STMicroelectronics STM32C: Cortex™-M3 Lab

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

For Keil MCBSTM32C™ Eval Board Fall 2013 Version 1.3 by Robert Boys, bob.boys@arm.com

For Keil MCBSTM32C™ & ST STM3210C-EVAL Evaluation

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_245.asp

Introduction:

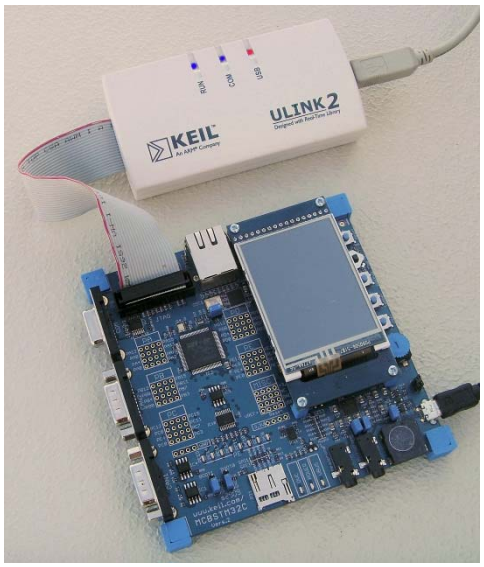
The purpose of this lab is to introduce you to the STMicroelectronics Cortex™-M3 processor using the ARM® Keil™ MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) and ETM trace on the Keil **MCBSTM32C** evaluation board. At the end of this tutorial, you will be able to confidently work with STM32 processors and Keil MDK™. This tutorial will also work with the STMicroelectronics **STM3210C-EVAL** board with no modifications.

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 now comes with a BSD license. Visit www.keil.com/st.

Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler. MDK is a turn-key product and includes examples.
2. Serial Wire Viewer and ETM trace capability is included. A full feature Keil RTOS called RTX is included with MDK.
3. RTX Kernel Awareness windows are updated in real-time. Kernel Awareness for Keil RTX, CMX, Quadros and Micrium.
4. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™ and Segger J-Link (version 6 or later). ST-Link V2 is supported with SWV. ULINK*pro* supports ETM instruction and data trace.
5. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.



Keil ULINK2 and MCBSTM32

This document details these features:

1. Serial Wire Viewer (SWV) and ETM Trace using ULINK*pro*™.
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: two kernel awareness windows for the Keil RTX RTOS. They are updated in real-time.

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-M3. 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 V1 adapter does not support SWV or ETM. ST-Link V2 supports SWV but not ETM trace.

Embedded Trace Macrocell (ETM):

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

Index:

STM32 Evaluation Boards:

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

Keil part number	URL for board info	Debug Connectors	ST board equivalent
MCBSTM32™	www.keil.com/mcbstm32/	JTAG/SWD	STM32F10X-128K-EVAL (color LCD)
MCBSTM32E™ ***replaced by EXL***	www.keil.com/mcbstm32EXL/	Cortex Debug and ETM	STM3210E-EVAL
MCBSTM32EXL™	www.keil.com/mcbstm32EXL/	Cortex Debug and ETM	STM3210E-EVAL
MCBSTM32C™	www.keil.com/mcbstm32c/	Cortex Debug and ETM	STM3210C-EVAL
MCBSTM32F200	www.keil.com/mcbstm32f200/	Cortex Debug and ETM	STM3220G-EVAL
MCBSTM32F400	www.keil.com/mcbstm32f400/	Cortex Debug and ETM	STM32429-EVAL

Keil MDK provides example projects for many STMicroelectronics boards:

Example projects for STMicroelectronics boards are found inside Keil MDK here: C:\Keil\ARM\boards\ST.

You can adapt the instructions in this document for any board using a STM32 processor including your own target.

STM32 processors need a special .ini file that configures the CoreSight Serial Wire Viewer and/or 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 needs to be configured for either SWO or 4 bit Trace Port operation. SWO is default. Instructions are provided later.

Visit www.keil.com/st for more information about Keil support of STMicroelectronics processors.

Software Installation:

This document was written for Keil MDK 4.14 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.

The current version is 4.72. Some windows will look different dues to product improvement.

To obtain a copy of MDK go to www.keil.com/arm and select "Evaluation Software" from the left column.

You can use the evaluation version of MDK and a ULINK2, ULINK-ME, ULINK*pro*, a J-Link or a ST-Link V2 for this lab. You must make certain adjustments for non-ULINK adapters 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 parts are equipped with ETM. All have SWV.

MDK 5.0:

ARM will release MDK 5.0 in the Fall 2013. There are substantial improvements. See www.keil.com/mdk5.

JTAG and SWD Definitions: It is useful to have a general 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 come out this one pin output.

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:

Keil provides examples for evaluation boards made by Keil in C:\Keil\ARM\boards\Keil. Most example projects are pre-configured to use a ULINK2 or a ULINK-ME. Serial Wire Viewer is not usually configured. Projects that contain a Ulp in their directory name are configured to use a ULINK*pro*. SWV and ETM are normally pre-configured. For instructions on converting a project from ULINK2 or ULINK-ME to a ULINK*pro* and vice versa please see the instructions under Part A.

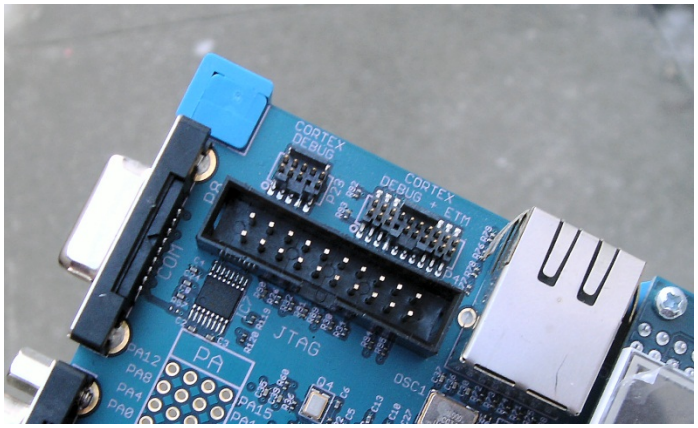
Example projects for STMicroelectronics boards are found under C:\Keil\ARM\boards\ST.

Most example projects will compile within the 32 K code and data limit of MDK. Exceptions are LCD_Demo and Demo. A compiled executable .axf file is provided to allow you to run, evaluate and debug these programs. If you attempt to compile these projects the .axf file will be erased. You must reinstall MDK to get the .axf file back unless you backed it up first.

Part A)

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

The Keil MCBSTM32C is equipped with the new ARM standard 10 and 20 pin CoreSight connectors for JTAG/SWD, SWO and ETM access as shown here:



The legacy 20 pin JTAG connector is provided. This provides JTAG, SWD and SWO access.

The 10 pin "Cortex Debug" 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.

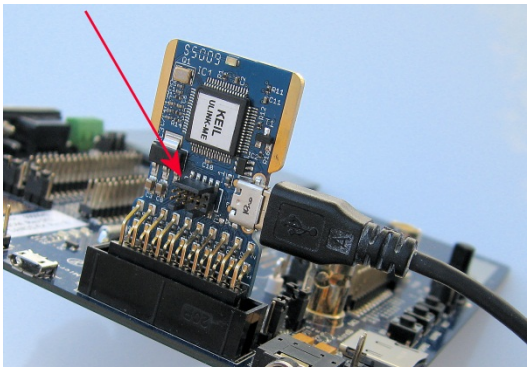
The 20 pin Cortex Debug + ETM" provides JTAG, SWD, SWO and adds 4 bit ETM support and connects to the ULINK*pro* adapter.

Connecting a ULINK2 or ULINK-ME:

Legacy 20 Pin JTAG Connector:

A ULINK2 plugged to a STM32 board is pictured on the first page of this document.

10 Pin Cortex Debug Connector: You will need to take the case off the ULINK2 and install the 10 pin cable. The ULINK-ME does not have a case and the cable can be directly installed on the 10 pin connector or directly to the legacy JTAG connector. The ULINK-ME is pictured here and the arrow points to the 10 pin connector.



20 Pin Cortex Debug + ETM Connector: Keil does have a 10 pin to 20 pin adapter cable available to connect to this connector. The first 10 pins on the 20 pin have an identical layout as the 10 pin.

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

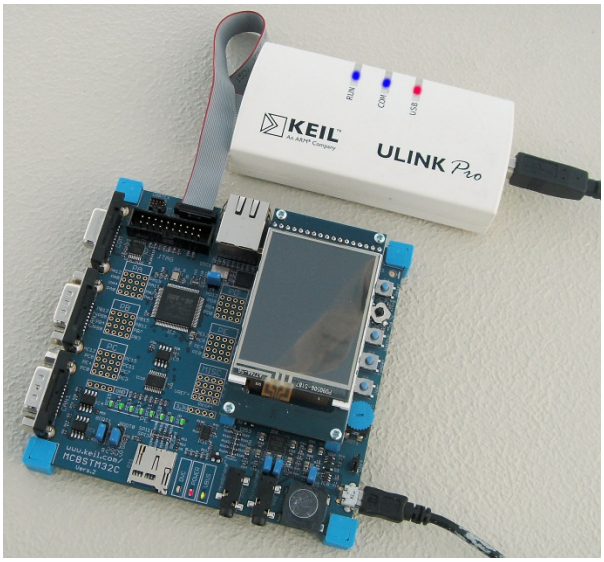
Connecting a ULINK*pro*:

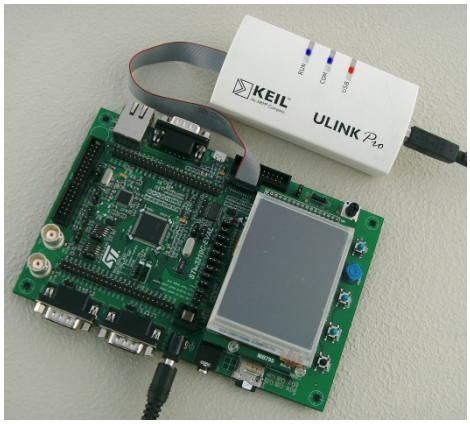
The ULINK*pro* connects to a STM32 board with its standard 20 pin Hi—Density connector or the standard JTAG connector with a supplied adapter.

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

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

Pictured is a ULINK*pro* with a MCBSTM32C (right) and a STM3210C-EVAL from STMicroelectronics (below).





2) ULINK2 or ULINK-ME and µVision Configuration:

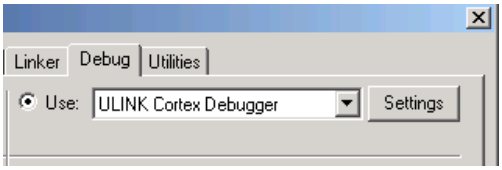
It is easy to select a USB debugging adapter in µVision. You must configure the connection to both the target and to Flash programming in two separate windows as described below. They are each selected using the Debug and Utilities tabs.

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

Serial Wire Viewer 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, displays all program counter values and an enhanced instruction trace 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 Keil MCBSTM32C board on page 1.

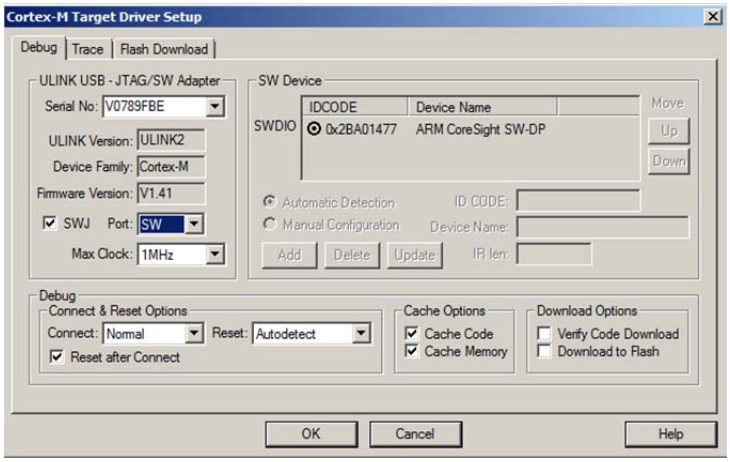


Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select ULINK as shown here:

2. 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).
3. In **Port:** select SWJ and SW. SWV will not work with JTAG selected.
4. 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.



Step 2) Configure the Keil Flash Programmer:

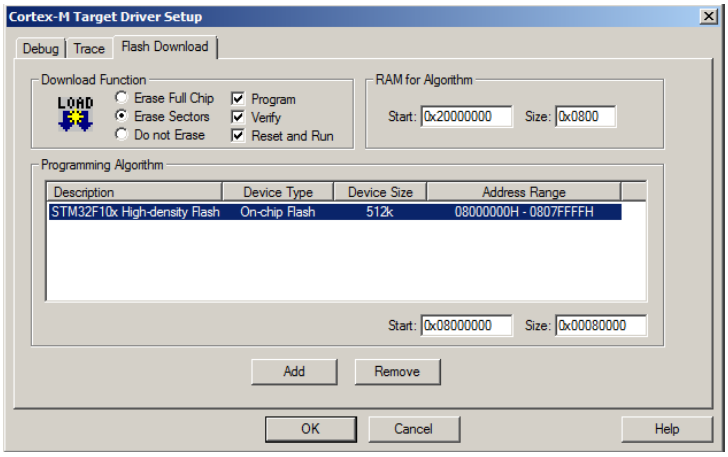
5. Click on OK once and select the Utilities tab.
6. Select the ULINK similar to Step 2 above.
7. Click Settings to select the programming algorithm if one is not visible.
8. SelecSTM32F10x High-density Flash as shown below or the one for your processor:
9. Click on OK once.

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

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

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

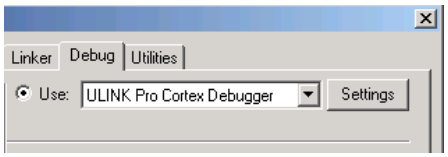
TIP: If you select ULINK or ULINK*pro*, and have the opposite ULINK physically connected to your PC; the error message will say "**No ULINK device found**". This message actually means that µVision found the wrong Keil adapter connected.



3) ULINK*pro* and µVision Configuration:

Step 1) Select the debug connection to the target:

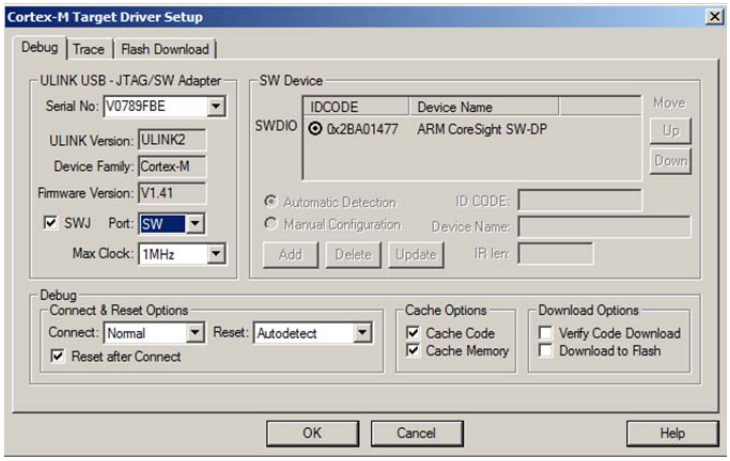
1. Assume the ULINK*pro* is connected to a powered up STM32 target board, µVision is running in Edit mode (as it is when first started – the alternative to Debug mode) and you have selected a valid project. The ULINK*pro* is shown connected to the MCBSTM32C on page 4.
2. Select Options for Target or ALT-F7 and select the Debug tab. In the drop-down menu box select the ULINK Pro Cortex Debugger as shown here:



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.

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.



Step 2) Configure the Keil Flash Programmer:

1. Click on OK once and select the Utilities tab.
2. Select the ULINK*pro* similar to Step 2 above.
3. Click Settings to select the programming algorithm if one is not visible.
4. Select STM32F10x High-density 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.

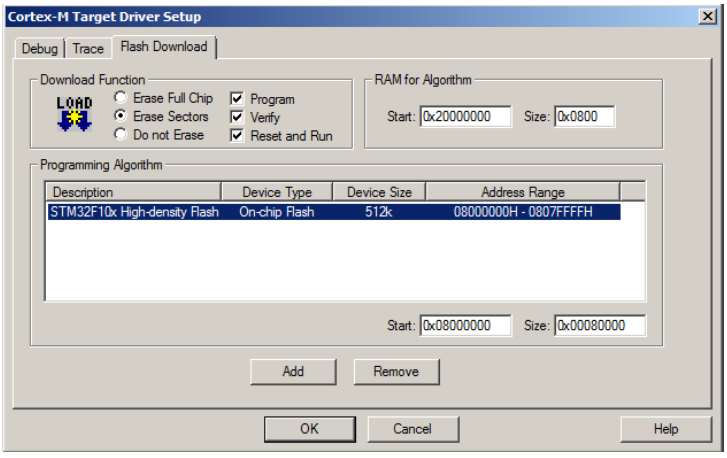
1. Click on OK to return to the µVision main screen.
2. You have successfully connected to the STM32 target processor and selected the µVision Flash programmer.

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 (Instruction Trace) will be quite different from the ULINK2 Trace Records as it offers additional features.

One feature is it 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.



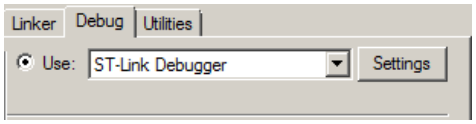
4) ST-Link from STMicroelectronics and µVision Configuration:

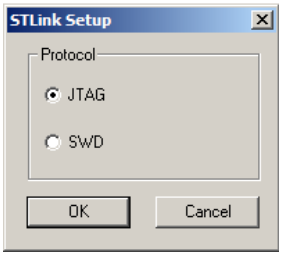
The economical ST-LINK V2 can be used with µVision to provide stable JTAG or SWD debugging. It provides Serial Wire Viewer, on-the-fly Watch and Memory updates and write capability, on-the-fly breakpoint setting and Watchpoints.

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.

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 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 need to select SWD if your target board only has the two SWD signals and not the full set of JTAG signals or you want to use Serial Wire Viewer.



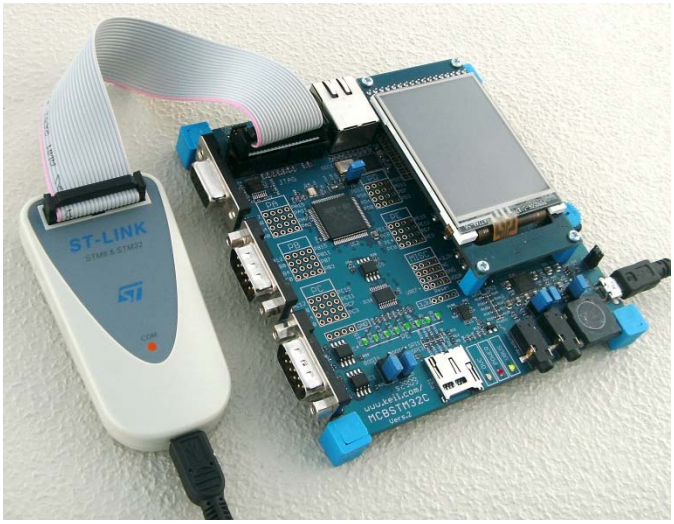


Step 2) Configure the Keil Flash Programmer:

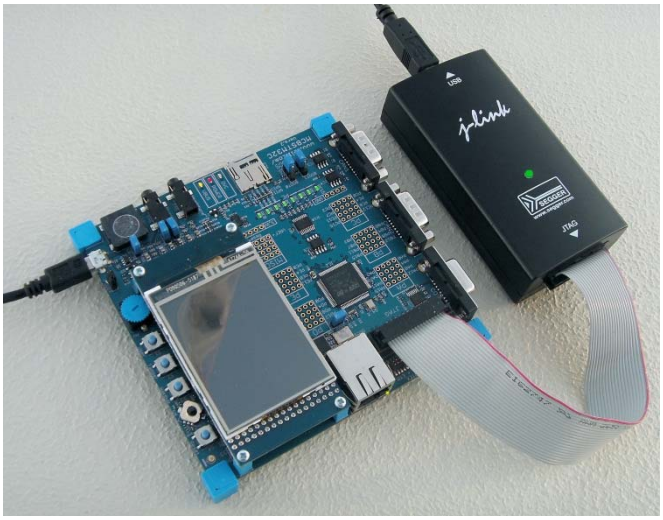
5. Click on OK once and select the Utilities tab.
6. Select the ST-Link Debugger similar to Step 2 above.
7. You do not select any Flash algorithm. ST-LINK does this automatically.
3. Click on OK twice to return to the µVision main screen.
4. You have successfully connected to the STM32 target processor 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 will need the Initialization ini file if you want to use Serial Wire Viewer. ST-Link V1 did not support SWV but ST-Link V2 does. µVision supports both of these adapters. See page 12: Configuring the Serial Wire Viewer (SWV): for instructions about using the .ini file.



ST-Link V1



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 does not provide ITM, data R/W trace frames in MDK 4.14. Segger has the new J-Link Ultra which provides faster operation. 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 provided by Segger on the J-Link.

SWV is easily overloaded 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 and display a fast changing variable. Try disabling the timestamps but some functions need it and the trace will stop operating.

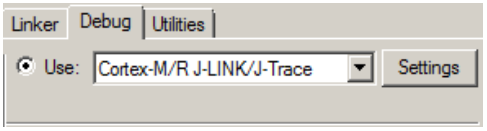
The J-Link is configured using very similar windows as with the ULINK2. This include SWV configuration. The J-Link uses an Instruction Trace window similar to the ULINK*pro*. If you double click on a PC Sample frame, that instruction will be highlighted in the Disassembly and Source windows. The J-Link 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:

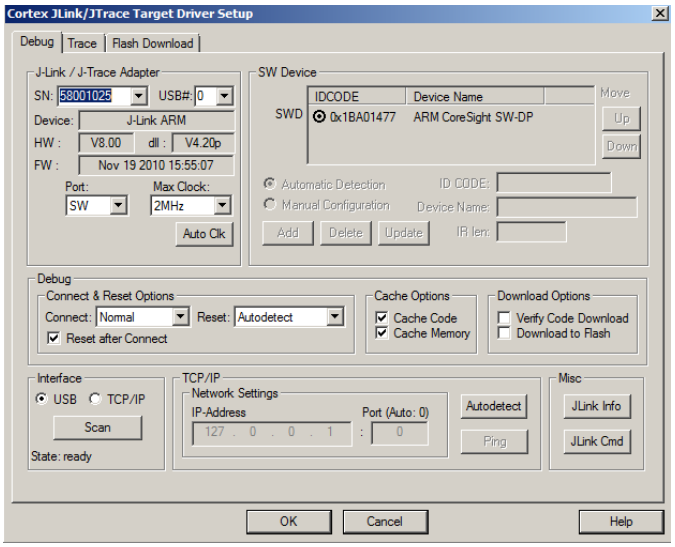
2. Select Options for Target  or ALT-F7 and select the Debug tab. In the drop-down menu box select the J-LINK or J-Trace as shown here:



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.

Step 2: Configure the Keil Flash Programmer:

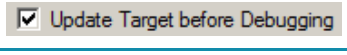
6. Click on OK once and select the Utilities tab.
7. Select the ST-Link Debugger similar to Step 2 above.
8. Click Settings to select the programming algorithm if one is not visible.
9. Select STM32F10x High-density Flash as shown in the directions for the ULINK2 or the algorithm for your processor.
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.

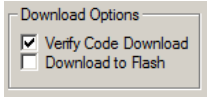


Configure the SWV Trace

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.

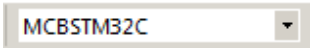
TIP: It is easy to miss programming the Flash with your latest .axf executable. Select either the Verify Code Download in the Target/Debug/Settings as shown here or select Update Target before Debugging:  or make sure you program the Flash manually by clicking on the Load icon.



Part B)

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

We will connect a Keil MDK development system using real target hardware and a ULINK2 or ULINK-ME. These instructions use a Keil MCBSTM32C board. It is possible to use the ULINK*pro* for this example but you must configure it. The project referenced below is pre-configured to use ULINK2. You can also use a ST-Link V1 or V2.

1. Connect the equipment as pictured on the first page.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Keil\MCBSTM32C\Blinky\Blinky.uvproj.
4. Make sure "MCBSTM32C" is selected. 

This is where you create and select different target configurations such as to execute a program in RAM or 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.

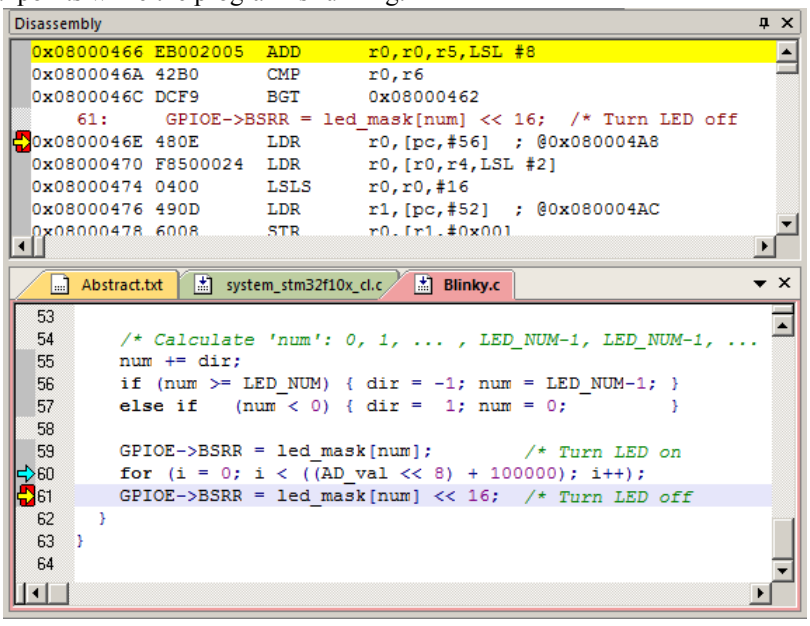
The LEDs on the STM32 board will now blink at a rate determined by the setting of POT1

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

STM3210C-EVAL Board: No LEDs will light but the program will run. Rotate RV1and monitor PE8 or PE9 with a meter.

2) Hardware Breakpoints:

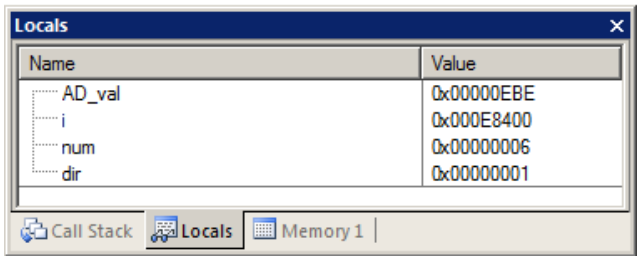
1. With Blink running, double-click in the left margin on a darker gray block in the source file Blinky.c between Lines 55 through 61 as shown below:
2. A red block is created and soon the program will stop at this point.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows.
4. The cyan arrow is a mouse selected pointer and is associated with the yellow band in the disassembly window.
5. Note you can set and unset hardware breakpoints while the program is running.
6. The STM32 has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to.



3) Locals Window:

Whenever the program is stopped, the Locals window will display the local variables for 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. Leave the hardware breakpoint active from the previous page.
2. AD_val contains the pot position value.
3. Rotate the pot to a new position and click on RUN.
4. Note AD_val is updated each time the program stops.



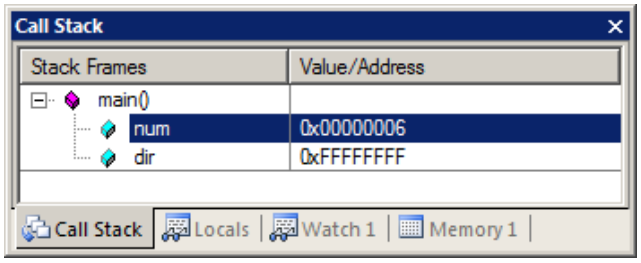
TIP: This is standard "Stop and Go" debugging. ARM CoreSight debugging technology is much better than this. You can display this variable updated in real-time while the program is running. No additions or changes to your code are required.

4) Call Stack Window:

This window displays the list of called functions 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.

The example Blinky.c does not have any function calls so the Call Stack window is a simple one.

1. Click on the Call Stack tab.
2. This widow opens up showing two local variables:
3. Each time you click on RUN this will be updated.
4. Remove the breakpoint by double-clicking on it.



5) 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. CoreSight and µVision can do this.

CoreSight cannot display local variables on-the-fly so these must first be converted into static or global variables.

How to convert Local Variables to be viewable in the Watch or Memory windows:

All you need to do is to make the variable AD_val static !

1. In the declaration for AD_val in Blinky.c, add the keyword static and make **i** a separate int variable like this:

```
int main (void) {
        static int AD_val;
        int i;
```

2. This will ensure that variable AD_val always exists and is visible to µVision.
3. Exit debug mode. **TIP:** You can edit files in edit or debug mode, but can compile them only in edit mode.
4. Compile the source files by clicking on the Rebuild icon. Hopefully they compile with no errors or warnings.
5. To program the Flash, click on the Load icon. 

. A progress bar will be displayed at the bottom left.

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

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

TIP: µVision in conjunction with CoreSight can display in real-time global and static variables, structures, peripheral registers and physical memory locations. Local variables cannot be displayed because they are active only when in scope while their function is executed. Convert locals to static or global variables to see them. This conversion usually means the variable is stored in volatile memory rather than a CPU register. There will be a small time penalty incurred.

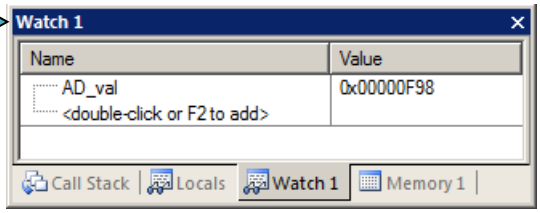
This feature is available on any Keil ULINK or the Segger J-Link version 6 or later.

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

The Watch and memory windows will display updated variable values in real-time. It does this through the ARM CoreSight debugging technology that is part of Cortex-M3 processors. It is also possible to "put" or insert values into these memory locations in real-time. It is possible to "drag and drop" variables into windows or enter them manually.

Watch window:

1. You can do the following steps while the CPU is running. Click on RUN if desired.
2. Find the static variable AD_val in Blinky.c. This was changed from a local to a static var in the previous example.
3. Open the Watch 1 window by clicking on the Watch 1 tab as shown or select View/Watch Windows/Watch 1.
4. In Blinky.c, block AD_val, click and hold and drag it into Watch 1. Release it and it will be displayed updating as shown here:



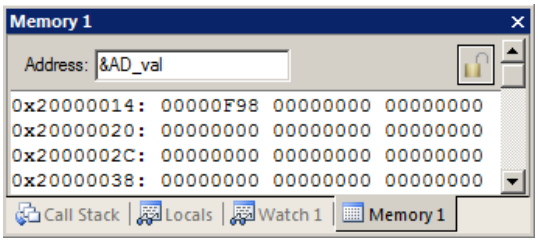
5. Rotate POT1 to see the value update.
6. You can also enter a variable manually by double-clicking or pressing F2 and using copy and paste or typing the variable.

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

6. Double click on the value for AD_val 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. You can also do this in the Memory window with a right-click and select Modify Memory.

Memory window:

1. Drag 'n Drop AD_val into the Memory 1 window or enter it manually. Rotate the pot and watch the window.
2. Note the value of AD_val 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_00014).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of AD_val is displayed as shown here:



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_val 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-M3 is a Harvard architecture. This means it has separate instruction and data buses. While the CPU is fetching instructions at full speed, there is plenty of time for the CoreSight debug module to read or write 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.

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.

STM3210C-EVAL Board: The Watch and Memory windows will display exactly the same on the ST board.

7) 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:

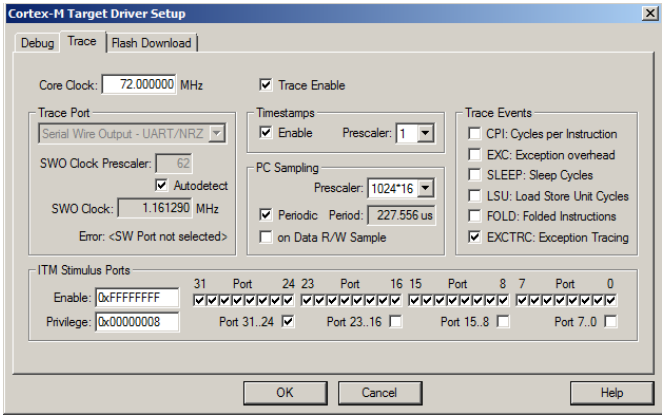
1. µVision must be stopped and in edit mode (not debug mode).
2. Select Options for Target or ALT-F7 and select the Debug tab.
3. In the box Initialization File: enter **..\Blinky_ULp\STM32_SWO.ini** You can use the Browse button:
4. Click on Settings: beside the name of your adapter (ULINK Cortex Debugger) 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 72 and select the Trace Enable box. This is the default frequency for many STM32 projects.
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: Any of these ini files will work in Step 3:
STM32_SWO.ini
C:\Keil\ARM\Boards\MCBSTM32C\Blinky_Ulp

STM32F10x_DBG.ini
C:\Keil\ARM\Boards\MCBSTM32E\Blinky

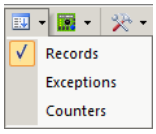
STM32DBG.ini
C:\Keil\ARM\Boards\MCBSTM32E\STLIB_Blinky

These are set by default to SWV operation. You must change them to use the Trace Port and ETM.



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:



TIP: If you do not see PC Samples as shown and either nothing or erratic 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.

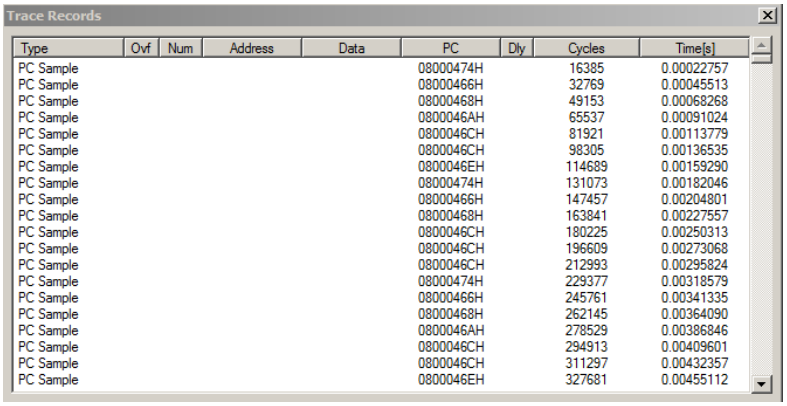
Double-click inside this window to clear it.

If you right click inside this window you can see how to filter various types of frames out. No other frames than PC Samples exist in this simple example.

TIP: SWV is easily overloaded as indicated by an "x" in the OVF or Dly column.

Select only that information needed.

There are more features of Serial Wire Viewer.

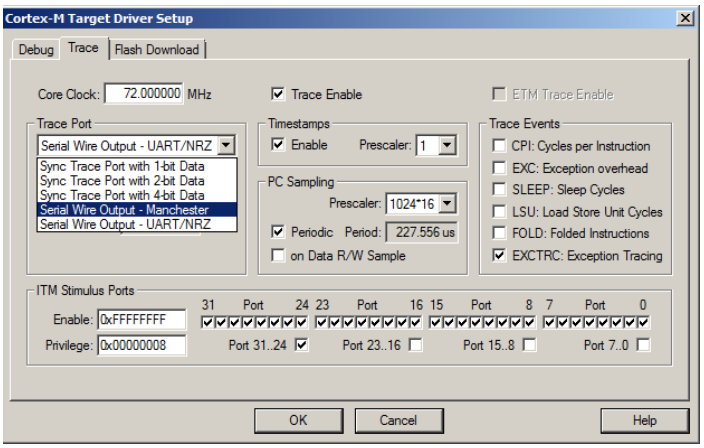


B) SWV for ULINK*pro*:

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).
2. Select Options for Target  or ALT-F7 and select the Debug tab.
3. In the box Initialization File: enter **..\Blinky_ULp\STM32_SWO.ini** You can use the Browse button: 
4. Click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side of the window.
5. Click on the Trace tab. The window below is displayed.
6. Core Clock: No need to enter anything. ULINK*pro* determines this automatically. Select the Trace Enable box.
7. In the Trace Port select Serial Wire Output – Manchester. Selecting UART/NRZ will cause an error.
8. Select Periodic and leave everything else at default. Selecting Periodic activates PC Samples.
9. Click on OK twice to return to the main µVision menu. SWV is now configured.

TIP: Sync Trace Port with 4 bit Data field sends the trace records out the 4 bit trace port rather than the single pin SWO. The Trace Port is faster and must be selected for ETM trace. It is available only with the ULINK*pro*. We will examine this setting later. You could use it now if you prefer.



Note: Any of these ini files will work in Step 3:
STM32_SWO.ini
C:\Keil\ARM\Boards\MCBSTM32C\Blinky_Ulp

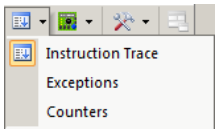
STM32F10x_DBG.ini
C:\Keil\ARM\Boards\MCBSTM32E\Blinky

STM32DBG.ini
C:\Keil\ARM\Boards\MCBSTM32E\STLIB_Blinky

These are set by default to SWV operation. You must change them to use the Trace Port and ETM.

Display Trace Records:

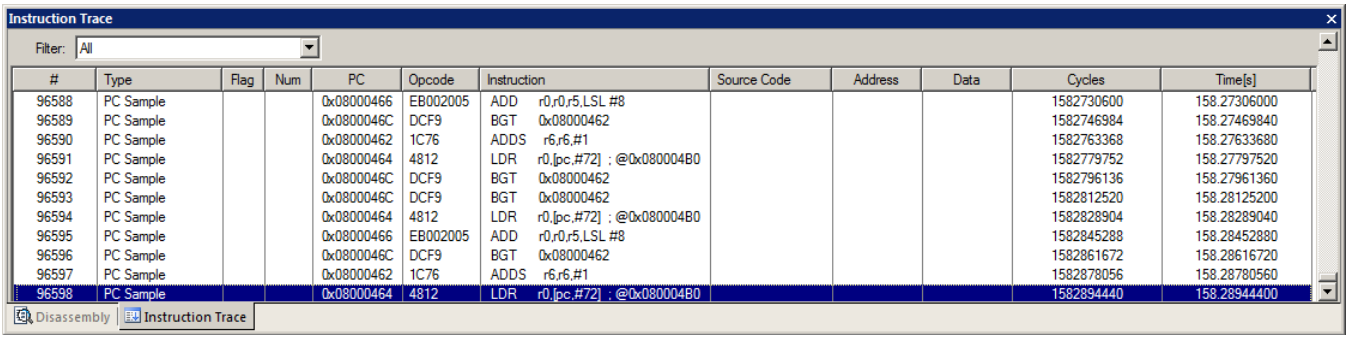
1. Enter Debug mode. 
2. Click on the RUN icon.  .
3. Open the Instruction Trace window by clicking on the small arrow beside the Trace icon:
4. The Instruction Trace window will open and display PC Samples as shown below:



TIP: The Instruction Trace window is different that the Trace Records window provided with the ULINK2. Note the disassembled instructions are displayed and if available, the source code is also displayed. Clicking on a PC Sample line will take you to that place in the source and disassembly windows.

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.

You cannot clear the Instruction Trace window by double-clicking inside it. To clear the trace, exit and re-enter debug mode.



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

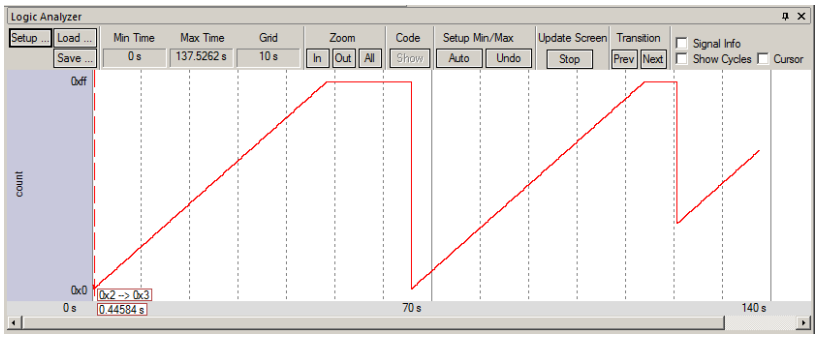
This example will use the ULINK2 with the Blinky example. Please connect a ULINK2 or ULKINK-ME to your STM32 board and configure it for SWV trace. If you want to use a ULINK*pro* you will have to make appropriate modifications.

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.

1. Create a global variable called count. Enter **int count;** before main() near line 16 in Blinky.c.
2. Enter count ++; just after the for loop near line 60 in Blinky.c:

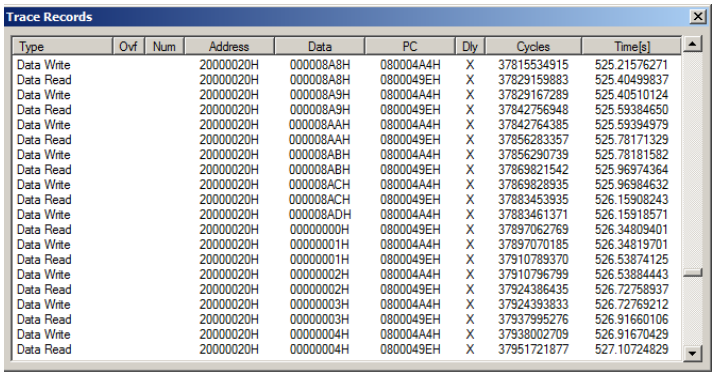
```
for (i = 0; i < ((AD_val << 8) + 100000); i++);
count ++;
```

3. Compile the source files by clicking on the rebuild icon. 
4. Program the STM32 flash by clicking on the Load icon: 
5. 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 **count** you created in Blinky.c. It is declared near line 16.
11. Block **count** 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 0xFF. Click on Close. The LA is completely configured now.
13. Drag and drop **count** into the Watch 1 window. It should be incrementing if Blinky is running.
14. Adjust the Zoom OUT icon in the LA window to about 1 second or so to get a nice ramp as shown below.
15. In the Watch 1 window, double-click on the **count** value and enter 0 and press Enter.
16. This value will be displayed in the LA window as shown here: You can enter any reasonable value into **count**.



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.
5. The window similar below opens up:
6. The first line below says:
   The instruction at 0x0800_04A4 caused a write of data 0x0000_08A8 to address 0x2000_0020 at the listed time in Cycles or Time.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000020H	000008A8H	080004A4H	X	37815534915	525.21576271
Data Read			20000020H	000008A8H	0800049EH	X	37829159883	525.40499837
Data Write			20000020H	000008A9H	080004A4H	X	37829167289	525.40510124
Data Read			20000020H	000008A9H	0800049EH	X	37842756948	525.59384650
Data Write			20000020H	000008AAH	080004A4H	X	37842764385	525.59394979
Data Read			20000020H	000008AAH	0800049EH	X	37856283357	525.78171329
Data Write			20000020H	000008ABH	080004A4H	X	37856290739	525.78181582
Data Read			20000020H	000008ABH	0800049EH	X	37869821542	525.96974364
Data Write			20000020H	000008ACH	080004A4H	X	37869828935	525.96984632
Data Read			20000020H	000008ACH	0800049EH	X	37883453935	526.15908243
Data Write			20000020H	000008ADH	080004A4H	X	37883461371	526.15918571
Data Read			20000020H	00000000H	0800049EH	X	37897062769	526.34809401
Data Write			20000020H	00000001H	080004A4H	X	37897070185	526.34819701
Data Read			20000020H	00000001H	0800049EH	X	37910789370	526.53874125
Data Write			20000020H	00000002H	080004A4H	X	37910796799	526.53884443
Data Read			20000020H	00000002H	0800049EH	X	37924386435	526.72758937
Data Write			20000020H	00000003H	080004A4H	X	37924393833	526.72769212
Data Read			20000020H	00000003H	0800049EH	X	37937995276	526.91660106
Data Write			20000020H	00000004H	080004A4H	X	37938002709	526.91670429
Data Read			20000020H	00000004H	0800049EH	X	37951721877	527.10724829

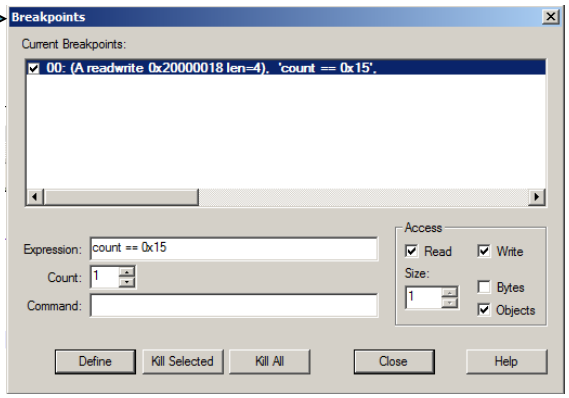
TIP: The PC column is activated when you selected On Data R/W Sample in Step 2. You can leave this unselected to save bandwidth on the SWO pin.

TIP: The ULINK*pro* will give a more sophisticated Instruction Trace window. Watchpoints are described on the next page.

9) Watchpoints: *Conditional Breakpoints*

The STM32 Cortex-M3 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: "**count** == 0x15" without the quotes. Select both the Read and Write Access boxes.
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. Set **count** in Watch 1 window to zero.
9. Click on RUN.
10. When **count** equals 3, the program will stop. This is how a Watchpoint works.
11. You will see **count** incremented to 0x15 in the Logic Analyzer as well as in the Watch window.



12. Note the data write of 0x15 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.
13. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.
14. To repeat this exercise, set **count** to a number less than 15 or click on RUN a few times to get past the trigger value of 0x15.
15. When finished, delete this Watchpoint by selecting Debug and select Breakpoints and select Kill All.
16. Leave Debug mode.

Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Read			20000018H	0000000BH	08000478H	X	1315659252	18.27304517
Data Write			20000018H	0000000CH	0800047EH	X	1315666645	18.27314785
Data Read			20000018H	0000000CH	08000478H	X	1318319299	18.30999026
Data Write			20000018H	0000000DH	0800047EH	X	1318326693	18.31009296
Data Read			20000018H	0000000DH	08000478H	X	1320959378	18.34665803
Data Write			20000018H	0000000EH	0800047EH	X	1320966777	18.34676079
Data Read			20000018H	0000000EH	08000478H	X	1323652705	18.38406535
Data Write			20000018H	0000000FH	0800047EH	X	1323660119	18.38416832
Data Read			20000018H	0000000FH	08000478H	X	1326279472	18.42054822
Data Write			20000018H	00000010H	0800047EH	X	1326286873	18.42065101
Data Read			20000018H	00000010H	08000478H	X	1328912895	18.45712354
Data Write			20000018H	00000011H	0800047EH	X	1328920323	18.45722671
Data Read			20000018H	00000011H	08000478H	X	1331533006	18.49351397
Data Write			20000018H	00000012H	0800047EH	X	1331540443	18.49361726
Data Read			20000018H	00000012H	08000478H	X	1334159773	18.52999685
Data Write			20000018H	00000013H	0800047EH	X	1334167197	18.53009996
Data Read			20000018H	00000013H	08000478H	X	1336813164	18.56684950
Data Write			20000018H	00000014H	0800047EH	X	1336820549	18.56695207
Data Read			20000018H	00000014H	08000478H	X	1339413307	18.60296260
Data Write			20000018H	00000015H	0800047EH	X	1339420705	18.60306535

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.

10) *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. If source code is required, this is included in the Keil RL-ARM™ Real-Time Library which also includes USB, CAN, TCP/IP networking and a Flash File system. 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.

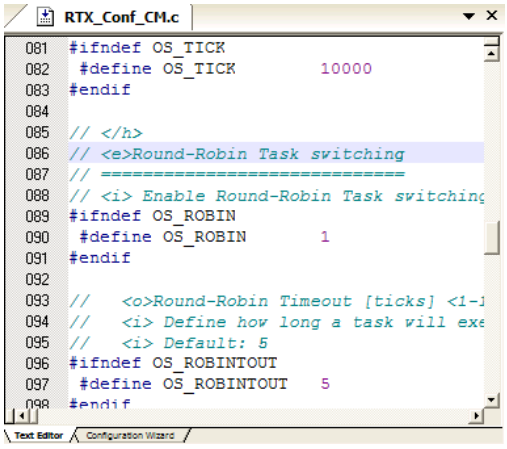
1. This exercise will work with the ST STM3210C-EVAL board except the LCD and LEDs will not be active.
2. Start µVision by clicking on its icon on your desktop if it is not already running.
3. Select Project/Open Project and open the file C:\Keil\ARM\Boards\Keil\MCBSTM32C\RTX_Blinky\Blinky.uvproj.
4. RTX_Blinky uses the ULINK2 as default: if you are using a ULINK*pro*, please configure it as described on page 4 and configure the Serial Wire Viewer on page 13. You only have to do this once for each project – it will be saved in the project file. Select File/Save All.
5. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
6. To program the Flash manually, click on the Load icon. A progress bar will be at the bottom left.
7. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
8. The LEDs will blink indicating the four waveforms of a stepper motor driver. Click on STOP.

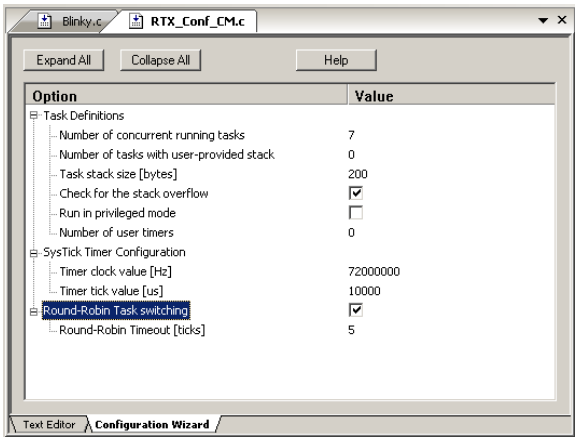
The Configuration Wizard for RTX:

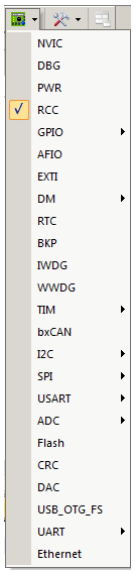
1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open.
2. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See 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.



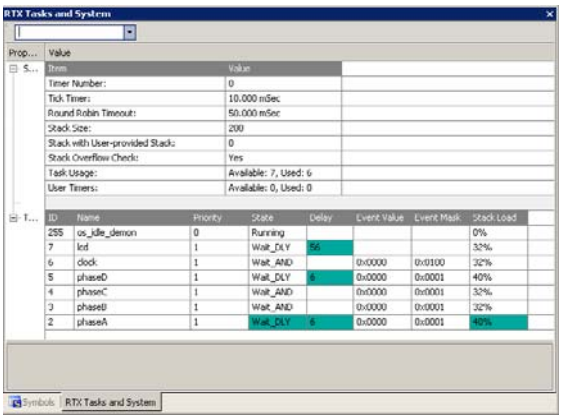




Text Editor: Source Code **Configuration Wizard** **System Viewer**

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

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

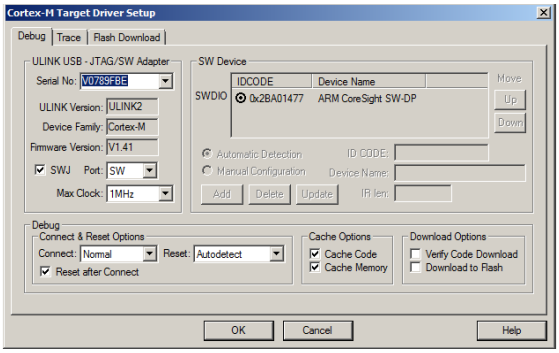


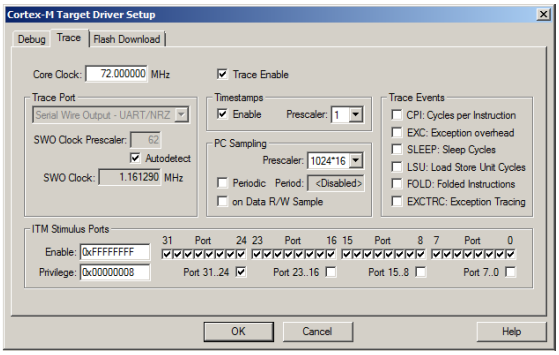
1. Run RTX_Blinky again by clicking on the Run icon.
2. Open Debug/OS Support and select RTX Tasks and System and the window on the right opens up. You might have to grab the window and move it into the center of the screen. These values are updated in real-time using the same technology as used in the Watch and Memory windows.
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.

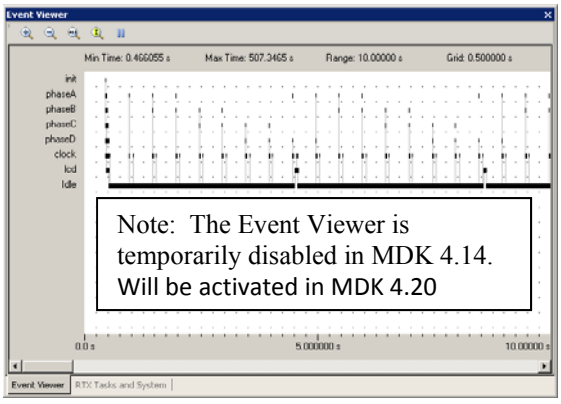
RTX Viewer: Configuring Serial Wire Viewer (SWV):

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

1. Stop the CPU and exit debug mode.
2. Click on the Options icon next to the target box.
3. Select the Debug tab. In the box Initialization File: enter **..\Blinky_ULp\STM32_SWO.ini** or use the Browse icon.
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 72 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 5 seconds by clicking on the ALL and then the + and – icons.







TIP: View/Periodic Window Update must be selected !

TIP: If Event Viewer doesn't work, open up the Trace Records and confirm there are good ITM frames present. Is the 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.

You will not have to stop the program to view this data. No CPU cycles are used. Your program runs at full speed. No instrumentation code need be inserted into your source. You will find this feature very useful ! Remember, RTX is included free with MDK.

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

12) 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:
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 081 and 084 (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.

```
029
030  #define LED_NUM     8
031  const long led_mask[] = { 1<<15,
032
033  unsigned int phasea;
034  unsigned int phaseb;
035  unsigned int phasec;
036  unsigned int phased;
037
038  /*----------------------------
039   * switch LED on
```

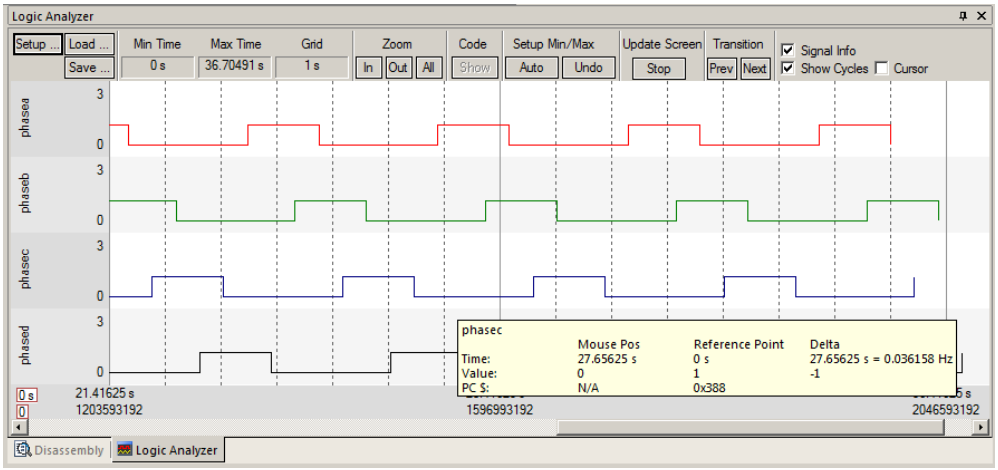
TIP: The Logic Analyzer can display static and global variables, structures and arrays. It can't see locals: just make them static. To see peripheral registers merely read or write to them and enter them into the Logic Analyzer.

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. 

```
077  /*----------------------------------------
078   *        Task 1 'phaseA': Phase A output
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  }
```

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 OUT and In buttons set the range to 20 seconds. Move the scrolling bar to the far right if needed.
15. You will see the following waveforms appear. 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:



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.

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

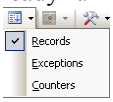
a) Data Reads and Writes: (Note: Data Reads but not Writes are disabled in the current version of µVision**).**

You have configured Serial Wire Viewer (SWV) two pages back in Section 11 under **Configuring the Serial Wire Viewer:**

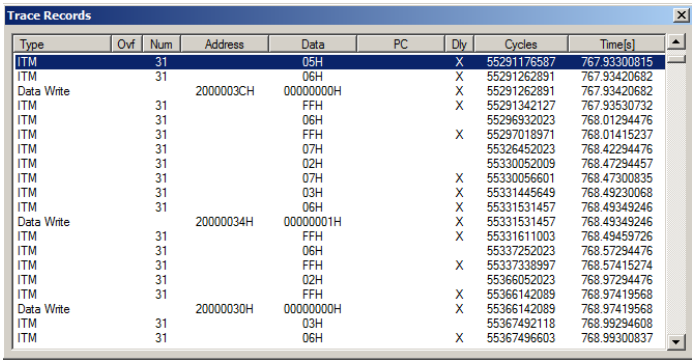
Now we will examine some of the features available to you. SWV works with µVision and a ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link V6 or higher. SWV is included with MDK and no other equipment must be purchased.

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.

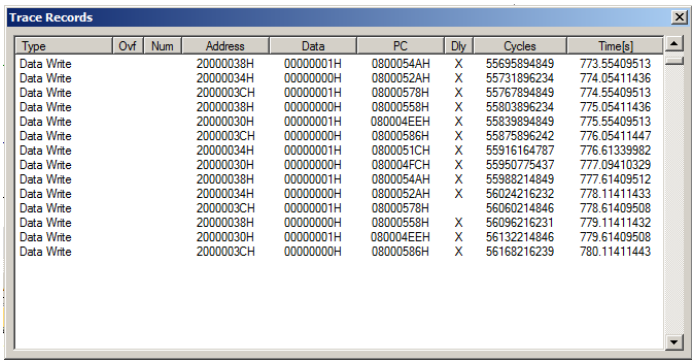
1. Use RTX_Blinky from the last exercise. Enter Debug mode and run the program if not already running.
2. Select View/Trace/Records or click on the Trace icon  and select Records.



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



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. **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.

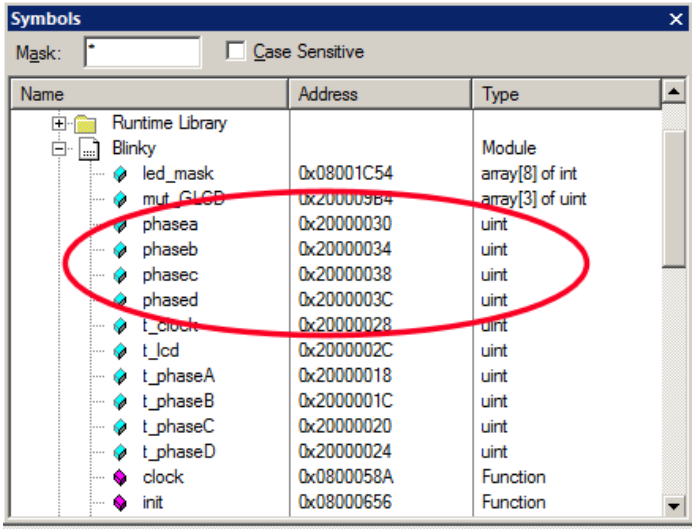


What is happening here ?

1. When variables are entered in the Logic Analyzer (remember phasea through phased ?), the reads and/or 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. 



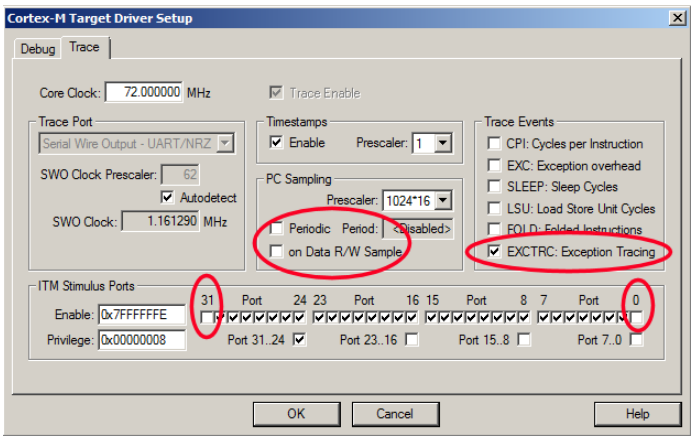
Note: You must have Browser Information selected in the Options for Target/Output tab to use the Symbol Browser.

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 the Cortex-M3 processor has many interrupts and it can be difficult to determine when they are being activated. SWV on the Cortex-M3 processor 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.

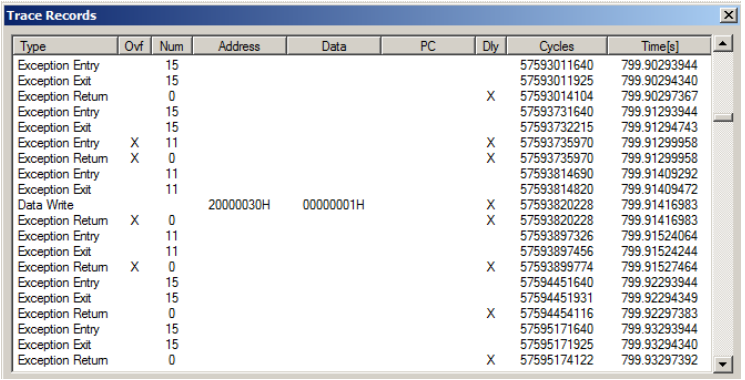


What Is Happening ?

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.

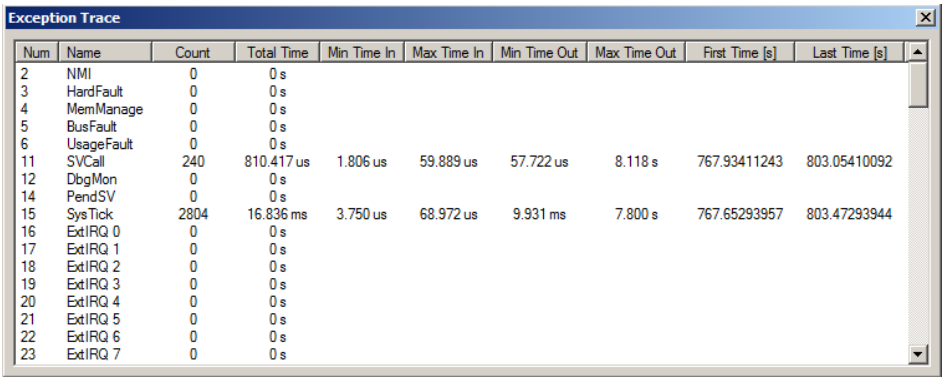
2. Num 11 is SVCall from the RTX calls.
3. Num 15 is the Systick timer.
4. In my example you can see one data write from the Logic Analyzer.
5. Note everything is timestamped.
6. The "X" in Ovf is an overflow and some data was lost. The "X" in Dly means the timestamps are delayed because too much information is being fed out the SWO pin.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15					57593011640	799.90293944
Exception Exit		15					57593011925	799.90294340
Exception Return		0				X	57593014104	799.90297367
Exception Entry		15					57593731640	799.91293944
Exception Exit		15					57593732215	799.91294743
Exception Entry	X	11				X	57593735970	799.91299958
Exception Return	X	0				X	57593735970	799.91299958
Exception Entry		11					57593814690	799.91409292
Exception Exit		11					57593814820	799.91409472
Data Write			20000030H	00000001H		X	57593820228	799.91416983
Exception Return	X	0				X	57593820228	799.91416983
Exception Entry		11					57593897326	799.91524064
Exception Exit		11					57593897456	799.91524244
Exception Return	X	0				X	57593899774	799.91527464
Exception Entry		15					57594451640	799.92293944
Exception Exit		15					57594451931	799.92294349
Exception Return		0				X	57594454116	799.92297383
Exception Entry		15					57595171640	799.93293944
Exception Exit		15					57595171925	799.93294340
Exception Return		0				X	57595174122	799.93297392

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 this 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.

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 !



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						

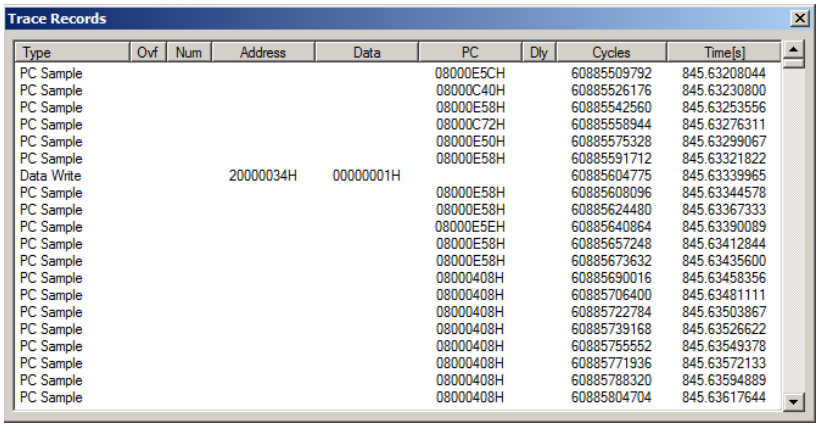
TIP: Num is the exception number: RESET is 1. External interrupts (ExtIRQ), which are normally attached to peripherals, start at Num 16. For example, Num 41 is also known as 41-16 = External IRQ 25. Num 16 = 16 – 16 = ExtIRQ 0.

c) PC Samples:

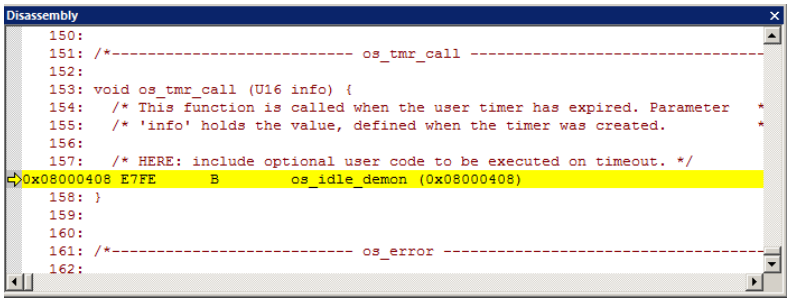
Serial Wire Viewer can display a sampling of the program counter. If you need to see all the PC values, use the ETM trace with a Keil ULINK*pro*. ETM trace also provides Code Coverage, Execution Profiling and Performance Analysis.

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:



6. Most of the PC Samples are 0x0800_0408 which is a branch to itself in a loop forever routine.
7. Stop the program and the Disassembly window will show this Branch:

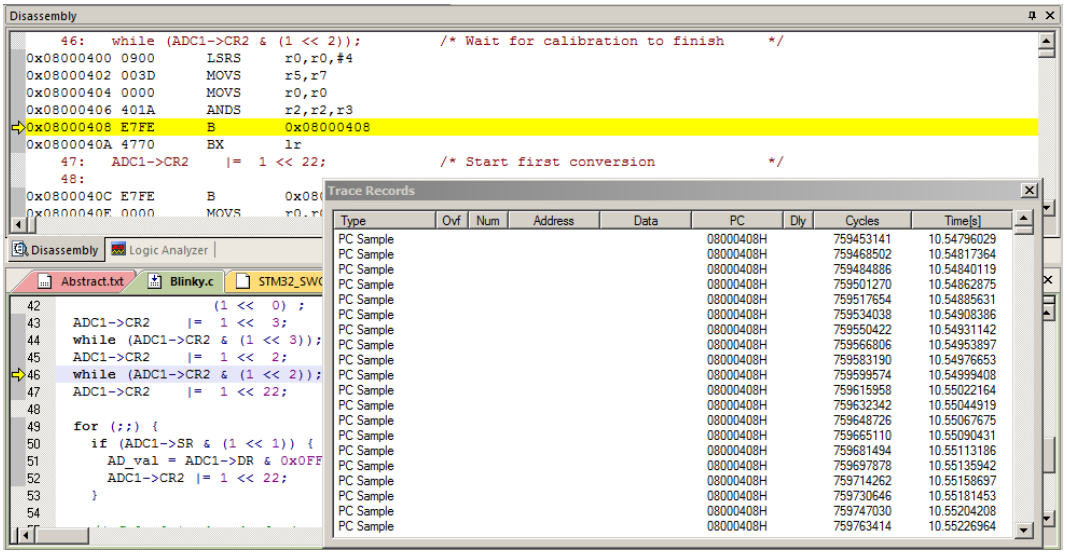


8. Not all the PCs will be captured. Still, PC Samples can give you some idea of where your program is; especially if it is caught in a tight loop like in this case.
9. **Note:** you can get different PC values depending on the optimization level set in µVision.
10. Set a breakpoint in one of the tasks.
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.

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 debugging than PC Samples.

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



14) 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.

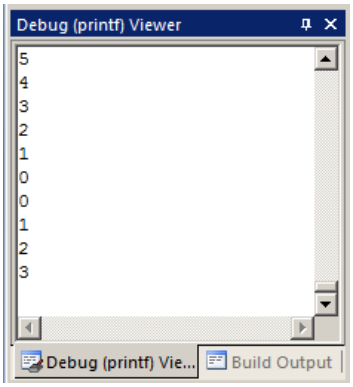
1. Open the project Blinky.uvproj (not RTX Blinky).
2. Add this code to Blinky.c. A good place is near line 17, just after the declaration of **count**.

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

3. In the main function in Blinky.c right after the variable **count** enter these lines after near line 63:

```
ITM_Port8(0) = num + 0x30;     /*  displays count value: +0x30 converts to ASCII */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```

4. Rebuild the source files, program the Flash memory and enter debug mode.
5. Open Debug/Debug Settings and select the Trace tab.
6. Unselect On Data R/W Sample, PC Sample 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 **num** appear.
11. Change the POT and the display rate will change.
12. How else could you detect this interesting effect of the program without SWV ?



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 Data R/W frames.

What Is This ?

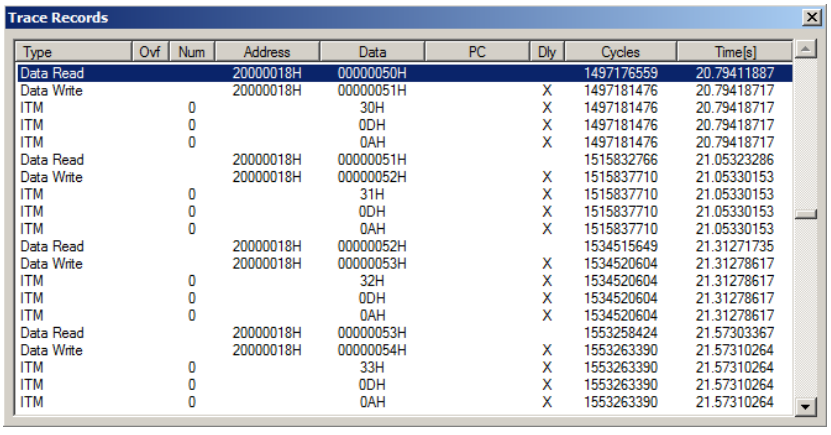
1. ITM 0 frames (Num column) are our ASCII characters from **num** with carriage return (0D) and line feed (0A) as displayed the Data column.
2. All these are timestamped in both CPU cycles and time in seconds.
3. Note the "X" in the Dly column. This means the timestamps might/are not be correct due 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]
Data Read			20000018H	00000050H			1497176559	20.79411887
Data Write			20000018H	00000051H		X	1497181476	20.79418717
ITM		0		30H		X	1497181476	20.79418717
ITM		0		0DH		X	1497181476	20.79418717
ITM		0		0AH		X	1497181476	20.79418717
Data Read			20000018H	00000051H			1515832766	21.05323286
Data Write			20000018H	00000052H		X	1515837710	21.05330153
ITM		0		31H		X	1515837710	21.05330153
ITM		0		0DH		X	1515837710	21.05330153
ITM		0		0AH		X	1515837710	21.05330153
Data Read			20000018H	00000052H			1534515649	21.31271735
Data Write			20000018H	00000053H		X	1534520604	21.31278617
ITM		0		32H		X	1534520604	21.31278617
ITM		0		0DH		X	1534520604	21.31278617
ITM		0		0AH		X	1534520604	21.31278617
Data Read			20000018H	00000053H			1553258424	21.57303367
Data Write			20000018H	00000054H		X	1553263390	21.57310264
ITM		0		33H		X	1553263390	21.57310264
ITM		0		0DH		X	1553263390	21.57310264
ITM		0		0AH		X	1553263390	21.57310264

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

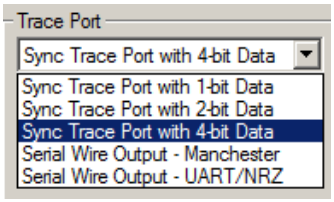
Part C)

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 Instruction Trace. More complete filtering is available.
2) The SWV (Serial Wire Viewer) data is sent out the SWO pin with the ULINK2 using UART encoding. The ULINK*pro* can send SWV data either out the SWO pin using Manchester encoding or through the 4 bit Trace Port. This is done so the ULINK*pro* can 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 data is also sent out this port.

ULINK*pro* offers:

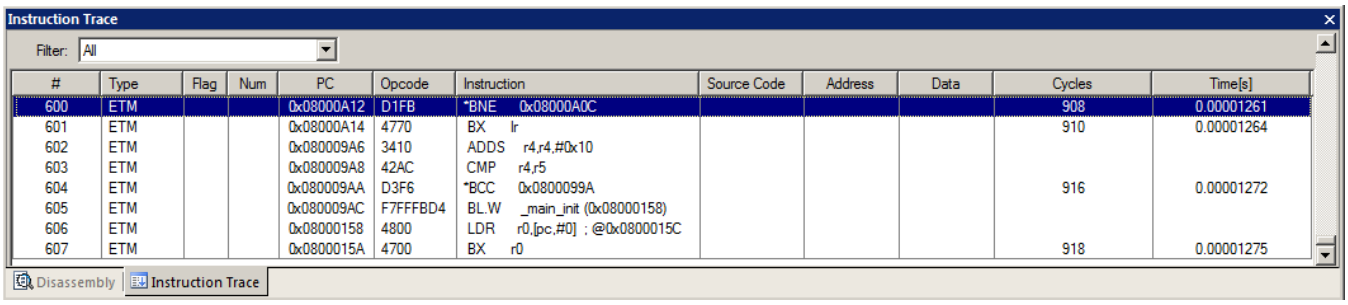


1) Faster Flash programming than the ULINK2.
2) All Serial Wire Viewer features as the ULINK2 does.
3) Adds ETM trace which provides records of all Program Counter values. ULINK2 provides only PC Samples and is not nearly as useful.
4) **Code Coverage**: were all assembly isntructions executed ?
5) **Performance Analysis**: where 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) Blinky_ULp Example:

The project in C:\Keil\ARM\Boards\Keil\MCBSTM32C\Blinky_ULp is preconfigured for the ULINK*pro*.

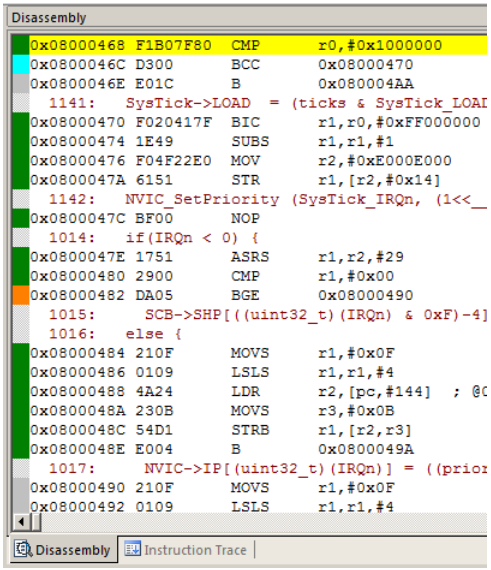
1. Connect the ULINK*pro* to the MCBSTM32C board using the Cortex Debug + ETM connector.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open the file C:\Keil\ARM\Boards\Keil\MCBSTM32C\Blinky_ULp\Blinky.uvproj.
4. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
5. Program the STM32 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
6. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
7. DO NOT CLICK ON RUN YET !!!
8. 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.
9. In this case, # 607 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_045A in my case). Click on Single Step once.



10. The instruction BL.W will display: | 0x0800045A | F7FFFE91 | BL.W SystemInit (0x08000180) | 109: SystemInit(); |
11. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.
12. If you use the Memory window at look at location 0x4, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x8000_0164 + 1 (+1 says it is a Thumb instruction).

2) Code Coverage:

13. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
14. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin:
15. 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 here.
6. RED: Breakpoint is set here.
7. Next instruction to be executed.

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

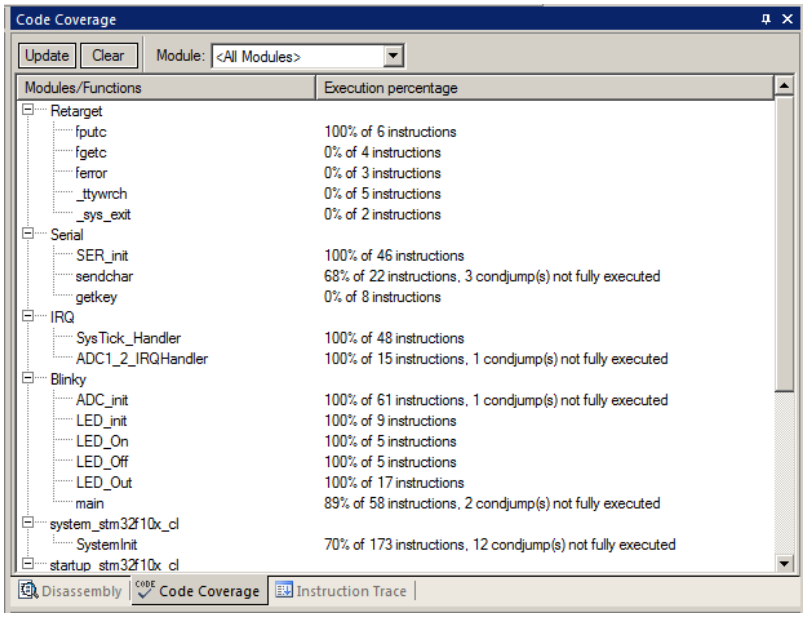
Why was 0x0800_046E never executed ? Or 0x0800_0490 ? You should devise tests to execute these instructions so you can test the effects.

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. You can Clear and Update this window with the buttons provided.

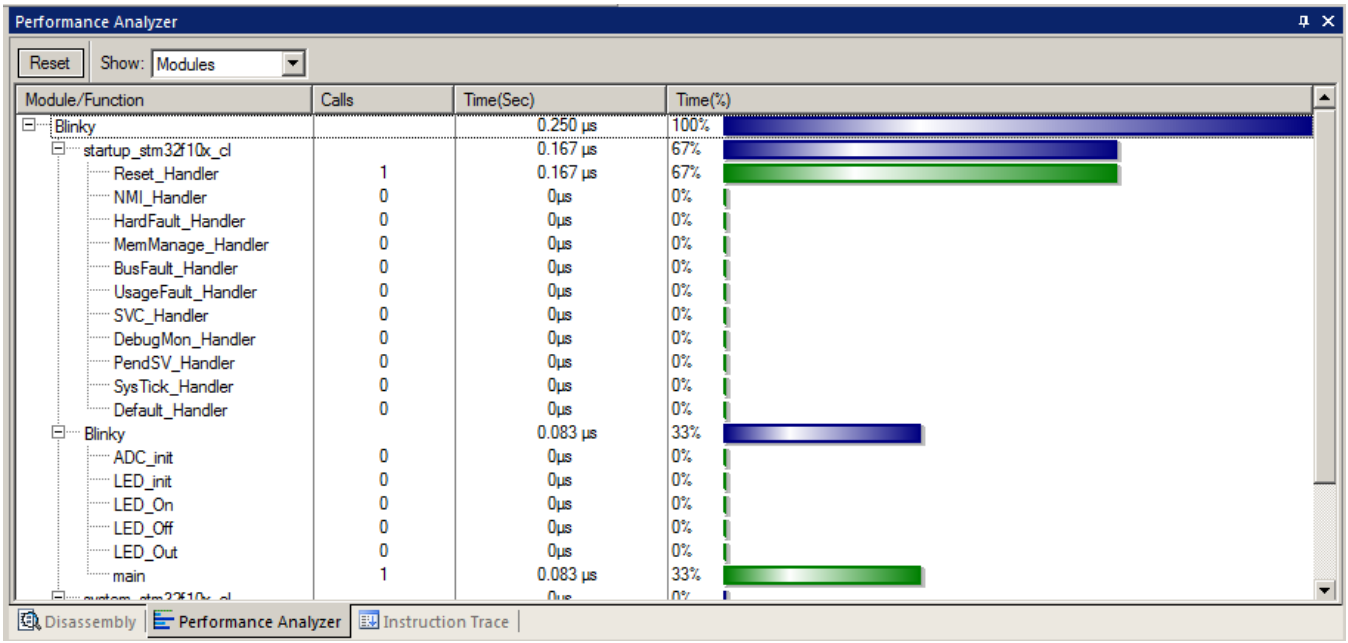


3) Performance Analysis (PA):

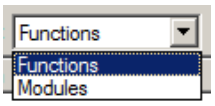
Performance Analysis tells you how much time was spent in each function. The data can be provided by either the SWV PC Samples or the ETM. If provided by the SWV, the results will be statistical and more accuracy is improved with longer runs. Small loops could be entirely missed. ETM provides complete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

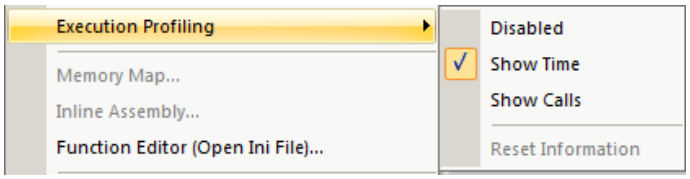


TIP: You can also click on the RESET icon  but the processor will stay at the initial PC and will not run to main(). You can type **g, main** in the Command window to accomplish this.

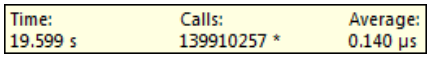
When you click on RESET, the Initialization File .ini will no longer be in effect and this can cause SWV and/or SWV to stop working. Exiting and re-entering Debug mode executes the .ini script again.

4) Execution Profiling:

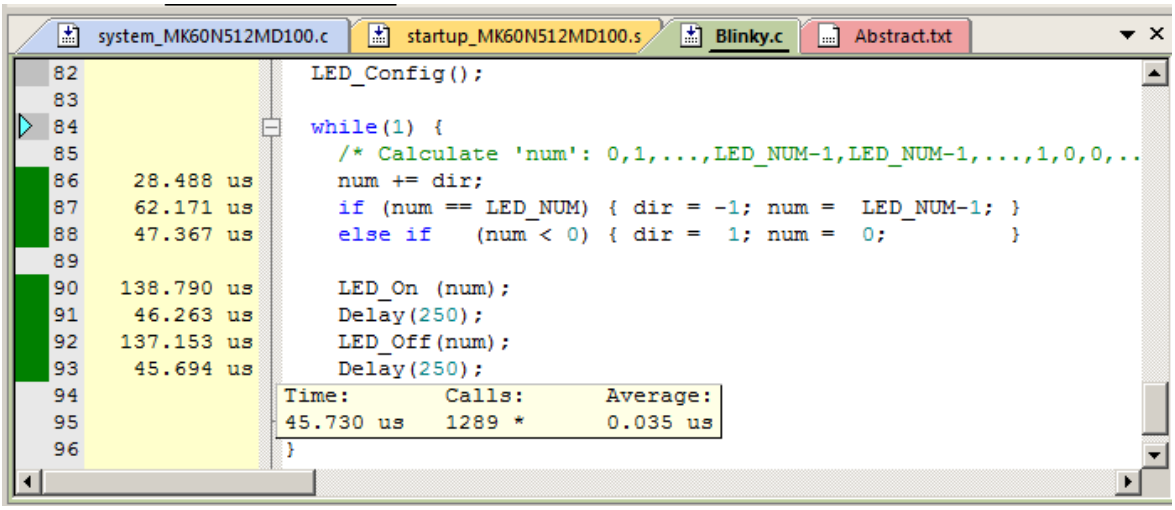
Execution Profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.
The µVision simulator also provides Execution Profiling.



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



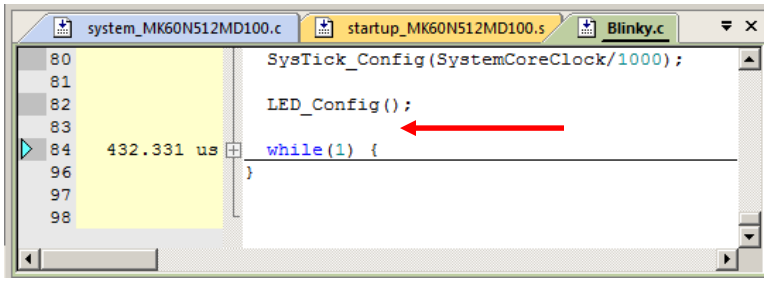
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



Outlining:

Each place there is a small square with a "-" sign  can be collapsed down to compress the associated source files together.

1) Click in the square near the while(1) loop near line 84 as shown here:
2) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
3) Click on the + to expand it.
4) Stop the program and exit Debug mode.



5) In-the-Weeds Example:

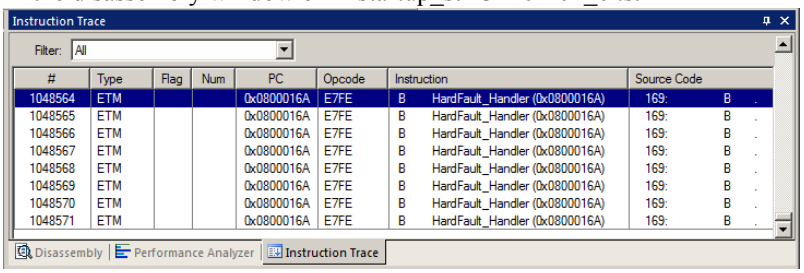
Some of the hardest problems to solve are those when a crash has occurred and you have no clue what caused this. You only know that it happened and the stack is corrupted or provides no useful clues. Modern programs tend to be asynchronous with interrupts and RTOS task switching plus unexpected and spurious 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 Bus Fault occurs in our example, the CPU will end up at 0x800_016A as shown in the disassembly window below. This is the Bus Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. The Instruction Trace window below shows this branch forever. This is not useful.

This exception vector is found in the file startup_stm3210f10x_cl.s. If we set a breakpoint by double-clicking on the Hard Fault handler and run the program: at the next Bus Fault event the CPU will again jump to the Hard Fault handler.

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

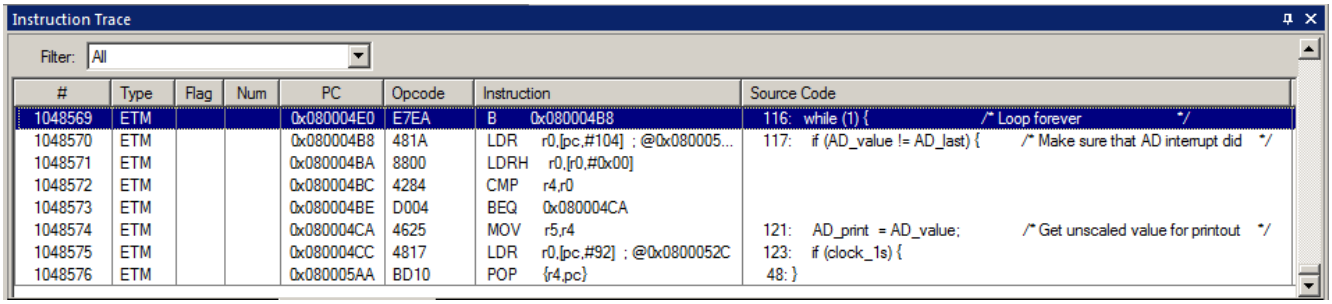
1. Using the Blinky example from the previous exercise, exit and re-enter Debug mode to clear the trace.
2. Locate the Hard fault vector near line 169 in the disassembly window or in startup_stm3210f10x_cl.s.
3. Set a breakpoint at this point. A red block will appear.
4. Run the Blinky example for a few seconds and click on STOP.
5. In the Disassembly window, scroll down until you find a POP instruction. I found one at 0x8000_05AA.
6. Right click on the POP instruction and select Set Program Counter. This will be the next instruction executed.
7. Click on RUN and immediately the program will stop on the Hard Fault exception branch instruction.
8. Examine the Instruction Trace window and you find this POP plus everything else that was previously executed.



How this was done:

To create the hard fault, a POP instruction was executed out of order. A breakpoint was set on the Hard Fault Handler location 0x800_016A. The program was run and stopped. A POP was located by scrolling down through the disassembly window. The PC was set to the POP at this location by right clicking on it and selecting Set PC. Click on RUN and the CPU immediately goes to the Hard Fault Handler and stops because the stack had a non-valid return PC address to be popped.

TIP: You might have to do a step-out-of to clear out all other running interrupt routines running otherwise you will just return from an interrupt rather than crash. See the µVision Call Stack window for information on interrupts running.



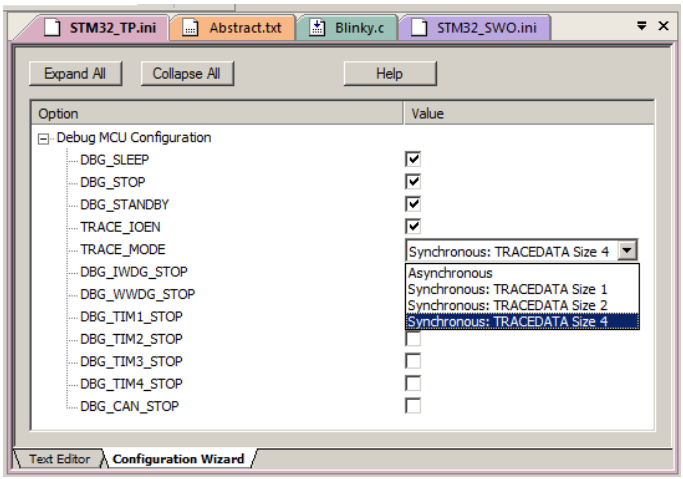
The frames above the POP are a record of all previous instructions executed and tells you the complete program flow.

6) Configuring the ULINK*pro* ETM Trace:

The ULINK*pro* was configured for SWV operation using the SWO pin and Manchester encoding on page 11. We will activate ETM trace here.

1) Select a project. Please use MCBSTM3220F Blinky.
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: an ini file will be there. Click on the Edit box. 

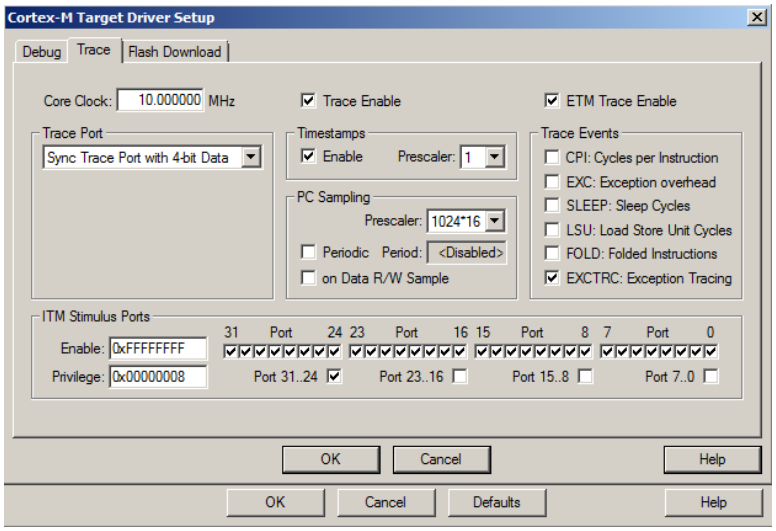
The specified ini file will open.
6) Click OK. At the bottom of the ini file, click on the Configuration Wizard tab.
7) Expand the menu and select Synchronous: Trace Data Size 4 as shown here:



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: No need to enter anything. ULINK*pro* determines this automatically
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.

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.



7) 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.

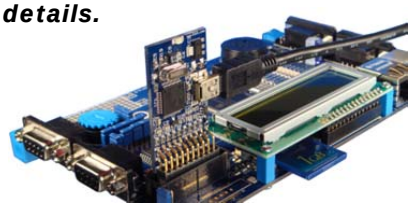
For complete information on CoreSight for the Cortex-M3: Search for **DDI0314F_coresight_component_trm.pdf** on www.arm.com.

8) Keil Products:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite™ (Evaluation version) $0
- ***NEW !!*** MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit) - $3,200
- MDK-Standard™ (unlimited compile and debug code and data size) - $4,895
- MDK-Professional™ (Includes Flash File, TCP/IP, CAN and USB driver libraries) $9,500

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



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* - $1,250 – Cortex-M*x* SWV & ETM trace.
- *MDK also supports ST-Link V2 and Segger J-Link Debug adapters.*

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

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

Keil provides free DSP libraries for the Cortex-M3 and Cortex-M4.

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

All products are available from stock.

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

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

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

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 for your local distributor see: www.keil.com/distis/

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

For more information:

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

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

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

For the latest version of this document, contact the author, Keil Technical support or www.keil.com.