

The latest version of this document is here: www.keil.com/apnotes/docs/apnt_314.asp

Introduction:

The purpose of this tutorial is to introduce you to the Microchip SAM L11 Cortex®-M23 processor using the Arm® Keil® MDK toolkit featuring the IDE µVision®. We will demonstrate all debugging features available on this processor including those pertaining to Arm TrustZone® Security Technology. At the end of this tutorial, you will be able to confidently work with these Arm processors and Keil MDK. www.keil.com/Microchip

Armv8-M Architecture Technical Overview: www.keil.com/apnotes/files/apnt_291.pdf

Getting Started MDK 5: www.keil.com/gsg. **Arm Compiler 6:** www.keil.com/apnotes/docs/apnt_298.asp

Keil MDK supports and has examples for most Microchip (and Atmel) Arm processors. Check the Keil Device Database® on www.keil.com/dd2. This list is also provided by the µVision Pack Installer utility. See www.keil.com/Microchip.

Many Microchip 8051 processors are supported by Keil. www.keil.com/dd/chips/atmel/8051.htm

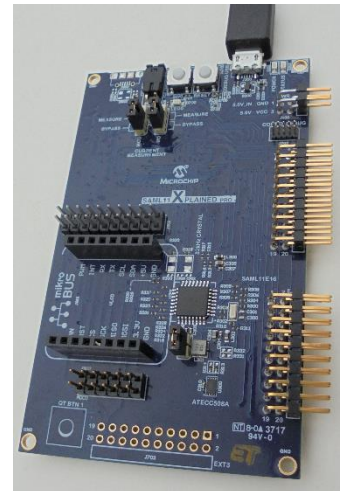
Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. No license is required for SAM L11.

RTX RTOS: All variants of MDK contain the full version of RTX with Source Code. 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 is also supported.

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 (AC5) and Arm Compiler 6 (AC6) (LLVM) are included.
3. GCC is supported and available on Developer.arm.com.
4. Dynamic Syntax checking on C/C++ source lines.
5. **Compiler Safety Certification Kit:** www.keil.com/safety/
6. TÜV certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
7. RTX RTOS Safety Certification will be available 2018. www.keil.com/safety/rtos
8. MISRA C/C++ support using PC-Lint. www.gimpel.com
9. **Keil Middleware:** Network, USB, Flash File and Graphics.
10. **NEW!** Event Recorder for Keil RTX RTOS and User programs.
11. CoreSight™ Serial Wire Viewer (SWV) for most Cortex-M processors.
12. ETM Instruction Trace: For some Cortex-M processors. Includes Code Coverage and Performance Analyzer. Consult your device datasheet availability.
13. Debug Adapters: On-board Microchip EDBG (CMSIS-DAP), Keil ULINK™2, ULINKplus, ULINKpro and J-Link.
14. Affordable perpetual and term licensing with support. Contact Keil sales for pricing options. Inside-Sales@arm.com
15. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
16. ULINKplus power analysis: www.keil.com/mdk5/ulink/ulinkplus/ Contact Keil sales.



This document includes details on these features plus more:

1. The sequences used to call a Secure function from the Non-secure state.
2. Real-time Read and Write to memory locations for the Watch, Memory and Peripheral windows.
3. Four Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
4. **NEW!** µVision Event Recorder. You can use this in your own programs. See pages 13 and 19.
5. printf using Event Recorder (EVR). No UART is required.
6. **NEW!** Power Measurement using Keil ULINKplus. <http://www.keil.com/ulinkplus>

General Information:

1. Microchip Evaluation Boards & Keil Evaluation Software:	3
2. MDK 5 Keil Software Information: MDK 5.26 or later:	3
3. Debug Adapters Supported:	3
4. CoreSight Definitions:	4

Keil Software and Software Packs:

5. Keil MDK Software Download and Installation:	5
6. μ Vision Software Pack Download and Install Process:	5
7. Install the L11 example	5
8. Other features of Software Packs:	6

Blinky Example using the Microchip Cortex-M23 SAM L11

9. Keil example using the Microchip Cortex-M23 SAM L11:	7
10. Review: Stepping from Non-Secure State to Secure State and back:	8

Arm CoreSight Debugging Features supported by Keil μ Vision:

11. Add some Useful Example Code:	9
12. Hardware Breakpoints and Single Stepping:	9
13. Call Stack & Locals window:	10
14. Watch and Memory windows and how to use them:	11
15. printf using Event recorder (no UART required)	12

***NEW !* Power Measurement using Keil ULINKplus:**

16. Power Measurement using Keil ULINKplus: Connecting to the Target SWD:	14
17. Connecting ULINKplus power measurement connections to Target SAM L11:	15
18. Displaying Power measurement data in System Analyzer Window:	16
19. Annotating Your Code to Display in System Viewer:	17
20. Measuring Power Consumption with Event Recorder Statistics:	18
21. Creating a New Options for Target to add a new Debug Adapter: <i>for reference:</i>	19

Other Useful Information:

22. Issues Encountered in Creating this Document:	20
23. Document Resources:	21
24. Keil Products and Contact Information:	22

1) Microchip Evaluation Boards & Keil Evaluation Software: www.keil.com/Microchip

Keil MDK provides board support for many Microchip Cortex-M processors that were previously offered by Atmel. See www.keil.com/Microchip for the complete list.

On the second last page of this document is an extensive list of resources that will help you successfully create your projects. This list includes application notes, books and labs and tutorials for other Microchip Arm boards.

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

Migrating from Arm Compiler 5 (AC5) to Arm Compiler 6 (AC6): www.keil.com/appnotes/docs/apnt_298.asp

ARM forums: <https://developer.arm.com> **Keil Forums:** www.keil.com/forum/

Programming the Microchip SAM L11 Cortex-M23: www.keil.com/appnotes/docs/apnt_315.asp

Microchip SAM L10 Tutorial: www.keil.com/appnotes/docs/apnt_313.asp

2) MDK 5 Keil Software Information: *This document uses MDK 5.26 or later.*

MDK 5 Core is the heart of the MDK toolchain. This will be in the form of MDK Lite which is the evaluation version. The addition of a Keil license will turn it into one of the commercial versions available. Contact Keil Sales for more information.

Device and board support are distributed via Software Packs. These Packs are downloaded from the web with the "Pack Installer", the version(s) selected with "Select Software Packs" and your project configured with "Run Time Environment" (RTE). These utilities are components of μ Vision.

A Software Pack is an ordinary .zip file with the extension changed to .pack. It contains various header, Flash programming and example files and more. Contents of a Pack are described by a .pdsc file in XML format. You can make your own Pack.

See www.keil.com/dd2/pack for the current list of available Software Packs. More Packs are always being added.

Example Project Files: This document uses examples provided outside of the Pack and are distributed with this document.

3) Debug Adapters Supported:

These are listed below with a brief description. Configuration instructions start on page 7.

1. **Microchip EDBG CMSIS-DAP:** Many Xplained boards contain EDBG: an on-board debug adapter that is CMSIS-DAP compliant. You do not need an external debugger such as a ULINK2 to do this lab. The SAM L11 does not have any feature such as SWV or ETM that requires a specific adapter. All other CoreSight functions found on SAM L11 are provided by EDBG CMSIS-DAP.
To add CMSIS-DAP to a custom board. See https://github.com/ARM-software/CMSIS_5.

This tutorial uses only the on-board EDBG CMSIS-DAP. For the SAM L11, this is all you need. The Keil ULINK $plus$ will provide power measurement for SAM L11. See below.

2. **ULINK2 and ULINK-ME:** ULINK-ME is only offered as part of certain evaluation board packages. ULINK2 can be purchased separately. These are electrically the same and both support Serial Wire Viewer (SWV), Run-time memory reads and writes for the Watch and Memory windows and hardware breakpoint set/unset on-the-fly.
3. **ULINK pro :** ULINK pro supports all SWV features and adds ETM Instruction Trace. ETM records all executed instructions. ETM provides Code Coverage, Execution Profiling and Performance Analysis features. ULINK pro also provides the fastest Flash programming times. Not all SAM processors have ETM. Consult your datasheet.
4. **NEW ! ULINK $plus$:** Power Measurement + High SWV performance and Test Automation.
See www.keil.com/ulink/ulinkplus/ for details.

External J702 CORTEX DEBUG connector:

An external debug adapter can be connected to the J702 Cortex Debug 10 pin connector.

J702 is a 10 pin CoreSight standard connector. For pinouts search the web for "Keil connectors". A special 10 to 20 cable is provided with ULINK pro and ULINK $plus$. ULINK2 and ULINK-ME will connect to J702.


Contact Segger for a special adapter board for the J-Link series.

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

Not all processors have all features. Cortex-M0/M0+/M23 do not have SWV, ITM or ETM trace. They have DAP read/write. Cortex-M3/M4/M7/M33 can have all or most of these features listed implemented. MTB may be found on certain Cortex-M0+. Consult your specific datasheet.

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 except Boundary Scan is not possible. SWD is referenced as SW in the μ Vision Cortex-M Target Driver Setup. The SWJ box must be selected in ULINK2/ME or ULINK pro . Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDO shares the same pin as SWO. The SWV data normally comes out the SWO pin or Trace Port.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **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 the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. μ Vision uses the DAP to update Memory, Watch, Peripheral and RTOS kernel awareness windows while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed. You do not need to configure or activate DAP. μ Vision configures DAP when you select a function that uses it. Do not confuse this with CMSIS_DAP which is an Arm on-board debug adapter standard.
5. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
6. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
7. **Trace Port:** A 4 bit port that ULINK pro uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by μ Vision, 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. μ Vision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK pro provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis. ETM is output on the Trace Port or accessible in the ETB (ETB has no Code Coverage or Performance Analysis).
10. **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 datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal user RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. Cortex-M3/M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 usually 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. The CPU is halted before the instruction is executed.
13. **Watchpoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can 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. There also referred to as Access Breaks in Keil documentation.

Read-Only Source Files:

Some source 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 files. This can cause difficult to solve problems. These files normally need no modification. μ Vision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

If you need to modify one, you can use Windows Explorer to modify its permission.

1. In the Projects window, 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 μ Vision editor.
6. It is a good idea to make the file read-only when you are finished modifications.

5) Keil MDK 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 tutorial 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. This tutorial requires a Keil ULINK_{pro} debug adapter.
5. You need a Keil MDK Pro (for Secure and Non-Secure) or MDK Essential (for Non-Secure) license for Armv8-M processors and this tutorial. You can get an instant 7 day license as found under File/License Management. Otherwise, for evaluation purposes, contact Keil Sales for a temporary license or purchase: Inside-Sales@arm.com
6. MDK uses only Arm Compiler 6 (LLVM based) for Armv8-M TrustZone processors.

Download MDK-Core Version 5

6) SAM L11 Software Pack Download and Install Process:

A Software Pack contain components such as header, Flash programming, documents and other files used in a project. The Pack is easily downloaded from a webserver using the μ Vision utility Pack Installer.

1) Start μ Vision and open Pack Installer and install The SAM L11 Software Pack:

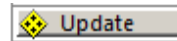
1. Start μ Vision:  Connect your PC to the Internet.
2. Open the Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.
3. This window opens up:
4. Select the Boards tab. Enter L11 in the Search: box.
5. Highlight SAML11-XPRO as shown.
6. Select the Packs tab. Keil::SAML11_DFP is shown.
7. Select Install opposite the SAML11 DFP.
8. The latest Pack is installed into μ Vision.

TIP: You can select your processor with the Device tab.

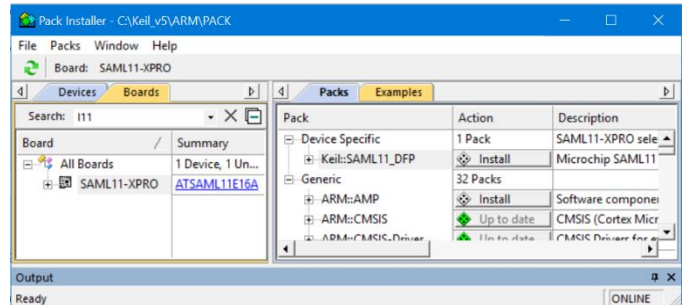
9. The Pack status is indicated by the “Up to date” icon:




10. Update means there is an updated Software Pack available for download.

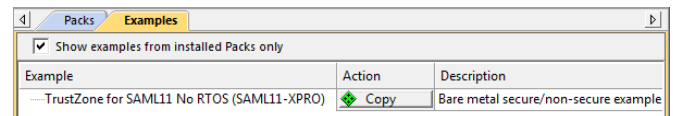


TIP: The left hand pane filters the selections displayed on the right pane. You can start with either Devices or Boards.




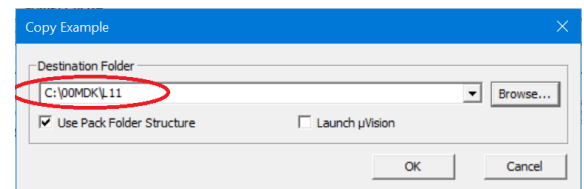
7) Install the L11 Example:

1. Select the Examples tab. The example is displayed:
2. Select Copy:  **Copy**
3. The Copy Example window shown below right opens:
4. For Destination Folder, enter C:\00MDK\L11. Unselect Launch μ Vision.
5. Click OK.
6. This puts the folder at the top of your tree at C:\00MDK\L11\mdk\examples\TrustZone_noRTOS\.



TIP: For simplicity, we will use the default folder of C:\00MDK\L11 in this tutorial. You can use any folder you prefer.


7. Close Pack Installer. You can open it any time by clicking on its icon.  If a dialog box says the Software Packs folder has modified, select Yes.

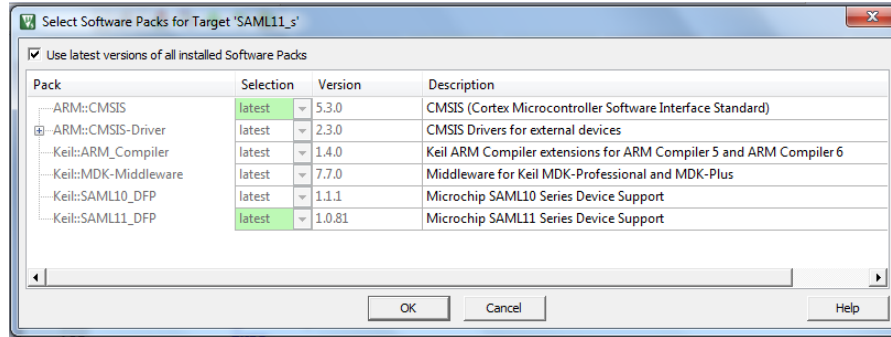



8) Other features of Software Packs: This page is for reference:

1) Select Software Pack Version:

This µVision utility provides the ability to choose among the various software pack versions installed in your computer.


1. Open the Select Software Pack by clicking on its icon: 
2. This window opens up. Note **Use latest versions ...** is selected. The latest version of the Pack will be used.
3. Unselect this setting and the window changes to allow various versions of Packs to be selected.

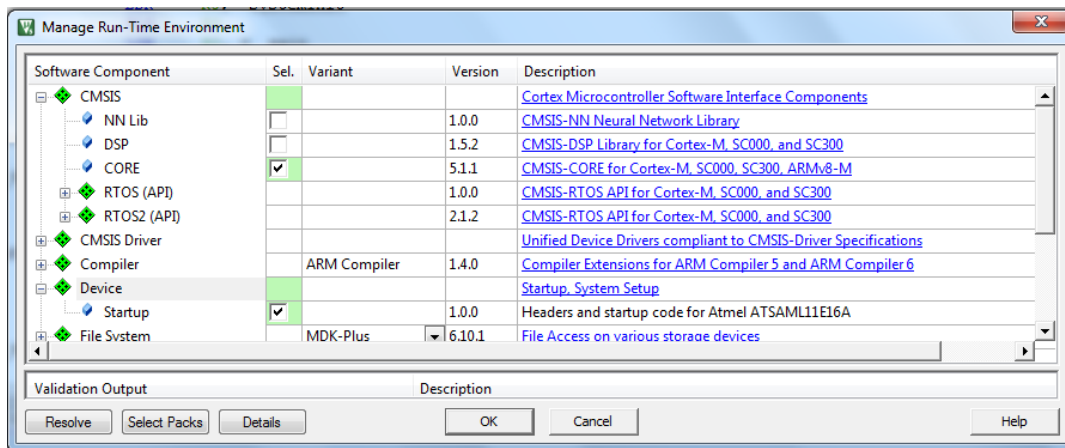


4. Note various options are visible as shown here: 
5. Select excluded and see the options as shown:
6. Select Use latest versions... Do not make any changes.
7. Click Cancel to close this window to make sure no changes are made.



2) Manage Run-Time Environment (MRTE):


1. From the main µVision menu, select Project/Open Project.
2. Open the project: C:\00MDK\L11\mdk\examples\TrustZone_noRTOS\NoRTOS.uvmpw.
3. Click on the Manage Run-Time Environment (MRTE) icon:  The window below opens:
4. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project.
5. Do not make any changes. Click Cancel to close this window.




TIP: µVision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

9) Keil example program using the Microchip SAM L11 board:

Now we will connect a Keil MDK development system using the SAM L11 board.



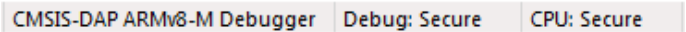
1. Connect a USB cable between your PC and the SAM L11 board J200 DEBUG USB connector.
2. The board will be powered by this USB connection. The green POWER D201 LED will illuminate.
3. Start μ Vision by clicking on its desktop icon if it is not already running. 

Open and Build the μ Vision Multi—Project:



1. Select Project/Open Project.
2. Select the file NoRTOS.uvmpw located in: C:\00MDK\L11\mdk\examples\TrustZone_noRTOS\
3. Highlight this project and click Open.
4. This is a μ Vision multi-project consisting of a Secure and a Non-Secure projects. They are named sApp and nsApp.
5. Compile the source files by clicking on the Batch Build icon. . Make sure you **do not** use Build or Re-Build.
6. The Build Output window will display: 2 succeeded. Ignore any warnings.
7. If you get an error here stating ARM compiler does not support Cortex-M23, you need to obtain a MDK Professional license. See page 5 for options.

TIP: Indications of Secure or Non-Secure code or states is indicated by the letters S, NS, _S, _NS in μ Vision, source code or documentation. Sometimes lowercase letters are used.



Run the Example:

1. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed.
2. The program will run to the start of the main() function in main_s.c.
3. Select the Registers tab:  You can see Secure is **bold**. The CPU is now in Secure mode.
4. This is also visible at the bottom right: 

TIP: Debug: Secure means the μ Vision debugger has access to both Secure and Non-Secure modes, memory and peripherals. This can be changed so μ Vision can see only Non-secure attributes or also to be able to access nothing.

5. Open the file main_ns.c by clicking on it in Projects or selecting its tab. Make sure you select **_ns.c** and **not _n.c** !
6. Set a breakpoint on the start of main() in main_ns.c near line 31. Click Run .
7. The program will run from the Secure main() to the start of the Non-Secure main() in main_ns.c. This will be displayed in the main_ns.c source window.
8. The CPU is now in Non-Secure mode. 
9. In the Registers window, Non-Secure will be **bold**.

Stepping from Non-Secure to Secure and back to Non-Secure States:

1. Note there are two calls to functions secure_func1 and secure_func2 near line 34 and 37.
2. These two functions are in Secure memory area.
3. Since we are now in Non-Secure mode, we will demonstrate how Non-Secure code can call Secure functions. There are specific methods to do this. Any deviation from these steps will result in a Hard Fault.
4. Click on Step (F11) twice. 
5. The program enters the Non-secure Callable Function in interface.c. The CPU will now be in Secure mode.
6. You can view the state in either the Registers window or the box at the bottom right of μ Vision.
7. Click on Step (F11)  until you return to main_ns.c at the start of the call to val2 = secure_func2 (7);
8. Step though val2 = near line 37 to see the same sequence. You will eventually end up in the while(1); loop at line 39.

This is how a Non-Secure code can call code in the Secure memory area. Details will follow.

Now you know how to compile a program, program it into the SAM L11 processor Flash, run it and stop it !

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


10) Review: Stepping from Non-Secure State to Secure State and back:

This section shows how a Secure function can be called from the Non-Secure state and its return to Non-Secure.




The sequence used is:

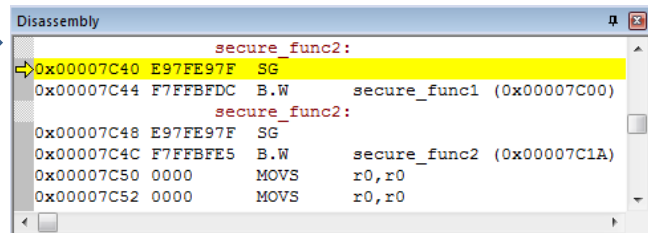
1. **Start in Non-Secure state.**
2. Branch to the veneer.
3. The instruction found here must be a SG (Secure Gate) else a Hard Fault will result.
4. **Enter the Secure State.**
5. Execute any user code provided.
6. Execute a BXNS LR instruction using Link Register.
7. **Enter the Non-Secure state.**

Calling A Secure Function from the Non-Secure State Demonstration:

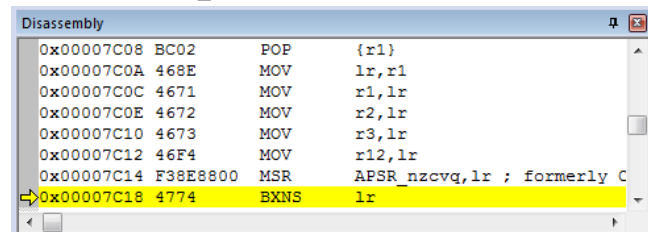
1. RESET the SAM L11 processor: 
2. **CPU is in IN SECURE STATE at RESET by design:**
3. Click Run and the program will halt on the breakpoint previously set on the start of the Non-Secure main().
4. **CPU ENTERED NON_SECURE STATE:**
5. At the bottom right note the CPU is in Non-Secure state: **CPU: Non-Secure**
6. Note the two function calls secure_func1 and secure_func2 near lines 35 and 38.
7. Normally the NS state is unable to access anything in the Secure (S) state which includes these two functions.
8. Select somewhere in the Disassembly window to bring it into focus. We want to step by instruction and not C line.

TIP: It is vital to keep the Disassembly window in focus. Otherwise you will start to step in C source. If this happens, click on RESET and then Run and start over.

9. Step  until you get to val1 = secure_func(1); near line 35.
10. Note the address of the instruction at start of call to in secure_func(1); is 0x0000 819E. This is Non-Secure memory.
11. Step until you get to the branch (BL) to secure_func1. 
12. Step one more time and you will be at the SG (Secure Gate) instruction: This is the veneer and you can see the SG for both of the functions.
13. **CPU ENTERED SECURE STATE:** 
14. Step and execute the SG instruction and note the CPU is now in the Secure state. **CPU: Secure**
15. The B.W branches to the secure_func1.
16. Step until you come to the BXNS (Branch and Exchange Non-secure) instruction as shown below:
17. This uses the LR register to return to Non-Secure.
18. Click Step one time and you will be back at the end of the call to secure_func1.
19. **CPU ENTERED NON_SECURE STATE:**
20. Note the CPU is back in Non-Secure mode: **CPU: Non-Secure**
21. You can step the same way through the call to func2 and back up to func1 again.



```
Disassembly
secure_func2:
0x00007C40 E97FE97F SG
0x00007C44 F7FFBFD C B.W secure_func1 (0x00007C00)
secure_func2:
0x00007C48 E97FE97F SG
0x00007C4C F7FFBFE5 B.W secure_func2 (0x00007C1A)
0x00007C50 0000 MOV r0,r0
0x00007C52 0000 MOV r0,r0
```





```
Disassembly
0x00007C08 BC02 POP {r1}
0x00007C0A 468E MOV lr,r1
0x00007C0C 4671 MOV r1,lr
0x00007C0E 4672 MOV r2,lr
0x00007C10 4673 MOV r3,lr
0x00007C12 46F4 MOV r12,lr
0x00007C14 F38E8800 MSR APSR_nzcvq,lr ; formerly C
0x00007C18 4774 BXNS lr
```

Later versions of this document will illustrate many more features of µVision and Armv8-M processors.





11) Add Some Interesting C Code:

We will add a few lines of code to be used in some experiments.

Add the New C Statements:



1. Stop the program  and exit Debug mode .
2. Click on the main_ns.c tab to bring it into focus. It will be underlined.
3. At the start of main() and near line 32, add this statement: while(1){
4. After the val2 function call near line 38, add this statement: val1 = val2 = 0;
5. In the next line near 39, add this curly brace: }
6. You can comment out the while(1); near line 40 as it will not be used any longer.

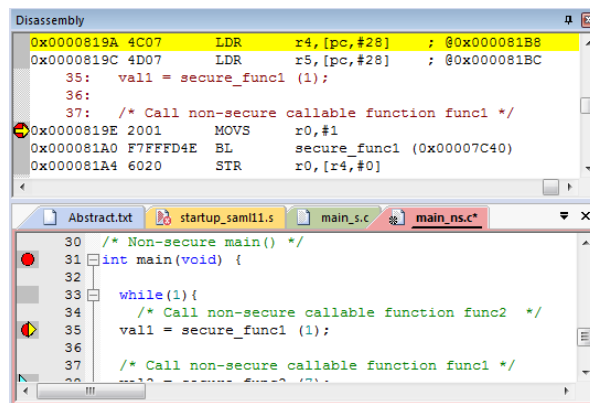
Build the Code and RUN:

1. Select File/Save All or click .
2. Build the source files by clicking on the Batch Build icon. .
3. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed.
4. Click on RUN  to get to main() in main_ns.c using the breakpoint you created on the previous page.
5. The program will now continually run. No LEDs blink in this example program. Continue with these exercises.

12) Hardware Breakpoints:

The SAM L11 has four hardware breakpoints that can be set or unset on the fly while the program is running. Arm hardware breakpoints are no-skid – that is, they do not execute an instruction they are set to when it is encountered.

1. With the TrustZone program running, in the main_ns.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 between near lines 33 through 39 is a good place: You can also click in the Disassembly window 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. With main_ns.c in focus (is underlined in its tab), click on Step  or F11 to enter secure_func1 or secure_func2 near lines 33 or 40 in interface.c.
5. Set a second breakpoint in one of these functions.
6. Every time you click on the RUN icon  the program will run until a breakpoint is again encountered.
7. One breakpoint is in the Non-Secure code and the other is in the Secure code area. This will be displayed in µVision.
8. You can see the CPU mode change as the program runs.
9. The yellow arrow is the current program counter value.
10. 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:
11. Open Debug/Breakpoints or Ctrl-B to see any breakpoints set. You can temporarily unselect them or delete them.
12. Delete all breakpoints *except* the one at the start of main(). Use the Kill Selected button.
13. Close the Breakpoint window if it is open.
14. You can also delete the breakpoints by clicking on the red circle.




TIP: If you set too many breakpoints, µVision will warn you.

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


Single-Stepping Notes:

1. The program will be stopped by one of the breakpoints you have set.

By Assembly Instruction:

2. Click on the top margin of the Disassembly window to bring it into focus. Clicking Step  or F11 advances the program counter one assembly instruction at a time.

By C/C++ Source Lines:

3. Note which C source line is in focus.
4. Click on the Step icon  or F11 a few times: You will see the program counter jumps a C line at a time. The yellow arrow indicates the next C line to be executed.


13) Call Stack + Locals Window:

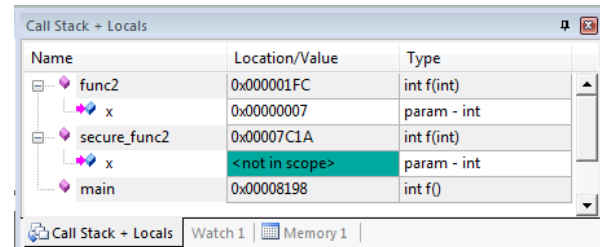
Local Variables:

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 or thread.

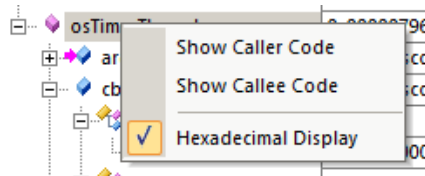
If possible, the values of the local variables will be displayed and if not the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

1. Click on the Call Stack + Locals tab if necessary to open it. Expand some of the entries.
2. The Call Stack + Locals window will show func2, secure_func2 and main(): Local variables are displayed.

3. Click on the Step icon  to enter a few functions.
4. As you click on Step In, you can see the program entering and perhaps leaving various functions. Note the local variables are displayed.
5. The functions as they were called are displayed. If these functions had local variables, they are displayed. If the functions are in scope, their values are displayed.



6. This program is very simple. Call Stack + Locals is very useful in sorting out stack issues in more complicated programs.
7. Right click on a function and select either Callee or Caller code and this will be highlighted in the source and disassembly windows.



8. When you ready to continue, remove all the hardware breakpoints except for the one at NS main() by clicking on each 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 only 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 because locals are normally kept in a CPU register. These cannot be read by the debugger while the program is running. Any local variable values are visible only when they are in scope.

If you need to monitor any variables, make it static or global and enter it in a Watch or Memory window.


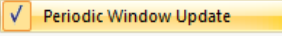
Do not forget to remove any hardware breakpoints except the one on Secure main() before continuing.

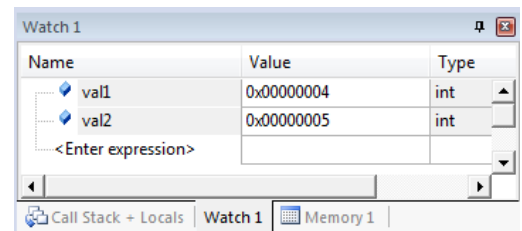
14) Watch and Memory Windows and how to use them:

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 variable values into a Watch or Memory window in real-time. It is possible to enter variable names into windows manually. You can also right click on a variable and select Add *varname* to.. and select the appropriate window. The System Viewer windows work using the same CoreSight technology. Call Stack, Watch and Memory windows can’t see local variables unless stopped in their function (in scope).

Watch window:

A global variable: The global variables `val1` and `val2` are declared in `main_ns.c` near line 28. They are volatile.

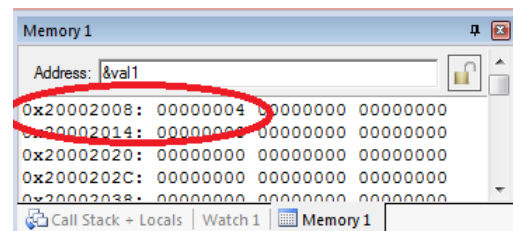
1. Run  the program.
2. Select View and select Periodic Window Update if necessary: 
3. You can configure a Watch or Memory window while the program is running or stopped.
4. In `main_ns.c`, right click on `val1` and select Add `val1` to ... and select Watch 1.
5. Watch 1 will automatically open. `val1` will be displayed:
6. Repeat for `val2`. Both will be updating when the program runs. They are changing very fast so that Watch1 might not be able to always display them.
7. `val1` and `val2` are updated in real-time – no CPU cycles are stolen.
8. You can set some breakpoints at appropriate places to see these variables update. Or you could add a timing loop to slow the program down.



TIP: A Watch or Memory window can display and update global and static variables, structures 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 μ Vision in real-time. To view a local variable in these windows, convert it to a static or global variable.

Memory window:

1. Right click on `val1` and select Add `val1` to ... and select Memory 1.
2. Note the value of `val1` 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 here is `0x2000_2008`.
4. Right click in the Memory window and select Unsigned/Int.
5. The data contents of `val1` is displayed as shown here:
6. The Memory window is also 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 or variable on-the-fly while the program is still running.



SystemCoreClock:



1. In the Watch1 window, double click on `<Enter Expression>` and type in `SystemCoreClock`.
2. Right click on `SystemCoreClock` and unselect Hexadecimal Display.
3. 4 MHz will be displayed. `SystemCoreClock` is provided by CMSIS Core to help determine the CPU clock frequency.

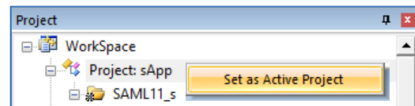
TIP: No CPU cycles are used to perform these operations.

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



15) printf using Event Recorder:

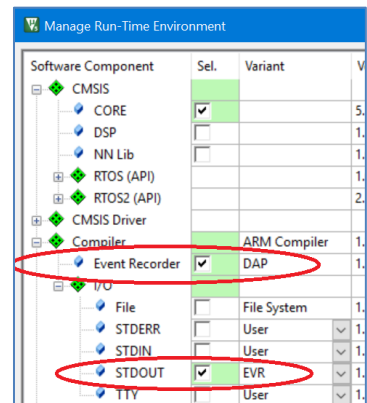
printf is provided by Event Recorder instrumentation that requires minimal user code. printf data will be displayed in the Debug (printf) Viewer and Event Recorder windows.

1. Stop the program  and exit Debug mode .
2. Right click on Project: sApp: in the Project pane and select Set as Active Project:



Add STDOUT File (retarget_io.c):

1. Open the Manage Run-Time Environment window (MRTE) .
2. Expand Compiler and I/O as shown here: .
3. Select Event Recorder and then DAP as shown:
4. Select STDOUT and then EVR. This adds the file retarget_io.c to the project.
5. Ensure all blocks are green and click OK to close the MRTE. Click Resolve if there are red or orange blocks.

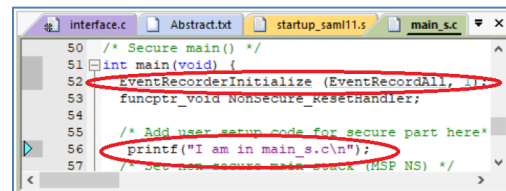


Add printf and ER statements to main_s.c:



1. At the top of main_s.c, right-click and select 'Insert #include file' and then select EventRecorder.h".
2. At the beginning of main() near line 54, *after* the line funcptr_void, add these two lines:
`EventRecorderInitialize (EventRecordAll, 1);`
`__DMB(); // or can use __NOP();`
3. Near line 58 after "Add user setup code for secure part here, add `printf ("I am in main_s.c\n");`

Increase the Heap for printf:

1. In the Project window, double-click on startup_SAM L11.s to open it.
2. Select the Configuration Wizard tab at the bottom of this window.
3. Change Heap to 0x100 bytes.








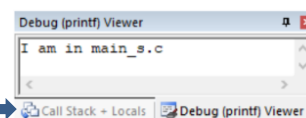
Create Uninitialized RAM: Modify the s_App Scatter File:

1. Select Options for Target  or ALT-F7 and select the Linker tab.
2. Select Edit opposite the Scatter File: box: .
3. Click OK to return to the main µVision menu. S_App.sct will now be displayed with the other source files.
4. Modify RW_IRAM1 and add RW_IRAM2 as follows: This provides some non-initialized RAM for Event Recorder.

```
RW_IRAM1 0x20000000 0x00001800 { ; RW data
    .ANY (+RW +ZI)
}
RW_IRAM2 0x20001800 UNINIT 0x00000800 { ; RW data
    EventRecorder.o (+ZI)
}
```

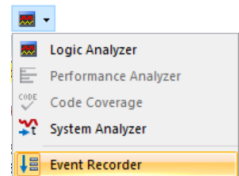
Compile and Run the Project:

1. Select File/Save All or click .
2. Batch Build the source files  and enter Debug mode .
3. The program will run to the start of main() in main_S.c.
4. Click on View/Serial Windows and select Debug (printf) Viewer.
5. Click RUN .
6. In the Debug (printf) Viewer you will see the printf statement appear as shown here: .
7. Right click on the Debug window and select Mixed Hex ASCII mode. Note other useful settings that are available.






View Event Recorder Window:

1. Select the small arrow beside the Analysis window icon and select Event Recorder.
2. Event Recorder opens and the ASCII printf frames are displayed as shown below:



TIP: Note the Timestamps in the Time column are all zeroes. This is because the Cortex-M0 and Cortex-M23 do not have a CoreSight DWT counter. Select SysTick Timer and specify the speed in EventRecorderConf.c to enable timestamps.

3. It is possible to annotate your code with Event Recorder with your own messages.
4. See www.keil.com/pack/doc/compiler/EventRecorder/html/
5. Stop the program and exit Debug mode.

Event Recorder				
Enable Recorder: <input checked="" type="checkbox"/>    Mark: All Operations				
Event	Time (sec)	Component	Event Property	Value
0	0.00000000	EvCtrl	EventRecorderInitialize	Restart Count = 4
1	0.00000000	STDIO	stdout	0x49,0x20,0x61,0x6D,0x20,0x69,0x6E,0x20
2	0.00000000	STDIO	stdout	0x6D,0x61,0x69,0x6E,0x5F,0x73,0x2E,0x63
3	0.00000000	STDIO	stdout	0x0A,0x00,0x00,0x00,0x00,0x00,0x00,0x00

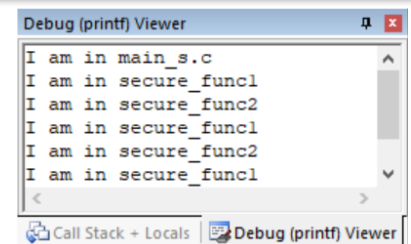
Add Delay() Function to main_s.c:





1. In main_s.c, near line 42 and before main(), add this code:

Add printf and Delay() calls to interface.c:

2. Open interface.c.
3. Near line 31, add this statement:
`extern void Delay (unsigned int Ticks);`
4. Near line 28, add this line:
`#include <stdio.h>`
5. Near line 35, before the return, add these lines:
`printf("I am in secure_func1\n");`
`Delay(150);`
6. Near line 42, before the return, add these lines:
`printf("I am in secure_func2\n");`
`Delay(150);`

```
/*-----  
delays the program  
*-----*/  
void Delay (unsigned int dlyTicks);  
void Delay (unsigned int dlyTicks) {  
    unsigned int volatile Ticks;  
    for (Ticks = 0; Ticks < (1000 * dlyTicks); Ticks++);  
}
```



7. Select File/Save All or click .
8. Batch Build the source files .
8. Enter Debug mode . Click RUN twice: . You must get past the second breakpoint at the start of main() in main_NS.c you set earlier.
9. Remove any breakpoints you encounter so this program can run free.
10. The printf Debug window will display the printf statements shown above right:

Next, we will demonstrate determining power consumption of the SAM L11 board using the Keil ULINKplus:

16) Power Measurement using a Keil ULINKplus:

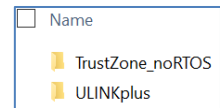
Power Measurement is provided by Keil ULINKplus. See www.keil.com/ulinkplus. The SAM L11, being a Cortex-M23, does not have Serial Wire Viewer. This means a few ULINKplus functions are not provided.

Getting the ULINKplus working on the SAM L11 board with SWD (Serial Wire Debug):

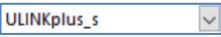
In this section, we will confirm the ULINKplus is connected to the SAM L11 processor debug connection SWD. Once this is working, we will connect the L11 CPU 3.3 v rail to the ULINKplus to measure power consumption.

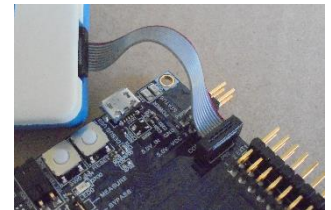
Load the Blinky_ULINKplus Example Project:

1. Obtain the ULINKplus example from here: www.keil.com/appnotes/docs/apnt_314.asp
2. Unzip this file into C:\00MDK\L11\mdk\examples\. A folder ULINKplus will be created.
3. Select Project/Open Project and navigate to C:\00MDK\L11\mdk\examples\ULINKplus\.
4. Open the µVision multi-project file NoRTOS.uvmpw. This version has parts of ULINKplus preconfigured.




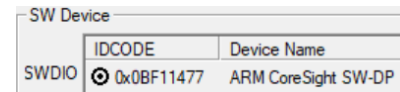
Connect the debug cable to the SAM L11 Xplained board:

1. Connect ULINKplus to the 10 pin CoreSight debug connector J702 as shown here:
2. Power ULINKplus: Connect a USB cable from DEBUG USB connector to your PC.
3. Power the SAM L11 board with a USB cable from your PC to J200 DEBUG USB.
4. Select ULINKplus_s from the Select Target dialog box: 
5. If you do not see ULINKplus_s in this box, create your own as shown in Section 21 page 19.








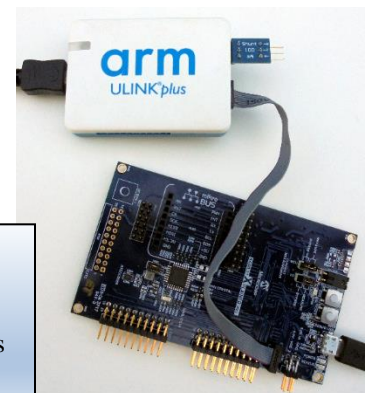
Confirm ULINKplus is connected to the SAM L11 Core:

1. Select Options for Target  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. Perhaps the 10 pin flat cable is rotated or the L11 board is not powered. Jumpers MCU and I/O on correctly ?
5. Click OK twice to return to the main µVision menu when everything is working correctly.



Build the Sources and Run the Program:

1. Compile the source files by clicking on the Batch Build icon: 
2. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed.
3. The program will run to the start of main() in main_s.c.
4. Click on the RUN icon. 
5. Both counter_S and counter_NS will increment in Watch 1 indicating the program is running.
6. Stop the program  and exit Debug mode .
7. At this point, we have the ULINKplus acting as the debug adapter controlling the program and the board. Now, we can connect the ULINKplus power measurement cables to the SAM L11 board as shown on the next page.



Arm Keil ULINKplus:
Power Measurement System.
Power connections to the
board are not yet made in this
photo. See the next page.

17) Connecting the ULINKplus Power Measurement Connections:



The SAM L11 board has on-board power measurement facilities but the ULINKplus does not need them. There are three jumpers to interrupt current paths: J102 (for I/O circuitry), J103 and J104 for the MCU Vcc 3.3v. J104 is for compatibility with other Xplained boards and is not used here. Here is a partial schematic for the SAM L11 Xplained board: Measured current using example is MCU ~ 0.35 ma and I/O about 0.33 ma when running.

We will measure the current on the CPU 3v3 power rail.

ULINKplus Shunt Resistors:




ULINKplus 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.

Connect ULINKplus Power pins to SAM L11 Board:


1. Remove the jumper on J104 MCU.
2. No shunt is needed with this board since internal shunt will handle up to
3. Connect three jumper cables to the ULINKplus.
4. Connect pin  to pin 2 JP104 MCU connector.
5. Connect pin  to pin 1 JP104 MCU connector.
6. Connect the ground jumper wire to GND on J100. You can also use pins 1 or 2 on J802 (these are the closest pins to the edge of PCB).

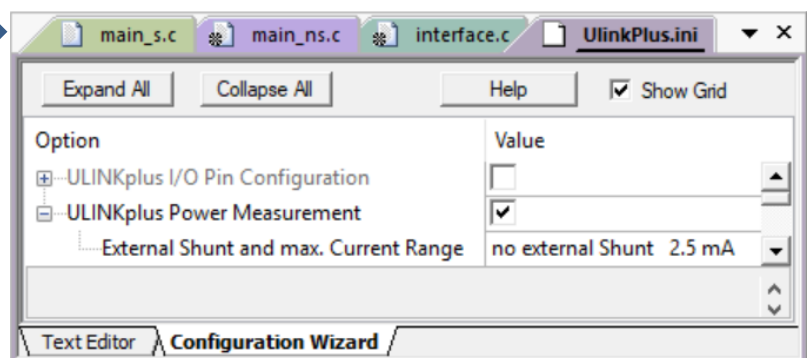
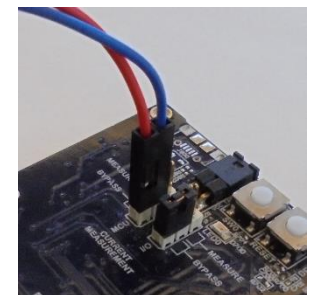
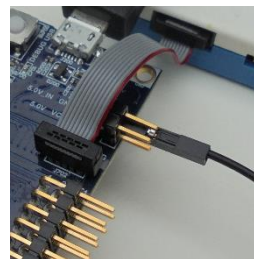
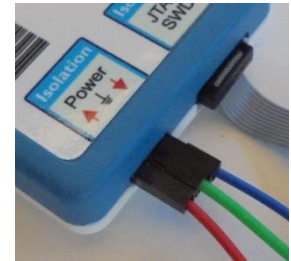
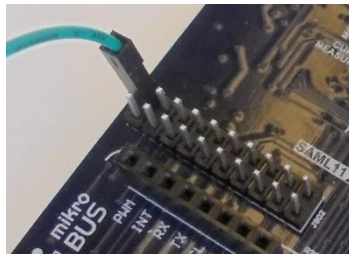
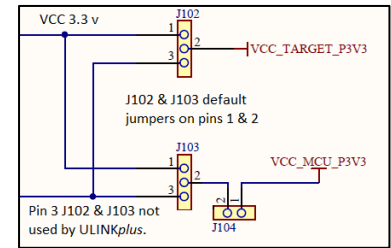
TIP: The grounds on JTAG/SWD, Power, I/O and USB connect are all isolated from each other. You must use a separate ground cable on both Power and I/O.

Configure ULINKplus with the .ini file:

1. Select the Options for Target icon .
2. Select the Debug tab.
3. In Initialization file box, Debug_all.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 μ Vision menu.
6. At the beginning of Debug_all.ini, add this line: include "..\UlinkPlus.ini"
7. Select File/Open and elect All Files (*.*) .
8. Go to C:\00MDK\L11\mdk\examples\ULINKplus and select ULINKplus.ini and click Open.
9. Select the Configuration Wizard tab at the bottom of this file.
10. Select no external shunt as shown here: 
11. Click Save All: 

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 .

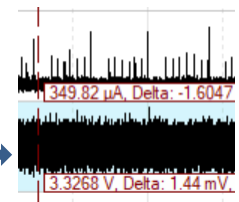
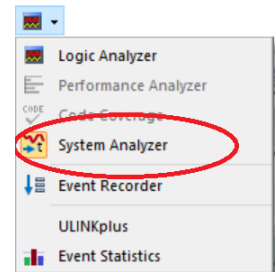


What we have to this point:

ULINKplus is now connected and configured. You are ready to build the L11 project and enter Debug mode and plot the power consumption of the SAM L11 Xplained Pro board.

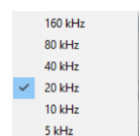
18) Displaying Power Measurements in System Analyzer (SA):

1. Batch 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 SAM L11 board, both current and voltage of the I/O rail are clearly displayed.



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

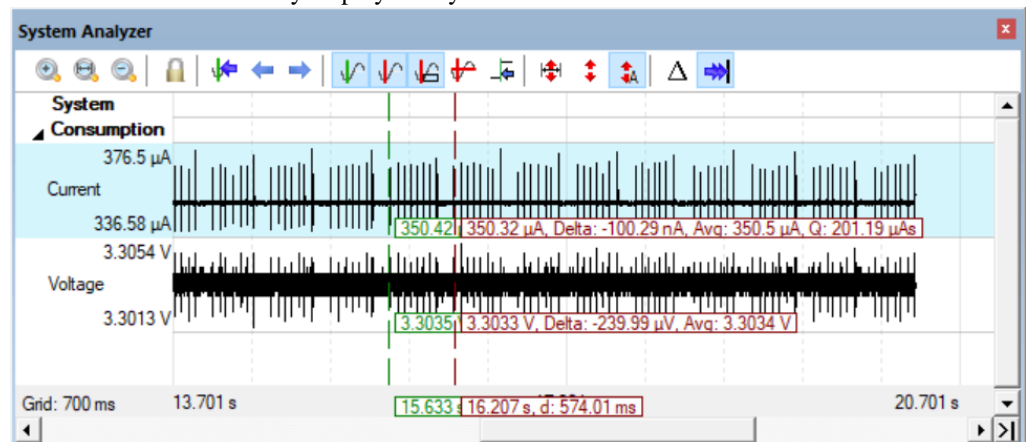
7. You can use the cursors to determine current, voltage and times wherever you hover your mouse.
8. You can stop the SA window while leaving the program running with the Lock icon:
9. With SA locked, you can right click on the Current or Voltage display and select Lo-pass filter:



TIP: Not all System Analyzer (SA) features are available with Cortex-M0/M0+ or Cortex-M23. These features are provided with Cortex-M3/M4/M7 and Cortex-M33.

Note in this screen there are 7 pulses in a group, with a period of about 574 msec. Each group is separated by 61 msec. The pulse current rises to 0.35 ma. These attributes are clearly displayed as you move the cursor about.

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



ULINKplus Window:



1. Select ULINKplus in the Analysis 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 ULINKplus window will be lost with a cycle of Debug mode.
6. Refer to the ULINKplus User Guide located here for more information: www.keil.com/support/man/docs/ulinkplus/
7. Stop the program and exit Debug mode .

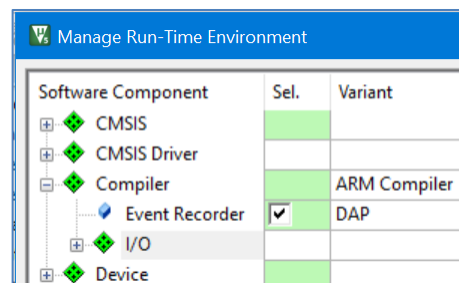
ULINKplus				
I/O				
Pin	Function	Out	In	
0	Digital Input	<input type="checkbox"/>	Low	
1	Digital Input	<input type="checkbox"/>	Low	
2	Digital Input	<input type="checkbox"/>	Low	
3	Digital Input	<input type="checkbox"/>	Low	
4	Analog Input		3.164649 V	
5	Digital Input	<input type="checkbox"/>	Low	
6	Digital Input	<input type="checkbox"/>	Low	
7	Digital Input	<input type="checkbox"/>	Low	
8	Digital Input	<input type="checkbox"/>	Low	
9	+3.3V Out	<input type="checkbox"/>		
Power				
Power Measurement				
Shunt R		33.20 Ohm		
Voltage		3.325 V		
Current		4.70 mA		
Power		0.02 W		
Enable		<input checked="" type="checkbox"/>		

19) Annotating Your Code to Display in System Viewer:


You can annotate your code and this will be displayed in the System Viewer and Event Viewer windows. Keil RTX 5 RTOS (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.

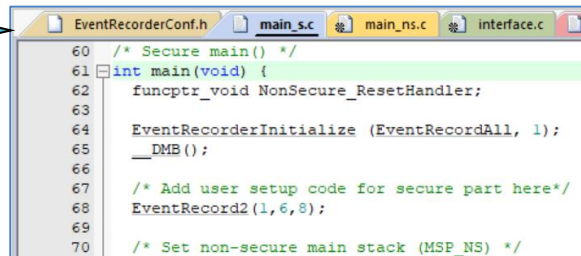


Add Event Recorder (ER) statements to main_s.c:

1. At the top of main_s.c near line 30, right-click and select 'Insert #include file' and then select EventRecorder.h.
2. At the beginning of main() near line 63, add these two lines: 

```
EventRecorderInitialize (EventRecordAll,1);  
__DMB ();
```
3. In the Add user setup code.. area near line 68 add:

```
EventRecord2 (1,6,8);
```









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

1. Near the top of interface.c near line 31, right-click and select 'Insert #include file' and then select EventRecorder.h.
2. In interface.c, near line 42 and just before the line `return func1(x);`, add this line: `EventRecord2 (2,6,8);`

Configure the SysTick Timer in EventRecorder.h: (in Project/Compiler tree)

1. Open EventRecorder.h (click on its tab) and click on the Configuration Wizard tab at the bottom.
2. Select Time Stamp Source to SysTick. Set Timestamp Clock Frequency to 4 Mhz (4000000).

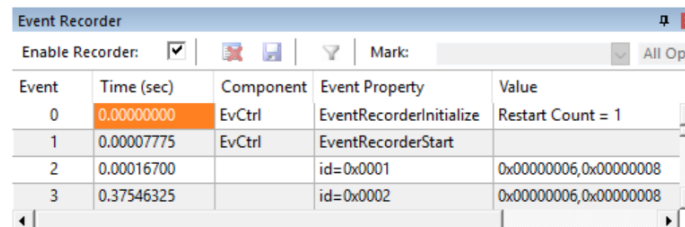
Compile and Run the Project:

1. Select File/Save All or click .
2. Batch Build the files.  Make sure you use Batch Build and not Build or Rebuild.
3. Enter Debug mode . Click RUN. .
4. Open System Analyzer (SA) if it is not already open: Click Freeze Data .
5. A new row is created called Event Record.
6. Adjust the SA window and you will see the Record2 ID = 2 icons in the Event Record row:
7. If you scroll to the left of this window you can see the 0001 Record: .
8. 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.



Event	Time (sec)	Component	Event Property	Value
0	0.00000000	EvCtrl	EventRecorderInitialize	Restart Count = 1
1	0.00007775	EvCtrl	EventRecorderStart	
2	0.00016700		id=0x0001	0x00000006,0x00000008
3	0.37546325		id=0x0002	0x00000006,0x00000008

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/

20) Measuring Power Consumption with Event Recorder 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:

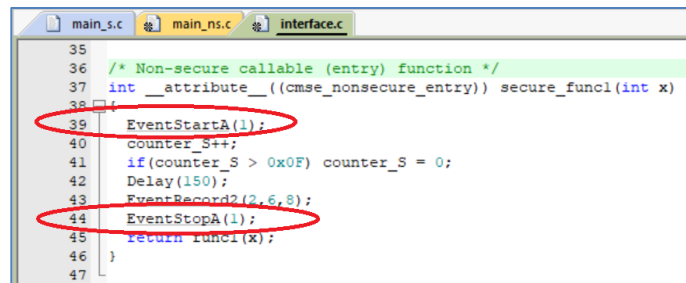
2. In interface.c, near line 39 add: EventStartA(1);
3. In interface.c, near line 39 add: EventStopA(1);

Batch Build and RUN the program:

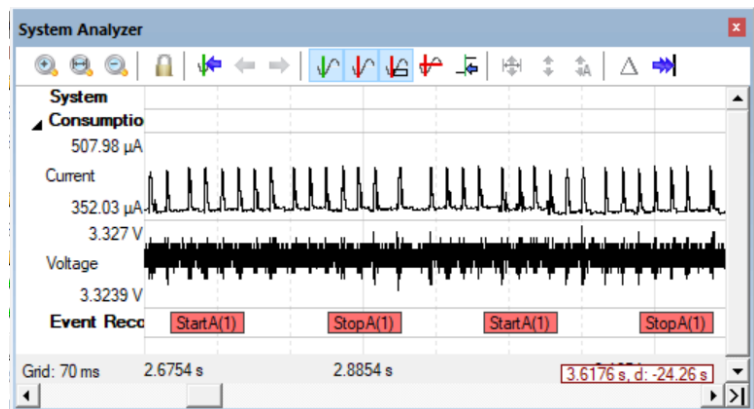
4. Select File/Save All or click .
5. Batch Build the files.  Make sure you use Batch Build and not Build or Rebuild.
6. Enter Debug mode . Click RUN. .

View System Analyzer:

7. Open the System Analyzer window. Position as necessary to comfortable view it.

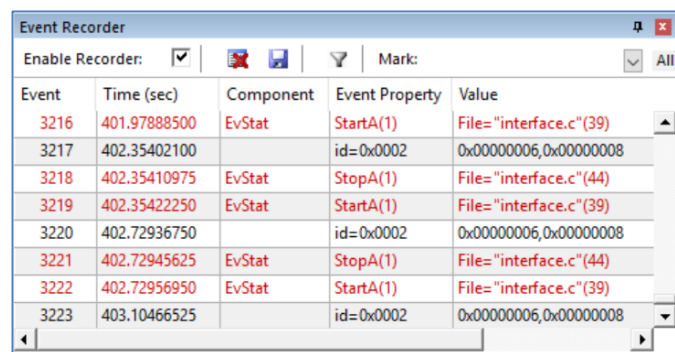


```
35
36 /* Non-secure callable (entry) function */
37 int __attribute__((cmse_nonsecure_entry)) secure_func1(int x)
38 {
39     EventStartA(1);
40     counter_S++;
41     if(counter_S > 0x0F) counter_S = 0;
42     Delay(150);
43     EventRecord2(2,6,8);
44     EventStopA(1);
45     return func1(x);
46 }
47
```



View Event Recorder Window:

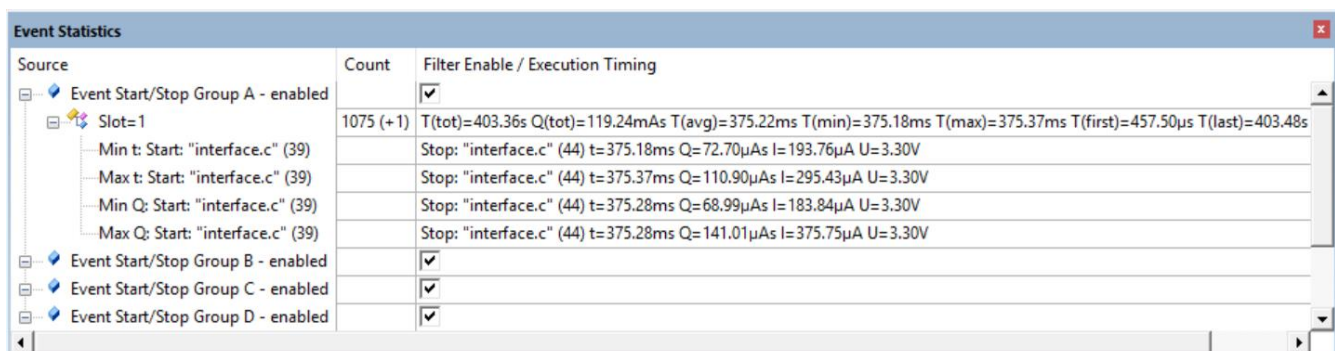
8. Open Event Recorder and notice that in addition to the EventRecords ID=2, you now have EventStart and EventStop frames displayed as shown here:
9. These frames update while the program is running.



Event	Time (sec)	Component	Event Property	Value
3216	401.97888500	EvStat	StartA(1)	File="interface.c"(39)
3217	402.35402100		id=0x0002	0x00000006,0x00000008
3218	402.35410975	EvStat	StopA(1)	File="interface.c"(44)
3219	402.35422250	EvStat	StartA(1)	File="interface.c"(39)
3220	402.72936750		id=0x0002	0x00000006,0x00000008
3221	402.72945625	EvStat	StopA(1)	File="interface.c"(44)
3222	402.72956950	EvStat	StartA(1)	File="interface.c"(39)
3223	403.10466525		id=0x0002	0x00000006,0x00000008

View the Statistics View window.

10. Statistics about the program power between the Start and Stop you entered is now displayed as shown below:
11. This is for Group A, Slot 1 which is what you specified.
12. You can have many of these events in various parts of your program.



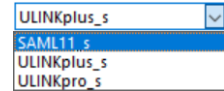
Source	Count	Filter Enable / Execution Timing
Event Start/Stop Group A - enabled		<input checked="" type="checkbox"/>
Slot=1	1075 (+1)	T(tot)=403.36s Q(tot)=119.24mAs T(avg)=375.22ms T(min)=375.18ms T(max)=375.37ms T(first)=457.50 μ s T(last)=403.48s
Min t: Start: "interface.c" (39)		Stop: "interface.c" (44) t=375.18ms Q=72.70 μ As I=193.76 μ A U=3.30V
Max t: Start: "interface.c" (39)		Stop: "interface.c" (44) t=375.37ms Q=110.90 μ As I=295.43 μ A U=3.30V
Min Q: Start: "interface.c" (39)		Stop: "interface.c" (44) t=375.28ms Q=68.99 μ As I=183.84 μ A U=3.30V
Max Q: Start: "interface.c" (39)		Stop: "interface.c" (44) t=375.28ms Q=141.01 μ As I=375.75 μ A U=3.30V
Event Start/Stop Group B - enabled		<input checked="" type="checkbox"/>
Event Start/Stop Group C - enabled		<input checked="" type="checkbox"/>
Event Start/Stop Group D - enabled		<input checked="" type="checkbox"/>

This is the end of the exercises (for now). More will be forthcoming.

21) Creating a New Options for Target to add a new Debug Adapter: *for reference:*

It is easy to add a new Debug adapter that is supported by μ Vision. You can change the debug adapter in an existing Options for Target or create a new one. Multiple Options for Target are selected by the pulldown menu:



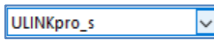
Each Options for Target can have many settings set according to your needs. They need not be differentiated by a debug adapter nor does a new one need a different debug adapter.

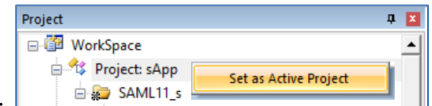


Each Options for target can be opened by clicking  or selecting ALT-F7. Select the tab of interest.




Add a New Debug Adapter: We will add a Keil ULINKpro by creating a new Options for Target.

Create a new Options Target:

1. μ Vision must not be in Debug mode.
2. Right click on Project: sApp: in the Project pane and select Set as Active Project:
3. Select Manage Project Items:  The Manage Project Items opens.
4. Select Insert:  or press the INSERT key.
5. Type in: ULINKpro_s (or whatever you want)
6. Click OK and this will be saved.
7. Select ULINKpro_s in the Select Target menu: 



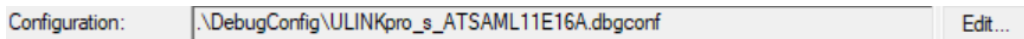
Configure the Debug tab in ULINKpro_s Options for Target:


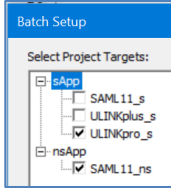




1. Select the Target Options icon  in μ Vision.
2. Select the Debug tab: Select ULINKpro Debugger: 
3. Click on Settings: 
4. If the ULINKpro and the board are connected and powered, you will see a SW Device displayed: This is good test:

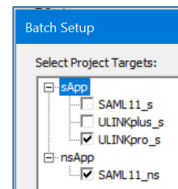
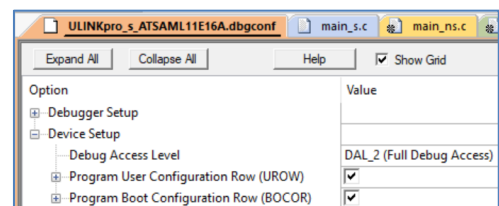
Configure the Debug Configuration .dbgconf File:

TIP: This must be done every time you add a new debug adapter or Options for target. The template from the L11 Pack will be copied to your project with the Options Target name added. It will not have the correct values for the example or for your own project. You must configure it each time to the specific addresses used. See the .sct scatter file in the Linker tab.

1. Select the Pack: tab. A new ULINKpro_s .debugconf file will be displayed (or for your debug adapter) as shown:




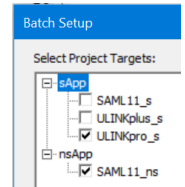
2. Select Edit... and this file will be displayed in μ Vision. Click OK twice to return to the main μ Vision menu.
3. In this .dbgconf file, click on the Configuration Wizard tab.
4. Expand Device Setup. Enable both UROW and BOCOR as shown:
5. Expand Program User Configuration Row (UROW).
6. Expand Application Flash.
7. Modify Secure Size from 0x0000_00FF to 0x0000_0080.
8. Modify Non-Secure Callable from 0x0 to 0x0000_0020.
9. Expand RAM.
10. Modify Secure Size from 0x000_007F to 0x0000_0040.
11. Select the small arrow beside the Batch Build icon: 
12. Select only ULINKpro_s and SAML11_ns Option Targets to save compile time: 
13. Click Save All: 
14. Compile the source files by clicking on the Batch Build icon: 
15. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed.
16. Click on the RUN icon.  The program should run as before. Both Watch 1 variables should increment.



22) Issues Encountered in Creating this Document:

This list contain hints that might be useful to you in following this tutorial:

1. If variables in windows update only when you stop the program, make sure View/Periodic Window Update is enabled.
2. Make sure project s_App is active and not ns_App. You want to do all of your work in this tutorial in the Secure App.
3. Ensure you use Batch Build and not Build or Rebuild. This way the sources from both Secure and Non-Secure sources are compiled. 
4. If you do not see the grey blocks on the left in a source window where you know there should be code, you might have failed to use BatchBuild.
5. Make sure the sources you want build are enabled in the Batch Build Setup menu. Enable only those you need in order to save compile time.
6. It is easy to get the Cortex Debug connector on backwards.
7. If you do not see any Event Records in the System Analyzer window, the SysTick timer might not be correctly set.



23) Document Resources:

See www.keil.com/Microchip

Books:

1. **NEW!** Getting Started with MDK 5: Obtain this free book here: www.keil.com/mdk5/
2. There is a good selection of books available on ARM: <https://developer.arm.com/support/arm-books>
µVision contains a window titled Books. Many documents including data sheets are located there.
3. **The Definitive Guide to the Arm Cortex-M0/M0+** by Joseph Yiu. Search the web for retailers.
4. **The Definitive Guide to the Arm Cortex-M3/M4** by Joseph Yiu. Search the web for retailers.
5. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
6. 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. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
4. Porting a mbed™ Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
5. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
6. GCC Arm Developer: <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm>
7. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
8. Cortex-M Processors for Beginners: <http://community.arm.com/docs/DOC-8587>
9. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
10. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
11. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
12. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
13. **NEW!** ARMv8-M Architecture Technical Overview www.keil.com/appnotes/files/apnt_291.pdf
14. **NEW!** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp
15. Migrating Arm Compiler 5 (AC5) to Arm Compiler 6 (AC6): www.keil.com/appnotes/docs/apnt_298.asp

Keil Tutorials for Microchip Boards:

www.keil.com/Microchip

1. Microchip SAM4S: www.keil.com/appnotes/docs/apnt_228.asp
2. Microchip SAM3X: www.keil.com/appnotes/docs/apnt_229.asp
3. Microchip SAMV71: www.keil.com/appnotes/docs/apnt_274.asp
4. Microchip SAM L10 : www.keil.com/appnotes/docs/apnt_313.asp
5. Microchip SAML11: www.keil.com/appnotes/docs/apnt_314.asp
6. Programming the Microchip SAM L11 Cortex-M23: www.keil.com/appnotes/docs/apnt_315.asp

Useful Arm Websites:

1. **NEW!** CMSIS 5 Standards: https://github.com/ARM-software/CMSIS_5 and www.keil.com/cmsis/
2. Forums: www.keil.com/forum <http://community.arm.com/groups/tools/content> <https://developer.arm.com/>
3. ARM University Program: www.arm.com/university. Email: university@arm.com
4. mbed™: <http://mbed.org>

24) Keil Products and contact information: See www.keil.com/Microchip

Keil Microcontroller Development Kit (MDK-ARM™) for Microchip Arm processors:

- **MDK-Lite™** (Evaluation version) up to 32K Code and Data Limit - \$0
- **New MDK-ARM-Essential™** For all Cortex-M series processors – unlimited code limit
- **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 Adapters (for Flash programming too)

- **ULINK2** - (ULINK2 and ME - SWV only – no ETM) **ULINK-ME** is equivalent to a ULINK2.
- **New ULinkplus-** Cortex-Mx High performance SWV & power measurement.
- **ULINKpro** - Cortex-Mx SWV & ETM instruction trace. Code Coverage and Performance Analysis.
- **ULINKpro D** - Cortex-Mx SWV no ETM trace. ULinkpro and ULinkpro D also work with Arm DS-5.

Many Xplained boards have an EDBG debug adapter which µVision supports. For Serial Wire Viewer (SWV), any Keil ULink or a J-Link is needed. For ETM, a ULinkpro is needed. For Power Measurement, a ULinkplus is needed.

Call Keil Sales listed below for more details on current pricing and any specials. All products are available.

All software products include Technical Support and Updates for 1 year. This can easily be renewed.

Keil RTX™ Real Time Operating System: www.keil.com/rtx

- RTX is provided free as part of Keil MDK.
- No royalties are required. RTX has an Apache 2.0 license.
- See https://github.com/ARM-software/CMSIS_5
- RTX source code is included with all versions of MDK.
- Kernel Awareness visibility windows are integral to µVision.
- FreeRTOS is also supported in MDK.

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor. For Microchip support: www.keil.com/Microchip

For Linux, Android, bare metal (no OS) and other OS support on Microchip Cortex-A series processors please see DS-5 and DS-MDK at www.arm.com/ds5/ and www.keil.com/ds-mdk.

Getting Started with DS-MDK: www.keil.com/mdk5/ds-mdk/install/



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/>

