Cortex™-M3 Hands-On LAB featuring Serial Wire Viewer

MDK-ARM™ Microcontroller Development Kit

Keil® MCBSTM32™ Evaluation Board with ULINK2 USB to JTAG Adapter

with the STMicroelectronics STM32F103RB Processor

Version 4.04 May, 2008

Table of Contents

Contacts:

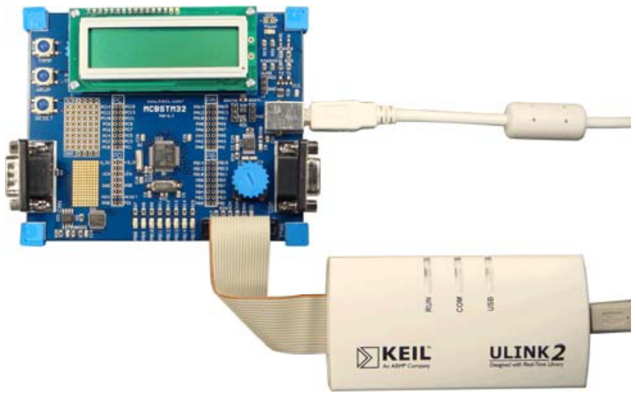
Robert Boys robert.boys@arm.com

Introduction:

1. The purpose of these hands-on exercises is to give you exposure to the RealView MDK® microprocessor development kit and enhance familiarity with the Cortex-M3 core. Emphasis is placed on demonstrating the Real-Time Trace: Serial Wire Viewer (SWV) which is a component of CoreSight™ on-chip debug and trace technology. You will learn some interesting facts about the Cortex-M3 processor.
2. The target processor is the STMicrolectronics STM32F103RB 32 bit ARM Coretx-M3 processor. The Cortex-M3 is the latest ARM processor and is targeted for the embedded market.
3. You will use Keil example programs to configure µVision, compile and load these programs into the processor Flash memory. You will then put µVision into debug mode and you can start and stop the program and observe various data inside the processor. Most will be viewed in real-time with the Serial Wire Viewer.
4. Where possible, we will use the Keil default settings. MDK is designed to offer "out of the box" operation. It takes a minimal effort to get projects configured, compiled, loaded and running. This saves much time. You do not need to write any new code.
5. This document requires the use of ARM® MDK Version 3.20. The IDE (Integrated Development Environment) used is Keil µVision®3. MDK uses the ARM RealView toolset including C, C++ compilers, libraries, assembler and linker and with the full version, the Keil RTX RTOS kernel is included.

   Later versions of MDK will work but the example programs are subject to change as is µVision. Earlier versions of MDK do not have the Serial Wire Viewer incorporated into µVision. Please contact M Onions or R Boys or Keil technical support for the latest version of this document.

6. You can download MDK free from www.keil.com. Without a license key, µVision will run in evaluation mode. Code size is then restricted to 16 Kbytes. This is sufficient to compile and run all the Keil examples.

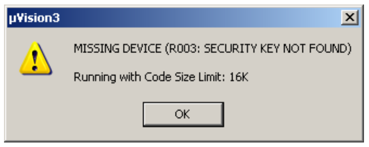


Installation:

1. **Software:**
   When installing MDK, please use the default directories. All the example files are installed with µVision in C:\Keil\Arm\Boards\Keil\MCBSTM32.

   Hardware:
   A Keil MCBSTM32 evaluation board and a ULINK2 or ULINK2-ME USB to JTAG adapter are required. Running the examples on other target boards or adapters is possible but is beyond the scope of this document.

2. All these examples will compile and link with the MDK evaluation version (no license is required). When the dialog box in Figure 1 appears when you enter debug mode, please click on OK. This box does not appear on a licensed copy of MDK.
3. Connect the hardware to your PC as shown in Figure 2. The USB cable connected to the MCBSTM32 board is only to power it. All communication to and from the board is through the Keil ULINK®2 USB-JTAG adapter.


µVision3

MISSING DEVICE (R003: SECURITY KEY NOT FOUND)

Running with Code Size Limit: 16K

Figure 1 Evaluation Mode

4. There must be two (2) USB cables attached to your computer as shown in Figure 2.
5. When all the connections are correct, the MCBSTM32 LCD displays some graphics or letters and a red LED will be on. On the ULINK2, only the red LED will be illuminated.
6. When you start µVision, if the ULINK2 needs its firmware upgraded – this will happen at this time. It will notify you of this fact and will only take a minute or so and then you can continue with the exercises.
7. Many of these exercises can be run without a target hardware using the µVision Simulator. The Serial Wire Viewer will not be available as a real Cortex-M3 chip must be used to access this valuable feature.
8. Please send any corrections and comments to robert.boys@arm.com.

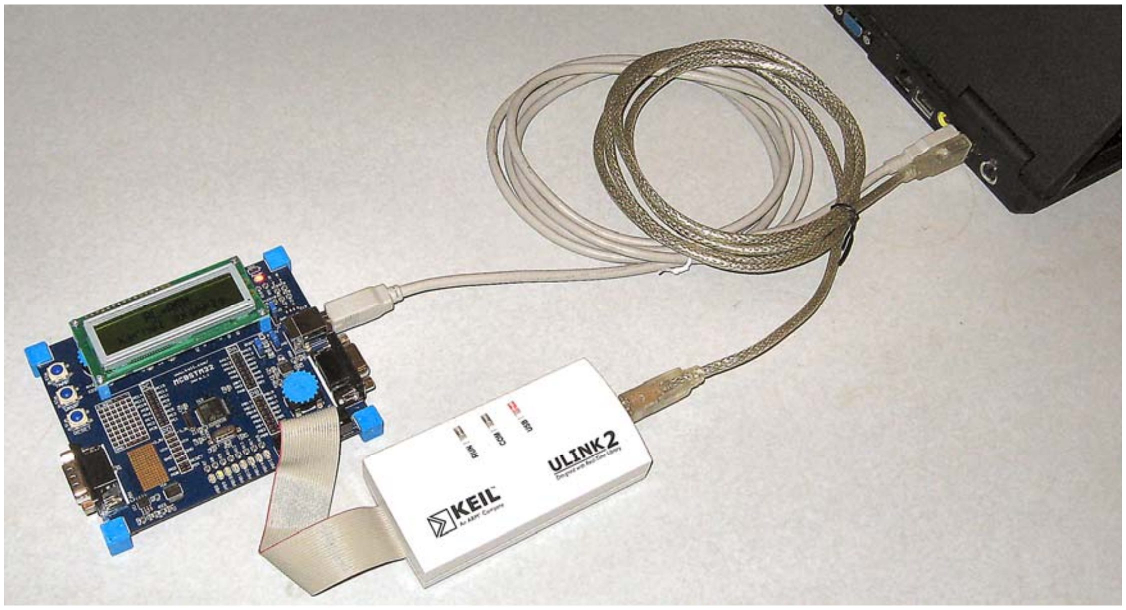


Figure 2 Hardware Connections: MCBSTM32 and ULINK2

Lab exercises introduction: General Information needed to run the exercises.

1. ULINK2 operation and LED meaning: (for reference only – do not start µVision yet)
    a. The “USB” led will illuminate (red) with power connected to its USB cable.
    b. With µVision running and in debug mode, the “COM” led will illuminate (blue).
    c. When a program is running under command of µVision, the “RUN” led will illuminate (blue).
2. If µVision locks up, temporarily disconnect the USB cable to the ULINK2 and the connection will be automatically restored. Control will then be restored to µVision.
3. Edit and Debug mode. µVision has two modes: Edit and Debug. Clicking on the debug icon toggles between these modes. When this icon has a square box around it – µVision is in debug mode.
    a. **Edit mode** is essentially for configuring much of µVision, creating and modifying source files and compiling them and programming the processor Flash memory.
    b. **Debug mode** is essentially for running and stopping the program and viewing registers and memory locations of various types. The RUN and STOP icons are available only in debug mode. The core registers are available only during debug mode. You will quickly learn to know which mode µVision is currently in. This will not create problems for you.
4. Important Steps that might hamper success:
    a. Simulator or MCBSTM32™ Target: Always use the target hardware box set to MCBSTM32 or “STM32 Trace C” for the STLIB_Blinky example and not this: Simulator This box is only changeable when µVision is in Edit mode and not in debugger mode.
    b. ULINK Cortex debugger must always be selected – not “Use Simulator”. This is described in Appendix A, Figure xyz.
    c. Serial Wire Viewer Configuration: See Appendix A. The five most important parts are circled in the three screen shots in Appendix A. They are:
        i. ULINK Cortex Debugger must be both visible and selected.
        ii. The Initialization file STM32DBG.ini must be entered.
        iii. SWJ and SW must be selected.
        iv. Core clock must be set to 72 MHz
        v. Trace Enable must be checked.

If you have problems check these first. The fastest method to get to these menus is to first make sure µVision is in Edit mode (not debugging) and click on the Options for Target icon.

1) Blinky:

This example obtains the value of the pot position on the board using the A/D converter and displays it on the LCD. You will learn how to open a project, compile (rebuild) the source files into an executable, load this into Flash and run it.

1. Start µVision by clicking on its icon.
2. Select Project/Open Project. Open project C:\Keil\ARM\Boards\Keil\MCBSTM32\Blinky\Blinky.Uv2.
3. ***Note:*** there are many Blinky.Uv2 files – make sure you hare loading the correct one ! Check your directories.
4. On the toolbar select MCBSTM32 as the target (not Simulator). MCBSTM32
5. Select the processor in Project/Select Device for Target MCBSTM32. Select STM32F103RB (not R8). Click OK.
6. Everything about this project is pre-configured but you must first compile it with the Rebuild command.
7. Click on Rebuild All Target Files.
8. This will return 0 Error(s) and 0 Warning(s) in the Build window at bottom left of your screen.
9. Now, the FLASH in the processor must be programmed with the executable file you just created.
10. First – make sure the ULINK Cortex debugger is selected as per instructions "Configuring SWV" in Appendix A: steps 1 through 6. Step 4 is optional and not used in this example.
11. Click on the FLASH load icon. . µVision will go through its erase, program and verify cycles.
12. If you get a JTAG Communication Error, repeat step 6 in Appendix A. (SWV might be reset back to JTAG).
13. Put µVision into debug mode. Click on this icon.
14. Run the program by clicking on the Run icon.
15. As you turn the pot on the evaluation board, you will see the LEDS change their speed and the bar on the LCD change as shown in Figure 3. This is correct. Congtartulations ! Your first µVision project !
16. Stop the program execution by clicking on the stop icon.

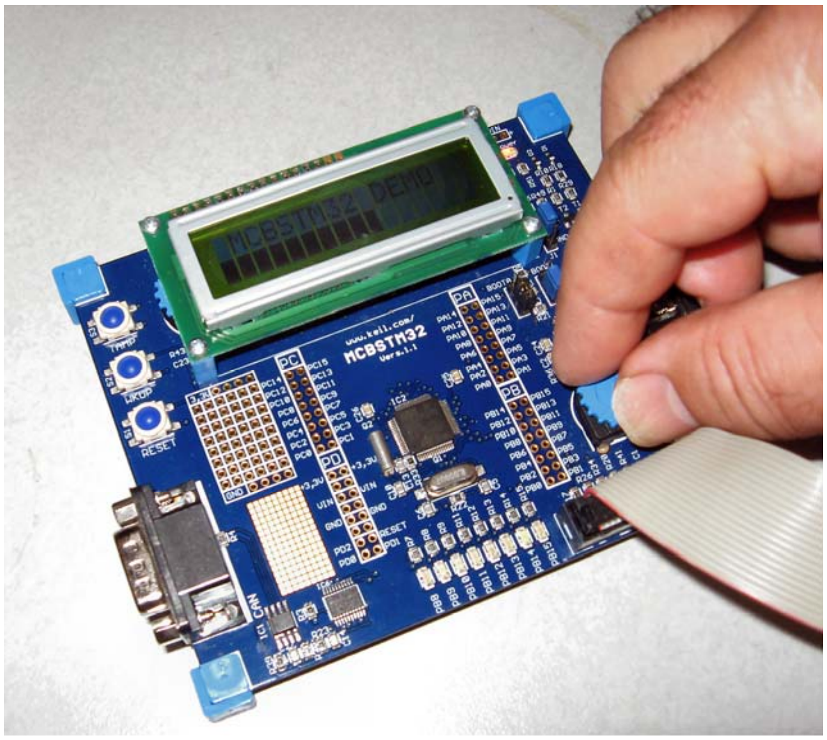
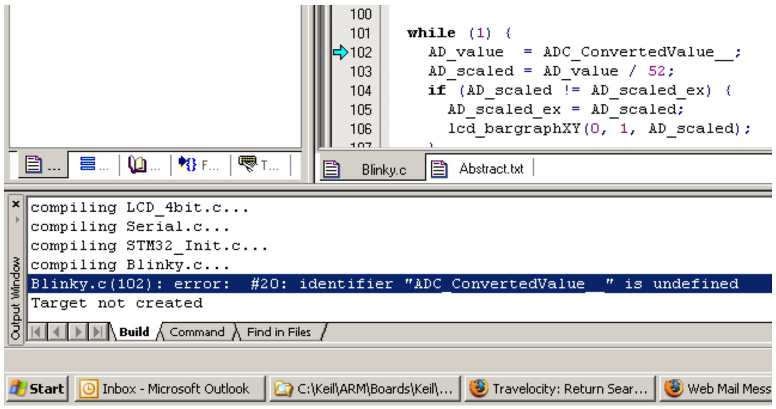


Figure 3 Blinky example – turning the pot.

Finding Programming Errors

1. Take µVision out of debugging mode by clicking on this icon.
2. Insert a programming error into a file. As an example, I added two underscores __ behind the variable ADC_ConvertedValue on line 102 of blinky.c.
3. Rebuild the project. You will get Figure 4.
4. Note the target file was not created because of the error described in the Output window in the line Blinky.c (102) (this line is outlined in Figure 4). Click on this line and µVision takes you to the error.
5. A turquoise arrow will display the offending line in order for you to easily locate and repair it. This is shown in Figure 4 in the file Blinky.c. Note: On some computers this arrow will not show but you are taken to the line.
6. Correct the error and rebuild.
7. **Neat Feature:** Click once on any curly bracket or parenthesis and note that it turns grey and so does its counterpart. This helps you keep track of which braces are related to each other.



```
100
101    while (1) {
102      AD_value  = ADC_ConvertedValue__;
103      AD_scaled = AD_value / 52;
104      if (AD_scaled != AD_scaled_ex) {
105        AD_scaled_ex = AD_scaled;
106        lcd_bargraphXY(0, 1, AD_scaled);
```

```
compiling LCD_4bit.c...
compiling Serial.c...
compiling STM32_Init.c...
compiling Blinky.c...
Blinky.c(102): error:  #20: identifier "ADC_ConvertedValue__" is undefined
Target not created
```

Blinky.c
Abstract.txt
Output Window
Build
Command
Find in Files
Start
Inbox - Microsoft Outlook
C:\Keil\ARM\Boards\Keil\...
Travelocity: Return Sear...
Web Mail Mess

Figure 4 Error messages

Finding the Definition and the Reference of a Variable

Neat Feature: Sometimes it can be quite difficult to find the definition and reference of a variable. µVision makes this very easy. To find the definition or reference of a variable: right click on it and select either definition or reference in the window that opens as shown in Figure 5. You will be taken to this location.

1. In Blinky.c as shown in Figure 4, right click on the variable ADC_ConvertedValue.
2. Select Definition and view the result.
3. Right click again on the definition of this variable and this time select Reference and view the result.
4. Try a harder one: scroll up to line 034 of Blinky.c and right click on DMA_Channel1 and select definition.
5. You will be taken to a different file – a header this time. It would have taken some time to find where this variable was defined without this feature.

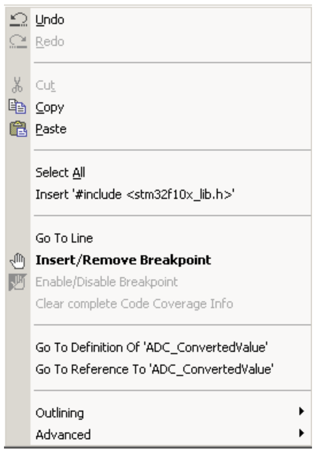
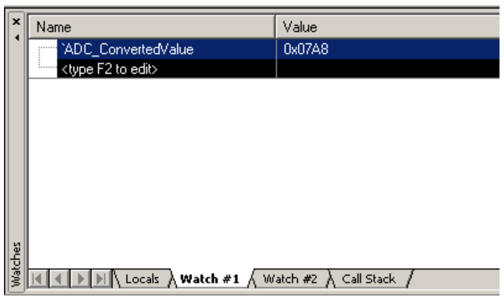

Undo
Redo
Cut
Copy
Paste
Select All
Insert '#include <stm32f10x_lib.h>'
Go To Line
Insert/Remove Breakpoint
Enable/Disable Breakpoint
Clear complete Code Coverage Info
Go To Definition Of 'ADC_ConvertedValue'
Go To Reference To 'ADC_ConvertedValue'
Outlining
Advanced

Figure 5 Finding Definitions and References of variables.

Watch Window

1. Put µVision into Debug mode.
2. If the Watch window shown in Figure 6 is not open in the bottom area of your screen, open it with View/Watch & Call Stack Window. Click on the Watch #1 tab.
3. Click on "type F2 to edit" and press F2. Enter **ADC_ConvertedValue.**
4. This will display the global variable ADC_ConvertedValue which is the value of the pot position.
5. Run the program and vary the pot. The value of ADC_ConvertedValue will also vary and in real-time.
6. Carefully click once or twice on the Value until it is blocked over.
7. Enter a new value and press Enter. This new value will be injected into this memory location in real-time. You will not see the effect since it is quickly over-written by the program. But this is how to do this.
8. Stop program execution and take µVision out of debug mode.



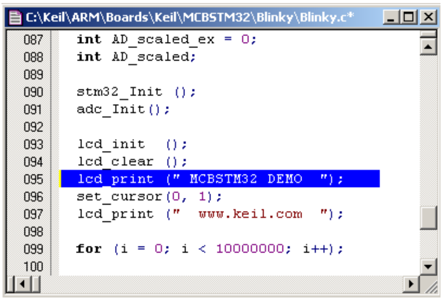
Name	Value
ADC_ConvertedValue	0x07A8
<type F2 to edit>	

Figure 6 Watch window

How It Works: µVision uses the Serial Wire Debug Port (as opposed to the JTAG port) to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive and therefore does not impact the program execution timings. The only time this will be intrusive is in the unlikely event the program running on the CPU and µVision read or write to the same memory location at exactly the same. Then the CPU will be stalled for only one clock cycle. In practice, this cycle stealing effectively never happens.

Changing text on the LCD

1. In the file blinky.c scroll to line number 095 as shown in Figure 7.
2. This is the C code line `lcd_print (" MCBSTM32 DEMO  ");`
3. You can open this file if not already visible by double-clicking on it in the Project Workspace in the Source directory. This is in the upper left hand corner of µVision.
4. Enter your own ASCII text inside the quotes up to 16 characters replacing "MCBSTM32 Demo".
5. Click on Rebuild and then program the processor flash memory.
6. Enter the debug mode and click on Run.
7. Your new text will now be visible on the LCD.
8. Stop program execution and take µVision out of debug mode to be ready for the next exercise.



C:\Keil\ARM\Boards\Keil\MCBSTM32\Blinky\Blinky.c*

```
087   int AD_scaled_ex = 0;
088   int AD_scaled;
089
090   stm32_Init ();
091   adc_Init();
092
093   lcd_init  ();
094   lcd_clear ();
095   lcd_print (" MCBSTM32 DEMO  ");
096   set_cursor(0, 1);
097   lcd_print ("  www.keil.com  ");
098
099   for (i = 0; i < 10000000; i++);
100
```

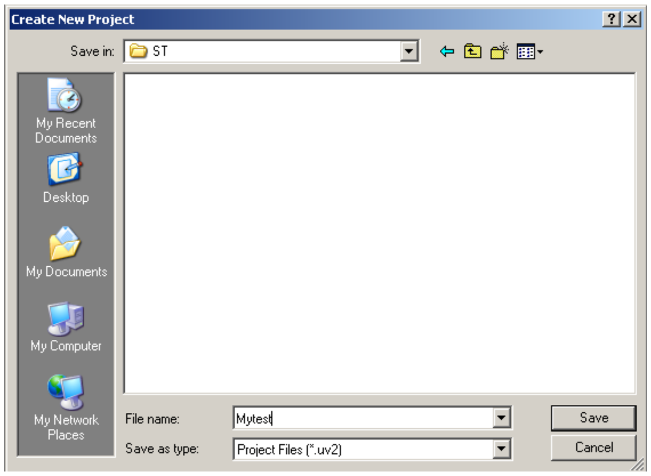
Figure 7 Changing the Text on the LCD

2) Creating a new project: PWM_2: Pulse Width Modulator (TIM4) Example

The pulse width example (PWM_2) is pre-configured as are all the examples provided by Keil. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you will build, load and run the PWM_2 example as usual. You can use this process to create any new project from your own source files created with µVision's editor or any other editor. If you prefer to try out this example without doing it from the beginning, go to Appendix B for instructions.

Create a new project called Mytest:

1. With µVision running and not in debug mode, select Project/New µVision Project.
2. In the window Create New Project go to the folder C:\Keil\ARM\Boards\Keil\MCBSTM32
3. Right click and create a new folder by selecting New/Folder. I named this new folder ST.
4. Double-click on the newly created folder "ST" to enter this folder as is shown in Figure 8.
5. Name your project. I called mine Mytest. You can choose your own name but you will have to keep track of it.
6. Click on Save.



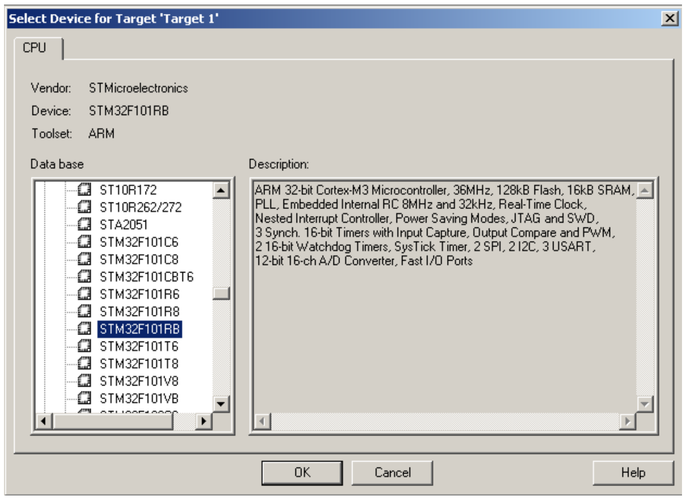
Save in: ST

File name: Mytest

Save as type: Project Files (*.uv2)

Figure 8 Creating a new project name: Mytest

7. The window "Select Device for Target 1" shown in Figure 9 opens up.
8. This is the Keil Device Database® which lists all the devices Keil directly (and publically) supports.
9. Locate the STMicroelectronics directory, open it and select STM32F103RB (not R8). Note the feature list for this device is displayed in Figure 9. You are able to add new devices yourself.
10. Click on OK.



Vendor: STMicroelectronics

Device: STM32F101RB

Toolset: ARM

ST10R172
ST10R262/272
STA2051
STM32F101C6
STM32F101C8
STM32F101CBT6
STM32F101R6
STM32F101R8
STM32F101RB
STM32F101T6
STM32F101T8
STM32F101V8
STM32F101VB

ARM 32-bit Cortex-M3 Microcontroller, 36MHz, 128kB Flash, 16kB SRAM,
PLL, Embedded Internal RC 8MHz and 32kHz, Real-Time Clock,
Nested Interrupt Controller, Power Saving Modes, JTAG and SWD,
3 Synch. 16-bit Timers with Input Capture, Output Compare and PWM,
2 16-bit Watchdog Timers, SysTick Timer, 2 SPI, 2 I2C, 3 USART,
12-bit 16-ch A/D Converter, Fast I/O Ports

Figure 9 Keil Device Database

11. A window opens up asking if you want to insert the default STM32 startup file to your project. Click on “Yes”. This will save you a lot of time.
12. In the µVision Project Workspace area shown below in Figure 10, open up the folders as shown below on the left by clicking on the “+” beside each folder.
13. We have now created a project called Mytest and the target hardware called Target 1.

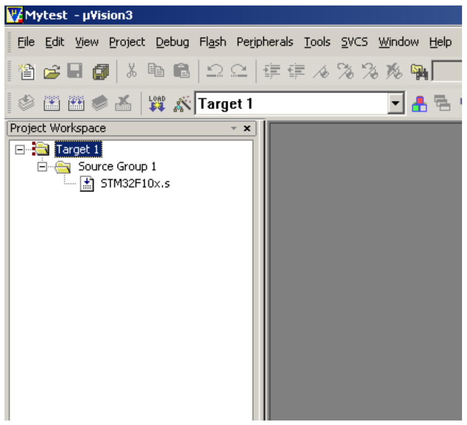


Figure 10 Project Workspace

14. Click once (carefully) on the name “Target 1” (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose MCBSTM32 as shown below in Figure 11. Click once on a blank part of the Project Workspace to accept this. Note the Target selector also changes. You can create many target hardware configurations including a simulator and easily select them for debugging purposes.

Select the source files:

1. Using MS Explore (right click on Windows Start icon), copy all the files from C:\Keil\ARM\Boards\Keil\MCBSTM32\PWM_2 to the ST folder except the file STM32F10x.s.
2. In the Project Workspace in the upper left hand of µVision, right-click on “Source Group 1” and select Add files to Group “Source Group 1”.
3. Select the files Pwm.c and STM32_Init.c and click on Add and then Close. These will show up in the Project Workspace as shown below. At this point you could build this project but let us finish setting µVision up first.

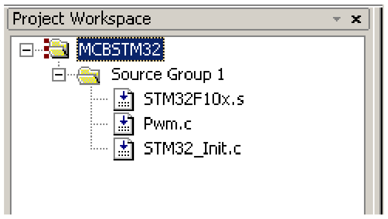


Figure 11 New Project Workspace for Target hardware MCBSTM32

Configure the ULINK2:

1) Select Project/Options for Target Cypress 1 or click on the Options icon.
2) Select the Debug tab.
3) Select the box labeled Use: beside the box Ulink Cortex Debugger. ***Very important step !***
4) Verify the SWV is configured as described in Appendix A.
5) Complete Steps 2 through 6 in Appendix A to configure the ULINK2.
6) You can skip 4 as trace isn’t needed for this example. But it won’t hurt of you complete it.
7) Steps 7 through 10 are for the SWV Trace and are optional for this example. They are not needed.
8) Do Steps 11 through 15. This completely configures the ULINK2 in µVision.
9) Now, you can continue with the PWM_2 example on the next page.
10) Click on File/Save All to save your project.

2a) PWM_2: Pulse Width Modulator (TIM4) Example

LEDs PB8 and PB9 are connected to the output of TIM4 Channel 3 and Channel 4 respectively. The PWM output makes the LEDs smoothly and slowly change brightness alternatively as the duty cycles are slowly changed..

1. The project is already open. If not, open Pwm.Uv2 from the directory you created: ST. ***Important step.***
2. Double-click on the file Pwm.c in the Project Workspace to make it visible.
3. Select MCBSTM32 in the Target box. Do not use Simulator.
4. Rebuild the project and load this into the processor flash.

Note: if you get a JTAG Communication Error at this point, open up the Options for Target box and select Debug and Settings. There is a possibility the SW settings have changed back to JTAG. Change these as shown in the circle on Figure 36 in Appendix A.

5. Put µVision into debug mode.
6. Click on RUN. Note the two LEDS slowly change brightness.

Variable Ccr determines the relative brightness of LED PB8 and 100-Ccr for PB9. Variable Dir determines if a LED is increasing or decreasing.

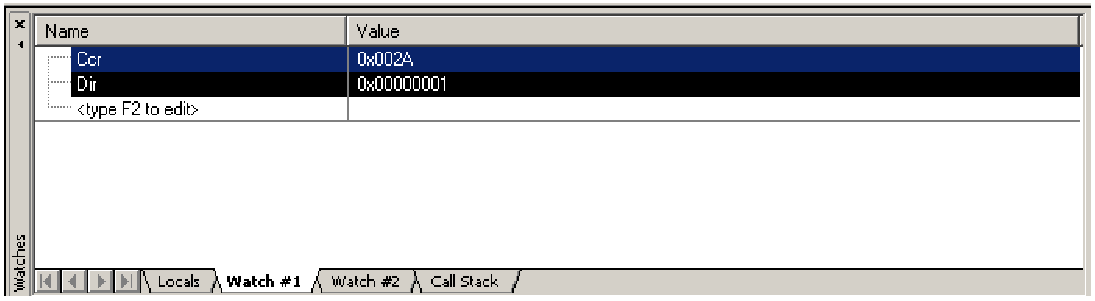
Watch Window:

1. In the Watch window (either #1 or #2) click on "Type F2 to edit" and press the key F2.
2. Enter one entry for Ccr and repeat for Dir. Note you do not have to stop the program to enter these values.

Neat Feature: You can block a variable in a source file, then click on it and drag it to a Memory or Watch window. You can also do this from the Symbol Viewer available while in debug mode from View/Symbol Window or its icon.

The Watch window will look like Figure 12: Note how the values change as the program runs.

How it works: This information is acquired from the CPU through the Serial Wire Viewer therefore is non-intrusive and in real-time. No code in the user space is used.
Other processors must use various intrusive methods to obtain this type of information without stopping which by definition is not real-time. Some projects such as disk drives, motor control, engine and transmission controllers and those that depend on time-outs with other systems often cannot be stopped.



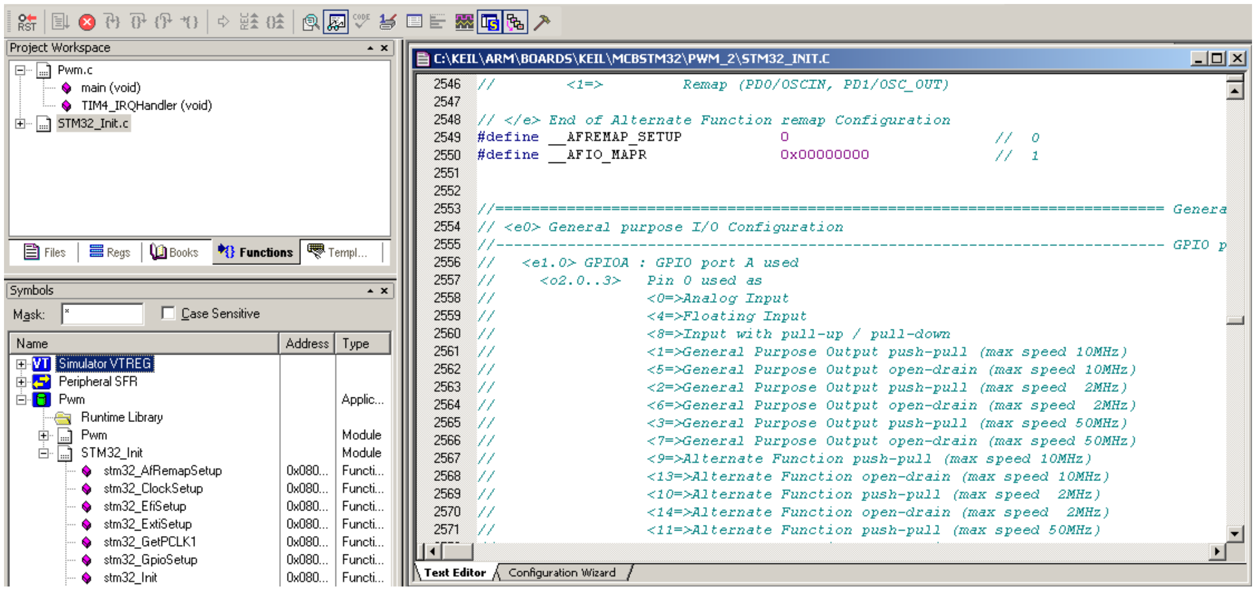
Name	Value
Ccr	0x002A
Dir	0x00000001
<type F2 to edit>	

Figure 12 Real Time Watch Window for variables Ccr and Dir

Configuration Wizard:

Neat Feature: µVision provides a method to easily edit initialization files to set program values.

1) Stop program execution and take µVision out of debug mode.
2) Open the file STM32_Init.c.
3) You can select this from the Project Workspace. Select either the Functions or Files tab at the bottom of the Project Workspace as shown in Figure 13, then double click on the filename to open it up.
4) You might have to move the right edge of this window to the right in order to see all the columns.
5) In the STM32_Init.c window shown in Figure 13, note the commented text (//) with tags surrounded with < >. This is an open scripting language that you can create yourself for any of your own source files.

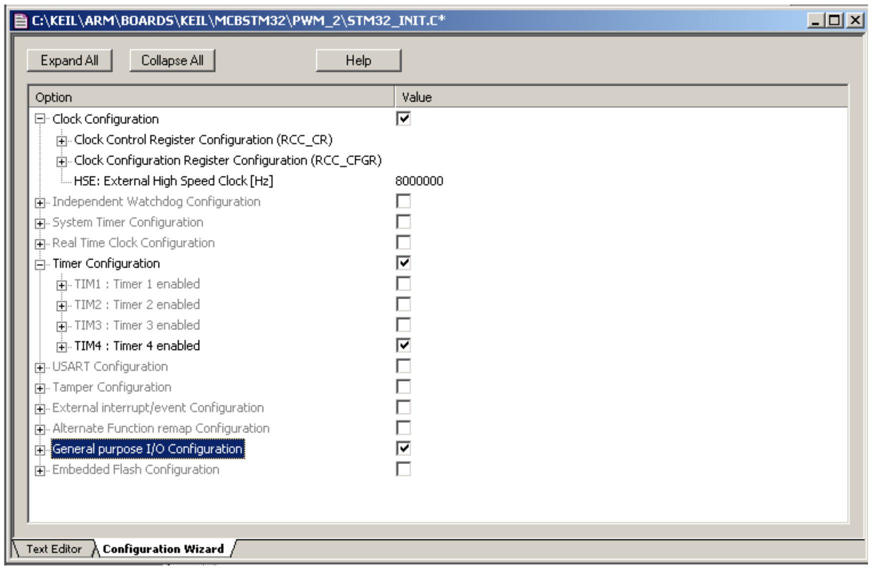


Name	Address	Type
Simulator VTREG		
Peripheral SFR		
Pwm		Applic...
Runtime Library		
Pwm		Module
STM32_Init		Module
stm32_AfRemapSetup	0x080...	Functi...
stm32_ClockSetup	0x080...	Functi...
stm32_EfiSetup	0x080...	Functi...
stm32_ExtiSetup	0x080...	Functi...
stm32_GetPCLK1	0x080...	Functi...
stm32_GpioSetup	0x080...	Functi...
stm32_Init	0x080...	Functi...

```
//          <1=>          Remap (PD0/OSCIN, PD1/OSC_OUT)

// </e> End of Alternate Function remap Configuration
#define __AFREMAP_SETUP            0                        //  0
#define __AFIO_MAPR                0x00000000               //  1


//============================================================================ Genera
// <e0> General purpose I/O Configuration
//---------------------------------------------------------------------------- GPIO p
//   <e1.0> GPIOA : GPIO port A used
//     <o2.0..3>   Pin 0 used as
//                 <0=>Analog Input
//                 <4=>Floating Input
//                 <8=>Input with pull-up / pull-down
//                 <1=>General Purpose Output push-pull (max speed 10MHz)
//                 <5=>General Purpose Output open-drain (max speed 10MHz)
//                 <2=>General Purpose Output push-pull (max speed  2MHz)
//                 <6=>General Purpose Output open-drain (max speed  2MHz)
//                 <3=>General Purpose Output push-pull (max speed 50MHz)
//                 <7=>General Purpose Output open-drain (max speed 50MHz)
//                 <9=>Alternate Function push-pull (max speed 10MHz)
//                 <13=>Alternate Function open-drain (max speed 10MHz)
//                 <10=>Alternate Function push-pull (max speed  2MHz)
//                 <14=>Alternate Function open-drain (max speed  2MHz)
//                 <11=>Alternate Function push-pull (max speed 50MHz)
```

Figure 13 Text Editor tab for a source file

6) Click on the Configuration Wizard tab at the bottom of the window STM32_INIT.c in Figure 13. Figure 14 opens up with check boxes. This is the same text but arranged differently. Changing the text in the Text Editor tab or the settings in the Configuration Wizard tab changes the settings in the other tab and vice versa.



Option	Value
Clock Configuration	✓
Clock Control Register Configuration (RCC_CR)	
Clock Configuration Register Configuration (RCC_CFGR)	
HSE: External High Speed Clock [Hz]	8000000
Independent Watchdog Configuration	☐
System Timer Configuration	☐
Real Time Clock Configuration	☐
Timer Configuration	✓
TIM1 : Timer 1 enabled	☐
TIM2 : Timer 2 enabled	☐
TIM3 : Timer 3 enabled	☐
TIM4 : Timer 4 enabled	✓
USART Configuration	☐
Tamper Configuration	☐
External interrupt/event Configuration	☐
Alternate Function remap Configuration	☐
General purpose I/O Configuration	✓
Embedded Flash Configuration	☐

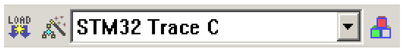
Figure 14 Configuration Wizard tab for the same source file

3) STLIB_Blinky Example

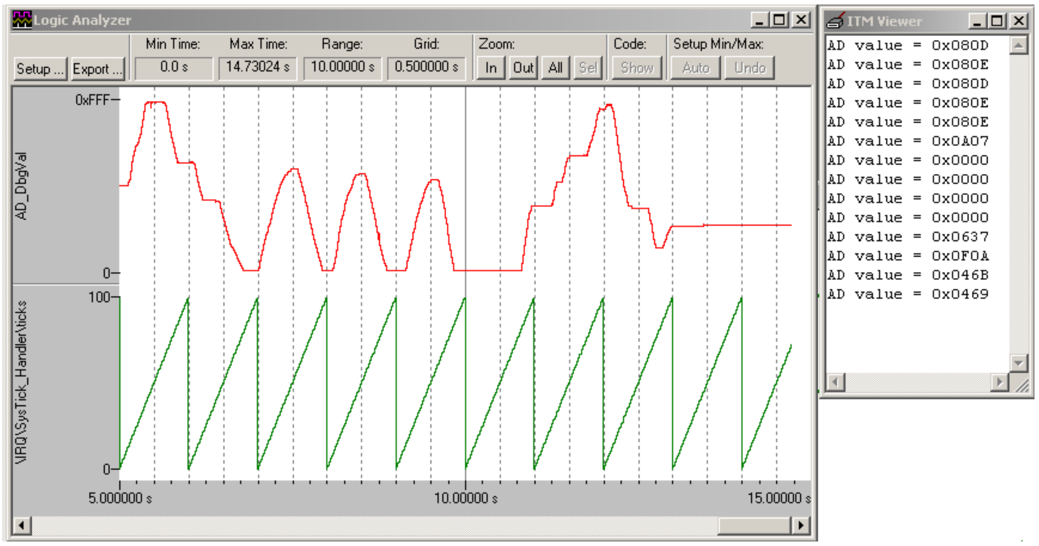
Serial Wire Viewer - Trace

The Serial Wire Viewer is a Cortex-M3 feature. It is a single bit trace feature offering flexible debugging power exceeded only by the ETM (Embedded Trace Macrocell). Both are part of the CoreSight debugging technology from ARM. ETM is an option and is not available on all ARM controllers. Consult the applicable processor data sheet.

1. Open the project Blinky.UV2 in MCBSTM32\STLIB_Blinky.
2. Select STM32 Trace C in the target window as shown here:
3. configure the SWV as described in Appendix A.
4. Rebuild (compile) the project.
5. Load the FLASH and put µVision into the debugging mode.
6. If necessary, open the Logic Analyzer window in View/Logic Analyzer Window.
7. Open the ITM Viewer in View/Serial Window if not already present.
8. Run the program and rotate the pot on the board.
9. If it doesn't work as described and shown in Figure 15: double check the SWV settings as in Appendix A.



How it works: The Logic Analyzer window shows the global variable AD_DbgVal representing the position of the pot (in red) and the Cortex-M3 SYSTICK timer (in green). This is displayed in real-time through the Serial Wire Viewer.



```
AD value = 0x080D
AD value = 0x080E
AD value = 0x080D
AD value = 0x080E
AD value = 0x080E
AD value = 0x0A07
AD value = 0x0000
AD value = 0x0000
AD value = 0x0000
AD value = 0x0000
AD value = 0x0637
AD value = 0x0F0A
AD value = 0x046B
AD value = 0x0469
```

Figure 15 Logic Analyzer and ITM Viewer displaying the variable AD_DbgVal

ITM Viewer

Neat Feature: The ITM Viewer window shown on the right in Figure 15 is written to by customer code and uses the ITM channel Port 0. This example code shown below is in the file Serial.c. In this example, it writes the ASCII value of the pot position to the ITM Viewer window. This is called serial debug or printf debugging. The value you write to this port appears in the ITM Viewer window. ITM is an acronym for Instrumentation Trace Macrocell.
Writing the value ch will take 2 CPU cycles. No cycles are used to transfer the data through the ITM out to µVision.

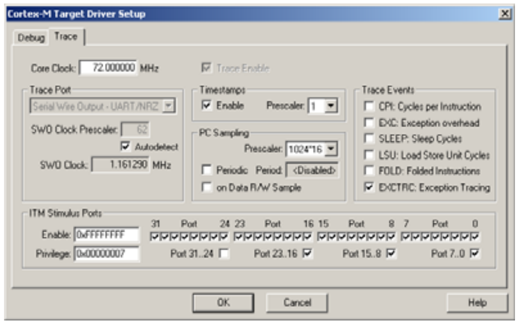
```
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))

if (DEMCR & TRCENA) {
        while (ITM_Port32(0) == 0);
        ITM_Port8(0) = ch;
                        }
```

10. Please STOP the program execution and take µVision out of debugger mode for the next exercise.

4) Tamper Example

Pressing the tamper button, TAMP on the evaluation board, causes an external interrupt 2 with the STM32 processor. This is also known as Int 18 in ARM documents. Tamper is a processor pin to prevent unauthorized access.

1. Open the Tamper project tamper.Uv2 and build it and load it into the processor Flash memory as before.
2. Configure the Serial Wire Viewer as described in the Appendix A. Make sure the ULINK and not simulator is selected. See the Figure 35 in Appendix A.
3. Enter the debug mode.
4. Open Peripherals/Target Settings and select the Trace tab and set this window as shown in Figure 16. Ignore the ITM settings. ***The EXCTRC must be set.*** Click on OK.
5. Open Peripherals/Trace and select Records. Repeat and select Exceptions and position the windows.


Cortex-M Target Driver Setup

Debug | Trace

Core Clock: 72.000000 MHz [✓ Trace Enable]

Trace Port: Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 62 [✓ Autodetect]
SWO Clock: 1.161290 MHz

Timestamps: [✓ Enable] Prescaler: 1

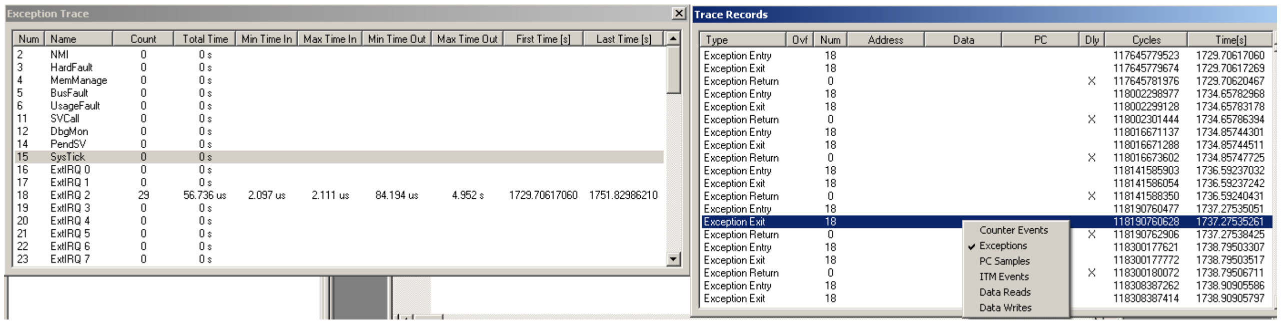
PC Sampling: Prescaler: 1024*16
[ ] Periodic Period: <Disabled>
[ ] on Data R/W Sample

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

ITM Stimulus Ports:
Enable: 0xFFFFFFFF
Privilege: 0x00000007
Port 31..24 [ ] Port 23..16 [✓] Port 15..8 [✓] Port 7..0 [✓]

Figure 16 Trace configuration for Tamper

6. Right click on the Trace Records window and set only the exceptions to be active as shown below in the small box.
7. Click on RUN and press the TAMP button on the board. Note the Exceptions displayed as shown in Figure 17.


Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	0	0 s						
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	0	0 s						
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	29	56.736 us	2.097 us	2.111 us	84.194 us	4.952 s	1729.70617060	1751.82986210
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						

Trace Records

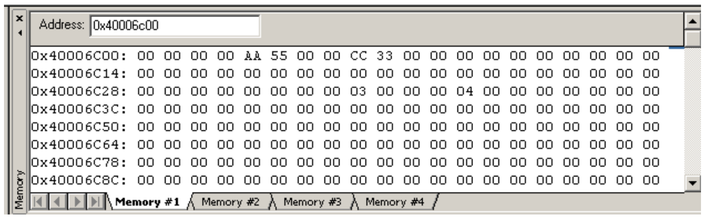
Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		18					117645779523	1729.70617060
Exception Exit		18					117645779674	1729.70617269
Exception Return		0				X	117645781976	1729.70620467
Exception Entry		18					118002298977	1734.65782968
Exception Exit		18					118002299128	1734.65783178
Exception Return		0				X	118002301444	1734.65786394
Exception Entry		18					118016671137	1734.85744301
Exception Exit		18					118016671288	1734.85744511
Exception Return		0				X	118016673602	1734.85747725
Exception Entry		18					118141585903	1736.59237032
Exception Exit		18					118141586054	1736.59237242
Exception Return		0				X	118141588350	1736.59240431
Exception Entry		18					118190760477	1737.27535051
Exception Exit		18					118190760628	1737.27535261
Exception Return		0				X	118190762906	1737.27538425
Exception Entry		18					118300177621	1738.79503307
Exception Exit		18					118300177772	1738.79503517
Exception Return		0				X	118300180072	1738.79506711
Exception Entry		18					118308387262	1738.90905586
Exception Exit		18					118308387414	1738.90905797

Figure 17 Exceptions displayed in real-time with the Serial Wire Viewer.

BKP - Backup Memory.

This memory will be cleared when the TAMPER is activated and is easily demonstrated with µVision.

1. Open a memory window (View/Memory Window) and set the address to 0x40006C00 as shown below.
2. Stop the program (unless it already is).
3. Click on RESET on the menu and then RUN.
4. Note several memory locations have data set up: AA 55 00 00 CC 33 as shown in Figure 18.
5. Press the TAMP button and watch these get cleared. This effect is part of the Tamper protection scheme.


Address: 0x40006c00

```
0x40006C00: 00 00 00 00 AA 55 00 00 CC 33 00 00 00 00 00 00 00 00 00 00
0x40006C14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40006C28: 00 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00 00 00 00 00
0x40006C3C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40006C50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40006C64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40006C78: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40006C8C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 18 Backup memory cleared by the Tamper feature.

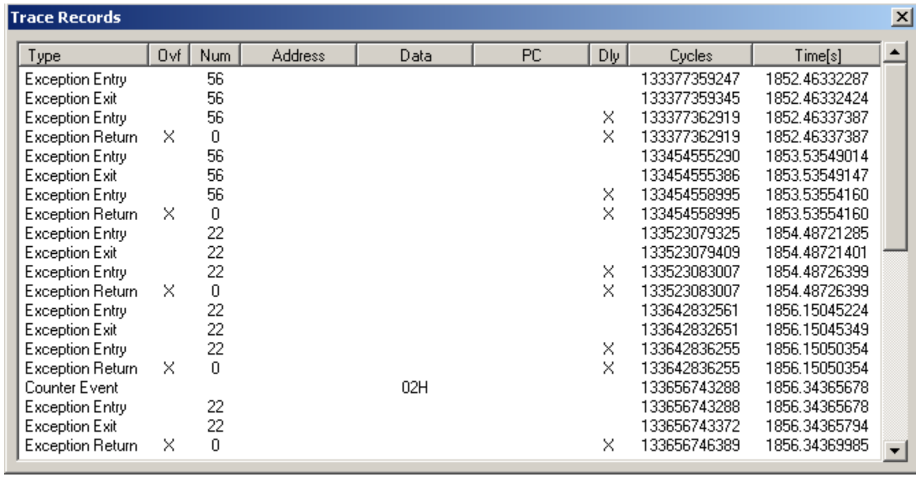
5) EXTI: External Interrupt Example

External interrupts are created by peripherals or external input pins. In this example, the WKUP and TAMP buttons activate two interrupt pins, PA0 (EXTI0) and PC13 (EXTI13).
Software interrupts are created with SVC instruction (System Service Call) with Cortex processors rather than a SWI in other ARM processors. Software interrupts are not discussed in this document.

1. STOP the program execution and take µVision out of debugger mode from the last exercise.
2. Open the EXTI project Exti.Uv2 and compile and load it as before.
3. Configure the Serial Wire Viewer as described in Appendix A.
4. Enter the debug mode.
5. Open Peripherals/Target Settings and select the Trace tab and set this as shown above in Figure 16 for the tamper example. Ignore the ITM settings. The EXCTRC must be set.
6. Click on OK.
7. Open Peripherals/Trace and select Records.
8. Repeat and select Exceptions. Position the windows as appropriate.
9. Click RUN to start program execution.

Pressing either the TAMP or WKUP buttons on the board creates an interrupt that controls the LED PB8.
Note the exceptions 22 and 56 entry and exit events are listed in the two trace windows.

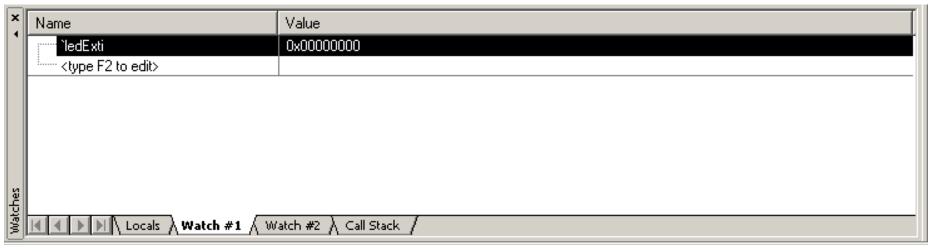
Neat Feature: The "X" in the overflow column indicates a message was lost. The ETM (Embedded Trace Macrocell) will capture all frames and is available on some Cortex-M3 processors. The "X" in the DLY (delay) column indicates the timestamp is not correct due to some delay in the SWV output. This is shown below in Figure 19.


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		56					133377359247	1852.46332287
Exception Exit		56					133377359345	1852.46332424
Exception Entry		56				X	133377362919	1852.46337387
Exception Return	X	0				X	133377362919	1852.46337387
Exception Entry		56					133454555290	1853.53549014
Exception Exit		56					133454555386	1853.53549147
Exception Entry		56				X	133454558995	1853.53554160
Exception Return	X	0				X	133454558995	1853.53554160
Exception Entry		22					133523079325	1854.48721285
Exception Exit		22					133523079409	1854.48721401
Exception Entry		22				X	133523083007	1854.48726399
Exception Return	X	0				X	133523083007	1854.48726399
Exception Entry		22					133642832561	1856.15045224
Exception Exit		22					133642832651	1856.15045349
Exception Entry		22				X	133642836255	1856.15050354
Exception Return	X	0				X	133642836255	1856.15050354
Counter Event				02H			133656743288	1856.34365678
Exception Entry		22					133656743288	1856.34365678
Exception Exit		22					133656743372	1856.34365794
Exception Return	X	0				X	133656746389	1856.34369985

Figure 19 Exceptions in Trace Records window

1. Open a Watch window. The program need not be stopped to do these steps. It can be done on-the-fly.
2. In the Watch window (either #1 or #2) click on "Type F2 to edit" and press the key F2.
3. **Enter the global variable name ledExti.**
4. Note that when you now press TAMP or WKUP this variable changes state.



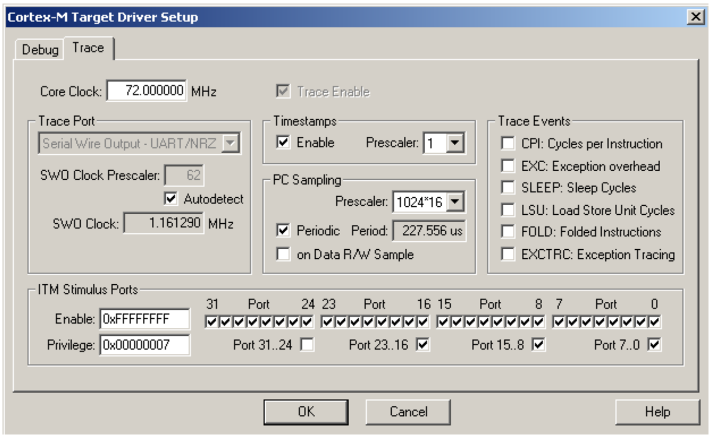
Name	Value
ledExti	0x00000000
<type F2 to edit>	

Figure 20 Variable ledExti displayed in real-time.

PC Sample Example

Neat Feature: The PC Sample is an output of the SWV and is useful for Performance or Profile Analysis (where does your program spend most of its time) and determining the PC (program counter) when your program has "gone into the weeds" and is caught in an infinite loop. The SWV (and ETM) are very good tools for these purposes.

1) Click on STOP to halt program execution. Leave µVision in Debug mode.
2) Open Peripherals/Target Settings and select the Trace tab to open the Cortex Target Driver Setup.
3) Configure it as shown below in Figure 21. Make sure the PC Sampling Periodic is checked. This turns on the PC Sampling feature. Ignore the ITM fields.
4) Click on OK. Click Yes when asked to take new values. This stops the program execution.



Cortex-M Target Driver Setup

Debug | Trace

Core Clock: 72.000000 MHz — [✓] Trace Enable

Trace Port: Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 62 — [✓] Autodetect
SWO Clock: 1.161290 MHz

Timestamps: [✓] Enable — Prescaler: 1

PC Sampling: Prescaler: 1024*16
[✓] Periodic — Period: 227.556 us
[ ] on Data R/W Sample

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

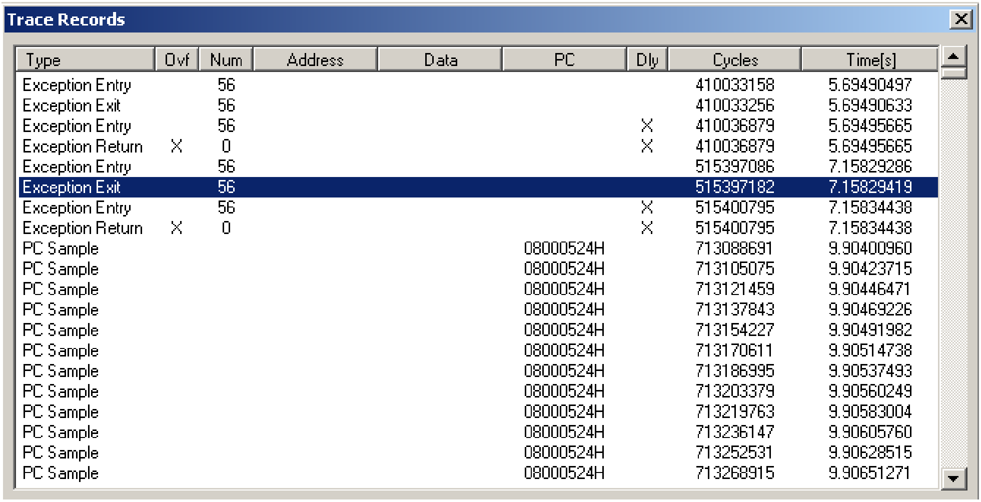
ITM Stimulus Ports:
Enable: 0xFFFFFFFF — Ports 31..0 all [✓]
Privilege: 0x00000007 — Port 31..24 [ ] Port 23..16 [✓] Port 15..8 [✓] Port 7..0 [✓]

Figure 21 Trace Configuration to Sample the PC

5) Make sure the Trace Records window is open. Select Peripherals/Trace/Records if it is not.
6) Right click in the Trace Records window and check PC Samples. Click anywhere to close.
7) Double click in the Trace Records window to clear it is necessary.
8) Click on RUN. The Trace Records window will immediately fill up with PC values as shown in Figure 21.

How it works: Clearly the PC is in a tight loop on itself at address 08000524H. This example is easy to find without a trace but if the loop was bigger and more complicated the track of the PC will be captured allowing detailed analysis. If the program is stopped due to a breakpoint, the instruction(s) that caused the problem will also be captured. Complicated "off into the weeds" problems are nearly impossible to find without a trace.

9) Stop program execution and open the Disassembly window in the View menu item. Note the program is indeed stopped at 0x080000524 and is a Branch instruction (E7FE) to itself.



Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		56					410033158	5.69490497
Exception Exit		56					410033256	5.69490633
Exception Entry		56				X	410036879	5.69495665
Exception Return	X	0				X	410036879	5.69495665
Exception Entry		56					515397086	7.15829286
Exception Exit		56					515397182	7.15829419
Exception Entry		56				X	515400795	7.15834438
Exception Return	X	0				X	515400795	7.15834438
PC Sample					08000524H		713088691	9.90400960
PC Sample					08000524H		713105075	9.90423715
PC Sample					08000524H		713121459	9.90446471
PC Sample					08000524H		713137843	9.90469226
PC Sample					08000524H		713154227	9.90491982
PC Sample					08000524H		713170611	9.90514738
PC Sample					08000524H		713186995	9.90537493
PC Sample					08000524H		713203379	9.90560249
PC Sample					08000524H		713219763	9.90583004
PC Sample					08000524H		713236147	9.90605760
PC Sample					08000524H		713252531	9.90628515
PC Sample					08000524H		713268915	9.90651271

Figure 22 Trace Records showing PC Sampling

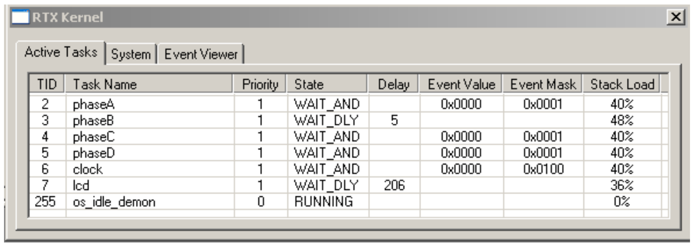
6) STLIB_RTX_Blinky Example – with the RTX RTOS

This example uses the Keil MDK RTX kernel to provide outputs to drive a stepper motor. Four LEDs blink in sequence. PB8 through PB11. This example simulates Half step driver mode and with CW rotation direction.

1. Open the STLIB_RTX_Blinky project Blinky.Uv2 and compile and load it as in previous examples.
2. Configure the Serial Wire Viewer as described in Appendix A.
3. Put µVision into debugger mode by clicking on the debug icon.
4. Select View/Periodic Window Update.
5. Open Peripherals/Target Settings and select the Trace tab and check EXCTRC and Periodic and on Data R/W Sample not checked. Ignore the ITM settings. Click on OK.
6. Open Peripherals/Trace and select Records. Repeat and select Exceptions. Position the windows.
7. Select Peripherals/RTX Kernel. A new window will open up as shown in Figure 23. Click on RUN.

The LEDS will blink simulating the motor driver. Click on STOP and the RTX window will update as shown (expect a delay) below depending on where the program happened to stop and also which task is active.

New! It is easy to make a RTX Event Viewer update in real-time using the Serial Wire Viewer. See Keil Application Note 197 www.keil.com/appnotes/docs/apnt_197.asp. Look for RTX Kernel Event Viewer. Needs MDK Version 3.20b.



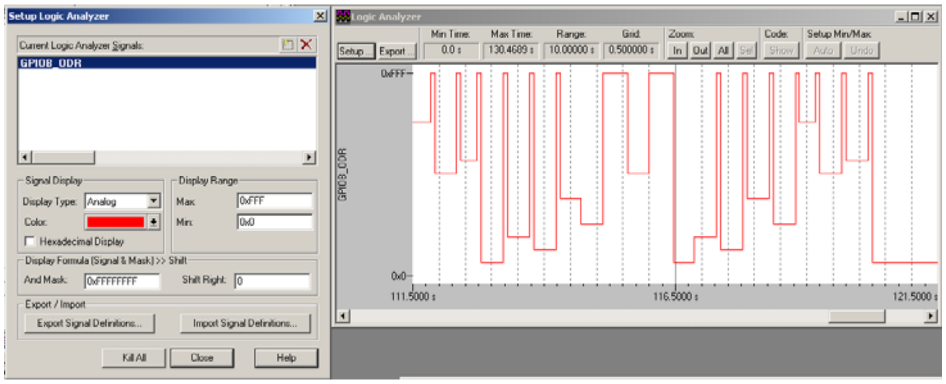
TID	Task Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
2	phaseA	1	WAIT_AND		0x0000	0x0001	40%
3	phaseB	1	WAIT_DLY	5			48%
4	phaseC	1	WAIT_AND		0x0000	0x0001	40%
5	phaseD	1	WAIT_AND		0x0000	0x0001	40%
6	clock	1	WAIT_AND		0x0000	0x0100	40%
7	lcd	1	WAIT_DLY	206			36%
255	os_idle_demon	0	RUNNING				0%

Figure 23 RTX Kernel Awareness

Logic Analyzer example with Serial Wire Viewer

We can display certain elements in real-time with the Serial Wire Viewer and display them graphically.

1. Stop program execution and leave µVision in debugger mode. You can close the Trace Records window.
2. Open the Logic Analyzer window in View/Logic Analyzer Window.
3. Click on Setup… the Setup Logic Analyzer window on the left side of Figure 24 opens up.
4. Click in the area under Current Logic Analyzer Signals and press the PC **Insert** key.
5. Enter GPIOB_ODR as shown below and press Enter key. Set MAX to 0xFFF and MIN to 0x0.
6. Click on CLOSE. Use IN and OUT to set the range to 10 seconds. Click on RUN.
7. Put the scroll bar to the right side if needed.
8. This displays the IO port LED output sequence. In real-time. Click on STOP when you are done.



GPIOB_ODR

Display Type: Analog
Color:
☐ Hexadecimal Display
Max: 0xFFF
Min: 0x0
And Mask: 0xFFFFFFFF
Shift Right: 0

Min Time: 0.0 s
Max Time: 130.4689 s
Range: 10.00000 s
Grid: 0.500000 s

Figure 24 Setup Logic Analyzer and data display window

7) CAN Example: Controller Area Network

How it works: This example uses one MCBSTM32 board. The STM32F103RB processor contains one CAN controller. Since a CAN controller is unable by definition to communicate with itself (i.e. receive its own transmitted messages), this example puts the controller into two special diagnostics modes called Loopback and Silent Modes. When these modes are active, the CAN controller is unable to communicate with any other CAN nodes. Its transmitted messages are treated as if they were transmitted by another CAN node. It is also isolated from the bus for both receive and transmit.

ST documents reference this as Test Mode and is configured in lines 107 and 108 in file CanDemo.c and is copied here. In our example here, we will use Test Mode activated with our one CAN node (the default). In 8), we will use Normal Mode.

```
 /* COMMENT THE LINE BELOW TO ENABLE DEVICE TO PARTICIPATE IN CAN NETWORK   */
CAN_testmode(CAN_BTR_SILM | CAN_BTR_LBKM);   // Loopback, Silent Mode (self-test)
```

This example creates a CAN message using data from the pot and displays both the transmit and receive values on the LCD. The CAN controller is in the above mentioned Test Mode (both Loopback and Silent modes).

1. Make sure µVision is not in debugger mode.
2. Open the project Can.Uv2 in the directory c:\Keil\Arm\Boards\Keil\MCBSTM32\Can.
3. On the toolbar select MCBSTM32 as the target (not Simulator). MCBSTM32
4. Turn on the Serial Wire Viewer as described in Appendix A.
5. Click on File/Save to save your project or press Ctrl-S.
6. Rebuild and load the project into the processor Flash memory as described in previous examples.
7. Put µVision into debugger mode by clicking on the debug icon.
8. Verify View/Periodic Window Update is checked and enabled. It might already be selected.
9. Click on RUN to execute the program.

As you turn the pot on the MCBSTM32 eval board the values transmitted change as shown in Figure 25:

The LCD displays the speed of the CAN transmission (500 Kbps) and the value of the first data byte (78 hex here) from both the receive and transmit buffers. It can do this because the CAN controller is in the special Test Mode.

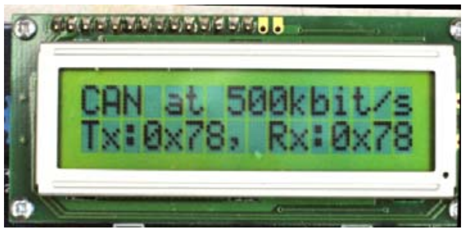


Figure 25 LCD showing CAN frames

How it works: The CAN ID is not shown here but is an 11 bit with the value of 0x21. You can see this value in the STID box in Figures 26 and 27. The DLC (data length control) is set to "1" indicating one data byte. The DLC is displayed in boxes labeled DLC in Figures 26 and 27.

It is possible to have up to 8 data bytes in CAN and either an 11 or 29 bit ID. All of these plus the data bytes are completely user software controlled. Review the source code in the Keil CAN example and you can see where these are configured.

CAN Speeds: 500 Kbps is the standard speed for the main CAN bus in passenger cars and light trucks (ISO 15765). Heavy duty vehicles (J1939) is set at 250 Kbps. Maximum CAN speed is 1 Mbps and the minimum is 10 Kbps. Any speed can be used in a custom design. All nodes must run at the same speed…always from controllers' initialization. CAN Speed is not dynamically changeable.

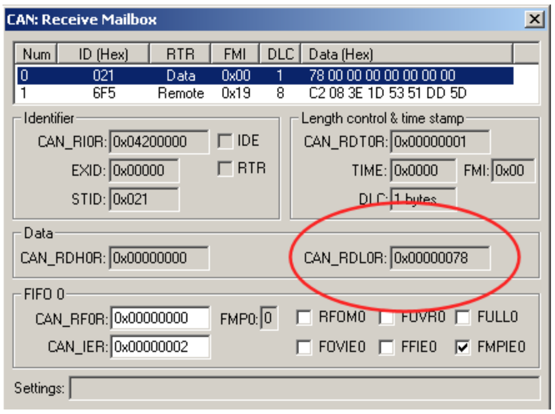
10. Click on the Stop icon to continue.

CAN Information Windows

There should be four windows open that display information about the CAN controllers. We will look at two of them and they are pictured below in Figures 26 and 27. They are the CAN Receive and CAN Transmit Mailbox windows respectively. These windows are updated when program execution is stopped. Contact Keil to use the Serial Wire Viewer. The other two CAN windows are for the Filters and the CAN controller initialization values. Both are outside of the scope of this tutorial.

CAN_RDL0R (that is a zero in there) in Figure 26 is one of 2 CAN receive FIFOs. CAN_TDL0R in Figure 27 is one of 4 CAN transmit buffers called mailboxes (in ST docs). The value of 0x78 as shown in the LCD in Figure 25 on the previous page is displayed in the appropriate window below. You can run and then stop your program to see these values are the same.

How it works: The entire CAN frame is displayed in both of these windows – they are highlighted below. The CAN frame itself consists of the fields labeled ID (hex), RTR, DLC and Data(hex). Note the first data byte is a 78 which is reflected in the fields CAN_RDL0R and CAN_TDL0R. This is the only byte valid out of a potential 8 as DLC is set to "1 byte".


CAN: Receive Mailbox

Num	ID (Hex)	RTR	FMI	DLC	Data (Hex)
0	021	Data	0x00	1	78 00 00 00 00 00 00 00
1	6F5	Remote	0x19	8	C2 08 3E 1D 53 51 DD 5D

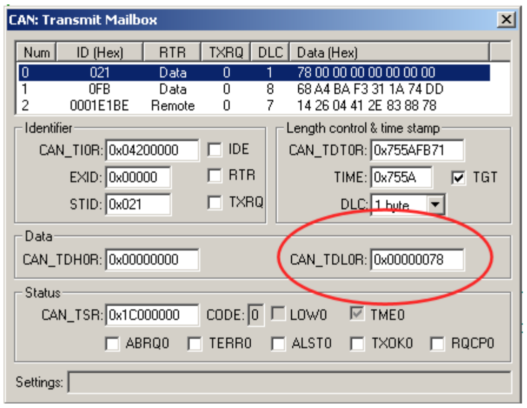
Identifier
CAN_RI0R: 0x04200000 ☐ IDE
EXID: 0x00000 ☐ RTR
STID: 0x021

Length control & time stamp
CAN_RDT0R: 0x00000001
TIME: 0x0000 FMI: 0x00
DLC: 1 bytes

Data
CAN_RDH0R: 0x00000000
CAN_RDL0R: 0x00000078

FIFO 0
CAN_RF0R: 0x00000000 FMP0: 0 ☐ RFOM0 ☐ FOVR0 ☐ FULL0
CAN_IER: 0x00000002 ☐ FOVIE0 ☐ FFIE0 ☑ FMPIE0

Settings:


CAN: Transmit Mailbox

Num	ID (Hex)	RTR	TXRQ	DLC	Data (Hex)
0	021	Data	0	1	78 00 00 00 00 00 00 00
1	0FB	Data	0	8	68 A4 BA F3 31 1A 74 DD
2	0001E1BE	Remote	0	7	14 26 04 41 2E 83 88 78

Identifier
CAN_TI0R: 0x04200000 ☐ IDE
EXID: 0x00000 ☐ RTR
STID: 0x021 ☐ TXRQ

Length control & time stamp
CAN_TDT0R: 0x755AFB71
TIME: 0x755A ☑ TGT
DLC: 1 byte

Data
CAN_TDH0R: 0x00000000
CAN_TDL0R: 0x00000078

Status
CAN_TSR: 0x1C000000 CODE: 0 ☐ LOW0 ☑ TME0
☐ ABRQ0 ☐ TERR0 ☐ ALST0 ☐ TXOK0 ☐ RQCP0

Settings:

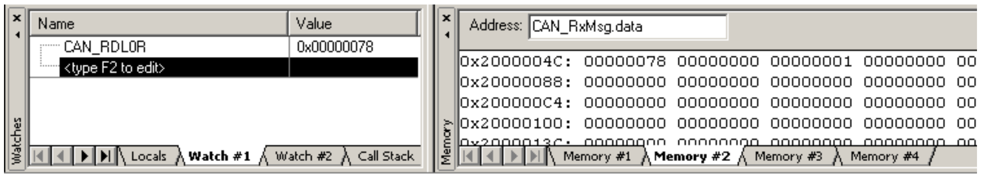
Figure 26 CAN Receive **Figure 27 CAN Transmit**

How it works: The data from these buffers is stored in the Keil example program in a structure defined in CAN.h with the synonym CAN_Msg and two instances called CAN_RxMsg and CAN_TxMsg. The data is stored in the structure member ***data*** which is an array of size 8 as shown here in its declaration:

```
typedef struct  {
unsigned int   id;                  // 11 bit identifier
unsigned char  data[8];             // Data field
unsigned char  len;                 // Length of data field in bytes
unsigned char  format;              // 0 - STANDARD, 1- EXTENDED IDENTIFIER
unsigned char  type;                // 0 - DATA FRAME, 1 - REMOTE FRAME
} CAN_msg;
```

A CAN message can have from 0 to 8 data bytes attached and is specified by the DLC field which is 4 bits. The number of data bytes can be dynamically changed from one frame to the next merely by changing the value of DLC.

Neat Feature: This data can be viewed in real-time with the Watch or Memory windows using the Serial Wire Viewer as shown in Figure 28 with the same data value of 0x78. Enter the shown variables as described in the Watch Window in Blinky. As the pot on the board is moved during program execution, these values will change in real-time with no CPU cycles used. You can also write to these memory locations with the Watch and Memory windows.



Name	Value
CAN_RDL0R	0x00000078
<type F2 to edit>	

Address: CAN_RxMsg.data

```
0x2000004C: 00000078 00000000 00000001 00000000 00
0x20000088: 00000000 00000000 00000000 00000000 00
0x200000C4: 00000000 00000000 00000000 00000000 00
0x20000100: 00000000 00000000 00000000 00000000 00
```

Figure 28 Watch and Memory windows

Neat Feature: You can also enter the structure instance name CAN_TxMsg or CAN_RxMsg as well as using CAN_RxMsg.data (or also the Tx structure) as shown above in the memory window.

CAN Example with Serial Wire Viewer:

We can easily view data in a graphical format in real-time using the Serial Wire Viewer. This does not steal any CPU cycles.

1. µVision must be in debug mode for the next steps.
2. Open the Logic Analyzer window. Click on View/Logic Analyzer Window to do this if necessary.
3. Click on the Setup box and enter these three variables: If necessary, see the **Logic Analyzer example** below Figure 23 for example instructions on setting this up. Steps 1 through 7 are the important ones.

CAN_RxRdy	(set the Display Range to 0x0 and 0x2)
CAN_RxMsg.data[0]	(set the Display Range to 0x0 and 0xFF)
CAN_TxMsg.data[0]	(set the Display Range to 0x0 and 0xFF)

4. Use the Zoom In and Out buttons to set Range to 10 seconds.
5. Open Peripherals/Target Settings and click on the Trace tab. Enable "on Data R/W sample". Click OK twice.
6. Click on Run to start program execution. You might have to bring the Logic Analyzer in focus to see it.
7. Note that as you turn the pot on the board the transmit trace CAN_TxMsg.data[0] varies as shown in Figure 29.
8. The CAN_RxRdy either doesn't change or hardly ever. This is the CAN peripheral bit indicating there is a new received valid CAN message in the buffer. This isn't being displayed properly because the Serial Wire Viewer is probably being overloaded by the setting in step 5 where the "on Data R/W sample" was set.
9. Go to Peripherals/Target Settings and select the Trace tab. Deselect "on Data R/W sample". Click OK.
10. Click on Run (program is automatically stopped when you change the Target Settings).

The CAN_RxRdy will display properly now. It is important to not push too much information down the SWV port (called SWO) as it is only one bit wide and is easily overloaded.

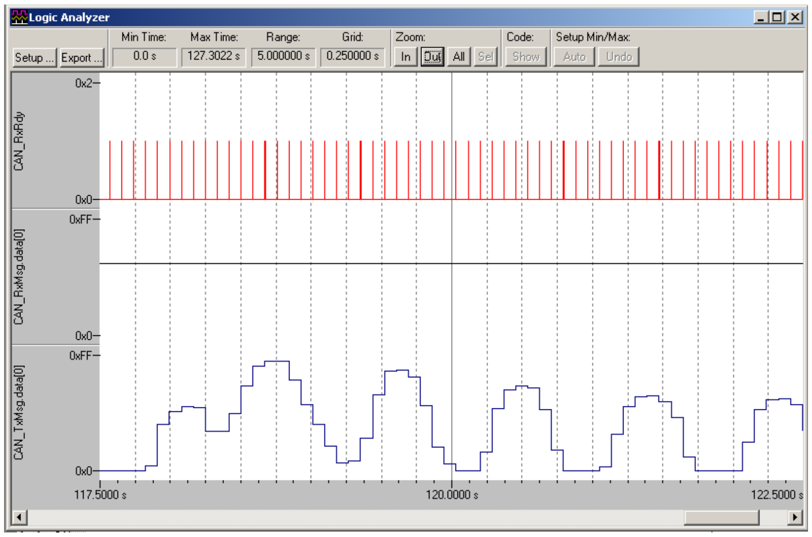

Min Time: 0.0 s
Max Time: 127.3022 s
Range: 5.000000 s
Grid: 0.250000 s

Figure 29

Important Notes about this CAN example:

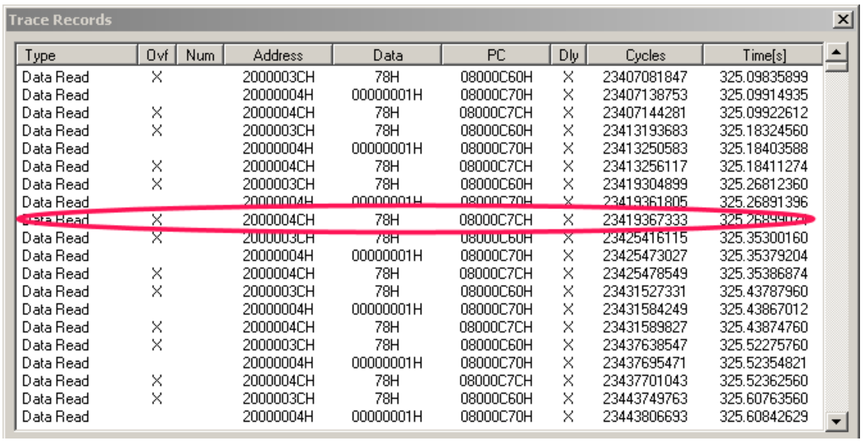
1. This example is sending only one CAN frame with one ID at a time. On a real network, there will probably be many more messages with different IDs and the example above takes the first data byte and displays it from every CAN message as it enters the receive buffer. Clearly the data displayed will become very erratic under such a situation. One way around this problem is to set the filters in the CAN controller to pass only the CAN ID(s) you want to display. This is beyond the scope of this document but the example provides the facilities to do this.
2. In order to see the graphic display CAN_RxMsg.data[0] you will need to repeat the example using two CAN boards and this is in the next example.
3. This example can be completely demonstrated with the µVision simulator. Change the target setting to Simulator and select Use Simulator in the Options for Target setting.
4. **Important Concept:** When you enter a variable in the Logic Analyzer, this action also sends it to the Trace Records window where it will be displayed as a data read or write as long as the appropriate filters are set.

CAN Trace Records and Interrupts

1. Stop the program execution and leave µVision in debugging mode.
2. Open Peripherals/Trace and open both the Records and Exceptions windows.
3. Open Peripherals/Target Settings and select the Trace tab.
4. Check the box labeled on data R/W Sample in the PC Sampling area of this window. Click on OK (twice).
   What it does: The instruction location causing the data reads and writes records will be added in the PC column in the Trace Records window.
5. Run the program by clicking on Run.
6. Recall in the Trace Records window you can filter out certain records by right-clicking on this window and selecting records for the small window that opens up. Do this and deselect Exceptions and Data Writes and any other records that are not Data Reads. Double click in the Trace Records window to clear it and it will fill up again.

How it works: A window similar to Figure 30 below will be displayed. Note the data read of 0x78 or 78H (in this example) from memory location 0x2000_004C, by the instruction at 0x8000_0C7C and is time-stamped both in CPU cycles and time.

Why it does this: The "x" in the Ovf column indicates there was some data overflow and some records were dropped. The "x" in the Dly column indicates the timestamp is delayed somewhat. This comes from the high amount of data reads performed by the Cortex-M3 processor. The Serial Wire Debug is having trouble recording such high speed traffic. It is difficult in some cases for the single bit port the Serial Wire Viewer uses (called the SWO) to track a ARM CPU at full speed. The information gained, as can be seen in Figure 29, it still very valuable. This can be alleviated by reducing the number of trace features employed in the Trace Configuration window in Figure 21 and increasing the speed of the SWV in Figure 36 in Appendix A.


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Read	X		2000003CH	78H	08000C60H	X	23407081847	325.09835899
Data Read			20000004H	00000001H	08000C70H	X	23407138753	325.09914935
Data Read	X		2000004CH	78H	08000C7CH	X	23407144281	325.09922612
Data Read	X		2000003CH	78H	08000C60H	X	23413193683	325.18324560
Data Read			20000004H	00000001H	08000C70H	X	23413250583	325.18403588
Data Read	X		2000004CH	78H	08000C7CH	X	23413256117	325.18411274
Data Read	X		2000003CH	78H	08000C60H	X	23419304899	325.26812360
Data Read			20000004H	00000001H	08000C70H	X	23419361805	325.26891396
Data Read	X		2000004CH	78H	08000C7CH	X	23419367333	325.2689[illegible]
Data Read	X		2000003CH	78H	08000C60H	X	23425416115	325.35300160
Data Read			20000004H	00000001H	08000C70H	X	23425473027	325.35379204
Data Read	X		2000004CH	78H	08000C7CH	X	23425478549	325.35386874
Data Read	X		2000003CH	78H	08000C60H	X	23431527331	325.43787960
Data Read			20000004H	00000001H	08000C70H	X	23431584249	325.43867012
Data Read	X		2000004CH	78H	08000C7CH	X	23431589827	325.43874760
Data Read	X		2000003CH	78H	08000C60H	X	23437638547	325.52275760
Data Read			20000004H	00000001H	08000C70H	X	23437695471	325.52354821
Data Read	X		2000004CH	78H	08000C7CH	X	23437701043	325.52362560
Data Read	X		2000003CH	78H	08000C60H	X	23443749763	325.60763560
Data Read			20000004H	00000001H	08000C70H	X	23443806693	325.60842629

Figure 30 Data READ Trace Records

1. **Neat Feature:** The "78" in Figure 29 is the output of the pot on the evaluation board.
2. Note what value you see on your particular pot setting.
3. Change the pot and ….
4. Double-click anywhere in the Trace records window to clear it.
5. Note that this data value will change corresponding to the pot setting position.
6. Repeat this sequence as many times as you want.
7. Stop program execution by clicking on Stop.

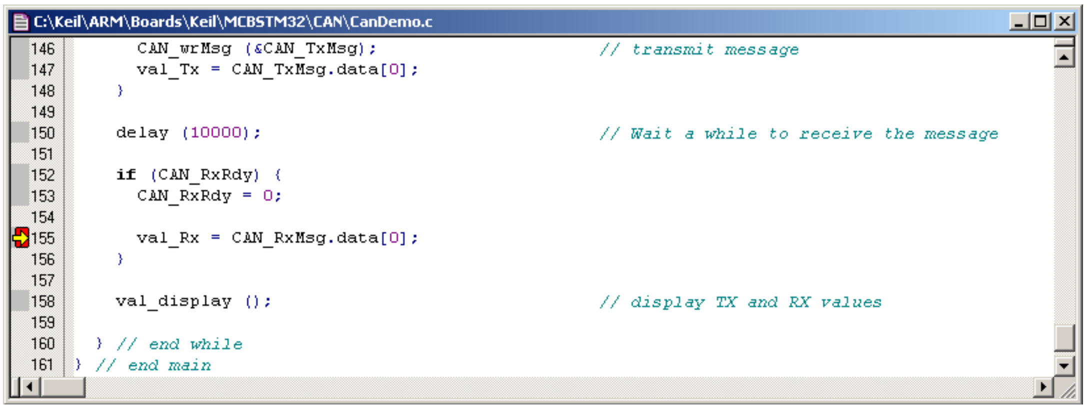
How it works: Every time you clear the trace a new set of trace records is entered. All the variables you have entered in the Logic Analyzer window will be displayed in the Trace Records window as either a Data Write or a Data Read.

Neat Feature: Right click anywhere in the Trace Records window and activate Exceptions. You will see the exceptions as they occurred.

CAN Programming Hint: It is common practice to always send CAN frames with the maximum data length of 8. Unused data bytes are often padded with 00 or FF. While the CAN controller automatically handles different numbers of data bytes, your software will be much simpler and less prone to have bugs if it always expects to see the full 8 bytes rather than 0 to 8.

1. Scroll in the CanDemo.c window to the C source code at line 155 as shown below in Figure 31. You might have to move the Trace windows away – they are always on top.
2. Double-click in the margin next to Line 155. A red block will appear signifying the breakpoint.
3. Double-click the Trace Records window to clear it.
4. Click on Run.

Each time you click on Run, the program will run to the breakpoint and various CPU cycles will show in the Trace Records window depending on how the filter box is configured. They will be time stamped if this is enabled in the Trace Configuration window.


C:\Keil\ARM\Boards\Keil\MCBSTM32\CAN\CanDemo.c

```
146       CAN_wrMsg (&CAN_TxMsg);                       // transmit message
147       val_Tx = CAN_TxMsg.data[0];
148     }
149
150     delay (10000);                                  // Wait a while to receive the message
151
152     if (CAN_RxRdy) {
153       CAN_RxRdy = 0;
154
155       val_Rx = CAN_RxMsg.data[0];
156     }
157
158     val_display ();                                 // display TX and RX values
159
160   } // end while
161 } // end main
```

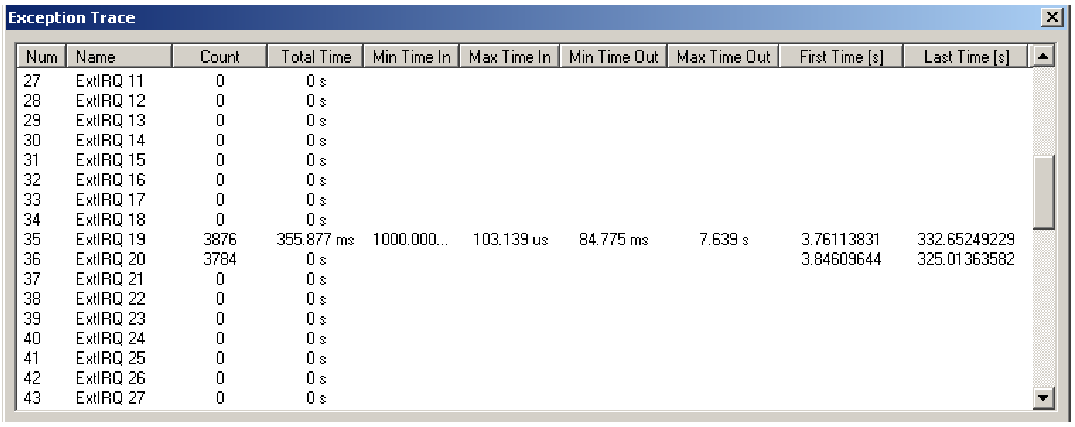
Figure 291 C Source to set breakpoint on.

Note that if you enable exceptions in the Trace Records window you can see a exception 35 but not the 36 as shown below in Figure 31. This is probably because the SWV is being overloaded.

1. Go to Peripherals/Target Settings and select the Trace tab.
2. Deselect the Timestamps, on Data R/W and Periodic. Select only the Exctrc Trace Events area.
3. Ignore the ITM settings.
4. Click on Run.
5. Note both Exceptions 35 and 36 will now be visible. They correspond to ST exceptions 19 and 20.
5. Remove the breakpoint by double-clicking on it again. You can also press Ctrl-B and clicking on Kill All.

Neat Feature: Note in the Exception Trace window as shown below in Figure 32 that two exceptions are counted: ExtIRQ 19 and 20. The number of occurrences and various times are recorded as shown. These values accumulate and can be cleared anytime by double-clicking anywhere inside the Exception Trace window.

The Cortex-M3 interrupts for the CAN transmit and receive buffers are displayed respectively.


Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
27	ExtIRQ 11	0	0 s						
28	ExtIRQ 12	0	0 s						
29	ExtIRQ 13	0	0 s						
30	ExtIRQ 14	0	0 s						
31	ExtIRQ 15	0	0 s						
32	ExtIRQ 16	0	0 s						
33	ExtIRQ 17	0	0 s						
34	ExtIRQ 18	0	0 s						
35	ExtIRQ 19	3876	355.877 ms	1000.000...	103.139 us	84.775 ms	7.639 s	3.76113831	332.65249229
36	ExtIRQ 20	3784	0 s					3.84609644	325.01363582
37	ExtIRQ 21	0	0 s						
38	ExtIRQ 22	0	0 s						
39	ExtIRQ 23	0	0 s						
40	ExtIRQ 24	0	0 s						
41	ExtIRQ 25	0	0 s						
42	ExtIRQ 26	0	0 s						
43	ExtIRQ 27	0	0 s						

Figure 30 Exception Trace: 19 and 20 shown. (Number 35 & 36)

8) CAN Example: using two MCBSTM32 boards.

In the preceding example, one CAN controller was used in Test Mode. This simulated a CAN node. It is possible to connect two boards together to form a real CAN node. You could also connect a CAN analyzer to this setup. Recall this source line:

```
 /* COMMENT THE LINE BELOW TO ENABLE DEVICE TO PARTICIPATE IN CAN NETWORK   */

CAN_testmode(CAN_BTR_SILM | CAN_BTR_LBKM);   // Loopback, Silent Mode (self-test)
```

How it works: Should the above line be commented out, the CAN controller will be put in Normal mode. In practice this means that in order for messages to be successfully transmitted or received, at least one other CAN node must be properly connected to the MCBSTM32 board in order to form a fully functioning CAN network. Another MCBSTM32 board will accomplish this and an example is now provided for a two node system using two MCBSTM32 boards.

1. Pins 2 and 7 on the DB9 connector labeled CAN are CAN_Lo and CAN_Hi respectively.
2. These two pins will be connected to the corresponding pins on the other board. A ground is not required.
3. You can do this experiment using only one ULINK2. The board will run the CAN program stand-alone.
4. A 120 ohm termination resistor is required at each end of a CAN network. This means a maximum of two resistors are required and each board has one installed (R4). If you add more than two boards, this resistor should be removed in each additional board. Do not add any resistors ! They are already included on the boards.
5. This example displays on the LCDs the pot value (Tx:) of one board to other board's Rx:.
6. To do this a trick is employed – since a given CAN controller can't see its own transmitted messages, the only Rx: messages displayed are from the opposite board.
7. Since both boards are transmitting the same ID, collisions either rarely happen or such collisions are handled by the CAN controllers' error mechanism and these messages are automatically resent in the background.
8. Having two nodes sending out messages with the same CAN ID is a rather poor engineering practice but is useful for illustrative purposes in this example.

Connecting the boards together:

1. Connect the two MCBSTM32 boards together as shown Figure 33. Pins 2 together and pins 7 together.
2. No ground is needed. The ground will add help in certain fault situations. You can add one.
3. Recall no external 120 ohm termination resistors are required. Each board already has one installed
4. CAN uses two wires plus an optional ground. These wires are called a differential twisted pair and offer great noise immunity useful in noisy environments.
5. You could connect a CAN analyzer to these two wires and view CAN frames and also inject them on the bus.

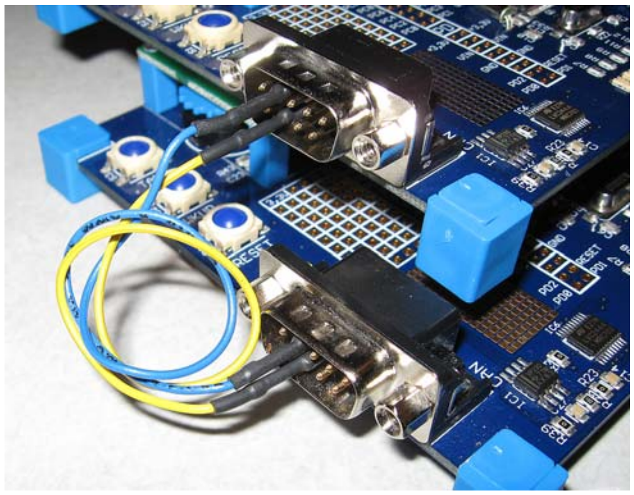


Figure 31 Connecting the two boards together

Programming the boards and running the CAN program:

Complete instructions start on the next page and for reference, this is the general format.

We will follow these steps using two MCBSTM32 boards, one ULINK2 and one PC as follows:

1. Connect one board up to the ULINK2 as usual.
2. Rebuild the project and load the program into the processor Flash memory.
3. Disconnect this board from the ULINK2. This board will run standalone so it still needs power from the USB cable.
4. Connect the second board to the ULINK2 and load the program to the processor Flash memory.
5. Run the example program on the second board and debug from it.
6. Now start the instructions on the next page.

Loading and Compiling (rebuilding) the Project:

1. Maker sure µVision is not in debugger mode.
2. Open the project CAN.Uv2 in the MCBSTM32\CAN directory.
3. If not already visible, open the file CanDemo.c by double clicking on its name in the Project Workspace.
4. Comment Line 106 to disable the Test Mode. You can use either // or the /* and */ pair. This line is `CAN_testmode(CAN_BTR_SILM | CAN_BTR_LBKM);`
5. Rebuild and load into the Flash memory of the processor.
6. Disconnect the board from the ULINK2, power it through another USB cable to allow the board to run stand-alone.
7. At this point turning the pot will change only the Tx: field on this board. If not, press the RESET button on the board. The Rx: must come from another board.
8. Connect the other board to the ULINK2 and load the Flash memory.
9. You might want to make sure that the SWV is configured as described in Appendix A.
10. Enter debug mode and click on Run.
11. The value of the pot in the Tx: field on one board will be displayed in the Rx: field on the other board. This is shown below in Figure 32.
12. If you configure (or have already configured) the Logic Analyzer as shown in **CAN Example with Serial Wire Viewer:** on page 19, both the transmit and receive waveforms will react to changes in the pot positions as shown in Figure 34 below.
    If you do not see the waveform, try enabling the Timestamps in Peripherals/Target Settings under the Trace tab.
13. This is because you now have a real CAN network setup and all variables used in the example programs are now available to the Serial Wire Viewer.

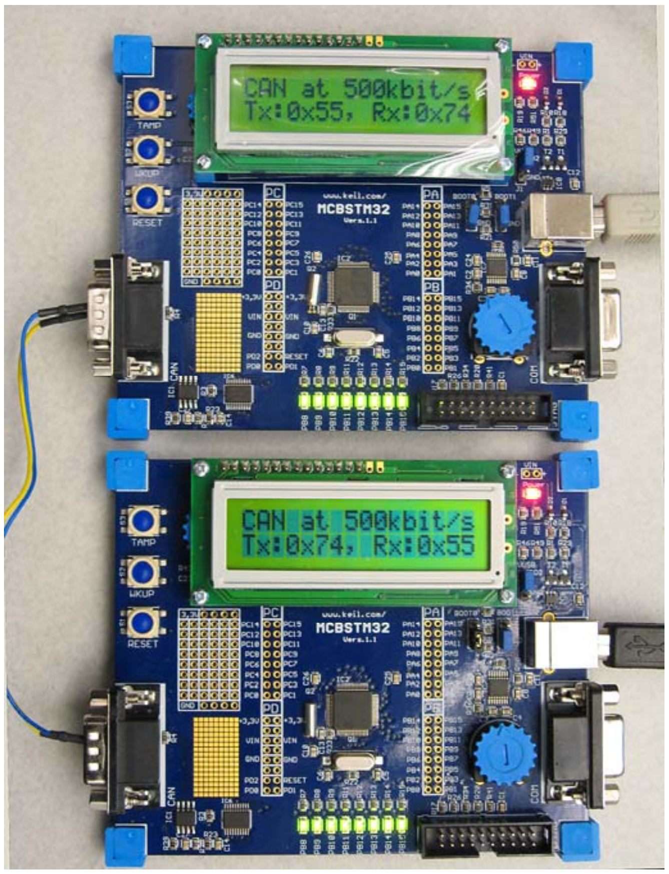


Figure 32 Two MCBSTM32 connected as 2 CAN nodes

In Figure 35 below, both pots are being turned and both their values are displayed:

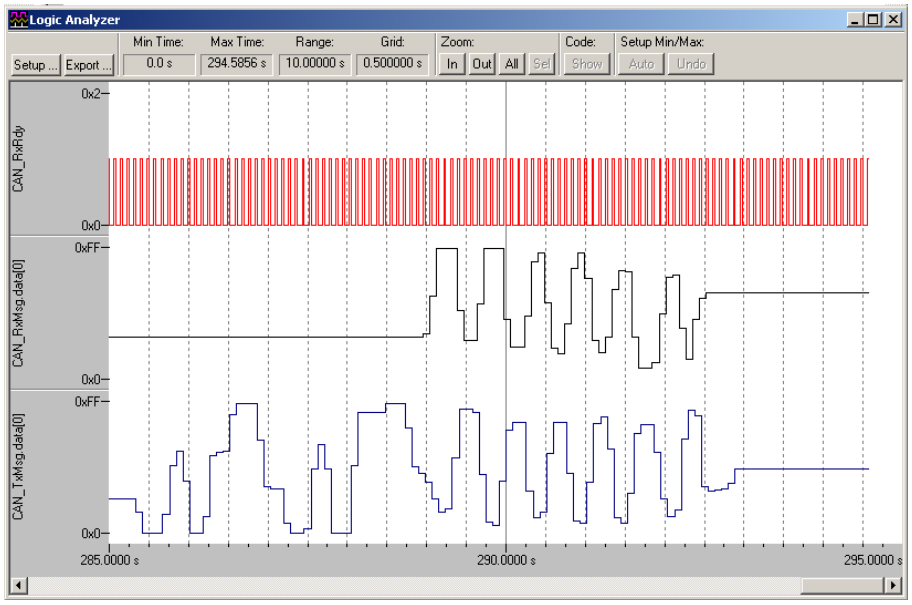
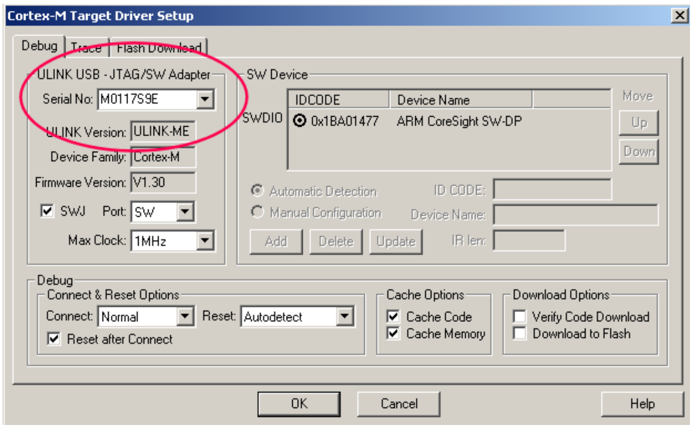


Figure 33 CAN data from each board

You can do the remaining experiments for the CAN example as listed in section 7).

Super Neat Feature: There are three ways to run multiple hardware configurations:

1. **Connect each board to a ULINK2 and to its own PC.** It is obvious how to do this.
2. **If you have only one ULINK2**, perform the steps for one board and then connect the other board up and repeat the programming steps. On RESET, without a ULINK2 connected, a properly programmed board will run the CAN program stand-alone. You can then debug the other board normally.
3. **If you have only one PC**, it is possible to run each board separately from one instance of uVision or run two separate instances of µVision using two ULINK2s. In both cases, each ULINK2 has a unique serial number that you can use to connect to it. Select this serial number as shown in the window in Figure 34:
    a. **One Instance:** You are able to select each ULINK2, one at a time, from the Cortex-M Target Driver Setup as shown below when not in debug mode. Just select the appropriate ULINK2 device in the Serial Number box and return to debug mode and you will connect to this ULINK2. Select the other serials number to change boards.
    b. **Two Instances of µµVision:** Start µVision up and select a ULINK serial number as previously described. Then, start up the second instance and select the other serial number (if you are given the choice). Debug each board by switching between the two instances of µVision.


Cortex-M Target Driver Setup

ULINK USB - JTAG/SW Adapter
Serial No: M0117S9E
ULINK Version: ULINK-ME
Device Family: Cortex-M
Firmware Version: V1.30
☑ SWJ Port: SW
Max Clock: 1MHz

SW Device

	IDCODE	Device Name
SWDIO	⦿ 0x1BA01477	ARM CoreSight SW-DP

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

Debug
Connect & Reset Options
Connect: Normal Reset: Autodetect
☑ Reset after Connect

Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash

Figure 34

Appendix A:

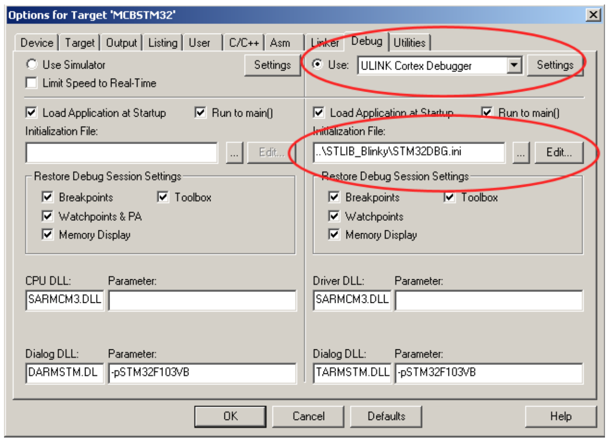
Configuring the Serial Wire Viewer (SWV):

The configuration of the SWV needs to be done for each project you use. It will be saved when you close a project or µVision. There is two parts to this setup: one to select the Serial Wire Debug Port instead of the JTAG port (step 6 below) and the other is configuring the Serial Wire Output (steps 4 and 7).

Steps:

µVision must not be in debugging mode for these steps. If it is, click on the debugger icon. The menus below will be grayed out if µVision is in debugger mode.

1) Select your project and make sure the correct processor is selected. (Project/Select Project).
2) Open the Debug setup menu. This can be accessed three ways:
   a) Click on Flash/Configure Flash Tools and select the Debug tab.
   b) Click on Project/Options for target "MCBSTM32". Select the Debug tab.
   c) Click on Options For Target icon on main µVision toolbar. Select the Debug tab.
      Number 3) is probably the easiest to use, but whichever one you use, Figure 35 opens up:

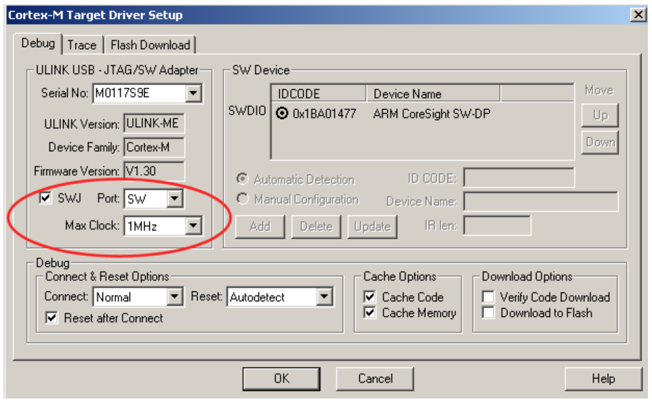

Options for Target 'MCBSTM32'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

( ) Use Simulator
[ ] Limit Speed to Real-Time
[✓] Load Application at Startup [✓] Run to main()
Initialization File:
Restore Debug Session Settings: [✓] Breakpoints [✓] Toolbox [✓] Watchpoints & PA [✓] Memory Display
CPU DLL: SARMCM3.DLL Parameter:
Dialog DLL: DARMSTM.DLL Parameter: -pSTM32F103VB

(•) Use: ULINK Cortex Debugger
[✓] Load Application at Startup [✓] Run to main()
Initialization File: ..\STLIB_Blinky\STM32DBG.ini
Restore Debug Session Settings: [✓] Breakpoints [✓] Toolbox [✓] Watchpoints [✓] Memory Display
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TARMSTM.DLL Parameter: -pSTM32F103VB

Figure 35

3) Select ULINK Cortex Debugger.
4) In the box Initialization File: select the file STM32DBG.ini as shown. This file is located in the directory C:\Keil\ARM\Boards\Keil\MCBSTM32\STLIB_Blinky and it configures the SWV ports. You can rename and/or relocate this file if you prefer. It will not hurt if this is enabled for every exercise.
5) Click on Settings to configure the ULINK2. Figure 36 opens up.


Cortex-M Target Driver Setup
Debug | Trace | Flash Download

ULINK USB - JTAG/SW Adapter
Serial No: M0117S9E
ULINK Version: ULINK-ME
Device Family: Cortex-M
Firmware Version: V1.30
[✓] SWJ Port: SW
Max Clock: 1MHz

SW Device

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

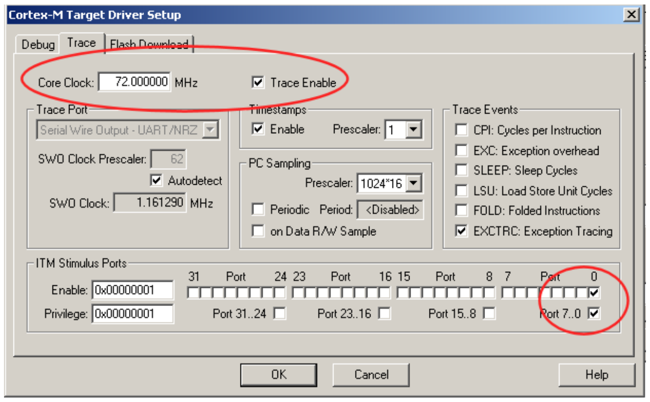
(•) Automatic Detection ID CODE:
( ) Manual Configuration Device Name:
IR len:

Debug
Connect & Reset Options: Connect: Normal Reset: Autodetect
[✓] Reset after Connect
Cache Options: [✓] Cache Code [✓] Cache Memory
Download Options: [ ] Verify Code Download [ ] Download to Flash

Figure 36

6) Select the SWJ box and set Port: to SW as shown. This must not be set to JTAG. SWV operates only through the Serial Wire debug port (SW). Max clock @ 1 MHz is correct. Up to 10 MHz will work.

7) Select the Trace tab and Figure 37 opens up to configure the SWV trace:



Core Clock: 72.000000 MHz
[✓] Trace Enable

Trace Port: Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 62
[✓] Autodetect
SWO Clock: 1.161290 MHz

Timestamps: [✓] Enable Prescaler: 1

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

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

ITM Stimulus Ports:
Enable: 0x00000001
Privilege: 0x00000001
Port 31..24 [ ] Port 23..16 [ ] Port 15..8 [ ] Port 7..0 [✓]

Figure 37

8) **Important Step:** Set the Core Clock to 72 MHz and check the Trace Enable box. Ensure in the ITM Stimulus Ports that at least Port 0 and Port 7..0 are selected. The rest are Don't Care for this exercise.

Note: if the Trace Enable box is grayed out – this means you are in Debug mode. You are not allowed to enable/disable Trace in debug mode. Close this window, leave debug mode (click on the debug icon) and click on the Target Options icon to enter the correct window. Now you will be able to set the Trace Enable box.

9) Click on OK twice. The Serial Wire Viewer Trace is now configured and ready to use !
10) Click on File/Save to save these settings.

NOTE: When in the debugger mode, if the SWV windows do not update when the program is running and only do when you stop the program execution, make sure View/Periodic Window Update is activated.

Trouble ?

- If you have trouble getting or maintaining communication with your evaluation board – please check these settings and Figures 35 and 36. Pay particular attention to the JTAG or SW setting.

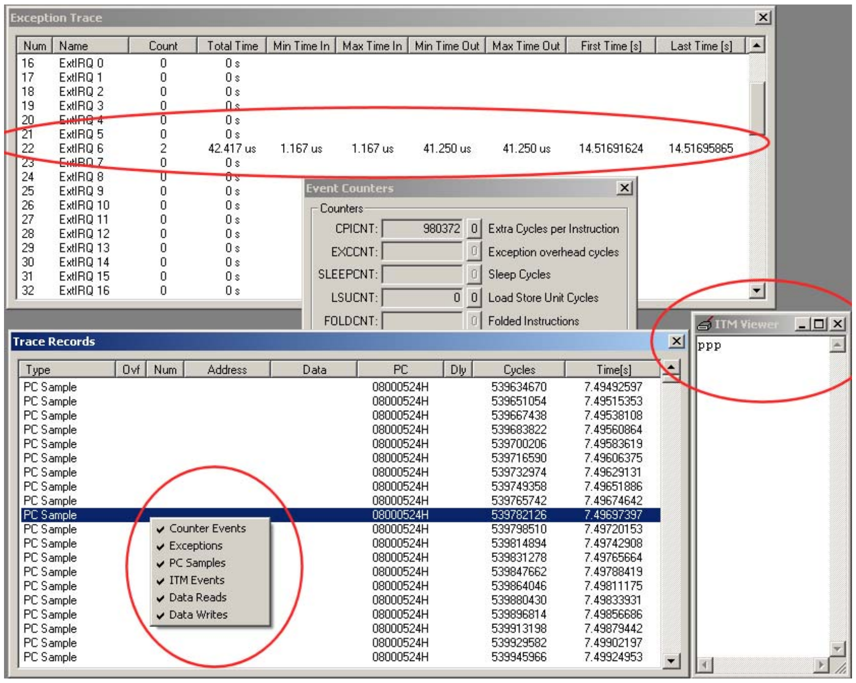
- If these windows are not correctly configured, especially with those items circled – the Serial Wire Viewer will not function. **No trace data will be displayed. If the Core Frequency is incorrect, the data will be sporadic and erroneous.**

- If you are unable to Stop program execution or µVision gets locked up – try disconnecting temporarily the ULINK2 USB cable. Usually this restores communication in a controlled fashion.

SWV Windows and data displayed (for reference only)

There are 4 windows associated with the Serial Wire Viewer and are shown in Figure 38 below.

1) ITM Viewer: writes from user code to ITM Port 0 are displayed in this window.
2) Trace Records: various cycles such as PC Samples, ITM and data reads/writes. Right click on the Trace Records window to display a filter box listing trace record types. Note timestamp columns.
3) Exception Trace: Shows when exceptions occur, how many times and how often.
4) Event Counters: Display various events as shown. These show the number of times the counters have rolled over except for CPICNT.



Exception Trace

Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	2	42.417 us	1.167 us	1.167 us	41.250 us	41.250 us	14.51691624	14.51695865
23	ExtIRQ 7	0	0 s						
24	ExtIRQ 8	0	0 s						
25	ExtIRQ 9	0	0 s						
26	ExtIRQ 10	0	0 s						
27	ExtIRQ 11	0	0 s						
28	ExtIRQ 12	0	0 s						
29	ExtIRQ 13	0	0 s						
30	ExtIRQ 14	0	0 s						
31	ExtIRQ 15	0	0 s						
32	ExtIRQ 16	0	0 s						

Event Counters

Counters

CPICNT:	980372	0	Extra Cycles per Instruction
EXCCNT:		0	Exception overhead cycles
SLEEPCNT:		0	Sleep Cycles
LSUCNT:	0	0	Load Store Unit Cycles
FOLDCNT:		0	Folded Instructions

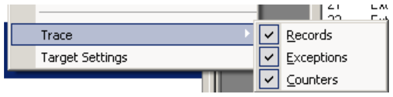
Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					08000524H		539634670	7.49492597
PC Sample					08000524H		539651054	7.49515353
PC Sample					08000524H		539667438	7.49538108
PC Sample					08000524H		539683822	7.49560864
PC Sample					08000524H		539700206	7.49583619
PC Sample					08000524H		539716590	7.49606375
PC Sample					08000524H		539732974	7.49629131
PC Sample					08000524H		539749358	7.49651886
PC Sample					08000524H		539765742	7.49674642
PC Sample					08000524H		539782126	7.49697397
PC Sample					08000524H		539798510	7.49720153
PC Sample					08000524H		539814894	7.49742908
PC Sample					08000524H		539831278	7.49765664
PC Sample					08000524H		539847662	7.49788419
PC Sample					08000524H		539864046	7.49811175
PC Sample					08000524H		539880430	7.49833931
PC Sample					08000524H		539896814	7.49856686
PC Sample					08000524H		539913198	7.49879442
PC Sample					08000524H		539929582	7.49902197
PC Sample					08000524H		539945966	7.49924953

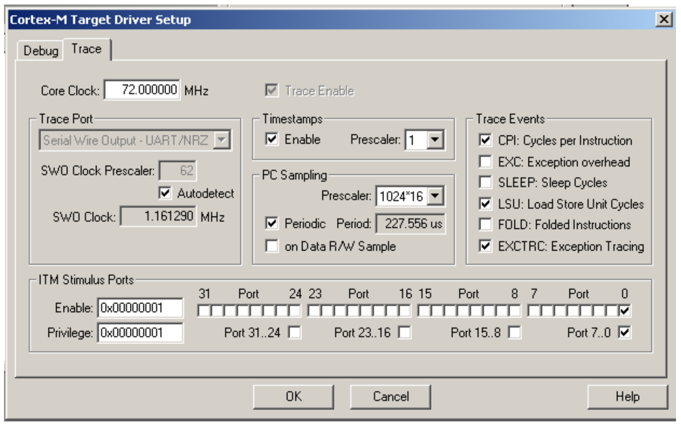
```
ppp
```

Figure 38

These windows are available only when µVision is in debug mode.
These windows are selected in Peripherals/Trace as shown here. If this menu is grayed out – the µVision is not in debug mode. In this case: please click on the debug icon: The ITM Viewer is selected in View/Serial menu.



Clicking on Target Settings will open up the Figure 39 window which you can edit. This is almost the same one you configured on the previous page but here you are unable to change the Trace Enable box.



Cortex-M Target Driver Setup

Debug | Trace

Core Clock: 72.000000 MHz   [✓] Trace Enable

Trace Port
- Serial Wire Output - UART/NRZ
- SWO Clock Prescaler: 62
- [✓] Autodetect
- SWO Clock: 1.161290 MHz

Timestamps
- [✓] Enable   Prescaler: 1

PC Sampling
- Prescaler: 1024*16
- [✓] Periodic   Period: 227.556 us
- [ ] on Data R/W Sample

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

ITM Stimulus Ports
- Enable: 0x00000001   31 Port 24 23 Port 16 15 Port 8 7 Port 0: only port 0 [✓]
- Privilege: 0x00000001   Port 31..24 [ ]   Port 23..16 [ ]   Port 15..8 [ ]   Port 7..0 [✓]

OK   Cancel   Help

Test Drive SWV (optional exercise)

1) Open the three SWV windows as shown on the previous page.
2) Open Cortex M Target Driver Setup in Peripherals/Target Settings and select the Trace tab.
3) Click the Periodic, CPI and EXC boxes in the Cortex M Target Driver Setup.
4) Click on OK.
5) Click on RUN and data will appear in at least some of the windows as shown on the previous page.
6) Note these are displayed in real-time. No CPU cycles are stolen for these operations.

Notes:

1) If you see a "x" in the Ovl (overflow) column in the trace records window – this means at least one entry was lost. If you need this information consider obtaining your trace information for the ETM. You can also unselect some of the options in the setup window to reduce the data flow.

Problems:

1) Is the correct frequency entered in the Cortex M Target Driver Setup window under the Trace tab ? Must be 72 MHz or SWV operation is not predicable. Erroneous data mat appear or none at all.
2) Is the SWJ and SW selected correctly under the Debug tab ? SWV will not work with JTAG selected.
3) Is your program running correctly ? Even if it isn't – you still should see something. That is one of the purposes of SWV – debugging when the situation is seriously wrong.

Serial Wire Viewer is easy to get operating correctly. Just check the steps described in this document and depending on your program, you should see some SWV activity.
If you still can't get it working – try the Blinky example in STLIB_Blinky. It is already configured for Serial Wire Viewer operation and is a good starting point.

What kinds of data can the Serial Wire Viewer see ?

1. Global variables.
2. Static variables.
3. Structures.
4. Peripheral registers – just read or write to them. Same is true for memory locations.
5. Can't see local variables. (just make them global or static)
6. Can't see DMA transfers. (because by definition these bypass the CPU and SWV can only see CPU actions)

Appendix B:

PWM_2 example from MCBSTM32 files. (use this if not starting from the beginning)

PWM_2: Pulse Width Modulator (TIM4) Example (only the 1st part)

LEDs PB8 and PB9 are connected to the output of TIM4 Channel 3 and Channel 4 respectively. The PWM output makes the LEDs smoothly and slowly change brightness alternatively.

7. Open the project PWM.Uv2 for the Keil MCBSTM32 board in the directory PWM_2.
8. Build the project and load this into the processor flash.
9. Put µVision into debug mode.
10. Click on GO. Note the two LEDS slowly change brightness.

Variable Ccr determines the relative brightness of LED PB8 and 100-Ccr for PB9. Variable Dir determines if a LED is increasing or decreasing.

Appendix C:

Serial Wire Viewer: What is Trace good for ?

The ARM Cortex-M3 core has a new low cost version of trace called SWV and is accessed through the same JTAG connector via the ULINK2. SWV is a superset of regular debugging via the JTAG module.
No special trace hardware is required. The standard Keil ULINK2 will perform both the JTAG and SWV functions at top speed by plugging into the standard JTAG connector on your target board.

The SWV trace has these features:

1) Data tracing. Read/write with instruction address and timestamps.
2) Hardware Breakpoints and Watchpoints.
3) ETM trigger (if the microcontroller is so equipped).
4) Program Counter or Data address sampler registers (Four 32 bit registers). (not all these can be sampled).
5) Instrumentation Trace: Through the ITM and is also called printf debugging.

In addition, the SWV trace module has counters for:

1. # of CPU clock cycles (32 bits).
2. # of folded instructions count (8 bits).
3. # of instruction cycles (8 bit).
4. # of cycles processor is sleeping (8 bit).
5. Total cycles spent in interrupt processing (8 bit).
6. Total cycles spent in load/store operations.

Each time one of these counters rolls over an event is generated and fed out to µVision. These counters can be used for various events such as measuring total elapsed time and number of events during a time period.

Usefulness of the Trace

Trace, either ETM or SWV, adds significant power to debugging efforts. Problems which may take hours, days or even weeks in big projects can often be found in a fraction of these times with a trace. Especially useful is 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). Usually, these techniques allow finding bugs without stopping the program. These are the types of problems that can be found with a quality trace:

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes).
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), corrupted stack.
4) Out of bounds data. Uninitialized variables and arrays.
5) Stack overflows. What causes the stack to grow bigger than it should ?
6) Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this.
7) Communication protocol and timing issues. System timing problems.
8) Profile analyzer.