

Introduction:

The purpose of this lab is to introduce you to the debugging features of the Nordic nrf52 Cortex®-M4 processor evaluation board 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/Nordic. Download the free **Getting Started MDK 5**: www.keil.com/gsg.

Keil MDK supports and has examples for many Nordic Arm processors. Check the Keil Device Database® on www.keil.com/dd2. This list is also provided by the µVision Pack Installer utility.

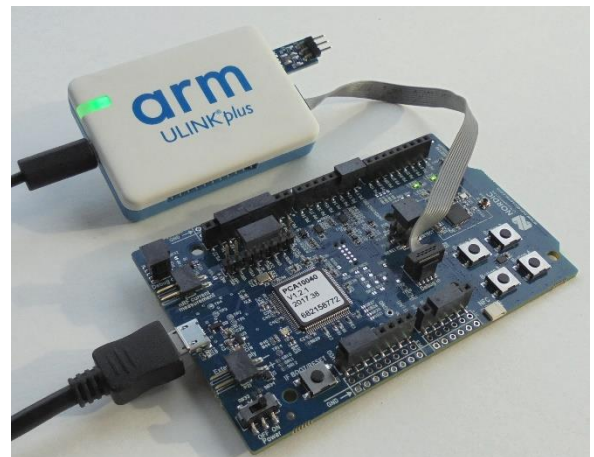
Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into one of the commercial versions.

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 Nordic 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 Nordic driver support is included.
2. Arm Compiler 5 and Arm Compiler 6 (LLVM) are included. GCC is supported. See Developer.arm.com
3. Dynamic Syntax checking on C/C++ source lines.
4. **Compiler Safety Certification Kit:** www.keil.com/safety/
5. TÜV certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
6. MISRA C/C++ support using PC-Lint. www.gimpel.com
7. **Keil Middleware:** Network, USB, Flash File and Graphics.
8. CoreSight™ Serial Wire Viewer (SWV). ETM instruction trace provides program flow debugging, Code Coverage and Performance Analysis.
9. Adapters: ULINK™2, ULINK-ME, ULINKplus, ULINKpro and J-Link. CMSIS-DAP is supported.
10. **NEW ! ULINKplus** power analysis: www.keil.com/mdk5/ulink/ulinkplus/
11. Affordable perpetual and term licensing with support. Contact Keil sales for pricing options. Inside-Sales@arm.com
12. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
13. **NEW ! Event Recorder:** Instrument your code. www.keil.com/pack/doc/compiler/EventRecorder/html/cv_use.html



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. **ETM Instruction Trace:** including Code Coverage and Performance Analysis. Uses a Keil ULINKpro.
3. 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.
4. Six Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
5. RTX tickless RTOS.
6. RTX and RTX Threads windows: kernel awareness for RTX that updates while your program is running.
7. printf using Serial Wire Viewer (SWV) ITM. No UART is required. Can also use Event Recorder.

General Information:

1. NordicSemiconductor Evaluation Boards & Keil Evaluation Software:	3
2. MDK 5 Keil Software Information:	3
3. Debug Adapters Supported:	3
4. CoreSight Definitions:	4

Keil Software and Software Packs:

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

Blinky Example using Nordic nrF52 Demonstrating Debug Features:

8. <i>Blinky</i> example using the Nordic nrF52 Evaluation Board:	8
9. Hardware Breakpoints and Single Stepping:	9
10. Call Stack & Locals window:	10
11. Watch and Memory windows and how to use them:	11
12. Peripheral System Viewer (SV):	12
13. Watchpoints: Conditional Breakpoints:	13

Serial Wire Viewer (SWV):

14. Serial Wire Viewer (SWV)	14
15. Displaying Variables Using the Logic Analyzer (LA):	15
16. Displaying Exceptions (including Interrupts) with SWV:	16
17. printf using ITM:	17
18. RTX System and Threads Viewer:	18
19. Event Viewer for RTX:	19

ETM Instruction Trace using Keil ULINKpro and Blinky:

20. Configuring the ETM example program using the Nordic nrF52 board:	20
1. Connecting a ULINKpro to the Nordic nrF52 board:	20
2. Configure ETM Trace in μ Vision:	20
3. Compile, RUN and View the ETM Instruction Trace:	21
21. Finding the Trace Frames you are looking for:	22
22. Trace Triggers:	23
23. Setting Trace Triggers: Capture only the frames you want:	24
24. Code Coverage:	25
25. Performance Analysis (PA):	27
26. Execution Profiling:	28
27. In-the-Weeds Example: <i>A most important feature of ETM Trace:</i>	29
28. Serial Wire Viewer and ETM Trace Summary:	30

Other Useful Information:

36. Document Resources:	31
37. Keil Products and contact information:	32

1) Nordic Evaluation Boards & Keil Evaluation Software:

Keil MDK provides board support for the Nordic Semiconductor nrF51 Cortex-M0 and nrF52 Cortex-M4 series. See www.keil.com/dd2 under Nordic Semiconductors for the complete list of devices. www.keil.com/Nordic

This document uses the PCA10040 nrF52 Development Kit. This note can be adapted to other boards.

The on-board J-Link debug adapter will be used for most exercises except for ETM instruction trace. For power measurement, use the new ULINK_{plus}. www.keil.com/ULINKplus

On the second last page of this document is an extensive list of resources that will help you successfully create your projects.

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

2) MDK 5 Keil Software Information: *This document uses MDK 5.25 or later.*

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

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

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

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 Blinky example project contained in the nrF52 Software Pack.

3) Debug Adapters Supported:

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

1. **CMSIS-DAP:** An extra processor on your board becomes a debug adapter compliant to CMSIS-DAP. You can add CMSIS-DAP to a custom board. See https://github.com/ARM-software/CMSIS_5. ULINK2, ULINK-ME and ULINK_{plus} are CMSIS-DAP compatible.
2. **ULINK2 and ULINK-ME:** ULINK-ME is only offered as part of certain evaluation board packages. ULINK2 can be purchased separately. These are electrically the same and both support Serial Wire Viewer (SWV), Run-time memory reads and writes for the Watch and Memory windows and hardware breakpoint set/unset on-the-fly.
3. **ULINK_{pro}:** ULINK_{pro} supports all SWV features and adds ETM Instruction Trace. ETM records all executed instructions and provides powerful program flow debugging. ETM provides Code Coverage, Execution Profiling and Performance Analysis features. ULINK_{pro} also provides the fastest Flash programming times.
4. **NEW ! ULINK_{plus}:** High SWV performance plus Power Measurement. See www.keil.com/ulink/ulinkplus/ for details.
5. **Segger J-Link:** The PCA10040 has an on-board J-Link. SWV data reads and writes are not currently supported with a J-Link. You do not need an external debugger such as a ULINK2 to do this lab. For faster SWV data trace, use a suitable Keil ULINK such as ULINK_{pro} or a ULINK_{plus}.

Debug Connections:


- 1) The internal Segger J-Link is connected to your PC via the USB J2 connector.
- 2) An external debug adapter must be connected to the P18 Debug In 10 pin connector.
- 3) For ETM instruction trace, a Keil ULINK_{pro} or a Segger J-Trace must be used. A custom adapter is needed to provide the 4 ETM signals plus the trace clock to the 20 pin CoreSight ETM connection.

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 μ Vision Cortex-M Target Driver Setup. The SWJ box must be selected in ULINK2/ME or ULINK pro . Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDO shares the same pin as SWO. The SWV data normally comes out the SWO pin or Trace Port.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **DAP:** Debug Access Port. This is a component of the Arm CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. μ Vision uses the DAP to update Memory, Watch, Peripheral and RTOS kernel awareness windows while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed. You do not need to configure or activate DAP. μ Vision configures DAP when you select a function that uses it. Do not confuse this with CMSIS_DAP which is an Arm on-board debug adapter standard.
5. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
6. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
7. **Trace Port:** A 4 bit port that ULINK pro uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by μ Vision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. μ Vision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK pro provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis. ETM is output on the Trace Port or accessible in the ETB (ETB has no Code Coverage or Performance Analysis).
10. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK pro . ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal user RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. Cortex-M3/M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 usually have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs. The CPU is halted before the instruction is executed.
13. **Watchpoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. There also referred to as Access Breaks in Keil documentation.

Read-Only μ Vision Source Files:

Some source files in the Project window will have a yellow key on them:  This means they are read-only. This is to help unintentional changes to these files. This can cause difficult to solve problems. These files normally need no modification. μ Vision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

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

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

5) Keil MDK Software Download and Installation:

1. Download MDK 5.25 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\nRF52 for the examples.
4. You do not need a debug adapter for the basic exercises: just the nRF52 board, a USB cable and MDK installed.
5. For the exercises using ETM trace, you need a ULINKpro and a suitable adapter for the 5 ETM signals.
6. You do not need a Keil MDK license for this tutorial. All examples will compile within the 32 K limit.

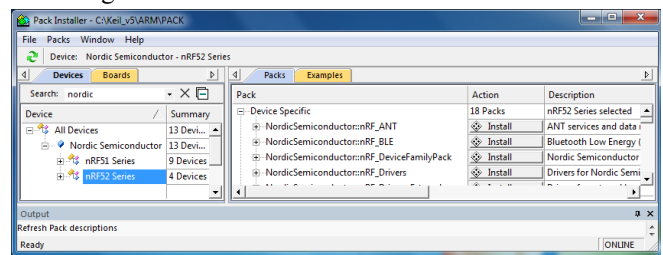
Download MDK-Core Version 5


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. There are several Packs needed for this tutorial. We will download the basic ones and let µVision choose the rest.

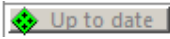
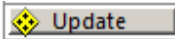
1) Start µVision and open Pack Installer:

1. Connect your computer to the internet. This is needed to download the Software Packs. Start µVision:
2. The very first time you start µVision, the Pack Installer utility will open automatically and download the Pack list. After the first time, you can always start Pack Installer by clicking on its icon from within µVision.
3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen may open. Read and close it.
4. This window opens up: Note “ONLINE” is displayed at the bottom right. If “OFFLINE” is displayed, connect to the Internet before continuing.
5. The current Pack list will be downloaded. This might take a few minutes.
6. Select the Devices tab:
7. In the Search: box, enter Nordic and then select nRF52 Series to highlight it as shown in this window:




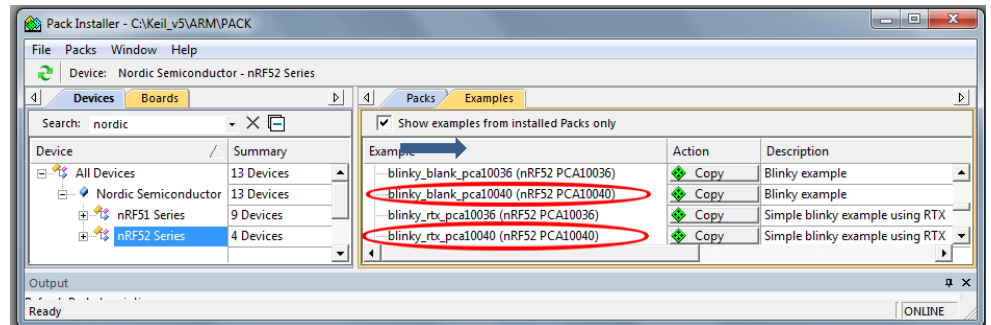
TIP: The left hand pane filters the selections displayed on the right pane. You can start with either Devices or Boards. If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or  to refresh once you have connected to the Internet.

2) Install The nRF52 Software Packs:

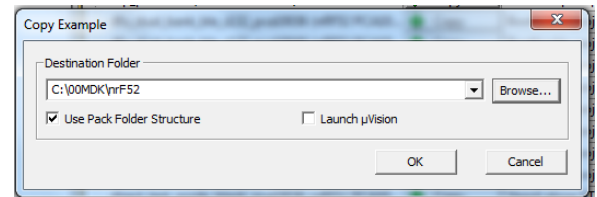
1. In the Packs tab, all the Packs available for the nRF52 you selected are visible. We do not need all of them now.
2. Make sure nRF52 Series is still selected. Select NordicSemiconductor::nRF_DeviceFamilyPack. Click Install. This Pack will download and install into MDK. Progress is indicated in the bottom left corner.
3. Make sure nRF52 Series is still selected. Select NordicSemiconductor::nRF_Examples and click Install. This Pack will also download and install into MDK.
4. A Pack's status is indicated by the “Up to date” icon: 
5. Update means there is an updated Software Pack available for download.  If your project requires older Pack components, you can select them anytime with the Select Software Packs utility.

3) Install the Blinky Examples:

1. Make sure nRF52 Series is still selected. Select the Examples tab:
2. Scroll down to find the two examples blinky_blank_pca10040 and blinky_rtx_pca10040 as shown here:
3. Click Copy beside blinky_blank_pca10040 (nRF52 PCA10040): 
4. The Copy Example window opens up: Select Use Pack Folder Structure. Unselect Launch µVision.
5. Type in C:\00MDK\nrF52\ as shown. Click OK to copy the project.



6. Select blinky_RTX_pca10040 (nRF52 PCA10040): and click Copy.
7. The two blinky examples were copied to C:\00MDK\nrF52\peripherals\ in this case.
8. “blinky_blank” is a no RTOS (bare metal) example. “blinky_RTX” uses Keil RTX. A FreeRTOS example is available but we will not use it here.




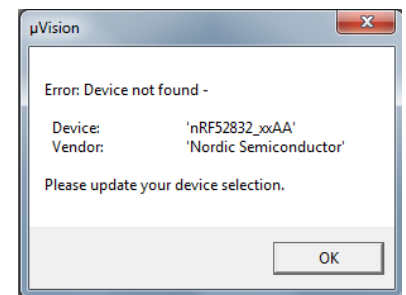
TIP: The default folder for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default folder of C:\00MDK\nrF52 in this tutorial. You can use any folder you prefer. Using 00MDK puts it conveniently at the top of your directory tree in Windows.

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


4) Install the Remaining Software Packs:

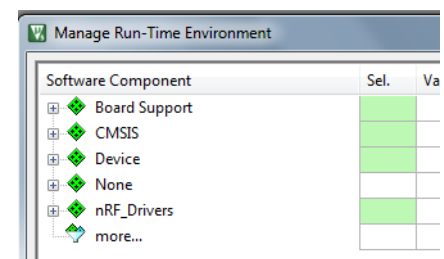
We did not install all the Packs needed for this project. Rather than select them manually, we will let µVision do this.

- 1) Start µVision if it is not running:  It might not be after the very first running of Pack Installer.
- 2) You might see this warning. Select OK and µVision will download any Packs that are needed for this project.
- 3) For each dialog window that opens, select OK or Yes as appropriate.
- 4) When this process is completed, you can close Pack Installer.



5) Open Manage Run-Time Environment to Check Installed Packs:


- 1) Open Manage Run-Time Environment by clicking on its icon: 
- 2) This window opens. If the correct Packs are installed all boxes will be green and not red or orange:
- 3) If there are red or orange – there will be text stating what is needed.
- 4) Close the MRE window.

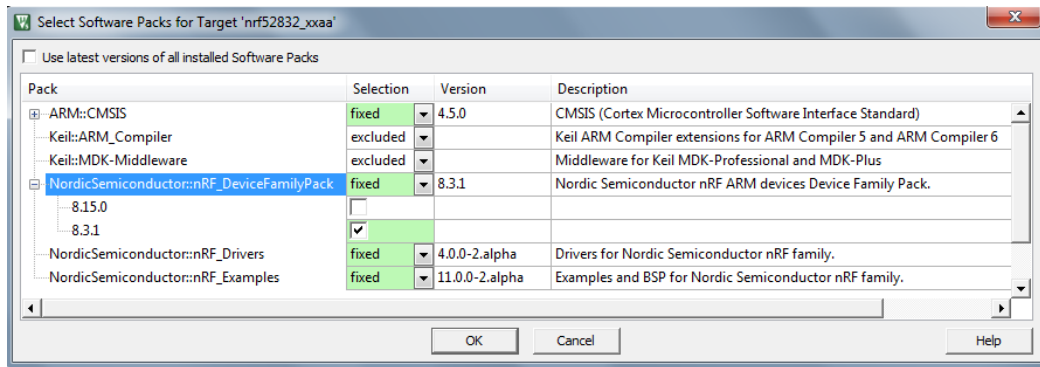



7) Other features of Software Packs:

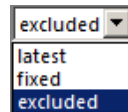
1) Select Software Pack Version:

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 not selected. The versions selected will be used.




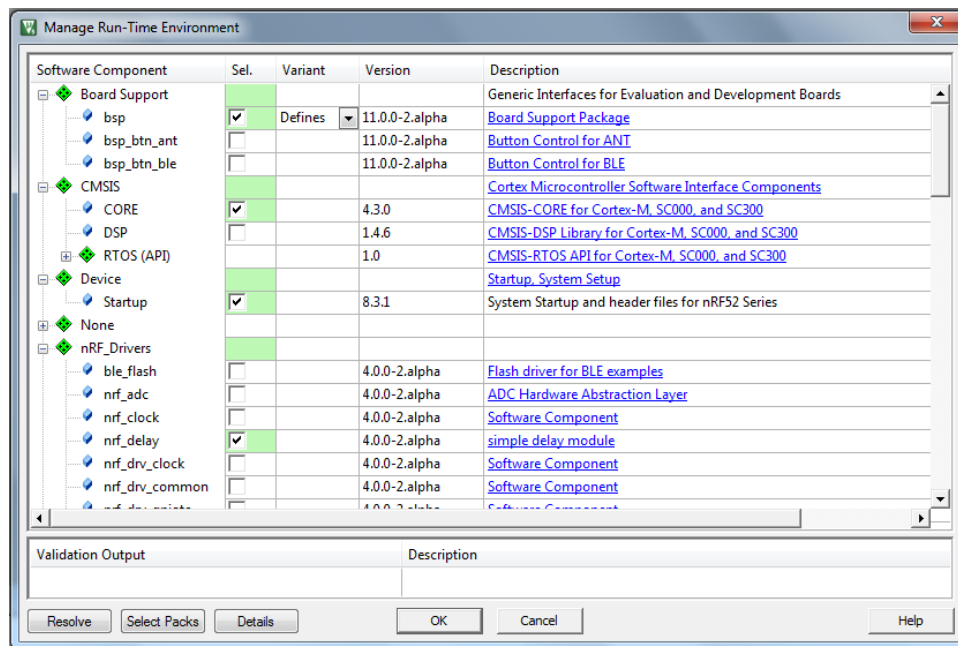
3. Note various options are visible as shown here: 
4. Select excluded and see the options as shown:
5. Do not make any changes.
6. Click Cancel to close this window to make sure no changes are made.



TIP: The correct software component versions are not selected, notification is indicated in the µVision OUTPUT window.

2) Manage Run-Time Environment (MRTE):





1. Click on the Manage Run-Time Environment (MRTE) icon:  The window below opens:
2. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project. You can select Nordic SDK files from under the nRF_Drivers header.
3. Do not make any changes. Click Cancel to close this window.



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

8) *Blinky_blank* example program using the Nordic nrF52 Development Kit:


Now we will connect a Keil MDK development system using the nrF52 board. This page will use the onboard J-Link. You can use other debug adapters if you prefer. Keil ULINK*plus* and ULINK*pro* offer various features.

1. Connect a USB cable between your PC and the nrF52 board J2 USB connector.
2. Start μ Vision by clicking on its desktop icon. 
3. Select Project/Open Project.
4. Navigate to C:\00MDK\nrF52\peripheral\blinky\pca10040\blank\arm5\.
5. Open the project blinky_blank_pca10040.uvprojx.
6. Compile the source files by clicking on the Rebuild icon. . You can also click on the BUILD icon.
7. There will be no warnings or errors indicated in the Build Output window.
8. 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.
9. Click on the RUN icon. 


The four LEDs will now blink in sequence on the board. LED1 through LED4.

Now you know how to compile a program, program it into the nrF52 processor Flash and run it !

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

10. Stop the program with the STOP icon. 

Single-Stepping:

1. With main.c in focus (the main.c tab is underlined), click on the Step icon  or F11 a few times: You will see the program counter jumps a C line at a time. The yellow arrow indicates the next C line to be executed.
2. Click on the top margin of the Disassembly window to bring it into focus. Clicking Step now jumps the program counter one assembly instruction at a time.

Debug Adapters: Which one to use ?

You can use a variety of debug adapters with your Nordic boards and μ Vision.

1. Keil ULINK2 or ULINK-ME (these are CMSIS-DAP capable)
2. Keil ULINK*plus*
3. Keil ULINK*pro*
4. CMSIS-DAP
5. Segger J-Link (on-board)

This document uses the on-board J-Link. ULINK*plus* and ULINK*pro* are featured explaining their capabilities.



ULINK*plus* offers faster Serial Wire Speed and Power consumption plus more. www.keil.com/ulinkplus

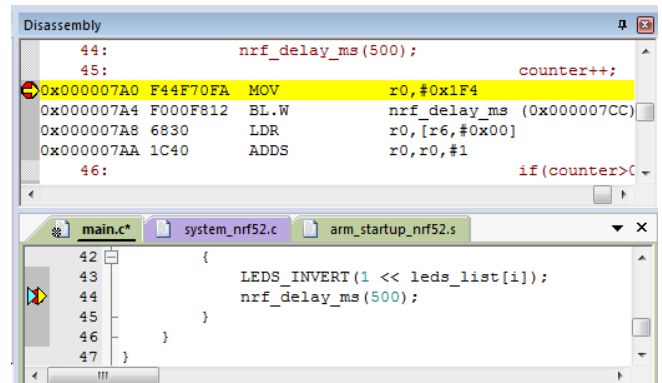
ULINK*pro* offers ETM Instruction Trace and fast SWV operation. A custom adapter is needed for the nrF52 boards.

For additional information on Keil debug adapters, see www.keil.com/mdk5/ulink

9) Hardware Breakpoints:

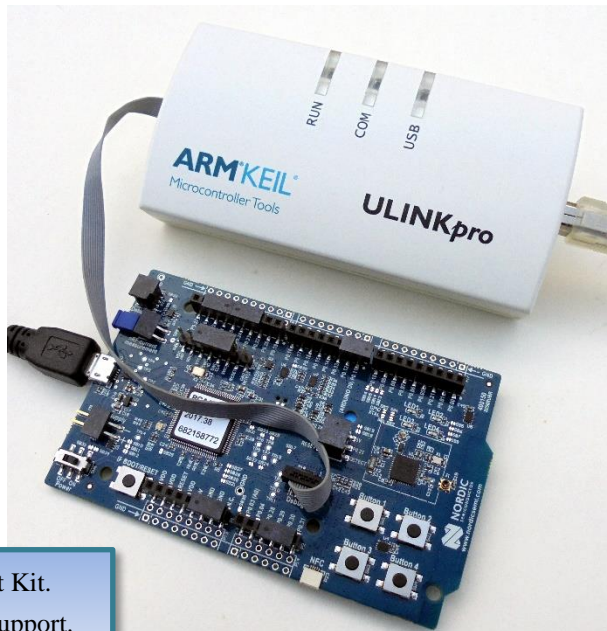
The nrF52 has six hardware breakpoints that can be set or unset on the fly while the program is running.

1. Click on the RUN icon. 
2. With Blinky running, in the main.c window, click on a darker grey block on the left on the C line nrf_delay_ms(500) near line 44. The gray blocks mean assembly instructions are present at these points. You can also click in the Disassembly window to set a breakpoint.
3. A red circle will appear and the program will presently stop.
4. Note the breakpoint is displayed in both the Disassembly and source windows as shown here:
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.





ULINKpro with Nordic nrF52 Development Kit.
10 Pin CoreSight connector shown. SWV support.
ETM needs an extra 5 pins connected to this board.

10) Call Stack + Locals Window:

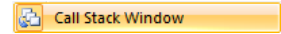
Local Variables:

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

1. Click on RUN . After a few seconds, click Stop. 


2. Click on the Call Stack and Locals tab or open it by selecting View/Call Stack Window.



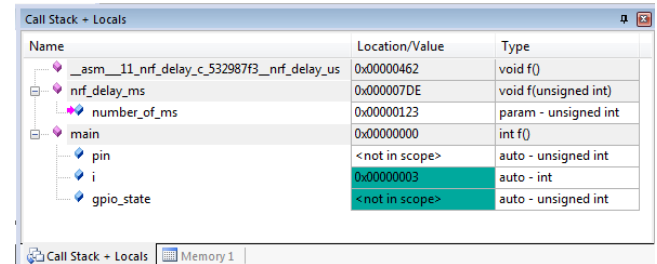
3. Expand some of the entries. This window will be similar to this one:


4. The functions as they were called are displayed.

5. If these functions have local variables, they are displayed. If they are in scope, their values will be displayed. If not in scope, this will be indicated:

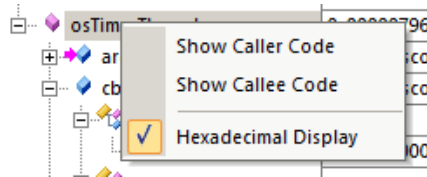
6. Click Step Out  to immediately exit a function.

7. You can see the program leaving various functions to go back to main().



8. Click on the Step In icon  to enter a few functions.

9. Right click on a function and select either Callee or Caller code and this will be highlighted in the source and disassembly windows.



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.

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





The Watch and Memory windows will display updated variable values in real-time while your program runs. 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 (peripherals) windows work using the same CoreSight technology. Call Stack, Watch and Memory windows can’t see local variables unless stopped in their function. They must be in scope to have their values displayed.

Watch Window:

Create the global variable counter:

1. Click Stop. . Exit Debug mode. 
1. In main.c, declare the global variable counter near line 29: `uint32_t counter = 0;`
2. In main.c, after the line `nrf_delay_ms(500);`, add these lines starting at near line 45:

```
45 counter++;  
46 if(counter > 0x0F) counter = 0;
```

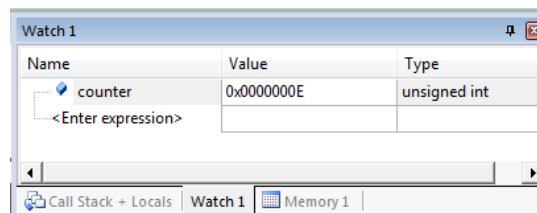
3. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon or F7.
4. Enter Debug mode by clicking on the Debug icon. 
5. In the toolbar, select View and enable Periodic Window Update.  Periodic Window Update If unselected, many windows update only when the program is stopped. This does not include the trace windows which must be stopped.
6. Click on the RUN icon. 

Enter counter into Watch1:

TIP: You can configure a Watch or Memory window while the program is running.

1. In main.c, right click on counter and select Add counter to ... and select Watch 1. Watch 1 will automatically open.
2. counter will be displayed as shown here updating in real time:
3. Select the counter Value (double click) and enter a different value. This will be inserted into the counter. You have to enter it quickly. It is easier to change values in a Memory window.

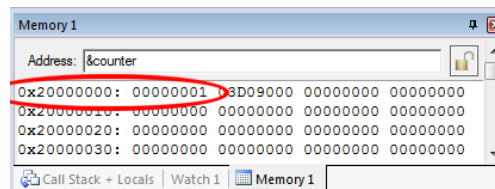
TIP: If Watch 1 does not update unless you stop the program, check Step 5 above: Periodic Window Update.



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

Memory window:

1. In main.c, right click on counter and select Add counter to ... and select Memory 1. Memory 1 will open.
2. Note the value of counter is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to but this not what we want to see at this time.
3. Add an ampersand “&” in front of the variable name and press Enter. The physical address here is 0x2000_0000.
4. Right click in the Memory window and select Unsigned/Int.
5. The data contents of counter is displayed as shown here: (0x01)
6. Both the Watch and Memory windows are updated in real-time.
7. Right-click with the mouse cursor over the desired data field and select Modify Memory. You can change a memory or variable on-the-fly while the program is still running.



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

12) 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**.

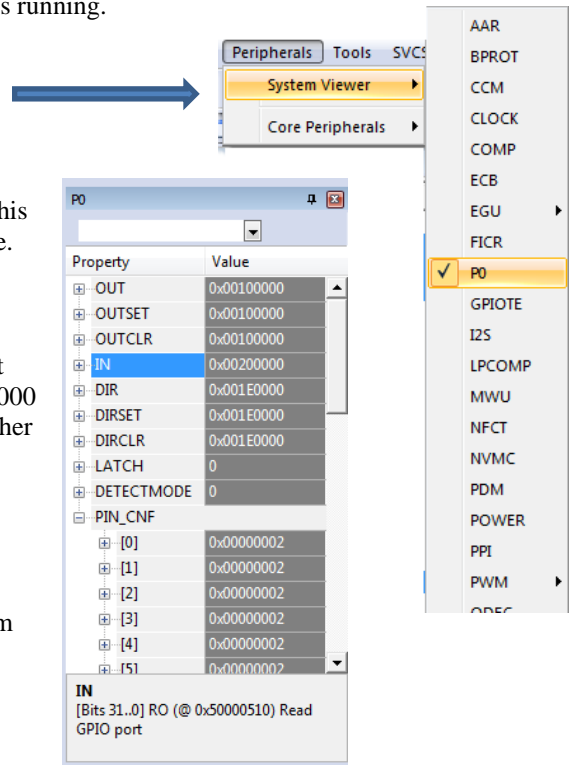
1. Click on RUN. You can open SV windows when your program is running.

Select P0:

2. Select Peripherals/System Viewer and then P0 as shown here.
3. This window below opens:
4. You can now see various registers update as counter 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. These are the GPIO port P0 pins connected to the user LEDs.

TIP: If you click on a register in the Property column, a description about this register will appear at the bottom of the window. IN is located at 0x5000 0510. This is a handy way to determine register physical addresses and other properties.

7. You can look at other Peripherals contained in the System View windows.
8. When you are done, stop the program  and close all the System Viewer windows that are open.



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

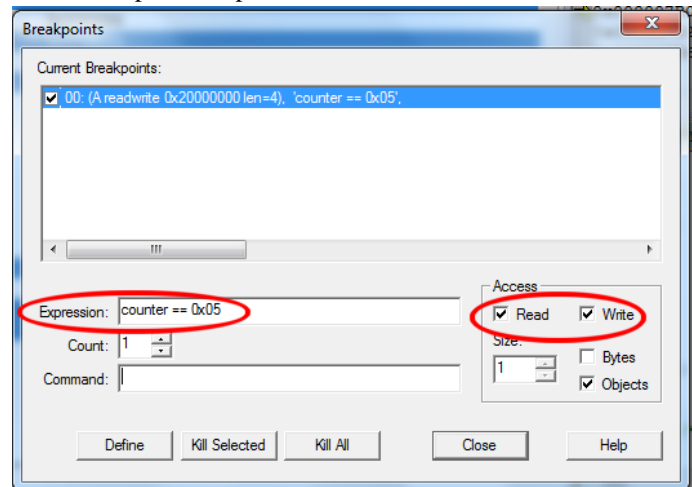
You must make sure a given peripheral register allows and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.

13) Watchpoints: Conditional Breakpoints




The nrF52 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 the equality test. Currently, you can set one Watchpoint with μ Vision.

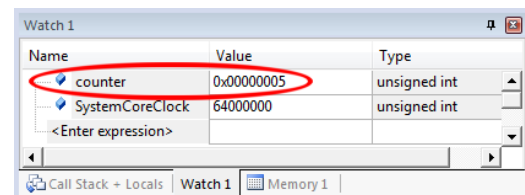
Configure a Watchpoint:

1. Use the same Blinky configuration as the previous page. You can configure a Watchpoint while the program is running or halted. With a J-Link, you must stop the program. All Keil ULINKs allow live configuration.
2. We will use the same global variable **counter** found in main.c you created to explore the Watch windows.
3. Select Debug in the main μ Vision window and then select Breakpoints or press Ctrl-B.
4. Select Access to Read and Write.
5. Enter: "counter == 0x05" 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.
7. Click on Close.
8. Enter the variable counter in Watch 1 if it is not already there.



Run the Program:

1. Click on RUN. 
2. When counter equals 0x05, the Watchpoint will stop the program. See Watch 1 shown below:
3. 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 any Read and/or Write as selected.
4. This is useful to detect stack pointer levels. Set a Watchpoint on a low stack address. If hit, the program will stop.
5. To repeat this exercise, change counter to something other than 0x05 in the Watch window and click on RUN.
6. Stop the CPU if it is running. 
7. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.
8. 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 physical addresses can be used with a Watchpoint. An example is: *((unsigned long *)0x20000004)

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

14) Serial Wire Viewer (SWV):

Serial Wire Viewer is a data trace including interrupts in real-time without any code stubs in your sources. SWV is output on the Serial Wire Output (SWO) pin. SWV can be limited by overruns. This example uses the on-board J-Link. For higher SWV performance, use a ULINKpro or a ULINKplus. For even higher throughput, use the ULINKpro 4 bit Trace Port.





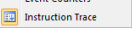



Configure SWV:

1. Stop the CPU if it is running.  Exit Debug mode. 
1. Obtain the file nrF52_SWO.ini from where you got this tutorial. This enables the Serial Wire Out pin (SWO) pin.
2. Copy it to C:\00MDK\nrF52\peripheral\blinky_blank\pca10040\blank\arm5\
2. Select Options for Target  or ALT-F7 and select the Debug tab. Your debugger must be displayed beside Use:..
3. Enter nrF52_SWO.ini in the right side Initialization File: use the Browse button: 
4. Select Settings:  on the right side of this window.
5. In the Target Driver setup window that opens, confirm Port: is set to SW and SWJ box is enabled if it is displayed. SWV will not work with JTAG. The nrF52 has only Serial Wire Debug and not JTAG.
6. Click on the Trace tab. The window below is displayed.
7. In Core Clock: enter 32 MHz. Select Trace Enable. This value *must* be set correctly to 1/2 CPU speed for the nrF52.

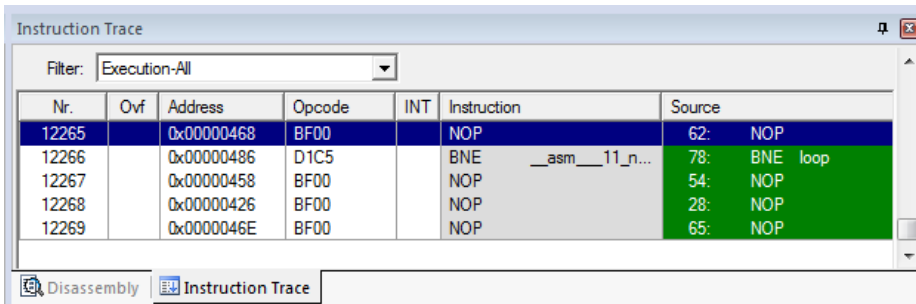
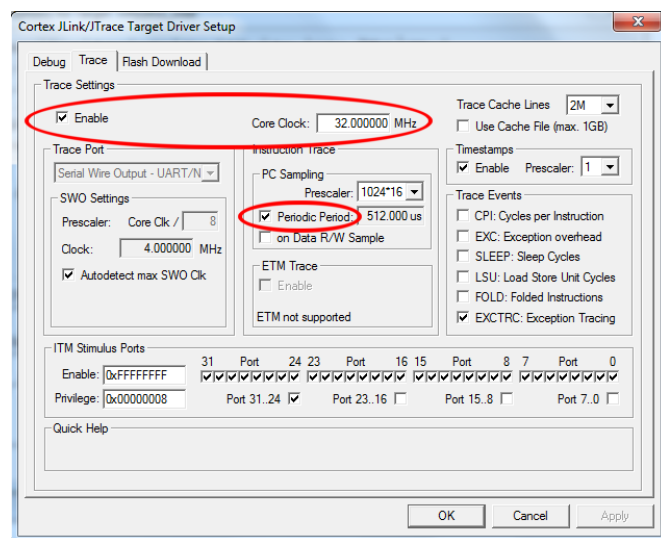
TIP: To find Core Clock frequency: Enter the global variable SystemCoreClock in a Watch window and run the program.

8. Select Periodic to enable PC Samples.
9. Leave all other settings at default.
10. Click on OK twice to return to the main µVision menu.
11. SWV is now configured and ready to use.

Display Trace Records:

1. Select File/Save All or click .
2. Enter Debug mode. 
3. Click RUN. 
4. Open Trace Records window by clicking on the small arrow beside the Trace icon and select Instruction Trace:  
5. The Instruction Trace window will open as shown below:
6. Stop the program  to display updated trace frames.
7. If you see instructions as shown, SWV is working correctly. If not, the most probable cause is a wrong Core Clock:.. In many Arm processors, Core Clock: is equal to the CPU clock. In the nrF52, it is 1/2 the CPU clock.
8. Keil ULINK adapters have different trace windows. Sometimes they are called Trace Data or Trace Records.
9. Select Options for Target  and select the Debug tab.
10. Select Settings: 
11. Select the Trace tab. Unselect periodic and click Close twice.

TIP: The only valid ITM frames are ITM 0 and ITM 31. If you see any other values such as ITM 13, this nearly always means the Core Clock: value is incorrect.



TIP: LED2 shares the same pin as SWO.




When this program runs, LED2 is dim and does not blink. If the writes to the LEDS cause problems just comment out this line in main.c: It is located near line 36 just inside the main() function: `LEDS_CONFIGURE(LEDS_MASK);`

15) Displaying Variables Using the Logic Analyzer (LA):

This example will demonstrate how to display four variables in a graphically using the Blinky example and the global variable counter you created. This page assumes you have the SWV configured and working as described on the previous page.

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 nrF52. Cortex-M0 does not have Serial Wire viewer. The nrF51 series are Cortex-M0. LA uses the same four comparators as Watchpoints so everything can't be used at same time.

Configure the Logic Analyzer:

1. SWV must be configured as found on the previous page.
2. µVision must be in Debug mode.  Run the program.  **TIP:** You can configure the LA while the program is running or stopped.
3. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 
4. Locate the global variable **counter** in main.c. You can use any instance of the variable.
5. Right click on **counter** and select counter 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. 32 mHz.

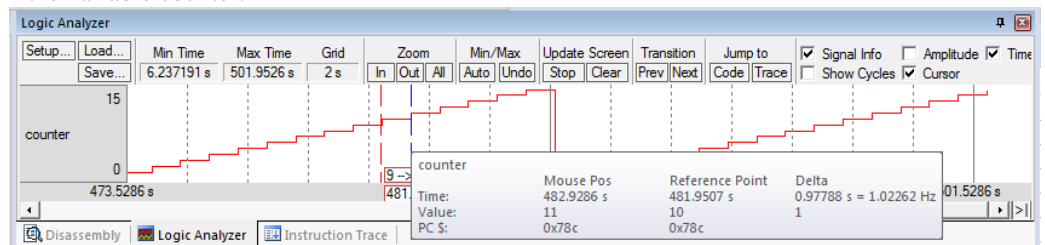
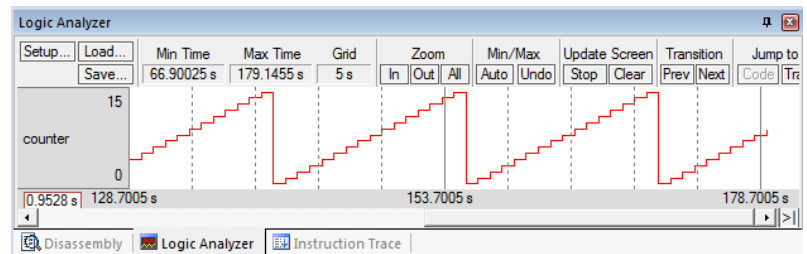
6. In the LA, click on Setup and set Max: in Display Range to 0x0F and Min to 0x0. Click on Close.
7. The LA is now configured to display counter in a graphical format and counter should be displaying now.
8. **counter** should still be in the Watch and Memory windows. It will be changing as counter increments.
9. 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. Cycle the board power.

TIP: The Logic Analyzer can display up to four static or 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 the register.

Measure Timings in LA:

1. Either stop the program or select Stop in the LA Update Screen box.
2. Select Cursor and Signal Info.
3. Click on an edge on a step on counter.
4. Move the cursor to the other edge and hover to open the time box as shown below:
5. Note this step is 0.99 s in the variable counter.
6. You can also measure amplitudes and other statistics of this waveform.



View Write of counter:

When a variable is added to the Logic Analyzer, the data write frames are sent to the Trace Data or Records window. To display these any Keil debug adapter is needed.

A J-Link does not support this feature.

1. The first line says data 0x0F was written to 0x2000_0000 by the instruction located at 0x078C.
2. The execution addresses in the SRC Code column is provided by selecting On Data R/W Sample in the trace configuration window.
3. Red **D** is SWV overload. Two timestamps have the same value. µVision recovers from SWV overloads.

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



TIP: SWV is easily overloaded as it is a lot of data to send out one pin. A ULINKplus or ULINKpro provides faster SWV.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
471.473 533 375 s	W: 0x20000000	0x0000000F	X: 0x0000078C	
472.426 003 813 s	W: 0x20000000	0x00000010	X: 0x0000078C	
D 472.426 007 813 s	W: 0x20000000	0x00000000	X: 0x00000792	
473.378 474 312 s	W: 0x20000000	0x00000001	X: 0x0000078C	
474.330 944 750 s	W: 0x20000000	0x00000002	X: 0x0000078C	
475.283 415 187 s	W: 0x20000000	0x00000003	X: 0x0000078C	
476.235 885 625 s	W: 0x20000000	0x00000004	X: 0x0000078C	
477.188 356 125 s	W: 0x20000000	0x00000005	X: 0x0000078C	
478.140 826 563 s	W: 0x20000000	0x00000006	X: 0x0000078C	

16) Displaying Exceptions (including interrupts) using SWV:

The Trace Exceptions window displays exceptions including interrupts firing with suitable timing information. These are also displayed in the trace window.

Configure SysTick Timer: This will provide some interrupts to display in this simple example program.

1. Click Stop. . Exit Debug mode. .
2. In main.c add these lines in appropriate places: This section should be before the main() function.

```
31 void SysTick_Handler(void) (  
32     static uint32_t ticks;  
33     ticks++;  
34     If (ticks > 0x0F0) ticks = 0;  
35 )
```


Inside the main() function (to configure and enable the SysTick timer):

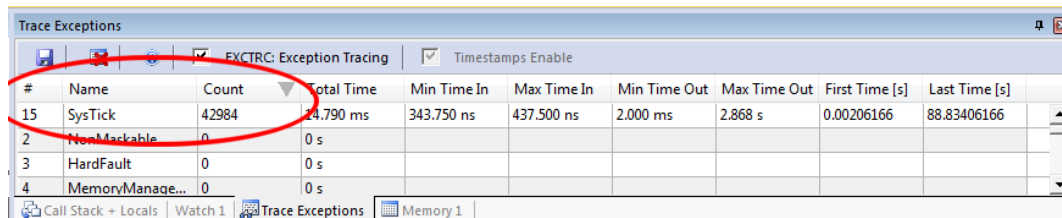
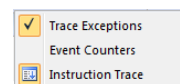
```
45 SysTick_Config(SystemCoreClock/1000); //1 msec
```

Compile and RUN:

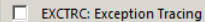
1. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon or F7.
2. Enter Debug mode by clicking on the Debug icon.  Run the program. .



Open the Trace Display Trace Exceptions:

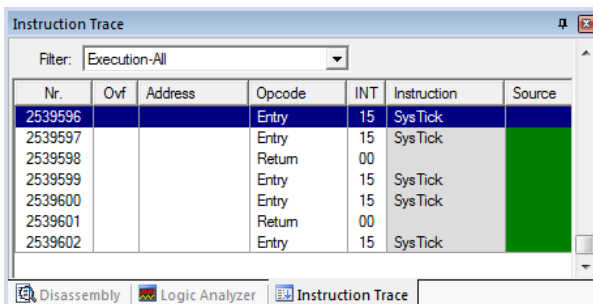
1. Open the Trace Exceptions window by selecting Trace Exceptions as shown here: .
2. **OR:** Select the Trace Exceptions tab (located beside the Watch and Memory windows).
3. Click in the Count column header to bring the active exceptions to the top as shown below:
4. Note the various timings displayed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



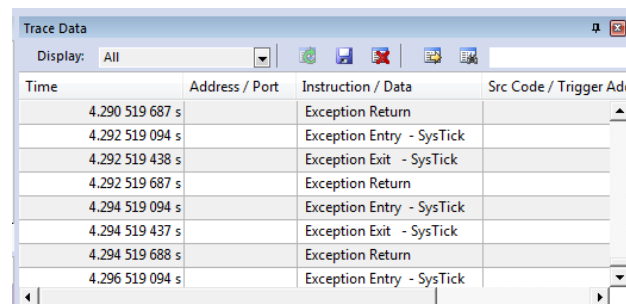
#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
15	SysTick	42984	24.790 ms	343.750 ns	437.500 ns	2.000 ms	2.868 s	0.00206166	88.83406166
2	NonMaskable	0	0 s						
3	HardFault	0	0 s						
4	MemoryManage...	0	0 s						

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 reduced on the SWO pin.

5. Select Instruction Trace or click on its tab if it is already available. This window opens:
6. Exceptions are displayed. The left picture is J-Link and the right is ULINKplus. Note ULINKplus displays the exception Entry, Exit and Return. This is useful to detect tail-chaining. Timestamps are also provided with ULINKs.
7. ULINKpro and ULINKplus will display interrupts graphically with RTX threads in the Event Viewer.
8. Click Stop. . Exit Debug mode. .



Nr.	Ovf	Address	Opcode	INT	Instruction	Source
2539596			Entry	15	SysTick	
2539597			Entry	15	SysTick	
2539598			Return	00		
2539599			Entry	15	SysTick	
2539600			Entry	15	SysTick	
2539601			Return	00		
2539602			Entry	15	SysTick	



Time	Address / Port	Instruction / Data	Src Code / Trigger Adc
4.290 519 687 s		Exception Return	
4.292 519 094 s		Exception Entry - SysTick	
4.292 519 438 s		Exception Exit - SysTick	
4.292 519 687 s		Exception Return	
4.294 519 094 s		Exception Entry - SysTick	
4.294 519 437 s		Exception Exit - SysTick	
4.294 519 688 s		Exception Return	
4.296 519 094 s		Exception Entry - SysTick	

17) printf using ITM (Instruction 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, no CPU cycles are required to get the data out of the processor and into μ Vision for display in the Debug (printf) Viewer window.

No UART is needed for this technique.

1. Add this line to main.c. A good place is near line 30, just after the declaration of `counter`.


```
#define ITM_Port8(n) (*(volatile unsigned char *) (0xE0000000+4*n))
```
2. In the main function in main.c after the counter test statement, enter these lines after near line 57:

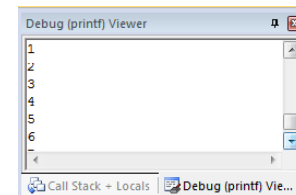
```
ITM_Port8(0) = counter + 0x30; /* displays count value: +0x30 converts to ASCII */  
while (ITM_Port8(0) == 0);  
ITM_Port8(0) = 0x0D;  
while (ITM_Port8(0) == 0);  
ITM_Port8(0) = 0x0A;
```

Compile and Enter Debug Mode:

1. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon or F7.
2. Enter Debug mode by clicking on the Debug icon. 

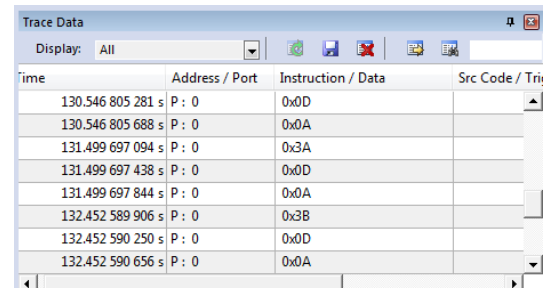
Configure SWV:

1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample and ITM Port 31. (this is to help not overload the SWO port)
3. Select EXCTRC and ITM Port 0. ITM Stimulus Port "0" enables the Debug (printf) Viewer.
4. Click OK twice.
5. Click on View/Serial Windows and select Debug (printf) Viewer.
6. Run the program. 
7. In the Debug (printf) Viewer you will see the ASCII values of counter appear:



Trace Records When Using a Keil ULINK:

1. Open the Trace Records if not already open. Double click on it to clear it.
2. You will see a window such as the one below with ITM frames. The ASCII data is clearly visible coming from Port 0 (P0).
3. This is the Trace Data window using ULINK_{plus}. The onboard J-Link does not display ITM frames.



Time	Address / Port	Instruction / Data	Src Code / Tri
130.546 805 281 s	P: 0	0x0D	
130.546 805 688 s	P: 0	0x0A	
131.499 697 094 s	P: 0	0x3A	
131.499 697 438 s	P: 0	0x0D	
131.499 697 844 s	P: 0	0x0A	
132.452 589 906 s	P: 0	0x3B	
132.452 590 250 s	P: 0	0x0D	
132.452 590 656 s	P: 0	0x0A	

TIP: To save ITM data to a file, see the command ITMLOG:

www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm

ITM Conclusion

The writes to ITM Stimulus Port 0 are intrusive and are usually one cycle. It takes no CPU cycles to get the data out the processor via the Serial Wire Output pin to μ Vision.

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enter only those features that you really need. You can use a ULINK_{plus} or a ULINK_{pro} for more SWV throughput.

Super TIP: ITM_SendChar is a useful function you can use to send characters. It is found in the header core_CM4.h.

It is possible to add the ability to use a printf statement in your code. Obtain the file retarget.c and adapt it to your program using ITM.

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.








ITM documentation is here: www.keil.com/pack/doc/CMSIS/Core/html/group_ITM_Debug_gr.html

18) RTX System and Threads Viewer:


This example uses the Arm RTX RTOS in tickless mode to save power. RTX has an BSD license and sources and documents are included. See www.keil.com/mdk5/cmsis/rtx. It is located on https://github.com/ARM-software/CMSIS_5

Keil uses the term "threads" instead of "tasks" for consistency. Using an RTOS is becoming increasingly common as projects complexity increases.

Running System and Threads Viewer:

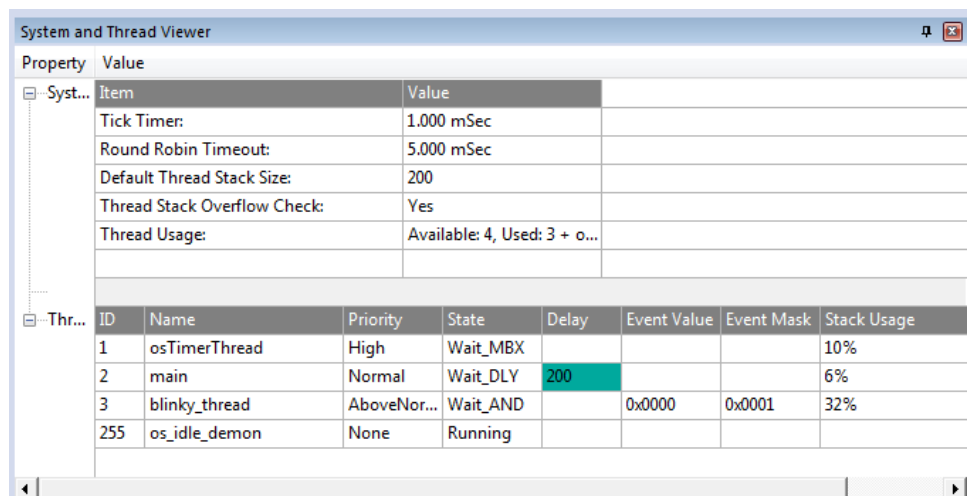
1. Stop the CPU if it is running.  Exit Debug mode. 
2. Navigate to C:\00MDK\nrF52\peripheral\blinky_RTX\pca10040\blank\arm5\.
3. Open the project blinky_RTX_pca10040.uvprojx.
4. µVision will want to download and install some additional Packs. Allow this to occur. Respond as appropriate.
5. Compile the source files by clicking on the Rebuild icon. . Enter Debug mode. 
6. In the toolbar, select View and enable Periodic Window Update.  Periodic Window Update
7. Click on the RUN icon.  LED1 and LED2 will blink as the threads are executed in turn.
8. Select Debug/OS Support and select 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 and resize the columns to comfortably view it.
9. This window updates as the various threads switch state. Only the Delay value for main thread seems to change.
10. There are three Threads listed under the Threads heading. osTimerThread, main and blinky_thread. These are located in main.c near lines 88, 98 and 64 respectively. It is easy to add more threads to RTX.
11. Stop the program.  It will probably stop in the os_idle_demon as shown by Running in the State column. This program spends most of its time in os_idle_demon. This can be adjusted to meet your needs.

Stopping in a Thread:

1. Set a breakpoint in two of the tasks in main.c by clicking in the left margin on a grey area.
 - a. Near line 75: nrf_gpio_pin_toggle(GPIO_OUTPUT_!1) in blinky_thread.
 - b. Near line 122: UNUSED_VARIABLE(osSignalSet (blinky_thread_id, SIGNAL_OUTPUT_!_TOGGLE));
2. Click on Run  several times and the program will stop 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 Running displayed beside the thread name in the State column. In the screen below, os_idle_demon is Running.
4. **Remove all breakpoints** by clicking on each one. You can use Ctrl-B and select Kill All.
5. 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. This will work with the nrF51 Cortex-M0 series.







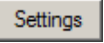
Property	Value
System	
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 4, Used: 3 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				10%
2	main	Normal	Wait_DLY	200			6%
3	blinky_thread	AboveNor...	Wait_AND		0x0000	0x0001	32%
255	os_idle_demon	None	Running				

19) Event Viewer: Uses SWV:




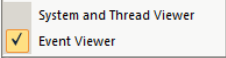

Event Viewer displays the RTX threads executing in a graphical format. It is easy to make measurements of various timings of RTX. FreeRTOS has a similar window in μ Vision. A ULINK pro and ULINK $plus$ have the advantage of also displaying interrupt routines. A ULINK $plus$ will display current draw related to threads and more information.

Configure Serial Wire Viewer (SWV): This is for J-Link. Other debug adapters are similar.

1. Stop the CPU if it is running.  Exit Debug mode. 
2. Obtain the file nrF52_SWO.ini from where you got this tutorial. This turns on the Serial Wire Out pin (SWO) pin.
3. Copy it to C:\00MDK\nrF52\peripheral\blinky_RTX\pca10040\blank\arm5\ or somewhere easy to access.
4. Select Options for Target  or ALT-F7 and select the Debug tab. Your debugger must be displayed beside Use:.
5. Enter nrF52_SWO.ini in the Initialization File: Use the Browse button: 
6. Select Settings  on the right side of this window.
7. Confirm Port: is set to SW and SWJ box is enabled if it is displayed. SWV will not work with JTAG.
8. Click on the Trace tab. The window below is displayed.
9. In Core Clock: enter 32 MHz. Select Trace Enable. This value *must* be set correctly to your CPU speed.
10. Click on OK twice to return to the main μ Vision menu. SWV is now configured and ready to use.

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

Display Event Viewer:



1. Select File/Save All or click .
2. Enter Debug mode.  Click on the RUN icon. 
3. Select Debug/OS Support and select Event Viewer. 
4. The Event Viewer opens: 
5. If you do not see the blue blocks, SWV is not working correctly. The most probably cause is a wrong Core Clock:.
6. Enable Timing Info and Cursor.
7. You can use the mouse to set two cursors for timing measurement.

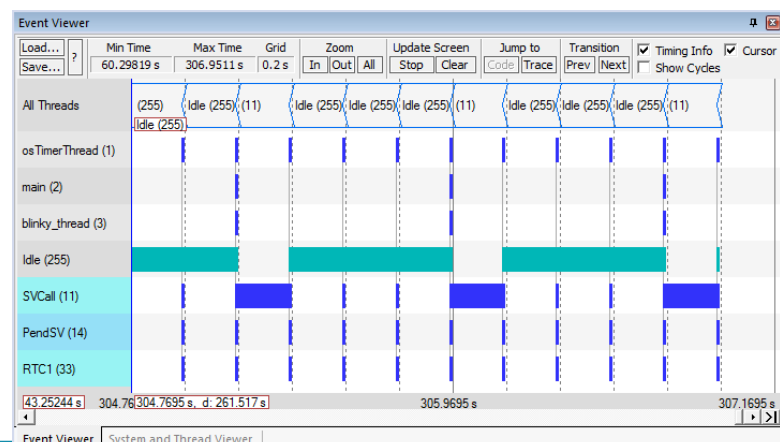
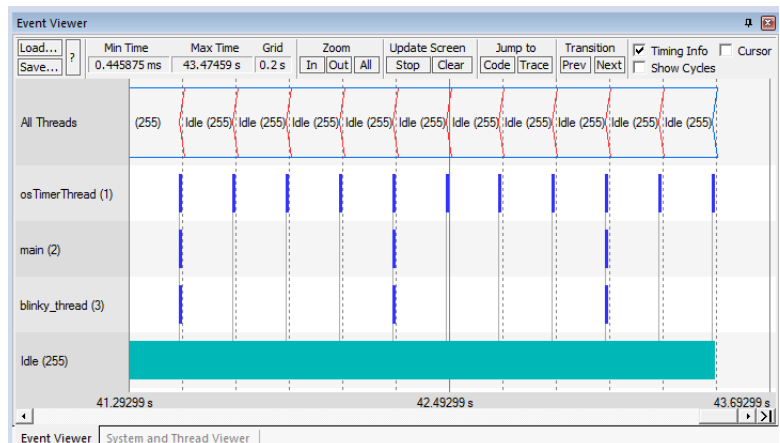
TIP: You can stop the update of Event Viewer without stopping the program.

Interrupt Displays: The bottom right window shows how interrupts are displayed when RTX and a ULINK pro or a ULINK $plus$ is used. This is invaluable to determine how long a handler runs. You can see RTC1, the timer for RTX tickless.

RTX tickless RTOS: This version of RTX is tickless. This means the program stops when it is in the idle demon. Source code for this is RTX_Conf_CM.c.

STOP and Start the Program:

1. If you stop the program with STOP , it will not start normally.
2. You can click RESET to start it. 
3. If Event Viewer stops, click RESET while the program is halted. Then RUN to start again.



20) Using ETM with blinky on the Nordic nrF52 PCA10040 board:

Now we will connect the Keil MDK development system using the nrF52 board. A Keil ULINK_{pro} must be used to collect the ETM instruction trace frames. ETM displays all the instructions executed.

1) Connecting ULINK_{pro} to ETM Trace Port with Custom Adapter:

At the time of this writing (November 2017) there is no adapter available to directly connect the 20 pin ULINK_{pro} cable to the 10 pin SWD/JTAG P18 DEBUG IN connector and the 5 ETM signals (D0-D3 and TrcClk). A suitable adapter will need to be constructed for this tutorial.




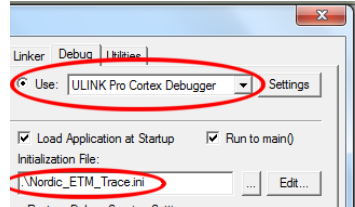




1. See www.keil.com/coresight/coresight-connectors/ for the connector pinouts.
2. The first 10 pins contain the JTAG/SWD debug control signals.
3. Pins 12 through 20 receive the ETM signals. Ground is provided on pins 15, 17 and 19.
4. ETM signals are output on various Port 0 pins as shown below:
5. ETM signals share LED2, LED4 and BTN2 through BTN4. These cannot be used at the same time.

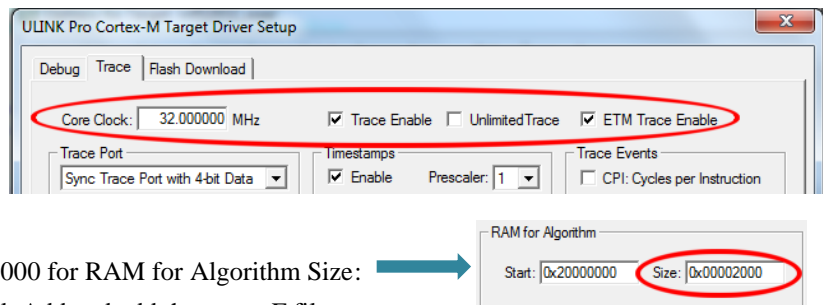
P0 Port on nrF52	ETM Signal	LED or BTN	20 pin ULINK _{pro} Connector
P014	D3	BTN2	20
P015	D2	BTN3	18
P016	D1	BTN4	16
P018	D0	LED2 / SWO	14
P020	TraceCLK	LED4	12

2) Connect ULINK_{pro} to the nrF52:

1. Connect a ULINK_{pro} to the nrF52 board using a custom connector assembly. Plug a USB cable to your PC.
2. Select Project/Open Project.
3. Navigate to C:\00MDK\nrF52\peripheral\blinky\pca10040\blank\arm5\.
4. Open the project blinky_blank_pca10040.uvprojx. You can use the one you modified previously in this tutorial.
5. In main.c, near line 37 in the main() function, comment this line out: LEDS_CONFIGURE(LED5_MASK); This is needed because LED1 and LED2 conflict with two ETM signals. See the chart above.

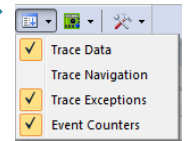
3) Configure ETM Trace in µVision:

1. Select Options for Target . Click on the C/C++ tab.
2. For easy debugging view, set Optimization to Level 0. 
3. Click on the Debug tab to select a debug adapter.
4. Select ULINK Pro Cortex Debugger... as shown: 
5. Obtain the file Nordic_ETM_Trace.ini from where you got this tutorial.
6. Copy it here: C:\00MDK\nrF52\peripheral\blinky\pca10040\blank\arm5 or somewhere handy.
7. Enter this .ini file in the Initialization File: box as shown above right partial window. Use the Browse button. 
8. Select Settings: 
9. Click on the Trace tab.
10. In Core Clock: enter 32 MHz.
11. Select Trace Enable.
12. Select Sync Trace Port with 4-bit Data:
13. Select ETM Trace Enable. 
14. Click the Flash Download tab. Enter 0x2000 for RAM for Algorithm Size: 
15. If Flash programming does not work, click Add and add the two nrF files.
16. Click OK twice to return the main µVision menu.
17. Select File/Save All or click . *Continued on the next page:*



4) Compile and RUN and View the ETM Instruction Trace Window:

1. Compile the source files by clicking on the Rebuild icon.
2. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Open ETM Trace window by clicking on the small arrow beside the Trace icon and select Trace Data:
3. **DO NOT** click on the RUN icon. The program will run to the start of main() and stop as shown below. If you do click on RUN, exit and reenter Debug to restart the program.



Trace Data				
Display:	All		in	All
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x0000748	CMP r2,#0x00		_scatterload_zeroinit
0.000 333 000 s	X : 0x000074A	*BNE 0x0000744		_scatterload_zeroinit
0.000 333 125 s	X : 0x000074C	BX lr		_scatterload_zeroinit
	X : 0x00004C2	ADDS r4,r4,#0x10		_scatterload
	X : 0x00004C4	CMP r4,r5		_scatterload
0.000 333 375 s	X : 0x00004C6	*BCC 0x00004B6		_scatterload
	X : 0x00004C8	BL.W __main_after_scatterload (0x0000408)		_scatterload
	X : 0x0000408	LDR r0,[pc,#0] ; @0x000040C		???
0.000 333 750 s	X : 0x000040A	BX r0		???

4. Examine the Trace Data window fields. The BX r0 is the last instruction executed. BX is the jump to the beginning of main(). A timestamp is shown and if an instruction is related to a C/C++ source line, it will be shown in the Src Code column. The Function column indicates the function an instruction is in. The first occurrence will be highlighted orange.
5. The source and Disassembly windows display the program as it was written. ETM displays the program as it ran.
6. Double-click on any instruction and it will be highlighted in the Disassembly and source window.
7. Scroll up to the top of this window and you can see the first instruction executed after RESET. In this case, it is a LDR at memory 0x0000_048C. All instructions executed from RESET to start of main() are recorded and displayed.
8. If you unselect **run to main()** in the Debug tab, ☒ Run to main() the program will not run and the PC will point to the first instruction. You can step through the program and the trace will display the instructions assuming each instruction can safely be stepped in the initialization code.
9. This window saves the last one million instructions. You can save all instructions but this will consume disk space.

Trace Data				
Display:	All		in	All
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
		TRACE RUN		
	X : 0x0000048C	LDR r0,[pc,#24] ; @0x000004A8	LDR R0,=SystemInit	Reset_Handler
0.000 000 375 s	X : 0x0000048E	BLX r0	BLX R0	Reset_Handler
	X : 0x00000500	PUSH {r4,lr}	{	SystemInit
	X : 0x00000502	BL.W errata_31 (0x00000718)	if (errata_31()){	SystemInit
	X : 0x00000718	LDR r0,[pc,#112] ; @0x0000078C	if (((*(uint32_t *)0xF0000FE0) & 0x000000FF) == ...	errata_31
	X : 0x0000071A	LDRB r0,[r0,#0x00]		errata_31
	X : 0x0000071C	CMP r0,#0x06		errata_31
0.000 000 563 s	X : 0x0000071E	*BNE 0x00000786		errata_31

10. Click in the Disassembly window and click Step. The program steps one instruction:
11. If a source window is in focus, Step will step one source line at a time and all instructions for this line are recorded.

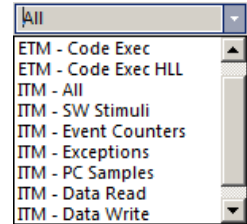
0.001 001 094 s	X : 0x0000040A	BX r0		???
		TRACE RUN		
0.001 334 781 s	X : 0x00000750	LDR r7,[pc,#52] ; @0x00000788	LEDS_INVERT(1 << leds_list[i]);	main

21) 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 μ Vision will allow more precise filters to be selected. Currently, with the nrF52, ETM instructions and SWV frames cannot be displayed at the same time unless the trace window is not full.




Find a Trace Record:

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

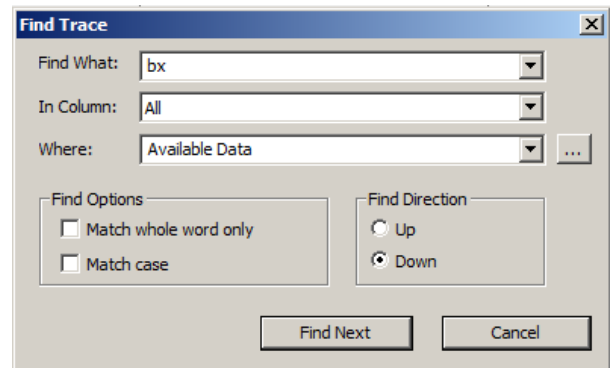






You can select properties where you want to search in the “in” box. "All" is shown in the screen above:

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.

This is shown in the screen below:



Trace Data				
Display: All     bx in All				
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x000004F0	LDR r0,[pc,#4] ; @0x000004F8	SystemCoreClock = _SYSTEM_CLOCK_64M;	SystemCoreClockUpdate
	X: 0x000004F2	LDR r1,[pc,#8] ; @0x000004FC		SystemCoreClockUpdate
	X: 0x000004F4	STR r0,[r1,#0x00]		SystemCoreClockUpdate
0.000 014 625 s	X: 0x000004F6	BX lr	}	SystemCoreClockUpdate
0.000 014 812 s	X: 0x000006CE	POP {r4,pc}	}	SystemInit
	X: 0x00000490	LDR r0,[pc,#24] ; @0x000004AC	LDR R0, __main	Reset_Handler
0.000 015 063 s	X: 0x00000492	BX r0	BX R0	Reset_Handler
	X: 0x00000400	LDR.W sp,[pc,#12] ; @0x00000410		???
	X: 0x00000404	BL.W __scatterload (0x000004B0)		???
	X: 0x000004B0	LDR r4,[pc,#24] ; @0x000004CC		__scatterload
	X: 0x000004B2	LDR r5,[pc,#28] ; @0x000004D0		__scatterload
	X: 0x000004B4	B 0x000004C4		__scatterload

22) Trace Triggering Background:

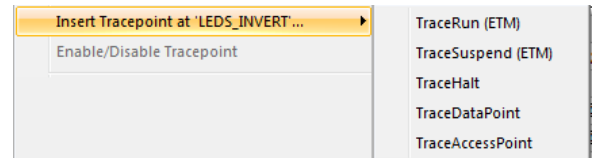
You can configure two sets of trace triggers. μ Vision has three types of instruction trace trigger and two data trace commands as listed here: They are set and unset in a source or Disassembly window or in the μ Vision Command window.

Instruction Trace Triggers:

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.

Data Trace Triggers: These work differently than instruction triggers.

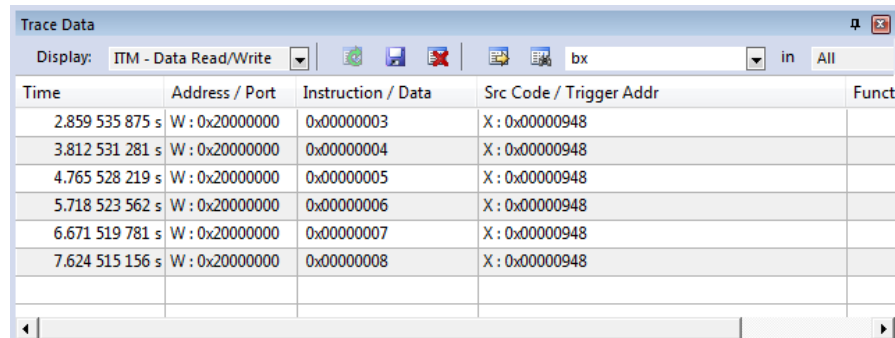
These Triggers capture memory accesses and display the in the Trace Data window. The trace system must be in SWV mode and not ETM mode. Unselect ETM Trace Enable. It is useful to use Trace data filters.



Trace Commands are used to manage triggers. This includes Trace Commands listed below.

1. **TraceDataPoint:** A data read and/or write to a specified address is recorded in the Trace Data window. Recorded are the timestamp, variable address, data and the address of the instruction causing the read or write.
Documentation: www.keil.com/support/man/docs/uv4/uv4_cm_tracedatapoint.htm
2. The **TraceAccessPoint** command is similar to TraceDataPoint.
www.keil.com/support/man/docs/uv4/uv4_cm_traceaccesspoint.htm

This window was created with the command **tdp counter**. It displays writes to the variable counter in the blinky program. Note the Display selection of ITM – Data Read/Write. Otherwise there will be many SysTick frames.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Funct
2.859 535 875 s	W : 0x20000000	0x00000003	X : 0x00000948	
3.812 531 281 s	W : 0x20000000	0x00000004	X : 0x00000948	
4.765 528 219 s	W : 0x20000000	0x00000005	X : 0x00000948	
5.718 523 562 s	W : 0x20000000	0x00000006	X : 0x00000948	
6.671 519 781 s	W : 0x20000000	0x00000007	X : 0x00000948	
7.624 515 156 s	W : 0x20000000	0x00000008	X : 0x00000948	

Using Instruction Trace Triggers:

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, ULINKpro 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 CoreSight behavior.

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.

23) Setting Trace Triggers:

Setup the ETM Example program: This uses the variable counter you created :

1. Stop the program  and stay in Debug mode.
2. You can set/unset trace triggers in either the Disassembly or a C/C++ source window.
3. In main.c, near line 52 is the statement: **counter++**; Double-click on this line to show it in the Disassembly window.
4. A window similar to this will open – your addresses might be different. This is with Optimization set to Level 0.
5. The C statement counter++; consists of instructions starting at 0x0940 and ending at 0x0948 in this case.

Setup the Trace Triggers:

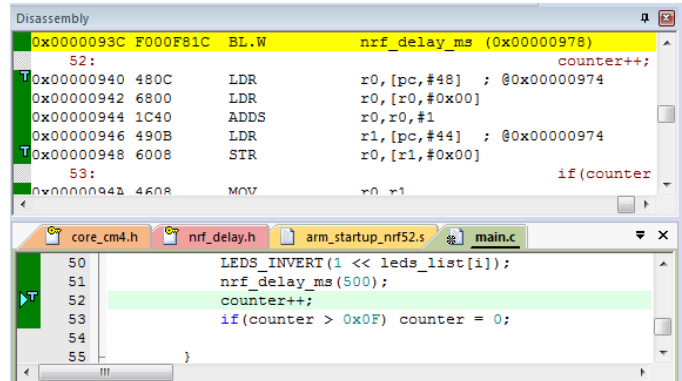
1. In the Disassembly window, right click on line 0x0940 as shown here:
2. Select Insert Tracepoint and select **TraceRun (ETM)**. A cyan **T** will appear as shown:
3. Right-click the line 0x0948 as shown:
4. Select Insert Tracepoint and select **TraceSuspend (ETM)**. A cyan **T** will appear as shown:
5. Clear the Trace Data window  for convenience. This is an optional step but makes the process clearer and easier to understand.
6. Click RUN and after a few seconds click STOP.
7. Filter the SysTick exceptions out by selecting ETM – Code Exec in the Display in the Trace Data window:
Display: **ETM - Code Exec**
8. Examine the Trace Data window as shown below:
9. You can see below where each TRACE RUN started on 0x940 and stopped on 0x94A: There is a skid of one instruction. All other frames are discarded. If there was an exception or a function call, all executed instructions would be included in the trace. The trace turns on at the specified address and captures all instructions until it is turned off. It is not a system where it records only addresses between those specified limits.
10. In the Command window, enter TL and press Enter. The two Trace points are displayed.
11. 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.










Trace Data				
Display: ETM - Code Exec				
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
13.821 591 219 s	X : 0x0000094A	MOV r0,r1	if(counter > 0x0F) counter = 0;	main
	TRACE RUN			
	X : 0x00000940	LDR r0,[pc,#48] ; @0x00000974	counter++;	main
	X : 0x00000942	LDR r0,[r0,#0x00]		main
	X : 0x00000944	ADDS r0,r0,#1		main
	X : 0x00000946	LDR r1,[pc,#44] ; @0x00000974		main
	X : 0x00000948	STR r0,[r1,#0x00]		main
14.298 088 719 s	X : 0x0000094A	MOV r0,r1	if(counter > 0x0F) counter = 0;	main
	TRACE RUN			
	X : 0x00000940	LDR r0,[pc,#48] ; @0x00000974	counter++;	main

24) Code Coverage (CC): CC is provided by ETM Instruction Trace:

1. In main.c, various colours are used to indicate Code Coverage.
2. Examine the Disassembly and main.c windows. Scroll and notice the different color blocks in the left margin: Most will be green. In the Disassembly window you can easily find examples of the other three colours.
3. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not. This is complete as opposed to system that use only PC Samples.

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.

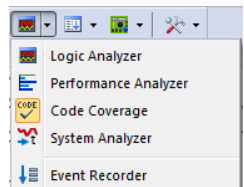
In the window above, why was 0x08D0 never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

Code Coverage tells what assembly instructions were executed. Or why the branches at 0x08CE or 0x08E0 are not complete ? 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 ULINKpro. CC can be saved in the .gecov protocol.

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

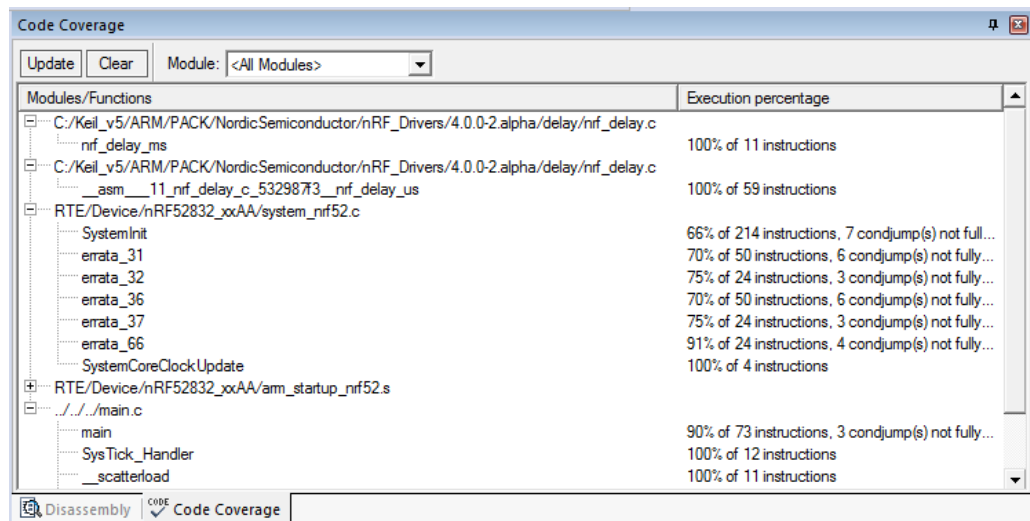
Code Coverage is captured by ETM.

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.

Click the Update button to update the CC window.



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.

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

```
\\nrf52832_xxaa\C:\Keil_v5\ARM\PACK\NordicSemiconductor\NRF_Drivers\4.0.0-  
2.alpha\delay\nrf_delay.c\nrf_delay_ms - 100% (11 of 11 instructions executed)  
7 condjump(s) or IT-block(s) not fully executed  
\\nrf52832_xxaa\RTE\Device\NRF52832_xxAA\system_nrf52.c\errata_31 - 70% (35 of 50 instructions executed)  
6 condjump(s) or IT-block(s) not fully executed  
\\nrf52832_xxaa\RTE\Device\NRF52832_xxAA\system_nrf52.c\errata_32 - 75% (18 of 24 instructions executed)  
3 condjump(s) or IT-block(s) not fully executed  
\\nrf52832_xxaa\RTE\Device\NRF52832_xxAA\system_nrf52.c\errata_36 - 70% (35 of 50 instructions executed)  
6 condjump(s) or IT-block(s) not fully executed  
\\nrf52832_xxaa\RTE\Device\NRF52832_xxAA\system_nrf52.c\errata_37 - 75% (18 of 24 instructions executed)  
3 condjump(s) or IT-block(s) not fully executed  
\\nrf52832_xxaa\RTE\Device\NRF52832_xxAA\system_nrf52.c\errata_66 - 91% (22 of 24 instructions executed)  
4 condjump(s) or IT-block(s) not fully executed
```

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

```
\\nrf52832_xxaa\C:\Keil_v5\ARM\PACK\NordicSemiconductor\NRF_Drivers\4.0.0-  
2.alpha\delay\nrf_delay.c\nrf_delay_ms - 0% (0 of 11 instructions executed)  
NE | 0x00000978 nrf_delay_ms:  
NE | 0x00000978 B501 PUSH {r0,lr}  
NE | 0x0000097A E006 B 0x0000098A  
NE | 0x0000097C 9800 LDR r0,[sp,#0x00]  
NE | 0x0000097E 1E40 SUBS r0,r0,#1  
NE | 0x00000980 9000 STR r0,[sp,#0x00]  
NE | 0x00000982 F24030E7 MOVW r0,#0x3E7  
NE | 0x00000986 F7FFFD45 BL.W __asm__11_nrf_delay_c_532987f3__nrf_delay_us (0x00000414)  
NE | 0x0000098A 9800 LDR r0,[sp,#0x00]
```

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)

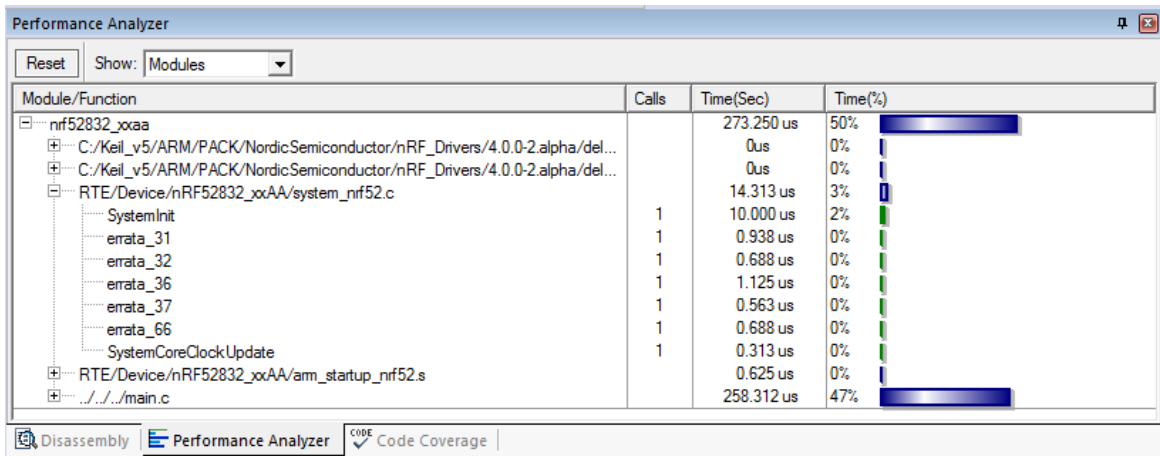
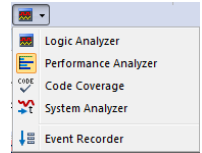
TIP: If you type in **coverage** in the command window, various options will be displayed in the info bar below.

25) Performance Analysis (PA): PA is provided by ETM Instruction Trace:


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 μ Vision 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. Shown is the number of calls and percentage of total time in this short run from RESET to the beginning of main().

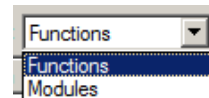





Module/Function	Calls	Time(Sec)	Time(%)
nrF52832_xxaa		273.250 us	50%
C:/Keil_v5/ARM/PACK/NordicSemiconductor/nRF_Drivers/4.0.0-2.alpha/del...		0us	0%
C:/Keil_v5/ARM/PACK/NordicSemiconductor/nRF_Drivers/4.0.0-2.alpha/del...		0us	0%
RTE/Device/nRF52832_xxaa/system_nrf52.c		14.313 us	3%
SystemInit	1	10.000 us	2%
errata_31	1	0.938 us	0%
errata_32	1	0.688 us	0%
errata_36	1	1.125 us	0%
errata_37	1	0.563 us	0%
errata_66	1	0.688 us	0%
SystemCoreClockUpdate	1	0.313 us	0%
RTE/Device/nRF52832_xxaa/arm_startup_nrf52.s		0.625 us	0%
./././main.c		258.312 us	47%

5. Click on the RUN icon. 
6. 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.



7. Select Functions from the pull down box as shown here and notice the difference.



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

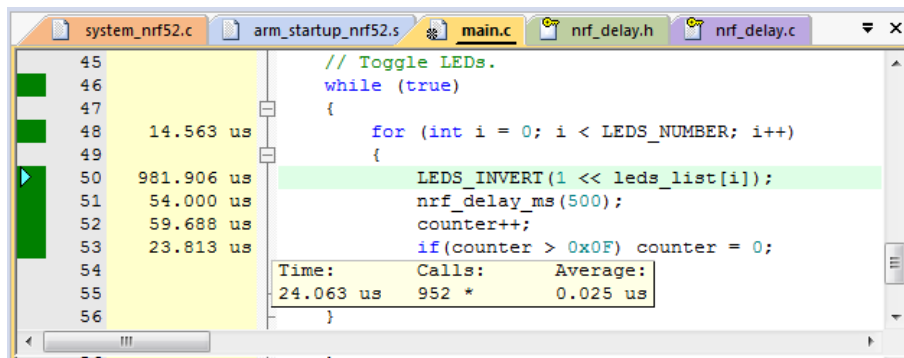
26) Execution Profiling:

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.

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 using ETM instruction trace.
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.

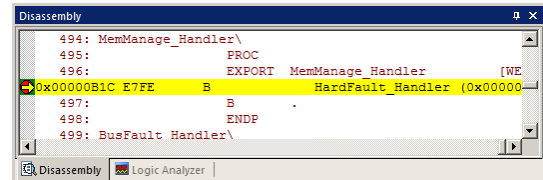


27) In-the-Weeds Example: (your addresses might not be the same as those shown below)

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 quickly 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 displayed might be different in your project.

The Hard Fault handler exception vector is found in the file `arm_startup_nrf52.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 be halted.



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 0x0496 in the Disassembly window or near line 353 in `arm_startup_nrf52.s`
3. Set a breakpoint at this point. This will be the instruction B – branch to itself. A red circle will appear.
4. We will use the BX near line 79 in the assembly routine in `nrf_delay_us` in `nrf_delay.h` as our Hard Fault trigger.

TIP: 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.

5. Set a breakpoint on the instruction **BNE loop** near line 78 in `nrf_delay.h` just before the BX.
6. Click RUN. The program will halt at the BNE instruction. Remove the breakpoint.
7. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
8. 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.
9. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
10. Scroll to the bottom of the Trace Data. In the Trace Data window, you will find the BX instruction near the end. When the function tried to return, the bogus value of `lr` caused a Hard Fault. The HardFault is displayed at the end.
11. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skip breakpoints.
12. Click on Single Step. You will now see the Hard Fault branch instruction as shown in the bottom screen:

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

Exception Entry: Note the Exceptions 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 353... and select TraceHalt. When Hard Fault is reached, trace collection will halt but the program will keep executing for testing Hard Fault handlers.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
1,196,730,605,656 s	X: 0x00000476	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x00000478	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x0000047A	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x0000047C	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x0000047E	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x00000480	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x00000482	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
	X: 0x00000484	NOP	NOP	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
1,196,730,605,719 s	X: 0x00000486	*BNE _asm_11_nrf_delay...	BNE loop	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
1,196,730,605,781 s	X: 0x00000488	BX lr	BX LR	_asm_11_nrf_delay_c_532987f3_nrf_delay_us
1,196,730,745,000 s		Exception Entry - SysTick		
1,196,730,745,625 s		Exception Exit - SysTick		
1,196,730,745,875 s		Exception Return		
D 1,196,731,557,375 s		Exception Entry - HardFault		

13. Exit Debug mode.

14. This is the end of the exercises.

TRACE RUN				
1,196,731,557,375 s	X: 0x00000496	B	HardFault_Handler (0x00000496)	B . HardFault_Handler

28) Serial Wire Viewer and ETM Trace Summary:

We have several debug systems implemented in Arm 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 μ Vision.
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 μ Vision via a ULINK2 or ULINKpro.

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 ULINKpro and RTX, DMA exceptions will display in the Event Viewer.

29) Document Resources:

See www.keil.com/Nordic

Books:

1. **NEW!** **Getting Started with MDK 5:** Obtain this free book here: www.keil.com/mdk5/
2. There is a good selection of books available on ARM: www.arm.com/support/resources/arm-books/index.php
3. μ Vision contains a window titled Books. Many documents including data sheets are located there.
4. **The Definitive Guide to the Arm Cortex-M0/M0+** by Joseph Yiu. Search the web for retailers.
5. **The Definitive Guide to the Arm Cortex-M3/M4** by Joseph Yiu. Search the web for retailers.
6. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
7. MOOC: Massive Open Online Class: University of Texas: <http://users.ece.utexas.edu/~valvano/>

Application Notes:

1. **NEW!** Arm Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer using Keil MCB170: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with μ Vision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with μ Vision <https://launchpad.net/gcc-arm-embedded>
8. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
9. Cortex-M Processors for Beginners: <http://community.arm.com/docs/DOC-8587>
10. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
11. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
12. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
13. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
14. **NEW!** ARMv8-M Architecture Technical Overview www.keil.com/appnotes/files/apnt_291.pdf
15. **NEW!** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.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>

30) Keil Products and contact information: See www.keil.com/Nordic

Keil Microcontroller Development Kit (MDK-ARM™) for Cortex-M processors:

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

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

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

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

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

You can use Segger J-Link on the nrF52 board. For ETM support, a ULINKpro is needed.

Call Keil Sales for more details on current pricing. All products are available.

For the Arm University program: go to www.arm.com/university Email: university@arm.com

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

Keil RTX™ Real Time Operating System

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

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

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

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



For more information:

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

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

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

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

