

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_313.asp

The latest SAM L11 tutorial is located here: www.keil.com/appnotes/docs/apnt_314.asp

Introduction:

The purpose of this tutorial is to introduce you to the Microchip SAM L10 Cortex®-M23 processor using the Arm® Keil® MDK toolkit featuring the IDE µVision®. We will demonstrate all debugging features available on this processor. At the end of this tutorial, you will be able to confidently work with these Arm processors and Keil MDK.

Getting Started MDK 5: www.keil.com/gsg. **Arm Compiler 6:** www.keil.com/appnotes/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. SAM L10/L11 needs a valid MDK license.

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 supported.

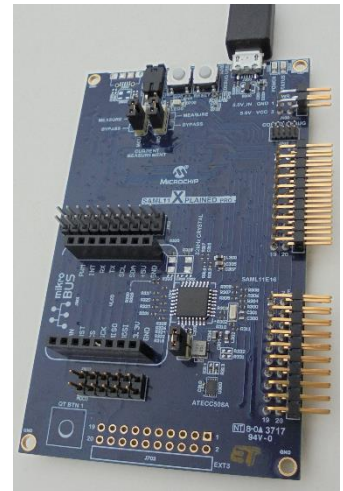
Why Use Keil MDK ?

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 2Q18.
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. Pages 13 & 19.**
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. 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. Four Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
3. RTX and RTX Threads windows: kernel awareness for RTX that updates while your program is running.
4. **NEW!** µVision Event Recorder. You can use this in your own programs. See pages 13 and 18.
5. printf using Event Recorder (EVR). No UART is required.
6. Create your own µVision project from scratch. Add Keil RTX to any project.



General Information:

| | |
|--|---|
| 1. Microchip Evaluation Boards & Keil Evaluation Software: | 3 |
| 2. MDK 5 Keil Software Information: MDK 5.25 or later: | 3 |
| 3. Debug Adapters Supported: | 3 |
| 4. Arm CoreSight Definitions: | 4 |

Keil Software and Software Packs:

| | |
|---|---|
| 5. Keil MDK Software Download and Installation: | 5 |
| 6. Install the μ Vision MDK Professional License: | 5 |
| 7. μ Vision Software Pack Download and Install Process: | 5 |
| 8. Install the Blinky example | 5 |
| 9. Other features of Software Packs: | 6 |

Blinky Example using the Microchip Cortex-M23 SAM L10

| | |
|---|---|
| 10. <i>Blinky</i> example using the Microchip Cortex-M23 SAM L10: | 7 |
|---|---|

Arm CoreSight Debugging Features supported by Keil μ Vision:

| | |
|---|----|
| 11. Hardware Breakpoints and Single Stepping: | 8 |
| 12. Call Stack & Locals window: | 9 |
| 13. Watch and Memory windows and how to use them: | 10 |
| 14. Peripheral System Viewer (SV): | 11 |
| 15. Watchpoints: Conditional Breakpoints: | 12 |
| 16. NEW ! printf using Event Recorder: | 13 |

NEW ! Power Measurement using Keil ULINKplus:

| | |
|--|----|
| 17. Getting ULINKplus Connected to SAM L10 Xplained board via SWD: | 14 |
| 18. Connecting ULINKplus to the Power Management Connectors | 15 |
| 19. Displaying Power Measurement in System Analyzer: | 16 |
| 20. Displaying Event Statistics: | 17 |
| 21. More Features of Event Statistics: | 18 |

Creating a New μ Vision® Project for SAM L10:

| | |
|---|----|
| 22. Creating a new μ Vision Project from Scratch: | 19 |
| 23. Adding Keil RTX RTOS to your Project: | 22 |
| 24. Adding a Second Thread (Task) to Your Project: | 23 |
| 25. NEW ! Event Recorder for RTX: | 24 |

Other Useful Information:

| | |
|--|----|
| 26. Document Resources: | 25 |
| 27. Keil Products and Contact Information: | 26 |

1) Microchip Evaluation Boards & Keil Evaluation Software:

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 on 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/

2) MDK 5 Keil Software Information: *This document uses MDK 5.25 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 "Manage Run-Time Environment" (MRTE). 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 L10 does not have any feature such as SWV or ETM that requires a specific adapter. All other CoreSight functions found on SAM L10 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 L10, this is all you need. The Keil ULINKplus will provide power measurement for SAM L10. 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. **ULINKpro:** ULINKpro supports all SWV features and adds ETM Instruction Trace. ETM records all executed instructions. ETM provides Code Coverage, Execution Profiling and Performance Analysis features. ULINKpro also provides the fastest Flash programming times. Not all SAM processors have ETM. Consult your datasheet.
4. **NEW ! ULINKplus:** Power Measurement + High SWV performance and Test Automation.
See www.keil.com/ulink/ulinkplus/ for details.
5. **Segger J-Link:** J-Link Version 6 (black) or later supports Serial Wire Viewer. SWV data reads and writes are not currently supported with a J-Link.

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 ULINKpro and ULINKplus. 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, Cortex-M7 and Cortex-M23 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 5.25 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. You do not need a debug adapter for the basic exercises: just the SAM L10 board, a USB cable and MDK installed.
5. You do not need a Keil MDK license for this tutorial. MDK-Lite, the evaluation version will work.

Download MDK-Core Version 5

6) Install the µVision MDK Professional License:



1. Select File/License Management. If you have not used this feature previously, this icon is displayed in lower left corner:
2. Click on this icon to obtain a temporary MDK Pro license. Cortex-M23 needs at least MDK Essential.
3. If this expires or is not available, please contact Keil Sales for temporary licenses: Inside-Sales@arm.com

Evaluate MDK Professional

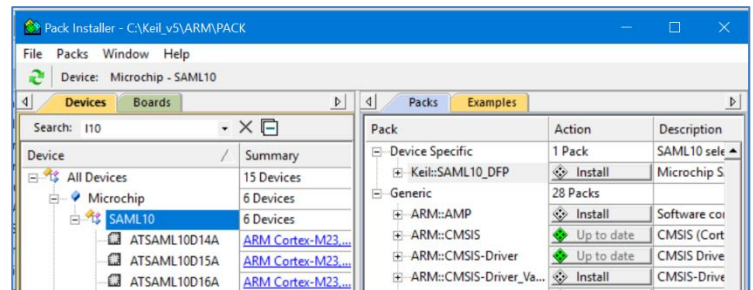
7) µVision Software Pack Download and Install Process:

A Software Pack contain components such as header, Flash programming, documents and other files used in a project.

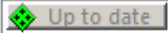
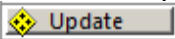
1) Start µVision and open Pack Installer and install The SAM L10 Software Pack from the Web:

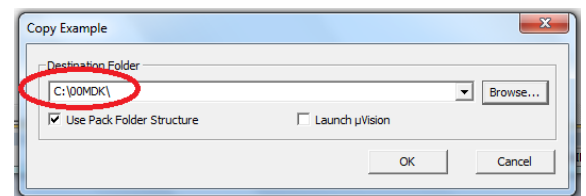
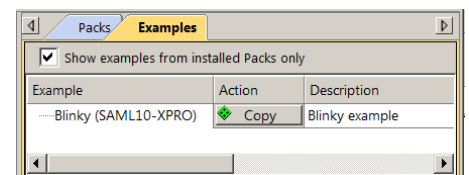
1. Start µVision:  Connect your PC to the Internet to download files.
2. Open the Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.
3. This window opens up: Select the Devices tab.
4. Enter L10 in the Search: box. Highlight as shown
5. In the Packs tab, select Install as shown:
6. The Pack will be installed into µVision.

TIP: You can also install the Pack by double-clicking on the .pack file. It will be recognized and installed.



8) Install the Blinky Example:

1. Select the Devices tab in Pack Installer. Select SAM L10 as shown above.
2. Select the Examples tab and one Blinky example will display:
3. Click on Copy. The Copy Example window shown here opens:
4. Enter C:\00MDK\ as shown. Unselect Launch µVision. Click OK.
5. Blinky will copy to C:\00MDK\mdk\boards\Microchip\SAML10-XPRO\
6. The Pack status is indicated by the “Up to date” icon: 
7. Update means there is an updated Software Pack available for download. 
8. You can update any files at this time with a yellow Update box.
9. Close Pack Installer. You can open it any time by clicking on its icon.
10. If a dialog box says Software Packs folder has modified, select Yes.




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

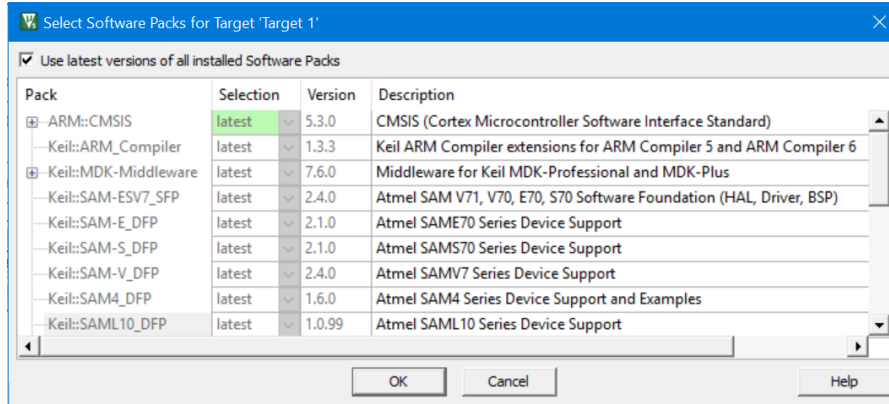
TIP: For simplicity, we will use the default folder of C:\00MDK\ in this tutorial. You can use any folder you prefer.

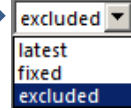
9) Other features of Software Packs: This page is for reference:

1) Select Software Pack Version:


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

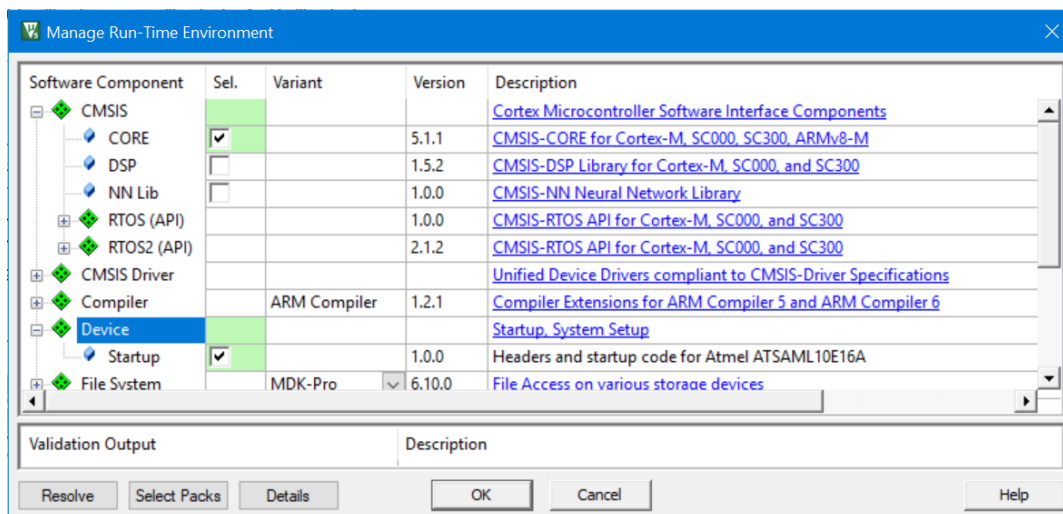
1. Open the Select Software Pack by clicking on its icon: 
2. This window opens up. Note **Use latest versions ...** is selected. The latest version of the Pack will be used.
3. Unselect this setting and the window changes to allow various versions of Packs to be selected.



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: Blinky.uvprojx in C:\00MDK\mdk\boards\Microchip\SAML10-XPRO\.
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

10) **Blinky example program using the Microchip SAM L10 board:**

Now we will connect a Keil MDK development system using the SAM L10 board. This page will use the Microchip EDBG CMSIS-DAP on-board debug adapter.

1. Connect a USB cable between your PC and the SAM L10 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. 
4. Select Project/Open Project.
5. Open the file Blinky.uvprojx located in: C:\00MDK\mdk\boards\Microchip\SAML10-XPRO\
6. Compile the source files by clicking on the Rebuild icon.  If you get an error here stating ARM compiler does not support Cortex-M23, you need to obtain a MDK Essential license or higher. See page 5.
6. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed. Progress will be indicated in the Output Window.
7. The program will run from the RESET vector to the start of main() and stop. This will be indicated in the main.c C source window.
8. Click on the RUN icon. 




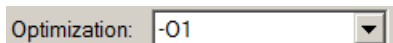




The yellow LED LEDO will now blink on the SAM L10 board.

Now you know how to compile a program, program it into the SAM L10 processor Flash, run it and stop it !

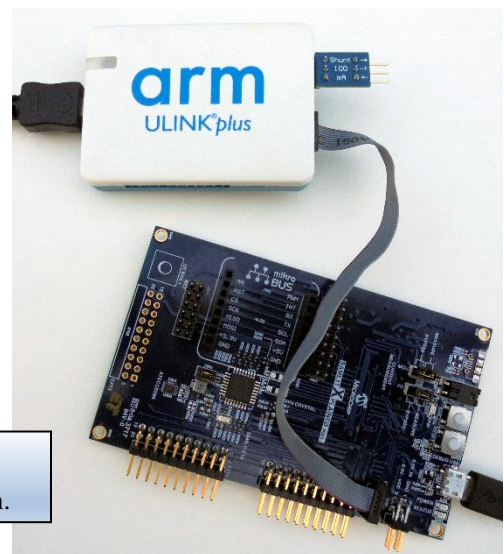
Note: The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

Set Optimization to Level -01:

This is needed for the breakpoints step on the next page.


1. Stop the program  and exit Debug mode .
2. Select Options for Target:  or ALT-F7.
3. Select the C/C++ (AC6) tab.
4. Select Optimization level -01 as shown here:

5. Click OK to close this window.
6. Select File/Save All or click .
7. Compile the source files by clicking on the Rebuild icon. .
8. Enter Debug mode.  The Flash memory will be programmed. Progress will be indicated in the Output Window.
9. Click on the RUN icon. 

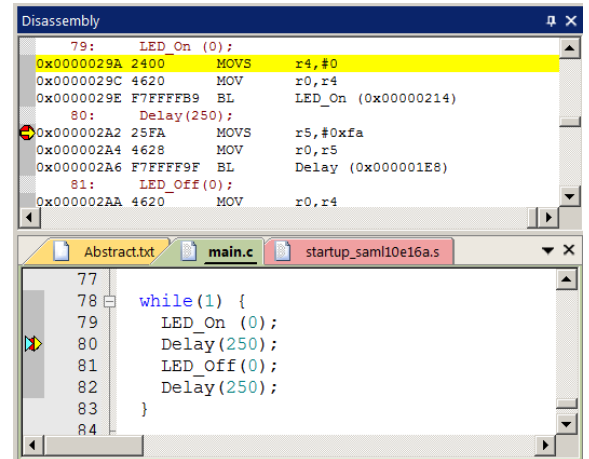
Keil ULINKplus:
Power Measurement System.



11) Hardware Breakpoints:

The SAM L10 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 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 between near lines 91 through 94 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. Set a second breakpoint in the while(1) 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.
9. **Delete all breakpoints.**
10. Close the Breakpoint window if it is open.
11. You can also delete the breakpoints by clicking on the red circle.




TIP: If you set too many breakpoints, μ Vision 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:

1. RUN icon  for a short time. Stop the program with the STOP icon. 
2. The program will probably stop inside the Delay() function starting near line 29 in Blinky.c.

By Assembly Instruction:

3. 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:



4. Click inside the Blinky.c source code window or better, on the Blinky.c tab.
5. Click on Step Out  to exit the Delay() function. Step is not able to correctly step inside this function.
6. 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.
7. When you enter the Delay(); function, stepping will no longer work. This is because interrupts are disabled during single-stepping and this function uses the SysTick handler which is interrupt driven.
8. Click Step Out  to exit the Delay function and return to main().
9. Stop the program  if it is running.



12) Call Stack + Locals Window:

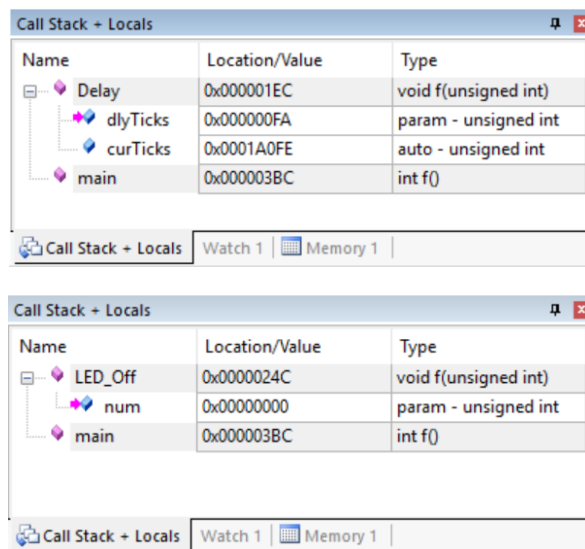
Local Variables:

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.

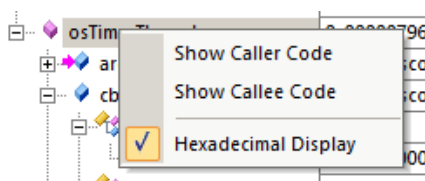
If possible, the values of the local variables will be displayed and if not the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

1. Run  and Stop  the Blinky program. It will probably stop inside the Delay() function.
2. Click on the Call Stack + Locals tab if necessary to open it. Expand some of the entries.
3. The Call Stack + Locals window will show Delay and main as shown here: Local variables are displayed.

4. Click on Step Out  to exit the Delay() function. Only main will remain.
5. Click on the Step In icon  to enter a few functions.
6. As you click on Step In, you can see the program entering and perhaps leaving various functions. Note the local variables are displayed.
7. Shown is an example Call Stack + Locals window while the program is in the LED_Off function:
8. 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.
9. The Blinky program is very simple. Call Stack + Locals is very useful in sorting out stack issues in more complicated programs



10. Right click on a function 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 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 before continuing.

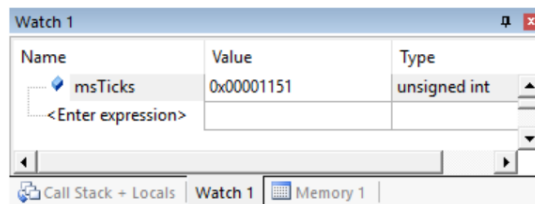
13) Watch and Memory Windows and how to use them:

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 variable `msTicks` is declared in `main.c` near line 17.

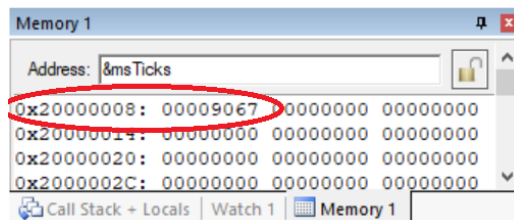
1. Leave Blinky running. Click Run:  The yellow LED will be blinking.
2. You can configure a Watch or Memory window while the program is running.
3. In `Blinky.c`, right click on `msTicks` and select Add `msTicks` to ... and select Watch 1.
4. Watch 1 will automatically open. `msTicks` will be displayed:
5. `msTicks` is updated in real-time – no CPU cycles are stolen.
6. Click and select the entire variable value of `msTicks`.
7. Enter 0 (or any other value) and press Enter.
8. The value of `msTicks` is set to zero. Incrementing will resume.



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 `msTicks` and select Add `msTicks` to ... and select Memory 1.
2. Note the value of `msTicks` 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_0008`.
4. Right click in the Memory window and select Unsigned/Int.
5. The data contents of `msTicks` 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 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.

14) Peripheral System Viewer (SV):

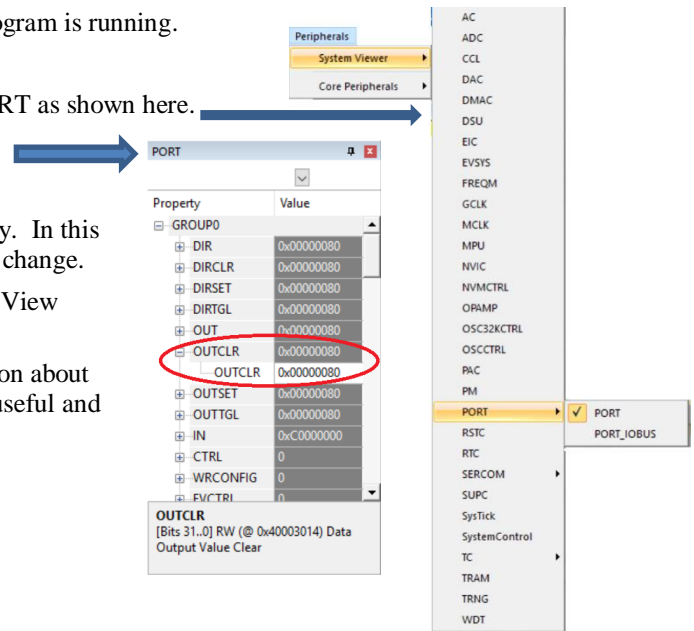
The System Viewer provides the ability to view certain registers in the CPU core and in peripherals. In most cases, these Views are updated in real-time while your program is running. These Views are available only while in Debug mode. There are two ways to access these Views: **a) View/System Viewer**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

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

Select ADC0:

1. Select Peripherals/System Viewer and then PORT and PORT as shown here.
2. This window opens up. Expand GROUP:
3. You can now see the ports update as the LED blinks.
4. You can change the values in the System Viewer on-the-fly. In this case, the values are updated quickly so it is hard to see the change.
5. You can look at other Peripherals contained in the System View windows.

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 very useful and is an easy way to find the address of a peripheral port.






SysTick Timer:

This program uses the SysTick timer as a tick timer for the Delay() function.

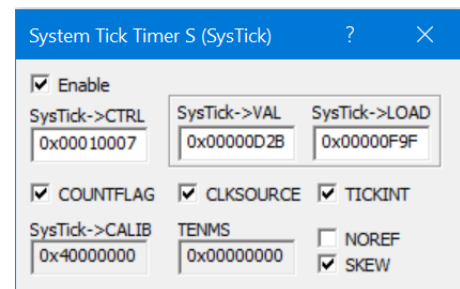
Blinky.c programs SysTick in main() near line 76 with this CMSIS function:

```
SysTick_Config(SystemCoreClock / 1000); /* Setup SysTick for 1 msec */
```

1. Select Peripherals/Core Peripherals and then select SysTick S Timer. “S” means Secure state.
2. The SysTick window shown below opens:
3. Note it also updates in real-time while your program runs. These windows use the same CoreSight DAP technology as the Watch, Memory and Peripheral windows.
4. Note the SysTick->LOAD register. This is the reload register value. This is set during the SysTick configuration.
5. Note that it is set to 0x0F9F. This is the same hex value of 4,000,000/1000-1 (0x0FA0)-1) that is programmed by Blinky.c. This is where this value comes from. Changing the variable passed to this function is how you change how often the SysTick timer creates its interrupt 15.
6. In the SysTick->LOAD register in the SysTick window, *while the program is running*, type in 0x300 and press Enter.
7. The blinking LED will speed up. This will convince you of the power of Arm CoreSight debugging.
8. Replace RELOAD with 0x0F9F. A CPU RESET  and RUN  will also do 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.



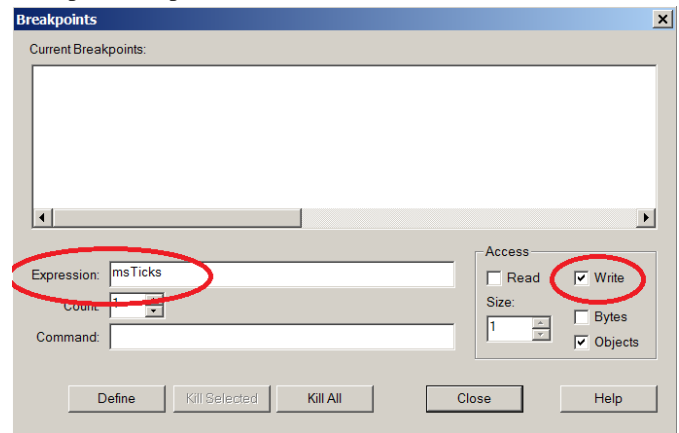
15) Watchpoints: Conditional Breakpoints

The SAM L10 Cortex-M23 processor has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil documents. Cortex-M23 processors do not have the equality test – only address matching. Using a data test causes μ Vision to stop the processor creating substantial program delays. Cortex-M3/M4/M7 and M33 have data tests implemented.

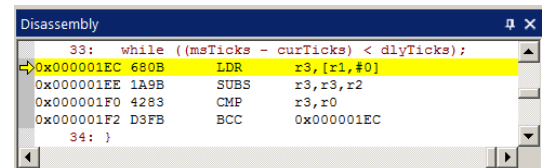
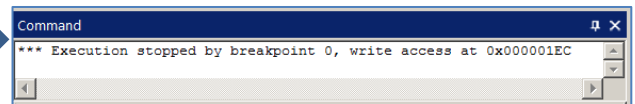
Watchpoints can be useful to detect a read or a write to a specified address. When the specified address is accessed, the program will stop. This can be used to determine the range of the Stack Pointer. It is also useful for determining instructions making reads or writes to memory. We will demonstrate the Stack Pointer here.

This test will put the program in the main() function. The first time Delay() function is called, a write to the Stack Pointer will trigger a Watchpoint and stop the processor.

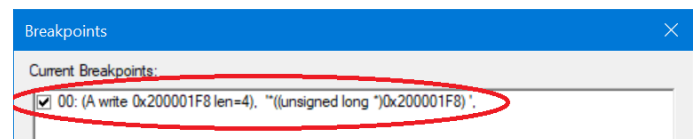
1. Stop the program . Normally you can configure a Watchpoint while the program is running or halted.
2. The program will probably stop in the Delay() function.
3. Click Step Out so that only main() is indicated in the Call Stack + Locals window.
4. Select Debug in the main μ Vision window and then select Breakpoints or press Ctrl-B.
5. Select Access to Write as shown here:
6. Enter: msTicks in the Expression box. This window will display:
7. Click on Define or press Enter and the expression will be accepted into the Current Breakpoints: box as shown below in the bottom Breakpoints window:
8. Click on Close.
9. Click on RUN. .
10. When the program writes to the msTick variable, the Watchpoint will detect this and stop the program.



11. The command window displays the cause of the halt:
12. This is the instruction at 0x0000 01EC that caused the write.
13. This instruction is displayed in the Disassembly window:
14. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.
15. Exit Debug mode.



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)

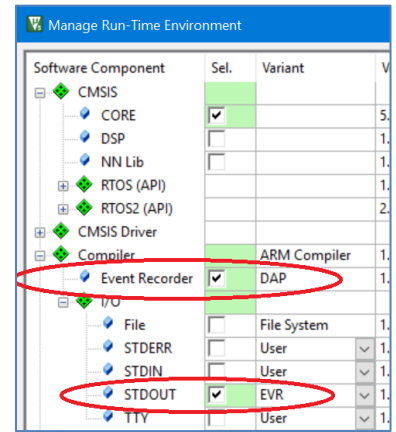
16) printf using Event Recorder:

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. If you are using a Cortex-M3/M4/M7/M33 you can send this data out the ITM port 0 using SWV (Serial Wire Viewer). Cortex-M0/M0+ and M23 do not have SWV. They use DAP.

1. Stop the program  and exit Debug mode .

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.



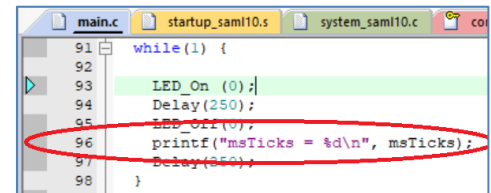
TIP: If you selected ITM instead of EVR, printf will require SWV. Cortex-M23 does not have SWV, only DAP.

Add printf and ER statements to Blinky.c:








1. At the top of main.c, right-click and select 'Insert #include file' and then select EventRecorder.h".
2. At the beginning of main() near line 73, add this line: EventRecorderInitialize (EventRecordAll, 1);
3. Inside the while(1) loop in main(); add this line: printf("msTicks = %d\n", msTicks);

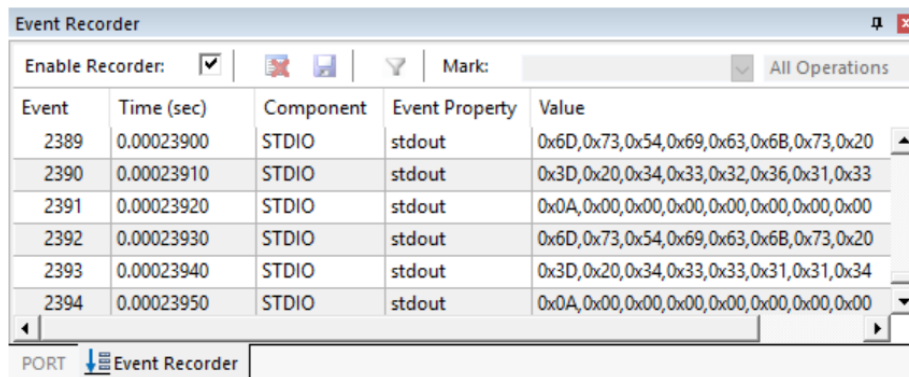
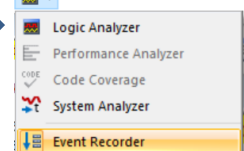
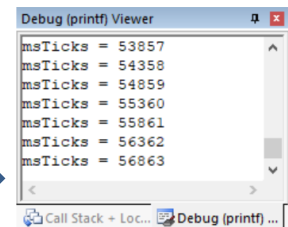
Increase the Stack and Heap:

1. In the Project window, double-click on startup_SAM L10.s to open it.
2. Select the Configuration Wizard tab at the bottom of this window.
3. Change Stack to 0x300 bytes and Heap to 0x100 bytes.



Compile and Run the Project:

1. Select File/Save All or click .
2. Rebuild the source files  and enter Debug mode .
3. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
4. In the Debug (printf) Viewer you will see the printf statements appear as shown here: .
5. Right click on the Debug window and select Mixed Hex ASCII mode. Note other useful settings that are available.
6. Select the small arrow beside the Analysis window icon and select Event Recorder. .
7. Event Recorder opens and the ASCII printf frames are displayed as shown below:
8. It is possible to annotate your code with Event Recorder with your own messages.
9. See www.keil.com/pack/doc/compiler/EventRecorder/html/
10. Stop the program  and exit Debug mode .



| Event | Time (sec) | Component | Event Property | Value |
|-------|------------|-----------|----------------|---|
| 2389 | 0.00023900 | STDIO | stdout | 0x6D,0x73,0x54,0x69,0x63,0x6B,0x73,0x20 |
| 2390 | 0.00023910 | STDIO | stdout | 0x3D,0x20,0x34,0x33,0x32,0x36,0x31,0x33 |
| 2391 | 0.00023920 | STDIO | stdout | 0x0A,0x00,0x00,0x00,0x00,0x00,0x00,0x00 |
| 2392 | 0.00023930 | STDIO | stdout | 0x6D,0x73,0x54,0x69,0x63,0x6B,0x73,0x20 |
| 2393 | 0.00023940 | STDIO | stdout | 0x3D,0x20,0x34,0x33,0x33,0x31,0x31,0x34 |
| 2394 | 0.00023950 | STDIO | stdout | 0x0A,0x00,0x00,0x00,0x00,0x00,0x00,0x00 |

17) Power Measurement using a Keil ULINKplus:

Power Measurement is provided by Keil ULINKplus. See www.keil.com/ulinkplus. The SAM L10, being a Cortex-M23, does not have Serial Wire Viewer. This means some ULINKplus functions are not provided.


Getting the ULINKplus working on the SAM L10 board with SWD (Serial Wire Debug):

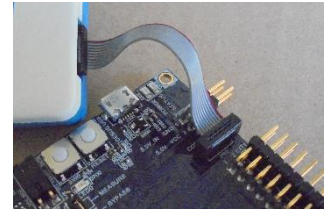
In this section, we will confirm the ULINKplus is connected to the SAM L10 processor debug connection SWD.






Load the Blinky_ULINKplus Example Project:

1. Obtain the file Blinky_ULINKplus.zip from www.keil.com/appnotes/docs/apnt_313.asp
2. Extract and save it in C:\00MDK\mdk\boards\Microchip\SAML10-XPRO\
3. Select Project/Open Project and navigate to C:\00MDK\mdk\boards\Microchip\SAML10-XPRO\
4. Open Blinky_ULINKplus.uvprojx. Double-click it or select Open to load it.


Connect the debug cable to the SAM L10 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 L10 board with a USB cable from your PC to J200 DEBUG USB.
4. Select ULINKplus from the Select Target dialog box: 
5. If you do not see ULINKplus in this box, create your own such Target with these steps:








- 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: 

Confirm ULINKplus is connected to the SAM L10 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.

| SW Device | | |
|-----------|------------|---------------------|
| | IDCODE | Device Name |
| SWDIO | 0x0BF11477 | ARM CoreSight SW-DP |

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 yellow LED LED0 will blink as it did on page 7, then when using the on-board EDBG debug adapter.
5. At this point, we have the ULINKplus acting as the debug adapter controlling the program and the board. The on-board EDBG debug adapter is bypassed. Now, we can connect the ULINKplus power measurement cables to the SAM L10 Xplained board. See the next page for these instructions.
6. Stop the program  and exit Debug mode .

18) Connecting the ULINKplus Power Measurement Connections:



The SAM L10 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 L10 Xplained board: Measured current using Blinky example is VCC ~ 0.32 ma and I/O maximum about 2.1 ma with the LED on.

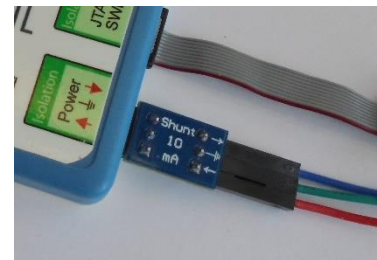
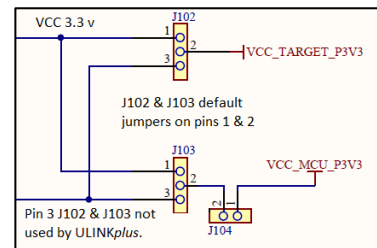
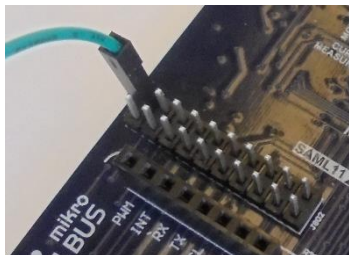
We will measure the current on the I/O 3v3 power rail. This includes the LED LED0.

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, Set_Ulinkplus.ini is provided to configure ULINKplus.



Connect ULINKplus Power pins to SAM L10 Board:

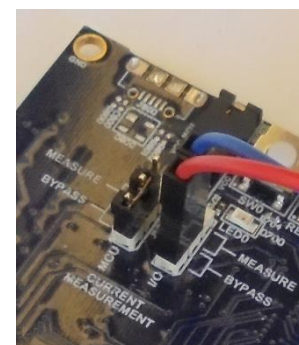
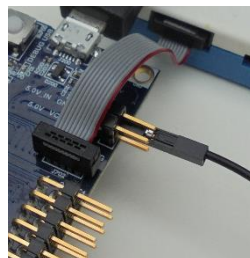
1. Remove the jumper on J102 I/O.
2. Plug the Shunt 10 ma board ULINKplus as shown here:
3. Connect three jumper cables to the Shunt board.
4. Connect pin  to pin 1 JP102 I/O connector.
5. Connect pin  to pin 2 JP102 I/O connector.
6. Connect the ground jumper wire to GND on J100. You can also use pins 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.

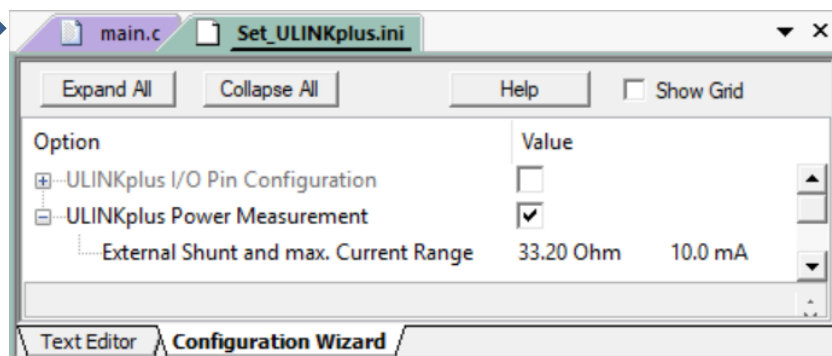
Configure ULINKplus with the .ini file:

1. Select the Target Options icon .
2. Select the Debug tab.
3. In Initialization file box, select SET_ULINKplus.ini using the Browse button.
4. Click Edit to open this file in the source files area.
5. Click OK to return to the main μ Vision menu.
6. Select the SET_ULINKplus.ini tab in the Source windows area to bring it into focus.
7. Select the Configuration Wizard tab at the bottom of this file.
8. Select 33.20 10 mA as shown here:
9. Click Save All: .



TIP: This .ini file is provided with Blinky. Debug_UlinkPlus.ini is also 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 .

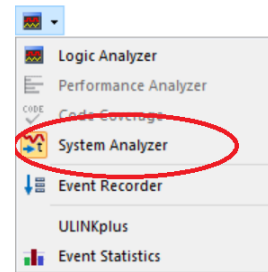


What we have to this point:

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 SAM L10 Xplained board.

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

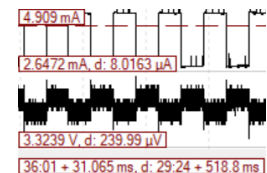
1. Build the files. Enter Debug mode: Click on RUN.
2. Open the System Analyzer (SA) window by selecting the small arrow beside Analysis Window icon as shown here:
3. Position SA as appropriate. To scroll to the waveform, click: Jump to End:
4. Use Zoom icons to size the waveform. You can also use a 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 SAM L10 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.

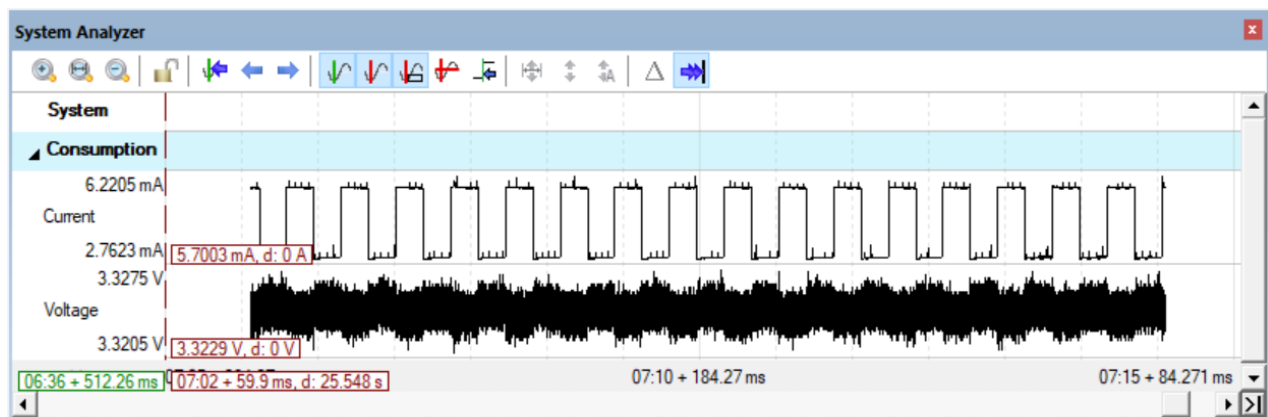
7. The square wave in the Current is obviously the LED turning off and on.

8. You can use the cursors to determine current, voltage and times wherever you hover your mouse.



9. You can stop the SA window while leaving the program running with the Lock icon:
10. With SA locked, you can right click on the Current or Voltage display and select Lo-pass filter:

TIP: 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.



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/

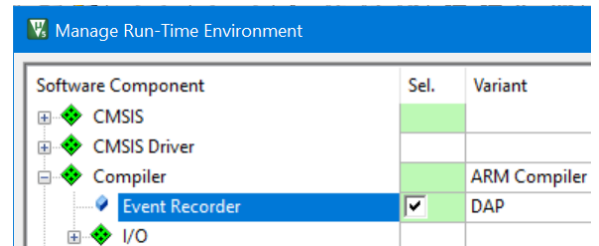
| ULINKplus | | | | |
|-------------------|---------------|-------------------------------------|------------|--|
| I/O | | | | |
| Pin | Function | Out | In | |
| 0 | Digital Input | <input type="checkbox"/> | Low | |
| 1 | Digital Input | <input type="checkbox"/> | Low | |
| 2 | Digital Input | <input 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"/> | | |

20) Displaying Event Statistics:






You can collect statistics from the ULINKplus Power measurement with Start and Stop functions. Data will be displayed in both Event Statistics and Event Recorder windows. www.keil.com/pack/doc/compiler/EventRecorder/html/es_use.html

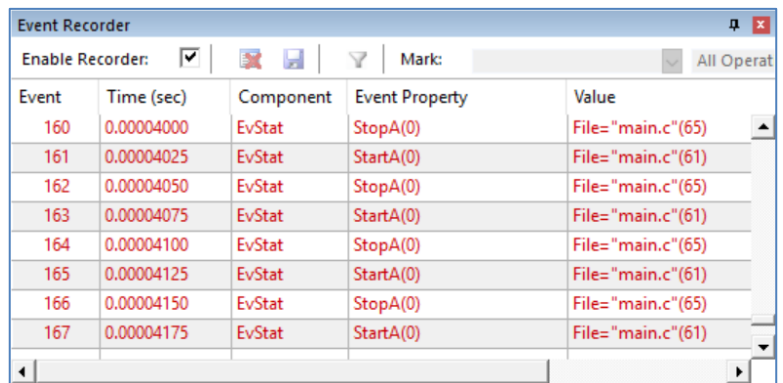
Configure Event Recorder:

1. Stop the program  and exit Debug mode .
2. Select Manage Run-Time Environment:  Expand as shown:
3. Select Event Recorder and click OK to close MRTE.
4. In main.c right click near line 7, and select Insert '#include file' and select EventRecorder.h.
5. In main.c near line 92 just before the while(1) loop, add this line: EventRecorderInitialize (EventRecordAll, 1);

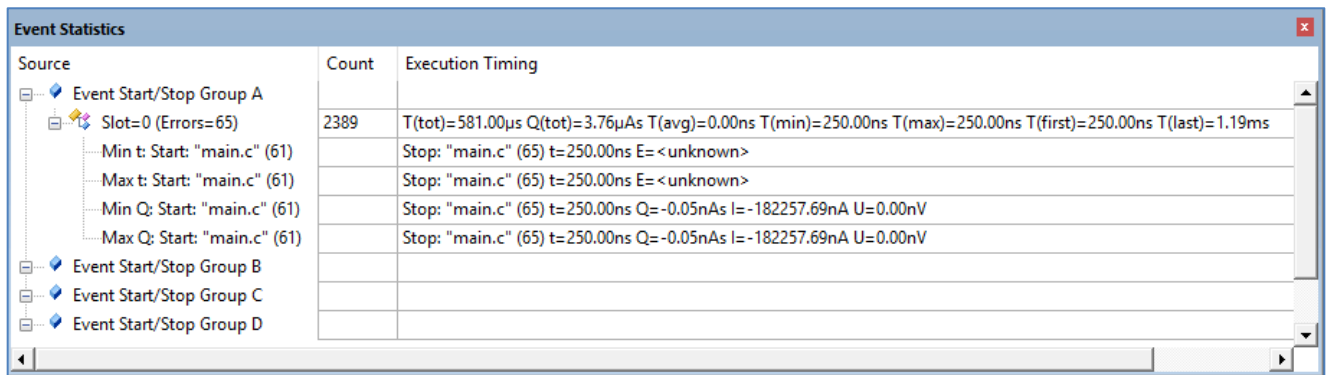


Instrument LED_ON() Function in main.c:

1. At the beginning of the function LED_ON near line 61, add this line: EventStartA(0);
2. Inside the end of LED_ON near line 65, add this line: EventStopA(0);
3. These are for Group A and Slot 0.
4. Click Save All:  Build the files. 
5. Enter Debug mode:  Click on RUN. 
6. The System Analyzer will display current and voltage waveforms as before.
7. Select Event Recorder to open it: 
8. Event Statistics will be displayed. The Start and Stop events are shown.
9. Select Event Statistics to open it. This window is displayed below:
10. You can see various timings (T) and accumulated power consumption (Q) in nAs.

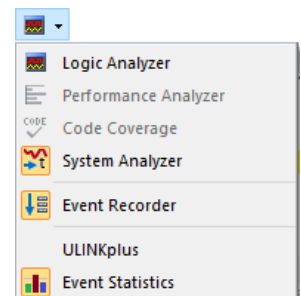


| Event | Time (sec) | Component | Event Property | Value |
|-------|------------|-----------|----------------|-------------------|
| 160 | 0.00004000 | EvStat | StopA(0) | File="main.c"(65) |
| 161 | 0.00004025 | EvStat | StartA(0) | File="main.c"(61) |
| 162 | 0.00004050 | EvStat | StopA(0) | File="main.c"(65) |
| 163 | 0.00004075 | EvStat | StartA(0) | File="main.c"(61) |
| 164 | 0.00004100 | EvStat | StopA(0) | File="main.c"(65) |
| 165 | 0.00004125 | EvStat | StartA(0) | File="main.c"(61) |
| 166 | 0.00004150 | EvStat | StopA(0) | File="main.c"(65) |
| 167 | 0.00004175 | EvStat | StartA(0) | File="main.c"(61) |



| Source | Count | Execution Timing |
|-----------------------------|-------|---|
| Event Start/Stop Group A | | |
| Slot=0 (Errors=65) | 2389 | T(tot)=581.00µs Q(tot)=3.76µAs T(avg)=0.00ns T(min)=250.00ns T(max)=250.00ns T(first)=250.00ns T(last)=1.19ms |
| Min t: Start: "main.c" (61) | | Stop: "main.c" (65) t=250.00ns E=<unknown> |
| Max t: Start: "main.c" (61) | | Stop: "main.c" (65) t=250.00ns E=<unknown> |
| Min Q: Start: "main.c" (61) | | Stop: "main.c" (65) t=250.00ns Q=-0.05nAs I=-182257.69nA U=0.00nV |
| Max Q: Start: "main.c" (61) | | Stop: "main.c" (65) t=250.00ns Q=-0.05nAs I=-182257.69nA U=0.00nV |
| Event Start/Stop Group B | | |
| Event Start/Stop Group C | | |
| Event Start/Stop Group D | | |

11. These values represent those measured between the Start and Stop function calls.
12. These values update in real-time while the program is running and if System Analyzer is locked or unlocked.



21) More Features with Event Statistics: Passing Up to Two Variables:

There are 4 Groups G = A, B, C and D. In each of these groups are 16 Slots S = 0...15. A EventStop of S = 15 will stop all slots in that group. See documentation at www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr_stat.htm

There are four versions of the EventStart and EventStop function calls:

No Variables Passed:

EventStartG(S);

EventSTOPG(S); // Start and stop Event Statistics. G = Group = A, B, C and D. S = Slot – 0 through 15

Examples:

EventStartA(3); Start event Group A Slot 3.

EventStopA(6); Stop event Group A Slot 6.






Two Variables Passed:

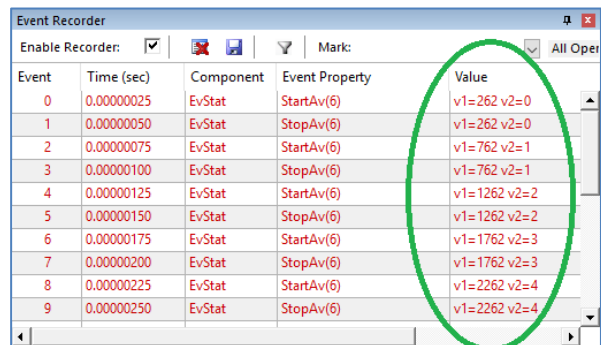
EventStartGv(S, v1, v2); // v = pass variable(s). Variables are labeled v1 and v2 respectively. G = Group, S = Slot number.

Examples:

EventStartC(5, msTicks, counter); //Two variables, msTicks and counter are displayed in Event Recorder and Event Statistics.

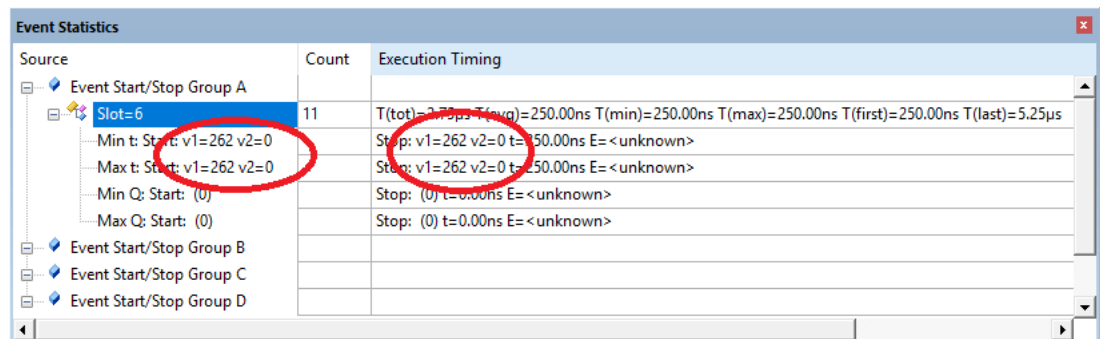
Modify the Start and Stop functions:

- 1) Stop the program  and exit Debug mode .
- 2) Change these two statements in the function LED_ON found near line 60 in main.c:
 - a) EventStartAv(0, msTicks, counter); //Group A, Slot 0, variables msTicks and counter
 - b) EventStopAv(0, msTicks, counter);
- 3) Click on File/Save All or .
- 4) Build the files. .
- 5) Click on RUN. .
- 6) In the Event Recorder and Event Statistics windows, note the value of each variable is displayed as v1 and v2 respectively.



| Event | Time (sec) | Component | Event Property | Value |
|-------|------------|-----------|----------------|--------------|
| 0 | 0.00000025 | EvStat | StartAv(6) | v1=262 v2=0 |
| 1 | 0.00000050 | EvStat | StopAv(6) | v1=262 v2=0 |
| 2 | 0.00000075 | EvStat | StartAv(6) | v1=762 v2=1 |
| 3 | 0.00000100 | EvStat | StopAv(6) | v1=762 v2=1 |
| 4 | 0.00000125 | EvStat | StartAv(6) | v1=1262 v2=2 |
| 5 | 0.00000150 | EvStat | StopAv(6) | v1=1262 v2=2 |
| 6 | 0.00000175 | EvStat | StartAv(6) | v1=1762 v2=3 |
| 7 | 0.00000200 | EvStat | StopAv(6) | v1=1762 v2=3 |
| 8 | 0.00000225 | EvStat | StartAv(6) | v1=2262 v2=4 |
| 9 | 0.00000250 | EvStat | StopAv(6) | v1=2262 v2=4 |

This is the end of the exercises with the Event Statistics window.



| Source | Count | Execution Timing |
|---------------------------|-------|--|
| Event Start/Stop Group A | | |
| Slot=6 | 11 | T(tot)=2.75ps T(avg)=250.00ns T(min)=250.00ns T(max)=250.00ns T(first)=250.00ns T(last)=5.25ps |
| Min t: Start: v1=262 v2=0 | | Stop: v1=262 v2=0 t=250.00ns E= <unknown> |
| Max t: Start: v1=262 v2=0 | | Stop: v1=262 v2=0 t=250.00ns E= <unknown> |
| Min Q: Start: (0) | | Stop: (0) t=0.00ns E= <unknown> |
| Max Q: Start: (0) | | Stop: (0) t=0.00ns E= <unknown> |
| Event Start/Stop Group B | | |
| Event Start/Stop Group C | | |
| Event Start/Stop Group D | | |

22) Creating your own MDK 5 project from scratch:

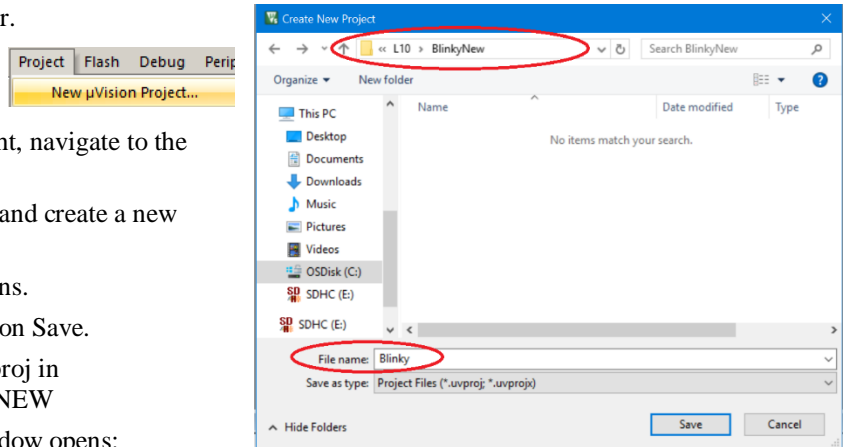
All examples provided by Keil are pre-configured. All you need to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run a simple Blinky example. 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.

Install the SAM L10 Software Pack for your processor:

1. Start μ Vision and leave in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Pack for the SAM L10 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 Folder and a New Project:

1. Click on Project/New μ Vision Project...
2. In the window that opens, shown to the right, navigate to the folder C:\00MDK\Boards\Microchip\L10\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Select Open and the folder BlinkyNew opens.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the MDK 4 project Blinky.uvproj in C:\00MDK\Boards\Microchip\L10\BlinkyNEW
7. As soon as you click on Save, the next window opens:

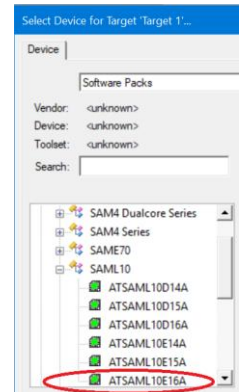


Select the Device you are using:

1. Expand Microchip, then SAM L10, then finally select ATSAM L10E16A as shown here:
2. Click OK. Blinky.uvproj (MDK 4) will now be changed to Blinky.uvprojx (MDK 5).
3. The Manage Run-Time window shown below bottom right opens.

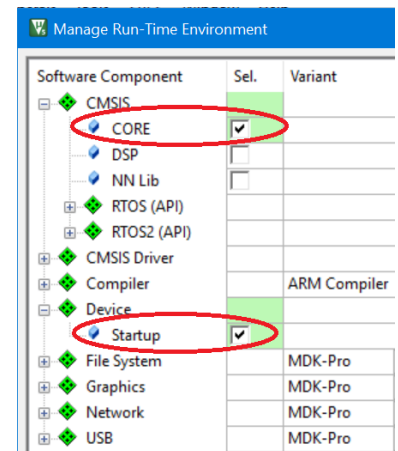
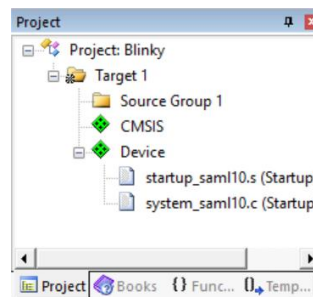
Select the CMSIS components you want in the MRTE:

1. Expand CMSIS and select Core.
2. Expand Device and select Startup as shown below.
3. They will be highlighted in Green indicating there are no other files needed. Click OK.
4. Click on File/Save All or select the Save All icon:
5. 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.
6. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
7. Click once on it and change its name to EDBG CMSIS-DAP Flash and press Enter. The Target selector name will also change.



What has happened to this point:


You have created a blank μ Vision project using MDK 5 Software Packs. All you need to do now is add your own source files. This is shown on the next page:

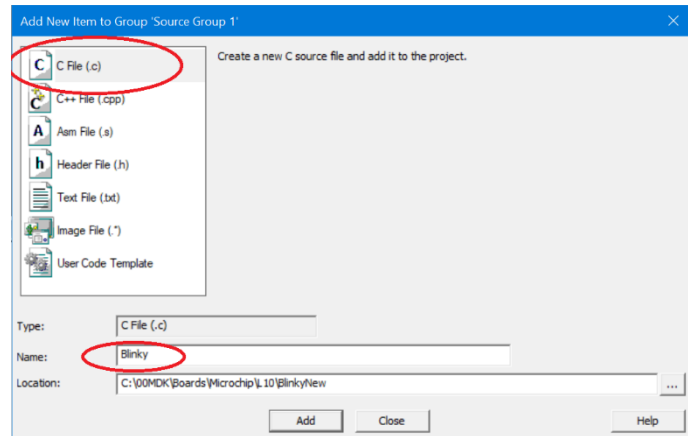


Create a blank C Source File:



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

Add New Item to Group 'Source Files'...


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.




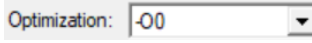





Add Some Code to Blinky.c:

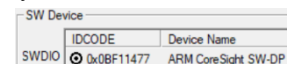
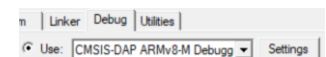
1. In the blank Blinky.c, add the C code below:
2. Click on File/Save All or 
3. Build the files.  There will be no errors and 4 not important warnings if all was entered correctly.

```
1 #include "sam.h" // Device header
2
3 extern unsigned int counter;
4 unsigned int counter = 0;
5
6 #if defined(__ARMCC_VERSION) && (__ARMCC_VERSION >= 6010050) // for Optimize -O0
7 __asm(" .global __ARM_use_no_argv\n");
8 #endif
9
10 int main(void){
11
12     while(1){
13         counter++;
14         if (counter > 0x0F) counter = 0;
15     }
16 }
```

TIP: You can also add existing source files:  No need to do this at this time.

Configure the Target EDBG CMSIS-DAP Flash: *Please complete these instructions carefully and completely:*





1. Select the Target Options icon .
2. Select the C/C++ AC6 tab.
3. Select Optimization -O0 as shown here: 
4. Select the Debug tab. Select CMSIS-DAP ARMv8-M Debugger in the Use: box:
5. Select the Settings: icon.
6. Select SW as shown here in the Port: box:  JTAG will not work with SAM L10. If your board is connected to your PC, you **must** now see a valid IDCODE and Device Name in the SW Device box. 
7. If you do not see this: you **must** fix this before you can continue. Make sure the Discovery board is connected.
8. Click on OK **once** to go back to the Target Configuration window once you have confirmed the connection is working.
9. **Select the Utilities tab.** Select Settings and select Add. Select the correct Flash algorithm: Shown below is the correct one for the SAM L10 series processors: 
10. Click on OK twice to return to the main menu.
11. Click on File/Save All or 
12. Build the files.  There will be no errors and 4 warnings if all was entered correctly. If not, please fix them !

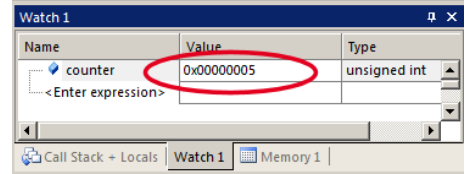


| Programming Algorithm | | | |
|-----------------------|-------------|---------------|-----------------------|
| Description | Device Size | Device Type | Address Range |
| ATSAML10 64kB Flash | 64k | On-chip Flash | 00000000H - 0000FFFFH |

The Next Step ? Let us run your program and see what happens ! Please turn the page....

Running Your Program:


1. Enter Debug mode by clicking on the Debug icon .  Flash memory will be programmed with progress indicated.
2. Click on the RUN icon.  **Note:** you stop the program with the STOP icon. 
3. No LEDs will blink since there is no source to accomplish this task. You could add such code yourself.
4. Right click on counter in Blinky.c and select Add counter to ... and select Watch 1.
5. counter will be updating as shown here: 
6. 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.
7. You could now be able to add your own source code to create a meaningful project. See the Blinky example.




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. Normally some sort of delay routine is used to slow this down.

Hints using this Tutorial with Arm Compiler 6 (AC6):

1. **Arm Compiler 6 (AC6) is used with all Armv8-M processors. AC5 does not support Armv8-M.**
2. **Migrating from AC5 to AC6:** www.keil.com/appnotes/docs/apnt_298.asp
3. **Optimization:** Using -O1 is recommended for general debugging with AC6.
4. **To allow the use of Optimization level -O0: (instead of -O1) Debugging is easier with -O0.**

```
1. #if defined(__ARMCC_VERSION) && (__ARMCC_VERSION >= 6010050) // for Optimize -O0
2. __asm(" .global __ARM_use_no_argv\n");
3. #endif
```
5. **Heap:** Add heap if printf or MicroLIB is used. Errors or warnings are generated. Heap in startup_SAM L10.s file.
6. **MicroLIB** is an effective way to produce significantly smaller code.
Select the Target Options icon . Select the Target tab and select/unselect Use MicroLIB.
7. **printf** sometimes requires use of Heap value in startup_SAM L10.s file.
8. **Declaring Variables:**

```
extern unsigned int counter;
unsigned int counter =0;
```



The second statement provides initialization of the variable which is a good idea. Uninitialized variables can sting.
9. **Hard Fault:** Try more Stack Pointer and/or Heap. These are found in startup_SAM L10.s.
10. **RTX:** Add more Default Thread Stack size in RTX_config.h. Use Configuration Wizard tab to select these easily.
11. **Preventing infinite loops from being optimized away:** Put this line in the loop: `__asm volatile("");`
12. **Functions:** It is a good idea to create a function prototype for your functions with AC6.
13. **Warnings:** AC6 creates more warnings than AC5 does. It is useful to turn warnings off so you can focus on errors.
Select the Target Options . Select the C/C++ AC6 tab and in the Warnings box, select AC5-like Warnings.

23) Adding Keil RTX5 to your Project:

The MDK Software Packs makes it easy to configure an RTX project. RTX 5 is ported to ArmV8-M processors. RTX5 has an Apache 2.0 license. RTX5 is located inside MDK here and here: https://github.com/ARM-software/CMSIS_5.

TIP: Keil documentation uses the term “threads” instead of “tasks” for consistency.

Enable RTX:


1. Using the same example from the preceding pages, Stop the program  and Exit Debug mode. 

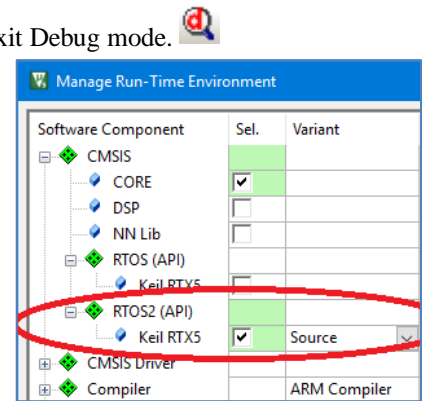
2. Open the Manage Run-Time Environment window: 

3. Expand all the elements as shown here: 

4. In RTOS2, select Keil RTX5 and Source as shown and click OK.

5. Appropriate RTX files will be added to your project. See the Project window.

6. Click on File/Save All or 



Add Source Lines to Blinky.c:

1. In Blinky.c, at the top near line 2, right-click and select: Insert ‘#include file’.

2. Select this line: #include "cmsis_os2.h"

Create a Thread named *thr_counter*:

1. Add the thread named *thr_counter* plus its prototype starting at line 11:

```
11 void thr_counter(void *argument); // thread prototype
12
13 __NO_RETURN void thr_counter(void *argument){
14     (void) argument;
15
16     for(;;) {
17         counter++;
18         if(counter > 0x0F) counter = 0;
19     }
20 }
21
```

Configure RTX in main() and Create the Thread *thr_counter*:


1. In the main() function, remove the two lines containing the counter increment and test.


2. In the main() function add these lines 24 through 30:

```
23 int main(void){
24     osKernelInitialize();
25     osThreadNew(thr_counter,NULL,NULL); //create the thread thr_counter
26     osKernelStart(); // start the thread execution
27
28     while(1){
29     }
30 }
```


3. Click on File/Save All or 

Build and Run Your RTX Program:

1. Build the files.  There will be no errors and many non-important warnings. (you can turn these off if you want)

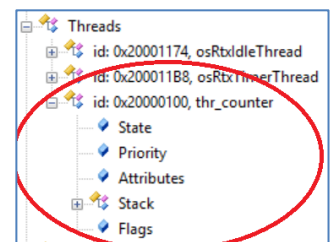
2. Enter Debug mode:  Click on RUN.  counter will still increment in Watch1 but now it is located in an RTX5 thread.

3. Select View/Watch Windows and select RTX RTOS. The RTX RTOS window opens.

4. Expand Threads and you can see the thread *thr_counter* as shown here: 



5. If you add threads, they will be automatically listed in this window.

6. This window updates in real-time as the program runs using DAP.



24) Adding a Thread to your RTX5 project:

It is easy to add threads to a project. This adds a simple thread that increments a variable counter2. These steps use the same configuration as in the preceding Blinky RTX5 example.

1. Using the same example from the preceding pages, Stop the program  and Exit Debug mode. 

Add the new Thread thr_counter2 source code:

1. In Blinky.c, add these lines near line 6 to declare a global variable counter2:

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

2. Near line 14, add this prototype for the new thread thr_counter2:

```
14     void thr_counter(void *argument);
```

3. After the first thread thr_counter, add this code for the thread thr_counter2:


```
26     __NO_RETURN void thr_countere2 (void *argument) {
27
28         (void) argument;
29
30         for(;;) {
31             counter2++;
32             if(counter2 > 0x0F) counter2 = 0;
33         }
34     }
```

Create the new Thread:




1. In Blinky.c, just after the osThreadCreate for thr_counter line: add this line near line 39:

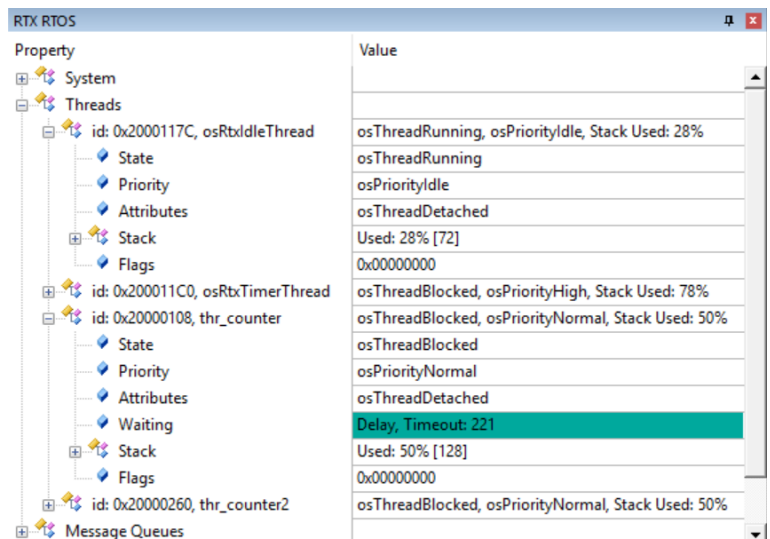
```
39     osThreadNew(thr_counter2, NULL, NULL);    //create the thread thr_counter2
```

Add a Delay to each Thread:

1. Add this line just after the if statement in the while(1) loop in each of the two threads: osDelay(500);
2. Click on File/Save All or 

Build and Run Your RTX Program:

1. Build the files.  There will be no errors and many non-important warnings. If there are errors, please repair them.
2. Enter Debug mode:  Click on RUN. 
3. Right click on the variable counter2 in Blinky.c and select 'add counter2' to Watch 1.
4. Open the RTX RTOS window.
5. Counter and counter2 are now being incremented inside the two threads as each thread is selected Running by the RTX kernel.
6. Stop the program and it will probably be in the osRtxIdleThread function in RTX_Config.c. This program spends most of its time in the idle daemon. You can change this.
7. Set a breakpoint in each thread to convince yourself these are being executed.
8. Open RTX_Config.h and click on the Configuration Wizard tab at the bottom. Note how you can configure RTX in this window.
9. Clear any breakpoints set.



The screenshot shows the RTX RTOS window with a tree view on the left and a table of values on the right.






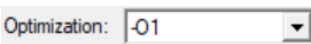

| Property | Value |
|----------------------------------|--|
| System | |
| Threads | |
| id: 0x2000117C, osRtxIdleThread | osThreadRunning, osPriorityIdle, Stack Used: 28% |
| State | osThreadRunning |
| Priority | osPriorityIdle |
| Attributes | osThreadDetached |
| Stack | Used: 28% [72] |
| Flags | 0x00000000 |
| id: 0x200011C0, osRtxTimerThread | osThreadBlocked, osPriorityHigh, Stack Used: 78% |
| State | osThreadBlocked, osPriorityNormal, Stack Used: 50% |
| Priority | osThreadBlocked |
| Attributes | osPriorityNormal |
| Waiting | osThreadDetached |
| Stack | Delay, Timeout: 221 |
| Flags | Used: 50% [128] |
| id: 0x20000260, thr_counter2 | 0x00000000 |
| Message Queues | osThreadBlocked, osPriorityNormal, Stack Used: 50% |

25) Event Recorder:





Event Recorder is a new μ Vision feature. Code annotations can be inserted into your code to send out messages to μ Vision and be displayed as shown below. Keil Middleware and RTX5 have these annotations already inserted. You can add Event Recorder annotations to your own source code. You already saw Event Recorder working on page 13.

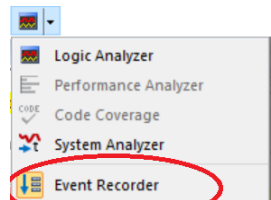
Documentation for Event Recorder is found here: www.keil.com/pack/doc/compiler/EventRecorder/html/

Demonstrating Event Recorder with RTX5_Blinky:



1. Stop the program  and Exit Debug mode. 
2. Open the Manage Run-Time Environment window (MRTE) 
3. Expand Compiler as shown here: 
4. Select Event Recorder and then DAP as shown:
5. Click OK to close the MRTE. Event Recorder files are added to your project under the Compiler header.
6. Near the top of Blinky.c, right-click and select Insert '#include' and select #include "EventRecorder.h"
7. As the **very first** line in main() in Blinky.c, add this line: EventRecorderInitialize (EventRecordAll, 1);
8. Select the Target Options icon 
9. Select the C/C++ AC6 tab.
10. Select Optimization -O1 as shown here: 
11. Click OK to close and return to the main μ Vision menu.
12. Click on File/Save All or 

Build and Run Your RTX Program:

1. Build the files.  There will be no errors and many non-important warnings. If there are any errors, please repair them before continuing.
2. Enter Debug mode:  Click on RUN. 
3. Open Event Recorder by selecting View/Analysis/Event Recorder or 
4. The Event Recorder opens and will display messages from RTX as shown here:



| Event Recorder | | | | |
|--|------------|--|--------------------------------------|------------------------------------|
| Enable Recorder: <input checked="" type="checkbox"/> | | Mark: <input type="checkbox"/> All Operations <input type="checkbox"/> Stopped | | |
| Event | Time (sec) | Component | Event Property | Value |
| 36 | 0.00000700 | RTX Thread | ThreadUnblocked | thread_id=0x20000720, ret_val=osOK |
| 37 | 0.00000710 | RTX Thread | ThreadPreempted | thread_id=0x2000163C |
| 38 | 0.00000720 | RTX Thread | ThreadSwitched | thread_id=0x20000720 |
| 39 | 0.00000730 | RTX Thread | ThreadDelay | ticks=500 |
| 40 | 0.00000740 | RTX Thread | ThreadBlocked | thread_id=0x20000720, timeout=500 |
| 41 | 0.00000750 | RTX Thread | ThreadSwitched | thread_id=0x2000163C |
| 42 | 0.00000760 | RTX Thread | ThreadDelayCompleted | |
| 43 | 0.00000770 | RTX Thread | ThreadUnblocked | thread_id=0x200005C8, ret_val=osOK |
| 44 | 0.00000780 | RTX Thread | ThreadPreempted | thread_id=0x2000163C |
| 45 | 0.00000790 | RTX Thread | ThreadSwitched | thread_id=0x200005C8 |

5. Stop the program. 
6. Exit Debug mode. 

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

26) 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

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>

27) 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. The Cortex-M23 does not have either SWV or ETM.

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. It is the full version of RTX – it is not restricted or crippled.
- 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/>

