

The latest version of this document is here: [www.keil.com/appnotes/docs/apnt\\_286.asp](http://www.keil.com/appnotes/docs/apnt_286.asp)

## Introduction:

The purpose of this tutorial is to introduce you to the STMicroelectronics ST-Link/V2 debug adapter using the ARM® Keil™ MDK toolkit featuring the IDE μVision®. We will use a STM32F401C Discovery Kit in this tutorial. This tutorial can be adapted to any board with an ST-Link/V2. ST-Link/V2 supports Serial Wire Viewer (SWV) data trace.

Keil MDK-Lite™ is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn MDK into a full commercial version. MDK is free for STM32 F0/L0 Cortex®-M0 processors. See [www.keil.com/st](http://www.keil.com/st) or contact Keil Sales for more information regarding evaluation licenses.

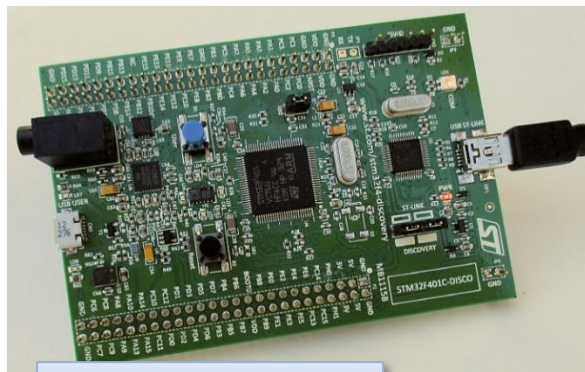
This tutorial describes practical methods to use ST-Link/V2 and SWV properly. It is in a hands-on format.

## Hardware and Software Requirements:

1. Any board with an on-board ST-Link/V2 or an external ST-Link/V2. Includes Discovery, Nucleo and EVAL boards. This tutorial uses a STM32F401 Discovery. An on-board ST-Link can be used with an external STM32 board. It should be easy to modify this tutorial for use with any STM32 board.
2. Keil MDK V 5.20 or later. The free evaluation version (MDK-Lite) will be used. No license is needed.
3. The appropriate Software Pack for your ST processor is installed in μVision.
4. A valid MDK 5 project. You can use any example provided by Keil in the Software Pack for your processor.

## This document details these features:

1. Serial Wire Viewer (SWV) data trace using ST-Link/V2. Real-time tracing is 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.
3. Six Hardware Breakpoints (can be set/unset on-the-fly)
4. Watch, Memory and System Viewer windows.
5. Display up to four variables with the Logic Analyzer.
6. A printf utility using ITM Port 0 or Event Recorder.
7. Display interrupts in Exception Trace and Trace Records.
8. Data Writes display in the Trace Records window.
9. RTX Viewers: Kernel awareness windows for the Keil RTX RTOS update while your program is running. FreeRTOS is now supported.
10. Determining and confirming the CPU frequency using SWV.



STM32F401C-Disco board  
with an on-board ST-Link/V2



Standalone ST-Link/V2  
connected to a Keil  
MCBSTM32F400 board.

## Serial Wire Viewer (SWV): ST-Link/V2 Supports SWV.

SWV provides to the μVision elements of your program such as Exceptions (interrupts), data writes, variable values and more all while your program is running. SWV is nearly always non-intrusive (Writes to ITM registers are not) and your program does not need to be stopped to display such elements. No code stubs are needed as CoreSight™ technology provides this in hardware.

SWV provides useful views into the internal workings of your program in order to more easily debug it. Four variables can be displayed graphically in the Logic Analyzer. Interrupt events are displayed live to help find tricky bugs. SWV is easy to configure and easy to interpret the results.

1. Keil Software Overview: MDK 5:	3
2. Keil MDK Core Software Download and Installation:	3
3. Getting Started Guide MDK 5:	3
4. STMicroelectronics Evaluation Boards:	3
5. Three main methods to create your own $\mu$ Vision projects:	3
6. DAP Debug Access Port:	3
7. Serial Wire Viewer (SWV):	3
8. Embedded Trace Macrocell (ETM):	3
9. CoreSight Definitions:	4
10. $\mu$ Vision Software Pack Download and Install Process:	5
11. Install the ST-Link V2 USB Drivers and Update the ST-Link Firmware:	6
12. Testing the ST-Link/V2 Connection:	6
13. <i>Blinky</i> example: STM32F401C Discovery board:	7
14. Blinky Program Highlights:	7
15. Hardware Breakpoints:	8
16. Display CPU Frequency:	8
17. Watch and Memory windows and how to use them:	9
18. System Viewer (SV): Peripheral Views:	10
19. Configuring Serial Wire Viewer (SWV):	11
20. Display Variables Graphically with the Logic Analyzer (LA):	12
21. Watchpoints: Access Breakpoints:	13
22. printf with ITM (Instrumentation Trace Macrocell):	14
23. <b>NEW!</b> Event Recorder printf for Cortex-M0 STM32F0 using DAP:	15
24. Trace Records Window (TR):	16
25. Exceptions (including Interrupts)	17
26. Data Writes in Trace Records:	18
27. RTX System and Threads Viewer:	19
28. RTX Event Viewer:	19
29. Using SWV to Determine and/or Confirm CPU Speed:	20
30. Serial Wire Viewer (SWV) Configuration Window:	21
31. Serial Wire Viewer and ETM summary:	22
32. Document Resources:	23
33. Keil Products and contact information:	24

## Notes on using this document:

1. The latest version of this document is located here: [www.keil.com/appnotes/docs/apnt\\_286.asp](http://www.keil.com/appnotes/docs/apnt_286.asp)
2. ARM MDK 5.20 and Software Pack STM32F7xx\_DFP 2.9.0 were used in the exercises in this document.
3. The ST-Link/V2 interfaces very well with Keil  $\mu$  Vision and its SWV performance is quite good.
4. The ST-Link (no V2 suffix) does not support Serial Wire Viewer.

## 1) Keil Software Overview: 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. A Software pack contains headers, Flash programming, CMSIS files and documents in a regular .zip file. The extension is renamed .pack. You need not wait for the next version of MDK or install patches to get the latest processor specific files. STM32CubeMX provides a project in MDK 5 format consistent with Software Packs. Keil Middleware supports STM32.

## 2) Keil MDK Core Software Download and Installation:

Download MDK-Core

1. Download MDK Core from the Keil website. [www.keil.com/mdk5/install](http://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. You do not need any debug adapters: just the Discovery board, a USB cable and MDK 5 installed on your PC.
4. You can use the evaluation version (MDK-Lite) for this lab. No license is needed.

## 3) Getting Started Guide MDK 5: Obtain this useful book here: [www.keil.com/gsg/](http://www.keil.com/gsg/).

## 4) STMicroelectronics Evaluation boards:

This tutorial can support the three types of ST evaluation boards or a custom board: A board needs is a ST-Link/V2 and a ST Cortex-M0, M3, M4 or M7 processor.

This tutorial uses a STM32F401C-Discovery board. For other boards you need to select the correct Software pack and example. Serial Wire Viewer (SWV) is not supported on a Cortex-M0. Only Cortex-M3, M4 and M7 are supported.

To use any ST-Link/V2 equipped board, download the appropriate Pack and the existing RTX\_Blinky example.

1. **Nucleo:** On many Nucleo boards the ST-Link can be "snapped off" and used with  $\mu$ Vision as an external ST-Link.
2. **Discovery:** As pictured on the front page. This tutorial uses this board.
3. **EVAL:** These boards are the high end and have all features implemented. They usually have an ETM connector.

## 5) There are three main methods to create your own $\mu$ Vision projects:

- 1) **STM32CubeMX.** This configures your processor and exports a  $\mu$ Vision project in MDK 5 format. You can obtain STM32CubeMX for your processor by going to [www.st.com](http://www.st.com) and search for stm32cube.
- 2) **Standard Peripheral Libraries** from ST (legacy). Contains extensive examples and source code for Keil MDK 5. It is recommended to use STM32CubeMX instead.
- 3)  **$\mu$ Vision Software Packs, examples and Keil Middleware.** A Software Pack includes examples and files that you can use. See [www.keil.com/pack/doc/STM32Cube/General/html/](http://www.keil.com/pack/doc/STM32Cube/General/html/)

## 6) DAP Debug Access Port:

DAP allows reads and writes to your target while the program is running through the JTAG or SWD ports. It is nearly always non-intrusive. No code stubs are needed in your program. Watch, Memory, System Viewer (peripherals) and RTX Threads and System windows use this feature. DAP reads are periodic and set by  $\mu$ Vision. Do not confuse DAP with CMSIS-DAP which is a debug adapter standard from ARM. All Cortex-M0, M3, M4 and M7 have DAP.

## 7) Serial Wire Viewer (SWV):

**Serial Wire Viewer** (SWV) displays PC Samples, Exceptions (includes interrupts), data reads and writes, ITM (printf), CPU counters and timestamps. This information comes from the ARM CoreSight™ debug module integrated into the STM32. SWV does not steal any CPU cycles and is completely non-intrusive. (except for the ITM Debug printf Viewer). SWV data is output on the 1 bit SWO (Serial Wire Output) pin on the debug connector with ST-Link/V2. SWV is not available on the Cortex-M0. It is available only on Cortex-M3, M4 and M7.

Up to four variables can be displayed graphically in the  $\mu$ Vision Logic Analyzer.

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

ETM records and displays all instructions that were executed. This is useful for debugging program flow problems such as "going into the weeds" and "how did I get here?". ETM requires a Keil ULINK™*pro* and a 20 pin ETM connector such as on STM32756G\_EVAL. Code Coverage and Performance Analysis is provided by ETM. ETM is not explored in this tutorial.

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

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

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except Boundary Scan is not possible. SWD is referenced as SW in the  $\mu$ Vision Cortex-M Target Driver Setup. See page 4, 2<sup>nd</sup> picture. The SWJ box must be selected in ULINK2/ME or ULINK $pro$ . 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.  $\mu$ Vision 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.  $\mu$ Vision 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. SWV and SWO are only on Cortex-M3, M4 and M7 processors. They are not on Cortex-M0 or M0+.
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 ULINK $pro$  uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by  $\mu$ Vision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations.  $\mu$ Vision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK $pro$  provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis.
10. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK $pro$ . ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal 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. Keil documents also refer to these as Access Breakpoints.
14. **ELF/DWARF:** The ARM compiler produces an .axf file which is ELF/DWARF compliant.  $\mu$ Vision 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  $\mu$ Vision.

### Additional Serial Wire Viewer (SWV) Information:

Serial Wire Viewer data (frames) are output on the 1 pin SWO. This pin is located on 13 on the 20 pin legacy connector (the BIG one). It is also available on the 10 or 20 pin CoreSight Debug connectors. SWO is on pin 6 on the 10 pin and pin 14 on the 20 pin. SWO is multiplexed with JTAG TDO pin. This means SWD (Serial Wire Debug) must be used and not JTAG mode. This is easily set in  $\mu$ Vision. SWD = SW in  $\mu$ Vision. ST-Link/V2 supports SWD only and not JTAG.

SWO is only one pin and it can be challenging to send a large amount of SWV data through it. A ULINK $pro$  using Manchester mode on the SWO pin is more efficient. For even more throughput, ULINK $pro$  can output SWV on the 4 bit Trace Port. This port is available on most ST Cortex-M3, M4 and M7. Cortex-M0 does not have SWV nor the Trace Port but has DAP read/write.


## 10) µVision Software Pack and Examples Download and Install Process:

Software Packs contain files needed for your project such as header, flash programming and example files. For complete information see [www.keil.com/pack/doc/CMSIS/Pack/html/](http://www.keil.com/pack/doc/CMSIS/Pack/html/)

These instructions reference a STM32F401C-DISCO board. If you are using a different board, install the appropriate Pack and example. If you already have the Pack and an example installed for your board or processor, you can skip this section.

**1) Start µVision and open Pack Installer (PI):** (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 might open. Read and close it.

4. This window opens up: Select the Boards tab. Type *stm* in the Search box to filter the listings: You can also select your processor under the Devices tab.

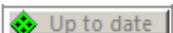
5. Select STM32F401C-Discovery:

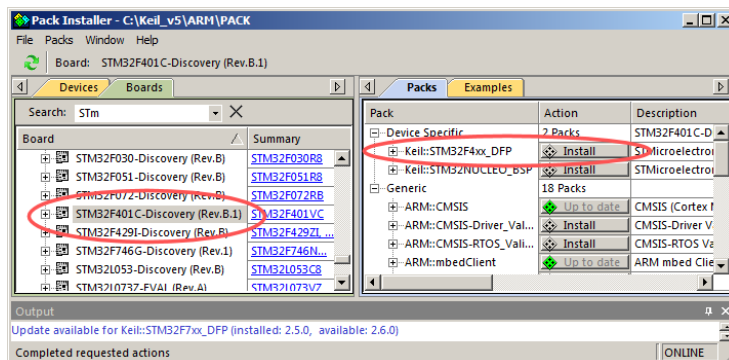
6. Select the Packs tab.

7. Click Install opposite STM32F4xx\_DFP:


8. The Pack will now install to your PC.

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

10. The Pack's status will then be indicated by the “Up to date” icon: 



**TIP:** What you select in the left side of the Pack Installer in the Devices or Boards tabs, determines what is filtered on the right side in the Packs and Examples tabs.

**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 CMSIS-RTOS RTX\_Blinky Example:

1. Select the Examples tab:


2. Select Copy beside CMSIS-RTOS Blinky: 

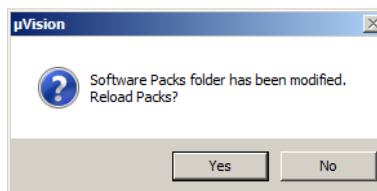
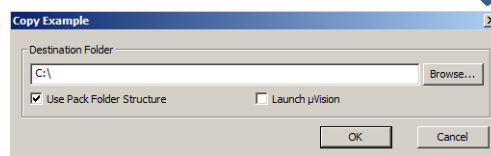
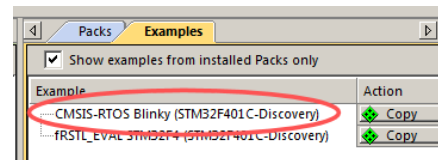
3. The Copy Example window below right opens up: Select Use Pack Folder Structure: Unselect Launch µVision: 

4. Type in C:\ as shown to the right: Click OK to copy Blinky into C:\MDK\Boards\ST\STM32F401C-Discovery\

**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.

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

6. When a window opens stating the Software Packs have changed, select Yes: 



## 11) Install the ST-Link V2 USB Drivers and Update the ST-Link Firmware:




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

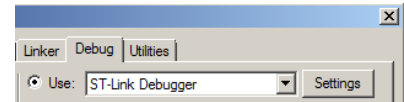
### 1) Install the ST-Link USB Drivers: *This step normally needs to be done just once !*

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 without much notice.
3. Plug in the Discovery board to USB CN14. The USB drivers will now finish installing in the normal fashion.

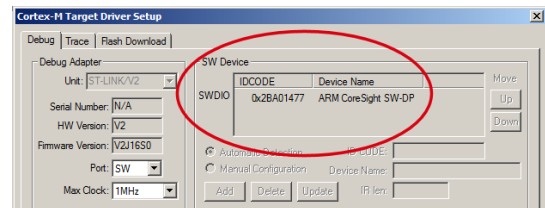
**2) Upgrading the ST-Link V2 Firmware:** The ST-Link V2 firmware updater utility ST-LinkUpgrade.exe is located here: C:\Keil\_v5\ARM\STLink\. It is a good idea to upgrade the ST-Link firmware. Find this file and double click on it to execute it. It will check and report the current firmware version. If you experience trouble, especially with SWV, update to the latest ST firmware and USB drivers.

## 12) Testing the ST-Link V2 Connection: (Optional)

1. Start  if it is not already running.
2. Connect the Discovery board USB ST-Link to your PC.
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 Project/Open Project.
6. Select the Blinky project C:\MDK\Boards\ST\STM32F401C-Discovery\Blinky\Blinky.uvprojx. (or any valid project)
7. Select Target Options  or ALT-F7 and select the Debug tab:
8. Select ST-Link Debugger: 
9. 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  $\mu$ Vision main menu.
10. A number in the SN: box means  $\mu$ Vision is connected to the ST-Link adapter. SW Device means  $\mu$ Vision is connected to the processor via ST-Link/V2.
11. If nothing or an error is displayed in this SW Device box, this *must* be corrected before you can continue. See the instructions above: Install the ST-Link USB Drivers:
12. Once you see a proper display, your ST-Link USB drivers are installed properly. Click OK twice to exit the Target Driver Setup windows. 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 JTAG mode.



### LED LD7 indication: (might not be applicable for all boards)

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).  $\mu$ Vision is not connected to ST-Link (i.e. in Debug mode).

LED is GREEN:  $\mu$ 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  $\mu$ 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  $\mu$ Vision.

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

### 13) Blinky example: STM32F401C Discovery board:

We will connect the Keil MDK development system to the Discovery board using the built-in ST-Link/V2 debug adapter.

1. Start  $\mu$ Vision by clicking on its desktop icon.  Connect your PC to the USB ST-Link connector.
2. Select Project/Open Project. Open C:\MDK\Boards\ST\STM32F4-Discovery\Blinky.uvprojx
3. By default, the ST-Link is selected. If this is the first time you have run  $\mu$ 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. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears. The Flash will be programmed. Flash programming progress will be indicated in the Output Window.
6. Click on the RUN icon.  Note: you stop the program with the STOP icon. 

**Four LEDs LD3 through LD6 on the STM32F401C Discovery board will blink in sequence. Press the blue *USER* button and the program will stop in an infinite loop in main thread.**

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

Other boards may have different or no LEDs. To confirm your program is running if no LEDs blink, set a breakpoint in an appropriate place in a source or Disassembly window. The program will stop when this instruction is fetched.

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

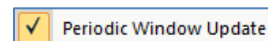
### 14) Blinky program highlights:

1. Blinky is running RTX RTOS with four threads: main, thread\_LED, os\_Timer\_Thread and os\_idle\_demon.
2. This example is sequences four LEDs on GPIO Port D.
3. Keil uses the term threads instead of tasks for consistency when discussing CMSIS-RTOS RTX.
4. There are similar Blinky examples for many other STM32 boards. These are selected in the Pack Installer.
5. This Blinky uses STM32Cube Framework (API) and STM32Cube HAL. See the Manage Run-Time Environment window for details.

**TIP:** Many  $\mu$ Vision windows update while the program is running. If a window updates only when you stop the program: make sure Periodic Window Update is enabled:

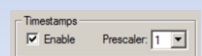
This is selected in Debug mode by selecting View in the main  $\mu$ Vision menu.

This window is enabled by default.







#### Top Reasons SWV does not work:

- Core Clock is incorrect. Use global variable SystemCoreClock to determine CPU speed. Note the CPU speed can change after the PLL is configured and SWV can stop. Use STM32CubeMX to calculate the clock tree.
- Check the Trace Records window: look for bogus ITM frames – should be only ITM 0 and 31 present.
- Enable PC Samples if necessary to create frames if interrupts or ITM are not used in your program.
- SWO pin is seriously overloaded. Unselect some SWV events. Always minimize active number of SWV events.
- Make sure View/Periodic Window Update is selected. Available only while in Debug mode.
- Make sure Timestamps are activated: Some, but not all, SWV features will turn off if unselected.
- Some ST boards have a solder bridge (SB) for SWO that needs to be soldered. The STM32F429I- Disco an example.



## 15) Hardware Breakpoints: **this uses neither SWV nor DAP:**

The STM32F4 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 a breakpoint is again encountered.
6. You can also click on Single Step (Step In) , Step Over  and Step Out .

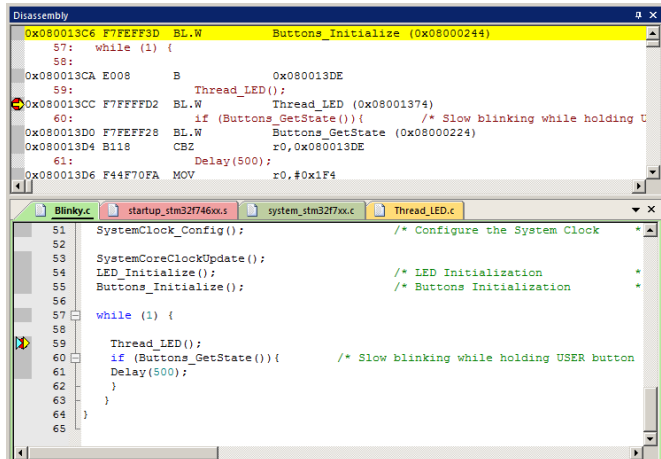
**TIP:** To single step (Step In) by assembly instruction, click on the Disassembly window to bring it into focus. To step by a C source line, bring any source window into focus.

**TIP:** A hardware breakpoint does not execute the instruction it is set to. Arm CoreSight breakpoints are no-skip. 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 by 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.



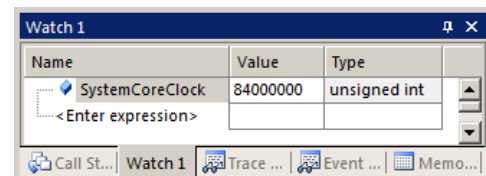
## 16) Display CPU Frequency: **Watch window uses DAP:**

The global variable SystemCoreClock contains the speed of the CPU. It is declared and set in system\_stm32f4xx.c.

1. Click on the RUN icon.  You can configure the Watch and Memory windows while your program is running.
2. Select View/Watch Windows/Watch 1 if Watch 1 is not already open.
3. Double click on <Enter expression> and enter SystemCoreClock and press the Enter key.
4. A value will appear. Right click on the variable name and unselect Hexadecimal Display.
5. The CPU frequency will now display in Hertz.
6. When Debug mode is first entered and the program counter is still at the start of main(), the CPU frequency will be 16 MHz. After SystemClock\_Config() is executed, it will be 84 MHz. Other boards may/will be different.

**TIP:** You can also open the file system\_stm32f4xx.c and right click on the variable SystemCoreClock. Select Add SystemCoreClock to ... and select Watch 1.

**TIP:** You can also find the CPU clock value using the Serial Wire Viewer configuration in the Target Driver Setup window. See page 19 for instructions.










## 17) Watch and Memory Windows and how to use them: Both use DAP:

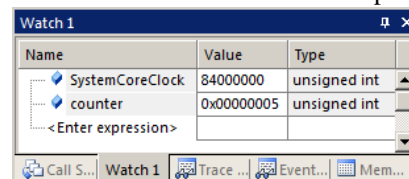
The Watch and Memory windows will display updated variable values in real-time. It is also possible to modify values into the Memory window while the program is running. You can modify a variable in a Watch window while the program is running if the variable is changing slowly. Otherwise, stop the program to make a change to a Watch variable.

### Watch window:

**Add a global variable:** The Watch and Memory windows can't see local variables unless the program is 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 89:
4. Select File/Save All or click . Click on Rebuild. .
5. Enter Debug mode.  Click on RUN . You can set Watch and Memory windows while the program is running.
6. 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: 
7. **counter** will increment to 0x0F and will update in real-time.
8. Note some values of counter might be skipped if it changes quickly because the Watch and Memory windows are updated periodically.

```
86 while (1) {  
87     while (Buttons_GetState() & 1U);  
88     osSignalSet(tid_Thread_LED, 1U);  
89     counter++;  
90     if (counter > 0x0F) counter = 0;  
91     osDelay(500);  
}
```



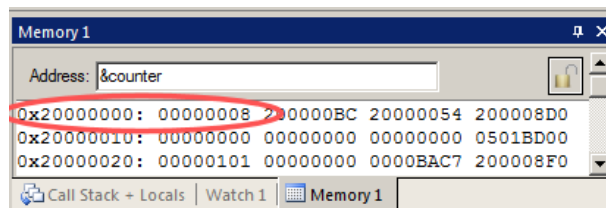
Name	Value	Type
SystemCoreClock	84000000	unsigned int
counter	0x00000005	unsigned int
<Enter expression>		

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

9. 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 (0x2000\_0000). This address might be different depending on memory and compiler settings.
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. Enter the new variable value and press Enter.



Address: &counter
0x20000000: 00000008 000000BC 20000054 200008D0
0x20000010: 00000000 00000000 00000000 0501BD00
0x20000020: 00000101 00000000 0000BAC7 200008F0

**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.

## 18) System Viewer (SV): Uses DAP:

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

It is possible to create your own custom System Viewer for specified peripherals:

[www.keil.com/support/man/docs/uv4/uv4\\_db\\_dbg\\_scvd\\_viewer.htm](http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_scvd_viewer.htm)

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

### GPIO Port D:

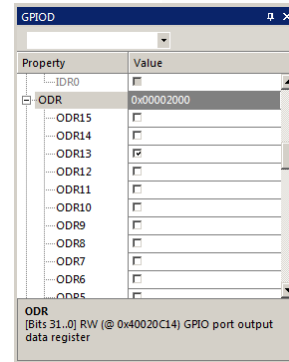
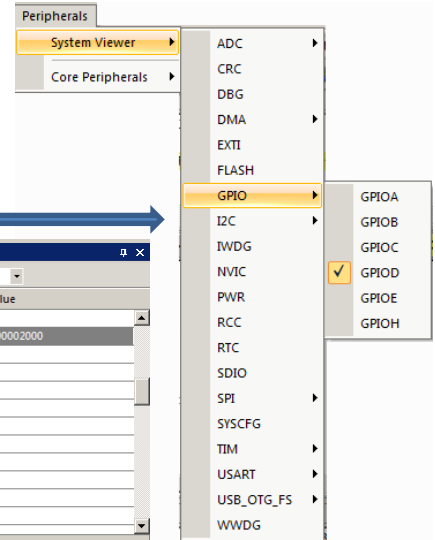
2. Select Peripherals/System Viewer, GPIO and then GPIOD as shown here:

3. This window opens up. Expand ODR:

4. You can see ODR12 through ODR15 update as the LEDs blink in succession: Other boards may/will use different ports.





5. These are updated periodically, not when the value changes. To view a register when it changes, you need to use the Logic Analyzer. It displays on a specified write operation.

6. You can modify register values while the program is running. Select one of the bits ODR15 through ODR12. The corresponding LED will come on until the program turns it off.



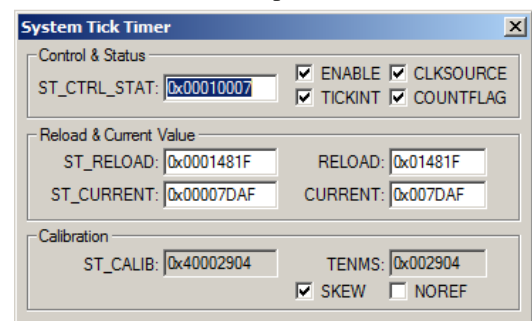
**TIP:** If you click on a register in the Property column, a description will appear at the bottom of the SV window. ODR is shown here:

**SysTick Timer:** This program uses the SysTick timer for RTX. This effectively controls how fast the program runs.

1. Select Peripherals/Core Peripherals and then select System Tick Timer.
2. The System Tick Timer window shown below opens: You might get different values with other boards or programs.
3. Note it also updates in real-time while your program runs. These windows use the same CoreSight DAP technology as the Watch and Memory windows. Do not confuse this DAP (Debug Access Port) with CMSIS-DAP.
4. Note the ST\_RELOAD and RELOAD registers. This is the reload register value set by RTX.
5. Note that it is set to 0x1481F. This is the hex value of 84,000,000/1000-1 that is programmed into SysTick in SysTick\_Config function in core\_cm4.h. The CPU clock in this example is 84 MHz.
6. In the RELOAD register in the SysTick window, *while the program is running*, type in 0x1000, press Enter and click inside ST\_RELOAD ! (or the other way around)
7. The blinking LEDs will speed up. This will convince you of the power of Arm CoreSight debugging.
8. Replace RELOAD with 0x1481F. A CPU RESET  and RUN  will also do this.
9. You can look at other Peripherals contained in the System View windows.
10. When you are done, stop the program  and close all the System Viewer windows that are open.
11. Exit Debug mode. 

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

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



## 19) Configuring Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) is a data trace. It is supported by ST-Link/V2 and  $\mu$ Vision. The output is on the 1 bit SWO pin. SWD must be used as JTAG will not work with SWV because of a conflict with the JTAG TDO pins. SWV is non-intrusive and does not need any code stubs in your program except for the ITM features. It is easy to configure.






**Serial Wire Viewer displays these items:**

1. Exceptions including Interrupts and the number and when they occur.
2. Data read and write operations. Currently  $\mu$ Vision displays only data writes.
3. Logic Analyzer (LA): display up to four variables in a graphical format. These are event driven.
4. printf: uses ITM Port 0 to output printf statements on a window in  $\mu$ Vision. RTX Event Viewer uses ITM Port 31.
5. CPU Counters. Display various events such as number of CPU cycles in Sleep mode.
6. PC Samples: A sampling of program counter values. Of limited value. ETM with a ULINKpro is much better.


**Important Things to Know about SWV:**

1. The 1 bit SWO is easily overloaded. It is important to enable only those features you really need.
2. You might need to sample variables used in LA since an event is created each the variable is accessed.
3. You must know the exact CPU frequency to configure the SWO pin to UART mode.
4. On a few boards (such as STM32F429I) you must short a solder bridge (SB.xx) on the SWO pin to get SWV to work.
5. A few boards might need an initialization file to configure I/O pins. This file is provided with the  $\mu$ Vision examples.

**Configure the SWV Trace:**

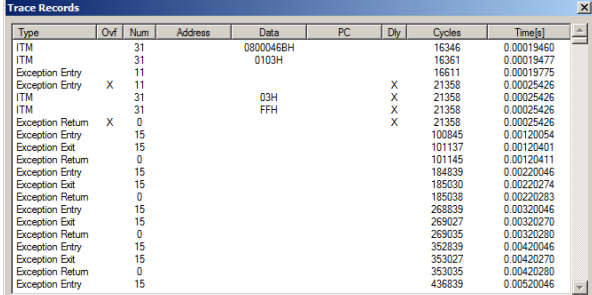
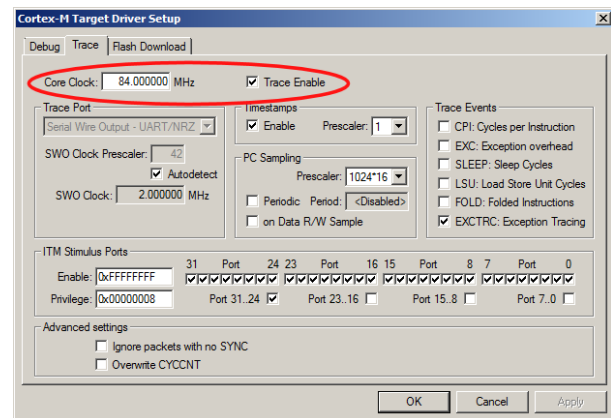
1. Stop the processor  and exit Debug mode. 
2. Select Target Options  or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window. Confirm SW is selected. SW selection is mandatory for SWV. ST-Link uses only SW.
3. Select the Trace tab.
4. In the Trace tab, select Trace Enable. Set Core Clock: to 84 MHz. Everything else is left at default.
5. Click OK twice to return to the Debug tab.
6. Enter Debug mode.  No need to recompile.
7. Click RUN  The program will run as before.

**Confirm SWV is working:**

1. Select View/Trace/Records or click on the arrow beside the Trace icon  and select Records.
2. The Trace Records window will open up:
3. If you see Exception 15 and ITM 31 frames, then SWV is working correctly.
4. If you see nothing or any ITM other than 0 or 31, the most probable cause is Core Clock: is incorrect. Check SystemCoreClock and enter the correct frequency. You can try potential numbers such as 8, 16, 42 MHz.

**You must get SWV working as described on this page before you can continue with this tutorial.**

1. Make sure the Core Clock: is correctly entered. This must be exact to configure the UART mode.
2. A few boards need a solder bridge connected to get SWV working. This connects the SWO pin.
3. Cycle the power to the board and restart  $\mu$ Vision.
4. Check for a signal on the SWO pin with an oscilloscope – this pin is found on the SWD connector.
5. If your program is not using exceptions or RTX, try enabling PC Samples in the trace setup window and try again.




Type	Cnt	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		0800046B			16346	0.00019460
ITM		31		0103H			16361	0.00019477
Exception Entry		11					16611	0.00019775
Exception Entry	X	11				X	21358	0.00025426
ITM		31		03H		X	21358	0.00025426
ITM		31		FFH		X	21358	0.00025426
Exception Return	X	0				X	21358	0.00025426
Exception Entry		15					100845	0.00120054
Exception Exit		15					101137	0.00120401
Exception Return		0					101145	0.00120411
Exception Entry		15					184839	0.00220046
Exception Exit		15					185030	0.00220274
Exception Return		0					185038	0.00220283
Exception Entry		15					268839	0.00320046
Exception Exit		15					269027	0.00320270
Exception Return		0					269035	0.00320280
Exception Entry		15					352839	0.00420046
Exception Exit		15					353027	0.00420270
Exception Return		0					353035	0.00420280
Exception Entry		15					436839	0.00520046

## 20) Display Variables graphically with the Logic Analyzer (LA): Uses SWV:


The Logic Analyzer can display up to four waveforms of a variable. This uses the same CoreSight hardware as the Watchpoints. An event is generated each time the variable is written to and the frame is output on the SWO pin to  $\mu$ Vision. This means a variable that changes quickly can overload the SWO pin. If you have this problem, you can write a routine to sample this variable. Display the sample variable in the LA.

1. Be in Debug mode.  Run the program. 

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

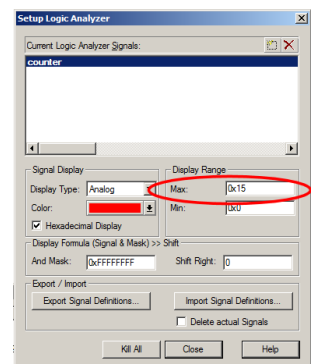
2. Select View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 
3. Position and size it for comfortable viewing.
4. Click on the Blinky.c tab. Right click on the **counter** and select Add counter to... and then select Logic Analyzer. You can also Drag and Drop or enter it manually.

**TIP:** If you are unable to enter a variable in LA, the cause is the SWV is not correctly configured. See the preceding page.

5. In the Logic Analyzer window, click on the Select box and the LA Setup window appears as shown here:
6. With `counter` selected, set Display Range Max: to 0x15 as shown here:
7. Click on Close.
8. Run the program if it is stopped. 
9. You should have a waveform visible at this point (or at least a red line)
10. In the LA window, click on Zoom Out until Grid is about 1 second.
11. 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 reprogram the Discovery board. Confirm the Core Clock: value is correct.

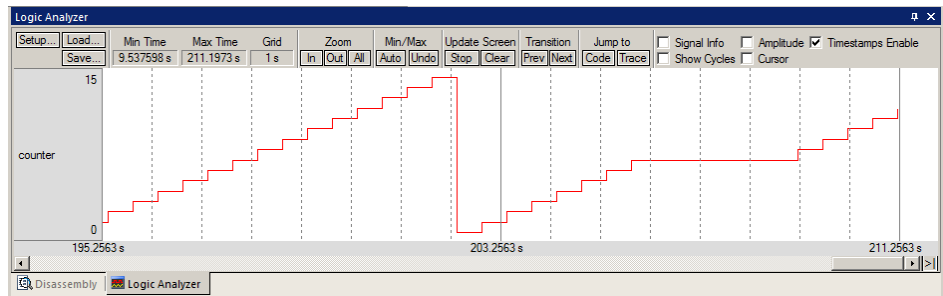
**TIP:** You can show up to 4 variables in the Logic Analyzer. These variables must be global, static or raw addresses such as \*((unsigned long \*)0x20000000). They can be a peripheral register memory address.



12. Press the blue User button and release: See counter stops incrementing as shown here:

13. Select Signal Info, Show Cycles, Amplitude and Cursor.

14. Stop the LA by clicking on the Stop icon in the Update Screen box. The program keeps running normally.



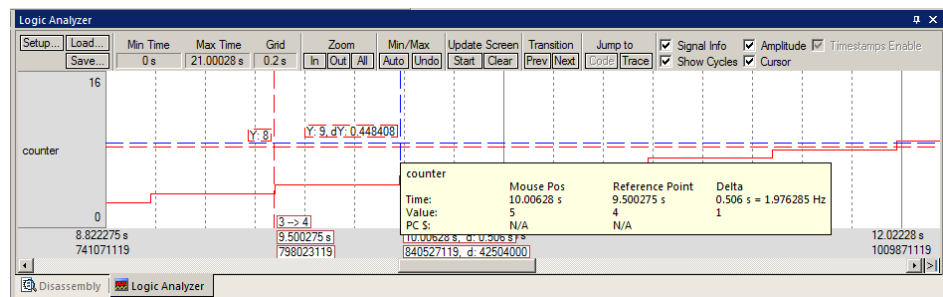
15. Click your mouse on the waveform at a point to measure time from.

16. Move the mouse cursor to another point to measure any particular time. Here each LED is on for 1/2 second.

17. When you are ready to continue, start the Update Screen.


18. On the next page we will look at the Trace Records window more closely.

**TIP:** When a variable is entered in the LA, the associated Data Write is also displayed in the Trace Records window.




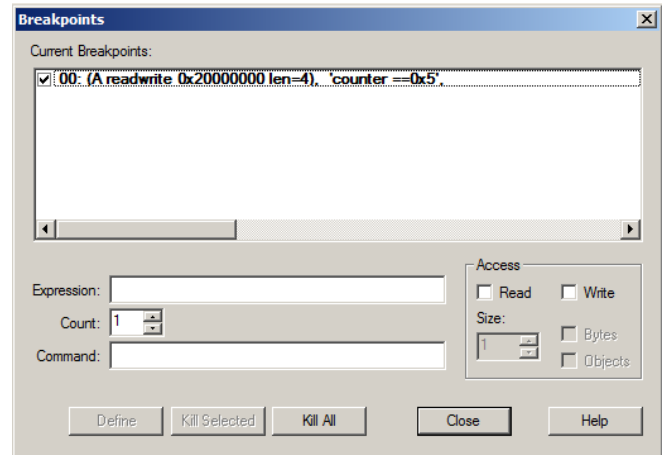
## 21) Watchpoints: Conditional Breakpoints: **uses neither SWV nor DAP:**

Recall STM32 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. The STM32 also have two Watchpoints.  $\mu$ Vision can currently use one. 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  $\mu$ Vision you must have two variables free in the Logic Analyzer to use Watchpoints. Watchpoints are also referred to as Access Breakpoints in Keil documentation.

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  $\mu$ Vision window and select Breakpoints or press Ctrl-B on your keyboard.
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. We unselect EXTRC (exceptions) to minimize SWO loading. This step may be optional.
10. Click on OK twice. Open the Trace Records window.




11. Double click inside the Trace Records to clear it.
12. Click on RUN. 
13. You will see **counter** increment in the Logic Analyzer as well as in the Watch 1 window.
14. When **counter** equals 0x5, the Watchpoint will stop the program.



**TIP:** To see a data write event without stopping the program, enter it into the Logic Analyzer (LA) window.

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. PC was enabled by on Data R/W Sample in step 9 above.
16. If you enter a variable with no assignment, the program will stop at the next access read or write or both as you select.
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.



Type	Of	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	00000002H	08000292H		672012173	4.0007246
Data Write			20000000H	00000003H	08000292H		714012170	4.25007244
Data Write			20000000H	00000004H	08000292H		756012170	4.50007244
Data Write			20000000H	00000005H	08000292H		798012170	4.75007244

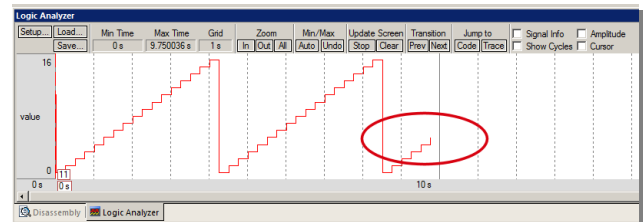
19. Leave Debug mode. 

**TIP:** You cannot set Watchpoints on-the-fly with ST-Link while the program is running. You can with a Keil ULINK.

**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.

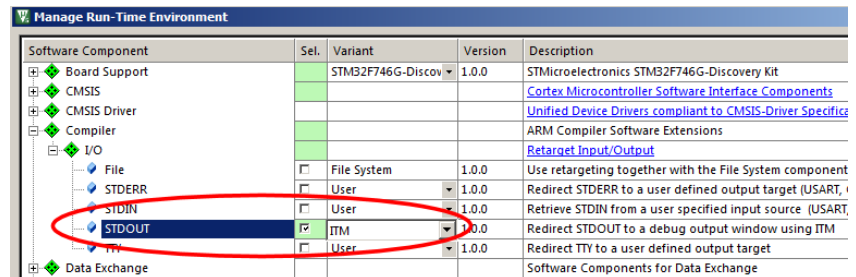
## 22) printf with ITM (Instrumentation Trace Macrocell): Uses SWV and ITM 0:






It is easy to incorporate printf using ITM and the  $\mu$ Vision utility Manage Run-Time Environment. This uses Serial Wire Viewer and works with Cortex-M3, M4, M7, M23 and M33 processors.

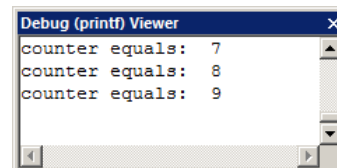
To provide printf for Cortex-M0 and M0+ processors, see the next page.

1. Stop the program if it is running  and exit Debug mode. .
2. Open the Manage Run-Time Environment utility.  This window opens:

3. Expand Compiler...I/O as shown.
4. Select STDOUT and ITM:
5. All the blocks should be green. If not, click on the Resolve button.
6. Click OK to close this window.
7. The file retarget\_io.c will be added to your project in the project window under the Compiler group.



8. In Blinky.c, near line 91 just after the if (counter>.... Line, add this line: printf("counter equals: %d\n", counter);
9. Select File/Save All or click .
10. Rebuild the source files .
11. Enter Debug mode . Click on RUN .
12. Select View/Serial Windows and select Debug (printf) Viewer.
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](http://www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm)

### Setting RAM for printf:

If your program doesn't work, you might have to do one or more of these items when implementing printf using ITM:

1. Increase RTX Thread Stack size. This is done in RTX\_Conf\_CM.C. Use the Configuration Wizard. Try 300 bytes.
2. If you do not have MicroLIB selected, you must add some heap in startup\_stm32f401xc.s or the correct startup.s file for your processor. Try adding a 200 byte heap. MicroLIB results in smaller executable size.

### 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](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.


**TIP:**  $\mu$ Vision icon meanings are found here: [www.keil.com/support/man/docs/uv4/uv4\\_ca\\_filegrp\\_att.htm](http://www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm)

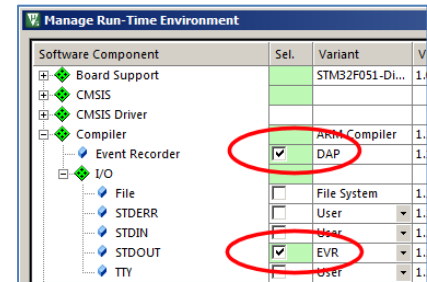
## 23) printf for STM32F0 Cortex-M0: Uses Event Recorder and CoreSight DAP:

**Event Recorder** is a new Vision feature that can be used to instrument your code. Keil RTX5 and Middleware is already instrumented with Event Recorder. Event Recorder can provide a printf utility using the DAP read/write abilities of the Debug Access Port. A UART is not used. This can be used with STM32F0 Cortex-M0 parts as well as any STM32F processor. This is the same technology used in Watch, Memory and Peripheral windows. This method can also be used with any Cortex-M processor. It does not use SWV as does the method on page 14.


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

### Configure Event Recorder:






2. Open the Manage Run-Time Environment utility.  This window opens:
3. Expand Compiler and I/O as shown.
4. Select Event Recorder and STDOUT and EVR as shown:
5. All the blocks should be green. If not, click on the Resolve button.
6. Click OK to close this window.
7. retarget\_io.c and EventRecorder.c will be added to your project under the Compiler group in the Project window.
8. Right click near the top of Blinky.c, and select Insert "#include" and select #include "EventRecorder.h".
9. At the beginning of the main() function, add this line: EventRecorderInitialize (EventRecordAll, 1);

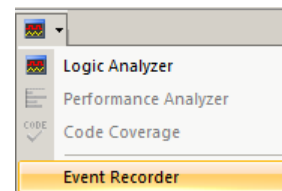
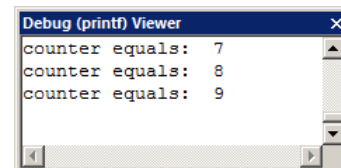


### Add a printf statement to Blinky.c:




1. In Blinky.c add #include "stdio.h" near the top of the file.
2. In Blinky.c, near line 91 just after the if (counter>.... Line, add this line: printf("counter equals: %d\n", counter);
3. Select File/Save All or click .

### Build and RUN the Blinky program and view printf:

1. Rebuild the source files .
2. Enter Debug mode . Click on RUN .
3. Select View/Serial Windows and select Debug (printf) Viewer.
4. The values of counter is displayed as seen here: 
5. Open the Event Recorder window: 
6. Information about the printf statements are displayed as shown below:
7. You can annotate your own sources and display events. See [www.keil.com/support/man/docs/uv4/uv4\\_db\\_dbg\\_evr.htm](http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr.htm)





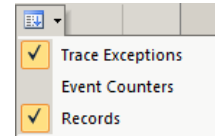
**TIP:** Keil Middleware and RTX5 are annotated using Event Recorder.

Event Recorder				
Enable Recorder: <input checked="" type="checkbox"/>    Mark: <input type="text"/> All Operations <input type="button" value="Recording"/>				
Event	Time (sec)	Component	Event Property	Value
0	0.00000000		Init Event	Restart Count=0x00000001
1	0.00000010	STDIO	stdout	0x63,0x6F,0x75,0x6E,0x74,0x65,0x72,0x20
2	0.00000020	STDIO	stdout	0x65,0x71,0x75,0x61,0x6C,0x73,0x3A,0x20
3	0.00000030	STDIO	stdout	0x31,0x0A,0x00,0x00,0x00,0x00,0x00,0x00
4	0.00000040	STDIO	stdout	0x63,0x6F,0x75,0x6E,0x74,0x65,0x72,0x20
5	0.00000050	STDIO	stdout	0x65,0x71,0x75,0x61,0x6C,0x73,0x3A,0x20
6	0.00000060	STDIO	stdout	0x32,0x0A,0x00,0x00,0x00,0x00,0x00,0x00
7	0.00000070	STDIO	stdout	0x63,0x6F,0x75,0x6E,0x74,0x65,0x72,0x20
8	0.00000080	STDIO	stdout	0x65,0x71,0x75,0x61,0x6C,0x73,0x3A,0x20

## 24) Trace Records (TR): Uses SWV:

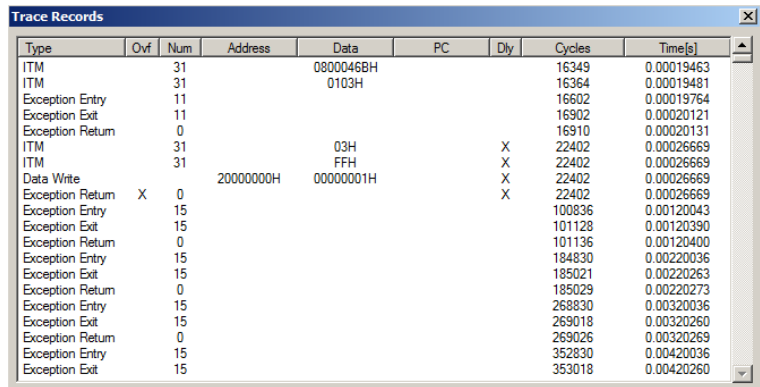
The Trace Records window displays the output of the SWO pin. The elements are selected in the Trace Config window. Any frame overloads or overruns are clearly shown.  $\mu$ Vision recovers gracefully from such issues.

1. Be in Debug mode.  Run the program. 
2. Open the Trace Records window by clicking on the small arrow beside the Trace icon:  
Select both Records and Trace Exceptions as shown here:
3. The Trace Records window will open as shown here:
4. Double-click inside this window to clear it.  
It will fill with more frames if the program is running.



### Frame Explanation:

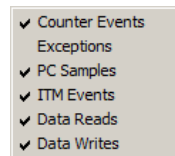
1. **ITM 31:** Data for the RTX Event Viewer.
2. **Exception 11:** SVC call exception.
3. **Exception 15:** SysTick timer.
  - **Entry:** when the exception enters.
  - **Exit:** When it exits or returns.
  - **Return:** When all the exceptions have returned to the main program. This is useful to detect tail-chaining.
4. **Data Write:** Result of the counter being incremented in the Logic Analyzer. The write triggers this event.
5. There are a few other frames not displayed in this window since they are not enabled in the Trace Config window.



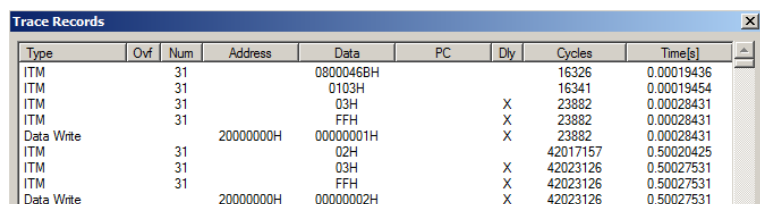
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		0800046BH			16349	0.00019463
ITM		31		0103H			16364	0.00019481
Exception Entry		11					16602	0.00019764
Exception Exit		11					16902	0.00020121
Exception Return		0					16910	0.00020131
ITM		31		03H		X	22402	0.00026669
ITM		31		FFH		X	22402	0.00026669
Data Write			20000000H	00000001H		X	22402	0.00026669
Exception Return	X	0				X	22402	0.00026669
Exception Entry		15					100836	0.00120043
Exception Exit		15					101128	0.00120390
Exception Return		0					101136	0.00120400
Exception Entry		15					184830	0.00220036
Exception Exit		15					185021	0.00220263
Exception Return		0					185029	0.00220273
Exception Entry		15					268830	0.00320036
Exception Exit		15					269018	0.00320060
Exception Return		0					269026	0.00320069
Exception Entry		15					352830	0.00420036
Exception Exit		15					353018	0.00420260

### Frame Filters:

1. Note there are some "x" in the Ovf and Dly columns. These are SWO overloads and can cause SWV display errors.
2. Right click in Trace Records to filter out frames. Unselect Exceptions.
3. Clear the Trace Records window by double-clicking inside it. More frames will be displayed.
4. Note the "x" do not go away in the DLY column. This is because this filter only affects the display of frames and not their output on the SWO pin. To reduce overloads you must turn off various events in the trace configuration window.



5. The Exception "x" are no longer visible because the Exceptions are no longer displayed but the overload is probably still present.
6. In this case, there is something still causing the overloads. It could be too many Data Writes or the still collected Exceptions.
7. Note sometimes the Cycles and/or Time(s) sometimes stay the same as the result of an overload. It is possible to turn off the timestamps but not all events will work without them.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		0800046BH			16326	0.00019436
ITM		31		0103H			16341	0.00019454
ITM		31		03H		X	23882	0.00028431
ITM		31		FFH		X	23882	0.00028431
Data Write			20000000H	00000001H		X	23882	0.00028431
ITM		31		02H			42017157	0.50020425
ITM		31		03H		X	42023126	0.50027531
ITM		31		FFH		X	42023126	0.50027531
Data Write			20000000H	00000002H		X	42023126	0.50027531

### Spurious Frames:

1. You might see some Counter frames. In this case, since all Counters are disabled therefore these frames are spurious and are caused by SWO pin overloads.
2. All ITM frames are either 31 or 0. Any other frame numbers are usually caused by the Core Clock: incorrect or possibly caused by SWO pin overloads.

**TIP:** You can filter out types of frames in the Trace Records. This does not help reduce overloads. To reduce overloads it is necessary to disable certain events in the Trace Configuration window. This will result in stopping and starting the program.

### Trace Overflow Display:

An indication of Trace overflows or not is displayed at the bottom of  $\mu$ Vision.

Trace: Running ...

Trace: Data Overflow

It is common to see Data Overflow displayed and this is not always a need for high concern.




## 25) Exceptions (includes interrupts): Uses SWV:

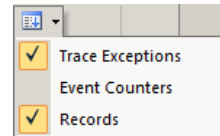
Exceptions include interrupts. These are displayed in two windows: 1) Trace Records and 2) Trace Exceptions. They are enabled/disabled in the Trace Configuration window or in the Trace Exceptions window. The mnemonic is EXCTRC.




The exceptions in the Trace Records have been shown on the previous page. If Exceptions are filtered out, they are still being output on the SWO pin potentially adding to overflows. The key to successful SWV operation is keeping the overflows down.

### Trace Exceptions Window:

It is very useful to see when exceptions are occurring and how many times. This is displayed most completely in the Trace Exceptions window.

1. Be in Debug mode.  Run the program.  The Trace Exceptions was opened on the previous page. If not:
2. Open the Trace Exceptions window by clicking on the small arrow beside the Trace icon:
3. Note the exceptions are listed by both number and name.
4. Click in the Count box and active exceptions will come to the top as shown below:
5. This window updates in real time while the program runs.
6. The Exceptions can be turned off/on by unselecting EXCTRC: Exception Tracing. The program will not stop.
7. Exceptions can also be enabled/disabled by selecting Debug/Debug Settings in the  $\mu$ Vision main menu and then selecting the Trace tab and then EXCTRC box. This action will cause the program to stop. Click RUN to start .



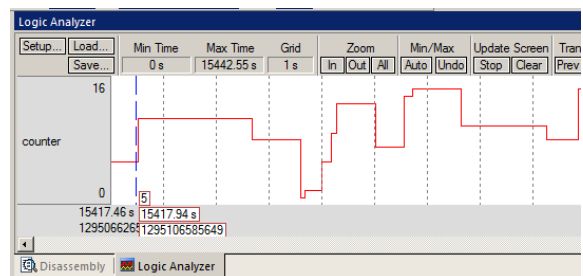
Trace Exceptions										
   <input checked="" type="checkbox"/> EXCTRC: Exception Tracing <input checked="" type="checkbox"/> Timestamps Enable										
#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]	
15	SysTick	151556	364.854 ms	2.226 us	5.214 us	994.857 us	997.774 us	0.00120408	151.55620401	
11	SVCALL	2	281.595 us					0.00020130	0.00048289	
97	FPU	0	0 s							
89	I2C3_ER	0	0 s							
88	I2C3_EV	0	0 s							

Call Stack + Locals | Debug (printf) Viewer | Watch 1 | **Trace Exceptions** | Event Counters | Memory 1

### Trace Overflow and how to minimize this: It is important to make sure overflows do not distort your SWV displays.

At this point we have these events coming out the 1 bit SWO pin:

- ITM 31(RTX Event Viewer) and ITM 0 (printf).
  - Exceptions 11 and 15 (SVCALL and SysTick respectively)
  - Data Write. This results from the global variable counter being entered in the LA.
1. There are some display problems as the SWO pin is seriously overloaded. There are many "x" in the Trace Records window especially for the Data Writes. The LA is distorted (in my example) as shown:
  2. In the Trace Exceptions window, deselect EXCTRC: box. In my case, the LA started displaying the original saw tooth waveform. The "x" on Data writes stopped.
  3. It might take a few seconds for the LA to fully recover.
  4. Turning off the exceptions freed up enough bandwidth to enable the LA to work correctly.




#### Rule Number 1 when using Serial Wire Viewer:

Enable as few events as possible to minimize SWO pin overflows. There are many SWV bits to push out the 1 bit SWO pin and this is easily overloaded. Events chosen to output on the SWO pin are selected in the Trace Config window.

If you are a heavy user of SWV, consider obtaining a Keil ULINKpro. This has much better SWO performance and can use either the SWO pin in Manchester mode or the 4 bit Trace Port for increased throughput.

## 26) Data Write: Uses SWV and Logic Analyzer Variables:

When a variable is successfully entered and displayed in the Logic Analyzer when the write event occurs, it is also displayed in the Trace Records window as a Data Write. This feature is very useful for finding unexpected data writes or values.

1. Right click in the Trace data window and unselect ITM Events. This is to make the Data Writes easier to see.
2. Select Debug/Debug Settings and select the Trace tab.
3. Select on Data R/W sample: ☒ on Data R/W Sample This will add the PC column into the Trace Records window.
4. Select Close. Click Yes when a window opens and states that Configuration has changed and asked if to take values. The program will stop.
5. Run the program.  Double click in the Trace Records window to clear it. It will fill up with new Data Writes.

These fields will be displayed in the Trace Data window:

- **Address:** The physical address of the variable.
- **Data:** The data value written to this location.
- **PC:** The location of the instruction that made the indicated data write. This field is optional. It is enabled by the on Data R/W Sample in the Trace Configuration window: ☒ on Data R/W Sample This is off by default.
- **Cycles and Time(s):** when the data write occurred. Data overflows can distort these times. They will often display the same cycles/time if incorrect because of overflows.

The first line in this Trace Data window says:

Data 0x0000\_0004 was written to address 0x2000\_0000 by the instruction located at address 0x0800\_0420 at the indicated time and CPU cycles.

With no "x": in the Dly or the Ovf columns means there are now data overflows. This is because the output of Exceptions to the SWO pin have been disabled.

Trace Records								
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	00000004H	08000420H		2811160946	33.46620174
Data Write			20000000H	00000005H	08000420H		2853160946	33.96620174
Data Write			20000000H	00000006H	08000420H		2895160946	34.46620174
Data Write			20000000H	00000007H	08000420H		2937160946	34.96620174
Data Write			20000000H	00000008H	08000420H		2979160946	35.46620174
Data Write			20000000H	00000009H	08000420H		3021160946	35.96620174
Data Write			20000000H	0000000AH	08000420H		3063160946	36.46620174

Address 0x0800\_0420 is a STR instruction as shown here:

```
89:                                     counter++;
0x08000418 480E    LDR      r0,[pc,#56] ; @0x08000454
0x0800041A 6800    LDR      r0,[r0,#0x00]
0x0800041C 1C40    ADDS     r0,r0,#1
0x0800041E 1300    LDR      r1,[pc,#52] ; @0x08000454
0x08000420 6008    STR      r0,[r1,#0x00]
90:                                     if (counter > 0x00) counter = 0;
```

## 27) RTX System and Threads Viewer: Uses DAP:

RTX is a CMSIS-RTX compliant RTOS. See [www.keil.com/rtx](http://www.keil.com/rtx). There are two kernel awareness windows available.

1. Have the program still running from the previous pages..
2. In the main  $\mu$ Vision menu, select Debug/OS Support and select System and Thread Viewer:
3. This window below opens: position and size it appropriately.
4. Shown are three threads in this implementation of RTX. ID of 1, 2 and 3 plus the idle daemon.
5. Various other RTX data is displayed. This window updates in real time.
6. This window is very useful to quickly see if your threads are configured correctly and running.

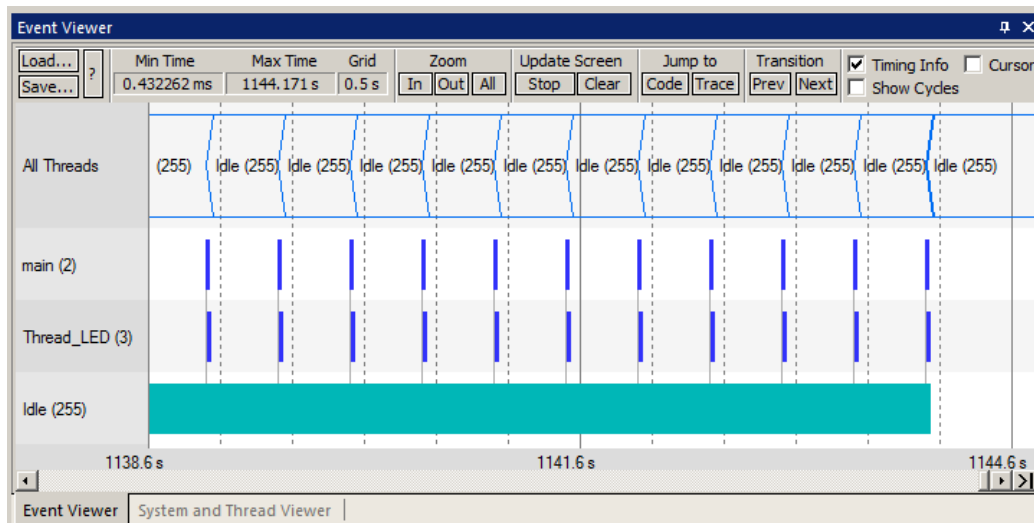
Property	Value
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 3 + os_idle_demon

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				32%
2	main	Normal	Wait_DLY	363			21%
3	Thread_LED	Normal	Wait_AND		0x0000	0x0001	40%
255	os_idle_demon	None	Running				

## 28) RTX Event Viewer: Uses SWV and ITM 31:

1. In the main  $\mu$ Vision menu, select Debug/OS Support and select Event Viewer:
2. This window opens. Adjust the Zoom with the In, Out and All icons for appropriate scaling.
3. This window is useful to see at a glance when various threads are running and how long they are active.



4. Select Cursor and Show Cycles. Select Stop in Update Screen. The program keeps running.
5. Select a place with a mouse click and move the cursor to determine timings. Hover over a spot to see a dialog.

**TIP:** If you use a ULINK $pro$ , interrupts and their handler timings will also be displayed in this window.

**TIP:** Event Viewer has high SWO bandwidth. It is very sensitive to SWO overflows. You may need to disable other events.

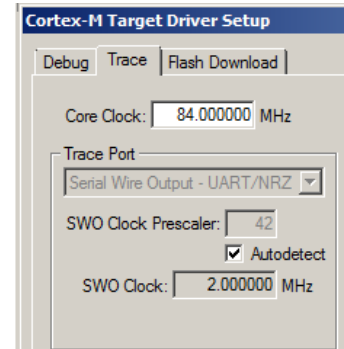
## 29) Using SWV to Determine and/or Confirm CPU Speed:

It is possible to use SWV to determine or confirm the speed of the CPU. The Serial Wire Viewer must be outputting a waveform on the SWO pin for this to work. The SWO output is a UART when used with ST-LINK/V2. The SWO Clock frequency is the UART speed. This method provides an accuracy expected when using a UART.

### How it Works:

The Trace Config window has area called TracePort: as shown here:

1. You enter what you believe is the correct CPU frequency in Core Clock:
2.  $\mu$ Vision takes this frequency and calculates a SWO Clock Prescaler value to create a SWO Clock frequency within the range of the debug adapter: a STLink/V2.
3. In this case,  $\mu$ Vision calculates a Prescaler of 42 is necessary and the resulting SWO Clock: will be 2 MHz.
  - a. The SWO Clock Prescaler is a CoreSight register in the processor.
  - b. The SWO Clock can be measured with an oscilloscope on the SWO pin.
  - c. **Core Clock: / 42 = SWO Clock.** In this example:  $84/42 = 2$  MHz.
  - d. With this information plus the actual SWO Clock: frequency, you can easily calculate what Core Clock: is.



### Steps to take:

1. Connect an oscilloscope to the SWO pin and obtain a signal of the SWO signal.
2. Measure the period of the narrowest pulse and invert it. This gives the actual SWO Clock:
3. Calculate CPU frequency as shown with this equation: **SWO frequency \* Prescaler = actual Core Clock:**

### Conclusions:

1. If you have valid and stable trace frames in the Trace Records window, the frequency in the Core Clock: is valid.
2. If you have no or erroneous frames in the Trace Records window, the Core Clock: is probably incorrect.
3. Erroneous frames include any ITM Ports other than 0 or 31 and exceptions known not to be active.

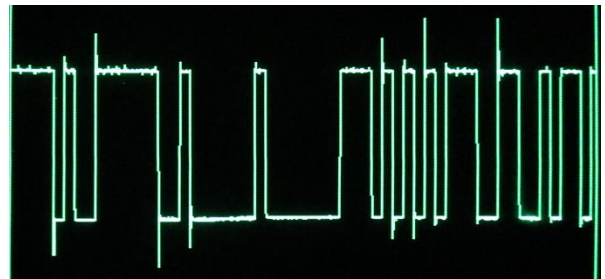
### Example:

1. With the current Blinky project, Core Clock: was set to 60 MHz. The actual CPU speed is still 84 MHz.
2. The resulting Trace Records window is shown below: Note the ITM 24, 4, 20, and so on are bogus. The Data Write data has a bogus value in the Data column. The Cycles and Time(s) columns are mostly all the same value. It is safe to assume the Core Clock: setting is probably incorrect.
3. Referring to the board schematic, SWO is on Pin 6 of CN2. Connect an oscilloscope and as carefully as possible measure the period of the shortest pulse in the SWO signal.

### Calculations:

4. Period of shortest pulse = 353 nsec.
5. Prescaler value is now 30.
6.  $(1/(353 \text{ nsec})) * 30 = 84.98 \text{ MHz}$ .
7. Enter rounded 85 MHz in Core Clock: and the Trace Records now works normally.
8. Shown is the SWO signal on the oscilloscope:  
It is approximately 3.3 VP-P.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		24		FFH		X	1759577039	22.06499194
ITM		4		FFC1H		X	1759577039	22.06499194
ITM		4		FFC1H		X	1759577039	22.06499194
ITM		4		BFC1H		X	1759577039	22.06499194
ITM		4		BFC1H		X	1759577039	22.06499194
Data Write			20000000H	25B02008H		X	1759577039	22.06499194
ITM		20		0DA0H			1759577042	22.06499199
ITM		23		BABFH			1759577042	22.06499199
ITM		23		A8B9H			1759577042	22.06499199
ITM		23		BAB8H			1759577042	22.06499199
ITM		7		FEH			1759577042	22.06499199





**TIP:** For CoreSight debug connector specifications see:  
[www.keil.com/coresight/coresight-connectors/](http://www.keil.com/coresight/coresight-connectors/)

Or

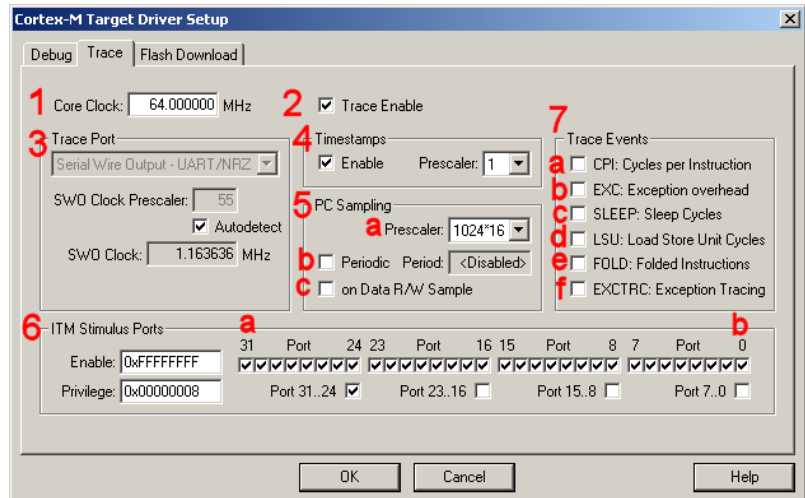
[www.arm.com](http://www.arm.com) and search for cortex\_debug\_connectors.pdf

### 30) 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  $\mu$ 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  $\mu$ 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 global variable SystemCoreClock can be viewed to provide the CPU frequency. This *must* be set correctly for ST-Link/V2.
- 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 16,384<sup>th</sup> PC is displayed. The rest are not collected.
- b. **Periodic:** Enables PC Sampling.
- 6) 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.
- 7) **ITM Stimulus Ports:** Enables the thirty-two 32 bit registers used to output data in a *printf* type statement to  $\mu$ 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  $\mu$ 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.
- 8) **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.

## 31) Serial Wire Viewer (SWV) and ETM Trace Summary:

### Serial Wire Viewer can see:

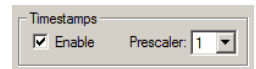
- Global variables.
- Static variables.
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers – DMA bypasses CPU and SWV by definition.

### Serial Wire Viewer displays in various ways:

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

### Top Reasons SWV does not work:

- Core Clock is incorrect. Use the global variable `SystemCoreClock` to determine CPU speed. Note the CPU speed can change after the PLL is configured and SWV can stop. Use STM32CubeMX to calculate the clock tree.
- Check Trace Records window: look for bogus ITM frames – should be only ITM 0 and 31.
- SWO pin is seriously overloaded. Unselect some SWV events. Always minimize active number of SWV events.
- Make sure View/Periodic Window Update is selected. Only while in Debug mode.
- Make sure Timestamps are activated: Some, but not all, SWV features will turn off with this.
- Some ST boards have a solder bridge that needs to be soldered. The STM32F429I- Disco is one example.



### ETM Trace is good for: A ULINKpro is needed to obtain ETM Instruction Trace:

- ETM Instruction 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).
- ETM facilitates Code Coverage, Performance Analysis and program flow debugging and analysis.

### 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.

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

## 32) Document Resources:

See [www.keil.com/st](http://www.keil.com/st)

### Books:

1. **NEW!** Getting Started MDK 5: Obtain this free book here: [www.keil.com/mdk5/](http://www.keil.com/mdk5/).
2. There is a good selection of books available on Arm: [www.arm.com/support/resources/arm-books/index.php](http://www.arm.com/support/resources/arm-books/index.php)
3.  $\mu$ Vision contains a window titled Books. Many documents including data sheets are located there.
4. A list of resources is located at: [www.arm.com/products/processors/cortex-m/index.php](http://www.arm.com/products/processors/cortex-m/index.php)  
Click on the Resources tab. Or select the Cortex-M processor you want in the Processor panel on the left.
5. Or search for the Cortex-M processor you want on [www.arm.com](http://www.arm.com).
6. The Definitive Guide to the Arm Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
7. The Definitive Guide to the Arm Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
8. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

### Application Notes:

1. Using Cortex-M3 and Cortex-M4 Fault Exceptions [www.keil.com/appnotes/files/apnt209.pdf](http://www.keil.com/appnotes/files/apnt209.pdf)
2. Segger emWin GUIBuilder with  $\mu$ Vision™ [www.keil.com/appnotes/files/apnt\\_234.pdf](http://www.keil.com/appnotes/files/apnt_234.pdf)
3. Porting mbed Project to Keil MDK™ [www.keil.com/appnotes/docs/apnt\\_207.asp](http://www.keil.com/appnotes/docs/apnt_207.asp)
4. MDK-Arm™ Compiler Optimizations [www.keil.com/appnotes/docs/apnt\\_202.asp](http://www.keil.com/appnotes/docs/apnt_202.asp)
5. GNU tools (GCC) for use with  $\mu$ Vision <https://launchpad.net/gcc-arm-embedded>
6. RTX CMSIS-RTOS in MDK 5 C:\Keil\_v5\ARM\PACK\ARM\CMSIS\4.5.0\CMSIS\RTOS\RTX  
(replace 4.5.0 with the RTX version you desire)
7. RTX CMSIS-RTOS [www.keil.com/rtx](http://www.keil.com/rtx) and [www.keil.com/cmsis](http://www.keil.com/cmsis)
8. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
9. Lazy Stacking on the Cortex-M4: [www.arm.com](http://www.arm.com) and search for DAI0298A
10. Cortex Debug Connectors: [www.arm.com](http://www.arm.com) and search for cortex\_debug\_connectors.pdf
11. Sending ITM printf to external Windows applications: [www.keil.com/appnotes/docs/apnt\\_240.asp](http://www.keil.com/appnotes/docs/apnt_240.asp)
12. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: [www.keil.com/appnotes/docs/apnt\\_270.asp](http://www.keil.com/appnotes/docs/apnt_270.asp)
13. **NEW!** Armv8-M Architecture Technical Overview <https://community.arm.com/docs/DOC-10896>
14. **NEW!** Determining Cortex-M CPU Frequency using SWV [www.keil.com/appnotes/docs/apnt\\_297.asp](http://www.keil.com/appnotes/docs/apnt_297.asp)

### Keil Training Courses:

1. Keil Training Courses, Workshops and Tradeshows: [www.keil.com/events/](http://www.keil.com/events/)

### Useful Arm Websites:

1. **NEW!** CMSIS Standards: [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5) and [www.arm.com/cmsis/](http://www.arm.com/cmsis/)
2. CMSIS Documents: [www.keil.com/pack/doc/CMSIS/General/html](http://www.keil.com/pack/doc/CMSIS/General/html)
3. Arm and Keil Community Forums: [www.keil.com/forum](http://www.keil.com/forum) and <http://community.arm.com/groups/tools/content>
4. Arm University Program: [www.arm.com/university](http://www.arm.com/university). Email: [university@arm.com](mailto:university@arm.com)
5. mbed™: <http://mbed.org>

For comments or corrections on this document please email [bob.boys@arm.com](mailto:bob.boys@arm.com).

### 33) Keil Products and Contact Information:

#### Keil Microcontroller Development Kit (MDK-ARM™)

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

Free MDK for STM32 F0/L0

For the latest MDK details see: [www.keil.com/mdk5/selector/](http://www.keil.com/mdk5/selector/)

Keil Middleware includes Network, USB, Graphics and File System. [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/)

#### USB-JTAG adapter (for Flash programming too)

- **ULINK2** - (ULINK2 and ME - SWV only – no ETM) **ULINK-ME** is equivalent to a ULINK2.
- **New ULINKplus-** Cortex-Mx High performance SWV & power measurement. Available late Oct 2017
- **ULINKpro** - Cortex-Mx SWV & ETM instruction trace. Code Coverage and Performance Analysis.
- **ULINKpro D** - Cortex-Mx SWV no ETM trace ULINKpro also works with ARM DS-5.

- **For special promotional or quantity pricing and offers, please contact Keil Sales.**
- **In USA and Canada, 1-800-348-8051.** [sales.us@keil.com](mailto:sales.us@keil.com)
- **Outside the US: +49 89/456040-20** [sales.intl@keil.com](mailto:sales.intl@keil.com)
- **Global Inside Sales Contact Point:** [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)

**RTOS:** Keil RTX RTOS is now provided as part of MDK: See [www.keil.com/rtx](http://www.keil.com/rtx)

**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](http://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](http://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](http://www.arm.com/ds5).



#### For more information:

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

Global Inside Sales Contact Point: [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)

Keil World Wide Distributors: [www.keil.com/distis/](http://www.keil.com/distis/)

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

CMSIS: [www.keil.com/cmsis](http://www.keil.com/cmsis) **New CMSIS Version 5:** [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)

