Microchip Atmel SMART | SAM V7: Cortex-M7
SAMV7 Xplained ULTRA evaluation board
ARM Keil MDK 5 Toolkit Summer 2018 V 2.0

bob.boys@arm.com

Introduction:

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_274.asp

The purpose of this lab is to introduce you to the Microchip Cortex®-M7 processor using the ARM® Keil® MDK toolkit featuring the IDE µVision®. We will demonstrate all debugging features available on this processer including Serial Wire Viewer and ETM instruction trace. You will be able to confidently work with these processors and Keil MDK.
We recommend you obtain the ***new*** **Getting Started MDK 5:** from here: www.keil.com/gsg/.

Keil Microchip Information Page: See www.keil.com/Microchip.

Keil MDK supports and has examples for most Microchip ARM processors and boards. Check the Keil Device Database® on www.keil.com/dd2 for the complete list. Additional information is listed in www.keil.com/Microchip/.

Linux: Microchip ARM processors running Linux and Android are supported by ARM DS-5™. http://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 a commercial version. Contact Keil Sales for details.

Microchip 8051 Processors: Keil has tools for many Microchip 8051 processors. See www.keil.com/Microchip/

Microchip | Start: µVision is compatible with the Microchip | START configuration program.

RTX RTOS: All variants of MDK contain the full version of RTX RTOS. RTX has a BSD or Apache 2.0 license and source code is provided. µVision has two kernel awareness windows that update while the program is running. www.keil.com/rtx.

Many other RTOSs are compatible with MDK as are applications with no OS.

Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is turn-key "out-of-the-box".
2. ARM Compiler 5 and ARM Compiler 6 (LLVM). GCC is supported.
3. **Compiler Safety Certification Kit:** www.keil.com/safety/
4. **TÜV** certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
5. Dynamic Syntax checking on C/C++ source lines.
6. MISRA C/C++ support using PC-Lint. www.gimpel.com
7. **Keil Middleware:** Network, USB, Flash File and Graphics.
8. ***NEW!*** **Event Recorder for Keil Middleware, RTX and User programs.**
9. ***NEW!*** **Power Measurement and Testing** with Keil ULINK*plus*.
10. CMSIS-RTOS RTX: RTX has a BSD or Apache 2.0 license with source code. www.keil.com/RTX and https://github.com/ARM-software/CMSIS_5 FreeRTOS CMSIS-RTOS is supported.
11. CoreSight™ Serial Wire Viewer (SWV). ETM instruction trace capability on appropriately equipped Microchip processors. ETM provides Instruction Debugging, Code Coverage and Performance Analysis.
12. Affordable perpetual and term licensing with support. Contact Keil sales for pricing options. Inside-Sales@arm.com
13. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.

This document includes details on these features plus more:

1. Real-time Read and Write to memory locations for the Watch, Memory and peripheral windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
2. Six Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
3. RTX and RTX Tasks windows: Two kernel awareness windows that update while your program is running.
4. A DSP example program using ARM CMSIS-DSP libraries and RTX.
5. ETM Instruction Trace including Performance Analyzer and Code Coverage.
6. How to create your own µVision projects with and without RTX and a list of document resources available.

General Information:

Software Packs:

Blinky Example and Debugging Features:

DSP Sine Example:

Keil Middleware:

Power Measurement with Keil ULINK*plus*:

Creating your own MDK 5 Blinky projects from scratch:

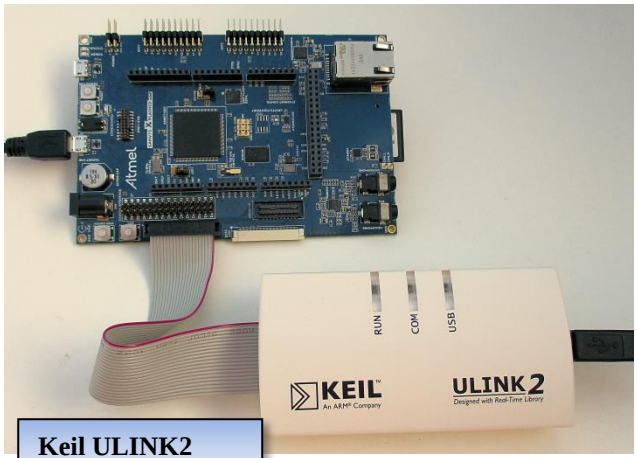
ETM Instruction Trace with ULINKpro:

Other Useful Information:

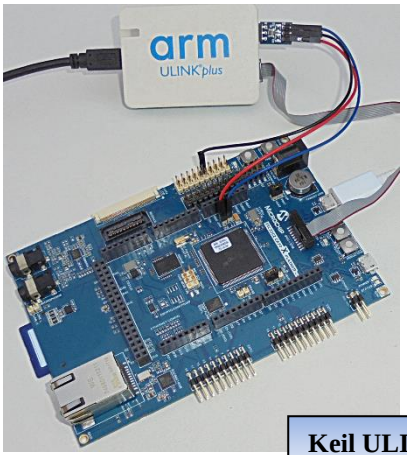
1) CoreSight Definitions: *It is useful to have a basic understanding of these terms:*

Cortex-M0 and Cortex-M0+ often have features 2), 3), 11, 12 and sometimes 10) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 have all features listed implemented except MTB. It is possible some processors have all features except ETM Instruction trace and the trace port. Consult your specific Microchip datasheet to determine its specific feature set.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities but no Boundary Scan. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDIO shares the same pin as SWO. The SWV data normally comes out the 1 bit SWO pin.
3. **DAP:** Debug Access Port. This is a component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. µVision uses DAP to update Memory, Watch, Peripheral and RTOS Thread Viewer windows in real-time while the processor is running. You can also modify variables on the fly. No CPU cycles are used, the program can be running and no source code stubs are needed. You do not need to configure or activate DAP. µVision configures DAP when you select a function that uses it.
4. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
5. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
6. **ITM:** Instrumentation Trace Macrocell: As used by µVision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
7. **Trace Port:** A 4 bit port that ULINK*pro* uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. A ULINK*pro* is needed for ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis.
9. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK*pro*. ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific Microchip datasheet.
10. **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. Only on certain Cortex-M0+ processors. Microchip Cortex-M3, Cortex-M4 and Cortex-M7 processors can provide ETM trace.
11. **Hardware Breakpoints:** The Microchip Cortex-M0+ has 2 breakpoints. The Microchip Cortex-M3, M4 and M74 have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs.
12. **WatchPoints:** Both the Microchip Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7 have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. Check your datasheet for specific details on implementation. Some Cortex-M0/M0+ do not have data compare, only address compare.



Keil ULINK2



Keil ULINK*plus*

2) MDK Keil Evaluation Software: MDK 5

MDK 5 uses Software Packs to distribute processor specific software, examples and middleware. First you install MDK 5 Core. The Software Packs you require are then downloaded with the "Packs Installer", the version(s) selected with "Select Software Packs" and configured with the "Manage Run Time Environment" (RTE) utilities in µVision. They can also be imported manually. You no longer need to wait for the next version of MDK or install patches to get the latest files.

Microchip | START provides software in MDK 5 format. MDK 4 projects are supported with a Legacy Pack.

MDK currently supports SAM L, E, S and V Microchip Cortex-M7 processors with Software Packs. Most other Microchip Cortex-M processors also have a Software Pack. See www.keil.com/Microchip for the current list.

Keil Middleware: Network, USB, File System and Graphics. For more information see www.keil.com/mdk5/middleware/

We recommend you obtain the latest Getting Started Guide for MDK5: It is available free of charge: www.keil.com/gsg/.

Microchip SAM V71 Xplained Ultra Evaluation Kit Part Number: **ATSAMV71-XULT**

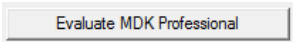
This tutorial uses this board. The SAM V71 Xplained Ultra evaluation kit is ideal for evaluating and prototyping for the SAM V71, SAM V70, SAM S70 and SAM E70. The SAM V71 is a superset of these processors.

Keil Middleware: Network

Forums: www.keil.com/forum http://community.arm.com/groups/tools/content https://developer.arm.com/

3) Keil Software Download and Installation:

1. Download MDK 5.26 or later from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5
3. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
4. If you install MDK into a different folder, you will have to adjust for the folder location differences.
5. You do not need a debug adapter: just the Microchip board, a USB cable and MDK installed on your PC.
6. You do not need a Keil license for this tutorial. All examples in this tutorial will compile within the 32 K limit.
7. You can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible: If you need additional time for evaluation purposes, please contact Keil sales. Contact information is on the last page of this tutorial.



4) USB Debug Adapters:

Keil manufactures several adapters. These are listed below with a brief description. See www.keil.com/ulink/

1. **ULINK2:** ULINK2 supports Serial Wire Viewer (SWV). Run-time memory reads and writes for the Watch, Memory and Peripheral windows plus hardware breakpoints can be set/unset on-the-fly are all provided. See page 3.
2. **ULINK*plus*:** ULINKplus adds Power Measurement and I/O. See www.keil.com/mdk5/ulink/ulinkplus
3. **ULINK*pro*:** This is pictured on page 1. ULINK*pro* supports all SWV features and adds ETM Instruction Trace support. ETM records all executed instructions. ETM provides complete Code Coverage, Execution Profiling and Performance Analysis features. ULINK*pro* also provides the fastest Flash programming times. See page 1.

Keil supports more adapters:

1. **EDBG: (CMSIS-DAP):** An extra processor on your board becomes a debug adapter compliant to CMSIS-DAP. Microchip EDBG has a CMSIS-DAP mode which is selected in the µVision Target Options menu under the Debug tab. EDBG currently does not support SWV or ETM.
2. **Microchip-ICE:** Microchip ICE is also CMSIS-DAP compliant. It is selected as CMSIS-DAP in µVision.
3. **SAM-ICE:** SAM-ICE (blue) is selected in µVision as a J-Link. Serial Wire Viewer is usable on V 6.0 and later. See page 6.
4. **Segger J-Link:** J-Link Version 6 (black) or older versions of J-Link Ultra do not support Cortex-M7. When you try to use such a J-Link, µVison will display a notice box. If a µVison box offering to update the J-Link firmware is displayed, your J-Link is Cortex-M7 compatible once this firmware is upgraded. Most J-Links support Serial Wire Viewer. SWV data reads and writes are not displayed with a J-Link. ETM trace is not supported. See page 3.

5) µVision Software Pack Download and Install Process: (do all steps this page)

A Software Pack contains components such as header files, Flash programming, documents and other files used in a project. A Pack is a standard zip archive with a .pack file extension. It contains a .pdsc description file in XML format.

TIP: If you get an error when downloading a Pack: make sure your internet connection allows an application such as Pack Installer to download .zip files. A few do not. In this case, download your Pack from www.keil.com/pack and install it manually by double clicking on it. Manually installed Packs are an ideal method to share your own private Packs.

Information on creating your own Software Pack is found here: www.keil.com/pack/doc/CMSIS/Pack/html

1) Start µVision and open Pack Installer (PI):

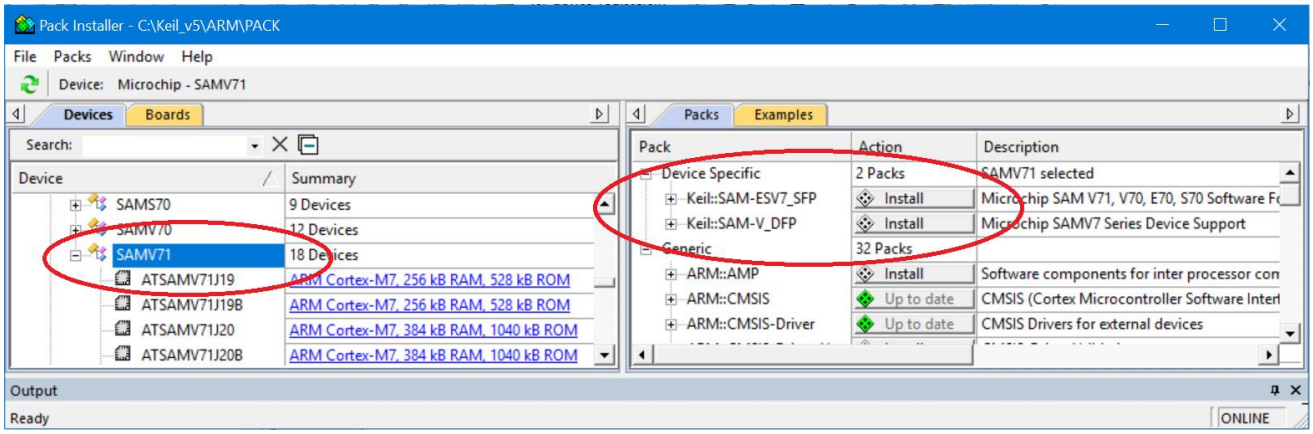
1. Connect your computer to the Internet. This is needed to download the Software Packs. Start µVision:
2. Open Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
3. The Pack Installer window opens up as shown below:
4. Note "ONLINE" is displayed at the bottom right of the Pack Installer window shown below: If "OFFLINE" is displayed, connect to the Internet before continuing.
5. Select the Update icon to refresh the Pack Installer listings. It is a good practice to do this regularly.

Install the SAM ESV7 Software Pack: (SFP = Software Foundation Pack)

This Pack contains the Chip Library from the Microchip Software Package for the SAM-SAM-E7x, SAM-S7x, and SAM-V7x. Information regarding these SFP Packs can be accessed here: www.keil.com/pack/doc/SAM_ESV7/General/html

1. Select the Devices tab on the left:
2. Select Microchip and then SAMV71 (or the processor SAME70, SAMS70 or V70 you are using) as shown below:
3. Select Keil::SAM_ES7_SFP under the Packs tab and click on Install. This Pack will download and install.

TIP: The left side (Devices and Boards) filters what is displayed on the right side (Packs and Examples).



2) Install the SAM V7 DFP Software Pack: (DFP = Device Family Pack)

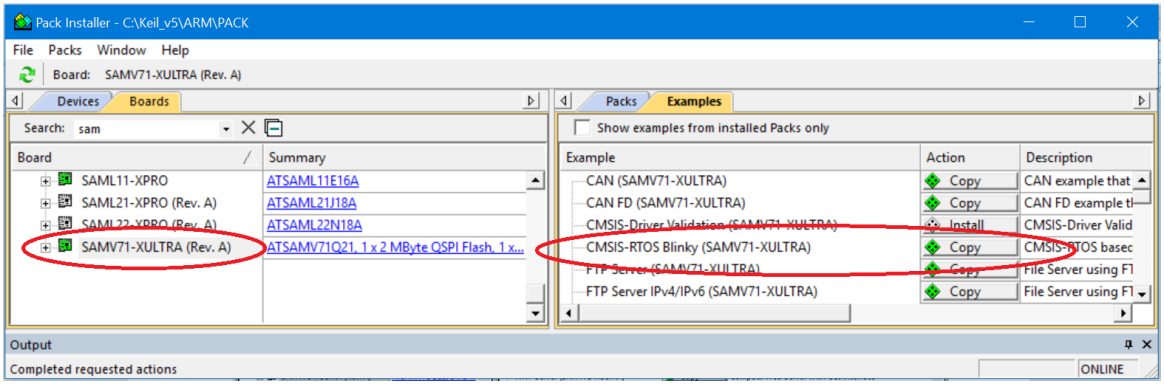
1. There are three DFP's available depending on the processor you are using. When you select the processor under Devices tab, the DFP will be automatically displayed on the right. We are using SAMV71 in this tutorial.
2. Select Keil::SAM-V_DFP under the Packs tab and click on Install. This Pack will download and install.
3. Its status will be indicated by the "Up to date" icon: Up to date
4. Update means there is an updated Software Pack available for download. Update
5. If a dialog box opens as asking to "Reload Packs" – click Yes.

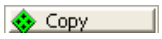
TIP: You have the option of selecting which version of a Pack you want to use. See page 7.

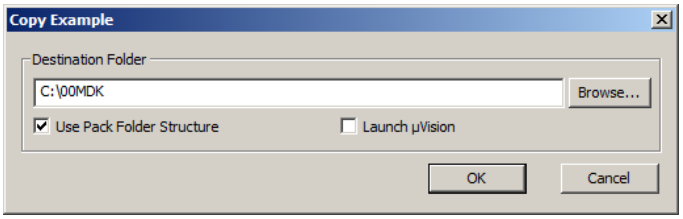
6) Examples Download and Install Process: (do 2 steps on this page)

1) Install the SAMV71 Xplained Ultra CMSIS-RTOS Blinky Example:

1. Select the Boards tab. Select SAM V71 XULTRA or the board you are using.
2. Select the Examples tab. All the examples for this board are now filtered and visible.



3. Highlight CMSIS-RTOS Blinky (SAMV71-XULTRA) It is shown above under the Example column.
4. Select Copy  opposite CMSIS-RTOS Blinky (SAMV71-XULTRA).
5. The Copy Example window shown below opens up: Select Use Pack Folder Structure. Unselect Launch µVision.
6. Type in C:\00MDK. Click OK to copy the CMSIS-RTOS Blinky project as shown below:
7. The CMSIS-Blinky example will now copy to C:\00MDK\Boards\Microchip\SAMV71-XULTRA. The folder created will be Blinky.
8. The rest of the path after C:\00MDK\ is calculated by Pack Installer.
9. You can download other examples later to try them using the Pack Installer.



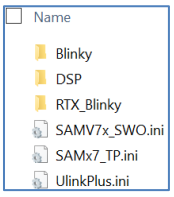
10. Close the Packs Installer. You can open it any time by clicking on its icon. 
11. If a window opens up saying "Software Pack folder has been modified": click on Yes.

TIP: The default folder for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default folder of C:\00MDK\ in this tutorial. You can use any folder you prefer.

2) Install the Examples provided by this Tutorial:

The DSP and RTX_Blinky examples are provided where you obtained this document: www.keil.com/appnotes/docs/apnt_274.asp

Extract them into C:\00MDK\Boards\Atmel\SAMV71-XULTRA. This folder will result:



SAM-ICE Version 8:

The three .ini files shown will be used to configure the processor for SWV and ETM trace operations and to configure the ULINK*plus* respectively. You will need them to add to your own projects that use SWV data or ETM instruction traces.

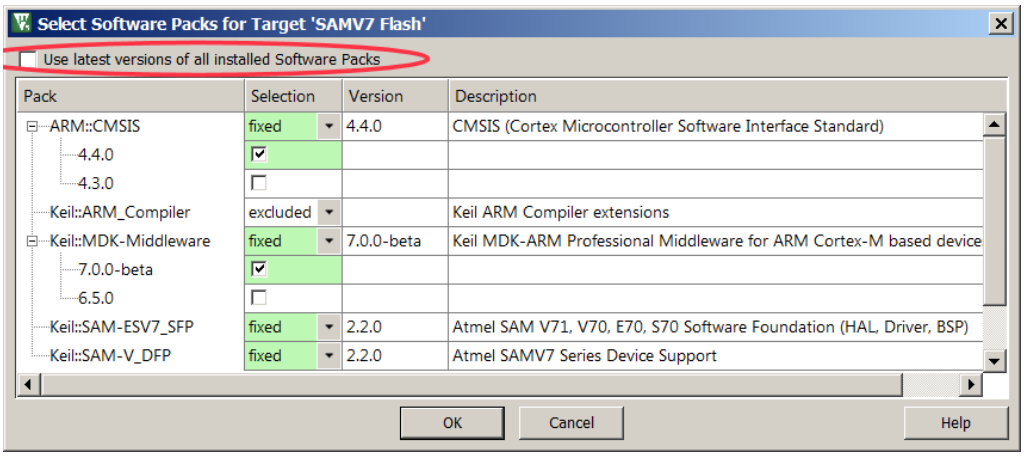
SWO = Serial Wire Output for SWV and **TP** = 4 bit Trace Port for SWV and ETM when using a Keil ULINK*pro*.

7) Other features of CMSIS-Pack Software Packs:

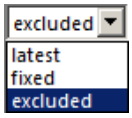
A) Select Software Pack version:

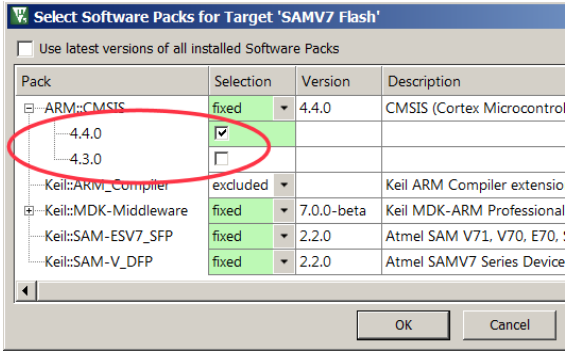
This feature provides the ability to choose various Software Pack versions installed in your computer. You can select the versions you want to use. **TIP:** You can create several sets by creating a new target under Projects/Manage/Project Items…

1. Open the Select Software Packs by clicking on its icon: 
2. This window opens up. Note Use latest versions of all Installed … is selected. The latest Packs will be used.
3. Unselect this setting and the window changes as shown below:



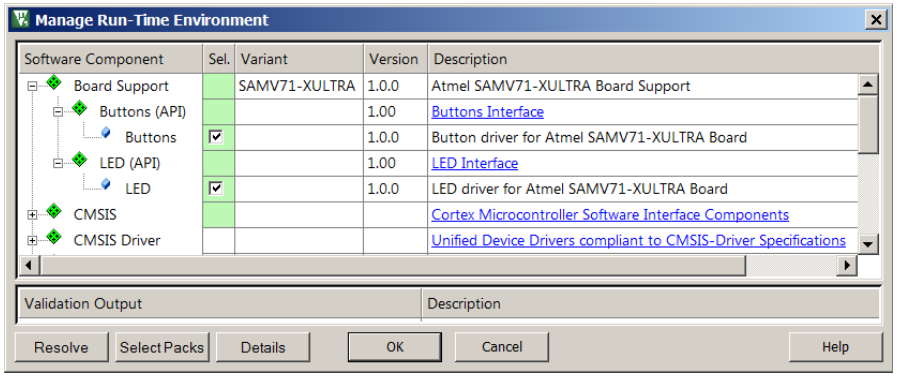
4. Expand the header ARM::CMSIS as shown above. Note various options are visible. You may see different versions depending on the Packs installed on your computer.
5. Select excluded and see the options as shown:
6. If you wanted to use V 4.3.0, you would select fixed and then select the check box opposite 4.3.0.
7. Select Use latest…
8. Close this window.





B) Manage Run-Time Environment:

1. Select the Blinky project in C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Blinky\Blinky.uvprojx.
2. Click on the Manage RTE icon:  The next window opens:
3. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project when you click OK. (do not do this now !)
4. Do not make any changes at this time. Click Cancel to close this window.



8) *Blinky* example using the Microchip SAMV71 Xplained and the EDBG adapter:

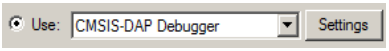
We will connect the Keil MDK development system to the Xplained board using the on-board EDBG as the debug adapter. Microchip EDBG is CMSIS-DAP compliant.

Connection and Preparation:

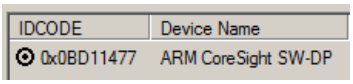
1. Connect a USB cable between your PC and USB connector DEBUG USB. The green POWER LED will illuminate. Some USB setup might occur.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project.
4. Open the file: C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Blinky\Blinky.uvprojx.

Select the EDBG CMSIS-DAP Debugger:

1. Select Target Options or ALT-F7 and select the Debug tab. Select CMSIS-DAP:



2. Select Settings: on the right side of this window. Confirm there is a valid IDCODE:



Note: If nothing or an error is displayed in the SW Device box, this ***must*** be corrected before you can continue. Check your USB connections. To refresh: switch to JTAG in Port: box and back to SW. It might take some time for the EDBG drivers to install.

3. Click OK twice to return to the main µVision menu when everything is working correctly.

Compile and RUN the Blinky Program:

1. Compile the source files by clicking on the Rebuild icon. You can also use the Build icon beside it.
2. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Progress will be indicated in the Output Window. Select OK if the Evaluation Mode box appears. The yellow LED STATUS will blink while in Debug mode.
3. Click on the RUN icon. Two yellow LEDs will start to blink.

   Note: You stop the program with the STOP icon.

LED0 and LED1 will now blink alternatively on the Microchip Xplained board.

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

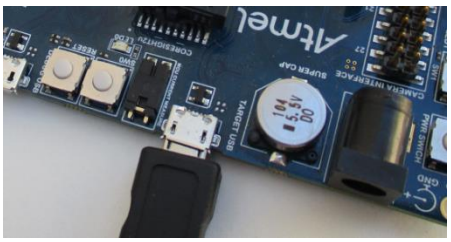
Note: The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

Each LED is on for 500 msec and off for 1.5 sec. This is mostly due to the osDelay(500) function calls in Blinky.c.

TIP: To use the on-board EDBG debugger, connect a USB cable to the USB connector DEBUG USB to power the board. To use an external debugger such as a Keil ULINK2, ULINK*pro*, SAM-ICE or a J-Link Plus or Ultra +, connect USB power to the connector labelled TARGET USB or to the VIN 5-14 V barrel power connector.

TIP: When using an external debugger and the green POWER LED blinks and a USB cable connected to TARGET USB connector, it is possible the board is using more power than your PC USB port can provide. Use a 5-14 volt external supply connected to VIN or a stronger USB source. You can try using DEBUG USB but risk conflict with the EDBG.

DEBUG USB to connect to EDBG Debugger

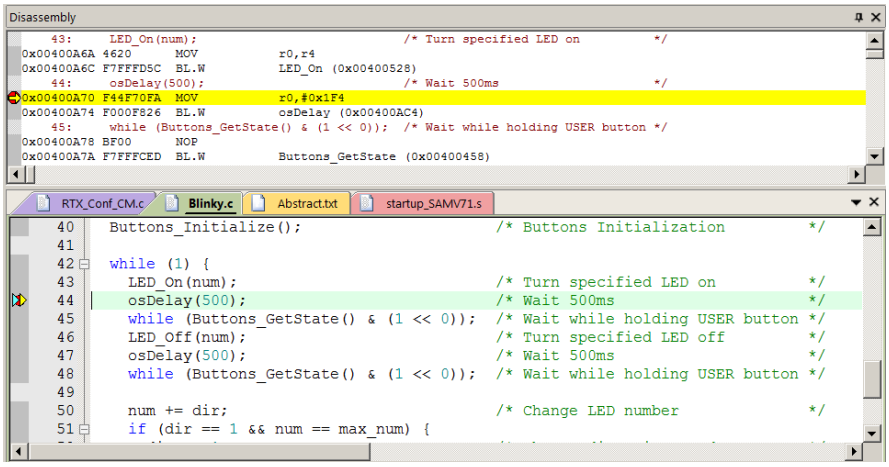


TARGET USB to power board when using an external debug adapter.

9) Hardware Breakpoints and Single Stepping:

The Microchip SAM V7 has six hardware breakpoints that can be set or unset on the fly while the program is running. This feature works using a CMSIS-DAP compatible such as EDBG or any Keil ULINK or a J-Link Plus or Ultra debug adapter.

1. With Blinky running, in the Blinky.c window, click on a darker grey block on the left on a suitable part of the source code. This means assembly instructions are present at these points. Inside the while (1) loop inside the main() function between near lines 42 through 55 is a good place: You can also click in the Disassembly window on a similar block to set a breakpoint.
2. A red circle will appear and the program will presently stop.
3. Note the breakpoint is displayed in both the Disassembly and source windows as shown here:
4. Set a second breakpoint in the while() loop as before.
5. Every time you click on the RUN icon the program will run until the breakpoint is again encountered.
6. The yellow arrow is the current program counter value.
7. Clicking in the source window will indicate the appropriate code line in the Disassembly window and vice versa. This is relationship indicated by the cyan arrow and the yellow highlight:



8. Open Debug/Breakpoints or Ctrl-B and you can see any breakpoints set. You can temporarily unselect them or delete them. This window also will contain any Watchpoints.
9. **Delete all breakpoints and close the Breakpoints window.**

TIP: If you set too many breakpoints, µVision will warn you. Microchip Cortex-M processors usually have six. The Cortex-M0 usually has only two.

TIP: ARM hardware breakpoints do ***not*** execute the instruction they are set to and land on. This will be the next instruction executed. Arm CoreSight hardware breakpoints are no-skid. This is a rather important feature for effective debugging.

Single-Stepping:

1. With Blinky.c in focus (Blinky.c tab is underlined), click on the Step icon or F11 a few times: You will see the program counter jumps one C line at a time. The yellow arrow indicates the next C line or instruction to be executed.

TIP: Step might not seem to do anything if the program is in the RTX idle daemon (this is likely in this example). The CPU is executing a Branch to itself instruction each time you click Step.
Set a breakpoint on one of the function calls osDelay(500); in Blinky.c. These are near lines 44 and 47.
The program will then stop outside of the idle daemon and Step will now function normally.

2. Click on the top margin of the Disassembly window to bring it into focus. Clicking Step now jumps the program counter one assembly instruction at a time.

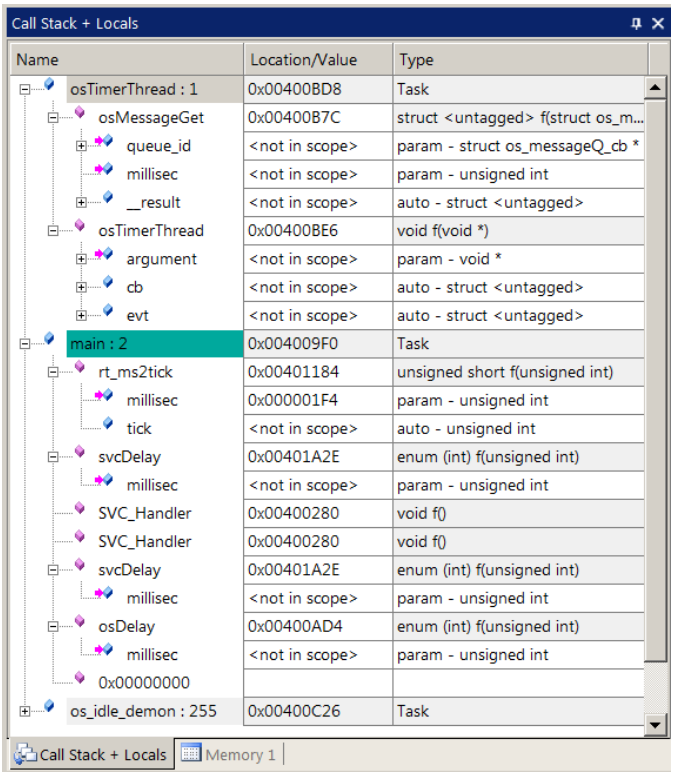
10) Call Stack + Locals Window:

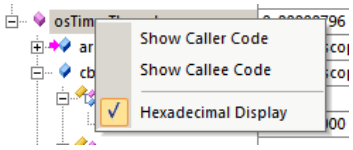
Local Variables:

The Call Stack and Locals 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 located in 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 in the main µVision window when in Debug mode.

1. Set a breakpoint on one of the function calls to osDelay(500); in Blinky.c. These are located near lines 44 and 47.
2. Click on RUN  .
3. Click on the Call Stack + Locals tab if necessary to open it. Expand some of the entries.
4. Shown is this Call Stack + Locals window:
5. The functions as they were called are displayed. If these functions had local variables, they would be displayed.
6. Click on the Step icon  or F11 a number of times:
7. As you click on Step, you can see the program entering and leaving various functions. Note the local variables are displayed.
8. If you get stuck in a delay or the os_idle_Demon, click on RUN to start over.
9. Note that this program has RTX RTOS running but there are only three threads. main(), os_idle_demon and osTimerThread are the three threads. osTimerThread was created but never active in this simple example.
10. Right click on a function in the Call Stack and Locals window and select either Callee or Caller code and this will be highlighted in the source and disassembly windows.





11. When you ready to continue, remove the hardware breakpoint by clicking on its red circle ! You can also type Ctrl-B, select Kill All and then Close.

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

TIP: This window is only valid when the processor is halted. It does not update while the program is running. Any local variable values are visible only when they are in scope.

TIP: Inspecting the stack is a useful method of debugging problems. This can be enhanced by using ETM instruction trace.

Do not forget to remove any hardware breakpoints before continuing !

11) 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 using the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to "put" or insert values into the Memory window in real-time. It is possible to "drag and drop" variable names into windows or enter them manually. You can also right click on a variable and select Add *varname* to.. and select the appropriate window. The System Viewer windows (peripheral views) and the System and Thread Viewer (RTX viewer) also use the same CoreSight technology.

A) Watch window:

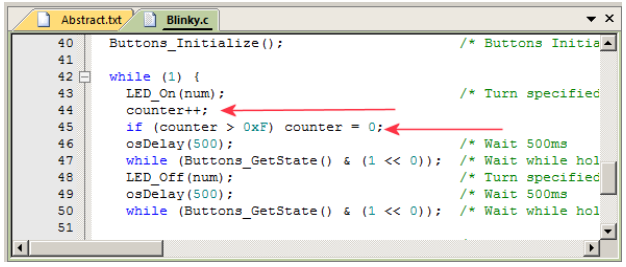
Add a global variable: Call Stack, Watch and Memory windows can't see local variables unless stopped in their function.

1. Stop the processor and exit Debug mode.
2. Declare a global variable called **`counter`** near line 19 in Blinky.c:

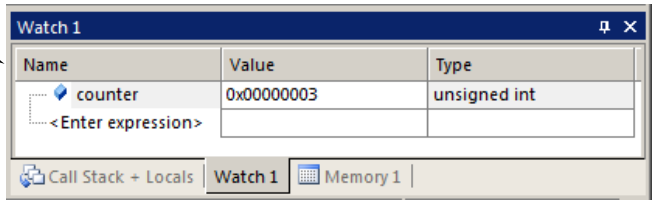
   uint32_t counter = 0;
3. Add these statements after LED_On(num); near line 44:

   counter++;

   if (counter > 0x0F) counter = 0;



4. Select File/Save All.
5. Click on Rebuild .
6. Enter Debug mode. The Flash will be programmed. Click on RUN . You can configure a Watch window while the program is running. You can also do this with a Memory window.
7. In Blinky.c, right click on any instance of **`counter`** and select Add counter to … and select Watch 1. Watch 1 will automatically open. **`counter`** will be displayed:
8. **`counter`** will update in real time.

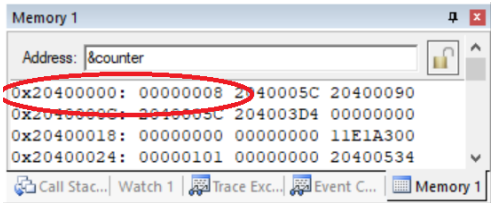


TIP: To Drag 'n Drop into a tab that is not active, pick up the variable and hold it over the tab you want to open; when it opens, move your mouse into the window and release the variable.

TIP: A Watch or Memory window can display and update global and static variables, structures and memory and peripheral addresses while the program is running. These are unable to display local variables because these are typically stored in a CPU register. These cannot be read by µVision in real-time. To view a local variable in these windows, convert it to a static or global variable or stop the program where it is in scope.

B) Memory window:

1. Right click on **`counter`** and select Add counter to … and select the Memory 1 window.
2. Note the value of **`counter`** 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 0x2040_0000.
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **`counter`** is displayed as shown here:
6. Both Watch and Memory windows are updated in real-time.
7. Right-click with the mouse cursor over the desired data field and select Modify Memory. You can change a memory location or variable on-the-fly while the program is still running.



TIP: No CPU cycles are used to perform these operations.

TIP: To view variables and their locations use the Symbol window. Select View/Symbol Window while in Debug mode.

12) System Viewer (SV):

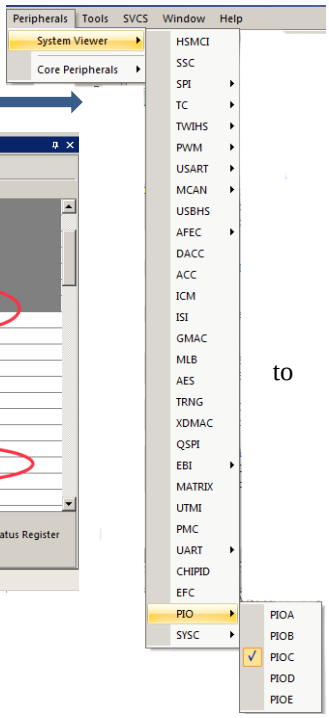
The System Viewer provides the ability to view certain registers in the CPU core and in peripherals. In most cases, these Views are updated in real-time while your program is running. These Views are available only while in Debug mode. There are two ways to access these Views: **a)** View/System Viewer and **b)** Peripherals/System Viewer. In the Peripheral/Viewer menu, the Core Peripherals are also available:

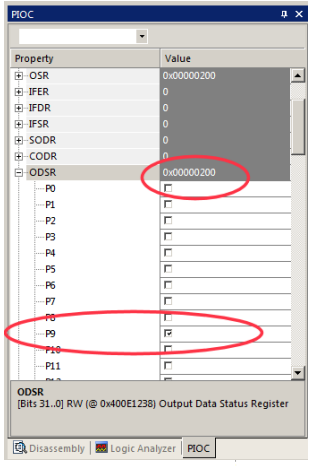
In our Blinky example, LED0 is connected to Port A pin 23 (PA23) and LED1 is connected to Port C pin 9 (PC09).

1. Click on RUN. You can open SV windows when your program is running.

GPIO Port C:

2. Select Peripherals/System Viewer/PIO and then PIOC as shown here:
3. This window opens up. Expand ODSR:
4. You can now see P9 update as the LEDs blink in succession:
5. You can change the values in the System Viewer on-the-fly. When LED1 is off, click in the P9 box and it will come on.
6. This window is updated using the same CoreSight DAP process as the Watch and Memory windows.
7. Look at other Peripherals contained in the System View windows to see what else is available.





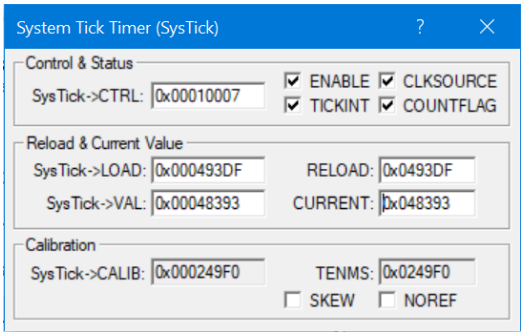
TIP: If you click on a register in the properties column, a description about this register will appear at the bottom of the window. This is an easy way to determine the physical address of a register. ODSR address is shown as 0x400E 1238.

SysTick Timer: This program uses the SysTick timer as a tick timer for RTX RTOS. The function osDelay(500); is used to slow the program down.

RTX has configured the SysTick timer using values contained in RTX_Conf_CM.c.

1. Select Peripherals/Core Peripherals and then select SysTick Timer. Run the program.
2. The SysTick window shown below opens:
3. Note it also updates in real-time while your program runs using CoreSight DAP technology.
4. Note the ST_RELOAD and RELOAD registers. This is the reload register value. This is set during the SysTick configuration by RTX using values set in RTX_Conf_CM.c.
5. Note that it is set to 0x493DF. This is created by 300 MHz/0x493DF = 0x3E8 = 1,000 which is the value in RTX_Conf_CM.c . Changing the reload value changes how often the SysTick timer creates its interrupt 15.
6. In the RELOAD register in the SysTick window, *while the program is running*, type in 0x7000 and click inside Current box.
7. The blinking LEDs will speed up. This will convince you of the power of ARM CoreSight debugging.
8. Replace RELOAD with 0x493DF. A CPU RESET 

will also accomplish this.
9. When you are done, stop the program  and close all the System Viewer windows that are open.



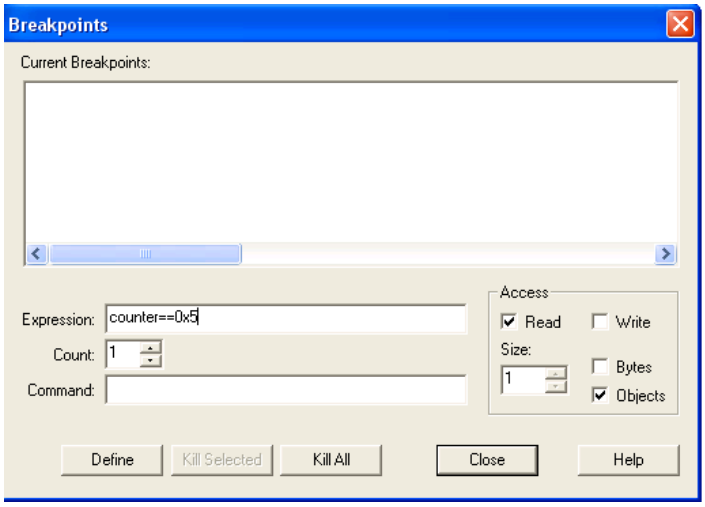
TIP: It is true: you can modify values in the SV while the program is running. This is very useful for making slight timing value changes instead of the usual modify, compile, program, run cycle.

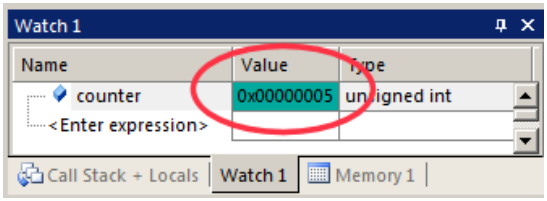
You must make sure a given peripheral register allows and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.

13) Watchpoints: *Conditional Breakpoints*

The Microchip Cortex processors have two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil documents. Cortex-M0+ Watchpoints are slightly intrusive. When the Watchpoint is hit, µVision must test the memory location. Cortex-M3/M4/M7 Watchpoints equality tests are not intrusive.

1. Use the same Blinky configuration as the previous page. Stay in Debug mode. You can set a Watchpoint on-the fly.
2. We will use the global variable **`counter`** you created in Blinky.c to explore Watchpoints.
3. Select Debug in the main µVision window and then select Breakpoints or press Ctrl-B.
4. Select Access to Read.
5. In the Expression box enter: "**`counter` == 0x5**" without the quotes. This window will display:
6. Click on Define or press Enter and the expression will be accepted as shown below in the bottom Breakpoints window:
7. Click on Close.
8. Enter the variable **`counter`** in Watch 1 if it is not already there.
9. Click on RUN.  .
10. When **`counter`** equals 0x5, the Watchpoint will stop the program. See Watch 1 shown below:
11. Watch expressions you can enter are detailed in the Help button in the Breakpoints window. Triggering on a data read or write is most common. You can leave out the variable value and trigger on any Read and/or Write.
12. To repeat this exercise, click on RUN again. If counter does not get past 0x05, change **`counter`** to something other than 0x05 in the Watch window and click on RUN.
13. Stop the CPU. 
14. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.
15. Exit Debug mode. 



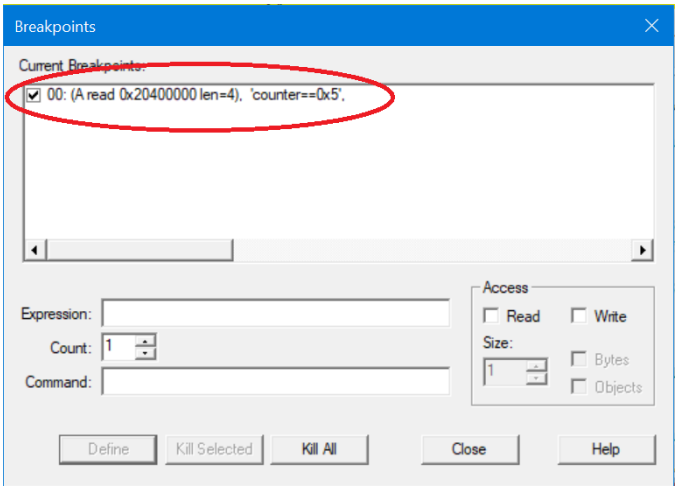


TIP: You can set/unset Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

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 be used with a Watchpoint. An example is: *((unsigned long *) 0x20000004)



14) Configuring Serial Wire Viewer (SWV) with External Debug Adapters:

This feature is only available with a Keil ULINK2, ULINK*plus*, ULINK*pro* or a J-Link. EDBG does not support SWV.

We will display the global variable counter you created earlier graphically in the Logic Analyzer.

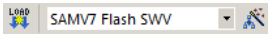
1. Stop the processor and exit Debug mode.
2. Connect a ULINK2, ULINK*plus*, ULINK*pro*, SAM-ICE or J-Link to the 20 pin SAMV71 DEBUG (SWD) or to the 20 pin connector CORESIGHT SWD and ETM depending on the adapters you have.
3. Connect a USB cable to connector DEBUG USB or Target USB to provide 5-14 volts power.

A) ULINK*pro*: Configure Serial Wire Viewer (SWV):

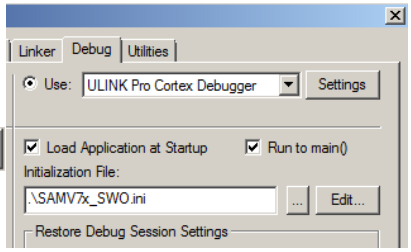
TIP: Using a ULINK*pro*, you can send SWV data either out the 1 bit SWO pin with Manchester encoding or out the 4 bit Trace Port. If sent out the Trace Port, you will need to slow the CPU speed down to 100 MHz or less. The Trace Port has a much greater throughput than the SWO pin. We will use the SWO pin in Manchester format in this tutorial.

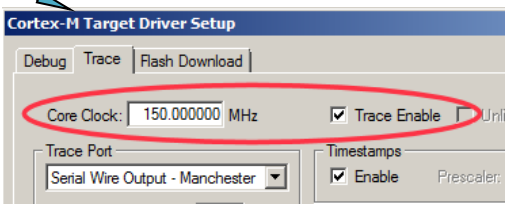
Create a new Target Options configuration:

1. Select Project/Manage/Project Items: 

2. Click on the Insert icon: Enter **SAMV7 Flash SWV** (or whatever you want). Enter and then Close.
3. Select the Target Option you just created: 

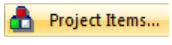
4. Select Target Options or ALT-F7. Select the Debug tab. Select ULINK Pro Cortex Debugger as shown here:
5. In the Initialization File: box, open SAMV7x_SWO.ini using Browse icon
6. Select Settings: on the right side of this window. Confirm SW is selected.
7. Select the Trace tab. In the Trace tab, select Trace Enable.
   Set Core Clock: to 150 MHz. (1/2 of 300 MHz)
   Set Trace Port to Serial Wire Output Manchester as shown here:
8. Click OK twice to return to the main µVision menu.
9. Select File/Save All.

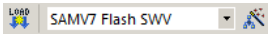




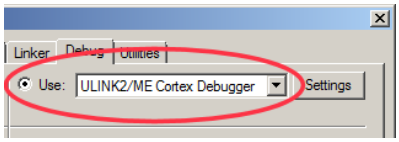
B) ULINK2, ULINK*plus*: Configure Serial Wire Viewer:

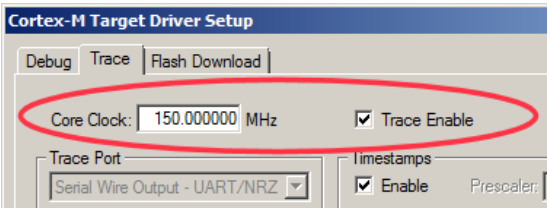
Create a new Target Options configuration:

1. Select Project/Manage/Project Items: 

2. Click on the Insert icon: Enter **SAMV7 Flash SWV** (or whatever you want). Enter and then Close.
3. Select the Target Option you just created: 

4. Select Target Options . Select the Debug tab.
5. Select Settings: on the right side of this window.
6. Select ULINK2/ME Cortex Debugger as shown here. If you are using a ULINK*plus*, select ULINKplus in this box.
7. In the Initialization File box, select SAMV7x_SWO.ini using the Browse button . This file is provided with the Blinky example. It configures the processor CoreSight for trace operation. It is run when Debug mode is entered.
8. Select Settings: on the right side of this window.
9. Confirm SW is selected.
10. Select the Trace tab. In the Trace tab, select Trace Enable. Set Core Clock: to 150 MHz as shown here: (1/2 of 300 MHz)
11. Click OK twice to return to the main µVision menu.
12. Select File/Save All.





C) SAM-ICE and J-Link: J-Link or SAM-ICE configuration is similar to ULINK2. Select J-Link J-Trace in Use:

TIP: The SWO speed is ½ of the CPU speed. Select Core Clock: ½ of CPU clock: this is 600/2 MHz in this example.

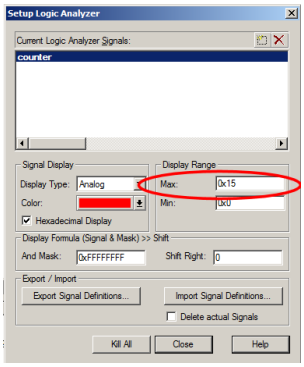
15) View Variables Graphically with the Logic Analyzer (LA):

Configure the Logic Analyzer:

1. Enter Debug mode.  Click on Run. 
2. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 

TIP: You can configure the LA while the program is running or stopped.

3. Click on the Blinky.c tab. Right click on **counter** and select Add 'counter' to… and then select Logic Analyzer. You can also Drag and Drop or enter it manually.
4. In the Logic Analyzer window, click on the Select box and the LA Setup window appears as shown here:
5. With `counter` selected, set Display Range Max: to 0x15 as shown here:
6. Click on Close.



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

1) Click on Zoom Out until Grid is about 5 seconds.
2) The variable `counter` will increment to 0x10 (decimal 16) and then set to 0.

TIP: If you do not see a waveform, exit and re-enter Debug mode to refresh the LA. You might also have to repower the Xplained board. Confirm the Core Clock: value is correct.

If you get stuck here: you probably forgot to include SAMV7x_SWO.ini:

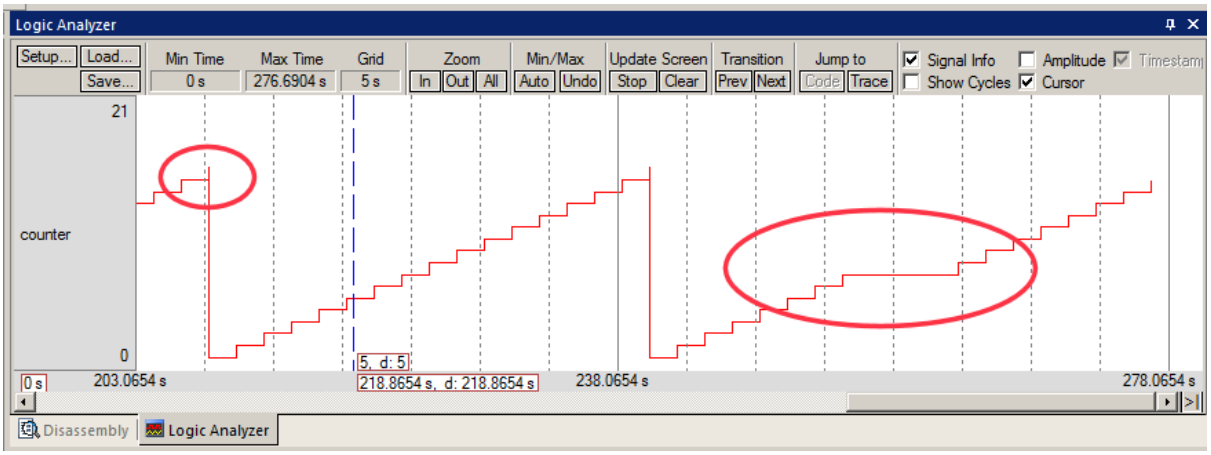
```
164   while (ITM_PORT31_U32 == 0U);
```

CoreClock Setting: In this project, this Core Clock: value must be ½ the CPU core clock. The trace circuitry in this processor in this project is clocked at half this rate. You can determine the CPU clock speed by viewing the global variable SystemCoreClock in a Watch window. The processor clocks are configured in system_SAMV71.c.

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) Press the SW0 User button and see counter increment stop as shown in the circle below:
4) Select Signal Info, Show Cycles, Amplitude and Cursor to see the measuring capabilities of the LA. You can stop the LA by clicking on the Stop icon in the Update Screen box.
5) Note counter briefly reaches 0x11 since the test is after the increment. This can be very useful to find unusual bugs.
6) When you are ready to continue, start the Update Screen.
7) Stop the CPU.  Exit Debug mode: 

TIP: The Data Write of counter will be displayed in the Trace window.



16) printf with ITM (Instrumentation Trace Macrocell): ITM uses Serial Wire Viewer:

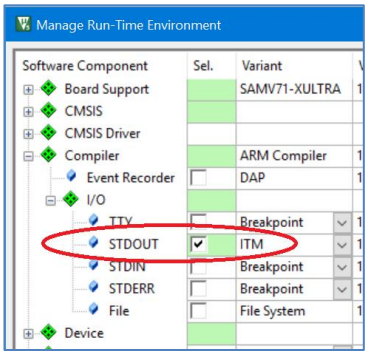
This feature is only available with a Keil ULINK2, ULINK*plus*, ULINK*pro* or a J-Link. EDBG does not support SWV.

It is easy to incorporate printf using ITM and the µVision utility Manage Run-Time Environment.

1. Stop the program if it is running and exit Debug mode.

Configure STDOUT and ITM:

2. Open the Manage Run-Time Environment utility. This window opens:
3. Expand Compiler…I/O.
4. Select STDOUT and ITM as shown here:
5. All the blocks should be green. If not, click on the Resolve button.
6. Click OK to close this window.
7. The file retarget_io.c will be added to your project in the Project window under the Compiler group.

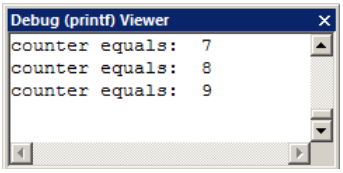


Add printf code and include:

1. In Blinky.c, near line 20, add **#include <stdio.h>**
2. In Blinky.c, near line 47 just after the if (counter>…. line, add this line: **printf ("counter equals: %d\n", counter);**

Increase RTX Stack:

1. Select RTX_Conf_CM.c, by clicking on its tab. Select the Configuration Wizard tab at its bottom.
2. Expand Thread Configuration. Increase the **Main Thread Stack size** to 300 bytes. It will change to 296 bytes.
3. Select File/Save All or click .
4. Rebuild the source files .
5. Enter Debug mode . Click on RUN .
6. Select View/Serial Windows and select Debug (printf) Viewer.
7. The values of counter is displayed as seen here:
8. Stop the program and exit Debug mode.



TIP: You can easily save ITM information to a file. See www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm

printf with Cortex-M0 and Cortex-M23 Processors:
These processors do not have SWV but you can get printf working by selecting EventRecorder and then EVR in STDOUT in the MRTE window shown above right.

Read-Only Source Files:
Some files in the Project window will have a yellow key on them: This means they are read-only. This is to help unintentional changes to these source files. This can cause difficult to solve problems. Most of these files will not need any modification in normal use.

If you need to modify a protected file, you can use Windows Explorer to modify its permission.

1. Double click on the file to open it in the Sources window.
2. Right click on its source tab and select Open Containing folder.
3. Explorer will open with the file selected.
4. Right click on the file and select Properties.
5. Unselect Read-only and click OK. You are now able to change the file in the µVision editor.
6. It is a good idea to make the file read-only when you are finished modifications.

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

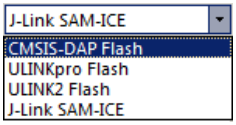
ARM CMSIS-DSP libraries are offered for Cortex-M0/M0+, Cortex-M3, Cortex-M4 and Cortex-M7 processors. DSP libraries are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS*x.x.x*\CMSIS\. CMSIS documentation is located at C:/Keil_v5/ARM/Pack/ARM/CMSIS/*x.x.x*/CMSIS/Documentation/General/html/index.html. On the web documentation is located here: www.keil.com/pack/doc/CMSIS/DSP/html.

This example creates a sine wave to which noise is added, and then the noise is filtered out leaving the original sine wave.

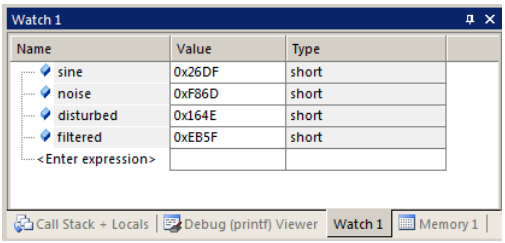
This example incorporates Keil RTX RTOS. RTX is free with a BSD or Apache 2.0 license. Source code is provided.

Running the DSP Example: The DSP example file will have been copied into your computer on page 6.

1. Open the project file sine: C:\00MDK\Boards\Microchip\SAMV71-XULTRA\DSP\sine.uvprojx.
2. Select the target for the debug adapter you are using: Select CMSIS-DAP to use the onboard EDBG adapter as shown here: If you have another adapter listed, use it.



3. Build the files.  There will be no errors or warnings.
4. Enter Debug mode by clicking on the Debug icon.  The Flash will be programed.
5. Click on the RUN icon. 
6. Open Watch 1 by selecting View/Watch/Watch 1 if necessary.
7. Four global variables will be displayed in Watch 1 as shown here: If these variables are changing the program is working properly.



8. Select View/Serial Windows/Debug (printf) Viewer. Some printf lines are displayed if you are using any ULINK2, ULINK*plus*, ULINK*pro* or a J-Link. This is implemented with SWV ITM printf.

TIP: Keil documentation uses the term "Thread" instead of "Task" for consistency.

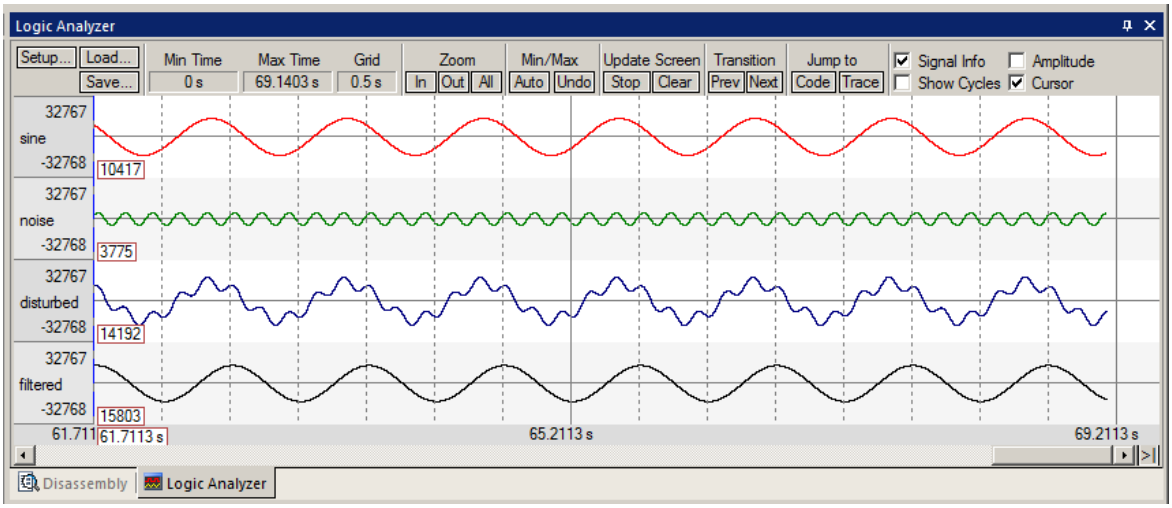
Serial Wire Viewer (SWV): (available with a ULINK2, ULINK*plus*, ULINK*pro*, SAM-ICE) EDBG does not support SWV.

If you selected either a ULINK*pro* or ULINK2 as your Target Options (see step 2 above) you will see the Logic Analyzer window below displaying the four global variables displayed in the Watch window.

The Microchip Cortex-M3, Cortex-M4 and Cortex-M7 processors have Serial Wire Viewer (SWV). This is data trace. If you use any Keil ULINK or a J-Link, you can use this Logic Analyzer window plus many other Serial Wire Viewer (SWV) features. Microchip Cortex-M7 processors also have ETM instruction trace. A ULINK*pro* is needed for this valuable ETM feature.

ETM trace will be shown later in this document.

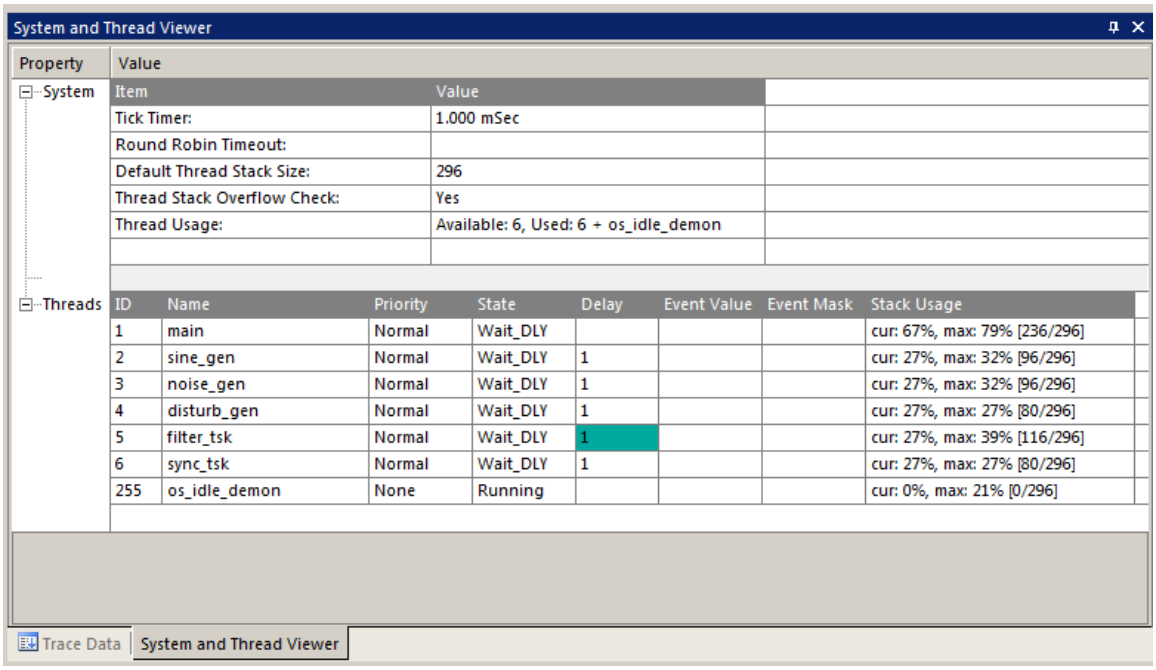
µC/Probe from Micrium might also be useful to display variables in a graphical format. It is CMSIS-DAP compliant. You do not need to be running a Micrium RTOS to use µC/Probe. See www.micrium.com/ucprobe/



RTX System and Thread Viewer: This feature works with the on-board EDBG, any ULINK or J-Link or SAM-ICE.

1. Click on RUN.
2. Open Debug/OS Support and select System and Thread Viewer. A window similar to below opens up.
3. This window updates in real-time. Note nearly all the processor time is spent in the idle daemon os_idle_demon. The processor spends relatively little time in each thread. You can change this if you need to.
4. Set a breakpoint in one of the threads in DirtyFilter.c by clicking in the left margin on a grey area.
5. Here are the four threads with their approximate starting line numbers:

   1) sine_gen (73) **2)** noise_gen (93) **3)** disturb_gen (114) **4)** filter_tsk (135)
6. The program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the noise_gen task: You can see that noise_gen was running when the breakpoint was activated.
7. os_idle_demon is Ready to run when noise_gen is finished and no other task is Ready.



TIP: os_idle_demon has a Priority of 0 which is the lowest priority possible. Every other task has a higher priority.

8. Set another breakpoint in a different task. Click on the RUN icon.
9. Each time you click on RUN, the program will stop at one of the two tasks and this is indicated by the Running state.
10. **Remove all breakpoints by clicking on them or Debug/Breakpoints or Ctrl—B and select Kill All.**

TIP: Recall this window uses the CoreSight DAP read and write technology to update this window in real-time.

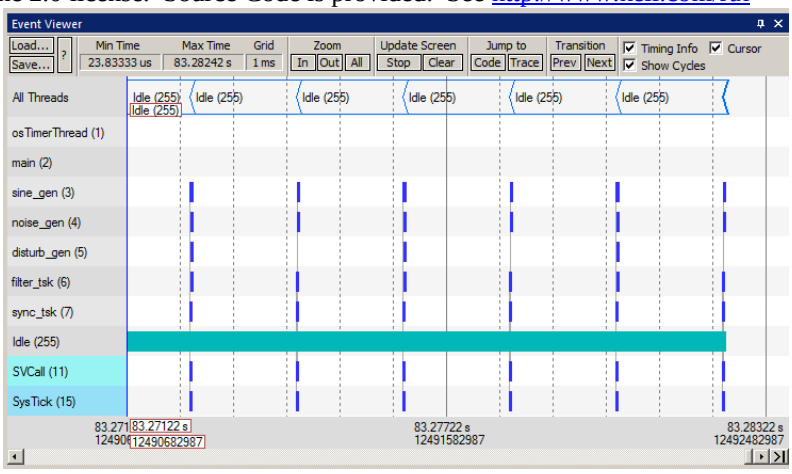
TIP: You might have noticed the Event Viewer in Debug/OS Support. This uses Serial Wire Viewer to get its information and this feature is available on Microchip Cortex-M3 , Cortex-M4 and Cortex-M7 processors with any Keil ULINK, J-Link or SAM-ICE (Version 6 or higher) debug adapter. This window is examined on the next page.

Event Viewer:

This feature is only available with a Keil ULINK2, ULINK*plus*, ULINK*pro* or a SAM-ICE. EDBG does not support SWV.

Serial Wire Viewer (SWV) must be configured for the Event Viewer to work. SWV is pre-configured in this DSP example.

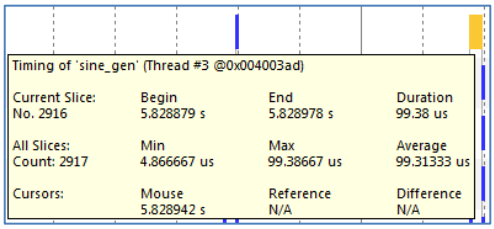
RTX is a free full feature RTOS with a BSD or Apache 2.0 license. Source Code is provided. See http://www.keil.com/rtx



1. Stop the program.
2. Select Debug/Debug Settings.
3. Click on the Trace tab.
4. Enable ITM Stimulus Port 31. Event Viewer uses this to collect its information.
5. Click OK twice.
6. If using a ULINK2, delete all variables in the LA to lessen SWO overloads.
7. Click on RUN.
8. Open Debug/OS Support and select Event Viewer. The window here opens up:

Key Features of the Event Viewer:

1. Note each thread is plotted. The program spends most of its time in the Idle daemon.
2. If you are using a ULINK*pro* or a ULINK*plus*, the timing of interrupts are also displayed. See SVCALL and Sys Tick above.
3. You can stop/start the update without stopping the program execution. Select Stop on the Update Screen box.
4. Hover your mouse on a block and it turns yellow and displays statistics. The Timing Info box must be enabled:
5. Click on a block on the Y axis it is on and a red line is created. Click on Jump to Code and this area in your source code is highlighted.
6. Select Timing Info. Move your mouse away from the red line and wait a bit for the position to register. A similar box displays statistics about the difference in pointer positions.



The Event Viewer makes it easy to view and adjust the timings of your RTX implementation.

Serial Wire Viewer Throughput: If the Event Viewer is empty or displays data that looks incorrect or unstable:

Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might occur.

TIP: It is easy to overload the Serial Wire Output pin cause it to drop trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some bandwidth. You might have to sample your own variables if they change too often. Using a ULINK*pro* in either Manchester mode or the 4 bit trace port will help.

ULINK*pro* and ULINK*plus* are much better with SWO bandwidth issues. ULINK*pro* uses the faster Manchester format than the slower UART mode that ST-Link, ULINK2, ULINK*plus* and J-Link uses. ULINK*plus* has very SWV high performance.

ULINK*pro* can also use the 4 bit Trace Port for faster operation for SWV. The Trace Port is mandatory for ETM trace.

If you are using Serial Wire Viewer extensively, the modest investment in a ULINK*pro* is easily justified.

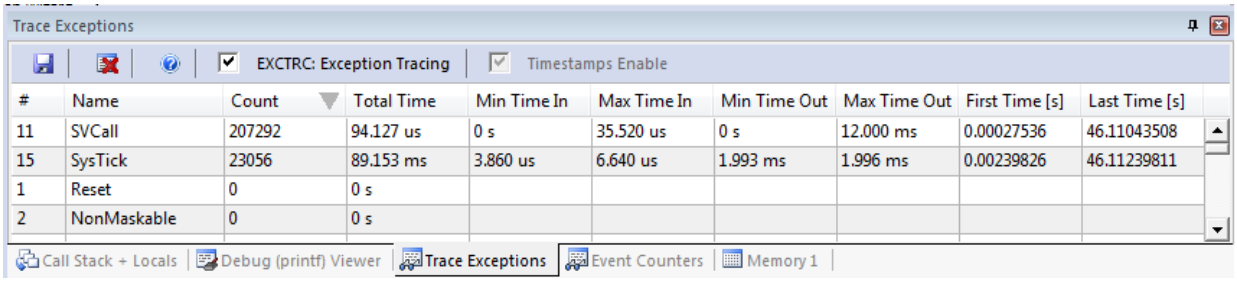
When there is a SWO or Trace Port overload and frames are lost, µVision recovers gracefully and continues displaying data.

TIP: The timestamps values in the LA are derived from the Core Clock: setting in the Trace Configuration window.

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

Trace Exceptions:

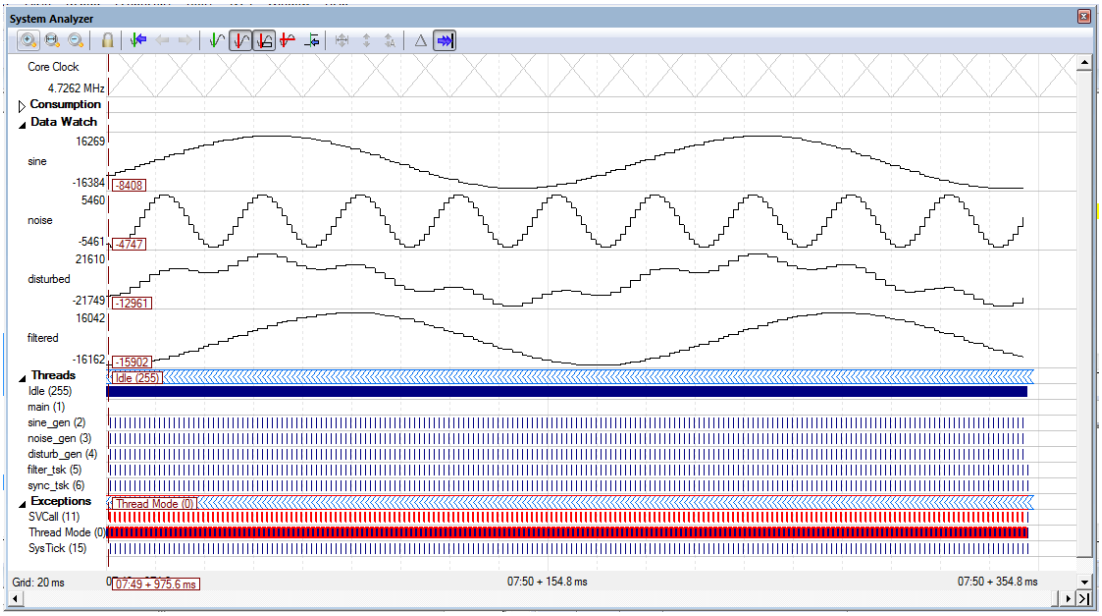
Interrupts and exceptions are displayed in the Event Viewer as shown on the previous page. They are also displayed in the Trace Exceptions window as shown here: Click on the Count column header to bring active interrupts to the top.



#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
11	SVCall	207292	94.127 us	0 s	35.520 us	0 s	12.000 ms	0.00027536	46.11043508
15	SysTick	23056	89.153 ms	3.860 us	6.640 us	1.993 ms	1.996 ms	0.00239826	46.11239811
1	Reset	0	0 s						
2	NonMaskable	0	0 s						

System Analyzer: Keil ULINK*plus* only:

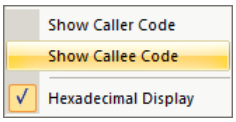
When using a ULINK*pro* or a ULINK*plus*, many SWV items are consolidated into one window as shown below. Besides the regular frames, power consumption is added (ULINK*plus* only). This is not shown here, but later in this tutorial.

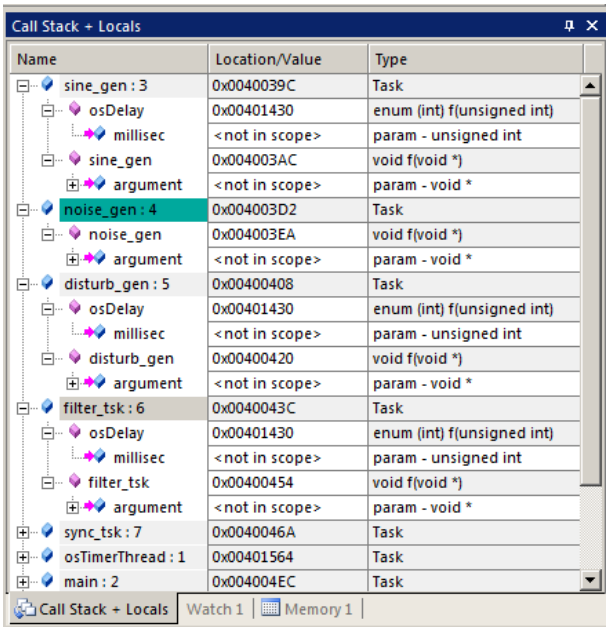


Call Stack and Locals:

1. Click on the Call Stack + Locals tab. This window opens up:

2. Each time you stop the program, the information is updated depending on which thread is running.
3. Right click on an element and select Callee or Caller Code to go there:





TIP: Recall the Call Stack and Locals window updates only when the program is stopped by one of the two breakpoints that were set on the previous page.

This is the end of the DSP example.

Using the Manage Run-Time Manager to select various CMSIS components, the correct libraries and sources will be added to your project according to the processor and/or board you selected.

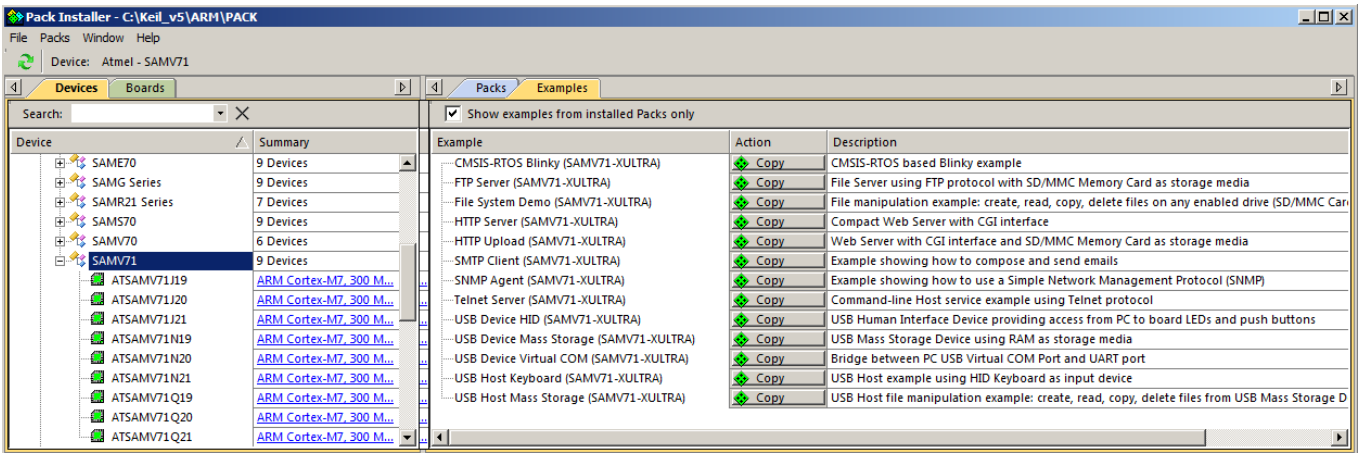
18) Keil Middleware Overview: Network: TCP/IP, Flash File, USB, Graphics:

MDK Professional provides commercial grade middleware with extensive capabilities designed for demanding applications. See www.keil.com/mdk5/middleware/ for more information.

Working examples are provided for various Microchip boards. You can access and copy these examples using Pack Installer.

Keil Middleware and RTX 5 contain annotations for Event Viewer. This is a useful debugging tool for your projects.

Here is the listing of examples for SAMV71:
This window is displayed by selecting SAMV71in the Devices tab in the Pack Installer:



License:

An MDK Pro license is needed for Keil Middleware. A 7 day one-time licenses is available in µVision under File/License Management. If you qualify, this button is displayed: Evaluate MDK Professional

To obtain a temporary MDK Professional license for evaluation purposes, contact Keil sales as listed on the last page.

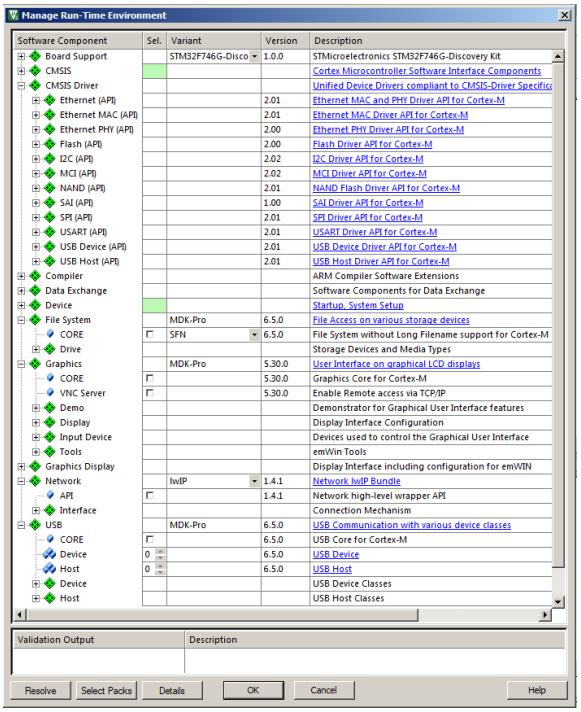
Instructions: Each example contains the file abstract.txt which provides instructions. These projects compile and run "out-of-the-box". If you have any questions regarding Keil Middleware operation during your evaluation phase, please contact Keil Technical Support as listed on the last page of this document.

Configuring the examples:

The Middleware examples make extensive use of the Configuration Wizard. This permits easy modifications to various settings with mouse clicks instead of digging through source code.

Selecting the Middleware:

Keil Middleware is selected using the Manage Run-Time Environment utility. This selects the various components you desire and place them into your project. Open the Manage Run-Time Environment utility  and this window is displayed:



Help and Documentation:

This is available both online and embedded in MDK:

www.keil.com/pack/doc/mw/General/html/

and

C:\Keil_v5\ARM\Pack\Keil\MDK-Middleware

On the next page, we will test the File System example.

19) Keil Middleware Examples USB Device HID Interface:

Running the USB Device HID Interface example:

This feature works with any Keil ULINK, SAM-ICE or EDBG. SWV is not used in this example.

You need a Keil MDK Pro or Essential license. Select File/License Management to see if you can obtain a 7 day license. Contact Keil Sales for a temporary license for evaluation purposes.

You can find additional instructions here: http://www.keil.com/pack/doc/MW/USB/html/dev_hid_tutorial.html

1. Stop the program and exit Debug mode.
2. Open the Pack Installer.
3. In the Devices tab, enter SAMV71 in the Search box.
4. Highlight SAMV71. This filters what appears on the right side of the Pack Installer.
5. Select the Examples tab and copy USB Device HID into C:\00MDK\
6. Close Pack Installer.

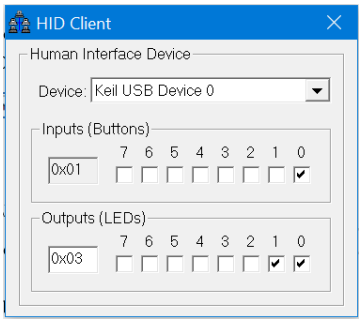
Configure the Example Project:

1. In µVision, open the HID project HID.uvprojx: C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Middleware\USB\Device\HID\
2. If you are using the on-board EDBG CMSIS-DAP adapter: select SAMV7 Flash-DAP: 

3. Connect your PC USB cable to the DEBUG USB connector.
4. If you are using another adapter, you will have to make suitable adjustments to these instructions. A preconfigured target options for ULINK*pro* is available. For others you can easily create your own.
5. Build the files. There will be no errors or warnings.
6. Enter Debug mode: Click RUN.
7. Connect another USB cable to the connector TARGET USB.
8. You will hear the USB enumerate tone. Everything is now running.

Run the HID Client Utility:

1. Using Microsoft Explorer, navigate to C:\Keil_v5\ARM\Utilities\HID_Client\Release and run HIDClient.exe.
2. The HID Client window opens as shown here:
3. Select Keil USB Device 0: Do not select EDBG.
4. Press button SW0 and Input 0 will be checked as shown here:
5. Select 0 and/or 1 in Output (LEDs) and the LEDs will illuminate.
6. If you do not see this or the Device is not recognized, cycle the USB connections.
7. If the green LED POWER blinks, supply 5 volts power to the barrel VIN connector or try the Debug USB connector. It is possible when using the Xplained board USB port it will exceed the power capabilities of your PC.



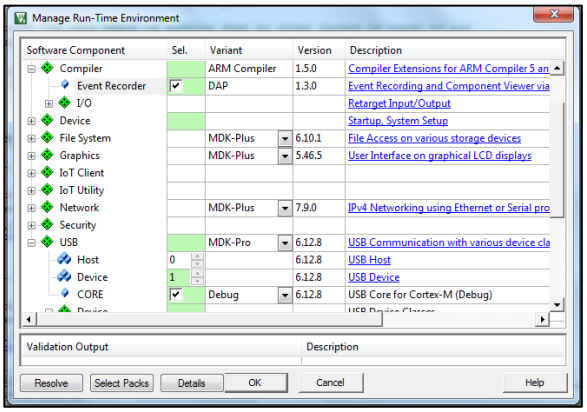
TIP: The program is now programmed into the SAMV7 Flash and will run stand-alone without µVision connected.

8. Stop the program and exit Debug mode.

Event Recorder: Now we will enable Event Recorder in the HID example The File System example also has this enabled.

NEW ! Event Recorder (ER): This feature works with any Keil ULINK, SAM-ICE or EDBG.

Event Recorder is a method to annotate your source code. ER uses DAP Read/writes and any debug adapter including EDBG CMSIS-DAP will work. Keil Middleware components have ER configured as does RTX 5.
See www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr.htm.



1. Stop the program and exit Debug mode.

Initialize and Configure Event Recorder:

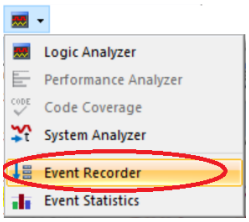
2. Open Manage Run-Time Environment.
3. Expand Compiler and USB as shown here:
4. Select Event Recorder DAP and CORE Debug as shown here:
5. Click OK when done to close MRTE window.
6. Open the file HID.c.
7. Near line 18, right click and select 'Insert #include' and add:

   #include "EventRecorder.h"

8. In the beginning of the main() function (near line 24), add this code near line 30, just after the Disable Watchdog code: EventRecorderInitialize (EventRecordAll, 1);

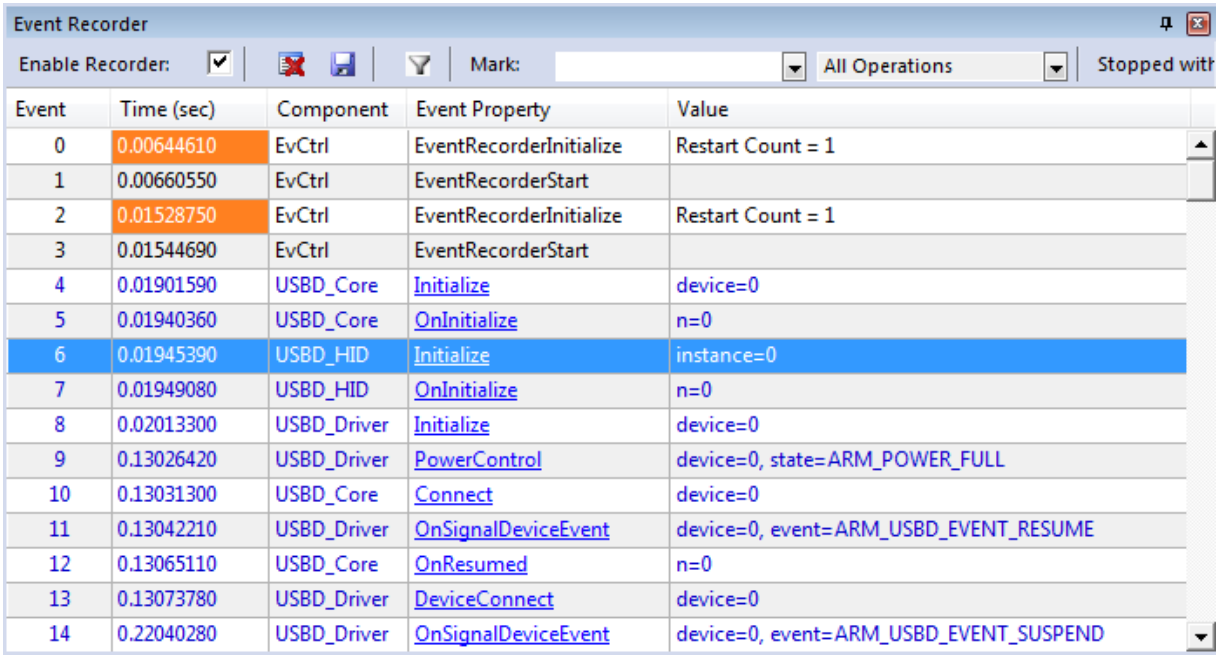
Build and RUN the Program:

1. Select File/Save All.
2. Build the files.
3. Enter Debug mode. Click on RUN.
4. Open the Event Recorder window:



5. Event Recorder will open and display information about the HID example:
6. As you press SWO button or turn the LEDs on/off, you can see the USB frames execute.
7. Stop the program and exit Debug mode.

TIP: You can create Event Recorder frames for your own source code.
See www.keil.com/pack/doc/compiler/EventRecorder/html



Event	Time (sec)	Component	Event Property	Value
0	0.00644610	EvCtrl	EventRecorderInitialize	Restart Count = 1
1	0.00660550	EvCtrl	EventRecorderStart	
2	0.01528750	EvCtrl	EventRecorderInitialize	Restart Count = 1
3	0.01544690	EvCtrl	EventRecorderStart	
4	0.01901590	USBD_Core	Initialize	device=0
5	0.01940360	USBD_Core	OnInitialize	n=0
6	0.01945390	USBD_HID	Initialize	instance=0
7	0.01949080	USBD_HID	OnInitialize	n=0
8	0.02013300	USBD_Driver	Initialize	device=0
9	0.13026420	USBD_Driver	PowerControl	device=0, state=ARM_POWER_FULL
10	0.13031300	USBD_Core	Connect	device=0
11	0.13042210	USBD_Driver	OnSignalDeviceEvent	device=0, event=ARM_USBD_EVENT_RESUME
12	0.13065110	USBD_Core	OnResumed	n=0
13	0.13073780	USBD_Driver	DeviceConnect	device=0
14	0.22040280	USBD_Driver	OnSignalDeviceEvent	device=0, event=ARM_USBD_EVENT_SUSPEND

20) Running the Middlewar File System File example:

This example works with any Keil ULINK or SAM-ICE. Since SWV is used, EDBG will not work.

You need a Keil MDK Pro or Essential license. Select File/License Management to see if you can obtain a 7 day license.

This example creates a flash memory using an SD card in the SAMV71 Xplained board.

TIP: If you are intrested in using a two-way terminal program with uVision and the Xplained board, examine this example to see how to accomplish this. A terminal program is created using the Serial Wire Viewer ITM channel.

Hardware Preparation:

1. Install a suitable SD card in the Xplained board:
2. Connect any Keil ULINK, SAM-ICE or J-Link to either the DEBUG(SWD) or the CoreSight 20 SWD+ETM connectors as appropriate for the connectors you have. Connect a USB cable between the debug adapter and the PC.
3. Connect a USB cable between the connector TARGET USB and your PC. This powers the board and the USB port.

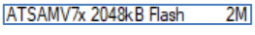
Download the File System Example and Open It:

1. Open the Pack Installer. 
2. In the Boards tab, enter **sam** in the Search: box.
3. Scroll down and select SAMV71-XULTRA or the board you are using.
4. Select the Examples tab and copy File System Demo for your board into C:\00MDK\ as shown below:
5. Close Pack Installer.
1. In µVision, open the File System project File_Demo.uvprojx: C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Middleware\FileSystem\File_Demo\File_Demo.uvprojx.

Create a new Target Options configuration:

If you are using a Keil ULINK*pro* adapter:

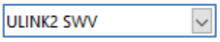
1. select SAMV7 Flash ULINK Pro: 

2. Select Target Options  . Select the Utilities tab to add the Flash programming algorithm.
3. Select Settings: and confirm ATSAMV7x 2048kB is entered. 

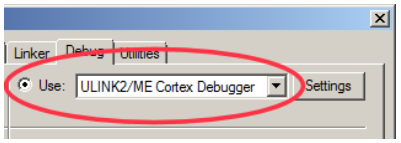
4. If not, please Add it now. Click OK twice to return to the main µVision menu. Skip the rest of this page.

If you are using a Keil ULINK2/ME, ULINK*plus*, SAM-ICE or a J-Link adapter:

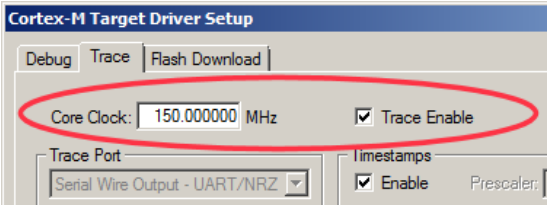
1. Select Project/Manage/Project Items: 

2. Click on the Insert icon:  Enter **ULINK2 SWV** (or whatever you want). Press Enter and then click Close.
3. Select the Target Option you just created: 

Configure the Debug Adapter: ULINK2/ME, ULINKplus, SAM-ICE or J-Link:



1. Select Target Options  . Select the Debug tab.
2. Select Settings: on the right side of this window.
3. Select the debug adapter you are using in the Use: box. A ULINK2/ME Cortex Debugger is shown here.
4. Select Settings: on the right side of this window.
5. Select the Trace tab. In the Trace tab, select Trace Enable. Set Core Clock: to 150 MHz as shown here: (1/2 of 300 MHz)
6. Below the SWO Clock Prescaler, select AutoDetect.
7. Click OK twice to return to the main µVision menu.
8. Select File/Save All. 



The next page describes how to build and run this program.

Building and Running the Project:

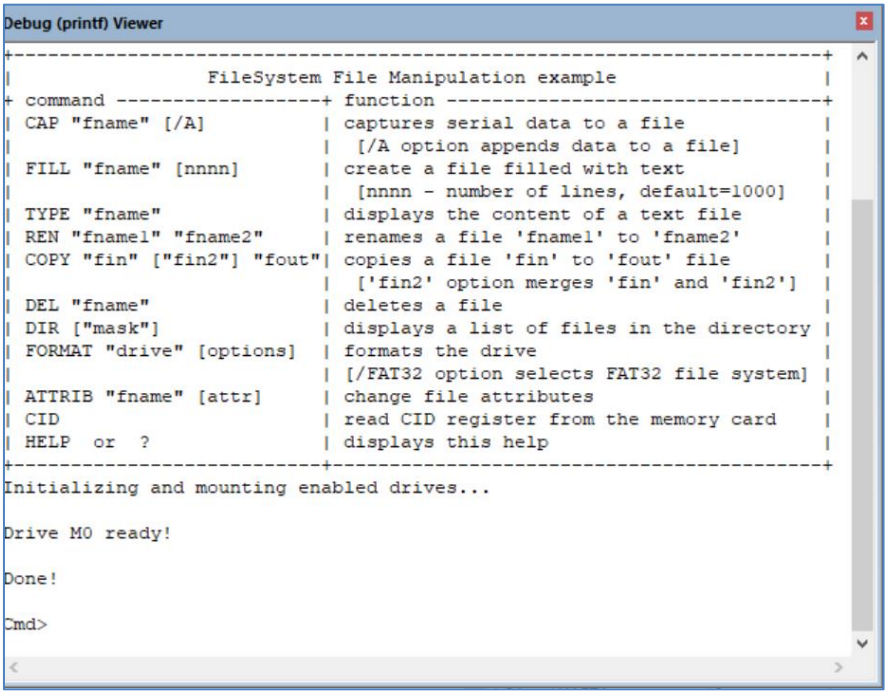
At this point, you should have an external debug adapter (any Keil ULINK or SAM-ICE) connected to the board on either debug connector depending on your adapters. There must be a USB cable connected from your PC to the TARGET USB connector. If your PC cannot supply enough power over the USB cable as indicated the green LED blinking, power the board via VIN barrel connector.

1. Build the files.  There will be no errors or warnings.
2. Enter Debug mode:  Click RUN. 
3. Everything is now running.
4. Select View/Serial Windows/Debug printf Viewer. This window opens up:
5. If there is text in the window, the program is operating correctly.

Important Note: Make sure Debug (printf) Viewer is in focus before you type on the keyboard. It is very easy to have a source window in focus and you might inadvertently modify your C source.

Testing the Example:

1. The prompt is Cmd>
2. Click on Debug (printf) Viewer to put it in focus.
3. Enter DIR and Enter and the contents of your SD card will be displayed.
4. Enter FILL MDK and Enter.
5. Enter TYPE MDK and the contents of your FILL command will be printed out.
6. The commands can be either upper or lower case.
7. Remember, make sure your cursor is in the Viewer and not a C source file.



What if it doesn't work ?

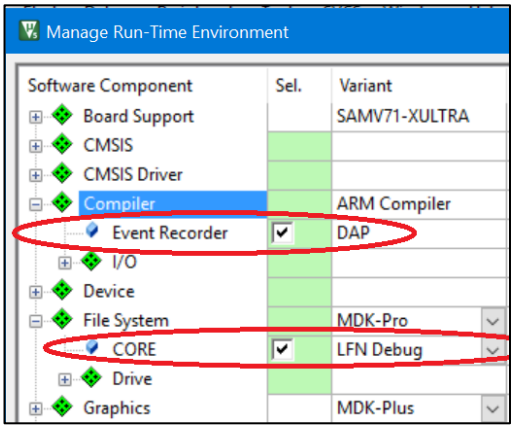
1. SWV is probably not configured correctly. Check in the Trace tab.
2. Make sure Core Clock: is set to 150 MHz. Ensure Trace is enabled and Autodetect is set. ITM Port 0 must be selected.
3. Open the Trace Data window – frames must be present.
4. You are trying to use EDBG and/or CMSIS-DAP. These will not work.
5. You failed to complete all instructions in the previous page correctly. Check carefully.
6. Exit and reenter Debug mode and try again. Cycle power to the adapter and board.

As mentioned in the Abstract.txt file, detailed instructions can be found here:
www.keil.com/pack/doc/MW/FileSystem/html/fs_examples.html#fs_standalone_example

The next page describes the Event Recorder:

NEW ! Event Recorder (ER):

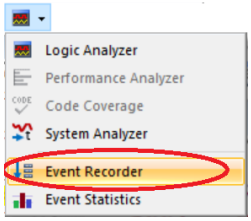
Event Recorder is a method to annotate your source code. ER uses DAP Read/writes and any debug adapter including EDBG CMSIS-DAP will work but not with this program since SWV is used for printf Viewer. Other programs not using SWV can use Event Recorder. Most Keil Middleware components have ER configured as does RTX 5. See www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr.htm.



1. Stop the program and exit Debug mode.
2. Open Manage Run-Time Environment.
3. Expand Compiler and File System as shown here:
4. Select Event Recorder DAP and LFN Debug as shown here:
5. Click OK when done to close MRTE window.
6. Open the file File_Demo.c.
7. Near line 16, right click and select 'Insert #include' and add:

   #include "EventRecorder.h"
8. In the beginning of the main() function (near line 643), add this line just after the Disable Watchdog code:

   EventRecorderInitialize (EventRecordAll, 1);
9. Select File/Save All.
10. Build the files.
11. Enter Debug mode. Click on RUN.
12. Open the Event Recorder window:
13. Event Recorder will open and display information about the File System example:
14. Enter some commands in the Debug (printf) Viewer and see the annotations display:
15. These have proved to be very useful in debugging complex code.
16. Stop the program and exit Debug mode.
17. You can create Event Recorder frames for your own source code.
    See www.keil.com/pack/doc/compiler/EventRecorder/html



Event Recorder

Enable Recorder: Mark: All Operations Missing Even

Event	Time (sec)	Compone...	Event Property	Value
0	0.00087420	EvCtrl	EventRecorderInitialize	Restart Count = 1
1	0.01658887	FsCore	finit	drive=0x00401470
2	0.01660183	FsFAT	InitDrive	drive=M0
3	0.01660963	FsMcMCI	InitDriver	instance=0, driver=Driver_MCI0
4	0.01664493	FsFAT	InitDriveSuccess	drive=M0
5	0.01664709	FsCore	fmount	drive=0x00401470
6	0.01665383	FsFAT	MountDrive	drive=M0
7	0.01665745	FsMcMCI	DevCtrl	instance=0, code=fsDevCtrlCodeCheckMedia, p=0x204006...
8	0.01666527	FsMcMCI	InitMedia	instance=0
9	0.02057848	FsMcMCI	MediaReset	instance=0
10	0.02126241	FsMcMCI	MediaDetectionSD	instance=0
11	0.02185332	FsMcMCI	MediaSD_V2	instance=0
12	0.05977918	FsMcMCI	MediaReady	instance=0, OCR=0xC0FF8000
13	0.08106677	FsMcMCI	InitSuccess	instance=0
14	0.08107194	FsFAT	ReadMBR	drive=M0. sector=0

21) Power Measurement using a Keil ULINK*plus*:

Power Measurement is provided by Keil ULINK*plus*. See www.keil.com/ulinkplus.

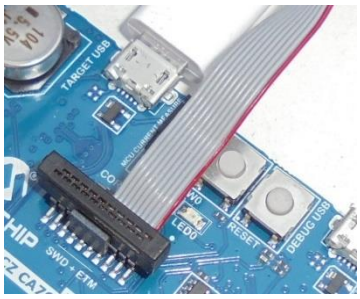
Getting the ULINK*plus* working on the SAM V71 board with SWD (Serial Wire Debug):

In this section, we will confirm the ULINK*plus* is connected to the SAM V71 processor debug connection SWD.

Load the Blinky_ULINKplus Example Project:

1. Select Project/Open Project and navigate to C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Blinky\
2. Open Blinky.uvprojx. Double-click it or select Open to load it.

Connect the debug cable to the SAMv71 Xplained board:



1. Using a 10 pin to 20 pin CoreSight adapter cable, connect ULINK*plus* to the 20 pin SWD + ETM connector as shown here:
2. Power ULINK*plus*: Connect a USB cable from DEBUG USB connector to your PC.
3. Power the SAMV71 board with a USB cable to either TARGET USB or DEBUG USB.

Configure ULINK*plus*:

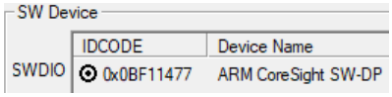
1. Select Manage Project Items:
2. Select Insert:
3. Type in: ULINKplus (or whatever you want)
4. Click OK and this will be saved.
5. Select ULINKplus in the Select Target menu: 

6. Select the Target Options icon in µVision.
7. Select the Debug tab: Select ULINKplus Debugger: 

8. Select ULINK*plus* from the Select Target dialog box: 

Confirm the ULINK*plus* is connected to the SAMV71 Cortex-M7 CPU Core:

1. Select Options for Target Options or ALT-F7.
2. Select the Debug tab: Select Settings: and the Target Options window opens:
3. You must see something in the SW Device box. If you see nothing or an error, you ***must*** fix this before continuing. You are probably not connected correctly.



4. Click OK twice to return to the main µVision menu when you have a valid IDCODE.

Build the Sources and Run the Program:

1. Compile the source files by clicking on the Rebuild icon. .
2. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed.
3. Click on the RUN icon.
4. The two yellow LEDs will blink. This means success to this point.
5. At this point, we have the ULINK*plus* acting as the debug adapter controlling the program and the board. The on-board EDBG debug adapter is bypassed. Now, we can connect the ULINK*plus* power measurement cables to the SAMV71 Xplained board. See the next page for these instructions.
6. Stop the program and exit Debug mode .

22) ULINK*plus* Power Measurement Connections:

The SAMV71 board has a jumper J201 to measure the CPU current. Measured current using Blinky is ~ 33 mA.

ULINKplus Shunt Resistors:

ULINK*plus* has an internal 100 Ω shunt resistor useful for up to and several supplied small boards each with a different shunt resistor. A file, Ulinkplus.ini is provided to configure ULINK*plus*.

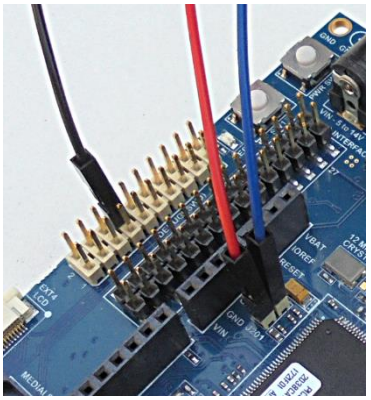
Copy Ulinkplus.ini Initialization File:

1. Locate the file Ulinkplus.ini in C:\00MDK\Boards\Microchip\SAMV71-XULTRA and copy it into \Blinky\.

Connect ULINK*plus* Power pins to SAMV71 Board:

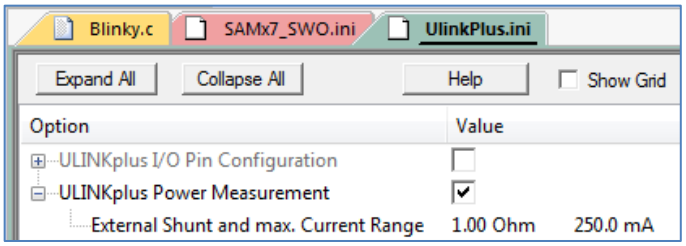
1. Remove the power from the SAMV71 board. Microchip recommends this step to not harm the processor.
2. Remove the jumper on J201.
3. Plug the Shunt 250 mA board in the ULINK*plus* as shown here:
4. Connect three jumper cables to the Shunt board.
5. Connect pin to pin 1 JP201 connector.
6. Connect pin to pin 2 JP201 connector.
7. Connect the ground jumper wire to any even pin from2 through 20 Debug (SWD) connector. These are the pins closest to the edge of the board. This is shown here: You can also the GND pin on the POWER connector.

TIP: The grounds on JTAG/SWD, Power, I/O and USB connect are all isolated from each other. You must use a separate ground cable on both Power and I/O. Do not depend on the USB ground as it is isolated.



Configure ULINK*plus* with the .ini file:

1. Select the Options for Target icon .
2. Select the Debug tab.
3. In Initialization file box, SAMx7_SWO.ini is entered.
4. Click the Edit icon to open this file in the source files area.
5. Click OK to return to the main µVision menu.
6. At the beginning of Debug_all.ini, add this line: include ".\\UlinkPlus.ini"
7. Select File/Open and select All Files (*.*).
8. Go to C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Blinky and select UlinkPlus.ini and click Open.
9. Select the Configuration Wizard tab at the bottom of this file.
10. Select 250 mA as shown here:
11. Click Save All:



TIP: The CPU voltage is 1.2 volts and it is important to use a low ohm shunt. 100 mA shunt or less will not work with this board due to an excessive voltage drop.

TIP: This .ini file is provided with the project. Debug_UlinkPlus.ini is provided at C:\Keil_v5\ARM\ULINK\Templates\. It is a similar version with extra examples.

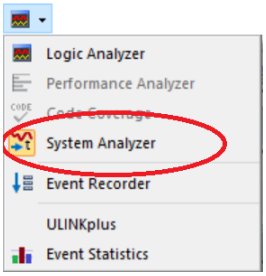
TIP: This file is executed when entering Debug mode. It is important to make sure this file is saved after making any modifications to it and before invoking Debug mode .

What we have to this point:

ULINKplus is now connected and configured. You are ready to build the Blinky project and enter Debug mode and plot the power consumption of the SAMV71 Xplained board.

23) Displaying Power Measurements in the System Analyzer (SA):

1. Build the files. Enter Debug mode: Click on RUN.
2. Open the System Analyzer (SA) window by selecting the small arrow beside Analysis Window icon as shown here:

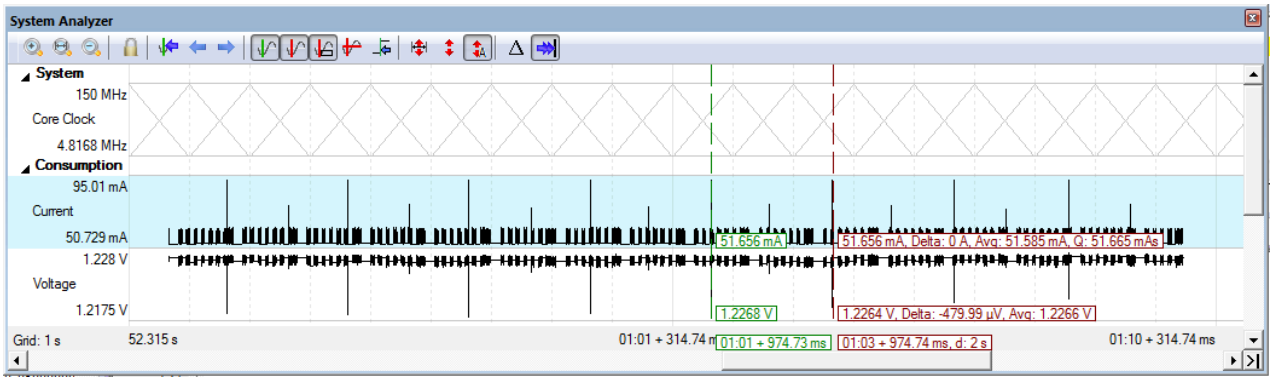


3. Position SA as appropriate. To scroll to the waveform, click: Jump to End:
4. Use Zoom icons to size the waveform. You can also use a scrolling wheel on your mouse if it is so equipped.
5. You will have a System Analyzer window similar to the one shown below:
6. With the SAMV71 board, both current and voltage of the I/O rail are clearly displayed.
7. The frequency is incorrect as the SAMV70 uses a clock that ULINK*plus* does not yet recognize.

TIP: If the Current values have a negative number, this means the Power In and Out connections are reversed.

8. Stop the SA window while leaving the program running with the Lock icon:
9. The pulses in Current are spaced 2 seconds apart indication possible LED turn on points.
10. You can use the cursors to determine current, voltage and times wherever you hover your mouse.
11. With SA locked, you can right click on the Current or Voltage display and select Lo-pass filter 160 kHz:

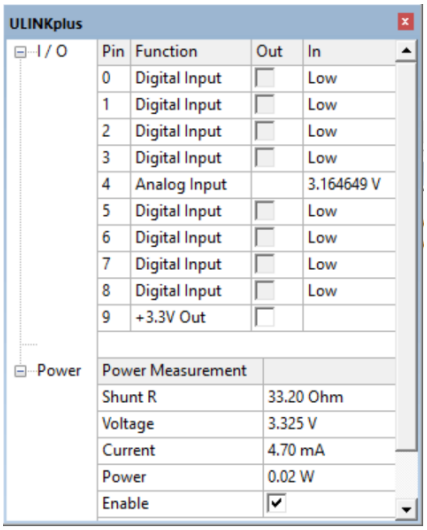
TIP: Some features in System Analyzer (SA) use SWV. The power and frequency do not need SWV.



TIP: You can measure power with or without a processor debug connection (SWD).

ULINK*plus* Window:

1. Select ULINKplus in the Analysis Select Window. See step 2 above and the first screen picture above. This window will open:
2. In the I/O section are various inputs and outputs you can use. In this case, Pin 4 is set to Analog Input and is measuring a pin on the board.
3. Note the values displayed under Power Measurement as shown here. These values update while the program runs. The SA can be locked.
4. You can change the Shunt R value at any time.
5. The values here are copied from the .ini file. Any changes made inside the ULINK*plus* window will be lost with a cycle of Debug mode.
6. Refer to the ULINK*plus* User Guide located here for more information: www.keil.com/support/man/docs/ulinkplus/



Viewing Data Write, Exceptions and RTX Threads in System Viewer:

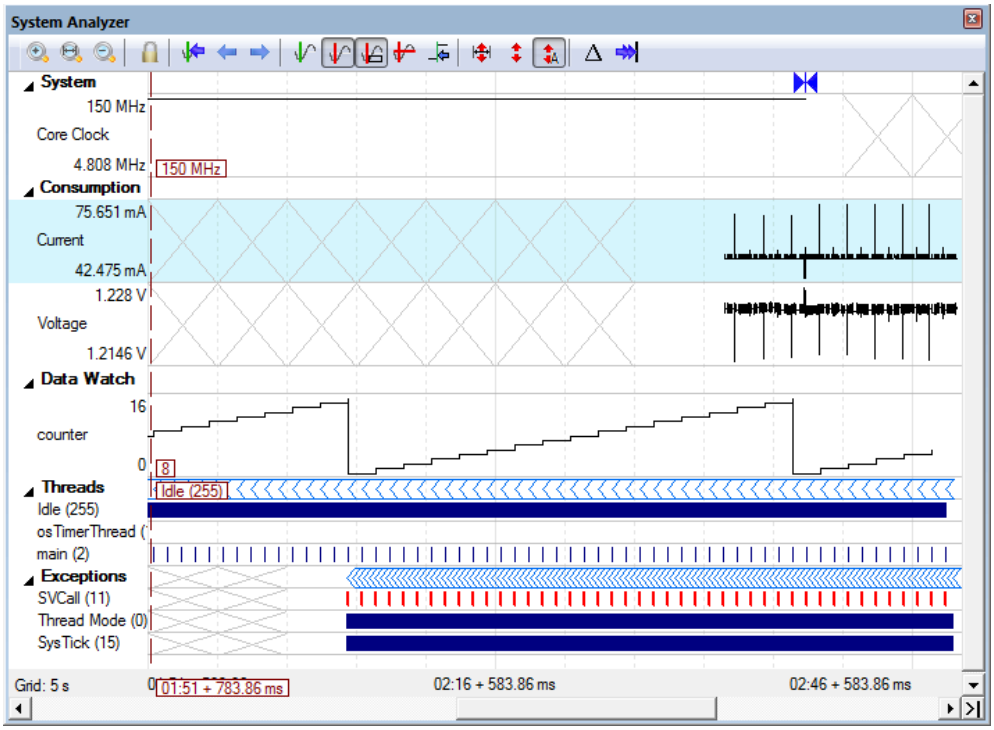
Various data will be displayed automatically in the System Viewer. You can measure times, voltages, current, total power consumed and more. The window below displays what is contained in the System Analyzer window with this example:

Timing Values: System shows the CPU clock is 150 MHz when it is really 300 MHz. This value is determined by µVision by running a test program. The clock µVision sees has a divide by 2 counter. You must therefore divide any timing values you find by 2.

This is not true for all processors.

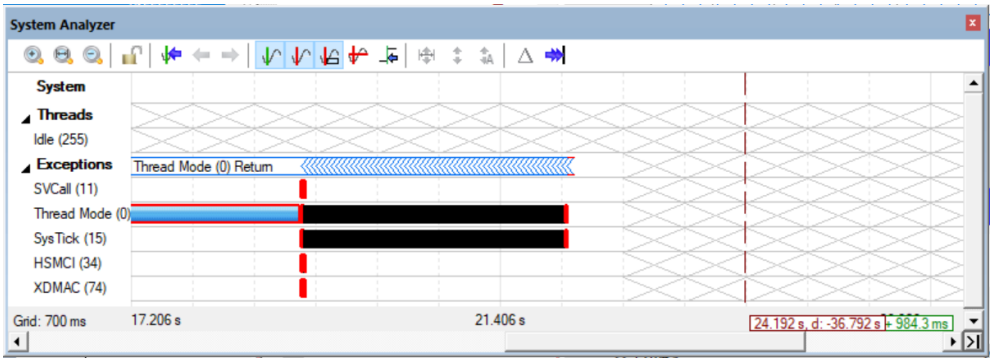
SystemCoreClock displays the correct clock frequency. This global variable can be inserted in a Watch window. It is calculated from the Systemxyz.c file and not calculated at run time.

Exceptions: All exceptions including interrupts are listed here. If you had any interrupts for DMA or any other peripheral running, their timing will be displayed here.

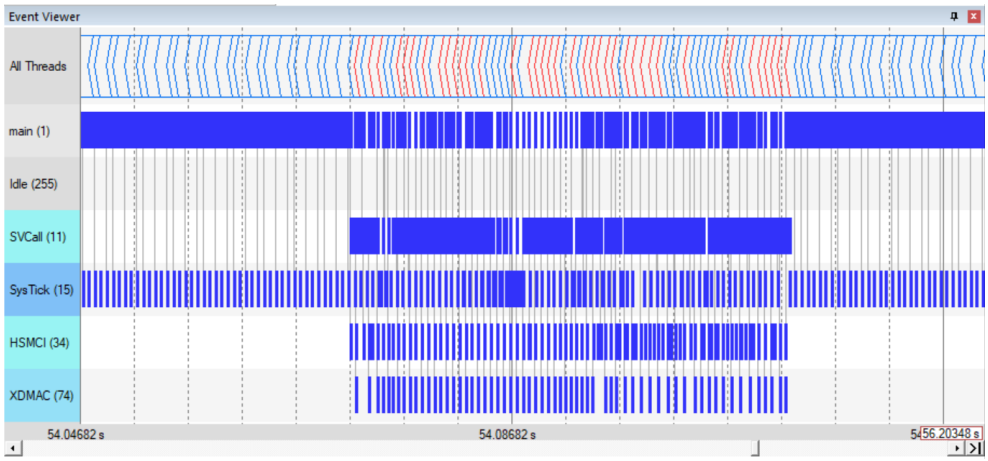


The second window displays the interrupts of the HSMCI and XDMAC peripherals. This can be expanded to show great detail.

ULINK*pro*: These waveforms except Consumption will also display with a ULINK*pro*.



Event Viewer: This window duplicates some of the data in the System Analyzer window. Event Viewer uses SWV and works with any Keil ULINK or J-Link. The interrupts will only display with a ULINK*plus* or a ULINK*pro*.

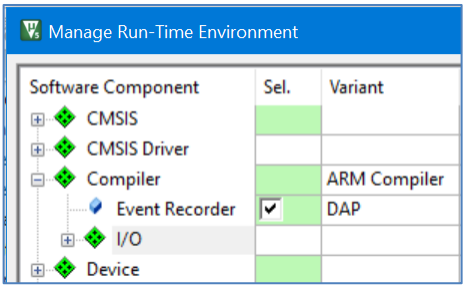


24) Annotating Your Code to Display in the System Analyzer:

You can annotate your code and this will be displayed in the System Viewer and Event Viewer windows. Keil RTX 5 RTOS (RTX 5 is TrustZone compatible) and Keil Middleware is already annotated.

Add STDOUT File (retarget_io.c):

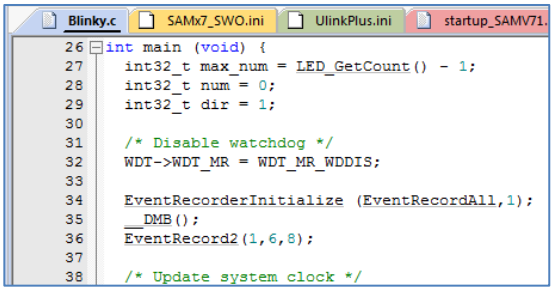
1. Open the Manage Run-Time Environment window (MRTE)  .
2. Expand Compiler as shown here:
3. Select Event Recorder DAP as shown:
4. Ensure all blocks are green and click OK to close the MRTE. Click Resolve if there are red or orange blocks.



Add Event Recorder (ER) statements to Blinky.c:

1. At the top of Blinky.c near line 21, right-click and select 'Insert #include file' and then select EventRecorder.h.
2. At the beginning of main() near line 34, after Watchdog disable, add these three lines:

```
EventRecorderInitialize (EventRecordAll,1);
__DMB();
EventRecord2(1,6,8);
```



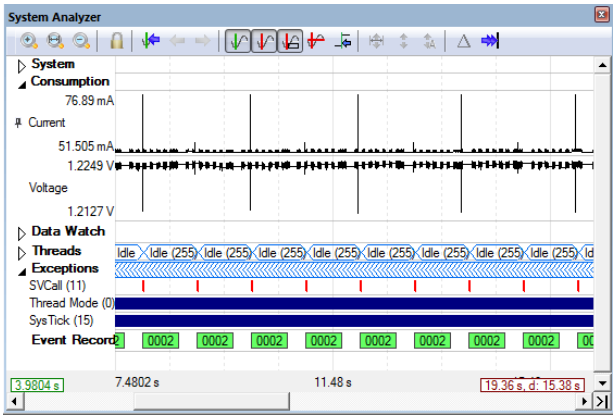
Add Event Recorder (ER) statements to Blinky.c:

1. At the beginning of the while(1) loop near line 47, add this line:

```
EventRecord2(2,6,8);
```

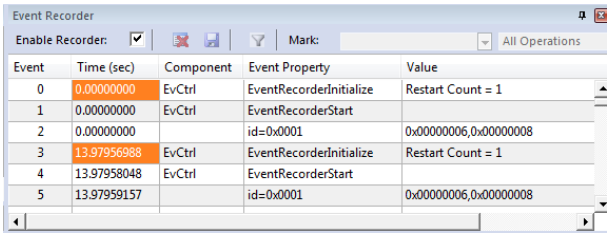
Compile and Run the Project:

1. Select File/Save All or click  .
2. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !
3. Enter Debug mode  . Click RUN. 
4. Open System Analyzer (SA) if it is not already open:
5. Click Freeze Data  :
6. Note I have collapsed some of the rows. You do not have to at this time.
7. A new row is created called Event Record.
8. Adjust the SA window and you will see the Record2 ID = 2 icons in the Event Record row:
9. If you scroll to the left of this window you can see the 0001 Record you put at the beginning of main(): 
10. You use this to see the code and waveform match:



Event Recorder Viewer:

1. Select the small arrow beside the Analysis window icon and select Event Recorder.
2. Event Recorder opens. Scroll to the top and you can see where these were initialized.
3. You can see the result of Record2 with ID 1 or 2.
4. Each has a data values of 6 and 8.



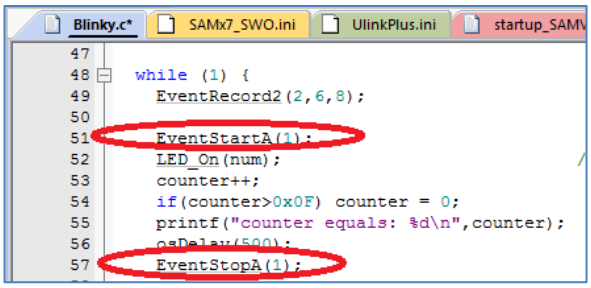
TIP: It is possible to annotate your code with Event Recorder with your own messages using the SCVD: format.
See: www.keil.com/pack/doc/compiler/EventRecorder/html/

25) Measuring Power Consumption with Event Statistics:

You have seen the current and voltage waveforms and how you can annotate your code to show a relationship between them. Using Event Statistics window, you can determine values over ranges of your code.

Documentation is here: www.keil.com/pack/doc/compiler/EventRecorder/html/es_use.html

1. Stop the program and exit Debug mode.

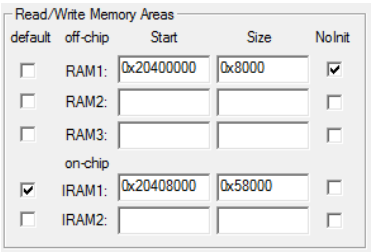


Create Event Stop and Start:

1. In Blinky.c, near line 51 add: EventStartA(1);
2. In Blinky.c, near line 57 add: EventStopA(1);

Allocate RAM for Event Recorder:

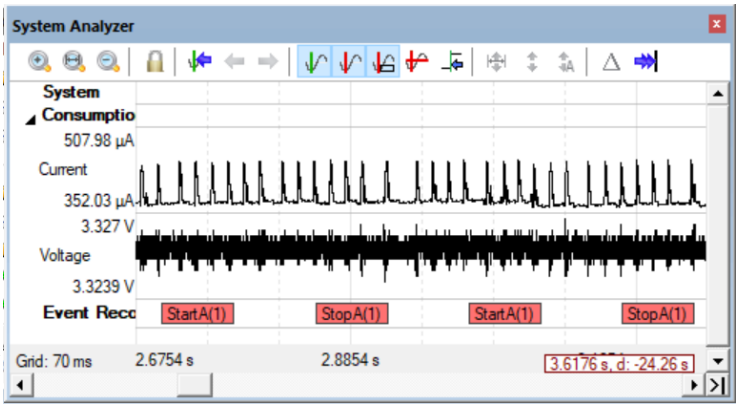
1. Select the Option for Target:
2. Select the Target tab.
3. Modify Read/Write Memory exactly as shown here:
4. Click OK to close this window.
5. Right click on Compiler in the Project window.
6. Select Options for Component Class 'Compiler'.
7. Select Event Recorder. Click the Memory tab.



8. In Zero Initialized Data: select RAM1…
9. Click OK to close this window.

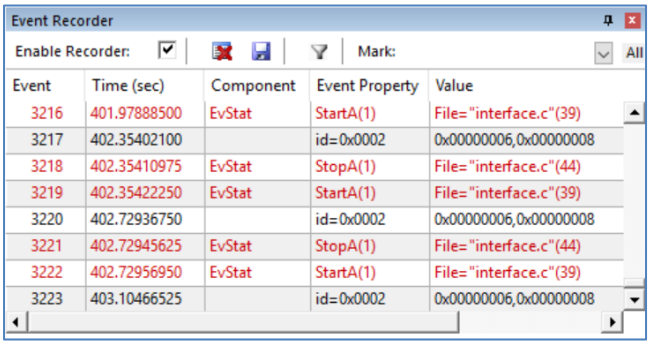
Build and RUN the program:

1. Select File/Save All or click.
2. Build the files.
3. Enter Debug mode. Click RUN.
4. Open the System Analyzer window.
5. Note Start and Stop events now displayed:



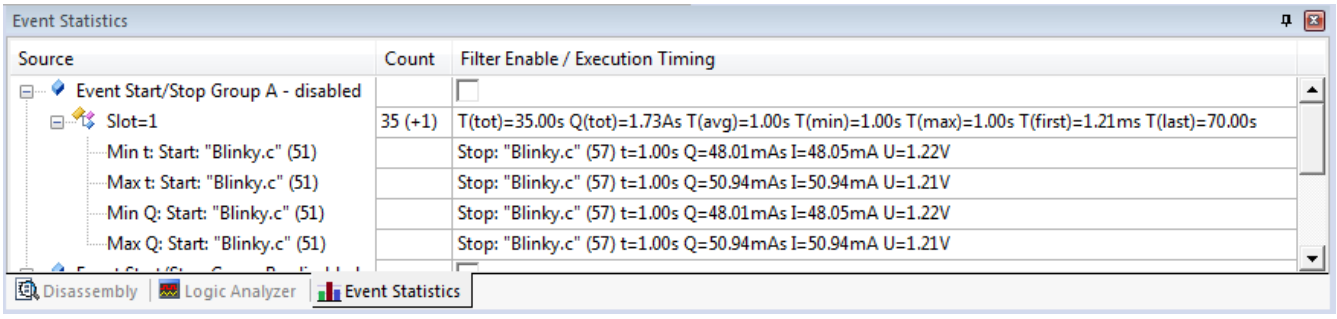
View Event Recorder Window:

1. Open Event Recorder and notice that in addition to the EventRecords ID=2,6,8 you now have EventStart and EventStop frames displayed as shown here:
2. These frames update while the program is running.



View the Statistics View window.

1. Statistics about the program power between the Start and Stop you entered is now displayed below:
2. This is for the specified Group A, Slot 1.
3. You can have many of these events in your program.

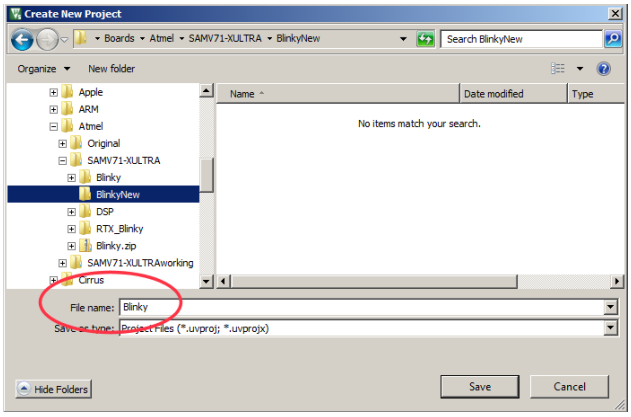


26) Creating your own MDK 5 project from scratch:

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 a project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run your example. It will have 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 as shown on page 26.

Install the Software Pack for your processor:

1. Start µVision and leave it in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Software Pack for your processor must be installed. This has already been done on page 5.
3. You do not need to copy any examples over.

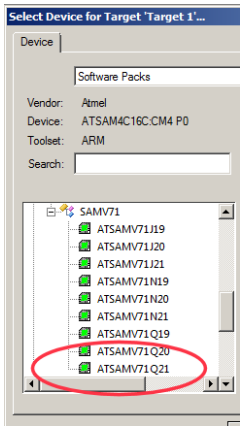


Create a new Directory and a New Project:

1. In the main µVision menu, click on Project/New µVision Project…
2. In the window that opens, go to the folder C:\00MDK\Boards\Atmel\SAMV71-XULTRA
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj.
7. As soon as you click on Save, the next window opens:

Select the Device you are using:

1. Expand Microchip SAMV71and then select ATSAMV71Q21or your processor as shown here:



TIP: Chip icons in colour are from MDK 5 Software Packs. Grey icons are from MDK 4.7x.

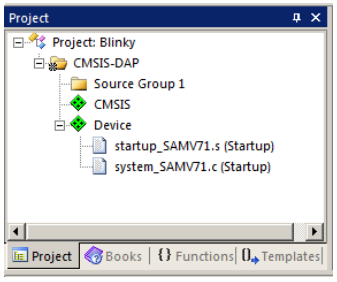
2. Click OK and the Manage Run Time window shown below bottom right opens.

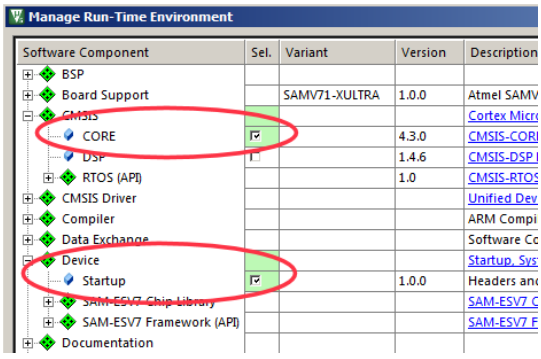
Select the CMSIS components you want:

1. Expand CMSIS and Device. Select CORE and Startup as shown below right. They will be highlighted in Green indicating there are no other files needed. Click OK to close.
2. Click on File/Save All or select the Save All icon: 
3. The project Blinky.uvproj will now be changed to Blinky.uvprojx.
4. You now have a new project list as shown on the bottom left below: The appropriate CMSIS files you selected have been automatically entered and configured into your project for your selected processor.
5. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
6. Click once on it and change its name to CMSIS-DAP and press Enter. The Target selector name will also change.

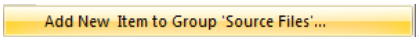
What has happened to this point:

You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files.

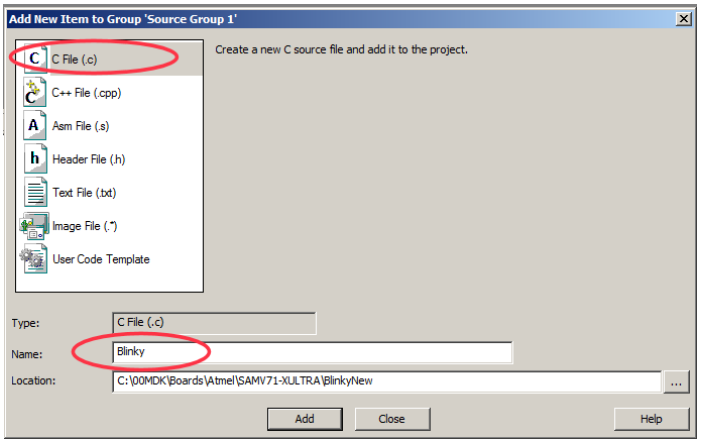




Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select 

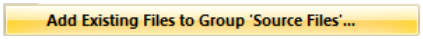
.
2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter Blinky.
5. Click on Add to close this window.
6. Click on File/Save All or 
7. Expand Source Group 1 in the Project window and Blinky.c will now display.
8. It will also open in the Source window.



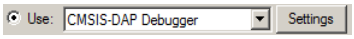
Add Some Code to Blinky.c:

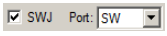
1. In the blank Blinky.c, add the C code below:
2. Click on File/Save All or 
3. Build the files.  There will be no errors or warnings if all was entered correctly.

```
#include "sam.h"                        // Device header
#include "RTE_Components.h"             // Component selection

unsigned int counter = 0;
/*----------------------------------------------
  MAIN function
 *----------------------------------------------*/
int main (void) {

  while(1) {
      counter++;
if (counter > 0x0F) counter = 0;
}
}          //make sure you add a CR Carriage Return or Enter after the last parentheses.
```

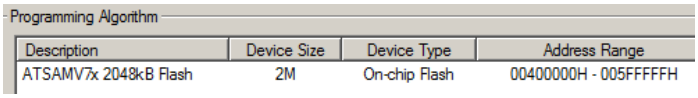
TIP: You can also add existing source files: 

Configure the Target CMSIS-DAP: ***Please complete these instructions carefully to prevent unusual problems…***

1. Select the Target Options icon  . Select the **Target** tab. Note the Flash and RAM addresses are already entered.
2. Click on the **Debug** tab. Select CMSIS-DAP Debugger in the Use: box: 

3. Select Settings: icon beside Use: CMSIS-DAP.
4. Set SWJ and SW as shown here: 

If your board is connected to your PC and Debug USB connector, you should now see a valid IDCODE and Device Name in the SW Device box.
5. Click on OK ***once*** to go back to the Target Configuration window.
6. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm is present: Shown are the correct ones for the SAMV71 Xplained board:
7. Click on OK twice to return to the main menu.
8. Click on File/Save All or 
9. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

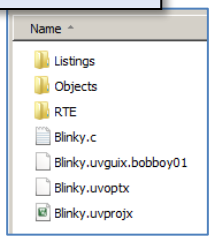


The Next Step ? First we will do a summary of what we have done so far and then….

Let us run your program and see what happens ! Please turn the page….

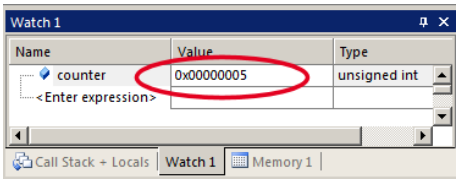
What we have so far ?

1. A MDK 5 project has been created in C:\00MDK\Boards\Microchip\SAMV71-XULTRA \BlinkyNEW
2. The folders have been created as shown here below:
3. RTE contains the CMSIS-Core startup and system files.
4. The Software Pack has automatically pre-configured many items in your new project.



Running Your Program:

1. Enter Debug mode by clicking on the Debug icon . The Flash will be programmed.
2. Click on the RUN icon. **Note:** you stop the program with the STOP icon.
3. Right click on counter in Blinky.c and select Add counter to … and select Watch 1.
4. counter should be updating as shown here:
5. You can also set a breakpoint in Blinky.c and the program should stop at this point if it is running properly. If you do this, remove the breakpoint.
6. You should now be able to add your own source code to create a meaningful project.



TIP: Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip sequential values you know must exist.

Configuring the CPU Clock and Enabling the Caches:

The CMSIS-CORE file system_SAMV71.c contains CPU clock configuration code with a global variable CoreSystemClock to indicate the CPU frequency. We will run the two functions in this file to configure the clock.

1. In Blinky.c, near line 10, add these lines: Add these before the while loop in main().

```
10      SystemCoreClockUpdate ();
11      SCB_EnableDCache();
12      SCB_EnableICache();
```

2. Click on File/Save All or
4. Build the files. There will be no errors or warnings.
5. Enter Debug mode by clicking on the Debug icon . The Flash will be programmed. **Do not click RUN yet !**
6. In Watch 1, double click on Enter Expression and enter SystemCoreClock. Press Enter.
7. Right click on SystemCoreClock and unselect Hexadecimal Display. The value of 4000000 will be displayed. This is the initial CPU clock speed of 4 MHz.
8. Click on the RUN icon. The clock increases to 300 MHz with the execution of the clock files.
9. The Data and Instruction caches are also enabled by calling the two CMSIS functions listed in Step 1.
10. Stop the CPU. and exit Debug mode.

What else can we do ?

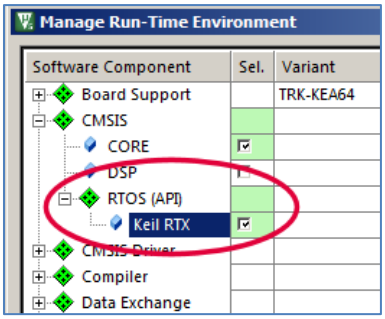
5. You can add your source files using the Add New Item window. See the top of the previous page.
6. You can add existing source files by right clicking on a Group name and selecting Add Existing Item.
7. If you use RTX or Keil middleware, source and template files are provided in the Add New window.
8. Microchip | START can be used to create entire µVision projects. http://start.Microchip.com

27) Creating your own RTX MDK 5 project from scratch:

The MDK Software Packs contains RTX software components. RTX is CMSIS-RTOS compliant.

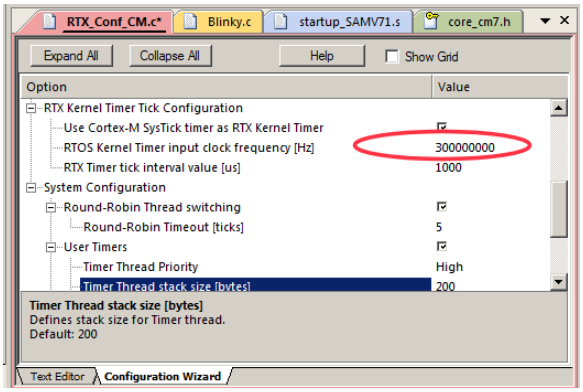
Creating and configuring RTX is easy in MDK 5. These steps use the preceding BlinkyNEW example you constructed.

1. Using the same example from the preceding pages, Stop the program and exit Debug mode.
2. Open the Manage Run-Time Environment window:
3. Expand all the elements as shown here:
4. Select Keil RTX as shown and click OK.
5. Appropriate RTX files will be added to your project. See the Project window.
6. In Blinky.c, right click near line 3 and select Insert '# include file'. Select "cmsis_os.h". This will be added to Blinky.c.
7. Click on File/Save All or



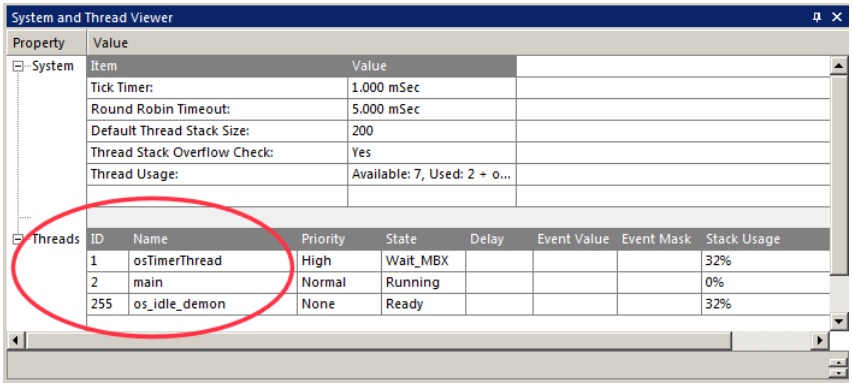
Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.
3. Select the Configuration Wizard tab: Select Expand All.
4. The window is displayed here:
5. Set RTOS Kernel Timer clock value: to 300 MHz as shown:
6. Use the defaults for the other settings.



Build and Run Your RTX Program:

1. Build the files. There will be no errors or warnings.
2. Enter Debug mode: Click on the RUN icon.
3. Select Debug/OS Support/System and Thread Viewer. The window below opens up:
4. You can see three threads: the main thread is the only one running. As you add more threads to create a real RTX program, these will automatically be added to this window.



What we have so far ?

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs. See the next page.
2. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/gsg/. It has very useful information on implementing RTX.
3. **RTX docs are located here:** www.keil.com/pack/doc/CMSIS/RTOS/html/index.html

TIP: The Configuration Wizard is a scripting language as shown in the Text Editor as comments starting such as a </h> or <i>. See www.keil.com/support/docs/2735.htm for instructions to add this to your own source code.

28) Adding a Thread to your RTX_Blinky:

We will create and activate a thread. We will add another variable counter2 to give it something to do.

1. In Blinky.c, add this line near line 6: unsigned int counter2=0;

```
6   unsigned int counter2=0;
```

Create the Thread job1:

2. Add this code to be the thread job1: Add this before the main() function. osDelay(500) delays the program by 500 clock ticks to slow it down so we can see the values of both counter and counter2 increment by 1.

```
8    void job1 (void  const *argument) {
9    for (;;) {
10           counter2++;
11           if (counter2 > 0xf) counter2=0;
12           osDelay(500);
13    }
14  }
```

Add osDelay to main():

3. Add this line just after the if statement near line 27:

```
27    osDelay(500);
```

Define and Create the Thread:

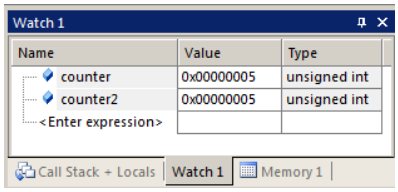
1. Add this line near line 15 just before main():

```
16    osThreadDef(job1, osPriorityNormal, 1, 0);
```

2. Create the thread job1 near line 23 just before the while(1) loop:

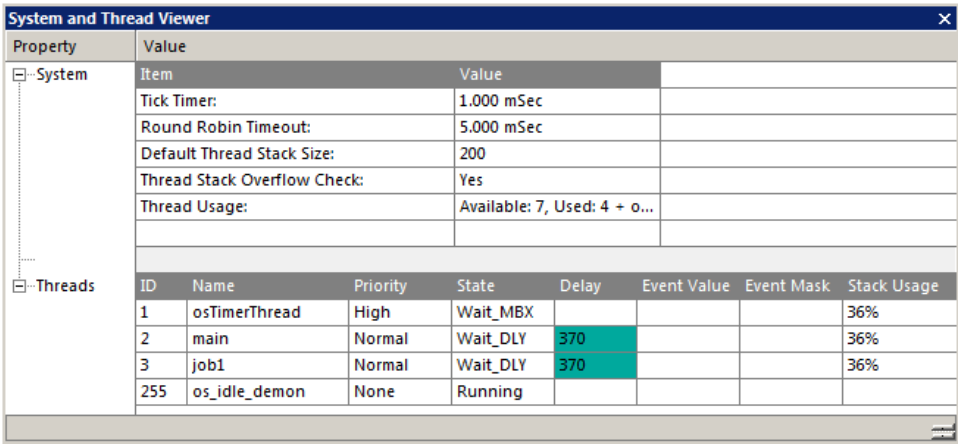
```
26    osThreadCreate(osThread(job1), NULL);
```

3. Click on File/Save All or 
4. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.
5. Enter Debug mode:  Click on the RUN icon. 
6. Right click on counter2 in Blinky.c and select Add counter2 to … and select Watch 1.
7. Both counter and counter2 will increment but slower than before: The two osDelay(500) function calls each slow the program down by 500 msec. This makes it easier to watch these two global variables increment. osDelay() is a function provided by RTX.



8. Open the System and Thread Viewer by selecting Debug/OS Support.
9. Note that job1 has now been added as a thread as shown below:
10. Note os_idle_demon is always labelled as Running. This is because the program spends most of its time there.
11. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
12. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
13. There are many attributes of RTX you can add. RTX help files are located here depending on your CMSIS version: C:/Keil_v5/ARM/Pack/ARM/CMSIS/4.3.0/CMSIS/Documentation/RTX/html/index.html.
14. This concludes the Making Your Own Project. Next is ETM Instruction Trace. You will need a ULINK*pro* for it.

.



29) ETM Trace with ULINK*pro*: Introduction:

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

1) The window containing the trace frames is now called Trace Data. More complete filtering is available.
2) With the ULINK2, SWV (Serial Wire Viewer) data is sent out the 1 bit SWO pin using UART encoding. The ULINK*pro* can send SWV data ***either*** out this same SWO pin using Manchester encoding or through the 4 bit Trace Port. This allows ULINK*pro* to support those Cortex-M processors that have SWV but no Trace Port. The Trace Port is found on the 20 pin Cortex connector and is configured in the Trace configuration window.
   ETM frames are always sent out the Trace Port and if this is the case, SWV data is also be sent out this port.

ULINK*pro* offers:

1) Faster Flash programming than the ULINK2 or EDBG adapters.
2) All Serial Wire Viewer features that ULINK2 provides but at a much faster data throughput.
3) Provides ETM Instruction Trace which provides a record of all executed instructions.
4) The Trace Data window has Trace start and stop, filtering and ability to save records to a file. *(in development)*
5) **Code Coverage**: Were all the assembly instructions executed ? Useful for program certification.
6) **Performance Analysis**: Displays where the processor spent its time in graphical and numeical formats.
7) **Execution Profiling**: How long instructions, ranges of instructions, functions or C source code took in both time and CPU cycles as well as number of times these were executed.

30) Configuring ULINK*pro* ETM Trace:

Configuring the Connection:

The ULINK*pro* was configured for SWV operation using the SWO pin and Manchester encoding on page 14. We will activate ETM trace in the next few pages. We will output the trace frames, including SWV, out the 4 bit Trace Port.

.ini File:

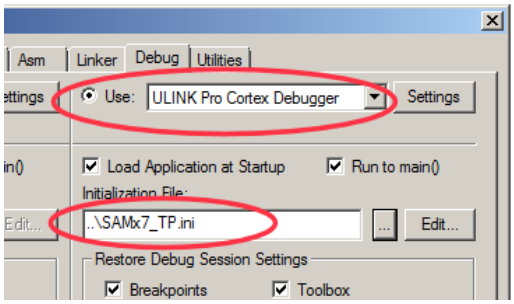
A script must be executed upon entering Debug mode to configure the ETM registers and GPIO ports. This ASCII script is SAMx7_TP.ini and a copy is found in C:\00MDK\Boards\Microchip\SAMV71-XULTRA\ as installed with the examples with this tutorial. This is an ASCII file. This file is specific to SAMV7x processors as their GPIO ports must be configured.

Entering the Initialization File:

1) Refresh the CMSIS-Blinky example program as done on page 6. Save your existing version if you prefer.
2) Select Project/Open Project. Open the file C:\00MDK\Boards\Microchip\SAMV71-XULTRA\Blinky\Blinky.uvprojx.
3) **Create a new Target Options configuration:** Select Project/Manage/Project Items: 

4) Click on the Insert icon:  Enter **ULINKpro ETM** (or something of your choice). Enter and then Close.
5) Select "Ulinkpro ETM: 

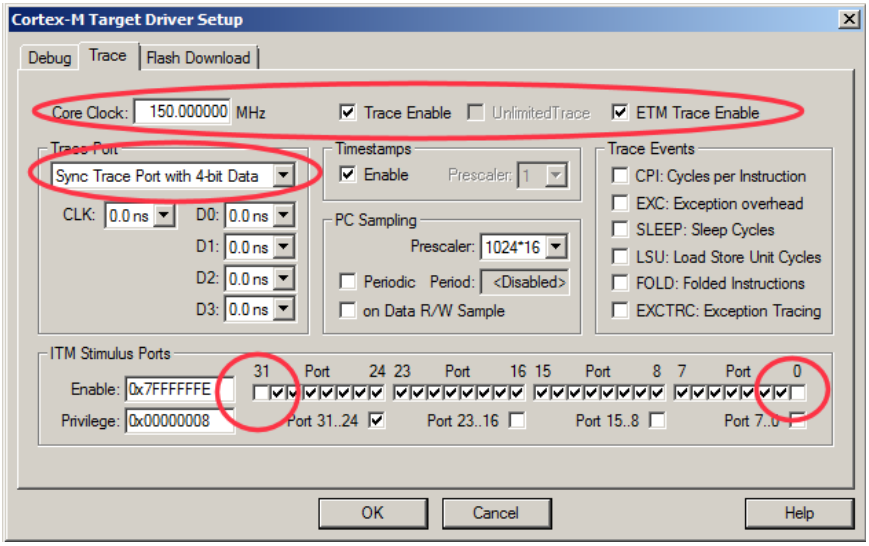
6) Click on the Target Options icon 
7) Click on the Debug tab.
8) Select ULINK Pro Cortex Debugger as shown here:
9) Enter the SAMx7_TP.ini in the Initialization file: box and shown here: Use the browse icon to select it from C:\00MDK\Boards\Microchip\SAMV71-XULTRA\
10) If you click on the Edit icon, TracePort.ini will be opened with the other source files. You can then view and edit it.
11) Leave this Debug tab open for the next page.



The next page describes how to configure ETM.

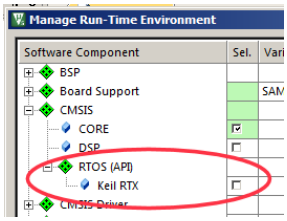
Configuring ETM Trace:

1) These instructions assume you are continuing from the previous page.
2) In the Target Options window left open at the Debug tab from the previous page, click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side of the window.
3) Click on the Trace tab. The window below is displayed.
4) Select Trace Enable.
5) In Core Clock: enter a speed of 150 MHz. µVision uses this to calculate displayed timestamps. This is actually the Trace (TPIU) input clock rather than the actual CPU speed of 300 MHz in this case.
6) In Trace Port select **Sync Trace Port with 4 bit Data** as shown below.
7) Select ETM Trace Enable.
8) Unselect EXCTRC, ITM 0 and 31. Leave everything else at default as shown below.
9) Click on OK twice to return to the main µVision menu. ETM is now configured through the 4 bit Trace Port.
10) Select File/Save All. 



Configure Software Attributes:

1. Open Manage Run-Time Environment: 
2. Unselect Keil RTX as shown here: We do this to simplify our example. µVision can trace all instructions through any RTOS such as RTX no matter how complicated.
3. Click on OK to close Manage Run-Time Environment window.
4. In Blinky.c do these to further simplify our trace window example:
   a. Comment out this line: `#include "cmsis_os.h"` (near line 16).
   b. Comment out both Cache enable function calls just inside the main() function near lines 36 and 37.
   c. There are two lines osDelay(500); in Blinky.c near lines 44 and 57. Comment both of these function calls out. These are part of RTX which has now been disabled. The program will now run very fast.
5. Select File/Save All or  .



The Microchip SAM V7 has JTAG support only for boundary scan use and not for debugging. SWD (Serial Wire Debug) is provided for debugging.

Be aware the five Trace Port pins are normally multiplexed with GPIO pins or other peripherals. You should make appropriate allowances for the use of these shared ports during debugging with ETM trace.

TIP: It is good engineering practice *during system design* to not use those pins shared with ETM for important purposes that will preclude normal operation with ETM enabled.

31) ETM Instruction Trace Configuration Notes:

The high speed of the SAMV7 processors can cause some issues when using ETM Instruction Trace. Here are some hints and other useful information.

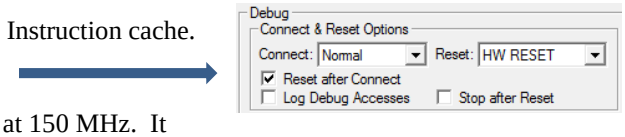
1) If everything is working, when you enter debug mode and before you click RUN, the Trace Data window should contain frames. If not, something is wrong. Exit and reenter Debug mode.
2) It is important to power up the board first, then the ULINK*pro*. Wait for the USB dual-tone from the board before plugging the ULINK*pro* in.
3) Try providing power to the VIN 5 – 14 volts barrel connector rather than through the USB connectors.
4) The external EMAC PHY chip is loading down one of the trace pins. It helps if you hold this device in RESET.
5) You can enter these statements into your code to isolate the PHY chip:

```
PIOC->PIO_OWER = (PIO_PC10);          /* Setup PC10 as ground */

PIOC->PIO_SODR= 0;
```

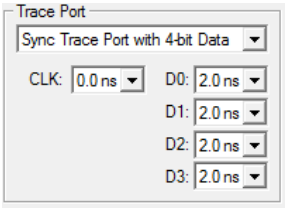
6) Or you can add this to the SAMx7_TP.ini file and they will be set at enter debug mode time:

```
_WDWORD (0x400E1230, 0x00);          //PC10_SODR

_WDWORD (0x400E12A0, 0x400);         //PC10_OWER
```

7) Sometimes the processor will end up in a Hard Fault. Check your Stack Pointer and other hints here. Enter and reenter Debug mode. Repower the board and ULINK*pro* as described in 2) above.
8) If you have a crash, repowering the board and ULINK*pro* is useful to remove any artifacts set in CoreSight module. It can help to have power off for a number of seconds especially if error is: "Trace HW Not Found".
9) If you receive a box stating Processing Trace Data: this sometimes, but not always, means the trace buffer is corrupted. Examine the Trace Data window after the download to see if valid or if the processor is in a Hard Fault.
10) It might help to disable the Cortex-M7 caches, especially the Instruction cache.
11) **RESET:** Try different values under the Debug tab/Settings:



12) The CPU clock runs at 300 MHz and the ETM TRCLK runs at 150 MHz. It may help to slow the CPU down.
13) The CORE Clock: This value is only used by µVision to calculate timings. It does not set the ETM speed. The clocks are provided by ETM or Manchester protocols.
14) The exception is: With ULINK2, ULINKplus and J-Link – for UART operation the Core Clock: must be set correctly or SWV will not work correctly. ETM is not supported by these debug adapters.
15) µVision supports the Segger J-Trace but this has not been tested with Microchip boards.
16) Sometimes the USB cable on the ULINK*pro* has had problems with the high data rates. Try another cable or obtain a high-speed cable. Reports are positive with a UGREEN USB Printer Cable.
17) If you design your own board, pay attention to the proper PCB layout for the ETM signals. Especially TRACECLK.
18) ETM signals (4 TRACEDATA + 1 TRACECLK) are multiplexed with other peripherals. If you are using such a peripheral, you cannot use ETM.
19) If the ETM signals are skewed, you can provide some adjustments under the Debug tab:



If you are still having pron]blems: consult Keil Technical Support.

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

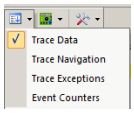
32) Blinky Example: ETM Frames starting at RESET and beyond:

The Blinky project has now been modified on the previous page to provide ETM Trace and the features it provides.

Power up the Xplained board and the ULINK*pro* *in this order*:

a. Both the Xplained board and ULINK*pro* are not powered at this point.
b. Connect the ULINK*pro* 20 pin cable to the CoresSight ETM & SWD connector on the Xplained board.
c. Power the Xplained board using USB cable at TARGET USB connector or VIN connector and then…..
d. ***Finally***, power the ULINK*pro*.
e. If these steps are not followed, you can get Trace: Data Stream Errors and no or few trace frames.

1. Compile the Blinky source files by clicking on the Rebuild icon.  . You can also use the Build icon beside it.
2. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
3. DO NOT CLICK ON RUN YET !!! If you did, simply exit and re-enter Debug mode.
4. Open the Trace Data window by clicking on the small arrow beside the Trace icon as shown here:



5. Examine the Trace Data window as shown below: This is a complete record of all the program flow since RESET until µVision halted the program at the start of main() since Run To main is selected.
6. In this case, the last instruction to be executed is. (BL.W). main In the Register window the PC will display the value of the next instruction to be executed (0x0040 05FC in my case).
7. Click on Single Step once. 

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x00400256	MOV r6,#0x00		__user_setup_stackheap
	X : 0x0040025A	MOV r7,#0x00		__user_setup_stackheap
	X : 0x0040025E	MOV r8,#0x00		__user_setup_stackheap
	X : 0x00400262	MOV r11,#0x00		__user_setup_stackheap
	X : 0x00400266	BIC r1,r1,#0x07		__user_setup_stackheap
	X : 0x0040026A	MOV r12,r5		__user_setup_stackheap
	X : 0x0040026C	STM r12!,{r6-r8,r11}		__user_setup_stackheap
	X : 0x00400270	STM r12!,{r6-r8,r11}		__user_setup_stackheap
	X : 0x00400274	STM r12!,{r6-r8,r11}		__user_setup_stackheap
	X : 0x00400278	STM r12!,{r6-r8,r11}		__user_setup_stackheap
	X : 0x0040027C	MOV sp,r1		__user_setup_stackheap
0.000 226 807 s	X : 0x0040027E	BX lr		__user_setup_stackheap
	X : 0x004001C4	MOV r1,r2		???
0.000 227 173 s	X : 0x004001C6	BL.W __rt_lib_init (0x004001B4)		???
	X : 0x004001B4	PUSH {r0-r4,lr}		???
0.000 227 453 s	X : 0x004001B6	BL.W _fp_init (0x0040065C)		???
	X : 0x0040065C	MOV r0,#0x3000000		_fp_init
	X : 0x00400660	VMSR FPSCR, r0		_fp_init
0.000 227 960 s	X : 0x00400664	BX lr		_fp_init
0.000 228 347 s	X : 0x004001BA	POP {r0-r4,pc}		???
0.000 228 640 s	X : 0x004001CA	BL.W main (0x004005FC)		???

8. The next instruction BL.w will display at 0x0040 05FC which is the first instruction inside main().

0.000 228 640 s	X : 0x004001CA	BL.W main (0x004005FC)		???
0.000 229 113 s	X : 0x004005FC	BL.W LED_GetCount (0x004002E0)	int32_t max_num = LED_GetCount() - 1;	main

9. Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is a LDR as shown below:
10. If you use the Memory window to look at location 0x4, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0040 01E1 - 1 = 0x0040 01E0 (+1 says it is a Thumb instruction). These first instructions after RESET are shown below: Note any source information available is displayed:

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x004001E0	LDR r0,[pc,#36] ; @0x00400208	LDR R0, =SystemInit	Reset_Handler
0.000 000 000 s	X : 0x004001E2	BLX r0	BLX R0	Reset_Handler
	X : 0x00400518	LDR r0,[pc,#188] ; @0x004005D8	SCB->CPACR |= (3UL << 10*2) | ...	SystemInit
	X : 0x0040051A	LDR r0,[r0,#0x00]		SystemInit

TIP: If you double-click on any trace line, this will be highlighted in both the Disassembly and source windows.

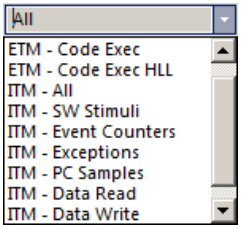
ETM trace provides a powerful tool for finding nasty bugs not easily found any other way. See page 38 for a list.

Finding the Trace Frames you are looking for:

Capturing all the instructions executed is possible with ULINK*pro* but this might not be practical. It is not easy sorting through millions and billions of trace frames or records looking for the ones you want. You can use Find, Trace Triggering, Post Filtering or save everything to a file and search with a different application program such as a spreadsheet.

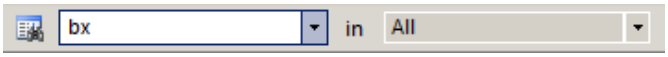
Trace Filters:

In the Trace Data window you can select various types of frames to be displayed. Open the Display: box and you can see the various options available as shown here: These filters are post collection. Future enhancements to µVision will allow more precise filters to be selected.



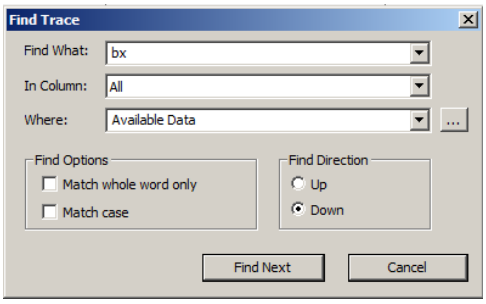
Find a Trace Record:

In the Find a Trace Record box enter bx as shown here:



Note you can select properties where you want to search in the "in" box. All is shown in the screen above

Select the Find a Trace Record icon  and the Find Trace window screen opens as shown here: Click on Find Next and each time it will step through the Trace records highlighting each occurrence of the instruction bx.



Trace Triggering: *coming soon for Microchip Cortex-M7*

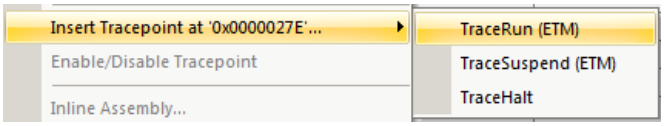
uVision has three trace triggers currently implemented:

TraceRun: Starts ETM trace collection when encountered.

TraceSuspend: Stops ETM trace collection when encountered. TraceRun has to have been encountered for this to have an effect.

These two commands have no effect on SWV or ITM. TraceRUN starts the ETM trace and TraceSuspend stops it.

TraceHalt: Stops ETM trace, SWV and ITM. Can be resumed only with a STOP/RUN sequence. TraceStart will not restart this.



How it works:

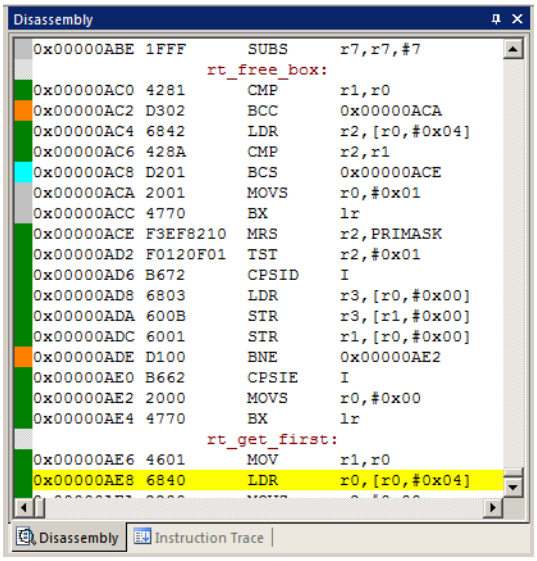
When you set a TraceRun point in assembly language point, ULINK*pro* will start collecting trace records. When you set a TraceSuspend point, trace records collection will stop there. EVERYTHING in between these two times will be collected. This includes all instructions through any branches, exceptions and interrupts.

33) Code Coverage:

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

Colour blocks indicate which assembly instructions have been executed.

1. Green: this assembly instruction was executed.
2. Gray: this assembly instruction was not executed.
3. Orange: a Branch is always not taken.
4. Cyan: a Branch is always taken.
5. Light Gray: there is no assembly instruction here.
6. RED: A hardware breakpoint is set here.
7. The PC: The next instruction to be executed.



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

TIP: Code Coverage is visible in both the disassembly and source code windows. Click on a line in one and this place will be matched in the other.

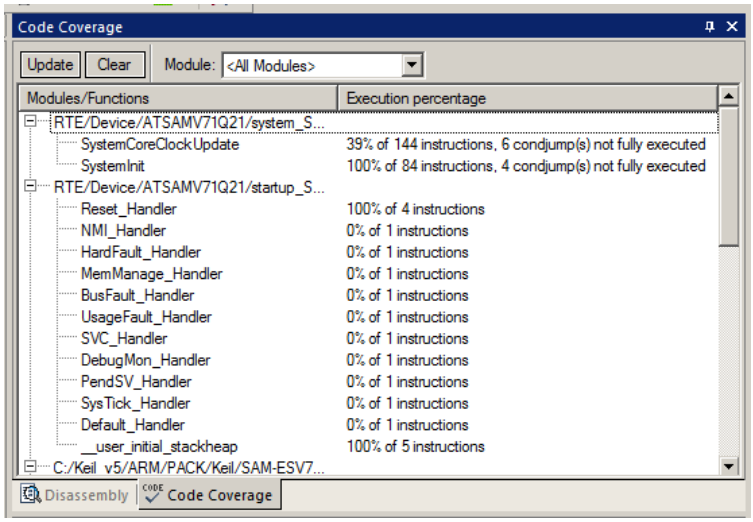
Why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

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

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

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

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage.
The next page describes how you can save Code Coverage information to a file.



Saving Code Coverage information:

Code Coverage information is temporarily saved during a run and is displayed in various windows as already shown.

It is possible to save this information in an ASCII or .gcov file to create a report or for use in other programs.

TIP: To get help on Code Coverage, type Coverage in the Command window and press the F1 key.

You can Save Code Coverage in two formats:

1. In a binary file that can be later loaded back into µVision. Use the command Coverage Save *filename*.
2. In an ASCII file. You can either copy and paste from the Command window or use the log command:
   1) log > c:\cc\test.txt ; send CC data to this file. The specified directory must exist.
   2) coverage asm ; you can also specify a module or function.
   3) log off ; turn the log function off.

1) Here is a partial display using the command **`coverage.`** This displays and optionally saves everything.

```
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
    3 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\Blinky.c\Delay - 100% (9 of 9 instructions executed)
\\Blinky\system_MK60N512MD100.c\SystemInit - 100% (34 of 34 instructions executed)
    2 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\system_MK60N512MD100.c\SystemCoreClockUpdate - 31% (36 of 116 instructions executed)
    5 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\startup_MK60N512MD100.s\__asm_0x27C - 47% (9 of 19 instructions executed)
\\Blinky\startup_MK60N512MD100.s\Reset_Handler - 100% (4 of 4 instructions executed)
\\Blinky\startup_MK60N512MD100.s\NMI_Handler - 0% (0 of 1 instructions executed)
\\Blinky\startup_MK60N512MD100.s\HardFault_Handler - 0% (0 of 1 instructions executed)
\\Blinky\startup_MK60N512MD100.s\MemManage_Handler - 0% (0 of 1 instructions executed)
```

2) The command **`coverage asm produces this listing (partial is shown):`**

```
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
 EX |  0x000002B8 SysTick_Handler:
 EX |  0x000002B8 483D      LDR      r0,[pc,#244]  ; @0x000003B0
 EX |  0x000002BA 6800      LDR      r0,[r0,#0x00]
 EX |  0x000002BC 1C40      ADDS     r0,r0,#1
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
    3 condjump(s) or IT-bcock(s) not fully executed
 EX |  0x000002C4 main:
 EX |  0x000002C4 F04F34FF  MOV      r4,#0xFFFFFFFF
 EX |  0x000002C8 2501      MOVS     r5,#0x01
 EX |  0x000002CA F000F8CB  BL.W     SystemCoreClockUpdate (0x00000464)
```

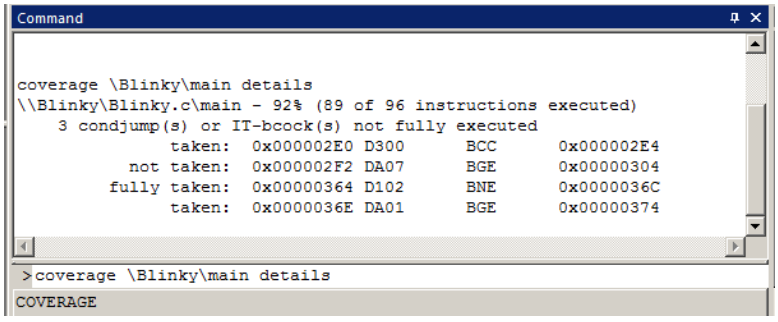
The first column above describes the execution as follows:

NE	Not Executed
FT	Branch is fully taken
NT	Branch is not taken
AT	Branch is always taken.
EX	Instruction was executed (at least once)

3) Shown here is an example using:
`coverage \Blinky\main details`

If the log command is run, this will be saved/appended to the specified file.

You can enter the command **coverage** with various options to see what is displayed.

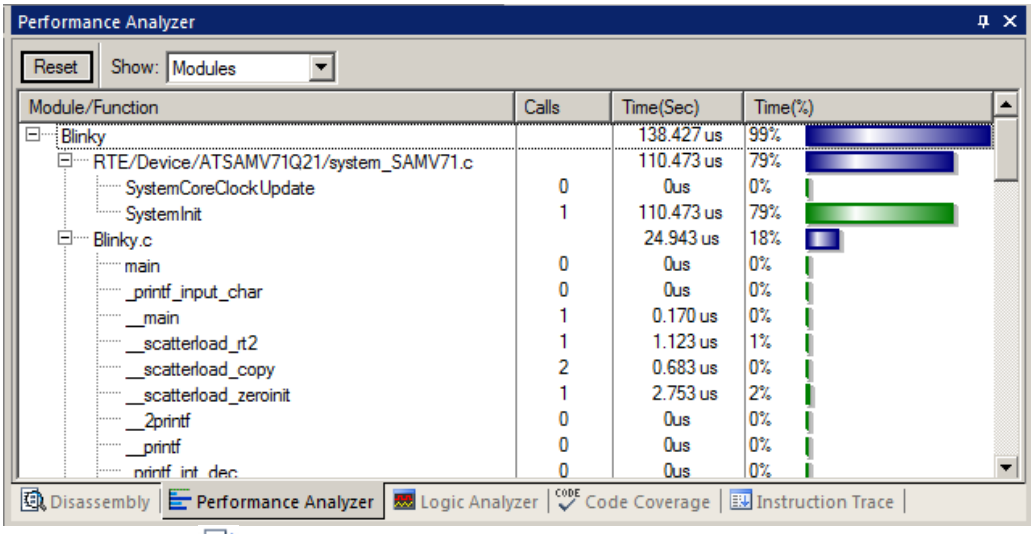


34) Performance Analysis (PA):

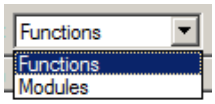
Performance Analysis tells you how much time was spent in each function. It is useful to optimize your code for speed.

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

1. Use the same setup as used with Code Coverage. Do not have the program running.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Click on RESET  . This resets the SAMV7 processor and keeps it at RESET,
4. Click on the RESET  in the Performance Analysis (PA) window as shown below. This clears the PA window.
5. Enter g,main in the Command window to run the program to the beginning of main() in Blinky.c.
6. Do ***not*** click on RUN yet Expand some of the module names as shown below.
7. Times and number of calls has been collected in this short run from RESET to main().



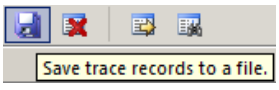
8. Click on the RUN icon. 
9. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the information. No code stubs are needed in your source files. Most time is spent in the Delay function.
10. Select Functions from the pull down box as shown here and notice the difference.



11. Click on the PA RESET icon.  Watch as new data is displayed in the PA window.
12. When you are done, STOP the program  and exit Debug mode. 

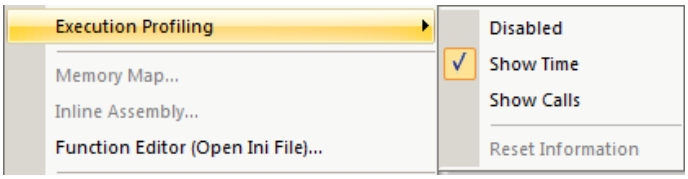
TIP: The Performance Analyzer uses ETM to collect its raw data.

TIP: You can save the Trace data to a file using Save Trace icon in the Trace Data window:



35) Execution Profiling:

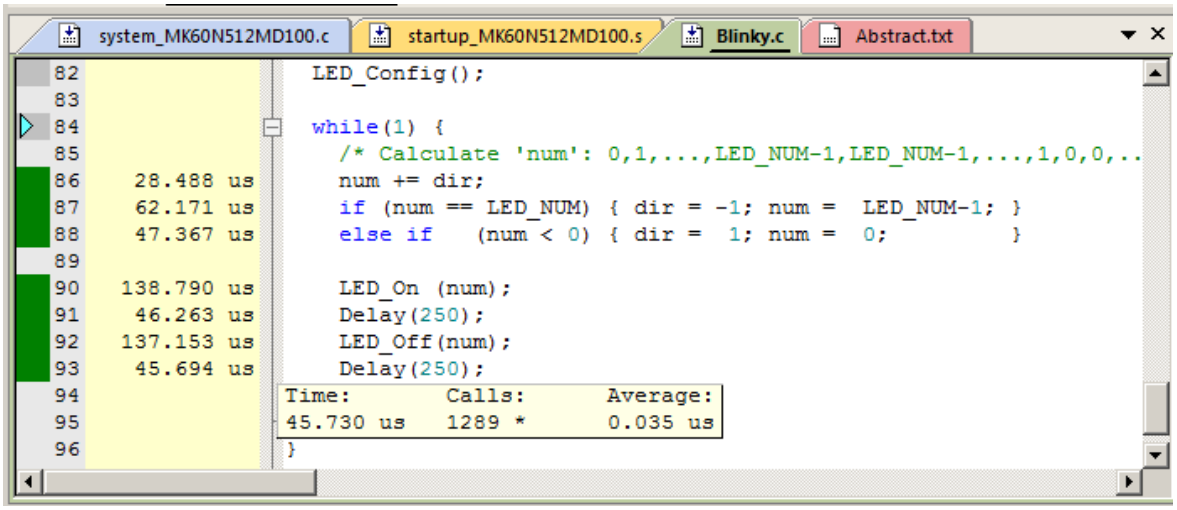
Execution profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.



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

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

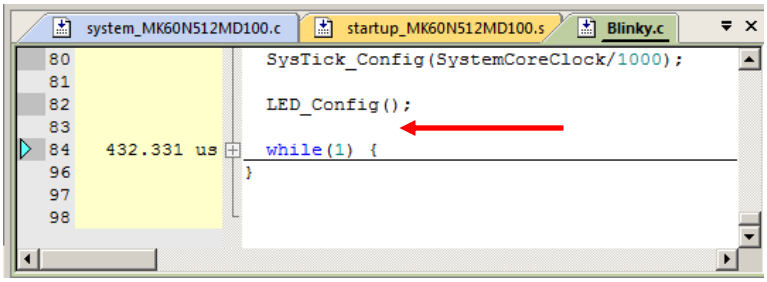
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



Outlining:

Each place there is a small square with a "-" sign  can be collapsed down to compress the associated source files together.

1) Click in the square near the while(1) loop near line 84 as shown here:
2) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
3) Click on the + to expand it.
4) Stop the program  .
5) Exit Debug mode  .



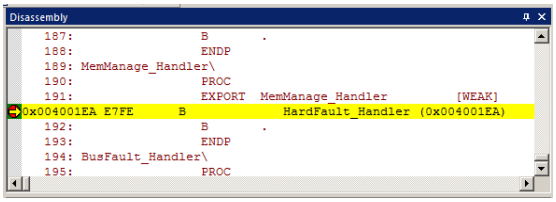
36) In-the-Weeds Example:

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

If a Bus Fault occurs in our example, the CPU will end up at 0x0040 01EA as shown in the disassembly window below. This is the Bus Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. This is not very useful.

This exception vector is found in the file startup_SAMV71.s. If we set a breakpoint by clicking on the Hard Fault handler and run the program: at the next Bus Fault event the CPU will jump to this location.

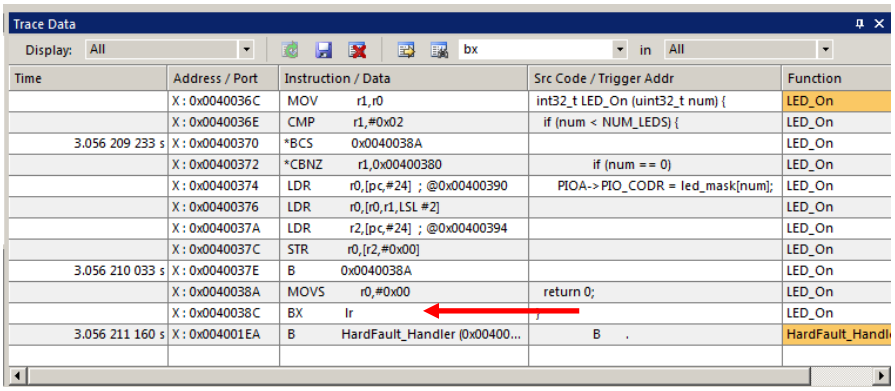
The breakpoint will stop the CPU and also the trace collection. The trace buffer will be visible and extremely useful to investigate and determine the cause of the crash.



1. Use the Blinky example from the previous exercise, enter Debug mode.
2. Locate the Hard fault vector near address 0x0040 01EA in the Disassembly window as shown above or near line 187 startup_SAMV71.s. You are looking for a B instruction with the label HardFault_Handler as shown above.
3. Set a breakpoint at this point. A red circle will appear.
4. Run the Blinky example for a few seconds and click on STOP. Single Step to enter either Led_On or Led_Off function. You can see this in the Call Stack and Locals window.

TIP: You need to enter any function that will process a few instructions and then return: We will put a bogus value (00) into the LR (Link Register) to create a Hard Fault on the next function return.

5. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
6. In the Register window, double-click on R14 (LR) register and set it to zero. LR contains the return address from a subroutine. This is guaranteed to cause a hard fault when the processor tries to execute an instruction at the initial SP location in Flash at 0x00. This will cause a real problem since this is not a real instruction: it contains an address in RAM which is the location of the initial Stack Pointer at RESET.
7. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch always instruction.
8. In the Trace Data window you will find the remainder of the LED function with the BX LR at the end.
9. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.



10. Single Step to run a B at the HardFault Handler which will be displayed at the end of the Trace as shown above. You can see this is the HardFault handler. This is also listed in the Function column.

This is admittedly a contrived example but it clearly shows how quickly and easily ETM trace can show you program flow problems.

These types of problems are very hard to solve and usually take a long time.

37) Serial Wire Viewer and ETM Trace 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.

These are the types of problems that can be found with a quality ETM trace:

- ETM facilitates Code Coverage, Performance Analysis and program flow debugging and analysis.
- 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).
- Pointer problems. Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), a corrupted stack.
  How did I get here ?
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. ETM trace is best for this. Especially where the Stack is corrupted or destroyed and unreliable.
- Communication protocol and timing issues. System timing problems.

38) Document Resources: See www.keil.com/Microchip

Books:

1. ***NEW!*** **Getting Started with MDK 5:** Obtain this free book here: www.keil.com/gsg/
2. There is a good selection of books available on ARM: www.arm.com/support/resources/arm-books/index.php
3. µVision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab. Or select the Cortex-M processor you want in the Processor panel on the left.
5. Or search for the Cortex-M processor you want on www.arm.com.
6. **The Definitive Guide to the ARM Cortex-M0/M0+** by Joseph Yiu. Search the web for retailers.
7. **The Definitive Guide to the ARM Cortex-M3/M4** by Joseph Yiu. Search the web for retailers.
8. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
9. MOOC: Massive Open Online Class: University of Texas: http://users.ece.utexas.edu/~valvano/

Application Notes:

1. ***NEW!*** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with µVision https://launchpad.net/gcc-arm-embedded
8. RTX CMSIS-RTOS Download www.keil.com/demo/eval/rtx.htm
9. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
10. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
11. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
12. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
13. ***NEW!*** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
14. ***NEW!*** ARMv8-M Architecture Technical Overview https://community.arm.com/docs/DOC-10896
15. ***NEW!*** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp
16. Migrating Arm Compiler 5 (AC5) to Arm Compiler 6 (AC6): www.keil.com/appnotes/docs/apnt_298.asp

Useful ARM Websites:

1. CMSIS Standards: https://github.com/ARM-software/CMSIS_5 and www.arm.com/cmsis/
2. ARM and Keil Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content
3. ARM University Program**:** www.arm.com/university. Email: university@arm.com
4. mbed™: http://mbed.org

Keil Direct Sales In USA: sales.us@keil.com or 800-348-8051. **Outside the US:** sales.intl@keil.com
Global Inside Sales Contact Point: Inside-Sales@arm.com ARM **Keil World Distributors:** www.keil.com/distis
Keil Distributors: See www.keil.com/distis/ **DS-5 Direct Sales Worldwide:** orders@arm.com
Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.

39) Keil Products and Contact Information: See www.keil.com/Microchip

Keil Microcontroller Development Kit (MDK-ARM™)

- **MDK-Lite™** (Evaluation version) 32K Code and Data Limit - $0
- *New* **MDK-ARM-Essential™** For all Cortex-M series processors.
- *New* **MDK-Plus™** MiddleWare Level 1. ARM7™, ARM9™, Cortex-M, SecureCore®.
- *New* **MDK-Professional™** MiddleWare Level 2. For details: www.keil.com/mdk5/version520.

For the latest MDK details see: www.keil.com/mdk5/selector/

Keil Middleware includes Network, USB, Graphics and File System. www.keil.com/mdk5/middleware/

USB-JTAG/SWD Debug Adapter (for Flash programming too)

- ULINK2 - *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM: is equivalent to a ULINK2.
- *New* ULINK*plus*- Cortex-M*x* High performance SWV & power measurement.
- ULINK*pro* - Cortex-M*x* SWV & ETM instruction trace. Code Coverage and Performance Analysis.
- ULINK*pro* D - Cortex-M*x* SWV no ETM trace ULINK*pro* also works with ARM DS-5.
- *MDK also supports Microchip SAM-ICE, Microchip-ICE, Microchip EDBG, CMSIS-DAP and Segger J-Link.*

- ***For special promotional or quantity pricing and offers, please contact Keil Sales. In USA, 1-800-348-8051. For rest of the world: +49 89/456040-20 or www.keil.com/distis/***

Keil RTX RTOS is now provided under a Berkeley BSD or Apache 2.0 license. This makes it free.

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

Keil provides free DSP libraries for the Cortex-M0, Cortex-M3 Cortex-M4 and Cortex-M7.

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

All products are available from stock.

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

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

Keil supports many other Microchip processors including ARM9™ and 8051. See www.keil.com/Microchip for the complete list of Microchip support.

For Linux, Android and bare metal (no OS) support on Microchip Cortex-A processors such as SAMA5D3**,** please see DS-5 or DS-MDK: www.arm.com/ds5.

For more information:

Sales In Americas: sales.us@keil.com or 800-348-8051. **Europe/Asia:** sales.intl@keil.com +49 89/456040-20

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

Global Inside Sales Contact Point: Inside-Sales@arm.com ARM **Keil World Distributors:** www.keil.com/distis

Forums: www.keil.com/forum http://community.arm.com/groups/tools/content https://developer.arm.com/

For more Microchip specific information, visit www.keil.com/Microchip.

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

arm KEIL