STMicroelectronics: Cortex™-M4 Training STM32F4 Discovery evaluation board using ARM® Keil™ MDK 5 toolkit

featuring Serial Wire Viewer Summer 2014 Version 1.2 Robert Boys, bob.boys@arm.com

ARM KEIL Microcontroller Tools

The latest version of this document is here:	www.keil.com/appnotes/docs/apnt_230.asp
For a CAN lab on the STM32F4 Discovery:	www.keil.com/appnotes/docs/apnt_236.asp

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. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. See www.keil.com/st.

Keil MDK supports and has examples for most ST ARM processors. Check the Keil Device Database® on www.keil.com/dd for the complete list which is also included in MDK: in µVision, select Project/Select Device for target…

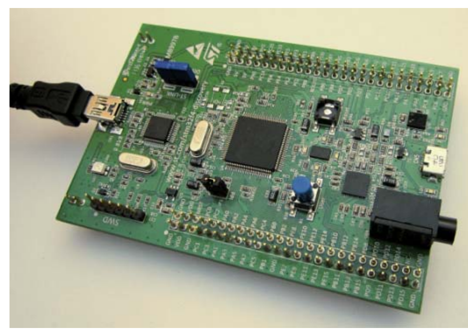
Linux: For ST processors running Linux, Android and bare metal are supported by ARM DS-5™. www.arm.com/ds5.

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

RTX RTOS: All variants of MDK contain the full version of RTX with source code. See www.keil.com/rl-arm/kernel.asp.

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is a turn-key product.
2. A full feature Keil RTOS called RTX is included with MDK. RTX comes with a BSD type license. Source code is provided.
3. Serial Wire Viewer and ETM trace capability is included.
4. RTX Kernel Awareness window. It is updated in real-time.
5. Keil Technical Support is included for one year and is easily renewable. This helps you get your project completed faster.



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. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
3. Six Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also known as Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS that updates while your program is running.
5. A DSP example program using ARM CMSIS-DSP libraries. www.arm.com/cmsis

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. This information comes from the ARM CoreSight™ debug module integrated into STM32 CPU. SWV does not steal any CPU cycles and is completely non-intrusive. (except for the ITM Debug printf Viewer).

CoreSight displays memory contents and variable values in real-time and these can be modified on-the-fly.

Embedded Trace Macrocell (ETM):

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?". Keil µVision uses ETM to provide Code Coverage, Performance Analysis and code execution times. ETM requires a special debug adapter such as the ULINK*pro*. The Discovery series do not have the ETM connector even though the processor has ETM. Most other ST and all Keil boards do have this connector.

Discovery Board Debug Adapter Connections:

The STM32F407 Discovery board lacks the standard ARM debugger connections. 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.

Notes on using this document:

1. The latest version of this document and the necessary example source files are available here: **www.keil.com/appnotes/docs/apnt_230.asp**
2. MDK 5.10 was used in the exercises in this document.
3. To use MDK 4.7x, see **www.keil.com/appnotes/docs/apnt_261.asp**
4. The on-board ST-Link V2 is used by default in this document. All you need install is the USB driver to your PC.
5. The ST-Link V2 interfaces very well with Keil µVision and its performance is quite good including SWV.

1) Keil Evaluation Software: MDK 4.7x and MDK 5

MDK 5 now uses Software Packs to distribute processor specific software, examples and middleware. MDK 5 is installed and the Packs you need will be downloaded from the web. They can also be imported manually.

MDK 4.7x is still available. MDK 4.7x contains all the files needed to run projects. The MDK 5 Software Packs are not used. The older version of this document written for MDK 4 is available here: www.keil.com/appnotes/docs/apnt_261.asp.

MDK 4 projects can be used with MDK 5 by using the Legacy install software available on www.keil.com. Then, MDK 5 can then run legacy MDK 4 projects without any Software Packs.

We recommend that MDK 5.10 Software packs are used if they are available. New ones are continually being added.

Keil has several labs for various STM32 processors including one using CAN. See www.keil.com/st for details.

This document uses only MDK 5.10 or later.

2) Keil Software Download and Installation:

Download MDK-Core Version 5

1. Download MDK 5.10 or later from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default directory. You can install into any directory, but this lab uses the default C:\Keil_v5
3. We recommend you use the default directories for this tutorial. We will use C:\MDK\ for the examples.
4. If you install MDK into a different directory, you will have to adjust for the directory differences.
6. The example DSP5 is available on the web where you got this document.
7. You can use the evaluation version (MDK-Lite) for this lab. No license is needed.
8. You do not need any debug adapters: just the Discovery board, a USB cable and MDK 5.10 installed on your PC.

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

The on-board ST-Link V2 is used exclusively in this lab. Instructions on configuring the ST-Link V2 are given. Page 5 contains a test for the ST-Link V2 drivers.

4) Example Programs:

MDK 5 Software Pack contains a Blinky and RTX_Blinky example programs. We will use only Blinky. An enhanced RTX_Blinky5 and the DSP example (DSP5) is available on the web where the latest version of this document is stored: www.keil.com/appnotes/docs/apnt_230.asp.

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

STM32F401C-DISCO Discovery Board:

STM32F4-Discovery: This tutorial is written for the STM32F4-Discovery board with a STM32F407VGT6 processor using a CPU speed of 168 MHz. The number MB997B or similar is marked on the board.

STM32F401C-DISCO: This newer board will work but the CPU clock speed must be reduced to 84 MHz. This board contains a STM32F401VCT6U processor. STM32F401C-DISCO is marked on the board.

Changing the clock speed:

In the file **system_stm32f4xx.c** near lines 254 and 256 are two #defines: #define PLL_M and #define PLL_Q.

Change these values according to the processor you are using as follows:

You will have to modify the Core Clock: value in the Trace Config window when using the Serial Wire Viewer. The DSP5 example runs at 57 MHz so no modification is needed.

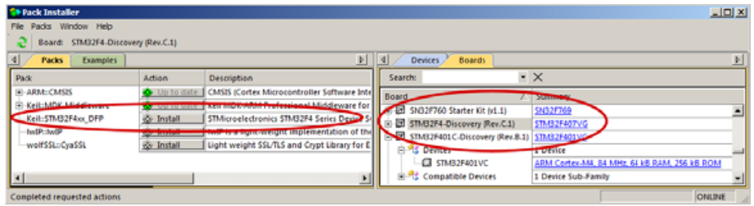
Note: The Software Pack for STM32F401C-DISCO contains Blinky and RTX_Blinky examples at 84 MHz.

	STM32F4-Discovery 168 MHz	**STM32F401C-DISCO 84 MHz**
#define PLL_M	8	336
#define PLL_Q	7	4

6) µVision Software Packs Download and Install Process:

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

1. Connect your computer to the internet. This is needed to download the Software Packs.
2. Start µVision by clicking on its desktop icon.
3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
4. This window opens up: Under the Boards tab, Select STM32F4-Discovery as shown below: This will filter the list under the Packs tab.
5. You can enter Discovery in Search to filter.
6. Note: "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet before continuing.
7. If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or to refresh once you have connected to the Internet.

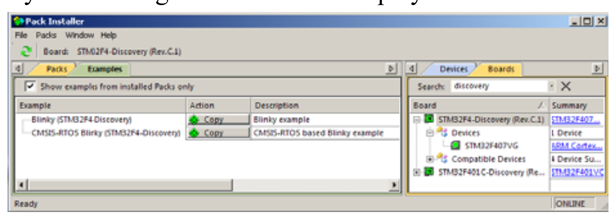


2) Install The STM32F4 Software Pack:

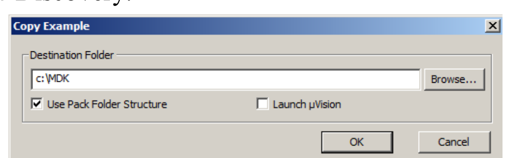
1. Click on the Packs tab. Initially, the Software Pack ARM::CMSIS is installed by default.
2. Select Keil::STM32F4xx_DFP and click on Install. This Software Pack will download and install to C:\Keil_v5\ARM\Pack\Keil\ST\ by default. This download can take two to four minutes.
3. Its status will be indicated by the "Up to date" icon: Up to date

TIP: If you click on the Devices tab, you can select the processor you are using and this will be displayed in the Packs tab.

3) Install the Blinky MDK 5.10 Example:



1. Select the Examples tab to display this window:
2. Select Blinky (STM32F4-Discovery):
3. Select Copy Copy as shown here:
4. The Copy Example window below opens up: Select Use Pack Folder Structure: Unselect Launch µVision:
5. Type in C:\MDK. Click OK to copy the Blinky project.
6. The Blinky example will now copy to C:\MDK\Boards\ST\ STMF32F4-Discovery.
7. We do not need to copy CMSIS-RTOS Blinky. We will use a special version that has 4 threads instead of two to make things interesting.

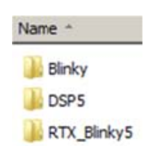


TIP: The default directory for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default directory of C:\MDK\ in this tutorial. You can use any directory you prefer.

8. Close the Packs Installer. You can open it any time by clicking on its icon.

4) Install the RTX_Blinky5 and DSP5 Examples from Keil.com:

1. Obtain the example software zip file from www.keil.com/appnotes/docs/apnt_230.asp.
2. Unzip this into the directory C:\MDK\Boards\ST\STMF32F4-Discovery\.
3. The DSP5 folder will be created with the Blinky and RTX_Blinky folder as shown here:



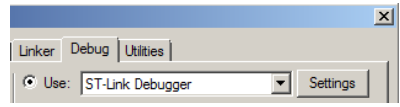
TIP: An Update icon means there is an updated Software Pack available for download. Update

TIP: If you look in the directory C:\Keil_v5\ARM\Pack\Keil\STM32F4xx_DFP\1.0.6\Boards\ST\STM32F4-Discovery, you will find another Blinky. This is a read-only version for backup purposes. Use only the projects you copied over from the Examples tab in the Pack Installer window to the directory you chose. In this tutorial we are using C:\MDK.

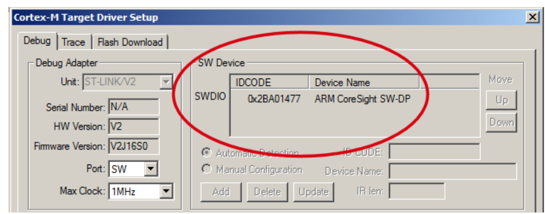
7) Testing the ST-Link V2 Connection:

1. Start µVision if it is not already running. Select Project/Open Project.
2. Connect the Discovery board to your PC with a USB cable as shown on the first page of this tutorial.
3. If the ST-Link USB drivers are installed correctly, you should hear the usual USB connected dual-tone. If not, you might have to install the drivers manually. See the directions below.
4. Two red LEDs will light:LD1 (COM) and LD2 (PWR)
5. Select the Blinky project C:\MDK\Boards\ST\STMF32F4-Discovery\Blinky.uvprojx.
6. Select STM32F407 Flash as shown here: 

7. Select Target Options or ALT-F7 and select the Debug tab:



8. Click on Settings: and the window below opens up: If an IDCODE and Device name is displayed, ST-Link is working. You can continue with the tutorial. Click on OK twice to return to the µVision main menu.
9. A number in the SN: box means µVision is connected to the ST-Link adapter.
10. If nothing or an error is displayed in this SW Device box, this ***must*** be corrected before you can continue. See the next step: Installing the ST-Link USB Drivers:
11. If you see a proper display, your ST-Link USB drivers are installed properly. Click OK twice to exit the Target Options windows and continue to the next page.



TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window.

TIP: The main difference between ST-Link and ST-Link V2 is the addition of Serial Wire Viewer (SWV) trace capability.

8) Installing ST-Link V2 USB Drivers: (might not be necessary if the test above passes)

Installing the ST-Link USB Drivers: (Only needed if the above test fails)

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 in the Discovery board to USB CN1. The USB drivers will now finish installing in the normal fashion.

Super TIP: The ST-Link V2 firmware updater utility ST-LinkUpgrade.exe is located here: C:\Keil_v5\ARM\STLink. If you want to update the ST-Link firmware, find this file and double click on it. It is easy to use. It will check and report the current firmware version. It is important you are using firmware V2.J16.S0 or later for proper SWV operation. Do not use V2.J19.S0. This version is incompatible to the latest drivers.

COM LED LD1 indication:

LED is blinking RED: the start of USB enumeration with the PC is taking place but not yet completed.
LED is RED: communication between the PC and ST-LINK/V2 is established (end of enumeration). µVision is not connected to ST-Link (i.e. in Debug mode).
LED is GREEN: µVision is connected in Debug mode and the last communication was successful.
LED is blinking GREEN/RED: data is actively being exchanged between the target and µVision.
LED is off, except for a brief RED flash while entering Debug mode and a brief flash when clicking on RUN happens when the SWV trace is enabled in µVision.

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

9) *Blinky* example program using the STM32F4 Discovery board:

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

1. Start µVision by clicking on its desktop icon. Connect your PC to the board with a USB cable to CN1.
2. Select Project/Open Project. Open the file C:\MDK\Boards\ST\STM32F4-Discovery\Blinky\Blinky.uvprojx
3. By default, the ST-Link is selected. If this is the first time you have run µVision and the Discovery board, you might have to install the USB drivers. See the configuration instructions on the previous page.
4. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
5. Program the STM32 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
6. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
   Note: You only need to use the Load icon to download to FLASH and not for RAM operation if it is chosen.
7. Click on the RUN icon. Note: you stop the program with the STOP icon.

The 4 LEDs on the* STM32F4 Discovery board *will now blink in succession.
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 !

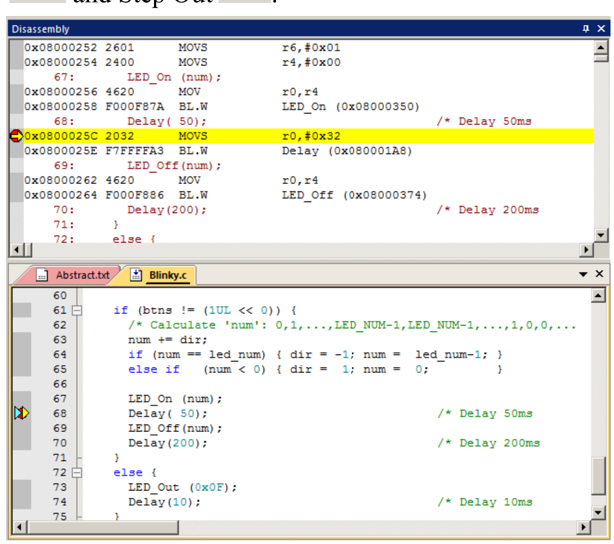
Note: The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

TIP: If you are using a STM32F401C-DISCO board and the LEDs do not blink, you might have the clock frequency set too high. See the instructions on page 3 under STM32F401C-DISCO Discovery Board:

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 59 through 74 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: If single step (Step In) doesn't work, click on the Disassembly window to bring it into focus. If needed, click on a disassembly line. This tells µVision you want to single step at the assembly level rather than at the C source level.

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

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

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

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

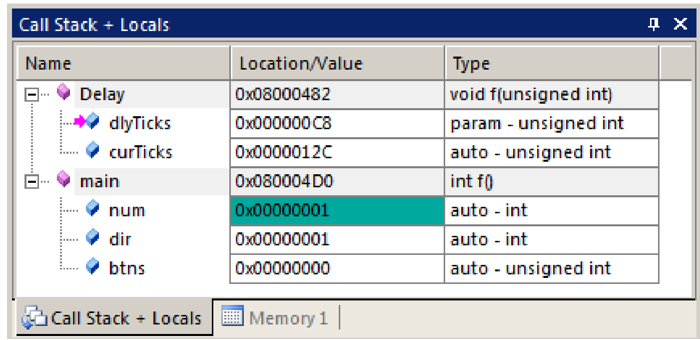
11) Call Stack + Locals Window:

Local Variables:

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

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

1. Run and Stop Blinky. Click on the Call Stack + Locals tab.
2. Shown is the Call Stack + Locals window.



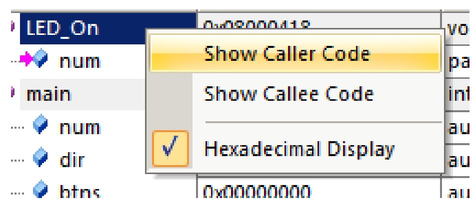
The contents of the local variables are displayed as well as names of active functions. Each function name will be displayed as it is called from the function before it or from an interrupt or exception.

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

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

This table is active only when the program is stopped.

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



TIP: If single step (Step In) does not work, click on the Disassembly window to bring it into focus. If needed, click on a disassembly line to step through assembly instructions. If a source window is in focus, you will step through the source lines instead.

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 ULINK*pro* 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 additionally provides Code Coverage, Performance Analysis and Execution Profiling.

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

Call Stack:

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

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

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

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

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.

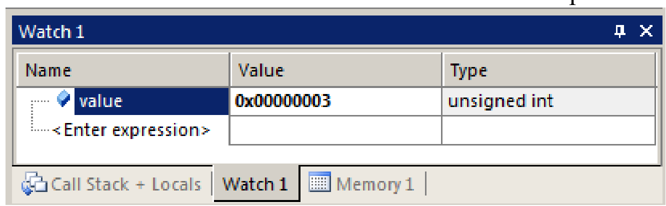
Watch window:

Add a global variable: Recall the Watch and Memory windows can't see local variables unless stopped in their function.

1. Stop the processor and exit Debug mode.
2. In Blinky.c, declare a global variable (I called it value) near line 20 like this: **unsigned int value = 0;**
3. Add the statements `value++;` and `if (value > 0x10) value = 0;` as shown here near line 71:

```
69        LED_Off(num);
70        Delay(200);
71        value++;
72        if (value > 0x10) value = 0;
73      }
74      else {
```

4. Select File/Save All or click .
5. Click on Rebuild. . Click on Load to program the Flash.
6. Enter Debug mode. Click on RUN . Recall you can set Watch and Memory windows while the program is running.
7. In Blinky.c, right click on the variable **`value`** and select Add value to … and select Watch 1. Watch 1 will open if needed and **`value`** will be displayed as shown here:
8. **`value`** will increment until 0x10 in real-time.



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

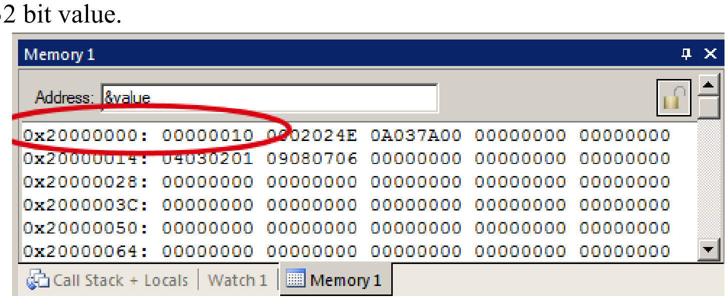
TIP: Make sure View/Periodic Window Update is selected.

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. Use the View/Symbols window to enter a variable fully qualified.

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 **`value`** and enter into the Memory 1 window or enter it manually. Select View/Memory Windows if necessary.
2. Note the value of **`value`** 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_0014).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **`value`** is now displayed as a 32 bit value.
6. Both the Watch and Memory windows are updated in real-time.
7. You can modify value in the Memory window with a right-click with the mouse cursor over the data field and select Modify Memory.



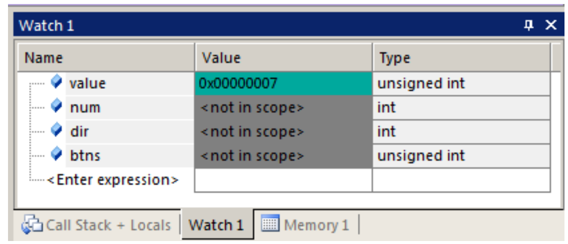
TIP: No CPU cycles are normally used to perform these operations. See the next page "How It Works" for an explanation on how DAP functions.

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

Serial Wire Viewer (SWV) 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.

13) How to view Local Variables in the Watch or Memory windows:

1. Make sure Blinky.c is running. We will use the local variables from main(): num, dir and btns.
2. Locate where the three local variables are declared in Blinky.c near line 46, at the start of the main function.
3. Enter each of the three variables into Watch 1 window by right-clicking on them. Note it says < not in scope > because µVision cannot access the CPU registers while running which is where value is probably located. If µVision says you are unable to add a variable, stop and start the Blinky program.
4. Set a breakpoint in the Blinky.c while loop. The problem will stop the program and the current variable values will appear.
5. Remove this breakpoint.
6. Set a breakpoint at `if (btns != (1UL << 0))` near line 61.
7. Hold down the blue USER button and start the program. The program will stop. A btns value of 1 will display. Without USER pressed, a 0 will be displayed if you click on Run again.



TIP: Remember: you can set and unset hardware breakpoints on-the-fly in the Cortex-M4 while the program is running !

8. µVision is unable to determine the value of these three variables when the program is running because they exist only when main is running. They disappear in functions and handlers outside of main. They are a local or automatic variable and this means it is probably stored in a CPU register which µVision is unable to access during run time.
9. ***Remove the breakpoint*** and make sure the program is not running. Exit Debug mode.

How to view local variables updated in real-time:

All you need to do is to make the local variables num, dir and btns global where it is declared in Blinky.c !

1. Move the declarations for num, dir and btns out of main() and to the top of Blinky.c to make them global variables:

```
unsigned int value = 0;
int32_t num = -1;
int32_t dir =  1;
uint32_t btns = 0;
```

TIP: You could also make the them static ! i.e. static `int32_t num = -1;`

TIP: You can edit files in edit or debug mode. However, you can compile them only in edit mode.

2. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
3. To program the Flash, click on the Load icon. A progress bar will be displayed at the bottom left.
4. Enter Debug mode. Click on RUN.
5. Now the three variables are updated in real-time. Press and release the User button and btns will update to 0 or 1. This is ARM CoreSight technology working.
6. You can read (and write) global, static variables and structures. Anything that stays around in a variable from function to function. This includes reads and writes to peripherals.
7. Stop the CPU and exit debug mode for the next step. and

TIP: View/Periodic Window Update must be selected. Otherwise variables update only when the program is stopped.

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.

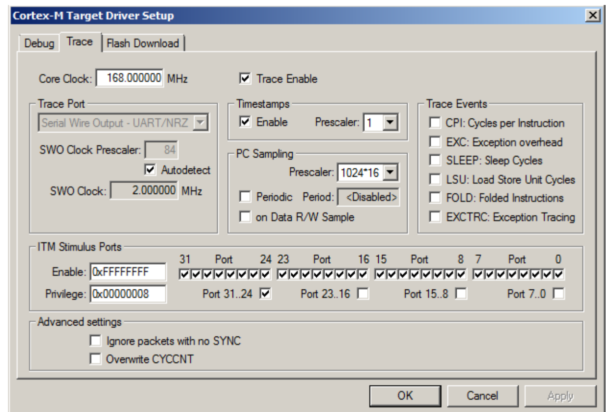
14) View Variables Graphically with the Logic Analyzer (LA):

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

1. Stop the processor and exit Debug mode.

Configure Serial Wire Viewer (SWV):

2. Select Target Options or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window. Confirm SW is selected. SW selection is mandatory for SWV. ST-Link uses only SW. Select the Trace tab.
3. In the Trace tab, select Trace Enable. Unselect Periodic and EXCTRC. Set Core Clock: to 168 MHz. Everything else is set as shown here:
4. Click OK once to return to the Debug tab.
5. Click OK return to the main menu. Enter debug mode.

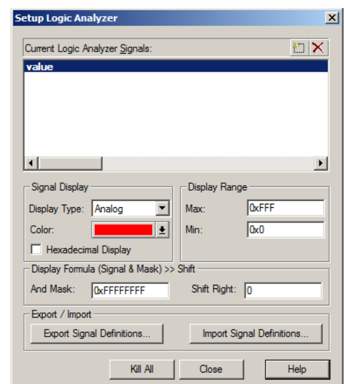


Configure Logic Analyzer:

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

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

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



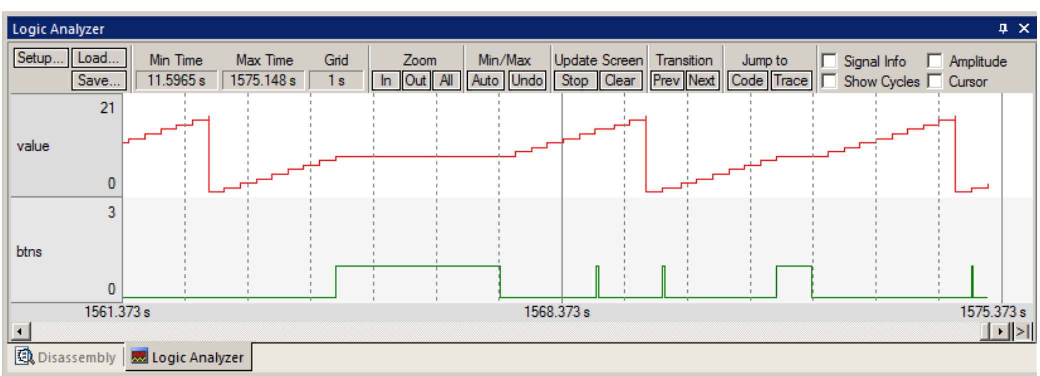
Run Program: Note: The LA can be configured while the program is running.

1) Click on Run. Click on Zoom Out until Grid is about 1 second.
2) The variable `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 show up to 4 variables in the Logic Analyzer. These variables must be global, static or raw addresses such as *((unsigned long *)0x20000000).

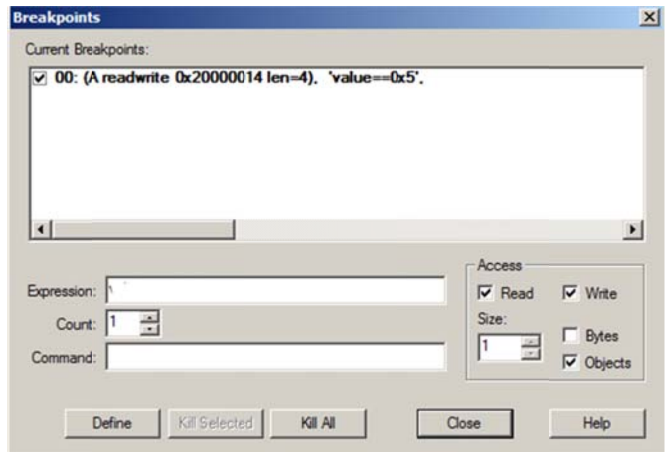
3) Enter the static variable btns into the LA and set the Display Range Max: to 0x2. Click on RUN and press the User button and see the voltages below:
4) Note the variable value stops incrementing while USER is held down. Also note how easy it is to view this effect.
5) Select Signal Info, Show Cycles, 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.
6) Stop the CPU.
7) Click on Setup in the LA and delete btns. You can use the Delete key or this icon:
8) Click on Close.

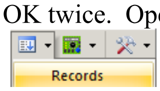


15) **Watchpoints:** *Conditional Breakpoints: This does not need or use Serial Wire Viewer:*

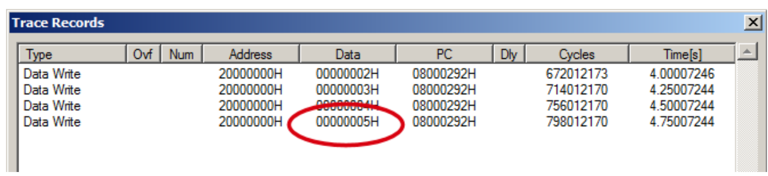
Recall STM32 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. The STM32 also have four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same comparators as Watchpoints in its operations and they must be shared. This means in µVision 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 **`value`** you created in Blinky.c to explore Watchpoints.
3. The SWV Trace does not need to be configured for Watchpoints. However, we will use it in this exercise.
4. The variable **`value`** should be still entered in the Logic Analyzer from the last exercise on the previous page.
5. Select Debug in the main µVision window and select Breakpoints or press Ctrl-B.
6. Select both the Read and Write Access. In the Expression box enter: "**`value`** = = 0x5" without the quotes.
7. Click on Define and it will be accepted as shown here: Click on Close.



8. Enter the **`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. Click on RUN.
12. You will see **`value`** change in the Logic Analyzer as well as in the Watch window.
13. When **`value`** equals 0x5, the Watchpoint will stop the program.
14. 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* or a J-Link (black case) will show a slightly different display.


Trace Records

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

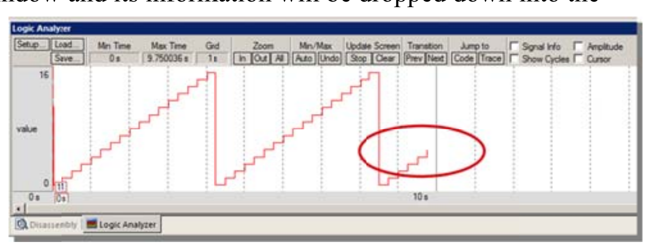
15. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window. Not all are currently implemented in µVision.
16. To repeat this exercise, click on RUN.
17. When you are finished, stop the program, click on Debug and select Breakpoints (or Ctrl-B) and Kill the Watchpoint.
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 **`value`** trigger point of 0x5. This is three runs.

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

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

NOTE: RTX_Blinky supplied with MDK 5 is a two task project that blinks one LED. Supplied with this document is an RTX_Blinky that has fours tasks and lights four LEDs. It is more interesting.

RTX and all its components are located here: C:\Keil_v5\ARM\Pack\ARM\CMSIS\3.20.4\CMSIS_RTX.

You must have copied RTX_Blinky5 to C:\MDK\Boards\ST\STM32F4-Discovery\ as described on page 4.

1. With µVision in Edit mode (not in debug mode): Select Project/Open Project.
2. Open the file C:\MDK\Boards\ST\STM32F4-Discovery\RTX_Blinky5\Blinky.uvprojx.
3. **If the Update Configuration Files window opens, select Cancel.**
4. This project is pre-configured for the ST-Link V2 debug adapter.
5. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
6. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
7. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
8. The four LEDs will blink in succession simulating the signals for a stepper motor.

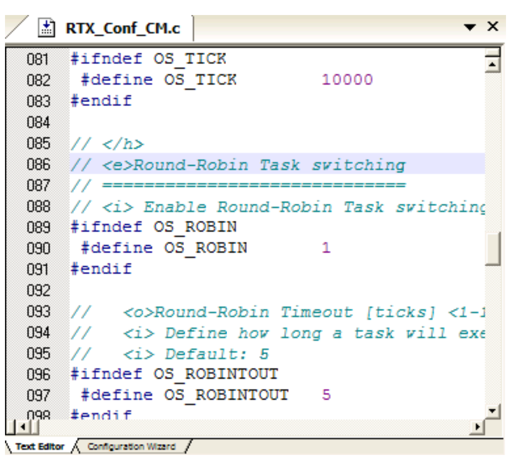
TIP: If you are using a STM32F401C-DISCO board and the LEDs do not blink, you might have the clock frequency set too high. See the instructions on page 3 under STM32F401C-DISCO Discovery Board:.

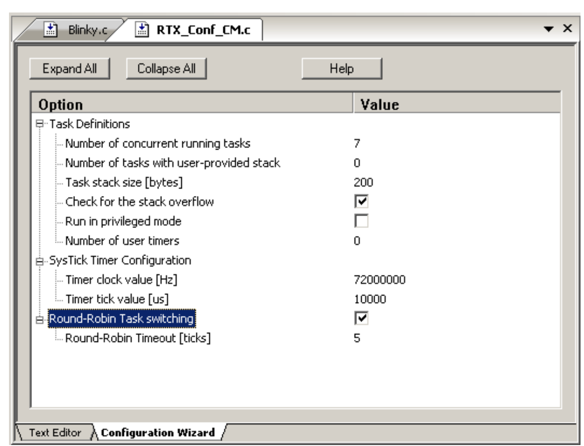
9. Click on STOP .

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

The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open if needed.
2. Click on the Configuration Wizard tab at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. Changing an attribute in one tab changes it in the other automatically. You should save a modified window.
6. You can create Configuration Wizards 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 for instructions.

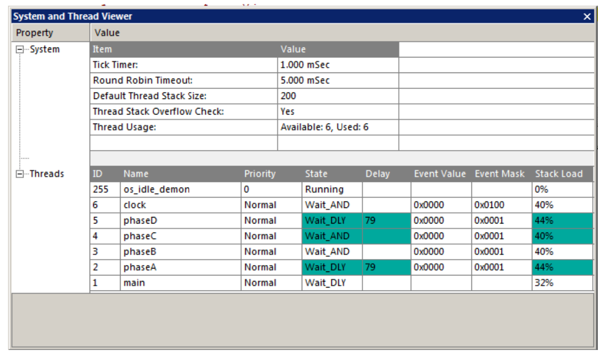




Text Editor: Source Code **Configuration Wizard**

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

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



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

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

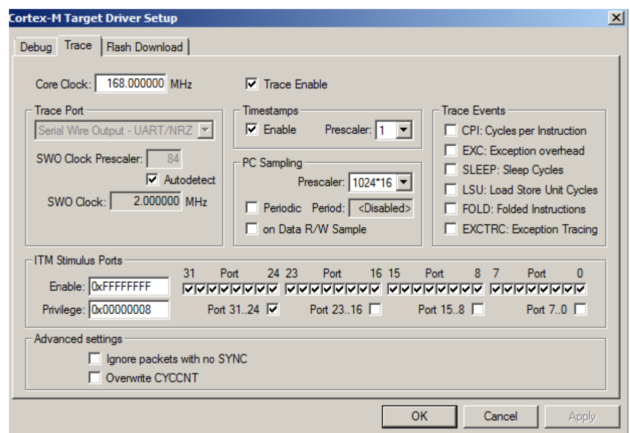
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.

RTX Viewer: Configuring Serial Wire Viewer (SWV):

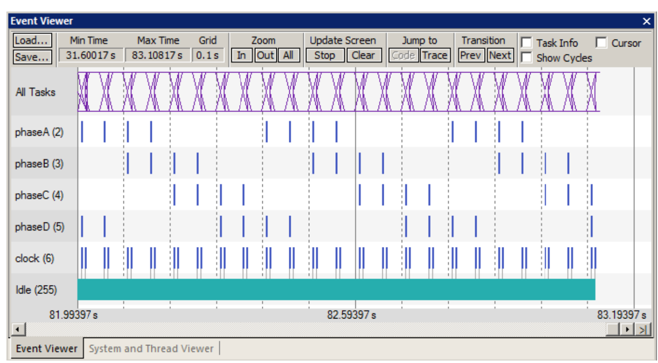
We must activate Serial Wire Viewer to get the Event Viewer working.

1. Stop the CPU and exit debug mode.
2. Click on the Target Options icon next to the target box. Select the Debug tab.
3. Click the Settings box next to ST-Link Debugger.
4. In the Debug window, make sure Port: is set to SW and not JTAG. SWV works only with SW mode.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 168 MHz (84 for STM32F401 board) and select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown:
8. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer. It is slightly intrusive.
9. Click on OK twice to return to µVision.

   The Serial Wire Viewer is now configured !



10. Enter Debug mode and click on RUN.
11. Select "Tasks and System" tab: the display is updated.
12. Click on the Event Viewer tab.
13. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 0.1 sec by clicking on the Zoom ALL and + and – icons.



TIP: If Event Viewer doesn't work, open up the Trace Records and confirm there are good ITM 31 frames present. Is Core Clock correct ? This project is running at 168 or 84 MHz depending on the board you are using.

Cortex-M3 Alert: µVision will update all RTX information in real-time on a target board due to its read/write capabilities as already described. The Event Viewer uses ITM and is slightly intrusive.

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

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

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

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

1. Close the RTX Viewer windows. Stop the program and exit debug mode.
2. Add 4 global variables **`unsigned int phasea`** through **`unsigned int phased`** to Blinky.c as shown here:

```
25  #define LED_CLK    7
26
27  unsigned int phasea=0;
28  unsigned int phaseb=0;
29  unsigned int phasec=0;
30  unsigned int phased=0;
31  /*---------------------
32   *          Function 'sig
```

3. Add 2 lines to each of the four threads phaseA through phaseD in Blinky.c as shown below: **phasea=1;** and **phasea=0;**. The first two lines are shown added near lines 50 and 53 (just after the LED_On and LED_Off function calls). For each of the three remaining threads, add the corresponding statements phaseb, phasec and phased.
4. Select File/Save All or click .
5. Rebuild the project. Program the Flash .
6. Enter debug mode .
7. You can run the program at this point.
8. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.

```
43  /*-----------------------------
44   *        Thread 1 'phaseA': Ph
45   *-----------------------------
46  void phaseA (void const *argume
47    for (;;) {
48      osSignalWait(0x0001, osWait
49      LED_On (LED_A);
50      phasea = 1;
51      signal_func(tid_phaseB);
52      LED_Off(LED_A);
53      phasea = 0;
54
```

Enter the Variables into the Logic Analyzer (LA):

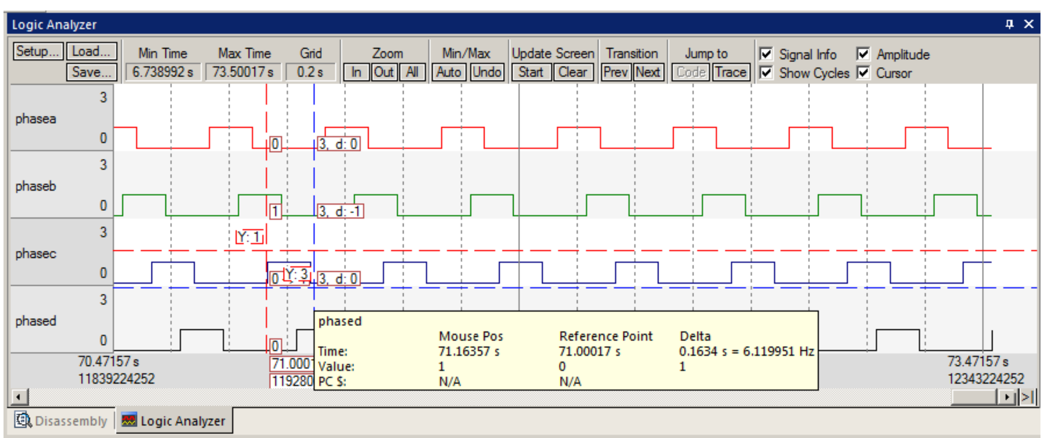
9. Click on the Blinky.c tab. Right click on **phasea**, select Add 'phasea' to… and finally select Logic Analyzer. Phasea will be added to the LA.
10. Repeat for **phaseb, phasec and phased.** These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.

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

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

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

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



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

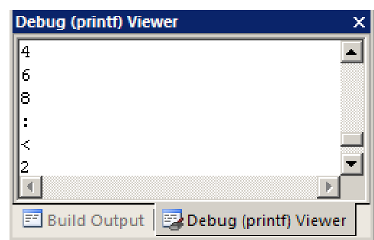
TIP: You can also enter these variables into the Watch and Memory windows to display and change them in real-time.

TIP: You can view signals that exist mathematically in a variable and not available for measuring in the outside world.

19) ITM (Instrumentation Trace Macrocell) ITM uses Serial Wire Viewer.

Recall that we showed you can display information about the RTOS in real-time using the RTX Viewer. This is done through ITM Stimulus Port 31. ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in its Debug (printf) Viewer window. Note: the global variable **`value`** from **12) Watch and Memory Windows …** must be entered and compiled in Blinky.c in order for this exercise to work.

1. Stop the program if it is running and exit Debug mode.
2. Open the project C:\MDK\Boards\ST\STM32F4-Discovery\Blinky\Blinky.uvprojx. (do not use RTX_Blinky).
3. Add this code to Blinky.c. A good place is near line 19, just after the #include "LED.h".
```
#define ITM_Port8(n)   (*((volatile unsigned char *)(0xE0000000+4*n)))
```
4. In the main function in Blinky.c after the second Delay(200); near line 72, enter these lines:
```
ITM_Port8(0) = value + 0x30;     /* displays value in ASCII */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```
5. Rebuild the source files, program the Flash memory and enter debug mode.
6. Open Select Target Options or ALT-F7 and select the Debug tab, and then the Trace tab.
7. Configure the Serial Wire Viewer as described on page 10. Use 168 MHz for the Core Clock.
8. Unselect On Data R/W Sample, EXCTRC and PC Sample. (this is to help not overload the SWO port)
9. Select ITM Port 0. ITM Stimulus Port "0" enables the Debug (prinftf) Viewer.
10. Click OK twice. Enter Debug mode.
11. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
12. In the Debug (printf) Viewer you will see the ASCII of value appear.
13. As value is incremented its ASCII character is displayed.



Trace Records

1. Open the Trace Records if not already open. Double click on it to clear it.
2. You will see a window such as the one below with ITM and Exception frames.

What Is This ?

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

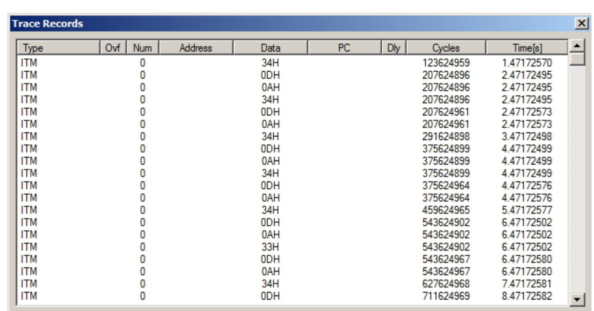
1. ITM 0 frames (Num column) are our ASCII characters from **`value`** with carriage return (0D) and line feed (0A) as displayed the Data column.
2. All these are timestamped in both CPU cycles and time in seconds.
3. When you are done, stop the processor and exit debug mode.

ITM Conclusion

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

TIP: It is important to select as few options in the Trace configuration as possible to avoid overloading the SWO pin. Enter only those features that you really need.

Super TIP: ITM_SendChar is a useful function you can use to send ITM characters. It is found in the header core.CM3.h.


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		0		34H			123624959	1.47172570
ITM		0		0DH			207624896	2.47172495
ITM		0		0AH			207624896	2.47172495
ITM		0		34H			207624896	2.47172495
ITM		0		0DH			207624961	2.47172573
ITM		0		0AH			207624961	2.47172573
ITM		0		34H			291624898	3.47172498
ITM		0		0DH			375624899	4.47172499
ITM		0		0AH			375624899	4.47172499
ITM		0		34H			375624899	4.47172499
ITM		0		0DH			375624964	4.47172576
ITM		0		0AH			375624964	4.47172576
ITM		0		34H			459624965	5.47172577
ITM		0		0DH			543624902	6.47172502
ITM		0		0AH			543624902	6.47172502
ITM		0		33H			543624902	6.47172502
ITM		0		0DH			543624967	6.47172580
ITM		0		0AH			543624967	6.47172580
ITM		0		34H			627624968	7.47172581
ITM		0		0DH			711624969	8.47172582

Super TIP: To see how to use printf in your code, see the DSP example.

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

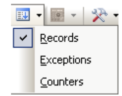
1) Data Reads and Writes: (Note: Data Writes but not Reads are enabled in the current version of µVision).

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

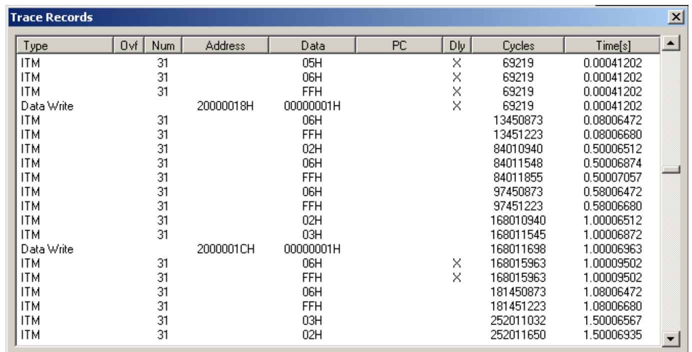
Now we will examine some of the features available to you. SWV works with µVision and ST-Link V2, ULINK2/ME, ULINK*pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

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

1. Use RTX_Blinky5 from the previous exercise. Enter Debug mode and Run the program if not already running.
2. Select View/Trace/Records or click on the Trace icon  and select Records.



3. The Trace Records window will open up as shown here:
4. The ITM frames are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. here:
5. To turn this off, select Debug/Debug Settings and click on the Trace tab. Unselect ITM Stimulus Port 31.
   TIP: Port 0 is used for Debug `printf` Viewer.
6. Unselect EXCTRC and Periodic.
7. Select On Data R/W Sample.
8. Click on OK twice to return.



Trace Records

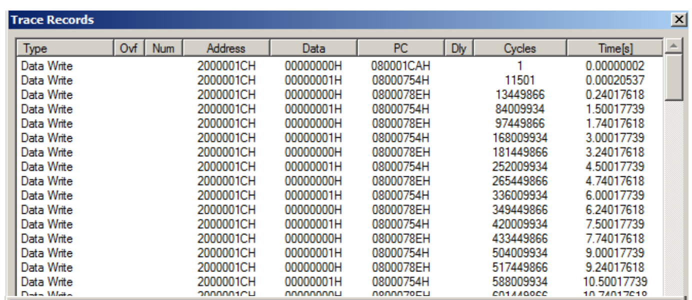
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		05H		X	69219	0.00041202
ITM		31		06H		X	69219	0.00041202
ITM		31		FFH		X	69219	0.00041202
Data Write			20000018H	00000001H		X	69219	0.00041202
ITM		31		06H			13450873	0.08006472
ITM		31		FFH			13451223	0.08006680
ITM		31		02H			84010940	0.50006512
ITM		31		06H			84011548	0.50006874
ITM		31		FFH			84011855	0.50007057
ITM		31		06H			97450873	0.58006472
ITM		31		FFH			97451223	0.58006680
ITM		31		02H			168010940	1.00006512
ITM		31		03H			168011545	1.00006872
Data Write			2000001CH	00000001H			168011698	1.00006963
ITM		31		06H		X	168015963	1.00009502
ITM		31		FFH		X	168015963	1.00009502
ITM		31		06H			181450873	1.08006472
ITM		31		FFH			181451223	1.08006680
ITM		31		03H			252011032	1.50006567
ITM		31		02H			252011650	1.50006935

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

If you are using the STM32F401 board see page 3.

9. Click on the RUN icon.
10. Double-click in Trace Records window to clear it.
11. Only Data Writes will appear now.

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

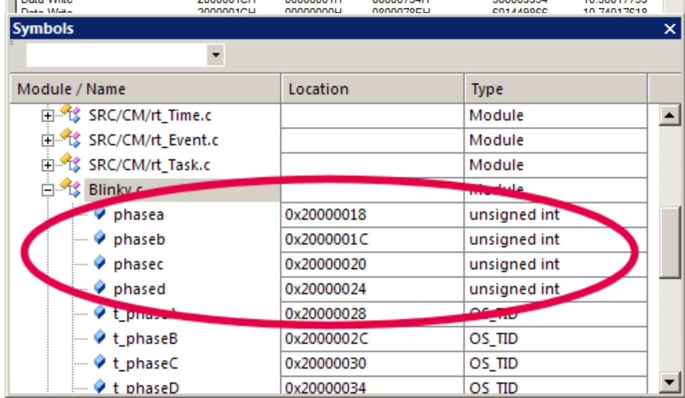


Trace Records

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

What is happening here ?

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



Symbols

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

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

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

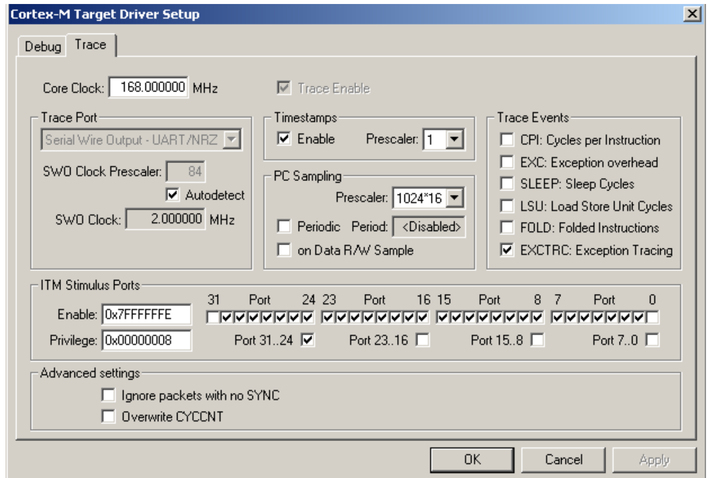
Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

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

2) Exceptions and Interrupts:

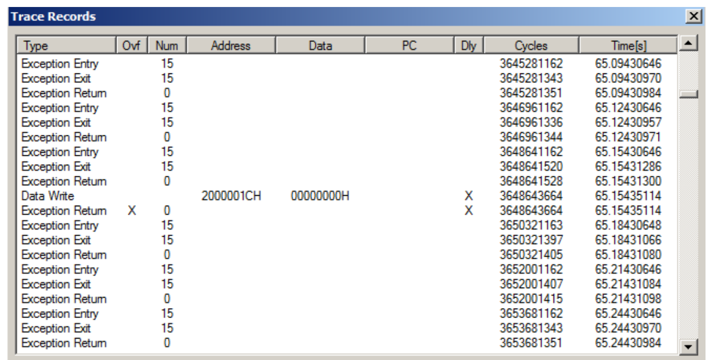
The STM32 family using the Cortex-M4 processor has many interrupts and it can be difficult to determine when they are being activated and how often. Serial Wire Viewer (SWV) on the STM32 family makes this task easy.

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



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

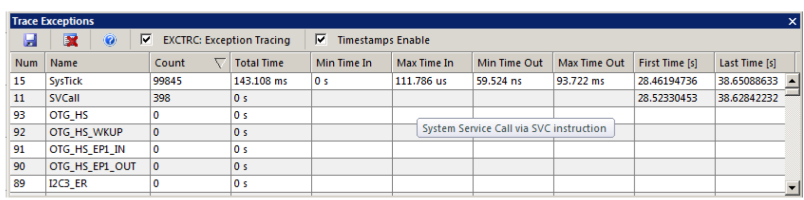
7. You will see a window similar to the one below with Exceptions frames displayed.



.What Is Happening ?

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

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



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

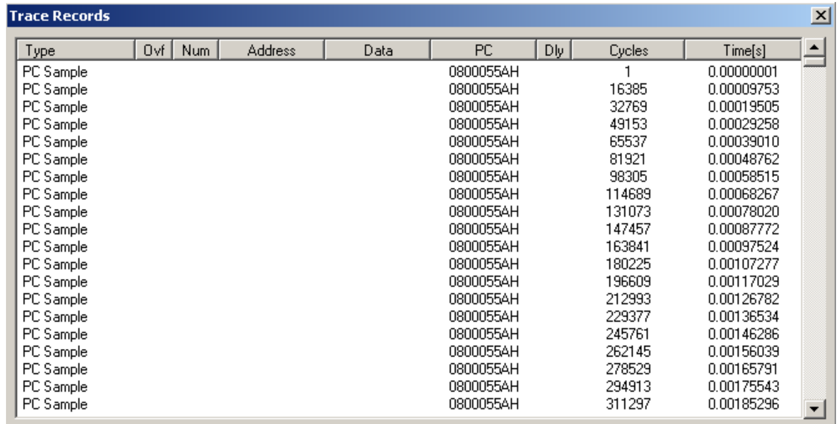
TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

3) PC Samples:

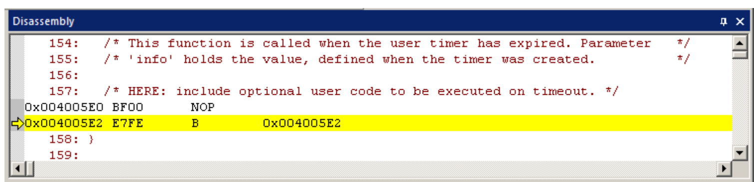
Serial Wire Viewer can display a sampling of the program counter.

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

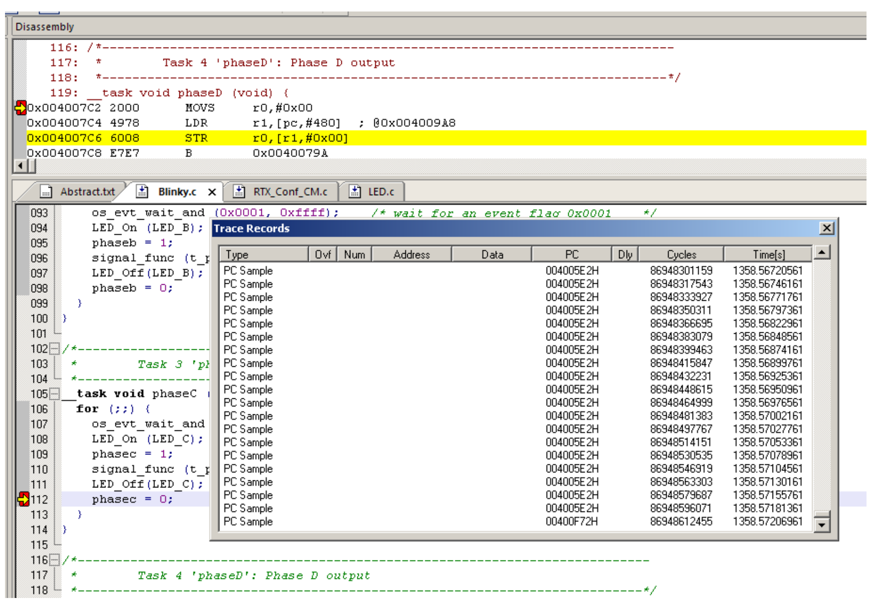
1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear.
5. Click on RUN and this window opens:
6. Most of the PC Samples in the example shown are 0x0800_055A which is a branch to itself in a loop forever routine.
   Note: the exact address you get depends on the source code and the compiler settings.



7. Stop the program and the Disassembly window will show this Branch as shown below:
8. Not all the PCs will be captured. Still, PC Samples can give you some idea of where your program is; especially if it is not caught in a tight loop like in this case.
9. Set a breakpoint in one of the tasks.
10. Run the program and when the breakpoint is hit, you might see another address at the bottom of the Trace Records window. See the screen below:



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

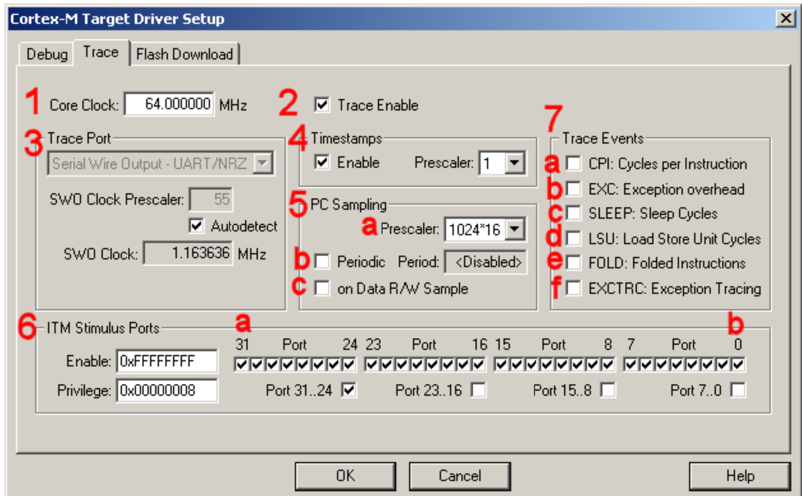


21) Serial Wire Viewer (SWV) Configuration window: (for reference)

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

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

B. **In Debug mode:** Select Debug/Debug Settings and then select the Trace tab. Debug mode is selected with .



1) **Core Clock:** The CPU clock speed for SWV. The CPU speed can be found in your startup code or in Abstract.txt. It is usually called SYSCLK or Main Clock. This ***must*** be set correctly for all adapters except ULINK*pro*.
2) **Trace Enable:** Enables SWV and ITM. It can only be changed in Edit mode. This does not affect the Watch and Memory window display updates.
3) **Trace Port:** This is preset for ST-Link**.**
4) **Timestamps:** Enables timestamps and selects the Prescaler. 1 is the default.
5) **PC Sampling:** Samples the program counter:
    - **a. Prescaler** 1024*16 (the default) means every 16,384th PC is displayed. The rest are not collected.
    - **b. Periodic:** Enables PC Sampling.
    - **c. On Data R/W Sample:** Displays the address of the instruction that caused a data read or write of a variable listed in the Logic Analyzer. This is not connected with PC Sampling but rather with data tracing.
6) **ITM Stimulus Ports**: Enables the thirty-two 32 bit registers used to output data in a *printf* type statement to µVision. Port 31 **(a)** is used for the Keil RTX Viewer which is a real-time kernel awareness window. Port 0 **(b)** is used for the Debug (printf) Viewer. The rest are currently unused in µVision.
    - **Enable:** Displays a 32 bit hex number indicating which ports are enabled.
    - **Privilege:** Privilege is used by an RTOS to specify which ITM ports can be used by a user program.
7) **Trace Events:** Enables various CPU counters. All except EXCTRC are 8 bit counters. Each counter is cumulative and an event is created when this counter overflows every 256 cycles. These values are displayed in the Counter window. The event created when a counter wraps around is displayed in the Instruction Trace window.
    - a. **CPI: Cycles per Instruction:** The cumulative number of extra cycles used by each instruction beyond the first, one including any instruction fetch stalls.
    - b. **Fold:** Cumulative number of folded instructions**.** These results from a predicted branch instruction where unused instructions are removed (flushed) from the pipeline giving a zero cycle execution time.
    - c. **Sleep:** Cumulative number of cycles the CPU is in sleep mode. Uses FCLK for timing.
    - d. **EXC:** Cumulative cycles CPU spent in exception overhead not including total time spent processing the exception code. Includes stack operations and returns.
    - e. **LSU:** Cumulative number of cycles spent in load/store operations beyond the first cycle.
    - f. **EXCTRC:** Exception Trace. This is different than the other items in this section. This enables the display of exceptions in the Instruction Trace and Exception windows. It is not a counter. This is a very useful feature to display exception events and is often used in debugging.

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

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

ARM CMSIS-DSP libraries are offered for ARM Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS. See www.arm.com/cmsis for more information. CMSIS is an acronym for Cortex Microcontroller Software Interface Standard. CMSIS is an ARM standard.

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

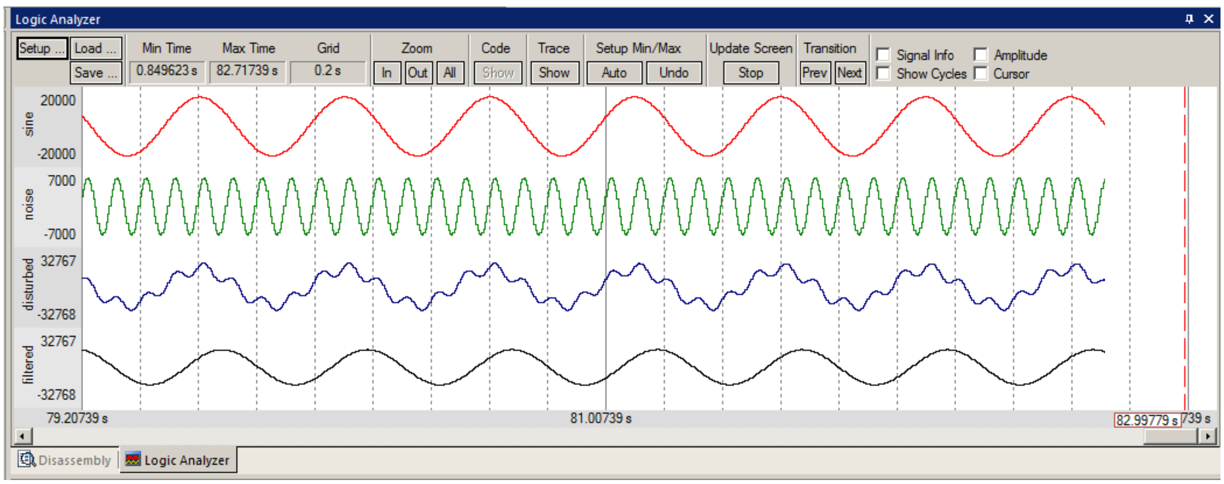
This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. RTX source code is provided.

To obtain this example file, go to **www.keil.com/appnotes/docs/apnt_230.asp** Extract DSP to …\STM32F4-Discovery\.

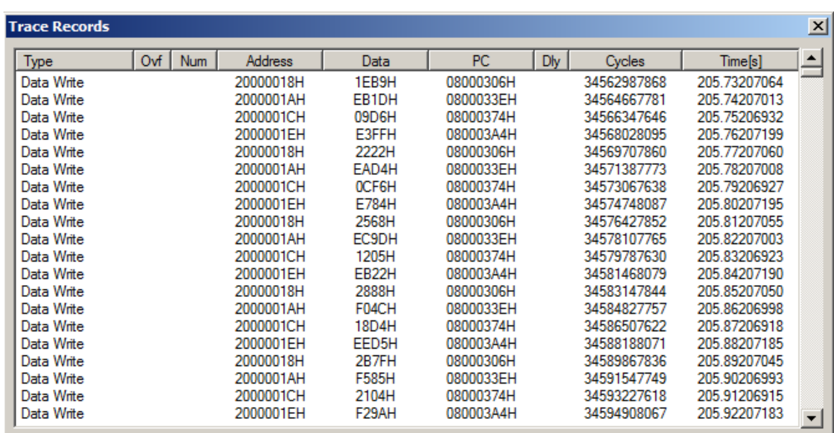
1. Open the project file sine: C:\Keil\ARM\Boards\ST\STM32F4-Discovery\DSP\sine.uvproj
2. Build the files. There will be no errors or warnings.
3. Program the STM32 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
4. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
5. Click on the RUN icon. Open the Logic Analyzer window.
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. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables.



8. Open the Trace Records window and the Data Writes to the four variables are listed as shown here:
9. You can right click in this window and deselect ITM Events to get only Data Writes displayed.
10. Leave the program running.
11. Close the Trace Records window.

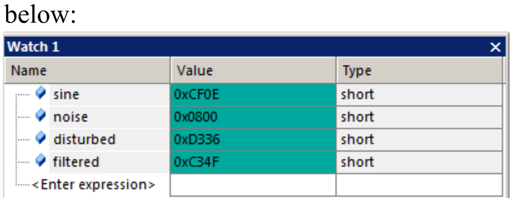


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000018H	1EB9H	08000306H		34562987868	205.73207064
Data Write			2000001AH	EB1DH	0800033EH		34564667781	205.74207013
Data Write			2000001CH	09D6H	08000374H		34566347646	205.75206932
Data Write			2000001EH	E3FFH	080003A4H		34568028095	205.76207199
Data Write			20000018H	2222H	08000306H		34569707860	205.77207060
Data Write			2000001AH	EAD4H	0800033EH		34571387773	205.78207008
Data Write			2000001CH	0CF6H	08000374H		34573067638	205.79206927
Data Write			2000001EH	E784H	080003A4H		34574748087	205.80207195
Data Write			20000018H	2568H	08000306H		34576427852	205.81207055
Data Write			2000001AH	EC9DH	0800033EH		34578107765	205.82207003
Data Write			2000001CH	1205H	08000374H		34579787630	205.83206923
Data Write			2000001EH	EB22H	080003A4H		34581468079	205.84207190
Data Write			20000018H	2888H	08000306H		34583147844	205.85207050
Data Write			2000001AH	F04CH	0800033EH		34584827757	205.86206998
Data Write			2000001CH	18D4H	08000374H		34586507622	205.87206918
Data Write			2000001EH	EED5H	080003A4H		34588188071	205.88207185
Data Write			20000018H	2B7FH	08000306H		34589867836	205.89207045
Data Write			2000001AH	F585H	0800033EH		34591547749	205.90206993
Data Write			2000001CH	2104H	08000374H		34593227618	205.91206915
Data Write			2000001EH	F29AH	080003A4H		34594908067	205.92207183

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

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

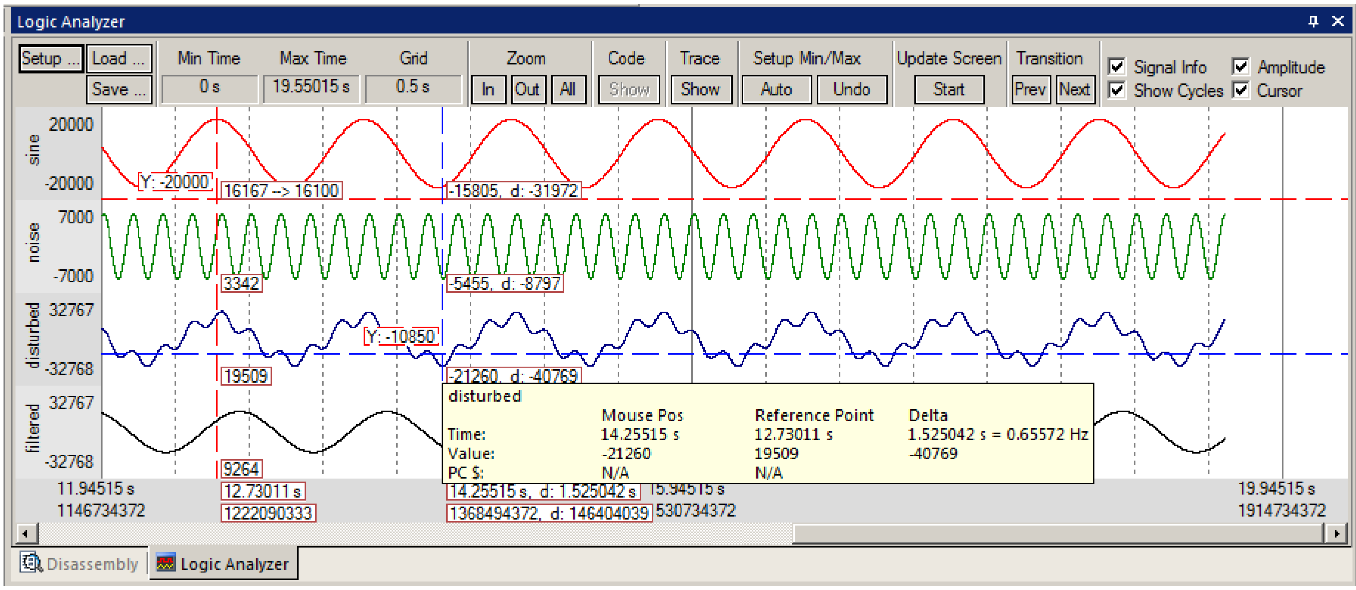


Watch 1

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

Signal Timings in Logic Analyzer (LA):

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



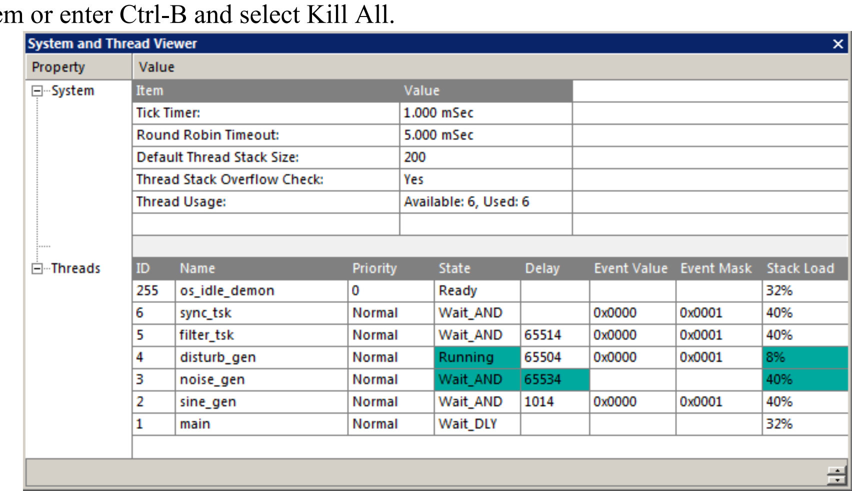
RTX Tasks and System:

5. Click on Start in the Update Screen box to resume the collection of data.
6. Open Debug/OS Support and select System and Thread Viewer. A window similar to below opens up. You probably have to click on its header and drag it into the middle of the screen.
7. Note this window does not update: nearly all the processor time is spent in the idle daemon: it shows it is Running. The processor spends relatively little time in the other threads. You will see this illustrated clearly on the next page.
8. Set a breakpoint in four of the threads in DirtyFilter.c by clicking in the left margin on a grey area.
9. Click on Run and the program will stop at each thread in turn and the Thread Viewer window will be updated accordingly. Here, I set a breakpoint in the disturb_gen thread:
10. Clearly you can see that disturb_gen was running when the breakpoint was activated.
11. Remove the breakpoints. Click on them or enter Ctrl-B and select Kill All.

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

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

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



System and Thread Viewer

Property | Value

System

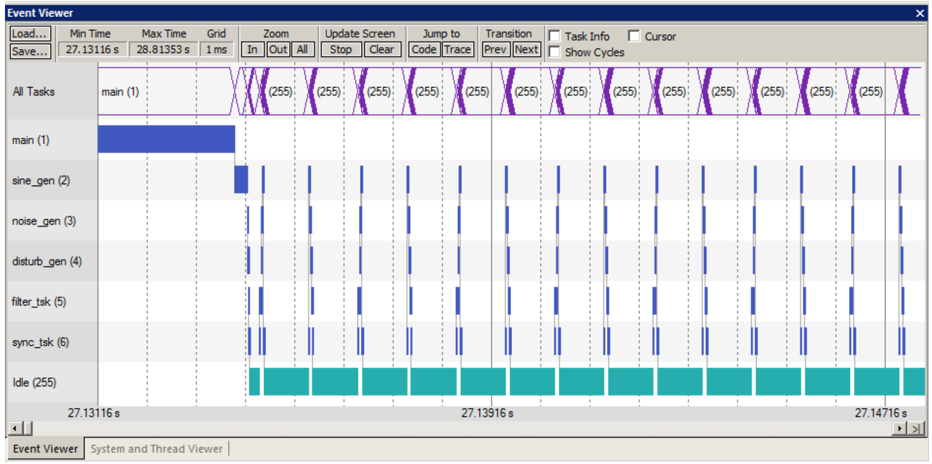
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 6, Used: 6

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Ready				32%
6	sync_tsk	Normal	Wait_AND		0x0000	0x0001	40%
5	filter_tsk	Normal	Wait_AND	65514	0x0000	0x0001	40%
4	disturb_gen	Normal	Running	65504	0x0000	0x0001	8%
3	noise_gen	Normal	Wait_AND	65534			40%
2	sine_gen	Normal	Wait_AND	1014	0x0000	0x0001	40%
1	main	Normal	Wait_DLY				32%

Event Viewer:

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



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

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

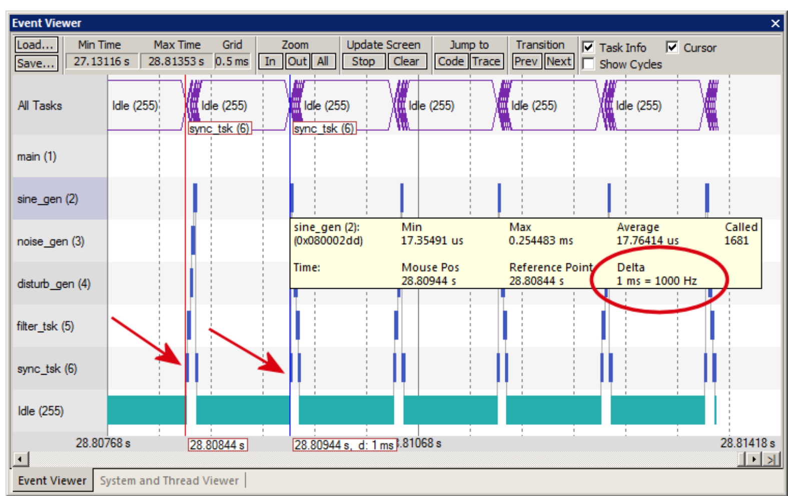
ULINK*pro* is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows. ULINK*pro* uses the faster Manchester format than the slower UART mode that ST-Link, ULINK2 and J-Link uses.

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

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

The 10 msec shown is the SysTick timer value. This value is set in RTX_Conf_CM.c . The next page describes how to change this.

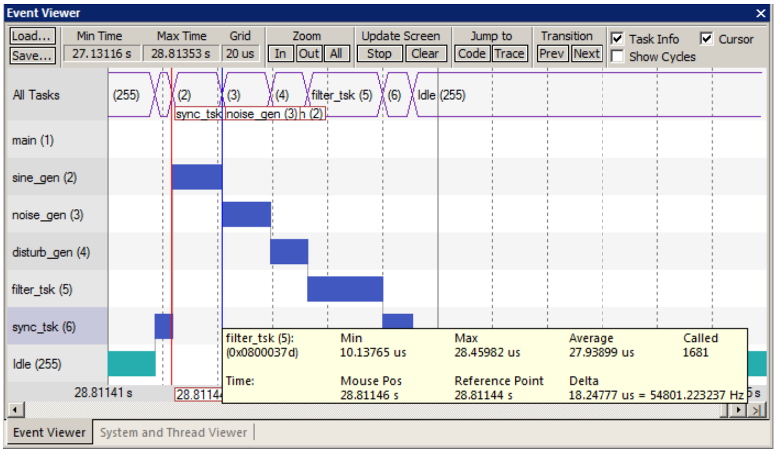
TIP: ITM Port 31enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port if you are not using the Event Viewer.



Event Viewer Timing:

1. Click on Zoom In until one set of tasks is visible as shown below:
2. Enable Task Info (as well as Cursor and Show Cycles from the previous exercise).
3. Note one entire sequence is shown. This screen is taken with ST-Link V 2with LA cleared of variables.
4. Click on a task to set the cursor and move it to the end. The time difference is noted. The Task Info box will appear.

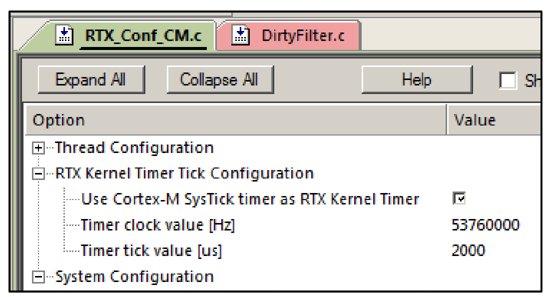
TIP: If the Event Viewer does not display correctly, the display of the variables in the Logic Analyzer window might be overloading the SWO pin. In this case, stop the program and delete all variables (Kill All) and click on Run.



The Event Viewer can give you a good idea if your RTOS is configured correctly and running in the right sequence.

Changing the SysTick Timer Value:

1. Stop the processor and exit debug mode.
2. Open the file RTX_Conf_CM.c from the Project window. You can also select File/Open in C:\Keil\ARM\Boards\ST\STM32F4-Discovery\DSP.
3. Select the Configuration Wizard tab at the bottom of the window. See page 12 for an explanation on how the Wizard works.
4. This window opens up. Expand SysTick Timer Configuration.
5. Note the Timer tick value is 1000 usec or 1 msec.
6. Change this value to 2,000.



TIP: The 5,376,000 is the CPU speed. The Discovery board has a 8 MHz crystal. This program was designed for 168 MHz with a 25 MHZ crystal. Therefore it runs 8/25 slower than designed for. The PLL is configured in CMSIS file system_stm32f4xx.c and is easily modified.

7. Rebuild the source files and program the Flash.
8. Enter debug mode and click on RUN .
9. When you check the timing of the tasks in the Event Viewer window as you did on the previous page, they will now be spaced at 2 msec.

TIP: The SysTick is a dedicated timer on Cortex-M processors that is used to switch tasks in an RTOS. It does this by generating an exception 15. You can view these exceptions in the Trace Records window by enabling EXCTRC in the Trace Configuration window.

1. Set the SysTick timer back to 1,000. You will need to recompile the source files and reprogram the Flash.
2. Click on File/Save All or select the Save All icon:

This ends the exercises. Thank you !

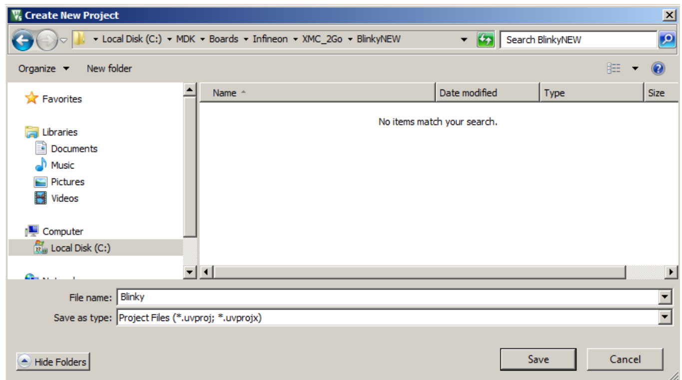
Next is how to make a new project from scratch, how ETM trace works and Keil product and contact information.

23) Creating your own MDK 5 project from scratch:

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

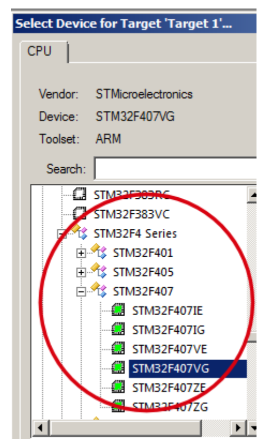
Install the STM32 Software Pack for your processor:

1. Start µVision 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.
3. You do not need to copy any examples over.



Create a new Directory and a New Project:

1. Click on Project/New µVision Project…
2. In the window that opens, shown below, go to the folder C:\MDK\Boards\ST\STMF32F4-Discovery\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj in C:\MDK\Boards\ST\STMF32F4-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 STM32F407 and then finally select STM32F407VG:

TIP: Processor icons in green are from the Software Packs. Grey icons are from MDK 4.7x.

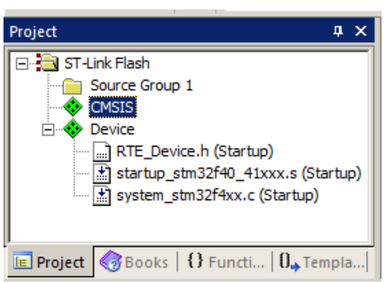
2. Click OK and the Manage Run Time window shown below bottom right opens.

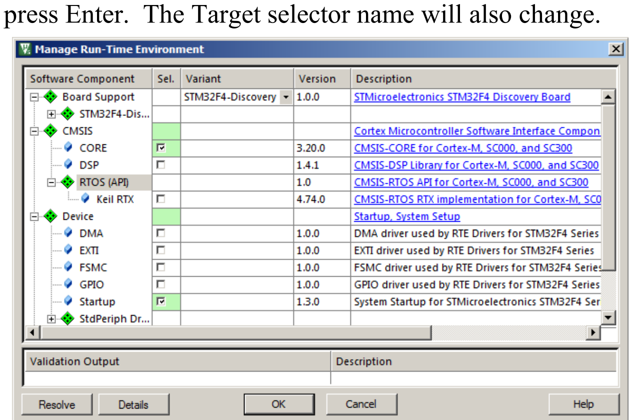
Select the CMSIS components you want:

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

What has happened to this point:

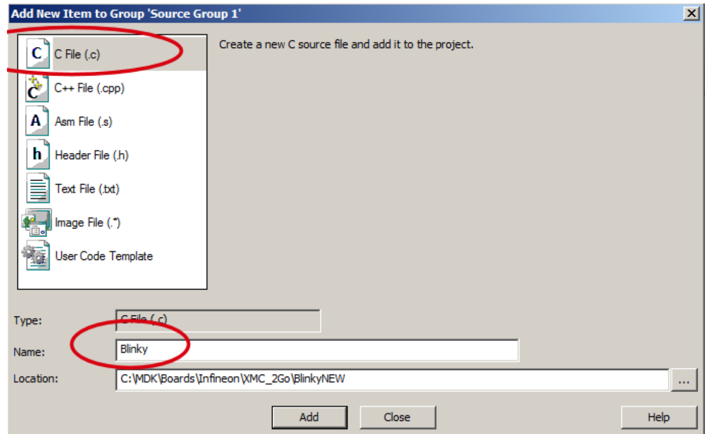
You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files.





Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select Add New Item to Group 'Source Files'... .
2. This window opens up:



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

Add Some Code to Blinky.c:

9. In the blank Blinky.c, add the C code below:
10. Click on File/Save All or
11. Build the files. There will be no errors or warnings if all was entered correctly.

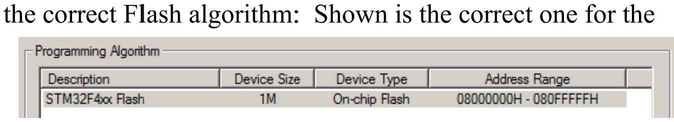
```
#include "stm32f4xx.h"
unsigned int counter = 0;
/*----------------------------------------------
  MAIN function
 *----------------------------------------------*/
int main (void) {

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

TIP: You can also add existing source files: Add Existing Files to Group 'Source Files'... No need to at this time.

Configure the Target CMSIS-DAP: ***Please complete these instructions carefully to prevent unusual problems…***

1. Select the Target Options icon . Select the **Target** tab.
2. Enter 8 in Xtal (MHz). This is used for timing calculations. Select Use MicroLIB to optimize for smaller code size.
3. Select the **Output** tab. Click on Select Folder for Objects…:
4. In the Browse for Folder window that opens: right click and create a new folder called Flash. Select Folder for Objects...
5. Double click on Flash to enter this folder and click OK. Compilation files will now be stored in this Flash folder.
6. Click on the **Listings** tab. Click on Select Folder for Objects…: Double click on Flash and click OK to close.
7. Click on the **Debug** tab. Select St-Link Debugger in the Use: box: Use: ST-Link Debugger Settings
8. Select the Settings: icon.
9. Select SW as shown here in the Port: box: Port: SW JTAG here will not work with SWV. If your board is connected to your PC, you ***must*** now see a valid IDCODE and Device Name in the SW Device box.
10. Click on OK ***once*** to go back to the Target Configuration window. Otherwise, fix the connection problem.
11. Click on the **Utilities** tab. Select Settings and Add the correct Flash algorithm: Shown is the correct one for the STM32Fx series processors:

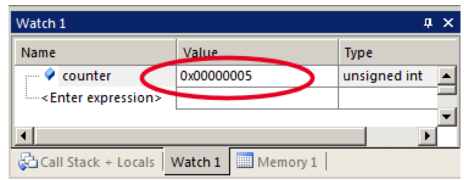


12. Click on OK twice to return to the main menu.
13. Click on File/Save All or
14. Build the files. There will be no errors or warnings if all was entered correctly. If there are, please fix them !

The Next Step ? Let us run your program and see what happens ! Please turn the page….

Running Your Program:

1. Program the ST Flash by clicking on the Load icon: Progress will be indicated in the Output Window.
2. Enter Debug mode by clicking on the Debug icon .
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.
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.



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

Clock Frequency: The CPU clock speed is 53.76 MHz. To change it to 168 MHz:

1. Open the file system_stm32f4xx.c.
2. Locate the line #define PLL_M 25 as shown below:
3. Change the value to 8.
4. Click on File/Save All or
5. Build the files.
6. The CPU speed is now 168 MHz.

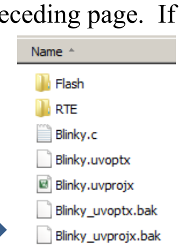
```
/************************* PLL Parameters *****************
/* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLL_M) * PLL_N */
#define PLL_M      25
/* USB OTG FS, SDIO and RNG Clock =  PLL_VCO / PLLQ */
#define PLL_Q      7
```

TIP: If PLL_M = 336 and PLL_Q = 4, then the clock speed is 84 MHz.

Cleaning up your Project: (you only need to do this once: this is not a critical step)

We modified the folder where the output and listings files are stored. This was in Steps 3 through 7 on the preceding page. If you did a Build before this was done, there will be files in your project root directory. Now we want them only in .\Flash.

1. Exit µVision. Otherwise, you can't delete files that it still has open.
2. Open Microsoft Explorer and navigate to: C:\MDK\Boards\ST\STMF32F4-Discovery\BlinkyNEW\.
3. Delete all files and folders except these: (you can delete Flash – a Build will recreate it.)
4. You can also leave any backup or µVision files that identify your computer to retain your settings.
5. Restart µVision. Having all compilation files stored in the .\Flash folder makes it cleaner.



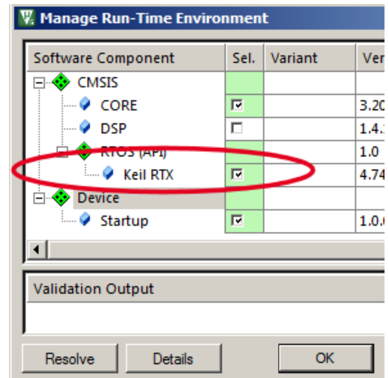
TIP: If you want to save or send the project files to someone, you can delete the folder Flash to reduce file size. This folder and its contents are easily reconstructed with a Build.

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

The MDK Software Packs makes it easy to configure an RTX project. There are two versions of RTX: The first comes with MDK 4.7x and earlier. The second comes with MDK 5.10 and later. This second one is CMSIS-RTOS compliant.

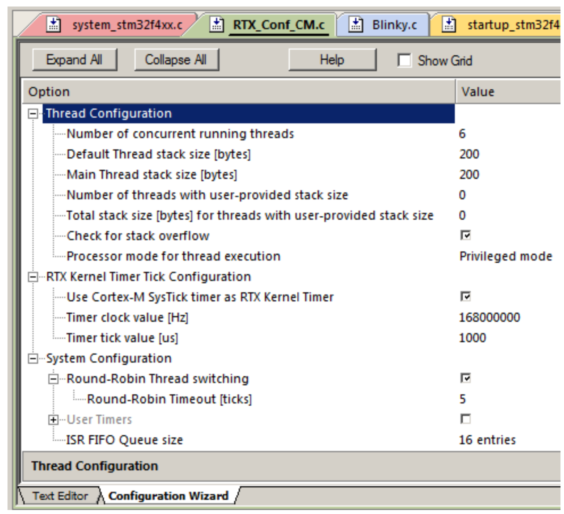
Configuring RTX is easy in MDK 5.10 and later. These steps use the same configuration as in the preceding Blinky example.

1. Using the same example from the preceding pages, Stop the program and Exit Debug mode.
2. Select ST-Link Flash: ST-Link Flash
3. In Blinky.c, at the top, add this line: #include "cmsis_os.h"
4. Open the Manage Run-Time Environment window:
5. Expand all the elements as shown here:
6. Select Keil RTX as shown and click OK.
7. Appropriate RTX files will be added to your project. See the Project window.
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 168000000 as shown: (168 MHz)
6. Unselect User Timers. Use defaults for the other settings.

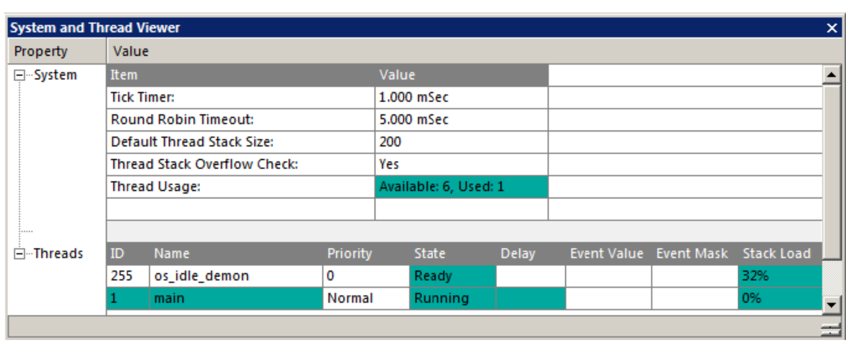


Build and Run Your RTX Program:

1. Build the files. Program the Flash: .
2. Enter Debug mode: Click on the RUN icon.
3. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
4. 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.
2. See the DSP5 and RTX_Blinky5 examples to use as templates and hints.
3. If you copy Blinky.c from the RTX_Blinky project, it will blink the LEDs. It has the RTX code incorporated into it.
4. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/mdk5/. It has very useful information on implementing and maintaining RTX.



This completes the exercise of creating your own RTX project from scratch.

25) ETM Trace Examples: *For reference only…**Note:** MDK 5 has enhanced windows and triggers:*

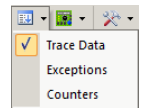
These examples were run on the STM3240G-EVAL evaluation board. These are applicable for the Keil MCBSTM32F400 board. These examples are included for reference. A ULINK*pro* debug adapter is required for ETM operation.

ETM provides serious debugging power as shown on the next few pages. It is worth the small added cost.

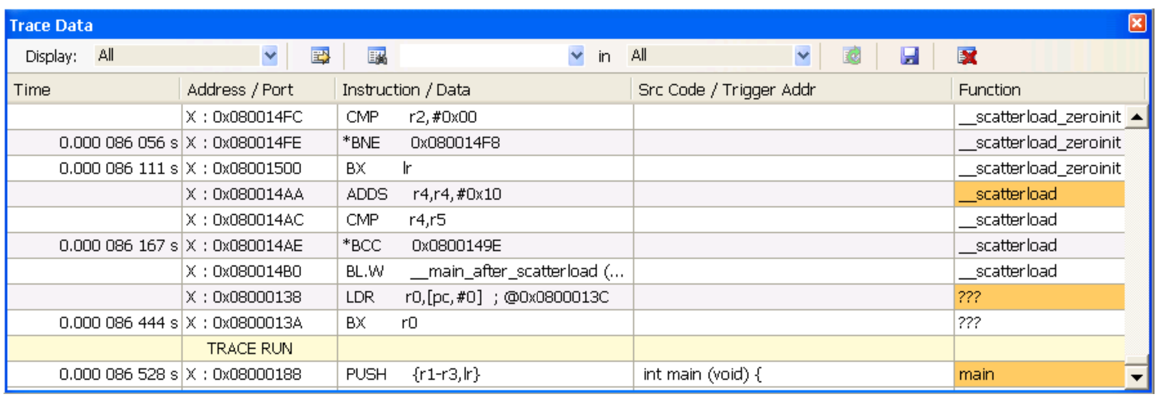
Most STM32 processors are ETM equipped.

1. Connect the ULINK*pro* to the STM3240G board using the 20 pin CN13 Trace connector.
2. Start µVision 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 µVision halted the program at the start of main() since Run To main is selected in µVision.
11. In this case, 086 444 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_0188 in my case). Click on Single Step once.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x080014FC	CMP r2,#0x00		__scatterload_zeroinit
0.000 086 056 s	X : 0x080014FE	*BNE 0x080014F8		__scatterload_zeroinit
0.000 086 111 s	X : 0x08001500	BX lr		__scatterload_zeroinit
	X : 0x080014AA	ADDS r4,r4,#0x10		__scatterload
	X : 0x080014AC	CMP r4,r5		__scatterload
0.000 086 167 s	X : 0x080014AE	*BCC 0x0800149E		__scatterload
	X : 0x080014B0	BL.W __main_after_scatterload (…		__scatterload
	X : 0x08000138	LDR r0,[pc,#0] ; @0x0800013C		???
0.000 086 444 s	X : 0x0800013A	BX r0		???
	TRACE RUN			
0.000 086 528 s	X : 0x08000188	PUSH {r1-r3,lr}	int main (void) {	main

12. The instruction PUSH will display as shown on the last line in the trace window. Note the function column diplsyas the function where the assembly isnstructions are located with the start highlighted in orange.
13. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.

TIP: You can set Trace Triggers to start and stop the Trace collection. This allows you to save only the frames you are interested in.

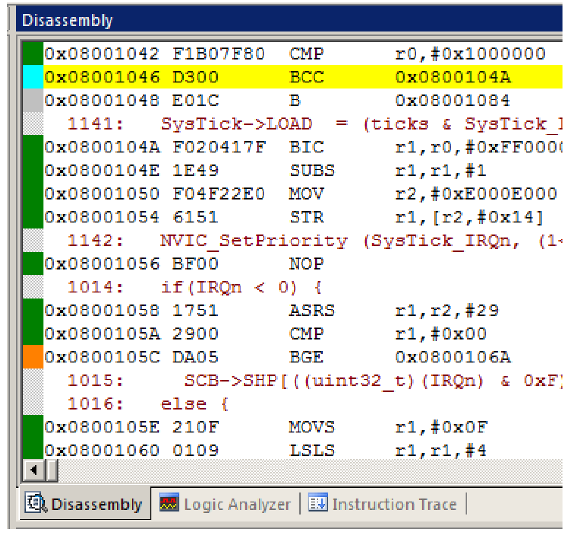
A STM3240G-EVAL board connected to a ULINK*pro* using the special CoreSight 20 pin ETM connector:

3) Code Coverage: *For reference only…*

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).

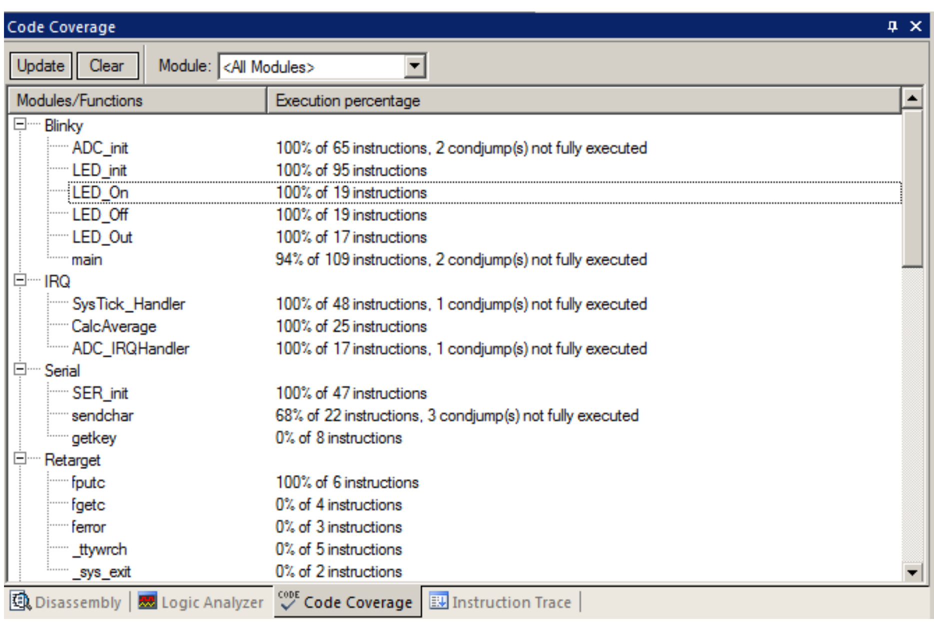
Why was the branch BCC always taken resulting in 0x0800_1048 never being executed ? Or why the branch BGE at 0x800_105C was never taken ? You should devise tests to execute these instructions so you can test them.

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.

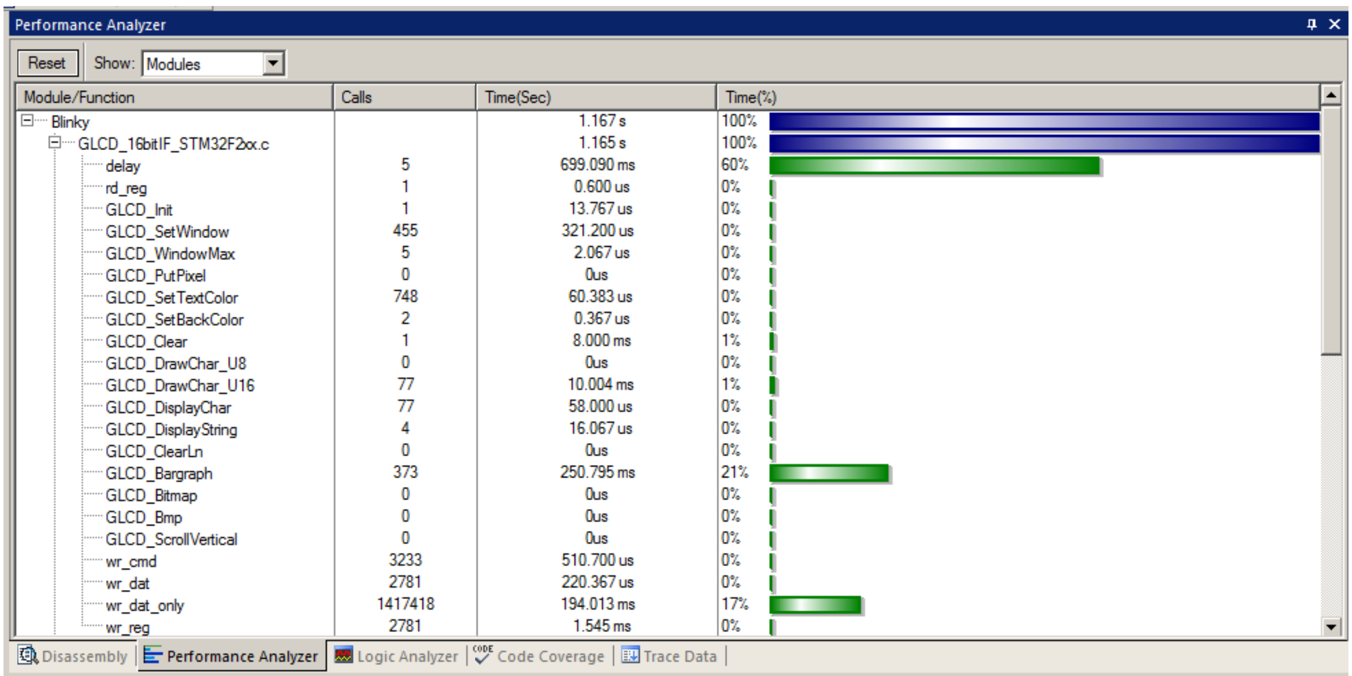


4) Performance Analysis (PA): *For reference only…*

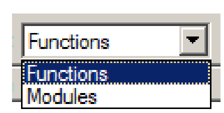
Performance Analysis tells you how much time was spent in each function. The data can be provided by either the SWV PC Samples or the ETM. If provided by the SWV, the results will be statistical and more accuracy is improved with longer runs. Small loops could be entirely missed. ETM provides complete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

Keil provides Performance Analysis with the µVision 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.

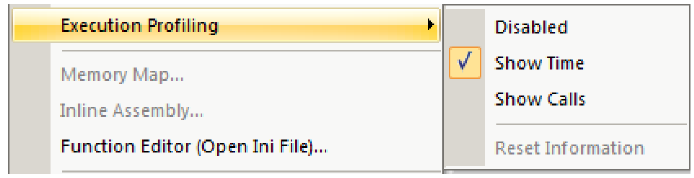


TIP: You can also click on the RESET icon but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

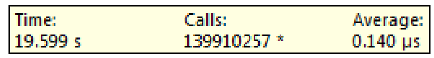
When you click on the RESET icon, the Initialization File .ini will no longer be in effect and this can cause SWV and/or ETM to stop working. Exiting and re-entering Debug mode executes the .ini script again.

5) Execution Profiling: *For reference only…*

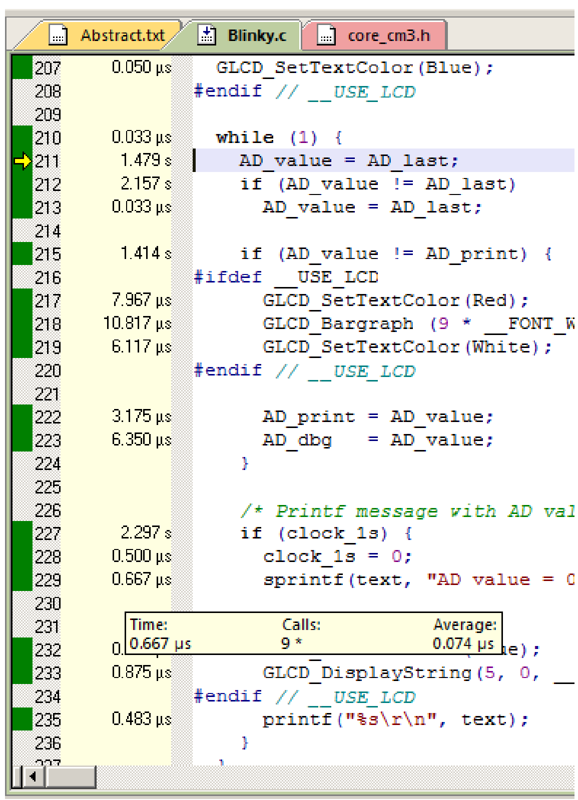
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace. It is possible to group source lines (called collapse) to get combined times and number of calls. This is called Outlining. The µVision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



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

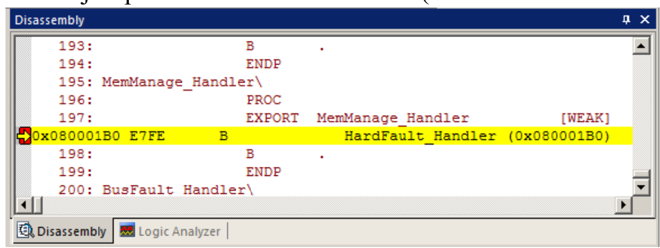


6) In-the-Weeds Example: *For reference only... A ULINKpro is needed for Instruction Trace:*

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

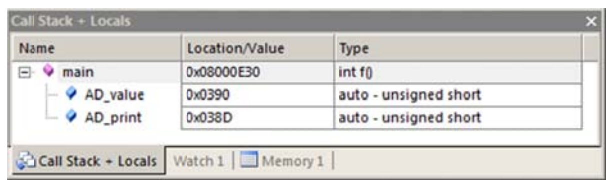
If a Hard Fault occurs, the CPU will end up at the address specified in the Hard Fault vector located at 0x00 000C. This address points to the Hard Fault handler. This is usually a branch to itself and this Branch instruction will run forever. The trace buffer will save millions of the same branch instructions. 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.

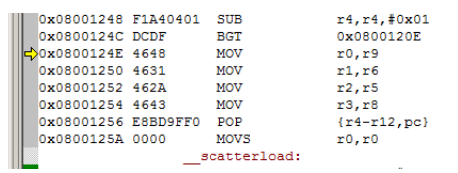
1. Open the Blinky_Ulp example, rebuild, program the Flash and enter Debug mode. Open the Data Trace window.
2. Locate the Hard fault vector near line 207 in the disassembly window or in startup_stm32f4xx.s.
3. Set a breakpoint at this point. A red block will appear as shown above.
4. Run the Blinky 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:

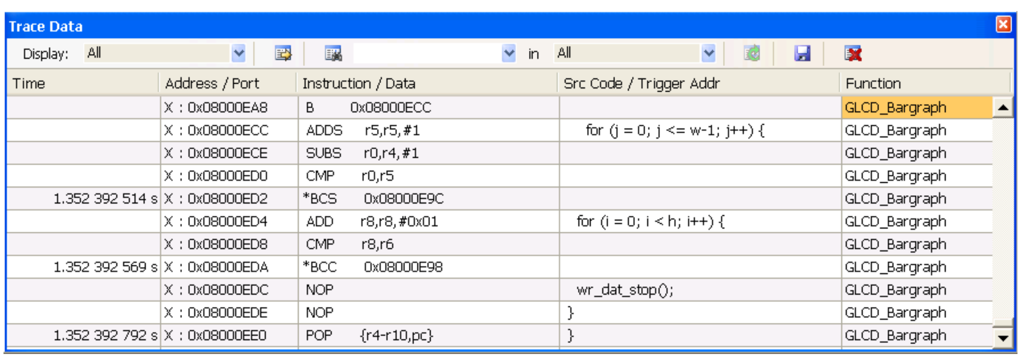


7. Right click on the POP instruction (or at the MOV at 0x0800 124E as shown below) and select Set Program Counter. This will be the next instruction executed.
8. Click on RUN and immediately the program will stop on the Hard Fault exception branch instruction.
9. Examine the Data Trace window and you find this POP plus everything else that was previously executed. In the bottom screen are the 4 MOV instructions plus the offending POP.

TIP: The addresses you get will be different than these ones.

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.





Trace Data

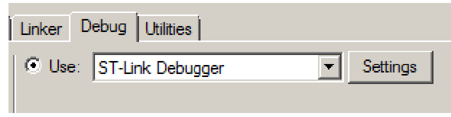
Display: All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x08000EA8	B 0x08000ECC		GLCD_Bargraph
	X : 0x08000ECC	ADDS r5,r5,#1	for (j = 0; j <= w-1; j++) {	GLCD_Bargraph
	X : 0x08000ECE	SUBS r0,r4,#1		GLCD_Bargraph
	X : 0x08000ED0	CMP r0,r5		GLCD_Bargraph
1.352 392 514 s	X : 0x08000ED2	*BCS 0x08000E9C		GLCD_Bargraph
	X : 0x08000ED4	ADD r8,r8,#0x01	for (i = 0; i < h; i++) {	GLCD_Bargraph
	X : 0x08000ED8	CMP r8,r6		GLCD_Bargraph
1.352 392 569 s	X : 0x08000EDA	*BCC 0x08000E98		GLCD_Bargraph
	X : 0x08000EDC	NOP	wr_dat_stop();	GLCD_Bargraph
	X : 0x08000EDE	NOP	}	GLCD_Bargraph
1.352 392 792 s	X : 0x08000EE0	POP {r4-r10,pc}	}	GLCD_Bargraph

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

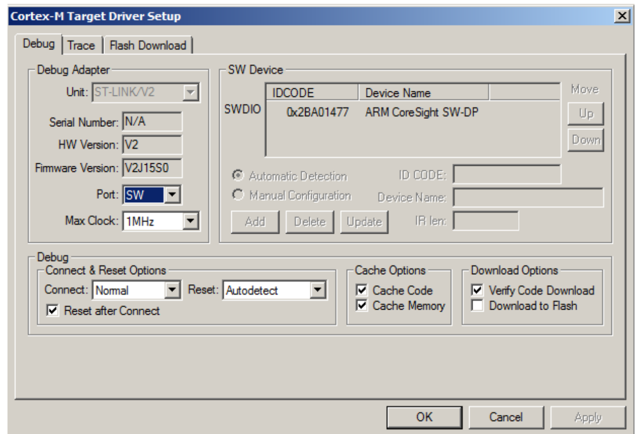
26) Configuring the ST-Link Debug Connection to the target: *The following steps are already done by default in the example programs. These instructions are provided for reference.*

1. Connect your PC to the Discovery board with a USB cable. Start µVision. 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 select ST-Link Debugger as shown here:



TIP: Do NOT select ST-Link (Deprecated Version).

3. Select Settings and the next window below opens up. This is the control panel for the ULINKs and ST-Link. (they are the same).
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 **MUST** be displayed. 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 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.

TIP: You can do everything with SW (SWD) as you can with JTAG except for boundary scan.

Next page: configure the Keil Flash programming tool:

Debug Adapters:

It is easy to select a USB debugging adapter in µVision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are each selected using the Debug and Utilities tabs.

Using other Debug Adapters: This document uses the on-board ST-Link. 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.

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 on another board. Its main purpose is not to connect an external tool such as a Keil ULINK2. Some adaptation is required but not difficult to do.

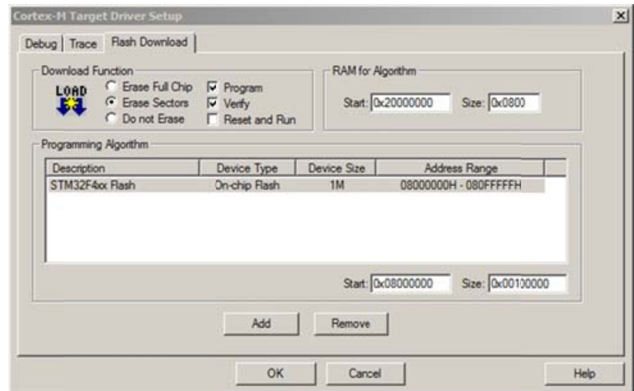
It is possible to use a Segger J-Link with µVision. Serial Wire Viewer is supported.

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

Serial Wire Viewer (SWV) is completely supported by ST-LINK Version 2. Firmware V2.16.S0 or later except V2.19.S0.

27) Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ST-Ling Debugger similar to Step 2 above.
8. Click Settings to select the programming algorithm if it is not visible or is the wrong one.
9. Select STM32F4xx Flash as shown here or the one for your processor:
10. Click on OK once.



TIP: To program the Flash every time you enter Debug mode, check Update target before Debugging.

TIP: If you select Use Debug Driver, the debugger you selected in the Debug tab will be used.

11. Click on OK to return to the µVision main screen.
12. Select File/Save All.
13. You have successfully connected to the STM32 target processor and configured the µVision Flash programmer.

TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV). You will learn to do this later.

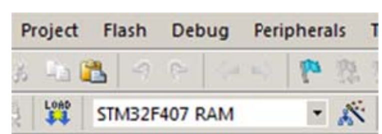
COM led LD1 indication:

LED is blinking RED: the first USB enumeration with the PC is taking place.
LED is RED: communication between the PC and ST-LINK/V2 is established (end of enumeration). µVision is not connected to ST-Link (i.e. in Debug mode).
LED is GREEN: µVision is connected in Debug mode and the last communication was successful.
LED is blinking GREEN/RED: data is actively being exchanged between the target and µVision.
No Led: ST-LINK/V2 communication with the target or µVision has failed. Cycle the board power to restart.

Running programs in the internal STM32 RAM:

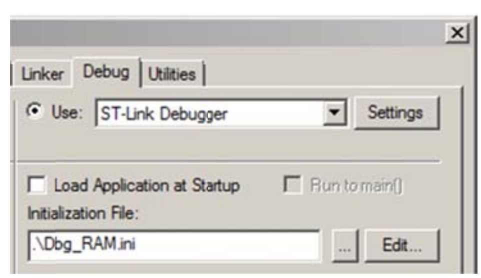
It is possible to run your program in the processor RAM rather than Flash. In this case, the Flash programming tool is not used nor is the Load icon. After successfully compiling the source files, click on Debug icon . An .ini file configures the processor and loads your executable into RAM.

The Discovery Blinky project has a RAM setting. Select STM32F407 RAM as shown here if you want to try this mode.



Loading and Running your program into RAM:

1. Select STM32F407 RAM as shown above.
2. Select Target Options or ALT-F7 and select the Debug tab.
3. The ini file is located in the Initialization File: box as shown here:
4. Click on Edit… to view its contents.
5. Click on the Target tab. Note the RAM at 0x2000_0000 is split between the R/O and R/W memory areas.
6. Click on OK to return to the main µVision window.
7. Return to the STM32F407 Flash setting.



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 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) pleas e visit www.keil.com/st for other labs discussing ETM.

29) Document Resources: See www.keil.com/st

Books:

1. ***NEW!*** **Getting Started MDK 5:** Obtain this free book here: www.keil.com/mdk5/.
2. There is a good selection of books available on ARM processors. A good list of books on ARM processors is found at www.arm.com/university by selecting "Teaching Resources". You can also select ARM Related Books but make sure to also select the "Books suited for Academia" tab to see the full selection.
3. µVision 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
   Click on the Resources tab. Or search for "Cortex-M3" on www.arm.com and click on the Resources tab.
5. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
6. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
7. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

Application Notes:

8. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
9. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
10. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
11. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
12. Using µVision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
13. RTX CMSIS-RTOS in MDK 5 C:\Keil_v5\ARM\Pack\ARM\CMSIS\3.20.4\CMSIS_RTXDownload
14. RTX CMSIS-RTX www.keil.com/demo/eval/rtx.htm **and** www.arm.com/cmsis
15. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
16. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
17. Cortex Debug Connectors: www.arm.com and search for cortex_debug_connectors.pdf
18. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp

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

Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content

ARM University program: www.arm.com/university. Email: university@arm.com

ARM Accredited Engineer Program: www.arm.com/aae

mbed: http://mbed.org

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

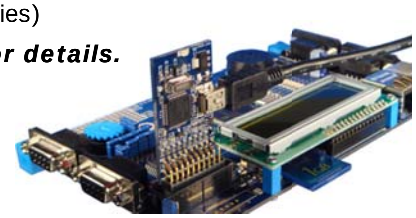
For more information on the ARM CMSIS standard: www.arm.com/cmsis,

30) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite™ (Evaluation version) $0
- ***NEW !!*** MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit)
- MDK-Standard™ (unlimited compile and debug code and data size)
- MDK-Professional™ (Includes Flash File, TCP/IP, CAN and USB driver libraries)

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)*
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINK*pro* – Cortex-M*x* SWV & ETM trace.

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

The Keil RTX RTOS is now provided under a Berkeley BSD type license. This makes it free.

All versions, including MDK-Lite, includes Keil RTX RTOS *with source code !*
www.keil.com/demo/eval/rtx.htm or C:\Keil_v5\ARM\Pack\ARM\CMSIS

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

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

All products 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 to view various programs and resources.

MDK supports STM32 Cortex-M3 and Cortex-M4 processors. Keil supports many other ST processors including 8051, ARM7, ARM9™ and ST10 processors. See the Keil Device Database® on www.keil.com/dd for the complete list of STMicroelectronics support. This information is also included in MDK.

Contact Keil Sales for USA prices. Contact sales.intl@keil.com for pricing in other countries.

For the entire Keil catalog see 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.

For more information:

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

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

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

For the latest version of this document and for more STMicroelectronics specific information, go to www.keil.com/st

CMSIS Version 3: See www.arm.com/cmsis and http://community.arm.com/groups/tools/content for more information.

Also see www.keil.com/st and www.keil.com/forum

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™ Intelligent Processors by ARM®

KEIL™ Tools by ARM