

Introduction:

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_280.asp

The purpose of this lab is to introduce you to the STMicroelectronics Cortex™-M7 processor using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) and the on-board ST-Link V2 Debug Adapter.

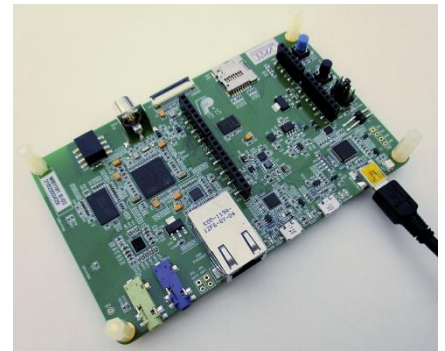
Keil MDK supports and has examples for most ST ARM processors. See www.keil.com/st.

Linux: Cortex-A processors running Linux, Android and no OS are supported by ARM DS-5™. www.arm.com/ds5.

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. MDK is free for STM32 F0/L0 Cortex-M0 processors. See www.keil.com/st or contact Keil Sales.

Why Use Keil MDK ? MDK provides these features particularly suited for Cortex-M processor users:

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is turn-key "out-of-the-box".
2. ARM Compiler 5 and ARM Compiler 6 (LLVM) are included. GCC support.
3. **STM32CubeMX** compatible.
4. **Keil Middleware:** Network, USB, Flash File and Graphics.
5. **NEW!** Event Recorder for Keil Middleware, RTX and User programs.
6. MISRA C/C++ support using PC-Lint. www.gimpel.com
7. Serial Wire Viewer data trace and ETM instruction trace capability included.
8. **NEW!** Power Measurement with Keil ULINKplus: www.keil.com/ulinkplus
9. **Compiler Safety Certification Kit:** www.keil.com/safety/
10. **NEW ! Run-Time System for Functional Safety Applications: FuSa RTS** www.keil.com/fusa-rts
11. TÜV certified. ISO 26262 ASIL D, IEC62304 Class C, IEC 61508 SIL 3, EN50128 SIL 4
12. Keil RTX is included with MDK. RTX has a BSD or Apache 2.0 license and source code is provided. www.keil.com/RTX and https://github.com/ARM-software/CMSIS_5
13. Two RTX Kernel Awareness windows. They are updated in real-time. **NEW!** FreeRTOS is now supported.
14. Keil Technical Support is included for one year and is easily renewable. This helps you get your project completed.
15. Affordable perpetual and term licensing. Contact Keil sales for pricing, options and current special offers.



This document details these features:

1. Serial Wire Viewer (SWV) data trace using ST-Link V2. Real-time tracing updated while the program is running.
2. Real-time Read and Write to memory locations for Watch, Memory and Peripherals windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
3. Six Hardware Breakpoints (they can be set/unset on-the-fly) and four Watchpoints (also known as Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS that updates while your program is running.
5. A DSP example program using ARM CMSIS-DSP libraries. www.arm.com/cmsis
6. ETM instruction trace including Performance Analysis, Code Coverage and a Hard Fault error demonstration.
7. Create a project with STM32CubeMX, Keil Software Packs, MDK Middleware or ST Standard Peripheral Libraries.

Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and timestamps. This information comes from the ARM CoreSight™ debug module integrated into STM32 CPU. Memory contents and variable values are displayed in real-time and these can be modified on-the-fly.

Embedded Trace Macrocell (ETM): (includes Code Coverage (gcov) and Performance Analysis)

1. ETM records and displays all instructions that were executed. This is very useful for debugging program flow problems such as "going into the weeds" and "how did I get here?" ETM requires a ULINKpro.

1. Keil Evaluation Software: MDK5	3
2. Keil Software Download and Installation:	3
3. On-board ST-Link V2 Debug Adapter:	3
4. Example Programs	3
5. Getting Started MDK 5 Manual:	3
6. STMicroelectronics evaluation boards:	3
7. μ Vision Software Pack Download and Install Process:	4
8. Software Pack Version Selection and Manage Run-Time Environment:	5
9. CoreSight Definitions:	6
10. Installing the ST-Link USB Drivers:	7
11. Testing the ST-Link V2 Connection:	7
12. <i>Blinky</i> example using the STM32F4 Discovery board:	8
13. Hardware Breakpoints:	8
14. Call Stack & Locals window:	9
15. Watch and Memory windows and how to use them:	10
16. View Variables Graphically with the Logic Analyzer (LA):	11
17. Watchpoints: <i>Conditional Breakpoints</i> (Access Breakpoints)	12
18. ITM (Instrumentation Trace Macrocell):	13
19. Printf with ITM (Instrumentation Trace Macrocell):	14
20. RTX_Blinky example: Keil RTX RTOS:	15
21. RTX Kernel Awareness using RTX Viewer:	16
22. Logic Analyzer: View variables real-time in a graphical format:	17
23. DSP Sine Example using ARM CMSIS-DSP Libraries	18
24. Keil Middleware: Network (TCP/IP), Flash File, USB, Graphics	21
25. Creating your own project from scratch:	22
26. Creating your own RTX RTOS project from scratch:	25
27. Adding a new Thread to your RTX project:	26
28. Using Event Viewer to examine the timing of RTX:	27
29. Using STM32CubeMX to create a simple Blinky Program:	28
30. Serial Wire Viewer (SWV) and how to use it:	30
1) Data Reads and Writes	30
2) Exceptions and Interrupts	31
3) PC Samples (program counter samples)	32
31. Serial Wire Viewer (SWV) Configuration:	33
32. ETM Trace and its benefits:	34
33. Serial Wire Viewer and ETM summary:	44
34. Document Resources:	45
35. Keil Products and contact information:	46

Notes on using this document:

1. The necessary example source files are available here: www.keil.com/appnotes/docs/apnt_280.asp
2. MDK 5.17 and Software Pack STM32F7xx_DFP 2.3.0 were used in the exercises in this document.
3. The ST-Link V2 interfaces very well with Keil μ Vision and its performance is quite good including SWV.

1) Keil Evaluation Software: MDK 5

MDK 5 uses Software Packs to distribute processor specific software, examples and middleware. MDK 5 Core is first installed and you then download the Software Packs you require from the web. They can also be imported manually. You no longer need to wait for the next version of MDK or install patches to get the latest processor specific files.

STM32CubeMX provides software in MDK 5 format consistent with Software Packs. Keil Middleware supports STM32F7.

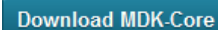
This document uses MDK 5.25 or later.

Summary of Keil software installation: This is a three step process:

- A. Download and install the MDK Core file. This is done in Step 2 below.
- B. Download and install the appropriate Software Pack for the processor you are using. This is done on the next page.
- C. In addition, you need to download and install the examples used in this tutorial. See the next page:


2) Keil MDK Core Software Download and Installation:

1. Download MDK Core from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5
3. We recommend you use the default folders for this tutorial. We will use C:\MDK\ for the examples.
4. If you install MDK or the examples into different folders, you will have to adjust for the folder location differences.
4. You do not need any debug adapters: just the Discovery board, a USB cable and MDK 5 installed on your PC.
5. For the ETM examples, you will need a ULINK_{pro} and a board with a 20 pin CoreSight ETM connector.



MDK Licensing:

1. You can use the evaluation version (MDK-Lite) for this lab. No license is needed.
2. You can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible:
3. This gives you access to the Keil Middleware as well as unlimited code size compilation. Contact Keil sales to extend this license for evaluation purposes.



3) The on-board ST-Link V2 Debug Adapter:

The on-board ST-Link V2 is used exclusively in this lab except for the ETM exercises. Instructions on configuring and testing the ST-Link V2 are on page 7. The Discovery board has no external debug adapter connector.

You connect to the on-board ST-Link V2 with a USB cable connected to CN14 and to your PC which also powers the board.

ETM Examples: An STM32756G-EVAL board with a ULINK_{pro} is used or similar with the 20 pin ETM connector.

4) Example Programs: These are obtained as described on the next page (page 4).

5) Getting Started MDK 5: Obtain this useful book here: www.keil.com/mdk5/.

6) STMicroelectronics evaluation boards:


This tutorial supports two STM32F7 boards:

1. **STM32F746G-Discovery:** As pictured on the front page. This tutorial uses this board except for the ETM exercises. Blinky and Middleware examples are provided.
2. **STM32756G_EVAL:** This is used only in the ETM examples in this tutorial. Blinky and Middleware examples are provided. This has a 20 pin connector for ULINK_{pro}. ETM has 4 bit signal + a clock + JTAG/SWD signals.

7) µVision Software Pack and Examples Download and Install Process:

1) **Start µVision and open Pack Installer:** (after the first MDK install is complete and if you are connected to the Internet, µVision and Software Packs will automatically startup. Otherwise, follow Steps 1 and 2 below)

1. Connect your computer to the Internet. This is normally needed to download the Software Packs.

2. Start µVision by clicking on its desktop icon. 

3. Open the Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.

1. This window opens up: Select the Boards tab. Type **discovery** in the Search box to filter the listings:

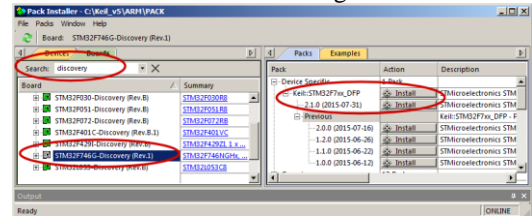
4. Select STM32F746G-Discovery as shown here: You can also select individual processors under the Devices tab.

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

TIP: If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or




to refresh once you have connected to the Internet.



2) Install The STM32F7 Software Pack:

1. Click on the Packs tab. Initially, the Software Pack ARM::CMSIS is installed by default.
2. Select Keil::STM32F7xx_DFP as shown above and click on Install. The latest Pack will download and install to C:\Keil_v5\ARM\Pack\Keil\ST\ by default. This download can take two to four minutes.

3. Its status will then be indicated by the “Up to date” icon: 

3) Install the Blinky MDK 5 Example:

2. Select STM32F746G-Discovery in the Boards tab.

3. Select the Examples tab:

4. Select CMSIS-RTOS Blinky Discovery:


5. Select Copy  Copy:

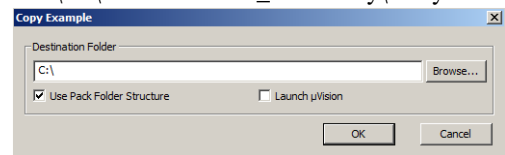
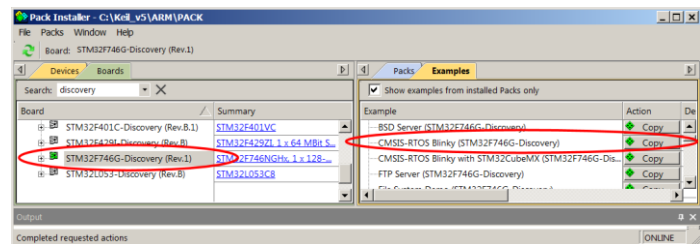
6. The Copy Example window below opens up: Select Use Pack Folder Structure: Unselect Launch µVision:

7. Type in C:\ as shown to the right: Click OK to copy into C:\MDK\Boards\ST\STM32F746G_Discovery\. If you are using the STM32756G_EVAL board, select it.

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

8. Close the Packs Installer. Open it any time by clicking on its icon.


9. If a window states the Software Packs have changed, select Yes. 

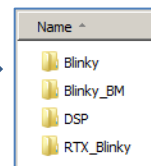


4) Install the Blinky_BM, RTX_Blinky and DSP Examples from Keil.com:

1. Obtain the example software zip file from www.keil.com/appnotes/docs/apnt_280.asp.

2. Extract this into the directory C:\MDK\Boards\ST\STM32F746G_Discovery\

3. The Blinky_BM, DSP and RTX_Blinky folders will be created with the Blinky folder: 



TIP: An Update icon means there is an updated Software Pack available for download. 


Select Packs/Check for Updates or  in the Pack Installer to refresh this list. It is not done automatically.

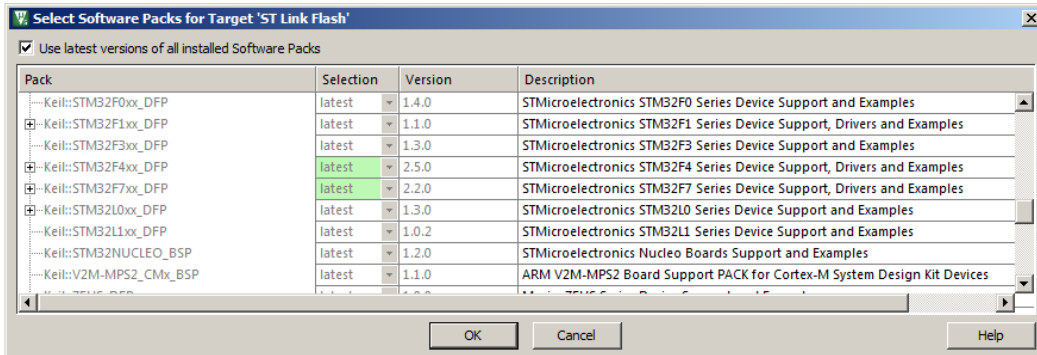
TIP: If you look in the directory C:\Keil_v5\ARM\Pack\Keil\STM32F7xx_DFP\2.1.0\MDK\Boards\ST\, you will find examples for two ST boards. This is a read-only version for backup purposes. Use only the projects you copied over from the Examples tab in the Pack Installer window to the directory you chose. In this tutorial we are using C:\MDK.

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

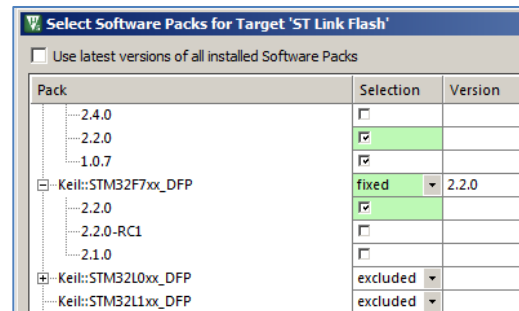
Select Software Pack Version:

This µVision utility provides the ability to choose various software pack versions installed in your computer. You can select the versions you want to use.


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

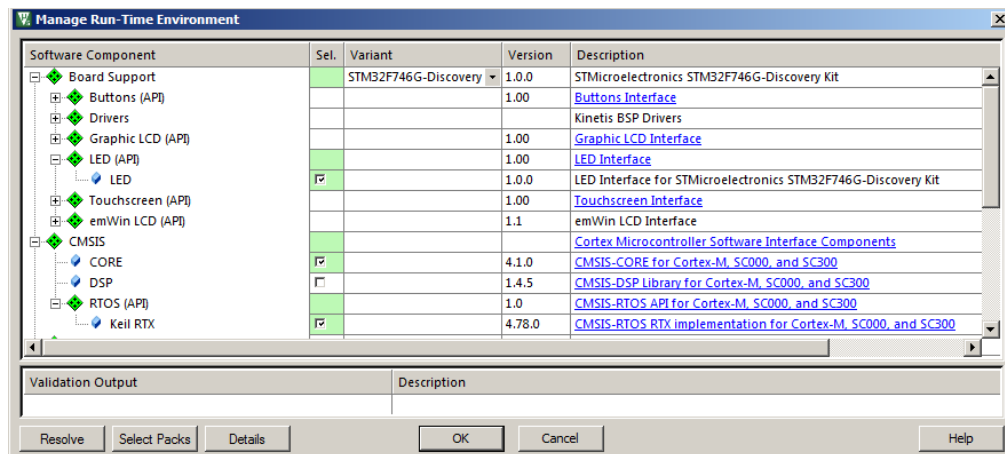


4. Expand the header Keil::STM32F7xx_DFP. Note the various versions visible. You may see different versions.
5. Select excluded and see the options as shown:
6. If you wanted to use V 2.1.0, you would select fixed and then select the check box opposite 2.1.0.
7. Select Use latest versions...
8. Close this window.



Manage Run-Time Environment:

1. Select Project/Open Project. Open the project:
C:\MDK\Boards\ST\STM32F746G_Discovery\Blinky_BM\Blinky.uvprojx.
2. Click on the Manage RTE icon:  The next window opens: This includes the board support package (BSP), Keil Middleware, selected open source software, RTX and CMSIS drivers for various peripherals.
3. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project.
4. Do not make any changes. Click Cancel to close this window.



There are three main methods to create your own µVision projects:

- 1) **STM32Cube.** This configures your processor and exports a µVision project in MDK 5 format. See Page 28. STM32CubeMX can be downloaded from www.st.com/stm32cubemx/
- 2) **Standard Peripheral Libraries** from ST. STM32CubeF7. Contains extensive examples and source code for Keil MDK 5. These libraries are also available from www.st.com/stm32cubemx/
- 3) **µVision Software Packs, examples and Keil Middleware.** A Software Pack includes examples and files that you can use. See Page 21 and www.keil.com/pack/doc/STM32Cube/General/html/index.html

ELF/DWARF: The ARM compiler produces a .axf file which is ELF/DWARF compliant. µVision can load similar compiler output such as from GCC with all debug information visible. You can also use GCC as your compiler of choice in µVision.

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

Cortex-M0 and Cortex-M0+ have only features 1) through 3) plus 11 and 12 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 STMicroelectronics datasheet to determine its feature set.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except Boundary Scan is not possible. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup. See page 4, 2nd picture. The SWJ box must be selected in ULINK2/ME or ULINKpro. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDIO shares the same pin as SWO. The SWV data normally comes out the SWO pin.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **DAP:** Debug Access Port. This is a component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. µVision uses the DAP to update memory, watch and RTOS kernel awareness windows in real-time 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 source code stubs are needed. You do not need to configure or activate DAP. µVision configures DAP when you select a function that uses it. Do not confuse this with CMSIS-DAP which is an 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 TDIO.
7. **Trace Port:** A 4 bit port that ULINKpro uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by µVision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINKpro provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and 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 ULINKpro. 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 RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. STM32 Cortex-M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 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.
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.

10) Installing ST-Link V2 USB Drivers: (this is not necessary if the test below passes)





ST-Link V2 USB drivers initially must be installed manually. Windows may attempt to install them but this might not work.

Installing the ST-Link USB Drivers: *This step normally needs to be done just once !*

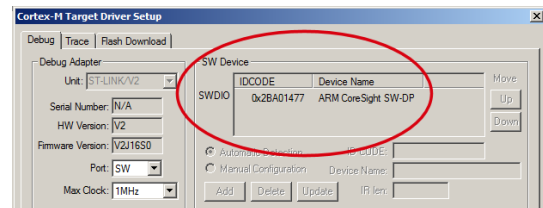
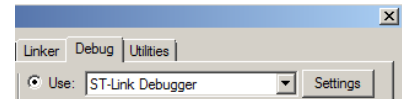
1. Do not have the Discovery board USB port connected to your PC at this time.
2. The USB drivers must be installed manually by executing stlink_winusb_install.bat. This file is found in C:\Keil_v5\ARM\STLink\USBDriver\. Double-click on this file. The drivers will install.
3. Plug in the Discovery board to USB CN14. The USB drivers will now finish installing in the normal fashion.

Upgrading the ST-Link V2 Firmware: The ST-Link V2 firmware updater utility ST-LinkUpgrade.exe is located here: C:\Keil_v5\ARM\STLink\. It is a good idea to upgrade the ST-Link firmware. Find this file and double click on it. It is easy to use. It will check and report the current firmware version. Version V2.J24.M10 is the current version supplied with MDK 5.16. If you experience trouble especially with Serial Wire Viewer (SWV), update to the latest drivers and firmware.

11) Testing the ST-Link V2 Connection: (Optional)

1. Start  if it is not already running. Select Project/Open Project.
2. Connect the Discovery board to your PC with a USB cable as shown on the first page of this tutorial.
3. If the ST-Link USB drivers are installed correctly, you should hear the usual USB connected dual-tone. If not, you might have to install the drivers manually. See the directions above.
4. Two red LEDs will light: LD7 and LD2 (PWR)
5. Select the Blinky project C:\MDK\Boards\ST\STM32F4-Discovery\Blinky.uvprojx. (or to any project)
6. Select STM32F407 Flash as shown here: 
7. Select Target Options  or ALT-F7 and select the Debug tab: 
8. Click on Settings: and the window below opens up: If an IDCODE and Device name is displayed, ST-Link is working. You can continue with the tutorial on the next page. Click OK twice to return to the μ Vision main menu.
9. A number in the SN: box means μ Vision is connected to the ST-Link adapter.
10. If nothing or an error is displayed in this SW Device box, this **must** be corrected before you can continue. See the instructions above: Installing the ST-Link USB Drivers:
11. Once you see a proper display, your ST-Link USB drivers are installed properly. Click OK twice to exit the Target Options windows and continue to the next page.

TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window. ST-Link V2 does not support the JTAG mode.



LED LD7 indication:

LED is blinking RED: the start of USB enumeration with the PC is taking place but not yet completed.

LED is RED: communication between the PC and ST-LINK/V2 is established (end of enumeration). μ Vision is not connected to ST-Link (i.e. in Debug mode).

LED is GREEN: μ Vision is connected in Debug mode and the last communication was successful.







LED is blinking GREEN/RED: data is actively being exchanged between the target and μ Vision.

LED is off, except for a brief RED flash while entering Debug mode and a brief flash when clicking on RUN happens when the SWV trace is enabled in μ Vision.

No Led: ST-LINK/V2 communication with the target or μ Vision has failed. Cycle the board power to restart.

12) Blinky example program using the STM32F7 Discovery board:

We will connect a Keil MDK development system using real target hardware using the built-in ST-Link V2 debug adapter.

1. Start μ Vision by clicking on its desktop icon.  Connect your PC to the board with a USB cable to CN14.
2. Select Project/Open Project. Open C:\MDK\Boards\ST\STM32F746G_Discovery\Blinky_BM\Blinky.uvprojx
3. By default, the ST-Link is selected. If this is the first time you have run μ Vision and the Discovery board, you might have to install the USB drivers. See the configuration instructions on the previous page.
4. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
5. Program the STM32 flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
6. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
Note: You only need to use the Load icon to download to FLASH and not for RAM operation if it is chosen.
7. Click on the RUN icon.  Note: you stop the program with the STOP icon. 





**The LED LD1 on the STM32F7 Discovery board will blink.
Press the blue *USER* button and it will blink slower.**

Now you know how to compile a program, program it into the STM32 processor Flash, run it and stop it !

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

13) Hardware Breakpoints:

The STM32F7 has six hardware breakpoints that can be set or unset on the fly while the program is running.

1. With Blinky running, in the Blinky.c window, click on a darker grey block in the left margin on a line in main() in the while loop. Between around lines 57 through 61 will suffice.
2. A red circle will appear and the program will stop.
3. Note the breakpoint is displayed in both the disassembly and source windows as shown below:
4. You can set a breakpoint in either the Disassembly or Source windows as long there is a gray rectangle indicating the existence of an assembly instruction at that point.
5. Every time you click on the RUN icon  the program will run until the breakpoint is again encountered.
6. You can also click on Single Step (Step In) , Step Over  and Step Out .

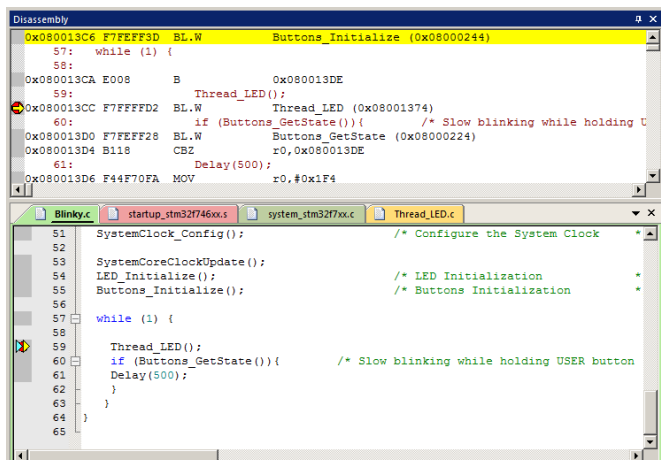
TIP: To single step (Step In) by assembly instruction, click on the Disassembly window to bring it into focus. To step by a C source line, bring any source window into focus.

TIP: A hardware breakpoint does not execute the instruction it is set to. ARM CoreSight breakpoints are no-skid. Your instructions in Flash are not substituted or modified. These are rather important features for efficient software development.

Remove all breakpoints when you are done for the next exercise by clicking on them again.

TIP: You can delete the breakpoints by clicking on them or selecting Debug/Breakpoints (or Ctrl-B) and selecting Kill All. Click on Close to return.

TIP: You can view the breakpoints set by selecting Debug/Breakpoints or Ctrl-B.




14) Call Stack + Locals Window:

Local Variables:

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

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

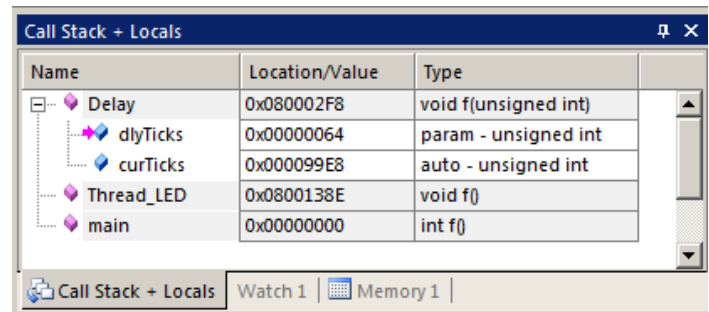
1. Run and Stop Blinky . Click on the Call Stack + Locals tab.
2. You will probably be stopped in the Delay() function as shown here in the Call Stack + Locals window:




The contents of the local variables are displayed as well as names of active functions. Each function name will be displayed as it is called from the function before it or from an interrupt or exception.

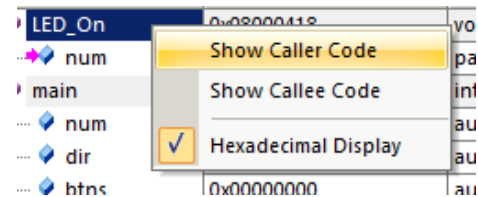
When a function exits, it is removed from the list.

The first called function is at the bottom of this table.

This table is active only when the program is stopped.



3. Click on the Step In icon  or F11:
4. Note the function different functions displayed as you step through them. If you get trapped in the Delay function, use Step Out  or Ctrl-F11 to exit it faster.
5. Click numerous times on Step In and see other functions.
6. Right click on a function name and try the Show Callee Code and Show Caller Code options as shown here:
7. Click on the StepOut icon  to exit all functions to return to main().



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

TIP: This is standard “Stop and Go” debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables updated in real-time while the program is running. No additions or changes to your code are required. Update while the program is running is not possible with local variables because they are usually stored in a CPU register. They must be converted to global or static variables so they always remain in scope.

If you have a ULINKpro and ETM trace, you can see a record of all the instructions executed. The Disassembly and Source windows show your code in the order it was written. The ETM trace shows it in the order it was actually executed. ETM additionally provides Code Coverage, Performance Analysis and Execution Profiling.

Changing a local variable to a static or global normally means it is moved from a CPU register to RAM. CoreSight can view RAM but not CPU registers when the program is running.

Call Stack:

The list of stacked functions is displayed when the program is stopped as you have seen. This is useful when you need to know which functions have been called and what return data is stored on the stack.

TIP: You can modify a local variable value when the program is stopped.

TIP: You can access the Hardware Breakpoint table by clicking on Debug/Breakpoints or Ctrl-B. This is also where Watchpoints (also called Access Points) are configured. You can temporarily disable entries in this table.







Selecting Debug/Kill All Breakpoints deletes Breakpoints but not Watchpoints.

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

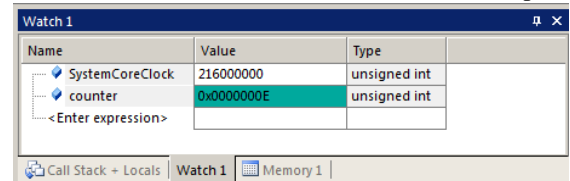
The Watch and Memory windows will display updated variable values in real-time. It does this through the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to modify values into the Memory window.

Watch window:

Add a global variable: Recall the Watch and Memory windows can't see local variables unless stopped in their function. Make local variables static or global to make them visible in a watch or Memory window.

1. Stop the processor  and exit Debug mode. .
2. In Blinky.c, declare a global variable (I called it counter) near line 20 like this: **unsigned int counter = 0;**
3. Add the statements `counter++;` and `if (counter > 0x10) counter = 0;` as shown here near line 60:
4. Select File/Save All or click .
5. Click on Rebuild. .
6. Enter Debug mode.  Click on RUN . You can set Watch and Memory windows while the program is runs.
7. In Blinky.c, right click on the variable **counter** and select Add counter to ... and select Watch 1. Watch 1 will open if needed and counter will be displayed as shown here:
8. **counter** will increment until 0x10 in real-time.
9. Note some values of counter will be missed because the watch and Memory windows are updated periodically. Press the Blue User button to demonstrate this.

```
59 Thread_LED();
60 counter++;
61 if (counter > 0x10) counter = 0;
62 if (Buttons_GetState()) {
```



Name	Value	Type
SystemCoreClock	216000000	unsigned int
counter	0x0000000E	unsigned int
<Enter expression>		

TIP: Enter SystemCoreClock and this displays the CPU clock frequency.

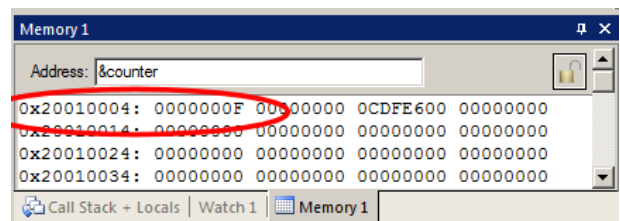
TIP: You can also block **counter**, click and hold and drag it into a Watch or Memory window.

TIP: Make sure View/Periodic Window Update is selected.

10. You can also enter a variable manually by double-clicking under Name or pressing F2 and using copy and paste or typing the variable. Use the View/Symbols window to enter a variable fully qualified.

Memory window:

1. Right-click on **counter** and similarly enter it into the Memory 1 window.
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 is shown (0x2001_0004).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **counter** is now displayed as a 32 bit value.
6. Both the Watch and Memory windows are updated in real-time without stealing CPU cycles.
7. You can modify counter in the Memory window with a right-click with the mouse cursor over the data field and select Modify Memory.



Address	Value
0x20010004: &counter	0000000F
0x20010014:	00000000
0x20010024:	00000000
0x20010034:	00000000

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

How It Works:

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



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

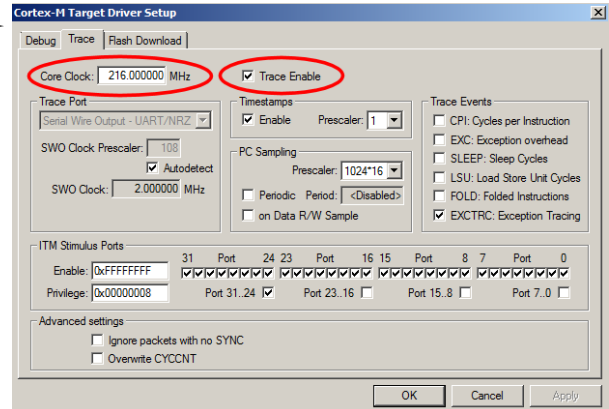
16) View Variables Graphically with the Logic Analyzer (LA):

We will display the global variable counter you created earlier in the Logic Analyzer. No code stubs in the user code will be used. This uses the Serial Wire Viewer (SWV) and therefore does not steal CPU cycles.


1. Stop the processor  and exit Debug mode. 

Configure Serial Wire Viewer (SWV):

2. Select Target Options  or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window. Confirm SW is selected. SW selection is mandatory for SWV. ST-Link uses only SW. Select the Trace tab.
3. In the Trace tab, select Trace Enable. Unselect Periodic and select EXCTRC. Set Core Clock: to 216 MHz. Everything else is set as shown here: 
4. Click OK once to return to the Debug tab.

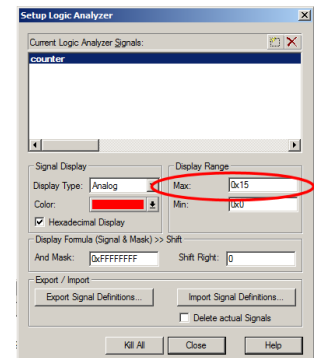


Configure the Logic Analyzer:


1. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 

TIP: You can configure the LA while the program is running.

2. Click on the Blinky.c tab. Right click on **counter** and select Add counter to... and then select Logic Analyzer. You can also Drag and Drop or enter it manually.
3. In the Logic Analyzer window, click on the Select box and the LA Setup window appears as shown here:
4. With **counter** selected, set Display Range Max: to 0x15 as shown here:
5. Click on Close.



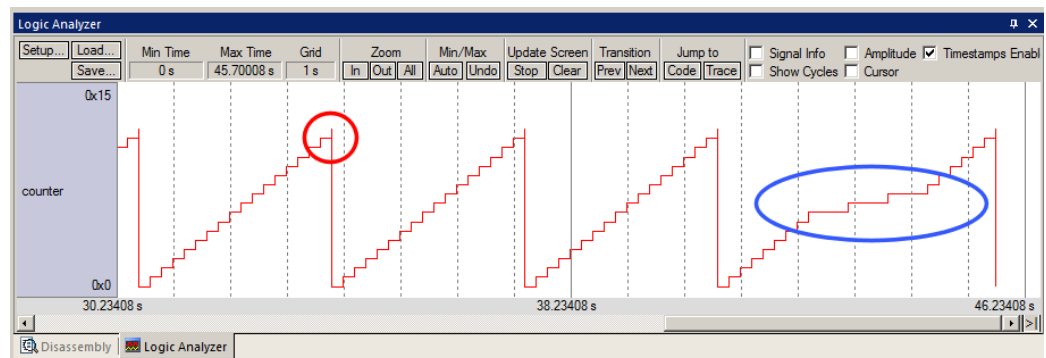
Run the Program: Note: The LA can be configured while the program is running.


- 1) Click on Run.  Click on Zoom Out until Grid is about 1 second.
- 2) The variable **counter** will increment to 0x10 (decimal 16) and then is set to 0.

TIP: If you do not see a waveform, exit and re-enter Debug mode to refresh the LA. You might also have to repower the Discovery board. Confirm the Core Clock: value is correct.

TIP: You can show up to 4 variables in the Logic Analyzer. These variables must be global, static or raw addresses such as *((unsigned long *)0x20000000).


- 3) Press the blue User button and see counter increment slower as shown in the blue circle below:
- 4) Select Signal Info, Show Cycles, Amplitude and Cursor to see the measuring capabilities of the LA. You can stop the LA by clicking on the Stop icon in the Update Screen box.



- 5) Note counter briefly reaches 0x11 since the test is after the increment.
- 6) When you are ready to continue, start the Update Screen.
- 7) Stop the CPU. 
- 8) Click on Close.

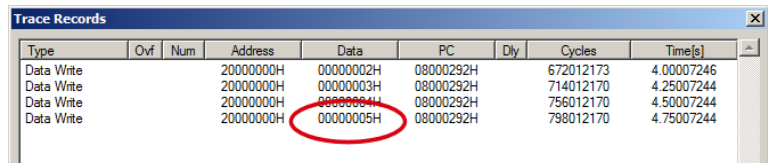
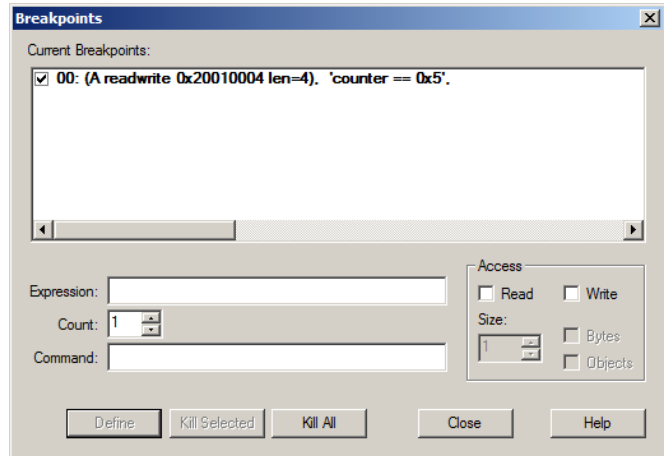
17) Watchpoints: Conditional Breakpoints: This does not need or use Serial Wire Viewer:

Recall STM32 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. The STM32 also have four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same comparators as Watchpoints in its operations and they must be shared. This means in μ Vision you must have two variables free in the Logic Analyzer to use Watchpoints. Watchpoints are also referred to as Access Breakpoints.

1. Use the same Blinky configuration as the previous page. Stop the program if necessary.  Stay in debug mode.
2. We will use the global variable `counter` you created in Blinky.c to explore Watchpoints.
3. The SWV Trace does not need to be configured for Watchpoints. However, we will use it in this exercise.
4. The variable `counter` should be still entered in the Logic Analyzer from the last exercise on the previous page.
5. Select Debug in the main μ Vision window and select Breakpoints or press Ctrl-B.
6. Select both the Read and Write Access. In the Expression box enter: "`counter == 0x5`" without the quotes.
7. Click on Define and it will be accepted as shown here: Click on Close.
8. Enter the `counter` to the Watch 1 window if it is not already listed.
9. Open Debug/Debug Settings and select the Trace tab. Check "on Data R/W sample" and uncheck EXTRC.
10. Click on OK twice. Open the Trace Records window.



11. Double click in the Trace Records to clear it.
12. Click on RUN.
13. You will see `counter` change in the Logic Analyzer as well as in the Watch window.
14. When `counter` equals 0x5, the Watchpoint will stop the program.
15. Note the data writes in the Trace Records window shown below. 0x5 is in the last Data column. Plus the address the data written to and the PC of the write instruction. This is with the ST-Link. A ULINK2 will show the same window. A ULINKpro or a J-Link (black case) will show a slightly different display.
16. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window. Not all are currently implemented in μ Vision.
17. To repeat this exercise, click on RUN.
18. When you are finished, stop the program, click on Debug and select Breakpoints (or Ctrl-B) and Kill the Watchpoint.
19. Leave Debug mode.



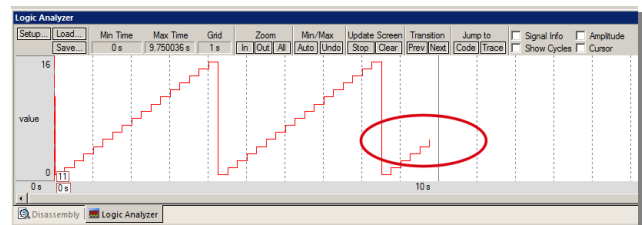
Type	Ofs	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	00000002H	08000292H		672012173	4.00007246
Data Write			20000000H	00000003H	08000292H		714012170	4.25007244
Data Write			20000000H	00000004H	08000292H		756012170	4.50007244
Data Write			20000000H	00000005H	08000292H		798012170	4.75007244

TIP: You cannot set Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

TIP: To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint. You should delete the old one by highlighting it and click on Kill Selected or try the next TIP:

TIP: The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it.



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



Shown above right is the Logic Analyzer window displaying the variable `counter` trigger point of 0x5. This is three runs.






18) ITM (Instrumentation Trace Macrocell): ITM uses Serial Wire Viewer:

ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into μ Vision for display in its Debug (printf) Viewer window. This exercise uses the Blinky_BM project.

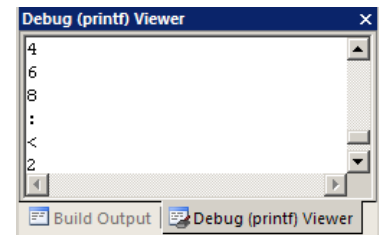
1. Stop the program if it is running  and exit Debug mode. .
2. Use the same Blinky_BM project we have been using. SWV must be configured.
3. Add this code to Blinky.c. A good place is near line 19, just after the #include "LED.h".
4. In the main function in Blinky.c after the second Delay(200); near line 72, enter these lines:

```
#define ITM_Port8(n)    (*((volatile unsigned char *) (0xE0000000+4*n)))

ITM_Port8(0) = counter + 0x30;    /* displays value in ASCII */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```

5. Select File/Save All or click . Rebuild the source files .
6. Open Select Target Options  or ALT-F7 and select the Debug tab, and then the Trace tab.
7. The Serial Wire Viewer should be still configured. Use 216 MHz for the Core Clock.
8. Select ITM Port 0. ITM Stimulus Port "0" enables the Debug (printf) Viewer.
9. Click OK twice to return to the main μ Vision menu. Enter Debug mode .
10. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN .
11. In the Debug (printf) Viewer you will see the ASCII of counter display:
12. As counter is incremented its ASCII character is displayed.

TIP: You can easily save ITM information to a file. For information see www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm



Trace Records

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 and Exception frames.
3. Right click inside the Trace Records window and unselect Exceptions to filter these out.

How does this work ?

You can see the ITM writes and Data writes (value being displayed in the LA).

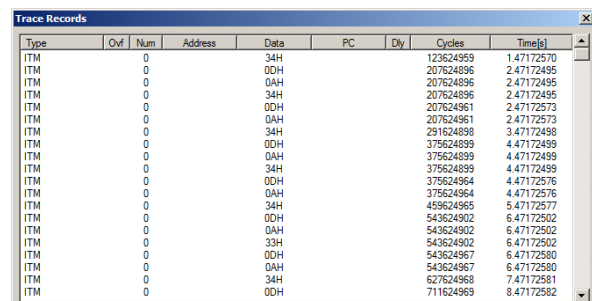
1. ITM 0 frames (Num column) are our ASCII characters from **counter** with carriage return (0D) and line feed (0A) as displayed the Data column.
2. All these are timestamped in both CPU cycles and time in seconds.
3. When you are done, stop the processor and exit Debug mode.

ITM Notes

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 and to your PC via the Serial Wire Output (SWO) pin.

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.








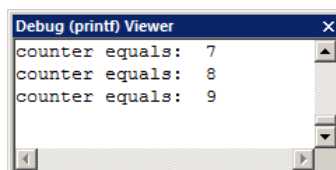
TIP: ITM_SendChar is a useful function you can use to send ITM characters. It is found in the header core.CM7.h.

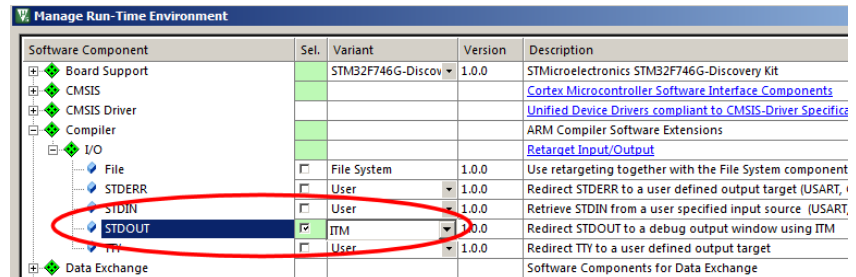


Type	Ovl	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM	0	34H	123624959	147172570				
ITM	0	00H	207624896	247172495				
ITM	0	0AH	207624896	247172495				
ITM	0	34H	207624896	247172495				
ITM	0	00H	207624961	247172573				
ITM	0	0AH	207624961	247172573				
ITM	0	34H	291624898	347172498				
ITM	0	00H	375624899	447172499				
ITM	0	0AH	375624899	447172499				
ITM	0	34H	375624899	447172499				
ITM	0	00H	375624964	447172576				
ITM	0	0AH	375624964	447172576				
ITM	0	34H	459624965	547172577				
ITM	0	00H	543624902	647172502				
ITM	0	0AH	543624902	647172502				
ITM	0	33H	543624902	647172502				
ITM	0	00H	543624967	647172580				
ITM	0	0AH	543624967	647172580				
ITM	0	34H	627624968	747172581				
ITM	0	00H	711624969	847172582				

19) printf with ITM (Instrumentation Trace Macrocell): ITM uses Serial Wire Viewer:

It is easy to incorporate printf using ITM and the μ Vision utility Manage Runtime Environment.

1. Stop the program if it is running  and exit Debug mode. 
2. Comment out all six C source lines you entered in Blinky on the previous page. They are not needed here.
3. Open the Manage Run-Time Environment utility.  This window opens:
4. Expand Compiler...I/O as shown.
5. Select STDOUT and ITM:
6. All the blocks should be green. If not, click on the Resolve button.
7. Click OK to close this window.
8. The file retarget_io.c will be added to your project in the project window under the Compiler group.
9. In Blinky.c, near line 63 just after the if (counter>.... Line, add this line: `printf("counter equals: %d\n", counter);`
10. Select File/Save All or click .
11. Rebuild the source files .
12. Enter Debug mode . Click on RUN .
13. The values of counter is displayed as seen here: 



TIP: You can easily save ITM information to a file. See www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm


Obtaining a character typed into the Debug printf Viewer window:

It is possible for your program to do this with the function ITM_ReceiveChar found in core.CM7.h.

See https://www.keil.com/pack/doc/CMSIS/Core/html/group__i_t_m__debug__gr.html.

A working example can be found in the File System Demo in Keil Middleware. Download this using the pack installer utility.

Read-Only Source Files:

Some 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 source files. This can cause difficult to solve problems. Most of these files will not need any modification.

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

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

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






20) RTX_Blinky Example Program with Keil RTX RTOS: A Stepper Motor example

Keil provides RTX, a full feature RTOS. RTX is included as part of Keil MDK including source. This example explores the RTX RTOS project. MDK will work with any RTOS. An RTOS is just a set of C functions that gets compiled with your project. RTX comes with a BSD type license and source code is provided with all versions of MDK.

NOTE: RTX_Blinky supplied with this document is an RTX_Blinky that has four threads simulating a stepper motor and blinks one LED.

RTX and all its components are located here: C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.3.0\CMSIS\RTOS

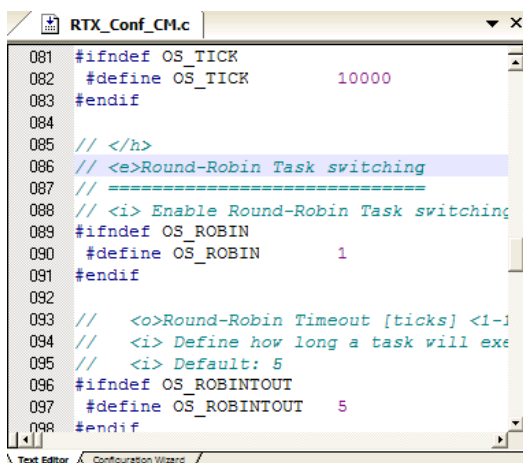
You must have copied RTX_Blinky to C:\MDK\Boards\ST\STM32F746G_Discovery as described on page 4.

1. With μ Vision in Edit mode (not in debug mode): Select Project/Open Project.
2. Open the file C:\MDK\Boards\ST\STM32F746G_Discovery\Blinky.uvprojx.
3. This project is pre-configured for the ST-Link V2 debug adapter.
4. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
7. The green LED LD1 will be toggled by Thread 1 (phaseA).
8. Click on STOP .

We will explore the operation of RTX with the Kernel Awareness windows.

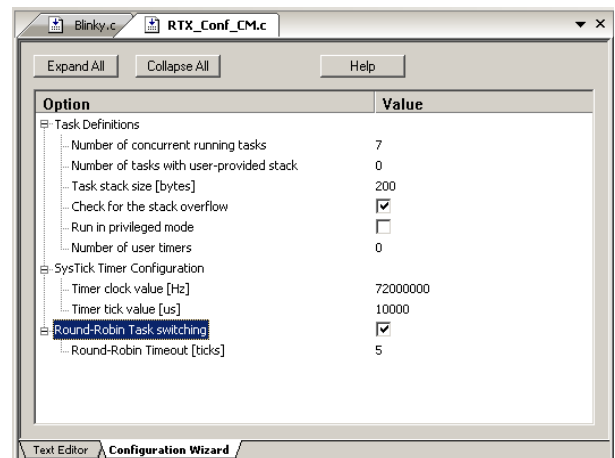
The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open if needed.
2. Click on the Configuration Wizard tab at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. Changing an attribute in one tab changes it in the other automatically. You should save a modified window.
6. You can create Configuration Wizards for any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a `</h>` or `<i>`.
See www.keil.com/support/man/docs/uv4/uv4_ut_configwizard.htm for instructions.



```
081 #ifndef OS_TICK
082 #define OS_TICK      10000
083 #endif
084
085 // </h>
086 // <e>Round-Robin Task switching
087 // =====
088 // <i> Enable Round-Robin Task switching
089 #ifndef OS_ROBIN
090 #define OS_ROBIN      1
091 #endif
092
093 // <o>Round-Robin Timeout [ticks] <1-1
094 // <i> Define how long a task will ex
095 // <i> Default: 5
096 #ifndef OS_ROBINTOUT
097 #define OS_ROBINTOUT  5
098 #endif
```

Text Editor: Source Code

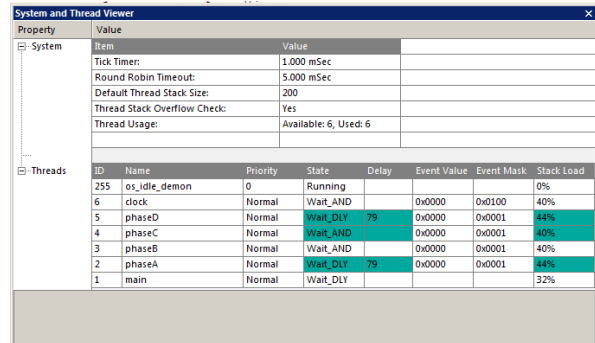


Configuration Wizard

21) RTX Kernel Awareness using Serial Wire Viewer (SWV):

Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides a Task Aware window for RTX. Other RTOS companies also provide awareness plug-ins for μ Vision.

1. Run RTX_Blinky by clicking on the Run icon.
2. Open Debug/OS Support and select System and Thread Viewer and the window on the right opens up. You might have to grab the window and move it into the center of the screen. These values are updated in real-time using the same read write technology as used in the Watch and Memory windows.



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: 6, Used: 6

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
6	clock	Normal	Wait_AND		0x0000	0x0100	40%
5	phaseD	Normal	Wait_DLY	79	0x0000	0x0001	44%
4	phaseC	Normal	Wait_AND		0x0000	0x0001	40%
3	phaseB	Normal	Wait_AND		0x0000	0x0001	40%
2	phaseA	Normal	Wait_DLY	79	0x0000	0x0001	44%
1	main	Normal	Wait_DLY				32%

Important TIP: View/Periodic Window Update must be selected !

3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.

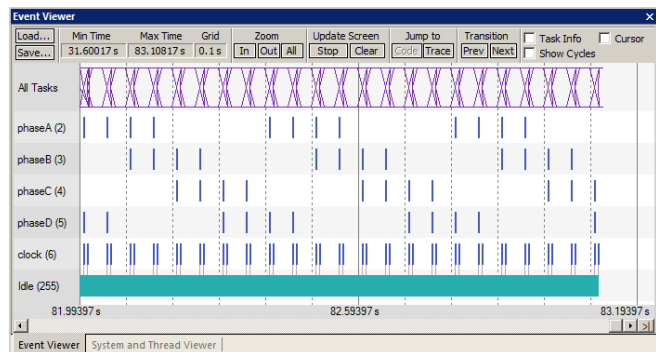
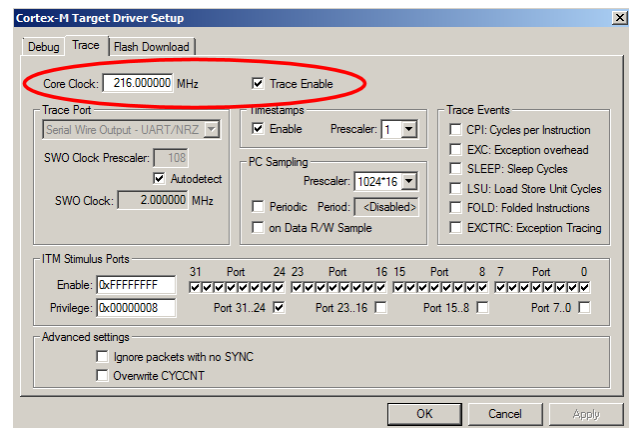
RTX Viewer: Configuring Serial Wire Viewer (SWV):

We must activate Serial Wire Viewer to get the Event Viewer working.

1. Stop the CPU and exit debug mode.
2. Click on the Target Options icon next to the target box. Select the Debug tab.
3. Click the Settings box next to ST-Link Debugger.
4. In the Debug window, make sure Port: is set to SW and not JTAG. SWV works only with SW mode.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 216 MHz.
7. Select Trace Enable.
8. Unselect the Periodic and EXCTRC boxes as shown:
9. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer. It is slightly intrusive.
10. Click on OK twice to return to μ Vision.

The Serial Wire Viewer is now configured !

11. Enter Debug mode and click on RUN.
12. Select "Tasks and System" tab: the display is updated.
13. Click on the Event Viewer tab.
14. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 0.1 sec by clicking on the Zoom ALL and + and - icons.
15. You can use the cursor settings to determine various timings of the Threads. You can select Stop Update Screen without stopping the CPU.



TIP: If Event Viewer doesn't work, open up the Trace Records and confirm there are good ITM 31 frames present. Is Core Clock correct ? This project is running at 216 MHz.

The data is updated while the program is running. No instrumentation code needs to be inserted into your source. You will find this feature very useful ! Remember, RTX with source code is included with all versions of MDK.

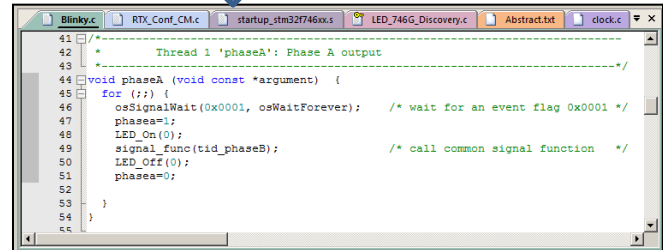
TIP: You can use a ULINK2, ULINK-ME, ULINKpro, ST-Link V2 or J-Link for these RTX Kernel Awareness windows.

22) Logic Analyzer Window: View variables real-time in a graphical format:

µVision has a graphical Logic Analyzer window. Up to four variables can be displayed in real-time using the Serial Wire Viewer in the STM32. RTX_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.

1. Leave the program running.
2. Four global variables `unsigned int phasea` through `unsigned int phased` are in Blinky.c as shown here:
3. Each of four threads `phasea` through `phased` represents one phase of a stepping motor driver. This is `phasea` and is the one that toggles the LED LD1.
4. Note the variable `phasea` being switched. You can examine the other three threads.
5. We will enter these for global variables into the Logic Analyzer for examination.

```
24 unsigned int phasea=0;
25 unsigned int phaseb=0;
26 unsigned int phasec=0;
27 unsigned int phased=0;
```



Enter the Variables into the Logic Analyzer (LA):

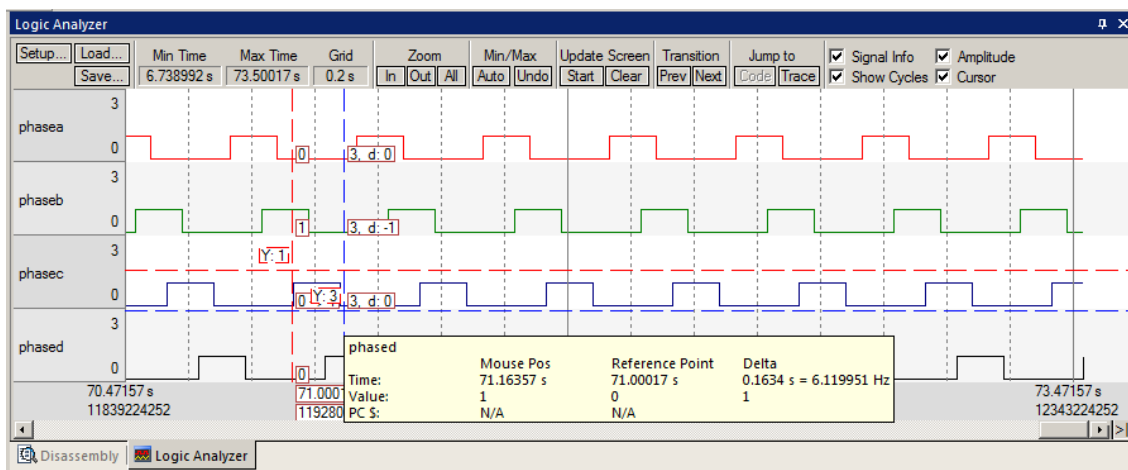
6. Click on the Blinky.c tab. Right click on `phasea`, select Add 'phasea' to... and finally select Logic Analyzer. Phasea will be added to the LA.
7. Repeat for `phaseb`, `phasec` and `phased`. These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.



TIP: If you can't get these variables entered into the LA, make sure the Trace Config is set correctly. The Serial Wire Viewer *must* be configured correctly in order to enter variables in the LA.

The Logic Analyzer can display static and global variables, structures and arrays.

It can't see locals: just make them static or global. To see peripheral registers read or write to them and enter them in the LA.

8. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
9. Click on Close to go back to the LA window.
10. Using the All, OUT and In buttons set the range to 0.2 second or so. Move the scrolling bar to the far right if needed.



11. Select Signal Info and Show Cycles. Click to mark a place move the cursor to get timings. Place the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled phasec:
12. Click on Stop in Update Screen to stop (and start) the data collection.
13. Stop the CPU  and exit debug mode. 

TIP: You can also enter these variables into the Watch and Memory windows to display and change them in real-time.





TIP: You can view signals that exist mathematically in a variable and not available for measuring in the outside world. This has proved to be very useful in debugging tricky problems.

23) DSP SINE example using ARM CMSIS-DSP Libraries:

ARM CMSIS-DSP libraries are offered for ARM Cortex-M series processors. DSP libraries are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS. See www.arm.com/cmsis for more information. CMSIS is an acronym for Cortex Microcontroller Software Interface Standard. CMSIS is an ARM standard.

This example creates a sine wave with noise added, and then the noise is filtered out. The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer.

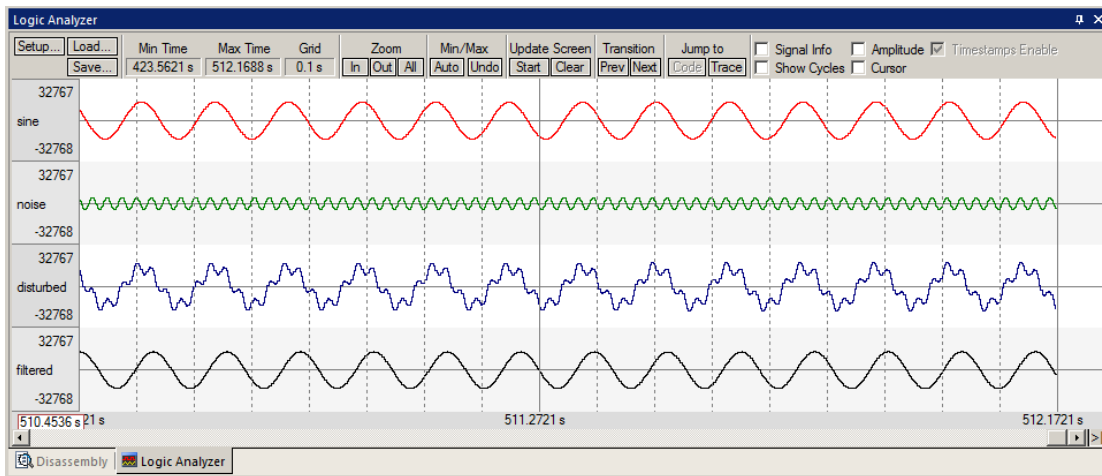
This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. RTX source code is provided.

1. Open the project file sine: C:\MDK\Boards\ST\STM32F746G_Discovery\DSP\sine.uvproj
2. Build the files.  There will be no errors or warnings.
3. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
4. Click on the RUN icon.  Open the Logic Analyzer window. 
5. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom Out for an appropriate display. Displayed are 4 global variables: sine, noise, disturbed and filtered.

TIP: If one variable shows no waveform, disable the ITM Stimulus Port 31 in the Trace Config window.

TIP: Serial Wire Viewer can be limited since all data comes out one pin. A Keil ULINK_{pro} handles SWV information much better since it uses Manchester mode at a higher frequency than the ST-Link. Use of the 4 bit trace port is even better.

6. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.



7. Open the Trace Records window and the Data Writes to the four variables are listed as shown here:
8. Leave the program running.
9. Close the Trace Records window.

TIP: The ULINK_{pro} trace display is different and the program must be stopped to update it.

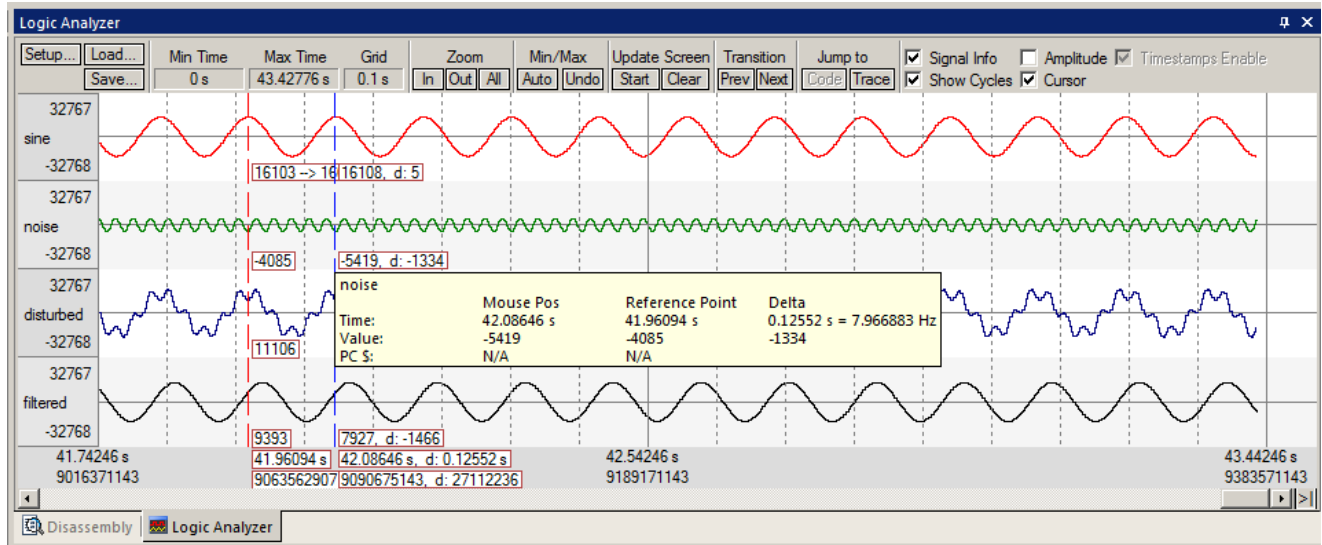
The Watch 1 window will display the four variables updating in real time as shown below:

Watch 1		
Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short
<Enter expression>		

Trace Records							
Type	Ofv	Num	Address	Data	PC	Dly	Cycles
Data Write			2001001AH	F5AFH			133242735769
Data Write			2001001CH	1912H			616.86451745
Data Write			2001001EH	3E43H			133242744210
Data Write			2001001EH	3D6CH			616.86456553
Data Write			2001001CH	1912H			133242756147
Data Write			2001001AH	F06EH			616.86461179
Data Write			20010018H	2004H			133243004800
Data Write			20010018H	1C83H			616.86576296
Data Write			2001001AH	ECB3H			133243016281
Data Write			2001001CH	0936H			616.86581612
Data Write			2001001EH	3DEDH			133243026290
Data Write			2001001EH	3CCFH			616.86586245
Data Write			2001001CH	0936H			133243035659
Data Write			2001001AH	EADDH			616.86590583
Data Write			20010018H	18E6H			133243278664
Data Write			20010018H	152FH			616.86703085
Data Write			2001001AH	EB17H			133243288709
Data Write			2001001CH	0046H			616.86707736
Data Write			2001001EH	3C84H			133243297110
Data Write			2001001EH	3B43H			616.86711625
Data Write			2001001EH				133243309047
Data Write			2001001EH				616.86717151
Data Write			2001001EH				133243358444
Data Write			2001001CH				616.86832613
Data Write			2001001AH				133243569925
Data Write			2001001AH				616.86837928
Data Write			20010018H				133243579934
Data Write			20010018H				616.86842562
Data Write			20010018H				133243589303
Data Write			20010018H				616.86846900
Data Write			2001001AH				133243831552
Data Write			2001001AH				616.86849900
Data Write			2001001CH				133243841597
Data Write			2001001CH				616.86963702
Data Write			2001001EH				133243849998
Data Write			2001001EH				616.86967592
Data Write			2001001EH				133243861935
Data Write			2001001EH				616.86973118
Data Write			2001001EH				133244110672
Data Write			2001001EH				616.87088274

Signal Timings in Logic Analyzer (LA):

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



RTX Tasks and System:

5. Click on Start in the Update Screen box to resume the collection of data.
6. Open Debug/OS Support and select System and Thread Viewer. A window similar to below opens up. You probably have to click on its header and drag it into the middle of the screen.
7. Set a breakpoint in four of the threads in DirtyFilter.c by clicking in the left margin on a grey area.
8. Click on Run and the program will stop at each thread in turn and the Thread Viewer window will be updated accordingly. Here, I set a breakpoint in the disturb_gen thread:
9. Clearly, below you can see that disturb_gen was running when the breakpoint was activated.
10. Remove the breakpoints. Click on them or enter Ctrl-B and select Kill All. Then click on Close.

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

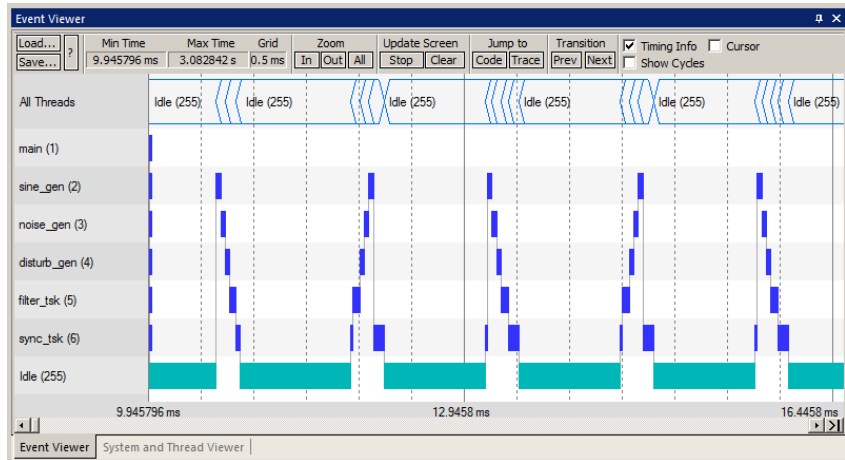
TIP: Recall this window uses the CoreSight DAP read and write technology to update this window. Serial Wire Viewer is not used and is not required to be activated for this window to display and be updated.

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

System and Thread Viewer							
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: 6, Used: 6				
Threads		ID	Name	Priority	State	Delay	Event Value
255	os_idle_demon	0	Ready				
6	sync_tsk	Normal	Wait_AND				
5	filter_tsk	Normal	Wait_AND	65514	0x0000	0x0001	40%
4	disturb_gen	Normal	Running	65504	0x0000	0x0001	8%
3	noise_gen	Normal	Wait_AND	65534			40%
2	sine_gen	Normal	Wait_AND	1014	0x0000	0x0001	40%
1	main	Normal	Wait_DLY				32%

Event Viewer:

1. Stop the program. Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might occur. If you like – you can leave the LA loaded with the four variables to see what the Event Viewer will look like.
2. Select Debug/Debug Settings.
3. Click on the Trace tab.
4. Enable ITM Stimulus Port 31. Event Viewer uses Port 31 to collect its information.
5. Click OK twice.
6. Click on RUN.
7. Open Debug/OS Support and select Event Viewer. The window here opens up:
8. Note the main(1) thread. This screen is scrolled to the beginning after RESET. Main() runs only once.



Important TIP: If SWV trace fails to work after this change, exit Debug, cycle the board power and re-enter Debug mode.

TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some bandwidth. Using a ULINKpro in either Manchester mode or using the 4 bit trace port will help.

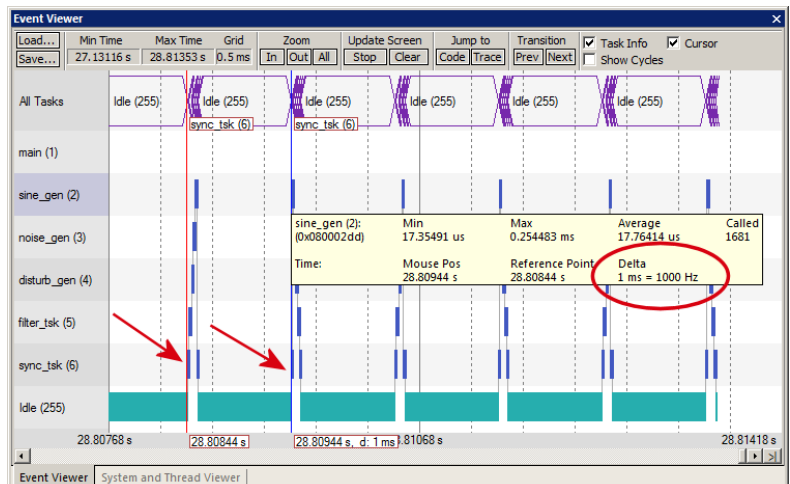
ULINKpro is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows.

ULINKpro uses the faster Manchester format than the slower UART mode that ST-Link, ULINK2 and J-Link uses.

ULINKpro can also use the 4 bit Trace Port for faster operation for SWV. The Trace Port is mandatory for ETM trace.

9. Note on the Y axis each of the 5 running tasks plus the idle daemon. Each bar is an active task and shows you what task is running, when and for how long.
10. Click Stop in the Update Screen box.
11. Click on Zoom In so three or four tasks are displayed.
12. Select Cursor. Position the cursor over one set of bars and click once. A red line is set at the first arrow:
13. Move your cursor to the right over the next set (where the second arrow is) and total time and difference are displayed.
14. Note, since you enabled Show Cycles, the total cycles and difference is also shown.
15. Hover your mouse on one of the threads blue block. It will turn yellow and display information about this event as shown below:

TIP: ITM Port 31 enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port if you are not using the Event Viewer.



24) Keil Middleware: Network (TCP/IP), Flash File, USB, Graphics:




















MDK Professional provides commercial grade middleware with extensive capabilities designed for demanding applications.

See www.keil.com/mdk5/middleware/ for more information.

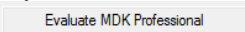
Working examples are provided for various ST boards including STM32F746G-Discovery and STM32F756G-Eval. You can

access and copy these examples using Pack Installer .  Here is the listing for STM32F746G-Discovery:

This window is displayed by selecting STM32F746G-Discovery in the Boards tab:

Packs Examples		
<input checked="" type="checkbox"/> Show examples from installed Packs only		
Example	Action	Description
BSD Client (STM32F746G-Discovery)	 Copy	Example using BSD sockets to send commands to remote server
BSD Server (STM32F746G-Discovery)	 Copy	Example using BSD sockets to accept commands from remote clients
CMSIS-RTOS Blinky (STM32F746G-Discovery)	 Copy	CMSIS-RTOS based Blinky example
CMSIS-RTOS Blinky with STM32CubeMX (STM32F746G-Discovery)	 Copy	CMSIS-RTOS based Blinky example configured with STM32CubeMX
FTP Server (STM32F746G-Discovery)	 Copy	File Server using FTP protocol with SD/MMC Memory Card as storage media
File System Demo (STM32F746G-Discovery)	 Copy	File manipulation example: create, read, copy, delete files on any enabled drive (SD/MMC Card, NOR/NAND Flash, RAM) and format each drive
HTTP Server (STM32F746G-Discovery)	 Copy	Compact Web Server with CGI interface
HTTP Upload (STM32F746G-Discovery)	 Copy	Web Server with CGI interface and SD/MMC Memory Card as storage media
SMTP Client (STM32F746G-Discovery)	 Copy	Example showing how to compose and send emails
SNMP Agent (STM32F746G-Discovery)	 Copy	Example showing how to use a Simple Network Management Protocol (SNMP)
Telnet Server (STM32F746G-Discovery)	 Copy	Command-line Host service example using Telnet protocol
USB Device HID (STM32F746G-Discovery)	 Copy	USB Human Interface Device providing access from PC to board LEDs and push buttons
USB Device Mass Storage (STM32F746G-Discovery)	 Copy	USB Mass Storage Device using RAM as storage media
USB Device Virtual COM (STM32F746G-Discovery)	 Copy	Bridge between PC USB Virtual COM Port and UART port
USB Host Keyboard (STM32F746G-Discovery)	 Copy	Measure example using USB HID Keyboard as input device
USB Host Mass Storage (STM32F746G-Discovery)	 Copy	USB Host file manipulation example: create, read, copy, delete files from USB Mass Storage Device and format the storage device
emWin Example (STM32F746G-Discovery)	 Copy	emWin Graphics simple example
emWin GUI Demo (STM32F746G-Discovery)	 Copy	emWin Graphics Demo example
emWin VNC Server with STM32CubeMX (STM32F746G-Discovery)	 Copy	emWin VNC Server example configured with STM32CubeMX

License:

An MDK Pro license is needed for Keil Middleware. A 7 day one-time license is available in µVision under File/License Management. If you qualify, this button is displayed: 

To obtain a temporary MDK Professional license for evaluation purposes, contact Keil sales as listed on the last page.

Instructions: Each example contains the file abstract.txt which provides instructions. These projects compile and run "out-of-the-box". If you have any questions regarding Keil Middleware operation during your evaluation phase, please contact Keil Technical Support as listed on the last page of this document.

Configuring the examples:

The Middleware examples make extensive use of the Configuration Wizard. This permits easy modifications to various settings with mouse clicks instead of digging through source code.

Selecting the Middleware:

Keil Middleware is selected using the Manage Run-Time Environment utility. This selects the various components you desire and place them into your project. Open the Manage Run-Time Environment utility



and this window is displayed:

Help and Documentation:

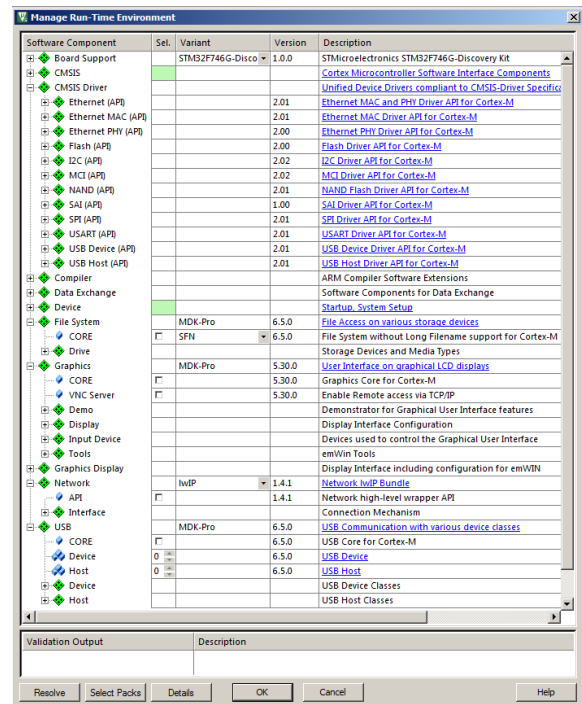
This is available both online and embedded in MDK:

www.keil.com/pack/doc/mw/General/html/index.html

and

C:\Keil_v5\ARM\Pack\Keil\MDK-Middleware

In each of the projects is a file abstract.txt that provides basic instructions and a link to more details.



25) Creating your own MDK 5 project from scratch:

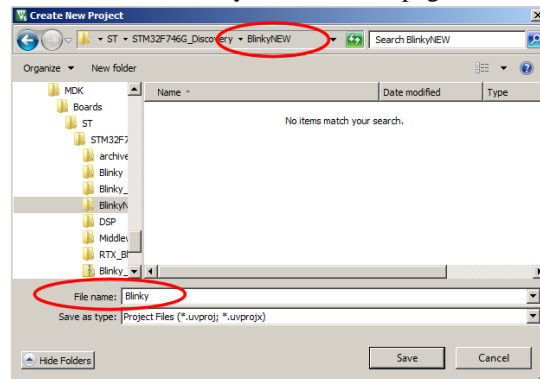
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run a bare Blinky example. It will have an empty main() function so it does not do much. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using an RTOS.

Install the STM32 Software Pack for your processor:

1. Start μ Vision and leave in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Pack for the STM32F7 processor must be installed. This has already been done on page 4.
3. You do not need to copy any examples over.

Create a new Directory and a New Project:

1. Click on Project/New μ Vision Project...
2. In the window that opens, shown here, go in:
C:\MDK\Boards\ST\STM32F746G_Discovery\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj in C:\MDK\Boards\ST\STM32F746G_Discovery\BlinkyNEW\.
7. As soon as you click on Save, the next window opens:



Select the Device you are using:

1. Expand STMicroelectronics, then STM32F7 Series, then STM32F746, then STM32F746NG and then finally select STM32F746NHHx:

TIP: You must select the deepest lever of processor else this will not work correctly.

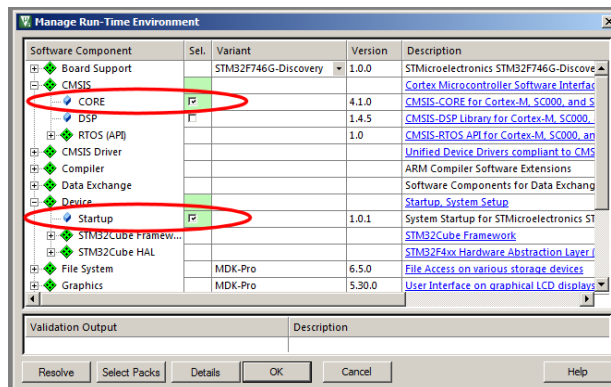
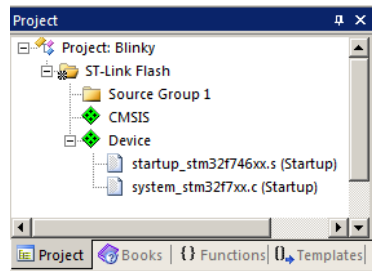
2. Click OK and the Manage Run Time window shown below bottom right opens.

Select the CMSIS components you want:


1. Expand CMSIS and Device as shown below. Select Core and Startup as shown below. They will be highlighted in Green indicating no other files are needed. Click OK.
2. Click on File/Save All or select the Save All icon:
3. The project Blinky.uvproj is now be changed to Blinky.uvprojx.
4. You now have a new project list as shown on the bottom left below: The appropriate CMSIS files you selected have been automatically entered and configured.
5. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
6. Click once on it and change its name to ST-Link Flash and press Enter. The Target selector name will also change.

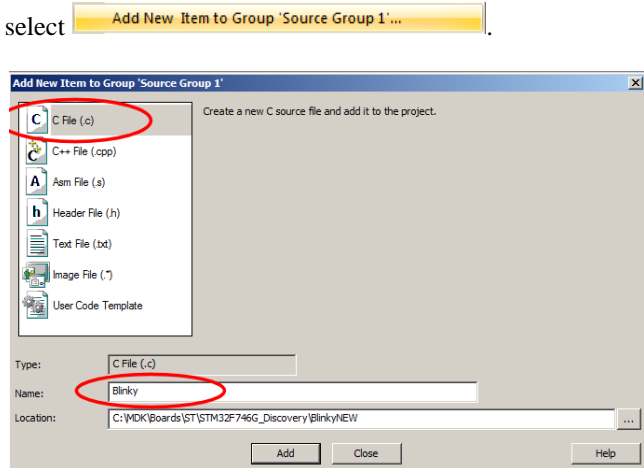
What has happened to this point:

You have created a blank μ Vision project using MDK 5 Software Packs. All you need to do now is add your own source files.





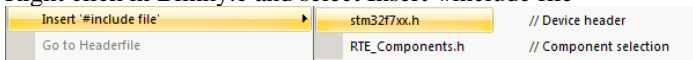
Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select
2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter Blinky.
5. Click on Add to close this window.
6. Click on File/Save All or 
7. Expand Source Group 1 in the Project window and Blinky.c will now display.
8. It will also open in the Source window.



Add Some Code to Blinky.c:

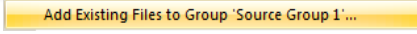
9. Right click in Blinky.c and select Insert '#include file'
10. Select stm32f7xx.h and then repeat for RTE_Components.h. These are added to Blinky.c.
11. In the blank portion of Blinky.c, add the C code below:
12. Click on File/Save All or 
13. Build the files.  There will be no errors or warnings if all was entered correctly.




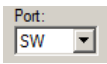



```
unsigned int counter = 0;

/*-----
  MAIN function
  *-----*/
int main (void) {





    while(1) {
        counter++;
        if (counter > 0x0F) counter = 0;
    }
}
```

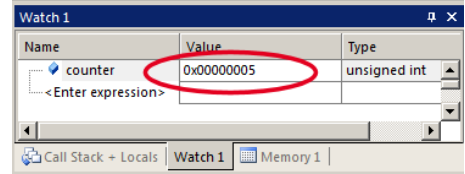
TIP: You can also add existing source files:  No need to at this time.

Configure the Target St Link Flash: *Please complete these instructions carefully to prevent unusual problems...*

1. Select the Target Options icon . Select the **Target** tab.
2. Select Use MicroLIB to optimize for smaller code size. Note the memory locations are entered for your convenience.
3. Select the Settings: icon.
4. Select SW as shown here in the Port: box:  JTAG here will not work with SWV. If your board is connected to your PC, you **must** now see a valid IDCODE and Device Name in the SW Device box.
5. Click on OK **once** to go back to the Target Configuration window. Otherwise, fix the connection problem.
6. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm as shown: Shown is the correct one for the STM32F7 series processors. 
7. Click on OK twice to return to the main menu.
8. Click on File/Save All or 
9. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

Running Your Program:

1. Enter Debug mode by clicking on the Debug icon . Your program will be programmed into the ST Flash.
2. Click on the RUN icon . Note: you stop the program with the STOP icon .
3. No LEDs will blink since there is no source to accomplish this task. You could add such code yourself.
4. Right click on counter in Blinky.c and select Add counter to ... and select Watch 1.
5. counter should be updating as shown here: 
6. You can also set a breakpoint in Blinky.c and the program should stop at this point if it is running properly. If you do this, remove the breakpoint.
7. You should now be able to add your own source code to create a meaningful project.



Since we did not configure any clocks, the CPU is running at the default of 16 mHz.

TIP: The Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip sequential values you know must exist.

TIP: If you want to save or send the project files to someone, you can delete the folder Flash to reduce file size. This folder and its contents are easily reconstructed with a Build.

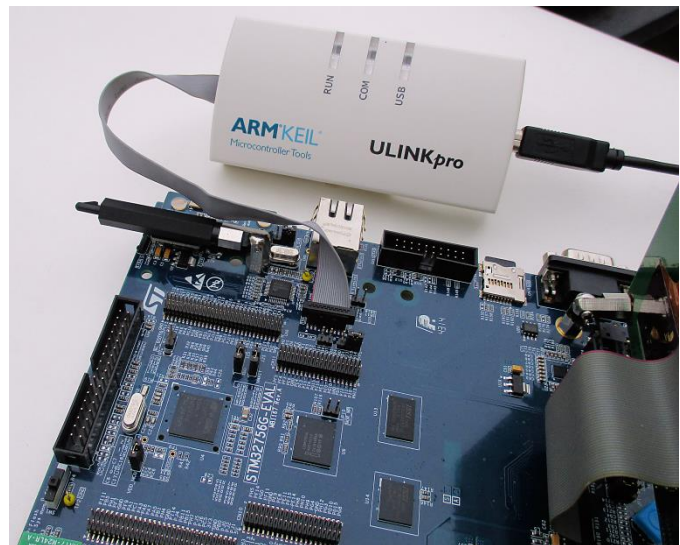
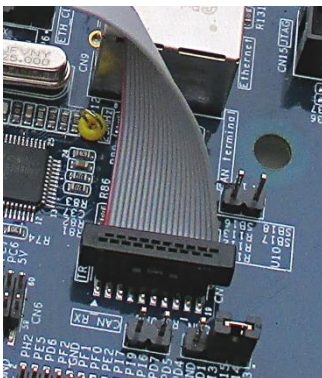
There are three main methods to create your own projects:

- 1) **STM32CubeMX.** This configures your processor and exports a μ Vision project in MDK 5 format. Using STM32CubeMX is the best way to configure your processor clocks.
- 2) **Standard Peripheral Libraries** from ST. Examples and source code. If in MDK 4 format, it can be converted into MDK 5 format with μ Vision Migrate feature. In μ Vision: Project/Manage/Migrate to MDK 5
www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00023896.pdf
- 3) **Using μ Vision Software packs, examples and Keil Middleware.** A Software pack includes many examples and header files that you can use. See
www.keil.com/pack/doc/STM32Cube/General/html/index.html

For more information on creating your projects, see www.keil.com/pack/doc/STM32Cube/General/html/index.html

An STM32756G_EVAL board connected to a Keil ULINKpro.

Close up of the connection to the 20 pin CoreSight ETM/SWD/JTAG connector:





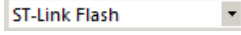
26) Creating your own RTX MDK 5 project from scratch:

The MDK Software Packs makes it easy to configure an RTX project. We will use RTX that is CMSIS-RTOS compliant.


Configuring RTX is easy in MDK 5. These steps use the same configuration as in the preceding Blinky example.

For RTX documentation see: www.keil.com/pack/doc/CMSIS/RTX/html/example_rtx_tutorial.html

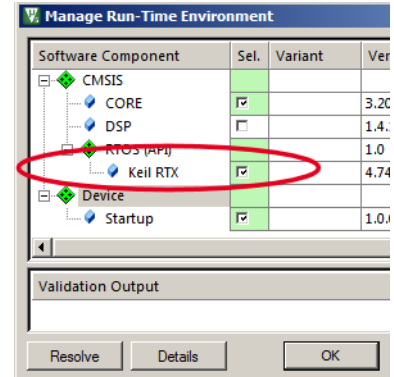
1. Using the same example from the preceding pages, Stop the program  and Exit Debug mode. 

2. Select ST-Link Flash: 

3. Open the Manage Run-Time Environment window: 


4. Expand all the elements as shown here: 

5. Select Keil RTX as shown and click OK.




6. Appropriate RTX files will be added to your project. See the Project window under the CMSIS group.






7. In Blinky.c, at the top, add this line: `#include "cmsis_os.h"`. You can also right-click inside Blinky.c and select Insert '#include' and select cmsis_os.h.

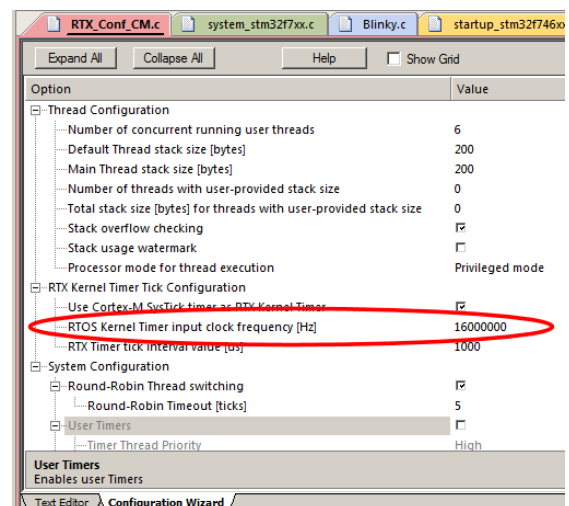
8. Click on File/Save All or 

Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.
3. Select the Configuration Wizard tab: Select Expand All.
4. The window is displayed here: 
5. Set Timer clock value: to 16000000 as shown: (16 MHz)
6. Unselect User Timers. Use defaults for the other settings.

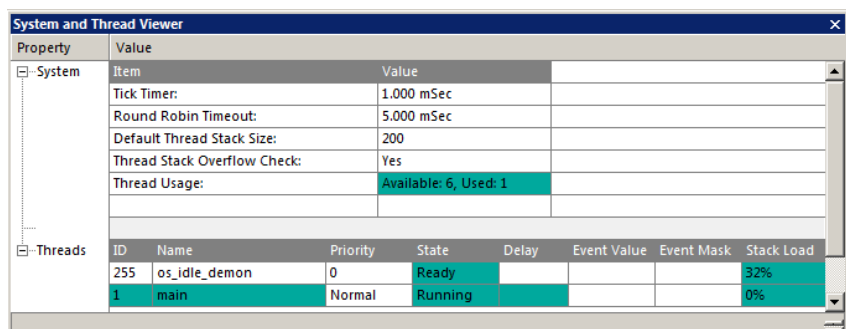
Build and Run Your RTX Program:

1. Click on File/Save All or 
2. Build the files.  Program the Flash: .
3. Enter Debug mode:  Click on the RUN icon. 
4. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
5. You can see two threads: the main thread is the only one running. As you add more threads to create a real RTX program, these will automatically be added to this window.



What you have to do now:

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs. See the next page.
2. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/mdk5/. It has very useful information on implementing and maintaining RTX.



27) Adding a Thread to your RTX_Blinky:

We will create and activate a thread. We will add another global variable counter2 to give it something to do.

1. In Blinky.c, add this line near line 6: unsigned int counter2=0;

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

Create the Thread job1:

2. Add this code to be the thread job1:

osDelay(500) delays the program by 500 clock ticks to slow it down so we can see the values of counter and counter2 increment by 1.

```
8 void job1 (void const *argument) {
9     for (;;) {
10         counter2++;
11         if (counter2 > 0xf) counter2=0;
12         osDelay(500);
13     }
14 }
```

Add osDelay to main():

3. Add this line just after the if statement near line 28: 28 osDelay(500);

Define and Create the Thread:


4. Define job1 near line 15 just before main():

```
15 osThreadDef(job1, osPriorityNormal, 1, 0);
```



5. Create the thread job1 near line 23 just before the while(1) loop:

```
23 osThreadCreate(osThread(job1), NULL);
```

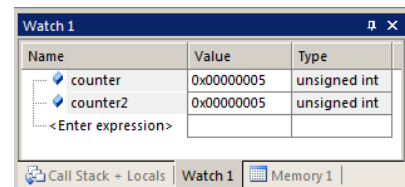
6. Click on File/Save All or 

7. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.

Run the Program and configure Watch 1 and see RTX running:

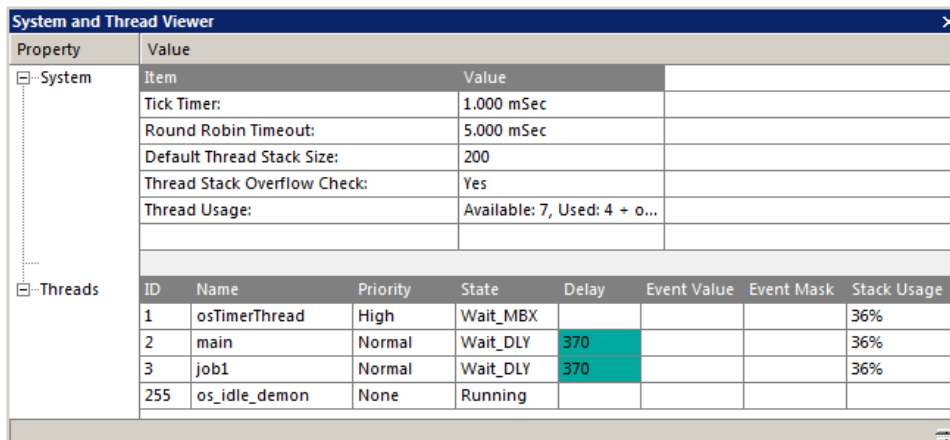
8. Enter Debug mode:  Click on the RUN icon. 
9. Right click on counter2 in Blinky.c and select Add counter2 to ... and select Watch 1.

10. Both counter and counter2 will increment but slower than before:
The two osDelay(500) function calls each slow the program down by 500 msec. This makes it easier to watch these two global variables increment.
OsDelay() is a function provided by RTX.



Name	Value	Type
counter	0x00000005	unsigned int
counter2	0x00000005	unsigned int
<Enter expression>		

11. Open the System and Thread Viewer by selecting Debug/OS Support.
12. Note that job1 has now been added as a thread as shown below:
13. Note os_idle_demon is always labelled as Running. This is because the program spends most of its time here.
14. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
15. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
16. There are many attributes of RTX you can add. RTX help files are located here depending on your CMSIS version:
C:/Keil_v5/ARM/Pack/ARM/CMSIS/4.3.0/CMSIS/Documentation/RTX/html/index.html.









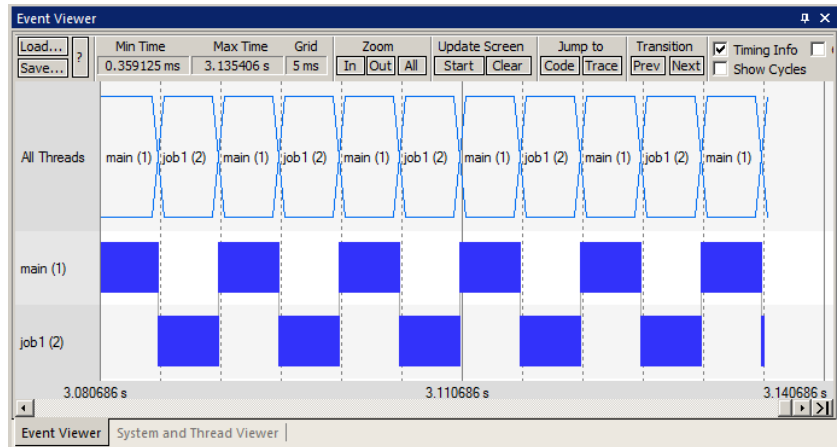
Property	Value
System	
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 4 + 0...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				36%
2	main	Normal	Wait_DLY	370			36%
3	job1	Normal	Wait_DLY	370			36%
255	os_idle_demon	None	Running				

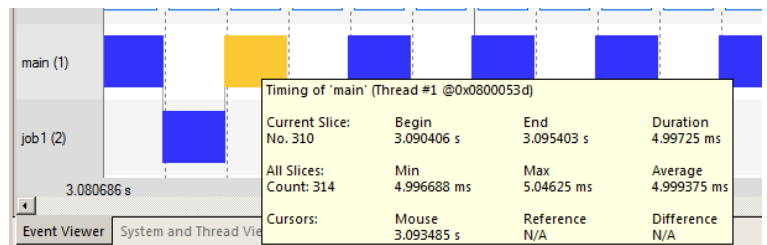
28) Using the Event Viewer to examine your RTX_Blinky Timing:

We will demonstrate the utility of the Event Viewer.

- 1) Stop the program  and Exit Debug mode. 
- 2) In Blinky.c there are two lines `osDelay();`. Comment both of these out. We will run the program really fast.
- 3) Open Select Target Options  or ALT-F7 and select the Debug tab, and then the Trace tab.
- 4) Set Core Clock: to 216 MHz. Unselect EXCTRC: and Periodic. Leave everything at their default settings.
- 5) Click on OK twice to return to the main μ Vision menu.
- 6) Build the files.  Enter Debug mode:  Click on the RUN icon. 
- 7) Open the Event Viewer by selecting Debug/OS Support and select Event Viewer. This window will open:
- 8) Adjust Zoom In and Out for a comfortable view.

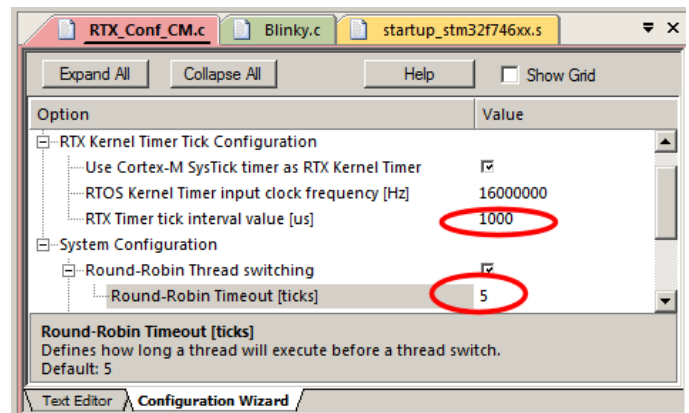


- 9) Hold your cursor over one of the blue blocks and an information window is displayed as shown below:
- 10) Note each thread duration is about 5 msec.



What is Happening Here:

- 1) Open RTX_Conf_CM.c and select the Configuration Wizard tab as shown below:
- 2) The CPU speed is set to 16 MHz with a Timer tick value of 1000 us or 1 msec.
- 3) Note Round Robin switching is selected with a tick timeout of 5 ticks.
- 4) The Thread timing is this 1 msec times 5 ticks = 5 msec.
- 5) Every 5 msec the Thread is switched to the next one and these sequences are displayed in the Event Viewer.
- 6) It is quite easy to view how RTX is running to make sure it is performing as you designed.




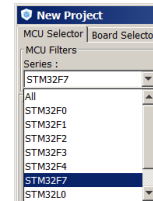
This concludes the lab exercises. Thank you !

29) Using STM32CubeMX to create a simple Blinky program:





Using STM32CubeMX to create your μ Vision project is a very good idea. It easily configures the multiplexed pins on your processor, calculate the clock settings and create a MDK 5 uvprojx using CMSIS to use as your starting point.

Download and Install STM32CubeMX:

1. Download and install STM32CubeMX from www.st.com. Search the web for STM32CubeMX.
2. Download and install [STM32CubeF7](#) Embedded Software package for the STM32F7 series.
3. Open STM32Cube MX  and select New Project: [New Project](#)



Create a new Project: select your processor and configure user LED Port PI pin 1 as GPIO:

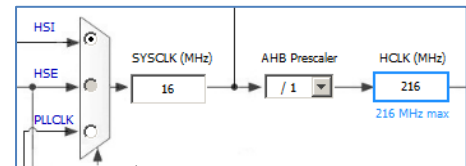
1. In the Series box, select STM32F7 as shown here: 
2. To the right, now select STM32F746NGHx as shown below:
3. Click OK. Wait for the Pinout window to display. It will have a graphic pinout diagram of your processor. The Pinout tab is selected.
4. In the Discovery board, the green user LED LD1 is attached to Port I pin 1 as found from the schematics. We will select and configure this pin for GPIO out.
5. In the Find box, enter Pi1 or PI1.  In the List box that opens, select PI1.
6. The PI1 circle (or pad) will turn darker grey.
7. Click on the circle PI1 and this menu selection opens: Select GPIO_Output as shown here: 
8. The PI1 circle will turn green with a check as shown here indicating a successful configuration: 

STM32F746IGTx	STM32F7x6	LQFP176	1024	320	0	140
STM32F746NEHx	STM32F7x6	TFBGA216	512	320	0	168
STM32F746NGHx	STM32F7x6	TFBGA216	1024	320	0	168
STM32F746VETx	STM32F7x6	LQFP100	512	320	0	82
STM32F746VGTx	STM32F7x6	LQFP100	1024	320	0	82

PI1
Reset_State
DCMI_D8
FMC_D25
I2S2_CK
LTDC_G6
SPI2_SCK
TIM8_BKIN2
GPIO_Input
GPIO_Output
GPIO_Analog
EVENTOUT
GPIO_EXTI1



Configure the Processor clocks: You set the CPU speed here. Default is 16 MHz.

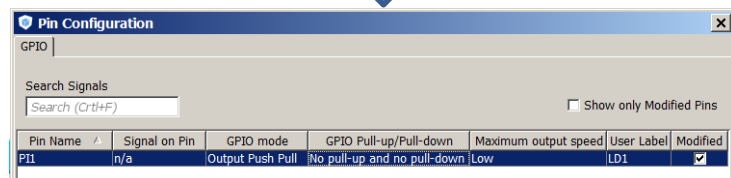
1. Select the Clock Configuration tab: [Clock Configuration](#)
2. Find the HCLK box and insert 216 and press Enter as shown here:
3. A info window will display stating no solution exists.. Select Yes to find a solution. (Selection must change from HSI to PLLCLK)
4. Note your final clock value is calculated backwards to obtain the various settings.



TIP: You can lock a value by right clicking on a box. This value will stay constant as you change other values.

Inspect Pin Configuration Settings: You can modify the settings of the pin items you have set here.

1. Select the Configuration tab: [Configuration](#)
2. Under the Systems column, select GPIO.  This window opens: 
3. Highlight GPIO: The bottom part opens:
4. Note the various items you can modify.
5. In User Label enter **LD1**.
6. Click OK to close this window.



Save your Project:



1. In the main window, select File/Save Project or Ctrl-S.
2. In the Project Name field, enter BlinkyCube (or whatever you prefer).
3. In the Toolchain/IDE field, select MDK-ARM V5. Note the Project Location shows where the project is stored.
4. Click on OK to save and close.

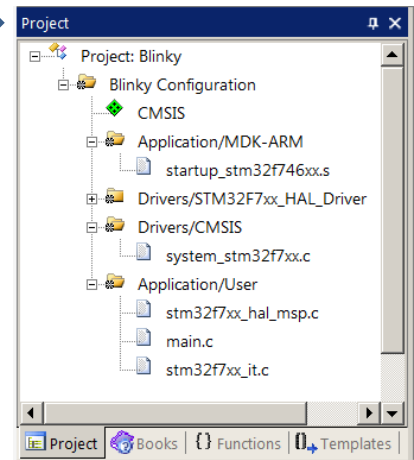
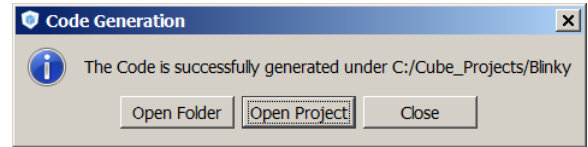
What you have so far: You have setup up GPIO Port I pin 1 to turn LD1 green LED on. The CPU speed is set to 216 MHz. Many other configuration items have been automatically set by STM32Cube for you. You can still modify them.

Generate µVision Project Code:

1. In the main STM32CubeMX menu, select Project/Settings. Confirm the Toolchain selected is MDK-AM V5.
2. Click on OK to close.
3. If you have µVision open to a previous version of this generated code, please close µVision now.
4. In the main STM32CubeMX menu, select Project/Generate Code. This will create the MDK 5 project files.
5. A progress bar will display. When it is finished this window will display: At this point, the µVision project exists.
6. Select Open Project and µVision will be started and load Blinky.

Examine and Build the Project Files:

1. Build the files.  The Build Output window will show no errors or warnings.
2. In the Project window in µVision the file structure is displayed. 
3. Expand the Application/User folder as shown below here:
4. Double click on main.c to open it. At this point, all you need to do is add your own source code. main.c has a main() function containing some calls to initialization files but an empty while(1) loop. There are sections created where you can enter your own source code.



Add Source Code to Blink LD1:

1. In main.c, in the while(1) loop near line 90 is a User Code section 3.
2. Add these two lines to near line 91:





```
91  HAL_GPIO_TogglePin(GPIOI, GPIO_PIN_1);
92  HAL_Delay(100);
```

3. Select File/Save All. 

TIP: You can see where these functions are located by right clicking on the call and selecting Go To Definition:

Go To Definition Of 'HAL_GPIO_TogglePin'

Running your Program:

1. Build the files.  The Build Output window will show no errors or warnings. If there are, please fix them now.
2. Connect your PC to the board with a USB cable to CN14.
3. By default, the ST-Link Debugger is selected. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
4. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears. The Flash will be programmed.
5. Click on the RUN icon. 

The LED LD1 on the STM32F7 Discovery board will blink.

Congratulations – you have created your first program with STM32CubeMX !

USING Keil RTX RTOS: At this point, you can easily add RTX to this project. See page 25 and the MDK 5 Getting Started Guide.

30) Serial Wire Viewer (SWV) and how to use it:


1) Data Reads and Writes: (Note: Data Writes but not Reads are enabled in the current version of μ Vision).

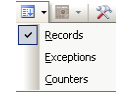
You have already configured Serial Wire Viewer (SWV) on page 13 under **RTX Viewer: Configuring the Serial Wire Viewer:**

Now we will examine some of the features available to you. SWV works with μ Vision and ST-Link V2, ULINK2/ME, ULINK pro or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. Your program runs at full speed and needs no code stubs or instrumentation software added to your source code. Screens are shown using a ST-Link.

1. Use RTX_Blinky5 from the previous exercise. Enter Debug mode and Run the program if not already running.

2. Select View/Trace/Records or click on the Trace icon  and select Records.



3. The Trace Records window will open up as shown here:

4. The ITM frames are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. here:

5. To turn this off, select Debug/Debug Settings and click on the Trace tab. Unselect ITM Stimulus Port 31.

TIP: Port 0 is used for Debug `printf` Viewer.

6. Unselect EXCTRC and Periodic.

7. Select On Data R/W Sample.

8. Click on OK twice to return.

TIP: If the SWV trace fails to work properly after this change, exit Debug mode, cycle the power to the Discovery board and re-enter Debug mode.

9. Click on the RUN icon.

10. Double-click in Trace Records window to clear it.

11. Only Data Writes will appear now.

TIP: You could have right clicked on the Trace Records window to filter the ITM frames out. Unselecting a feature is better as it reduces SWO pin traffic and trace overflows.

What is happening here ? 

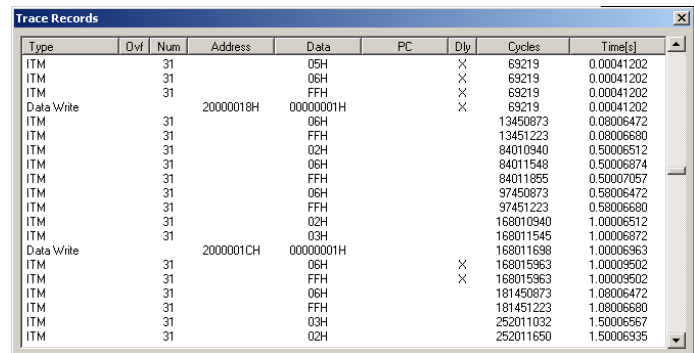
1. When variables are entered in the Logic Analyzer (remember phasea ?), the reads and/or writes will appear in Trace Records.
2. The Address column shows where the variable is.
3. The Data column displays the data values written to phasea.
4. PC is the address of the instruction causing the writes. You activated it by selecting On Data R/W Sample in the Trace configuration window.
5. The Cycles and Time(s) columns are when these events happened.

TIP: You can have up to four variables in the Logic Analyzer and subsequently displayed in the Trace Records window. You must have all Watchpoints off.

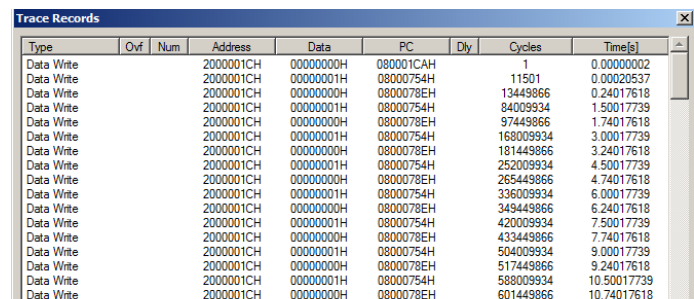
TIP: If you select View/Symbol Window you can see where the addresses of the variables are located.

Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

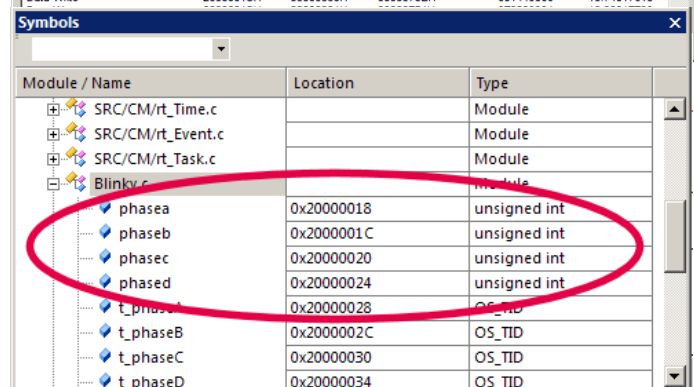
TIP: ULINK pro and Segger J-Link adapters display the trace frames in a slightly different style trace window. The J-Link currently does not display Data reads or writes.



Type	Ofs	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		05H		X	69219	0.00041202
ITM		31		06H		X	69219	0.00041202
ITM		31		FFH		X	69219	0.00041202
Data Write			20000018H	00000001H		X	69219	0.00041202
ITM		31		06H			13450873	0.08006472
ITM		31		FFH			13451223	0.08006680
ITM		31		02H			84010940	0.50006512
ITM		31		06H			84011548	0.50006874
ITM		31		FFH			84011855	0.50007057
ITM		31		06H			97450873	0.58006472
ITM		31		FFH			97451223	0.58006680
ITM		31		02H			168010940	1.00006512
ITM		31		03H			168011545	1.00006872
Data Write			2000001CH	00000001H			168011638	1.00006963
ITM		31		06H		X	168015363	1.00009502
ITM		31		FFH		X	168015363	1.00009502
ITM		31		06H			181450873	1.08006472
ITM		31		FFH			181451223	1.08006680
ITM		31		03H			252011032	1.50006567
ITM		31		02H			252011650	1.50006935



Type	Ofs	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			2000001CH	00000000H	080001CAH		1	0.00000002
Data Write			2000001CH	00000001H	08000754H		11501	0.00020537
Data Write			2000001CH	00000000H	0800078EH		13449866	0.24017618
Data Write			2000001CH	00000001H	08000754H		84009934	1.50017739
Data Write			2000001CH	00000000H	0800078EH		97449866	1.74017618
Data Write			2000001CH	00000001H	08000754H		168009934	3.00017739
Data Write			2000001CH	00000000H	0800078EH		181449866	3.24017618
Data Write			2000001CH	00000001H	08000754H		252009934	4.50017739
Data Write			2000001CH	00000000H	0800078EH		265449866	4.74017618
Data Write			2000001CH	00000001H	08000754H		336009934	6.00017739
Data Write			2000001CH	00000000H	0800078EH		349449866	6.24017618
Data Write			2000001CH	00000001H	08000754H		420009934	7.50017739
Data Write			2000001CH	00000000H	0800078EH		433449866	7.74017618
Data Write			2000001CH	00000001H	08000754H		504009934	9.00017739
Data Write			2000001CH	00000000H	0800078EH		517449866	9.24017618
Data Write			2000001CH	00000001H	08000754H		588009934	10.50017739
Data Write			2000001CH	00000000H	0800078EH		601449866	10.74017618



Module / Name	Location	Type
SRC/CM/rt_Time.c		Module
SRC/CM/rt_Event.c		Module
SRC/CM/rt_Task.c		Module
Blinky.c		Module
phasea	0x20000018	unsigned int
phaseb	0x2000001C	unsigned int
phases	0x20000020	unsigned int
phased	0x20000024	unsigned int
t_phaseA	0x20000028	OS_TID
t_phaseB	0x2000002C	OS_TID
t_phaseC	0x20000030	OS_TID
t_phaseD	0x20000034	OS_TID

2) Exceptions and Interrupts:

The STM32 family using the Cortex-M4 processor has many interrupts and it can be difficult to determine when they are being activated and how often. Serial Wire Viewer (SWV) on the STM32 family makes this task easy.

1. Stop the RTX_Blinky example program. Be in Debug mode. Open Debug/Debug Settings and select the Trace tab.
2. Unselect On Data R/W Sample, PC Sample and ITM Ports 31 and 0. (this is to minimize overloading the SWO port)
3. Select EXCTRC as shown here:
4. Click OK twice.
5. Double click on Trace Records to clear it.
6. Click RUN to start the program.

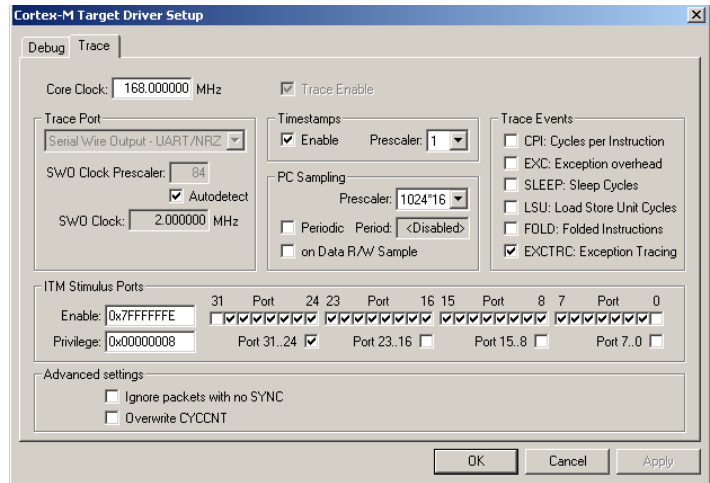
TIP: If the SWV trace fails to work properly after this change, exit and re-enter Debug mode.

7. You will see a window similar to the one below with Exceptions frames displayed.

.What Is Happening ?

1. You can see two exceptions happening.
 - **Entry:** when the exception enters.
 - **Exit:** When it exits or returns.
 - **Return:** When all the exceptions have returned to the main program. This is useful to detect tail-chaining.
2. Num 11 is SVCcall from the RTX calls.
3. Num 15 is the SysTick timer.
4. In my example you can see one data write from the Logic Analyzer.
5. Note everything is timestamped.
6. The “X” in Ovf is an overflow and some data was lost. The “X” in Dly means the timestamps are delayed because too much information is being fed out the SWO pin. Always limit the SWV features to only those you really need.

TIP: The SWO pin is one pin on the Cortex-M4 family processors that all SWV information is fed out. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate. μ Vision easily recovers gracefully from these overflows. Overflows are shown when they happen. Using a ULINK μ pro helps reduce overruns, especially if the 4 bit Trace Port connection is used rather than the 1 bit SWO pin.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					3645281162	65.09430646
Exception Exit		15					3645281343	65.09430970
Exception Return		0					3645281351	65.09430984
Exception Entry		15					3646961162	65.12430646
Exception Exit		15					3646961336	65.12430957
Exception Return		0					3646961344	65.12430971
Exception Entry		15					3648641162	65.15430646
Exception Exit		15					3648641520	65.15431286
Exception Return		0					3648641528	65.15431300
Data Write			2000001CH	00000000H		X	3648643664	65.15435114
Exception Return	X	0				X	3648643664	65.15435114
Exception Entry		15					3650321163	65.18430648
Exception Exit		15					3650321397	65.18431066
Exception Return		0					3650321405	65.18431080
Exception Entry		15					3652001162	65.21430646
Exception Exit		15					3652001407	65.21431084
Exception Return		0					3652001415	65.21431098
Exception Entry		15					3653681162	65.24430646
Exception Exit		15					3653681343	65.24430970
Exception Return		0					3653681351	65.24430984

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
15	SysTick	99845	143.108 ms	0 s	111.786 us	59.524 ns	93.722 ms	28.46194736	38.65088633
11	SVCcall	398	0 s					28.52330453	38.62842232
93	OTG_HS	0	0 s						
92	OTG_HS_WKUP	0	0 s						
91	OTG_HS_EPI_IN	0	0 s						
90	OTG_HS_EPI_OUT	0	0 s						
89	I2C3_ER	0	0 s						

1. Select View/Trace/Exceptions or click on the Trace icon and select Exceptions.
2. The next window opens up and more information about the exceptions is displayed as shown below:
3. Note the number of times these have happened under Count. This is very useful information in case interrupts come too fast or slow. Click on Count to bring the most active exceptions to the top of the window.
4. ExtIRQ are the peripheral interrupts.
5. You can clear this trace window by double-clicking on it.
6. All this information is displayed in real-time and without stealing any CPU cycles or stubs in your code !

TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 - 16 = ExtIRQ 0.

3) PC Samples:

Serial Wire Viewer can display a sampling of the program counter.

SWV can display at best every 64th instruction but usually every 16,384 is more common. It is best to keep this number as high as possible to avoid overloading the Serial Wire Output (SWO) pin. This is easily set in the Trace configuration.

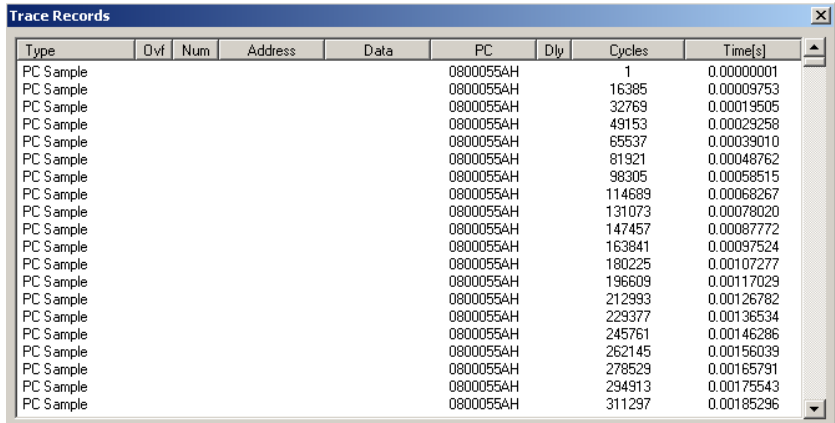
1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear.

5. Click on RUN and this window opens:
6. Most of the PC Samples in the example shown are 0x0800_055A which is a branch to itself in a loop forever routine.

Note: the exact address you get depends on the source code and the compiler settings.

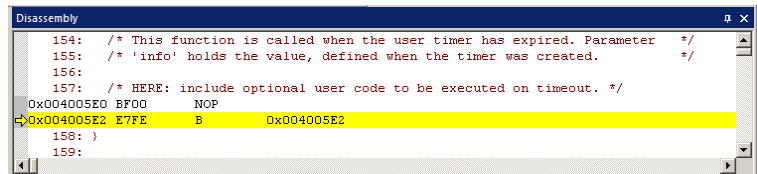
7. Stop the program and the Disassembly window will show this Branch as shown below:
8. Not all the PCs will be captured. Still, PC Samples can give you some idea of where your program is; especially if it is not caught in a tight loop like in this case.

9. Set a breakpoint in one of the tasks.
10. Run the program and when the breakpoint is hit, you might see another address at the bottom of the Trace Records window. See the screen below:

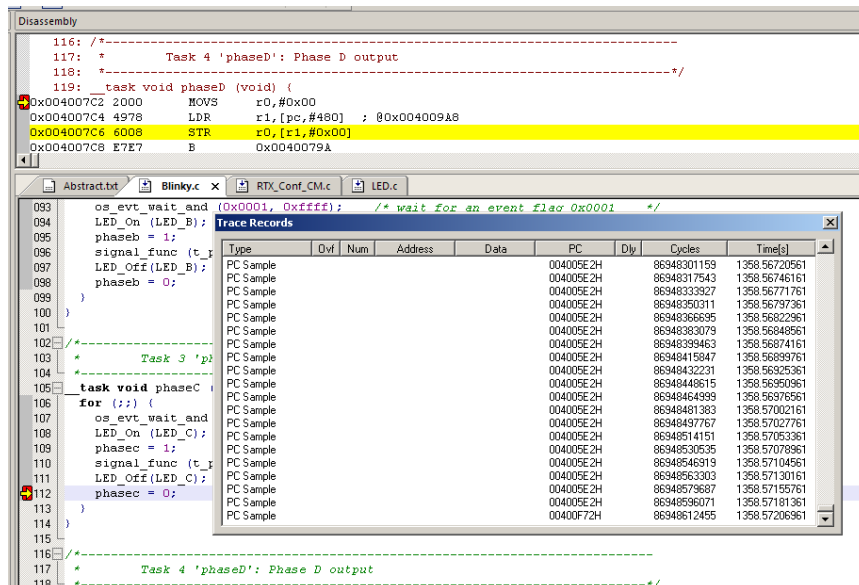


Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					0800055AH		1	0.00000001
PC Sample					0800055AH		16385	0.00009753
PC Sample					0800055AH		32769	0.00019505
PC Sample					0800055AH		49153	0.00029258
PC Sample					0800055AH		65537	0.00039010
PC Sample					0800055AH		81921	0.00048762
PC Sample					0800055AH		98305	0.00058515
PC Sample					0800055AH		114689	0.00068267
PC Sample					0800055AH		131073	0.00078020
PC Sample					0800055AH		147457	0.00087772
PC Sample					0800055AH		163841	0.00097524
PC Sample					0800055AH		180225	0.00107277
PC Sample					0800055AH		196609	0.00117029
PC Sample					0800055AH		212993	0.00126782
PC Sample					0800055AH		229377	0.00136534
PC Sample					0800055AH		245761	0.00146286
PC Sample					0800055AH		262145	0.00156039
PC Sample					0800055AH		278529	0.00165791
PC Sample					0800055AH		294913	0.00175543
PC Sample					0800055AH		311297	0.00185296

11. Scroll to the bottom of the Trace Records window and you might (probably not) see the correct PC value displayed. Usually, it will be a different PC depending on when the sampling took place.
12. To see all the instructions executed, you can use the ETM instruction trace with a ULINK^{pro}.
13. Remove the breakpoint.
14. Stop the program.
15. Leave Debug mode.



```
154: /* This function is called when the user timer has expired. Parameter */
155: /* 'info' holds the value, defined when the timer was created. */
156:
157: /* HERE: include optional user code to be executed on timeout. */
0x004005E0 BFD0 NOP
0x004005E2 E7FE B 0x004005E2
158:
159:
```





```
116: /*-----
117: * Task 4 'phased': Phase D output
118: *-----*/
119: task void phased (void) {
0x004007C2 2000 MOVS r0,#0x00
0x004007C4 4978 LDR r1,[pc,#480] ; 0x004009A8
0x004007C6 6008 STR r0,[r1,#0x00]
0x004007C8 E7E7 B 0x0040079A
```

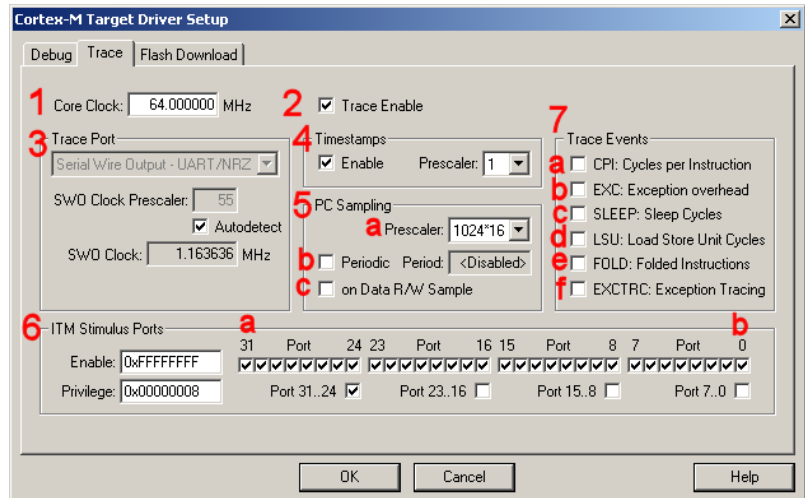
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					004005E2H		86348301159	1358.56720561
PC Sample					004005E2H		86348317543	1358.56746151
PC Sample					004005E2H		86348333827	1358.56771761
PC Sample					004005E2H		86348350311	1358.56797361
PC Sample					004005E2H		86348366695	1358.56822961
PC Sample					004005E2H		86348383079	1358.56848561
PC Sample					004005E2H		86348399463	1358.56874161
PC Sample					004005E2H		86348415847	1358.56899761
PC Sample					004005E2H		86348432231	1358.56925361
PC Sample					004005E2H		86348448615	1358.56950961
PC Sample					004005E2H		86348464999	1358.56976561
PC Sample					004005E2H		86348481383	1358.57002161
PC Sample					004005E2H		86348497767	1358.57027761
PC Sample					004005E2H		86348514151	1358.57053361
PC Sample					004005E2H		86348530535	1358.57078961
PC Sample					004005E2H		86348546919	1358.57104561
PC Sample					004005E2H		86348563303	1358.57130161
PC Sample					004005E2H		86348579687	1358.57155761
PC Sample					004005E2H		86348596071	1358.57181361
PC Sample					00400F2H		86348612455	1358.57206961

31) Serial Wire Viewer (SWV) Configuration window: (for reference)

The essential place to configure the trace is in the Trace tab as shown below. You cannot set SWV globally for μ Vision. You must configure SWV for every project and additionally for every target settings within a project you want to use SWV. This configuration information will be saved in the project. There are two ways to access this menu:

- A. **In Edit mode:** Select Target Options  or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window and then the Trace tab. Edit mode is selected by default when you start μ Vision.
- B. **In Debug mode:** Select Debug/Debug Settings and then select the Trace tab. Debug mode is selected with .

- 1) **Core Clock:** The CPU clock speed for SWV. The CPU speed can be found in your startup code or in Abstract.txt. It is usually called SYSCLK or Main Clock. This *must* be set correctly for all adapters except ULINKpro.
- 2) **Trace Enable:** Enables SWV and ITM. It can only be changed in Edit mode. This does not affect the Watch and Memory window display updates.
- 3) **Trace Port:** This is preset for ST-Link.
- 4) **Timestamps:** Enables timestamps and selects the Prescaler. 1 is the default.
- 5) **PC Sampling:** Samples the program counter:



- a. **Prescaler** 1024*16 (the default) means every 16,384th PC is displayed. The rest are not collected.
- b. **Periodic:** Enables PC Sampling.
- c. **On Data R/W Sample:** Displays the address of the instruction that caused a data read or write of a variable listed in the Logic Analyzer. This is not connected with PC Sampling but rather with data tracing.
- 6) **ITM Stimulus Ports:** Enables the thirty-two 32 bit registers used to output data in a *printf* type statement to μ Vision. Port 31 (a) is used for the Keil RTX Viewer which is a real-time kernel awareness window. Port 0 (b) is used for the Debug (printf) Viewer. The rest are currently unused in μ Vision.
 - **Enable:** Displays a 32 bit hex number indicating which ports are enabled.
 - **Privilege:** Privilege is used by an RTOS to specify which ITM ports can be used by a user program.
- 7) **Trace Events:** Enables various CPU counters. All except EXCTRC are 8 bit counters. Each counter is cumulative and an event is created when this counter overflows every 256 cycles. These values are displayed in the Counter window. The event created when a counter wraps around is displayed in the Instruction Trace window.
 - a. **CPI: Cycles per Instruction:** The cumulative number of extra cycles used by each instruction beyond the first, one including any instruction fetch stalls.
 - b. **Fold:** Cumulative number of folded instructions. These results from a predicted branch instruction where unused instructions are removed (flushed) from the pipeline giving a zero cycle execution time.
 - c. **Sleep:** Cumulative number of cycles the CPU is in sleep mode. Uses FCLK for timing.
 - d. **EXC:** Cumulative cycles CPU spent in exception overhead not including total time spent processing the exception code. Includes stack operations and returns.
 - e. **LSU:** Cumulative number of cycles spent in load/store operations beyond the first cycle.
 - f. **EXCTRC:** Exception Trace. This is different than the other items in this section. This enables the display of exceptions in the Instruction Trace and Exception windows. It is not a counter. This is a very useful feature to display exception events and is often used in debugging.

TIP: Counters will increment while single stepping. This can provide some very useful information. You can read these counters with your program as they are memory mapped.


32) ETM Trace Examples:


These examples were run on the STM32756G-EVAL evaluation board. A ULINK_{pro} debug adapter is required for ETM operation. You can use a ULINK_{pro} on your own custom board with the addition of the 5 pins Trace Port (4 data + clock). Many STM32 processors have ETM. ETM is a separate CoreSight component for Serial Wire Viewer (SWV).

ETM provides serious debugging power as shown on the next few pages. It is worth the small added cost. Most STM32 processors are ETM equipped.

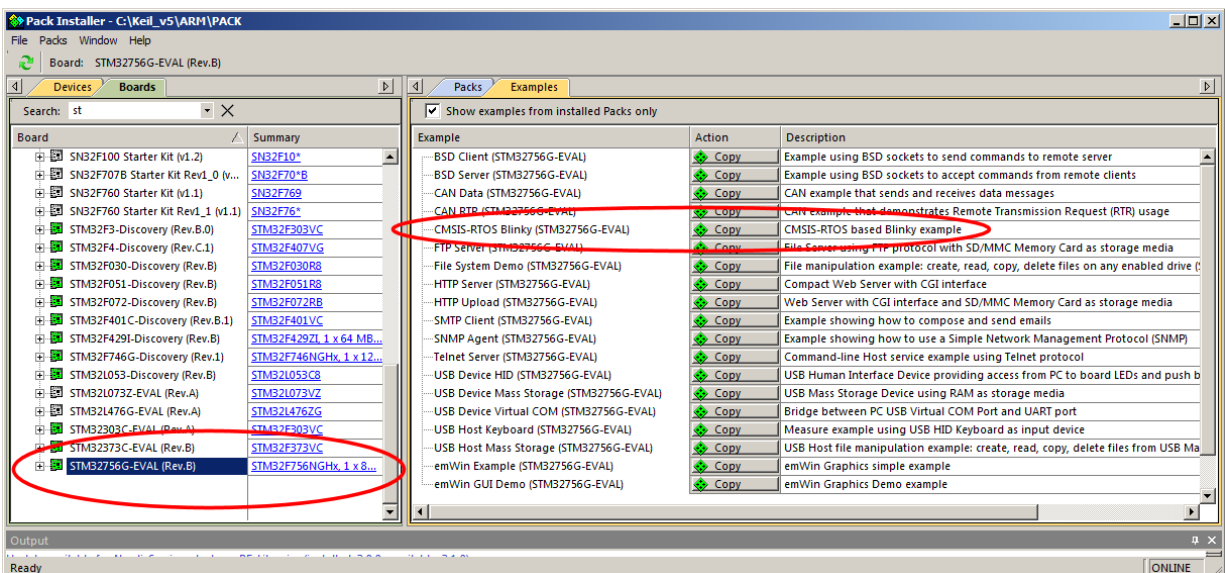
1) Install STM32756G-EVAL Examples:

6. Connect your computer to the internet. This is normally needed to download the Software Pack examples.


7. Start μ Vision by clicking on its desktop icon. 

8. Open the Pack Installer (PI) by clicking on its icon: 

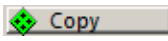
9. This window opens up: Under the Boards tab, in the Search box, type STM32756G-EVAL as shown below: This will filter the list under the Packs and Examples tabs.



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

11. Packs/Check for Updates or  to refresh once you have connected to the Internet. It is not done automatically.

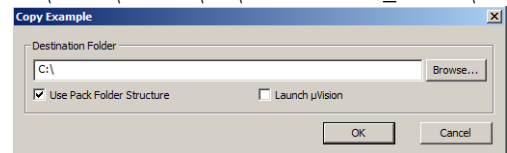
12. Select the Examples tab and note the many examples you can try. The middleware examples need a MDK license.

13. Beside CMSIS-RTOS Blinky (STM32756G-EVAL), click on Copy. 

10. The Copy Example window below opens up: Select Use Pack Folder Structure: Unselect Launch μ Vision:

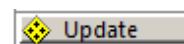
14. Type in C:\ as shown to the right: Click OK. Blinky will be copied to C:\MDK\Boards\ST\STM32756G_EVAL\

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



11. Close the Packs Installer. Open it any time by clicking on its icon. 

TIP: An Update icon means there is an updated Software Pack available for download.





1) Configure and RUN ETM Trace for the Blinky Example:



Connect ULINKpro and load the Blinky example:

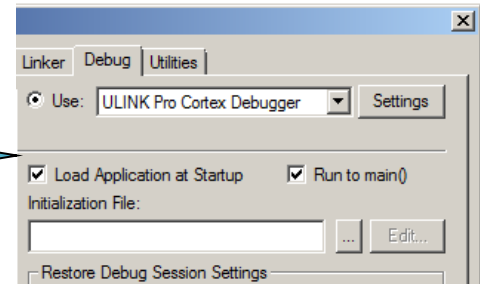
1. Connect the ULINKpro to the STM3240G board using the 20 pin CN13 Trace connector. See below:
2. Start μ Vision by clicking on its desktop icon. 
3. Select Project/Open Project. Open C:\MDK\Boards\ST\STM32756G_EVAL\Blinky.uvprojx.

Create a new Target Options configuration:

1. Select Project/Manage/Project Items: 
4. Click on the Insert icon:  Enter **ULINKpro Flash** (or whatever you want). Enter and then Close.
5. Select the Target Option you just created: 




Configure the ETM Trace:

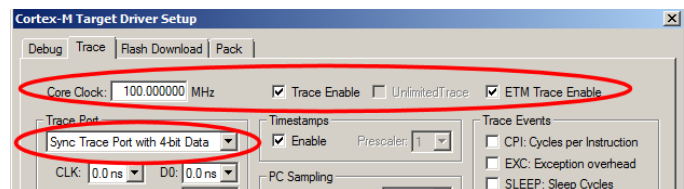
6. **ULINKpro** Select Target Options  or ALT-F7. Select the Debug tab. Select ULINK Pro Cortex Debugger as shown here: 
7. Select Settings: on the right side of this window.
8. In the next window Confirm SW is selected and a valid ID Code and Device Name are displayed. If not, you **must** fix this before continuing.
9. Select the Trace tab.
10. In the Trace window, select Trace Enable as shown below. Set Core Clock: to 100 MHz.
11. In the Trace Port box, select Sync Trace Port with 4 bit Data as shown below:
12. Click OK twice to return to the main μ Vision menu. ETM trace is now configured.



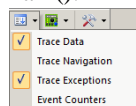
13. Select File/Save All. 

Compile Blinky and Enter Debug Mode:

14. In Blinky near line 60 change PLLN to 200. This is to slow the CPU clock to 100 MHz.
15. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
16. Enter Debug mode by clicking on the Debug icon. . The program will run to the first instruction in main().
17. **DO NOT CLICK ON RUN YET !!!**
18. Open the Data Trace window by clicking on the small arrow beside the Trace Windows icon. 



```
60  RCC_OscInitStruct.PLL.PLLN = 200;
```



19. The Trace Data window will open and some frames will be displayed. These are explained on the next page.

TIP: How to Easily Configure the STM32F7 Clocks:

The smartest way is to utilize STM32CubeMX to determine the values to be used for a clock speed. Select the Clock Configuration tab after you have chosen your processor. This works backwards: select the final clock speed you desire in the HCLK box and press Enter. The various divider values will be calculated. You can also right-click on a box to force it to remain (lock) the same.

An STM32756G_EVAL board connected to a Keil ULINKpro using the special CoreSight 20 pin ETM connector CN12:



Examine the Trace Data Window:

1. Examine the Instruction Trace window as shown below: This is a complete record of all the program flow since RESET until μ Vision halted the program at the start of main() since Run To main is selected in μ Vision.
2. The last two frames are actually SWV Exception frames and not instruction frames. Open the Exception Trace window and click on the Count column to bring SVCALL to the top. In my case, two exceptions were executed.
3. In this case, 100 060 s shows the last instruction to be executed. (BX lr). In the Register window the PC will display the location of the next instruction to be executed (0x0800_2428 in my case).
4. Features of The Trace Data window:
 - a. Address of the instruction with its mnemonic is shown with other information.
 - b. Any Serial Wire Viewer (SWV) events are displayed – see the last two frames.
 - c. Source Code if available is displayed.
 - d. The function the instruction belongs to is displayed. The start of a sequence is in orange.
5. Click on an instruction frame and this will be displayed in the Disassembly and source windows.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x08003CC4	LDR r0,[r0,#0x00]	(os_tsk.run->stack[0] != MAGIC_WORD)) {	rt_stk_check
	X: 0x08003CC6	LDR r1,[pc,#16] ; @0x0800...		rt_stk_check
	X: 0x08003CC8	CMP r0,r1		rt_stk_check
0.000 099 490 s	X: 0x08003CCA	BEQ 0x08003CD2		rt_stk_check
	X: 0x08003CD2	BX lr	}	rt_stk_check
	X: 0x08000354	POP {r2-r3}	POP {R2,R3}	SVC_Handler
	X: 0x08000356	STR r2,[r3,#0x00]	STR R2,[R3] ; os_tsk.run = os_tsk.new	SVC_Handler
	X: 0x08000358	LDR r12,[r2,#0x28]	LDR R12,[R2,TCB_TSTACK] ; os_tsk.new->tsk_stack	SVC_Handler
	X: 0x0800035C	LDM r12!,[r4-r11]	LDMIA R12!,[R4-R11] ; Restore New Context	SVC_Handler
	X: 0x08000360	LDRB r0,[r2,#0x25]	LDRB R0,[R2,TCB_STACKF] ; Stack Frame	SVC_Handler
	X: 0x08000364	CMP r0,#0x00	CMP R0,#0 ; Basic/Extended Stack Frame	SVC_Handler
	X: 0x08000366	ITEE EQ	MVNEQ LR,#NOT:0xFFFFFFFDD ; set EXC_RETURN value	SVC_Handler
	X: 0x08000368	MVNEQ lr,#0x02		SVC_Handler
	X: 0x0800036C	MVNNE lr,#0x12	MVNNE LR,#NOT:0xFFFFFFFDD	SVC_Handler
	X: 0x08000370	VLDMIANE r12!,[s16-s31]	VLDMIANE R12!,[S16-S31] ; restore VFP hi-registers	SVC_Handler
	X: 0x08000374	MSR PSP,r12	MSR PSP,R12 ; Write PSP	SVC_Handler
0.000 100 060 s	X: 0x08000378	BX lr	BX LR	SVC_Handler
0.000 100 290 s		Exception Exit - SVCALL		
0.000 100 640 s		Exception Return		

6. Click on Single Step once. The instruction at 0x0800 2428 is executed as shown below: This is a BL.W instruction branching to the MPU_Config function.

7.

	X: 0x08000374	MSR PSP,r12	MSR PSP,R12 ; Write PSP	SVC_Handler
0.000 100 060 s	X: 0x08000378	BX lr	BX LR	SVC_Handler
0.000 100 290 s		Exception Exit - SVCALL		
0.000 100 640 s		Exception Return		
0.000 100 690 s	X: 0x08002428	BL.W MPU_Config (0x0800...	MPU_Config(); /* Configure the MPU */	main

8. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x08000424	LDR r0,[pc,#36] ; @0x0800...	LDR R0,=SystemInit	Reset_Handler
0.000 000 000 s	X: 0x08000426	BLX r0	BLX R0	Reset_Handler
	X: 0x080021A0	LDR r0,[pc,#80] ; @0x0800...	SCB->CPACR = ((3UL << 10*2) (3UL << 11*2)); /* set CP10 a...	SystemInit
	X: 0x080021A2	LDR r0,[r0,#0x00]		SystemInit
	X: 0x080021A4	ORR r0,r0,#0xF00000		SystemInit
	X: 0x080021A8	LDR r1,[r0,#72] ; @0x0800...		CustomInit

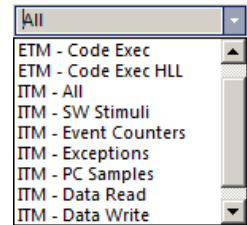
9. Note the RESET_Handler at location 0x0800 0424. If you enter 0x0800 0000 in the Memory window, the second word is the initial PC. It will be 0x0800 0425. The least significant bit means this is a Thumb2 instruction.

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.




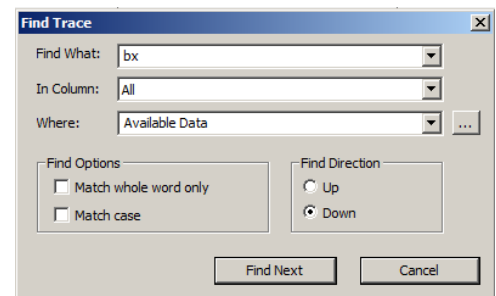
Find a Trace Record:

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



Note 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.



Trace Triggering: *coming soon for Cortex-M7*

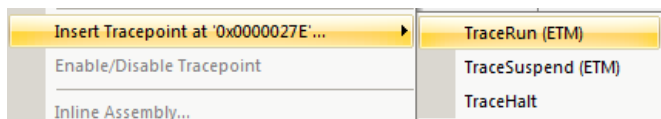
μ Vision has three trace triggers currently implemented:

TraceRun: Starts ETM trace collection when encountered.

TraceSuspend: Stops ETM trace collection when encountered. TraceRun has to have been encountered for this to have an effect.

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

TraceHalt: Stops ETM trace, SWV and ITM. Can be resumed only with a STOP/RUN sequence. TraceStart will not restart this.





How it works:

When you set a TraceRun point in assembly language point, ULINK_{pro} will start collecting trace records. When you set a TraceSuspend point, 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.








How to Set Trace Triggers:

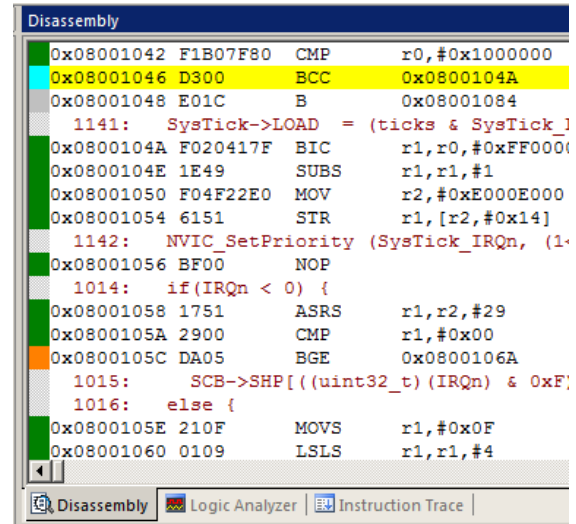
Coming Soon for Cortex-M7 Processors:

Code Coverage:

10. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
11. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin:
12. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not.

Colour blocks indicate which assembly instructions have been executed.

-  1. Green: this assembly instruction was executed.
-  2. Gray: this assembly instruction was not executed.
-  3. Orange: a Branch was always not taken.
-  4. Cyan: the Branch was always taken.
-  5. Light Gray: there is no assembly instruction here.
-  6. RED: Breakpoint is set here.
-  7. Next instruction to be executed.



```
Disassembly
0x08001042 F1B07F80 CMP      r0,#0x1000000
0x08001046 D300    BCC      0x0800104A
0x08001048 E01C    B        0x08001084
1141: SysTick->LOAD = (ticks & SysTick_1
0x0800104A F020417F BIC      r1,r0,#0xFF0000
0x0800104E 1E49    SUBS     r1,r1,#1
0x08001050 F04F22E0 MOV      r2,#0xE000E000
0x08001054 6151    STR      r1,[r2,#0x14]
1142: NVIC_SetPriority (SysTick_IRQn, (1
0x08001056 BF00    NOP
1014: if (IRQn < 0) {
0x08001058 1751    ASRS      r1,r2,#29
0x0800105A 2900    CMP      r1,#0x00
0x0800105C DA05    BGE      0x0800106A
1015: SCB->SHP[ (uint32_t) (IRQn) & 0xF
1016: else {
0x0800105E 210F    MOVS      r1,#0x0F
0x08001060 0109    LSLS     r1,r1,#4
```

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).

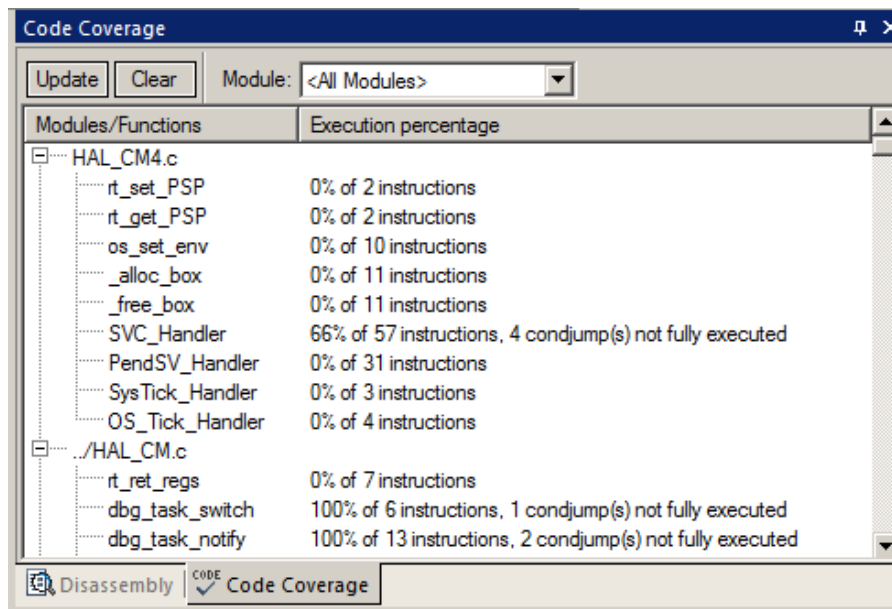
Why was the branch BCC always taken resulting in 0x0800_1048 never being executed ? Or why the branch BGE at 0x800_105C was never taken ? You should devise tests to execute these instructions so you can test them.

Code Coverage tells what assembly instructions were executed. It is important to ensure all assembly code produced by the compiler is executed and tested. You do not want a bug or an unplanned circumstance to cause a sequence of untested instructions to be executed. The result could be catastrophic as unexecuted instructions cannot be tested. Some agencies such as the US FDA require Code Coverage for certification.

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

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

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage. Note your display may look different due to different compiler options.



Modules/Functions	Execution percentage
HAL_CM4.c	
rt_set_PSP	0% of 2 instructions
rt_get_PSP	0% of 2 instructions
os_set_env	0% of 10 instructions
_alloc_box	0% of 11 instructions
_free_box	0% of 11 instructions
SVC_Handler	66% of 57 instructions, 4 condjump(s) not fully executed
PendSV_Handler	0% of 31 instructions
SysTick_Handler	0% of 3 instructions
OS_Tick_Handler	0% of 4 instructions
../HAL_CM.c	
rt_ret_regs	0% of 7 instructions
dbg_task_switch	100% of 6 instructions, 1 condjump(s) not fully executed
dbg_task_notify	100% of 13 instructions, 2 condjump(s) not fully executed

Saving Code Coverage information:

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.

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 binary file that can be later loaded back into μ Vision. Use the command Coverage Save *filename*.
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 ; you can also specify a module or function.
 - 3) log off ; turn the log function off.

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

```
\\Blinky\HAL_CM4.c\rt_set_PSP - 0% (0 of 2 instructions executed)
\\Blinky\HAL_CM4.c\rt_get_PSP - 0% (0 of 2 instructions executed)
\\Blinky\HAL_CM4.c\os_set_env - 0% (0 of 10 instructions executed)
\\Blinky\HAL_CM4.c\alloc_box - 0% (0 of 11 instructions executed)
\\Blinky\HAL_CM4.c\free_box - 0% (0 of 11 instructions executed)
\\Blinky\HAL_CM4.c\SVC_Handler - 66% (38 of 57 instructions executed)
    4 condjump(s) or IT-block(s) not fully executed
\\Blinky\HAL_CM4.c\PendSV_Handler - 0% (0 of 31 instructions executed)
\\Blinky\HAL_CM4.c\SysTick_Handler - 0% (0 of 3 instructions executed)
\\Blinky\HAL_CM4.c\OS_Tick_Handler - 0% (0 of 4 instructions executed)
\\Blinky\..\HAL_CM.c\rt_ret_regs - 0% (0 of 7 instructions executed)
\\Blinky\..\HAL_CM.c\dbg_task_switch - 100% (6 of 6 instructions executed)
    1 condjump(s) or IT-block(s) not fully executed
\\Blinky\..\HAL_CM.c\dbg_task_notify - 100% (13 of 13 instructions executed)
```

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

```
EX | 0x080004D6 468D      MOV          sp,r1
EX | 0x080004D8 4770      BX            lr
\\Blinky\Blinky.c\__use_no_semihosting_swi - 0% (0 of 1 instructions executed)
NE | 0x080004DA __I$use$semihosting:
NE | 0x080004DA 4770      BX            lr
\\Blinky\Blinky.c\_fp_init - 100% (3 of 3 instructions executed)
EX | 0x0800443C _fp_init:
EX | 0x0800443C F04F7040 MOV          r0,#0x3000000
EX | 0x08004440 EEE10A10 VMSR          FPSCR, r0
EX | 0x08004444 __fplib_config_fpu_vfp:
EX | 0x08004444 4770      BX            lr
```

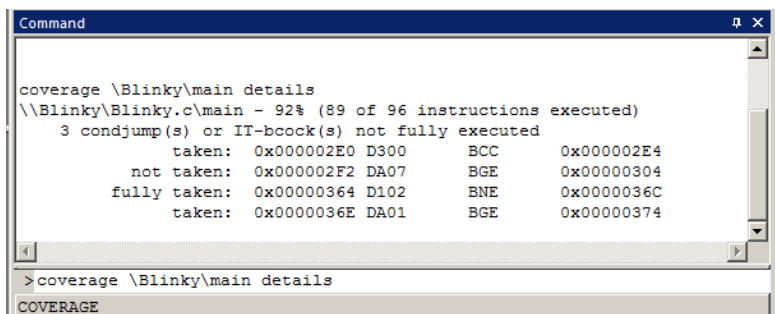
3) The first column above describes the execution as follows:

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)

4) Shown here is an example using:
coverage \Blinky\main details

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

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




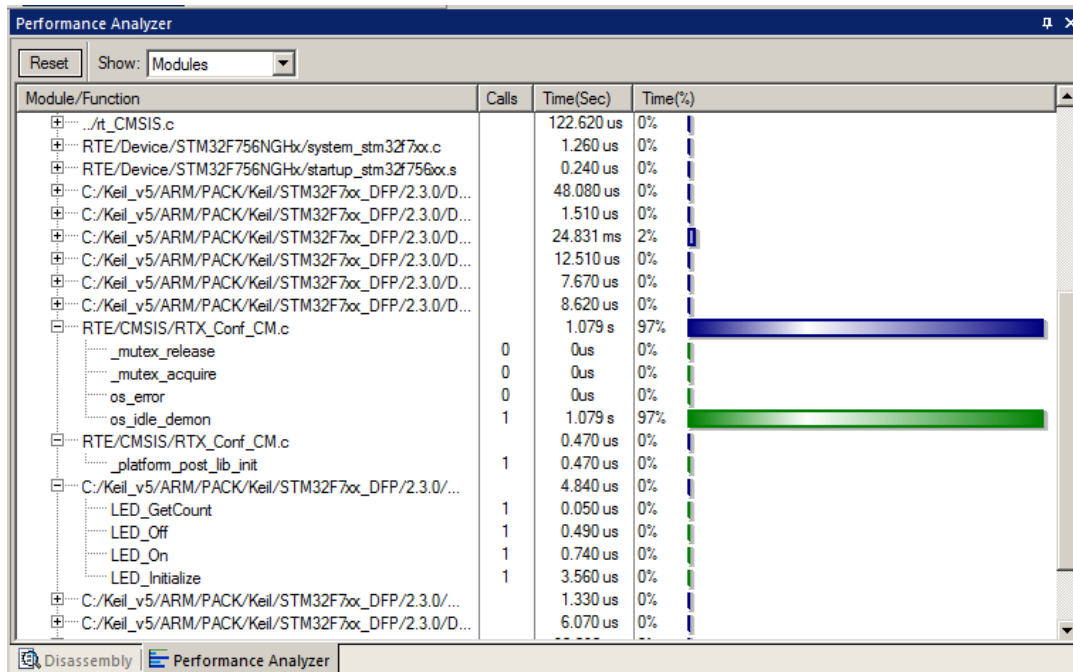
```
Command
coverage \Blinky\main details
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
    3 condjump(s) or IT-block(s) not fully executed
        taken: 0x000002E0 D300      BCC      0x000002E4
        not taken: 0x000002F2 DA07      BGE      0x00000304
        fully taken: 0x00000364 D102      BNE      0x0000036C
        taken: 0x0000036E DA01      BGE      0x00000374
>coverage \Blinky\main details
COVERAGE
```


Performance Analysis (PA):

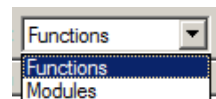
Performance Analysis tells you how much time was spent in each function. The data can be provided by either the SWV PC Samples or the ETM. If provided by the SWV, the results will be statistical and more accuracy is improved with longer runs. Small loops could be entirely missed. ETM provides complete Performance Analysis. Keil provides only ETM PA.


Keil provides Performance Analysis with the μ Vision simulator or with ETM and the ULINK μ pro. SWV PA is not offered. 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. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it.  This clears the PA window and resets the STM32 and reruns it to main() as before. Or select the Reset icon in the PA window to clear it. Run the program for a short time.
4. Expand some of the module names as shown below.
5. Note the execution information that has been collected in this initial short run. Both times and number of calls is displayed.
6. We can tell that most of the time at this point in the program has been spent in the os_idle-demon.




7. Click on the RUN icon. 
8. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the information. No code stubs are needed in your source files.
9. Select Functions from the pull down box as shown here and notice the difference.
10. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.
11. When you are done, exit Debug mode.

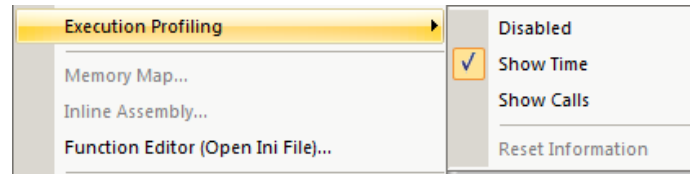


TIP: You can also click on the RESET icon  but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

Execution Profiling:

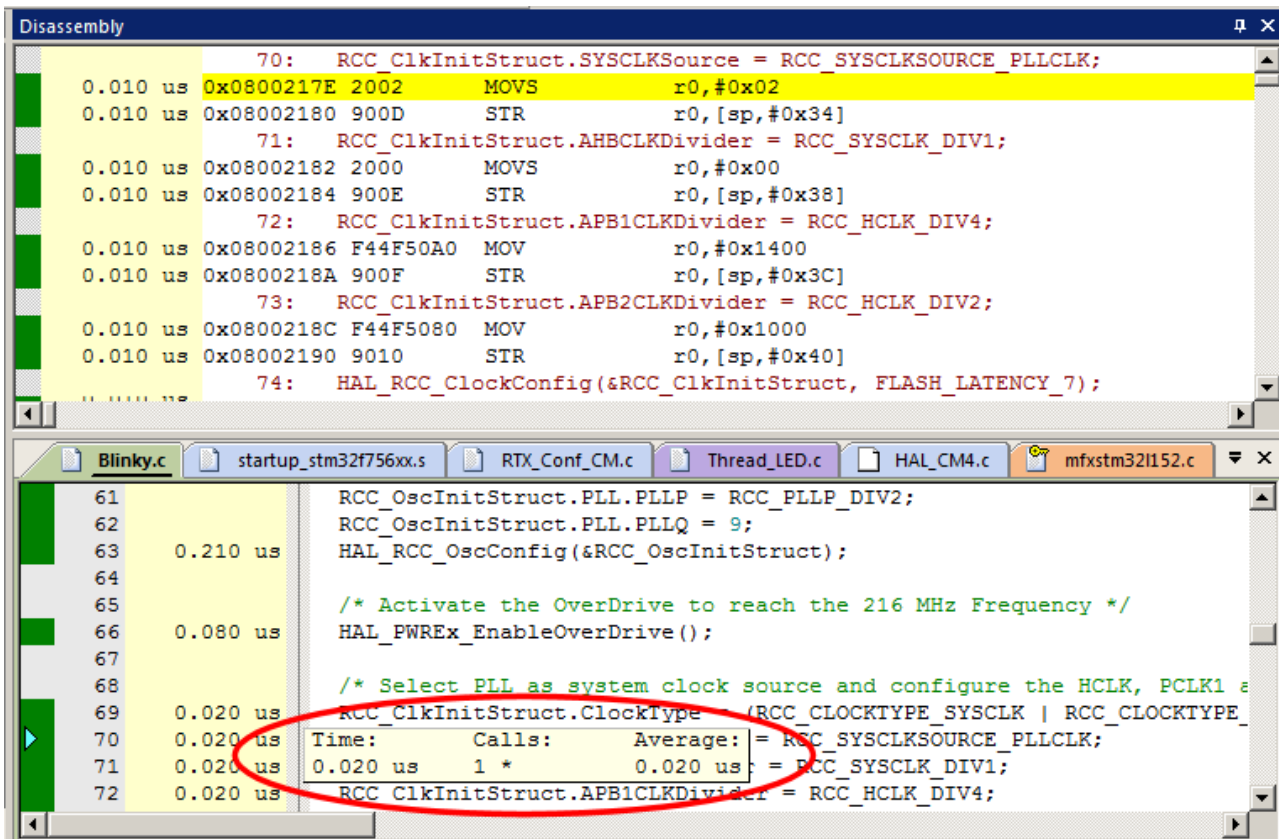
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace. It is possible to group source lines (called collapse) to get combined times and number of calls. This is called Outlining. The μ Vision simulator also provides Execution Profiling.

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



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

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



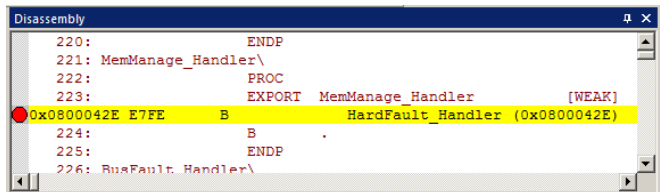
In-the-Weeds Example: A ULINKpro is needed for ETM Instruction Trace:

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 task switching plus unexpected and spurious events. Having a recording of the program flow is useful especially when a problem occurs and the consequences are not immediately visible. Another problem is detecting race conditions and determining how to fix them. ETM trace handles these problems and others easily and is not hard to use.

If a Hard Fault occurs, the CPU will end up at the address specified in the Hard Fault vector located at 0x0800 042E. This address points to the Hard Fault handler. This is usually a branch to itself and this Branch instruction will run forever. The trace buffer will save millions of the same branch instructions. This is not useful. We need to stop the CPU at this point.



This exception vector is found in the file startup_stm32f756xx.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 Hard Fault handler (in this case located at 0x0800 042E as shown to the right) and stop.

The CPU and also the trace collection will stop. The trace buffer will be visible and extremely useful to investigate and determine the cause of the crash.

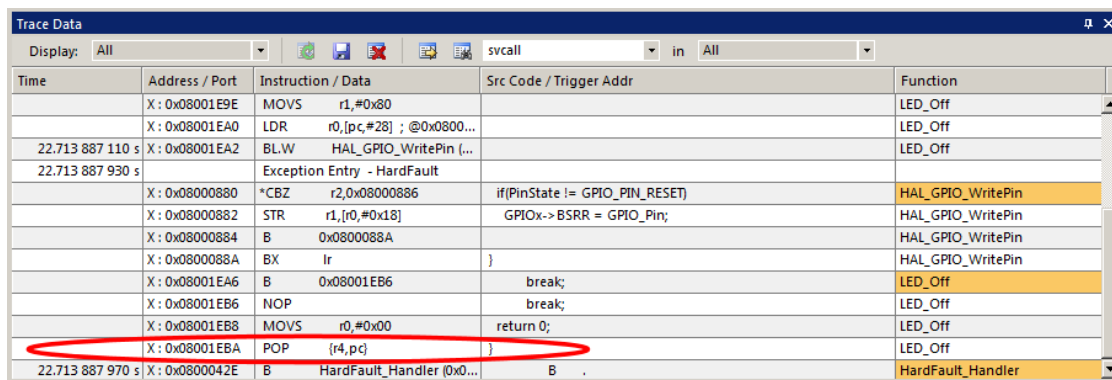


1. Open the Blinky example and enter Debug mode. Open the Trace Data window.
2. Locate the Hard Fault vector near line 223 in the disassembly window or in startup_stm32f756xx.s near line 219.
3. Set a breakpoint at this point. A red circle will appear as shown above.


TIP: An alternative to setting a breakpoint on the Hard fault vector: www.keil.com/support/docs/3782.htm

4. In Thread_LED.c, set another breakpoint on call to LED_Off(led_num); near line 29.
5. Click on Step Into  to enter the function LED_Off. You can verify this in the Call Stack and Locals window.
6. In the Registers window, set R14(LR) to 0x0. This will guarantee a Hard Fault on the next return.
7. Clear the Trace Data window for clarification purposes.
8. Click on RUN  and immediately the program will stop on the Hard Fault exception branch instruction.
9. Examine the Trace Data window and you find a POP plus everything else that was previously executed.
10. Double click on the POP instruction and this will be displayed in the Disassembly window.

TIP: The addresses you get might be different than these ones depending on compiler settings.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x08001E9E	MOVS r1,#0x80		LED_Off
	X: 0x08001EA0	LDR r0,[pc,#28] ; @0x0800...		LED_Off
22.713 887 110 s	X: 0x08001EA2	BLW HAL_GPIO_WritePin [...]		LED_Off
22.713 887 930 s		Exception Entry - HardFault		
	X: 0x08000880	*CBZ r2,0x08000886	if(PinState != GPIO_PIN_RESET)	HAL_GPIO_WritePin
	X: 0x08000882	STR r1,[r0,#0x18]	GPIOx->BSRR = GPIO_Pin;	HAL_GPIO_WritePin
	X: 0x08000884	B 0x0800088A		HAL_GPIO_WritePin
	X: 0x0800088A	BX lr	}	HAL_GPIO_WritePin
	X: 0x08001EA6	B 0x08001EB6	break;	LED_Off
	X: 0x08001EB6	NOP	break;	LED_Off
	X: 0x08001EB8	MOVS r0,#0x00	return 0;	LED_Off
	X: 0x08001EBA	POP {r4,pc}	}	LED_Off
22.713 887 970 s	X: 0x0800042E	B HardFault_Handler (0x0...	B .	HardFault_Handler

11. The Branch at the Hard Fault does not show in the trace window because a hardware breakpoint does not execute the instruction it is set to. It is therefore not recorded in the trace buffer. Click Step Into  and it will be executed as and displayed as shown above.
12. The frames above the POP are a record of all previous instructions executed and tells you the complete program flow from when you clicked RUN to when the event that caused the Hard Fault occurred.
13. Instruction Trace is very useful for locating difficult bugs. This is a contrived example but it is clear to see the usefulness of ETM. See the next page for a list of uses ETM and SWV can be.

33) Serial Wire Viewer and ETM Trace Summary:

Serial Wire Viewer can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers – DMA bypasses CPU and SWV by definition.

Serial Wire Viewer displays in various ways:

- PC Samples.
- Data reads and writes.
- Exception and interrupt events.
- CPU counters.
- Timestamps.

ETM Trace is good for:

- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened* as opposed to the way the program was written.
- Trace can often find nasty problems very quickly. Weeks or months can be replaced by minutes.
- Especially 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).

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

- Pointer problems. Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), a corrupted stack.
How did I get here ?
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. ETM trace is best for this.
- Communication protocol and timing issues. System timing problems.
- ETM facilitates Code Coverage, Performance Analysis and program flow debugging and analysis.

For information on Instruction Trace (ETM) please visit www.keil.com/st for other labs discussing ETM.

34) Document Resources:

See www.keil.com/st

Books:

1. **NEW!** Getting Started MDK 5: Obtain this free book here: www.keil.com/mdk5/.
2. There is a good selection of books available on Keil.com: www.keil.com/books/armbooks.asp
3. and on Arm.com: www.arm.com/resources/education/textbooks
4. μ Vision contains a window titled Books. Many documents including data sheets are located there.
5. A list of Arm CPUs is located at: <https://www.arm.com/products/silicon-ip-cpu>
6. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
7. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
8. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

Application Notes: www.keil.com/appnotes

9. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
10. Segger emWin GUIBuilder with μ Vision™ www.keil.com/appnotes/files/apnt_234.pdf
11. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
12. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
13. Using μ Vision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
14. RTX CMSIS-RTOS in MDK 5 C:\Keil_v5\ARM\Pack\ARM\CMSIS\3.20.4\CMSIS_RTXDownload
15. RTX CMSIS-RTX www.keil.com/demo/eval/rtx.htm and www.arm.com/cmsis
16. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
17. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
18. Cortex Debug Connectors: www.arm.com and search for cortex_debug_connectors.pdf
19. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
20. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
21. **NEW!** ARMv8-M Architecture Technical Overview <https://community.arm.com/docs/DOC-10896>
22. **NEW!** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp
23. Arm GCC: <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>

Keil Tutorials for STMicroelectronics Boards: see www.keil.com/st

Useful ARM Websites:

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

For comments or corrections on this document please email bob.boys@arm.com.

35) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- **MDK-Lite™** (Evaluation version) 32K Code and Data Limit - \$0
- **MDK-ST:** for STM32L0 and STM32F0: www.keil.com/st **\$0**
- **New MDK-ARM-Essential™** For all Cortex-M series processors – unlimited code limit
- **New MDK-Plus™** MiddleWare Level 1. ARM7™, ARM9™, Cortex-M, SecureCore®.
- **New MDK-Professional™** MiddleWare Level 2. For details: www.keil.com/mdk5/version520.

Free MDK for STM32 F0/L0

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

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

USB-JTAG adapter (for Flash programming too)

- **ULINK2** - (ULINK2 and ME - SWV only – no ETM) **ULINK-ME** – Equivalent to a ULINK2.
- **New ULINKplus-** Cortex-Mx High performance SWV & power measurement.
- **ULINKpro** - SWV & ETM trace. Fast Flash programming speed. Also works with DS-5.
- **SULINKpro D** - SWV & no ETM. Fast Flash programming speed. Also works with DS-5.

- **For special promotional or quantity pricing and offers, please contact Keil Sales.**
- **In USA and Canada, 1-800-348-8051.** sales.us@keil.com
- **Outside the US: +49 89/456040-20** sales.intl@keil.com

RTOS: Keil RTX RTOS is now provided as part of CMSIS 5: **New** https://github.com/ARM-software/CMSIS_5

DSP: Keil provides free DSP libraries with source code for Cortex-M processors.

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

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

Call Keil Sales for special university pricing. Go to www.arm.com/university to view various programs and resources.

MDK supports all STM32 Cortex-M0, M3, M4 and M7 processors. Keil supports many other ST processors including 8051, ARM7, ARM9™ and ST10 processors. See the Keil Device Database® on www.keil.com/dd.

For Linux, Android, other OSs and no OS support on ST Cortex-A processors such as SPEAr, see DS-5 www.arm.com/ds5



For more information:

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

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

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

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

