NXP S32K 148: Cortex-M4 Tutorial
With the S32K148 EVB evaluation board

Arm Keil MDK 5 Toolkit Summer 2018 V 1.2 bob.boys@arm.com

The latest version of this document is here:	www.keil.com/appnotes/docs/apnt_305.asp
NXP Cookbook Example Ports for MDK:	www.keil.com/appnotes/docs/apnt_304.asp
A lab for the S32K-144 EVB:	www.keil.com/appnotes/docs/apnt_299.asp

Introduction:

The purpose of this lab is to introduce you to the NXP S32K-148 Cortex®-M4 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 processors and Keil MDK. See www.keil.com/NXP and www.keil.com/s32K. ***New*** **Getting Started MDK 5:** www.keil.com/gsg.

Keil MDK supports and has examples for most NXP Arm processors. Check Keil Device Database® on www.keil.com/dd2.

NXP i.MX series processors are supported by Arm DS-MDK™. www.keil.com/ds-mdk.

Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into one of the commercial versions.

RTX RTOS: All variants of MDK contain the full version of RTX with Source Code. RTX has a BSD or Apache 2.0 license with source code. www.keil.com/RTX and https://github.com/ARM-software/CMSIS_5 FreeRTOS is also supported.

Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the Arm® Compiler toolchain. MDK is turn-key "out-of-the-box". Examples, board and NXP SDK support is included.
2. Arm Compiler 5 and Arm Compiler 6 (LLVM) are included. GCC: https://developer.arm.com/tools-and-software
3. Dynamic Syntax checking on C/C++ source lines.
4. MISRA C/C++ support using PC-Lint. www.gimpel.com
5. **Keil Middleware:** Network, USB, Flash File and Graphics.
6. ***NEW!*** **Event Recorder for Keil Middleware, RTOS and User programs. Page 17.**
7. CoreSight™ Serial Wire Viewer (SWV).
8. ETM instruction trace: Code Coverage and PA.
9. **Compiler Safety Certification Kit: www.keil.com/safety/** ***NEW !*** **FuSa RTS www.keil.com/fusa-rts**
10. **TÜV** certified. ISO 26262 ASIL D, IEC62304 Class C, IEC 61508 SIL 3, EN50128 SIL 4
11. Adapters: OpenSDA (CMSIS-DAP or P&E mode), ULINK™2, ULINK-ME, ULINK*plus*, ULINK*pro* and J-Link.
12. Affordable perpetual and term licensing with support. Contact Keil sales for pricing options. Inside-Sales@arm.com
13. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
14. ULINK*plus* power analysis: www.keil.com/mdk5/ulink/ulinkplus/ Contact Keil sales.

This document includes details on these features plus more:

1. Serial Wire Viewer (SWV) data trace. Includes Exceptions (interrupts), Data writes, graphical Logic Analyzer.
2. 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.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
4. RTX and RTX Threads windows: kernel awareness for RTX that updates while your program is running.
5. ***NEW!*** µVision Event Recorder (EVR) You can use this in your own programs too.
6. printf using SWV ITM or Event Recorder (EVR). No UART is required.
7. A DSP example program using Arm CMSIS-DSP libraries.
8. **ETM Instruction Tracing including Code Coverage and Performance Analysis.**

General Information:

Keil Software and Software Packs:

Using Debug Adapters:

Blinky Example and Debugging Features:

Serial Wire Viewer (SWV):

DSP Example:

ETM Instruction Trace:

Other Useful Information:

1) NXP Evaluation Boards & Keil Evaluation Software:

Keil MDK provides board support for many NXP Cortex-M processors. For the i.MX series see www.keil.com/ds5-mdk

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 NXP boards.

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

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

2) MDK 5 Keil Software Information: *This document uses MDK 5.24a or later.*

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

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

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 is described by a .pdsc file in XML format.

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

Example Project Files: This document uses the RTX5_Blinky example project contained in the S32K Software Pack.

3) Debug Adapters Supported:

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

1. **OpenSDA:** OpenSDA is the on-board debug adapter. NXP OpenSDA has a P&E and a CMSIS-DAP mode depending on the firmware loaded into the OpenSDA processor U510. You do not need an external debugger such as a ULINK2 to do this lab. If you want to use Serial Wire Viewer (SWV), you need any ULINK or a J-Link. You can add CMSIS-DAP to a custom board. See https://github.com/ARM-software/CMSIS_5.

2. **CMSIS-DAP:** An extra processor on your board becomes a debug adapter compliant to CMSIS-DAP. The S32K and many other NXP boards OpenSDA can be configured for CMSIS-DAP mode.

3. **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.

ULINK2

4. **ULINK*pro*:** ULINK*pro* supports all SWV features and adds ETM Instruction Trace. ETM records all executed instructions. ETM provides Code Coverage, Execution Profiling and Performance Analysis features. ULINK*pro* also provides the fastest Flash programming times. Only S32K-148 has ETM. Consult your datasheet.

5. ***NEW !*** **ULINK*plus*:** High SWV performance plus Power Measurement. See www.keil.com/ulink/ulinkplus/ for details.

ULINK*pro*

6. **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.

Debug Connections: An external debug adapter must be connected to the J10 Cortex Debug ETM 20 pin connector. This is a 20 pin CoreSight standard connector. For pinouts search the web for "Keil connectors". A special 10 to 20 cable is provided with ULINK2 and ULINKplus. ULINK*pro* connector has a 20 pin cable by default.


J10 CoreSight Debug ETM

by

ULINK*plus*

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:

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 µVision 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. µVision 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. µVision 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 µVision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
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. µVision 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 µVision 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 5.24 Lite or later from the Keil website. www.keil.com/mdk5/install 

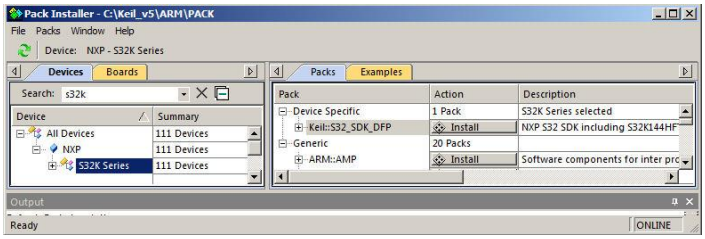
2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5
3. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
4. You do not need a debug adapter for the basic exercises: just the S32K board, a USB cable and MDK installed.
5. For the exercises using SWV, you need a Keil ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* or a J-Link.
6. For the exercises using ETM trace, you need a ULINK*pro*.
7. You do not need a Keil MDK license for this tutorial. All examples will compile within the 32 K limit.

6) µVision Software Pack Download and Install Process:

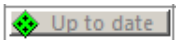
A Software Pack contain components such as header, Flash programming, documents and other files used in a project.

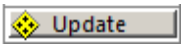
1) Start µVision and open Pack Installer:

1. Connect your computer to the internet. This is needed to download the Software Packs. Start µVision: 
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. Note "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet before continuing.
5. 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.



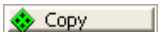
2) Install The S32K Software Pack:

1. In the Devices tab, select NXP and then S32K Series as shown above: The devices supported are displayed.
2. Select Keil::S32_SDK_DFP and click Install. This Pack will download and install into MDK. This download can take several minutes. This installs both S32K-144 and S32K-148 processor software.
3. Its status is indicated by the "Up to date" icon: 

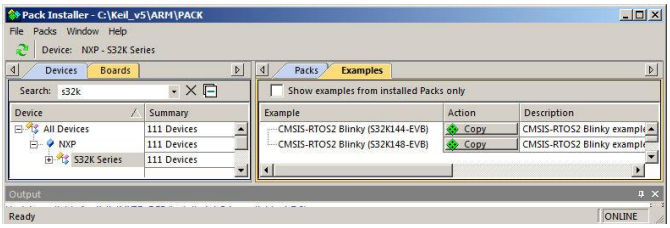
4. Update means there is an updated Software Pack available for download. 

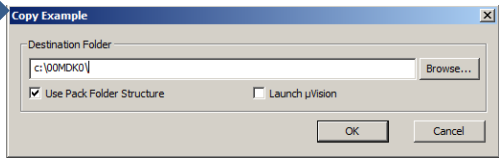
TIP: The left hand pane filters the selections displayed on the right pane. You can start with either Devices or Boards.

3) Install the RTX5_Blinky Example:

1. Select the Examples tab:
2. Opposite CMSIS-RTOS2 Blinky (S32K148-EVB): select Copy: 

3. The Copy Example window opens up: Select Use Pack Folder Structure. Unselect Launch µVision.
4. Type in C:\00MDK\. Click OK to copy the RTX5_Blinky project.
5. The RTX5_Blinky example will now copy to C:\00MDK\addon_mdk\Boards\NXP\S32K148-EVB\





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

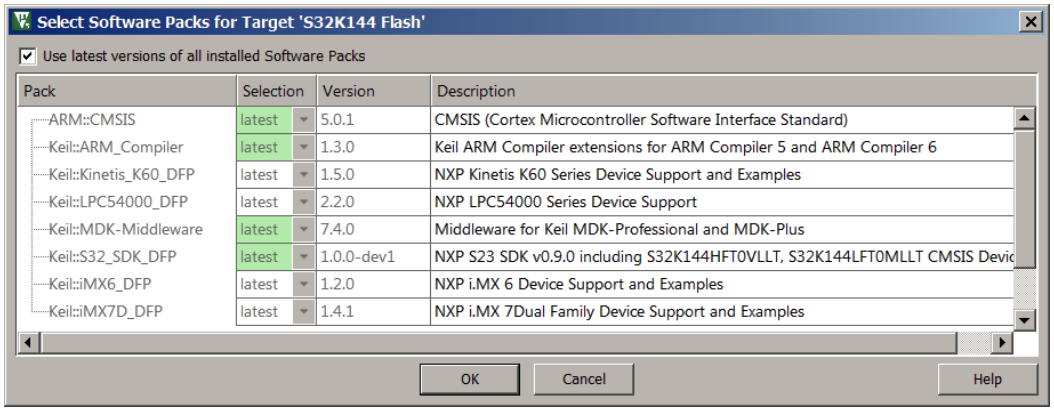
6. Close the Pack Installer. You can open it any time by clicking on its icon. 

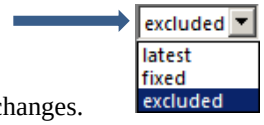
7) Other features of Software Packs:

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.

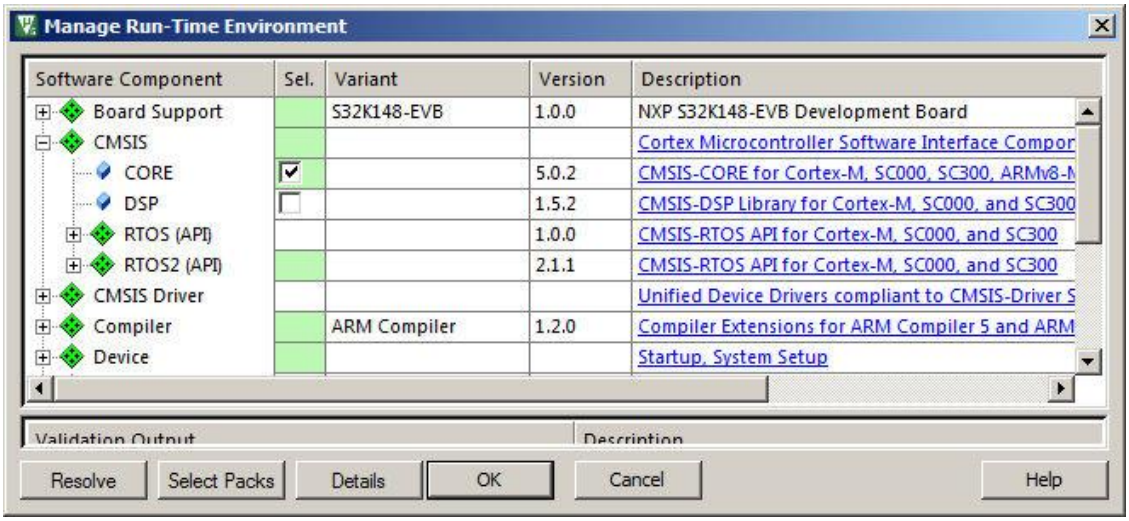


4. Note various options are visible as shown here: 

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. Select Project/Open Project.
2. Open the project: C:\00MDK\addon_mdk\Boards\NXP\S32K148-EVB\RTX5_Blinky\Blinky.uvprojx.
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. You can select NXP SDK files from under the Device header.
5. Do not make any changes. Click Cancel to close this window.



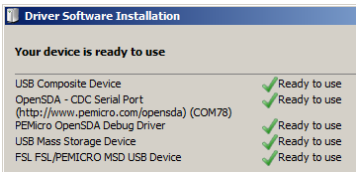
TIP: µVision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

8) Configuring OpenSDA in P&E Mode:

If you are using any Keil ULINK, CMSIS-DAP or J-Link as your debug adapter: you can skip this page:
µVision supports OpenSDA on this board in either P&E or CMSIS-DAP mode. This allows debugging the S32K board with a USB cable. No external adapter is required. The P&E firmware is installed on the S32K148 board at delivery. OpenSDA in CMSIS-DAP mode is described on the next page.

If you decide to use a ULINK2, ULINK-ME or ULINK*plus*, you will get Serial Wire Viewer (SWV). With a ULINK*pro*, ETM Trace is added which records all executed instructions and provides Code Coverage and Performance Analysis.

Install the P&E USB drivers:



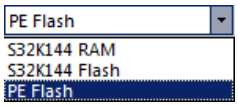
1. Plug a USB cable from your PC to J7 on the S32K board.
2. Windows will automatically install the necessary P&E USB drivers.
   Green LEDs D2 and D3 will light. If Blinky is installed, LED D11 will blink.

Start µVision and select the Blinky Project:

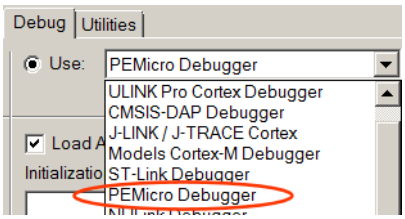
1. Start µVision by clicking on its desktop icon. 
2. Select Project/Open Project.
3. Open the project: C:\00MDK\addon_mdk\Boards\NXP\S32K144-EVB\RTX5_Blinky\Blinky.uvprojx.

Create a new Target Selection for P&E OpenSDA:

1. Select Project/Manage/Project Items… or select: 
2. In the Project Targets area, select NEW  or press your keyboard INSERT key.
3. Enter **PE Flash** and press Enter. Click OK to close this window.
4. In the Target Selector menu, select the PE Flash selection you just made: 

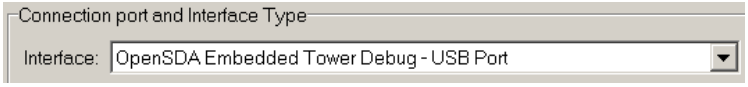


5. Select Options for Target  or ALT-F7. Click on the Debug tab to select a debug adapter.
6. Select PEMicro Debugger… as shown here: <an important step> 

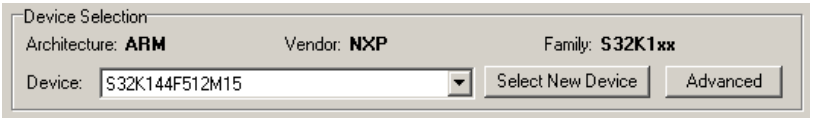


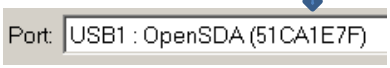
Configure the P&E Connection Manager: (the board must be connected)

1. Click on Settings: The P&E Connection Manager window opens.
2. In the Interface box, select OpenSDA Embedded Tower Debug: USB Port: as shown here:

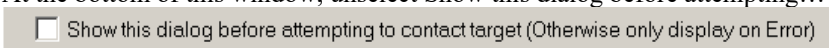


3. Click the Select New Device box, and select your exact processor. In this case it will be S32K-**148**xxx not as shown here: This step is very important.
4. Click on the Refresh List and you will get a valid Port: box:





5. This means µVision is connected to the target S32K processor using P&E OpenSDA.
6. At the bottom of this window, unselect Show this dialog before attempting… as shown below:



7. Click on OK to close this window.
8. If you see **Undetected** in Port:, this means µVision is not connected to the target. Problems can be the S32K board is not connected to USB, or the wrong device is selected. Fix the problem and click Refresh List to try again.
9. Select File/Save All or click  . P&E OpenSDA is now completely configured including Flash programming.
10. You can go to page 10 to compile and run Blinky !

TIP: You can program the on-board OpenSDA debug adapter U510 to run in CMSIS-DAP mode. This procedure is described on the next page. CMSIS-DAP currently has more features than P&E.

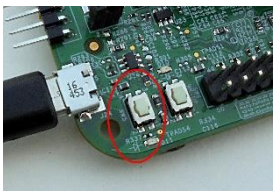
9) Configuring OpenSDA in CMSIS-DAP mode:

If you are using any Keil ULINK, or a J-Link as your debug adapter: you can skip this page:
This document will use NXP OpenSDA as an CMSIS-DAP debug adapter. This will replace the P&E debugger that comes pre-installed on a new S32K board. Target connection by µVision will be via a standard USB cable connected to USB connector J7. The on-board Kinetis K20 processor U510 acts as the CMSIS-DAP on-board debug adapter.

Program the K20 with the CMSIS-DAP application file CMSIS-DAP.S19:

1) Locate the file CMSIS-DAP.S19:

1. CMSIS-DAP.S19 is located where this document is located. www.keil.com/appnotes/docs/apnt_305.asp. You will now copy this file into the S32K board USB device.



2) Put the S32K Board into Bootloader: Mode:

2. Hold RESET button SW5 on the board down and connect a USB cable to J24 USB.
3. When you hear the USB dual-tone, release the RESET button to enter bootloader mode.
4. The **S32K board** will act as a USB mass storage device called BOOTLOADER connected to your PC. Open this USB device with Windows Explorer.

3) Copy CMSIS-DAP.S19 into the S32K Board:

5. Copy and paste or drag and drop CMSIS-DAP.S19 into this Bootloader USB device.

4) Exit Bootloader Mode:

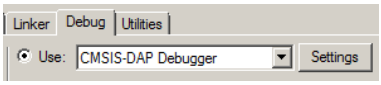
6. Cycle the power to the board by removing and reconnecting the USB cable while ***not*** holding RESET button down.
7. The **S32K** board is now ready to connect to µVision as a CMSIS-DAP debug adapter.

TIP: This application will remain in the U510 K20 Flash each time the board power is cycled with RESET not pressed. The next time board is powered with RESET held on, it will be erased. CMSIS-DAP.S19 is the CMSIS application in the Motorola S record format that loads and runs on the K20 OpenSDA processor.

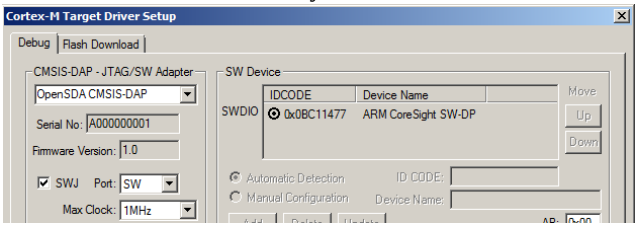
TIP: If you must later re-program CMSIS-DAP.S19 and it still does not work with µVision: check that Port: is set to SW and not JTAG. See the **TIP:** below.

10) Testing The OpenSDA CMSIS-DAP Connection: (Optional Exercise)

1. Start µVision if it is not already running. Select Project/Open Project.
2. Select the Blinky project C:\00MDK\addon_mdk\Boards\NXP\S32K148-EVB\RTX5_Blinky\Blinky.uvprojx.
3. Connect a 12 volt supply to J9 barrel connector. See **Board Power** below.
4. Connect USB to your PC.



5. Select Target Options or ALT-F7 and select the Debug tab:
6. Select CMSIS-DAP Debugger as shown here:
7. Click on Settings: and the window below opens up: Select SW in the Port box as shown below. An IDCODE and Device name will then be displayed indicating connection to the CoreSight DAP. This means CMSIS-DAP OpenSDA is working. You can continue with the tutorial. Click on OK twice to return to the µVision main menu.
8. If nothing or an error is displayed in this SW Device box, this ***must*** be corrected before you can continue.
9. Select File/Save All or .



TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window. CMSIS-DAP will not work with JTAG selected, only SW. But this is a useful way to refresh the SW setting.

Board Power: The NXP documentation for this board states that to power the main circuitry with the OpenSDA connector R537 must be removed. The alternative is to power the board with an external 12 volt supply. Since this external supply is needed for many board operations such as CAN, we will use such a supply in this lab. Refer to the Sk32K-148 docs.

11) Configuring External Debug Adapters in µVision:

It is easy to configure for a variety of Debug Adapters such as **ULINK2/ME, ULINK*plus*, ULINK*pro*, or a J-Link:**

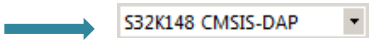
The RTX5_Blinky example is preconfigured for a Keil ULINK*pro* with three pre-selections for RAM, Flash and ETM trace. If you are using a ULINK*pro*, just select S32K148 Flash as in Step 1 and then jump to "Testing the Debug Connection" below. You can add a configuration for a ULINK*pro*, a J-Link or OpenSDA in either P&E or CMSIS-DAP modes easily.

Prerequisites:

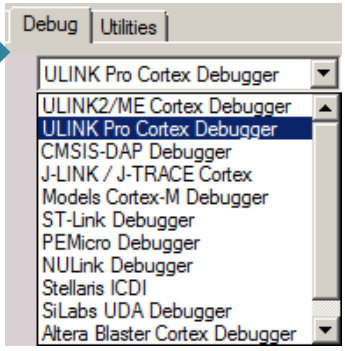
µVision must be running and in Edit mode (not Debug mode). Your project must be loaded. We will use Blinky.

Create a new Target Selection:

1. Select S32K148 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 your debug adapter name (we will use S32K-148 CMSIS-DAP for example) and press Enter.
5. Click OK to close this window.
6. In the Target Selector menu, select the menu item you just made: 

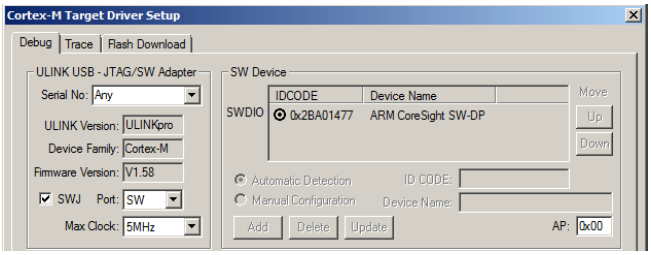
7. Select Options for Target  or ALT-F7. Click on the Debug tab to select a debug adapter. (use CMSIS-DAP now)
8. Select your debug adapter. Valid options are ULINK2/ME, ULINK*plus*, ULINK Pro Cortex Debugger, CMSIS-DAP, J-Link/JTRACE or PEMicro as shown:



TIP: For the selections CMSIS-DAP or PEMicro Debugger, the onboard OpenSDA K20 processor must have the proper firmware installed as described on the previous two pages.

Testing the Debug Connection:

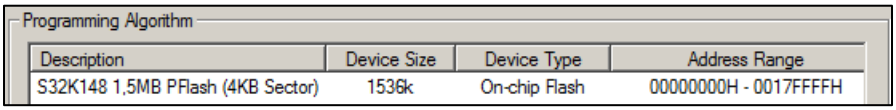
1. Select Settings: to test the connection to the target board. Connect the appropriate Debug Adapter connected to the 20 pin Cortex Debug ETM connector J10 or for OpenSDA, connect a USB cable to J24. Power the board with 12 volts.
2. Click on Settings: and the window below opens up: Select SW in the Port box and SWJ as shown below. An IDCODE and Device name will be displayed indicating connection to the processor.
3. If nothing or an error is displayed in this SW Device box, this ***must*** be corrected before you can continue. An RDDI error usually means the target processor is not powered. Attach an external 12 volt supply to the barrel connector.
4. Select File/Save All or  .



TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window. CMSIS-DAP will not work with JTAG selected, only SW. But this is a useful way to refresh the SW setting.

Verify the Flash Program Algorithm: This is preset by the Software Pack when you select the processor.

1. Select the Flash Download tab. You can also access this page with the Utilities tab beside the Debug tab.
2. The window that opens will display the correct algorithm. This is selected automatically according to the processor selected in the Device tab.
3. Below is the correct algorithm for the S32K148 processor:
4. Click OK twice to return to the main µVision window.



The new Debug Adapter is now ready to use.

TIP: You have now created a new Target Option. You can make any changes in the Options Target windows and save it. These are easily recalled by selecting the appropriate Options Target in the drop down menu as described above.

12) *Blinky* example program using the NXP S32K148 EVB board:

Now we will connect a Keil MDK development system using the S32K board. This page will use the OpenSDA CMSIS-DAP mode you configured on the previous page. You can use any debug adapter that you have programmed and connected according to one of the three preceding pages.

1. Connect a USB cable between your PC and the S32K board J24 USB connector OpenSDA **OR**.
2. If you are using an external debug adapter, connect it to J10 Cortex Debug ETM connector.
3. Power the board with 12 volts to J10 barrel connector J9.
4. Start µVision by clicking on its desktop icon.
5. Select Project/Open Project.
6. Open the file: C:\00MDK\addon_mdk\Boards\NXP\S32K148-EVB\RTX5_Blinky\Blinky.uvprojx.
7. Choose your debug adapter that you configured previously: We will use CMSIS-DAP: S32K148 CMSIS-DAP
8. Compile the source files by clicking on the Rebuild icon.
9. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Progress will be indicated in the Output Window or in the P&E window. Select OK if the Evaluation Mode box appears.

TIP: If the Flash programs with P&E but does not enter debug mode, select Debug mode again:

10. Click on the RUN icon.

The tri-colour LED will now blink in sequence on the S32K board.

Now you know how to compile a program, program it into the S32K 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.

11. Stop the program with the STOP icon.

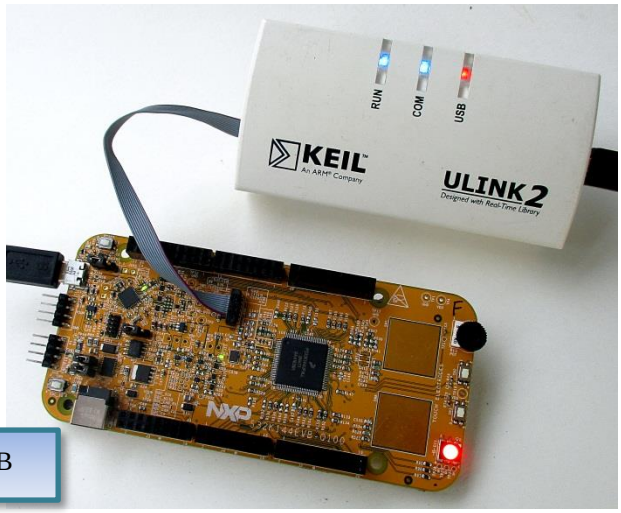
Single-Stepping:

1. With Blinky.c in focus (the Blinky.c tab is underlined), click on the Step In 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.
2. Click on the top margin of the Disassembly window to bring it into focus. Clicking Step Into now jumps the program counter one assembly instruction at a time.

Debug Adapters: Which one to use ?

You can use a variety of debug adapters with your S32K and µVision. Their feature list increases as follows:
The one with the higher number has the most performance.

1. P&E OpenSDA (on-board)
2. CMSIS-DAP OpenSDA (on-board)
3. Keil ULINK2 or ULINK-ME
4. Segger J-Link
5. Keil ULINK*plus*
6. Keil ULINK*pro*

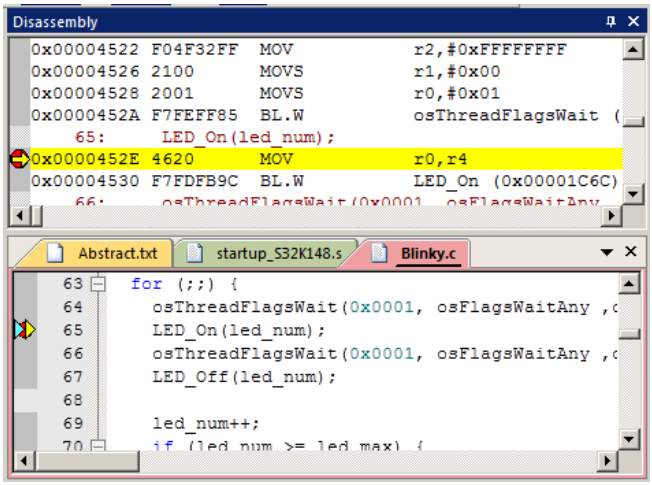


Keil ULINK2 with NXP S32K144 EVB

13) Hardware Breakpoints:

The S32 has six hardware breakpoints that can be set or unset on the fly while the program is running and using a CMSIS-DAP, any Keil ULINK or a J-Link debug adapter. **You must stop the program to set/unset breakpoints with P&E.**

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 for loop in the thread **thrLED** between near lines 64 through 71 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. Remember to restart the program if using P&E.
3. Note the breakpoint is displayed in both the Disassembly and source windows as shown here:
4. Set a second breakpoint in the for (;;) 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, µVision will warn you.

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

Keil ULINK*pro* with NXP S32K144 EVB

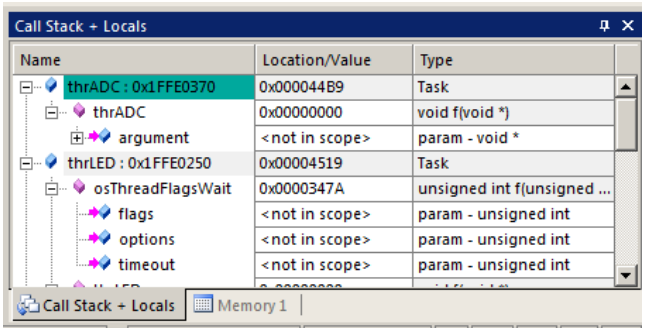
14) Call Stack + Locals Window:

Local Variables:

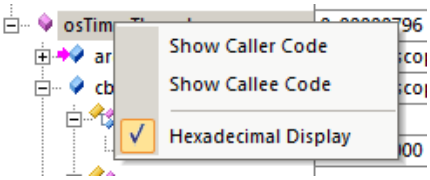
The Call Stack and Locals windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables located in the active function or thread.

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. Set a breakpoint on one of the lines inside the for loop in the thread **thrADC** in Blinky.c near lines 46 through 50.
2. Click on RUN  . The program will stop on the breakpoint.
3. Click on the Step In icon  to enter a few functions.
4. Click on the Call Stack + Locals tab if necessary to open it. Expand some of the entries.
5. As you click on Step In, you can see the program entering and perhaps leaving various functions. Note the local variables are displayed
6. Shown is an example Call Stack + Locals window:
7. The functions as they were called are displayed. If these functions had local variables, they would be displayed.
8. If you get stuck in a delay or the os_idle_Daemon, click on RUN to start over.



9. If using P&E and the program does not stop, click Stop. 
10. Click Step Out  to immediately exit a function.
11. Right click on a function and select either Callee or Caller code and this will be highlighted in the source and disassembly windows.



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

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

TIP: This window is only valid when the processor is halted. It does not update while the program is running 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.

Do not forget to remove any hardware breakpoints before continuing.

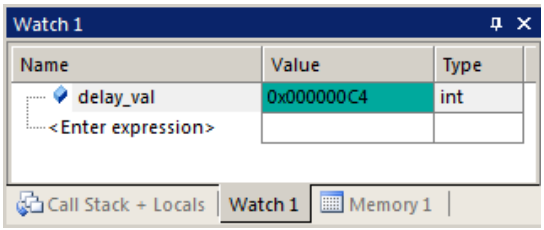
15) Watch and Memory Windows and how to use them:

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.

Watch window:

A global variable: The global variable delay_val is declared in Blinky.c near line 34.

1. Leave Blinky running.
2. You can configure a Watch or Memory window while the program is running.
3. **Note:** With P&E, you must stop the program to configure Watch window and to view the variable.
4. In Blinky.c, right click on delay_val and select Add delay_val to … and select Watch 1. Watch 1 will automatically open. delay_val will be displayed as shown here:
5. Vary the ADC POT R186 and delay_val will update in real time.
6. With P&E, you must stop the program to update the window.

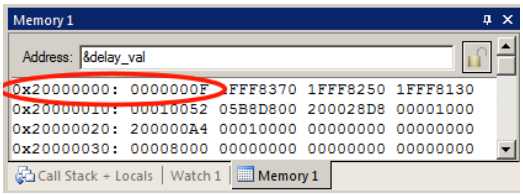


TIP: To Drag 'n Drop into a tab that is not active, pick up the variable and hold it over the tab you want to open; when it opens, move your mouse into the window and release the variable.

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

Memory window:

1. Right click on delay_val and select Add delay_val to … and select Memory 1.
2. Vary ADC POT R186. Note: With P&E you must stop the program to see the updated variable.
3. Note the value of delay_val 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.
4. Add an ampersand "&" in front of the variable name and press Enter. The physical address here is 0x2000_0000.
5. Right click in the Memory window and select Unsigned/Int.
6. The data contents of delay_val is displayed as shown here:
7. Both the Watch and Memory windows are updated in real-time.
8. 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. You will not see any change as this variable is constantly updated.



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.

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. 96 MHz will be displayed. SystemCoreClock is provided by CMSIS to help determine the CPU clock frequency.

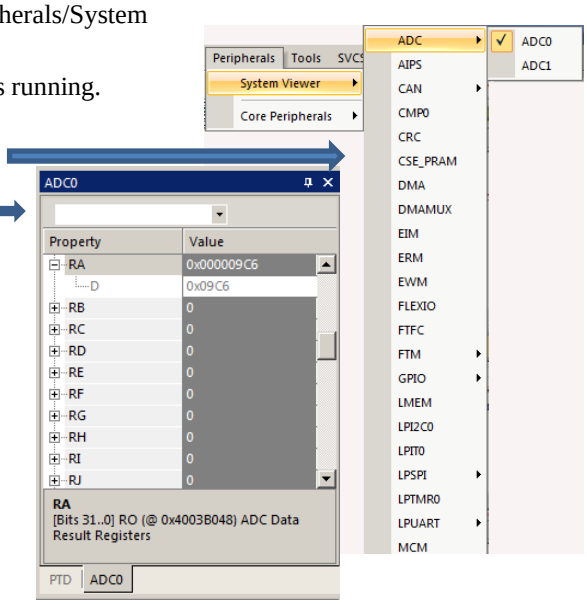
16) Peripheral System Viewer (SV):

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:

2. Select Peripherals/System Viewer and then ADC0 as shown here.
3. This window opens up. Expand RA:
4. You can now see RA update as the pot is changed.
5. 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.
6. 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.

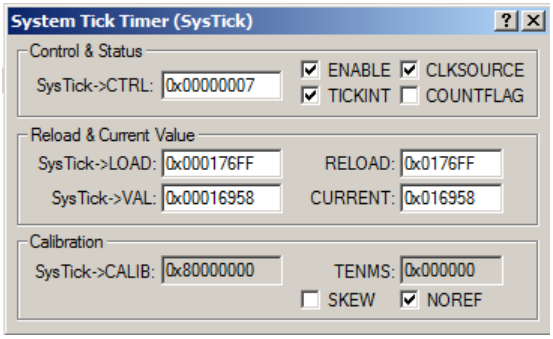
TIP: You can also open GPIO PTE to monitor the LEDs blinking.

SysTick Timer: This program uses the SysTick timer as a tick timer for RTX. RTX has configured the SysTick timer in RTX_Config.h.

1. Select Peripherals/Core Peripherals and then select SysTick Timer.
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 ST_RELOAD and RELOAD registers. This is the reload register value. This is set during the SysTick configuration by RTX using values set in RTX_Config.h Kernel Tick Frequency and the CPU clock.
5. Note that it is set to 0x176FF. This is the same value hex value of 96,000,000/1000-1 (0x17700-1) that is programmed into RTX_Config.h. 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 RELOAD register in the SysTick window, *while the program is running*, type in 0x5000 and click inside ST_RELOAD ! (or the other way around)
7. The blinking LEDs will speed up. This will convince you of the power of Arm CoreSight debugging.
8. Replace RELOAD with 0x176FF. A CPU RESET 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.

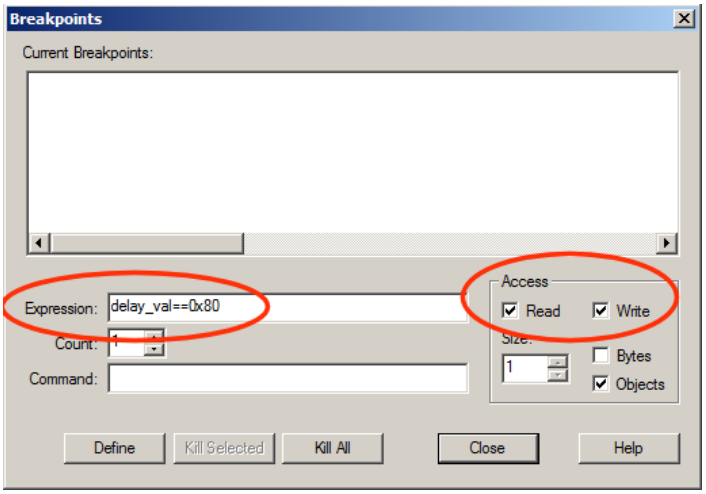
You must make sure a given peripheral register allows and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.



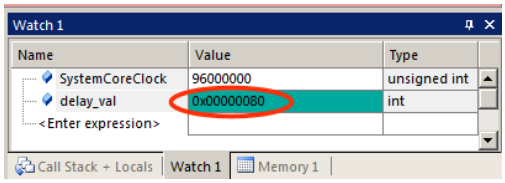
17) **Watchpoints:** *Conditional Breakpoints*

The S32K Cortex-M4 processor has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil documents. Cortex-M3/M4/M7 Watchpoints are not intrusive for equality test. Currently, you can set one Watchpoint with µVision.

1. Use the same Blinky configuration as the previous page. You can configure a Watchpoint while the program is running or halted.
2. We will use the same global variable **`delay_val`** found in Blinky.c you used to explore the Watch windows.
3. Select Debug in the main µVision window and then select Breakpoints or press Ctrl-B.
4. Select Access to Read and Write.
5. Enter: "delay_val == 0x80" without the quotes in the Expression box. This window will display:
6. 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:
7. Click on Close.
8. Enter the variable delay_val in Watch 1 if it is not already there.
9. Click on RUN.  .
10. Vary ADC POT R186 until delay_val equals 0x80 as displayed in the Watch window.



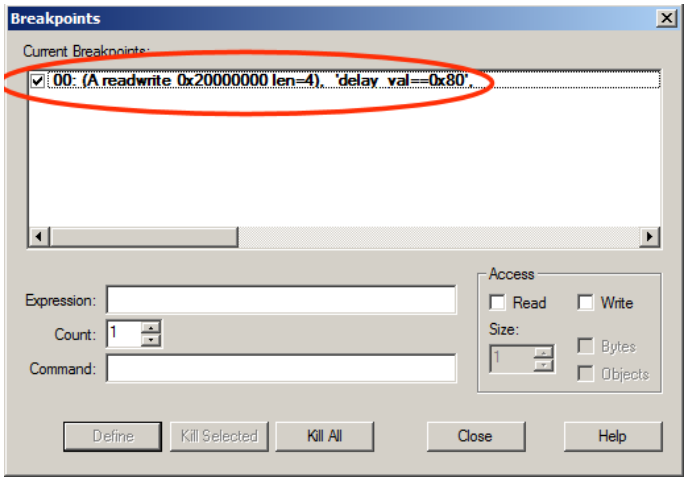
11. When delay_val equals 0x80, the Watchpoint will stop the program. See Watch 1 shown below:
12. Watchpoint expressions you can enter are detailed in the Help button in the Breakpoints window. Triggering on a data read or write is most common. You can leave out the value and trigger on just a Read and/or Write as you select.
13. This is useful to detect stack pointer levels. Set a Watchpoint on a low stack address. If hit, the program will stop.
14. To repeat this exercise, change `delay_val` to something other than 0x80 in the Watch window and click on RUN.
15. Stop the CPU if it is running. 
16. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.
17. 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)



18) RTX System and Threads Viewer: Need OpenSDA P&E or CMSIS-DAP, any ULINK or J-Link.

This example uses the new Arm RTX 5 RTOS. It has an Apache 2.0 license and sources and documents are included. See http://www2.keil.com/mdk5/cmsis/rtx. It is also located on https://github.com/ARM-software/CMSIS_5

With previous versions of RTX, the System and Thread Viewer was opened in Open Debug/OS Support.

With RTX5, this was moved to View/Watch/RTX RTOS.

Keil uses the term "threads" instead of "tasks" for consistency. You do not need to run RTX or any RTOS in your project. Using an RTOS is becoming increasingly common as projects complexity increases.

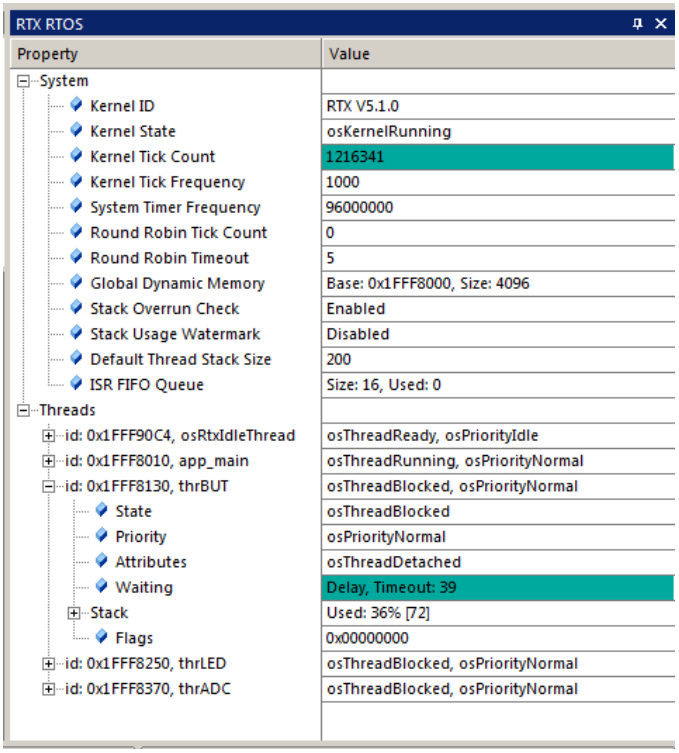
NOTE: With OpenSDA in P&E mode, the program must be stopped to open the RTX RTOS window and to see updates. OpenSDA in CMSIS-MODE displays this window updating in real-time while the program runs.

Running System and Threads Viewer:

1. µVision must be in Debug mode and the program running. If using P&E, stop the program.
2. Select View/Watch/RTX RTOS. A window similar to below opens up. You may have to click on its header and drag it into the middle of the screen and resize the columns to comfortably view it.
3. This window updates as the various threads switch state. (except with P&E).
4. There are three Threads listed under the Threads heading. thrADC to operate the ADC to measure the pot R186 position. thrLED to change the LEDS and thrBUT is for the buttons, and These are all located in Blinky.c near lines 43, 59 and 80 respectively. It is easy to add more threads to RTX.
5. Stop the program.  It will probably stop in app_main thread in the for (;;) as shown by osThreadRunning. This program spends most of its time in app_main. This can be adjusted to meet your needs.

Stopping in a Thread:

1. Set a breakpoint in one of the three tasks in Blinky.c by clicking in the left margin on a grey area. Do not select the for (;;) statement as this will not stop the program. This line (NOP) is executed only once at the start of the program.
2. Click on Run  and the program will stop at this thread and the System and Threads Viewer will be updated.
3. You will be able to determine which thread is running when the breakpoint was activated. This is shown by osThreadRunning displayed beside the thread name.
4. If you set a breakpoint in another thread, each time you click on RUN, the next task will display as Running.
5. Remove all the breakpoints by clicking on each one. You can use Ctrl-B and select Kill All.
6. Stay in Debug mode for the next page.



TIP: You can set/unset hardware breakpoints while the program is running.

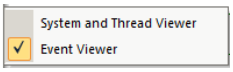
TIP: Recall this window uses CoreSight DAP read and write technology to update this window. Serial Wire Viewer is not used and is not required to be activated for this window to display and be updated.

19) Event Viewer: Uses SWV: ULINK2, ULINK*plus*, ULINK*pro* or J-Link:

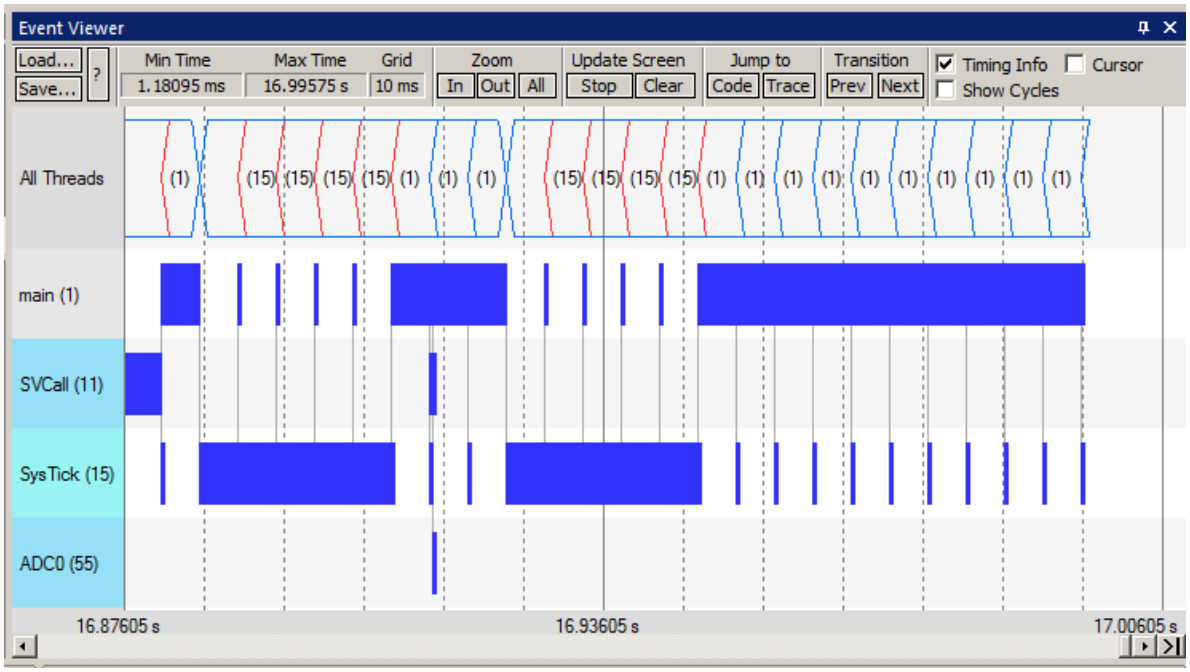
Event Viewer displays the RTX threads executing in a graphical format. It is easy to make measurements of various timings of RTX. FreeRTOS has a similar window in µVision. A ULINK*pro* has the advantage of also displaying interrupt routines.

Viewing Event Viewer:

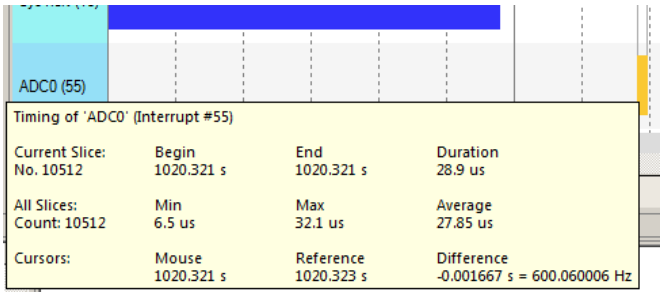
1. Run RTX5_Blinky with a suitable debug adapter connected and configured for Serial Wire Viewer operation.
2. Select Debug/OS Support and then select Event Viewer.



3. The Event Viewer window below will open:
4. Note: In This version of Event Viewer, only the main() thread is implemented for RTX5. This window is still under development. The other threads will be added in a future version of µVision. will be added.
5. Event Viewer for other versions of RTX are fully implemented.
6. The interrupt handlers for SVCall, SysTick and ADC0 are displayed as a ULINK*pro* was used. This will also work with a ULINK*plus* which adds power sensing capability.



7. Select Cursor and you can make various measurements of the waveforms.
8. If you hover your mouse over a blue block, statistics will be displayed and the block turned yellow as shown below:
9. In this case, the mouse was over the ADC0 interrupt #55.
10. In this case, there were 10,512 ADC0 events. They lasted from 6.5 through 32.1 µs with 27.85 µs as the average time.



TIP: You can stop the update of this Event Viewer without stopping the program.

20) Event Recorder: Uses DAP R/W: Works with any debug adapter.

Event Recorder is a new µVision feature. Code annotations can be inserted into your code to send out messages to µVision 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.

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

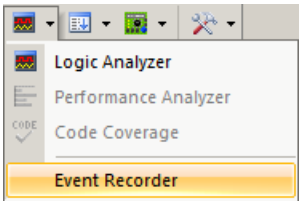
Event Recorder

Enable Recorder: ☑ Mark: All Operations Stopped

Event	Time (sec)	Component	Event Property	Value
0	0.01588850		Init Event	Restart Count=0x00000001
1	0.01593770	RTX Kernel	KernelInitialize	
2	0.01632660	RTX Thread	ThreadNew	func=app_main, argument=0x00000000, attr=0x00000000
3	0.01658320	RTX Kernel	KernelGetState	state=osKernelInactive
4	0.01663840	RTX Kernel	KernelStart	
5	0.01686830	RTX Thread	ThreadNew	func=thrBUT, argument=0x00000000, attr=0x00000000
6	0.01713370	RTX Thread	ThreadNew	func=thrLED, argument=0x00000000, attr=0x00000000
7	0.01740950	RTX Thread	ThreadNew	func=thrADC, argument=0x00000000, attr=0x00000000
8	0.07453070	RTX Thread	ThreadDelay	ticks=500
9	0.07466330	RTX Thread	ThreadFlagsWait	flags=0x00000001, options=0x00000000, timeout=-1
10	0.07487220	RTX Thread	ThreadDelay	ticks=20
11	0.26659260	RTX Thread	ThreadDelay	ticks=20
12	0.45859260	RTX Thread	ThreadDelay	ticks=20
13	0.65059220	RTX Thread	ThreadDelay	ticks=20
14	0.84259260	RTX Thread	ThreadDelay	ticks=20
15	1.03459260	RTX Thread	ThreadDelay	ticks=20
16	1.22659260	RTX Thread	ThreadDelay	ticks=20
17	1.41859260	RTX Thread	ThreadDelay	ticks=20
18	1.61059260	RTX Thread	ThreadDelay	ticks=20

Demonstrating Event Recorder with RTX5_Blinky:

1. µVision must be running Blinky from the previous page.
2. Open Event Recorder by selecting View/Analysis/Event Recorder or



3. Since Event Recorder is activated in Blinky.c, the window above will display.
4. Various RTX events will display as they happen. You can do this for your own code.

Event Recorder Features:

1. Stop and start Event Recorder while the program is running: Enable Recorder: ☑
2. Clear the window when the program is not running:
3. Stop the program.
4. In the Mark: box, enter ThreadDelay and these frames will be highlighted as shown here: This is useful to find events that do not occur frequently.
5. If you click on a frame in the Event Property column, you will be taken to Help for this event.
6. Hover your mouse over an event in the Value column and a hint will display such as this one:

osThreadFlagsSet function was called.

Event Recorder

Enable Recorder: ☑ Mark: ThreadDelay All Operations Stopped

Event	Time (sec)	Component	Event Property	Value
0	36978.77539070	RTX Thread	ThreadDelay	ticks=20
1	36978.92899500	RTX Thread	ThreadFlagsSet	thread_id=0x1FFF8250, flags=0x00000001
2	36978.92917570	RTX Thread	ThreadDelay	ticks=70
3	36978.98654190	RTX Thread	ThreadFlagsWait	flags=0x00000001, options=0x00000000, timeout=-1
4	36978.98676130	RTX Thread	ThreadDelay	ticks=20
5	36979.17859070	RTX Thread	ThreadDelay	ticks=20
6	36979.37059070	RTX Thread	ThreadDelay	ticks=20

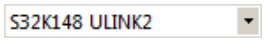
7. Stop the program. Close any Event Recorder and Threads and Event (RTX RTOS) windows.
8. Exit Debug mode.

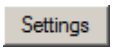
21) Serial Wire Viewer (SWV) Configuration: For Keil ULINK2/ME, ULINK*plus* or J-Link.

Serial Wire Viewer is a data trace including interrupts in real-time without any code stubs in your sources. SWV is output on the 1 bit SWO pin found on the JTAG/SWD connectors, either 10 or 20 pin. SWO is shared with JTAG TDO pin. This means you must use SWD (Keil SW) and not JTAG for debugging to avoid this conflict. SWD has essentially the same abilities as JTAG except for Boundary Scan. It is possible to use the 4 bit Trace Port. See the next page.

These instructions are not for ULINK*pro*: they are on the next two pages.

Configure SWV:

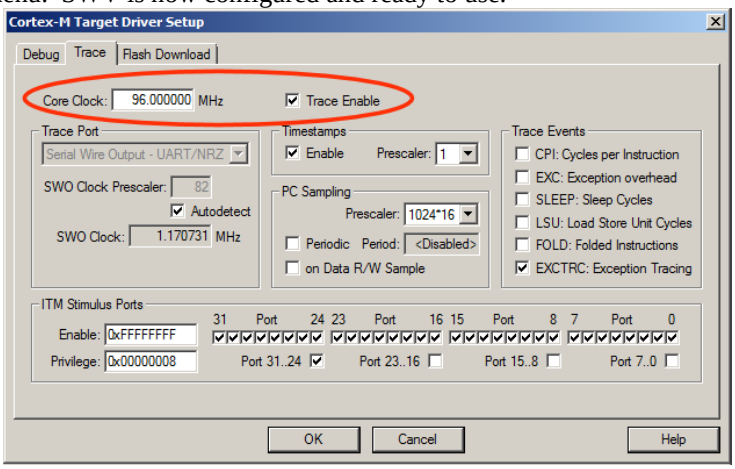
1. Connect a ULINK2/ME, ULINK*plus* or J-Link to the 20 pin connector Cortex Debug ETM. You will need a 10 pin to 20 pin adapter cable.
2. µVision must be stopped and in Edit mode (not Debug mode).
3. Configure your debug adapter as described on page 9: **11) Configuring External Debug Adapters in µVision:**
4. Select the Debug adapter you just configured in Step 3: Here is a ULINK2 example: 

2. Select Options for Target  or ALT-F7 and select the Debug tab. Your debugger must be displayed beside Use:.
3. Select Settings:  on the right side of this window.

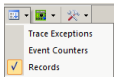
4. Confirm Port: is set to SW and SWJ box is enabled for SWD operation. SWV will not work with JTAG.
5. Click on the Trace tab. The window below is displayed.
6. In Core Clock: enter 96 MHz. Select Trace Enable. This value ***must*** be set correctly to your CPU speed.

TIP: To find Core Clock frequency: Enter the global variable SystemCoreClock in a Watch window and run the program.

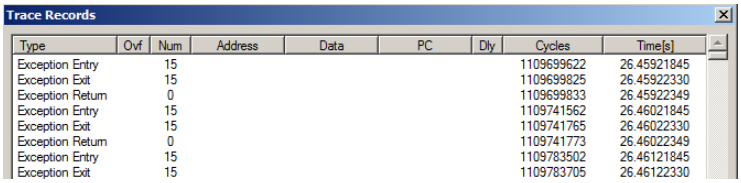
7. Click on OK twice to return to the main µVision menu. SWV is now configured and ready to use.



Display Trace Records:

1. Select File/Save All or click  .
2. Enter Debug mode. 
3. Click on the RUN icon. 
4. Open Trace Records window by clicking on the small arrow beside the Trace icon  and select Records: 

5. The Trace Records window will open:
6. If you see Exceptions as shown, SWV is working correctly. If not, the most probably cause is a wrong Core Clock:.
7. Double-click inside Trace Records to clear it.
8. Exception 15 is the SYSTICK timer. It is the timer provided for RTOS use.
9. All frames have a timestamp displayed.



You can see two exceptions happening:

1. Num 11 is SVCall from the RTX calls.
2. Num 15 is the Systick timer.
   - **Entry:** when the exception enters.
   - **Exit:** When it exits or returns.
   - **Return:** When all the exceptions have returned to the main program. This is useful to detect tail-chaining.

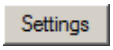
TIP: The only valid ITM frames are ITM 0 and ITM 31. If you see any other values, this nearly always means the Core Clock: value is incorrect. Since we are using a UART for SWO in this case, the frequency must be correct.

22) Serial Wire Viewer (SWV) Configuration with ULINK*pro* using 1 bit SWO Port:

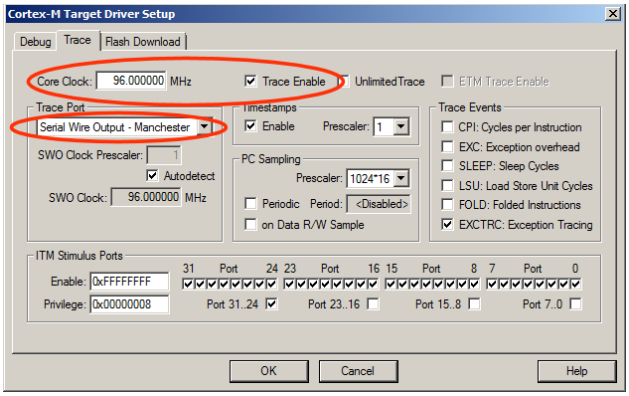
This configures the S32K148 to output out the 1 bit SWO port. For the faster 4 bit Trace Port, see the next page.

1) Configure SWV: (You can also use the 4 bit ETM Trace Port in suitably equipped NXP processors with a ULINK*pro*.)

1. µVision must be stopped and in Edit mode (not Debug mode). RTX_Blinky must be loaded.
2. Select S32K-148 Flash: 

This is preconfigured for ULINK*pro* but trace is not enabled:
3. Connect a ULINK*pro* to the 20 pin connector Cortex Debug ETM connector J10 as pictured on previous page.
4. Select Options for Target  or ALT-F7 and select the Debug tab. ULINK*pro* must be visible in the dialog box.
5. Select Settings: 

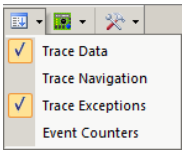
on the right side of this window.
6. Click on the Trace tab. The window below is displayed.
7. Set Core Clock: to 96 MHz. ULINK*pro* uses this only to calculate timings displayed in some windows.
8. Select the Trace Enable box.
9. In Trace Port, select Serial Wire Output - Manchester.
10. Select EXTRC to display exceptions and interrupts.
11. Click on OK twice to return to the main µVision menu. SWV is now configured and ready to use.
12. In this configuration, SWV data will be output on the 1 bit SWO pin as opposed to the 4 bit trace port.
13. Manchester mode is used rather than UART mode as used with other Keil ULINKs or a J-Link.



TIP: If Sync Trace Port with 4-bit Data is chosen in Trace Port: box, SWV data is sent out the 4 bit Trace Port pins if available on your processor. This has much higher data throughput than the 1 bit SWO pin. It is easy to overflow 1 bit trace data. A ULINK*pro* works the best at high SWO data rates. A ULINK*plus* provides high performance SWV capabilities for when there is no 4 bit Trace Port.

The 1 bit SWO port can be useful for high CPU speeds that ETM is unable to handle.

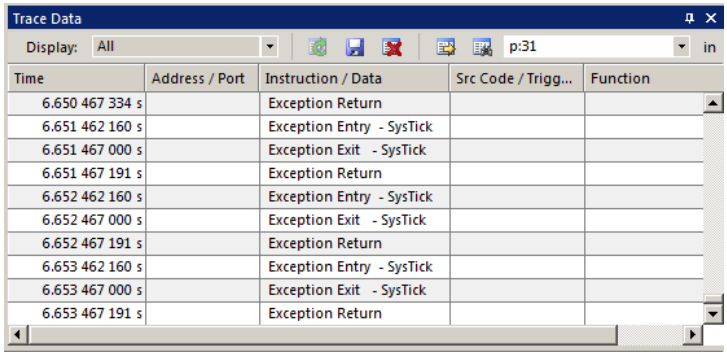
2) Display the Trace Data window:

1. Select File/Save All or click  . It is not necessary to rebuild the project.
2. Enter Debug mode.  Click on the RUN icon.  .
3. Open the Trace Data window by clicking on the small arrow beside the Trace icon: 


4. The Trace Data window shown below will open.
5. STOP  the program to display the Exceptions as shown below:

TIPS:

1. The Trace Data window is different than the Trace Records window provided with ULINK2.
2. Clear the Trace Data window by clicking 
3. The contents of the Trace Data window can be saved to a file. 
4. ULINK*pro* does not update the Trace Data window while the program is running. You must stop the program to view the Trace Data window.



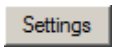
5. The Trace Port outputs SWV data faster than the 1 bit SWO with UART (ULINK2) or Manchester with ULINK*pro*. The 1 bit SWO port can still be useful for very high CPU speeds that ETM is unable to handle. (> ~ 100 MHz)

23) Serial Wire Viewer (SWV) Configuration with ULINK*pro* using 4 bit Trace Port:

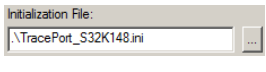
This configures the S32K148 to output out the 4 bit trace port. This adds ETM instruction trace to the SWV frames. For output on 1 bit SWO pin, see the preceding page. The target configuration contains a configured ETM entry but we will show you how to do it manually on this page.

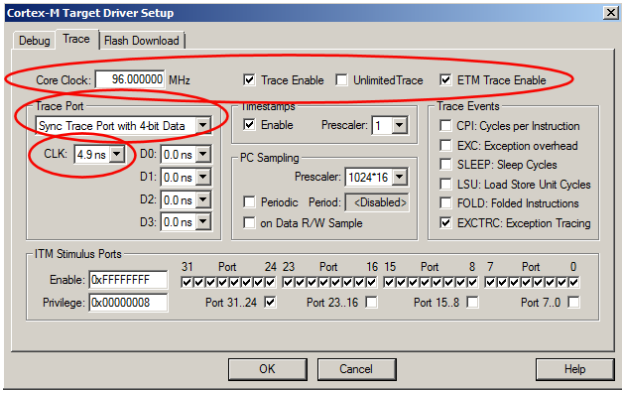
Configure SWV:

1. µVision must be stopped and in Edit mode (not Debug mode). RTX_Blinky must be loaded.
2. Select S32K-148 Flash: 

This is preconfigured for ULINK*pro* but trace is not enabled:
3. Connect a Keil ULINK*pro* to the 20 pin connector Cortex Debug ETM connector J10.
4. Select Options for Target  or ALT-F7 and select the Debug tab. ULINK*pro* must be visible in the dialog box.
5. In the Initialization File, use the browse button to select the script TracePort_S32K148.ini included in RTX_Blinky:
6. Select Settings: 

on the right side of this window.
7. Click on the Trace tab. The window below is displayed.
8. Set Core Clock: to 96 MHz. ULINK*pro* uses this only to calculate timings displayed in some windows.
9. Select the Trace Enable box.
10. In Trace Port, select Sync Trace Port with 4 bit Data.
11. Select ETM Trace Enable: Unselect Unlimited Trace.
12. Select 4.9 nsec in the CLK: box
13. Select EXTRC to display exceptions and interrupts.
14. Click on OK twice to return to the main µVision menu. The Trace Port is now configured and ready to use.
15. In this configuration, SWV data will be output on the 4 bit trace port. This gives much better SWV performance over the SWO port.



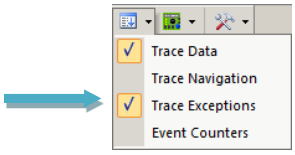


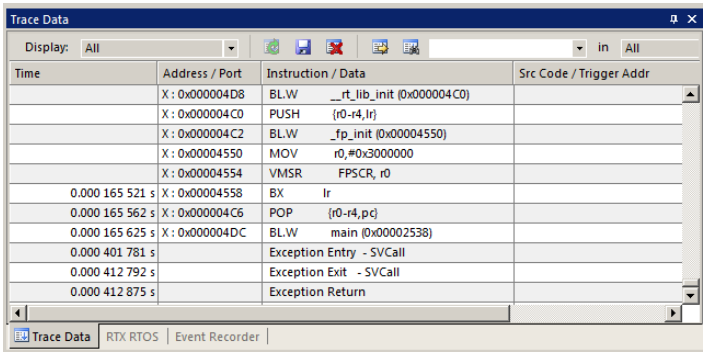
TIP: It is easy to overflow 1 bit trace data. A ULINK*pro* provides the highest SWV data rates. The 1 bit SWO port can be useful for high CPU speeds that ETM is unable to handle.

TIP: If you unselect ETM Trace Enable, you will only capture SWV data frames and not ETM instruction frames.

2) Display the Trace Data window:

1. Select File/Save All or click  . It is not necessary to rebuild the project.
2. Enter Debug mode.  The program will run to main() and collect trace frames.
3. Open the Trace Data window by clicking on the small arrow beside the Trace icon:
4. The Trace Data window shown below will open.
5. The Exceptions and Instructions created since RESET are displayed as shown below:
6. The last three frames are examples of SWV. All the other frames are ETM instruction trace frames.





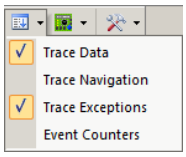
TIPS:

6. The Trace Data window is different than the Trace Records window provided with ULINK2.
7. Clear the Trace Data window by clicking 
8. The contents of the Trace Data window can be saved to a file. 
9. ULINK*pro* does not update the Trace Data window while the program is running.

24) Displaying Exceptions (includes interrupts) with SWV:

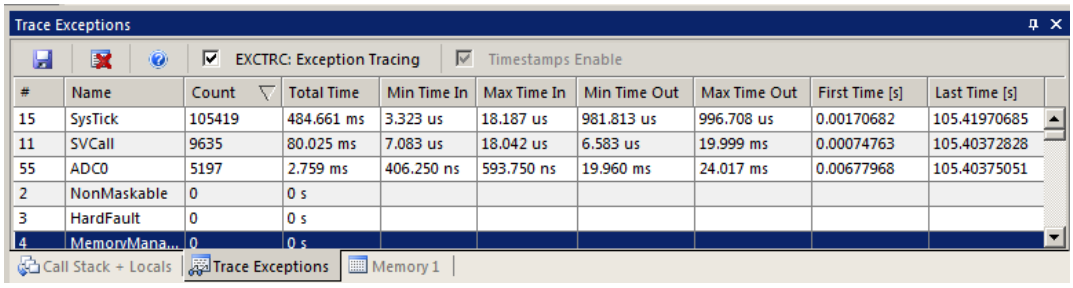
This exercise needs a ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* or a J-Link. Does not work with OpenSDA.

The Trace Exceptions window displays exceptions firing with suitable timing information.



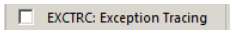
Display Trace Exceptions:

1. Open the Trace Exceptions window by selecting Trace Exceptions in the Trace window as shown here:
2. **OR:** Select the Trace Exception tab (located beside the Watch and Memory windows).
3. Click in the Count column header to bring the active exceptions to the top as shown below:



#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
15	SysTick	105419	484.661 ms	3.323 us	18.187 us	981.813 us	996.708 us	0.00170682	105.41970685
11	SVCall	9635	80.025 ms	7.083 us	18.042 us	6.583 us	19.999 ms	0.00074763	105.40372828
55	ADC0	5197	2.759 ms	406.250 ns	593.750 ns	19.960 ms	24.017 ms	0.00677968	105.40375051
2	NonMaskable	0	0 s						
3	HardFault	0	0 s						
4	MemoryMana...	0	0 s						

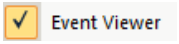
4. Note the various timings displayed. If you are using a ULINK*pro* and RTX, you can see these in graphical form in the Event Viewer. See below:

TIP: To quickly disable Trace Exceptions unselect EXCTRC: 

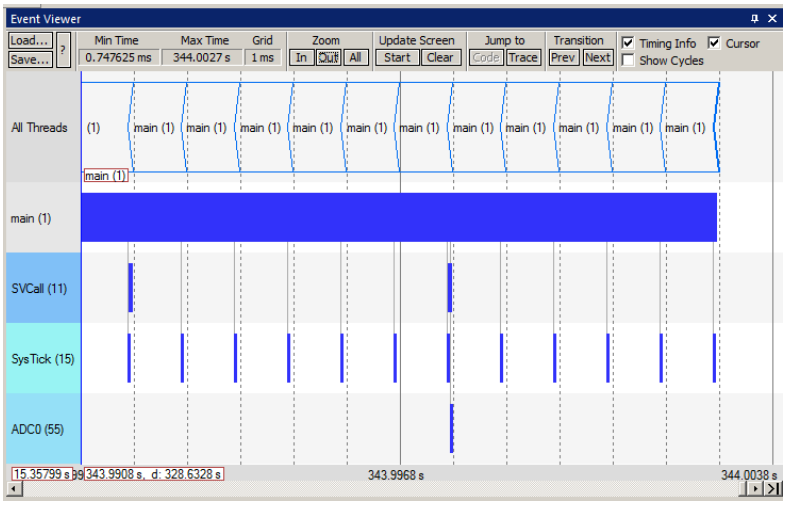
This is a quick way to disable Trace Exceptions if you have SWO overload. Exception frames are not captured and the bus load is less on the single bit SWO pin.

Display Trace Exceptions Graphically with a ULINK*pro* and Event Viewer: This needs a ULINK*pro*.

Event Viewer normally displays RTX threads and with a ULINK*pro*, exception timings are added. You can clearly see when the interrupt handlers were active in relation to each other and how long they were active in their respective handler routines.

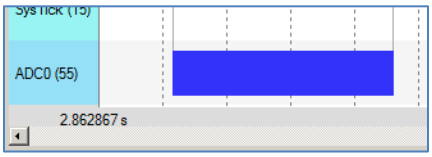
1. With a ULINK*pro* connected and RTX5_Blinky running, select Debug/OS Support/Event Viewer. 

2. This window opens up. Select a time of about 1 ms in Grid using the In and Out buttons.
3. You can see the three exceptions activated. The width of the blue bars indicate the exception handler execution times.

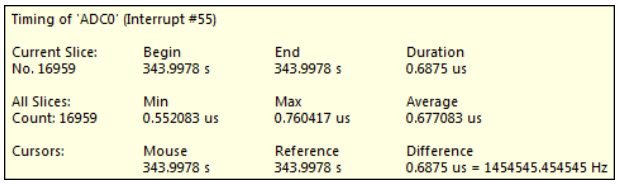


Measuring a Time:

1. Select Cursor and Timing Info.
2. Select Stop Update Screen. The Blinky program continues to run.
3. Find an ADC0 exception and click on it. This anchors it in the window.
4. Select In in the Zoom dialog until ADC0 is about as wide as in the screen below right:
5. Set the cursor by clicking on the left edge of the ADC0 blue block.
6. Hover the cursor on the block and it turns yellow/orange.
7. A box opens displaying various times as shown bottom left:



Current Slice: measures the yellow box time. **Cursors:** measures the cursor positions.



25) Using the Logic Analyzer (LA) with ULINK2/ME, ULINK*plus*, ULINK*pro* or J-Link:

This example will use a ULINK2, ULINK*plus*, ULINK*pro* or a J-Link with the Blinky example. Please connect a your debug adapter to your S32K board and configure it for Serial Wire Viewer (SWV) trace as described previously on pages 19 to 21.

µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using the Serial Wire Viewer as implemented in the S32K. LA uses the same comparators as Watchpoints so all can't be used at same time.

Configure and Use the Logic Analyzer:

1. SWV must be configured as found on pages 18, 19 or 20 for the debug adapter you are using.
2. µVision must be in Debug mode. If **Event Viewer** is open from the previous example, close it.
3. Run the program. **TIP:** You can configure the LA while the program is running or stopped.
4. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.
5. Locate the global variable **delay_val** in Blinky.c near line 34.
6. Right click on **delay_val** and select Add delay_val to… and select Logic Analyzer.

TIP: If an error results when adding counter to the LA, the most probable cause is SWV is not configured correctly.

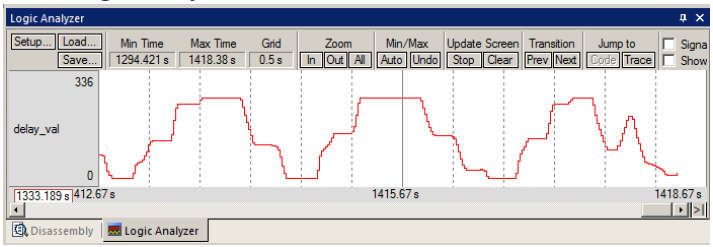
7. In the LA, click on Setup and set Max: in Display Range to 0x150 and Min to 0x0. Click on Close.
8. The LA is now configured to display delay_val in a graphical format.
9. **delay_val** should still be in the Watch and Memory windows. It will be changing as you vary the ADC Pot R186.
10. Adjust the Zoom OUT icon in the LA window to about 0.5 sec or so to display data in the LA as shown below:

TIP: If the LA is blank, exit and reenter Debug mode to refresh the CoreSight comparators.

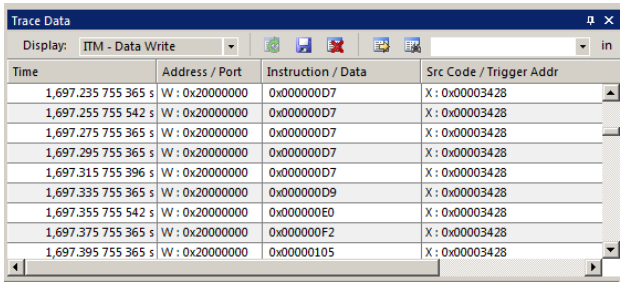
TIP: The Logic Analyzer can display up to four static and global variables, structures and arrays. It can't see locals: just make them static or global. To see Peripheral registers, enter them into the Logic Analyzer and write data to them.

View Data Write of delay_val:

When a variable is added to the Logic Analyzer, the data write frames are sent to the Trace Data or Records window.



1. Select Debug/Debug Settings. Select the Trace tab.
2. Select On Data R/W Sample.
3. Click OK twice. This adds execution addresses to the Src Code/Trigger Addr column.
4. Clear the Trace Data or Trace Records window. Double click for ULINK2 or for ULINK*pro* click:
5. RUN the program. STOP the program.
6. Open the Trace Data or Trace Records window.
7. The window similar to this opens up: This one is for ULINK*pro*. ULINK2 is different and updates while the program is running.
8. In the Display box, select ITM Data Write:
   For ULINK2, right click in the Trace Records window and unselect Exceptions.



9. The first line in ***this*** Trace Data window means:

   The instruction at 0x0000 3428 caused a write of data 0xD7 to address 0x2000 0000 at the listed time in seconds.
10. If using a ULINK*pro*, in the Trace Data window, double click on a data write frame and the instruction causing this write will be highlighted in the Disassembly and the appropriate source window.
11. You might have to turn off ETM Trace Enable if you do not see the correct SWV frames because of overload.

TIP: The Src Code/Trigger Addr column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin if you are not interested in it. With a ULINK2, this column is called PC.

TIP: Raw addresses can also be entered into the Logic Analyzer. An example is: *((unsigned long *)0x20000000)

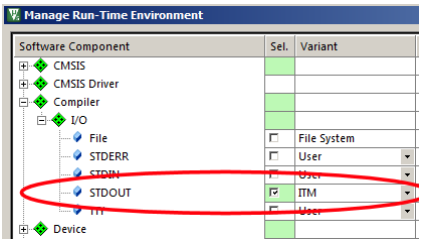
26) *printf* using ITM 0 (Instrumentation Trace Macrocell):

ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in the Debug (printf) Viewer window. It is possible to send ITM data to a file: www.keil.com/appnotes/docs/apnt_240.asp.

1. Stop the program and exit Debug mode.

Add STDOUT File (retarget_io.c):

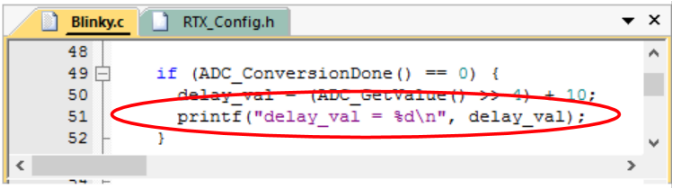
1. Open the Manage Run-Time Environment window (MRTE).
2. Expand Compiler and I/O as shown here:
3. Select STDOUT and ITM. This adds the file retarget_io.c to the project.
4. Ensure all blocks are green and click OK to close the MRTE.



TIP: If you select EVR instead of ITM, printf will not require SWV. It will use DAP and will work using CMSIS-DAP.

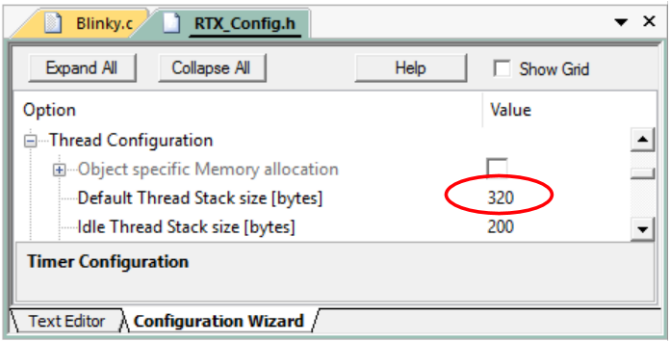
Add printf to Blinky.c:

1. Inside the thread thrADC found starting near line 43, add this line at line 51: printf("delay_val = %d\n", delay_val);



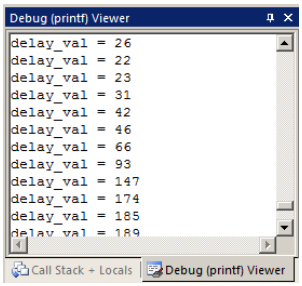
Increase the thread stack size in RTX:

1. Open the file RTX_Config.h.
2. Select Configuration Wizard tab at its bottom.
3. Change Default Thread Stack size: to 320 bytes as shown below right: (360 for EVR)



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 below right:
5. Right click on the Debug window and select Mixed Hex ASCII mode. Note other useful settings that are available.



Obtaining a character typed into the Debug printf Viewer window from your keyboard:

It is possible for your program to input characters from a keyboard with the function ITM_ReceiveChar in core.CM4.h.

This is documented here: www.keil.com/pack/doc/CMSIS/Core/html/group__ITM__Debug__gr.html

A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer.

TIP: ITM_SendChar is a useful function you can also use to send characters out ITM. It is found in core.CM4.h.

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enable only those SWV features that you need. If you need higher performance SWV, a ULINK*pro* using 4 bit Trace Port or a ULINK*pro* or a ULINK*plus* using the SWO pin provides the fastest speed.

27) DSP SINE Example:

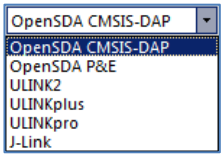
Arm CMSIS-DSP libraries are offered for Arm Cortex-M0, Cortex-M3, Cortex-M4 and Cortex-M7 processors. DSP libraries plus all sources are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS\.
See www.keil.com/cmsis and https://github.com/ARM-software/CMSIS_5.

This example creates a ***sine*** wave, then creates a second to act as ***noise***, which are then added together (***disturbed***), and then the noise is filtered out (***filtered***). The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer.

This example incorporates the Keil RTOS RTX. RTX has a BSD or Apache 2.0 license. All source code is provided.

This program will run with OpenSDA but to see the interesting and useful SWV features you need any ULINK or a J-Link.

1. Get the example from www.keil.com/appnotes/docs/apnt_305.asp. Copy it to C:\00MDK\Boards\NXP\S32K\DSP.
2. Open the project file C:\00MDK\Boards\NXP\S32K\DSP\sine.uvprojx.
3. **To use OpenSDA P&E:** connect a USB cable to USB connector. See page 7 to install P&E on the S32K board. No SWV including the LA will function with OpenSDA. The program must be stopped to update the Watch window.
4. **To use OpenSDA CMSIS-DAP:** connect a USB cable to USB connector. See page 8 to install CMSIS-DAP.
5. **For any ULINK or J-Link:** Connect it to the Cortex Debug ETM connector J10 on the 32K-148 board.
6. Power the board by connecting 12 volts to the J9 barrel connector.
7. Select your debug adapter from the pull-down menu as shown here:



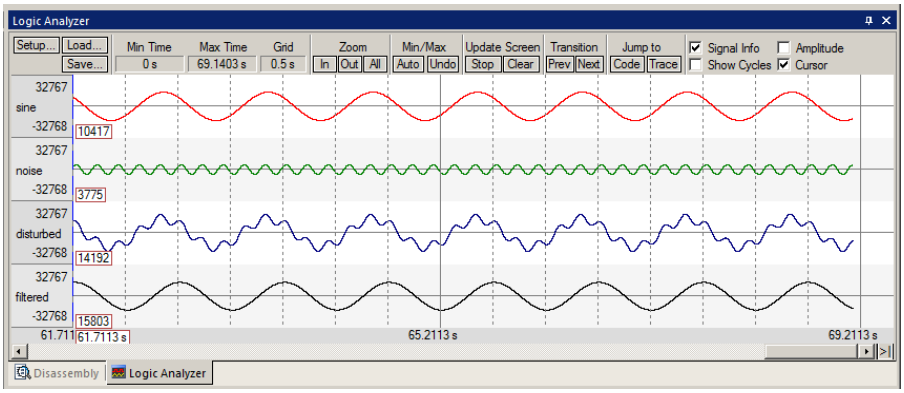
8. Compile the source files by clicking on the Rebuild icon.  .
9. Enter Debug mode by clicking on the Debug icon.  The Flash will be programmed.

TIP: The default Core Clock: is 80 MHz for use by the Trace configuration window under the Trace tab.

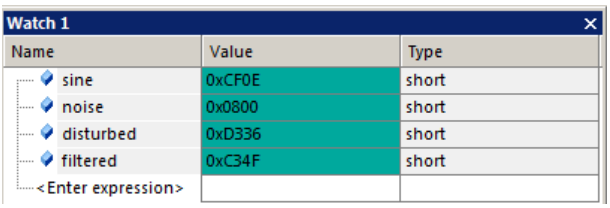
10. Click on the RUN icon.  Open the Logic Analyzer window  and the Watch 1 window: View/Watch/
11. This project has Serial Wire Viewer configured and the Logic Analyzer and Watch 1 loaded with the four variables.
12. If the variables in Watch 1 are changing, the program is running correctly. Stop the program if using P&E.
13. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom for an appropriate display. Displayed are 4 global variables: **`sine`**, **`noise`**, **`disturbed`** and **`filtered`**.

Trouble: If one or two variables display no waveform, disable ITM Stimulus Port 31 in the Trace Config window and/or Exceptions in the Trace Exceptions window. The SWO pin is probably overloaded if you are using a ULINK2. A ULINK*plus* or a ULINK*pro* handles SWV data faster than a ULINK2 or J-Link can. Make sure the Core Clock is set to 80.

14. Select View/Watch Windows and select Watch 1. The four variables are displayed updating as shown below:



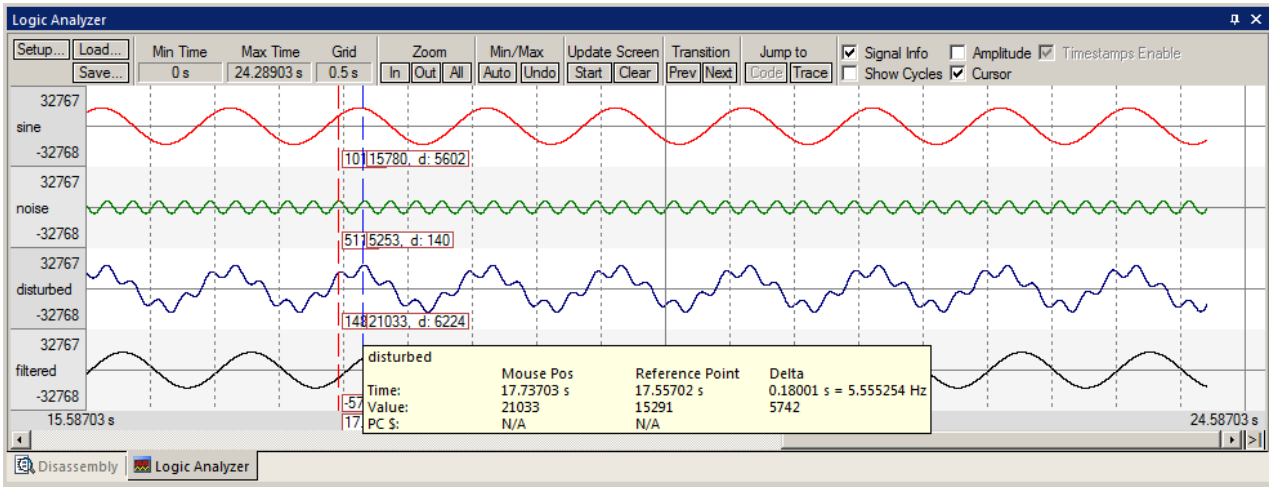
15. Open the Trace Records window and the Data Writes to the four variables are displayed using Serial Wire Viewer. When you enter a variable in the LA, its data write is also displayed in the Trace window. With ULINK*pro* you must stop the program to display the data Trace Data window. J-Link does not display any data read or write frames. OpenSDA P&E has no SWV support.



16. Select View/Serial Windows/Debug (printf) Viewer. ASCII data is displayed from the printf statements in DirtyFilter.c. Not with P&E.
17. Leave the program running.
18. Close the Trace Records window if open.

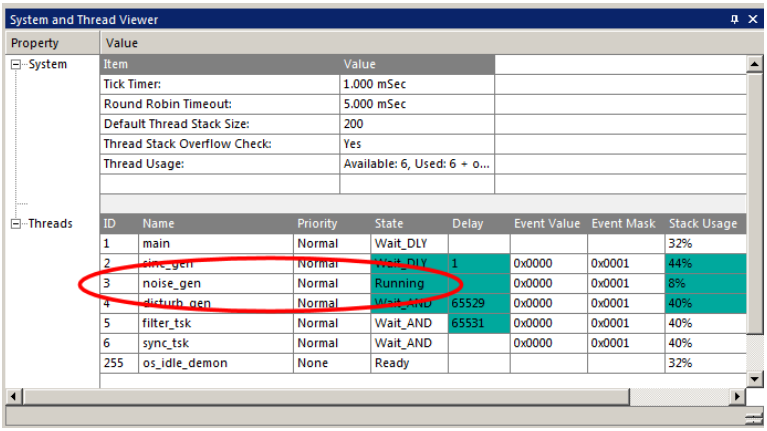
Signal Timings in Logic Analyzer (LA):

1. In the LA window, select Signal Info, Show Cycles, Amplitude and Cursor.
2. Click on STOP in the Update Screen box. You could also stop the program but leave it running in this case.
3. Click somewhere interesting in the LA to set a reference cursor line.
4. Note as you hover the cursor various timing information is displayed as shown below:



RTX System and Threads Viewer: This works with OpenSDA CMSIS-DAP, Keil ULINK and J-Link.

1. Click on Start in the Update Screen box to resume the collection of data. The program must be running.
2. Open Debug/OS Support and select RTX System and Thread Viewer. A window similar to below opens up. You may have to click on its header and drag it into the middle of the screen to comfortably view it.
3. As the various threads switch state this is displayed. Note most of the CPU time is spent in the idle daemon: it shows as Running. The processor spends relatively little time in other tasks. You will see this illustrated clearly on the next page. It is possible to adjust these timings to give more CPU time to various threads as needed.
   NOTE: With OpenSDA in P&E mode, the program must be stopped to update this window.
4. Set a breakpoint in each of the four tasks in DirtyFilter.c by clicking in the left margin on a grey area. Do not select while(1) as this will not stop the program.
5. Click on Run and the program will stop at a thread and the System and Threads Viewer will be updated accordingly. In the screen below, the program stopped in the noise_gen task:
6. Clearly you can see that noise_gen was Running when the breakpoint was activated.
7. Each time you click on RUN, the next task will display as Running.
8. Remove all the breakpoints by clicking on each one. You can use Ctrl-B and select Kill All.
9. Stay in Debug mode for the next page.



TIP: You can set/unset hardware breakpoints while the program is running.

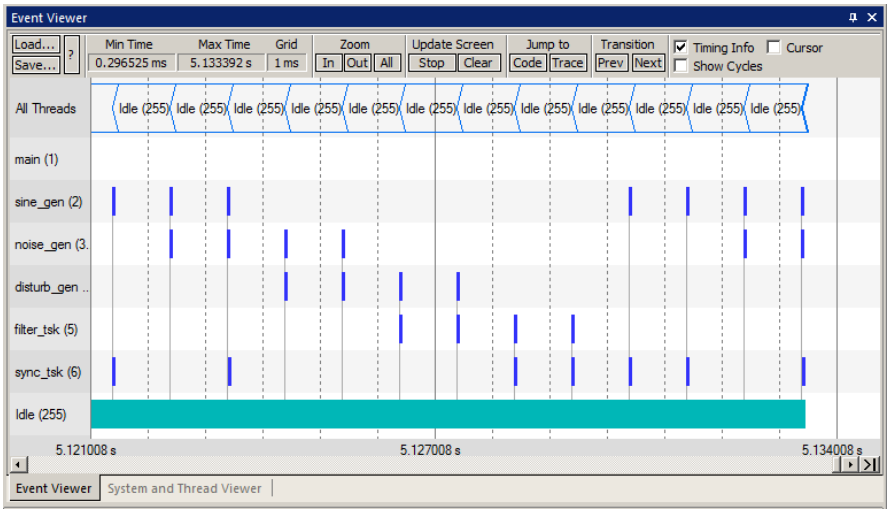
TIP: Recall this window uses CoreSight DAP read and write technology to update this window. Serial Wire Viewer is not used and is not required to be activated for this window to display and be updated.

The DSP example uses RTX4 and not RTX5 as used in the RTX5_Blinky example. This means the System and Thread Viewer has a different format.

The Event Viewer does use SWV and this is demonstrated on the next page.

RTX Event Viewer (EV): ULINK2, ULINK*plus*, ULINK*pro* and J-Link: Not with OpenSDA.

1. ***If you are using a* ULINK*pro*, *skip this step unless you want to see SWV overload.:*** Stop the program. Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables and select Close. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might/will occur.
2. Select Debug/Debug Settings.
3. Click on the Trace tab.
4. Enable ITM Stimulus Port 31. Event Viewer uses this port to collect its information.
5. Click OK.
6. Click on RUN  .
7. Open Debug/OS Support and select Event Viewer. The window here opens up:

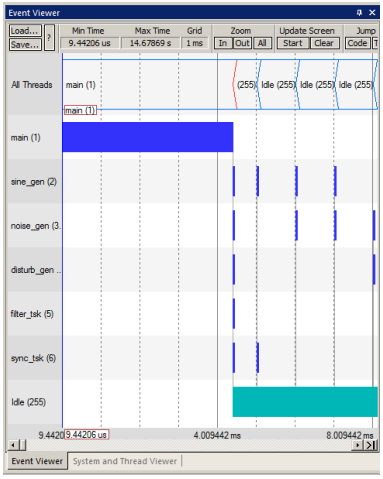


TIP: If Event Viewer is still blank, exit and re-enter Debug mode. 

Main Thread:

1. Select Stop in the Update Screen. Scroll to the beginning of the Event Viewer.
2. The first thread in this program was main() as depicted in the Event Viewer. The main thread is the main() function in DirtyFilter.c It runs some RTX initialization code at the beginning and is stopped with osDelay(osWaitForever);.

TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some SWO or Trace Port bandwidth. Try turning off the exceptions with EXTRC.



3. The 5 running threads plus the idle daemon are displayed on the Y axis. Event Viewer shows which thread is running, when and for how long.
4. Click Stop in the Update Screen box.
5. Click on Zoom In so three or four threads are displayed as shown here:
6. Select Cursor. Position the cursor over one set of bars and click once. A red line is set here:
7. Move your cursor to the next set and total time and difference are displayed.
8. Since you enabled Show Cycles, the total cycles and difference is also shown.

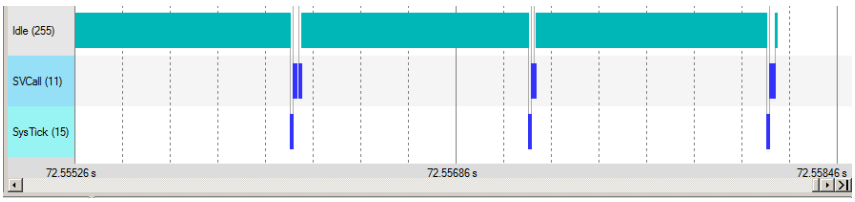
The 1 msec shown is the SysTick timer value which is set in RTX_Conf_CM.c in the OS_CLOCK and OS_TICK variables.

Using a Keil ULINK*pro* to view Interrupt Handler execution times:

SWV Throughput: ULINK*pro* is much better with SWO bandwidth issues. It has been able to display both the EV and LA windows. The ULINK*pro* ETM and ULINK*pro* Manchester modes are preconfigured in Options for Target.

ULINK*pro* can also use the 4 bit Trace Port for even faster operation for SWV. Trace Port use is mandatory for ETM trace. A ULINK*pro* in ETM mode provides program flow debugging, Code Coverage and Performance Analysis. ULINK*pro* also supports ETB (Embedded Trace Buffer) as found in many Kinetis processors.

Exceptions: A ULINK*pro* displays exceptions at the bottom of the Event Viewer. Shown here are the SysTick and SVCall exceptions. You can measure the duration of the time spent in the handlers. Any other exception events such as DMA will also be displayed here.

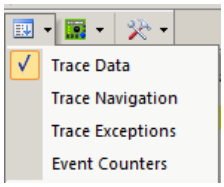


28) Running the ETM example program using the NXP S32K148 board:

Now we will connect the Keil MDK development system using the S32K-148 board. A Keil ULINK*pro* must be used to collect the ETM instruction trace frames. The RTX5_Blinky example will be used as it is already ETM example is preconfigured. The addresses you see might be different than those displayed in the next few pages depending on compiler settings.

1. Connect a Keil ULINK*pro* to J10 Cortex Debug ETM connector as shown here:
2. Power the board with 12 volts to the J9 connector.
3. Start µVision by clicking on its desktop icon.
4. Select Project/Open Project.
5. Go to: C:\00MDK\addon_mdk\Boards\NXP\S32K148-EVB\RTX5_Blinky\
6. Open the project Blinky.uvprojx.
7. Select Flash ETM: ULINK*pro* is preconfigured: S32K148 Flash ETM
8. Compile the source files by clicking on the Rebuild icon.
9. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Progress will be indicated in the Output Window. Select OK if the Evaluation Mode box appears.
10. **DO NOT** click on the RUN icon. The program will automatically run to the start of main() and stop. If you do click on RUN, exit and reenter Debug to restart the program.

Opening the Trace Data Window and Confirming ETM is Working:



1. Open the Trace Data window it by clicking on the small arrow beside the Trace icon as shown:
2. The Trace Data window will look similar to the one below if it is working correctly:
3. If you do see trace frames in this window, you must fix this before you can continue. See page 37 for instructions on how the ETM should be configured.

TIP: You must stop the program to view the trace frames in the Trace Data window.

Trace Data Window Description:

1. The frames above the blue line (in this case) are executed instructions with source if available.
2. The Function column displays the name of its function with the first instance highlighted in orange.
3. The frames below the blue lines are Serial Wire Viewer (SWV). SWV and ETM frames can be more mixed.
4. The last instruction executed is the POP (r4,pc). This is part of RTX5 and indicated in the Function column.

Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
0.000 970 300 s	X : 0x00001ED2	*CBNZ r0,0x00001ED8		EventCheckFilter
	X : 0x00001ED4	MOVS r0,#0x00	return 0U;	EventCheckFilter
0.000 970 400 s	X : 0x00001ED6	BX lr	}	EventCheckFilter
0.000 970 900 s	X : 0x00001F30	*CBNZ r0,0x00001F38		EventRecord2
	X : 0x00001F32	MOVS r0,#0x01	return 1U;	EventRecord2
0.000 971 000 s	X : 0x00001F34	POP {r4-r8,pc}	}	EventRecord2
0.000 971 400 s	X : 0x000023BE	POP {r4,pc}	}	EvrRtxKernelGetState
	X : 0x0000454A	LDR r0,[pc,#4] ; @0x00004550	return ((osKernelState_t)(osRtxInfo...	svcRtxKernelGetState
	X : 0x0000454C	LDRB r0,[r0,#0x08]		svcRtxKernelGetState
0.000 971 800 s	X : 0x0000454E	POP {r4,pc}	}	svcRtxKernelGetState
	X : 0x0000356C	B 0x0000355C		osKernelGetState
0.000 972 600 s	X : 0x0000355C	POP {r4,pc}	}	osKernelGetState
0.001 180 950 s		Exception Entry - SVCall		
0.001 232 200 s		Exception Exit - SVCall		
0.001 235 450 s		Exception Return		
0.001 248 300 s		Exception Entry - SVCall		
0.001 267 550 s		Exception Exit - SVCall		
0.001 296 400 s		Exception Return		

29) Viewing ETM Frames starting at RESET:

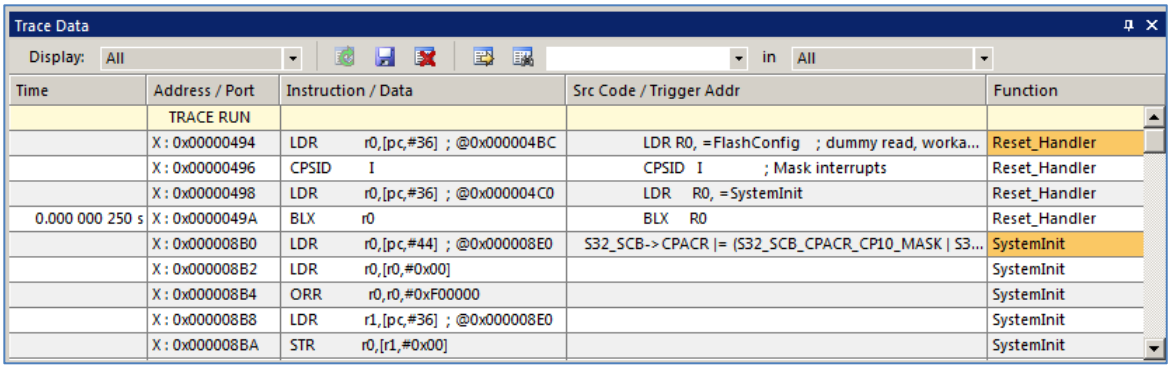
Single-Step:

1. In the Register window, the PC will display the location of the next instruction to be executed (0x34B8 in my case).
2. Click anywhere in the Disassembly window to bring it into focus. This ensures steps are at the instruction level.
3. Click on Single Step once. 
4. The instruction POP at 0x000 34B8 is executed and displayed in the Trace Data window as shown here:

	TRACE RUN			
0.002 428 400 s	X : 0x000034B8	PUSH {r4,lr}	int main (void) {	main

View RESET Sequence:

1. Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is a LDR instruction in the RESET_Handler function as shown below:
2. If you use the Memory window to look at location 0x04, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0000 00494 + 1 = 0x0495 (+1 says it is a Thumb® instruction). These first instructions after RESET are shown below: Note the source information is displayed including which function they belong to. The first occurrence in a function is highlighted in orange to make the start of functions easier to find.
3. Any source code associated with an instruction is displayed in the Src Code column.
4. If you double-click on any line, this will be highlighted in both the Disassembly and the relevant source window.



Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x00000494	LDR r0,[pc,#36] ; @0x000004BC	LDR R0, =FlashConfig ; dummy read, worka...	Reset_Handler
	X : 0x00000496	CPSID I	CPSID I ; Mask interrupts	Reset_Handler
	X : 0x00000498	LDR r0,[pc,#36] ; @0x000004C0	LDR R0, =SystemInit	Reset_Handler
0.000 000 250 s	X : 0x0000049A	BLX r0	BLX R0	Reset_Handler
	X : 0x000008B0	LDR r0,[pc,#44] ; @0x000008E0	S32_SCB->CPACR |= (S32_SCB_CPACR_CP10_MASK | S3...	SystemInit
	X : 0x000008B2	LDR r0,[r0,#0x00]		SystemInit
	X : 0x000008B4	ORR r0,r0,#0xF00000		SystemInit
	X : 0x000008B8	LDR r1,[pc,#36] ; @0x000008E0		SystemInit
	X : 0x000008BA	STR r0,[r1,#0x00]		SystemInit

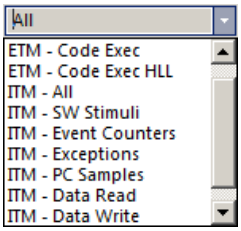
TIP: If you unselect Run to main() in the Debug tab, no instructions will be executed when you enter Debug mode. The PC will equal 0x0494. You can run or single-step from that point and this will be recorded in the Trace Data window.

30) Finding the Trace Frames you are looking for:

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

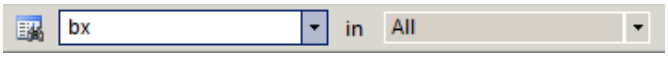
Trace Filters:

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



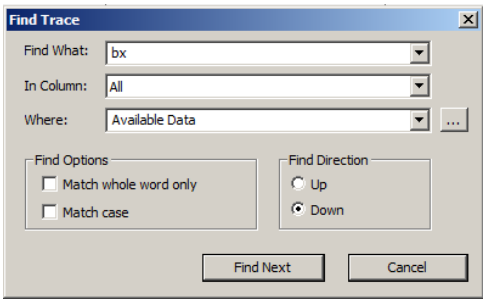
Find a Trace Record:

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



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

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



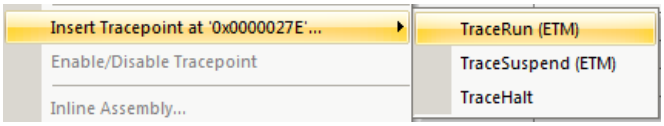
TIP: Or you can press Enter to go to the next occurrence of the search term.

Trace Triggering:

You can configure two sets of trace triggers. µVision has three types of trace trigger commands as listed here: They are set and unset in a source or Disassembly window or in the µVision Command window.

1. **TraceRun:** Starts ETM trace collection when encountered.
2. **TraceSuspend:** Stops ETM trace collection when encountered. TraceRun has to have first been set and encountered to start the trace collection for this trigger to have an effect.
3. **TraceHalt:** Stops ETM trace, SWV and ITM. Trace collection can be resumed only with a STOP/RUN sequence. TraceStart will not restart it.

They are selected from the menu shown here:
This menu is accessed by right-clicking on a valid assembly instruction in the Disassemble window or a C source line.



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

How it works:

When a TraceRun is encountered on an instruction while the program is running, ULINK*pro* will start collecting trace records. When a TraceSuspend is encountered, trace records collection will stop there. ***EVERYTHING*** in between these two times will be collected. This includes all instructions through any branches, exceptions and interrupts.

Sometimes there is some skid past the trigger point which is normal.

Trace Commands: *This part is important in order to effectively manage Tracepoints.*

There are a series of Trace Commands you can enter in the Command window while in Debug mode.

See www.keil.com/support/man/docs/uv4/uv4_debug_commands.htm

TL - Trace List: list all tracepoints.

TK - Trace Kill: tk* kills all tracepoints or tk *number* only a specified one i.e. tk 2.

TIP: TK* is very useful for deleting tracepoints when you can't find them in the source or Disassembly windows.

TL is useful for finding any tracepoints set. Results are displayed in the Command window.

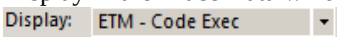
31) Setting Trace Triggers:

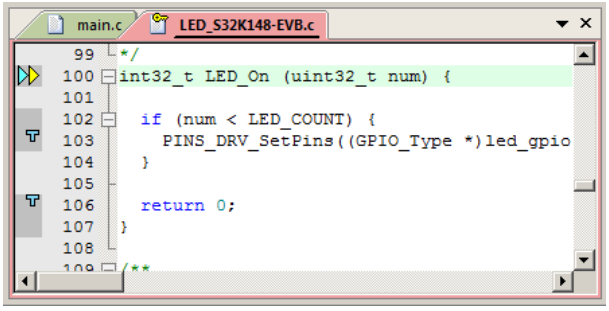
Setup the ETM Example program:

1. Stop the program  and stay in Debug mode.
2. In Blinky.c, set a breakpoint near line 65 LED_On(led_num); A breakpoint is selected by clicking on the grey or colour box to the left of the C or assembly line. A red circle will be displayed at the breakpoint.
3. Click on the RUN icon.  The program will run to the breakpoint and stop.
4. Remove the breakpoint by clicking on the red circle.
5. Click on Single Step  once to enter the function LED_On as shown below:

Setup the Trace Triggers:

6. In the file LED_S32K148-EVB.c, right click on the grey (or green) block opposite near line 103 as shown here:
7. Select Insert Tracepoint at or near line 103 and select **TraceRun (ETM).** A cyan **T** will appear as shown below:
8. Right-click on the gray (or green) block opposite line 106 (return 0;) as shown here:
9. Select Insert Tracepoint at line 106 and select **TraceSuspend (ETM).** A cyan **T** will appear as shown:
10. Clear the Trace Data window  for convenience. This is an optional step.
11. Click RUN and after a few seconds click STOP.
12. Filter exceptions out by selecting ETM – Code Exec in the Display in the Trace Data window:





13. Examine the Trace Data window as shown below:

You can see below where the trace started on 0x2A20 and stopped on 0x2A38: All other frames are discarded.

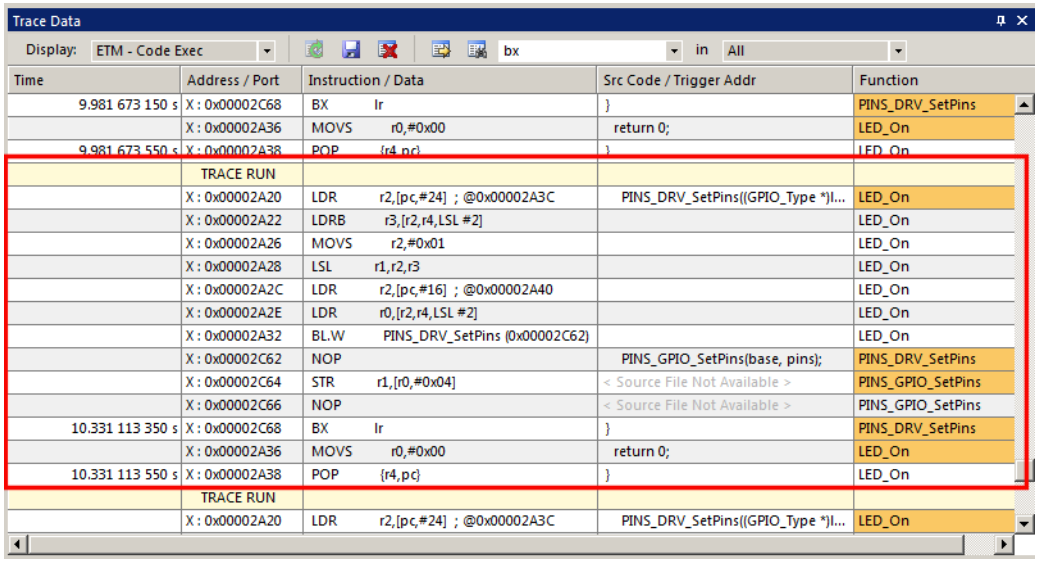
14. In the Command window, enter TL and press Enter. The two Trace points are displayed.
15. **Enter TK* in the Command window and press Enter to delete all Tracepoints.**

Trace Skid:

The trace triggers use the same CoreSight hardware as the Watchpoints. This means that it is possible a program counter skid might happen. The program might not start or stop on the exact location you set the trigger to.

You might have to adjust the trigger point location to minimize this effect.

This is because of the nature of the comparators in the CoreSight module and it is normal behavior.

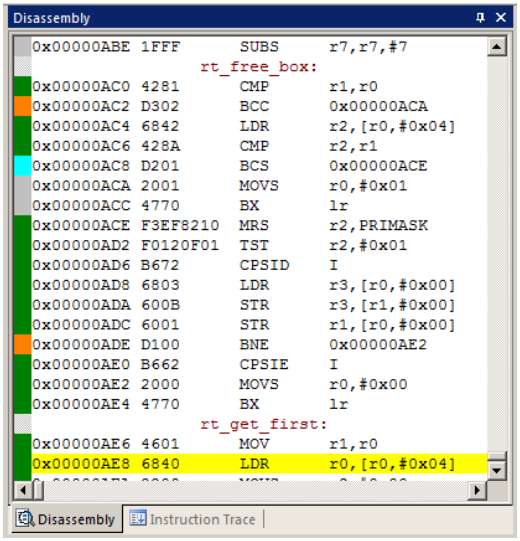


32) Code Coverage:

1. In Blinky.c, set a breakpoint near line 65 LED_On(led_num);
2. Click on the RUN icon.  The program will stop on this breakpoint. Remove the breakpoint.
3. Examine the Disassembly and Blinky.c windows. Scroll and notice the different color blocks in the left margin:
4. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not.

Colour blocks indicate which assembly instructions have been executed.

1. Green: This assembly instruction was executed.
2. Gray: This assembly instruction was not executed.
3. Orange: A Branch is never taken.
4. Cyan: A Branch is always taken.
5. Light Gray: There is no assembly instruction here.
6. RED: A Breakpoint is set here.
7. This points to the next instruction to be executed.



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

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

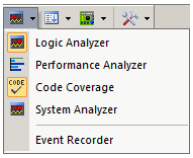
In the window above, why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

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

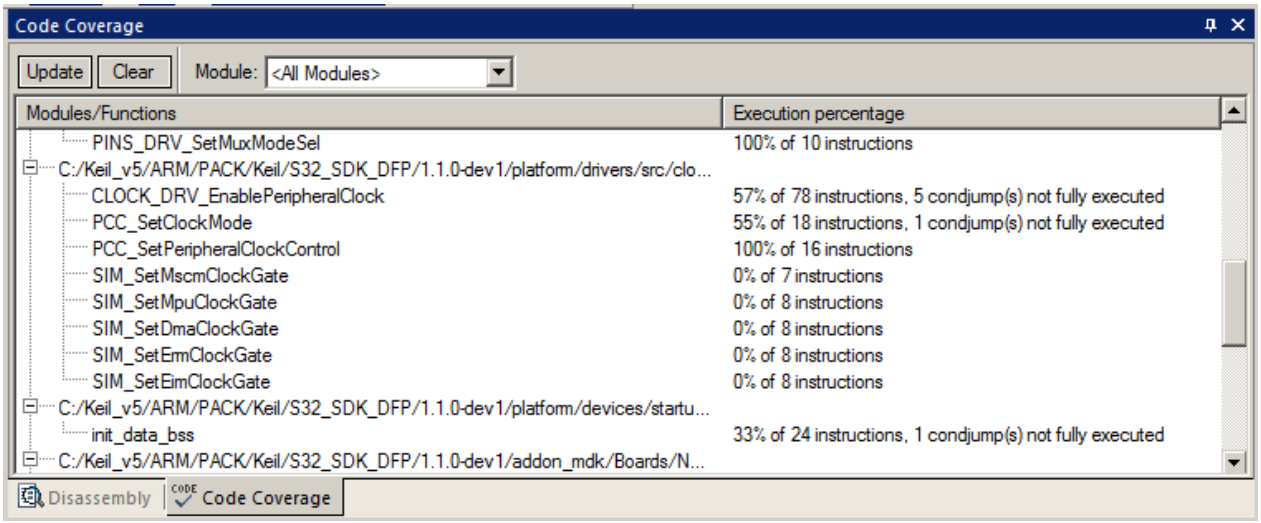
Good programming practice requires that these unexecuted instructions be identified and tested.

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

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage or by selecting the Analysis icon as shown here:



The next page describes how you can save Code Coverage information to a file.



Saving Code Coverage: See www.keil.com/support/man/docs/uv4/uv4_cm_coverage.htm

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

It is possible to save this information in an ASCII file for use in other programs. gcov is supported.

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

You can Save Code Coverage in two formats:

1. In a .gcov file: Use the command COVERAGE GCOV *module or* COVERAGE GCOV *.
2. In an ASCII file. You can either copy and paste from the Command window or use the log command:
   1) log > c:\cc\test.txt ; send CC data to this file. The specified directory must exist.
   2) coverage asm ; provides the data for log. you can also specify a module or function.
   3) log off ; turn the log function off.

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

```
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/system_S32K148.c\SystemCoreClockUpdate - 31% (18 of 58 instructions
executed)
    2 condjump(s) or IT-block(s) not fully executed
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/system_S32K148.c\SystemInit - 100% (20 of 20 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\Reset_Handler - 66% (6 of 9 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\NMI_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\HardFault_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\MemManage_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\BusFault_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\UsageFault_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\SVC_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\DebugMon_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\PendSV_Handler - 0% (0 of 1 instructions executed)
\\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\SysTick_Handler - 0% (0 of 1 instructions executed)
```

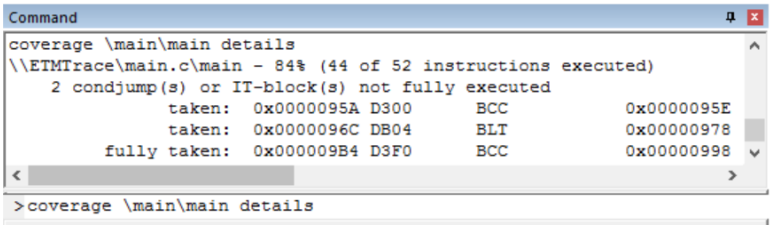
2) The command **`coverage asm`** produces this listing (partial is shown):

```
 \\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\Reset_Handler - 66% (6 of 9 instructions executed)
  EX |  0x00000494 Reset_Handler:
  EX |  0x00000494 4809      LDR           r0,[pc,#36]  ; @0x000004BC
  EX |  0x00000496 B672      CPSID         I
  EX |  0x00000498 4809      LDR           r0,[pc,#36]  ; @0x000004C0
  EX |  0x0000049C 4809      LDR           r0,[pc,#36]  ; @0x000004C4
  EX |  0x0000049E 4780      BLX           r0
  NE |  0x000004A0 B662      CPSIE         I
  NE |  0x000004A2 4809      LDR           r0,[pc,#36]  ; @0x000004C8
  NE |  0x000004A4 4700      BX            r0
 \\ETMTrace\RTE/Device/S32K148UAxxxLQx/startup_S32K148.s\NMI_Handler - 0% (0 of 1 instructions executed)
```

The first column above describes the execution as follows:

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

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



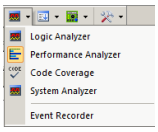
If the log command is run, this will be saved/appended to the specified file.

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

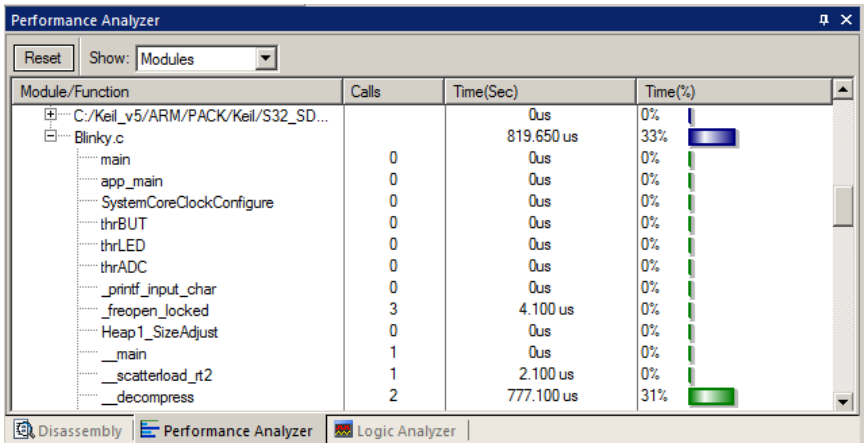
33) Performance Analysis (PA):

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

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



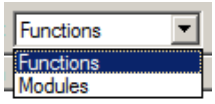
1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis or select Performance Analyzer from here:
3. A window similar to the one below will open up.
4. Exit Debug mode and re-enter it. This clears the PA window and resets the S32K processor and reruns to main(). Do ***not*** click on RUN yet Expand some of the module names as shown below.
5. Shown is the number of calls and percentage of total time in this short run from RESET to the beginning of main().



6. Click on the RUN icon. See the PA window below:
7. Note the display changes in real-time while the program is running. There is no need to stop the processor to collect the data. No code stubs are needed in your source files.

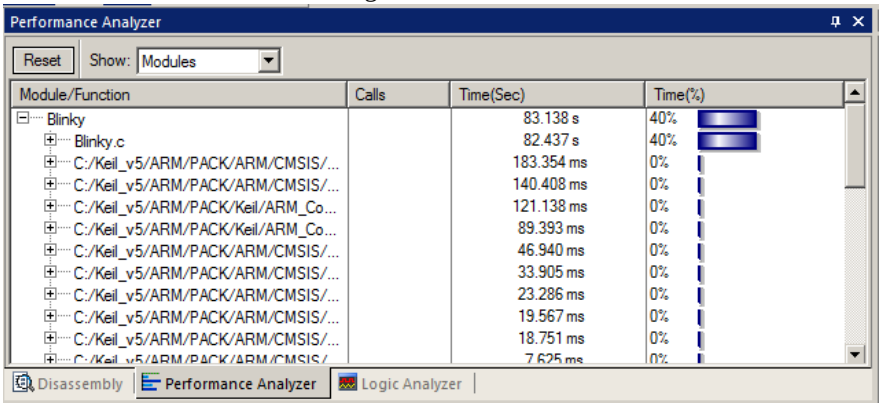
TIP: Double click on any Function or Module name and this will be highlighted in the Disassembly or source windows.

8. Select Functions from the pull down box as shown here and notice the difference.
9. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.



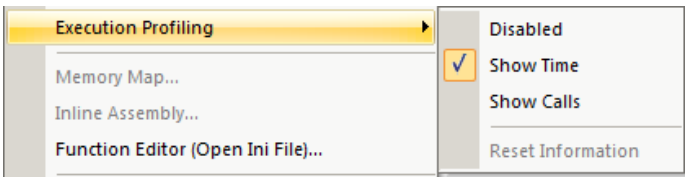
TIP: You can also click on the RESET icon RST but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

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



34) Execution Profiling:

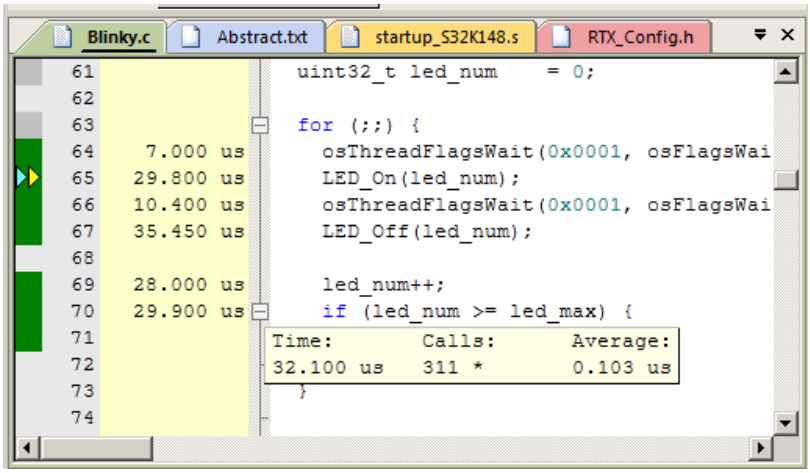
Execution profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.



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

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

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

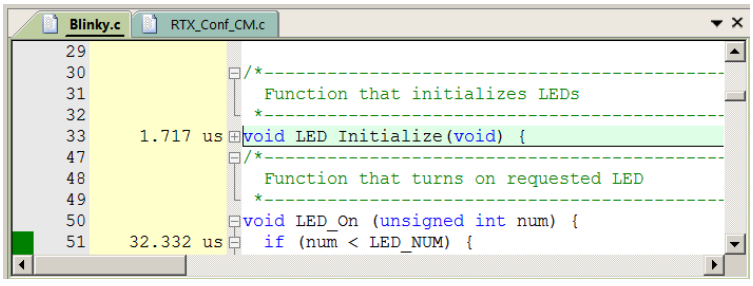


-

Outlining:

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

1) Click in the square near the while(1) loop near line 33 as shown here:
2) The C source in the function LED_Initialize is now collapsed into one line.
3) The times are added together to 1.717 usec in this case:
4) This can be useful to hide sections of code to simplify the window you are reading.
5) Click on the + to expand it.
6) Stop the program
7) Exit Debug mode.

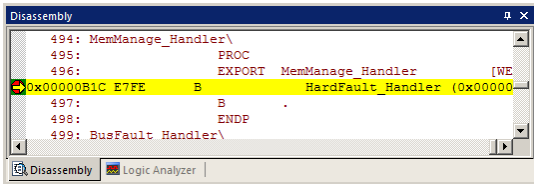


For more information see: www.keil.com/support/man/docs/uv4/uv4_ui_outline.htm

35) In-the-Weeds Example: (your addresses might not be the same as those shown below)

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

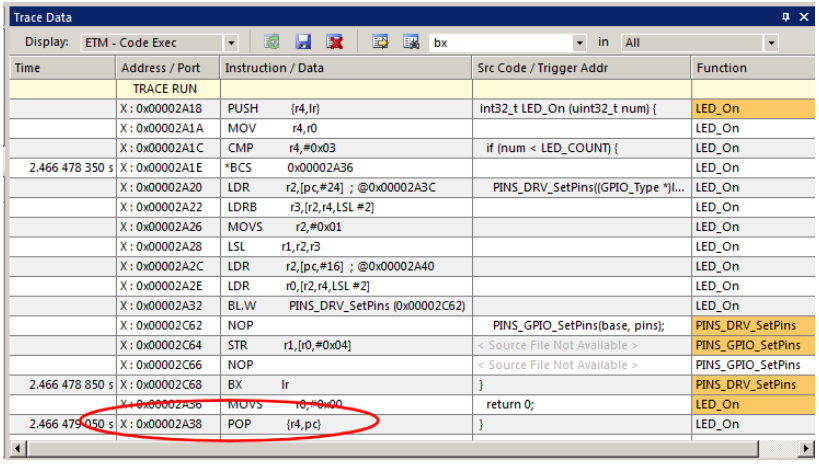
If a Hard Fault occurs in our example, the CPU will end up at 0x0B1C as shown in the Disassembly window below. This is the Hard Fault handler. This is a branch to itself and will run this instruction forever. The trace buffer will save millions of the same branch instructions. This is not very useful. The addresses used might be different in your project.



The Hard Fault handler exception vector is found in the file startup_S32K148.s. If we set a breakpoint by clicking on the Hard Fault Handler and run the program: at the next Hard Fault event the CPU will jump to the HardFault_Handler and halt processing.

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

1. Use the ETM example from the previous exercise, enter Debug mode.
2. Locate the Hard Fault near address 0x00B1C in the Disassembly window or near line 492 in startup_S32K148.s.
3. Set a breakpoint at this point. A red circle will appear.
4. In the Command window enter: **g, LED_On** and press ENTER. This will put the PC at the start of this function. LED_On returns with a POP instruction; near address 0x2A38 which we will use to create a Hard Fault with lr = 0.
5. The assembly and sources in the Disassembly window do not always match up and this is caused by anomalies in ELF/DWARF specification. In general, scroll downwards in this window to provide the best match.
6. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
7. In the Register window, double-click on the R14 (LR) register and set it to zero. This will cause a Hard Fault when the processor places lr = 0 into the PC and tries to execute the non-existent instruction at memory location 0x00.
8. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
9. In the Trace Data window, you will find the LED_On function with the POP instruction at the end. When the function tried to return, the bogus value of lr caused a Hard Fault.
10. The B instruction at the Hard Fault vector was not executed because ARMCoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.
11. Click on Single Step. You will now see the Hard Fault branch as shown in the bottom screen:

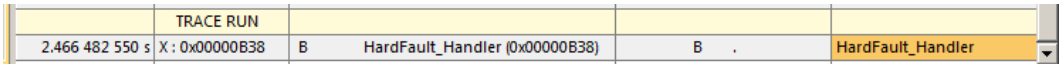


This example clearly shows how quickly ETM trace can help debug program flow bugs.

Exception Entry: Note the Exception Entry at the bottom of the Trace data. This entry is part of SWV. The timestamps of ETM and SWV are not the same. They cannot be correlated together.

TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 492… and select TraceHalt. When Hard Fault is reached, trace collection will halt but the program will keep executing for testing and Hard fault handlers.

12. Exit Debug mode.

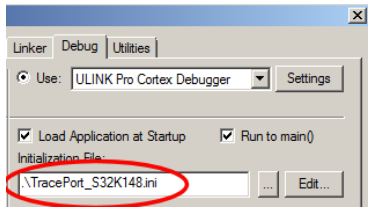


36) Configuring ETM Trace with ULINK*pro*:

The ETM Example program comes configured for ETM operation. These instructions are provided for reference. You can use these instructions to confirm proper settings if you are having trouble getting ETM to operate.

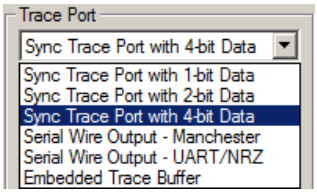
Select ULINK*pro* and enter the TracePort_S32K144.ini file:

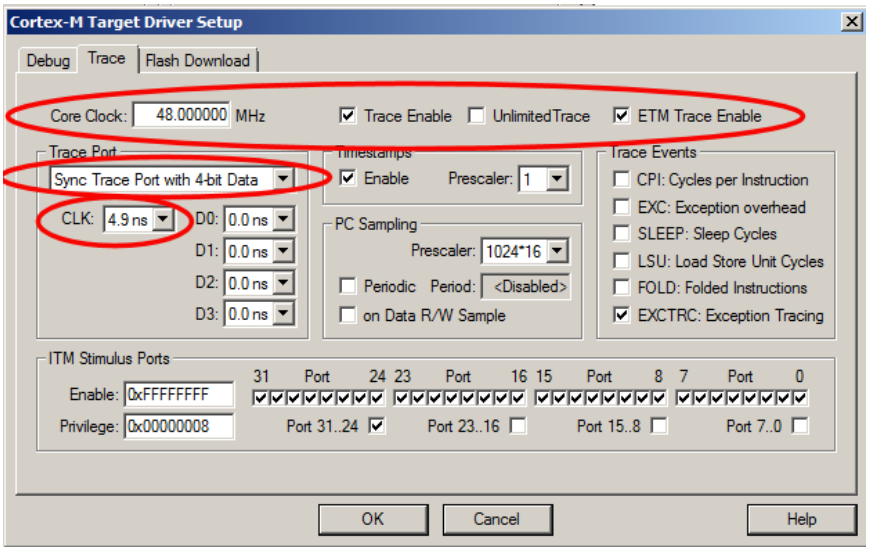
1. Select Options for Target or ALT-F7. Click on the Debug tab to select a debug adapter.
2. Select ULINK Pro Cortex Debugger as shown below:
3. Click on the browse icon and navigate to C:\00MDK\Boards\NXP\S32K\ETMTrace
4. Click on TracePort_S32K148.ini. It will be entered in the Initialization box:
5. If you click on the Edit icon, TracePort_S32K148.ini will be opened with the other source files. You can then view it. Do not modify it at this time.
6. Note you can select which pins ETM is output on.



Configuring ETM Trace (and SWV) using the 4 bit Trace port:

1. In the Options for Target window under the Debug tab, click on Settings:
2. Click on the Trace tab. The window at the bottom of this page is displayed.
3. Select Trace Enable. This selects SWV data trace.
4. Set Core Clock: to 96 MHz. ULINK*pro* uses this to calculate timings in various windows. This is the CPU speed for this project. The S32K-148 can use various clocks for ETM. They can be set in the file TracePort_S32K148.ini with the Configuration Wizard tab.
5. In Trace Port, select Sync Trace Port with 4-bit data. It is best to use the widest size which is 4 bits. It might be useful to use 2 bit trace if you need a pin for other uses.
6. Set Trace Port CLK: to 4.9 ns clock delay. D0 through D3 should be set to 0.0 ns as shown below but : D0 – D3 might be set to 0.9 ns on some boards. **Note:** not all NXP processors need these delay times.
7. Select ETM Trace Enable. This activates ETM Trace.
8. The last 1 million trace frames will be saved with the older ones being overwritten. If you need to save them all, select Unlimited Trace. Your hard drive space will determine how many are saved. ULINK*pro* is a streaming trace.
9. Everything else in this window relates to Serial Wire Viewer. Leave everything else at default as shown below. Only ITM 31 and 0 are used by µVision at this time for the RTOS Event Viewer and printf respectively.
10. Click on OK twice to return to the main µVision menu.
11. ETM and SWV are now configured through the 4 bit Trace Port.
12. Select File/Save All.





37) Serial Wire Viewer and ETM Trace Summary:

We have several debug systems implemented in Kinetis Cortex-M4 devices:

1. SWV and ITM data output on the SWO pin located on the JTAG/SWD 10 pin CoreSight debug connector. The 20 pin connector adds ETM trace.
2. ITM is a printf type viewer. ASCII characters are displayed in the Debug printf Viewer in µVision.
3. Non-intrusive Memory Reads and Writes in/out the JTAG/SWD ports (DAP).
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.
5. ETM provides a record of all instructions executed. ETM also provides Code Coverage and Performance Analysis.

These features are completely controlled through µVision via a ULINK2 or ULINK*pro*.

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

SWV Trace adds significant power to debugging efforts. Problems which may take hours, days or even weeks in big projects can often be found in a fraction of these times with a trace. Especially useful is where the bug occurs a long time before the consequences are seen or where the state of the system disappears with a change in scope(s) or RTOS thread switches.

Usually, these techniques allow finding bugs without stopping the program. These are the types of problems that can be found with a quality trace: Some of these items need ETM trace.

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes). How I did I get to this Fault vector ?
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs). ***How did I get here ?***
4) A corrupted stack.
5) Out of bounds data. Uninitialized variables and arrays.
6) Stack overflows. What causes the stack to grow bigger than it should ?
7) **Runaway programs:** your program has gone off into the weeds and you need to know what instruction caused this. ***This is probably the most important use of trace.***
8) Communication protocol and timing issues. System timing problems.
9) Trace adds significant power to debugging efforts. Tells you where the program has been.
10) Weeks or months can be replaced by minutes.
11) Especially where the bug occurs a long time before any consequences are seen.
12) Or where the state of the system disappears with a change in scope(s).
13) Plus - don't have to stop the program to test conditions. Crucial to some applications.
14) A recorded history of the program execution ***in the order it happened.*** Source and Disassembly is as it was written.
15) Trace can often find nasty problems very quickly.
16) Profile Analysis and Code Coverage is provided. Available only with ETM trace.

What kind of data can the Serial Wire Viewer display ?

1. Global variables.
2. Static variables.
3. Structures.
4. Can see Peripheral registers – just read or write to them. The same is true for memory locations.
5. Can see executed instructions. SWV only samples them. Use ETM to capture all instructions executed.
6. CPU counters. Folded instructions, extra cycles and interrupt overhead.

What Kind of Data the Serial Wire Viewer can't display…

1. Can't see local variables. (just make them global or static).
2. Can't see register operations. PC Samples records some of the instructions but not the data values.
3. SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU. SWV and ETM can only see CPU actions. If using a ULINK*pro* and RTX, DMA exceptions will display in the Event Viewer.

38) 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 Keil.com: www.keil.com/books/armbooks.asp
3. and on Arm.com: www.arm.com/resources/education/textbooks
4. µVision contains a window titled Books. Many documents including data sheets are located there.
5. A list of Arm CPUs is located at: https://www.arm.com/products/silicon-ip-cpu
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

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 using Keil MCB1700: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with µVision https://launchpad.net/gcc-arm-embedded
8. GCC Arm Developer: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads
9. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
10. Cortex-M Processors for Beginners: http://community.arm.com/docs/DOC-8587
11. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
12. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
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 www.keil.com/appnotes/files/apnt_291.pdf
17. ***NEW!*** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp

Keil Tutorials for NXP Boards: www.keil.com/NXP

1. KL25Z Freedom www.keil.com/appnotes/docs/apnt_232.asp
2. K20D50M Freedom Board www.keil.com/appnotes/docs/apnt_243.asp
3. Kinetis K60N512 Tower www.keil.com/appnotes/docs/apnt_239.asp
4. Kinetis K60D100M Tower www.keil.com/appnotes/docs/apnt_249.asp
5. Kinetis FRDM-K64F Freedom www.keil.com/appnotes/docs/apnt_287.asp
6. Kinetis K64F120M Tower www.keil.com/appnotes/docs/apnt_288.asp
7. S32K-144 EVB Tutorial: www.keil.com/appnotes/docs/apnt_299.asp
8. NXP Cookbook Example Ports for MDK: www.keil.com/appnotes/docs/apnt_304.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

39) Keil Products and contact information: See www.keil.com/NXP

Keil Microcontroller Development Kit (MDK-ARM™) for NXP 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* **ULINK*plus***- Cortex-M*x* High performance SWV & power measurement.
- **ULINK*pro*** - Cortex-M*x* SWV & ETM instruction trace. Code Coverage and Performance Analysis.
- **ULINK*pro* D** - Cortex-M*x* SWV no ETM trace ULINK*pro* also works with Arm DS-5.

You can use OpenSDA on the S32K board. For Serial Wire Viewer (SWV), a ULINK2, ULINK-ME or a J-Link is needed. For ETM support, a ULINK*pro* is needed. OS-JTAG or OpenSDA do not support either SWV or ETM.

Call Keil Sales for more details on current pricing. 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. It has a BSD or Apache 2.0 license.
- 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.

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

For Linux, Android, bare metal (no OS) and other OS support on NXP i.MX and Vybrid series processors please see DS-5 and DS-MDK at www.arm.com/ds5/.

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/

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™ Intelligent Processors by ARM®

arm KEIL