

## Introduction:

The purpose of this lab is to introduce you to the STMicroelectronics Cortex™-M4 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. You will be able to confidently work with these processors and Keil MDK. See [www.keil.com/st](http://www.keil.com/st).

Keil MDK supports and has examples for nearly all ST ARM processors. Check the Keil Device Database® on [www.keil.com/dd2](http://www.keil.com/dd2) for the complete list which is also included in MDK: in µVision, select Pack Installer.

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.

## Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is turn-key "out-of-the-box".
2. MDK is **STM32CubeMX** compatible.
3. ARM Compiler 5 and ARM Compiler 6 (LLVM) are included. GCC is supported. <https://launchpad.net/gcc-arm-embedded>
4. Dynamic Syntax checking on C/C++ source lines.
5. **Keil Middleware:** Network, USB, Flash File, Graphics and CAN.
6. **Event Recorder** for Keil Middleware, RTX and User programs.
7. MISRA C/C++ support using PC-Lint. [www.gimpel.com](http://www.gimpel.com)
8. **Compiler Safety Certification Kit:** [www.keil.com/safety/](http://www.keil.com/safety/)
9. TÜV certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
10. Keil RTX and RTX 5 is included. RTX has a BSD or Apache 2.0 license respectively with source code. [www.keil.com/RTX](http://www.keil.com/RTX) and [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)
11. CoreSight™ Serial Wire Viewer (SWV). ETM instruction trace capability on appropriately equipped ST processors.
12. Debug Adapters: ST-LINK/V2, Keil ULINK™2, ULINK-ME, ULINKpro, CMSIS-DAP and J-Link.
13. MDK includes board support for ST processors and boards.
14. Affordable perpetual and term licensing with support. Contact Keil sales for pricing options. [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)
15. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
16. Micrium µC/Probe compatible with ULINK2, ULINKplus or ULINKpro. [www.micrium.com/ucprobe](http://www.micrium.com/ucprobe)

### This document details these features:

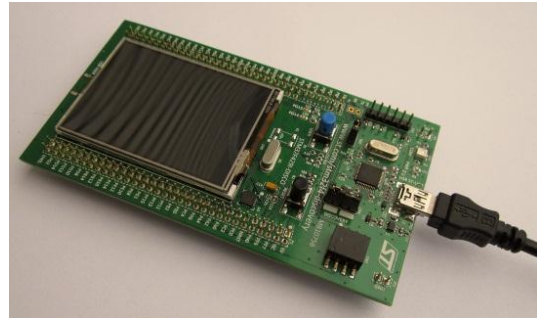
1. Serial Wire Viewer (SWV) and ETM trace. Real-time tracing updated while the program is running.
2. Real-time Read and Write to memory locations for Watch, Memory and RTX Tasks windows.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and four Access Breakpoints (also known as Watchpoints).
4. RTX Viewer: Two kernel awareness windows for the Keil RTX RTOS that update while your program is running.
5. A DSP example program using ARM CMSIS-DSP libraries. For info on CMSIS-DSP, see [www.arm.com/cmsis](http://www.arm.com/cmsis).

## Serial Wire Viewer (SWV):

**Serial Wire Viewer (SWV)** displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and a timestamp. SWV does not steal CPU cycles and is non-intrusive. (except for the ITM Debug printf Viewer).

## Embedded Trace Macrocell (ETM): ETM requires a Keil ULINKpro.

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 also provides Code Coverage and Performance Analysis. Use a STM32429I-EVAL board and a Keil ULINKpro to evaluate or use ETM.



## Index:

1. STMicroelectronics Evaluation Board Support:	3
2. MDK 5 Keil Evaluation Software and Software Pack Summary:	3
3. The on-board ST-LINK/V2 Debug Adapter:	3
4. Keil MDK Software Download and Installation:	4
5. $\mu$ Vision Software Pack Download and Install Process plus Examples:	4
1. Start uVision and Open Pack Installer:	4
2. Install STM32F4xxx_DFP Pack:	4
3. Copy CMSIS-RTOS Example:	5
4. Install Blinky_BM, DSP and RTX5_Blinky examples:	5
6. CoreSight Definitions:	6
7. Debug Adapters Supported with ST Processors: for reference:	7
8. Installing the ST-LINK/V2 USB Drivers:	8
9. <i>Blinky</i> example using the STM32F429I Discovery board:	9
10. Hardware Breakpoints:	9
11. Call Stack & Locals window:	10
12. Watch and Memory windows and how to use them:	11
13. View Variables Graphically with the Logic Analyzer (LA):	12
14. printf using ITM 0 and Serial Wire Viewer:	13
15. Watchpoints: <i>Conditional Breakpoints</i>	14
16. Peripherals System Viewer (SV):	15
17. RTOS Kernel Awareness: Keil RTX and FreeRTOS:	16
18. RTOS System and Thread Viewer:	17
19. RTOS Event Viewer:	17
20. Exceptions and Interrupts:	18
21. DSP Sine Example using graphical Logic Analyzer:	19
22. Creating your own MDK 5 Project from scratch:	20
23. Adding RTX RTOS to your Project:	23
24. Adding a Thread to your RTX Project:	24
25. Using Event Viewer to Examine your RTX Timings:	25
26. <b>NEW !</b> Event Recorder:	26
27. ETM Trace Examples and Features: <i>for reference</i>	27
28. Serial Wire Viewer and ETM Trace summary:	32
29. Document Resources:	33
30. Keil Products and Contact information:	34

## 1) ST Evaluation Boards:

Keil MDK provides board support including examples for many ST Cortex-M processors.

This lab used an STM429I-discovery board with the on-board ST-LINK/V2 as the debug adapter.

If you want to use or evaluate ETM, get a board such as the STM32429I-EVAL. It has the required 20 pin CoreSight ETM connector. You will also need a Keil ULINK<sub>pro</sub>. ETM is supported by all versions of Keil MDK. There are no extra license options to buy.

On the second last page of this document is an extensive list of resources that will help you successfully create your projects. This list includes application notes, books and labs and tutorials for other ST processors and boards.

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

**ARM forums:** <https://developer.arm.com>

**Keil Forums:** [www.keil.com/forum/](http://www.keil.com/forum/)

---

## 2) MDK 5 Keil Software and Software Packs Information: *This document used MDK 5.23.*

MDK 5 Core is the heart of the MDL toolchain. This will be in the form of MDK Lite which is the evaluation version with a compile limit of 32K. The addition of a Keil license will turn it into one of the commercial versions available. Contact Keil Sales for more information.

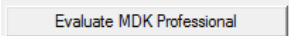
Device and board support are distributed via Software Packs. These Packs are downloaded from the web with the "Pack Installer", the version(s) selected with "Select Software Packs" and your project configured with the "Run Time Environment" (RTE) utilities. These are components of  $\mu$ Vision.

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

See [www.keil.com/dd2/pack](http://www.keil.com/dd2/pack) or open the Pack Installer  for the current list of available Software Packs.

### MDK Licensing:

1. You can use the evaluation version (MDK-Lite) for this lab. No license is needed except for the Keil Middleware examples if you choose to try them.
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 examples as well as unlimited code size compilation. Contact Keil sales to extend this license for evaluation purposes or to purchase an MDK license.



---

## 3) The on-board ST-LINK/V2 Debug Adapter:

The on-board ST-LINK/V2 is used exclusively in this lab. Instructions on configuring and testing the ST-LINK/V2 are on page 8. This Discovery board does not have an external debug adapter connector. If you need to connect an external adapter you will have to connect to connector CN2 or CN4 and perhaps adjust some solder bridges. This would normally be needed if you want to connect a Keil ULINK<sub>pro</sub> or ULINK<sub>plus</sub> for faster SWV performance or to view interrupts in a graphical format.

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<sub>pro</sub> can be used or similar with the 20 pin ETM connector.

**USB Drivers:** Normally, you must install the USB drivers manually. This is not done automatically. Instructions are on page 8. The current drivers and firmware updates for the ST-LINK/V2 are provided with MDK.

**ST-LINK/V2 and Serial Wire Viewer:** [www.keil.com/appnotes/docs/apnt\\_286.asp](http://www.keil.com/appnotes/docs/apnt_286.asp)

---

## 4) Keil MDK Software Download and Installation:




1. Download MDK 5.23 or later 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. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
4. If you install MDK into a different folder, you will have to adjust for the folder location differences.
5. You do not need a debug adapter: just the ST board, a USB cable and MDK installed on your PC.

Download MDK-Core Version 5

## 5) µVision Software Pack Download and Install Process plus Examples:

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

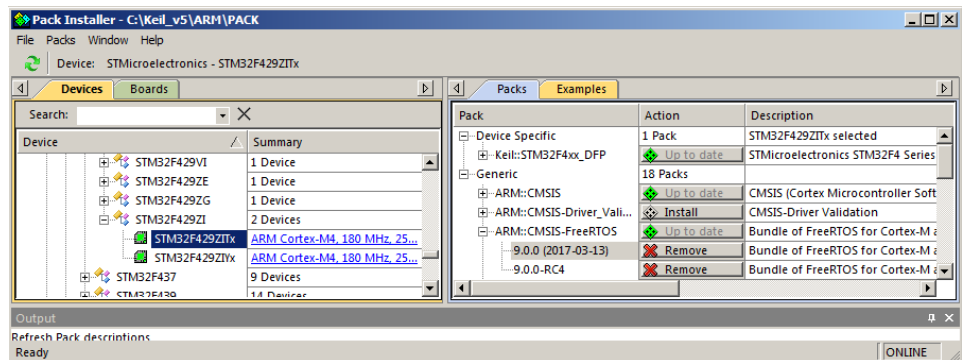
### 1) Start µVision and open Pack Installer:

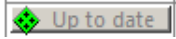
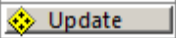
1. Connect your computer to the internet. This is needed to download the Software Packs. Start µVision: 
2. Open the Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.
3. This window opens up: Select the Devices tab:
4. Note “ONLINE” is displayed at the bottom right.  
If “OFFLINE” is displayed, connect to the Internet before continuing.
5. If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or  to refresh once you have connected to the Internet.

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


### 2) Install The STM32F4xx\_DFP Software Pack:

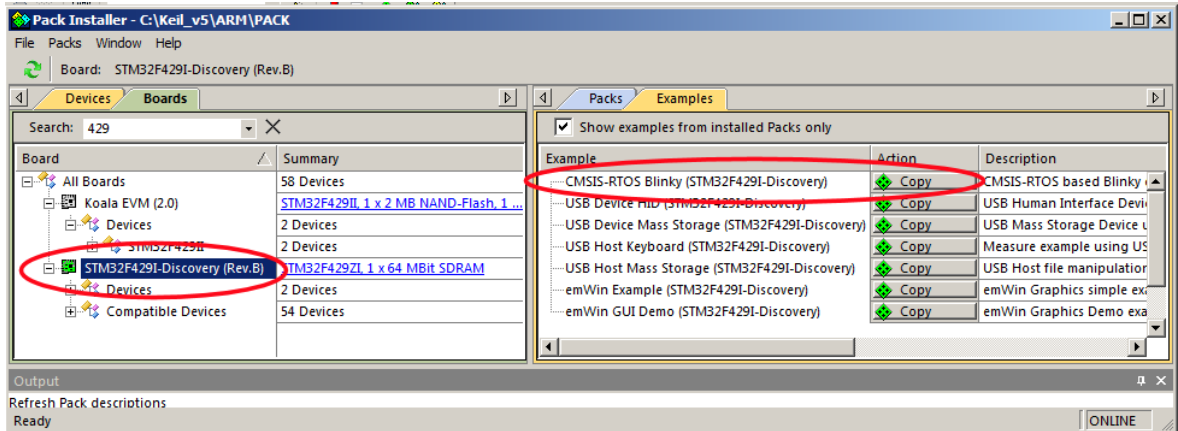
1. In the Devices tab, select ST and then STM32F429 Series. Select the processor STM32F429ZITx as shown below:
2. All the devices supported are displayed.
3. Select the Packs tab.



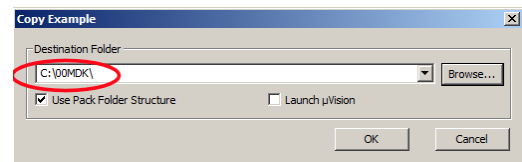
4. Beside Keil::STM32F4xx\_DFP click Install. This Pack will download and install in the MDK folders. This download can take several minutes.
5. Its status is indicated by the “Up to date” icon: 
6. Update means there is an updated Software Pack available for download. 
7. **Leave Pack Installer open to copy the example files on the next page:**

### 3) Install the CMSIS-RTOS RTX Blinky Example:


1. Select the Boards tab.
2. In the Search box enter 429.
3. Select STM32F429-Disco.
4. Select the Examples tab:
5. Opposite CMSIS-RTOS Blinky: select Copy: 




6. The Copy Example window below opens up: Select Use Pack Folder Structure: Unselect Launch  $\mu$ Vision:
7. Type in C:\00MDK\ as shown on the right: Click OK to copy the selected example. This will be copied into the destination folder C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery\.
8. Note there are other examples that use Keil Middleware. If you want to look at and/or try them copy them in the same manner.

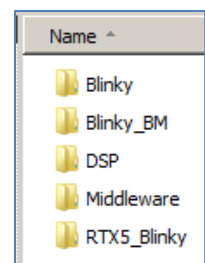


**TIP:** The default directory for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default directory of C:\MDK\ in this tutorial. You can use any directory you prefer.

9. Close the Packs Installer. You can open it any time by clicking on its icon. 
10. If a window opens and states the Software Packs have changed and need reloaded, select Yes.



### 4) Install the Blinky\_BM and DSP Examples from Keil.com:

1. Obtain the example software zip file from [www.keil.com/appnotes/docs/apnt\\_280.asp](http://www.keil.com/appnotes/docs/apnt_280.asp).
2. Extract this into the directory C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery\
3. The Blinky and RTX\_Blinky folders will be created with the Blinky folder: 



#### Notes:

1. The Blinky\_BM is a bare metal example (no RTOS) included for your reference.
2. RTX5\_Blinky uses the new RTX v5. Event Viewer is demonstrated with this example on page 16.

**TIP:** An Update icon means there is an updated Software Pack available for download.  Update  
Select Packs/Check for Updates or  in the Pack Installer to refresh this list. It is not done automatically.




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

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

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

### Read-Only Source Files:

Some source files in the Project window will have a yellow key on them:  This means they are read-only. This is to help unintentional changes to these files. This can cause difficult to solve problems. These files normally need no modification.  $\mu$ 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)

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

1. In the Projects window, double click on the file to open it in the Sources window.
  2. Right click on its source tab and select Open Containing folder.
  3. Explorer will open with the file selected.
  4. Right click on the file and select Properties.
  5. Unselect Read-only and click OK. You are now able to change the file in the  $\mu$ Vision editor.
  6. It is a good idea to make the file read-only when you are finished modifications.
1. The Copy Example window opens up: Select Use Pack Folder Structure. Unselect Launch  $\mu$ Vision.

## 7) Debug Adapters Supported with ST Processors: for reference:

### Discovery Board Debug Adapter Connections:

The STM32F429I Discovery board lacks the standard ARM debugger connections. It was meant to normally be used with the on-board ST-LINK/V2 debug adapter. This means it is not easy to connect a ULINK2, ULINK*pro* or J-Link to these boards. In order to use features like ETM trace, it is easier to obtain a board such as the Keil MCBSTM32 series or a STM32xxx-EVAL board. Versions are available with Cortex-M3 and Cortex-M4 processors. Keil MDK has examples and labs for these boards. This document uses only the on-board ST-LINK. See [www.keil.com/st](http://www.keil.com/st).

1. **ST Link V2:** ST-LINK/V2 is an on-board or external debug adapter. You do not need an external debugger such as a ULINK2 to do this lab. Serial Wire Viewer (SWV) is supported by ST-LINK/V2 and any Keil ULINK or a J-Link.
2. **CMSIS-DAP:** This is an ARM open source debug adapter specification. You can add CMSIS-DAP to a custom board. See [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)
3. **ULINK2 and ULINK-ME:** ULINK-ME is only offered as part of certain evaluation board packages. ULINK2 can be purchased separately. These are electrically the same and both support Serial Wire Viewer (SWV), Run-time memory reads and writes for the Watch and Memory windows and hardware breakpoint set/unset on-the-fly.
4. **ULINK*pro*:** ULINK*pro* supports all SWV features and adds ETM Instruction Trace. ETM records all executed instructions. ETM provides Code Coverage, Execution Profiling and Performance Analysis features.  
If you are using a RTX RTOS, it is possible to view interrupts in a graphical format. The Event Recorder is also available with a ULINK*pro*. ULINK*pro* also provides the fastest Flash programming times. Most ST Cortex-M3, M4 and M7 devices have ETM. Consult your datasheet.
5. **NEW ! ULINK*plus*:** High SWV performance plus Power Measurement. See [www.keil.com/ulink/ulinkplus/](http://www.keil.com/ulink/ulinkplus/) for complete details.
6. **Segger J-Link:** J-Link Version 6 (black) or later supports Serial Wire Viewer. SWV data reads and writes are not currently supported with a J-Link.



**TIP: Serial Wire Viewer Operation (SWV):** Solder Bridge SB9 must be shorted in order for SWV to operate. It is not connected by default.

### Using other Debug Adapters than the on-board ST-LINK/V2:

This document will use the on-board ST-LINK/V2. You can use a ULINK2 or a ULINK*pro* with suitable adjustments. You would need a suitable adapter to connect a different adapter to the SWD connector on the Discovery board. Some step(s) to turn off the on-board ST-Link adapter might also be necessary to avoid conflicts. It is reported that shorting solder bridge SB10 will hold the ST-Link processor in RESET allowing external adapter operation.

1. If your debugging sessions are unreliable, please check for additional conflicts or loading on the SWD pins. The SWD connector provides the ability to use the Discovery board as a debug adapter for another board. Its main purpose is not to connect an external tool such as a Keil ULINK2. Some adaptation is required but is not difficult to do.
2. It is possible to use a Segger J-Link with  $\mu$ Vision. Serial Wire Viewer is supported.

## 8) Installing the ST-LINK/V2 USB Drivers:

The ST-Link is selected as the default debug adapter for the Keil examples for this Discovery board.



### Step 1) Installing the ST-Link USB Drivers: *(you only need to do this the first time)*

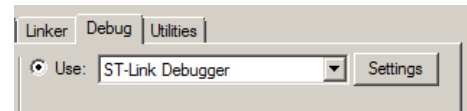
1. Do not have the Discovery board USB port connected to your PC at this time.
2. The USB drivers must be installed manually by executing **stlink\_winusb\_install.bat**. This file is found in C:\Keil\_v5\ARM\STLink\USBDriver\. Find this file and double click on it. The drivers will install.
3. Plug a USB cable from your PC to USB ST Link CN1 on the Discovery board. The USB drivers will now finish installing. ST-LINK/V2 is now ready to use. You can test it below.

**ST-LINK/V2 Firmware Update:** The ST-LINK/V2 firmware update files are located here: C:\Keil\_v5\ARM\STLink. This updates the Discovery ST-Link firmware by executing ST-LinkUpgrade.exe. Find this file and double click on it. It will check and report the current firmware version. It is important you are using firmware V2.J27.S0 or later for proper SWV operation. Always use the later version of the firmware whether from Keil or from STMicroelectronics.

### Step 2) Testing the debug connection to the target:

You can use these instructions to confirm you are connected to the STM32 CoreSight debug DAP.

1. Connect your PC to the Discovery board with a USB cable. Start  $\mu$ Vision. It must be in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. Blinky will do fine.
2. Select Target Options  or ALT-F7 and select the Debug tab. In the drop-down menu box confirm ST-Link Debugger is selected as shown here: 
3. Select Settings and the next window below opens up. This is the debug control panel.
4. In **Port:** select SW. JTAG is not a valid option for ST-Link and this board. SW is also known as SWD.
5. In the SW Device area: ARM CoreSight SW-DP with a valid IDCODE **MUST** be displayed as shown below. This confirms you are connected to the target processor. If there is an error displayed or it is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the board.



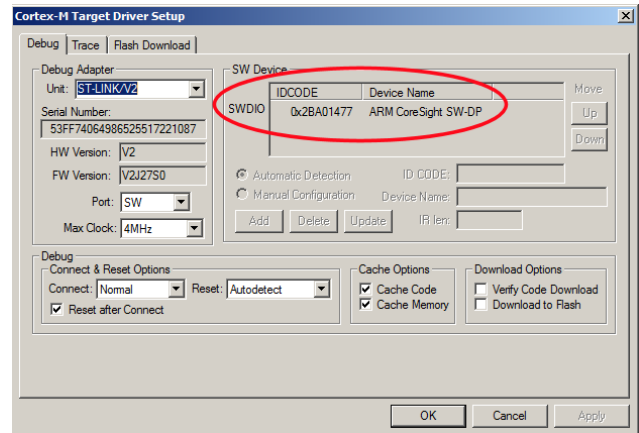
**TIP:** To refresh this screen select Port: and change it or click OK once to leave and then click on Settings again.

**Serial Wire Viewer Operation (SWV):** Solder Bridge SB9 must be shorted in order for SWV to operate. It is not connected by default.

### Step 3) Confirm the proper Flash Programmer:

These instructions are provided for reference.

1. Click on OK once and select the Utilities tab.
2. You can also select Flash Download in the above picture.
3. Click Settings to confirm the programming algorithm.
4. Below is the proper algorithm for the STM32F429 processor.
5. If it is not visible or is the wrong one select Add to select the proper one..
6. Click on OK once.
7. Click on OK to return to the  $\mu$ Vision main screen.
8. Select File/Save All.
9. You have successfully connected to the STM32 target processor and configured the  $\mu$ Vision Flash programmer.









Programming Algorithm			
Description	Device Size	Device Type	Address Range
STM32F4xx 2MB Flash	2M	On-chip Flash	08000000H - 081FFFFFFH

**TIP:** The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this later. The ETM trace is also selected here but this is not used in this lab.



## 9) Blinky example program using the STM32F429I Discovery board:

We will connect a Keil MDK development system to the target hardware with the built-in ST-LINK/V2 debug adapter.

1. Start  $\mu$ Vision by clicking on its desktop icon.  Connect your PC to the board with a USB cable to CN1.
2. Select Project/Open Project. Open C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery\Blinky\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 will need to install the USB drivers or you will get a fail to connect error. See the configuration instructions on page 4.
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: 
6. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed. Progress will be indicated in the Output Window. Select OK if the Evaluation Mode box appears.
7. Click on the RUN icon.  Note: you stop the program with the STOP icon. 





**The LEDs LD3 and LD4 on the STM32F4 Discovery board will now blink.  
Press the blue USER button and they will all come on.**

**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.

## 10) Hardware Breakpoints:

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 87 through 89 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 .

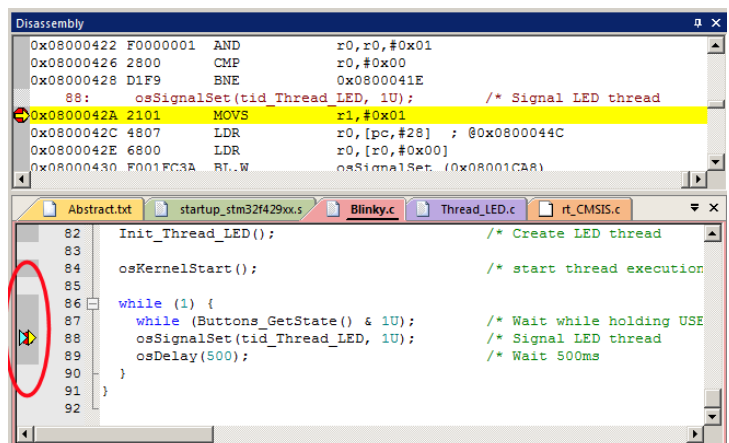
**TIP:** To step an assembly instruction, click on the Disassembly window to bring it into focus. If a C source window is in focus, it steps one C line at a time.

**TIP:** A hardware breakpoint does not execute the instruction it is set to. ARM CoreSight breakpoints are no-skip. These are rather important features for efficient software development.

### 7. Leave one breakpoint active for the next exercise.

**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.



## 11) Call Stack + Locals Window:

### Local Variables:

The Call Stack and Locals windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables 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 can be toggled by selecting View/Call Stack window.

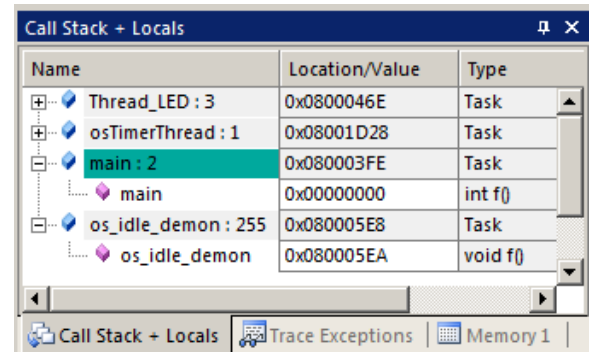
1. Click on the Call Stack + Locals tab.
2. Run Blinky. The program will halt on a line in the while() loop because of the breakpoint set on the previous page.
3. The program is halted in the main thread. This program is running RTX and main() is a thread.




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.

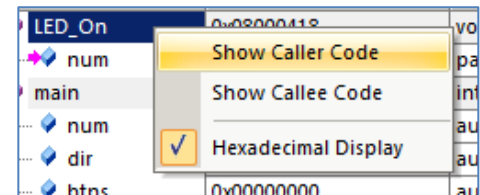
When a function exits, it is removed from the list.

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

This table is active only when the program is stopped.



4. Click on the Step In icon or F11:  with Blinky.c in focus. Blinky.c in its tab will be underlined.
5. 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.
6. Click numerous times on Step In and see other functions.
7. Right click on a function name and try the Show Callee Code and Show Caller Code options as shown here:
8. Click on the StepOut icon  to exit all functions to return to main().
9. **Remove all breakpoints to continue on the next page.**



**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 executed. ETM 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.

## 12) Watch and Memory Windows and how to use them:

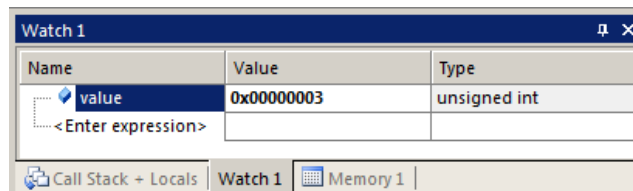
The Watch and memory windows will display updated variable values in real-time. It does this through the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to “put” or insert values into these memory locations in real-time. It is possible to “drag and drop” variable names into windows or enter them manually.

### Watch window:

**Add a global variable:** Recall the Watch and Memory windows can’t see local variables unless stopped in their function. To see locals, convert them to static or global variables. Structures, unions and memory can be viewed.

1. Stop the processor  and exit Debug mode. 
2. In Blinky.c declare a global variable (I called it counter) near line 26 like this: **unsigned int counter = 0;**
3. Add the statements `counter++;` and `if (counter > 0x10) counter = 0;` as shown here near line 90 and 91:
4. Click on Rebuild.  Click on Load  to program the Flash.
5. Enter Debug mode.  Click on RUN . Recall you can set Watch and Memory windows while the program is running.
6. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
7. In Blinky.c, right-click on the variable counter and select Add counter to ... and select Watch 1. counter will be displayed as shown here:
8. counter will increment until 0x10 in real-time.

```
86 while (1) {  
87     while (Buttons_GetState() & 1U);  
88     osSignalSet(tid_Thread_LED, 1U);  
89     osDelay(500);  
90     counter++;  
91     if (counter > 0x0F) counter = 0;  
92 }
```



**TIP:** You can also block **counter**, click and hold and drag it into Watch 1 or a Memory window.

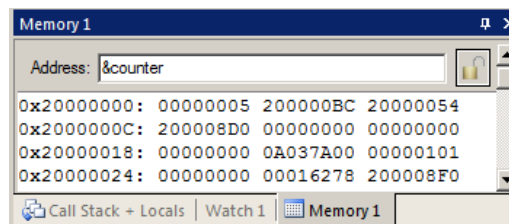
**TIP:** Make sure View/Periodic Window Update is selected.

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.

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

### Memory window:

1. Right-click on the variable counter and select Add counter to ... and select the Memory 1. counter will be displayed as shown here:
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).
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.
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. You can modify a Watch if the variable changing very slowly or the program is stopped.



Serial Wire Viewer does not need to be configured in order for the Memory and Watch windows to operate as shown. This mechanism uses a different feature of CoreSight than SWV. These Read and Write accesses are handled by the Serial Wire Debug (SWD) or JTAG connection via the CoreSight Debug Access Port (DAP), which provides on-the-fly memory accesses.

### 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.




### 13) View Variables Graphically with the Logic Analyzer (LA): Uses SWV:

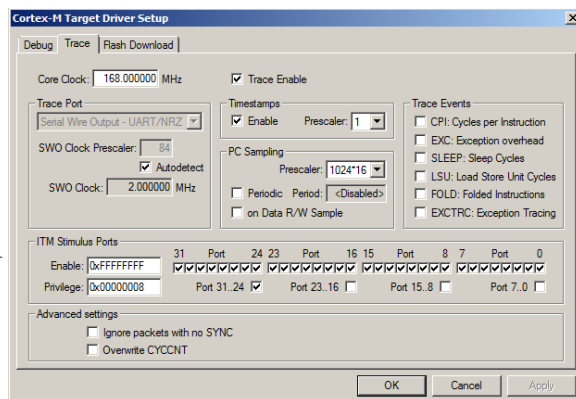
We will display the global variable counter you created earlier in the Logic Analyzer. You display up to four waveforms in the LA. **TIP:** You can configure the LA while the program is running. For more details on using SWV with STMicroelectronics processors see [www.keil.com/appnotes/docs/apnt\\_286.asp](http://www.keil.com/appnotes/docs/apnt_286.asp)

**NOTE:** Solder bridge SB9 MUST be shorted in order for SWV to work ! This is not done by default: you must do it !


1. Stop the processor  and exit Debug mode. 

#### Configure the Serial Wire Viewer (SWV):


1. Select Target Options  and select the Debug tab.
2. Select Settings: on the right side of this window.
3. Confirm SW is selected. SW selection is mandatory for SWV.
4. Select the Trace tab. This window opens: 
5. In the Trace tab, Set Core Clock: to 168 MHz. Select Trace Enable. Leave everything else at default.
6. Click OK once to return to the Debug tab.
7. Click OK return to the main menu. Enter Debug mode. 



#### Configure the Logic Analyzer:

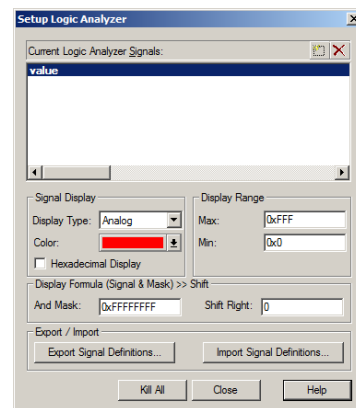
1. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 
2. Click on the Blinky.c tab. Right click on **counter** and select Add counter to... and then select Logic Analyzer. You can also Drag and Drop or enter it manually.

**TIP:** The Number 1 reason you cannot configure the LA is if the Core Clock: value in the Trace tab is incorrect.



3. Click on the Select box and the LA Setup window appears:
4. With counter selected, set Display Range Max: to 0x15 as shown here:
- 1) Click on Close.
- 2) Click on Run.  Click on Zoom Out until Grid is about 1 second.
- 3) The variable value will increment to 0x10 (decimal 16) and then is set to 0.

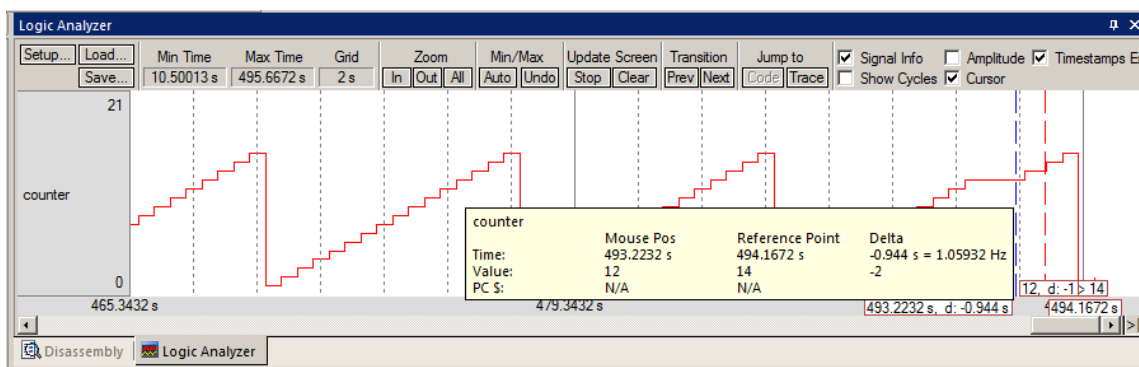
**TIP:** If you do not see a waveform, exit and re-enter Debug mode to refresh the LA. You might also have to repower the Discovery board. Confirm the Core Clock: value is correct.

**TIP:** You can display up to 4 variables in the Logic Analyzer. These variables must be global, static or raw addresses such as \*((unsigned long \*)0x20000000). No locals.



#### Measuring Times:

- 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. The program will keep running.
- 5) Hover the mouse over the waveform and a yellow box with statistics will appear as shown below:
- 6) Stop the CPU.  Exit debug mode. 





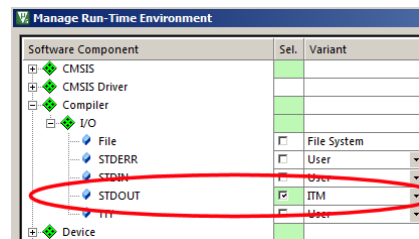
## 14) **printf** using ITM 0 (Instrumentation Trace Macrocell) SWV is required:

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  $\mu$ Vision for display in the Debug (*printf*) Viewer window. It is possible to send ITM data to a file: [www.keil.com/appnotes/docs/apnt\\_240.asp](http://www.keil.com/appnotes/docs/apnt_240.asp).

1. Stop the program  and exit Debug mode .

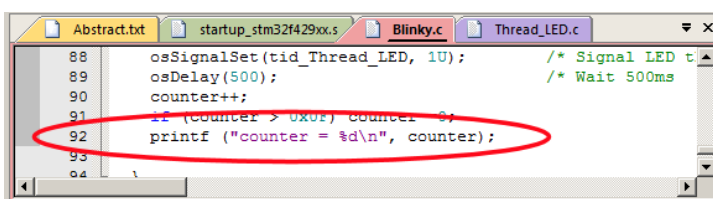
### Add STDOUT File (retarget\_io.c):

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







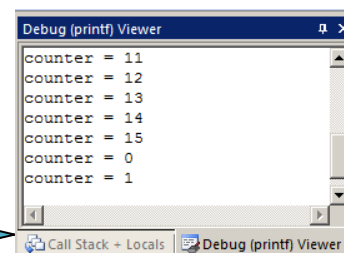
### Add **printf** and **#include <stdio.h>** to Blinky.c:

1. In Blinky.c, add **#include <stdio.h>** near line 20.
2. Inside the while()loop in Blinky.c, near line 92, add this line: `printf("counter = %d\n", counter);`



### Compile and Run the Project:

1. Select File/Save All or click .
2. Rebuild the source files  and enter Debug mode .
3. Click on View/Serial Windows and select Debug (*printf*) Viewer and click on RUN.
4. In the Debug (*printf*) Viewer you will see the *printf* statements appear. .
5. Right click on the Debug window and select Mixed Hex ASCII mode. Note other settings.



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

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

This is documented here: [www.keil.com/pack/doc/CMSIS/Core/html/group\\_ITM\\_Debug\\_gr.html](http://www.keil.com/pack/doc/CMSIS/Core/html/group_ITM_Debug_gr.html)

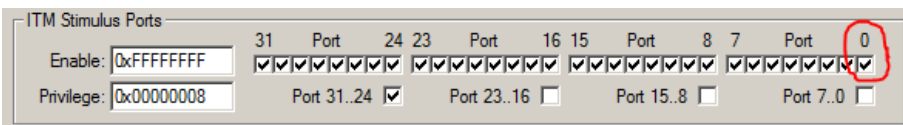
A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer.

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

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

**TIP:** ITM Port 0 can be selected or not in the Trace Configuration window as shown here: ITM *printf* will be disabled or not.

**TIP:** ITM Port 31 is used for the RTX Tasks and System window. All other ports are currently unused by  $\mu$ Vision.

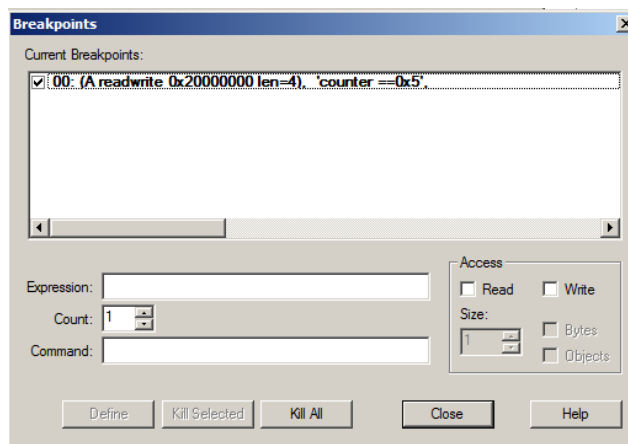





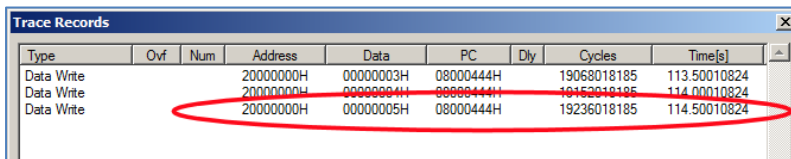
## 15) Watchpoints: Conditional Breakpoints: This does not need or use Serial Wire Viewer:

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  $\mu$ 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 to display counter.
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 then select Breakpoints or press Ctrl-B.
6. In the Expression box enter: "**counter** == 0x5" without the quotes. Select both the Read and Write Access.
7. Click on Define and it will be accepted as shown here: Click on Close.
8. Enter the variable **value** 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. Right-click in the Trace Records window and unselect ITM Events. This hides the ITM entries to clean up the display.



12. Click on RUN. 
13. You will see **value** change in the Logic Analyzer as well as in the Watch window.
14. When **value** 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 ULINK $pro$  will show a different display. A J-Link (black case) does not display data reads or writes.
16. To repeat this exercise, click on RUN several times to get past the value of counter equal five.
17. When you are finished, stop the program, click on Debug and select Breakpoints (or Ctrl-B) and Kill the Watchpoint.



Type	Ofv	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000000H	00000003H	08000444H		19068018185	113.50010824
Data Write			20000000H	00000004H	08000444H		19162018185	114.00010824
Data Write			20000000H	00000005H	08000444H		19236018185	114.50010824

18. Leave Debug mode. 

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

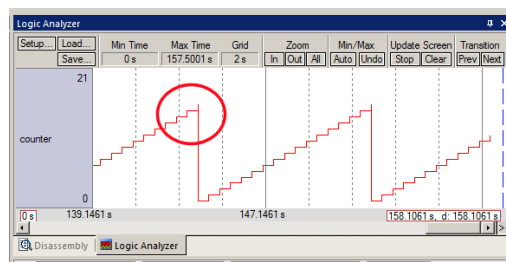
**TIP:** To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint. You should delete the old one by highlighting it and click on Kill Selected or try the next TIP:

**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. Note the value of 0x10 being displayed indicating when the test of counter > 0x0F is done.

**Stack:** Setting a Watchpoint with a test to a memory address in your stack is a good way to see how low the stack becomes. When the location is written to, the program will halt.

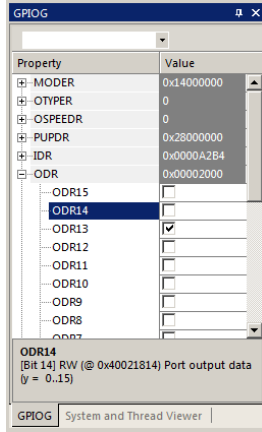


## 16) Peripherals System Viewer (SV):

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




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

### Select GPIOG:

2. Select Peripherals/System Viewer and then GPIOG as shown here.
3. This window opens up. Expand ODR: 
4. You can now see bits 13 and 14 update as the LEDs blink.
5. You can change the values in the System Viewer on-the-fly.
6. Click on one of these bits when it is off and the respective LED will illuminate. It will soon be turned off by the program.
7. The blue button is connected to GPIOA. You can see the button pressed in the IDR register.

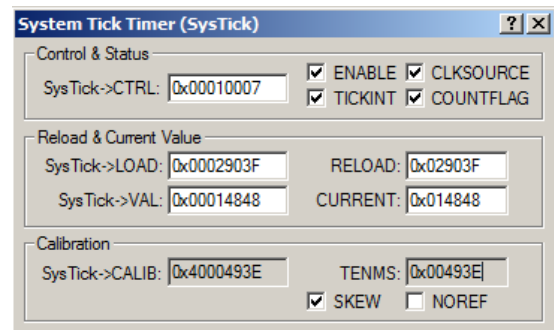
**TIP:** If you click on a register in the properties column, a description about this register will appear at the bottom of the window.

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

1. Select Peripherals/Core Peripherals and then select SysTick Timer.
2. The SysTick window shown below opens:
3. Note it also updates in real-time while your program runs. These windows use the same CoreSight DAP technology as the Watch, Memory and Peripheral windows.
4. Note the ST\_RELOAD and RELOAD registers. This is the reload register value. This is set during the SysTick configuration by RTX using values set in RTX\_Config.h Kernel Tick Frequency and the CPU clock.
5. Note that it is set to 0x2903F. This is the same value hex value of 168,000,000/1000-1 that is programmed into RTX\_Config.h. This is where this value comes from. Changing the variable passed to this function is how you change how often the SysTick timer creates its interrupt 15.
6. In the RELOAD register in the SysTick window, *while the program is running*, type in 0x5000 and click inside ST\_RELOAD ! (or the other way around)
7. The blinking LEDs will speed up. This will convince you of the power of ARM CoreSight debugging.
8. Replace RELOAD with 0x2903F. 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.

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

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




## 17) RTOS Kernel Awareness: Keil RTX and FreeRTOS:

Keil provides RTX and RTX 5, 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. RTX comes with a BSD or Apache 2.0 license and source code. See: [www.keil.com/RTX](http://www.keil.com/RTX) and [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)

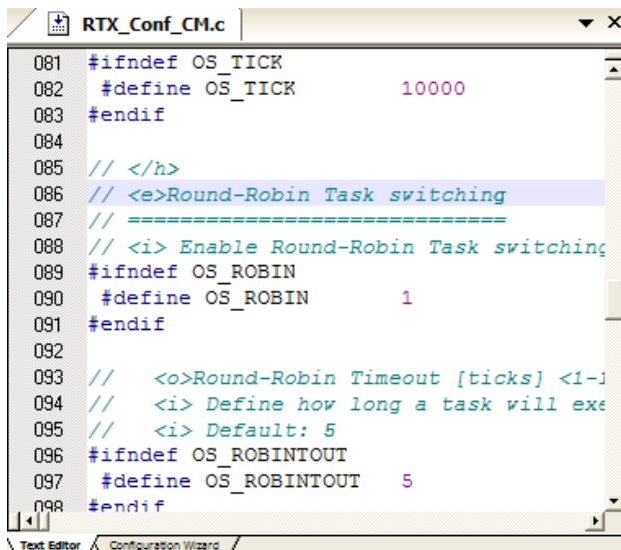
The project you have been using contains RTX. We will examine  $\mu$ Vision kernel awareness with it.

**TIP:** Keil uses the term Threads instead of Tasks for consistency.

1. Have the Blinky project loaded from the previous page. Be in Debug mode and stopped. .

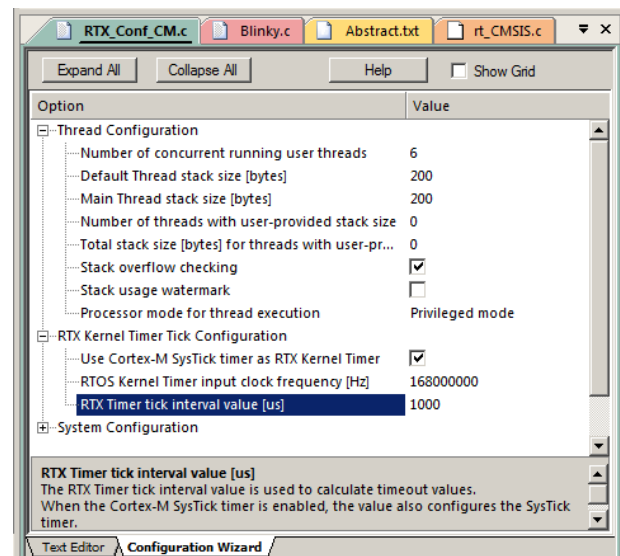
### The Configuration Wizard for RTX:

1. Open the file RTX\_Conf\_CM.c . You can click on it in the Project window. Selects its tab to bring it in focus.
  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. It is easy 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 in 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/docs/2735.htm](http://www.keil.com/support/docs/2735.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

### Serial Wire Viewer Hints:

1. In the Trace Records window, 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. Note often the timestamps are frozen with too many Dly entries. Always limit the SWV features to only those you really need. Using a ULINK $pro$  provides better SWV operation.
2. If the SWV trace fails to work properly after any changes are made while the program is still running (this is allowed), the CoreSight block might be confused. Exit and re-enter Debug mode to refresh it.
3. 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.  $\mu$ Vision easily recovers gracefully from these overflows. Overflows are shown when they happen. Using a ULINK $pro$  helps reduce overruns by using the SWO pin in Manchester mode or better, the 4 bit Trace Port.

## 18) RTOS System and Thread Viewer: SWV is not needed for this window:

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 Task Aware windows for RTX and FreeRTOS. Other RTOS companies also provide awareness plug-ins for  $\mu$ Vision.

This window uses DAP Reads and Writes. This is the same as in Watch, Memory and Peripheral windows. SWV is not used.

1. Run RTX\_Blinky again by clicking on the Run icon.
2. Open Debug/OS Support and select RTX System and Thread Viewer. The window below 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.

1. Stop the program.
2. There are three threads plus the idle daemon.
3. Note the idle demon is Running in the State column. In this project, the program spends most of its time in the idle state. This is easily adjusted.
4. Set a breakpoint in the while() loop in Blinky.c.
5. RUN the program. When it stops, the main thread will be Running.
6. Remove the breakpoint.

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: 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	469			32%
3	Thread_LED	Normal	Wait_AND		0x0000	0x0001	40%
255	os_idle_demon	None	Running				

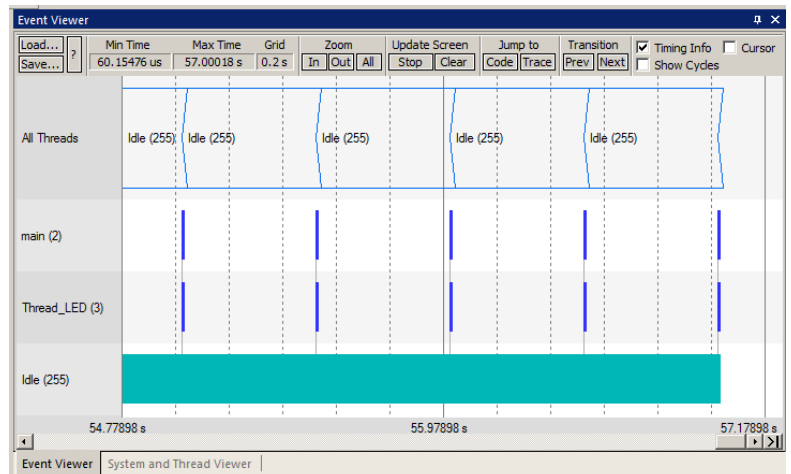
**TIP:** View/Periodic Window Update must be selected ! Otherwise, windows update only when the program is stopped.

## 19) Event Viewer: Needs SWV: For more information on SWV see: [www.keil.com/appnotes/docs/apnt\\_286.asp](http://www.keil.com/appnotes/docs/apnt_286.asp)

1. Serial Wire Viewer must be configured as described on page 12. **Configuring SWV:**
2. Open Debug/OS Support and select Event Viewer. The window below will open.
1. 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 0.5 seconds by clicking on the Zoom ALL and then the + and – icons.
2. You can tell at a glance if your implementation of RTX is running as you expect.
3. You can make timing measurements by selecting Timing Info and Cursor. Click and/or hover the mouse over an interesting waveform. See the screen below.
4. Use Start and Stop in the Update Screen area. The program will keep running.
5. Note the CPU spends most of its time in the idle demon.

**Trouble?:** If Event Viewer doesn't work, open up the Trace Records and confirm there are good ITM 31 frames present with not too many "x" in the Dly or OVF columns. ITM Stimulus Port 31 must be selected. Is Core Clock set correctly ? This project is running at 168 MHz. If the waveform is distorted, try unselecting EXCTRC and any variables in the Logic Analyzer. Check the Trace Records window for any frames you did not turn on.

Use of a ULINKplus or ULINKpro will provide better SWV performance if necessary.



This panel shows the time between each thread active is approximately 0.5 seconds. →


Timing of 'Thread_LED' (Thread #3 @0x080004a2)			
All Slices:	Min	Max	Average
Count: 4632	8.047619 us	8.047619 us	8.047619 us
Cursors:	Mouse	Reference	Difference
	2314.503 s	2314 s	0.5028 s = 1.988862 Hz

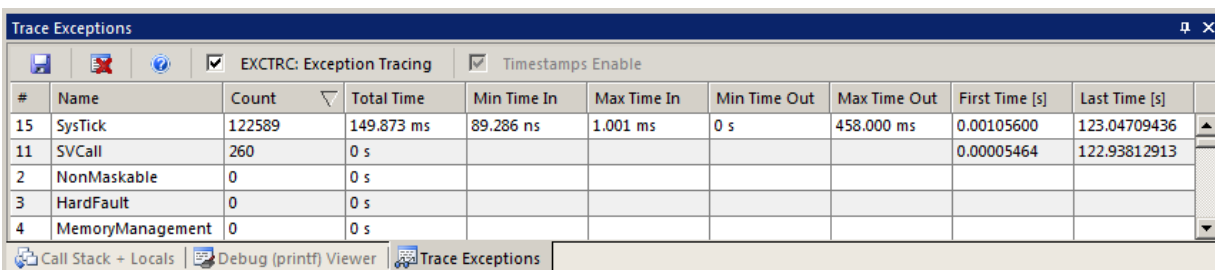
## 20) Exceptions and Interrupts: SWV is needed:

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 processors makes this task easy.

Exceptions can be displayed in Trace Records and Trace Exceptions windows. If using a ULINK $pro$  and RTX (soon FreeRTOS), exceptions can be displayed in a graphical format in the Event Viewer.

### Trace Exceptions Window:

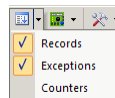
1. Click RUN  to start the program.
2. EXCTRC can be activated in the Trace Configuration window or the Trace Exceptions window.
3. Open the Trace Exceptions window. It is normally grouped with the Call Stack, Watch and Memory windows.
4. Select EXCTRC Exception Tracing: ☒ EXCTRC: Exception Tracing
5. With Blinky running, click on the column header containing Count. This brings active exceptions to the top.
6. SysTick and SVCcall will be displayed as shown below:
7. If SVCcall does not increment often, the SWO pin is overloaded. Delete the variable in the Logic Analyzer.



#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
15	SysTick	122589	149.873 ms	89.286 ns	1.001 ms	0 s	458.000 ms	0.00105600	123.04709436
11	SVCcall	260	0 s					0.00005464	122.93812913
2	NonMaskable	0	0 s						
3	HardFault	0	0 s						
4	MemoryManagement	0	0 s						

### Trace Records Window:

1. Open Trace Records. Exceptions will be listed.
2. This window is shown below on the right side.
  - **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.



Right-click in this window to filter out other frames.

**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.

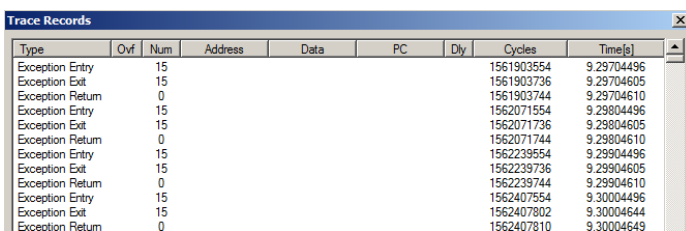
### Viewing Exceptions Graphically with ULINK $pro$ :

1. If you are using RTX and a ULINK $pro$ , you can display exceptions graphically. This is useful to view how long exception/interrupt handlers take. This feature will be expanded to include FreeRTOS and no RTOS (bare metal).
2. The window below right displays four exceptions at the bottom of the Event Viewer.

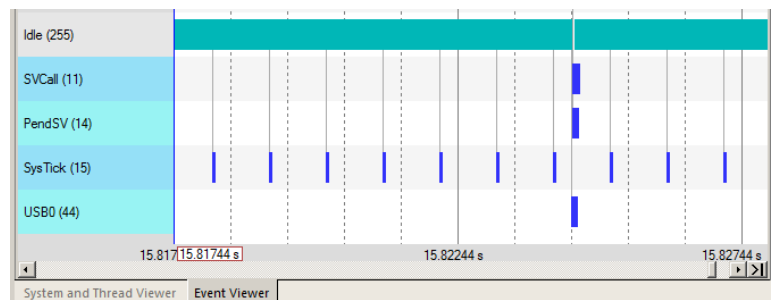
Shown below right are the SysTick, PendSV, SVCcall and USB0 exceptions. You can easily measure the duration of the time spent in the handlers. Any other exception events such as DMA will also be displayed here.

You will be able to easily measure the time a handler runs with the Event Viewer techniques you have learned.

In this window, USB0 (interrupt 44) handler runs for approximately 10 msec.



Type	Ofs	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					1561903554	9.29704496
Exception Exit		15					1561903736	9.29704605
Exception Return		0					1561903744	9.29704610
Exception Entry		15					1562071554	9.29804496
Exception Exit		15					1562071736	9.29804605
Exception Return		0					1562071744	9.29804610
Exception Entry		15					1562239554	9.29904496
Exception Exit		15					1562239736	9.29904605
Exception Return		0					1562239744	9.29904610
Exception Entry		15					1562407554	9.30004496
Exception Exit		15					1562407802	9.30004644
Exception Return		0					1562407810	9.30004649





## 21) DSP SINE example using ARM CMSIS-DSP Libraries:

ARM CMSIS-DSP libraries are offered for all ARM Cortex-M processors. DSP libraries are provided in MDK in C:\Keil\_v5\ARM\PACK\ARM\CMSIS\5.0.1\CMSIS\DSP\_Lib. They are easily selected with the MRTE utility. Documentation is located here: [www.keil.com/pack/doc/cmsis/DSP/html/](http://www.keil.com/pack/doc/cmsis/DSP/html/).





CMSIS is an acronym for Cortex Microcontroller Software Interface Standard. CMSIS is an ARM standard.

This example creates a sine wave, then a second higher frequency sine wave is created and added to the first one, and then it is filtered out. The waveform in each step is displayed in the Logic Analyzer using Serial Wire Viewer.

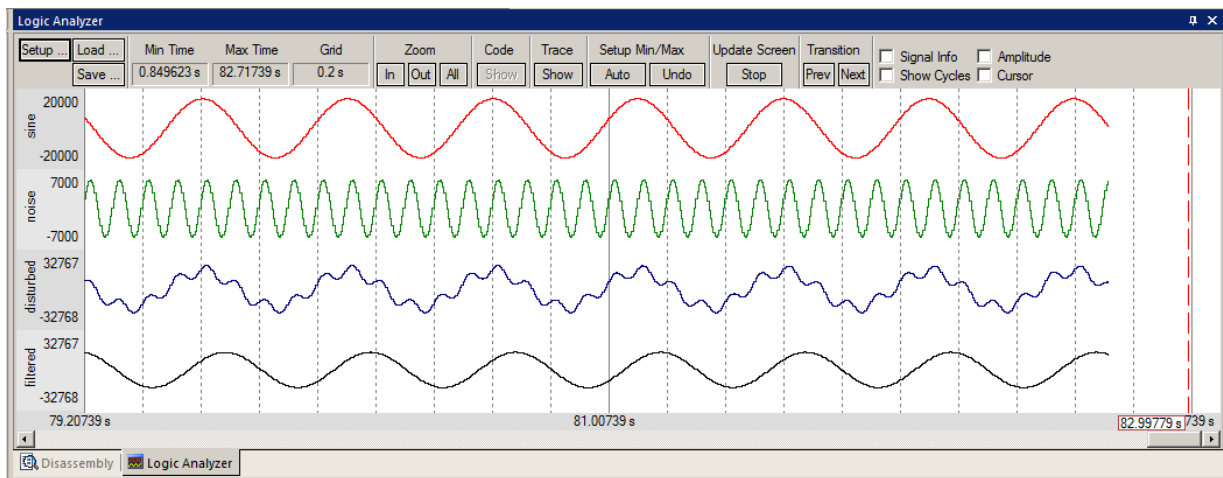
This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. RTX source code is provided.

To obtain this example file, go to [www.keil.com/appnotes/docs/apnt\\_253.asp](http://www.keil.com/appnotes/docs/apnt_253.asp) and get the example file.

Extract the zip to C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery to create a \DSP directory.

1. Open the project file sine: C:\00MDK\MDK\Boards\ST\STM32F429I-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. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.
6. 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.



7. Open the Trace Records window and the Data Writes to the four variables are listed as shown here:

**TIP:** The ULINK<sub>pro</sub> trace display is different and the program must be stopped to update it.

Select View/Watch Windows and select Watch 1 and it 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>		

***This ends the exercises. Thank you !***

Next is how to make a new project from scratch (almost scratch) and Keil product and contact information.

Trace Records							
Type	Ovf	Num	Address	Data	PC	Dly	Cycles
Data Write			20000000H	2C2DH	00400252H		9741265867
Data Write			20000002H	F081H	00400280H	X	9741274850
Data Write	X		20000006H	F326H	004002CEH	X	9741274850
Data Write			20000000H	2EF4H	00400252H		9741745582
Data Write			20000002H	F5CBH	00400280H	X	9741754550
Data Write	X		20000006H	F6FFH	004002CEH	X	9741754550
Data Write			20000000H	318CH	00400252H		9742225305
Data Write			20000002H	FC15H	00400280H	X	9742234332
Data Write	X		20000006H	FAE2H	004002CEH	X	9742234332
Data Write			20000000H	33F1H	00400252H		9742705028
Data Write			20000002H	02CDH	00400280H	X	9742714032
Data Write	X		20000006H	FECBH	004002CEH	X	9742714032
Data Write			20000000H	3621H	00400252H		9743184839
Data Write			20000002H	0927H	00400280H	X	9743193814
Data Write	X		20000006H	02B7H	004002CEH	X	9743193814
Data Write			20000000H	381AH	00400252H		9743664482
Data Write			20000002H	0EA9H	00400280H	X	9743673432
Data Write	X		20000006H	06A2H	004002CEH	X	9743673432
Data Write			20000000H	39DAH	00400252H		9744144197
Data Write			20000002H	12BAH	00400280H	X	9744153214

## 22) Creating your own MDK 5 project from scratch:

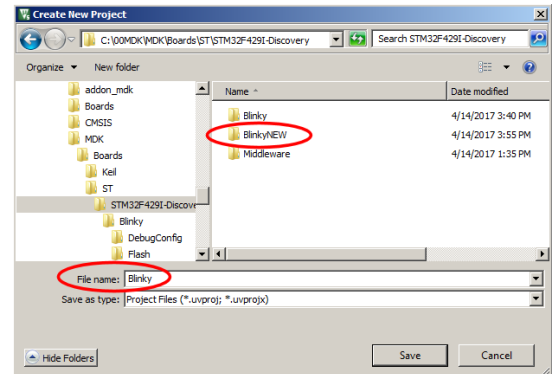
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load 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 such as RTX.

### Install the STM32 Software Pack for your processor:

1. Start  $\mu$ Vision and leave in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Pack for the STM32F4 processor must be installed. This has already been done on page 4.

### Create a new Directory and a New Project:

1. Click on Project/New  $\mu$ Vision Project...
2. In the window that opens, shown here, go in:  
C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on Blinky to open it or select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj in C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery\BlinkyNEW\.
7. As soon as you click on Save, the next window opens:



### Select the Device you are using:

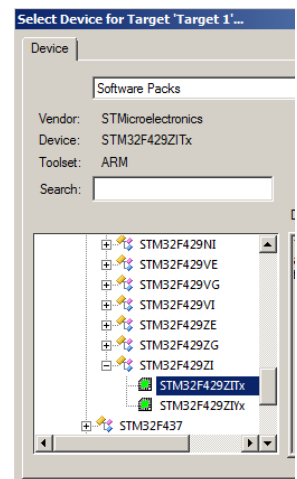
1. Expand STMicroelectronics, then STM32F4 Series, then STM32F429, then STM32F429ZI and then finally select STM32F429ZITx:

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

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

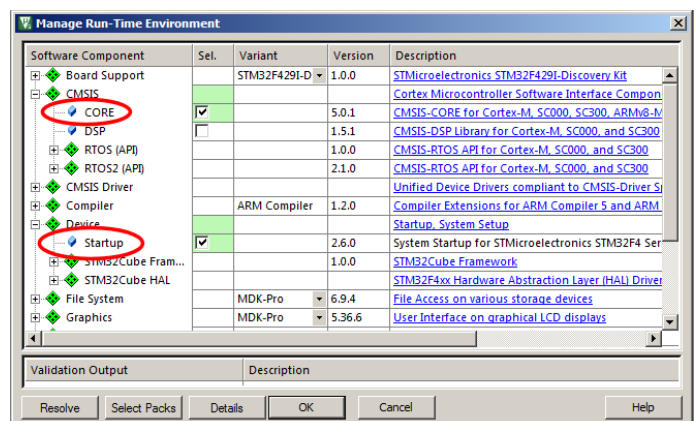
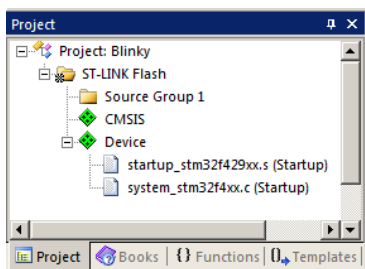
### Select the CMSIS components you want:

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.
2. Select your Discovery board in Board Support. Click OK to close this window.
3. The project Blinky.uvproj is now 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.
7. Click on File/Save All or select the Save All icon:




### What has happened to this point:

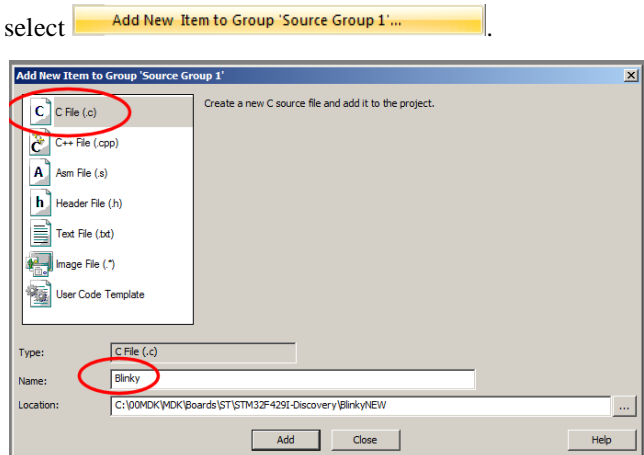
You have created a blank  $\mu$ Vision project using MDK 5 Software Packs. All you need to do now is add your own source files.



**Continued on the next page...**

## 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 as a Source window.





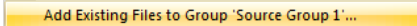
## Add Some C Code to Blinky.c:

1. Right click in Blinky.c and select 'Insert #include file'
2. Select stm32f4xx.h and then repeat for RTE\_Components.h. These are added to Blinky.c.
3. In the blank portion of Blinky.c, add the C code below:


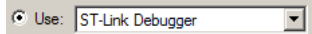




```
unsigned int counter = 0;
/*-----
MAIN function
-----*/
int main (void) {

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



4. Click on File/Save All or 
5. Build the files.  There will be no errors or warnings if all was entered correctly.


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


## Configure ST-Link, and confirm the Correct St Link Flash: *Complete these instructions carefully...*

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 Debug tab. Select **ST-Link Debugger** in the Use: selection box. 
4. Select the Settings: icon.
5. Select SW as shown here in the Port: box:  JTAG 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.
6. Click on OK **once** to go back to the Target Configuration window. Otherwise, fix the connection problem.
7. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm as shown: Shown is the correct one for the STM32F429 series processors. 
8. Click on OK twice to return to the main menu.
9. Click on File/Save All or 
10. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

## Running Your Program:

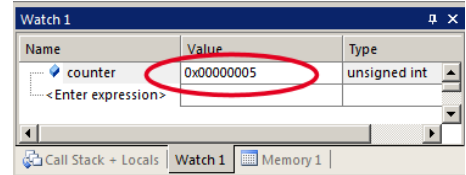
1. Enter Debug mode by clicking on the Debug icon .  Your program will be programmed into the ST Flash.
2. The program will run to the beginning of main() and stop.
3. Click on the RUN icon. 

**Note:** you stop the program with the STOP icon. 

4. No LEDs will blink since there is no source to accomplish this task. You could add such code yourself later.
5. Right click on counter in Blinky.c and select Add counter to ... and select Watch 1.
6. counter should be updating as shown here: 
7. 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.**

8. 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:** 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. Counter is changing faster than the Logic Analyzer can display. This is because SWV is so overloaded that the timestamps are the same.

**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:

We are using 3) in this exercise:

- 1) **STM32CubeMX.** 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/](http://www.st.com/stm32cubemx/). For information on creating projects with STM32CubeMX see: [www.keil.com/pack/doc/STM32Cube/General/html/](http://www.keil.com/pack/doc/STM32Cube/General/html/)
- 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/](http://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](http://www.keil.com/pack/doc/STM32Cube/General/html/index.html)

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

**MDK 5 and MDK 4 projects:** MDK 5 uses Software packs and MDK 4 does not. This tutorial uses MDK 5 projects which have a filename extension .uvprojx. Legacy MDK 4 projects (with an extension .uvproj) can be converted to MDK 5: Select **Project/Manage/Migrate to Version 5 format...** You will be prompted to do this when a MDK 4 project is loaded.

You can also use the MDK 5 Legacy support for processors not supported with a Software Pack. Go to [www.keil.com/mdk](http://www.keil.com/mdk) and select MDK v4 Legacy Support. This adds all the files required for MDK 4 projects.

**ELF/DWARF:** ARM Compiler 5 and ARM Compiler 6 (LLVM) both produce an .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.



*Now, we will add RTX.*

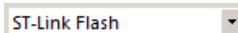
## 23) Adding RTX to your MDK 5 project:

The MDK Software Packs makes it easy to configure an RTX project. We will use the 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.

See [www.keil.com/pack/doc/CMSIS/RTOS/html/](http://www.keil.com/pack/doc/CMSIS/RTOS/html/) and [www.keil.com/pack/doc/CMSIS/RTOS2/html/rtx5\\_impl.html](http://www.keil.com/pack/doc/CMSIS/RTOS2/html/rtx5_impl.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: 

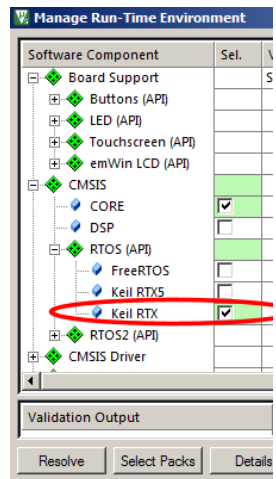
4. Expand the elements as shown here: 

5. Select Keil RTX as shown and click OK. Do not select RTX 5.


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 

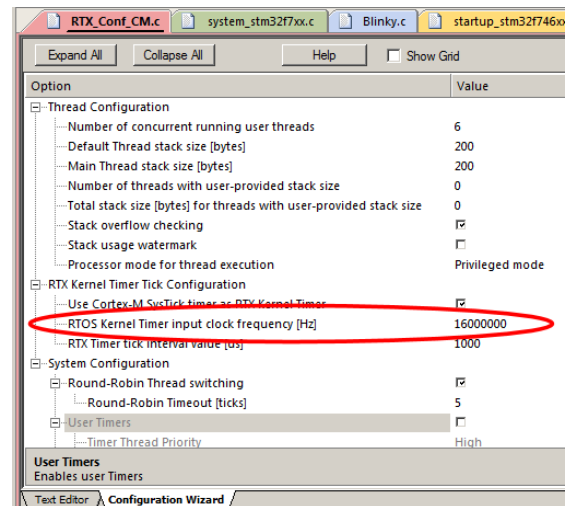


### Configure RTX:

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.

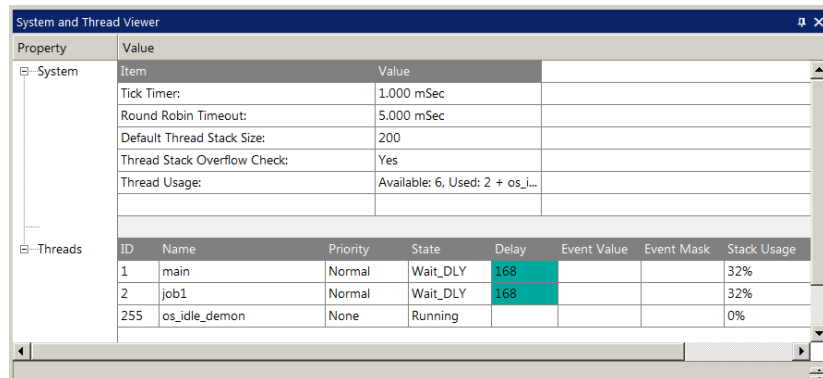
### Build and Run Your RTX Program:

1. Click on File/Save All or 
2. Build the files. 
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 to add a thread.
2. **Getting Started Guide MDK**  
Obtain this useful book here: [www.keil.com/gsg/](http://www.keil.com/gsg/). It has useful information on implementing RTX.



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: 6, Used: 2 + os_i...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	main	Normal	Wait_DLY	168			32%
2	job1	Normal	Wait_DLY	168			32%
255	os_idle_demon	None	Running				0%

5:  
very



## 24) 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. Stop the program  and Exit Debug mode. 

2. In Blinky.c, add this line near line 6 before the main() function:

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

### Create the Thread job1:

3. Add this code to be the thread **job1** before main():

**TIP:** osDelay(1000) delays the program by 1000 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 > 0x0F) counter2=0;
12         osDelay(1000);
13     }
14 }
```

### Add osDelay to main():

4. Add this line just after the if statement near line 21:

```
21 osDelay(1000);
```

### Define and Create the Thread:


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

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

6. Create the thread job1 near line 18 just before the while(1) loop:

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

7. Click on File/Save All or 

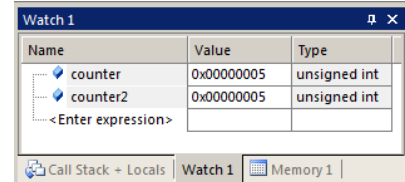
8. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.

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

9. Enter Debug mode:  Click on the RUN icon. 

10. Right click on counter2 in Blinky.c and select Add counter2 to ... and select Watch 1.

11. Both counter and counter2 will increment but slower than before:  
The two osDelay(1000) function calls each slow the program down by 1000 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>		

12. Open the System and Thread Viewer by selecting Debug/OS Support.

13. Note that job1 has now been added as a thread as shown below:

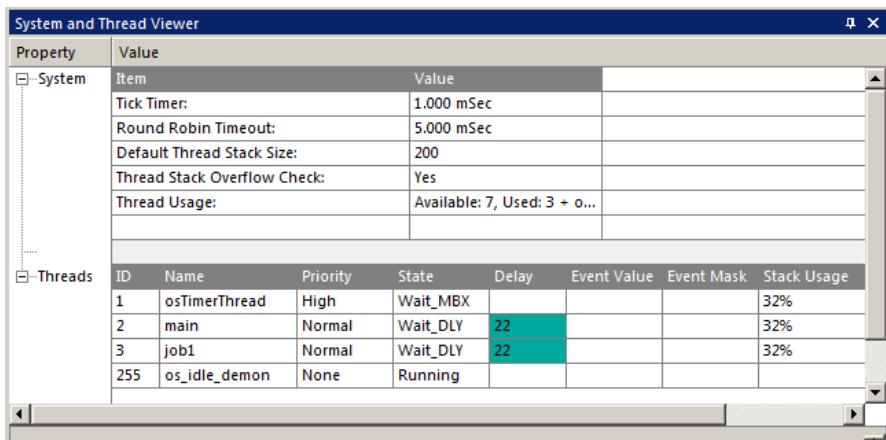
14. Note os\_idle\_demon is always labelled as Running. This is because the program spends most of its time here.

15. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.

16. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.

17. Remove all breakpoints before continuing.

18. There are many attributes of RTX you can add. See the RTX documentation mentioned on the previous page and the MDK 5 Getting Started Guide.









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: 3 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				32%
2	main	Normal	Wait_DLY	22			32%
3	job1	Normal	Wait_DLY	22			32%
255	os_idle_demon	None	Running				

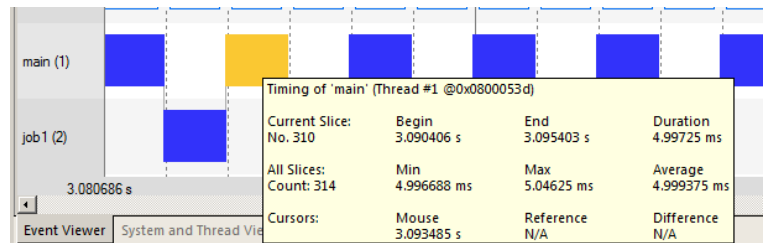
## 25) Using Event Viewer to examine your RTX\_Blinky Timing: Needs SWV

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(1000);`. Comment both of these out. We will run the program really fast.
- 3) Open Select Target Options  or ALT-F7. Select the Debug tab and the Settings. Select the Trace tab.
- 4) Set Core Clock: to 16 MHz. Unselect EXCTRC: and Periodic. Leave everything else at their default settings.
- 5) Click on OK twice to return to the main µVision 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.

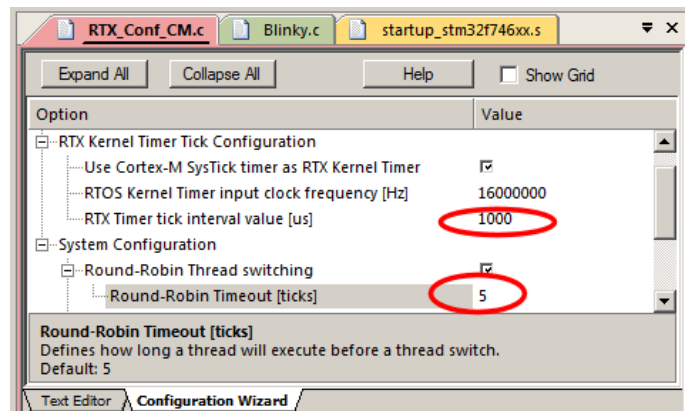


- 9) Hold your cursor over one of the blue blocks and a yellow information window is displayed as shown here:
- 10) Note each thread duration is about 5 msec.



### 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  $1 \text{ msec} * 5 \text{ ticks} = 5 \text{ msec}$ .
- 5) Every 5 msec the Thread is switched to the next one and these sequences are displayed in the Event Viewer. There are other ways to switch a thread.
- 6) It is quite easy to view how RTX is running to make sure it is performing as you designed.
- 7) There are many other RTX features you can use. Refer to the extensive RTX documentation.
- 8) Stop the program  and exit Debug mode. 

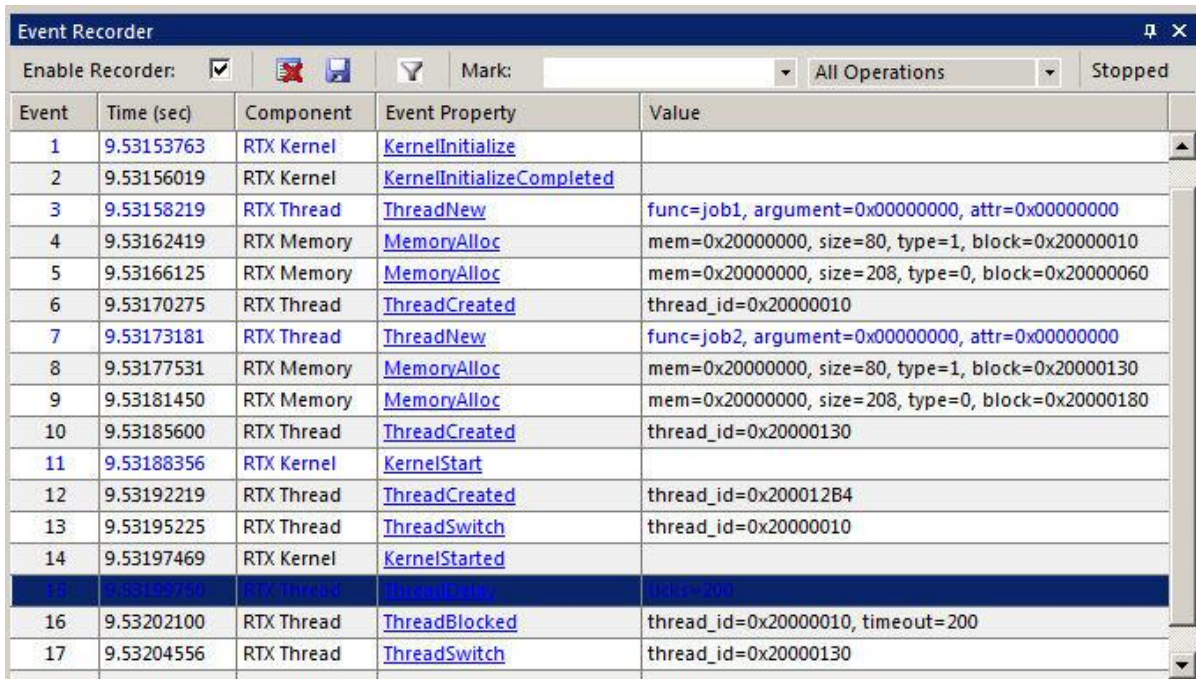


The next page demonstrates the new Event Recorder. We will use a preconfigured Blinky running the new RTX v5.

## 26) Event Recorder:





Event Recorder is a new  $\mu$ Vision feature. Code annotations can be inserted into your code to send out messages to  $\mu$ Vision and be displayed as shown below. Keil Middleware and RTX5 have these annotations already inserted. You can add Event Recorder annotations to your own source code. SWV is not used. DAP Reads and Writes are used.

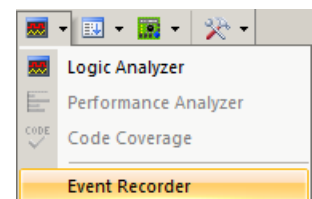
Documentation for Event Recorder is found here: [www.keil.com/pack/doc/compiler/EventRecorder/html/](http://www.keil.com/pack/doc/compiler/EventRecorder/html/)



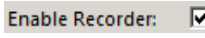


Event	Time (sec)	Component	Event Property	Value
1	9.53153763	RTX Kernel	KernelInitialize	
2	9.53156019	RTX Kernel	KernelInitializeCompleted	
3	9.53158219	RTX Thread	ThreadNew	func=job1, argument=0x00000000, attr=0x00000000
4	9.53162419	RTX Memory	MemoryAlloc	mem=0x20000000, size=80, type=1, block=0x20000010
5	9.53166125	RTX Memory	MemoryAlloc	mem=0x20000000, size=208, type=0, block=0x20000060
6	9.53170275	RTX Thread	ThreadCreated	thread_id=0x20000010
7	9.53173181	RTX Thread	ThreadNew	func=job2, argument=0x00000000, attr=0x00000000
8	9.53177531	RTX Memory	MemoryAlloc	mem=0x20000000, size=80, type=1, block=0x20000130
9	9.53181450	RTX Memory	MemoryAlloc	mem=0x20000000, size=208, type=0, block=0x20000180
10	9.53185600	RTX Thread	ThreadCreated	thread_id=0x20000130
11	9.53188356	RTX Kernel	KernelStart	
12	9.53192219	RTX Thread	ThreadCreated	thread_id=0x200012B4
13	9.53195225	RTX Thread	ThreadSwitch	thread_id=0x20000010
14	9.53197469	RTX Kernel	KernelStarted	
15	9.53200750	RTX Thread	ThreadBlocked	thread_id=0x20000010, timeout=200
16	9.53202100	RTX Thread	ThreadBlocked	thread_id=0x20000010, timeout=200
17	9.53204556	RTX Thread	ThreadSwitch	thread_id=0x20000130

### Demonstrating Event Recorder with RTX5\_Blinky:

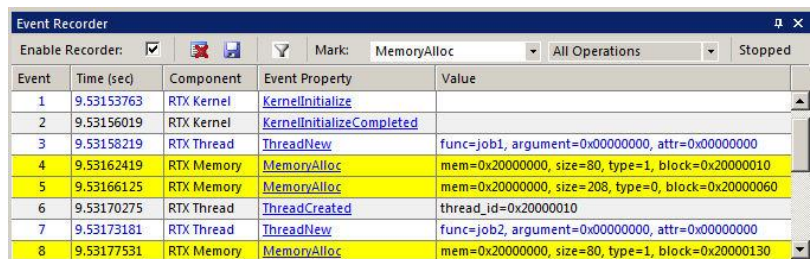
1. Open Blinky.uvprojx in C:\00MDK\MDK\Boards\ST\STM32F429I-Discovery\RTX5\_Blinky\.
2. Click on Rebuild. . Enter Debug mode.  Click on RUN .
3. Open Event Recorder by selecting View/Analysis/Event Recorder or  →
4. Since Event Recorder is activated in Blinky.c, the window above will display.
5. Various RTX events will display as they happen. You can do this for your own code.





### Event Recorder Features:

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

osThreadFlagsSet function was called.



Event	Time (sec)	Component	Event Property	Value
1	9.53153763	RTX Kernel	KernelInitialize	
2	9.53156019	RTX Kernel	KernelInitializeCompleted	
3	9.53158219	RTX Thread	ThreadNew	func=job1, argument=0x00000000, attr=0x00000000
4	9.53162419	RTX Memory	MemoryAlloc	mem=0x20000000, size=80, type=1, block=0x20000010
5	9.53166125	RTX Memory	MemoryAlloc	mem=0x20000000, size=208, type=0, block=0x20000060
6	9.53170275	RTX Thread	ThreadCreated	thread_id=0x20000010
7	9.53173181	RTX Thread	ThreadNew	func=job2, argument=0x00000000, attr=0x00000000
8	9.53177531	RTX Memory	MemoryAlloc	mem=0x20000000, size=80, type=1, block=0x20000130


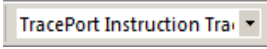




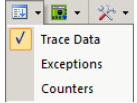
1. Stop the program.  Close any Event Recorder and Threads and Event (RTX RTOS) windows.
2. Exit Debug mode.  **This is the end of the exercises. Next page shows how ETM Instruction Trace works.**

## 27) ETM Trace Examples: For convenient reference...

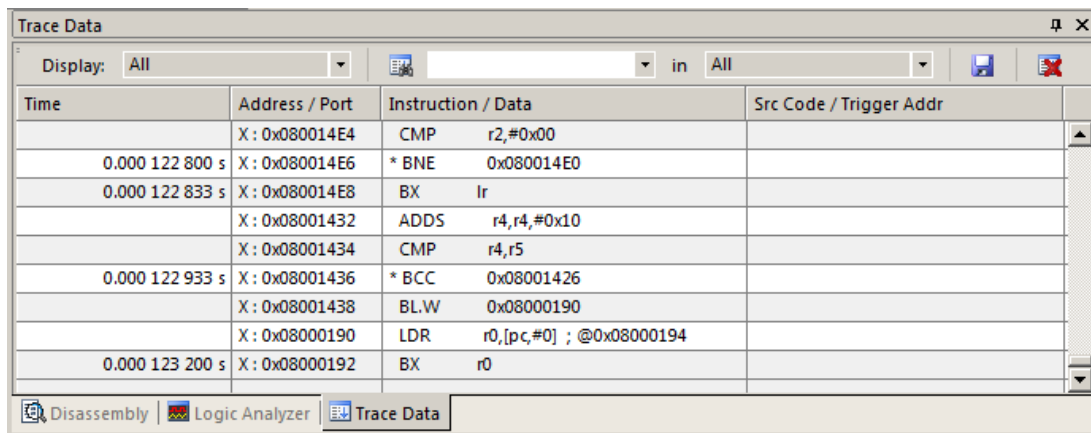
These examples were run on the STM3240G-EVAL evaluation board. These are applicable for the Keil MCBSTM32F400 board. These examples are included for reference. A ULINK<sub>pro</sub> debug adapter is required for ETM operation.

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. Connect the ULINK<sub>pro</sub> to the STM3240G board using the 20 pin CN13 Trace connector.
2. Start  $\mu$ Vision by clicking on its desktop icon. 
3. Select Project/Open Project. Open C:\Keil\ARM\Boards\ST\STM3240G-EVAL\Blinky\_Ulp\Blinky.uvproj.
4. Select TracePort Instruction Trace in the Target Options box as shown here: 
5. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
6. Program the STM32 flash by clicking on the Load icon: . Progress will be indicated in the Output Window.
7. Enter Debug mode by clicking on the Debug icon. . Select OK if the Evaluation Mode box appears.
8. DO NOT CLICK ON RUN YET !!!
9. Open the Data Trace window by clicking on the small arrow beside the Trace Windows icon.  

10. Examine the Instruction Trace window as shown below: This is a complete record of all the program flow since RESET until  $\mu$ Vision halted the program at the start of main() since Run To main is selected in  $\mu$ Vision.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x080014E4	CMP r2,#0x00	
0.000 122 800 s	X: 0x080014E6	* BNE 0x080014E0	
0.000 122 833 s	X: 0x080014E8	BX lr	
	X: 0x08001432	ADDS r4,r4,#0x10	
	X: 0x08001434	CMP r4,r5	
0.000 122 933 s	X: 0x08001436	* BCC 0x08001426	
	X: 0x08001438	BL.W 0x08000190	
	X: 0x08000190	LDR r0,[pc,#0] ; @0x08000194	
0.000 123 200 s	X: 0x08000192	BX r0	

11. In this case, 123 200 s shows the last instruction to be executed. (BX r0). In the Register window the PC will display the value of the next instruction to be executed (0x0800\_0192 in my case). Click on Single Step once. 
12. The instruction PUSH will display: | 0x080011DA | PUSH (r3,lr) | int main(void) { /\* Main Program \*/ |
13. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.



**A STM3240G-EVAL board connected to a ULINK<sub>pro</sub> using the special CoreSight 20 pin ETM connector:**










nearly



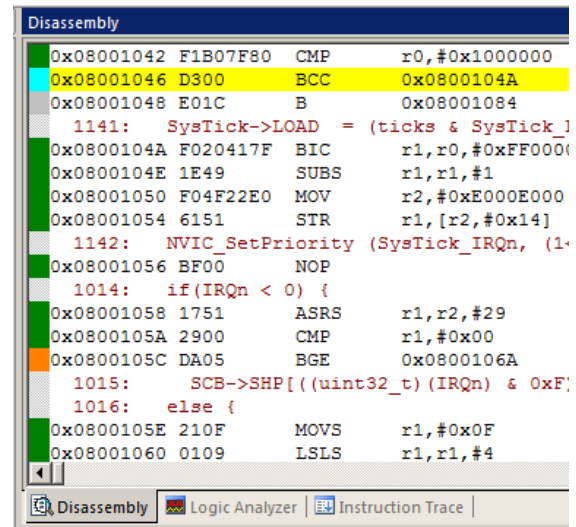
## 1) Code Coverage: For convenient reference ...

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

Colour blocks indicate which assembly instructions have been executed.

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

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



```
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
```

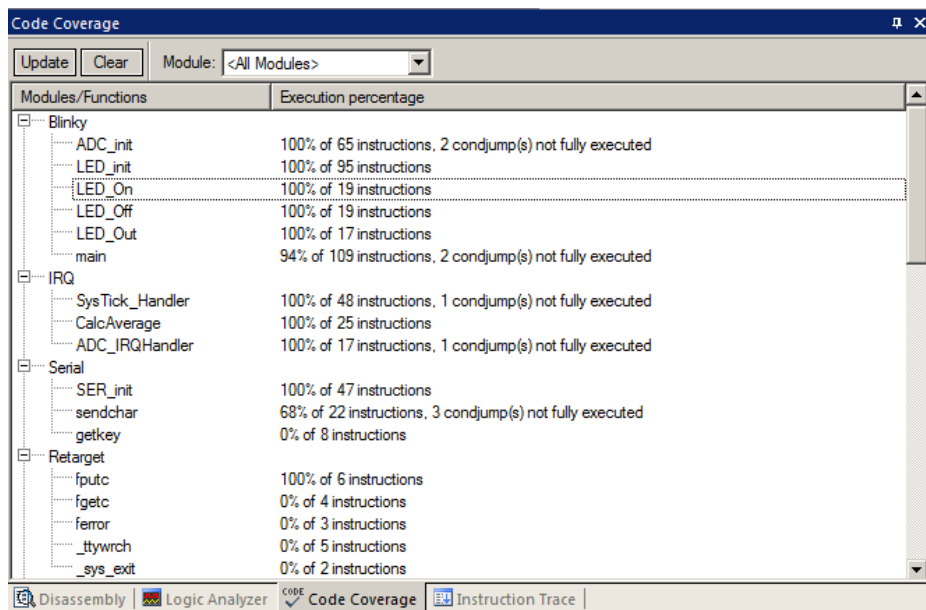
Why was the branch BCC always taken resulting in 0x0800\_1048 never being executed ? Or why the branch BGE at 0x800\_105C was never taken ? You should devise tests to execute these instructions so you can test them.

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
Blinky	
ADC_init	100% of 65 instructions, 2 condjump(s) not fully executed
LED_init	100% of 95 instructions
LED_On	100% of 19 instructions
LED_Off	100% of 19 instructions
LED_Out	100% of 17 instructions
main	94% of 109 instructions, 2 condjump(s) not fully executed
IRQ	
SysTick_Handler	100% of 48 instructions, 1 condjump(s) not fully executed
CalcAverage	100% of 25 instructions
ADC_IRQHandler	100% of 17 instructions, 1 condjump(s) not fully executed
Serial	
SER_init	100% of 47 instructions
sendchar	68% of 22 instructions, 3 condjump(s) not fully executed
getkey	0% of 8 instructions
Retarget	
fputc	100% of 6 instructions
fgetc	0% of 4 instructions
ferror	0% of 3 instructions
_ttywrch	0% of 5 instructions
_sys_exit	0% of 2 instructions

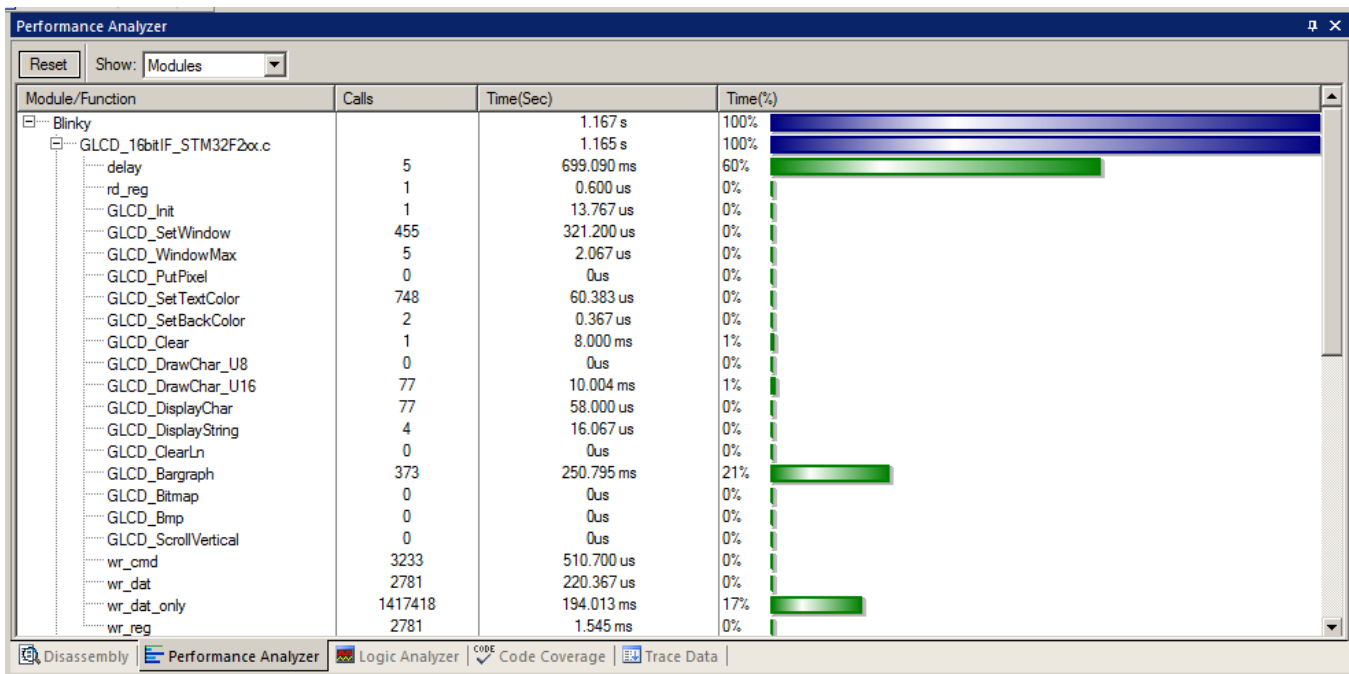



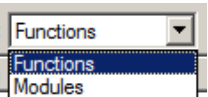
## 2) Performance Analysis (PA): For convenient reference ...


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  $\mu$ Vision simulator or with ETM and the ULINK $pro$ . 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 GLCD routines.



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.

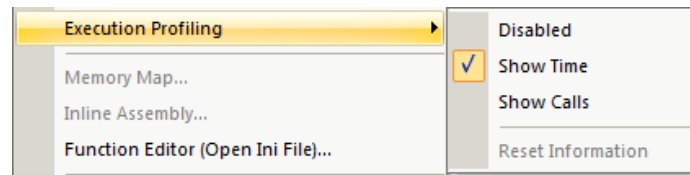
**TIP:** You can also click on the RESET icon  but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

When you click on the RESET icon, the Initialization File .ini will no longer be in effect and this can cause SWV and/or ETM to stop working. Exiting and re-entering Debug mode executes the .ini script again.

### 3) Execution Profiling: *For reference only...*

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  $\mu$ Vision simulator also provides Execution Profiling.

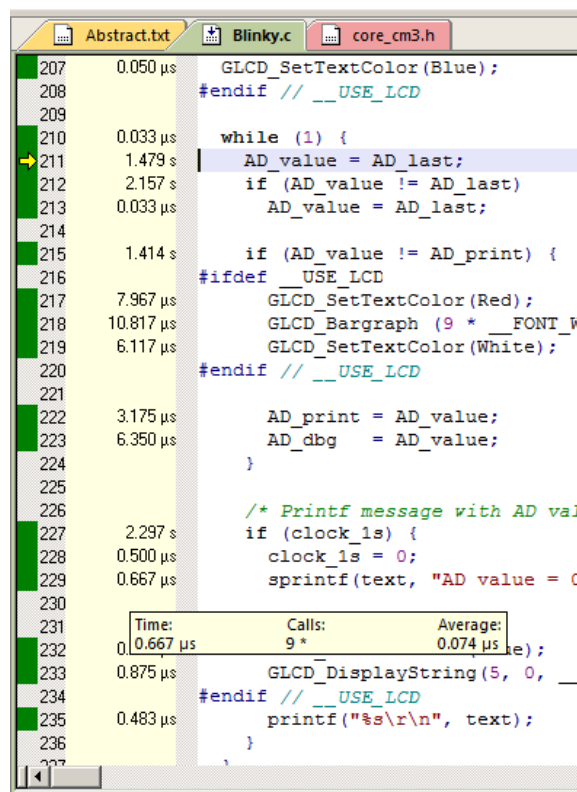
1. Enter Debug mode.
2. Select Debug/Execution Profiling/Show Time.
3. In the left margin of the disassembly and C source windows will display various time values.
4. Click on RUN.



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 $\mu$ s

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



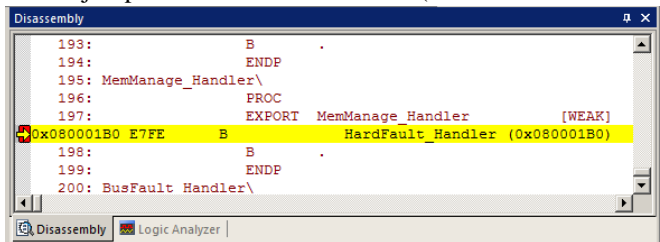
#### 4) In-the-Weeds Example: For reference only...


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.

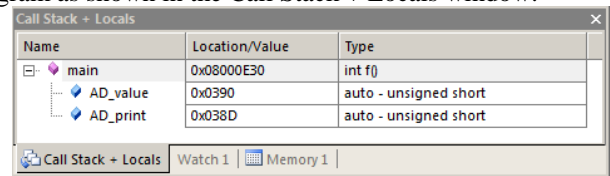
If a Hard Fault occurs, the CPU will end up at the address specified in the Hard Fault vector located at 0x00 000C. This address points to the Hard Fault handler. This is usually a branch to itself and this Branch instruction will run forever. The trace buffer will save millions of the same branch instructions. This is not useful. We need to stop the CPU at this point.

This exception vector is found in the file startup\_stm32f4xx.s. If we set a breakpoint by double-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 01B0 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\_Ulp example, rebuild, program the Flash and enter Debug mode. Open the Data Trace window.
2. Locate the Hard fault vector near line 207 in the disassembly window or in startup\_stm32f4xx.s.
3. Set a breakpoint at this point. A red block will appear as shown above.
4. Run the Blinky example for a few seconds and click on STOP.
5. Click on the Step\_Out icon  to go back to the main() program as shown in the Call Stack + Locals window:
6. In the Disassembly window, scroll down until you find a POP instruction. I found one at 0x0800 1256 as shown below in the third window:
7. Right click on the POP instruction (or at the MOV at 0x0800 124E as shown below) and select Set Program Counter. This will be the next instruction executed.
8. Click on RUN and immediately the program will stop on the Hard Fault exception branch instruction.
9. Examine the Data Trace window and you find this POP plus everything else that was previously executed. In the bottom screen are the 4 MOV instructions plus the offending POP.
10. Note the Branch at the Hard Fault does not show in the trace window because a hardware breakpoint does execute the instruction it is set to therefore it is not recorded in the trace buffer.



```
0x08001248 F1A0401 SUB    r4,r4,#0x01
0x0800124C DCFD    BGT    0x0800120E
0x0800124E 4648    MOV    r0,r9
0x08001250 4631    MOV    r1,r6
0x08001252 462A    MOV    r2,r5
0x08001254 4643    MOV    r3,r8
0x08001256 E8BD9FF0 POP    {r4-r12,pc}
0x0800125A 0000    MOVS   r0,r0
                _scatterload:
```

Trace Data			
Display: All in All			
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X: 0x0800DD4	LDRB r1,[r0,#0x00]	
	X: 0x0800DD6	CBZ r1,0x0800DE0	
1.215 414 190 s	X: 0x0800DE0	MOV r0,#0xFFFFFFFF	return -1; /* Conv. in pro...
1.215 414 210 s	X: 0x0800DE4	BX lr	
	X: 0x0800124E	MOV r0,r9	
	X: 0x08001250	MOV r1,r6	
	X: 0x08001252	MOV r2,r5	
	X: 0x08001254	MOV r3,r8	
1.215 420 300 s	X: 0x08001256	POP {r4-r12,pc}	

The frames above the POP are a record of all previous instructions executed and tells you the complete program flow.

## 28) 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 the 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*.
- 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).
- Plus - don't have to stop the program. Crucial to some projects.

### 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](http://www.keil.com/st) for other labs discussing ETM.

## 29) Document Resources:

**www.keil.com/st**

### Books:

1. **NEW!** **Getting Started Guide MDK 5:** Obtain this free book here: [www.keil.com/gsg/](http://www.keil.com/gsg/).
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:

9. Using Cortex-M3 and Cortex-M4 Fault Exceptions [www.keil.com/appnotes/files/apnt209.pdf](http://www.keil.com/appnotes/files/apnt209.pdf)
10. Segger emWin GUIBuilder with  $\mu$ Vision™ [www.keil.com/appnotes/files/apnt\\_234.pdf](http://www.keil.com/appnotes/files/apnt_234.pdf)
11. Porting mbed Project to Keil MDK™ [www.keil.com/appnotes/docs/apnt\\_207.asp](http://www.keil.com/appnotes/docs/apnt_207.asp)
12. MDK-ARM™ Compiler Optimizations [www.keil.com/appnotes/docs/apnt\\_202.asp](http://www.keil.com/appnotes/docs/apnt_202.asp)
13. Using  $\mu$ Vision with CodeSourcery GNU [www.keil.com/appnotes/docs/apnt\\_199.asp](http://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\_RTXXDownload
15. RTX CMSIS-RTX [www.keil.com/demo/eval/rtx.htm](http://www.keil.com/demo/eval/rtx.htm) and [www.arm.com/cmsis](http://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](http://www.arm.com) and search for DAI0298A
18. Cortex Debug Connectors: [www.arm.com](http://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](http://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](http://www.keil.com/appnotes/docs/apnt_270.asp)
21. **NEW!** ARMv8-M Architecture Technical Overview <https://community.arm.com/docs/DOC-10896>
22. **NEW!** Using ST-LINK/V2 with Serial Wire Viewer: [www.keil.com/appnotes/docs/apnt\\_286.asp](http://www.keil.com/appnotes/docs/apnt_286.asp)
23. **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 Tutorials for STMicroelectronics Boards:** see [www.keil.com/st](http://www.keil.com/st)

### Useful ARM Websites:

1. CMSIS Standards: [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5) and [www.keil.com/cmsis/](http://www.keil.com/cmsis/)
2. **Forums:** [www.keil.com/forum](http://www.keil.com/forum) <https://developer.arm.com/> <http://community.arm.com/groups/tools/content>
3. ARM University Program: [www.arm.com/university](http://www.arm.com/university). Email: [university@arm.com](mailto:university@arm.com)
4. **mbed™:** <http://mbed.org>

**Keil Direct Sales In USA:** [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. **Outside the US:** [sales.intl@keil.com](mailto:sales.intl@keil.com)

**Keil Distributors:** See [www.keil.com/distis/](http://www.keil.com/distis/) **DS-5 Direct Sales Worldwide:** [orders@arm.com](mailto:orders@arm.com)

**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).

For comments, additions or corrections please email [bob.boys@arm.com](mailto:bob.boys@arm.com)



### 30) 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](http://www.keil.com/st) **\$0** Free MDK for STM32 F0/L0
- **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).

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

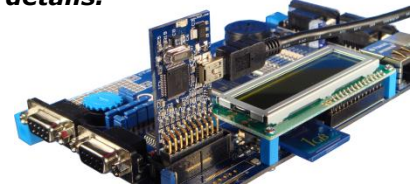
Keil Middleware includes Network, USB, Graphics and File System. [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/)

For special promotional pricing and offers, please contact Keil Sales for details.

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

- ULINK2 - (ULINK2 and ME - SWV only – no ETM)
- **New** ULINKplus -
- ULINKpro - Cortex-Mx SWV & ETM trace.

MDK also supports ST-Link and Segger J-Link Debug adapters.



The Keil RTX RTOS is provided under a BSD or Apache 2.0 license. This makes it free.

All versions, including MDK-Lite, includes Keil RTX RTOS with source code !

[www.keil.com/demo/eval/rtx.htm](http://www.keil.com/demo/eval/rtx.htm)

Keil provides free DSP libraries for all 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 are available from stock.

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-M 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 the complete list of STMicroelectronics support. This information is also included in MDK.

For the entire Keil catalog see [www.keil.com](http://www.keil.com) or contact Keil or your local distributor.

For Linux, Android and bare metal (no OS) support on ST processors such as SPEAr, please see DS-5 [www.arm.com/ds5](http://www.arm.com/ds5).



#### For more information:

Keil Sales In USA: [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. Outside the US: [sales.intl@keil.com](mailto:sales.intl@keil.com)

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).

For the latest version of this document and for more STMicroelectronics specific information, go to [www.keil.com/st](http://www.keil.com/st)

CMSIS Version 5: [www.keil.com/cmsis](http://www.keil.com/cmsis)

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

