

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_280.asp

Introduction:

The purpose of this lab is to introduce you to the STMicroelectronics Cortex™-M7 processor using the ARM® Keil® MDK toolkit featuring the IDE µVision®. We will use the Serial Wire Viewer (SWV) and the on-board ST-Link V2 Debug Adapter. You can also use a ULINK2 or a J-Link. For ETM instruction trace: use a Keil ULINKpro. See www.keil.com/st.

Keil MDK supports and has examples for most ST ARM processors. Check the Keil Device Database® on www.keil.com/dd2 for the complete list which is also included in MDK: in the µVision main menu, select Project/Select Device for target...

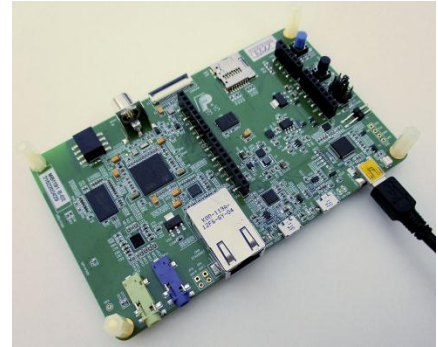
MDK supports the ELF/DWARF format. GCC and LLVM executables can be directly imported or used in µVision.

Linux: Cortex-A processors running Linux, Android and no OS are supported by ARM DS-5™. www.arm.com/ds5.

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

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM® Compiler toolchain. MDK is a turn-key "out-of-the-box" solution.
2. **Dynamic Syntax Checking** on C source lines.
3. **STM32CubeMX** compatible. MDK 5 projects are created.
4. **Compiler Safety Certification Kit:** www.keil.com/safety/
5. TÜV certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
6. **Keil Middleware:** TCP/IP, USB and Flash File. Easily configured.
7. A full feature Keil RTOS called RTX is included with MDK. RTX has a BSD license and source code is provided. See www.keil.com/rtx/.
8. Two RTX Kernel Awareness windows. They are updated in real-time.
9. CoreSight™ Serial Wire Viewer and ETM trace capability is included.
10. Keil Technical Support is included for one year and is easily renewable. This helps you get your project completed.
11. Affordable perpetual and term licensing. Contact Keil sales for pricing, options and current special offers.



This document details these features:

1. Serial Wire Viewer (SWV) data trace using ST-Link V2. Real-time is tracing updated while the program is running.
2. Real-time Read and Write to memory locations for Watch, Memory and Peripherals windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
3. Six Hardware Breakpoints (they can be set/unset on-the-fly) and four Watchpoints (also known as Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTX RTOS that updates while your program is running.
5. A DSP example program using ARM CMSIS-DSP libraries. www.arm.com/cmsis
6. ETM instruction trace including Performance Analysis, Code Coverage and a Hard Fault error demonstration.
7. Create a project with STM32CubeMX, Keil Software Packs, MDK Middleware or ST Standard Peripheral Libraries.

Serial Wire Viewer (SWV):

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM (printf), CPU counters and timestamps. This information comes from the ARM CoreSight™ debug module integrated into STM32 CPU. SWV does not steal any CPU cycles and is completely non-intrusive. (except for the ITM Debug printf Viewer).

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

ETM records and displays all instructions that were executed. This is very useful for debugging program flow problems such as "going into the weeds" and "how did I get here?" ETM requires a ULINKpro and an STM32756G_EVAL board.

1. Three Methods used to create μ Vision Projects:	3
2. CoreSight Definitions:	3
3. Overview: Keil MDK Software: MDK 5	4
4. Keil MDK Core Software Download and Installation:	4
5. Software Pack Download and Install Process:	4
6. Install the MDK 5 Blinky Example from the Software Packs:	5
7. Install the Blinky_BM, RTX_Blinky and DSP Examples from the web:	5
8. Getting Started Guide MDK 5 manual:	5
9. STMicroelectronics evaluation boards:	5
10. Install the ST-Link V2 USB Drivers:	6
11. Testing the ST-Link V2 Connection:	6
12. Software Pack Version Selection and Manage Run-Time Environment:	7
13. Blinky example using the STM32F746G Discovery board:	8
14. Hardware Breakpoints:	8
15. Call Stack & Locals window:	9
16. Watch and Memory windows and how to use them:	10
17. View Variables Graphically with the Logic Analyzer (LA):	11
18. Watchpoints: <i>Conditional Breakpoints</i> (Access Breakpoints)	12
19. ITM (Instrumentation Trace Macrocell):	13
20. Printf with ITM (Instrumentation Trace Macrocell):	14
21. RTX_Blinky example: Keil RTX RTOS:	15
22. RTX Kernel Awareness using RTX Viewer:	16
23. Logic Analyzer: View variables real-time in a graphical format:	17
24. DSP Sine Example using ARM CMSIS-DSP Libraries	18
25. Keil Middleware: Network (TCP/IP), Flash File, USB, Graphics	21
26. Creating your own MDK 5 project from scratch:	22
27. Creating your own RTX RTOS project from scratch:	25
28. Adding a new Thread to your RTX project:	26
29. Using Event Viewer to examine the timing of RTX:	27
30. Using STM32CubeMX to create a simple Blinky Program:	28
31. ETM Instruction Trace: with ULINKpro:	30
1) Configure ETM	31
2) Searching for Trace Frames	33
3) Trace Triggers:	33
4) Code Coverage (CC):	35
5) Performance Analysis (PA):	37
6) "In-the-Weeds" example: Finding a A Hard Fault cause:	39
32. Serial Wire Viewer (SWV) and ETM Summary and SWV Configuration:	40
33. Document Resources:	42
34. Keil Products and contact information:	43

Note: MDK 5.17 and Software Pack STM32F7xx_DFP 2.3.0 were used in the exercises in this document.

1) Three Methods used to create μ Vision Projects:

There are three main methods to create your own μ Vision projects:

- 1) **STM32CubeMX**. This configures your processor and exports a μ Vision project in MDK 5 format. See Page 28. STM32CubeMX can be downloaded from www.st.com/stm32cubemx/
- 2) **Standard Peripheral Libraries** from ST. STM32CubeF7. Contains extensive examples and source code for Keil MDK 5. These libraries are available from www.st.com/stm32cubemx/
- 3) **μ Vision Software Packs, examples and Keil Middleware**. A Software Pack includes examples and files that you can use. See Page 21 and www.keil.com/pack/doc/STM32Cube/General/html/index.html

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

MDK 5 and MDK 4 projects: MDK 5 uses Software Packs and MDK 4 does not. This tutorial uses MDK 5 projects which have a filename extension .uvprojx. Legacy MDK 4 projects (with an extension .uvproj) can be converted to MDK 5: Select **Project/Manage/Migrate to Version 5 format...** You can also use MDK 5 Legacy support for older processors.

ELF/DWARF: The ARM compiler produces a .axf file which is ELF/DWARF compliant. μ Vision can load similar compiler output such as from GCC with all debug information visible. You can also use GCC as your compiler of choice in μ Vision.

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

Cortex-M0 and Cortex-M0+ have only features 2) through 3) plus 11 and 12 implemented. Cortex-M3, Cortex-M4 and Cortex-M7 can have all features listed implemented. MTB is normally found on Cortex-M0+. It is possible some processors have all features except ETM Instruction trace and the 4 bit trace port. Consult your specific STMicroelectronics datasheet to determine its specific feature set.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except Boundary Scan is not possible. SWD is referenced as SW in the μ Vision Cortex-M Target Driver Setup. The SWJ box must be selected if it is displayed. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDIO shares the same pin as SWO. The SWV data normally comes out the SWO pin and a conflict will arise.
3. **DAP:** Debug Access Port. This is a component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of DAP is the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. μ Vision uses the DAP to update memory, watch and RTOS kernel awareness windows in real-time while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no source code stubs are needed. You do not need to configure or activate DAP. μ Vision configures DAP when you select a function that uses it.
4. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
5. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
6. **Trace Port:** A 4 bit port that ULINK pro uses to output ETM frames and optionally SWV (rather than SWO pin).
7. **ITM:** Instrumentation Trace Macrocell: As used by μ Vision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. μ Vision uses Port 0 for printf and Port 31 for the RTX Event Viewer. The data can be saved to a file.
8. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK pro provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis.
9. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK pro . ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
10. **MTB:** Micro Trace Buffer. Found only on Cortex-M0+ processors. A portion of the device internal RAM is used for an instruction trace buffer. STM32 Cortex-M3, M4 and M7 processors provide ETM trace instead.
11. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. Cortex-M3, M4 and M7 processors usually have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs.
12. **WatchPoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can have 2 Watchpoints. These are conditional breakpoints. They stop the program when a value is read and/or written to an address or variable.

3) Overview: Keil MDK Software: MDK 5 This document uses MDK 5.17 or later.

MDK 5 uses Software Packs to distribute processor specific software, examples and middleware. MDK 5 Core is first installed and you then download the Software Packs you require from the web. They can also be imported manually. You no longer need to wait for the next version of MDK or install patches to get the latest processor specific files.

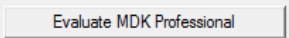
4) Keil MDK Core Software Download and Installation:



1. Download MDK Core from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default folder C:\Keil_v5.
3. We recommend you use the default folders for this tutorial. We will use C:\MDK\ for the examples.
4. If you install MDK or the examples into different folders, you will have to adjust for the folder location differences.

MDK Licensing:

1. You can use the evaluation version (MDK-Lite) for this lab. No license is needed.
2. You can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible:
3. This gives you access to the Keil Middleware as well as unlimited code size compilation. Contact Keil sales to extend this license for evaluation purposes.



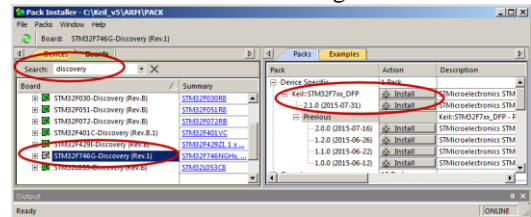
5) Software Pack Download and Install Process:

1) Start μ Vision and open Pack Installer: After the first MDK install is complete, μ Vision and Software Packs will startup. Otherwise, follow Steps 1 and 2 below.

1. Connect your computer to the Internet. This is normally needed to download the Software Packs.
2. Start μ Vision by clicking on its desktop icon.
3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
4. This window opens up: Select the Boards tab. Type *discovery* in the Search box to filter the listings:

TIP: What is selected on the left side in the Devices and Boards tabs filters what is displayed on the right side in the Packs and Examples tabs.

5. Select STM32F746G-Discovery as shown here: You can also select individual processors under the Devices tab.
6. Note: "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet.

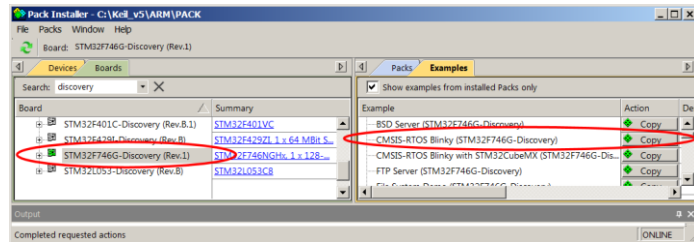


TIP: If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or to refresh once you have connected to the Internet.




2) Install The STM32F7 Software Pack:

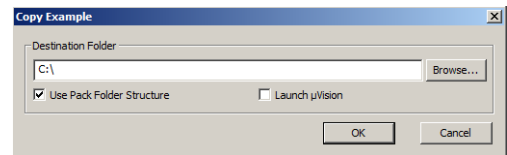
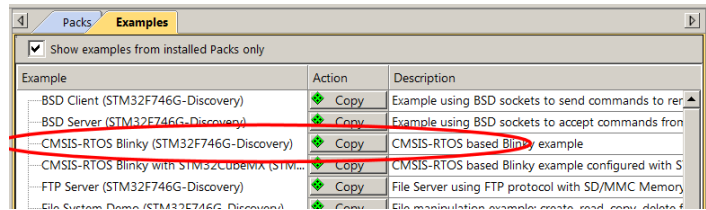
1. Click on the Packs tab. Initially, the Software Pack ARM::CMSIS is installed by default.
2. Select Keil::STM32F7xx_DFP as shown above and click on Install. The latest Pack will download and install to C:\Keil_v5\ARM\Pack\Keil\ST\ by default. This download can take two to four minutes.
3. Its status will then be indicated by the "Up to date" icon:
4. We will install the examples on the next page: Please leave Pack Installer open.

What is a Software Pack made up of ?
 A Pack is an ordinary zip file with a .pack filename extension. This file can contain various headers and other source files, Flash programming algorithms, examples, RTOS, Middleware, documentation and Board Support Packages (BSP). A .pdsc file in XML format describes the Pack contents.




6) Install the MDK 5 Blinky Example from the Software Pack:

1. You must still have Pack Installer open from the previous page. If not, open it now: 
2. Select the Boards tab. In the Search: box, enter *discovery* to filter the entries displayed.
3. Select STM32F746G-Discovery in the Boards tab.
4. Select the Examples tab:
5. In the Example tab: select CMSIS-RTOS Blinky:
6. Select Copy : Do not select the Blinky example using STM32CubeMX.
7. The Copy Example window below opens up:
8. In this window, select Use Pack Folder Structure: Unselect Launch μ Vision:
9. Type in C:\ as shown to the right: Click OK to copy into  C:\MDK\Boards\ST\STM32F746G_Discovery\.





TIP: When you specify C:\ in the Copy Example window, μ Vision uses information in the Software Pack to create the rest of the folder tree. In this case, it creates the folders MDK\Boards\ST\STM32F746G_Discovery\ in C:\.

10. Close Packs Installer. You can open it any time by clicking on its icon. 
11. If a window opens stating the Software Packs folder has been modified. Reload Packs?, select Yes.


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

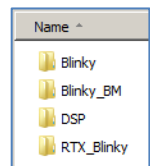
TIP: An "Update" icon means there is an updated Software Pack available for download.  Update

The "Up to date" icon means exactly that: 

Select Packs/Check for Updates or this icon  in the Pack Installer to refresh this list. This is not done automatically. You must be connected to the Internet. Remember a Pack can be imported manually as well as from the web. You can create your own Pack for unannounced devices or for your own proprietary boards or source code and distribute it privately.

7) Install the Blinky_BM, RTX_Blinky and DSP Examples from the web:

1. Obtain the example software zip file from www.keil.com/appnotes/docs/apnt_280.asp.
2. Extract this into the directory C:\MDK\Boards\ST\STM32F746G_Discovery\
3. The Blinky_BM, DSP and RTX_Blinky folders will be created along with the Blinky folder: 



8) Getting Started Guide MDK 5 manual: Obtain this useful book here: www.keil.com/gsg

9) STMicroelectronics evaluation boards:

This tutorial supports two STM32F7 boards: many others can be used with slight modifications.

STM32F746G-Discovery: As pictured on the front page. This tutorial uses this board except for the ETM exercises. Blinky and Middleware examples are provided. You do not need any debug adapters: just the Discovery board, a USB cable and MDK 5 installed on your PC. You connect to the on-board ST-Link V2 debug adapter with a USB cable connected to CN14 and to your PC which also powers the board.

STM32756G_EVAL: This is used in the ETM examples in this tutorial. Blinky and Middleware examples are provided. This has a 20 pin CoreSight connector for ULINK $_{pro}$. ETM has 4 bit signal + a clock + JTAG/SWD signals.

Custom or other boards: Using MDK with other STM32F7 boards is easy. For information concerning debug connectors visit www.keil.com/coresight/coresight-connectors/

On-board Debug Adapters: To install your own on-board debug adapter on a custom board: see CMSIS-DAP: www.keil.com/pack/doc/CMSIS/DAP/html/

10) Install the ST-Link V2 USB Drivers: (this is not necessary if the test below passes)





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

Installing the ST-Link USB Drivers: *This step normally needs to be done just once !*

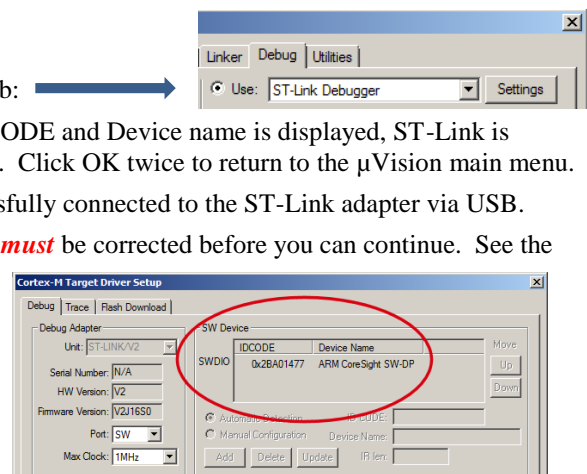
1. Do not have the Discovery board USB port connected to your PC.
2. The USB drivers must be installed manually by executing stlink_winusb_install.bat. This file is found in C:\Keil_v5\ARM\STLink\USBDriver\. Double-click on this file. The drivers will install.
3. Connect a PC to the Discovery board USB CN14. The USB drivers will now finish installing in the normal fashion.

Upgrading the ST-Link V2 Firmware on the board: The ST-Link V2 firmware updater utility ST-LinkUpgrade.exe is located here: C:\Keil_v5\ARM\STLink\. It is a good idea to upgrade the ST-Link firmware. Find this file and double click on it. It is easy to use. It will check and report the current firmware version. If you experience trouble especially with Serial Wire Viewer (SWV), try updating to the latest drivers and firmware.

11) Test the ST-Link V2 Connection: (Optional)

1. Start μ Vision  if it is not already running.
2. Connect the Discovery board to your PC with a USB cable as shown on the first page of this tutorial.
3. If the ST-Link USB drivers are installed correctly, you should hear the usual USB connected dual-tone. If not, you might have to install the drivers manually. See the directions above.
4. Two red LEDs will light: LD7 and LD2 (PWR).
5. Select Project/Open Project.
6. Select the Blinky project C:\MDK\Boards\ST\STM32F4-Discovery\Blinky\Blinky.uvprojx. (or any project)
7. Select STM32F407 Flash as shown here: 
8. Select Target Options  or ALT-F7 and select the Debug tab: 
9. Click on Settings: and the window below opens up: If an IDCODE and Device name is displayed, ST-Link is working. You can continue with this tutorial on the next page. Click OK twice to return to the μ Vision main menu.
10. A number in the Serial Number: box means μ Vision is successfully connected to the ST-Link adapter via USB.
11. If nothing or an error is displayed in this SW Device box, this *must* be corrected before you can continue. See the instructions above: Installing the ST-Link USB Drivers:
12. Once you see a proper display, your ST-Link USB drivers are installed properly. Click OK twice to exit the Target Options windows and continue to the next page.

TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window. ST-Link V2 does not support JTAG mode, only SWD.



LED LD7 indication:

LED is blinking RED: the start of USB enumeration with the PC is taking place but not yet completed.

LED is RED: communication between the PC and ST-LINK/V2 is established (end of enumeration). μ Vision is not connected to ST-Link (i.e. in Debug mode).

LED is GREEN: μ Vision is connected in Debug mode and the last communication was successful.

LED is blinking GREEN/RED: data is actively being exchanged between the target and μ Vision.

LED is off, except for a brief RED flash while entering Debug mode and a brief flash when clicking on RUN happens when the SWV trace is enabled in μ Vision.


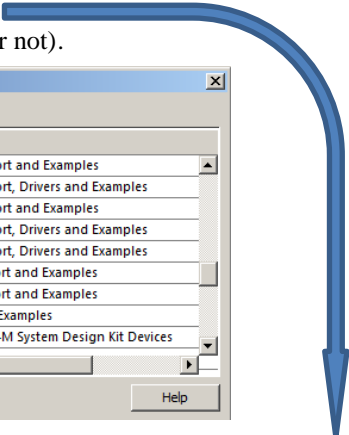
No Led: ST-LINK/V2 communication with the target or μ Vision has failed. Cycle the board power to restart.

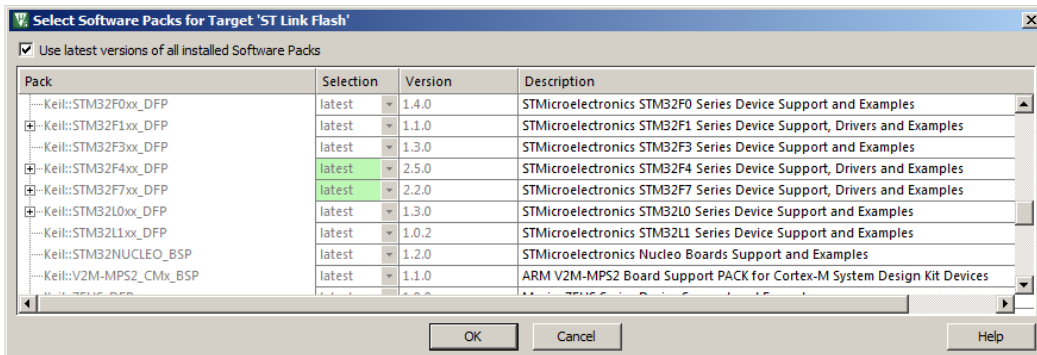
12) Software Pack Version Selection and Manage Run-Time Environment:

This section is presented for reference only.

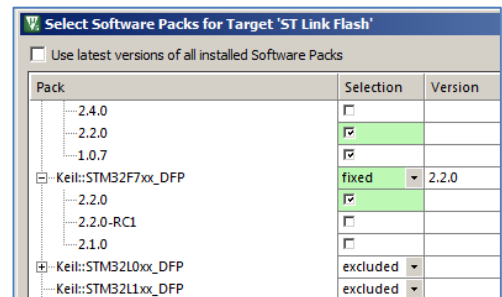
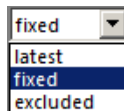
Select Software Pack Version:

This μ Vision utility provides the ability to choose various software pack versions installed in your computer. You can select the various versions you want to use. You are able to "freeze" software versions for your development purposes.


2. Select Project/Open Project in μ Vision.
3. Open the project Blinky.uvprojx located in C:\MDK\Boards\ST\STM32F746G_Discovery\Blinky\.
4. Open Select Software Pack by clicking on its icon: 
5. This window opens up. Note **Use latest versions ...** is selected. The latest version of the Pack will be used. The Packs used in the active project in μ Vision are highlighted in green as shown below:
6. Unselect this setting and the window changes similar to that shown below right:  You probably will see a different screen depending on what Packs are installed (or not).



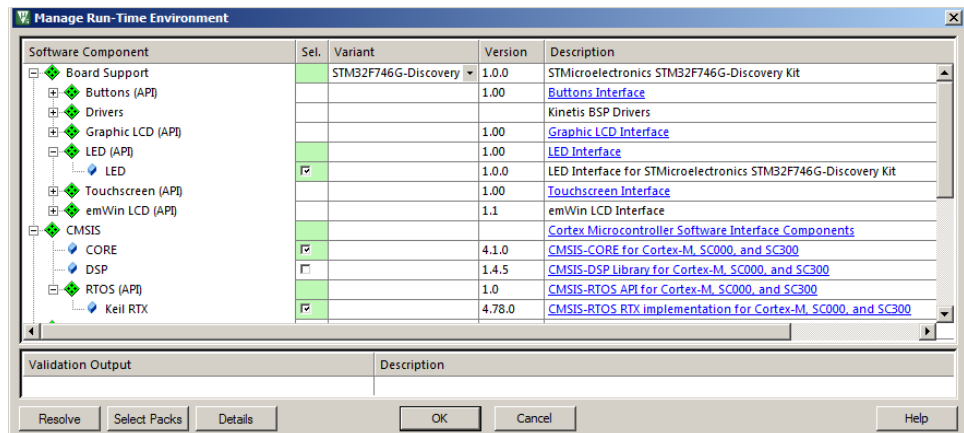
7. Expand the header Keil::STM32F7xx_DFP. Note the various versions visible. You probably will see a different screen depending on what Packs are installed or not.
8. .Select excluded and see the options as shown:
9. If you wanted to use V 2.1.0, you would select fixed and then select the check box opposite 2.1.0.
10. Re-select Use latest versions...
11. Close this window.



Manage Run-Time Environment:






1. Click on the Manage RTE icon:  The next window opens: This includes the board support package (BSP), Keil Middleware, selected open source software, RTX and CMSIS drivers for various peripherals. This window is created by the .pdsc file in the Software Pack.

2. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project.
3. Components selected are shown in green.
4. Do not make any changes. Click Cancel to close this window.



13) Blinky example program using the STM32F746G Discovery board:

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

1. Start μ Vision by clicking on its desktop icon.  Connect to your PC with a USB cable to USB ST-Link CN14.
2. Select Project/Open Project. Open C:\MDK\Boards\ST\STM32F746G_Discovery\Blinky_BM\Blinky.uvprojx
3. By default, ST-Link is selected. If this is the first time you have run μ Vision and the Discovery board, you might have to install the USB drivers. See the configuration instructions on page 6.
4. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears. The Flash will be erased and programmed and progress will be indicated in the Output Window.
6. Click on the RUN icon.  Note: you stop the program with the STOP icon. 





The LED LD1 on the STM32F7 Discovery board will blink.
Press the blue USER button and it will blink slower.

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

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

14) Hardware Breakpoints:

The STM32F7 has six hardware breakpoints that can be set or unset on the fly while the program is running.

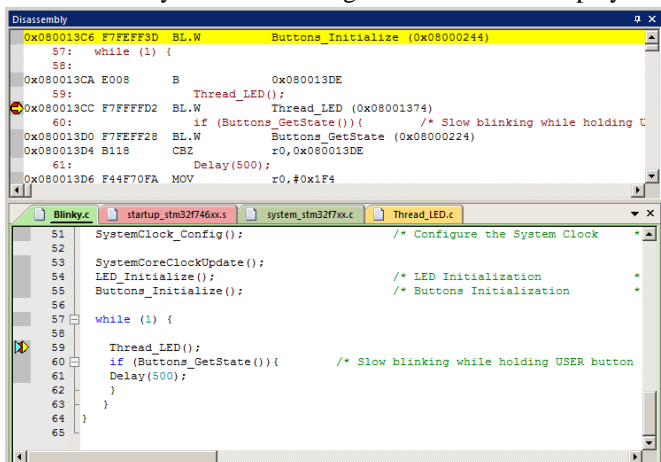
1. With Blinky running, in the Blinky.c window, click on a darker grey block in the left margin on a line in main() in the while loop. Between near lines 57 through 61 will suffice.
2. A red circle will appear and the program will stop.
3. Note the breakpoint is displayed in both the disassembly and source windows as shown below:
4. You can set a breakpoint in either the Disassembly or Source windows as long there is a gray rectangle indicating the existence of an assembly instruction at that point.
5. Every time you click on the RUN icon  the program will run until the breakpoint is again encountered.
6. You can also click on Single Step (Step In) , Step Over  and Step Out .
7. **Remove all breakpoints when you are done for the next exercise by clicking on them again.**

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

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

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

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




15) Call Stack + Locals Window:

Local Variables:

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

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

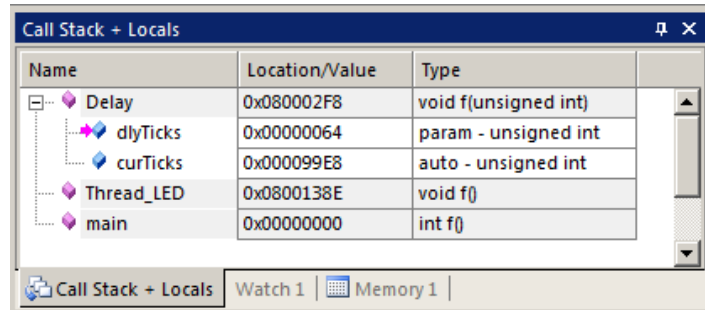
1. Run and Stop Blinky . Click on the Call Stack + Locals tab.
2. You will probably be stopped in the Delay() function as shown here in the Call Stack + Locals window:




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

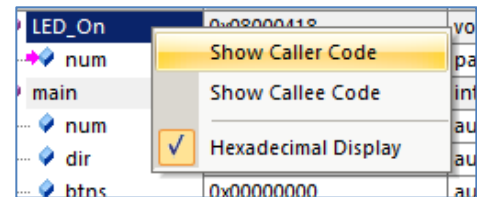
When a function exits, it is removed from the list.

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

This table is active only when the program is stopped.



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



TIP: You can modify a variable value in the Call Stack & Locals window when the program is stopped.

TIP: This is standard “Stop and Go” debugging. ARM CoreSight debugging technology can do much better than this. CoreSight can display global or static variables updated in real-time while the program is running. No additions or changes to your code are required. Updating variables while the program is running is not possible with local variables because they are usually stored in a CPU register. They must be converted to global or static variables so they always remain in scope.

If you have a ULINK_{pro} with ETM trace, you can see a record of all the instructions executed. The Disassembly and Source windows show your code in the order it was written. The ETM trace shows it in the order it was actually executed with timestamps. This is very useful for finding tricky and complicated program flow problems. See page 44 for more uses. ETM also provides Code Coverage, Performance Analysis and Execution Profiling.

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

Call Stack:

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

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







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

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

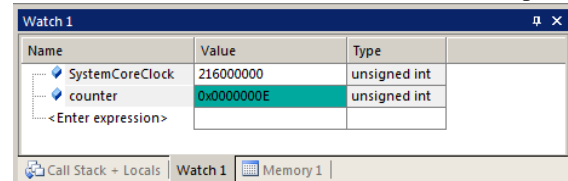
The Watch and Memory windows display updated variable values in real-time. It does this through the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to modify values in the Memory window.

Watch window:

Add a global variable: Recall the Watch and Memory windows can't see local variables unless stopped in their function. Make local variables static or global to make them visible in a Watch or Memory window while the program is running.

1. Stop the program  and exit Debug mode. .
2. In Blinky.c, declare a global variable (I called it counter) near line 20 like this: **unsigned int counter = 0;**
3. Add the statements `counter++;` and `if (counter > 0x10) counter = 0;` as shown here near line 60:
4. Select File/Save All or click .
5. Click on Rebuild. .
6. Enter Debug mode.  Click on RUN . You can set Watch and Memory windows while the program runs.
7. In Blinky.c, right click on the variable **counter** and select Add counter to ... and select Watch 1. Watch 1 will open if needed and counter will be displayed as shown here:
8. **counter** will increment in real-time.
9. Note some values of counter will be missed because the Watch and Memory windows are updated periodically. Press the Blue User button to demonstrate this by slowing the program down.

```
59 Thread_LED();
60 counter++;
61 if (counter > 0x10) counter = 0;
62 if (Buttons_GetState()){
```



Name	Value	Type
SystemCoreClock	216000000	unsigned int
counter	0x0000000E	unsigned int
<Enter expression>		

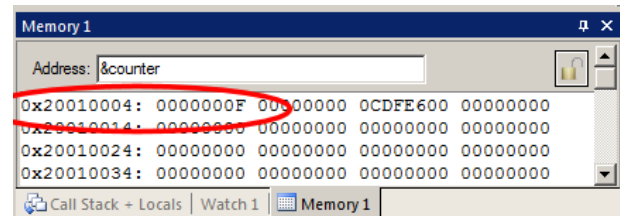
TIP: Note SystemCoreClock is visible and this displays the CPU clock frequency as shown above.

TIP: If an update occurs only when the program is stopped, make sure View/Periodic Window Update is selected.

TIP: You can also block **counter**, click, hold and drag it into a Watch or Memory window. You can also enter a variable manually by double-clicking under Name or pressing F2 and using copy and paste or typing the variable. Use the View/Symbols window to enter a variable fully qualified if needed.

Memory window:

1. Right-click on **counter** and similarly enter it into the Memory 1 window.
2. Note the value of **counter** is displayed in the address column in Memory 1 as if it is a pointer. This is useful to see what memory address a pointer is pointing to; but this not what we want to see at this time.
3. Add an ampersand "&" in front of the variable name and press Enter. The physical address is shown (0x2001_0004).
4. Right click in the Memory window and select Unsigned/Int.
5. The value of **counter** is now displayed as a 32 bit value.
6. Both the Watch and Memory windows are updated in real-time without stealing CPU cycles.
7. You can modify counter in the Memory window while the program is running with a right-click with the mouse cursor over the data field and select Modify Memory.



Address	0x20010004	0x20010014	0x20010024	0x20010034
Value	0000000F 00000000 0CDFE600 00000000	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000

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

How It Works:

µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and does not impact the program execution timings. The Cortex-M/M4/M7 series are a Harvard architecture. This means it has separate instruction and data buses. While the CPU is fetching instructions at full speed, there is plenty of time for the CoreSight debug module to read or write data values without stealing any CPU cycles.

This can be slightly intrusive in the unlikely event the CPU and µVision reads or writes to the same memory location at exactly the same time. Then, the CPU will be stalled for one clock cycle. In practice, this cycle stealing never happens.

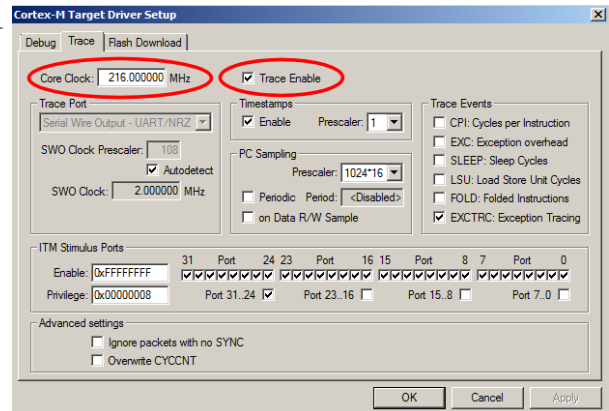
17) View Variables Graphically with the Logic Analyzer (LA):

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

1. Stop the processor and exit Debug mode.

Configure Serial Wire Viewer (SWV):

2. Select Target Options or ALT-F7 and select the Debug tab. Select Settings: on the right side of this window. Confirm SW is selected. SW selection is mandatory for SWV. ST-Link uses only SW. Select the Trace tab.
3. In the Trace tab, set Core Clock: to 216 MHz. Select Trace Enable. Unselect Periodic and select EXCTRC. Everything else is set as shown here:
4. Click OK once to return to the Debug tab.
5. Click OK return to the main menu. Enter Debug mode.

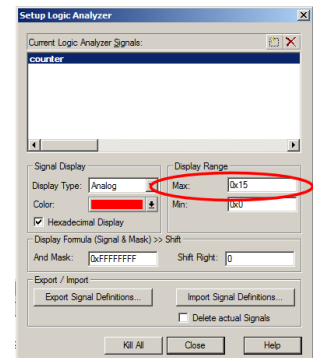


Configure the Logic Analyzer:

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

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

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



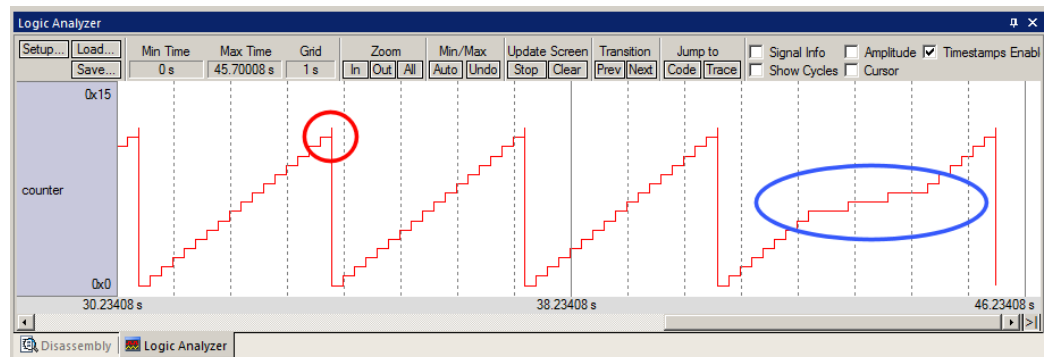
Run the Program:

1. Click on Run. Click on Zoom Out until Grid is about 1 second.
2. The variable `counter` will increment to 0x10 (decimal 16) and then is set to 0.

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

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


1. Press the blue User button and see counter increment slower as shown in the blue circle below:
2. Select Signal Info, Show Cycles, Amplitude and Cursor to see the measuring capabilities of the LA. You can stop the LA from acquiring data by clicking on the Stop icon in the Update Screen box. Your program can keep running.



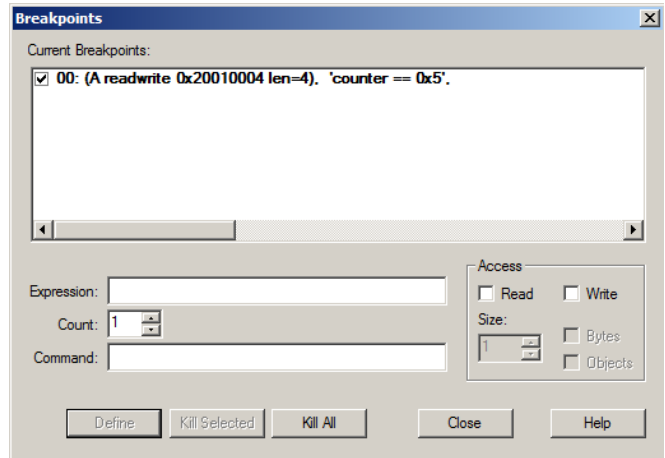
3. Note counter briefly reaches 0x11 since the test is after the increment.
4. When you are ready to continue, start the Update Screen.
5. Stop the CPU.



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

Recall STM32 processors have 6 hardware breakpoints. These breakpoints can be set on-the-fly without stopping the CPU. The STM32 also have four Watchpoints. Watchpoints can be thought of as conditional breakpoints. The Logic Analyzer uses the same comparators as Watchpoints in its operations and they must be shared. This means in you must have two variables free in the Logic Analyzer to use Watchpoints. Watchpoints are also referred to as Access Breakpoints in Keil documentation.

1. Use the same Blinky configuration as on the previous page. Stop the program if necessary.  Stay in Debug mode.
2. We will use the global variable `counter` you created in Blinky.c to explore Watchpoints.
3. The SWV Trace does not need to be configured for Watchpoints. However, we will use it in this exercise.
4. The variable `counter` should be still entered in the Logic Analyzer from the last exercise on the previous page.
5. Select Debug in the main μ Vision window and select Breakpoints or press Ctrl-B.
6. Select both the Read and Write Access. In the Expression box enter: "`counter == 0x5`" without the quotes.
7. Click on Define and it will be accepted as shown here: Click on Close.


8. Enter the `counter` to the Watch 1 window if it is not already listed. It should still be there from before.
9. Open Debug/Debug Settings and select the Trace tab. Select "on Data R/W sample" and unselect EXTRC.
10. Click on OK twice to return to the main menu.



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

16. Note the data writes in the Trace Records window shown below. 0x5 is in the last Data column. Also the address the data written (Address) to and the PC of the write instruction (PC) are displayed. This is with the ST-Link. A ULINK2 will show the same window. A ULINKpro or a J-Link (black case) will show a slightly different display.

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

17. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window. Not all are currently implemented in μ Vision.
18. To repeat this exercise, click on RUN.
19. When finished, stop the program , click Debug/Breakpoints (or Ctrl-B) and Kill the Watchpoint.

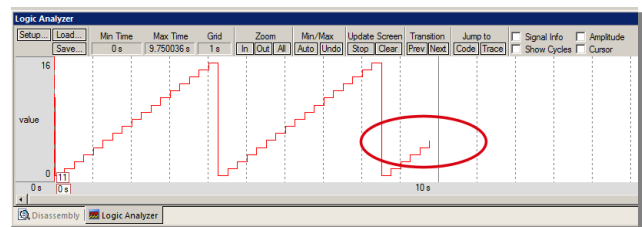
20. Leave Debug mode. 

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

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

TIP: The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it.



TIP: Raw addresses can also be entered into the Logic Analyzer. An example is: *((unsigned long *)0x20000000)



Shown above right is the Logic Analyzer window displaying the variable `counter` trigger point of 0x5.

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






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 μ Vision for display in its Debug (printf) Viewer window. This exercise uses the Blinky_BM project. _BM means Bare Metal – no RTOS is used for simplicity.

1. Stop the program if it is running  and exit Debug mode. .
2. Use the same Blinky_BM project we have been using. SWV must be configured. Delete the Watchpoint.
3. Add this code to Blinky.c. A good place is near line 20, just after the #include "clock.c".


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

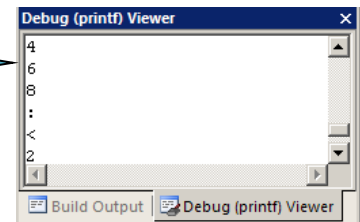
4. In the main function in Blinky.c after the if statement near line 65, enter these lines:

```
ITM_Port8(0) = counter + 0x30; /* displays value in ASCII */
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```

5. Select File/Save All or click . Rebuild the source files .
6. Open Select Target Options  or ALT-F7 and select the Debug tab, and then the Trace tab.
7. The Serial Wire Viewer should be still configured. Use 216 MHz for the Core Clock.
8. Confirm ITM Port 0 is selected. ITM Stimulus Port “0” is the port the Debug (printf) Viewer uses to pass data.
9. Click OK twice to return to the main μ Vision menu. Enter Debug mode .
10. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN .

11. In the Debug (printf) Viewer you will see the ASCII of counter display:

12. As counter is incremented, its ASCII character is displayed. 





TIP: You can easily save ITM information to a file. For information see www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm

Trace Records

1. Open the Trace Records window if not already open. Double click on it to clear it.
2. You will see a window such as the one below with ITM and Exception frames.
3. Right click inside the Trace Records window and unselect Exceptions to filter these out.

How does this work ?

You can see the **ITM writes and Data writes** (counter is also being displayed in the LA as shown below).

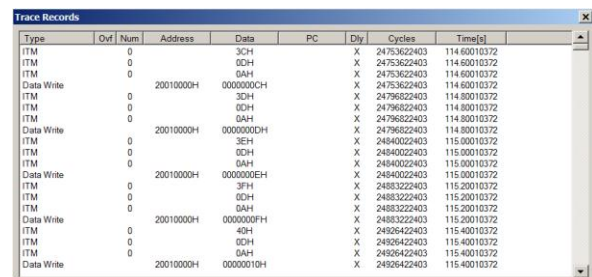
1. ITM 0 frames (Num column) are our ASCII characters from counter with carriage return (0D) and line feed (0A) as displayed the Data column.
2. All these are timestamped in both CPU cycles and time in seconds.
3. When you are done, stop the processor  and exit Debug mode .

ITM Notes

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

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




TIP: ITM_SendChar is a useful function you can use to send ITM characters. It is found in the header core.CM7.h. See the next page.



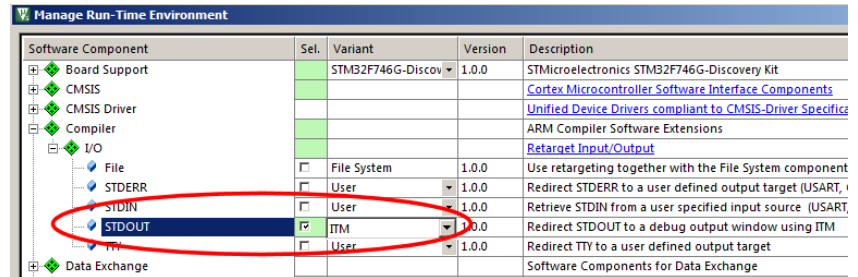
Type	Of	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM	0		3CH			X	24753622403	114.60010372
ITM	0		0DH			X	24753622403	114.60010372
ITM	0		0AH			X	24753622403	114.60010372
Data Write		20010000H	0000000CH			X	24753622403	114.60010372
ITM	0		3DH			X	24796822403	114.80010372
ITM	0		0DH			X	24796822403	114.80010372
ITM	0		0AH			X	24796822403	114.80010372
Data Write		20010000H	0000000DH			X	24796822403	114.80010372
ITM	0		3EH			X	24840022403	115.00010372
ITM	0		0DH			X	24840022403	115.00010372
ITM	0		0AH			X	24840022403	115.00010372
Data Write		20010000H	0000000EH			X	24840022403	115.00010372
ITM	0		3FH			X	24883222403	115.20010372
ITM	0		0DH			X	24883222403	115.20010372
ITM	0		0AH			X	24883222403	115.20010372
Data Write		20010000H	0000000FH			X	24883222403	115.20010372
ITM	0		4CH			X	24926422403	115.40010372
ITM	0		0DH			X	24926422403	115.40010372
ITM	0		0AH			X	24926422403	115.40010372
Data Write		20010000H	00000010H			X	24926422403	115.40010372






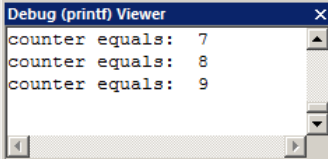


20) printf with ITM (Instrumentation Trace Macrocell): ITM uses Serial Wire Viewer:

It is easy to incorporate printf using ITM and the μ Vision utility Manage Runtime Environment.

1. Stop the program if it is running  and exit Debug mode. 
2. Comment out all six C source lines you entered in Blinky on the previous page. They are not needed here. That exercise was useful to show you how ITM works inside. Here is an easier method you can use.
3. Open the Manage Run-Time Environment utility.  This window opens:

4. Expand Compiler...I/O as shown.
5. Select STDOUT as shown here:
6. In the Variant column, select ITM:
7. All the blocks should be green. If not, click on the Resolve button.
8. Click OK to close this window.
9. The file `retarget_io.c` will be added to your project in the Project window in the Compiler group.




10. In `Blinky.c`, near line 65 just after the `if (counter>....` line, add this line: `printf("counter equals: %d\n", counter);`
11. Select File/Save All or click .
12. Rebuild the source files .
13. Enter Debug mode . Click on RUN .
14. The values of counter is displayed as seen here:  
15. When you are done, stop the processor  and exit Debug mode .

TIP: You can easily save ITM information to a file. See www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm

Obtaining a character typed into the Debug printf Viewer window:

It is possible for your program to input characters from a keyboard with the function `ITM_ReceiveChar` found in `core.CM7.h`. This is documented here: www.keil.com/pack/doc/CMSIS/Core/html/group_itm_debug_gr.html. A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer utility.

Read-Only Source Files:

Some files in the Project window will have a yellow key on them:  This means they are read-only. This is to help unintentional changes to these source files. This can cause difficult to solve problems. Most of these files will not need any modification.

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

1. Double click on the file to open it in the Sources window.
2. Right click on its source tab and select Open Containing folder.
3. Explorer will open with the file selected.
4. Right click on the file and select Properties.
5. Unselect Read-only and click OK. You are now able to change the file in the μ Vision editor.
6. It is a good idea to make the file read-only when you are finished modifications.

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

Keil provides RTX, a full feature RTOS. RTX is included as part of Keil MDK and separately. This example explores the RTX RTOS project. MDK will work with any RTOS. RTX comes with a BSD license and all source code is provided with all versions of MDK.





NOTE: This RTX_Blinky example has four threads simulating a stepper motor and blinks one LED.

RTX information and documentation are available here: www.keil.com/rtx The Getting Started Guide MDK 5 is useful.

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

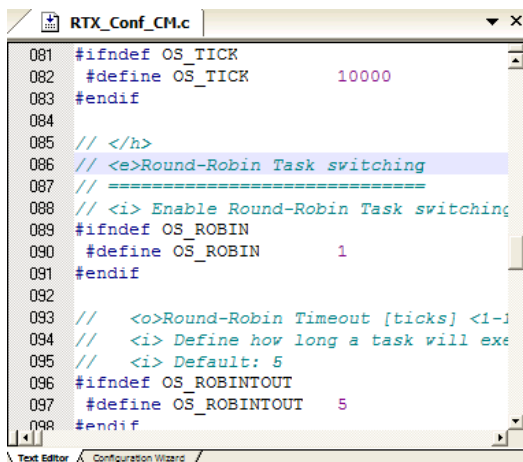
You must have copied the RTX_Blinky example to C:\MDK\Boards\ST\STM32F746G_Discovery as described on page 5.

Run the RTX_Blinky example:

1. With μ Vision in Edit mode (not in debug mode): Select Project/Open Project.
2. Open the file C:\MDK\Boards\ST\STM32756G_EVAL\RTX_Blinky\Blinky.uvprojx.
3. This project is pre-configured for the ST-Link V2 debug adapter.
4. Compile the source files by clicking on the Rebuild icon . They will compile with no errors or warnings.
5. Enter the Debug mode by clicking on the debug icon . The Flash is programmed and progress is displayed.
6. Click on the RUN icon. 
7. The green LED LD1 will be toggled by Thread 1 (phaseA).
8. Open Watch 1 and the four phases phasea through phased will display changing as well as the CPU clock speed.
9. Click on STOP . We will explore the operation of RTX with the Kernel Awareness windows.

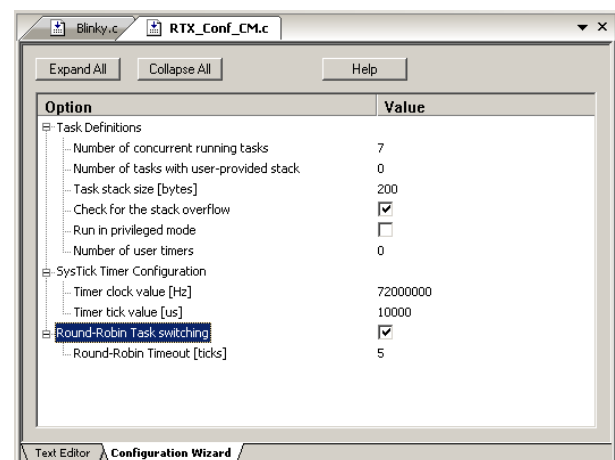
The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. To open this file, double click on it in the Project window under the CMSIS header or open it with File/Open.
2. Click on the Configuration Wizard tab at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. Changing an attribute in one tab changes it in the other automatically. You should save any changes and rebuild.
6. You can create Configuration Wizards for any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a `</h>` or `<i>`. See www.keil.com/support/man/docs/uv4/uv4_ut_configwizard.htm for instructions.



```
081 #ifndef OS_TICK
082 #define OS_TICK 10000
083 #endif
084
085 // </h>
086 // <e>Round-Robin Task switching
087 // =====
088 // <i> Enable Round-Robin Task switching
089 #ifndef OS_ROBIN
090 #define OS_ROBIN 1
091 #endif
092
093 // <o>Round-Robin Timeout [ticks] <1-1
094 // <i> Define how long a task will exe
095 // <i> Default: 5
096 #ifndef OS_ROBINTOUT
097 #define OS_ROBINTOUT 5
098 #endif
```

Text Editor: Source Code



Configuration Wizard

22) 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 two Task Aware windows for RTX. Other RTOS companies also provide awareness plug-ins for μ Vision.

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

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

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
6	clock	Normal	Wait_AND		0x0000	0x0100	40%
5	phaseD	Normal	Wait_DLY 79		0x0000	0x0001	44%
4	phaseC	Normal	Wait_AND		0x0000	0x0001	40%
3	phaseB	Normal	Wait_AND		0x0000	0x0001	40%
2	phaseA	Normal	Wait_DLY 79		0x0000	0x0001	44%
1	main	Normal	Wait_DLY				32%

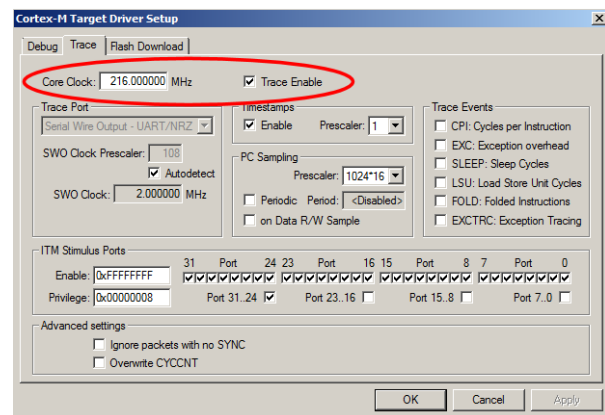
Important TIP: View/Periodic Window Update must be selected !

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

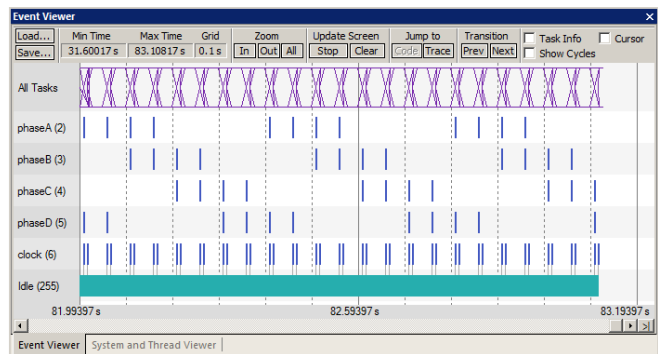
RTX Viewer: Configuring Serial Wire Viewer (SWV):

We must activate Serial Wire Viewer to get the Event Viewer working. The System and Threads Viewer uses DAP R/W.

1. Stop the CPU and exit Debug mode.
2. Click the Target Options icon. Select the Debug tab.
3. Click the Settings box next to ST-Link Debugger.
4. In the Debug window, make sure Port: is set to SW and not JTAG. SWV works only with SW mode.
5. Click on the Trace tab to open the Trace window.
6. Set Core Clock: to 216 MHz. Select Trace Enable.
7. Unselect the Periodic and EXCTRC boxes as shown:
8. ITM Stimulus Port 31 must be checked. This is the method the RTX Viewer gets the kernel awareness information out to be displayed in the Event Viewer. It is slightly intrusive and requires a small amount of code.
9. Click on OK twice to return to μ Vision main menu.
The Serial Wire Viewer is now configured !
10. Click on File/Save All or select the icon:



11. Enter Debug mode and click on RUN.
12. Click on the Event Viewer tab if not in focus.
13. This window displays task events in a graphical format as shown in the RTX Kernel window here: You probably have to change the Range to about 0.1 sec by clicking on the Zoom ALL and + and - icons.
14. You can use the cursor settings to determine various timings of the Threads. You can select Stop Update Screen to stop the LA from collecting data without stopping the CPU to make measuring easier. The program execution is not stopped.



TIP: If Event Viewer doesn't work, open the Trace Records window and confirm there are good ITM 31 frames present. Is Core Clock correct ? This project is running at 216 MHz. Exit and re-enter Debug mode. Cycle the power to the board.

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

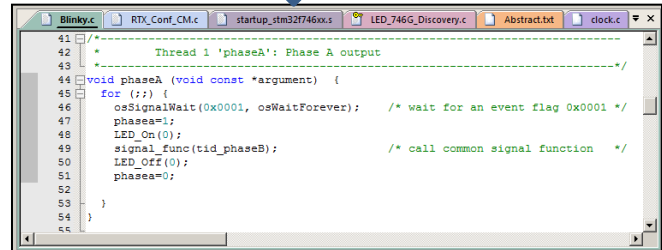
TIP: You can use a ULINK2, ULINK-ME, ULINK pro , ST-Link V2 or J-Link for these RTX Kernel Awareness windows.

23) Logic Analyzer Window (LA): 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 four tasks will be used to create the waveforms. We will graph these four waveforms.

1. Leave the program running from the previous page.
2. Four global variables `unsigned int phasea` through `unsigned int phased` are in Blinky.c as shown here:
3. Each of four threads `phasea` through `phased` represents one phase of a stepping motor driver. This is the `phasea` code and is the one that toggles the LED LD1.
4. Select View/Watch Windows/Watch 1.
5. Note the 4 variables changing in Watch 1 as each thread is executed.
6. We will enter these for global variables into the Logic Analyzer for examination.

```
24 unsigned int phasea=0;
25 unsigned int phaseb=0;
26 unsigned int phasec=0;
27 unsigned int phased=0;
```



Enter the Variables into the Logic Analyzer (LA):

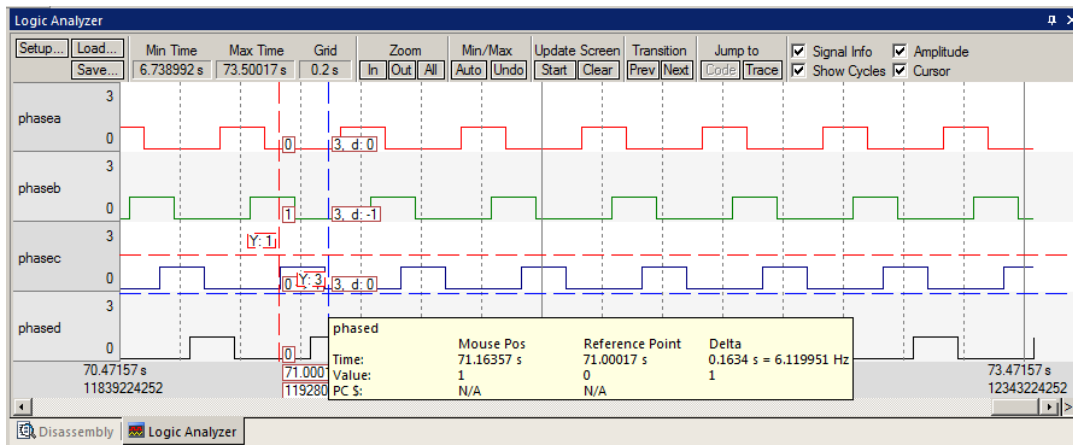
7. Click on the Blinky.c tab. Near line 23, right click on `phasea`, select Add 'phasea' to... and finally select Logic Analyzer. `Phasea` will be added to the LA.
8. Repeat for `phaseb`, `phasec` and `phased`. These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling. Do not use `phasea` instead of `phasea`. They are different variables.

Help: If you can't get these variables entered into the LA, make sure the Trace Configuration is set correctly. The Serial Wire Viewer *must* first be configured correctly in order to enter variables in the LA. The Core Clock value is especially important.

Help: If there is no display, stop the program and exit and re-enter Debug mode to refresh all CoreSight registers. You can try unselecting ITM Stimulus 31 in the Trace Configuration window to stop Event Viewer data from overloading the SWO pin.

TIP: LA can't see locals: make them static or global. For peripheral registers, read or write to them and enter them in the LA.

9. Click on the Setup icon and click on each of the four variables and in turn set Max. in the Display Range: to 0x3.
10. Click on Close to go back to the LA window.
11. Using the All, OUT and In buttons set the range to 2 seconds or so. Move the scrolling bar to the far right if needed.



12. Click on Stop in Update Screen to stop (and start) the data collection.
13. Select Signal Info and Show Cycles. Click to mark a place and move the cursor to get timings. Hover the cursor on one of the waveforms and get timing and other information as shown in the inserted box labeled `phased` above:
14. Stop the CPU and exit Debug mode.

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 are not available for measuring in the outside world. This has proved to be very useful in debugging tricky problems.

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

ARM CMSIS-DSP libraries are offered for ARM Cortex-M series processors. DSP libraries are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS. See www.keil.com/cmsis/dsp for DSP documentation. CMSIS is an acronym for Cortex Microcontroller Software Interface Standard. CMSIS is an ARM standard.

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

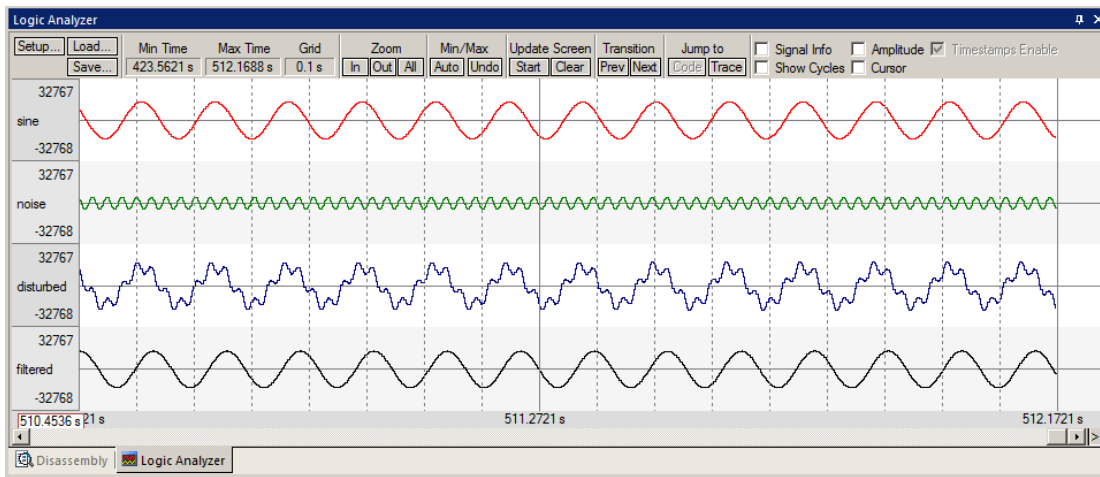
This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. RTX source code is provided.

1. Open the project file sine: C:\MDK\Boards\ST\STM32F746G_Discovery\DSP\sine.uvprojx
2. Build the files.  There will be no errors or warnings.
3. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
4. Click on the RUN icon.  Open the Logic Analyzer window. 
5. Four waveforms will be displayed in the Logic Analyzer using the Serial Wire Viewer as shown below. Adjust Zoom Out for an appropriate display. Displayed are 4 global variables: sine, noise, disturbed and filtered.

TIP: If one variable shows no waveform, disable the ITM Stimulus Port 31 in the Trace Config window. (is by default)

TIP: Serial Wire Viewer can be limited since all data comes out one pin. A Keil ULINK_{pro} handles SWV information much better since it uses Manchester mode at a higher frequency than the ST-Link. Using the 4 bit trace port is even better.

6. The project provided has Serial Wire Viewer configured and the Logic Analyzer loaded with the four variables. Watch 1 window is configured to display the four global variables as shown below:



7. Open the Trace Records window and the Data Writes to the four variables are listed as shown here:
8. Leave the program running.
9. Close the Trace Records window.

TIP: The ULINK_{pro} trace display is different and the program must be stopped to update it.

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

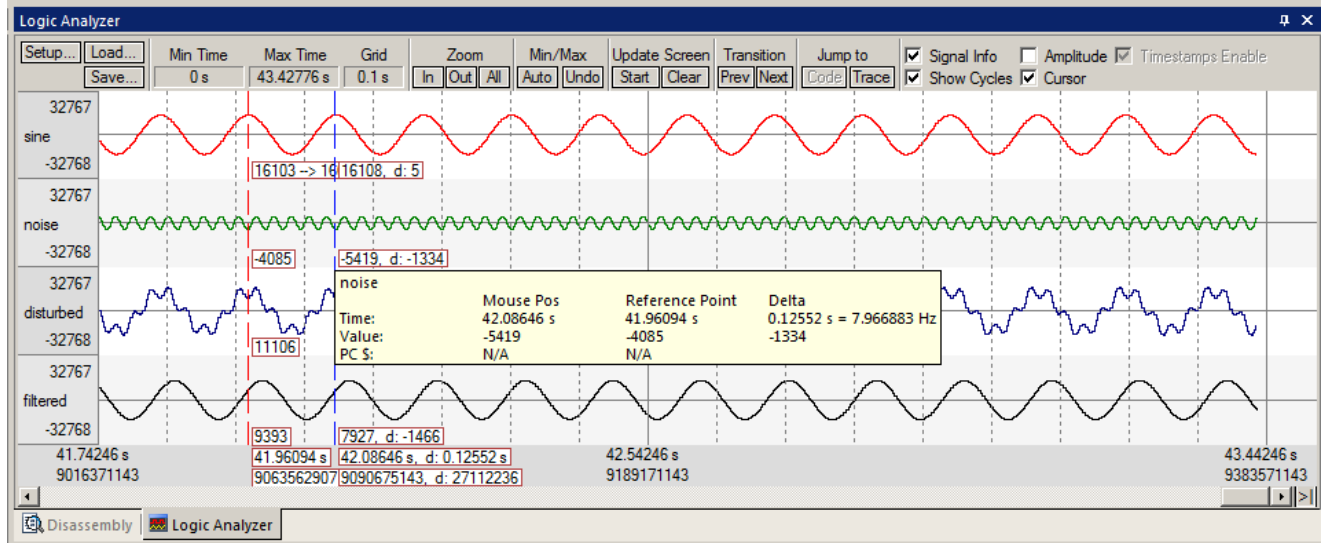
If these values are changing, the program is probably working correctly.

Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			2001001AH	F5AFH			133242735769	616.86451745
Data Write			2001001CH	1912H			133242744210	616.86455653
Data Write			2001001EH	3E43H			133242756147	616.86461179
Data Write			2001001EH	3D6CH			133243004800	616.86576296
Data Write			2001001CH	1912H			133243016281	616.86581612
Data Write			2001001AH	F06EH			133243026290	616.86586245
Data Write			20010018H	2004H			133243035659	616.86590583
Data Write			20010018H	1C83H			133243278664	616.86703085
Data Write			2001001AH	ECB3H			133243288709	616.86707736
Data Write			2001001CH	0936H			133243297110	616.86711625
Data Write			2001001EH	3DEDH			133243309047	616.86717151
Data Write			2001001EH	3CCFH			133243558444	616.86832613
Data Write			2001001CH	0936H			133243569925	616.86837928
Data Write			2001001AH	EADDH			133243579934	616.86842562
Data Write			20010018H	18E6H			133243589303	616.86846900
Data Write			20010018H	152FH			133243831552	616.86959052
Data Write			2001001AH	EB17H			133243841597	616.86963702
Data Write			2001001CH	0046H			133243849998	616.86967592
Data Write			2001001EH	3C84H			133243861935	616.86973118
Data Write			2001001EH	3B43H			133244110672	616.87088274

Signal Timings in Logic Analyzer (LA):

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



RTX Tasks and System:

5. Click on Start in the Update Screen box to resume the collection of data.
6. Open Debug/OS Support and select System and Thread Viewer. A window similar to below opens up. You probably have to click on its header and drag it into the middle of the screen.
7. Set a breakpoint in each of four of the threads in DirtyFilter.c by clicking in the left margin on a grey area.
8. Click on Run and the program will stop at each thread in turn and the Thread Viewer window will be updated accordingly. In this case, I set a breakpoint in the thread disturb_gen.
9. Clearly, below you can see that disturb_gen was running when the breakpoint was activated.
10. Remove the breakpoints. Click on them or enter Ctrl-B and select Kill All. Then click on Close to close this window.

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

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

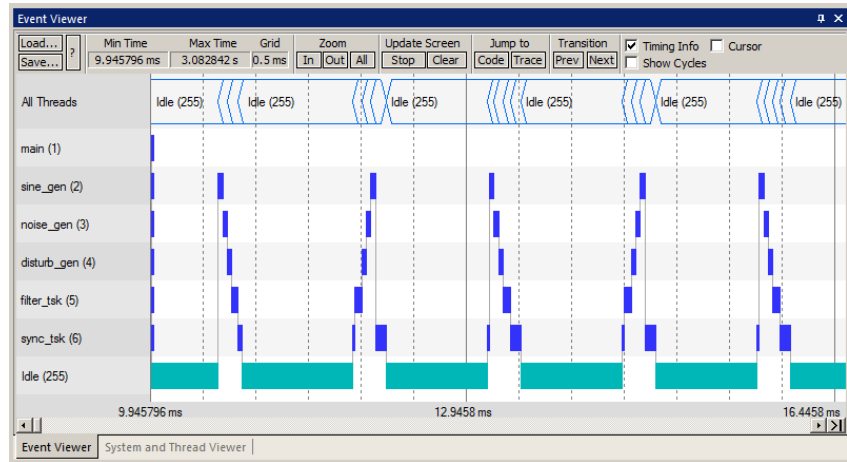
The Event Viewer does use SWV and this is demonstrated on the next page.

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

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

Event Viewer:

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



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

TIP: If Event Viewer is blank or erratic, or the LA variables are not displaying or blank: this is likely because the Serial Wire Output pin is overloaded and dropping trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some bandwidth. Using a ULINK pro in either Manchester mode or using the 4 bit trace port will help.

If you see <No Address> Threads, this is because of SWO overload. Using a ULINK pro is a good solution.

ULINK pro is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows.

ULINK pro uses the Manchester format at a higher speed than the UART mode that ST-Link, ULINK2 and J-Link uses.

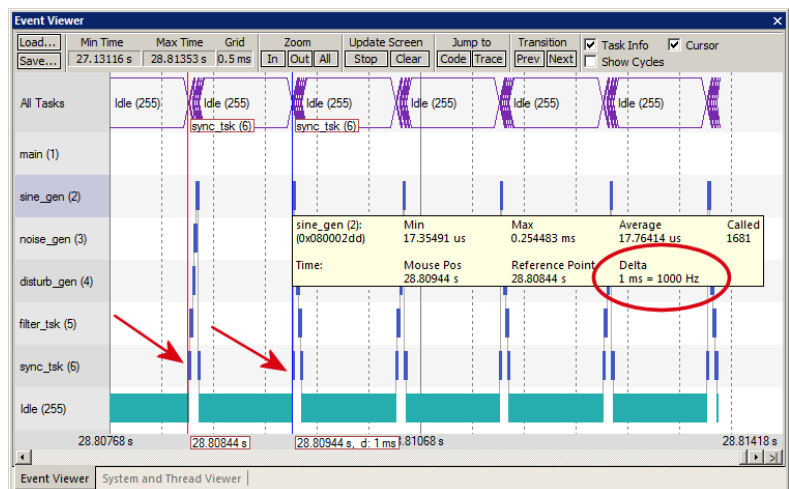
ULINK pro can also use the 4 bit Trace Port for faster operation for SWV. The Trace Port is mandatory for ETM trace.

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

15. Hover your mouse on one of the threads blue block. It will turn yellow and display information about this event as shown below:

TIP: ITM Port 31 enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port if you are not using the Event Viewer because data is still sent. Port 0 is used for ITM printf operations.

Timing of 'sine_gen' (Thread #2 @0x0800031d)			
Current Slice:	Begin	End	Duration
No. 4294	93.00956 s	93.00961 s	46.06019 us
All Slices:	Min	Max	Average
Count: 14522	45.11574 us	0.963153 ms	46.67593 us
Cursors:	Mouse	Reference	Difference
	93.0096 s	93.01105 s	-0.001456 s = 686.891815 Hz



25) Keil Middleware: Network (TCP/IP), Flash File, USB, CAN and Graphics:

MDK Professional provides commercial grade middleware with extensive capabilities designed for demanding applications.

See www.keil.com/mdk5/middleware/ for more information.

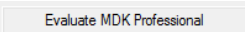
Working examples are provided for various ST boards including STM32F746G-Discovery and STM32F756G-Eval. You can

access and copy these examples using Pack Installer .  Here is the listing for STM32F746G-Discovery:

This window is displayed by selecting STM32F746G-Discovery in the Boards tab:

Example	Action	Description
BSD Client (STM32F746G-Discovery)	Copy	Example using BSD sockets to send commands to remote server
BSD Server (STM32F746G-Discovery)	Copy	Example using BSD sockets to accept commands from remote clients
CMSIS-RTOS Blinky (STM32F746G-Discovery)	Copy	CMSIS-RTOS based Blinky example
CMSIS-RTOS Blinky with STM32CubeMX (STM32F746G-Discovery)	Copy	CMSIS-RTOS based Blinky example configured with STM32CubeMX
FTP Server (STM32F746G-Discovery)	Copy	File Server using FTP protocol with SD/MMC Memory Card as storage media
File System Demo (STM32F746G-Discovery)	Copy	File manipulation example: create, read, copy, delete files on any enabled drive (SD/MMC Card, NOR/NAND Flash, RAM) and format each drive
HTTP Server (STM32F746G-Discovery)	Copy	Compact Web Server with CGI interface
HTTP Upload (STM32F746G-Discovery)	Copy	Web Server with CGI interface and SD/MMC Memory Card as storage media
SMTP Client (STM32F746G-Discovery)	Copy	Example showing how to compose and send emails
SNMP Agent (STM32F746G-Discovery)	Copy	Example showing how to use a Simple Network Management Protocol (SNMP)
Telnet Server (STM32F746G-Discovery)	Copy	Command-line Host service example using Telnet protocol
USB Device HID (STM32F746G-Discovery)	Copy	USB Human Interface Device providing access from PC to board LEDs and push buttons
USB Device Mass Storage (STM32F746G-Discovery)	Copy	USB Mass Storage Device using RAM as storage media
USB Device Virtual COM (STM32F746G-Discovery)	Copy	Bridge between PC USB Virtual COM Port and UART port
USB Host Keyboard (STM32F746G-Discovery)	Copy	Measure example using USB HID Keyboard as input device
USB Host Mass Storage (STM32F746G-Discovery)	Copy	USB Host file manipulation example: create, read, copy, delete files from USB Mass Storage Device and format the storage device
emWin Example (STM32F746G-Discovery)	Copy	emWin Graphics simple example
emWin GUI Demo (STM32F746G-Discovery)	Copy	emWin Graphics Demo example
emWin VNC Server with STM32CubeMX (STM32F746G-Discovery)	Copy	emWin VNC Server example configured with STM32CubeMX

License:

An MDK Pro license is needed for Keil Middleware. A 7 day one-time license is available in μ Vision under File/License Management. If you qualify, this button is displayed: 

To obtain a temporary MDK Professional license for evaluation purposes, contact Keil sales as listed on the last page.


Instructions: Each example contains the file abstract.txt which provides instructions. These projects compile and run "out-of-the-box". If you have any questions regarding Keil Middleware operation during your evaluation phase, please contact Keil Technical Support as listed on the last page of this document.

Configuring the examples:

The Middleware examples make extensive use of the Configuration Wizard. This permits easy modifications to various settings with mouse clicks instead of digging through source code.

Selecting the Middleware:

Keil Middleware is selected using the Manage Run-Time Environment utility. This selects the various components you desire and place them into your project. Open the Manage Run-Time Environment utility

 and this window is displayed:

Software Component	Sel.	Variant	Version	Description
Board Support	<input checked="" type="checkbox"/>	STM32F746G-Disco	1.0.0	STMicroelectronics STM32F746G-Discovery Kit
CMSIS	<input checked="" type="checkbox"/>			Cortex Microcontroller Software Interface Components
CMSIS Driver	<input checked="" type="checkbox"/>			Unified Device Drivers compliant to CMSIS-Driver Specific
Ethernet (API)	<input checked="" type="checkbox"/>		2.01	Ethernet MAC and PHY Driver API for Cortex-M
Ethernet MAC (API)	<input checked="" type="checkbox"/>		2.01	Ethernet MAC Driver API for Cortex-M
Ethernet PHY (API)	<input checked="" type="checkbox"/>		2.00	Ethernet PHY Driver API for Cortex-M
Flash (API)	<input checked="" type="checkbox"/>		2.00	Flash Driver API for Cortex-M
I2C (API)	<input checked="" type="checkbox"/>		2.02	I2C Driver API for Cortex-M
MCI (API)	<input checked="" type="checkbox"/>		2.02	MCI Driver API for Cortex-M
NAND (API)	<input checked="" type="checkbox"/>		2.01	NAND Flash Driver API for Cortex-M
SAI (API)	<input checked="" type="checkbox"/>		1.00	SAI Driver API for Cortex-M
SP (API)	<input checked="" type="checkbox"/>		2.01	SP Driver API for Cortex-M
USART (API)	<input checked="" type="checkbox"/>		2.01	USART Driver API for Cortex-M
USB Device (API)	<input checked="" type="checkbox"/>		2.01	USB Device Driver API for Cortex-M
USB Host (API)	<input checked="" type="checkbox"/>		2.01	USB Host Driver API for Cortex-M
Compiler	<input checked="" type="checkbox"/>			ARM Compiler Software Extensions
Data Exchange	<input checked="" type="checkbox"/>			Software Components for Data Exchange
Device	<input checked="" type="checkbox"/>			Startup_System Setup
File System	<input checked="" type="checkbox"/>		6.5.0	File Access on various storage devices
Drive	<input checked="" type="checkbox"/>	SFN	6.5.0	File System without Long Filename support for Cortex-M Storage Devices and Media Types
Graphics	<input checked="" type="checkbox"/>		MDK-Pro	User Interface on graphical LCD displays
Graphics Core for Cortex-M	<input checked="" type="checkbox"/>		5.30.0	Graphics Core for Cortex-M
VNC Server	<input checked="" type="checkbox"/>		5.30.0	Enable Remote access via TCP/IP
Demo	<input checked="" type="checkbox"/>			Demonstrator for Graphical User Interface features
Display	<input checked="" type="checkbox"/>			Display Interface Configuration
Input Device	<input checked="" type="checkbox"/>			Devices used to control the Graphical User Interface
Tools	<input checked="" type="checkbox"/>			emWin Tools
Graphics Display	<input checked="" type="checkbox"/>			Display Interface including configuration for emWIN
Network	<input checked="" type="checkbox"/>		lwIP	Network lwIP Bundle
API	<input checked="" type="checkbox"/>		1.4.1	Network high-level wrapper API
Interface	<input checked="" type="checkbox"/>			Connection Mechanism
USB	<input checked="" type="checkbox"/>		MDK-Pro	USB Communication with various device classes
USB Core for Cortex-M	<input checked="" type="checkbox"/>		6.5.0	USB Core for Cortex-M
USB Device	<input checked="" type="checkbox"/>		6.5.0	USB Device
USB Host	<input checked="" type="checkbox"/>		6.5.0	USB Host
Device Classes	<input checked="" type="checkbox"/>			USB Device Classes
Host Classes	<input checked="" type="checkbox"/>			USB Host Classes

Help and Documentation:

This is available both online and embedded in MDK:

www.keil.com/pack/doc/mw/General/html/index.html

and

C:\Keil_v5\ARM\Pack\Keil\MDK-Middleware

In each of the projects is a file abstract.txt that provides basic instructions and a link to more details.

26) Creating your own MDK 5 project from scratch:

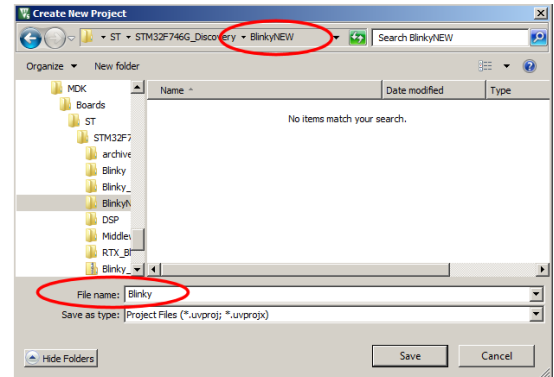
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load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

Install the STM32 Software Pack for your processor:

1. Start μ Vision and leave in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Pack for the STM32F7 processor must be installed. This has already been done on page 4.

Create a new Directory and a New Project:

1. Click on Project/New μ Vision Project...
2. In the window that opens, shown here, go in: C:\MDK\Boards\ST\STM32F746G_Discovery\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj in C:\MDK\Boards\ST\STM32F746G_Discovery\BlinkyNEW\.
7. As soon as you click on Save, the next window opens:



Select the Device you are using:

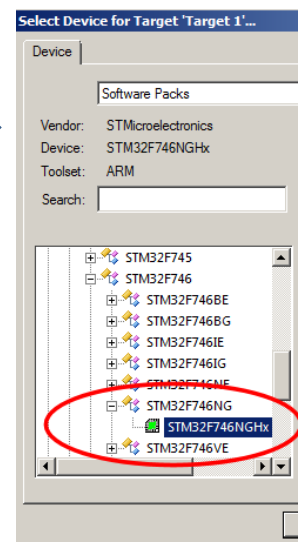
1. Expand STMicroelectronics, then STM32F7 Series, then STM32F746, then STM32F746NG and then finally select STM32F746NGHx:

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

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

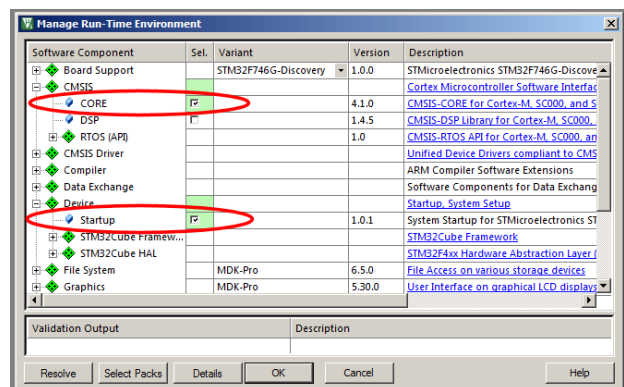
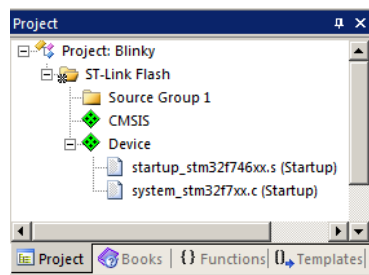
Select the CMSIS components you want:

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

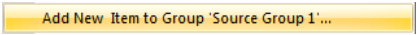



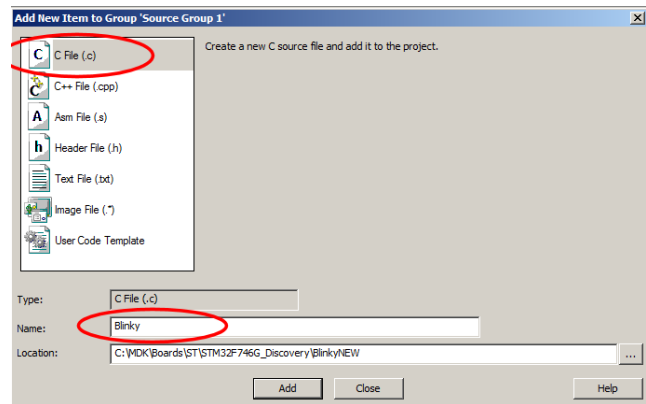
What has happened to this point:

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



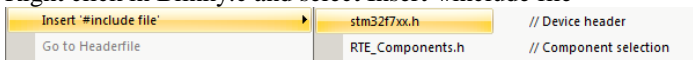
Create a blank C Source File:

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



Add Some Code to Blinky.c:

9. Right click in Blinky.c and select Insert '#include file'



10. Select stm32f7xx.h and then repeat for RTE_Components.h. These are added to Blinky.c.
11. In the blank portion of Blinky.c, add the C code below:



```

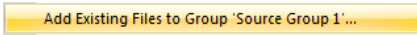
unsigned int counter = 0;

/*-----
  MAIN function
  *-----*/
int main (void) {


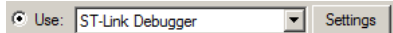




    while(1) {
        counter++;
        if (counter > 0x0F) counter = 0;
    }
}

```

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

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



Configure the Target St Link Flash: *Please complete these instructions carefully to prevent unusual problems...*

1. Select the Target Options icon . Select the **Target** tab.
2. Select Use MicroLIB to optimize for smaller code size. Note the memory locations are entered for your convenience.
3. Select the Debug tab. Select **ST-Link Debugger** in the Use: selection box. 
4. Select the Settings: icon.
5. Select SW as shown here in the Port: box:  JTAG will not work with SWV. If your board is connected to your PC, you **must** now see a valid IDCODE and Device Name in the SW Device box.
6. Click on OK **once** to go back to the Target Configuration window. Otherwise, fix the connection problem.
7. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm as shown: Shown is the correct one for the STM32F7 series processors: 
8. Click on OK twice to return to the main menu.
9. Click on File/Save All or 
10. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !


Programming Algorithm			
Description	Device Size	Device Type	Address Range
STM32F7xx 1MB Flash	1M	On-chip Flash	08000000H - 080FFFFFFH

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

Running Your Program:

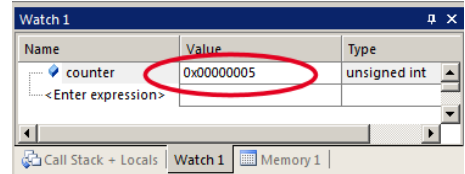
1. Enter Debug mode by clicking on the Debug icon . Your program will be programmed into the ST Flash.
2. Click on the RUN icon . It will run to the beginning of main() and stop.

Note: you stop the program with the STOP icon .

3. No LEDs will blink since there is no source to accomplish this task. You could add such code yourself later.
4. Right click on counter in Blinky.c and select Add counter to ... and select Watch 1.
5. counter should be updating as shown here: 
6. You can also set a breakpoint in Blinky.c and the program should stop at this point if it is running properly.

If you do this, remove the breakpoint.

7. You should now be able to add your own source code to create a meaningful project.



Since we did not configure any clocks, the CPU is running at the default of 16 mHz.

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

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

There are three main methods to create your own projects:

We are using 3) in this exercise:

- 1) **STM32CubeMX.** This configures your processor and exports a μ Vision project in MDK 5 format. See Page 28. STM32CubeMX can be downloaded from www.st.com/stm32cubemx/. For information on creating projects with STM32CubeMX see: www.keil.com/pack/doc/STM32Cube/General/html/
- 2) **Standard Peripheral Libraries** from ST. STM32CubeF7. Contains extensive examples and source code for Keil MDK 5. These libraries are also available from www.st.com/stm32cubemx/
- 3) **μ Vision Software Packs, examples and Keil Middleware.** A Software Pack includes examples and files that you can use. See Page 21 and www.keil.com/pack/doc/STM32Cube/General/html/index.html

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

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

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



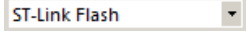



ELF/DWARF: The ARM compiler produces an .axf file which is ELF/DWARF compliant. μ Vision can load similar compiler output such as from GCC with all debug information visible. You can also use GCC as your compiler of choice in μ Vision.

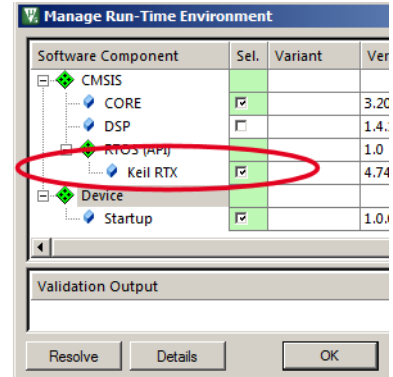
27) Creating your own RTX MDK 5 project from scratch:

The MDK Software Packs makes it easy to configure an RTX project. We will use the RTX that is CMSIS-RTOS compliant.


Configuring RTX is easy in MDK 5. These steps use the same configuration as in the preceding Blinky example.

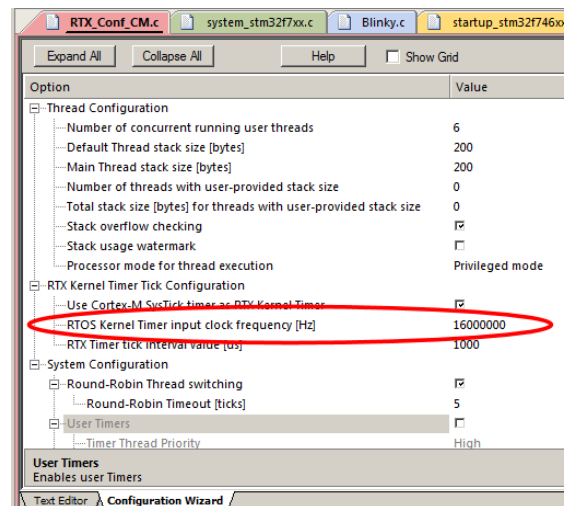
For RTX documentation see: www.keil.com/pack/doc/CMSIS/RTOS/html/example_rtx_tutorial.html

- Using the same example from the preceding pages, Stop the program  and Exit Debug mode. 
- Select ST-Link Flash: 
- Open the Manage Run-Time Environment window: 
- Expand all the elements as shown here: 
- Select Keil RTX as shown and click OK.
- Appropriate RTX files will be added to your project. See the Project window under the CMSIS group.
- In Blinky.c, at the top, add this line: `#include "cmsis_os.h"`. You can also right-click inside Blinky.c and select Insert '#include' and select cmsis_os.h.
- Click on File/Save All or 







Configure RTX:

- In the Project window, expand the CMSIS group.
- Double click on RTX_Conf_CM.c to open it.
- Select the Configuration Wizard tab: Select Expand All.
- The window is displayed here: 
- Set Timer clock value: to 16000000 as shown: (16 MHz)
- Unselect User Timers. Use defaults for the other settings.



Build and Run Your RTX Program:

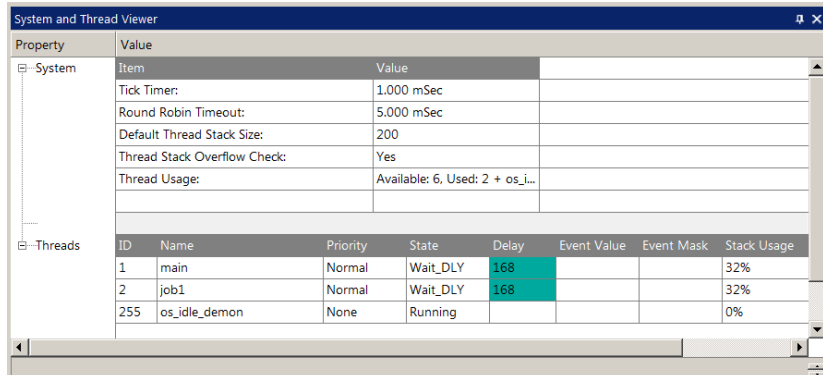
- Click on File/Save All or 
- Build the files. 
- Enter Debug mode:  Click on the RUN icon. 
- Select Debug/OS Support/System and Thread Viewer. The window below opens up.
- You can see two threads: the main thread is the only one running. As you add more threads to create a real RTX program, these will automatically be added to this window.

What you have to do now:

- You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs. See the next page.

2. Getting Started Guide MDK



Obtain this useful book here: www.keil.com/gsg/. It has useful information on implementing RTX.



5:
very

28) Adding a Thread to your RTX_Blinky:

We will create and activate a thread. We will add another global variable counter2 to give it something to do.

1. Stop the program  and Exit Debug mode. 
2. In Blinky.c, add this line near line 6 before the main() function:

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

Create the Thread job1:

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

TIP: osDelay(500) delays the program by 500 clock ticks to slow it down so we can see the values of counter and counter2 increment by 1.

Add osDelay to main():

4. Add this line just after the second if statement near line 19:

```
19 osDelay(500);
```


Define and Create the Thread:

5. Define job1 near line 15 just before main():
6. Create the thread job1 near line 18 just before the while(1) loop:



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

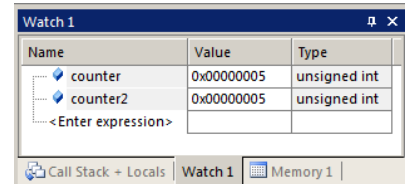
```
23 osThreadCreate(osThread(job1), NULL);
```

7. Click on File/Save All or 

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

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

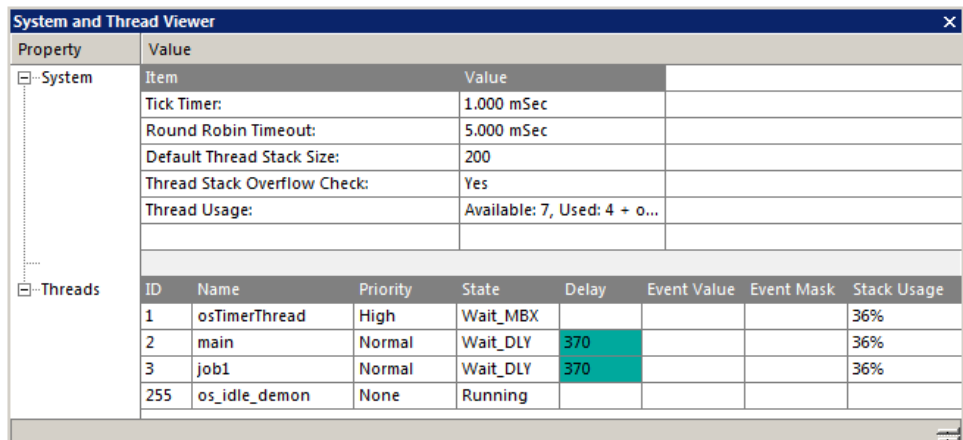
9. Enter Debug mode:  Click on the RUN icon. 
10. Right click on counter2 in Blinky.c and select Add counter2 to ... and select Watch 1.
11. Both counter and counter2 will increment but slower than before: The two osDelay(500) function calls each slow the program down by 500 msec. This makes it easier to watch these two global variables increment. OsDelay() is a function provided by RTX.



Name	Value	Type
counter	0x00000005	unsigned int
counter2	0x00000005	unsigned int
<Enter expression>		

12. Open the System and Thread Viewer by selecting Debug/OS Support.
13. Note that job1 has now been added as a thread as shown below:
14. Note os_idle_demon is always labelled as Running. This is because the program spends most of its time here.
15. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
16. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
17. Remove all breakpoints before continuing.

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









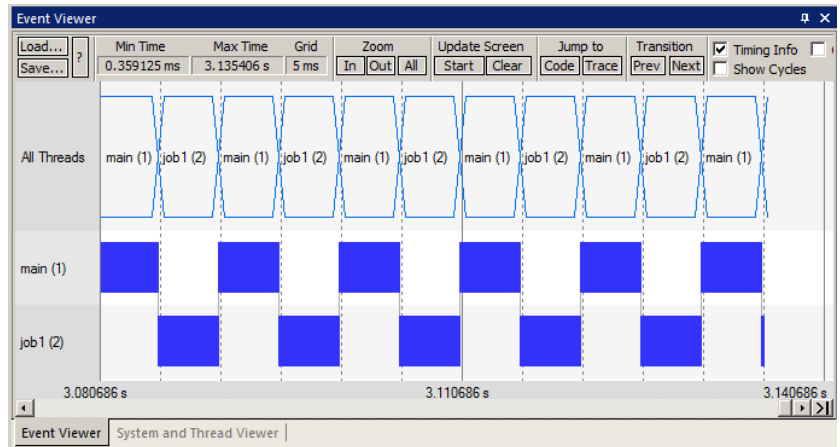
Property	Value
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 4 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				36%
2	main	Normal	Wait_DLY	370			36%
3	job1	Normal	Wait_DLY	370			36%
255	os_idle_demon	None	Running				

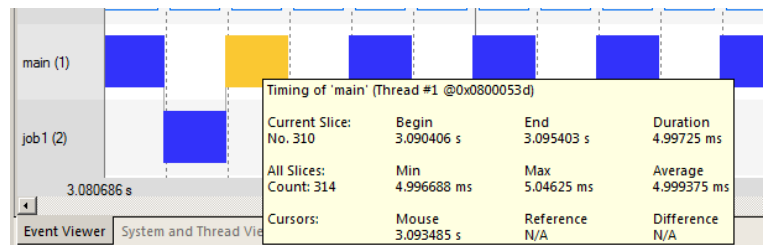
29) Using Event Viewer to examine your RTX_Blinky Timing:

We will demonstrate the utility of the Event Viewer.



- 1) Stop the program  and exit Debug mode. 
- 2) In Blinky.c there are two lines `osDelay();`. Comment both of these out. We will run the program really fast.
- 3) Open Select Target Options  or ALT-F7. Select the Debug tab and the Settings. Select the Trace tab.
- 4) Set Core Clock: to 16 MHz. Unselect EXCTRC: and Periodic. Leave everything else at their default settings.
- 5) Click on OK twice to return to the main μ Vision menu.
- 6) Build the files.  Enter Debug mode:  Click on the RUN icon. 
- 7) Open the Event Viewer by selecting Debug/OS Support and select Event Viewer. This window will open:
- 8) Adjust Zoom In and Out for a comfortable view.

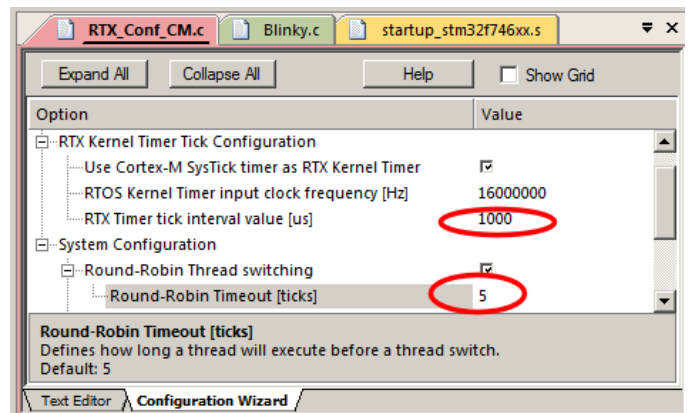


- 9) Hold your cursor over one of the blue blocks and a yellow information window is displayed as shown here:
- 10) Note each thread duration is about 5 msec.



What is Happening Here:


- 1) Open RTX_Conf_CM.c and select the Configuration Wizard tab as shown below:
- 2) The CPU speed is set to 16 MHz with a Timer tick value of 1000 us or 1 msec.
- 3) Note Round Robin switching is selected with a tick timeout of 5 ticks.
- 4) The Thread timing is this $1 \text{ msec} * 5 \text{ ticks} = 5 \text{ msec}$.
- 5) Every 5 msec the Thread is switched to the next one and these sequences are displayed in the Event Viewer. There are other ways to switch a thread.
- 6) It is quite easy to view how RTX is running to make sure it is performing as you designed.
- 7) There are many other RTX features you can use. Refer to the extensive RTX documentation.
- 8) Stop the program  and exit Debug mode. 



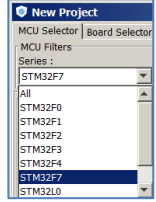

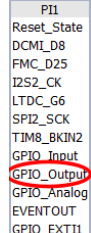

30) Using STM32CubeMX to create a simple Blinky program:

Using STM32CubeMX to create your μ Vision project is a very good idea. It configures the multiplexed pins on your processor, calculates the clock settings and creates a MDK 5 project using CMSIS to use as your starting point.

Download and Install STM32CubeMX:

1. Download and install STM32CubeMX from www.st.com/stm32cubemx
2. Download and install [STM32CubeF7](#) Embedded Software package for the STM32F7 series from the same place.
3. Open STM32Cube MX  and select New Project: [New Project](#)

Create a new Project: select your processor and configure user LED Port PI pin 1 as GPIO:

1. In the New Project window, in the Series pulldown menu, select STM32F7 as shown here: 
2. In the MCUs List window, select STM32F746NGHx as shown below: 
3. Click OK. Wait for the Pinout window to display. It will have a graphic pinout diagram of your processor. The Pinout tab is selected.
4. The green user LED LD1 is attached to Port I pin 1 as found from the Discovery board schematics. We will select and configure this pin as GPIO output.
5. In the Find box, enter Pi1 or PI1. In the List box that opens, select PI1.
6. The PI1 circle (or pad) will turn darker grey and blink.
7. Click once on the circle PI1 and this menu selection opens: Select GPIO_Output as shown here: 
8. The PI1 circle will turn green with a check as shown here indicating a successful configuration: 

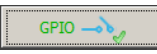

Configure the Processor clocks: You set the CPU speed here. The default CPU clock speed is 16 MHz.

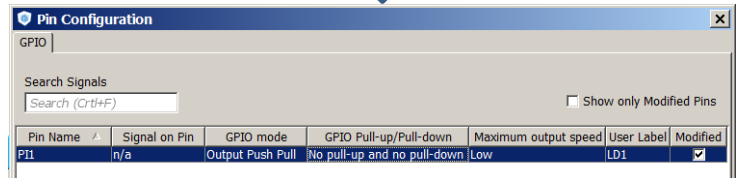
1. Select the Clock Configuration tab: [Clock Configuration](#)
2. Find the HCLK box and insert 216 and press Enter as shown here:
3. A info window will display stating no solution exists. Select Ok to find a solution. (Clock source will be changed from HSI to PLLCLK)
4. Note your final clock value is calculated backwards to obtain the correct clock register settings.



TIP: You can lock a value by right clicking on a box. This value will stay constant as you change other values.

Inspect Pin Configuration Settings: You can modify the settings of the pin items you have set here.

1. Select the Configuration tab: [Configuration](#)
2. Under the System column, select GPIO.  This window opens: 
3. Highlight GPIO: The bottom part opens:
4. Note the various items you can modify.
5. In User Label enter **LD1**.
6. Click Ok to close this window.



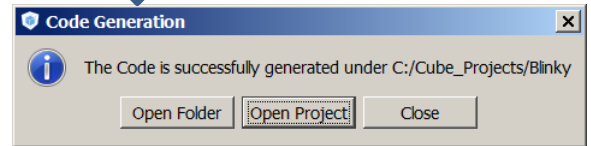
Configure your Project:

1. In the main STM32CubeMX window, select Project/Settings or press Alt-P. The Project Settings window opens.
2. In Project Name box enter Blinky.
3. For Project Location box, browse to C:\Cube_Projects.
4. In the Toolchain/IDE field, select MDK-ARM V5.
5. Click on Ok to save and close. Blinky.ioc will be created in C:\Cube_Projects\Blinky\.



What you have so far: You have setup up GPIO Port I pin 1 to turn LD1 green LED on. The CPU speed is set to 216 MHz. Many other configuration items have been automatically set by STM32Cube for you. You can still modify them.

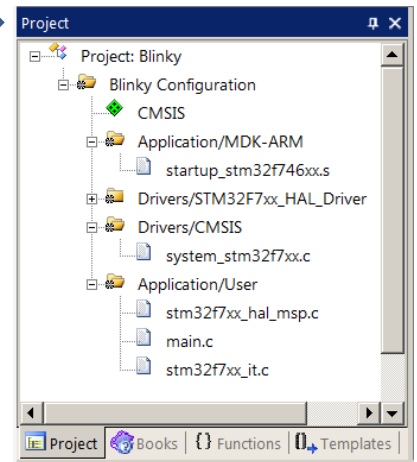
Generate µVision Project Code:

1. Close µVision. Otherwise, you will end up with two instances of µVision running and this can be confusing.
2. In the main STM32CubeMX menu, select Project/Generate Code. This will create the MDK 5 project files.
3. A progress bar will display. When it is finished this window will display:
4. Select Open Project and µVision will be started and load Blinky.



Examine and Build the Project Files:

1. Build the files.  The Build Output window will show no errors or warnings.
2. In the Project window in µVision the file structure is displayed. 
3. Expand the Application/User folder as shown below here:
4. Double click on main.c to open it. At this point, all you need to do is add your own source code. main.c has a main() function containing some calls to initialization files but an empty while(1) loop. There are sections created where you can enter your own source code.



Add Source Code to Blink LED LD1:

1. In main.c, in the while(1) loop near line 88 is a User Code section 3.
2. Add these two lines near line 89:




```
88 HAL_GPIO_TogglePin(GPIOI, GPIO_PIN_1);
89 HAL_Delay(100);
```

3. Select File/Save All. 

TIP: You can see where these functions are located by right clicking on the function name and selecting Go To Definition:

Go To Definition Of 'HAL_GPIO_TogglePin'

Running your Program:

1. Connect your PC to the board with a USB cable to CN14.
2. By default, the ST-Link Debugger is selected.
3. Compile the source files by clicking on the Rebuild icon.  You can also use the Build icon beside it. The Build Output window will show no errors or warnings. If there are, please fix them now.
4. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears. The Flash will be programmed with progress indicated in the Build Output window.
5. Click on the RUN icon. 

**The LED LD1 on the STM32F7 Discovery board will blink.
Congratulations – you have created your first program with STM32CubeMX !**




See various STM32CubeMX tutorials and documentation available on www.st.com/stm32cubemx

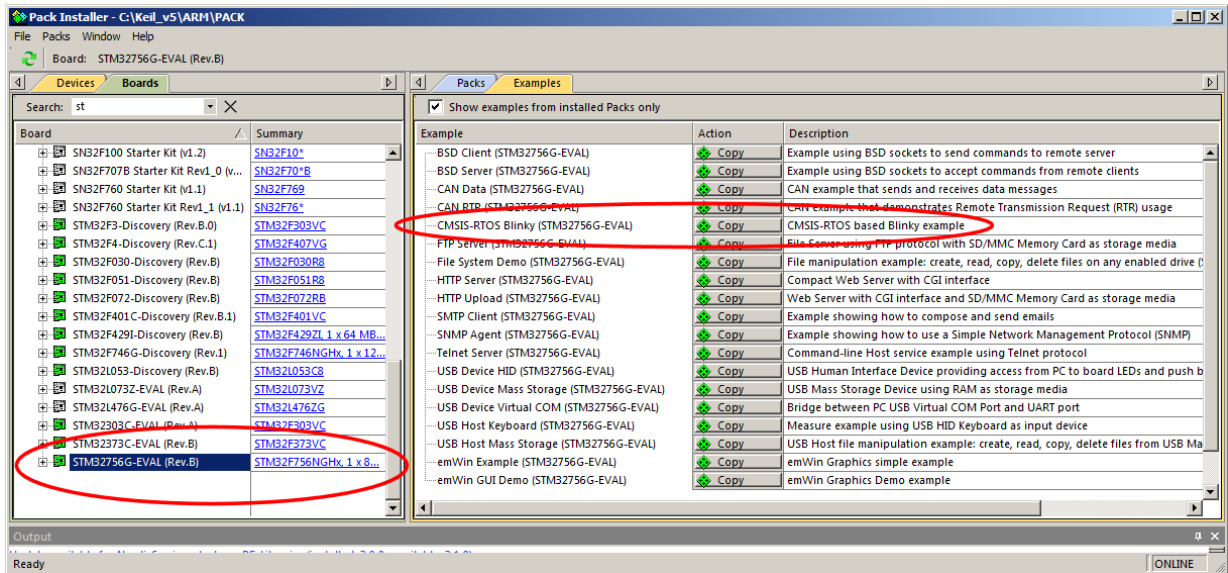
USING Keil RTX RTOS: At this point, you can easily add RTX to this project. See page 25 and the MDK 5 Getting Started Guide. RTX online documentation: www.keil.com/pack/doc/CMSIS/RTX/html/example_r_t_x_tutorial.html

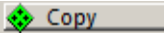
31) ETM Trace Examples:

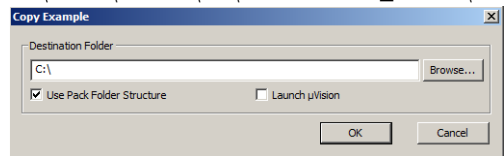
These examples were run on the STM32756G-EVAL evaluation board. A ULINK_{pro} debug adapter is required for ETM operation. You can use a ULINK_{pro} on your own custom board if it has the 20 pin CoreSight ETM connector to provide the 5 signal Trace Port (4 data + clock). Many STM32 processors have ETM. ETM and Serial Wire Viewer are separate CoreSight components. ETM and SWV are available on nearly every STM32 processor except for Cortex-M0. ETM provides serious debugging capabilities as shown on the next few pages. It is worth the small added cost given all the advantages provided.

Install STM32756G-EVAL Examples:

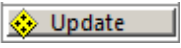
1. Connect your computer to the Internet. This is normally needed to download the Software Pack examples.
2. Start μ Vision by clicking on its desktop icon. 
3. Open the Pack Installer (PI) by clicking on its icon:  The Pack Installer window opens as shown below:
4. **Note:** "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, please connect to the Internet.
5. Packs/Check for Updates or  to refresh once you have connected to the Internet. It is not done automatically.
6. Under the Boards tab, in the Search box, type ST as shown below:
7. Select STM32756G-EVAL: This will filter the list under the Packs and Examples tabs.



8. Select the Examples tab and note the many examples you can try. The Middleware examples need an MDK license.
9. Beside CMSIS-RTOS Blinky (STM32756G-EVAL), click on Copy. 
10. The Copy Example window below opens up: Select Use Pack Folder Structure: Unselect Launch μ Vision:
11. Type in C:\ as shown to the right: Click OK. Blinky will be copied to C:\MDK\Boards\ST\STM32756G_EVAL\
12. Close the Packs Installer. Open it any time by clicking on its icon.



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

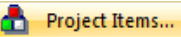


TIP: An Update icon means there is an updated Software Pack available for download. 

Configure and RUN ETM Trace for the Blinky Example:




Connect the ULINKpro and load the Blinky example:

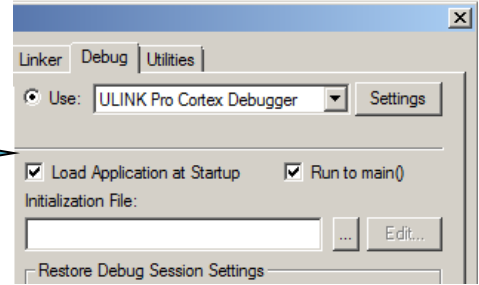
1. Connect the ULINKpro to the STM3240G board using the 20 pin CN13 Trace connector. See the photos below:
2. Select Project/Open Project. Open C:\MDK\Boards\ST\STM32756G_EVAL\Blinky.uvprojx.


Create a new Target Options configuration:

1. Select Project/Manage/Project Items: 
3. Click on the Insert icon:  Enter **ULINKpro Flash** (or whatever you want). Enter and then click Ok to close.
4. Select the Target Option you just created: 





Configure the ETM Trace:

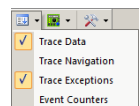
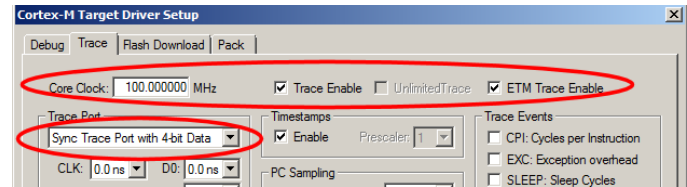
5. Select Target Options  or ALT-F7. Select the Debug tab. Select ULINK Pro Cortex Debugger as shown here: 
6. Select Settings: on the right side of this window.
7. In the next window, confirm that a valid ID Code and Device Name are displayed. If not, you **must** fix this before continuing.
8. Select the Trace tab.
9. In the Trace window, select Trace Enable as shown below. Set Core Clock: to 100 MHz.
10. In the Trace Port box, select Sync Trace Port with 4 bit Data as shown below: 
11. Select ETM Trace Enable as shown here:
12. Click OK twice to return to the main μ Vision menu. ETM trace is now configured.



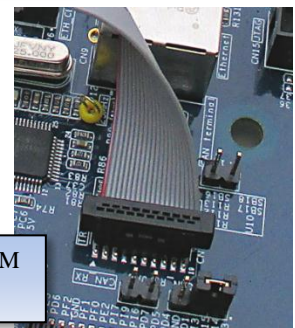
13. Select File/Save All. 

Compile Blinky and Enter Debug Mode:

14. In Blinky near line 60 change PLLN to 200. This is to slow the CPU clock to 100 MHz.  `60 RCC_OscInitStruct.PLL.PLLN = 200;`
15. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
16. Enter Debug mode by clicking on the Debug icon. . The program will run to the first instruction in main().
17. **DO NOT CLICK ON RUN YET !!!**
18. Open the Data Trace window by clicking on the small arrow beside the Trace Windows icon. 
19. The Trace Data window will open and some frames will be displayed. Please turn the page.



An STM32756G_EVAL board connected to a Keil ULINKpro using the special CoreSight 20 pin ETM connector CN12



Close-up of the CoreSight ETM 20 pin connector:

Examine the Trace Data Window:

1. Examine the Instruction Trace window as shown below: This is a complete record of all the program flow since RESET to the start of main(). μ Vision halted the program at the start of main() since Run To main is selected.
2. The last two frames are actually SWV Exception frames and not instruction frames. Open the Exception Trace window and click on the Count column to bring SVCALL to the top. In my case, two exceptions were executed.
3. In this case, 100 060 s shows the last instruction to be executed. (BX lr). In the Register window, the PC will display the location of the next instruction to be executed (0x0800_2428 in my case).
4. Features of The Trace Data window:
 - a. Address of the instruction with its mnemonic is shown with other information.
 - b. Any Serial Wire Viewer (SWV) events are displayed – see the last two frames.
 - c. Source Code if available is displayed.
 - d. The function the instruction belongs to is displayed. The start of a sequence is in orange as shown:
5. Click on an instruction frame and it will be displayed in the Disassembly and source windows.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x08003CC4	LDR r0,[r0,#0x00]	(os_tsk.run->stack[0] != MAGIC_WORD) {	rt_stk_check
	X: 0x08003CC6	LDR r1,[pc,#16] ; @0x0800...		rt_stk_check
	X: 0x08003CC8	CMP r0,r1		rt_stk_check
0.000 099 490 s	X: 0x08003CCA	BEQ 0x08003CD2		rt_stk_check
	X: 0x08003CD2	BX lr	}	rt_stk_check
	X: 0x08000354	POP {r2-r3}	POP {R2,R3}	SVC_Handler
	X: 0x08000356	STR r2,[r3,#0x00]	STR R2,[R3] ; os_tsk.run = os_tsk.new	SVC_Handler
	X: 0x08000358	LDR r12,[r2,#0x28]	LDR R12,[R2,#TCB_TSTACK] ; os_tsk.new->tsk_stack	SVC_Handler
	X: 0x0800035C	LDM r12!,[r4-r11]	LDMIA R12!,[R4-R11] ; Restore New Context	SVC_Handler
	X: 0x08000360	LDRB r0,[r2,#0x25]	LDRB R0,[R2,#TCB_STACKF] ; Stack Frame	SVC_Handler
	X: 0x08000364	CMP r0,#0x00	CMP R0,#0 ; Basic/Extended Stack Frame	SVC_Handler
	X: 0x08000366	ITEE EQ	MVNEQ LR,#NOT:0xFFFFFFF0 ; set EXC_RETURN value	SVC_Handler
	X: 0x08000368	MVNEQ lr,#0x02		SVC_Handler
	X: 0x0800036C	MVNNE lr,#0x12	MVNNE LR,#NOT:0xFFFFFFF0	SVC_Handler
	X: 0x08000370	VLDMIANE r12!,[s16-s31]	VLDMIANE R12!,[S16-S31] ; restore VFP hi-registers	SVC_Handler
	X: 0x08000374	MSR PSP,r12	MSR PSP,R12 ; Write PSP	SVC_Handler
0.000 100 060 s	X: 0x08000378	BX lr	BX LR	SVC_Handler
0.000 100 290 s		Exception Exit - SVCcall		
0.000 100 640 s		Exception Return		

6. Click on Single Step once. The instruction at 0x0800 2428 is executed as shown below: This is a BL.W instruction branching to the MPU_Config function which is the 1st function call in main().

	X: 0x08000374	MSR PSP,r12	MSR PSP,R12 ; Write PSP	SVC_Handler
0.000 100 060 s	X: 0x08000378	BX lr	BX LR	SVC_Handler
0.000 100 290 s		Exception Exit - SVCcall		
0.000 100 640 s		Exception Return		
0.000 100 690 s	X: 0x08002428	BLW MPU_Config (0x0800...	MPU_Config(); /* Configure the MPU */	main

8. Scroll to the top of the Instruction Trace window to frame # 1. This is the first instruction executed after RESET.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x08000424	LDR r0,[pc,#36] ; @0x0800...	LDR R0,=SystemInit	Reset_Handler
0.000 000 000 s	X: 0x08000426	BLX r0	BLX R0	Reset_Handler
	X: 0x080021A0	LDR r0,[pc,#80] ; @0x0800...	SCB->CPACR [(SUL << 10*2)](SUL << 11*2); /* set CP10 a...	SystemInit
	X: 0x080021A2	LDR r0,[r0,#0x00]		SystemInit
	X: 0x080021A4	ORR r0,r0,#0xF00000		SystemInit
	X: 0x080021A8	LDR r1,[r0,#77] ; @0x0800...		SystemInit

9. Note the RESET_Handler at location 0x0800 0424. If you enter 0x0800 0000 in the Memory window, the second word is the initial PC. It will be 0x0800 0425. The least significant bit means this is a Thumb2 instruction.

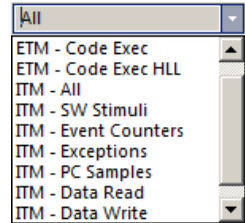
Concept: Think of the Source and Disassembly windows as displaying the source code as it was written. ETM instruction trace displays the source code in the order it was actually executed making this feature extremely useful for debugging.

Searching for Trace Frames:

Capturing all the instructions executed is possible with ULINK_{pro} but this might not be practical. It is not easy sorting through millions and billions of trace frames or records looking for the ones you want. You can use Find, Trace Triggering, Post Filtering or save everything to a file and search with a different application program such as a spreadsheet.

Trace Filters:


In the Trace Data window you can select various types of frames to be displayed. Open the Display: box and you can see the various options available as shown here: These filters are post collection. Future enhancements to μ Vision will allow more precise filters to be selected.

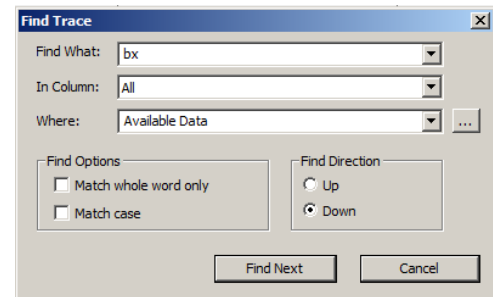


Find a Trace Record:

1. In the Find a Trace Record box enter bx as shown here:



2. Select the Find a Trace Record icon  and the Find Trace window screen opens as shown here: Click on Find Next and each time it will step through the Trace records highlighting each occurrence of the instruction bx.
3. Click on Cancel to close the Find Trace window.



TIP: You can select properties where you want to search in the “in” box. All is shown in the screen above

Trace Triggering: *coming soon for Cortex-M7*

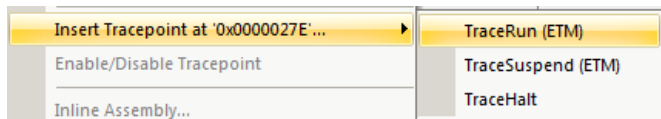
μ Vision has three trace triggers currently implemented:

TraceRun: Starts ETM trace collection when encountered.

TraceSuspend: Stops ETM trace collection when encountered. TraceRun has to have been encountered for this to have an effect.

These two commands have no effect on SWV or ITM. TraceRun starts the ETM trace and TraceSuspend stops it.

TraceHalt: Stops ETM trace, SWV and ITM. Can be resumed only with a STOP/RUN sequence. TraceStart will not restart this.



How it works:

When you set a TraceRun point in assembly language point, ULINK_{pro} will start collecting trace records. When you set a TraceSuspend point, trace records collection will stop there. EVERYTHING in between these two times will be collected. This includes all instructions through any branches, exceptions and interrupts.



TIP: How to *Easily* Configure the STM32F7 Clocks:

The smartest way is to utilize STM32CubeMX to determine the values to be used for a clock speed. Select the Clock Configuration tab after you have chosen your processor. This works backwards: select the final clock speed you desire in the HCLK box and press Enter. The various divider values will be calculated. You can then fill them in the appropriate places in Blinky.c and you did on the previous page 31. You can also right-click on a box to force it to remain (lock) the same. This is useful if you need to keep a clock value for a peripheral such as USB or CAN a specified value and the CPU clock different.








How to Set Trace Triggers:

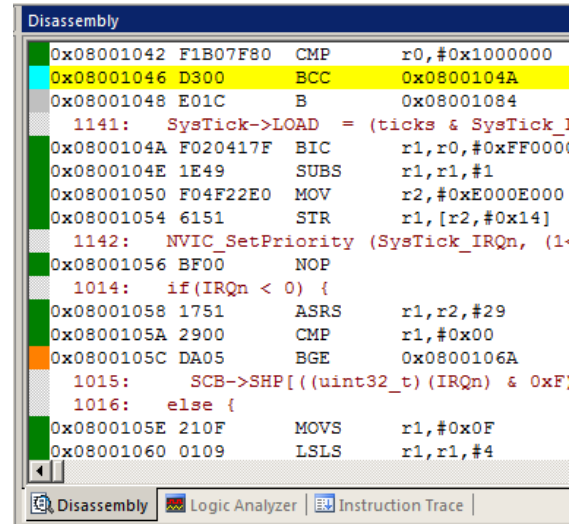
Coming Soon for Cortex-M7 Processors:

Code Coverage (CC):

10. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
11. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin.
12. 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 was always not taken.
-  4. Cyan: the Branch was always taken.
-  5. Light Gray: there is no assembly instruction here.
-  6. RED: a Breakpoint is set here.
-  7. The next instruction to be executed.



```

0x08001042 F1B07F80 CMP      r0,#0x1000000
0x08001046 D300    BCC     0x0800104A
0x08001048 E01C    B      0x08001084
          1141: SysTick->LOAD = (ticks & SysTick_1
0x0800104A F020417F BIC     r1,r0,#0xFF0000
0x0800104E 1E49    SUBS   r1,r1,#1
0x08001050 F04F22E0 MOV     r2,#0xE000E000
0x08001054 6151    STR    r1,[r2,#0x14]
          1142: NVIC_SetPriority (SysTick_IRQn, (1-
0x08001056 BF00    NOP
          1014: if (IRQn < 0) {
0x08001058 1751    ASRS   r1,r2,#29
0x0800105A 2900    CMP    r1,#0x00
0x0800105C DA05    BGE   0x0800106A
          1015: SCB->SHP[((uint32_t)(IRQn) & 0xF)-
          1016: else {
0x0800105E 210F    MOVS  r1,#0x0F
0x08001060 0109    LSLS  r1,r1,#4
  
```

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

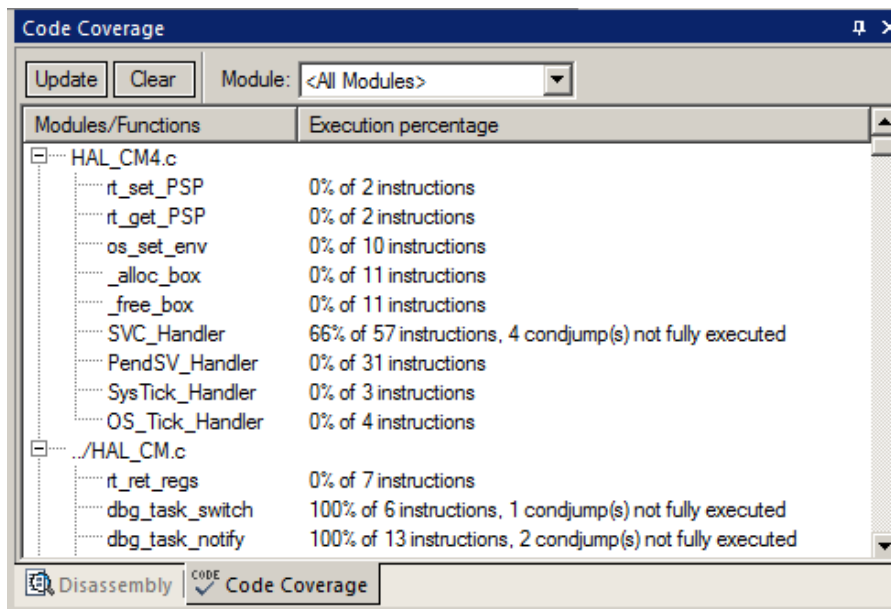
Why was the branch BCC always taken resulting in 0x0800_1048 never being executed ? Or why the branch BGE at 0x800_105C was never taken ? You should devise tests to execute these instructions so you can test them.

Code Coverage indicates 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 obviously have not been 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 for proper operation.

Code Coverage is provided by ETM instruction trace. 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. Your display may look different due to different compiler options.



Modules/Functions	Execution percentage
HAL_CM4.c	
rt_set_PSP	0% of 2 instructions
rt_get_PSP	0% of 2 instructions
os_set_env	0% of 10 instructions
_alloc_box	0% of 11 instructions
_free_box	0% of 11 instructions
SVC_Handler	66% of 57 instructions, 4 condjump(s) not fully executed
PendSV_Handler	0% of 31 instructions
SysTick_Handler	0% of 3 instructions
OS_Tick_Handler	0% of 4 instructions
./HAL_CM.c	
rt_ret_regs	0% of 7 instructions
dbg_task_switch	100% of 6 instructions, 1 condjump(s) not fully executed
dbg_task_notify	100% of 13 instructions, 2 condjump(s) not fully executed

Saving Code Coverage (CC) information:

Code Coverage information is temporarily saved during a run and is displayed in various windows as already shown. It is possible to save this information in an ASCII file for use in other programs.

TIP: To get help on Code Coverage, type Coverage in the Command window and press the F1 key.

You can Save Code Coverage in two formats:

1. In a binary file that can be later loaded back into μ Vision. Use the command Coverage Save *filename*.
2. In an ASCII file. You can either copy and paste from the Command window or use the log command:
 - 1) log > c:\cc\test.txt ; send CC data to this file. The specified directory must exist.
 - 2) coverage asm ; you can also specify a module or function.
 - 3) log off ; turn the log function off.

1) Here is a partial display using the command **coverage**. This displays and optionally saves everything.

```
\\Blinky\HAL_CM4.c\rt_set_PSP - 0% (0 of 2 instructions executed)
\\Blinky\HAL_CM4.c\rt_get_PSP - 0% (0 of 2 instructions executed)
\\Blinky\HAL_CM4.c\os_set_env - 0% (0 of 10 instructions executed)
\\Blinky\HAL_CM4.c\alloc_box - 0% (0 of 11 instructions executed)
\\Blinky\HAL_CM4.c\free_box - 0% (0 of 11 instructions executed)
\\Blinky\HAL_CM4.c\SVC_Handler - 66% (38 of 57 instructions executed)
    4 condjump(s) or IT-block(s) not fully executed
\\Blinky\HAL_CM4.c\PendSV_Handler - 0% (0 of 31 instructions executed)
\\Blinky\HAL_CM4.c\SysTick_Handler - 0% (0 of 3 instructions executed)
\\Blinky\HAL_CM4.c\OS_Tick_Handler - 0% (0 of 4 instructions executed)
\\Blinky\..\HAL_CM.c\rt_ret_regs - 0% (0 of 7 instructions executed)
\\Blinky\..\HAL_CM.c\dbg_task_switch - 100% (6 of 6 instructions executed)
    1 condjump(s) or IT-block(s) not fully executed
\\Blinky\..\HAL_CM.c\dbg_task_notify - 100% (13 of 13 instructions executed)
```

2) The command **coverage asm** produces this listing (partial is shown):

```
EX | 0x080004D6 468D    MOV        sp,r1
EX | 0x080004D8 4770    BX         lr
\\Blinky\Blinky.c\__use_no_semihosting_swi - 0% (0 of 1 instructions executed)
NE | 0x080004DA __I$use$semihosting:
NE | 0x080004DA 4770    BX         lr
\\Blinky\Blinky.c\_fp_init - 100% (3 of 3 instructions executed)
EX | 0x0800443C      _fp_init:
EX | 0x0800443C F04F7040 MOV        r0,#0x3000000
EX | 0x08004440 EEE10A10 VMSR      FPSCR, r0
EX | 0x08004444 __fplib_config_fpu_vfp:
EX | 0x08004444 4770    BX         lr
```

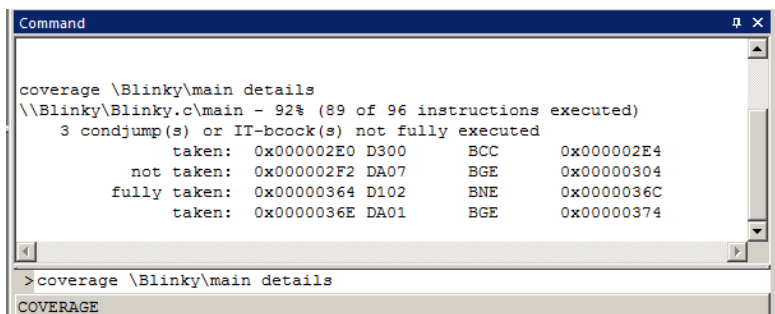
3) The first column above describes the execution as follows:

NE	Not Executed
FT	Branch is fully taken
NT	Branch is not taken
AT	Branch is always taken.
EX	Instruction was executed (at least once)

4) Shown here is an example using:
coverage \Blinky\main details

If the log command is run, this will be saved/appended to the specified file.

You can enter the command coverage with various options to see what is displayed.






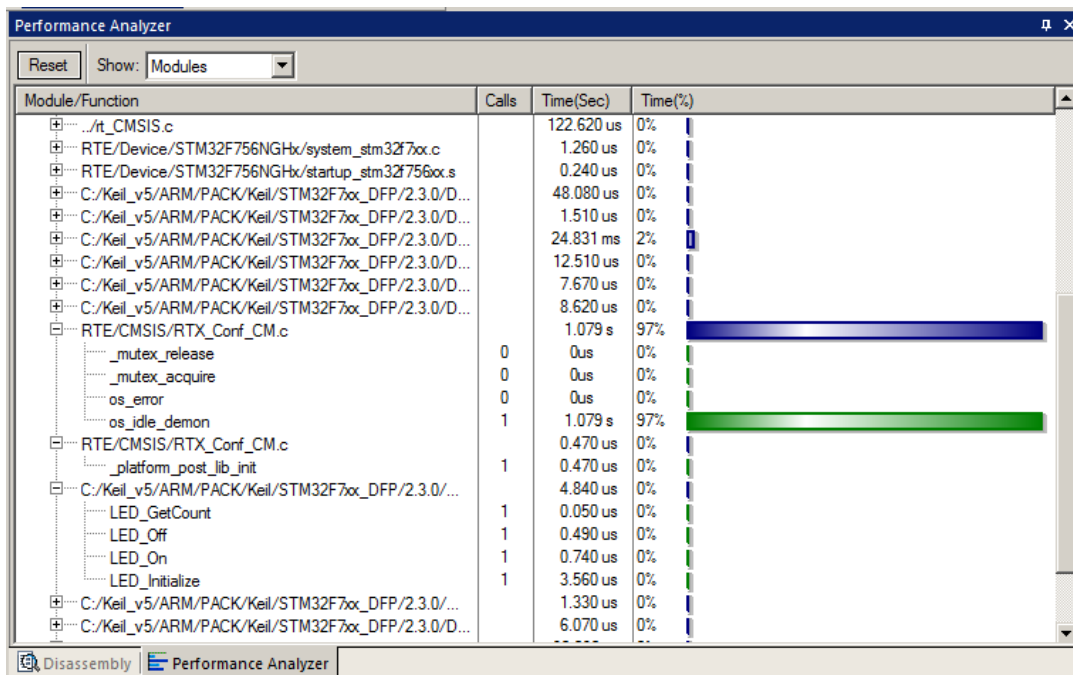
```
Command
coverage \Blinky\main details
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
    3 condjump(s) or IT-block(s) not fully executed
        taken: 0x000002E0 D300    BCC    0x000002E4
        not taken: 0x000002F2 DA07    BGE    0x00000304
        fully taken: 0x00000364 D102    BNE    0x0000036C
        taken: 0x0000036E DA01    BGE    0x00000374
>coverage \Blinky\main details
COVERAGE
```

Performance Analysis (PA):


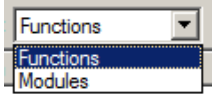


Performance Analysis tells you how much time was spent in each function. The data can be provided by either the SWV PC Samples or the ETM. If provided by the SWV, the results will be statistical and accuracy is improved with longer runs. Small loops could be entirely missed. ETM provides complete Performance Analysis. Keil provides ETM and not SWV PA.

Keil provides Performance Analysis with the μ Vision simulator or with ETM and the ULINK pro . The number of total calls made as well as the total time spent in each function is displayed. A graphical display is generated for a quick reference. If you are optimizing for speed, work first on those functions taking the longest time to execute.

1. Use the same setup as used with Code Coverage.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Exit Debug mode and immediately re-enter it.  This clears the PA window and resets the STM32 and reruns it to main() as before. Or select the Reset icon in the PA window to clear it.
4. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
5. Expand some of the module names as shown below:
6. Note the execution information that has been collected in this initial short run. Both times and number of calls are displayed. You will probably see a different set of data in your program.
7. We can tell that most of the time at this point in the program has been spent in the os_idle-demon.





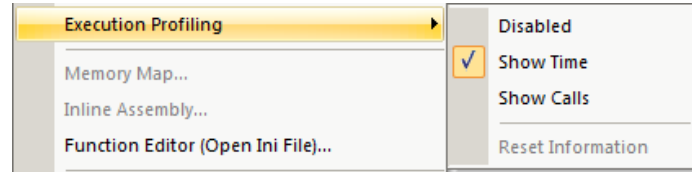
Module/Function	Calls	Time(Sec)	Time(%)
../rt_CMSIS.c		122.620 us	0%
RTE/Device/STM32F756NGHx/system_stm32f7xx.c		1.260 us	0%
RTE/Device/STM32F756NGHx/startup_stm32f756xx.s		0.240 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		48.080 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		1.510 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		24.831 ms	2%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		12.510 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		7.670 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		8.620 us	0%
RTE/CMSIS/RTX_Conf_CM.c		1.079 s	97%
_mutex_release	0	0us	0%
_mutex_acquire	0	0us	0%
os_error	0	0us	0%
os_idle_demon	1	1.079 s	97%
RTE/CMSIS/RTX_Conf_CM.c		0.470 us	0%
_platform_post_lib_init	1	0.470 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/...		4.840 us	0%
LED_GetCount	1	0.050 us	0%
LED_Off	1	0.490 us	0%
LED_On	1	0.740 us	0%
LED_Initialize	1	3.560 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/...		1.330 us	0%
C:/Keil_v5/ARM/PACK/Keil/STM32F7xx_DFP/2.3.0/D...		6.070 us	0%

8. Click on the RUN icon. 
9. 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.
10. Select Functions from the pull down box as shown here and notice the difference. 
11. Exit and re-enter Debug mode again and click on RUN. Note the different data set displayed.
12. When you are done, Stop the program  and exit Debug mode. 

Execution Profiling:

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

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



Time:	Calls:	Average:
19.599 s	139910257 *	0.140 μ s

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

Disassembly

```

70:  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
0.010 us  0x0800217E 2002  MOVS    r0,#0x02
0.010 us  0x08002180 900D  STR    r0,[sp,#0x34]
71:  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
0.010 us  0x08002182 2000  MOVS    r0,#0x00
0.010 us  0x08002184 900E  STR    r0,[sp,#0x38]
72:  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
0.010 us  0x08002186 F44F50A0 MOV    r0,#0x1400
0.010 us  0x0800218A 900F  STR    r0,[sp,#0x3C]
73:  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
0.010 us  0x0800218C F44F5080 MOV    r0,#0x1000
0.010 us  0x08002190 9010  STR    r0,[sp,#0x40]
74:  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);

```

Blinky.c startup_stm32f756xx.s RTX_Conf_CM.c Thread_LED.c HAL_CM4.c mfxstm32l152.c

```

61  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
62  RCC_OscInitStruct.PLL.PLLQ = 9;
63  0.210 us HAL_RCC_OscConfig(&RCC_OscInitStruct);
64
65  /* Activate the OverDrive to reach the 216 MHz Frequency */
66  0.080 us HAL_PWREx_EnableOverDrive();
67
68  /* Select PLL as system clock source and configure the HCLK, PCLK1 a
69  0.020 us RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_
70  0.020 us Time:      Calls:      Average:  = RCC_SYSCLKSOURCE_PLLCLK;
71  0.020 us 0.020 us  1 *      0.020 us  = RCC_SYSCLK_DIV1;
72  0.020 us RCC_ClkInitStruct.APB1CLKDivide = RCC_HCLK_DIV4;

```

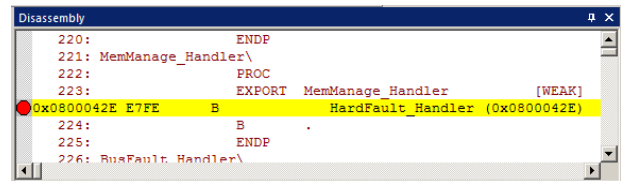
In-the-Weeds Example: A ULINKpro is needed for ETM Instruction Trace:

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

If a Hard Fault occurs, the CPU will end up at the address specified in the Hard Fault vector located at 0x0800 042E. This address points to the Hard Fault handler. This is a branch to itself by default and this Branch instruction will run forever. The trace buffer will save millions of the same branch instructions. This is not useful. We need to stop the CPU at this point.

This exception vector is found in the file startup_stm32f756xx.s. If we set a breakpoint by clicking on the Hard Fault handler and run the program: at the next Hard Fault event the CPU will jump to the Hard Fault handler (in this case located at 0x0800 042E as shown to the right) and stop.




The CPU and also the trace collection will stop. The trace buffer will be visible and is useful to find the cause of the crash.



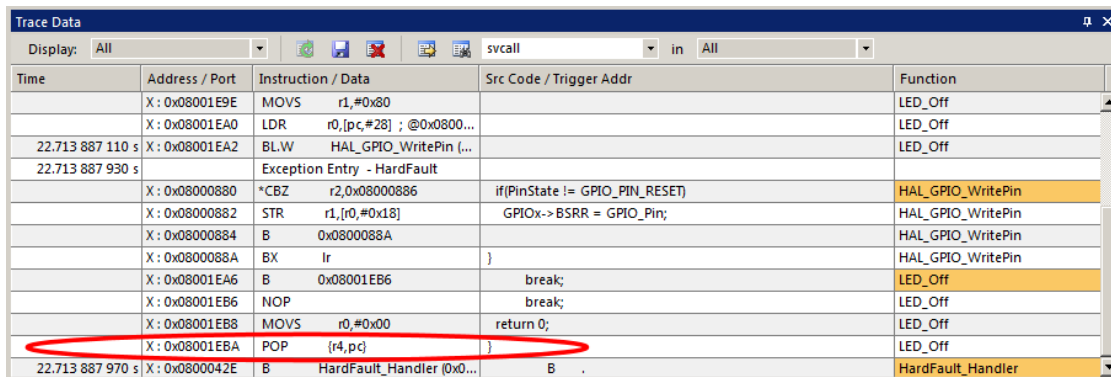
```
Disassembly
220:      ENDP
221: MemManage_Handler\
222:      PROC
223:      EXPORT MemManage_Handler [WEAK]
● 0x0800042E E7FE B HardFault_Handler (0x0800042E)
224:      B
225:      ENDP
226: BusFault_Handler\
```

1. Open the Blinky example and enter Debug mode. Open the Trace Data window.
2. Locate the Hard Fault vector in startup_stm32f756xx.s near line 219 or at 0x0800 042E in the Disassembly window.
3. Set a breakpoint at this point. A red circle will appear as shown above.


TIP: An alternative to setting a breakpoint on the Hard Fault vector is described here: www.keil.com/support/docs/3782.htm

4. In Thread_LED.c, set another breakpoint on the call to LED_Off(led_num); near line 29.
5. Click on RUN . The program will run to the breakpoint in the LED_Off function call.
6. Click on Step Into  **once** to enter the function LED_Off. Verify this in the Call Stack and Locals window.
7. In the Registers window, set R14(LR) to 0x0. This will guarantee a Hard Fault on the next return.
8. Clear the Trace Data window to make it easier to see what is happening. This is an optional step.
9. Click on RUN  and immediately the program will stop on the Hard Fault exception branch instruction.
10. Examine the Trace Data window and you find a POP plus everything else that was previously executed.
11. Double click on the POP instruction and this will be displayed in the Disassembly window.

TIP: The addresses you get might be different than these ones depending on compiler settings.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x08001E9E	MOVS r1,#0x80		LED_Off
	X: 0x08001EA0	LDR r0,[pc,#28] ; @0x0800...		LED_Off
22.713 887 110 s	X: 0x08001EA2	BL.W HAL_GPIO_WritePin (...)		LED_Off
22.713 887 930 s		Exception Entry - HardFault		
	X: 0x08000880	*CBZ r2,0x08000886	if(PinState != GPIO_PIN_RESET)	HAL_GPIO_WritePin
	X: 0x08000882	STR r1,[r0,#0x18]	GPIOx->BSRR = GPIO_Pin;	HAL_GPIO_WritePin
	X: 0x08000884	B 0x0800088A		HAL_GPIO_WritePin
	X: 0x0800088A	BX lr	}	HAL_GPIO_WritePin
	X: 0x08001EA6	B 0x08001EB6	break;	LED_Off
	X: 0x08001EB6	NOP	break;	LED_Off
	X: 0x08001EB8	MOVS r0,#0x00	return 0;	LED_Off
	X: 0x08001EBA	POP {r4,pc}	}	LED_Off
22.713 887 970 s	X: 0x0800042E	B HardFault_Handler (0x0...	B .	HardFault_Handler

12. The Branch at the Hard Fault does not show in the trace window because a hardware breakpoint does not execute the instruction it is set to. It is therefore not recorded in the trace buffer. Click Step Into  and it will be executed as and displayed as shown above. The Hard Fault will also be displayed in the Call Stack window.
13. The frames above the POP are a record of all previous instructions executed and tells you the complete program flow from when you clicked RUN to when the event that caused the Hard Fault occurred. Your problem is in there.

Instruction Trace is very useful for locating difficult bugs. This is a simple contrived example but it is clear to see the usefulness of ETM. See the page 40 for a list of uses ETM and SWV can be. **This is the end of the exercises.**

32) Serial Wire Viewer and ETM Trace Summary:

Serial Wire Viewer can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers and physical memory – 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.
- ITM for printf.

ETM Trace is good for:

- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened* as opposed to the way the program was written.
- Trace can often find nasty problems very quickly. Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time before the consequences are seen.
- Or where the state of the system disappears with a change in scope(s).



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

- Pointer problems. Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), a corrupted stack.
How did I get here ?
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. ETM trace is best for this.
- Communication protocol and timing issues. System timing problems.
- ETM facilitates Code Coverage, Performance Analysis and program flow debugging and analysis.

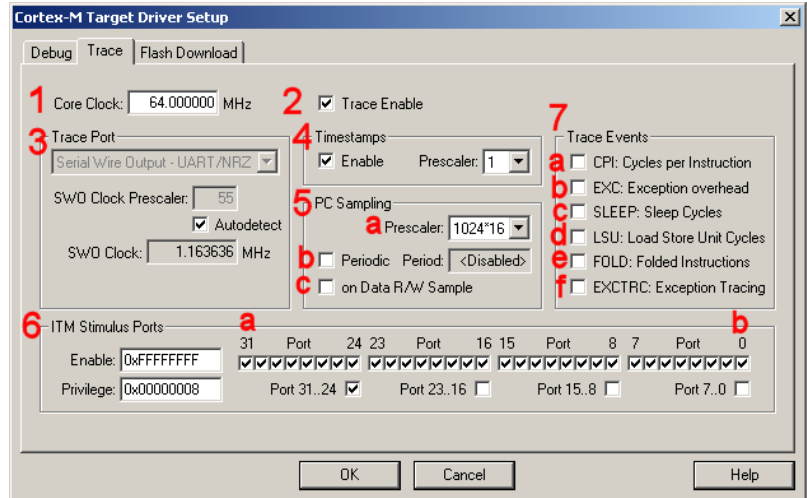
For information on Instruction Trace (ETM) please visit www.keil.com/st for other labs discussing ETM.

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

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

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

- 1) **Core Clock:** The CPU clock speed for SWV. The CPU speed can be found in your startup code or in Abstract.txt. It is usually called SystemCoreClock and can be viewed in a Watch window. This **must** be set correctly for all adapters except ULINKpro.
- 2) **Trace Enable:** Enables SWV and ITM. It can only be changed in Edit mode. This does not affect the Watch and Memory window display updates.
- 3) **Trace Port:** This is preset for ST-Link.
- 4) **Timestamps:** Enables timestamps and selects the Prescaler. 1 is the default.
- 5) **PC Sampling:** Samples the program counter:



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

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

33) Document Resources:

www.keil.com/st

Books:

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

Application Notes:

9. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
10. Segger emWin GUIBuilder with μ Vision™ www.keil.com/appnotes/files/apnt_234.pdf
11. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
12. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
13. Using μ Vision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
14. RTX CMSIS-RTOS in MDK 5 C:\Keil_v5\ARM\Pack\ARM\CMSIS\3.20.4\CMSIS_RTXDownload
15. RTX CMSIS-RTX www.keil.com/demo/eval/rtx.htm **and** www.arm.com/cmsis
16. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
17. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
18. Cortex Debug Connectors: www.arm.com and search for cortex_debug_connectors.pdf
19. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
20. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp

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

Keil Online CMSIS Documentation: www.keil.com/pack/doc/CMSIS/General/html

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

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

mbed: <http://mbed.org>

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

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

34) Keil Products and Contact Information:

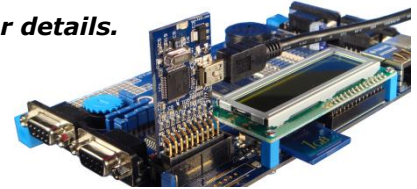
Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite™ (Evaluation version) \$0
- MDK- for STM32L0 and STM32F0: www.keil.com/st \$0 Free MDK for STM32 F0/L0
- MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit)
- MDK-Standard™ (unlimited compile and debug code and data size)
- MDK-Professional™ (Includes Flash File, TCP/IP, CAN and USB driver libraries)

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

USB-JTAG adapters www.keil.com/ulink

- ULINK2 - (ULINK2 and ME - SWV only – no ETM)
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINKpro – Cortex-Mx SWV & ETM trace. Very fast Flash programming.
- ULINKpro D – same as ULINKpro but without ETM trace.
- MDK also supports ST-Link, CMSIS-DAP and Segger J-Link Debug adapters.



All versions, including MDK-Lite, includes Keil RTX RTOS with source code! and a BSD license. www.keil.com/rtx or C:\Keil_v5\ARM\Pack\ARM\CMSIS

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

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

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

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

MDK supports all STM32 Cortex-M0, M3, M4 and M7 processors. Keil supports many other ST processors including 8051, ARM7, ARM9™ and ST10 processors. See the Keil Device Database® on www.keil.com/dd for the complete list of STMicroelectronics support. This information is also included in MDK.

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

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

For Linux, Android, other OSs and no OS support on ST Cortex-A processors such as SPEAr, see DS-5 www.arm.com/ds5.



For more information:

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

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

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

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

CMSIS Version 4: See www.arm.com/cmsis and <http://community.arm.com/groups/tools/content> for more information.

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

