STMicroelectronics STM32: Cortex™-M4 Lab

ARM® Keil™ MDK Toolkit *featuring Serial Wire Viewer and ETM Trace*

For the STM3240G-EVAL board Version 0.72

Robert Boys bob.boys@arm.com

For the ST STM3240G-EVAL Evaluation Board with STM32F407

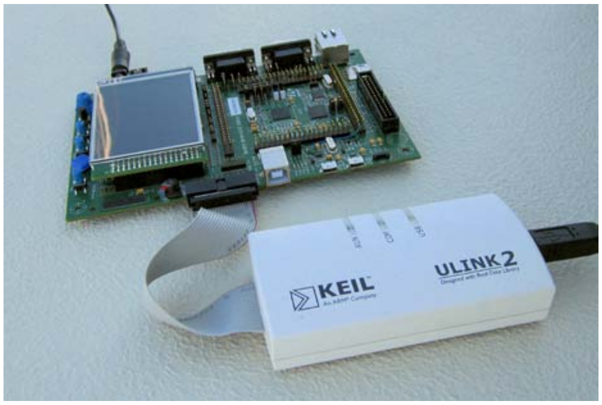
Introduction:

The purpose of this lab is to introduce you to the STMicroelectronics Cortex™-M4 processor family using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) and ETM trace on the STM3240G-EVAL evaluation board from STMicroelectroncs. At the end of this tutorial, you will be able to confidently work with STM32 processors and MDK. Keil offers a similar board: MCBSTM32F400™. Examples are provided for both boards.

Keil MDK comes in an evaluation version that limits code and data size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a license number will turn it into the full, unrestricted version. Contact Keil sales for a temporary full version license if you need to evaluate MDK with programs greater than 32K. MDK includes a full version of Keil RTX™ RTOS. No royalty payments are required. RTX source code is now included with all versions of Keil MDK™.

Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M3 and Cortex-M4 users:



1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is a turn-key product with included examples.
2. Serial Wire Viewer and ETM trace capability is included. A full feature Keil RTOS called RTX is included with MDK with source code.
3. RTX Kernel Awareness window is updated in real-time. Kernel Awareness exists for Keil RTX, CMX, Quadros and Micrium. All RTOSs can compile with MDK. Awareness can be provided by the supplier.
4. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™ or Segger J-Link (version 6 or later). ST-Link is supported but it has no SWV or ETM support at this time. SWV for ST-Link is planned for 4Q11.
5. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.

This document details these features:

1. Serial Wire Viewer (SWV) with ULINK2, ULINK-ME and ULINK*pro*. ETM Trace using ULIN*Kpro*.
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 called Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS that updates while the program is running.

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 the Cortex-M4. SWV is output on the Serial Wire Output (SWO) pin found on the JTAG/SWD adapter connector.

SWV does not steal any CPU cycles and is completely non-intrusive except for ITM Debug printf Viewer. SWV is provided by the Keil ULINK2, ULINK-ME, ULINK*pro* and the Segger J-Link. Best results are with a ULINK family adapter. The STMicroelectronics ST-Link adapter does not support SWV at this time.

Embedded Trace Macrocell (ETM):

ETM adds all the program counter values to the data provided by SWV. This allows advanced debugging features including timing of areas of code (Execution Profiling), Code Coverage, Performance Analysis and program flow debugging and analysis. ETM requires a special debugger adapter such as the ULINK*pro* or Segger J-Trace. This document uses a ULINK*pro* for ETM. A ULINK2 or ULINK-ME is used for the Serial Wire Viewer exercises in this lab.

Part A: Connecting and Configuring to the target board:

Part B: Blinky Example Programs using a ULINK2 or ULINK-ME:

Part C: Using the ULINK*pro* with ETM Trace

STM32 Evaluation Boards:

Keil makes six STM32 evaluation boards plus several with STR7 and STR9 processors. Examples are provided.

Keil part number	Processor	Characteristics	Debug Connectors	ST board equivalent
MCBSTM32™	STM32F103VB	monochrome LCD	JTAG/SWD	STM32F10X-128K-EVAL (color LCD)
MCBSTM32E™ ***replaced by EXL***	STM32F103ZE	color LCD	Cortex Debug and ETM	STM3210E-EVAL
MCBSTM32EXL™	STM32F103ZG	color LCD	Cortex Debug and ETM	STM3210E-EVAL
MCBSTM32C™	STM32F107VC	color touch LCD	Cortex Debug and ETM	STM3210C-EVAL
MCBSTM32F200™	**Cortex-M4:**	MCBSTM32F400™		

Keil MDK provides example projects for these STMicroelectronics boards:

CQ-STARM	EK-STM32F	STM32-Discovery	STM32F10X-EVAL	STM32L152-EVAL
STM32100E-EVAL	**Cortex-M4:**	**STM3240G-EVAL**	**STM32F4-Discovery (MDK has examples for this board)**	

Five Steps to Get Connected and Configured:

1. Physically connect a ULINK to the STM3240G or other target board. Power both of these appropriately.
2. Configure µVision to use a ULINK2, ULINK-ME or ULINK*pro* to communicate with the JTAG or SWD port.
3. Configure the Flash programmer inside µVision to program the STM32 internal flash memory.
4. If desired, configure the Serial Wire Viewer. Add the STM32F4xx_SWO.ini initialization file (see below).
5. If desired, configure the ETM trace with the ULINK*pro*. Add the STM32F4xx_TP.ini initialization file (see below).

STM32 processors need a special .ini file that configures the CoreSight Serial Wire Viewer and ETM trace. If you do not intend to use SWV or ETM you do not need this file. It is entered in the Options for Target window under the Debug tab. It can be configured for either SWO or 4 bit Trace Port operation. Instructions are provided on Page 30.

Software Installation:

This document was written for Keil MDK 4.22a or later which contains µVision 4. The evaluation copy of MDK is available free on the Keil website. Do not confuse µVision4 with MDK 4.0. The number "4" is a coincidence.

To obtain a copy of MDK go to www.keil.com/arm and select the Download icon: Download

You can use the evaluation version of MDK and a ULINK2, ULINK-ME, ULINK*pro* or J-Link for this lab. You must make certain adjustments for non-ULINK adapters such as the ST-Link and not all features shown here will be available.

The addition of a license number converts the evaluation into a full, unrestricted copy of MDK.

The ULINK*pro* adds Cortex-M3 ETM trace support. It also adds faster programming time and better trace display. Most STMicroelectronics Cortex-M3/M4 parts are equipped with ETM. All have SWV.

JTAG and SWD Definitions: It is useful to have an understanding of these terms:

JTAG: JTAG provides access to the CoreSight debugging module located on the STM32 processor. It uses 4 to 5 pins.

SWD: Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except no Boundary Scan. SWD is referenced as SW in the µVision Cortex-M Target Driver Setup. See page 5, middle picture.

SWV: Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.

SWO: Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.

Trace Port: A 4 bit port that ULINK*pro* uses to output ETM frames and optionally SWV (rather than the SWO pin).

ETM: Embedded Trace Macrocell: Provides all the program counter values. Only the ULINK*pro* works with ETM.

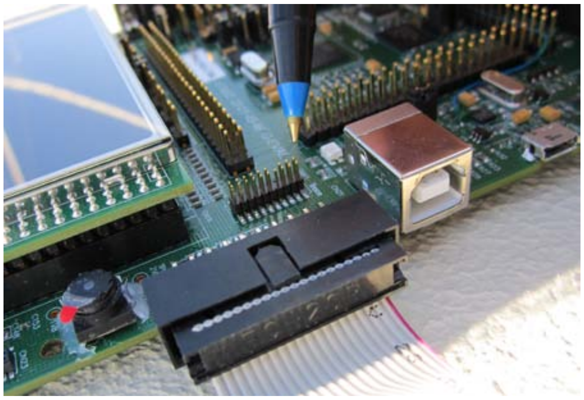
Example Programs: See www.keil.com/st for additional information.

Example projects for STMicroelectronics boards are found in C:\Keil\ARM\boards\ST and in C:\Keil\ARM\boards\Keil for Keil boards. Most example projects are pre-configured to use a ULINK2 or a ULINK-ME. Serial Wire Viewer is not usually configured: you can do this yourself easily. Projects that contain a Ulp in their name are configured to use a ULINK*pro* and SWV and ETM are pre-configured. It is easy to select different debug adapters in µVision.

Most example projects will compile within the 32 K code and data limit of the evaluation version of MDK. An exception is LCD_Blinky. A compiled executable called Blinky.axf file is provided to allow you to run, evaluate and debug these programs. If you attempt to compile these projects, the Blinky.axf file will be erased. It is a good idea to back this file up.

Part A)

1) Connecting ULINK2, ULINK-ME or ULINK*pro*:

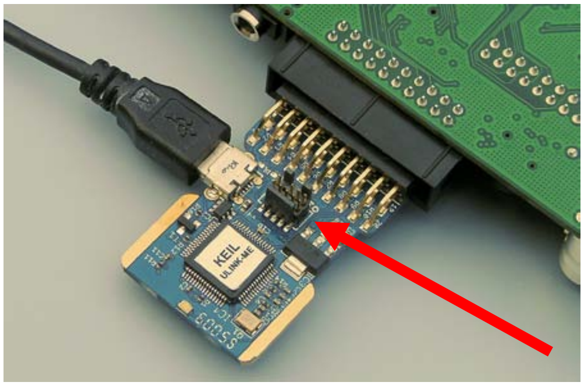


The STM3240G is equipped with the new ARM standard 20 pin Hi-Density connector for JTAG/SWD, SWO and ETM access as shown pointed to by the pen. It is labeled **CN13: Trace**

The legacy 20 pin JTAG connector is also provided. This provides JTAG, SWD and SWO access. No ETM is available on this connector.

A 10 pin connector in the same form factor as the 20 pin Hi-Density exists but is not provided on the STM3240G. This 10 pin provides JTAG, SWD and SWO access in a much smaller footprint. This connector is supported by ULINK2 and ULINK-ME with a special supplied cable. It is shown in the ULINK-ME photo below indicated by the red arrow.

The 20 pin connector CN13 provides JTAG, SWD, SWO and adds 4 bit ETM support and connects to the ULINK*pro*.



Connecting a ULINK2 or ULINK-ME:

Legacy 20 Pin JTAG Connector:

A ULINK2 plugged to the STM3240G board is pictured on the first page of this document.

The ULINK-ME is pictured here and the arrow points to the 10 pin Hi-Density connector.

20 Pin Connector: Keil does have a 10 pin to 20 pin adapter cable available to connect to this connector and is supplied with ULINK-ME. The first 10 pins on the 20 pin replicate the 10 pin.

The second 10 pins on the 20 pin contain the five ETM signals.

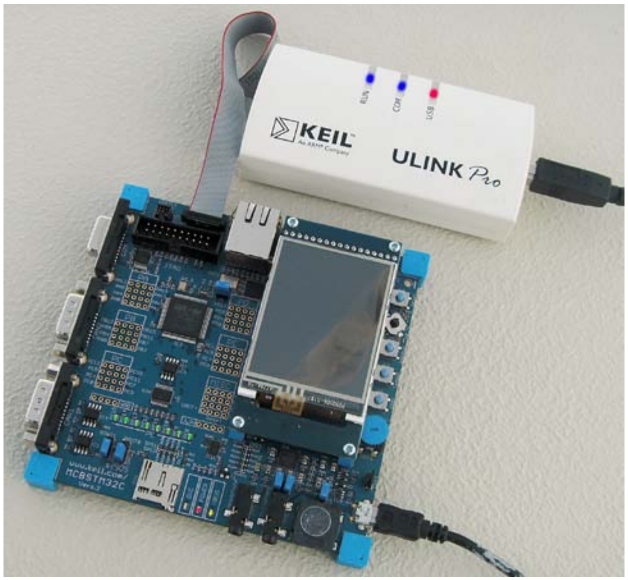
Connecting a ULINK*pro*:

The ULINK*pro* connects to a STM32 board with its standard 20 pin Hi—Density connector or the standard JTAG connector with a supplied adapter.

In order to use ETM trace you must connect the ULINK*pro* to the 20 pin Hi-density connector as shown below:

If you use the legacy 20 pin connector you can use JTAG, SWD and SWV but not ETM.

Pictured is a ULINK*pro* with the STM3220F-EVAL from STMicroelectronics (below) and the Keil MCBSTM32C (right).



2) ULINK2 or ULINK-ME and µVision Configuration:

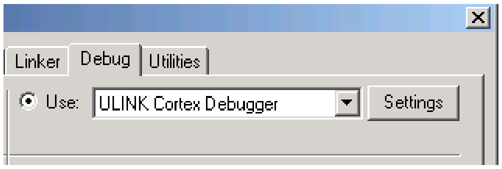
It is easy to select a USB debugging adapter in µVision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are each selected using the Debug and Utilities tabs.

This document will use a ULINK2 or ULINK-ME as described. You can substitute a ULINK*pro* with suitable adjustments.

Serial Wire Viewer (SWV) is completely supported by these two adapters. They are essentially the same devices electrically and any reference to ULINK2 in this document includes the ME. STM32 processors require an .ini file to configure the SWV or ETM features. The ULINK*pro*, which is a Cortex-Mx ETM trace adapter, has all the features of a ULINK2 with the advantages of faster programming time, adds ETM trace support and an enhanced Trace Data window.

Step 1) Select the debug connection to the target:

1. Assume the ULINK2 is connected to a powered up STM32 target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK2 is shown connected to the STM3240G-EVAL board on page 1.
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select ULINK as shown here:
3. Select Settings and the next window below opens up. This is the control panel for the ULINK 2 and ULINK-ME (they are the same).

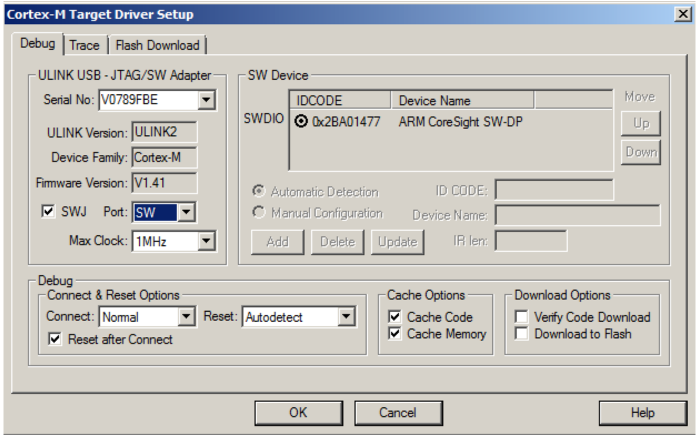


● Use: ULINK Cortex Debugger

4. In **Port:** select SWJ and SW. Serial Wire Viewer (SWV) will not work with JTAG selected. If you are not going to use SWV with the SWO port or will use it with the 4 bit trace port selected, you can use JTAG.
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 can continue. Check the target power supply. Cycle the power to the ULINK and 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 regular debugging using JTAG. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode.



ULINK USB - JTAG/SW Adapter

Serial No: V0789FBE

ULINK Version: ULINK2

Device Family: Cortex-M

Firmware Version: V1.41

☑ SWJ Port: SW

Max Clock: 1MHz

SW Device

	IDCODE	Device Name
SWDIO	● 0x2BA01477	ARM CoreSight SW-DP

● Automatic Detection ID CODE:

○ Manual Configuration Device Name:

IR len:

Debug

Connect & Reset Options: Connect: Normal Reset: Autodetect

☑ Reset after Connect

Cache Options: ☑ Cache Code ☑ Cache Memory

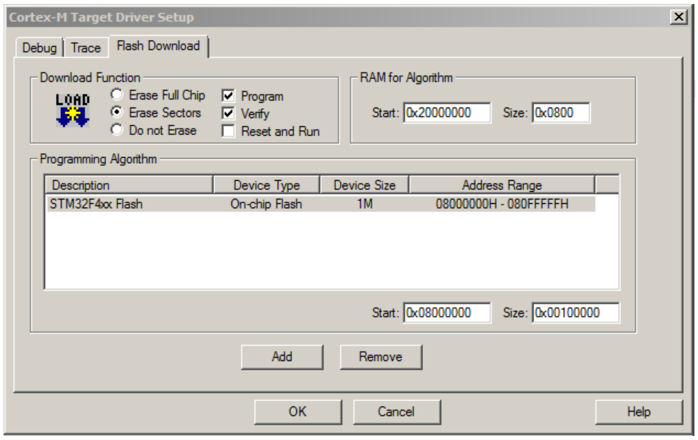
Download Options: ☐ Verify Code Download ☐ Download to Flash

Step 2) Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ULINK 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 below 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.

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



Download Function

○ Erase Full Chip ● Erase Sectors ○ Do not Erase

☑ Program ☑ Verify ☐ Reset and Run

RAM for Algorithm: Start: 0x20000000 Size: 0x0800

Programming Algorithm

Description	Device Type	Device Size	Address Range
STM32F4xx Flash	On-chip Flash	1M	08000000H - 080FFFFFH

Start: 0x08000000 Size: 0x00100000

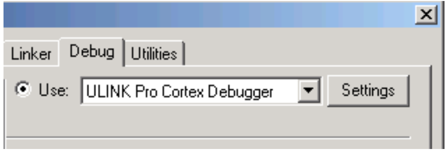
TIP: The Trace tab is where you configure the Serial Wire Viewer (SWV) and ETM trace if you have a ULINK*pro*. You will learn to do this later.

TIP: If you select ULINK or ULINK*pro*, and have the opposite ULINK physically connected to your PC; the error message will say "**No ULINK device found**". This message actually means that µVision found the wrong Keil adapter connected. Merely select the one connected.

3) ULINK*pro* and µVision Configuration:

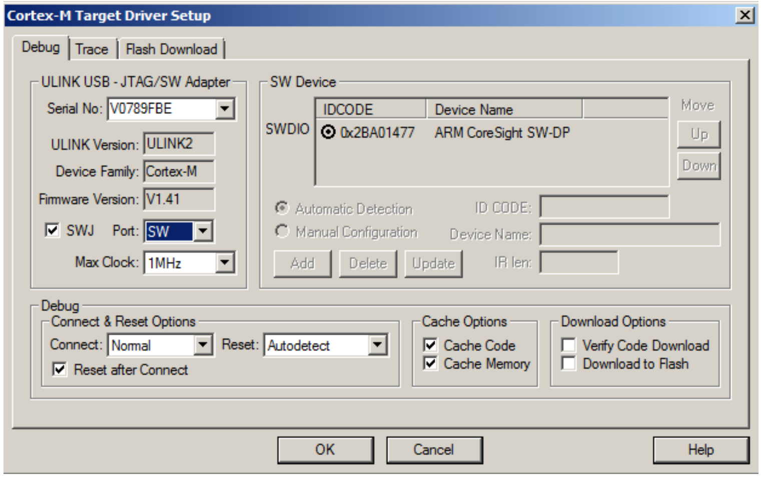
Step 1) Select the debug connection to the target:

1. Assume the ULINK*pro* is connected to a powered up STM32 target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK*pro* is shown connected to the STM3240G-EVAL board on page 4.
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select the ULINK Pro Cortex Debugger as shown here:



● Use: ULINK Pro Cortex Debugger

3. Select Settings and Target Driver window below opens up:
4. In **Port:** select SWJ and SW. SWV will not work with JTAG selected.
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 is blank this **must** be fixed before you can continue. Check the target power supply. Cycle the power to the ULINK and the board.



Serial No: V0789FBE
ULINK Version: ULINK2
Device Family: Cortex-M
Firmware Version: V1.41
☑ SWJ Port: SW
Max Clock: 1MHz

	IDCODE	Device Name
SWDIO	● 0x2BA01477	ARM CoreSight SW-DP

● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
IR len:

Connect: Normal Reset: Autodetect
☑ Reset after Connect
☑ Cache Code
☑ Cache Memory
□ Verify Code Download
□ Download to Flash

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

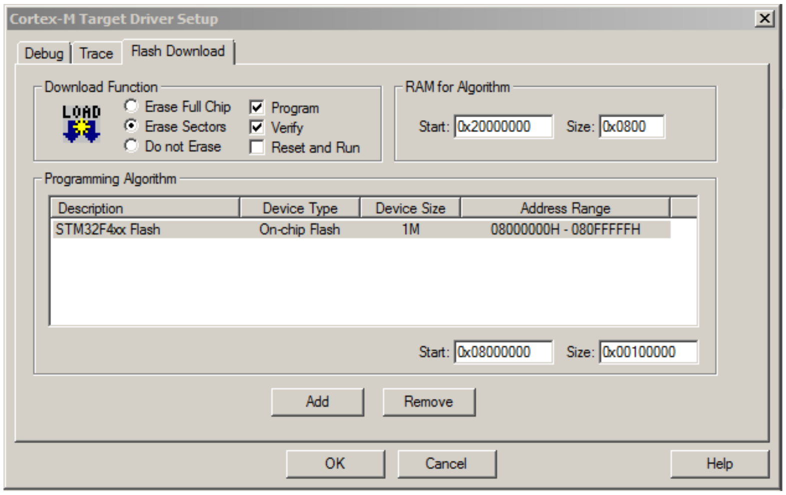
TIP: You can do regular debugging using JTAG. SWD and JTAG operate at approximately the same speed. Serial Wire Viewer (SWV) will not operate in JTAG mode unless the ULINK*pro* is using the Trace Port to output the trace frames.
This option is selected in the Trace tab.

Step 2) Configure the Keil Flash Programmer:

1. Click on OK once and select the Utilities tab.
2. Select ULINK*pro* similar to Step 2 above.
3. Click Settings to select the programming algorithm if one is not visible or is the wrong one.
4. Select STM32F4xx Flash as shown below or the one for your processor:
5. Click on OK once. Select File/Save All.

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

1. Click on OK to return to the µVision main screen.
2. You have successfully connected to the STM32 target processor and selected the µVision Flash programmer.



○ Erase Full Chip ☑ Program
● Erase Sectors ☑ Verify
○ Do not Erase □ Reset and Run

RAM for Algorithm: Start: 0x20000000 Size: 0x0800

Description	Device Type	Device Size	Address Range
STM32F4xx Flash	On-chip Flash	1M	08000000H - 080FFFFFH

Start: 0x08000000 Size: 0x00100000

TIP: If you select ULINK or ULINK*pro*, and have the opposite ULINK physically connected; the error message will say "No ULINK device found". This message actually means that µVision found the wrong Keil adapter connected.

TIP: A ULINK*pro* will act very similar to a ULINK2. The trace window (Trace Data) will be quite different from the ULINK2 Trace Records as it offers additional features.

Trace Data is linked to the Disassembly and Source windows. Double-click on a trace frame and it will be located and highlighted in the two windows.

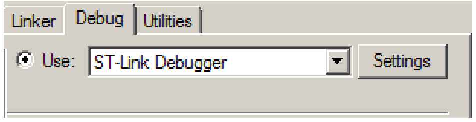
TIP: µVision windows can be floated anywhere. You can restore them by setting Window/Reset Views to default.

4) ST-Link from STMicroelectronics and µVision Configuration:

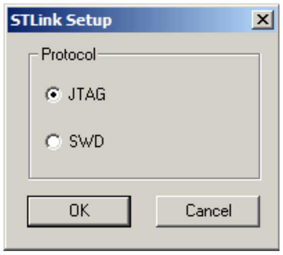
The economical ST-LINK can be used with µVision to provide stable JTAG or SWD debugging. It does not provide Serial Wire Viewer, on-the-fly Watch and Memory updates and write capability, on-the-fly breakpoint setting or Watchpoints.

Step 1) Select the debug connection to the target:

1. Assume the ST-LINK is connected to a powered up STM32 target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project.
   Important TIP: The STM3240G contains a built-in ST-LINK V2 via the USB port. You can use this device instead of an external ST-Link.
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box, select the ST-LINK Debugger as shown here:
3. Select Settings and Target Driver window below opens up:
4. In **Protocol** select either JTAG or SWD. You would only have to select SWD if your target board only has the two SWD signals and not the full set of JTAG signals. ST-LINK does not yet support SWV.



● Use: ST-Link Debugger



STLink Setup

Protocol

● JTAG

○ SWD

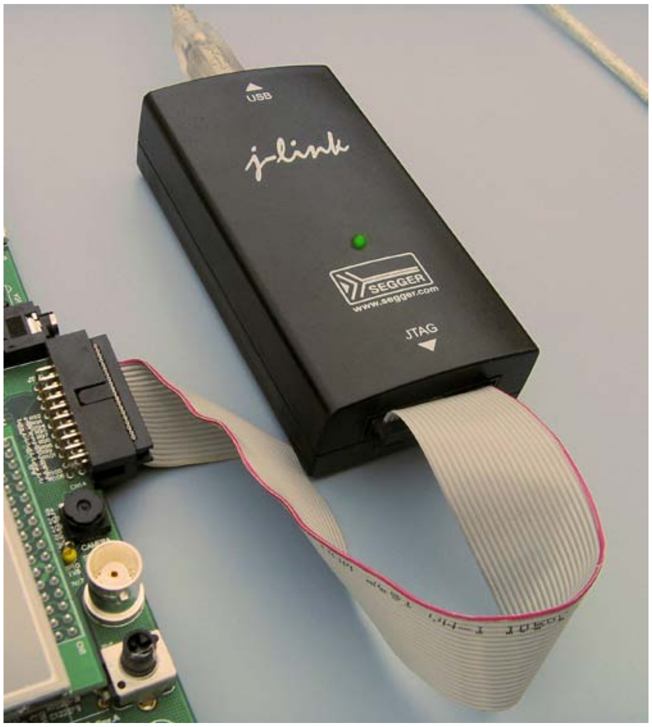
Step 2) Configure the Keil Flash Programmer:

5. Click on OK once and select the Utilities tab.
6. Select the ST-Link Debugger similar to Step 2 above.
7. You do not select any Flash algorithm. ST-LINK does this automatically.
3. Click on OK twice to return to the µVision main screen.
4. You have successfully connected to the STM32 target processor and selected the ST-Link as your debugger.
5. Select File/Save All.

TIP: You do not need to click on the Load icon to program the Flash. Simply enter Debug mode and the Flash will be automatically programmed.

TIP: You do not need the Initialization ini file since the ST-Link does not yet support either SWV or ETM trace.

ST-Link



Segger J-Link

5) Segger J-Link and µVision Configuration:

The J-Link (black box) version 6 or higher provides Serial Wire Viewer capabilities. It provides all debug functions that the Keil ULINK2 provides. This includes breakpoints, watchpoints, memory read/write and the RTX Viewer. J-Link displays exceptions and PC Samples but does not provide ITM, data R/W trace frames in MDK 4.22. Segger has the new J-Link Ultra which is faster. Segger also provides a J-Trace for the Cortex-M3 ETM trace but this has not been tested with MDK for this document. µVision will do an automatic firmware update on the J-Link if necessary. µVision contains all needed drivers.

The J-Link is challenged by a high output on the SWO pin especially where the Logic Analyzer is concerned. Make sure you select only that data that you really need. Disable all others and that can include ITM 31 and 0. Lower the rate the variable is changed or sample it if it is changing too fast. Try disabling the timestamps but some functions need them to operate and the trace may then stop operating. Sometimes you must stop the program to see trace frames. We are working on these issues.

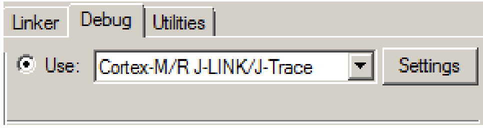
The J-Link is configured using very similar windows as with the ULINK2. This include SWV configuration. The J-Link uses an Instruction Trace window similar to the ULINK*pro*. If you double click on a PC Sample frame, that instruction will be highlighted in the Disassembly and Source windows. The J-Link does not display any ETM frames. Use the J-Trace.

If you have trouble installing the J-Link USB drivers, go to C:\Keil\ARM\Segger\USBDriver and execute InstallDrivers.exe. If the green LED on the J-Link blinks quickly, this means the USB drivers are not installed correctly. This LED should normally be on steady when in Edit mode and off with periodic blinks when in Debug mode.

1. Assume the J-Link is connected to a powered up STM32 target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project.

Step 1: Select the debug connection to the target:

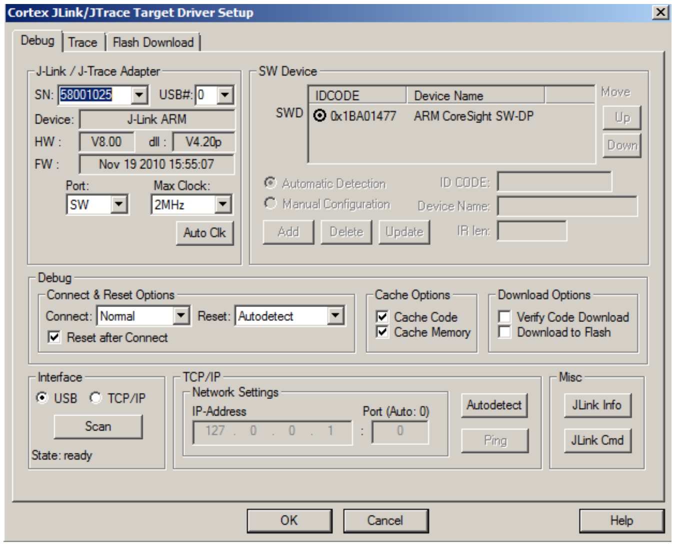
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select the J-LINK or J-Trace as shown here:
3. Select Settings and Target Driver window below opens up:
4. In **Port:** select SW. SWV will not work with JTAG selected.
5. In the SW Device area: ARM CoreSight SW-DP **MUST** be displayed. This confirms you are connected to the target processor. If there is an error displayed or is blank this **must** be fixed before you can continue. Check the target power supply. Cycle power to the J-Link and the board.



● Use: Cortex-M/R J-LINK/J-Trace

Step 2: Configure the Keil Flash Programmer:

6. Click on OK once and select the Utilities tab.
7. Select the ST-Link Debugger similar to Step 2 above.
8. Click Settings to select the programming algorithm if one is not visible or is the wrong one.
9. Select STM32F4xx Flash as shown in the directions for the ULINK2 or the algorithm for your processor.
6. Click OK twice to return to the main screen.
7. You have now selected the J-Link as your adapter, successfully connected to the STM32 target processor and configured the Flash programmer.



SN: 58001025 USB#: 0

Device: J-Link ARM

HW : V8.00 dll : V4.20p

FW : Nov 19 2010 15:55:07

Port: SW Max Clock: 2MHz

	IDCODE	Device Name
SWD	● 0x1BA01477	ARM CoreSight SW-DP

Connect: Normal Reset: Autodetect

☑ Reset after Connect

☑ Cache Code
☑ Cache Memory

□ Verify Code Download
□ Download to Flash

● USB ○ TCP/IP

State: ready

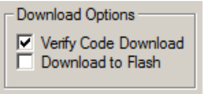
IP-Address 127 . 0 . 0 . 1 : Port (Auto: 0) 0

Configure the SWV Trace

This is done the same way as the ULINK2 or ULINK-ME. J-Link has an extra setting in the trace configuration window to store trace information on your hard drive called Use Cache File. Hover your mouse over this to get an explanation.

It is important with all Serial Wire Viewer configurations to choose as few signals as possible. The single wire SWO pin is easily overloaded.

TIP: It is easy to miss programming the Flash with your latest .axf executable. Select either the Verify Code Download in the Target/Debug/Settings as shown here or select Update Target before Debugging: ☑ Update Target before Debugging or make sure you program the Flash manually by clicking on the Load icon.



☑ Verify Code Download
□ Download to Flash

Part B)

1) *Blinky* Example Programs using a ULINK2 or ULINK-ME:

We will connect a Keil MDK development system using the STM3240G-EVAL and a ULINK2 or ULINK-ME. It is possible to use the ULINK*pro* for this example but you must configure it as in 3) ULINK*pro* and µVision Configuration: on page 6. This project is pre-configured to use ULINK2.

1. Connect the ULINK2 as pictured on the first page to the JTAG connector CN14.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\ST\STM3240G-EVAL\Blinky\Blinky.uvproj. If this file is not included with your version of MDK, please visit www.keil.com/st.
4. Make sure "STM32F207 Flash" is selected.
   This is where you can create and select different target configurations such as to execute a program in RAM or Flash. If you want to run in RAM, select STM32F407 RAM.
   You then omit the Load step in number 7.

STM32F407 Flash

5. This project is configured by default to use either the ULINK2 or ULINK-ME.
6. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
7. Program the STM32 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
8. 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 or the simulator.
9. Click on the RUN icon. Note: you stop the program with the STOP icon.

The LEDs on the STM32 board will now blink at a rate determined by the setting of RV1.

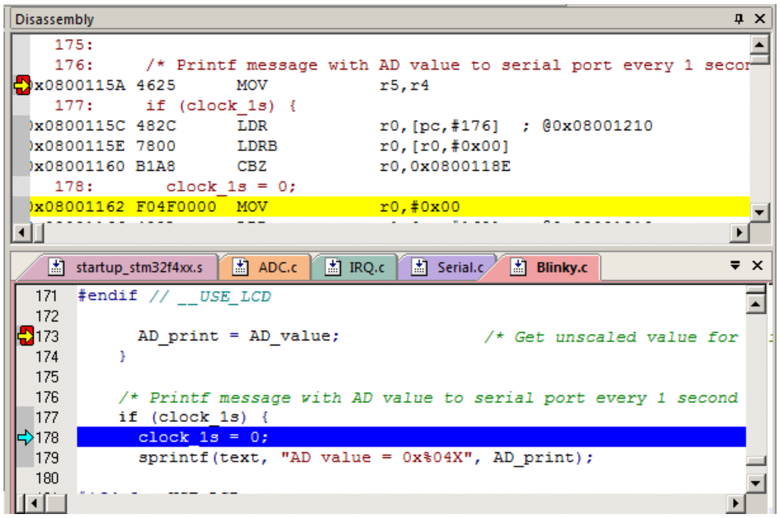
Rotate the potentiometer RV1. The LCD screen will display the value of Ad_value as shown below.

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

2) Hardware Breakpoints:

1. With Blinky running, double-click in the left margin on a darker gray block somewhere appropriate between Lines 162 through179 in the source file Blinky.c as shown below:
2. A red block is created and soon the program will stop at this point.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows.
4. The cyan arrow is a mouse selected pointer and is associated with the yellow band in the disassembly window. Click on a line in one window and this place will be indicated in the other window.
5. Note you can set and unset hardware breakpoints while the program is running. ARM CoreSight technology does this.
6. The STM32 has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to.

TIP: If you get multiple cyan arrows or can't understand the relationship between the C source and assembly, try lowering the compiler optimization to Level 0 and rebuilding your project. The level is set in Options for Target under the C/C++ tab.



```
   175:
   176:      /* Printf message with AD value to serial port every 1 secor
)x0800115A 4625      MOV           r5,r4
   177:      if (clock_1s) {
)x0800115C 482C      LDR           r0,[pc,#176]  ; @0x08001210
)x0800115E 7800      LDRB          r0,[r0,#0x00]
)x08001160 B1A8      CBZ           r0,0x0800118E
   178:        clock_1s = 0;
)x08001162 F04F0000  MOV           r0,#0x00
```

```
171  #endif // __USE_LCD
172
173      AD_print = AD_value;               /* Get unscaled value for
174    }
175
176    /* Printf message with AD value to serial port every 1 second
177    if (clock_1s) {
178      clock_1s = 0;
179      sprintf(text, "AD value = 0x%04X", AD_print);
180
```

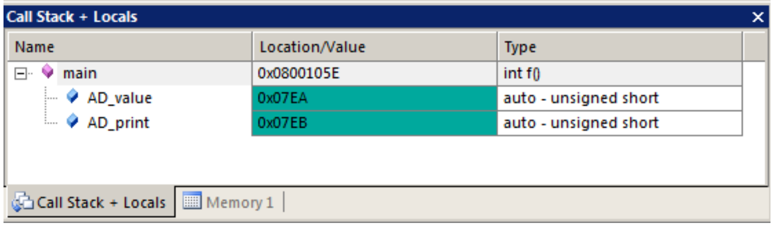
3) Call Stack + Locals Window:

Local Variables:

Starting with MDK 4.22 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 <out of scope> will be displayed.

1. Shown is the Locals window for the main function with the hardware breakpoint active from the previous page.
2. The contents of the local variables AD_value and AD_Print are displayed.
3. With the breakpoint set as in the previous page, as you click on RUN, these locals will update as appropriate.

TIP: This is standard "Stop and Go" debugging. ARM CoreSight debugging technology is 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. This is not possible with local variables.

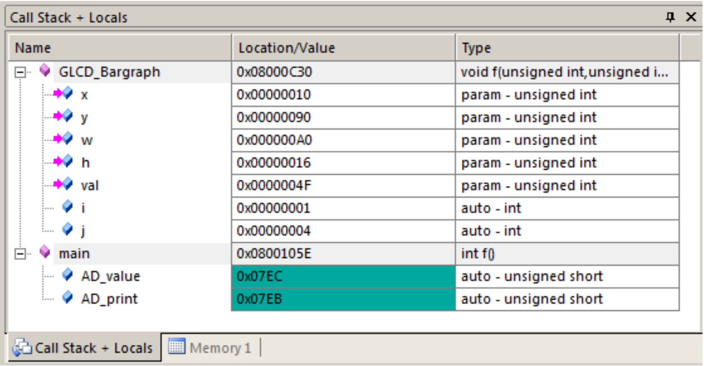


Name	Location/Value	Type
main	0x0800105E	int f()
AD_value	0x07EA	auto - unsigned short
AD_print	0x07EB	auto - unsigned short

Call Stack:

The list of called functions is displayed when the program is stopped. This is very useful for debugging when you need to know which functions have been called and are stored on the stack.

1. **Remove the hardware breakpoint by double-clicking on it.**
2. Click on RUN and then STOP
3. A window such as this one shows two functions and their local variables.
4. Each time you click on RUN and then STOP, this window will be updated or changed depending where the program happens to stop.



Name	Location/Value	Type
GLCD_Bargraph	0x08000C30	void f(unsigned int,unsigned i...
x	0x00000010	param - unsigned int
y	0x00000090	param - unsigned int
w	0x000000A0	param - unsigned int
h	0x00000016	param - unsigned int
val	0x0000004F	param - unsigned int
i	0x00000001	auto - int
j	0x00000004	auto - int
main	0x0800105E	int f()
AD_value	0x07EC	auto - unsigned short
AD_print	0x07EB	auto - unsigned short

4) Variables for Watch and Memory Windows:

It would be more useful if we could see the values of variables while the program is still running. Even more valuable would be the ability to change these values while the program is running. Even more valuable than that would be the ability to do this without any code stubs in your sources. CoreSight and µVision can do this and more as we shall soon see.

µVision can display global and static variables, structures and peripheral ports as well as physical memory addresses. It cannot display local variables which are constantly moving in and out of scope as their functions are called.

The locals must first be converted into static or global variables. This is easy to do by moving the variable out of all functions (including main) to create a global variable or by adding the static keyword in front of the variable declaration. An example is: `static int variable_name;`

1. This will ensure that variable always exists and is visible to µVision.
2. If you make this change, you must rebuild your project and Flash the processor again.

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

The next page describes how to enter variables in the Watch and Memory windows.

TIP: You will have to re-enter any variables you converted into a window after modifying it because it isn't the same variable anymore – it is a static variable or global now instead of a local. Drag 'n Drop is the fastest way as you will see on the next page.

TIP: Converting locals to static or global variables usually means the variable is now stored in volatile memory (RAM) rather than a CPU register. There will be a small time penalty incurred even if on-chip RAM is used..

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

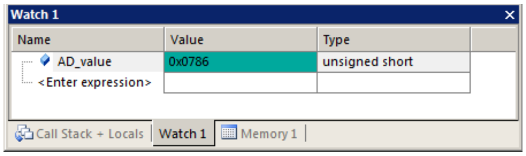
The Watch and memory windows will display updated variable values in real-time. It does this through the ARM CoreSight debugging technology. It is also possible to “put” or insert values into these memory locations in real-time. It is possible to “drag and drop” variables or enter physical addresses into windows or enter them manually while the program is running.

Watch window:

1. Stop the processor if running and exit debug mode.
2. Find the local variable AD_value in Blinky.c. This will be near line 140 at the start of the main function. Separate it from AD_print and change it to static as such:

```
unsigned short static AD_value = 0;
unsigned short AD_print = 0;
```

3. Rebuild the project. Program the Flash (Load) and re-enter debug mode. Click on RUN.
4. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
5. In Blinky.c, block AD_value, click and hold and drag it into Watch 1. Release it and it will be displayed updating as shown here:
6. Rotate the pot RV1 to see AD_value update in real-time.
7. You can also enter a variable manually by double-clicking and using copy and paste or typing the variable manually.



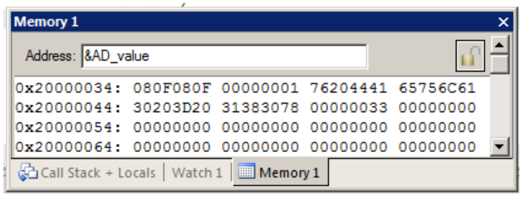
Name	Value	Type
AD_value	0x0786	unsigned short
<Enter expression>		

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.

6. Double click on the value for AD_value in the Watch window. Enter the value 0 and press Enter. 0 will be inserted into memory in real-time. It will quickly change as the variable is updated often by the program so you probably will not see this happen. You can also do this in the Memory window with a right-click and select Modify Memory.

Memory window:

1. Drag ‘n Drop AD_value into the Memory 1 window or enter it manually. Rotate the pot and watch the window.
2. Note the value of AD_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. Now the physical address is shown (0x2000_00034).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of AD_value is displayed as shown here:



Address: &AD_value

```
0x20000034: 080F080F 00000001 76204441 65756C61
0x20000044: 30203D20 31383078 00000033 00000000
0x20000054: 00000000 00000000 00000000 00000000
0x20000064: 00000000 00000000 00000000 00000000
```

TIP: You are able to configure the Watch and Memory windows and change their values while the program is still running in real-time without stealing any CPU cycles.

1. AD_value is now updated in real-time. This is ARM CoreSight technology working.
2. 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.

This is just a small example of the capabilities of Serial Wire Viewer. We will demonstrate more features..

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-M3and M4 are 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 to memory 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.

Remember you are not able to view local variables while the program is running. They are visible only when the program is stopped in their respective functions. You must change them to a different type of variable to see them update.

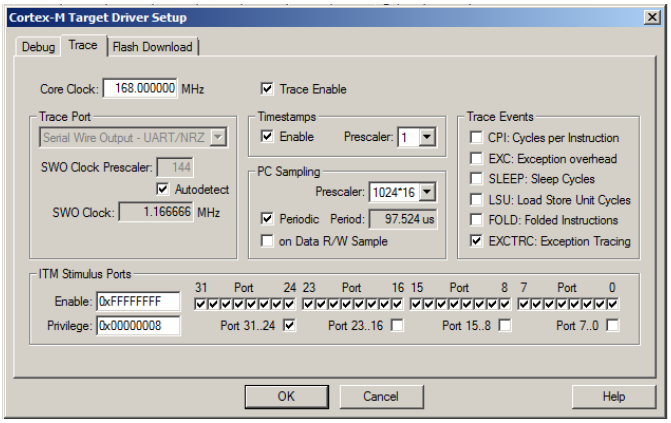
6) Configuring the Serial Wire Viewer (SWV):

Serial Wire Viewer provides program information in real-time.

A) SWV for ULINK2 or ULINK-ME: (ULINK*pro* instructions are on the next page)

Configure SWV:

1. µVision must be stopped and in edit mode (not debug mode).
2. Select Options for Target or ALT-F7 and select the Debug tab.
3. In the box Initialization File: enter **..\Blinky_ULp\STM32F4xx_SWO.ini** You can use the Browse button: ...
4. Click on Settings: beside the name of your adapter (ULINK Cortex Debugger) on the right side of the window.
5. Select the SWJ box and select SW in the Port: pulldown menu.
6. In the area **SW Device** must be displayed: ARM CoreSight SW-DP. SWV will not work with JTAG.
7. Click on the Trace tab. The window below is displayed.
8. In Core Clock: enter 168 and select the Trace Enable box.



Cortex-M Target Driver Setup

Core Clock: 168.000000 MHz ☑ Trace Enable

Trace Port: Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 144 ☑ Autodetect

SWO Clock: 1.166666 MHz

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16 ☑ Periodic Period: 97.524 us ☐ on Data R/W Sample

Trace Events:
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

ITM Stimulus Ports: 31 Port 24 23 Port 16 15 Port 8 7 Port 0

Enable: 0xFFFFFFFF ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑

Privilege: 0x00000008 Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

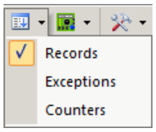
TIP: See Section 8) on page 32 for instructions on changing the CPU clock speed.

9. Select Periodic and leave everything else at default. Periodic activates PC Samples.
10. Click on OK twice to return to the main µVision menu. SWV is now configured.

Note: The ini file is set to SWV/SWO operation by default. You must edit it to use the Trace Port and ETM or select the file STM32F4xx_TP.ini. You can use the Configuration Wizard when you select Edit in the Debug tab.

To Display Trace Records:

1. Enter Debug mode.
2. Click on the RUN icon. .
3. Open Trace Records window by clicking on the small arrow beside the Trace icon:



4. The Race Records window will open and display PC Samples as shown below:
5. When you have completed this page, click on STOP.

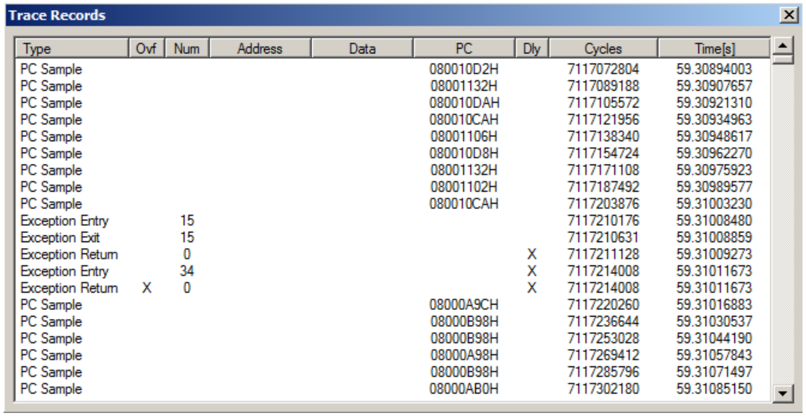
TIP: If you do not see PC Samples and Exceptions as shown and instead either nothing or frames with strange data, the trace is not configured correctly. The most probable cause is the Core Clock: frequency is wrong.

All frames have a timestamp displayed in CPU cycles and accumulated time.

Double-click this window to clear it.

If you right click inside this window you can see how to filter various types of frames out. Unselect PC Samples and you will see only exception frames displayed.

Did you know Exception 15 and 34 were being activated ? Now you do. This is a very useful tools for displaying how many times an exception is firing and when.



Trace Records

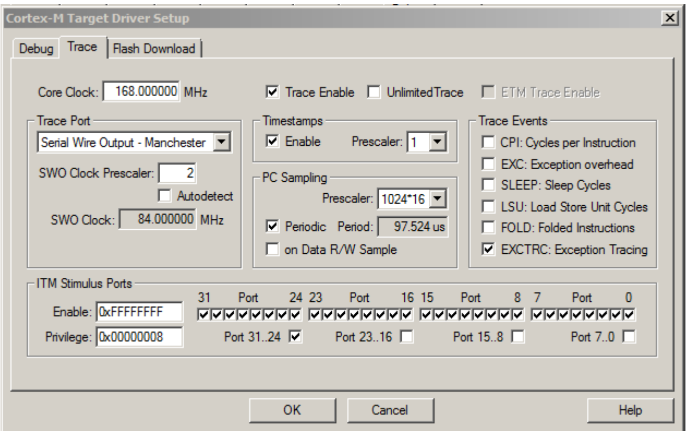
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					080010D2H		7117072804	59.30894003
PC Sample					08001132H		7117089188	59.30907657
PC Sample					080010DAH		7117105572	59.30921310
PC Sample					080010CAH		7117121956	59.30934963
PC Sample					08001106H		7117138340	59.30948617
PC Sample					080010D8H		7117154724	59.30962270
PC Sample					08001132H		7117171108	59.30975923
PC Sample					08001102H		7117187492	59.30989577
PC Sample					080010CAH		7117203876	59.31003230
Exception Entry		15					7117210176	59.31008480
Exception Exit		15					7117210631	59.31008859
Exception Return		0				X	7117211128	59.31009273
Exception Entry		34				X	7117214008	59.31011673
Exception Return	X	0				X	7117214008	59.31011673
PC Sample					08000A9CH		7117220260	59.31016883
PC Sample					08000B98H		7117236644	59.31030537
PC Sample					08000B98H		7117253028	59.31044190
PC Sample					08000A98H		7117269412	59.31057843
PC Sample					08000B98H		7117285796	59.31071497
PC Sample					08000AB0H		7117302180	59.31085150

TIP: SWV is easily overloaded as indicated by an “x” in the OVF or Dly column. Select only that information needed to reduce overloading. There are more useful features of Serial Wire Viewer as we shall soon discover.

B) SWV for ULINK*pro*:

Configure SWV: This uses the SWO output pin rather than the 4 bit Trace Port that is normally used with the ULINK*pro*.

1. µVision must be stopped and in edit mode (not debug mode) and with a ULINK*pro* connected to CN13 or CN14.
2. Select Project/Manage/Components, Environment, Books…
3. In Project Targets box select the Insert icon and enter ULINKpro and press Enter and then OK.
4. Select ULINKpro on the Select Target drop down box. ULINKpro Changes will not affect other targets.
5. Select Options for Target or ALT-F7 and select the Debug tab.
6. In the Use: box select ULINK Pro Cortex debugger. In the Utilities tab, select ULINK Pro Cortex debugger.
7. Select Settings: select Add and then select STM32F4xx Flash and Add. Click on OK once. Select the Debug tab.
8. In the box Initialization File: enter **..\Blinky_ULp\STM32F4xx_SWO.ini** You can use the Browse button: ...
9. Click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side.
10. Make sure SWJ and SW are selected. This exercise will not work with JTAG selected.
11. Click on the Trace tab. The window below is displayed.
12. Core Clock: ULINK*pro* uses this for calculating timing values. Enter 168 MHz. Select the Trace Enable box.
13. In the Trace Port select Serial Wire Output – Manchester. Selecting UART/NRZ will cause an error.
14. Unselect Autodetect. Enter 2 into SWO Clock Prescaler: as shown here.
15. Select Periodic and leave everything else at default. Selecting Periodic activates PC Samples.
16. Click on OK twice to return to the main µVision menu. SWV is now configured for the ULINK*pro*.
17. Rebuild this project and program the Flash with the Load icon. Select File/Save All.



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 168.000000 MHz ☑ Trace Enable ☐ UnlimitedTrace ☐ ETM Trace Enable

Trace Port: Serial Wire Output - Manchester
SWO Clock Prescaler: 2
☐ Autodetect
SWO Clock: 84.000000 MHz

Timestamps: ☑ Enable Prescaler: 1

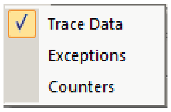
PC Sampling: Prescaler: 1024*16
☑ Periodic Period: 97.524 us
☐ on Data R/W Sample

Trace Events:
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

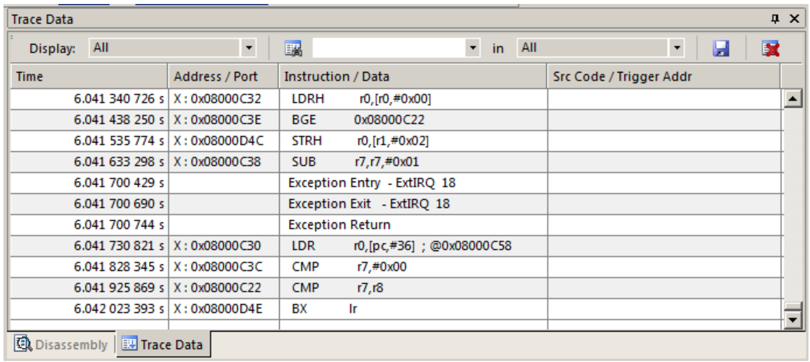
ITM Stimulus Ports
Enable: 0xFFFFFFFF 31 Port 24 ☑☑☑☑☑☑☑☑ 23 Port 16 ☑☑☑☑☑☑☑☑ 15 Port 8 ☑☑☑☑☑☑☑☑ 7 Port 0 ☑☑☑☑☑☑☑☑
Privilege: 0x00000008 Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

Display Trace Records:

1. Enter Debug mode.
2. Click on the RUN icon.
3. Open the Trace selection window by clicking on the small arrow beside the Trace icon:
4. Select Trace Data and the Trace Data window shown below will open.
5. Stop the processor and frames are displayed as shown below:
6. Select various types of frames with the Display: box to filter out various types of frames. The default is ALL.



TIP: The Trace Data window is different than the Trace Records window provided with the ULINK2. Trace Records display only SWV frames and Trace Data can display both SWV and ETM instruction frames. Note the disassembled instructions are displayed and if available, the source code is also displayed. If you want to see all the program counter values, use the ETM trace available with most STM32 processors. A Ulink*pro* using ETM trace will also provide Code Coverage, Performance Analysis and Execution Profiling in real time.



Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
6.041 340 726 s	X : 0x08000C32	LDRH r0,[r0,#0x00]	
6.041 438 250 s	X : 0x08000C3E	BGE 0x08000C22	
6.041 535 774 s	X : 0x08000D4C	STRH r0,[r1,#0x02]	
6.041 633 298 s	X : 0x08000C38	SUB r7,r7,#0x01	
6.041 700 429 s		Exception Entry - ExtIRQ 18	
6.041 700 690 s		Exception Exit - ExtIRQ 18	
6.041 700 744 s		Exception Return	
6.041 730 821 s	X : 0x08000C30	LDR r0,[pc,#36] ; @0x08000C58	
6.041 828 345 s	X : 0x08000C3C	CMP r7,#0x00	
6.041 925 869 s	X : 0x08000C22	CMP r7,r8	
6.042 023 393 s	X : 0x08000D4E	BX lr	

Clear and Save Trace Records:

You can clear the Trace Data window by clicking on the Clear icon.

You can also save the contents by clicking on the Save icon.

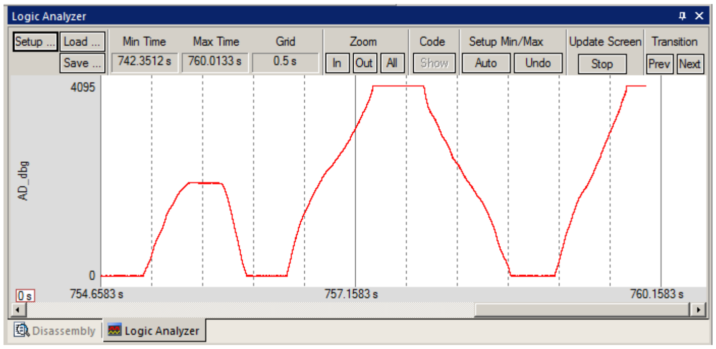
7) Using the Logic Analyzer (LA) with the ULINK2 or ULINK-ME:

This example will use the ULINK2 with the Blinky example. Please connect a ULINK2 or ULKINK-ME to your STM32 board and configure it for SWV trace. If you want to use a ULINK*pro* you will have to make appropriate modifications.

µVision has a graphical Logic Analyzer (LA) window. Up to four variables can be displayed in real-time using the Serial Wire Viewer. The Serial Wire Output pin is easily overloaded with many data reads and/or writes and data can be lost.

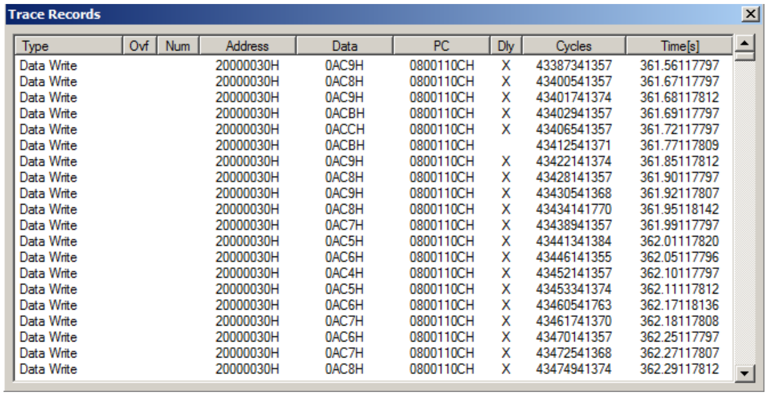
This exercise assumes AD_value is still a static variable as modified in Step 2 on page 11.

1. The project Blinky should still be open and is probably still in Debug mode. Click on STOP. Exit Debug mode.
2. Make sure STM32F407 Flash is selected. STM32F407 Flash
3. Create a global variable AD_dbg near line 40 in Blinky.c: **unsigned int AD_dbg:**
4. Near line 174 add this line, also in Blinky.c, just after AD_print = AD_value; **AD_dbg = AD_value;**
5. Rebuild the source files, program into Flash using the Load icon and enter debug mode.
6. Select Debug/Debug Settings and select the Trace tab.
7. Unselect Periodic and EXCTRC. This is to prevent overload on the SWO pin. Click OK twice.
8. Run the program. . **Note:** You can configure the LA while the program is running or stopped.
9. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.
10. Locate the variable AD_dbg you declared in Step 3 above in Blinky.c.
11. Block AD_dbg and drag it into the LA window and release it. Or click on Setup in the LA and enter it manually.
12. Click on Setup and set Max: in Display Range to 0xFFF. Click on Close. The LA is completely configured now.
13. Drag and drop AD_dbg into the Watch 1 window. Its value will now track the pot setting.
14. Adjust the Zoom OUT or the All icon in the LA window to provide a suitable scale of about 0.5 second.
15. Rotate the pot to obtain an interesting waveform as shown here:



TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: make them static or global. To see peripheral registers, enter them into the Logic Analyzer and read or write to them.

1. Select Debug/Debug Settings and select the Trace tab.
2. Select On Data R/W Sample. Click OK twice.
3. Run the program. .
4. Open the Trace Records window and clear it by double clicking in it.
5. The window similar below opens up:


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000030H	0AC9H	0800110CH	X	43387341357	361.56117797
Data Write			20000030H	0AC8H	0800110CH	X	43400541357	361.67117797
Data Write			20000030H	0AC9H	0800110CH	X	43401741374	361.68117812
Data Write			20000030H	0ACBH	0800110CH	X	43402941357	361.69117797
Data Write			20000030H	0ACCH	0800110CH	X	43406541357	361.72117797
Data Write			20000030H	0ACBH	0800110CH		43412541371	361.77117809
Data Write			20000030H	0AC9H	0800110CH	X	43422141374	361.85117812
Data Write			20000030H	0AC8H	0800110CH	X	43428141357	361.90117797
Data Write			20000030H	0AC9H	0800110CH	X	43430541368	361.92117807
Data Write			20000030H	0AC8H	0800110CH	X	43434141770	361.95118142
Data Write			20000030H	0AC7H	0800110CH	X	43438941357	361.99117797
Data Write			20000030H	0AC5H	0800110CH	X	43441341384	362.01117820
Data Write			20000030H	0AC6H	0800110CH	X	43446141355	362.05117796
Data Write			20000030H	0AC4H	0800110CH	X	43452141357	362.10117797
Data Write			20000030H	0AC5H	0800110CH	X	43453341374	362.11117812
Data Write			20000030H	0AC6H	0800110CH	X	43460541763	362.17118136
Data Write			20000030H	0AC7H	0800110CH	X	43461741370	362.18117808
Data Write			20000030H	0AC6H	0800110CH	X	43470141357	362.25117797
Data Write			20000030H	0AC7H	0800110CH	X	43472541368	362.27117807
Data Write			20000030H	0AC8H	0800110CH	X	43474941374	362.29117812

6. The first line below says:
   The instruction at 0x0800_110C caused a write of data 0x0AC9 to address 0x2000_0030 at the listed time in CPU Cycles or accumulated Time in seconds.

TIP: The PC column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin.

TIP: The ULINK*pro* will give a more sophisticated Trace Data window.

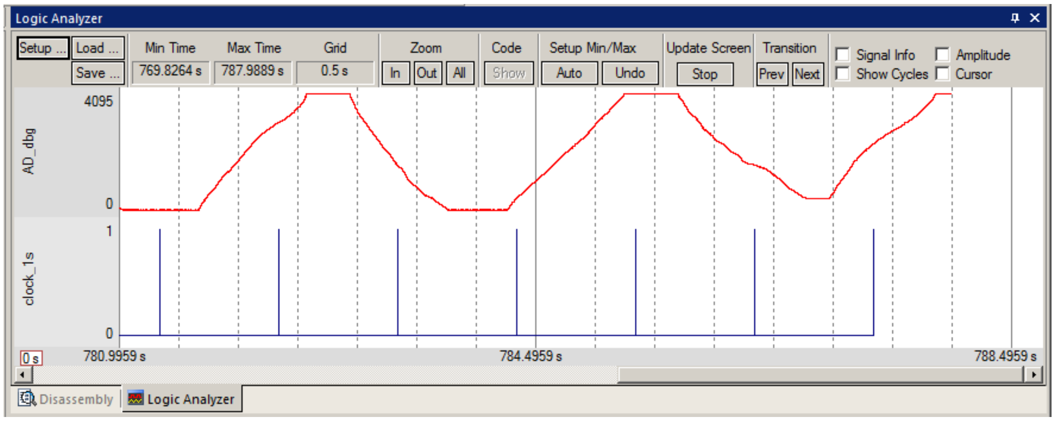
Watchpoints are described on the next page.

The next page also demonstrates another use of the Logic Analyzer.

Another Use for the Logic Analyzer:

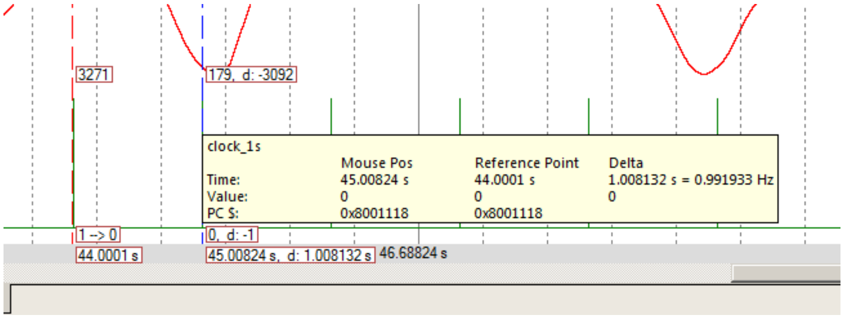
In Blinky.c, there is a global variable called clock_1s that is imported from IRQ.c. It is activated once a second for use as a timer. A question might arise: what is the duty cycle of this variable ? You could examine the code to determine this or instrumentate your code to send it out to a GPIO port to see with a scope…or put the variable in the Logic Analyzer.

1. Locate the global variable clock_1s in Blinky.c around Line 39.
2. Enter this into the Logic Analyzer: either manually or by dragging and dropping. You can do this while the program is running.
3. Open Setup and either set the Display Range: Max to 0x3 or select Display Type: to Bit. Click on Close.
4. Adjust the Zoom with IN, Out or All for a suitable display as shown below: You can easily see the duty cycle.



5. Stop the program. Select the Cursor checkbox.
6. Click on one of the clock_1s spikes and note a fixed red line is created.
7. A blue line follows the mouse and times are displayed as shown below.
8. Select Signal Info.
9. Hover the cursor for a few seconds and timing information is displayed as shown below in the yellow box.

TIP: The data writes to the variables listed in the LA will be visible in the Trace Records window.



clock_1s

	Mouse Pos	Reference Point	Delta
Time:	45.00824 s	44.0001 s	1.008132 s = 0.991933 Hz
Value:	0	0	0
PC $:	0x8001118	0x8001118	

Optional Exercise:

1. Create an unsigned int global variable in Blinky.c. An example is `unsigned int counter;`
2. In main() add `counter++;` as shown here near Line 182:

   clock_1s = 0;

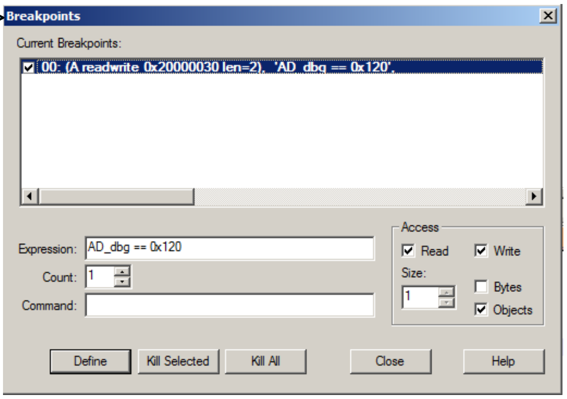
   counter++;

3. Exit Debug mode, rebuild, program the Flash (Load) and enter debug mode. Click on RUN.
4. Remove all variables from the Logic Analyzer window. You can use the Kill All icon.
5. Add counter to the Logic Analyzer window and set Max: to 0xFF. Click on RUN if necessary.
6. Add counter to the watch Window.
7. Counter will increment and will display as a ramp in the LA. Modify the value in the Watch window and see this change in real-time in the Logic Analyzer. You do not need to stop the processor to modify the value of counter.
8. Remove all variables from the Logic Analyzer window for the next exercise.

8) Watchpoints: Conditional Breakpoints

The STM32 Cortex-M4 processors have four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses watchpoints in its operations. This means in µVision you must have two variables free in the Logic Analyzer to use Watchpoints.

1. Using the example from the previous page, stop the program. Stay in Debug mode.
2. Click on Debug and select Breakpoints or press Ctrl-B.
3. The SWV Trace does not need to be configured to use Watchpoints. However, we will use it in this exercise.
4. In the Expression box enter: "**`AD_dbg`** == 0x120" without the quotes. Select both the Read and Write Access.
5. Click on Define and it will be accepted as shown here:
6. Click on Close.
7. Double-click in the Trace Records window to clear it.
8. Enter AD_dbg into the Logic Analyzer window.
9. Set its Max Display to 0xFFF. Click on Close.
10. Click on RUN.
11. Turn the pot to get AD_dbg to equal 0x120.
12. When **`AD_dbg`** equals 0x120, the program will stop. This is how a Watchpoint works.
13. You will see **`AD_dbg`** displayed as 0x120 in the Logic Analyzer as well as in the Watch window.



Current Breakpoints:

☑ 00: (A readwrite 0x20000030 len=2), 'AD_dbg == 0x120',

Expression: AD_dbg == 0x120

Count: 1

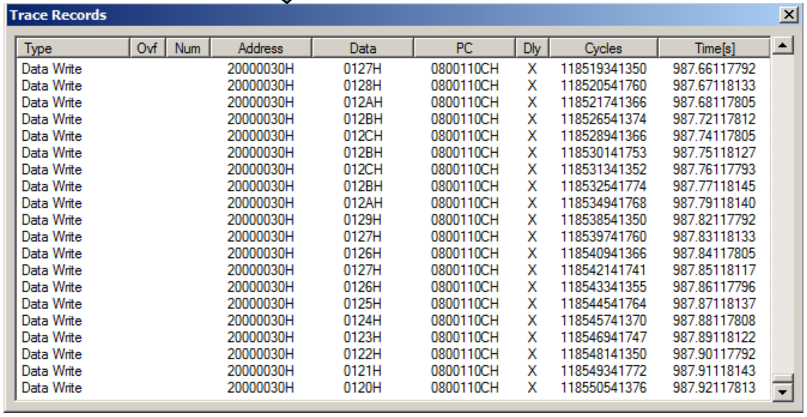
Command:

Access: ☑ Read ☑ Write

Size: 1 ☐ Bytes ☑ Objects

TIP: There will probably be a different value displayed in the LCD. This is because difference because of a delay when the LCD is updated. The trigger point represents the correct value and this will be displayed in the Watch window as well.

14. Note the data write of 0x120 in the Trace Records window shown below in the Data column. The address the data written to and the PC of the write instruction is displayed as well as the timestamps. This is with a ULINK2 or ULINK-ME. The ULINK*pro* will display a different window.
15. There are other types of expressions you can enter and they are detailed in the Help button in the Breakpoints window.
16. To repeat this exercise, click on RUN. If the program stops immediately, enter a different value in AD_dbg in Watch 1 window and try again.
17. **When finished, click on STOP and delete this Watchpoint by selecting Debug and select Breakpoints and select Kill All.**
18. Leave Debug mode.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000030H	0127H	0800110CH	X	118519341350	987.66117792
Data Write			20000030H	0128H	0800110CH	X	118520541760	987.67118133
Data Write			20000030H	012AH	0800110CH	X	118521741366	987.68117805
Data Write			20000030H	012BH	0800110CH	X	118526541374	987.72117812
Data Write			20000030H	012CH	0800110CH	X	118528941366	987.74117805
Data Write			20000030H	012BH	0800110CH	X	118530141753	987.75118127
Data Write			20000030H	012CH	0800110CH	X	118531341352	987.76117793
Data Write			20000030H	012BH	0800110CH	X	118532541774	987.77118145
Data Write			20000030H	012AH	0800110CH	X	118534941768	987.79118140
Data Write			20000030H	0129H	0800110CH	X	118538541350	987.82117792
Data Write			20000030H	0127H	0800110CH	X	118539741760	987.83118133
Data Write			20000030H	0126H	0800110CH	X	118540941366	987.84117805
Data Write			20000030H	0127H	0800110CH	X	118542141741	987.85118117
Data Write			20000030H	0126H	0800110CH	X	118543341355	987.86117796
Data Write			20000030H	0125H	0800110CH	X	118544541764	987.87118137
Data Write			20000030H	0124H	0800110CH	X	118545741370	987.88117808
Data Write			20000030H	0123H	0800110CH	X	118546941747	987.89118122
Data Write			20000030H	0122H	0800110CH	X	118548141350	987.90117792
Data Write			20000030H	0121H	0800110CH	X	118549341772	987.91118143
Data Write			20000030H	0120H	0800110CH	X	118550541376	987.92117813

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 in Current Breakpoints as shown above allows you to temporarily unselect or disable a Watchpoint without deleting it.

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

Keil provides RTX, a full feature RTOS. RTX is included for no charge as part of the Keil MDK full tool suite. It can have up to 255 tasks and no royalty payments are required. Source code is provided with all versions of MDK. This example explores the RTOS project. Keil will work with any RTOS. A RTOS is just a set of C functions that gets compiled with your project. A real-time awareness viewer for RTX is provided inside µVision.

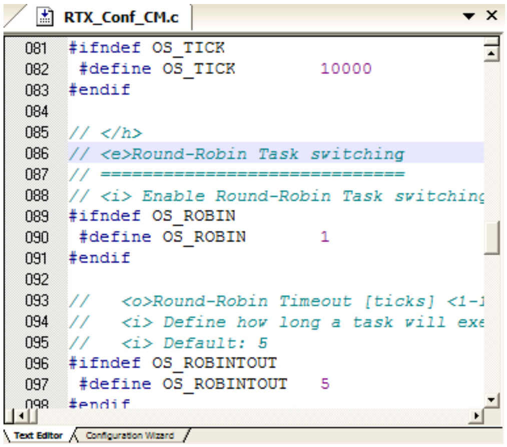
1. Start µVision by clicking on its icon on your desktop if it is not already running.
2. Select Project/Open Project and open C:\Keil\ARM\Boards\ST\STM3240G-EVAL\RTX_Blinky\Blinky.uvproj.
3. RTX_Blinky uses the ULINK2 as default: if you are using a ULINK*pro*, please configure it as described on page 3 and configure the Serial Wire Viewer as on page 13. You only have to do this once for each project – it will be saved in the project file by selecting File/Save All.
4. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
6. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
7. The LEDs and LCD will blink indicating the four waveforms of a stepper motor driver.
8. Click on STOP .

The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left below. You can open it with File/Open.
2. Click on Configuration Wizard 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. This is a great feature as it is much easier changing items here than in the source code.
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>.
8. The new µVision4 System Viewer windows are created in a similar fashion. Select View/System Viewer or click on the icon. The window similar to the on the far right opens.

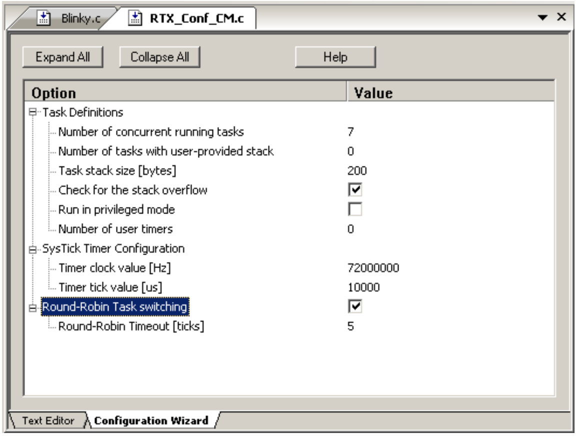
TIP: If you don't see any System Viewer entries, either the System Viewer is not available for your processor or you are using an older example project and it needs to be "refreshed" by the following instructions:

Exit Debug mode. Click on the Target Options icon and select the Device tab. Note which processor is currently selected. Select a different one, reselect the original processor and click on OK. System Viewer is now activated. Close this window and select File/Save All.



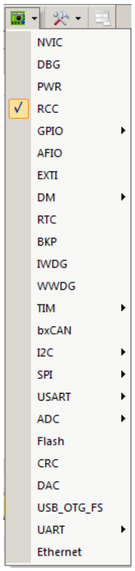
```
#ifndef OS_TICK
 #define OS_TICK         10000
#endif

// </h>
// <e>Round-Robin Task switching
// ================================
// <i> Enable Round-Robin Task switching
#ifndef OS_ROBIN
 #define OS_ROBIN        1
#endif

//   <o>Round-Robin Timeout [ticks] <1-1
//   <i> Define how long a task will exe
//   <i> Default: 5
#ifndef OS_ROBINTOUT
 #define OS_ROBINTOUT    5
#endif
```

Text Editor: Source Code



Option	Value
Task Definitions	
Number of concurrent running tasks	7
Number of tasks with user-provided stack	0
Task stack size [bytes]	200
Check for the stack overflow	☑
Run in privileged mode	☐
Number of user timers	0
SysTick Timer Configuration	
Timer clock value [Hz]	72000000
Timer tick value [us]	10000
Round-Robin Task switching	☑
Round-Robin Timeout [ticks]	5

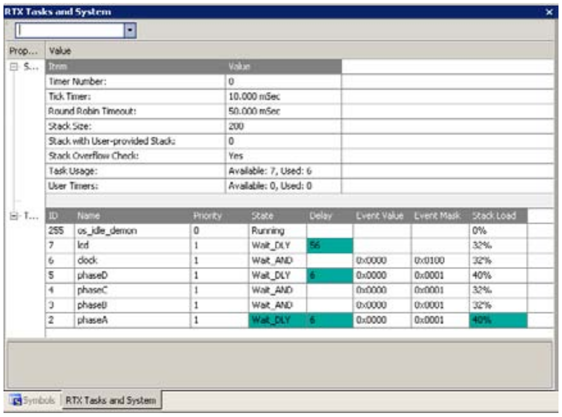
Configuration Wizard



System Viewer

10) 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.



RTX Tasks and System

Item	Value
Timer Number:	0
Tick Timer:	10.000 mSec
Round Robin Timeout:	50.000 mSec
Stack Size:	200
Stack with User-provided Stack:	0
Stack Overflow Check:	Yes
Task Usage:	Available: 7, Used: 6
User Timers:	Available: 0, Used: 0

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
7	lcd	1	Wait_DLY	56			32%
6	clock	1	Wait_AND		0x0000	0x0100	32%
5	phaseD	1	Wait_DLY	6	0x0000	0x0001	40%
4	phaseC	1	Wait_AND		0x0000	0x0001	32%
3	phaseB	1	Wait_AND		0x0000	0x0001	32%
2	phaseA	1	Wait_DLY	6	0x0000	0x0001	40%

1. Run RTX_Blinky again by clicking on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System 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.

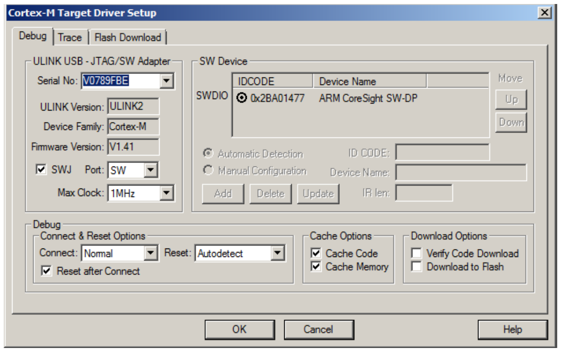
Important TIP: View/Periodic Window Update must be selected !

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.
3. Select the Debug tab. In the box Initialization File: enter **.\Blinky_ULp\STM32F4xx_SWO.ini** or use the Browse **...**
4. Click the Settings box next to ULINK Cortex Debugger.
5. In the Debug window as shown here, make sure SWJ is checked and Port: is set to SW and not JTAG.
6. Click on the Trace tab to open the Trace window.
7. Set Core Clock: to 168 MHz and select Trace Enable.
8. Unselect the Periodic and EXCTRC boxes as shown here:
9. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer. It is slightly intrusive.
10. Click on OK twice to return to µVision.
    The Serial Wire Viewer is now configured in µVision.
11. Enter Debug mode and click on RUN to start the program.
12. Select "Tasks and System" tab: note the display is updated.
13. Click on the Event Viewer tab.
14. This window displays task events in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 1 seconds by clicking on the ALL and then the + and – icons.



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

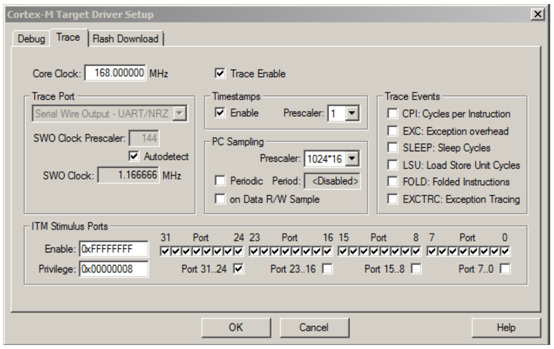
ULINK USB - JTAG/SW Adapter
Serial No: V0789FBE
ULINK Version: ULINK2
Device Family: Cortex-M
Firmware Version: V1.41
☑ SWJ Port: SW
Max Clock: 1MHz

SW Device

	IDCODE	Device Name
SWDIO	● 0x2BA01477	ARM CoreSight SW-DP

● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
Add Delete Update IR len:

Debug
Connect & Reset Options: Connect: Normal Reset: Autodetect
☑ Reset after Connect
Cache Options: ☑ Cache Code ☑ Cache Memory
Download Options: □ Verify Code Download □ Download to Flash



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 168.000000 MHz ☑ Trace Enable

Trace Port: Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 144 ☑ Autodetect
SWO Clock: 1.166666 MHz

Timestamps: ☑ Enable Prescaler: 1

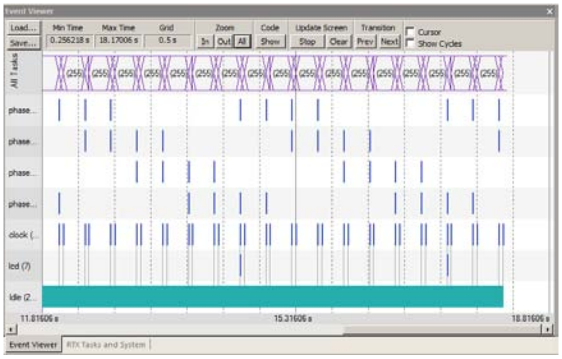
PC Sampling: Prescaler: 1024*16
□ Periodic Period: <Disabled>
□ on Data R/W Sample

Trace Events
□ CPI: Cycles per Instruction
□ EXC: Exception overhead
□ SLEEP: Sleep Cycles
□ LSU: Load Store Unit Cycles
□ FOLD: Folded Instructions
□ EXCTRC: Exception Tracing

ITM Stimulus Ports
Enable: 0xFFFFFFFF 31 Port 24 ☑☑☑☑☑☑☑☑ 23 Port 16 ☑☑☑☑☑☑☑☑ 15 Port 8 ☑☑☑☑☑☑☑☑ 7 Port 0 ☑☑☑☑☑☑☑☑
Privilege: 0x00000008 Port 31..24 ☑ Port 23..16 □ Port 15..8 □ Port 7..0 □

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 ?

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.



Event Viewer

Min Time 0.256218 s Max Time 18.17006 s Grid 0.5 s

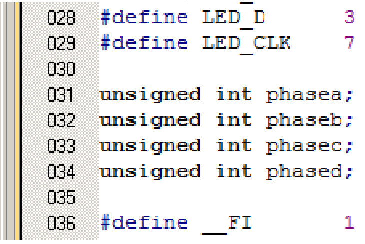
The data is updated while the program is running. No instrumentation code needs to be inserted into your source. You will find this feature very useful ! Remember, RTX with source code is included with all versions of MDK.

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

11) 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.

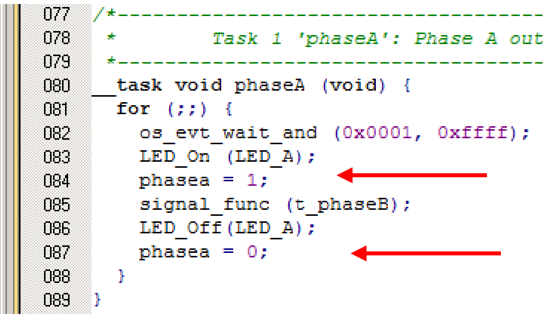
1. Close the RTX Viewer windows. Stop the program and exit debug mode.
2. **Add 4 global variables `unsigned int phasea` through `unsigned int phased` to Blinky.c as shown here:**



```
028 #define LED_D       3
029 #define LED_CLK     7
030
031 unsigned int phasea;
032 unsigned int phaseb;
033 unsigned int phasec;
034 unsigned int phased;
035
036 #define __FI        1
```

3. Add 2 lines to each of the four tasks Task1 through Task4 in Blinky.c as shown below: **phasea=1;** and **phasea=0;** :the first two lines are shown added at lines 084 and 087 (just after LED_On and LED_Off function calls). For each of the four tasks, add the corresponding variable assignment statements phasea, phaseb, phasec and phased.
4. We do this because in this simple program there are not enough suitable variables to connect to the Logic Analyzer.

TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

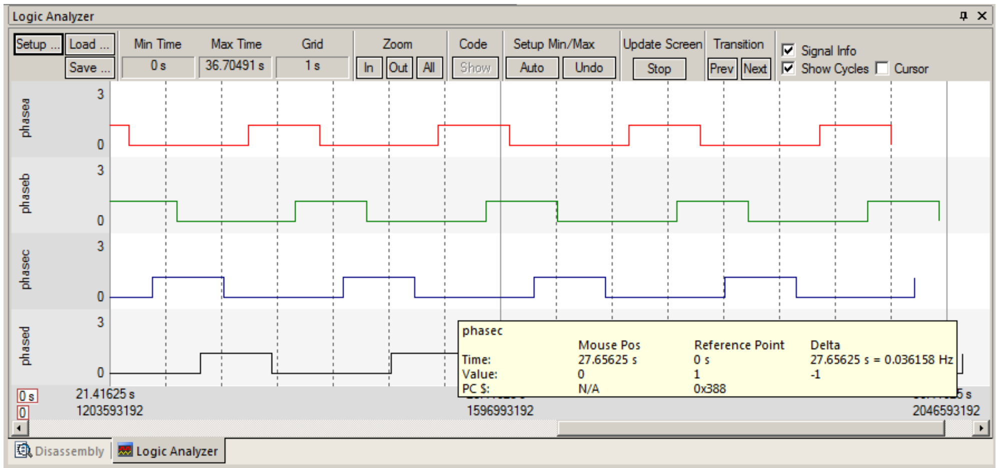


```
077 /*----------------------------------------
078  *        Task 1 'phaseA': Phase A out
079  *----------------------------------------
080 __task void phaseA (void) {
081   for (;;) {
082     os_evt_wait_and (0x0001, 0xffff);
083     LED_On (LED_A);
084     phasea = 1;
085     signal_func (t_phaseB);
086     LED_Off(LED_A);
087     phasea = 0;
088   }
089 }
```

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.

Enter the Variables into the Logic Analyzer:

9. Click on the Blinky.c tab. Block `phasea`, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
10. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release.
11. 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.
12. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
13. Click on Close to go back to the LA window.
14. Using the All, OUT and In buttons set the range to 1second or so. Move the scrolling bar to the far right if needed.
15. 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:



Min Time: 0 s
Max Time: 36.70491 s
Grid: 1 s
☑ Signal Info
☑ Show Cycles ☐ Cursor

0 s 21.41625 s
0 1203593192 1596993192 2046593192

```
phasec
        Mouse Pos     Reference Point   Delta
Time:   27.65625 s    0 s               27.65625 s = 0.036158 Hz
Value:  0             1                 -1
PC $:   N/A           0x388
```

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.

12) Serial Wire Viewer (SWV) and how to use it: (with ULINK2)

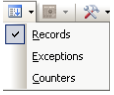
a) Data Reads and Writes: (Note: Data Reads but not Writes are disabled in the current version of µVision).

You have configured Serial Wire Viewer (SWV) two pages back in Page 12 under **Configuring the Serial Wire Viewer:**

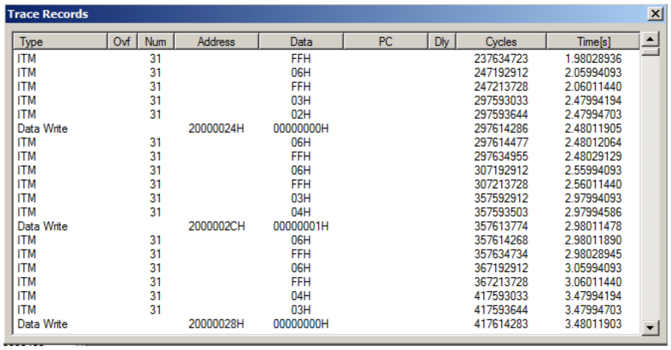
Now we will examine some of the features available to you. SWV works with µVision and a ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

Everything shown here is done without stealing any CPU cycles and is completely non-intrusive. A user program runs at full speed and needs no code stubs or instrumentation software added to your programs.

1. Use RTX_Blinky from the last 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.

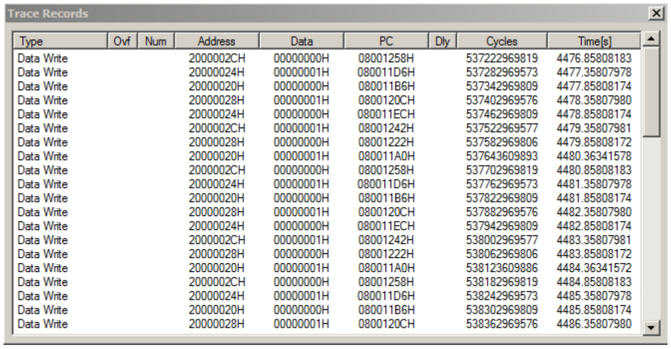


3. The Trace Records window will open up as shown here:


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		31		FFH			237634723	1.98028936
ITM		31		06H			247192912	2.05994093
ITM		31		FFH			247213728	2.06011440
ITM		31		03H			297593033	2.47994194
ITM		31		02H			297593644	2.47994703
Data Write			20000024H	00000000H			297614286	2.48011905
ITM		31		06H			297614477	2.48012064
ITM		31		FFH			297634955	2.48029129
ITM		31		06H			307192912	2.55994093
ITM		31		FFH			307213728	2.56011440
ITM		31		03H			357592912	2.97994093
ITM		31		04H			357593503	2.97994586
Data Write			2000002CH	00000001H			357613774	2.98011478
ITM		31		06H			357614268	2.98011890
ITM		31		FFH			357634734	2.98028945
ITM		31		06H			367192912	3.05994093
ITM		31		FFH			367213728	3.06011440
ITM		31		04H			417593033	3.47994194
ITM		31		03H			417593644	3.47994703
Data Write			20000028H	00000000H			417614283	3.48011903

4. The ITM frames are the data from the RTX Kernel Viewer which uses Port 31 as shown under Num. To turn this off select Debug/Debug Settings and click on the Trace tab. Unselect ITM Stim. Port 31. Or you can right click in the trace records window and unselect ITM frames to filter them out.
 TIP: Port 0 is used for Debug `printf` Viewer.
5. Unselect EXCTRC and Periodic.
6. Select On Data R/W Sample.
7. Click on OK to return.
8. Click on the RUN icon.
9. Double-click anywhere in the Trace records window to clear it.
10. Only Data Writes will appear now.
 TIP: You could have also right clicked on the Trace Records window to filter the ITM frames out but this will not reduce any SWO pin overloads.


Trace Records

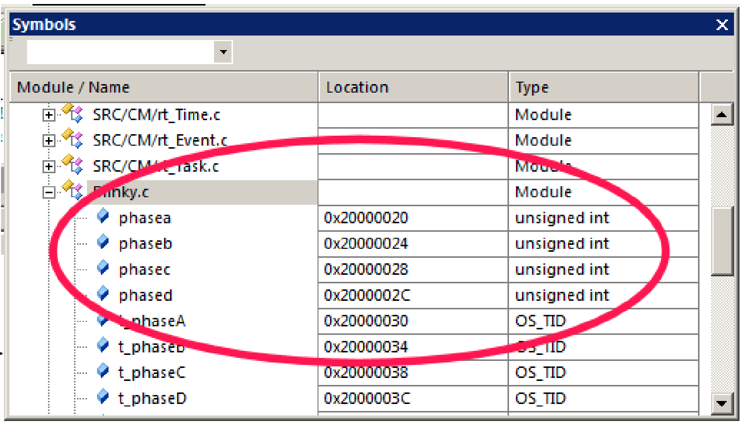
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			2000002CH	00000000H	08001258H		537222969819	4476.85808183
Data Write			20000024H	00000001H	080011D6H		537282969573	4477.35807978
Data Write			20000020H	00000000H	080011B6H		537342969809	4477.85808174
Data Write			20000028H	00000001H	0800120CH		537402969576	4478.35807980
Data Write			20000024H	00000000H	080011ECH		537462969809	4478.85808174
Data Write			2000002CH	00000001H	08001242H		537522969577	4479.35807981
Data Write			20000028H	00000000H	08001222H		537582969806	4479.85808172
Data Write			20000020H	00000001H	080011A0H		537643609893	4480.36341578
Data Write			2000002CH	00000000H	08001258H		537702969819	4480.85808183
Data Write			20000024H	00000001H	080011D6H		537762969573	4481.35807978
Data Write			20000020H	00000000H	080011B6H		537822969809	4481.85808174
Data Write			20000028H	00000001H	0800120CH		537882969576	4482.35807980
Data Write			20000024H	00000000H	080011ECH		537942969809	4482.85808174
Data Write			2000002CH	00000001H	08001242H		538002969577	4483.35807981
Data Write			20000028H	00000000H	08001222H		538062969806	4483.85808172
Data Write			20000020H	00000001H	080011A0H		538123609886	4484.36341572
Data Write			2000002CH	00000000H	08001258H		538182969819	4484.85808183
Data Write			20000024H	00000001H	080011D6H		538242969573	4485.35807978
Data Write			20000020H	00000000H	080011B6H		538302969809	4485.85808174
Data Write			20000028H	00000001H	0800120CH		538362969576	4486.35807980

What is happening here ?

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

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

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


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	0x20000020	unsigned int
phaseb	0x20000024	unsigned int
phasec	0x20000028	unsigned int
phased	0x2000002C	unsigned int
t_phaseA	0x20000030	OS_TID
t_phaseB	0x20000034	OS_TID
t_phaseC	0x20000038	OS_TID
t_phaseD	0x2000003C	OS_TID

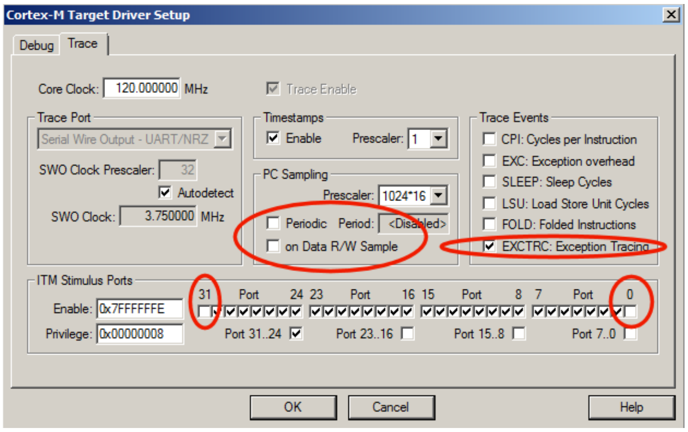
Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

TIP: ULINK*pro* and the Segger J-Link adapters display the trace frames in a different style trace window.

b) Exceptions and Interrupts:

The STM32 family using Cortex-M3 or M4 processors has many interrupts and it can be difficult to determine when they are being activated and when. Serial Wire Viewer (SWV) makes the display of exceptions and interrupts easy.

1. 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. The Trace Records should still be open. Double click on it to clear it.
6. Click RUN to start the program.
7. You will see a window similar to the one below with Exceptions frames displayed.



Cortex-M Target Driver Setup

Debug | Trace

Core Clock: 120.000000 MHz ☑ Trace Enable

Trace Port: Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 32 ☑ Autodetect

SWO Clock: 3.750000 MHz

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16

☐ Periodic Period: <Disabled>

☐ on Data R/W Sample

Trace Events:

☐ CPI: Cycles per Instruction

☐ EXC: Exception overhead

☐ SLEEP: Sleep Cycles

☐ LSU: Load Store Unit Cycles

☐ FOLD: Folded Instructions

☑ EXCTRC: Exception Tracing

ITM Stimulus Ports

Enable: 0x7FFFFFFE

Privilege: 0x00000008

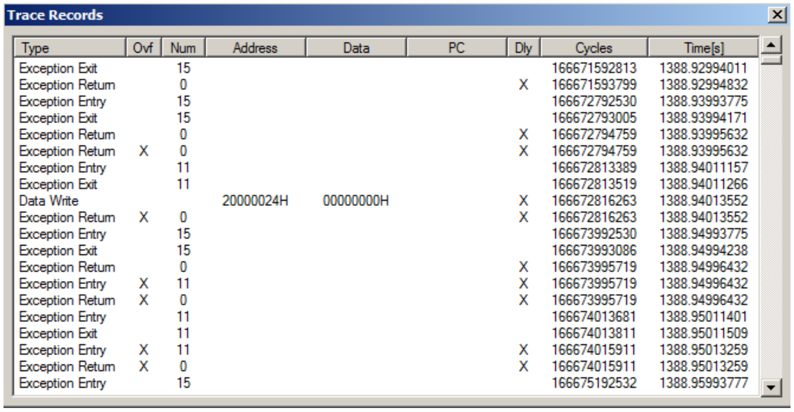
31 ☐ ☑☑☑☑☑☑☑ 24 23 ☑☑☑☑☑☑☑☑ 16 15 ☑☑☑☑☑☑☑☑ 8 7 ☑☑☑☑☑☑☑ ☐ 0

Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

What Is Happening ?

1. You can see two exceptions (11 & 15) happening.

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned. This is useful to detect tail-chaining.

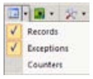


Trace Records

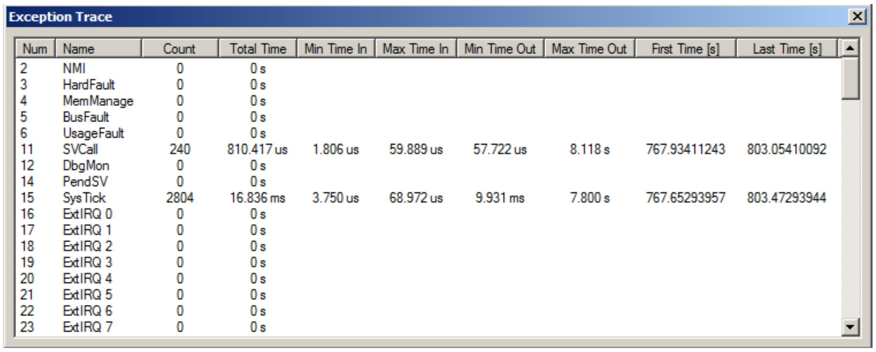
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Exit		15					166671592813	1388.92994011
Exception Return		0				X	166671593799	1388.92994832
Exception Entry		15					166672792530	1388.93993775
Exception Exit		15					166672793005	1388.93994171
Exception Return		0				X	166672794759	1388.93995632
Exception Return	X	0				X	166672794759	1388.93995632
Exception Entry		11					166672813389	1388.94011157
Exception Exit		11					166672813519	1388.94011266
Data Write			20000024H	00000000H		X	166672816263	1388.94013552
Exception Return	X	0				X	166672816263	1388.94013552
Exception Entry		15					166673992530	1388.94993775
Exception Exit		15					166673993086	1388.94994238
Exception Return		0				X	166673995719	1388.94996432
Exception Entry	X	11				X	166673995719	1388.94996432
Exception Return	X	0				X	166673995719	1388.94996432
Exception Entry		11					166674013681	1388.95011401
Exception Exit		11					166674013811	1388.95011509
Exception Entry	X	11				X	166674015911	1388.95013259
Exception Return	X	0				X	166674015911	1388.95013259
Exception Entry		15					166675192532	1388.95993777

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.

TIP: The SWO pin is one pin on the Cortex-M3 family processors that all SWV information is fed out. The exception is the ULINK*pro* which can also send SWV out the 4 bit Trace Port. There are limitations on how much information we can feed out this one pin. These exceptions are happening at a very fast rate. Overloads are gracefully handled.



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.
3. Note the number of times these have happened under Count. This is very useful information in case interrupts come at rates different from what you expect.
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 CPU cycles !



Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	240	810.417 us	1.806 us	59.889 us	57.722 us	8.118 s	767.93411243	803.05410092
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	2804	16.836 ms	3.750 us	68.972 us	9.931 ms	7.800 s	767.65293957	803.47293944
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						

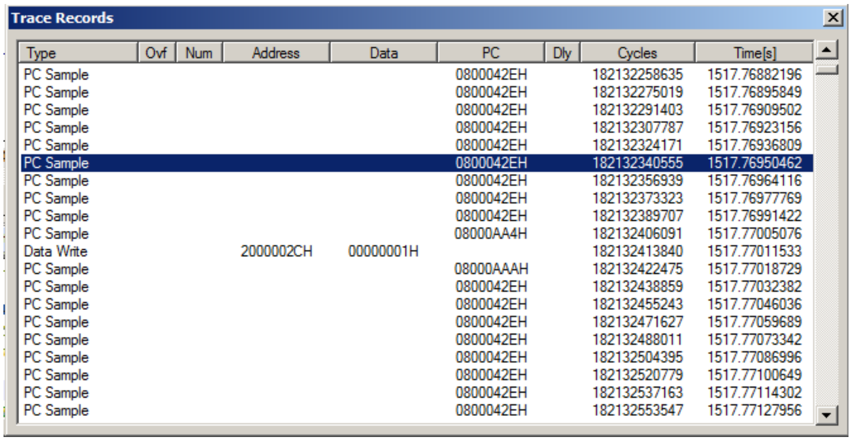
TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

c) PC Samples:

Serial Wire Viewer can display a sampling of the program counter. If you need to see all the PC values, use the ETM trace with a Keil ULINK*pro*. ETM trace also provides Code Coverage, Execution Profiling and Performance Analysis.

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.

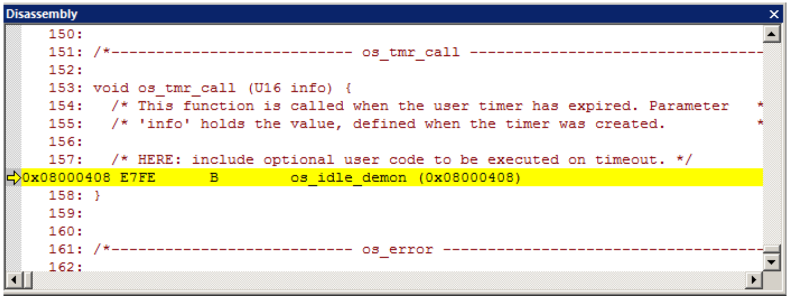
1. Open Debug/Debug Settings and select the Trace tab.
2. Unselect EXCTRC, On Data R/W Sample and select Periodic in the PC Sampling area.
3. Click on OK twice to return to the main screen.
4. Close the Exception Trace window and leave Trace Records open. Double-click to clear it.
5. Click on RUN and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



Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					0800042EH		182132258635	1517.76882196
PC Sample					0800042EH		182132275019	1517.76895849
PC Sample					0800042EH		182132291403	1517.76909502
PC Sample					0800042EH		182132307787	1517.76923156
PC Sample					0800042EH		182132324171	1517.76936809
PC Sample					0800042EH		182132340555	1517.76950462
PC Sample					0800042EH		182132356939	1517.76964116
PC Sample					0800042EH		182132373323	1517.76977769
PC Sample					0800042EH		182132389707	1517.76991422
PC Sample					08000AA4H		182132406091	1517.77005076
Data Write			2000002CH	00000001H			182132413840	1517.77011533
PC Sample					08000AAAH		182132422475	1517.77018729
PC Sample					0800042EH		182132438859	1517.77032382
PC Sample					0800042EH		182132455243	1517.77046036
PC Sample					0800042EH		182132471627	1517.77059689
PC Sample					0800042EH		182132488011	1517.77073342
PC Sample					0800042EH		182132504395	1517.77086996
PC Sample					0800042EH		182132520779	1517.77100649
PC Sample					0800042EH		182132537163	1517.77114302
PC Sample					0800042EH		182132553547	1517.77127956

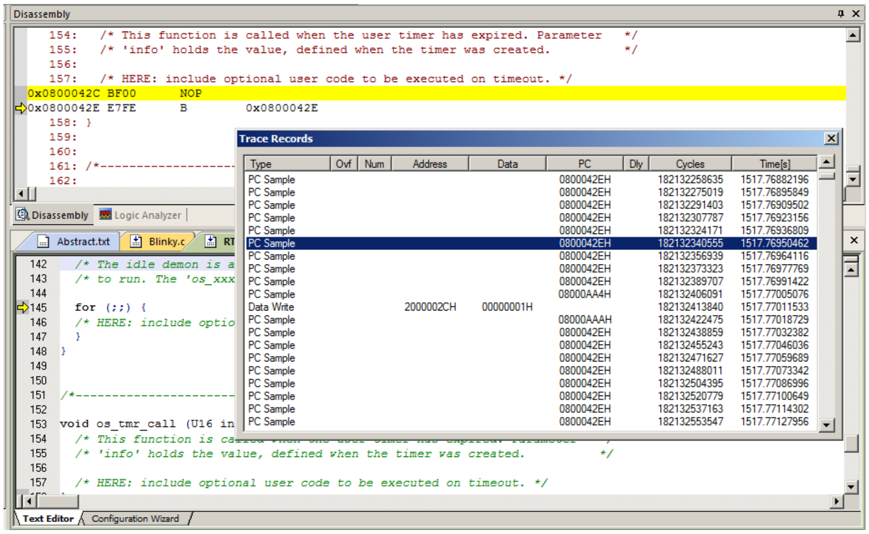
6. Most of the PC Samples are 0x0800_042E which is a branch to itself in a loop forever routine.
7. Stop the program and the Disassembly window will show this Branch:



Disassembly

```
   150: 
   151: /*---------------------------- os_tmr_call ----------------------------------
   152: 
   153: void os_tmr_call (U16 info) {
   154:   /* This function is called when the user timer has expired. Parameter   *
   155:   /* 'info' holds the value, defined when the timer was created.          *
   156: 
   157:   /* HERE: include optional user code to be executed on timeout. */
0x08000408 E7FE      B         os_idle_demon (0x08000408)
   158: }
   159: 
   160: 
   161: /*---------------------------- os_error -------------------------------------
   162: 
```

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 caught in a tight loop like in this case.
9. **Note:** you can get different PC values depending on the optimization level set in µVision.
10. Set a breakpoint in one of the tasks in Blinky.c.
11. Run the program and when the breakpoint is hit, the program and trace collection is stopped.
12. Scroll to the bottom of the Trace Records window and you might see the correct PC value displayed. Usually, it will be a different PC depending on when the sampling took place.
13. Remove the breakpoint for the next step.



Disassembly

```
   154:   /* This function is called when the user timer has expired. Parameter   */
   155:   /* 'info' holds the value, defined when the timer was created.          */
   156: 
   157:   /* HERE: include optional user code to be executed on timeout. */
0x0800042C BF00      NOP
0x0800042E E7FE      B         0x0800042E
   158: }
   159: 
   160: 
   161: /*--------------------
   162: 
```

```
  /* The idle demon is a
  /* to run. The 'os_xxx

  for (;;) {
  /* HERE: include optio
  }
}


/*---------------------

void os_tmr_call (U16 in
  /* This function is called when the user timer has expired. Parameter
  /* 'info' holds the value, defined when the timer was created.          */

  /* HERE: include optional user code to be executed on timeout. */
```

Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					0800042EH		182132258635	1517.76882196
PC Sample					0800042EH		182132275019	1517.76895849
PC Sample					0800042EH		182132291403	1517.76909502
PC Sample					0800042EH		182132307787	1517.76923156
PC Sample					0800042EH		182132324171	1517.76936809
PC Sample					0800042EH		182132340555	1517.76950462
PC Sample					0800042EH		182132356939	1517.76964116
PC Sample					0800042EH		182132373323	1517.76977769
PC Sample					0800042EH		182132389707	1517.76991422
PC Sample					08000AA4H		182132406091	1517.77005076
Data Write			2000002CH	00000001H			182132413840	1517.77011533
PC Sample					08000AAAH		182132422475	1517.77018729
PC Sample					0800042EH		182132438859	1517.77032382
PC Sample					0800042EH		182132455243	1517.77046036
PC Sample					0800042EH		182132471627	1517.77059689
PC Sample					0800042EH		182132488011	1517.77073342
PC Sample					0800042EH		182132504395	1517.77086996
PC Sample					0800042EH		182132520779	1517.77100649
PC Sample					0800042EH		182132537163	1517.77114302
PC Sample					0800042EH		182132553547	1517.77127956

TIP: In order to see all the program Counter values, use ETM trace with the ULINK*pro*. Most STM32 processors have ETM.

ETM is much superior for program flow debugging than PC Samples.

µVision with a ULINK*pro* uses ETM to provide Code Coverage, Execution Profiling and Performance Analysis.

13) ITM (Instruction Trace Macrocell) a `printf` Feature:

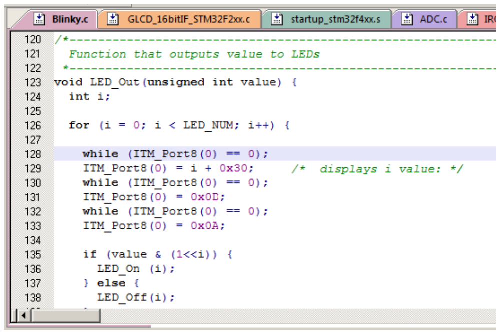
Recall that we showed you can display information about the RTOS in real-time using the RTX Viewer. This is done through ITM Stimulus Port 31. ITM Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into µVision for display in the Debug (printf) Viewer window.

1. Open the project Blinky.uvproj (not RTX Blinky).
2. Add this #define to Blinky.c. A good place is near line 34, just after the declaration of **`char text[40];`**.

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

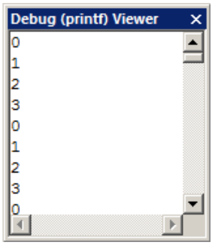
3. In the function LED_Out in Blinky.c, enter these lines starting at near line 128:

```
while (ITM_Port8(0) == 0);
ITM_Port8(0) = i + 0x30; /*  displays i value: */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```



```
/*----------------------------------------------------------------------------
  Function that outputs value to LEDs
 *----------------------------------------------------------------------------
void LED_Out(unsigned int value) {
  int i;

  for (i = 0; i < LED_NUM; i++) {

    while (ITM_Port8(0) == 0);
    ITM_Port8(0) = i + 0x30;     /*  displays i value: */
    while (ITM_Port8(0) == 0);
    ITM_Port8(0) = 0x0D;
    while (ITM_Port8(0) == 0);
    ITM_Port8(0) = 0x0A;

    if (value & (1<<i)) {
      LED_On (i);
    } else {
      LED_Off(i);
```

4. Rebuild the source files, program the Flash memory and enter debug mode. Select File/Save All.
5. Open Debug/Debug Settings and select the Trace tab.
6. Unselect On Data R/W Sample, Periodic and ITM Port 31. (this is to help not overload the SWO port)
7. Select EXCTRC and ITM Port 0. ITM Stimulus Port "0" enables the Debug (prinftf) Viewer.
8. Click OK twice.
9. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
10. In the Debug (printf) Viewer you will see the ASCII value of **`i`** appear.



```
0
1
2
3
0
1
2
3
0
```

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. You may have to scroll to see them.

What Is This ?

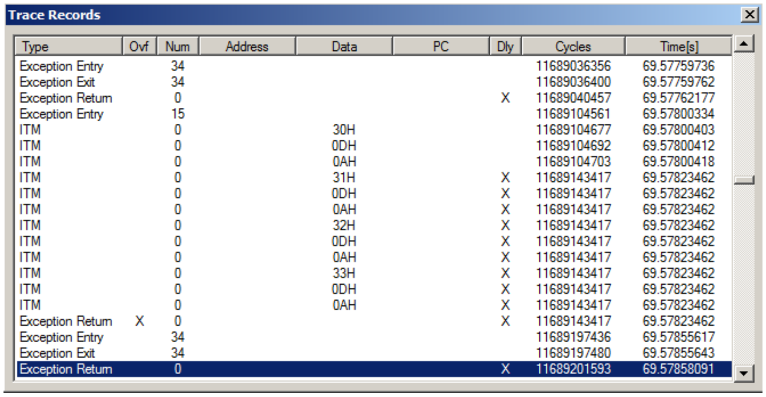
1. ITM 0 frames (Num column) are our ASCII characters from **`num`** with carriage return (0D) and line feed (0A) as displayed the Data column.
2. All these are timestamped in both CPU cycles and time in seconds.
3. Note the "X" in the Dly column. This means the timestamps might/are not be correct due to SWO pin overload.

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 STM32 processor via the Serial Wire Output pin.

This is much faster than using a UART and none of your peripherals are used.

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.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		34					11689036356	69.57759736
Exception Exit		34					11689036400	69.57759762
Exception Return		0				X	11689040457	69.57762177
Exception Entry		15					11689104561	69.57800334
ITM		0		30H			11689104677	69.57800403
ITM		0		0DH			11689104692	69.57800412
ITM		0		0AH			11689104703	69.57800418
ITM		0		31H		X	11689143417	69.57823462
ITM		0		0DH		X	11689143417	69.57823462
ITM		0		0AH		X	11689143417	69.57823462
ITM		0		32H		X	11689143417	69.57823462
ITM		0		0DH		X	11689143417	69.57823462
ITM		0		0AH		X	11689143417	69.57823462
ITM		0		33H		X	11689143417	69.57823462
ITM		0		0DH		X	11689143417	69.57823462
ITM		0		0AH		X	11689143417	69.57823462
Exception Return	X	0				X	11689143417	69.57823462
Exception Entry		34					11689197436	69.57855617
Exception Exit		34					11689197480	69.57855643
Exception Return		0				X	11689201593	69.57858091

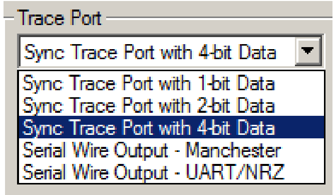
Super TIP: ITM_SendChar is a useful function you can use to send characters. It is found in the header core.CM3.h.

Part C)

Using the ULINK*pro* with ETM Trace:

The examples previously shown with the ULINK2 will also work with the ULINK*pro*. There are two major differences:

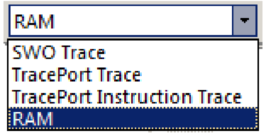
1) The window containing the trace frames is now called Trace Data. More complete filtering is available.
2) The SWV (Serial Wire Viewer) data is sent out the SWO pin to the ULINK2 using UART encoding. The ULINK*pro* can send SWV data either out the SWO pin using Manchester encoding or out the 4 bit Trace Port. This is done so the ULINK*pro* can still support those Cortex-M3 processors that have SWV but not ETM. The trace port is found on the 20 pin Hi-density connector. It is configured in the Trace configuration window as shown below. ETM data is always sent out the Trace Port and if ETM is being used, SWV frames must also sent out this port.


Trace Port
Sync Trace Port with 4-bit Data
Sync Trace Port with 1-bit Data
Sync Trace Port with 2-bit Data
Sync Trace Port with 4-bit Data
Serial Wire Output - Manchester
Serial Wire Output - UART/NRZ

ULINK*pro* offers:

1) Faster Flash programming than the ULINK2.
2) All Serial Wire Viewer features as the ULINK2 does.
3) Adds ETM trace which provides records of all Program Counter values. ULINK2 provides only PC Samples and is not nearly as useful.
4) **Code Coverage**: were all the assembly isntructions executed ? Untested code can be dangerous.
5) **Performance Analysis**: where did the processor spent its time ?
6) **Execution Profiling**: How long instructions, ranges of instructions, functions or C source code took in both time and CPU cycles as well as number of times these were executed.

1) Target Selector Box:



Beside the Target Options icon is the Target Selector drop down menu as shown here. The entries shown select between different settings in the Target Options menus and source files associations. You can select an entry and then look in the Target Options menus to see what is selected. This is a handy method to rapidly select different configurations.

To create your own Target Options, select Project/Manage and select Components,…. You can name them anything.

1. **SWO Trace:** SWV frames are sent out the SWO pin just as with the ULKINK2. ETM trace is not enabled. Manchester format is used (ULINK*pro* does not use UART mode). ULINKpro does not use the Core Clock: setting in the Trace tab to determine what frequency to sample the SWO pin. It does use this value to determine various trace timings. The file STM32F4xx_SWO.ini is selected in the Initialization File: box.
2. **TracePort Trace:** SWV frames are sent out the 4 bit Trace Port. ETM is not enabled. The file STM32F4xx_TP.ini is selected in the Initialization File: box.
3. **TracePort Instruction Trace:** Both SWV and ETM are enabled and sent out the 4 bit Trace Port. The file STM32F4xx_TP.ini is selected in the Initialization File: box.

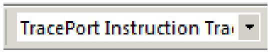
   Note: The STM3240G-EVAL board does not reliably output frames out the Trace Port with CPU speeds above approximately 60 MHz. The Keil MCBSTM32F400 does not have this limitation. The suspect is improper trace layout on the PCB and probably with the TrcClk signal and not the STM32 processor.

   Please see **8) Modifying processor speed for SWO and ETM with STM3240G-EVAL:** on page 32.
4. **RAM:** Loads the program into RAM instead of Flash when entering Debug mode. You do not select the Load icon. The memory settings are in the Target tab in the Target Options menu. The file Dbg_RAM.ini is selected in the Initialization File: box.
   When you switch from RAM to Flash and vice versa, you must rebuild and re-flash the project. This is because the addresses where the executable is located is very different.

2) Blinky_ULp Example:

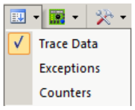
The project in C:\Keil\ARM\Boards\ST\STM3240G-EVAL\Blinky_Ulp is preconfigured for the ULINK*pro*.

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:

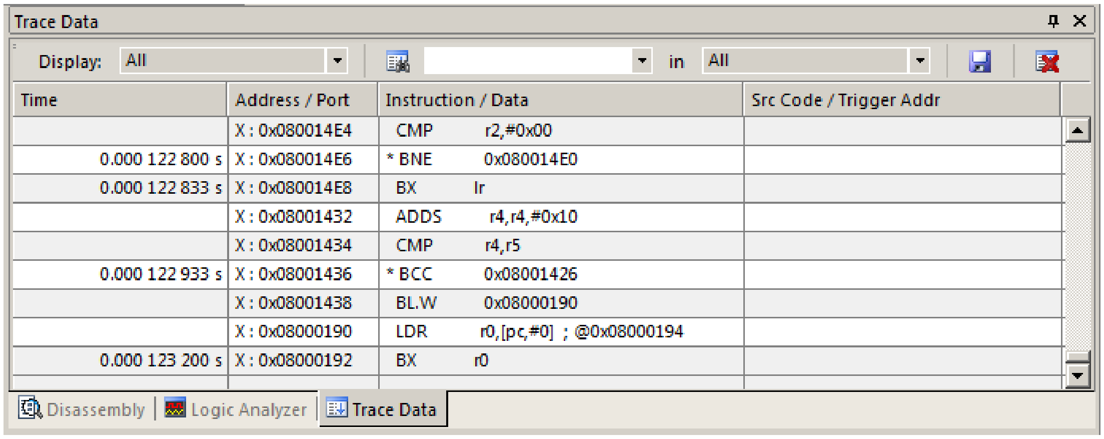


TracePort Instruction Tra

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.



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

11. In this case, 123 200 s shows the last instruction to be executed. (BX r0). In the Register window the PC will display the value of the next instruction to be executed (0x0800_0192 in my case). Click on Single Step once.
12. The instruction PUSH will display: | 0x080011DA | PUSH (r3,lr) | int main(void) { /* Main Program */ |
13. Scroll to the top of the Instruction Trace window to frame # 1. This is nearly the first instruction executed after RESET.

Note: The STM3240G-EVAL does not reliably output frames out the Trace Port with CPU speeds above approximately 60 MHz. The Keil MCBSTM32F400 does not have this limitation. The suspect is improper trace layout on the PCB and probably with the TrcClk signal and not the STM32 processor.

Changing CPU Speed:

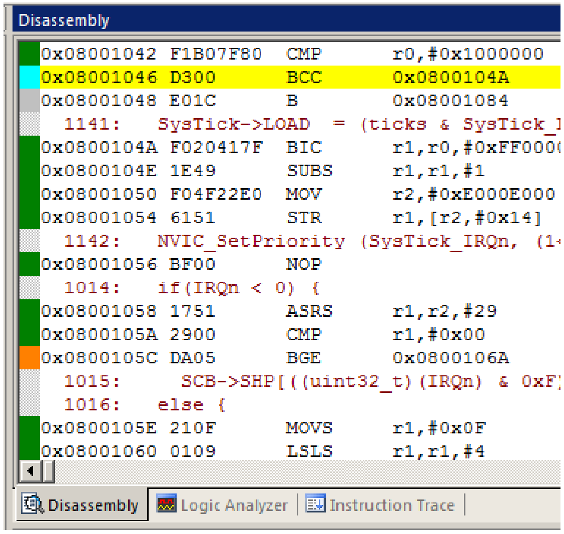
Please see **8) Modifying processor speed for SWO and ETM with STM3240G-EVAL:** on page 32.
The Blinky project in Blinky_Ulp is configured to run at 60 MHz.

3) Code Coverage:

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.



Disassembly

```
0x08001042 F1B07F80   CMP      r0,#0x1000000
0x08001046 D300       BCC      0x0800104A
0x08001048 E01C       B        0x08001084
  1141:     SysTick->LOAD  = (ticks & SysTick_I
0x0800104A F020417F   BIC      r1,r0,#0xFF000(
0x0800104E 1E49       SUBS     r1,r1,#1
0x08001050 F04F22E0   MOV      r2,#0xE000E000
0x08001054 6151       STR      r1,[r2,#0x14]
  1142:     NVIC_SetPriority (SysTick_IRQn, (1<
0x08001056 BF00       NOP
  1014:     if(IRQn < 0) {
0x08001058 1751       ASRS     r1,r2,#29
0x0800105A 2900       CMP      r1,#0x00
0x0800105C DA05       BGE      0x0800106A
  1015:       SCB->SHP[((uint32_t)(IRQn) & 0xF)
  1016:     else {
0x0800105E 210F       MOVS     r1,#0x0F
0x08001060 0109       LSLS     r1,r1,#4
```

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

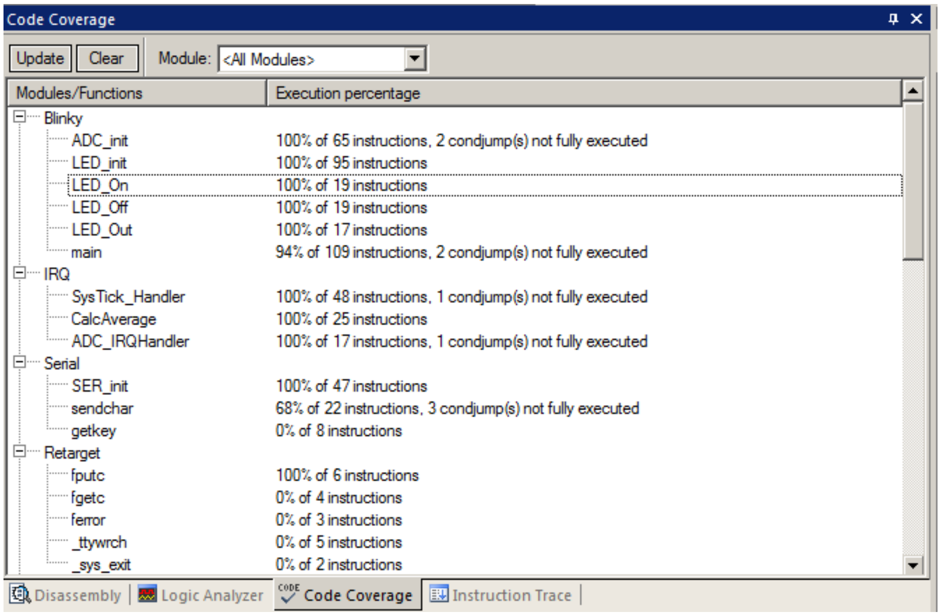
Why was the branch BCC always taken resulting in 0x0800_1048 never being executed ? Or why the branch BGE at 0x800_105C was never taken ? You should devise tests to execute these instructions so you can test them.

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

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

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

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



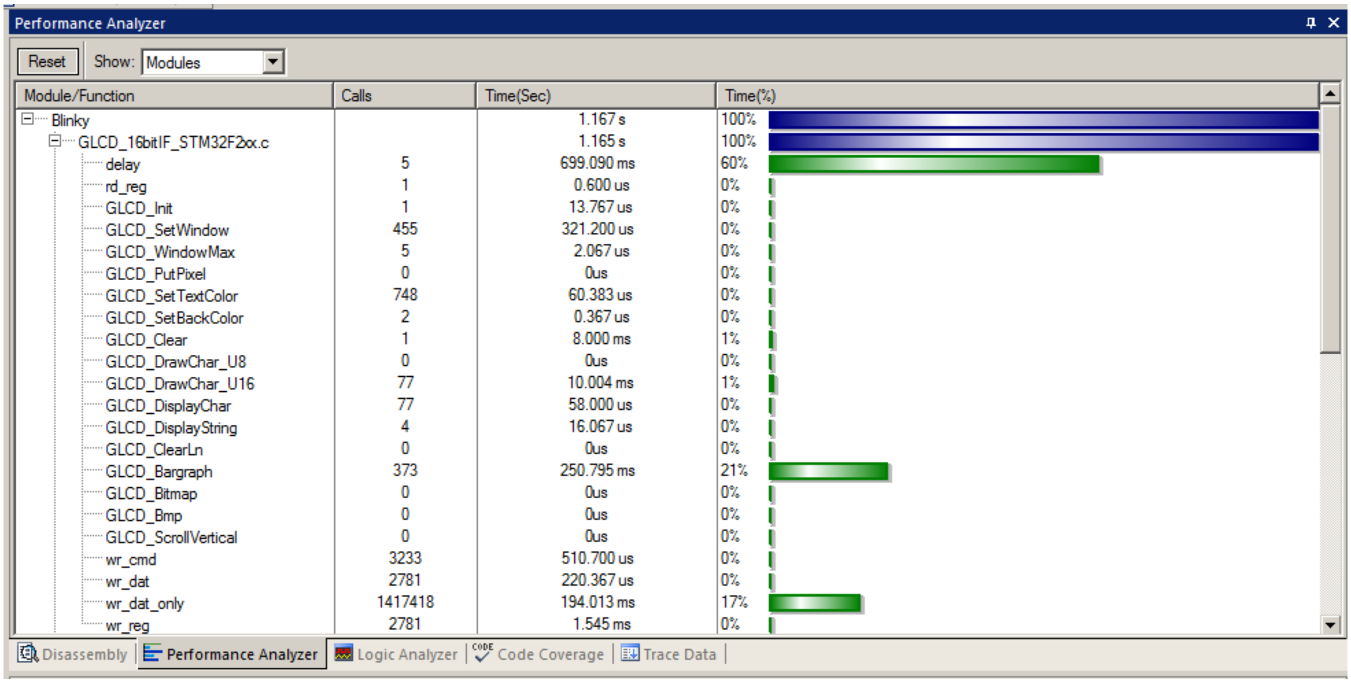
Modules/Functions	Execution percentage
Blinky	
ADC_init	100% of 65 instructions, 2 condjump(s) not fully executed
LED_init	100% of 95 instructions
LED_On	100% of 19 instructions
LED_Off	100% of 19 instructions
LED_Out	100% of 17 instructions
main	94% of 109 instructions, 2 condjump(s) not fully executed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

4) Performance Analysis (PA):

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

Keil provides Performance Analysis with the 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.

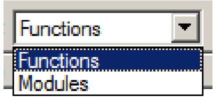
1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it. This clears the PA window and resets the STM32 and reruns it to main() as before. Or select the Reset 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.



Show: Modules

Module/Function	Calls	Time(Sec)	Time(%)
Blinky		1.167 s	100%
GLCD_16bitIF_STM32F2xx.c		1.165 s	100%
delay	5	699.090 ms	60%
rd_reg	1	0.600 us	0%
GLCD_Init	1	13.767 us	0%
GLCD_SetWindow	455	321.200 us	0%
GLCD_WindowMax	5	2.067 us	0%
GLCD_PutPixel	0	0us	0%
GLCD_SetTextColor	748	60.383 us	0%
GLCD_SetBackColor	2	0.367 us	0%
GLCD_Clear	1	8.000 ms	1%
GLCD_DrawChar_U8	0	0us	0%
GLCD_DrawChar_U16	77	10.004 ms	1%
GLCD_DisplayChar	77	58.000 us	0%
GLCD_DisplayString	4	16.067 us	0%
GLCD_ClearLn	0	0us	0%
GLCD_Bargraph	373	250.795 ms	21%
GLCD_Bitmap	0	0us	0%
GLCD_Bmp	0	0us	0%
GLCD_ScrollVertical	0	0us	0%
wr_cmd	3233	510.700 us	0%
wr_dat	2781	220.367 us	0%
wr_dat_only	1417418	194.013 ms	17%
wr_reg	2781	1.545 ms	0%

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.

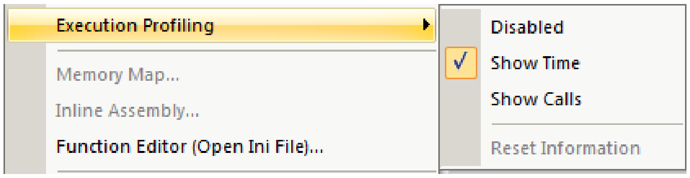


TIP: You can also click on the RESET icon but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

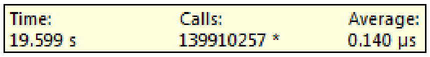
When you click on the RESET icon, the Initialization File .ini will no longer be in effect and this can cause SWV and/or ETM to stop working. Exiting and re-entering Debug mode executes the .ini script again.

5) Execution Profiling:

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.

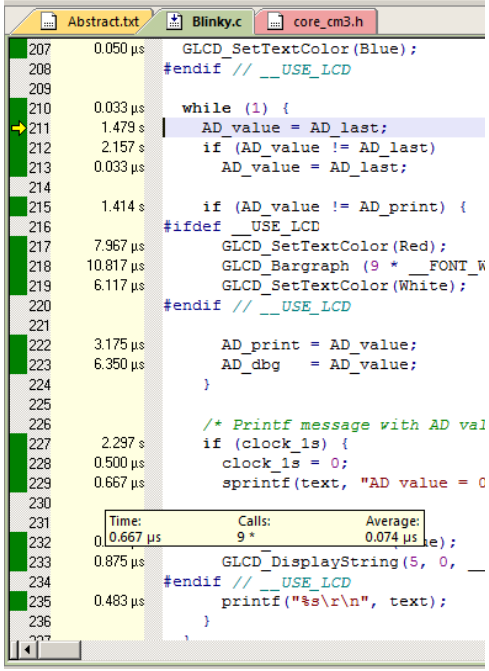


1. Enter Debug mode.
2. Select Debug/Execution Profiling/Show Time.
3. In the left margin of the disassembly and C source windows will display various time values.
4. Click on RUN.
5. The times will start to fill up as shown below right:
6. Click inside the yellow margin of Blinky.c to refresh it.
7. 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:



```
Time:          Calls:           Average:
19.599 s       139910257 *      0.140 µs
```

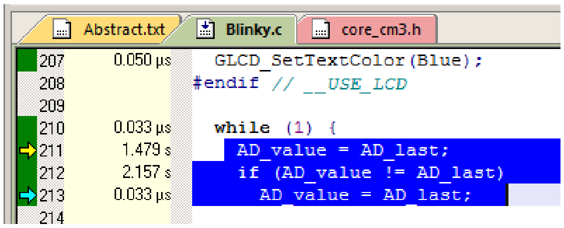
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the margin.



```
0.050 µs      GLCD_SetTextColor(Blue);
            #endif // __USE_LCD

0.033 µs      while (1) {
1.479 s         AD_value = AD_last;
2.157 s         if (AD_value != AD_last)
0.033 µs          AD_value = AD_last;

1.414 s         if (AD_value != AD_print) {
            #ifdef __USE_LCD
7.967 µs          GLCD_SetTextColor(Red);
10.817 µs         GLCD_Bargraph (9 * __FONT_W
6.117 µs          GLCD_SetTextColor(White);
            #endif // __USE_LCD

3.175 µs          AD_print = AD_value;
6.350 µs          AD_dbg   = AD_value;
                }

                /* Printf message with AD val
2.297 s         if (clock_1s) {
0.500 µs          clock_1s = 0;
0.667 µs          sprintf(text, "AD value = 0

              Time:          Calls:     Average:
0.            0.667 µs       9 *        0.074 µs  e);
0.875 µs          GLCD_DisplayString(5, 0, _
            #endif // __USE_LCD
0.483 µs          printf("%s\r\n", text);
                }
```

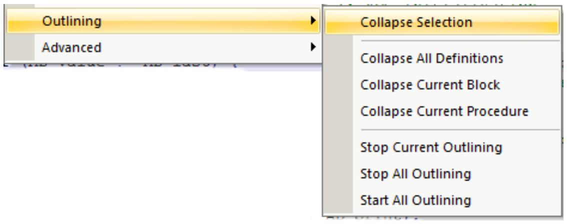
Outlining:

1) Block a section of source as similar to this:

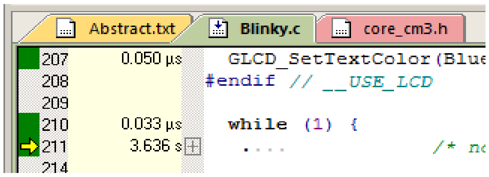


```
0.050 µs      GLCD_SetTextColor(Blue);
            #endif // __USE_LCD

0.033 µs      while (1) {
1.479 s         AD_value = AD_last;
2.157 s         if (AD_value != AD_last)
0.033 µs          AD_value = AD_last;
```

2) Right click on the blue block and select Outlining and then Collapse Section as shown below:



3) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
4) Click on the + to expand it.
5) Stop the program and exit Debug mode.



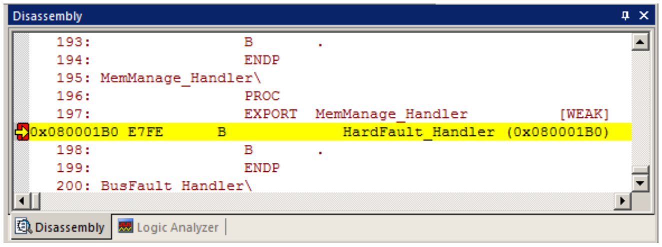
```
0.050 µs      GLCD_SetTextColor(Blue
            #endif // __USE_LCD

0.033 µs      while (1) {
3.636 s ⊞       ....               /* nc
```

6) In-the-Weeds Example:

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.

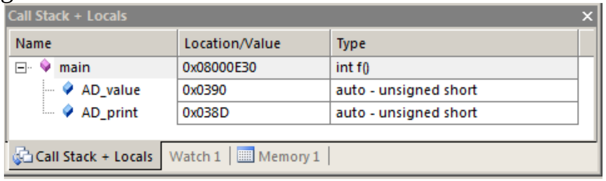
This exception vector is found in the file startup_stm32f4xx.s. If we set a breakpoint by double-clicking on the Hard Fault handler and run the program: at the next Hard Fault event the CPU will jump to the Hard Fault handler (in this case located at 0x0800 01B0 as shown to the right) and stop.



```
   193:                  B       .
   194:                  ENDP
   195: MemManage_Handler\
   196:                  PROC
   197:                  EXPORT  MemManage_Handler           [WEAK]
0x080001B0 E7FE       B                 HardFault_Handler (0x080001B0)
   198:                  B       .
   199:                  ENDP
   200: BusFault Handler\
```

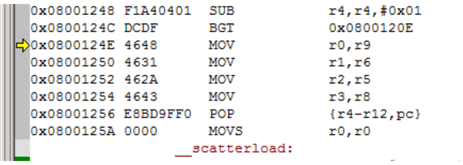
The CPU and also the trace collection will stop. The trace buffer will be visible and extremely useful to investigate and determine the cause of the crash.

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:

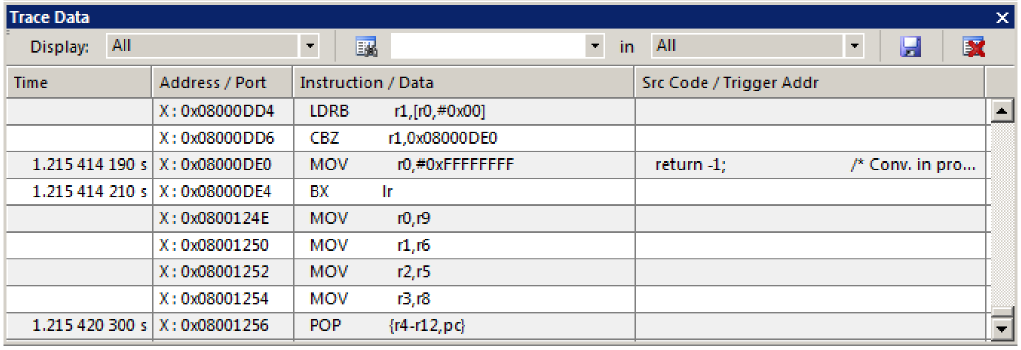


Name	Location/Value	Type
main	0x08000E30	int f()
AD_value	0x0390	auto - unsigned short
AD_print	0x038D	auto - unsigned short

6. In the Disassembly window, scroll down until you find a POP instruction. I found one at 0x0800 1256 as shown below in the third window:
7. Right click on the POP instruction (or at the MOV at 0x0800 124E as shown below) and select Set Program Counter. This will be the next instruction executed.
8. Click on RUN and immediately the program will stop on the Hard Fault exception branch instruction.
9. Examine the Data Trace window and you find this POP plus everything else that was previously executed. In the bottom screen are the 4 MOV instructions plus the offending POP.
10. Note the Branch at the Hard Fault does not show in the trace window because a hardware breakpoint does execute the instruction it is set to therefore it is not recorded in the trace buffer.



```
0x08001248 F1A40401  SUB              r4,r4,#0x01
0x0800124C DCDF      BGT              0x0800120E
0x0800124E 4648      MOV              r0,r9
0x08001250 4631      MOV              r1,r6
0x08001252 462A      MOV              r2,r5
0x08001254 4643      MOV              r3,r8
0x08001256 E8BD9FF0  POP              {r4-r12,pc}
0x0800125A 0000      MOVS             r0,r0
                          __scatterload:
```



Display: All in All

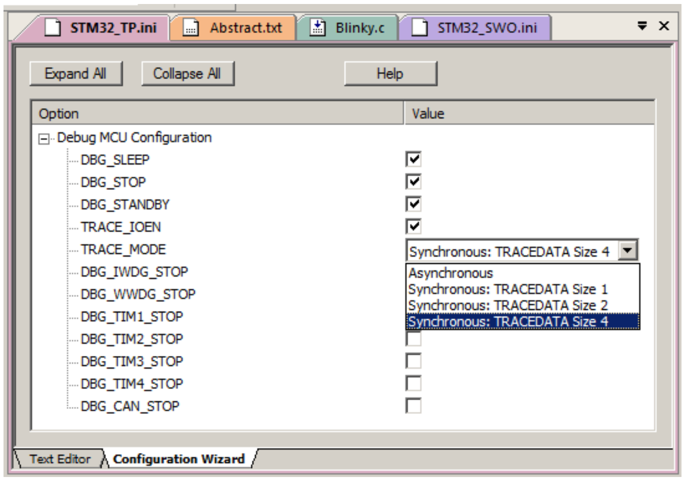
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr
	X : 0x08000DD4	LDRB r1,[r0,#0x00]	
	X : 0x08000DD6	CBZ r1,0x08000DE0	
1.215 414 190 s	X : 0x08000DE0	MOV r0,#0xFFFFFFFF	return -1; /* Conv. in pro...
1.215 414 210 s	X : 0x08000DE4	BX lr	
	X : 0x0800124E	MOV r0,r9	
	X : 0x08001250	MOV r1,r6	
	X : 0x08001252	MOV r2,r5	
	X : 0x08001254	MOV r3,r8	
1.215 420 300 s	X : 0x08001256	POP {r4-r12,pc}	

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

7) Configuring the ULINK*pro* ETM Trace:

The ULINK*pro* was configured for SWV operation using the SWO pin and Manchester encoding on page 11. The project Blinky_Ulp is pre-configured for ETM trace. We will activate ETM trace here manually to illustrate how it is done.

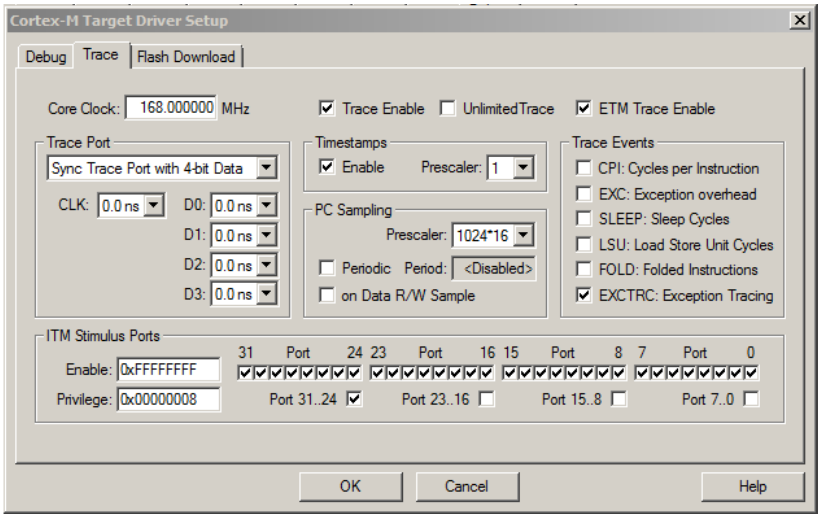
1) Select the STM3240G-EVAL Blinky project.
2) Configure ULINK*pro* for the STM32 processor as described on page 6: **ULINK*pro* and µVision Configuration:** Do not forget to configure the Flash programmer as well.
3) µVision must be stopped and in edit mode (not debug mode).
4) Select Options for Target or ALT-F7 and select the Debug tab.
5) In the box Initialization File: insert STM32F4xx_TP.ini. Click on the Edit box. Edit... The specified ini file will open.
6) Click OK. At the bottom of the ini file, click on the Configuration Wizard tab.
7) Expand the menu and select Synchronous: Trace Data Size 4 as shown here:



Option	Value
Debug MCU Configuration	
DBG_SLEEP	☑
DBG_STOP	☑
DBG_STANDBY	☑
TRACE_IOEN	☑
TRACE_MODE	Synchronous: TRACEDATA Size 4 Asynchronous Synchronous: TRACEDATA Size 1 Synchronous: TRACEDATA Size 2 Synchronous: TRACEDATA Size 4
DBG_IWDG_STOP	
DBG_WWDG_STOP	
DBG_TIM1_STOP	
DBG_TIM2_STOP	□
DBG_TIM3_STOP	□
DBG_TIM4_STOP	□
DBG_CAN_STOP	□

TIP: Asynchronous is used to select the SWO port and is needed for the ULINK2 or ULINK-ME.

8) Click on File/Save All to enable this file. It will be executed when you enter Debug mode.
9) Select Options for Target or ALT-F7 and select the Debug tab (again).
10) Click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side of the window.
11) Click on the Trace tab. The window below is displayed.
12) Core Clock: Enter 168 MHz. ULINK*pro* uses this value only to calculate various timing values.
13) In Trace Port select Sync Trace Port with 4 bit data. It is possible to use other bit sizes but best to use the largest.
14) Select Trace Enable and ETM Trace Enable. Unselect Periodic and leave everything else at default as shown below.
15) Click on OK twice to return to the main µVision menu. Both ETM and SWV are now configured.
16) Select File/Save All.



Core Clock: 168.000000 MHz
☑ Trace Enable □ Unlimited Trace ☑ ETM Trace Enable

Trace Port: Sync Trace Port with 4-bit Data
CLK: 0.0 ns D0: 0.0 ns D1: 0.0 ns D2: 0.0 ns D3: 0.0 ns

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16
□ Periodic Period: <Disabled>
□ on Data R/W Sample

Trace Events:
□ CPI: Cycles per Instruction
□ EXC: Exception overhead
□ SLEEP: Sleep Cycles
□ LSU: Load Store Unit Cycles
□ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

ITM Stimulus Ports
Enable: 0xFFFFFFFF
Privilege: 0x00000008
31 Port 24 23 Port 16 15 Port 8 7 Port 0
☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑
Port 31..24 ☑ Port 23..16 □ Port 15..8 □ Port 7..0 □

The ETM is now configured.

Note: The STM3240G-EVAL board has challenges sending data to the Trace Port connector. At 168 MHz, trace collection is unreliable. Slow the CPU down to 60 MHz. See the instructions on page 32.

TIP: We said that you must use SWD (also called SW) in order to use the Serial Wire Viewer. With the ULINK*pro* and with the Trace Port selected, you can also select the JTAG port as well as the SWD port since no conflict between SWO and TDIO signals will no longer occur.

8) Serial Wire Viewer 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 for these.

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 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.

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

- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), 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 especially oif the stack is corrupted.
- **ETM trace with the ULINK*pro* is best for solving program flow problems.**
- Communication protocol and timing issues. System timing problems.

For complete information on CoreSight for the Cortex-M3: Search for **DDI0314F_coresight_component_trm.pdf** on www.arm.com.

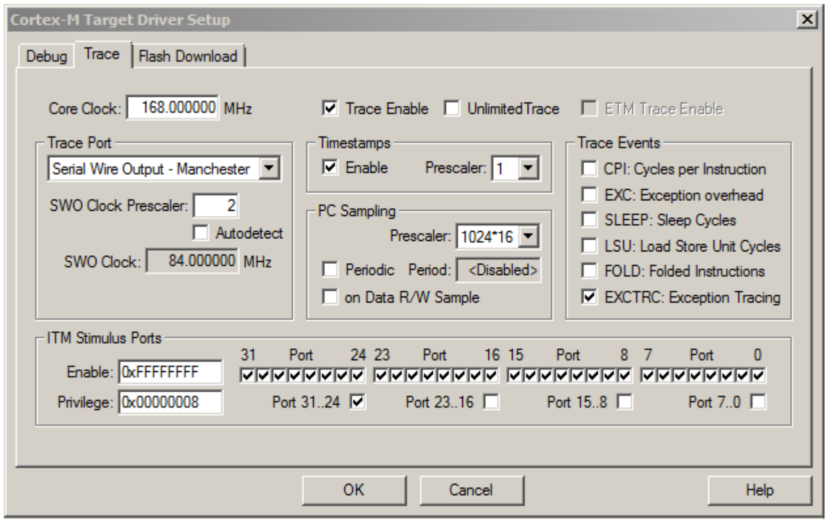
9) Modifying the processor speed for SWO and ETM with STM3240G-EVAL:

The STM3240G-EVAL has an issue with high speed SWV and ETM trace when using the ULINK*pro*. The Keil MCBSTM32F400 does not have this limitation. The ULINK2 and ULINK-ME are not affected as they cause the SWV to be output the SWO pin at relatively slow data rats.

This might be a board layout issue: it is important that the ETM signal traces, especially the ETM clock, be as short in length as possible. Recall the ULINKpro can access SWV information out either the one pin SWO port using the Manchester protocol or the 4 bit Trace Port.

Using the SWO pin using the Manchester protocol:

1) Select Options for Target or ALT-F7 and select the Debug tab.
2) In the drop-down menu box select the ULINK Pro Cortex Debugger.
3) Select the file STM32F4xx_SWO.ini.
4) Select Settings and the Target Driver window below opens up:
5) In the Trace setup window shown here: Select Serial Wire Output – Manchester.
6) Uncheck Autodetect.
7) Set SWO Clock Prescaler to 2.
8) The signal from the SWO pin will be reduced to 84 MHz. Click OK twice.
9) SWO will now function properly.



Core Clock: 168.000000 MHz
☑ Trace Enable ☐ UnlimitedTrace ☐ ETM Trace Enable

Trace Port: Serial Wire Output - Manchester
SWO Clock Prescaler: 2
☐ Autodetect
SWO Clock: 84.000000 MHz

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16
☐ Periodic Period: <Disabled>
☐ on Data R/W Sample

Trace Events:
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

ITM Stimulus Ports:
Enable: 0xFFFFFFFF
Privilege: 0x00000008
Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

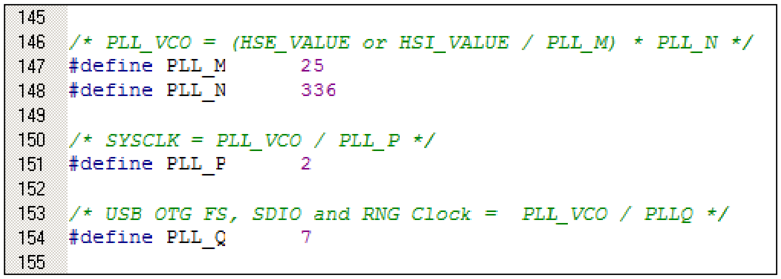
Using 4 bit Trace Port:

The CPU clock must be slowed to 60 MHz.

In the file system_stm32f4xx.c there are three variables to change to modify the clock speed.

PLL_M, PLL_N and PLL_P. Shown are the values for 60, 120 and 168 MHz. PLL_Q is shown for reference.

	60	120	168 MHz
PLL_M	25	25	25
PLL_N	240	240	336
PLL_P	4	2	2
PLL_Q	5	5	7



```

/* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLL_M) * PLL_N */
#define PLL_M      25
#define PLL_N      336

/* SYSCLK = PLL_VCO / PLL_P */
#define PLL_P      2

/* USB OTG FS, SDIO and RNG Clock =  PLL_VCO / PLLQ */
#define PLL_Q      7

```

1) Modify the values in system_stm32f4xx.c for 60 MHz, rebuild to source files and program the Flash.
2) The file STM32F4xx_TP.ini must be entered in the ini box in the Debug tab.
3) Re-enter debug mode and the ETM trace will now work.

ETM with this board works reliably at 60 MHz. With the Keil MCBSTM32F4 it works reliably to 168 MHz. On your own custom board, place the ETM connector as close to the CPU as practical. Practice appropriate high speed PCB design. The ETM TraceCLK is the most important and most easily corrupted signal.

10) Keil Products:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) Promotion with ULINK*pro* until December 31, 2011 - $9,995 Please contact Keil Sales for more details.
- MDK-Standard (no compile or debug limit) **-** $4,895
- MDK-Basic (256K Compiler Limit, No debug Limit) - $2,695
- MDK-Lite (Evaluation version) $0

All versions include Keil RTX RTOS *with source code !*

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

Call Keil Sales for special university pricing.

For the ARM University program: go to www.arm.com and search for university.

All products include Technical Support for 1 year. This can be renewed.

USB-JTAG adapter (for Flash programming too)

- ULINK2 - $395 *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINK*pro* - $1,395 – Cortex-Mx SWV & ETM trace

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.

Prices are for reference only and are subject to change without notice.

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor.

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, contact the author, Keil Technical support or www.keil.com/st

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

KEIL™
Tools by ARM