

Introduction:

The purpose of this lab is to introduce you to the NXP Cortex™-M33 processor family using the ARM Keil MDK toolkit featuring the µVision® IDE. At the end of this tutorial, you will work confidently with NXP processors and Keil MDK™.

Keil MDK has an evaluation version that limits code and data size to 32 Kbytes. Most Keil examples are under this 32K. The addition of a license number will turn it into a full, unrestricted version. Contact Keil sales for a temporary full version license to evaluate MDK with larger programs or Keil Middleware. MDK includes a full version of Keil RTX™ RTOS with all source code included. See www.keil.com/NXP for information concerning Keil support of NXP products.

Why Use Keil MDK ?

MDK provides these features particularly suited for NXP Cortex-M0, M0+, M3, M4, M7, M23 and Cortex-M33 processor users:

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is turn-key "out-of-the-box".
2. ARM Compiler 5 and Compiler 6 (LLVM) are included. GCC is supported. <https://launchpad.net/gcc-arm-embedded>
3. Dynamic Syntax checking on C/C++ source lines.
4. **Keil Middleware:** Network, USB, Flash File and Graphics.
5. CMSIS-RTOS RTX is included. www.keil.com/RTX.
6. **NEW! Event Recorder for Keil Middleware.**
7. MISRA C/C++ support using PC-Lint. www.gimpel.com
8. **Compiler Safety Certification Kit:** www.keil.com/safety/ **NEW ! FuSa RTS** www.keil.com/fusa-rts
9. TÜV certified. ISO 26262 ASIL D, IEC62304 Class C, IEC 61508 SIL 3, EN50128 SIL 4
10. CoreSight™ Serial Wire Viewer and ETM trace capability on appropriately equipped NXP processors.
11. Choice of adapters: LPC-Link2 (CMSIS-DAP), ULINK™2, ULINK-ME, ULINKpro, ULINKplus and J-Link.
12. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
13. Affordable perpetual and term licensing. Contact Keil sales for pricing, options and current special offers.



Significant features shown in this document:

1. A Blinky example showing basic operation of µVision and its debugging capabilities.
2. Serial Wire Viewer (SWV) with the ULINK2, ULINK-ME, ULINKpro, ULINKplus and J-Link.
3. Real-time Read and Write memory accesses for Watch, Memory, Peripheral and RTX Tasks windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
4. A TrustZone example showing a function call from NS space to S space and then returning back to NS space.
5. Keil Event Recorder. Power Measurement with ULINKplus

Serial Wire Viewer (SWV): Use a ULINK2, ULINKpro, ULINKplus or a J-Link for this feature.

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, printf (ITM), CPU counters and a timestamp. SWV is non-intrusive and no code stubs are needed in your source code.

Keil Event Recorder (EVR):

Event Recorder (EVR) is a system to annotate your source code and display messages and data in various windows. If used with a Keil ULINKplus, Power Measurement is added. www.keil.com/mdk5/debug/eventrecorder

TIP: A MDK-Pro license is needed to compile Armv8-M Cortex-M23 and Cortex-M33 projects. Please contact Keil Sales for a complimentary license. Contact information is on the last two pages.

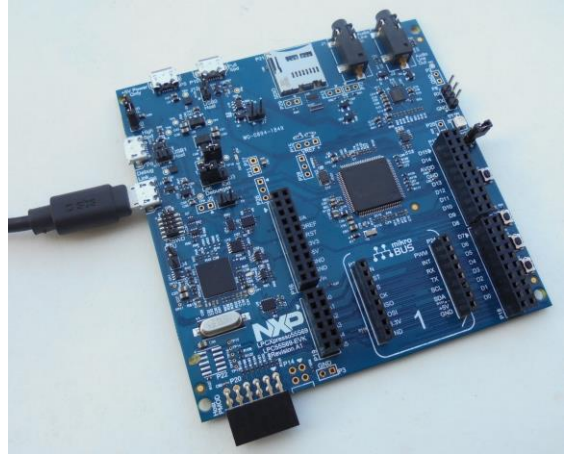
General Information, Hardware and Software:	3
1. NXP Evaluation Boards, Summary, Keil Software Installation:	3
2. Software Pack Installation Process:	4
3. Copy the Examples to your PC using Pack Installer:	5
4. Software Pack Version Selection and Manage Run-Time Environment:	6
5. CoreSight Definitions:	8
6. External Debug Adapter Types:	9
7. External Debug Adapter Configuration:	10
 Blinky Example Program using LPC55S69:	 11
1. Open, Compile and RUN Blinky:	11
2. Hardware Breakpoints:	12
3. Call Stack + Locals Window:	13
4. Watch and Memory Windows and how to use them:	14
5. System Viewer (Peripherals):	15
6. Watchpoints: Conditional Breakpoints:	16
7. Printf without using a UART:	17
 hello_world TrustZone Example Program using LPC55S69:	 18
1. Armv8-M TrustZone example hello_world	18
2. Investigating Secure and Non-Secure Modes:	19
3. Examining and Modifying TrustZone Project Files:	20
4. Switch from Secure to Non-Secure: Call a function in Secure Mode:	20
5. A Non-Secure program calling a function in Secure Space:	21
6. Blank Page:	22
 General Information:	 23
1. Document Resources:	23
2. Keil Products and Contact Information:	24

1) NXP ARM Cortex Processor Evaluation Boards:

This tutorial is for the NXP LPCXpresso55S69 evaluation board. You can adapt this for other members of the LPC55xx family. Keil supports and makes boards with other NXP processors. See www.keil.com/NXP for more information.

All NXP processors implement SWV with the exception of Cortex-M0 and M0+, which have neither SWV nor ETM. The Cortex-M0+ (LPC800, LPC81x, LPC82x and LPC83x) has a MTB Micro Trace Buffer. All have non-intrusive read/write to memory locations (for Watch, Memory and Peripheral windows), hardware breakpoints and Watchpoints. Many NXP Cortex-M3, M4 and M7 have ETM instruction trace which also provides Code Coverage and Performance Analysis.

NXP LPCXpresso55S69 Evaluation Board:




Summary: Five Easy Steps to Get Connected and Configured:

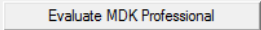
1. Obtain and install Keil MDK Core (evaluation version) on your PC. Use the default directory C:\Keil_v5\.
2. Download the Software Packs for LPC55S69 or LPC55S66 (DFP and BSP) and examples for your board.
3. Configure μ Vision to use a ULINK2, ULINKpro, ULINKplus, LPC-Link2 or a J-Link debug adapter.
4. If desired, configure the Serial Wire Viewer with a ULINK2, ULINKpro, ULINKplus or a J-Link as appropriate. LPC-Link2 in CMSIS-DAP mode does not currently support SWV.
5. If a debug adapter is used, connect it to your board. Power both of these appropriately with USB cables.

These steps are detailed in the next few pages.

Keil MDK Core Software Download and Installation: This tutorial used MDK 5.28.

1. Download MDK Core from the Keil website. www.keil.com/mdk5/install 
2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5\
3. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
4. If you install MDK or the examples into different folders, you will have to adjust for the folder location differences.

Licensing:

1. You need a MDK-Pro license for all Arm v8-M TrustZone processors including Cortex-M23 and Cortex-M33.
2. You can obtain a one-time free 7 day MDK Pro license in File/License Management. If you are eligible, this button is visible: 
3. This gives you unlimited code size compilation and access to Keil Middleware. Contact Keil Sales to extend this license for evaluation purposes.

For CMSIS-Pack documentation: www.keil.com/pack/doc/CMSIS/Pack/html/

For complete CMSIS documentation: www.keil.com/CMSIS/



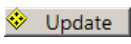
For CMSIS 5 on GitHub: https://github.com/ARM-software/CMSIS_5

2) Software Pack Installation Process:

A Software Pack is a zip file with a .pack file extension. It can contain header files, Flash programming algorithms, examples, documentation and more. The contents of a Pack are described by a .pdsc file contained inside the Pack. A DFP (Device Family Pack) contains contents for device(s) and optionally a BSP. A BSP (Board Support Package) contains information related to a specific board. A Pack is installed using the μ Vision Pack Installer utility or installed manually.

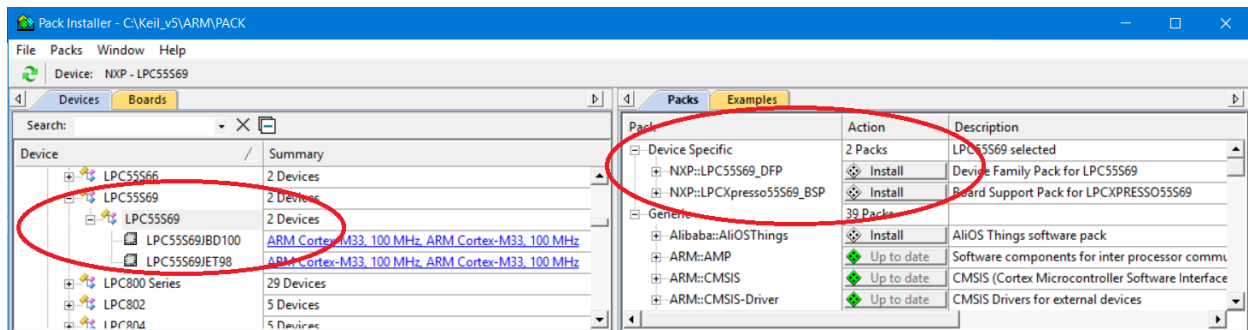
1) Start μ Vision and open Pack Installer:


When the first MDK install is complete and if you are connected to the Internet, μ Vision and Software Packs will automatically start. Otherwise, follow Steps 1 and 2 below.

1. Connect your computer to the Internet. This is normally needed to download the Software Packs unless you have a standalone Pack. In this case, import it with Pack Installer or double-click on it to manually install it.
2. Start μ Vision by clicking on its desktop icon.  The Pack descriptions will download on the initial μ Vision run.
3. Open the Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.
4. If there are any Updates available, you can download them now if they are applicable. 
5. The window below opens up: Select the Devices tab. Scroll down and expand NXP. Select LPC55S69 as shown below. You could select a specific NXP processor but in this case one Pack supports all of them at this time.
6. Alternatively, you can click on the Boards tab and select LPCXpresso55S69 to filter the Packs or Examples tabs.

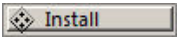
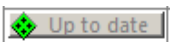
TIP: The Devices and Boards tabs are used to filter the items displayed on the right side in the Packs and Examples tabs.

7. Note: “ONLINE” is displayed at the bottom right. If “OFFLINE” is displayed, connect to the Internet.



TIP: If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or  to refresh once you have connected to the Internet. This is not done automatically.

2) Install NXP LPC54000 Device Family Pack (NXP::LPC55S69_DFP) and LPCXpresso55S69 BSP:

1. Click on the Packs tab.
2. Select the Install icon  beside NXP::LPC55S69_DFP as shown above.
3. The latest Pack will download and install to C:\Keil_v5\ARM\PACK\NXP\ by default. This download can take two to four minutes depending on your Internet connection speed.
4. Click Install on NXP::LPCXpresso55S69_BSP board support package as shown above.
5. The two Packs status will then be indicated by the “Up to date” icon: 
6. Leave Pack Installer open to copy the examples on the next page.

TIP: You can also install a Pack manually. A Pack has a file extension of .pack. It is actually an ordinary zip file with the extension changed so it is recognized by μ Vision. You can download the Pack from the web or transfer the file in any other way. Double click on this file and it will automatically be recognized (.pack) and installed by Pack Installer.

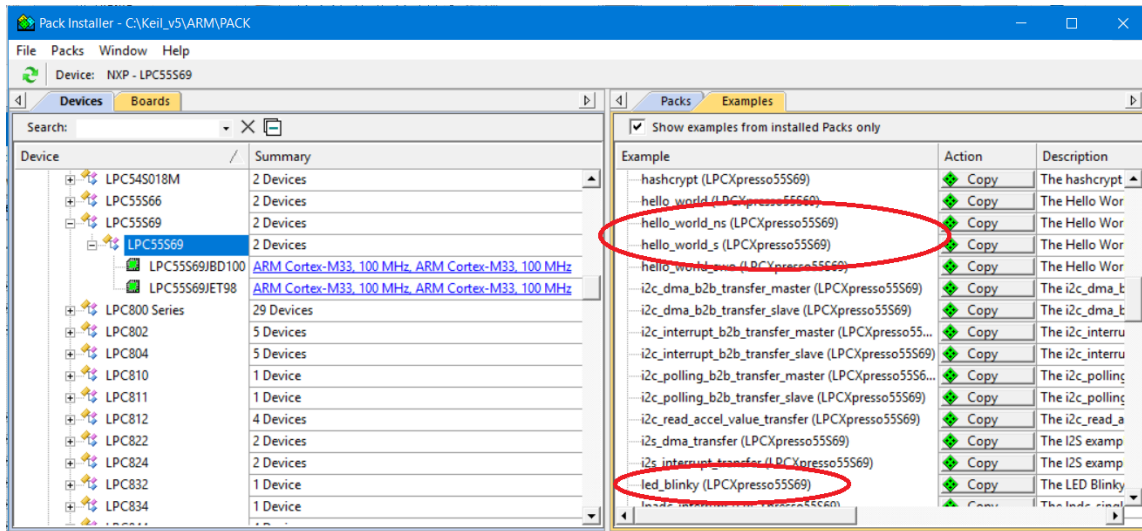
TIP: You can create your own Pack to distribute a DFP, BSP and an SDK. This is a way to distribute confidential material.




For CMSIS-Pack documentation: www.keil.com/pack/doc/CMSIS/Pack/html/

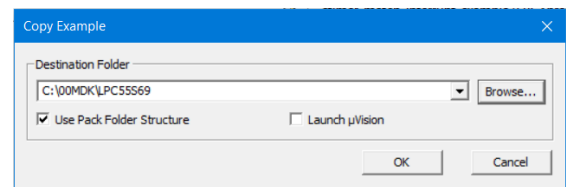
3) Copy the Examples to your PC using Pack Installer:

1) Copy led_blinky and hello_world examples:

1. Select the Boards tab. In the Search box enter LPC55S69. Highlight LPC55S69 as shown below on the left:
2. Select the Examples tab: There are many examples available. We will use two of them.



3. Beside hello_world_ns (LPCXpresso55S69) as shown above: Select Copy:  **Copy**
4. The Copy Example window below opens up: Select Use Pack Folder Structure. Unselect Launch µVision:
5. Type in C:\00MDK\LPC55S69 as shown here: 
6. Click OK to copy this example file into this folder:
7. Pack Installer creates the appropriate subfolders.
8. Repeat for hello_world_s and led_blinky.
9. Close Pack Installer. Open it any time by clicking: 
10. If a dialog box opens stating the Software Packs folder has been modified, select Yes to "Reload Packs ?"



EXAMPLES FOLDERS:

hello_world:

C:\00MDK\LPC55S69\trustzone_examples\hello_world\cm33_core0\hello_world_s\mdk\
C:\00MDK\LPC55S69\trustzone_examples\hello_world\cm33_core0\hello_world_ns\mdk\

led_blinky:

C:\00MDK\LPC55S69\demo_apps\led_blinky\cm33_core0\mdk\

TIP: The default directory for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default folder of C:\00MDK\LPC55S69\ in this tutorial. You can use any folder of your choosing.

TIP: Arm v8-M Cortex-M23 and Cortex-M33 have two modes: Secure and Non-Secure. These can be noted **n** and **ns** respectively. This nomenclature can be used for filenames, projects and variables. Other variants exist such as **_N** and **_NS**.

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

At this point, you have everything loaded and installed to run some examples and start developing your own µVision projects.

4) Software Pack Version Selection and Manage Run-Time Environment:


These three sections are provided only for reference. These three utilities are useful for managing and using Software Packs. This page is for the LPC54000. The LPC55xx series options and components are similarly configured and maintained.

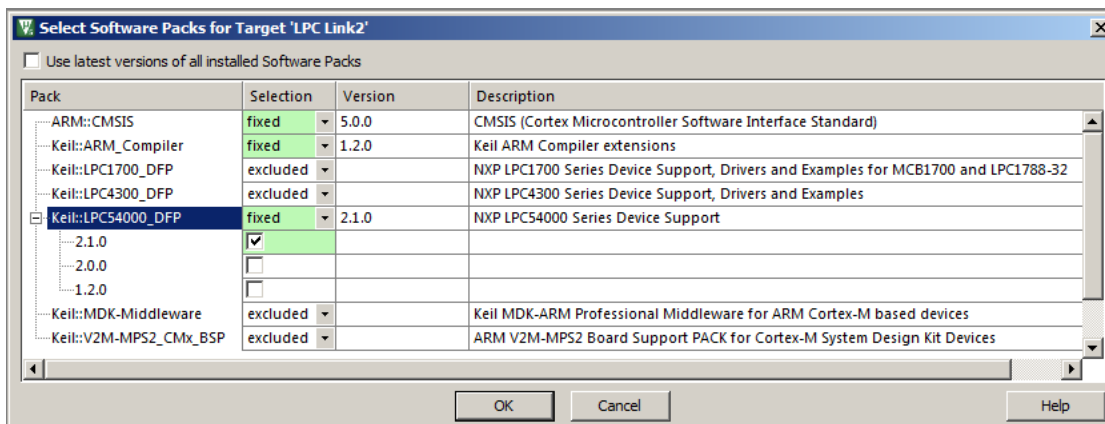
This section contains three parts on this page and the next one:

- A) Select Software Pack Version:
- B) Manage Run-Time Environment:
- C) Updating Source Files:

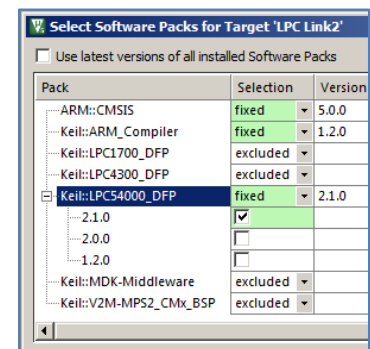
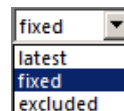
A) Select Software Pack Version: This section is provided for reference:

This μ Vision utility provides the ability to choose various Software Pack versions installed in your computer. You can select the versions you want to use. You must have μ Vision running and any project open for the following exercises:

1. Select Project/Open Project and navigate to C:\00MDK\Boards\Keil\MCB54110\RTX_Blinky.
2. Open Blinky.uvprojx. Blinky will load into μ Vision.
3. Open Select Software Packs by clicking on its icon: 
4. The window below opens: Note **Use latest versions ...** is selected.
5. Unselect this setting and the window changes as shown similar to the one below right:




6. Expand the header Keil::LPC54000_DFP as shown here:
7. You will see only one version – the one you installed. If you had installed others, you would see them listed like this:
8. Select the fixed pull-down menu and see the three options as shown below:
9. If you wanted to use a different version, you would select fixed and then select the check box opposite the version you wanted to use in your project. These settings are stored with your project. This is a good way to freeze software components for your designs. These are saved with the project selected.

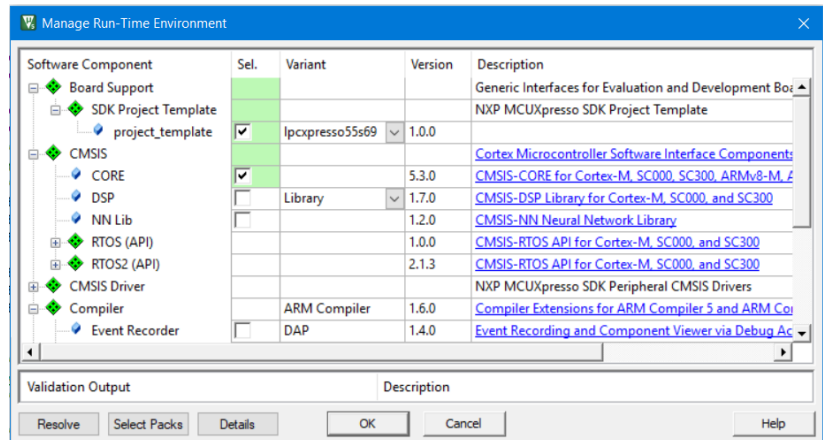


10. Re-select Use latest versions... **Do not make any changes at this time.**
11. Click OK or Cancel to close this window.

B) Manage Run-Time Environment: This section is provided for reference:

The Manage Run-Time Environment utility selects components from various Software Packs and inserts them into your Project list. From here, you can access these files in the usual manner usually by calling functions.

1. Click on the Manage RTE icon:  The window below opens: This includes Keil Middleware, selected open source software, RTX and CMSIS drivers for various peripherals. Not all Packs offer all options but more are being added.
2. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project.
3. Note CMSIS/Core (system.c), Keil RTX and Device/Startup (startup.s) files are selected. You can see the appropriate files listed in the μ Vision Project window.
4. **Do not make any changes.** Click Cancel to close this window.



Select column colour meanings:

TIP: Different colors represent messages:



Green: all required files are located.



Yellow: some files need to be added.

Click the Resolve button to add them automatically or add them manually.



Red: some required files could not be found. Obtain these files or contact Keil technical support for assistance.

The Validation Output area at the bottom of this window gives some information about needed files and current status.

C) Updating Source Files: *this section is provided for reference:*

Some of the files provided by a Software Pack are stored in your project in .\RTE

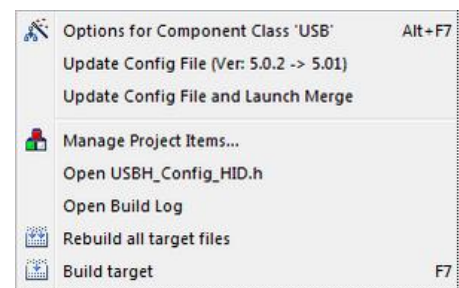
If you update a Software Pack, some of these files might need to be updated from the new Pack. These files will show new icons as shown below and described here: www.keil.com/support/man/docs/uv4/uv4_ca_updswcmpfiles.htm

Updating Source Files: **Note:** Any changes you made to a file will be lost when it is replaced with the new one.

1. Right click on a file you want to update. A window similar to the one below right opens:
2. Select the appropriate Update selection. Remember, any changes you made to the original file are lost.
3. This procedure is described here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

Icons for Software Components (SWC). Refer to [Update Software Component Files](#) for SWC containing modifications.

- SWC is available for the selected microcontroller.
- SWC is not available for the selected microcontroller. The SWC is part of another project target using another microcontroller.
- SWC has been selected previously, but the Software Pack containing this SWC has been uninstalled.
- Contains files with fully backward compatible corrections.
- Contains files with fully backward compatible extensions or new features.
- Contains files with incompatible modifications requiring modifications in the application code.
- File with fully backward compatible corrections. A file update is not required.
- File with fully backward compatible extensions or new features. A file update is recommended.
- File with incompatible modifications. A file update is required.




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

Cortex-M0 and Cortex-M0+ may have only features 2) and 4) plus 11), 12) and 13) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 can have all features listed implemented. MTB is normally found on Cortex-M0+. It is possible some processors have all features except ETM Instruction trace and the trace port. 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.

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.

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

6) External Debug Adapter Types:

A) LPC-Link2 (This has both CMSIS-DAP and J-Link Lite modes):

A small processor located on the target board acts as the debug adapter. On the LPCXpresso55S69 board, this extra processor is U2. Connect a USB cable to connector P6 Debug Link to use LPC-LINK2. CMSIS-DAP provides run control debugging, Flash and RAM programming and Watchpoints. Hardware breakpoints can be set/unset while the program runs. Reads and writes in the Watch, Memory and Peripheral windows are updated in real-time as well as the RTX System kernel awareness viewer. At this time, the LPC-Link2 version of CMSIS-DAP does not support Serial Wire Viewer (SWV). LPC-Link2 also has a J-Link Lite mode. Both can be installed with LPCScript. The LPCXpresso55S69 is preprogrammed with a LPC-LINK2 which is CMSIS-DAP compliant.

You are able to incorporate a CMSIS-DAP design on your own custom target boards. For documents and software go to https://github.com/ARM-software/CMSIS_5 Click on the CMSIS folder and then on DAP/Firmware.

This document was prepared using MDK 5.28.

A) Keil ULINK2/ME:

This is a hardware JTAG/SWD debugger. It connects to various connectors found on boards populated with ARM processors. With NXP Cortex-M3, M4 and M7 processors, ULINK2 supports Serial Wire Viewer (SWV). ULINK-ME is equivalent to a ULINK2.

B) Keil ULINKpro:

ULINKpro is Keil's most advanced debug adapter. ULINKpro provides Serial Wire Viewer (SWV) and ETM Instruction Trace support for NXP Cortex-M3, Cortex-M4 and Cortex-M7 processors. Code Coverage, Performance Analysis and Execution Profiling are then provided using ETM. ULINKpro programs the Flash memories very fast. ARM DS-MDK can use a ULINKpro or a ULINKpro D to debug NXP i.MX Cortex-A processors.

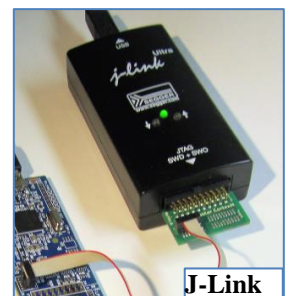
C) Keil ULINKplus:

ULINKplus is Keil's latest debug adapter. Power Measurement is its main feature. It also measures the clock frequency and has very fast SWV performance. It displays exceptions (interrupts). ULINKplus provides very fast Serial Wire Viewer (SWV) performance.

D) Segger J-Link:

µVision supports J-link and J-Link Ultra (black case only) Version 6.0 or later.

1. Select the Target Options icon .
2. Click on the Debug tab. Select J-Link/J-Trace in the Use: box:



External Debug Adapter USB Connections: **Important !**

LPC-Link2:

1. Use P6 Debug Link USB connector.

ULINK2, ULINKpro, ULINKplus or a J-Link:

2. Use USB connector P5 +5V Power Only to supply board power.
3. Connect a Debug Adapter to the 10 pin Cortex Debug P7 SWD connector.
4. Do **not** use Debug Link P6 as a conflict may arise between external and internal debug hardware.

7) External Debug Adapter Configuration:

Once you know which debug adapter you will be using, you can quickly select and test it in μ Vision.




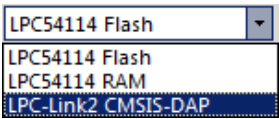

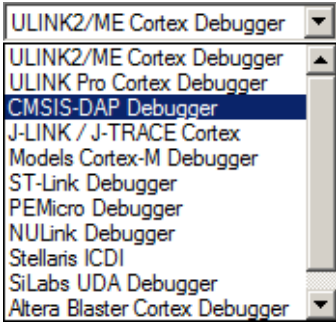
It is easy to configure μ Vision for a variety of supported Debug Adapters.

Prerequisites:

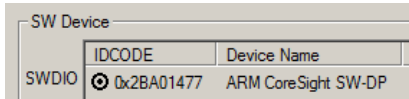
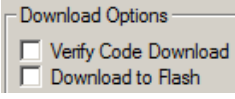

μ Vision must be running and in Edit mode (not Debug mode). Your project must be loaded.

We will use the Blinky example and add LPC-Link2 CMSIS-DAP support. You can choose your own debug adapter by using these instructions.


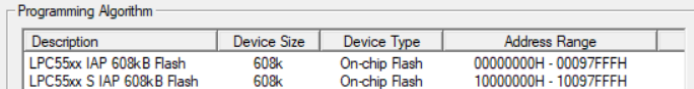

Create a new Target Selection:

1. Select a Target Option LPC54114 Flash to use as the template adapter: 
2. Select Project/Manage/Project Items... or select: 
3. In the Project Targets area, select NEW  or press your keyboard INSERT key.
4. Enter **LPC-Link2 CMSIS-DAP** and press Enter. Click OK to close this window.
5. In the Target Selector menu, select the **LPC-Link2 CMSIS-DAP** selection you just created: 
6. Select Options for Target  or ALT-F7. Click on the Debug tab.
7. Select CMSIS-DAP Debugger... as shown here: 
8. If you are using a different debug adapter, choose it here now.

Test the Debug Adapter:

1. Connect your debug adapter as described on the previous page.
2. Power the board appropriately.
3. In this case, for CMSIS-DAP, connect USB to the USB Link connector.
4. Click on Settings: Confirm SW is selected in the Port: box and not JTAG.
5. A valid SW Device **must** be displayed as shown here. 
6. **Trouble ?** If nothing or an error is displayed, check the LPCXpresso firmware programming and USB connections. See the previous page for instructions. This **must** be solved before you can continue.
7. If using LPC-Link2, unselect Verify Code Download as shown here: 
8. If this selected, an error may occur during the Flash programming process.
9. When completed, click OK twice to close the Options for Target windows.
10. Select File/Save All or click: 

Verify the Flash Program Algorithm: This step is optional. It is useful if Flash programming fails.

1. Select Options for Target  or ALT-F7. Select the Utilities tab.
2. Select Settings:
3. The window that opens will display the correct algorithm. This is selected automatically according to the processor selected in the Device tab.
4. This is the correct algorithms for the LPC55S69. 
5. If it is not visible, select Add to add it.
6. Click OK twice to return to the main μ Vision window.
7. Select File/Save All or click .


The new Debug Adapter is now ready to use.

1) **Blinky example using LPCXpresso55S69:**



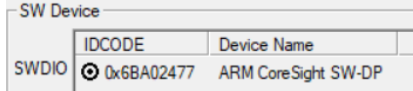
Now we will run the Blinky example on the Keil MDK development system using the LPCXpresso55S69 evaluation board. You can adapt these instructions for use on other LPC55xx boards including custom boards.

This example uses the Arm TrustZone set to Secure mode. It will run in a similar fashion like any other Cortex-M processor. TrustZone and Non-Secure mode are not used. The processor has access to all components directly.

Open the led_blinky example program:

1. Start μ Vision by clicking on its desktop icon. 
2. Select Project/Open Project from the main menu.
3. Open the file C:\00MDK\LPC55S69\demo_apps\led_blinky\cm33_core0\mdk\led_blinky.uvprojx.
4. Connect your PC USB cable to P6 Debug Link connector. You will hear the standard USB dual tone sound.

Confirm Target Processor Connection:

1. Select Target Options  or ALT-F7. Select the Debug tab:
2. Select Settings on the right where your debug adapter is displayed: 
3. Confirm SW is selected in the Port: box and not JTAG.
4. Confirm Verify Code Download is unselected.
5. A valid SW Device IDCODE *must* be displayed as shown here. 

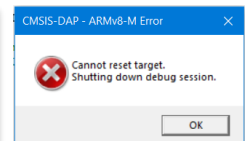
Trouble ? If nothing or an error is displayed, check the LPC-Link2 firmware programming and USB connections. See the previous page for instructions. This *must* be solved before you can continue. Make sure the jumpers are all set to their defaults.





6. When completed, click OK twice to close the Options for Target windows.

7. Select File/Save All or click: 

Compile, Load and Run the Project:

Error RESETing the Processor ? If you get an error box indicating the device cannot be reset when programming the Flash or entering Debug mode: hold the ISP button S1 down and cycle the board power. Release ISP S1 and try programming the Flash again.




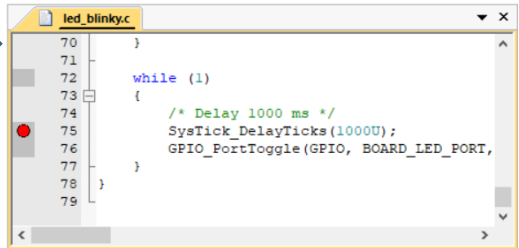

1. Compile the source files by clicking on the Rebuild icon.  Progress is indicated in the Build Output window.
2. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears. Flash programming progress is indicated in bottom left corner.
3. Click on the RUN icon  to start the Blinky program. If the LED RGB D8 blinks, **Blinky is now running !**
4. Stop the program with the STOP icon. 

The blue LED will blink !

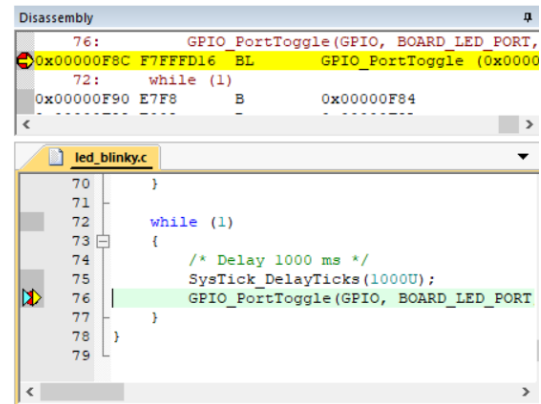
Now you know how to compile a program, load it into the LPC55S69 Flash, run it and stop it.

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


2) Hardware Breakpoints:

1. Click on the RUN icon .
2. Bring led_blinky.c in focus by clicking on its tab. If it is not visible, double-click on it in the Project window.
3. In led_blinky.c, scroll down to inside the while (1) loop found starting near line 72 as shown here: 
4. Note on the left of the line numbers are darker grey blocks. This indicates there is assembly code present and you can set a hardware breakpoint on these lines. You can also see these blocks in the Disassembly window.
5. While the program is still running, click to the left of a suitable line in the while loop and a red circle will be created. This is a hardware breakpoint. LPC55S69 has 8 hardware breakpoints. µVision will warn you if you exceed this limit.
6. The program will soon stop at the line where you set the breakpoint on as shown below. The yellow arrow is the current program counter position. This will be the next instruction executed. The cyan arrow is a placeholder you can use to explore the relationship between a source window and the Disassembly window.
7. Add another breakpoint in while (1) or in a suitable place in one of the other threads.
8. Each time you click on RUN , the program will cycle to the next breakpoint.


TIP: You can set/unset hardware breakpoints with µVision while the program is running or stopped. This is an important feature.

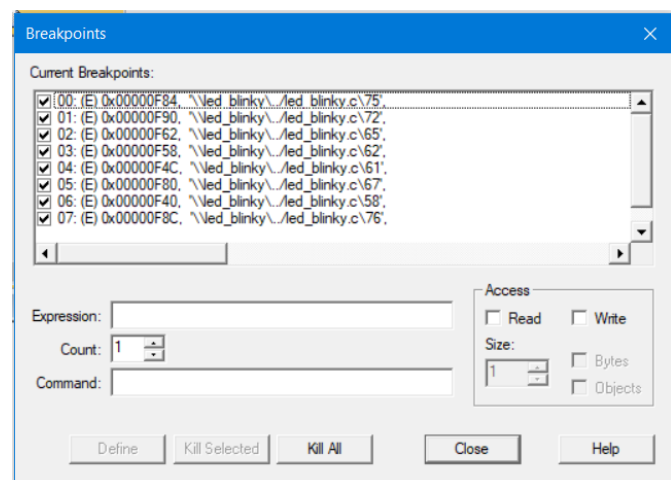


Managing Breakpoints:

1. Select Debug/Breakpoints or Ctrl-B. This window opens:
2. You can temporarily unselect, delete or create breakpoints in this window. It is easier to create a breakpoint by clicking in a source file or the Disassembly window.
3. Watchpoints are also created in this window. This is discussed in a few pages.
4. **Select Kill All to remove all breakpoints.**
5. Click Close.
6. Click RUN  for the next exercise.

TIP: If you get multiple cyan arrows or can't understand the relationship between the C source and assembly, try lowering the compiler optimization to Level 0 and rebuilding your project. Compilers can do strange things while optimizing code.

The optimization level is set in Options for Target  under the C/C++ tab when not in Debug mode.




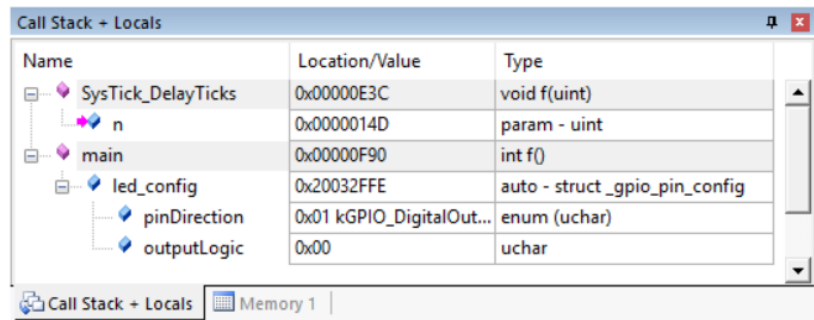
3) Call Stack + Locals Window:


Local Variables:

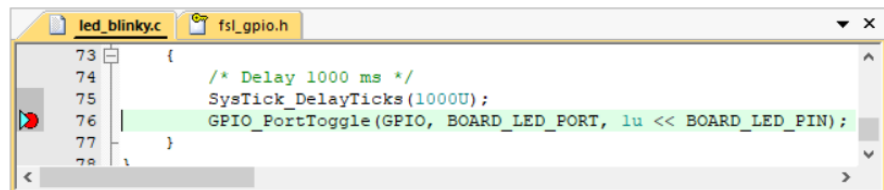
The Call Stack and Local windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function.



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

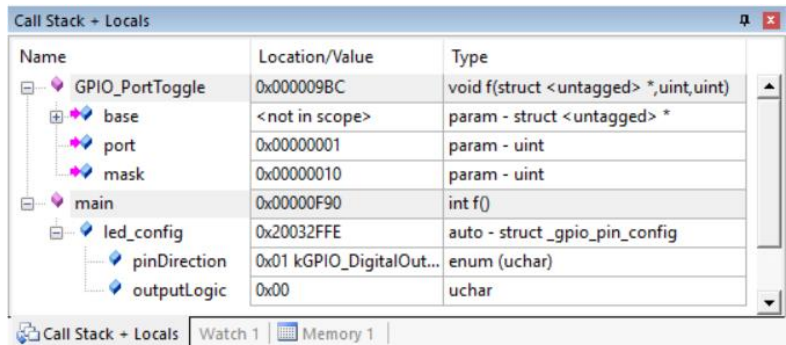
1. Stop the program with the STOP icon.  The program will probably stop in the SysTick_DelayTicks function.
2. Click on the Call Stack + Locals tab in the bottom right corner of µVision.
3. Inspect the various entries in the Call Stack + Locals window as shown below right in this very simple example. Local variables are displayed only when they are in scope.




4. Set a breakpoint in the while(1) loop near line 76 on the call to the GPIO_PortToggle function as shown here:
5. Click RUN .
6. The program will stop here:




7. Click Step  or F11: (with led_blinky.c in focus (its name in its tab is underlined)).
8. You will enter this function as shown here:
9. Note the variables displayed.
10. Click Step  or F11 a few more times:
11. You will return to the SysTick_DelayTicks function.
12. **REMOVE** the Breakpoint to continue.



TIP: You can also use Step Out icon  to exit a function. This is will be indicated in the Call Stack window.

TIP: This is standard “Stop and Go” debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables and structures updated in real-time while the program is running. No additions or changes to your code are required. They must be converted to global or static variables so they always remain in scope.

TIP: Single Step:  (F11): If you click inside a source window to bring it into focus, the program will step one source line at a time. If you click inside the Disassembly window to bring it into focus, it will then step one instruction at a time.

Call Stack Uses:

The list of stacked functions is displayed when the program is stopped. This is when you need to know which functions have been called and are stored on the stack. This is useful to find program flow problems such as crashes.


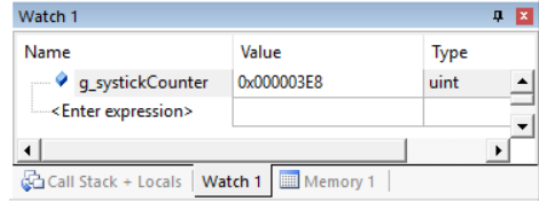
Normal functions are displayed only when they are on the Stack. They are removed and added as appropriate. When RTX is used, the threads are always displayed, even if not running. The active thread name and its number are highlighted in green.

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

The Watch and Memory windows will display updated variable values in real-time. It does this using ARM CoreSight DAP debugging technology. It is also possible to “put” or insert values into these memory locations in real-time using the Memory window while the program is running.

There is a global variable `g_systickCounter` located in `Blinky.c` near line 25 we can use in the Watch and Memory windows.

Watch window:

1. Click on the RUN icon  to start the Blinky program.
2. Right click on `g_systickCounter` in `led_blinky.c` near line 26 and select Add ‘`g_systickCounter`’ to... and select Watch 1.
3. `g_systickCounter` will be displayed in Watch 1 as shown here: 
4. Select View in the main menu and enable Periodic Window

Update:  Values of `g_systickCounter` are updated in real-time.

TIP: You do not need to stop the program to enter variables, raw addresses or structures in a Watch or Memory window.

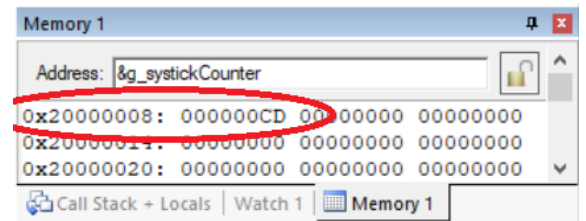
5. You can modify the value in a Watch window when the program is stopped or changing slowly. You can modify a variable in a Memory window anytime. Step 5 below illustrates this technique.

SystemCoreClock:



CMSIS provides this global variable which stores the CPU clock. Double-click on `<Enter Expression>` and enter `SystemCoreClock`. Right-click and disable hex display and 12 MHz is now displayed.

Memory window:

1. Right click on `g_systickCounter` and select Add ‘`g_systickCounter`’ to... and select Memory 1.
2. Note the changing value of `g_systickCounter` 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. Right click in Memory 1 and select Unsigned Long to see the data field as 32-bit numbers.
4. Add an ampersand “&” in front of the variable name `g_systickCounter` and press Enter. Now the physical address is shown (0x2000_0008) in this case. This physical address could change with different compilation optimizations.
5. The data contents of `g_systickCounter` is displayed as shown:
6. Right click on the memory data value and select Modify Memory. Enter a value and this will be pushed into `g_systickCounter`. Since `g_systickCounter` is updated often, you will not see a new value displayed. But the update did occur.



TIP: You are able to configure the Watch and Memory windows while the program is running in real-time without stealing any CPU cycles. You can change a Memory window value on-the-fly.

7. The global variable `g_systickCounter` is updated in real-time. This is ARM CoreSight technology working.
8. Stop the CPU  and exit Debug mode  for the next step.

TIP: View/Periodic Window Update must be selected. Otherwise variables update only when the program is stopped.

How It Works:

µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. The Cortex-M3, M4 and M7 are Harvard architectures. This means they have separate instruction and data buses. While the CPU is fetching instructions at full speed, there is plenty of time for the CoreSight debug module to read or write to memory without stealing any CPU cycles.


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

Remember you are not able to view local variables while the program is running. They are visible only when the program is stopped in their respective functions. You must change them to a different type of variable such as global to see them update.


5) System Viewer (SV):


System Viewer provides the ability to view certain registers in peripherals and the CPU core. In many cases, these windows are updated in real-time while your program is running. They are available only while in Debug mode. Select Peripherals/System Viewer to open the peripheral windows. Select CPU core registers by opening Peripherals/Core Peripherals. Both of these are illustrated here:

In our led_blinky example, the LED RGB is output on PIO1.

1. Click on RUN . You can open SV windows running or not.

PIO Port 1:


2. Select Peripherals/System Viewer and then GPIO then GPIO as shown: 

3. The GPIO window opens up: 

4. Expand B[1] as shown in the GPIO window:

5. This value changes (slowly) as the program runs.

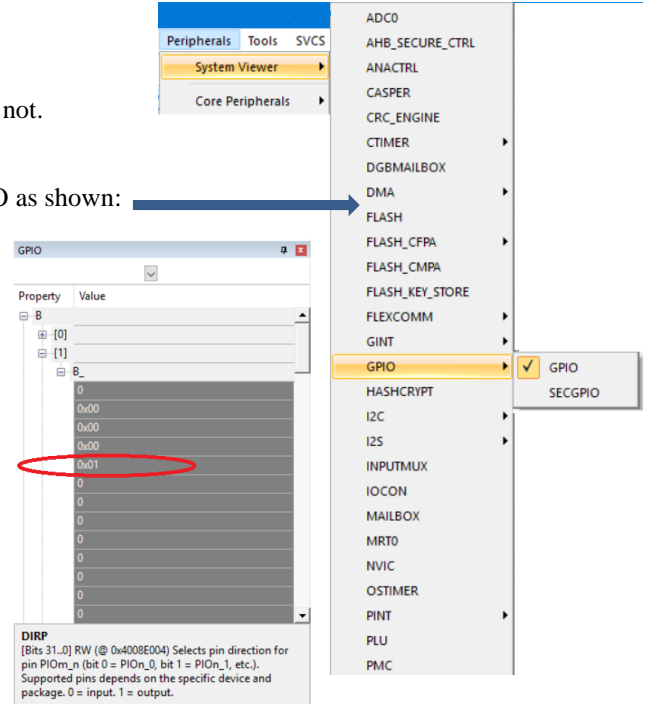
6. This window is updated periodically using the same CoreSight DAP technology as the Watch and Memory windows. For fast changing data, some values might be skipped.

7. Stop the CPU 

8. When you change this to a “1”, the LED goes out. Change it to a “0” and it comes on.

9. You can change the values in the System Viewer while the program is running or stopped. It will be difficult to see this as these values in this case are updated so often that your changes will be overwritten.

10. Some peripheral registers might not allow you to change them.



TIP: If you click on a register in the Property column, a description about this register will appear at the bottom of the window as shown here for GPIO. This is an easy method to find physical registers for various peripherals.

SysTick Timer: The led_blinky example program uses the Cortex SysTick timer S for a delay function.


1. Select Peripherals/Core Peripherals and then select SysTick Timer S. Run the program.
2. The SysTick Timer S window shown below opens: This is for the Secure mode.
3. Note it also updates in real-time while your program runs using CoreSight DAP technology.
4. Note the SysTick ->Load register. This is the reload register value. This value is set in led_blinky at line 65:
if (SysTick_Config(SystemCoreClock / 1000U))

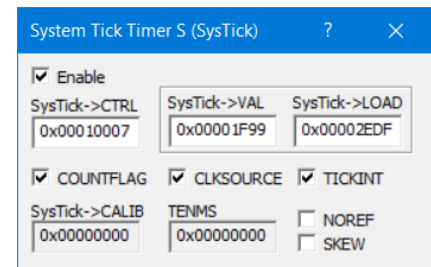
5. Note that it is set to 0x2EDF = dec 11,999. This is created by 12,000 MHz/1000-1 = 11,999. 1000 is specified as the timer tick value. A SysTick S interrupt 15 will occur every 1 msec. Changing the reload value changes how often the SysTick interrupt 15 occurs.

6. In the RELOAD register in the SysTick window, *while the program is running*, type in 0x1000 and press Enter.

7. The blinking LED will speed up. This will convince you of the power of ARM CoreSight debugging. Remember that a Cortex-M33 has two SysTick timers for S and NS modes.

8. Replace RELOAD with 0x0000_2EDF. A CPU RESET  will also accomplish this easier.

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.

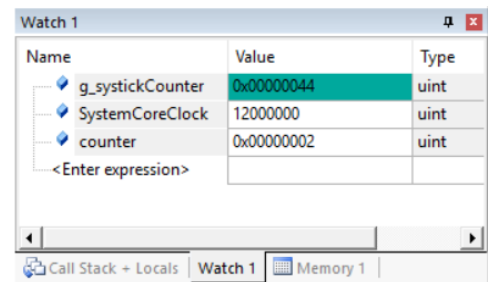
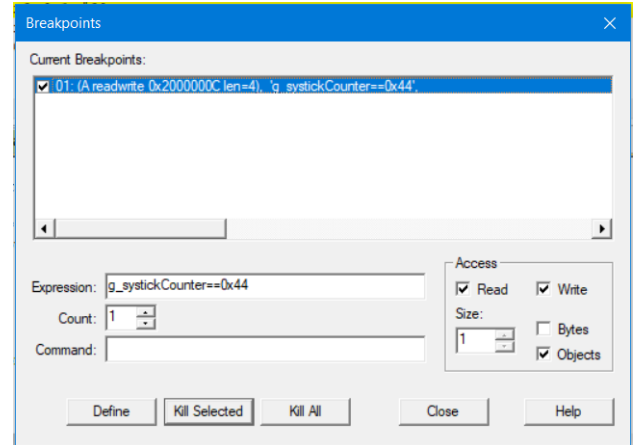
You must make sure a given peripheral register allows for and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.

6) Watchpoints: Conditional Breakpoints (Access Breakpoints)

Most NXP Cortex-M3, M4, M7 and M33 processors have four data comparators. Since each Watchpoint uses two comparators, you can configure two complete Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same comparators in its operations. This means in μ Vision you must have two variables free in the Logic Analyzer to use Watchpoints. Currently, μ Vision supports only one Watchpoint.

Configure Watchpoint:

1. Using the example from the previous page, Stop the program  Stay in Debug mode.
2. In the main menu, select Debug/Breakpoints... or select Ctrl-B on your keyboard. The Breakpoints window opens.
3. In the Expression box, enter `g_systickCounter==0x44` as shown below.
4. Select both the Read and Write Access.
5. Click on Define and it will be accepted as shown here:
(the Expression: box will go blank) 
6. Click on Close.
7. Click on RUN .
8. When `g_systickCounter = 0x44` the program will halt.
This is how a Watchpoint works.
9. You will see `g_systickCounter` displayed with a value of `0x44` in the Watch window as shown below right:
10. **Delete the Watchpoint** by selecting Debug and select Breakpoints and select Kill All.
11. Select Close.
12. Leave Debug mode. .



Using the Watchpoint in the Stack Space:

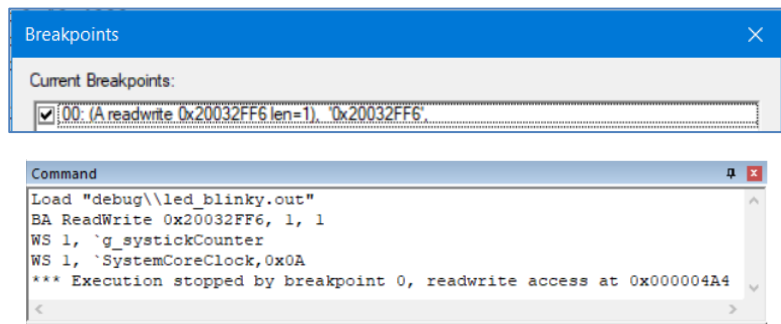
If you put a RAM address as the expression with no value, the next read and/or write (as you selected) will cause the program to halt.

This can be particularly useful in the Stack. Set an address at some limit and if the program reads or writes this address, the program stops.

1. In this example a Watchpoint is created with address `0x2003_2FF6`.
2. When the program is run the first R or W will stop the processor.
3. The Command window shows the setting of this Watchpoint and its execution including the approximate instruction location. In this case it is `0x04A4`.

TIP: SKID: The instruction noted will not be the instruction causing the break as it has not been executed yet. It will be noted in Disassembly.

The instruction causing the break is probably a few instructions before this one in the execution stream.



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 use the next TIP:

TIP: The checkbox beside the expression in Current Breakpoints as shown above allows you to temporarily unselect or disable a Watchpoint without deleting it.



TIP: You can create a Watchpoint with a raw address and no variable value. This is useful for detecting stack overruns. Physical addresses can be entered as `*((unsigned long *)0x20000000)`. Or simply enter the address as shown above.

7) printf without using a UART:


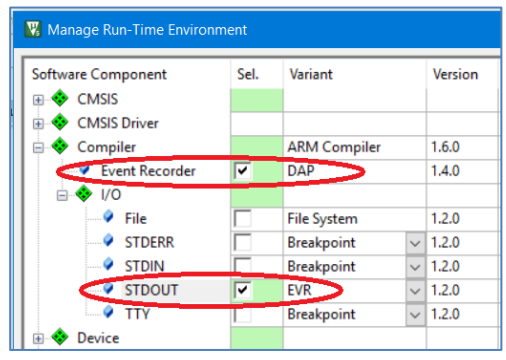
µVision provides a simple printf utility using Keil Event Recorder. No UART is needed and it is much faster and takes less code than a standard UART or USB COM port. Text is displayed in the Debug (printf) Viewer shown below.

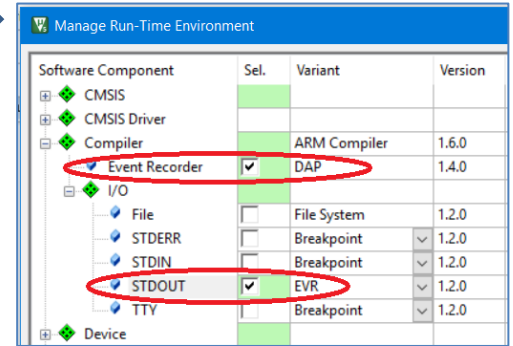
Event Recorder (EVR) is a new µVision feature that can be used to instrument your code. Keil RTX5, CMSIS-FreeRTOS and Middleware is already instrumented with Event Recorder. This can be used with any Cortex-M processors. DAP is the same technology used in Watch, Memory and Peripheral windows. LPC-Link2 supports Event Recorder.

Configure Event Recorder:


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

Configure Event Recorder:








1. Open the Manage Run-Time Environment utility.  This opens: 
2. Expand Compiler and I/O as shown.
3. Select Event Recorder (DAP) and STDOUT (EVR) as shown:
4. All the blocks should be green. If not, click on the Resolve button.
5. Click OK to close this window.
6. retarget_io.c and EventRecorder.c will be added to your project under the Compiler group in the Project window.
7. Right click near the top of led_blinky.c, at line 10 and select Insert "#include" and select #include "EventRecorder.h".
8. In the main() function near line 72 just before the second while(1) loop, add this line:
EventRecorderInitialize (EventRecordAll, 1);
9. Right after that line, add this line near line 14: EventRecorderStart ();

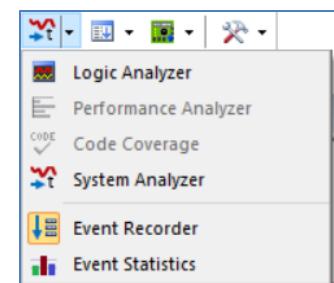


Add a printf statement to Blinky.c:

1. In Blinky.c add #include "stdio.h" near the top of the file near line 4.
2. In Blinky.c, near/at line 78 just after the SysTick_DelayTicks(1000U);, add this line: printf("Hello World !\n");
3. Select File/Save All or click .

Build and RUN the Blinky program and view printf:

1. Rebuild the source files . Ignore the two warnings.
2. Enter Debug mode . Click on RUN .
3. Select View/Serial Windows and select Debug (printf) Viewer.
4. The values of counter are displayed as seen here: 
5. Open the Event Recorder window: 
6. Information about the printf statements are displayed.
7. Stop the program.  Exit Debug mode .



TIP: If you get a Hard Fault error, you must either select MicroLIB *or* add some heap in the correct startup.s file or the scatter file if used for your processor.

Try adding a 200 byte heap. MicroLIB results in a smaller executable size and is a very useful compiler setting.

8) Armv8-M TrustZone Demonstration:

The NXP LPC55xx series of Cortex-M33 processors are full implementations of Armv8-M TrustZone technology.

There are two modes or states of operation: Secure and Non-Secure. These modes are memory mapped. A program running in Secure designated memory is running in Secure mode. It has access to both Secure and Non-Secure components. A program running in Non-Secure memory is in Non-Secure mode. It has access to only non-secure configured memory (including peripherals). A Non-Secure program is able to call a function in Secure mode by a prescribed method. If this is not followed exactly which might indicate a hacking attempt, a Secure Fault is generated.


µVision is capable of debugging both modes. This section will demonstrate how a Non-Secure program can call a function in the Secure mode and return back to Non-Secure mode.

1) Hello World example program using LPCXpresso55S69:

Now we will run the Hello World example on the Keil MDK development system using the LPCXpresso55S69 evaluation board. You can adapt these instructions for use on other LPC55xx boards including custom boards.

This example use the Arm TrustZone set to Secure mode. It will run in a similar fashion like any other Cortex-M processor.



Open the led_blinky example program:

1. Start µVision by clicking on its desktop icon. 
2. Select Project/Open Project from the main menu.
3. Open the file **hello_world.uvmpw** which is found in this folder tree:
C:\00MDK\LPC55S69\trustzone_examples\hello_world\cm33_core0\hello_world_s\mdk\.




TIP: This is a Keil Multi-project Workspace. It consists of two projects: a Secure and Non-Secure projects.

4. Connect your PC USB cable to P6 Debug Link connector. You will hear the standard USB dual tone sound.

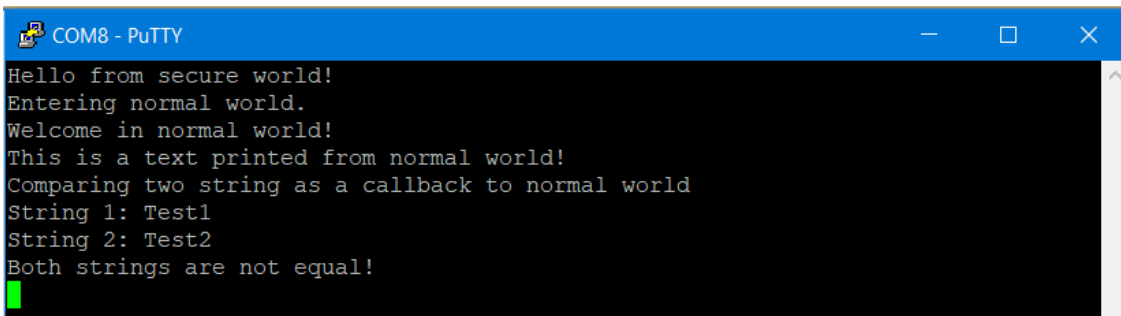
Compile, Load and Run the Project:

1. Compile the source files by clicking on the Batch Build icon: 
2. The Command window will display: **Batch-Build summary: 4 succeeded, 0 failed, 0 skipped**
3. Enter Debug mode.  Flash programming progress is indicated in bottom left corner.
4. The LPC-Link2 has a virtual COM port. Connect a terminal program such as PuTTY in the usual manner.

Error RESETing the Processor ? If you get an error box indicating the device cannot be reset: hold the ISP button S1 down and cycle the board power. Release ISP S1 and try programming the flash again.

5. Click RUN  to start the hello_world program.
6. The terminal program will display the results in the terminal window as shown below.
7. If you see this: you have just run your first Armv8-M TrustZone program on the NXP LPX55S69
8. Stop the program.  Exit Debug mode. 

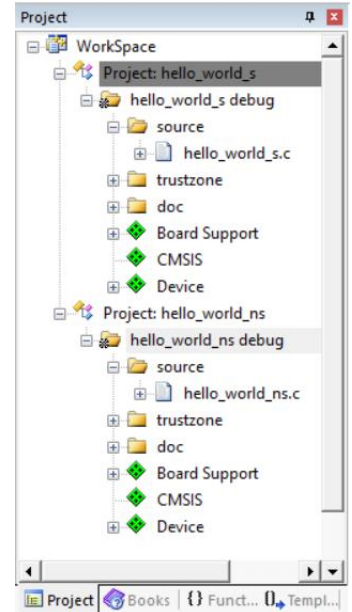
TIP: The board will run Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.



2) Investigating Secure and Non-Secure Modes:

This is some general information about this program:

1. The project file is a µVision Multi Project workspace.
2. There are two main programs: hello.world_s and hello_world_ns as shown here:
3. Each has one C source file: hello_world_s.c and hello_world_ns.c\
4. Each of these has its own main() function as required by the C language.
5. Some features such as SysTick, NVIC are duplicated to serve either S or NS mode.
6. The active project is indicated by the black highlight: Project: hello_world_s.
7. You can change this by right clicking over a Project and selected Set as Active project.
8. Sometimes there are a different file (but of the same name) in each project. You must be careful when modifying or referencing these files to make sure you are accessing the file you think you are. One example: there is a file startup_LPC55S69_cm33_core0.s in the Secure project and another of the same name in the Non-Secure project. The same situation for veneer_table.h. These are all different files. See them below: Similarly, make sure you are working with the correct project set as Active.
9. µVision can debug both Secure and Non-Secure modes. It is possible to disable Secure debugging and allow only Non-Secure debugging. Or no debugging access.
10. Arm Compiler 6 (AC6) which is LLVM based is used for all Armv8-M processors.



When this program is run these events happen:

1. At RESET the CPU is in Secure mode. Various configuration items are run.
2. Memory (including peripherals, code and RAM) are mapped to Secure or Non-Secure.
3. The CPU then is switched into Non-Secure mode.
4. The Secure program has access to all processor features.
5. The CPU will run in Non-Secure mode unless the program requires a switch to Secure.
6. The Non-Secure program can only see memory and peripherals set as Non-Secure.
7. It is possible to make functions calls into the Secure mode is a prescribed fashion.
8. **CMSE = Cortex Microcontroller Security Extensions.** An acronym definition.
9. In general, most user programs will run in the less -trusted Non-Secure mode. Code and data you want to protect will run in the Secure mode. You can put things like proprietary WiFi stacks for instance in the Secure mode.

We will demonstrate how a Non-Secure program calls a function into the Secure mode and return to the Non-Secure mode.

Compiler Attributes:

These are two important compiler attributes used to create TrustZone projects:

1. `__attribute__((cmse_nonsecure_entry))`
2. `__attribute__((cmse_nonsecure_call))`

3) Examining and Modifying the TrustZone Project Files:


hello_world_s.c:

µVision will be open at this point with the Multi Project Workspace project hello_world.uvmpw loaded.

hello_world_s is set to be the Active project: it is highlighted dark gray in the Projects window as shown on the previous page.

Modify Source Code:

This program runs only one time. It ends up in a while(1) loop at the end of hello_world_ns.c.

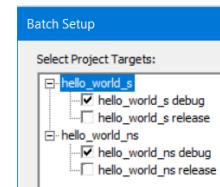
1. Move the while(1) line and its first brace to line 40 and 41. The program will run continuously.
2. Select File/Save All or click .

```
38 PRINTF_NSE("Welcome in normal world!\r\n");
39 PRINTF_NSE("This is a text printed from normal world!\r\n");
40 while (1)
41 {
42     result = StringCompare_NSE(&strcmp, "Test1\r\n", "Test2\r\n");
43     if (result == 0)
44     {
45         PRINTF_NSE("Both strings are equal!\r\n");
46     }
47 }
```


Compile Multi Project Workspace:



1. Select the small arrow beside Batch Build and then select Batch Setup... to open the configuration window.
2. Unselect the release projects as we do not need to compile them now.
3. Select the Rebuild box. The two projects will be successfully compiled.




Examining the Program: hello_world_s.c:

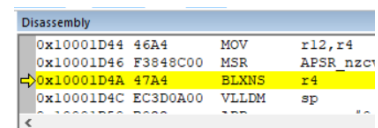
1. Enter Debug mode.  Flash will be programmed.
2. The program counter will be at the start of main() in hello_world_s.c.

Error RESETing the Processor ? If you get an error box indicating the device cannot be reset: hold the ISP button S1 down and cycle the board power. Release ISP S1 and try programming the flash again.

3. The Secure mode copy of startup_LPC55S69_cm33_core0.s has already run.
4. The program will be in Secure mode as indicated by bold of Secure and this box at µVision bottom right:
5. **Debug: Secure** means the debugger can see both Secure and Non-Secure code and memory. **CPU: Secure** states which mode the CPU is currently in. CMSIS-DAP ARMv8-M Debugger Debug: Secure CPU: Secure
6. Inspect the code as you scroll down. You can see various configuration items pertaining to Secure mode.
7. At line 67 the Non-Secure mode starts to be configured.
8. The program branches to hello_world_ns.c and switches to Non-Secure mode by Reset_Handler() at line 79. Specifically it is the BXNS instruction at address 0x1000_1D4A that does this branch and switch.

4) Switch from Secure to Non-Secure Mode:

1. Set a breakpoint at 79 Reset_Handler() near line 79.
2. Click RUN  to run to breakpoint at this function call.
3. Click in the Disassembly window to bring it into focus. **This is important.** If it is not in focus steps will be by C line and not assembly instruction which is what we want to observe. The yellow arrow in the PC and this has not been executed.
4. Click Step or F11 about 16 times until you come to the BXNS instruction:
5. Note the CPU is in Secure mode. CPU: Secure
6. Execute this instruction with Step or F11 and the CPU switches to Non-Secure mode. CPU: Non-Secure
7. The PC will now be in the Non-Secure mode copy of startup_LPC55S69_cm33_core0.s at Reset_Handler PROC.
8. From Arm website: The BXNS instruction causes a branch to an address and instruction set specified by a register and causes a transition from the Secure to the Non-secure domain.




Disassembly			
0x10001D44	46A4	MOV	r12, r4
0x10001D46	F3848C00	MSR	APSR_nzcv
0x10001D4A	47A4	BXNS	r4
0x10001D4C	EC3D0A00	VLLDM	sp

TIP: If you make a mistake, exit and re-enter Debug mode to start over. You might have to cycle the board power with ISP button held down. Biggest problem is usually stepping by C source line because Disassembly window is not in focus.

5) A Non-Secure program calling a function in Secure Space:

At this point the program is in Non-Secure mode and sitting on an instruction (CPSID) in the Non-Secure startup file.



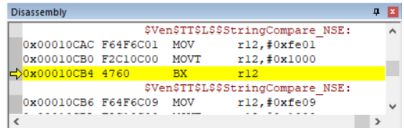

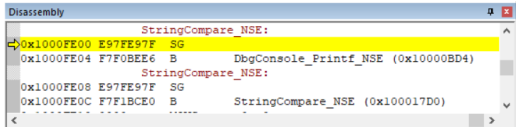

Run to main() in hello_world_NS.c:

1. Set a breakpoint at the start of main() in hello_world_ns.c. This will be at line 33 (or close to it).
2. Click RUN  to run to breakpoint at the Non-Secure main().
3. Note there are four PRINTF_NSE statements. Each of these is a call to a function in Secure mode. We will exercise these to demonstrate a non-Secure program making a call to the Secure space.
4. Set breakpoints at the two PRINTF_NSE statements at lines 45 and 49.

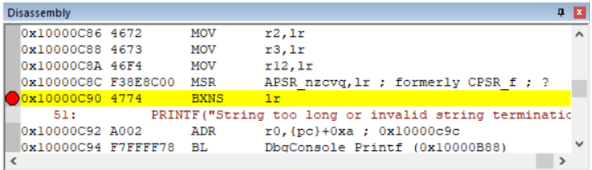


TIP: The program in the Non-Secure code is unable to access or read code, memory or peripherals in the Secure mode. In this example the μ Vision debugger is able to see both modes and all memory and peripherals. It is possible to restrict the debugger to Non-Secure mode and see only Non-Secure code and memory and peripherals.

We will now show how a Non-Secure program can call a Secure function that it cannot directly see.

Step into A Secure Function:

1. Click RUN  to run to breakpoint at either of the PRINTF_NSE function calls.
2. Click in the Disassembly window to bring it into focus. **This is important.** If it is not in focus steps will be by C line and not assembly instruction which is what we want to observe. The yellow arrow in the PC and this has not been executed.
3. Click Step  or F11 about 6 times until you come to this BX instruction: 
4. Note the CPU is in Non-Secure mode. **CPU: Non-Secure**
5. Click Step  or F11 **once** to come to the SG instruction:
6. SG = Secure Gateway. The Non-Secure makes a call to a function in the Secure area. The processor goes to a veneer code where the first instruction must be a SG. If it is not, a Secure Fault (7) will occur.

The Non-Secure code is not able to see this veneer that contains SG instructions – it only knows to go here.
7. Note the CPU is still in Non-Secure mode. **CPU: Non-Secure**
8. Click on Step  or F11 one time to execute SG.
9. Note the CPU is now in Secure mode. **CPU: Secure**
10. The CPU can now execute code that it was not able to access while in Non-Secure mode. The code in the Secure mode is trusted and cannot be modified.
11. At some point, the Secure code will return back to the Non-Secure mode using a BLXNS instruction.

Returning to Non-Secure Mode:

12. In the Disassembly window, right-click and select Show Disassembly at Address ... and enter 0x1000_0C90
13. You will find a BXNS instruction here that will take you back to the Non-Secure mode.

14. Click RUN  to run to breakpoint at the BXNS.
15. Note the CPU is still in Secure mode. **CPU: Secure**
16. Click on Step  or F11 one time to execute BXNS.
17. Note the CPU is now in Non-Secure mode. **CPU: Non-Secure** and you are back into hello_world_ns.c.

This is the sequence to allow a non-secure program call a function in the Secure mode.

This is the end of the exercises (for now)

Check the website for upcoming additions including ULINKplus power measurement.

www.keil.com/appnotes/docs/apnt_322.asp

Blank page for enhancements

1) Document Resources:

See www.keil.com/NXP

Books:

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

Application Notes:

1. **NEW!** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with μ Vision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with μ Vision <https://launchpad.net/gcc-arm-embedded>
8. RTX CMSIS-RTOS Download www.keil.com/demo/eval/rtx.htm
9. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
10. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
11. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
12. Sending ITM printf to external Windows applications: http://www.keil.com/appnotes/docs/apnt_240.asp
13. FlexMemory configuration using MDK www.keil.com/appnotes/files/apnt220.pdf
14. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
15. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
16. **NEW!** ARMv8-M Architecture Technical Overview <https://community.arm.com/docs/DOC-10896>
17. **NEW!** Using TrustZone on ARMv8-M with Keil MDK: www.keil.com/appnotes/docs/apnt_291.asp
18. **NEW!** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp

Useful ARM Websites:

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

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

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

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

2) Keil Products and Contact Information:

See www.keil.com/NXP

Keil Microcontroller Development Kit (MDK-ARM™)

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

- **ULINK2** -(ULINK2 and ME - SWV only – no ETM) **ULINK-ME** – Electrically equal to ULINK2.
- **New ULINKplus-** Cortex-Mx High performance SWV & power measurement.
- **ULINKpro** - Cortex-Mx SWV & ETM trace.
- **ULINKpro D** – The same as a ULINKpro without ETM support. It can be used with µVision or DS-5.

For Serial Wire Viewer (SWV), a ULINK2 or a J-Link is needed. For ETM support, a ULINKpro is needed.

All ULINK products support MTB (Micro Trace Buffer) with NXP Cortex-M0+.

The Keil RTX RTOS is provided with a BSD or Apache 2.0 license. This makes it free. All source code is provided.

See https://github.com/ARM-software/CMSIS_5 and www.keil.com/gsg.
RTX documentation is free. www.keil.com/pack/doc/CMSIS/RTOS/html/
For the feature list see: www.keil.com/RTX

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

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

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

Call Keil Sales for special university pricing. Go to www.arm.com/university

Keil supports many other NXP processors including ARM7™ and ARM9™ series processors. See the Keil Device Database® on www.keil.com/dd for the complete list of NXP support. Also see www.keil.com/NXP

NXP i.MX processors are supported by ARM DS-MDK™. www.keil.com/ds-mdk or DS-5: www.arm.com/ds5



For more information:

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

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

Global Inside Sales Contact Point: Inside-Sales@arm.com

Keil Distributors: www.keil.com/distis/

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

