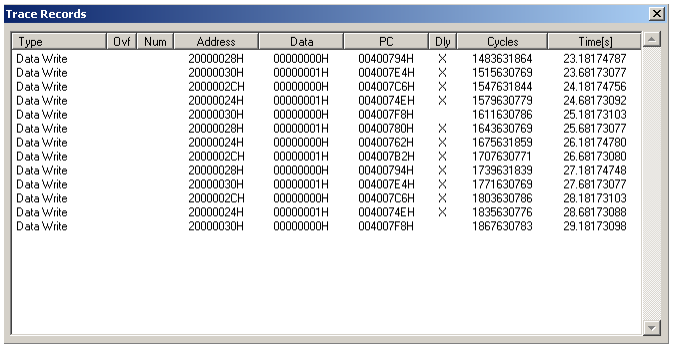
Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. A user program runs at full speed and needs no code stubs or instrumentation software added to your programs.

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



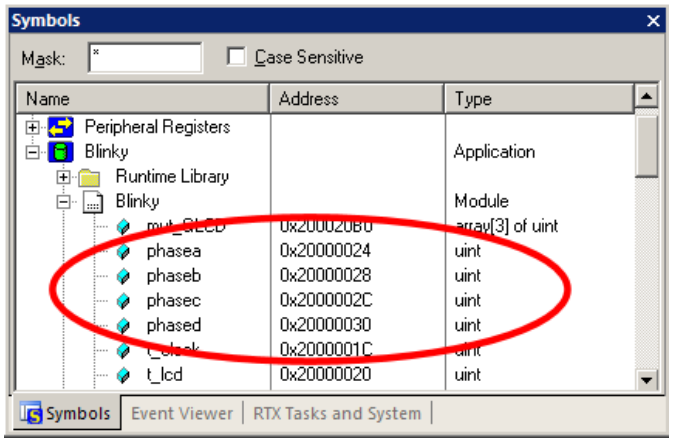
3. The Trace Records window will open up as shown here:
4. The ITM frames are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. To turn this off select Debug/Debug Settings and click on the Trace tab. Unselect ITM Stimulus Port 31. **TIP:** Port 0 is used for Debug **`printf`** Viewer.
5. Unselect EXCTRC and Periodic.
6. Select On Data R/W Sample.
7. Click on OK 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.





What is happening here ?

1. When variables are entered in the Logic Analyzer (remember phasea through phased ?), the reads and/or writes will appear in Trace Records.
2. The Address column shows where the four variables are located.
3. The Data column 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.

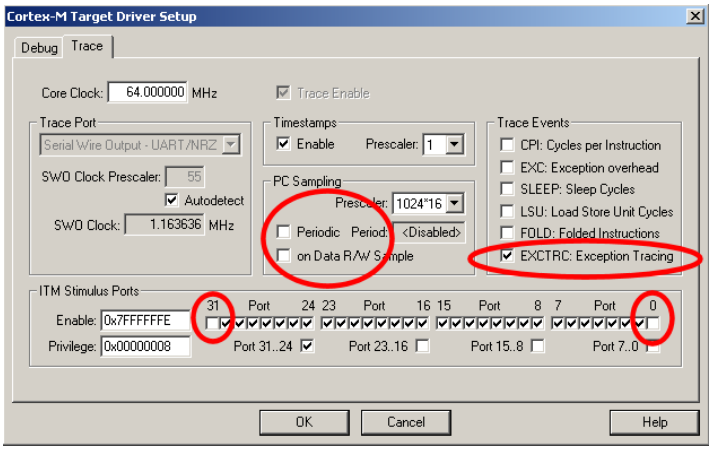
Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

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

2) Exceptions and Interrupts:

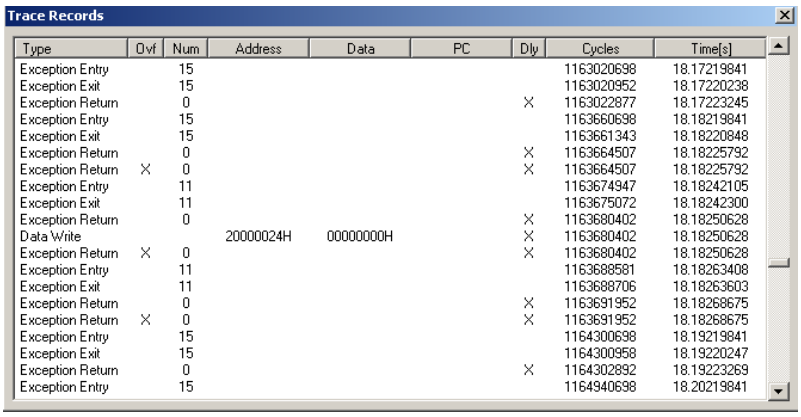
The ATSAM3 family using the Cortex-M3 processor has many interrupts and it can be difficult to determine when they are being activated. SWV on the ATSAM3 family makes this easy.

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample and ITM Ports 31 and 0. (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.



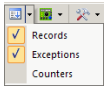
What Is Happening ?

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

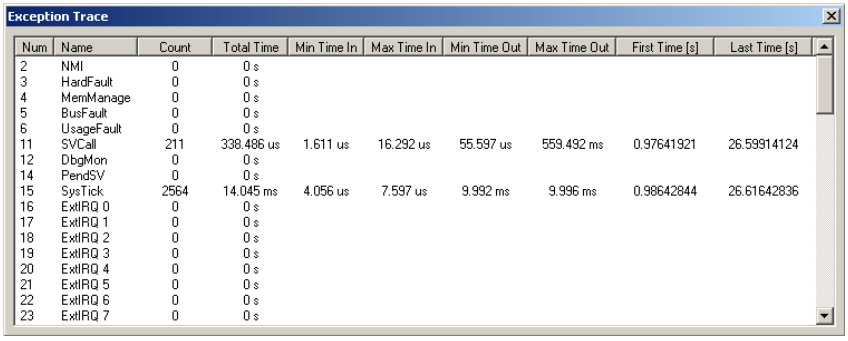


TIP: The SWO pin is one pin on the Cortex-M3 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate.

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 peripheral interrupts.
5. You can clear this trace window by double-clicking on it.



6. All this information is displayed in real-time and without stealing CPU cycles !

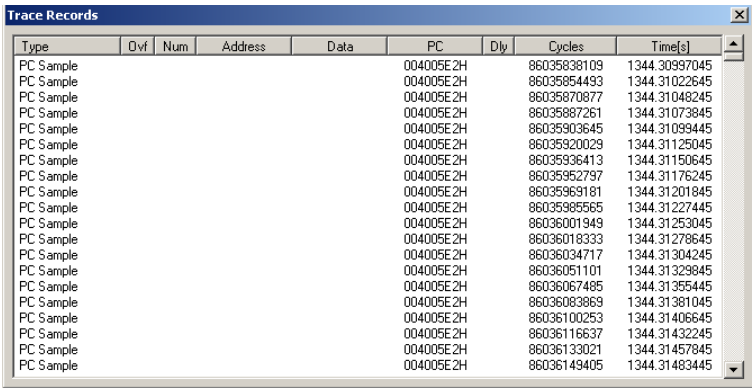
TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

3) PC Samples:

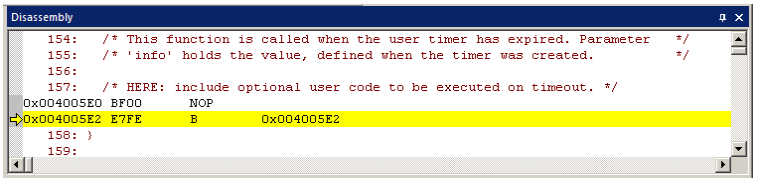
Serial Wire Viewer can display a sampling of the program counter.

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.

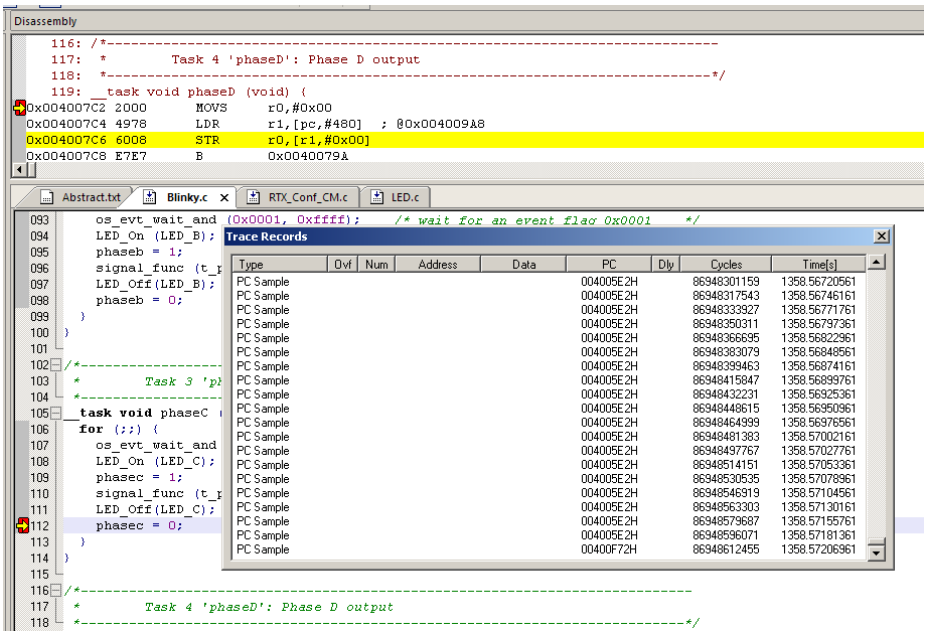
1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear it.
5. Click on RUN and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



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.



9) 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.

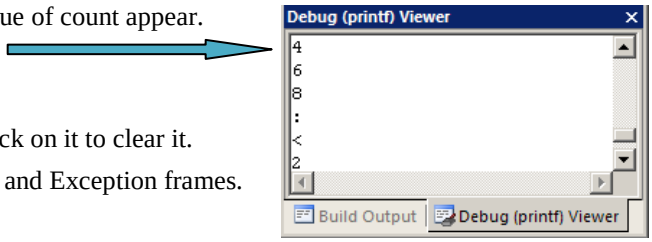
1. Exit Debug mode and open the project BlinkyB/Blinky.uvproj (not RTX_Blinky).
2. Configure the Serial Wire Viewer as described in Steps 1 through 9 on page 10. Leave EXCTRC selected.
3. Add this code to Blinky.c. A good place is near line 16, just after the declaration of **count**.

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

4. In the main function in Blinky.c right after the second Delay(500) enter these lines after near line 61:

```
ITM_Port8(0) = count + 0x30;     /*  displays count value: +0x30 converts to 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.

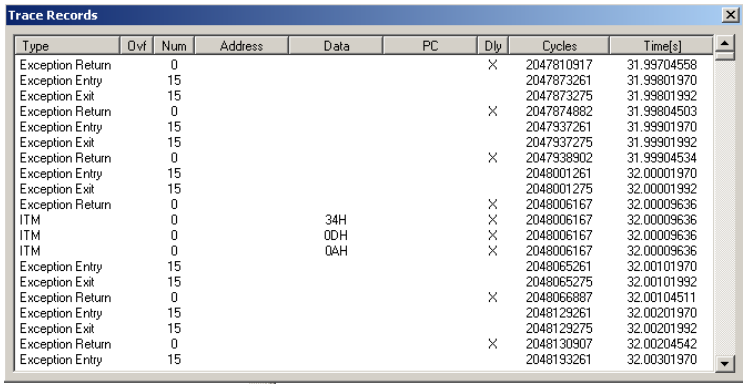


Trace Records

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.



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.

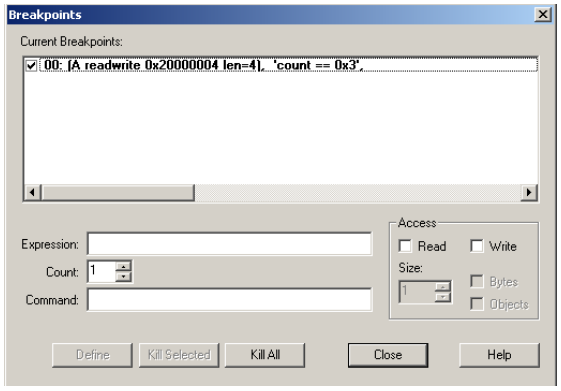
TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enter only those features that you really need.

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

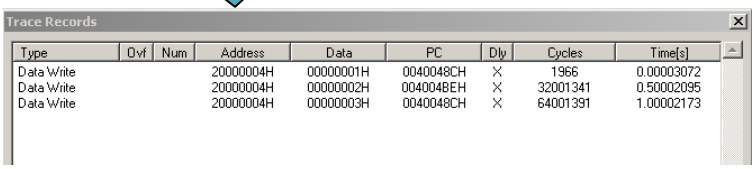
10) Watchpoints: *Conditional Breakpoints*

ATSAM3 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. Often µVision will take one and perhaps two or more breakpoints for its operations. The ATSAM3S also has four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses watchpoints in its operations. This means in µVision you must have two variables free in the Logic Analyzer to use Watchpoints.

1. Open the project Blinky that you used before if not already open. Be in Edit mode and not Debug.
2. We will use the global variable **`count`** 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 **`count`** to the Logic Analyzer. Click on Setup and set the Display Range to Min: 0x0 and Max: 0x10.
6. Click on Close to return. Set Range 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: "**`count`** == 3" 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 **`count`** 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" and uncheck EXTRC and ITM 31.
13. Click on OK. Open the Trace Records window. Click on RUN. J-Link or SAM-ICE do not have data tracing enabled..
14. When **`count`** equals 3, the Watchpoint will stop the program.
15. You will see **`count`** incremented in the Logic Analyzer as well as in the Watch window.
16. Note the three data writes in the Trace Records window shown below. 0, 1, 2 and 3 in the Data column. Plus the address the data written to and the PC of the write instruction. This is with a ULINK2 or ULINK-ME.



17. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.
18. To repeat this exercise, click on RUN several times to get past the stored value (3) of **`count`**.
19. When finished leave Debug mode.

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

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

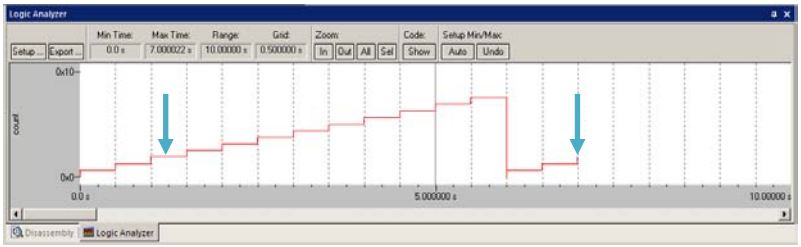
TIP: The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it.

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 **`count`**. The program was run twice using STOP and RUN.

The cyan arrows point to the two times Watchpoint trigger **`count == 3`**.

You should have no trouble configuring the Logic Analyzer to display **`count`**.



Note: The Logic Analyzer does not work very well with J-Link or SAM-ICE at this time. We are working on this.

11) USB HID (Human Interface Device):

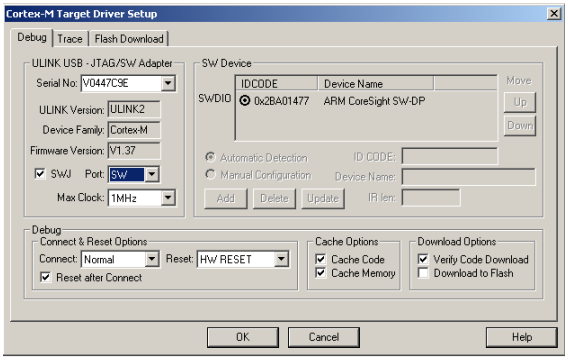
This USB example provided with MDK demonstrates two way communication over USB using the HID interface. A HID client program is provided to run on your PC and communicate with the SAM3S-EK. Pressing the USRPB1 and USRPB2 will activates corresponding boxes in the HID Client window. Selecting output boxes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

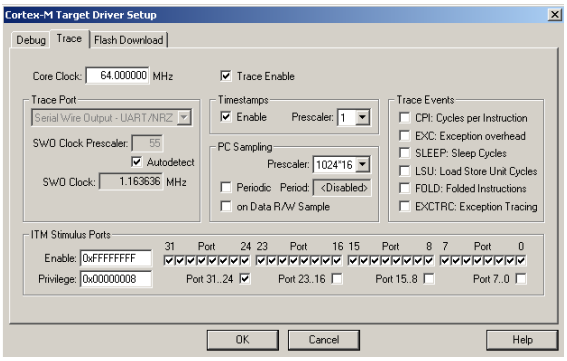
Configure the Serial Wire Viewer (SWV):

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.

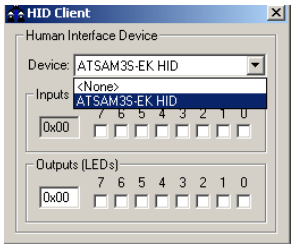


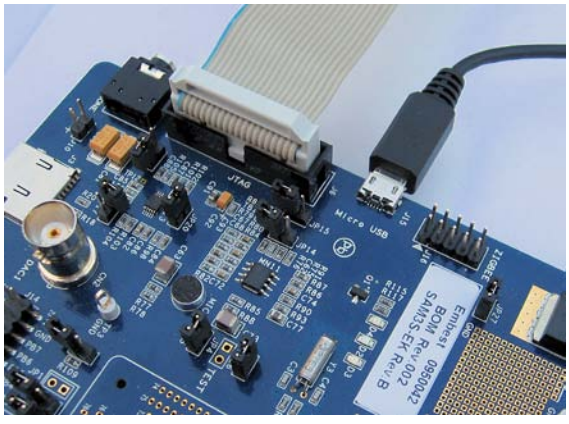
Connect the USB:

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 Client as shown below. Select this device. The boxes will not be grayed out which would mean loss of the USB connection.
13. As you select Outputs 0, 1 or 2, the LEDs will toggle. RED is normally ON, the others OFF.
14. Pressing USRPB1 and USRPB2 (or both) will cause an effect in the Inputs (Buttons) boxes. Try not to press the RESET button !
15. If you stop the CPU, the USB connection will stop. To reactivate the HID Client, reselect the Device option.



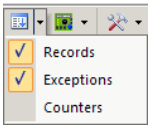


16. Turn the page for more details of this USB exercise including using the Serial Wire Viewer.

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.
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 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.

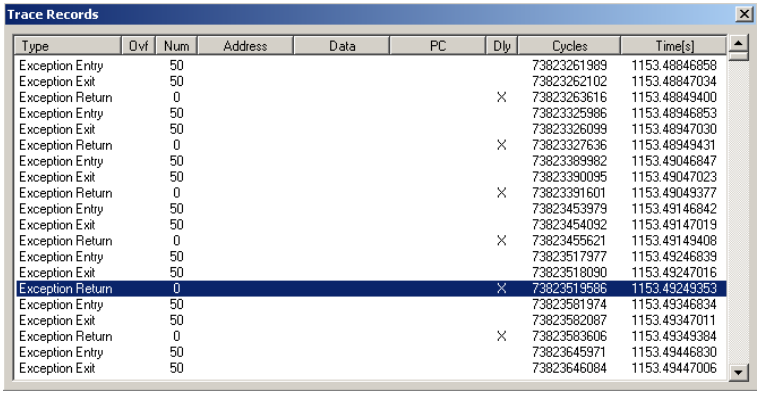
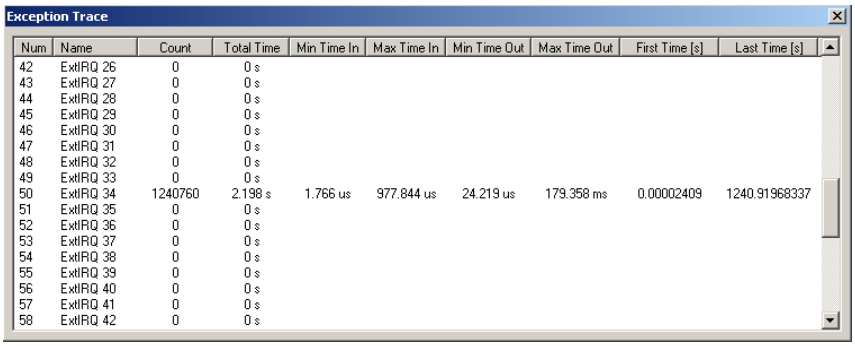


Table 37-3. Peripheral IDs

Instance	ID
UDP	34



RL-ARM: Complete USB support is in Keil RL-ARM. Its components include the source code for RTX (RTX itself is included with MDK), USB, a Flash file system, Ethernet and CAN drivers. Not all components of RL-ARM are applicable to all Atmel processors. No royalty payments or per-project fees are required with RL-ARM.

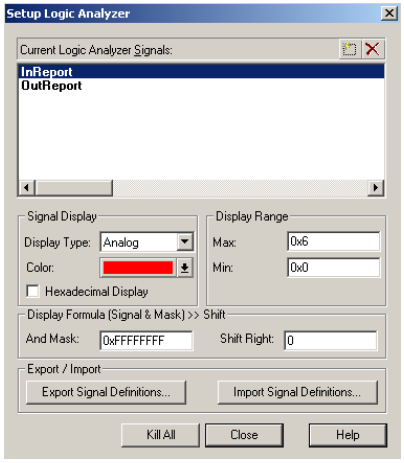
Viewing Global Variables as they change using Serial Wire Viewer:

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

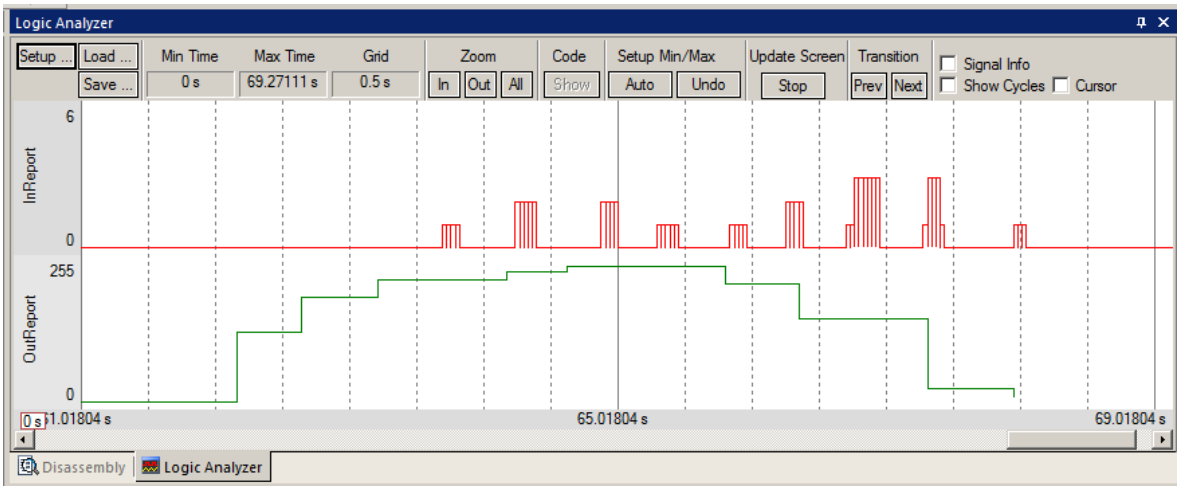
InReport – two bits contain the button status.

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

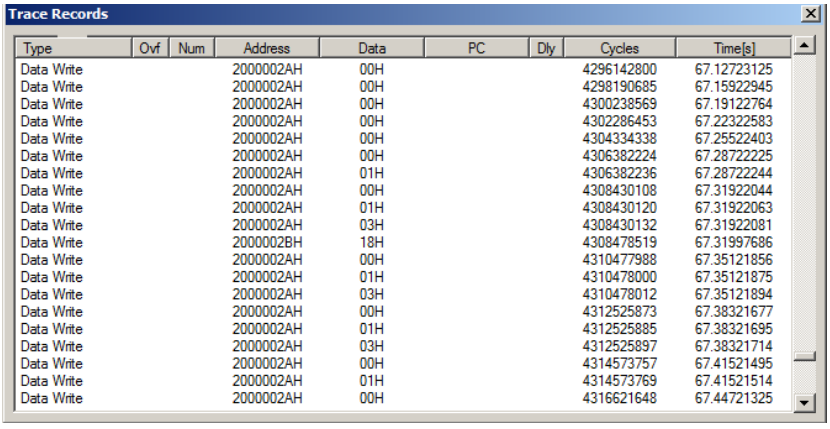
1. In Debug mode: Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC and Periodic if set. Select Sample "on R/W Sample". Click on OK to close this window.
3. Close the Exception Trace window and click on RUN to start the program.
4. Open the Logic Analyzer window and enter both the 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. Note: J-Link and SAM-ICE have problems with InReport.
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.

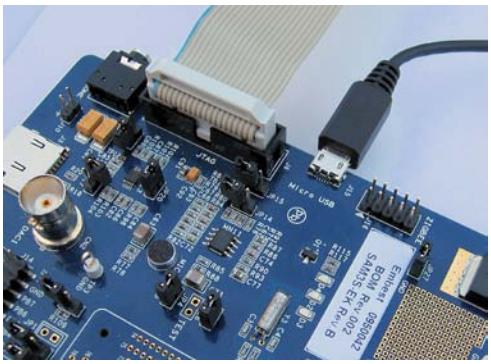


TIP: If you enable EXCTRC, you can have the USB interrupts display. You must be aware that enabling too many SWV features can lead to missing frames.

12) USB Memory:

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

1. Open the project C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\USBMem\Memory.uvproj.
2. Configure the USB JTAG adapter you are using if not the ULINK2 or ULINK-ME.
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.
5. Connect a USB cable to the Micro USB connector on the SAM3S-EK board to your PC as shown here:
6. Enter Debug mode.  Click on Run  to start the program.
7. A Windows Explorer window similar to the one below will open up. It will show a file readme.txt in the USB stick.
8. **Note:** You might have to unplug and reinsert the USB cable.
9. Right click in the area underneath README.TXT and select Properties. The Properties window will open up.
10. You can move files in and out of the USB stick as long as you stay within the memory limits.
11. This could be made larger with external RAM.



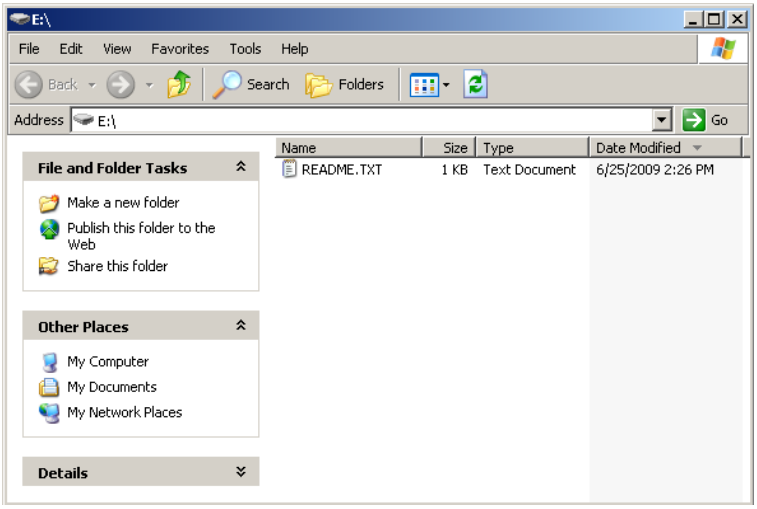
For information on Keil RL-ARM Real-Time Library please visit:

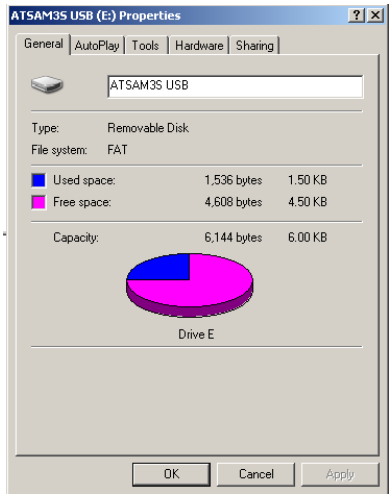
www.keil.com/arm/rl-arm/

Remember that Keil MDK includes a full version of RTX.

RL-USB Features: RL-USB is a component of RL-ARM 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
  - Low Speed (1.5Mbit/s), Full Speed (12Mbit/s), & High-Speed (480Mbit/s)
- Common USB Device Class support
  - Human Interface Devices (HID), Mass Storage Class (MSC),
  - Audio Device (ADC), Communication Device (CDC), & Composite Device



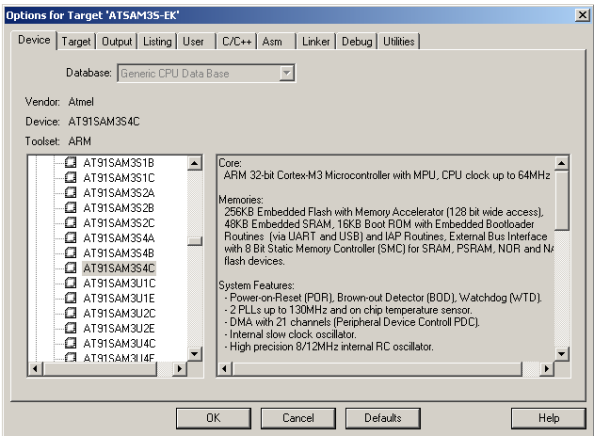


13) Creating a new project: Using the Blinky source files:

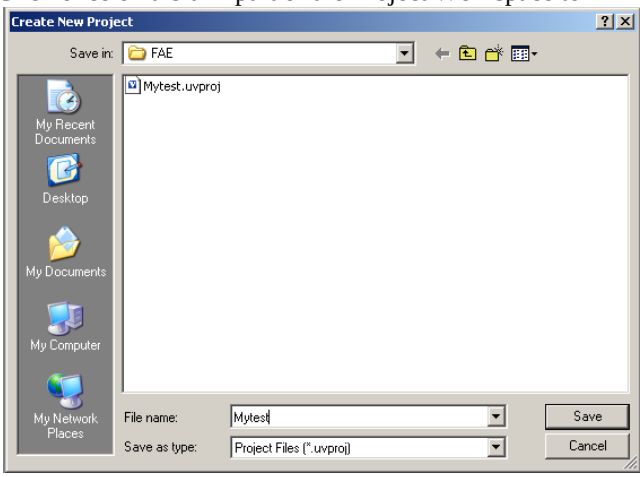
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run the Blinky example as usual. You can use this process to create any new project from your own source files created with µVision's editor or any other editor.

Create a new project called Mytest:

1. With µVision running and not in debug mode, select Project/New µVision Project.
2. In the window Create New Project go to the folder C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK.
3. Right click and create a new folder by selecting New/Folder. I named this new folder FAE.
4. Double-click on the newly created folder "FAE" to enter this folder as is shown below.
5. Name your project. I called mine Mytest. You can choose your own name but you will have to keep track of it.
6. Click on Save.



7. "Select Device for Target 1" shown here opens up.
8. This is the Keil Device Database® which lists all the devices Keil supports (plus some secret ones).
9. Locate the Atmel directory, open it and select ATSAM34C. Note the device features are displayed.
10. Click on OK.
11. A window opens up asking if you want to insert the default ATSAM3S startup file to your project. Click on "Yes". This will save you a great deal of time.
12. In the Project Workspace in the upper left hand of µVision, open up the folders by clicking on the "+" beside each folder.
13. We have now created a project called Mytest and the target hardware called Target 1 with one source file startup_SAM3S.s.
14. Click once (carefully) on the name "Target 1" (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose ATSAM3S. Click once on a blank part of the Project Workspace to accept this. Note the Target selector also changes to ATSAM3S. You can create many target hardware configurations including a simulator and easily select them.



Select the source files:

1. Using MS Explore (right click on Windows Start icon), copy blinky.c and system_ATSAM3S.c from C:\Keil\ARM\Boards\Atmel\ATSAM3S-EK\Blinky to the \Atmel\ATSAM3S-EK\FAE folder.
2. In the Project Workspace in the upper left hand of µVision, right-click on "ATSAM3S" and select "Add Group". Name this new group "Source Files" and press Enter.
3. Right-click on "Source Files" and select **Add files to Group "Source Files"**.
4. Select the file Blinky.c and system_ ATSAM3S.c and click on Add (once!) and then Close. These will show up in the Project Workspace when you click on the + beside Source Files..
5. Select Options For Target and select the Debug tab. Make sure ULINK Cortex Debugger is selected. Select this by checking the circle just to the left of the word "Use:".
6. At this point you could build this project and run it on your SAM3S-EK board.

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

Contact Keil Technical Support for a similar tutorial on configuring a RTX project from scratch.

14) 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.

For complete information on CoreSight for the Cortex-M3: Search for **DDI0314F_coresight_component_trm.pdf** on www.arm.com.

15) Keil Products:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Professional™ (Includes RL-ARM) Promotion with ULINK*pro* until September 30, 2011 - $9,995
- MDK-Standard™ (with included RTX RTOS) (MDK has a great simulator) - $4,895
- MDK-Basic™ (256K Compiler Limit, No debug Limit) No included RTX - $2,695
- MDK-Lite™ (Evaluation version) 32K Code and Data Limit - $0

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.

Prices are for reference only and are subject to change without notice.

Call Keil Sales for more details on current pricing. All products are available.

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

Keil Real Time Library (RL-ARM™)

Available only as a component of MDK-Professional.

- RTX sources, Flash File, TCP/IP, CAN, USB driver libraries.
- Contact Keil Sales for individual TCP/IP pricing.

USB-JTAG adapter (for Flash programming too)

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

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

ARM has an extensive university program. Call Keil Sales for special university pricing.

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

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

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

MDK supports Atmel ARM7™ and SAM9 processors. Check www.keil.com/dd for a complete list of device supported.

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.

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

KEIL™
Tools by ARM