TI MSP432E: Cortex™-M4 Tutorial *with* ETM
Using the MSP-EXP432E401Y LaunchPad Board
and ARM Keil MDK 5 Toolkit

Winter 2018 V 1.0 bob.boys@arm.com

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_309.asp

The MSP432P401 lab is here: www.keil.com/appnotes/docs/apnt_276.asp

Introduction: MSP432

The purpose of this lab is to introduce you to the TI MSP432E processor using the ARM® Keil® MDK toolkit featuring the IDE µVision®. This tutorial features ETM Instruction trace with Keil ULINK™*pro* and power monitoring with the Keil ULINK*plus*. At the end of this tutorial, you will be able to confidently work with this processor and Keil MDK.

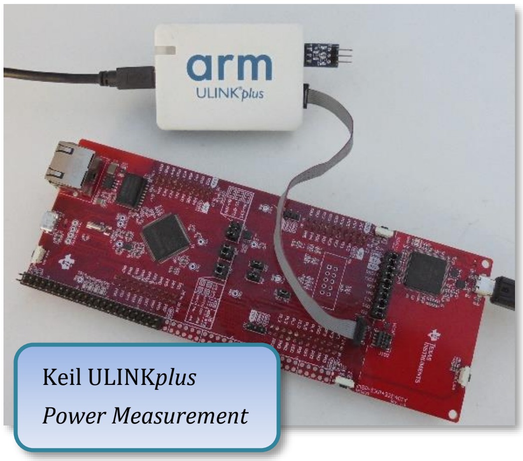
Keil MDK supports TI Cortex-M processors. Check the Keil Device Database® on www.keil.com/dd2 for the complete list. Software Packs for MSP432 series, LM3S, LM4F and Tiva C are available here: www.keil.com/dd2/pack/.

See www.keil.com/TI for more details. Keil also has TI 8051 support. DS-5™ supports TI Cortex-A. 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 specific commercial version.

Why Use Keil MDK ?

MDK provides these features particularly suited for TI Cortex-M users:


Keil ULINK*plus*
Power Measurement

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM Compiler/assembler/linker toolchain.
   MDK is a complete turn-key "out-of-the-box" tool solution.
2. You can use the ARM Compiler 5, ARM Compiler 6 (LLVM) or GCC in µVision. ELF/DWARF is supported.
3. Dynamic Syntax Checking and Code Completion.
4. **Compiler Safety Certification Kit:** www.keil.com/safety/
5. **TÜV** certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
6. MISRA C/C++ support using PC-Lint. www.gimpel.com
7. A full feature Keil RTX RTOS is included with MDK. RTX has an Apache 2.0 license. Source code is provided. www.keil.com/rtx
8. Two Kernel Awareness windows are available for RTX and are updated in real-time while your program runs.
9. ARM CoreSight™ debugging technology is supported.
10. Serial Wire Viewer (SWV) with any Keil ULINK™ or J-Link. ETM trace is on TI MSP432E401 with ULINK*pro*.
11. Debug Adapters: Launchpad XDS110, Keil ULINK™2, ULINK-ME, ULINK*plus*, ULINK*pro* and J-Link.
12. ***NEW !*** ULINK*plus*: Power Measuring. www.keil.com/ulinkplus
13. Keil Technical Support is included for one year and is easily renewable. This helps your project get completed faster.
14. Affordable perpetual and term licensing. Contact Keil sales for pricing options. See the last page in this document.
15. ***NEW !*** Event Recorder: Instrument your code. www.keil.com/pack/doc/compiler/EventRecorder/html/cv_use.html

This document includes details on these features plus more:

1. **ETM Instruction Trace.** Provides program flow debugging, Code Coverage and Performance Analysis.
2. **Serial Wire Viewer (SWV):** Data trace. View interrupts, variables graphically (Logic Analyzer) and data writes.
3. Power measurement with Keil ULINK*plus*. www.keil.com/ulinkplus
4. Real-time read and write operations to memory locations in the Watch, Memory and Peripheral windows.
5. Six Hardware Breakpoints (can be set/unset on-the-fly) and one Watchpoint (also known as Access Breaks).
6. A DSP example program using ARM CMSIS-DSP libraries including ITM *printf*. Event Recorder.
7. Create your own project with and without RTX. You can also add FreeRTOS.

General Information:

Install Keil MDK and Software Packs:

Connecting a Debug Adapter:

Blinky Example and Debugging Features:

RTX_Blinky Example with Keil RTX:

DSP Sine Example:

printf using SWV or DAP – no UART required:

ETM Instruction Trace:

Creating your own MDK 5 Blinky projects from scratch:

Other Useful Information:

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

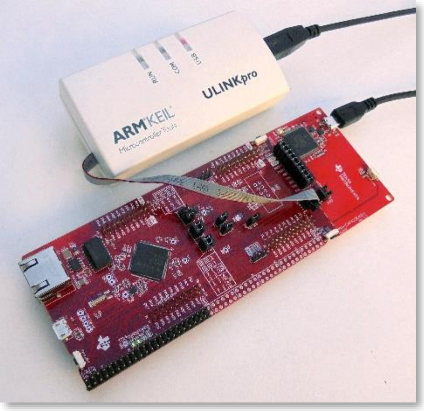
Various Cortex-M processors will have specific features. Consult your TI datasheet for feature implementation.

ETM requires a 20 pin CoreSight connector that is not usually installed on evaluation boards. Most boards have the legacy 20 pin JTAG and 10 pin CoreSight connectors. See www.keil.com/coresight/coresight-connectors It is possible to make a simple adapter to connect the ETM signals from the processor to the ULINK*pro* 20 pin CoreSight connector.

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 µVision Cortex-M Target Driver Setup. The SWJ box must be selected in ULINK2/ME or ULINK*pro*. 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 or optionally out the 4 bit Trace Port.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **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 the DAP are the memory read and write access which provides on-the-fly memory accesses without the need for processor core intervention. µVision uses the DAP to update Memory, Watch, System Viewer (peripherals) 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 and no source code stubs are used.
   You do not need to configure or activate DAP. µVision configures DAP when you select a function that uses it.
   Do not confuse this with CMSIS_DAP which is an ARM on-board debug adapter standard.
5. **SWV:** Serial Wire Viewer: a Data trace providing display of reads, writes, exceptions, PC Samples and printf.
6. **ITM:** Instrumentation Trace Macrocell: As used by µVision, ITM has 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. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
7. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
8. **Trace Port:** A 4 bit port that ULINK*pro* uses to collect ETM frames and optionally SWV (rather than SWO pin).
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK*pro* provides ETM using a suitable Cortex-M3/M4/M7 processor. ETM requires a special 20 pin CoreSight connector mounted on the target hardware. ETM also provides Code Coverage, Execution Timing and Performance Analysis. Many Cortex-M processors have ETM. Check your specific datasheet for details of features implemented by the manufacturer.
10. **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. Only on certain Cortex-M0+ processors. No special debug adapter such as a ULINK*pro* is needed. Just JTAG or SWD.
11. **Hardware Breakpoints:** Most Cortex-M0+ have 2 breakpoints. Most Cortex-M3, M4 and M7 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. The exact number of breakpoints is selected by the manufacturer at design time.
12. **WatchPoints:** Both Cortex-M0+, Cortex-M4 and Cortex-M7 have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. They are also referenced as Access Breakpoints in Keil documents.

The MSP432E401 LaunchPad board connected to a Keil ULINK*pro*. A 20 to 10 pin adapter cable is used. This cable is normally provided with the ULINK*pro*.

A custom adapter will be needed to provide the five ETM signals from the board to the 20 pin CoreSight cable on the ULINK*pro*.



2) TI MSP432E401Y LaunchPad Board & Keil Evaluation Software:

Keil provides board support for many TI Cortex-M3 and Cortex-M4 processors. This is contained in the Software Packs.

On the second last page of this document is an extensive list of resources that will help you successfully finish your projects. This list includes application notes, books, labs and tutorials.

We recommend you obtain the latest Getting Started Guide for MDK5: It is available here: www.keil.com/gsg/.

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

If you experience problems with these examples, please see Troubleshooter: in Page 39.

3) Keil MDK 5 Information: This document used *MDK 5.25.*

MDK 5 uses CMSIS compliant software. CMSIS is an ARM standard that downloads software including middleware, header and configuration files, RTX and documents from a webserver. Packs are downloaded with "Pack Installer", versions selected with "Select Software Packs" and components are selected with "Manage Run Time Environment" (MRTE).

Most TI Cortex-M processors have a Software Pack. See www.keil.com/dd2/pack for the current list.

Example Files: The examples are available where this document is stored: www.keil.com/appnotes/docs/apnt_309.asp.

RTX: RTX is a Keil RTOS that is provided with all source files and a BSD or Apache 2.0 license. This makes it free. RTX is distributed with MDK 5 and is CMSIS-RTOS compliant. You can use other RTOSs with MSP432 processors and MDK.

This tutorial used Software Pack 3.2.2.

4) USB Debug Adapters: For Serial Wire Viewer (SWV) use a ULINK2, ULINK*plus,* ULINK*pro* or J-Link.

Keil manufactures and supports several adapters. These are listed below with a brief description.

1. **Keil ULINK2 and ULINK-ME:** ULINK-ME is offered only as part of certain evaluation board packages. These are electrically the same and both support Serial Wire Viewer (SWV). SWV and DAP update non-intrusively while the program is running. Run-time memory reads and writes for the Watch, Memory and System Viewer windows are non-intrusive. Hardware breakpoints can be set/unset on-the-fly.
2. ***NEW !*** **ULINK*plus*:** High SWV performance plus Power Measurement. It is pictured on page 1. See www.keil.com/ulink/ulinkplus/ for details.
3. **Keil ULINK*pro*:** ULINK*pro* is pictured on page 3. It supports all SWV features and adds ETM Instruction Trace. ETM records all executed instructions and provides Code Coverage, Execution Profiling and Performance Analysis features. ULINK*pro* also provides the fastest Flash programming times.
4. **TI XDS110 Emulator:** MSP432 LaunchPad boards contain an on-board XDS110 debug adapter. Currently, XDS110 does not support Serial Wire Viewer. Use any ULINK or a J-Link to utilize this useful feature.
5. **Segger J-Link:** J-Link Version 6 (black case) or later supports Serial Wire Viewer. SWV data reads and writes are not currently supported with a J-Link. Program execution must be stopped to view the trace window. J-Link USB drivers must be manually installed. See C:\Keil_v5\ARM\Segger\USBDriver\ J-Link SWV configuration windows are slightly different from any Keil ULINK. Segger offers a 20 pin to 10 pin connector adapter needed to connect to the LaunchPad to the 10 pin EXT CoreSight Debug connector J11. Contact www.segger.com to obtain this adapter. This adapter is shown here: →

CMSIS-DAP: This is an ARM standard for debug adapters. SWV is a new option. CMSIS-DAP can be used to put a debug adapter on a custom board. See CMSIS-DAP is widely supported by development tools. The TI XDS110 emulator currently support CMSIS-DAP. ***New*** See https://github.com/ARM-software/CMSIS_5

Serial Wire Viewer (SWV): This is an ARM CoreSight feature. SWV provides data trace, the ability to graph four variables and see exceptions and interrupts plus other events occurring in real-time. No code stubs are needed in your code. Keil ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* and J-Link all provide SWV. TI XDS110, at this time, does not.

ETM Instruction Trace: ETM collects all the instructions executed. This is useful for program flow debugging. Code Coverage and Performance Analysis is provided. A Keil ULINK*pro* or Segger J-Trace debug adapter is needed. Only the MSP432E401 has ETM.

5) Download and Install Keil MDK Core Software:

Download MDK-Core Version 5

1. Download MDK 5.25 or later from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default folder. You can install into any folder, but this tutorial uses the default C:\Keil_v5.
1. We recommend you use the default folders for this tutorial. We will put the examples in C:\00MDK\TI\MSP432E\
3. You do not need a debug adapter: just the LaunchPad board, a USB cable and MDK installed on your PC. With a Keil ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link, you can use Serial Wire Viewer. A ULINK*pro* adds ETM instruction trace. A ULINLK*plus* adds Power Measurement.
4. You do not need an MDK license for this tutorial. All examples will compile within the 32 K limit.
5. For projects larger than 32K or to evaluate Keil Middleware, you can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible: Evaluate MDK Professional
6. If you need additional time for evaluation purposes, please contact Keil Sales as listed on the last page.

6) Install the MSP432E Software Pack:

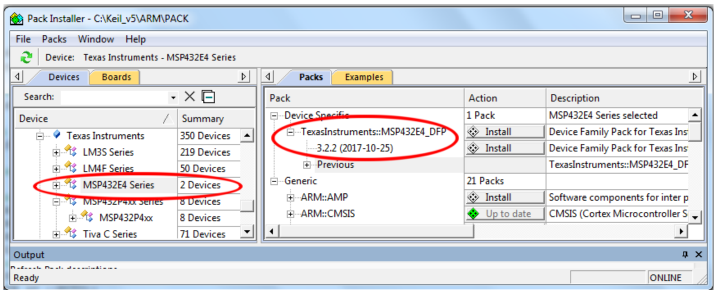
A Software Pack contains components such as headers, Flash programming, documents and other files used in a project.

TIP: A Pack is an ordinary zip file with an extension .pack. A few IT departments do not allow an application such as Pack Installer to download a zip file. In this case, download the Pack directly from www.keil.com/dd2/pack or contact Keil tech support to obtain it. Then, install it manually by double clicking on the Pack or select File/Import in Pack Installer.

2. Start µVision.
3. Open the Pack Installer (PI) by clicking on its icon: This window opens up:
4. Select the Devices tab. Highlight Texas Instruments and then MSP432E4 Series as shown:

TIP: What is entered in the Devices or Boards tabs filters what is displayed in the Packs and Examples tabs.

5. Select the Packs tab. Install
6. Click on the 3.2.2 Pack (or later) Install.
7. It will install to the µVision install folders.



Pack Installer - C:\Keil_v5\ARM\PACK

Device: Texas Instruments - MSP432E4 Series

Device	Summary
Texas Instruments	350 Devices
LM3S Series	219 Devices
LM4F Series	50 Devices
MSP432E4 Series	2 Devices
MSP432P4xx Series	8 Devices
MSP432P4xx	8 Devices
Tiva C Series	71 Devices

Pack	Action	Description
Device Specific	1 Pack	MSP432E4 Series selected
TexasInstruments::MSP432E4_DFP	Install	Device Family Pack for Texas Ins
3.2.2 (2017-10-25)	Install	Device Family Pack for Texas Ins
Previous		TexasInstruments::MSP432E4_DF
Generic	21 Packs	
ARM::AMP	Install	Software components for inter p
ARM::CMSIS	Up to date	CMSIS (Cortex Microcontroller S

7) Install the MSP432E Examples

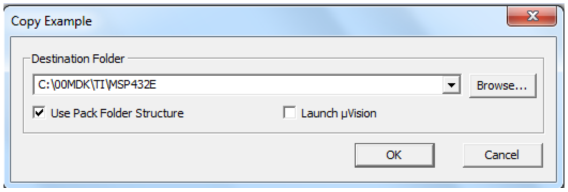
Install the Pack Examples:

8. Select the Examples tab. Two examples are visible: BlinkyLED and EmptyMain. Select Copy for each of them: Copy
9. The Copy Example window opens: Enter C:\00MDK\TI\MSP432E\ Unselect Launch µVision. Click OK.
10. Repeat so both examples are copied to C:\00MDK\TI\MSP432E\Examples\
11. Close the Packs Installer. You can open it any time by clicking on its icon.
12. If a dialog box opens stating: "Software Pack folder has been modified. Reload Packs?" Click Yes.

TIP: A Pack's status will be indicated by the "Up to date" icon: Up to date

The "Update" icon means there is an updated Software Pack available for download. Update If you update the files and later need older ones, you can select them in Select Software Pack.

TIP: Select "Check for Updates" or File/Refresh in PI to check for Packs updates. You must be connected to the Internet.



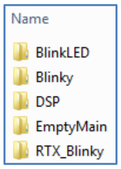
Copy Example

Destination Folder: C:\00MDK\TI\MSP432E

☑ Use Pack Folder Structure □ Launch µVision

Install the Keil MSP432E Examples:

1. The Keil MSP432E examples are available on the Keil website.
2. Obtain the example zip file from www.keil.com/appnotes/docs/apnt_309.asp.
3. Create the folder C:\00MDK\TI\MSP432E\Examples\ and extract the .zip file into it:
4. You will have these files in the specified folder if you include the examples from the Pack:



Name
BlinkLED
Blinky
DSP
EmptyMain
RTX_Blinky

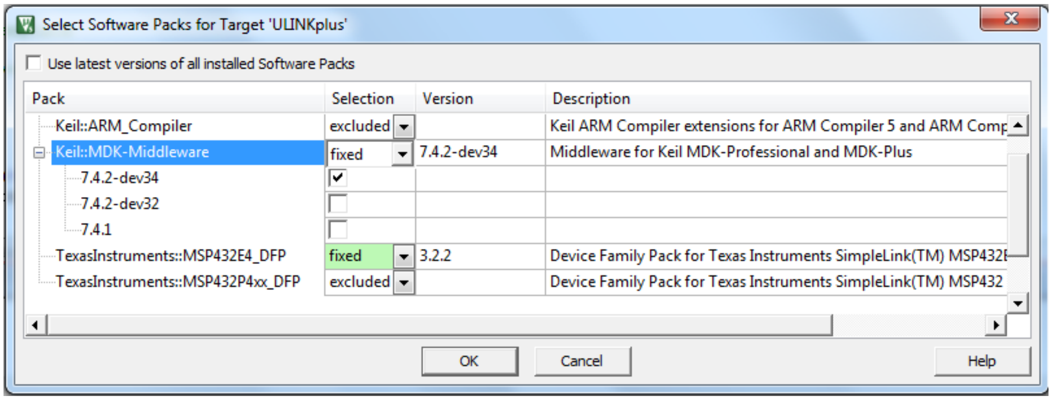
TIP: Obtain TI code here: www.ti.com/tool/download/SIMPLELINK-MSP432E4-SDK

8) Other features of CMSIS-Pack Software Packs: (for reference)

A) Select Software Packs (version selection): Use this to select the Pack version you want to use.

This feature provides the ability to choose various Software Pack versions installed in your computer. You can select the versions you want to use. "Use the latest versions" is the default. This is easily overridden for custom version selection.

1. Open Select Software Pack by clicking on its icon: You can also it open with Project/Manage/Select Software Packs.
2. This window opens up. Note Use latest versions … is selected.
3. Unselect this setting. Expand one of the entries – Keil::MDK – Middleware is a good one to try.

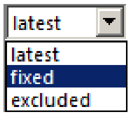


Select Software Packs for Target 'ULINKplus'

☐ Use latest versions of all installed Software Packs

Pack	Selection	Version	Description
Keil::ARM_Compiler	excluded		Keil ARM Compiler extensions for ARM Compiler 5 and ARM Comp
⊟ Keil::MDK-Middleware	fixed	7.4.2-dev34	Middleware for Keil MDK-Professional and MDK-Plus
7.4.2-dev34	☑		
7.4.2-dev32	☐		
7.4.1	☐		
TexasInstruments::MSP432E4_DFP	fixed	3.2.2	Device Family Pack for Texas Instruments SimpleLink(TM) MSP432E
TexasInstruments::MSP432P4xx_DFP	excluded		Device Family Pack for Texas Instruments SimpleLink(TM) MSP432

1. The Middleware Pack has three versions installed on this computer at this time: You may see different versions. If there were more than one Pack for the MSP series, they would be visible and selectable.
2. Select **fixed** and then the version you want to use. In this case, 7.4.2dev34.
3. ***Re-select Use latest versions of …***
4. Close this window with Cancel. **Do not make any changes at this time.** We will use the latest version.



B) Manage Run-Time Environment: Use this to add various software components to your project.

1. Select Project/Open Project and select Blinky.uvprojx in C:\00MDK\TI\MSP432E\Examples\Blinky\
2. Click on the Manage RTE icon: The next window opens:
3. Expand various headers and note the selections you can make. A selection made here will automatically select and insert the appropriate source files into your project for your convenience. Core and Startup are very important files.
4. Do not make any changes. Click Cancel to close this window.

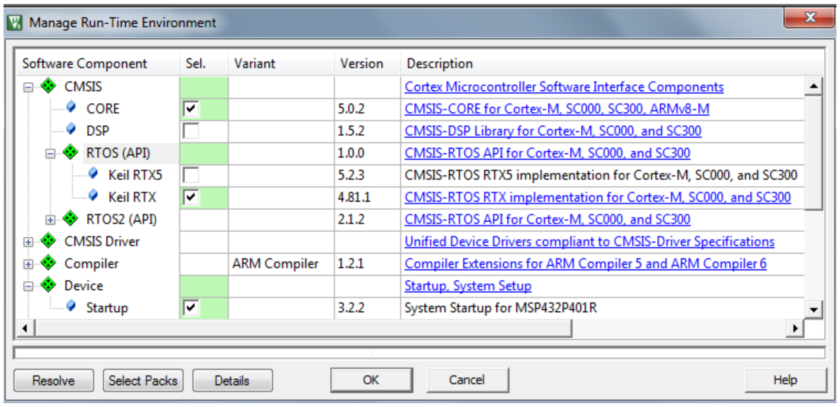
TIP: Different colors represent messages:

☑ Green: all required files are located.

☑ Yellow: some files need to be added. Click the Resolve button to add them automatically or add them manually.

☑ Red: some required files could not be found. Obtain these files or contact Keil technical support for assistance.

The Validation Output area at the bottom of this window gives some information about needed files and current status.



Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
⊟ CMSIS				Cortex Microcontroller Software Interface Components
CORE	☑		5.0.2	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M
DSP	☐		1.5.2	CMSIS-DSP Library for Cortex-M, SC000, and SC300
⊟ RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
Keil RTX5	☐		5.2.3	CMSIS-RTOS RTX5 implementation for Cortex-M, SC000, and SC300
Keil RTX	☑		4.81.1	CMSIS-RTOS RTX implementation for Cortex-M, SC000, and SC300
⊞ RTOS2 (API)			2.1.2	CMSIS-RTOS API for Cortex-M, SC000, and SC300
⊞ CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
⊞ Compiler		ARM Compiler	1.2.1	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
⊟ Device				Startup, System Setup
Startup	☑		3.2.2	System Startup for MSP432P401R

9) Install the XDS110 Drivers and Update the LaunchPad XDS firmware:

If you are going to use the on-board XDS110 debug adapter, the XDS110 drivers ***must*** be installed at C:\ti\.

Skip this page if you are going to use a Keil ULINK2, ULINK*plus*, ULINK*pro*, CMSIS-DAP or a J-Link, you do not need to install the XDS110 drivers. ULINK drivers are already installed and J-Link must be installed manually.

Download and Install the XDS110 Emulation Software Package (emupack):

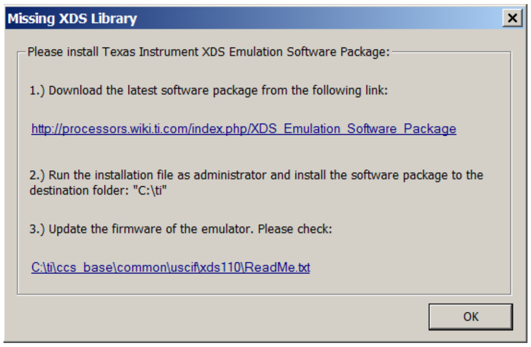
1. Go to http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package
2. On this website, download the latest XDS Emulation Software for Windows. You need a free TI account.
3. The filename will be similar to **ti_emupack_setup_6.0.228.1_win_32.exe** *or later*.
4. Run this executable. It will install files to the default folder C:\ti\. Do not change this default folder. µVision looks in this folder for the needed XDS110 driver software.

Install/Update the Firmware in the LaunchPAD XDS110 Processor:

1. Open the ReadMe file: C:\ti\ccs_base\common\uscif\xds110\ReadMe.txt
2. This file contains instructions on how to update the Launchpad firmware using the Windows Command prompt.

TIP: It is a good idea to use the same versions of software and firmware.

TIP: If the XDS110 debug adapter is selected, and if the appropriate XDS110 files are not present in C:\ti\: when you attempt to enter Debug mode in µVision this alert window opens: The above instructions are repeated here to install the drivers and to update the LaunchPad firmware.



Missing XDS Library

Please install Texas Instrument XDS Emulation Software Package:

1.) Download the latest software package from the following link:

http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package

2.) Run the installation file as administrator and install the software package to the destination folder: "C:\ti"

3.) Update the firmware of the emulator. Please check:

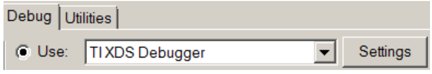
C:\ti\ccs_base\common\uscif\xds110\ReadMe.txt

Using the LaunchPad XDS110 Emulator with µVision:

1. Start µVision if it is not already running.
2. Select Project/Open Project and select Blinky.uvprojx in C:\00MDK\TI\MSP432E\Examples\Blinky\.
3. Select XDS in the µVision Target Selector: XDS
4. Connect a USB cable to the LaunchPad USB connector and your PC as shown here:
5. J101 must have all 10 jumpers populated.
6. A green LED will illuminate. Test the XDS110 connection as described below:

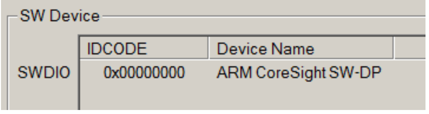
Testing the Debug Connection: You need any valid µVision project open for this step.

1. Select Target Options or ALT-F7 and select the Debug tab: Select your debugger. This selection is for the XDS110 on-board debug adapter:



● Use: TI XDS Debugger

2. Click on Settings: and the Target Driver Setup window opens:
3. Select SW in the Port box.
4. An IDCODE and Device name will then be displayed indicating a valid connection to the CoreSight DAP. **This means the debug adapter is working correctly !** You can continue with the tutorial.



SW Device

	IDCODE	Device Name
SWDIO	0x00000000	ARM CoreSight SW-DP

5. If nothing or an error is displayed in this SW Device box, this ***must*** be corrected before you can continue. Check your connections and settings. Reinstall the firmware. All 10 jumpers of J101 must be installed for XDS110 operation. Remove RST through TDI for an external adapter such as any Keil ULINK or Segger J-Link if there is a conflict.
6. Click on OK twice to return to the µVision main menu.

TIP: You can use these steps to test the connection of any debug adapter such as any ULINK or a J-Link.

TIP: The TI XDS110 Emulator does not yet support Serial Wire Viewer (SWV). For SWV, use a Keil ULINK2, ULINK*plus*, ULINK*pro* or a J-Link as described on the next page. SWV is a very useful debugging tool and is worth using.

10) Configuring a Debug Adapter:

You can use an external debug adapter. The main benefit is the ability to use Serial Wire Viewer (SWV). SWV is data trace and will display exceptions and interrupts, date read and write operations, graphical display of variables and more. ETM instruction trace is a powerful debugging technology and provides program flow debugging, Performance Analysis and Code Coverage. The source windows display code as it was written and the ETM window shows it as it was executed.

You can use a Keil ULINK2, ULINK -ME, ULINK*plus*, ULINK*pro* or a J-Link and many CMSIS-DAP compatible adapters.

This page uses the example BlinkLED. It is configured with ULINK2 by default. We will add another debug adapter.

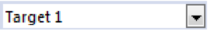
Connect your external debug adapter and Select the Project:

1. If you want to use the on-board XDS110 adapter, plug in the USB cable as described on the previous page.
2. For all external adapters, connect to the 10 pin CoreSight debug connector J11.
3. Power the board with a USB cable and power your external adapter (if using one) by connecting to your PC.
4. Select Open Project and navigate to C:\00MDK\TI\MSP432E\Examples\BlinkLED\ARM.
5. Select BlinkLED.uvprojx. Click Open. This project will open in µVision.

Adding a Target Hardware Configuration:

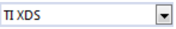
The Target Options configures many items related to your project and processor. These can all be individually saved and recalled with the Select Target drop down menu.

1. In Edit mode, in the Select Target box, choose the adapter you want use as a template.

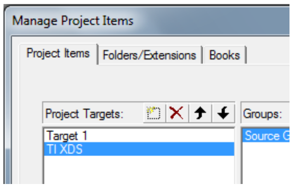


Target 1

2. Select Open Project/Manage and then select Project Items… or click
3. Click on the Insert icon or the Insert key and enter a name:
4. We used TI XDS as an example.
5. Click on close and this name will now be available in the Select Target box.
6. Select the target name you just created.



TI XDS

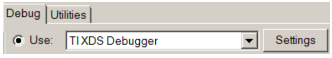


Manage Project Items

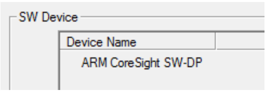
Project Targets:
Target 1
TI XDS

Configuring Your Debug Adapter:

1. Click on the Options for Target or ALT-F7. Select the Debug tab.
2. Select your Debug Adapter in the Use: box as shown here:
3. Click Settings: In the SW Device box a valid Device Name must be visible:
4. An IDCODE can be visible with other debug adapters.
5. Click on the Flash Download tab and confirm a program algorithm is present:
6. Click OK twice to return to the main µVision menu.



● Use: TI XDS Debugger

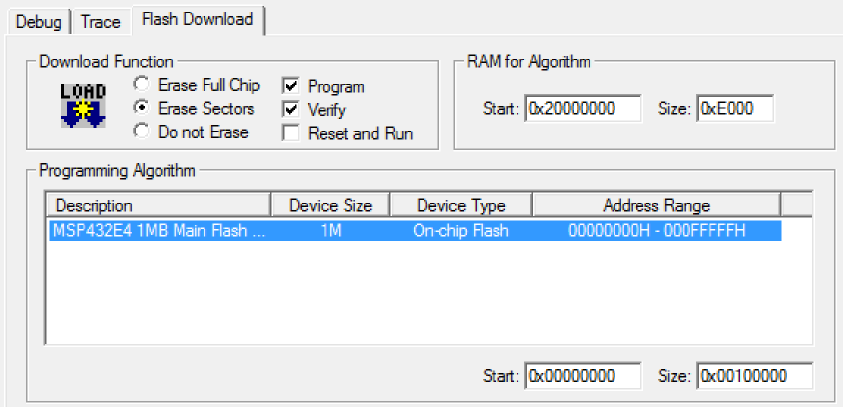


SW Device

Device Name
ARM CoreSight SW-DP

Problems with Flash Programming:

1. RAM Start is at a valid address. In this case it is 0x2000_0000.
2. Increase RAM size by 0x1000.
3. Under the Debug tab, try various options in the Reset: box.
4. Try various Connect options.
5. Unselect Verify Code Download.
6. Select slower Max Clock: speed.
7. In case of conflict with an external debug adapter, Remove jumpers RST, TMS, TCK, TDO and TDI on J101 to disconnect the onboard XDS110 adapter.



Download Function
○ Erase Full Chip
● Erase Sectors
○ Do not Erase
☑ Program
☑ Verify
☐ Reset and Run

RAM for Algorithm
Start: 0x20000000 Size: 0xE000

Programming Algorithm

Description	Device Size	Device Type	Address Range
MSP432E4 1MB Main Flash ...	1M	On-chip Flash	00000000H - 000FFFFFH

Start: 0x00000000 Size: 0x00100000

Segger J-Link/J-Trace Notes:

1. J-Link USB drivers must be manually installed (unless they are already installed). The drivers are located here: C:\Keil_v5\ARM\Segger\USBDriver\InstDrivers.exe or C:\Keil_v5\ARM\Segger\USBDriver\CDC\.

11) *Blinky* example program using the MSP432E401Y LaunchPad:

Using the Keil Blinky example, we will run our first program on the MSP432E Launchpad.

1. Select Project/Open Project and select Blinky.uvprojx in C:\00MDK\TI\MSP432E\Examples\Blinky\.
2. Select your debug adapter in the Select Target box in µVision.
3. Connect an external debug adapter to J11 if used. Remove J101 jumpers RST, TMS, TCK, TDO and TDI to disconnect the onboard XDS110 debug adapter. For the on-board XDS110, all ten J101 jumpers must be installed.
4. Power the board with a USB cable from your PC.
5. Compile the source files by clicking on the Rebuild icon. You can also use the Build icon beside it.
6. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Progress will be indicated in the Output Window. Select OK if the Evaluation Mode box appears.
7. Click on the RUN icon. **Note:** you stop the program with the STOP icon.

The green LEDs D1 and D2 will now blink on the LaunchPad board.

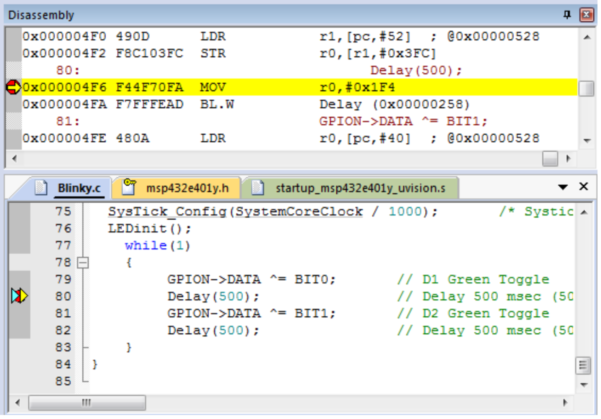
Now you know how to compile a program, program it into the TI processor Flash, run it and stop it !

TIP: The board will run BlinkLED stand-alone. BlinkLED is now permanently programmed in the Flash until reprogrammed.

12) Hardware Breakpoints:

The MSP432E 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 on the left on a suitable part of the source code. This grey box indicates assembly instructions are present at these points. Inside the while (1) loop inside the main() function between near lines 79 through 82 are suitable places: You can also click in the Disassembly window to set or unset a breakpoint on a specific assembly instruction.
2. A red circle will appear and the program will presently stop if the CPU tries to access this instruction.
3. Note the breakpoint is displayed in both the Disassembly and Blinky.c windows as shown here:
4. Set a second breakpoint in the while() loop as before.
5. Every time you click on the RUN icon the program will run until the breakpoint is again encountered.
6. The yellow arrow is the current program counter value.
7. Open Debug/Breakpoints or Ctrl-B and you can see any breakpoints set. You can unselect them or delete them here.
8. Delete all breakpoints and close the Breakpoint window.
9. Clicking in the source window will indicate the appropriate code line in the Disassembly window and vice versa. This relationship is indicated by the cyan arrow and the yellow highlight:
10. **Make sure you remove the breakpoints !**



```
0x000004F0 490D      LDR          r1,[pc,#52]  ; @0x00000528
0x000004F2 F8C103FC  STR          r0,[r1,#0x3FC]
    80:                                 Delay(500);
0x000004F6 F44F70FA  MOV          r0,#0x1F4
0x000004FA F7FFFEAD  BL.W         Delay (0x00000258)
    81:                           GPION->DATA ^= BIT1;
0x000004FE 480A      LDR          r0,[pc,#40]  ; @0x00000528
```

```
75    SysTick_Config(SystemCoreClock / 1000);       /* Systic
76    LEDinit();
77      while(1)
78      {
79          GPION->DATA ^= BIT0;        // D1 Green Toggle
80          Delay(500);                 // Delay 500 msec (50
81          GPION->DATA ^= BIT1;        // D2 Green Toggle
82          Delay(500);                 // Delay 500 msec (50
83      }
84  }
85
```

TIP: ARM hardware breakpoints do **not** execute the instruction they are set to and land on. CoreSight hardware breakpoints are no-skid. This is an important feature for effective debugging as is the ability to set/unset them while the program runs.

Single-Stepping:

1. With Blinky.c in focus (click anywhere inside Blinky.c), click on the Step In icon or F11 a few times: The program counter steps one C line at a time. The yellow arrow indicates the next C line or instruction to be executed.
2. Click on the top margin of the Disassembly window to bring it into focus. Clicking Step Into now jumps the program counter PC one assembly instruction at a time.

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

The Watch and Memory windows will display updated variable values in real-time. It does this by using ARM CoreSight DAP debugging technology that is part of Cortex-M processors. It is also possible to “put” or insert values into a Watch or Memory window in real-time. It is possible to “drag and drop” variable names into windows or enter them manually. You can also right click on a variable and select Add *varname* to.. and select the appropriate window. The System Viewer window works using the same CoreSight DAP technology. DAP works with all debug adapters including XDS110.

Watch window:

Add a global variable: Call Stack, Watch and Memory windows can’t see local variables unless stopped in their function.

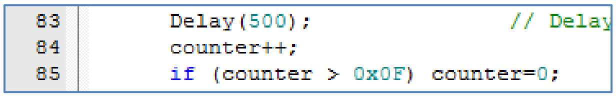
1. Stop the processor and exit Debug mode.
2. In Blinky.c, declare a global variable called **`counter`** near line 10:

   unsigned int counter = 0;

3. Add these statements near Line 83 in the while(1) loop in the main() function:

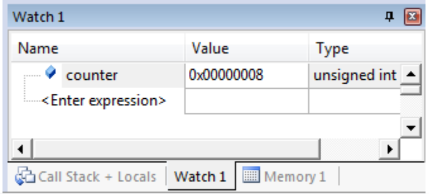
   counter++;

   if (counter > 0x0F) counter = 0;



```
83          Delay(500);                    // Delay
84          counter++;
85          if (counter > 0x0F) counter=0;
```

4. Select File/Save All.
5. Click on Rebuild . There will be no errors or warnings: If there are, please fix them before continuing.
6. Enter Debug mode. The Flash will be programmed. Click on RUN . You can configure a Watch or Memory window while the program is running. The program must be stopped to remove a variable from a Watch window.
7. In Blinky.c, right click on **`counter`** and select Add 'counter' to … and select Watch 1. Watch 1 will automatically open. **`counter`** will be displayed as shown here:
8. **`counter`** is immediately updated in real time !



Watch 1

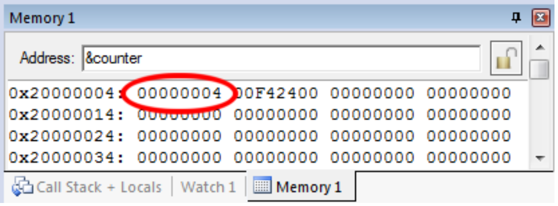
Name	Value	Type
counter	0x00000008	unsigned int
<Enter expression>		

TIP: It is possible to change a variable n a Watch window while the program runs if the value is not changing too fast. If it is too difficult to do this, just use a Memory window as described below.

TIP: A Watch or Memory window can display and update global and static variables, structures, raw addresses and peripheral addresses while the program is running. These are unable to display local variables because these are typically stored in a CPU register which cannot be read by µVision in real-time. To view a local variable in these windows, convert it to a static or global variable.

Memory window:

1. Right click on **`counter`** in Blinky.c and select Add 'counter' to … and select Memory 1.
2. Note the value of **`counter`** is displaying its value in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to, but this not what we want to see at this time.
3. Add an ampersand “&” in front of the variable name and press Enter. The physical address here is 0x2000_0004.
4. Right click in the Memory window and select Unsigned/Int.
5. The data contents of **`counter`** is displayed as shown here:
6. Both the Watch and Memory windows are updated in real-time.
7. Right-click the mouse cursor over the desired data field and select Modify Memory. You can change a memory or variable on-the-fly while the program is still running.



Memory 1

Address: &counter

```
0x20000004: 00000004 00F42400 00000000 00000000
0x20000014: 00000000 00000000 00000000 00000000
0x20000024: 00000000 00000000 00000000 00000000
0x20000034: 00000000 00000000 00000000 00000000
```

TIP: No CPU cycles are usually used to perform these operations.

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

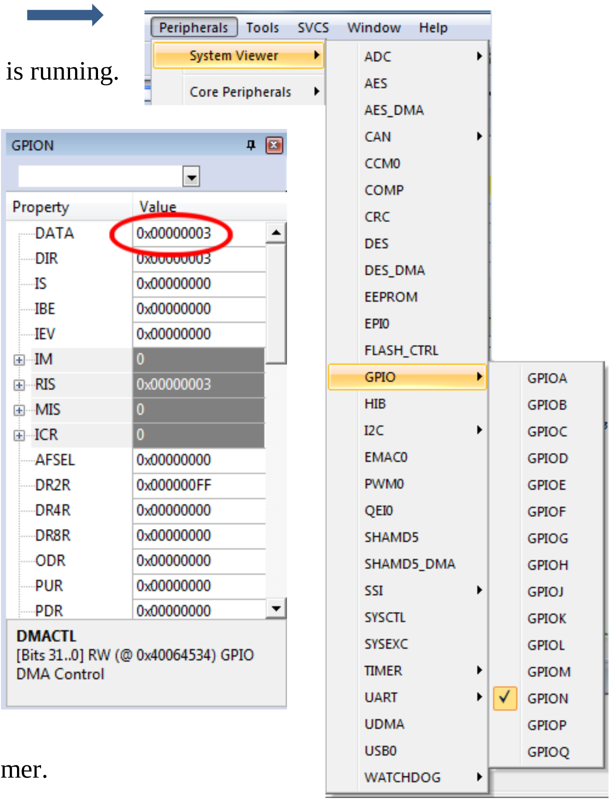
14) System Viewer (SV):

System Viewer provides the ability to view registers in the CPU core and in peripherals. In most cases, these peripherals are updated in real-time while your program is running. These Views are available only while in Debug mode. There are two ways to access these Views: **a)** View/System Viewer and **b)** Peripherals/System Viewer. In the Peripheral/Viewer menu, the Core Peripherals are also available: Note the various peripherals available.

1. Click on RUN. You can open SV windows when the program is running.

GPIO Port N:

2. Select Peripherals/System Viewer/GPIO/GPION.
3. This window opens up. Note the Data value changing:
4. You can now see D1 and D2 update as the LEDs blink.
5. These are updated periodically, not when the value changes. To view a register when it changes, you need to use Serial Wire Viewer and the Logic Analyzer. A ULINK2, ULINK*plus*, ULINK*pro* or a J-Link is needed.
6. You can change the values in the System Viewer on-the-fly. In this case, the values are updated quickly so they are hard to see.



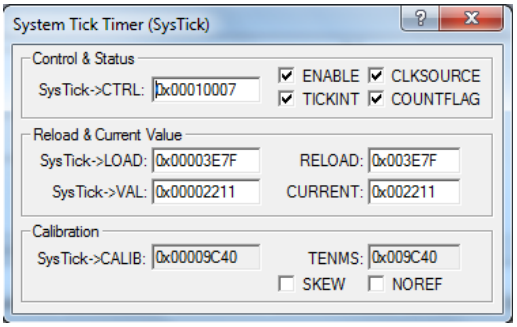
Property	Value
DATA	0x00000003
DIR	0x00000003
IS	0x00000000
IBE	0x00000000
IEV	0x00000000
IM	0
RIS	0x00000003
MIS	0
ICR	0
AFSEL	0x00000000
DR2R	0x000000FF
DR4R	0x00000000
DR8R	0x00000000
ODR	0x00000000
PUR	0x00000000
PDR	0x00000000

DMACTL
[Bits 31..0] RW (@ 0x40064534) GPIO DMA Control

TIP: If you click on a register in the Property column, a description about this register will appear at the bottom of the SV window. DMACTL is shown: This is an easy way to find the physical address of a register.

SysTick Timer: This program uses the SysTick timer as a timer for the Delay function.. This is configured and activated in Blinky.c near line 75: SysTick_Config(SystemCoreClock / 1000);

1. Select Peripherals/Core Peripherals and then select System Tick Timer.
2. The System Tick Timer window shown below opens:
3. Note it also updates in real-time while your program runs. These windows use the same CoreSight DAP technology as the Watch, Memory and SV windows. Do not confuse this DAP (Debug Access Port) with CMSIS-DAP.
4. Note the ST_RELOAD and RELOAD registers. This is the reload register value set by the SysTick_Config function.
5. Note that it is set to 0x3E7F. This is the hex value of 16,000,000/1000-1 that is programmed into SysTick_Config() in main() in Blinky.c. The CPU clock in this example is 16 MHz. Changing the variable passed to this function is how you change how often the SysTick timer triggers interrupt 15.
6. In the RELOAD register in the SysTick window, *while the program is running*, type in 0x1000, press Enter and click inside ST_RELOAD ! (or the other way around)
7. The blinking LEDs will speed up. This will convince you of the power of ARM CoreSight debugging.
8. Replace RELOAD with 0x3E7F. A CPU RESET will also do this.
9. You can look at other Peripherals contained in the System View windows.
10. When you are done, stop the program and close all the System Viewer windows that are open.
11. Exit Debug mode.



Control & Status
SysTick->CTRL: 0x00010007 ☑ ENABLE ☑ CLKSOURCE ☑ TICKINT ☑ COUNTFLAG

Reload & Current Value
SysTick->LOAD: 0x00003E7F RELOAD: 0x003E7F
SysTick->VAL: 0x00002211 CURRENT: 0x002211

Calibration
SysTick->CALIB: 0x00009C40 TENMS: 0x009C40
☐ SKEW ☐ NOREF

TIP: It is true: you can modify values in the System Viewer while the program is running. This is very useful for making slight timing value changes instead of the usual modify, compile, program, run cycle.

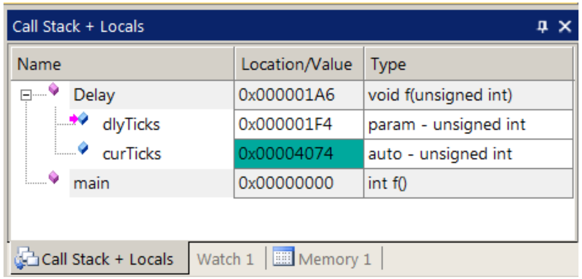
You must make sure a given peripheral register allows for and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.

15) Call Stack + Locals Window:

The Call Stack and Locals 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 located in 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 can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

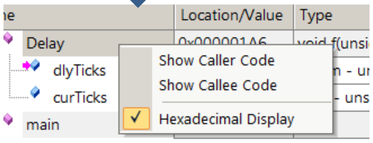
1. Use the same Blinky project as from the previous page.
2. Enter the Debug mode . The Flash will be programmed only if changes to the program executable were made.
3. Click on RUN. After a second or two, stop the CPU.
4. Click on the Call Stack + Locals tab if necessary to open it. Expand some of the entries.
5. Shown is a Call Stack + Locals window similar to this one: The program will most likely stop in the Delay function.



Call Stack + Locals

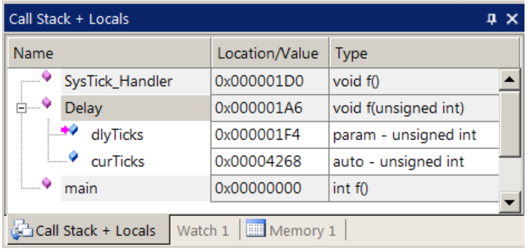
Name	Location/Value	Type
Delay	0x000001A6	void f(unsigned int)
dlyTicks	0x000001F4	param - unsigned int
curTicks	0x00004074	auto - unsigned int
main	0x00000000	int f()

6. The functions are displayed in the order they were called.
7. Note the two local variables in Delay() are displayed with their values since the program is stopped in this function.
8. Right click on the Delay name and select either Callee or Caller code and this will be highlighted in the source and Disassembly windows.



ne	Location/Value	Type
Delay	0x000001A6	void f(unsi
dlyTicks		n - ur
curTicks		- uns
main		

9. Click on Step Out and the Delay() function will not be on the stack as evidenced in the Call Stack window.
10. Set a breakpoint in the SysTick handler near line 60 in Blinky.c.
11. Click on RUN and the program will run to SysTick_Handler() and stop.
12. The Call Stack window will now show main(), Delay() and SysTick_Handler as being present on the Stack:



Call Stack + Locals

Name	Location/Value	Type
SysTick_Handler	0x000001D0	void f()
Delay	0x000001A6	void f(unsigned int)
dlyTicks	0x000001F4	param - unsigned int
curTicks	0x00004268	auto - unsigned int
main	0x00000000	int f()

13. Remove any breakpoints. Click on each breakpoint to remove it or select Ctrl-B, Kill All and then Close.
14. Stop the CPU and exit Debug mode. .
15. Disconnect the USB cable for the next page.

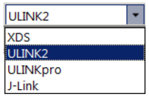
TIP: You can modify a variable value in the Call Stack & Locals window only when the program is stopped.

TIP: This window is only valid when the processor is halted. It does not update while the program is running. Any local variable values are visible only when they are in scope. This is also true for locals in the Watch and Memory windows.

16) Watchpoints: *Conditional Breakpoints* **Watchpoints are only supported with a ULINK or J-Link.**

The MSP432E Cortex-M4 processor has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil documents. Cortex-M3/M4/M7 Watchpoints are not intrusive as they are implemented in CoreSight™ hardware. XDS110 does not currently support Watchpoints in µVision.

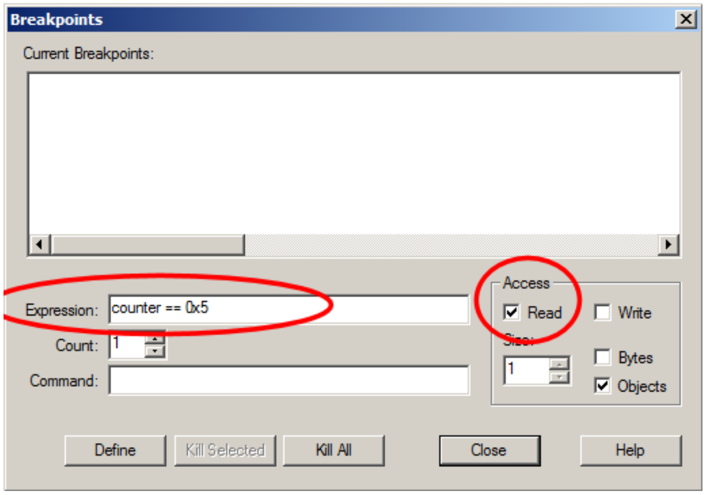
1. Remove J101 jumpers RST, TMS, TCK, TDO and TDI to disconnect the onboard XDS110 debug adapter.
2. Connect a ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* or a J-Link to the CoreSight connector J8.
3. Power the TI board with a USB cable to your PC.
4. In the Target dialog box, select your debug adapter: →


ULINK2
XDS
ULINK2
ULINKpro
J-Link

5. Use the same Blinky configuration as the previous page. Enter Debug mode .

TIP: You can configure Watchpoints while the program is running if you are using any Keil ULINK.

6. We will use the global variable **`counter`** you created in Blinky.c to explore Watchpoints.
7. In the main µVision window, select Debug/Breakpoints or press Ctrl-B.
8. Select Access to Read.
9. In the Expression box enter: "**`counter`** **== 0x5**" without the quotes. This window will display:
10. Click on Define or press Enter and the expression will be accepted as shown below in the Breakpoints window at the bottom of this page:
11. Click on Close.
12. Enter the variable **`counter`** in Watch 1 if it is not already there.
13. Click on RUN .
14. When **`counter`** equals 0x5, the Watchpoint will stop the program. See Watch 1 shown below:
15. Currently you can enter an equality test (= =) as a Watchpoint. Entering no test will stop the processor when the address is accessed with no regard to its value.


Breakpoints

Current Breakpoints:

Expression: counter == 0x5

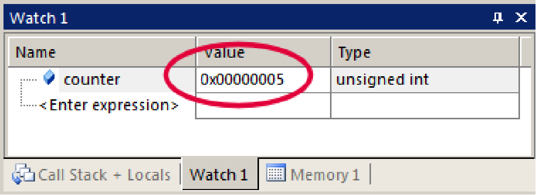
Count: 1

Command:

Access: ☑ Read ☐ Write

Size: 1 ☐ Bytes ☑ Objects

16. To repeat this exercise, change **`counter`** to something other than 0x05 in the Watch window and click on RUN. Or click on RUN a few times to clear past the Watchpoint comparator.
17. Stop the CPU if it is running.
18. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All.
19. Select Close.
20. Exit Debug mode.

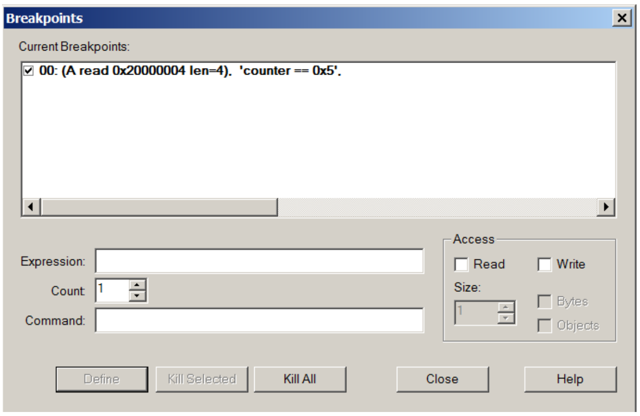

Watch 1

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

TIP: To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint. You must then 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 be used with a Watchpoint. An example is: *((unsigned long *)0x20000004)


Breakpoints

Current Breakpoints:

☑ 00: (A read 0x20000004 len=4), 'counter == 0x5',

Expression:

Count: 1

Command:

Access: ☐ Read ☐ Write

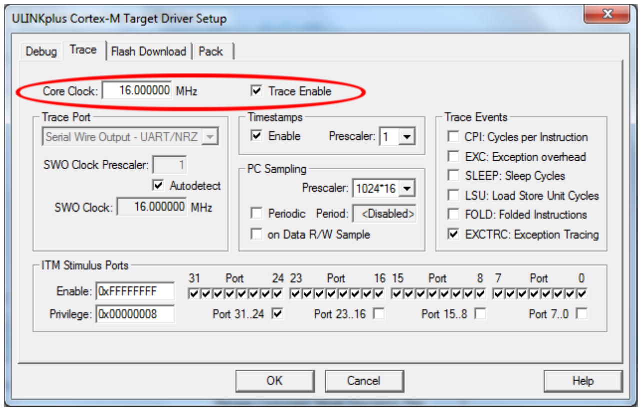
Size: 1 ☐ Bytes ☐ Objects

17) Serial Wire Viewer (SWV): (you need a ULINK2, ULINK*plus*, ULINK*pro* or a J-Link for this page)

Serial Wire Viewer is a data trace found on most Cortex-M3/M4/M7 processors and is a separate technology from the DAP reads and writes used in the Watch and other windows. It can display variables in a graphical format, data reads and writes, exceptions (including interrupts), PC sampler and various hardware counters. The ITM printf feature uses SWV.

Configure SWV: A ULINK2, ULINK*plus*, ULINK*pro* and J-Link each have slightly different but similar menus.

1. Connect a ULINK2, ULINK*pplus*, ULINK*pro* or J-Link to J11 which is a CoreSight 10 pin debug connector.
2. Remove J101 jumpers RST, TMS, TCK, TDO and TDI to disconnect the onboard XDS110 debug adapter.
3. Select your adapter in the Select Target box. This is for ULINK2: ULINK2
4. Click on the Options icon next to the Target box in the main µVision window.
5. Select the Debug tab and then click the Settings box next to ULINK2/ME Cortex Debugger dialog.
6. In the Debug window that opens, set Port: to SW. SWV will not work with JTAG. Only with SWD.
7. Click on the Trace tab to open the Trace window.
8. Select Trace Enable.
9. Set Core Clock: to 16 MHz. With a ULINK2 or J-Link, this ***must*** be set to the exact CPU frequency.
10. Leave everything else at default as shown here:
11. Click on OK twice to return to the µVision main menu. ***The Serial Wire Viewer is now configured in µVision.***
12. Click on File/Save All or
13. Power the board with a USB cable to your PC.
14. Enter Debug mode . Click on RUN .



ULINKplus Cortex-M Target Driver Setup

Core Clock: 16.000000 MHz ☑ Trace Enable

Trace Port: Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 1
☑ Autodetect
SWO Clock: 16.000000 MHz

Timestamps: ☑ Enable Prescaler: 1

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

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

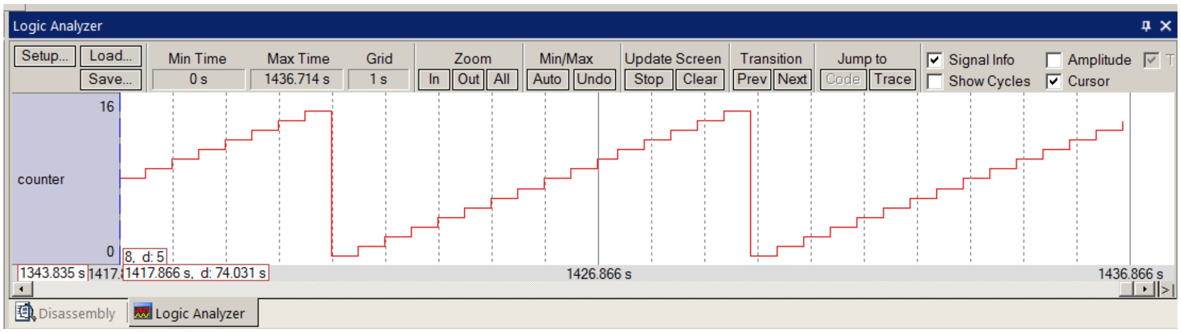
ITM Stimulus Ports:
Enable: 0xFFFFFFFF 31 Port 24 ☑☑☑☑☑☑☑☑ 23 Port 16 ☑☑☑☑☑☑☑☑ 15 Port 8 ☑☑☑☑☑☑☑☑ 7 Port 0 ☑☑☑☑☑☑☑☑
Privilege: 0x00000008 Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

Configure the Logic Analyzer (LA) with counter:

1. In Blinky.c, right click on `counter` and select Add 'counter' to … and select Logic Analyzer. The LA window will open.

TIP: If you are unable to enter a variable in the LA, this usually means SWV is not configured properly. Number one reason is the Core Clock: value is incorrectly set for ULINK2 and J-Link. A ULINK*pro* will work with the wrong clock value but timing values displayed in µVision will be incorrect. This is because ULINK*pro* uses Manchester encoding instead of UART.

2. In the Logic Analyzer (LA), click on Setup:
3. Set the Display Range Max: to 0x10. Close this window.
4. Click on Zoom out to get an appropriate display as shown below:
5. counter is displayed and updated in real-time. You can display up to four variables.
6. Select Cursor and Signal Info to make time and voltage measurements. Stop the Update Screen if needed. This enables you to examine the LA contents and have the program continue running.

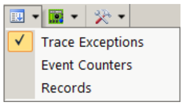


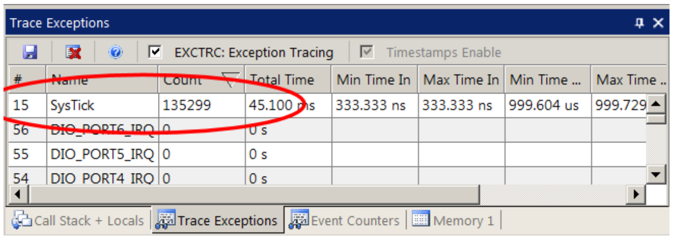
Logic Analyzer

Min Time: 0 s Max Time: 1436.714 s Grid: 1 s
☑ Signal Info ☐ Amplitude ☐ Show Cycles ☑ Cursor

Display Exceptions (including interrupts):

1. Open the Trace Exceptions window by clicking on its tab. If it is not visible, select View/Trace/Trace Exceptions. You can also click on the small arrow beside the Trace icon and select Trace Exceptions.
2. Click on the Count column header and SysTick will come to the top as shown below:
3. Note other interrupts are labelled by name along with their interrupt numbers.
4. This window updates in real-time and uses SWV.

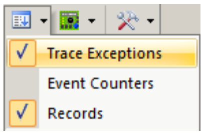




#	Name	Count	Total Time	Min Time In	Max Time In	Min Time ...	Max Time ..
15	SysTick	135299	45.100 ms	333.333 ns	333.333 ns	999.604 us	999.729
56	DIO_PORT6_IRQ	0	0 s				
55	DIO_PORT5_IRQ	0	0 s				
54	DIO_PORT4_IRQ	0	0 s				

> **Note:** The directions below are for using a ULINK2. A Keil ULINK*pro* or a Segger J-Link will require slightly different operations. Additionally, the program must be stopped to display any trace frames and started again to show new ones. The J-Link does not currently display data reads or writes.

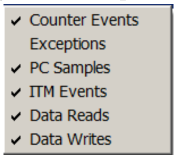
Display Trace Records:

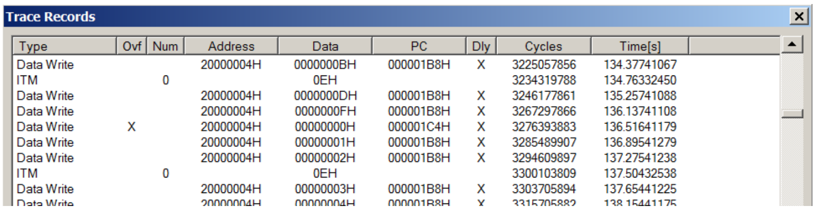


1. Select the small arrow beside the Trace icon and select Records as shown here: (for ULINK2)
2. SysTick Exception 15 will display in the Trace Records window. There are data writes present but they are swamped out by the more numerous SysTick interrupts..

TIP: With ULINK*plus*, ULINK*pro* or J-Link, the program must be stopped to see the Trace Data window.

3. For ULINK2, right click and unselect Exceptions.
4. For ULINK*plus*, ULINK*pro*, select ITM Data Write.
5. Now the Data Writes to counter are displayed (they were swamped out before and are not easy to see.). See below:
6. Note the numerous "x" in DLY and Ovf columns. This indicates data overflow out the single bit SWO port.
7. In the Trace Exceptions window, unselect EXCTRC:.
8. The collection of exceptions including interrupts will not be sent out the SWO port. This will reduce overloading.
9. Double click in the ULINK2 Trace Records to clear it. The "x" are gone indicating much less overloading.





Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			20000004H	0000000BH	000001B8H	X	3225057856	134.37741067
ITM		0		0EH			3234319788	134.76332450
Data Write			20000004H	0000000DH	000001B8H	X	3246177861	135.25741088
Data Write			20000004H	0000000FH	000001B8H	X	3267297866	136.13741108
Data Write	X		20000004H	00000000H	000001C4H	X	3276393883	136.51641179
Data Write			20000004H	00000001H	000001B8H	X	3285489907	136.89541279
Data Write			20000004H	00000002H	000001B8H	X	3294609897	137.27541238
ITM		0		0EH			3300103809	137.50432538
Data Write			20000004H	00000003H	000001B8H	X	3303705894	137.65441225
Data Write			20000004H	00000004H	000001B8H	X	3315705882	138.15441175

10. In the main µVision menu, select Debug/Debug Settings and select the Trace tab.
11. Select on Data R/W Sample. This will add the location of the write in the program. Click Close.
12. Allow "take on new values" and click on RUN.
13. Double click in the Trace Records window to clear it. Now the PC column in Trace Records will now be filled.
14. In the Trace Records window above, the first line means:

    A Data Write occurred to memory 0x2000 0004, with the data value 0x0B by the assembly instruction located at 0x 0000 01B8 at the Cycles or Time indicated.

15. Select Click on STOP . Exit Debug mode. .

TIP: You must stop program execution with a ULINK*plus*, ULINK*pro* or a J-Link to update trace frames. The LA and Exceptions windows still update in real time. ULINK2 and ULINK-ME update the Trace Records continuously.

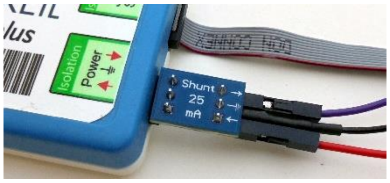
A ULINK*plus*, ULINK*pro* provides the best SWV operation with least amount of overloading.

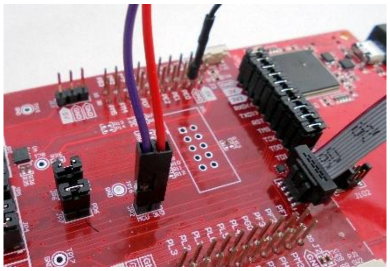
18) Power Measurement with ULINK*plus*: (you need a Keil ULINK*plus* for this page)

The ULINK*plus* debug adapter provides high quality power measurement as well as other information about your system. The data is displayed graphically in the System Analyzer window. Outstanding SWV performance. See www.keil.com/ulinkplus.

Connections to the MSP432E401Y Launchpad board:

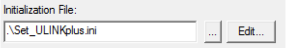
1. There are several small boards that come with ULINK*plus*. Connect the 25 mA shunt to the ULINKplus as shown:
2. Run jumpers from Power In and Out to JP2 MCU 3V3 as shown below. Do not worry about their polarity. If backwards the Current display will be negative and you merely need to swap them. No harm is done.
3. Connect the middle ground pin to a ground pin on the board. There are many to choose from and are labeled GND.
4. Connect the JTAG/SWD cable to J11.
5. Power the board with a USB cable.





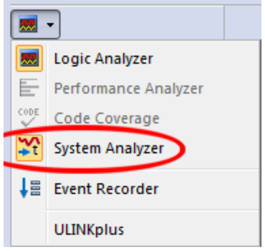
Configuring ULINK*plus*:

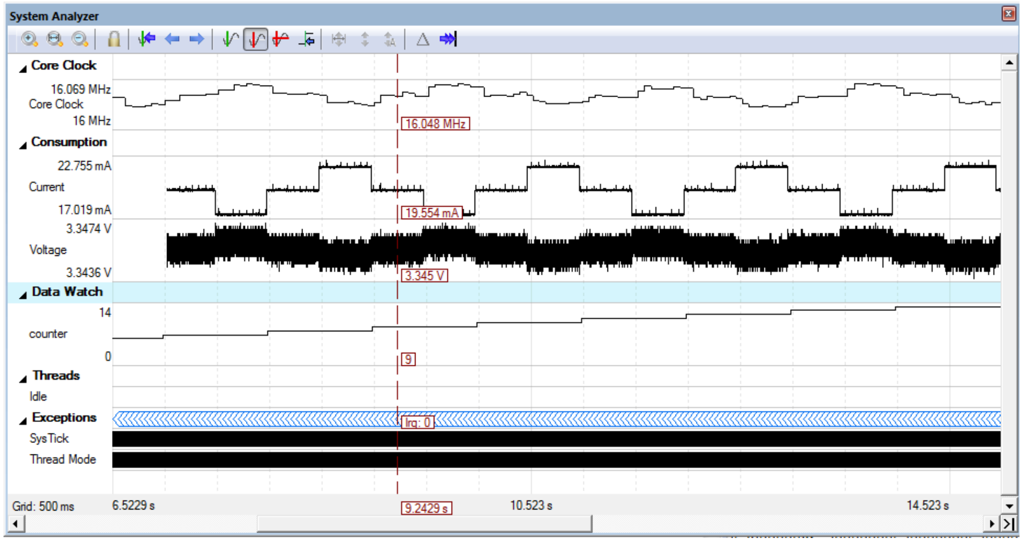
1. Use the same Blinky example from the preceding pages.
2. Click on the Options for Target or ALT-F7 to open the Options for Target menus..
3. Select the Debug tab.
4. In the Initialization File: box enter **Set_ULINKplus.ini**. to configure ULINK*plus*.
5. Click OK to return to the main µVision menu.



Initialization File:
.\Set_ULINKplus.ini ... Edit...

6. Compile the source files. . Enter Debug mode . The Flash will be programmed with a progress bar.
7. Click on RUN. LEDs D1 and D2 will blink in sequence.
8. Open the System Analyzer window:
9. The System Analyzer window opens as shown below:
10. Use Zoom and Jump to End to position the Current waveform.
11. The upper two levels represent one LED on and two LEDs on. Bottom one both LEDs off.
12. Show Show Cursor and various elements will display beside it.
13. Select Show Marker and you can plant a reference point with your mouse and use to to find changes.
14. Select Show Y Axis and this will be added to your cursor.
15. Select Click on STOP . Exit Debug mode .





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

Keil provides RTX, a full feature RTOS as a component of MDK. RTX comes with a BSD license. RTX with source code is provided in all versions of MDK as is all documentation. C:\Keil_v5\ARM\Pack\ARM\CMSIS\x.x.x\CMSIS\RTOS\RTX.

For CMSIS 5, see https://github.com/ARM-software/CMSIS_5

Running the RTX Program:

RTX_Blinky has target options pre-configured for the TI XDS110, ULINK2, ULINK*pro* and J-Link.

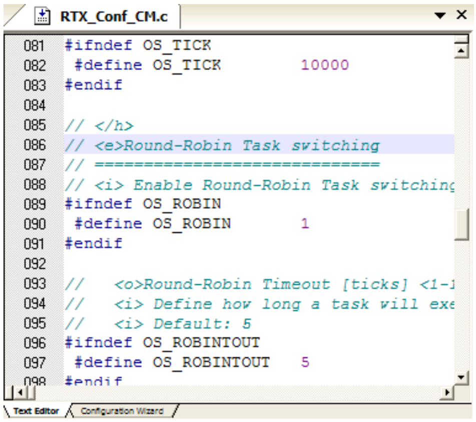
1. If you are using the on-board TI XDS110: make sure all J101 jumpers are populated.
2. If using a ULINK2, ULINK*pro* or a J-Link, connect it to J11.
3. Connect a USB cable to the LaunchPad board and to your PC to power the board.
1. Start µVision and select Project/Open Project.
2. Open C:\00MDK\TI\MSP432E\Examples\RTX_Blinky\Blinky.uvprojx.
3. Select your debug adapter: This is for the TI XDS110: XDS
4. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
5. Enter Debug mode . The Flash will be programmed with a progress bar.
6. Click on RUN.
7. Two LEDs will blink in sequence indicating two of the four waveforms of a stepper motor driver changing.
8. Click on STOP .

TIP: A blog on RTX: https://e2e.ti.com/blogs_/b/msp430blog/archive/2015/06/09/using-arm-keil-rtx-with-msp432-mcus

TIP: If you have trouble with a ULINK or J-Link, remove J101 jumpers RST, TMS, TCK, TDO and TDI to disconnect XDS.

The Configuration Wizard for RTX:

1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open.
2. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available. Do not change anything !
4. See www.keil.com/support/docs/2735.htm for instructions on how to add Configuration Wizard to your own sources.

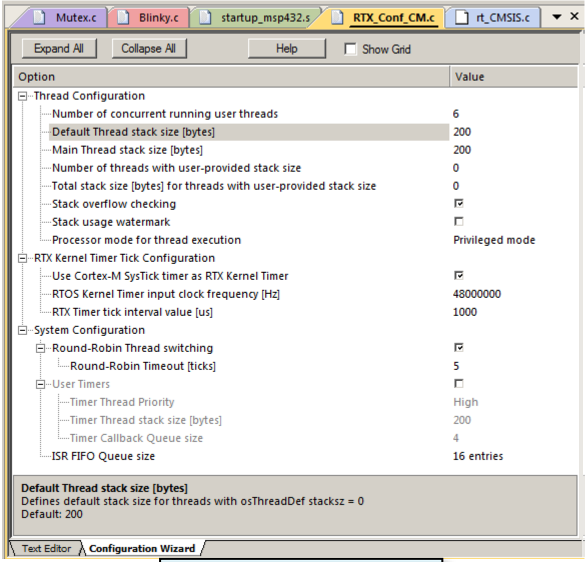


```
#ifndef OS_TICK
 #define OS_TICK        10000
#endif

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

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

Text Editor: Source Code



Option	Value
Thread Configuration	
Number of concurrent running user threads	6
Default Thread stack size [bytes]	200
Main Thread stack size [bytes]	200
Number of threads with user-provided stack size	0
Total stack size [bytes] for threads with user-provided stack size	0
Stack overflow checking	☑
Stack usage watermark	□
Processor mode for thread execution	Privileged mode
RTX Kernel Timer Tick Configuration	
Use Cortex-M SysTick timer as RTX Kernel Timer	☑
RTOS Kernel Timer input clock frequency [Hz]	48000000
RTX Timer tick interval value [us]	1000
System Configuration	
Round-Robin Thread switching	☑
Round-Robin Timeout [ticks]	5
User Timers	□
Timer Thread Priority	High
Timer Thread stack size [bytes]	200
Timer Callback Queue size	4
ISR FIFO Queue size	16 entries

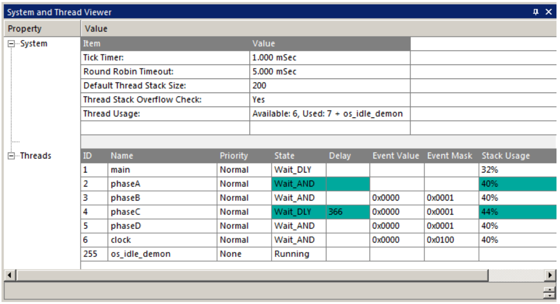
Default Thread stack size [bytes]
Defines default stack size for threads with osThreadDef stacksz = 0
Default: 200

Configuration Wizard

20) RTX Kernel Awareness

Users often want to know the number of the current operating thread and the status of the other threads. This information is usually stored in a structure or memory area by the RTOS. µVision provides two Kernel Awareness windows for RTX.

1. Run RTX_Blinky by clicking on the Run icon.
2. Open Debug/OS Support and select System and Thread Viewer. The window on the right opens up. You might have to grab the window and move it into the center of the screen.
3. Open Debug/OS Support and select Event Viewer. There is probably no data displayed because SWV is not configured.



System and Thread Viewer

Property	Value
System	

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 6, Used: 7 + os_idle_demon

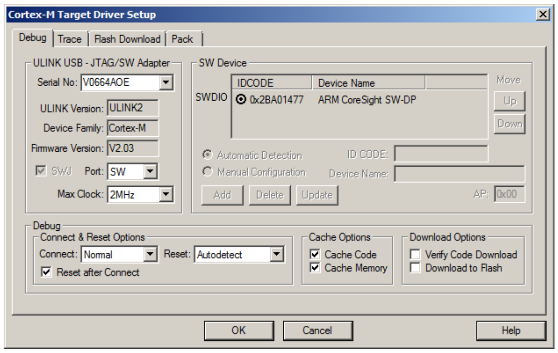
Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	main	Normal	Wait_DLY				32%
2	phaseA	Normal	Wait_AND				40%
3	phaseB	Normal	Wait_AND		0x0000	0x0001	40%
4	phaseC	Normal	Wait_DLY	366	0x0000	0x0001	44%
5	phaseD	Normal	Wait_AND		0x0000	0x0001	40%
6	clock	Normal	Wait_AND		0x0000	0x0100	40%
255	os_idle_demon	None	Running				

Event Viewer: *available only with any Keil ULINK or J-Link:*

We must activate Serial Wire Viewer to get the Event Viewer working. These steps are for ULINK2. Others are different.

15. Stop the CPU and exit Debug mode.
16. Connect a ULINK2, ULINK*plus*, ULINK*pro* or J-Link to J11. Cycle the power to the board.
17. Select your adapter in the Select Target box. ULINK2
18. Click on the Options for Target next to the Target box.
19. Select the Debug tab and then click the Settings box.
20. In the Debug window as shown here, set Port: to SW.
21. Click on the Trace tab to open the Trace window.
22. Select Trace Enable. Set Core Clock: to 16 MHz. With a ULINK2, ULINK*plus* or J-Link, this ***must*** be set to the CPU frequency or SWV will not function.



Cortex-M Target Driver Setup

Debug | Trace | Flash Download | Pack

ULINK USB - JTAG/SW Adapter
Serial No: V0664AOE
ULINK Version: ULINK2
Device Family: Cortex-M
Firmware Version: V2.03
☑ SWJ Port: SW
Max Clock: 2MHz

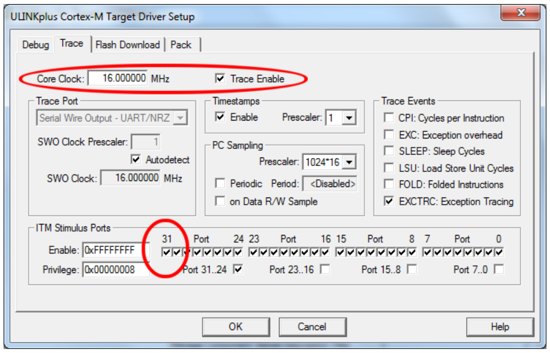
SW Device

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

◉ Automatic Detection ID CODE:
○ Manual Configuration Device Name:
AP: 0x00

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

23. Unselect the Periodic and EXCTRC boxes as shown here:
24. ITM Stimulus Port 31 must be checked. This is the port used to output the kernel awareness information to the Event Viewer. ITM is slightly intrusive from calls made by RTX.
25. Click on OK twice to return to the µVision main menu. ***The Serial Wire Viewer is now configured in µVision.***



ULINKplus Cortex-M Target Driver Setup

Debug | Trace | Flash Download | Pack

Core Clock: 16.000000 MHz ☑ Trace Enable

Trace Port
Serial Wire Output - UART/NRZ
SWO Clock Prescaler: 1
☑ Autodetect
SWO Clock: 16.000000 MHz

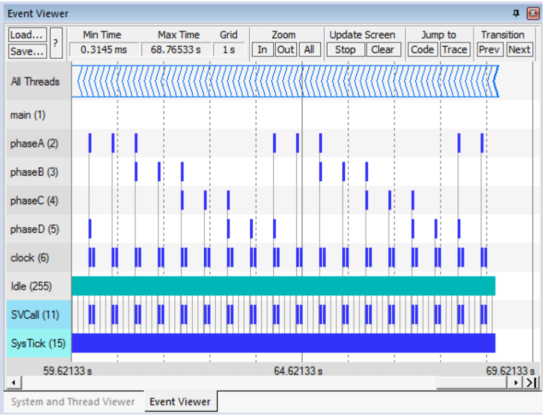
Timestamps
☑ Enable Prescaler: 1

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

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

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

26. Click on File/Save All or .
27. Enter Debug mode . Click on RUN .
28. Select "System and Threads" tab: note the display is updated.
29. Click on the Event Viewer tab.
30. This window displays the threads in a graphical format as shown in the RTX Kernel window below. You probably have to change the Range to about 0.5 seconds by clicking on the ALL and then the + and – icons.
31. Stop the CPU and exit Debug mode.



Event Viewer

Min Time: 0.3145 ms Max Time: 68.76533 s Grid: 1 s

TIP: If Event Viewer doesn't work, open up the Trace Records and confirm there are good ITM 31 frames present. The most probable cause for no valid ITM frames is the Core Clock: is not set correctly

Interrupts: With a ULINK*plus* or ULINK*pro*, exception/interrupt handler times are displayed as shown for SVCall and SysTick. You do not have to stop the program to view this data. Your program runs at nearly full speed. No instrumentation code need be inserted into your source. You will find this feature very useful to see if RTX is behaving as you expect it to and to view results of your configuration modifications.

21) DSP SINE example using ARM CMSIS-DSP Libraries and RTX:

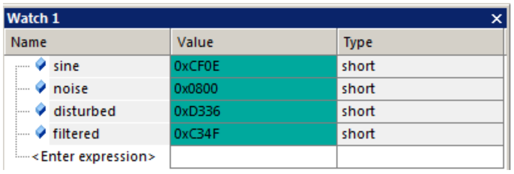
ARM CMSIS-DSP libraries are offered for Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7 processors. DSP libraries are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS\x.x.x\CMSIS\. For documentation see www.keil.com/pack/doc/CMSIS/DSP/html/. For CMSIS 5 see https://github.com/ARM-software/CMSIS_5

This example creates a sine wave to which noise is added, and then the noise is filtered out leaving the original sine wave.

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

a) Running the DSP Example:

1. Stop the CPU and exit Debug mode. Select Project/Open Project.
2. Open the project file sine.uvprojx in C:\00MDK\TI\MSP432E\Examples\DSP\
3. Select the debugger you are using in the in the Select Target box: This is for the XDS110: XDS
4. For the on-board XDS110-ET, populate all positions of jumper J101.
5. Connect an external debugger if used to J11, select it in the Select target box.
6. Remove J101 jumpers RST, TMS, TCK, TDO and TDI. This action disconnects the XDS110 from the processor.
7. Build the files. There will be no errors or warnings.
8. Enter Debug mode by clicking on the Debug icon. The Flash will be programed.
9. Click on the RUN icon.
10. Open Watch 1 by selecting View/Watch Windows/Watch 1.
11. Four global variables will be displayed in Watch 1 as shown here: If these variables are changing, the program is working properly.
12. Stop the CPU and exit Debug mode.



Watch 1

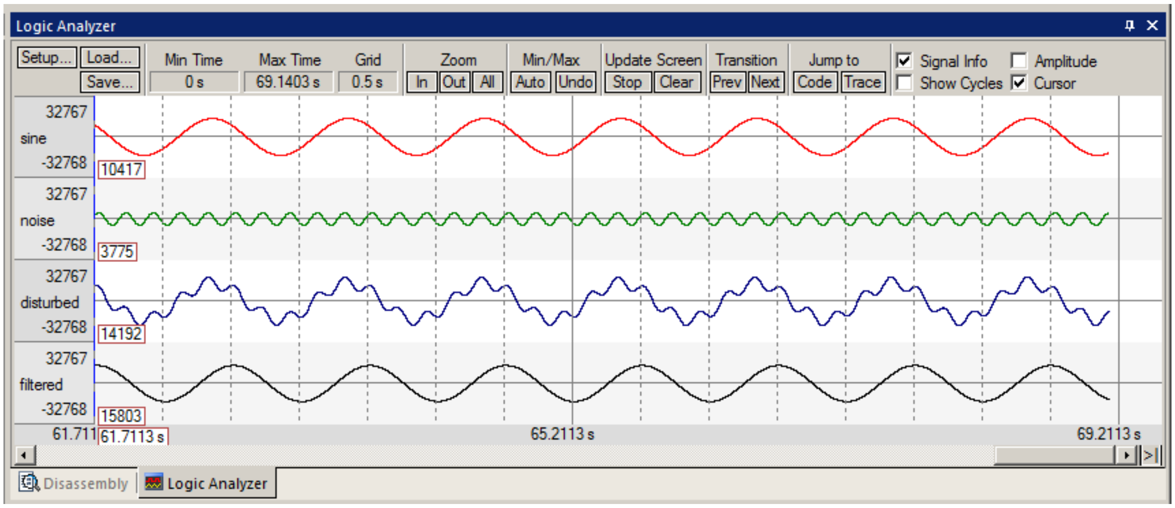
Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short
<Enter expression>		

b) Serial Wire Viewer (SWV) (requires a Keil ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* or J-Link):

If you are using a ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* or a J-Link, you will see the screen below. This has been configured by default in this project. You can open this window by selecting View/Analysis Windows/Logic Analyzer or . The Cortex-M3, Cortex-M4 and Cortex-M7 processors have Serial Wire Viewer (SWV).

The values of the global variables shown above will be displayed graphically in the Logic Analyzer as shown below. There are many other SWV features you can use to debug your programs. Currently the TI XDS110 does not support SWV.

TIP: If there are distortions in the waveforms, it is usually because of SWO overload. Solutions are to use a ULINK*plus* or ULINK*pro* or turn off one of the four waveforms. Limit the SWV features configured to those you really need.



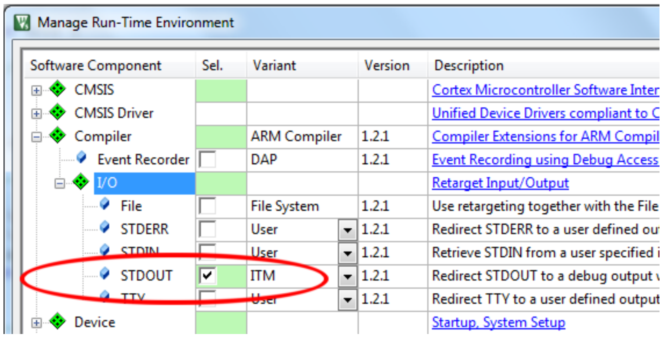
22) printf with ITM (Instrumentation Trace Macrocell): Uses SWV and ITM 0:

The DSP example provids some printf statements in it to indicate initialization progress. µVision provides two methods of providing printf support: One uses DAP read/write and will work with XDS110 and all ULINKs. The second uses SWV and works only with any ULINK or J-Link. **XDS110 does not currently support SWV. Use the DAP method on next page.**

The DSP example is pre-configured with ITM printf. This page is provided for reference so you can add it to any program.

1. Stop the program if it is running and exit Debug mode.

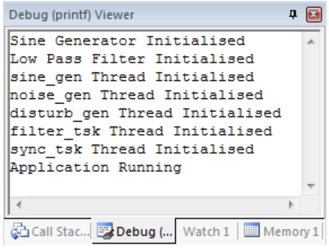
1. Open the Manage Run-Time Environment utility. This window opens: Expand Compiler…I/O as shown.
2. Select STDOUT and ITM:
3. All the blocks should be green. If not, click on the Resolve button.
4. Click OK to close this window.
5. The file retarget_io.c will be added to your project in the project window under the Compiler group.
6. In DirtyFilter.c, add this line: #include <stdio.h>.
7. Add your standard printf statements. In the DSP example, they are already included.



Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Inter
CMSIS Driver				Unified Device Drivers compliant to C
Compiler		ARM Compiler	1.2.1	Compiler Extensions for ARM Compil
Event Recorder	□	DAP	1.2.1	Event Recording using Debug Access
I/O				Retarget Input/Output
File	□	File System	1.2.1	Use retargeting together with the File
STDERR	□	User	1.2.1	Redirect STDERR to a user defined ou
STDIN	□	User	1.2.1	Retrieve STDIN from a user specified i
STDOUT	☑	ITM	1.2.1	Redirect STDOUT to a debug output v
TTY	□	User	1.2.1	Redirect TTY to a user defined output
Device				Startup, System Setup

8. Select File/Save All or click .
9. Rebuild the source files .
10. Enter Debug mode . Click on RUN .
11. Select View/Serial Windows and select Debug (printf) Viewer.
12. The values of counter is displayed as seen here:



```
Sine Generator Initialised
Low Pass Filter Initialised
sine_gen Thread Initialised
noise_gen Thread Initialised
disturb_gen Thread Initialised
filter_tsk Thread Initialised
sync_tsk Thread Initialised
Application Running
```

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

See ITMLOG command: www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm

Setting RAM for printf:

If your program doesn't work, you might have to do one or more of these items when implementing printf using ITM:

1. Increase RTX Thread Stack size. This is done in RTX_Conf_CM.C. Use the Configuration Wizard. Try 300 bytes.
2. If you do not have MicroLIB selected, you must add some heap in startup_stm32f401xc.s or the correct startup.s file for your processor. Try adding a 200 byte heap. MicroLIB results in smaller executable size.

Obtaining a character typed into the Debug printf Viewer window:

It is possible for your program to do this with the function ITM_ReceiveChar found in core.CM7.h.

See https://www.keil.com/pack/doc/CMSIS/Core/html/group___i_t_m___debug__gr.html.

A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer utility.

TIP: µVision icon meanings are found here: www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm

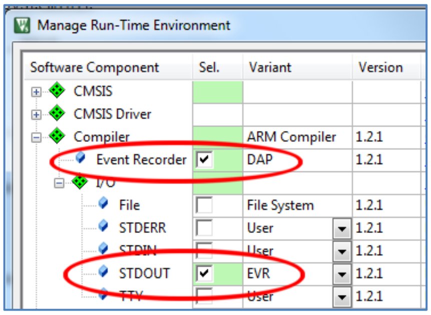
23) printf for XDS110 (no SWV): Uses Event Recorder and CoreSight DAP:

Event Recorder is a new Vision feature that can be used to instrument your code. Keil RTX5 and Keil Middleware is already instrumented with Event Recorder. Event Recorder can provide a printf utility using the DAP read/write abilities of the Debug Access Port. A UART is not used. This can be used with TI XDS-110 emulator as it does not yet support SWV as well as any ULINK. This is the same technology used in Watch, Memory and Peripheral windows. It does not use SWV. This page modifies the DSP example that uses SWV to allow the use of DAP.

1. Stop the program if it is running and exit Debug mode.

Configure Event Recorder:

2. Open the Manage Run-Time Environment utility. This window opens:
3. Expand Compiler and I/O as shown.
4. Select Event Recorder and STDOUT and EVR as shown:
5. All the blocks should be green. If not, click on the Resolve button.
6. Click OK to close this window.
7. retarget_io.c and EventRecorder.c will be added to your project under the Compiler group in the Project window.
8. Right click near the top of DirtyFilter.c, and select Insert "#include" and select #include "EventRecorder.h".
9. In DirtyFilter.c add #include "stdio.h" near the top of the file. The DSP example already has this added.
10. At the beginning of the main() function, add this line: EventRecorderInitialize (EventRecordAll, 1);



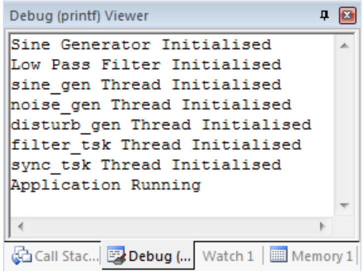
Software Component	Sel.	Variant	Version
CMSIS			
CMSIS Driver			
Compiler		ARM Compiler	1.2.1
Event Recorder	☑	DAP	1.2.1
I/O			
File	□	File System	1.2.1
STDERR	□	User	1.2.1
STDIN	□	User	1.2.1
STDOUT	☑	EVR	1.2.1
TTY	□	User	1.2.1

Add a printf statement to DirtyFilter.c:

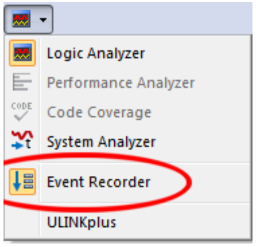
1. The DSP example already has printf statements included. You can your own if you like.
2. Select File/Save All or click .

Build and RUN the Blinky program and view printf:

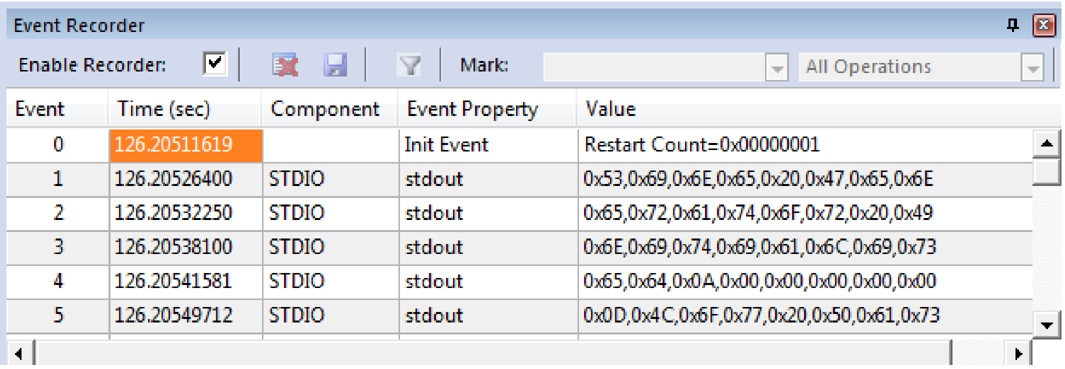
1. Rebuild the source files .
2. Enter Debug mode . Click on RUN .
3. Select View/Serial Windows and select Debug (printf) Viewer.
4. The values of counter is displayed as seen here:
5. Open the Event Recorder window:
6. Information about the printf statements are displayed as shown below:
7. You can annotate your own sources and display events. See www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr.htm



```
Sine Generator Initialised
Low Pass Filter Initialised
sine_gen Thread Initialised
noise_gen Thread Initialised
disturb_gen Thread Initialised
filter_tsk Thread Initialised
sync_tsk Thread Initialised
Application Running
```



TIP: Keil Middleware and RTX5 are annotated using Event Recorder.



Enable Recorder: ☑

Event	Time (sec)	Component	Event Property	Value
0	126.20511619		Init Event	Restart Count=0x00000001
1	126.20526400	STDIO	stdout	0x53,0x69,0x6E,0x65,0x20,0x47,0x65,0x6E
2	126.20532250	STDIO	stdout	0x65,0x72,0x61,0x74,0x6F,0x72,0x20,0x49
3	126.20538100	STDIO	stdout	0x6E,0x69,0x74,0x69,0x61,0x6C,0x69,0x73
4	126.20541581	STDIO	stdout	0x65,0x64,0x0A,0x00,0x00,0x00,0x00,0x00
5	126.20549712	STDIO	stdout	0x0D,0x4C,0x6F,0x77,0x20,0x50,0x61,0x73

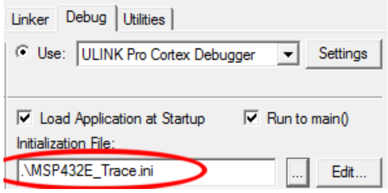
24) Running the ETM example program using the TI MSP432E401Y board:

Now we will connect the Keil MDK development system using the MSP432E401Y board. A Keil ULINK*pro* must be used to collect the ETM instruction trace frames. The MSP432E401 board does not have the required 20 pin CoreSight connector. You will need to create an adapter to add the 5 signals of ETM (4 data + clock) to headers on the board.

1. Connect a Keil ULINK*pro* to the MSP board with a custom adapter. Power the board with a USB cable.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open Blinky.uvprojx in C:\00MDK\TI\MSP432E\Examples\Blinky\
4. Select ULINKpro: in the Target Options box:
ULINKpro

Enter the Trace Initialization file:

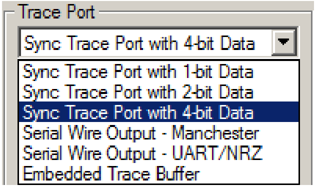
1. Select Options for Target or ALT-F7.
2. Click on the Debug tab.
3. In the Initialization File: area, click on the browse icon and select MSP432E_Trace.ini: It is included with this example in its root folder. It configures GPIO ports (PF0 – PF4) to be the ETM 4 bit traceport.
4. Click on Open and it will be entered in the Initialization box:
5. If you click on the Edit icon, MSP432E_Trace.ini will be opened with the other source files. You can then view it. Do not modify it at this time.



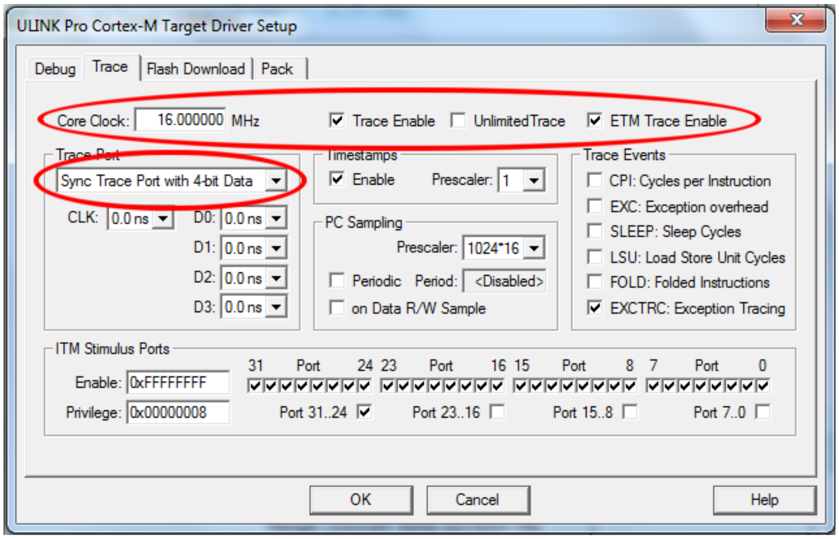
● Use: ULINK Pro Cortex Debugger
☑ Load Application at Startup ☑ Run to main()
Initialization File: .\MSP432E_Trace.ini

Configuring ETM Trace (and SWV) using the 4 bit Trace port:

1. In the Options for Target window under the Debug tab, click on the right side Settings: icon.
2. Click on the Trace tab. The window at the bottom of this page is displayed.
3. Select Trace Enable. This selects SWV data trace.
4. Set Core Clock: to 16 MHz. ULINK*pro* uses this to calculate timings in various windows.
5. In Trace Port, select Sync Trace Port with 4-bit data. It is best to use the widest size which is 4 bits. It might be useful to use 2 bit trace if you need a pin for other uses.
6. Select ETM Trace Enable. This activates ETM Trace.
7. The last 1 million or so trace frames will be saved with the older ones being overwritten. If you need to save them all, select Unlimited Trace. Your hard drive space will determine how many are saved. ULINK*pro* is a streaming trace.
8. Everything else in this window relates to Serial Wire Viewer. Leave everything else at default as shown below. Only ITM 31 and 0 are used by µVision at this time for the RTOS Event Viewer and printf respectively.
9. Click on OK twice to return to the main µVision menu.
10. ETM and SWV are now configured through the 4 bit Trace Port.
11. Select File/Save All.


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

TIP: The traceport is multiplexed on GPIO P0 through P4 with other peripheral outputs. This means if ETM is used, these peripherals can't be used. You should check your data sheet to prepare for this if ETM is required.



Core Clock: 16.000000 MHz ☑ Trace Enable □ Unlimited Trace ☑ ETM Trace Enable

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

Timestamps: ☑ Enable Prescaler: 1

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

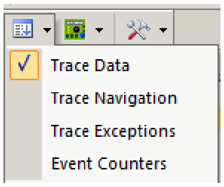
Trace Events:
□ CPI: Cycles per Instruction
□ EXC: Exception overhead
□ SLEEP: Sleep Cycles
□ LSU: Load Store Unit Cycles
□ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

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

Enter Debug mode and RUN the Program:

1. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Progress will be indicated in the Output Window. Select OK if the Evaluation Mode box appears.
2. **DO NOT** click on the RUN icon. The program will automatically run to the start of main() and stop. If you do click on RUN, exit and reenter Debug to restart the program.

Opening the Trace Data Window and Confirming ETM is Working:

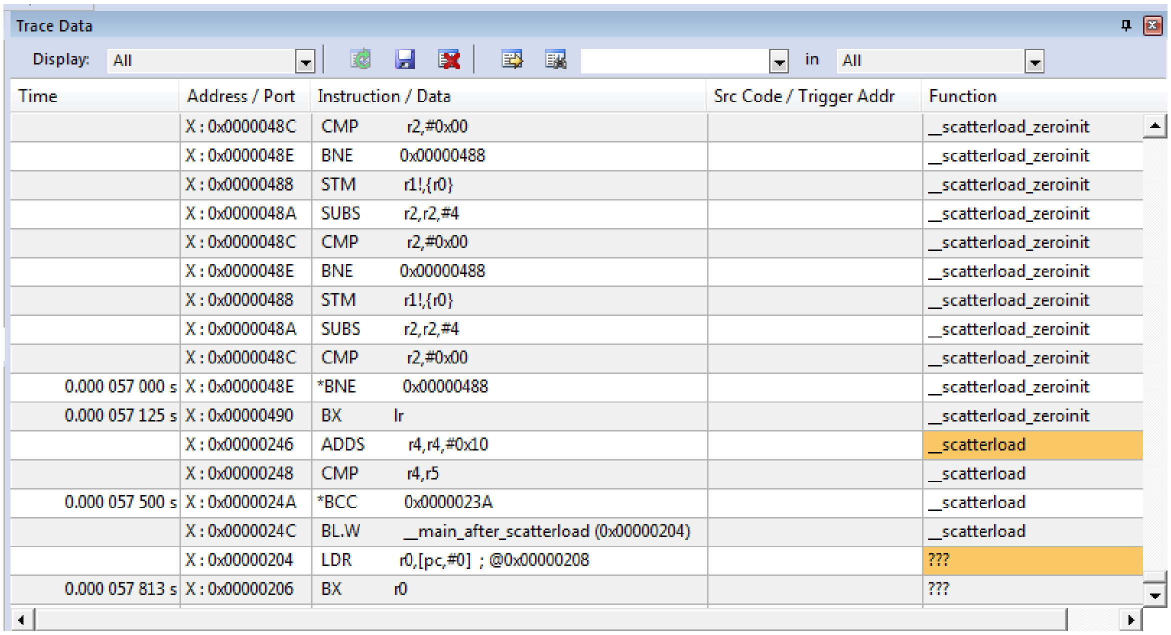


1. Open the Trace Data window it by clicking on the small arrow beside the Trace icon as shown:
2. The Trace Data window will look similar to the one below if it is working correctly:
3. If you do see trace frames in this window, you must fix this before you can continue. Check your settings and conections. ETM is easy to get working.

TIP: You must stop the program to view the trace frames in the Trace Data window.

Trace Data Window Description:

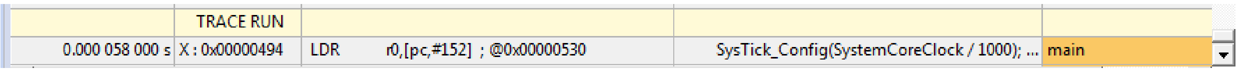
1. The frames above the blue line (in this case) are executed instructions with source if available.
2. The Function column displays the name of its function with the first instance highlighted in orange.
3. The last instruction executed is the BX r0. This is displayed at the end of the Trace Data window. This is the branch from the Cortex-M4 initialization code to the first instruction in the main() function.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x0000048C	CMP r2,#0x00		__scatterload_zeroinit
	X : 0x0000048E	BNE 0x00000488		__scatterload_zeroinit
	X : 0x00000488	STM r1!,{r0}		__scatterload_zeroinit
	X : 0x0000048A	SUBS r2,r2,#4		__scatterload_zeroinit
	X : 0x0000048C	CMP r2,#0x00		__scatterload_zeroinit
	X : 0x0000048E	BNE 0x00000488		__scatterload_zeroinit
	X : 0x00000488	STM r1!,{r0}		__scatterload_zeroinit
	X : 0x0000048A	SUBS r2,r2,#4		__scatterload_zeroinit
	X : 0x0000048C	CMP r2,#0x00		__scatterload_zeroinit
0.000 057 000 s	X : 0x0000048E	*BNE 0x00000488		__scatterload_zeroinit
0.000 057 125 s	X : 0x00000490	BX lr		__scatterload_zeroinit
	X : 0x00000246	ADDS r4,r4,#0x10		__scatterload
	X : 0x00000248	CMP r4,r5		__scatterload
0.000 057 500 s	X : 0x0000024A	*BCC 0x0000023A		__scatterload
	X : 0x0000024C	BL.W __main_after_scatterload (0x00000204)		__scatterload
	X : 0x00000204	LDR r0,[pc,#0] ; @0x00000208		???
0.000 057 813 s	X : 0x00000206	BX r0		???

Single-Step:

1. In the Register window, the PC will display the location of the next instruction to be executed (0x0494 in my case).
2. Click anywhere in the Disassembly window to bring it into focus. This ensures steps are at the instruction level.
3. The instruction at 0x0494 in the Disassembly window is LDR r0,[pc,#152] and is the next instruction to be executed.
4. It is the first instruction in the SysTick timer configuration statement near line 75 in Blinky.c.
5. Click on Single Step once.
6. The instruction LDR instruction is executed and displayed in the Trace Data window as shown here:

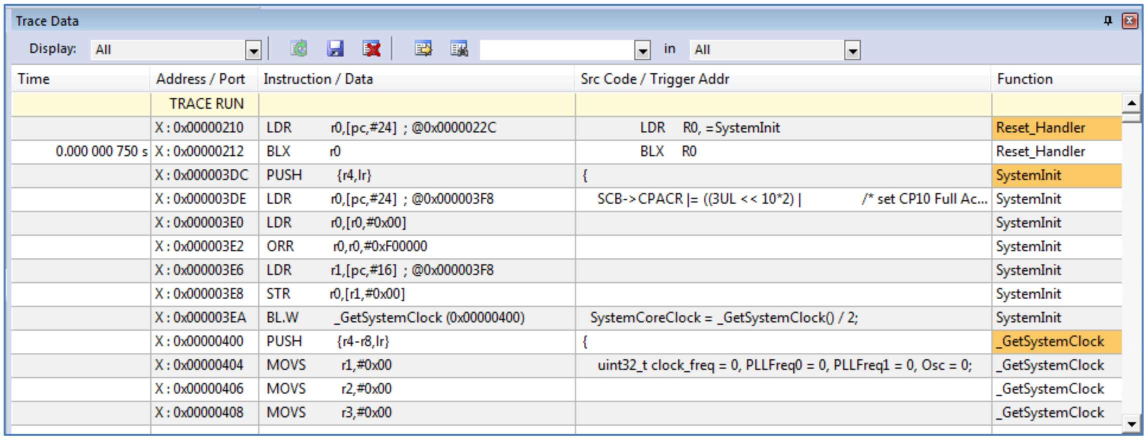


	TRACE RUN			
0.000 058 000 s	X : 0x00000494	LDR r0,[pc,#152] ; @0x00000530	SysTick_Config(SystemCoreClock / 1000); ...	main

25) Viewing ETM Frames starting at RESET:

View the RESET Sequence:

1. Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is an LDR instruction in the RESET_Handler function as shown below:
2. If you use the Memory window to look at location 0x04, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0000_00210 + 1 = 0x0211 (+1 says it is a Thumb® instruction). These first instructions after RESET are shown below: Note the source information is displayed including which function they belong to. The first occurrence in a function is highlighted in orange to make the start of functions easier to find.
3. Any source code associated with an instruction is displayed in the Src Code column.
4. If you double-click on any line, this will be highlighted in both the Disassembly and the relevant source window.



Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	TRACE RUN			
	X : 0x00000210	LDR r0,[pc,#24] ; @0x0000022C	LDR R0, =SystemInit	Reset_Handler
0.000 000 750 s	X : 0x00000212	BLX r0	BLX R0	Reset_Handler
	X : 0x000003DC	PUSH {r4,lr}	{	SystemInit
	X : 0x000003DE	LDR r0,[pc,#24] ; @0x000003F8	SCB->CPACR |= ((3UL << 10*2) | /* set CP10 Full Ac...	SystemInit
	X : 0x000003E0	LDR r0,[r0,#0x00]		SystemInit
	X : 0x000003E2	ORR r0,r0,#0xF00000		SystemInit
	X : 0x000003E6	LDR r1,[pc,#16] ; @0x000003F8		SystemInit
	X : 0x000003E8	STR r0,[r1,#0x00]		SystemInit
	X : 0x000003EA	BL.W _GetSystemClock (0x00000400)	SystemCoreClock = _GetSystemClock() / 2;	SystemInit
	X : 0x00000400	PUSH {r4-r8,lr}	{	_GetSystemClock
	X : 0x00000404	MOVS r1,#0x00	uint32_t clock_freq = 0, PLLFreq0 = 0, PLLFreq1 = 0, Osc = 0;	_GetSystemClock
	X : 0x00000406	MOVS r2,#0x00		_GetSystemClock
	X : 0x00000408	MOVS r3,#0x00		_GetSystemClock

TIP: If you unselect Run to main() in the Debug tab, no instructions will be executed when you enter Debug mode. The PC will equal 0x0210. You can run or single-step from that point and this will be recorded in the Trace Data window.

Be aware that sometimes initialization code produced by a compiler might not single step without errors. In such cases, certain sections in the initialization code must be executed in a contiguous fashion.

LEDS D3 and D4: Note when you enter Debug mode, D3 and D4 illuminate. This is because two of the ETM data lines are shared with these two LEDs. In this case it does not appear to affect the ETM trace operation. Normally, at higher clock speeds this can cause problems and hardware such as this must be disconnected from the shared pins.

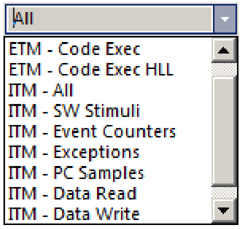
You are not able to use any peripherals shared on GPIO P0 through P4 if ETM is being used. It is possible to use a 1 or 2 bit traceport but throughput can be significantly reduced.

26) Finding the Trace Frames you are looking for:

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.

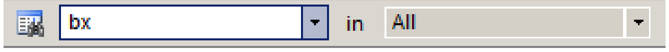
Trace Filters:

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 µVision will allow more precise filters to be selected.



Find a Trace Record:

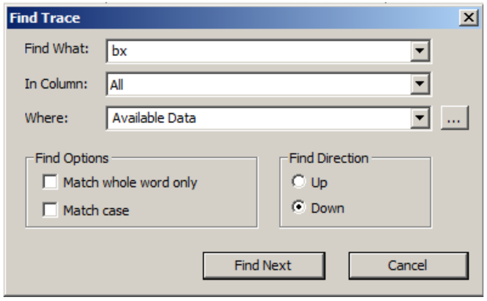
In the Find a Trace Record box, enter **bx** as shown here:



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

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.

TIP: Or you can press Enter to go to the next occurrence of the search term.



Find What: bx

In Column: All

Where: Available Data

Find Options

☐ Match whole word only

☐ Match case

Find Direction

○ Up

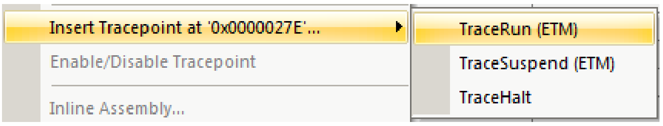
● Down

Trace Triggering:

You can configure two sets of trace triggers. µVision has three types of trace trigger commands as listed here: They are set and unset in a source or Disassembly window or in the µVision Command window.

1. **TraceRun:** Starts ETM trace collection when encountered.
2. **TraceSuspend:** Stops ETM trace collection when encountered. TraceRun has to have first been set and encountered to start the trace collection for this trigger to have an effect.
3. **TraceHalt:** Stops ETM trace, SWV and ITM. Trace collection can be resumed only with a STOP/RUN sequence. TraceStart will not restart it.

They are selected from the menu shown here:
This menu is accessed by right-clicking on a valid assembly instruction in the Disassemble window or a C source line.


Insert Tracepoint at '0x0000027E'...
Enable/Disable Tracepoint
Inline Assembly...
TraceRun (ETM)
TraceSuspend (ETM)
TraceHalt

TIP: These trace commands have no effect on SWV or ITM. TraceRUN starts the ETM trace and TraceSuspend and TraceHalt stops it.

How it works:

When a TraceRun is encountered on an instruction while the program is running, ULINK*pro* will start collecting trace records. When a TraceSuspend is encountered, 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.

Sometimes there is some skid past the trigger point which is normal.

Trace Commands: *This part is important in order to effectively manage Tracepoints.*

There are a series of Trace Commands you can enter in the Command window while in Debug mode.

See www.keil.com/support/man/docs/uv4/uv4_debug_commands.htm

TL - Trace List: list all tracepoints.

TK - Trace Kill: tk* kills all tracepoints or tk *number* only a specified one i.e. tk 2.

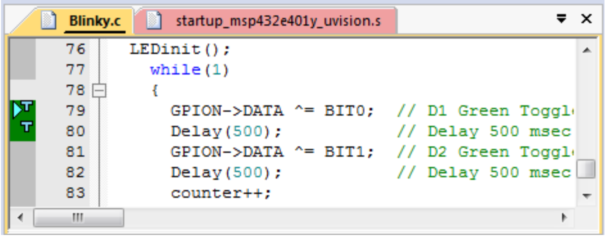
TIP: TK* is very useful for deleting tracepoints when you can't find them in the source or Disassembly windows.

TL is useful for finding any tracepoints set. Results are displayed in the Command window.

27) Setting Trace Triggers: They are called TracePoints:

Setup the Trace Triggers to capture the statement GPION->DATA ^= BIT0;

1. In the file Blinky.c, right click on the grey (or green) block opposite near line 79 as shown here:
2. Select Insert Tracepoint at line 79 and select **TraceRun (ETM).** A cyan T will appear as shown below:
3. Right-click on the gray (or green) block opposite line 80 (return 0;) as shown here:
4. Select Insert Tracepoint at line 80 and select **TraceSuspend (ETM).** A cyan T will appear as shown:
5. Clear the Trace Data window for convenience. This is an optional step.
6. Click RUN and after a few seconds click STOP.
7. Filter exceptions out by selecting ETM – Code Exec in the Display in the Trace Data window: Display: ETM - Code Exec
8. Examine the Trace Data window as shown below:



Blinky.c | startup_msp432e401y_uvision.s

```
76   LEDinit();
77     while(1)
78     {
79       GPION->DATA ^= BIT0;  // D1 Green Toggl
80       Delay(500);           // Delay 500 msec
81       GPION->DATA ^= BIT1;  // D2 Green Toggl
82       Delay(500);           // Delay 500 msec
83       counter++;
```

You can see below where the trace started on 0x04EA and stopped on 0x04F6: All other frames are discarded.

9. In the Command window, enter TL and press Enter. The two Trace points are displayed.
10. **Enter TK* in the Command window and press Enter to delete all Tracepoints.**

Trace Skid:

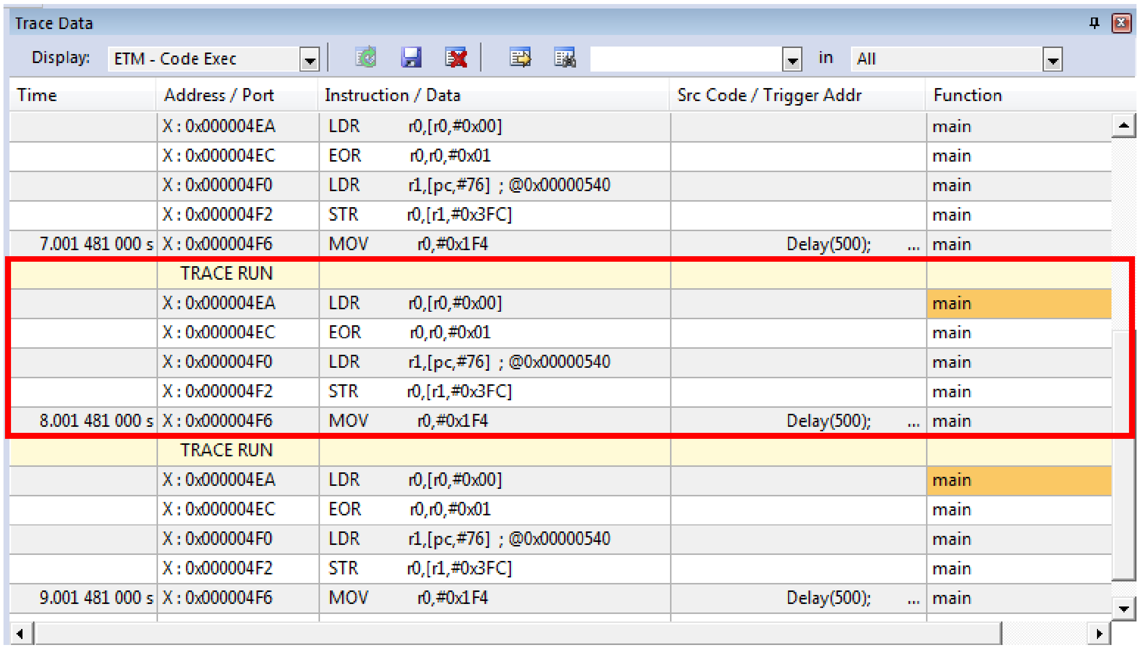
The trace triggers use the same CoreSight hardware as the Watchpoints. This means that it is possible a program counter skid might happen. The program might not start or stop on the exact location you set the trigger to.

You might have to adjust the trigger point location to minimize this effect.

This is because of the nature of the comparators in the CoreSight module and it is normal behavior.

In this case, the first instruction of the statement GPION->DATA ^= BIT0 was not recorded. It is a LDD instruction.

Click on the lines in the trace data window to see these lines in the Disassembly and Blinky.c files.



Trace Data

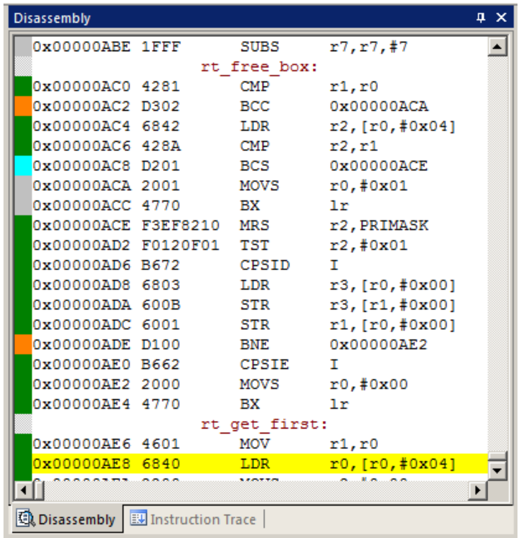
Display: ETM - Code Exec in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X : 0x000004EA	LDR r0,[r0,#0x00]		main
	X : 0x000004EC	EOR r0,r0,#0x01		main
	X : 0x000004F0	LDR r1,[pc,#76] ; @0x00000540		main
	X : 0x000004F2	STR r0,[r1,#0x3FC]		main
7.001 481 000 s	X : 0x000004F6	MOV r0,#0x1F4	Delay(500); ...	main
	TRACE RUN			
	X : 0x000004EA	LDR r0,[r0,#0x00]		main
	X : 0x000004EC	EOR r0,r0,#0x01		main
	X : 0x000004F0	LDR r1,[pc,#76] ; @0x00000540		main
	X : 0x000004F2	STR r0,[r1,#0x3FC]		main
8.001 481 000 s	X : 0x000004F6	MOV r0,#0x1F4	Delay(500); ...	main
	TRACE RUN			
	X : 0x000004EA	LDR r0,[r0,#0x00]		main
	X : 0x000004EC	EOR r0,r0,#0x01		main
	X : 0x000004F0	LDR r1,[pc,#76] ; @0x00000540		main
	X : 0x000004F2	STR r0,[r1,#0x3FC]		main
9.001 481 000 s	X : 0x000004F6	MOV r0,#0x1F4	Delay(500); ...	main

28) Code Coverage:

1. Click on the RESET icon.
2. Click on the RUN icon. After a few seconds, stop the program.
3. Examine the Disassembly and Blinky.c windows. Scroll and notice the different color blocks in the left margin:
4. 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.



Disassembly

```
0x00000ABE 1FFF      SUBS     r7,r7,#7
               rt_free_box:
0x00000AC0 4281      CMP      r1,r0
0x00000AC2 D302      BCC      0x00000ACA
0x00000AC4 6842      LDR      r2,[r0,#0x04]
0x00000AC6 428A      CMP      r2,r1
0x00000AC8 D201      BCS      0x00000ACE
0x00000ACA 2001      MOVS     r0,#0x01
0x00000ACC 4770      BX       lr
0x00000ACE F3EF8210  MRS      r2,PRIMASK
0x00000AD2 F0120F01  TST      r2,#0x01
0x00000AD6 B672      CPSID    I
0x00000AD8 6803      LDR      r3,[r0,#0x00]
0x00000ADA 600B      STR      r3,[r1,#0x00]
0x00000ADC 6001      STR      r1,[r0,#0x00]
0x00000ADE D100      BNE      0x00000AE2
0x00000AE0 B662      CPSIE    I
0x00000AE2 2000      MOVS     r0,#0x00
0x00000AE4 4770      BX       lr
               rt_get_first:
0x00000AE6 4601      MOV      r1,r0
0x00000AE8 6840      LDR      r0,[r0,#0x04]
```

🟩 1. Green: This assembly instruction was executed.
⬜ 2. Gray: This assembly instruction was not executed.
🟧 3. Orange: A Branch is never taken.
🟦 4. Cyan: A Branch is always taken.
▨ 5. Light Gray: There is no assembly instruction here.
🔴 6. RED: A Breakpoint is set here.
⇨ 7. This points to the next instruction to be executed.

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

TIP: Code Coverage is visible in both the Disassembly and source code windows. Click on a line in one window and this place will be matched in the other window.

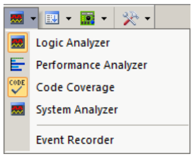
In the window above, why was 0x0000_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

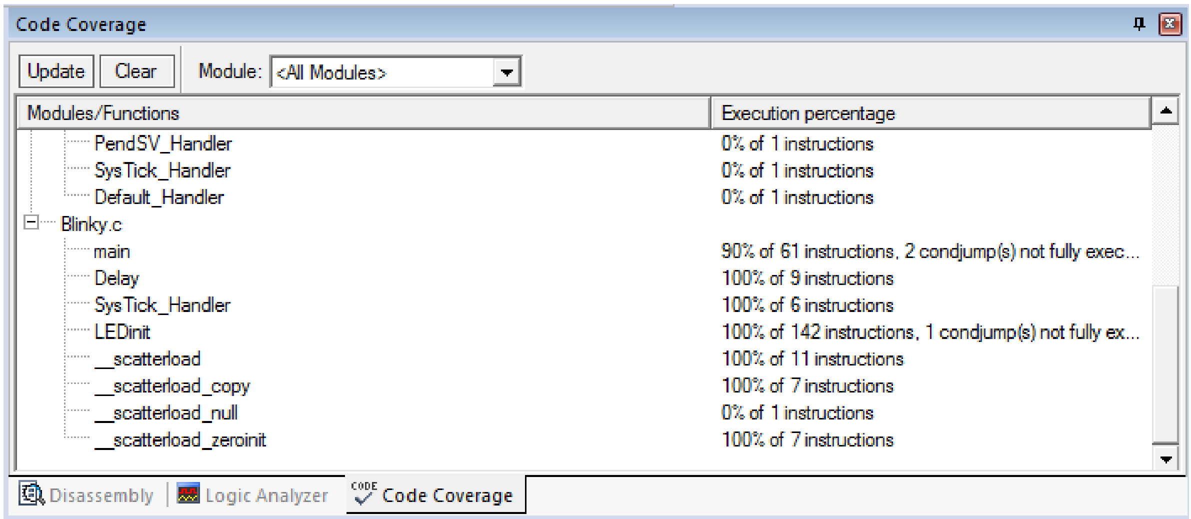
Code Coverage tells what assembly instructions were executed. It is important to ensure all assembly code produced by the compiler is executed and tested. You do not want a bug or an unplanned circumstance to cause a sequence of untested instructions to be executed. The result could be catastrophic as unexecuted instructions have not been tested. Some agencies such as the US FDA and FAA require Code Coverage for certification. This is provided in MDK µVision using ULINK*pro*.

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

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

A Code Coverage window is available as shown below. Click on Update to refresh it. This window is available in View/Analysis/Code Coverage or by selecting the Analysis icon as shown here:
The next page describes how you can save Code Coverage information to a file.





Code Coverage

Update Clear Module: <All Modules>

Modules/Functions	Execution percentage
PendSV_Handler	0% of 1 instructions
SysTick_Handler	0% of 1 instructions
Default_Handler	0% of 1 instructions
Blinky.c	
main	90% of 61 instructions, 2 condjump(s) not fully exec...
Delay	100% of 9 instructions
SysTick_Handler	100% of 6 instructions
LEDinit	100% of 142 instructions, 1 condjump(s) not fully ex...
__scatterload	100% of 11 instructions
__scatterload_copy	100% of 7 instructions
__scatterload_null	0% of 1 instructions
__scatterload_zeroinit	100% of 7 instructions

Saving Code Coverage: See www.keil.com/support/man/docs/uv4/uv4_cm_coverage.htm

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. gcov is supported.

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 .gcov file: Use the command COVERAGE GCOV *module or* COVERAGE GCOV *.
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 ; provides the data for log. 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.

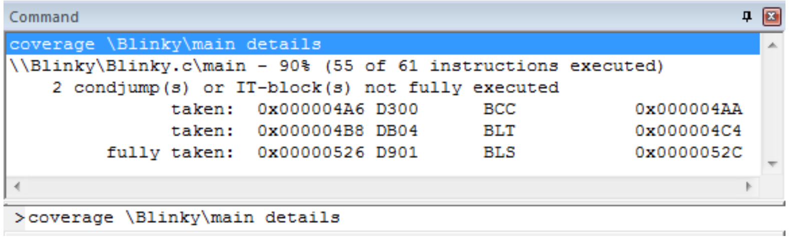
```
\\Blinky\RTE/Device/MSP432E401Y/startup_msp432e401y_uvision.s\Default_Handler-0%(0 of 1 inst executed)
\\Blinky\Blinky.c\main - 90% (55 of 61 instructions executed)
    2 condjump(s) or IT-block(s) not fully executed
\\Blinky\Blinky.c\Delay - 100% (9 of 9 instructions executed)
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
\\Blinky\Blinky.c\LEDinit - 100% (142 of 142 instructions executed)
    1 condjump(s) or IT-block(s) not fully executed
\\Blinky\Blinky.c\__scatterload - 100% (11 of 11 instructions executed)
\\Blinky\Blinky.c\__scatterload_copy - 100% (7 of 7 instructions executed)
\\Blinky\Blinky.c\__scatterload_null - 0% (0 of 1 instructions executed)
\\Blinky\Blinky.c\__scatterload_zeroinit - 100% (7 of 7 instructions executed)
```

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

```
EX |  0x000003DC SystemInit:
 EX |  0x000003DC B510      PUSH          {r4,lr}
 EX |  0x000003DE 4806      LDR           r0,[pc,#24]  ; @0x000003F8
 EX |  0x000003E0 6800      LDR           r0,[r0,#0x00]
 EX |  0x000003E2 F4400070  ORR           r0,r0,#0xF00000
 EX |  0x000003E6 4904      LDR           r1,[pc,#16]  ; @0x000003F8
 EX |  0x000003E8 6008      STR           r0,[r1,#0x00]
 EX |  0x000003EA F000F809  BL.W          _GetSystemClock (0x00000400)
 EX |  0x000003EE 0840      LSRS          r0,r0,#1
 EX |  0x000003F0 4902      LDR           r1,[pc,#8]  ; @0x000003FC
 EX |  0x000003F2 6008      STR           r0,[r1,#0x00]
 EX |  0x000003F4 BD10      POP           {r4,pc}
\\Blinky\RTE/Device/MSP432E401Y/system_msp432e401y.c\_GetSystemClock-44%(16 of 36 instructions executed)
    1 condjump(s) or IT-block(s) not fully executed
```

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)



```
coverage \Blinky\main details
\\Blinky\Blinky.c\main - 90% (55 of 61 instructions executed)
    2 condjump(s) or IT-block(s) not fully executed
              taken:  0x000004A6 D300      BCC           0x000004AA
              taken:  0x000004B8 DB04      BLT           0x000004C4
        fully taken:  0x00000526 D901      BLS           0x0000052C
```

```
>coverage \Blinky\main details
```

2) Shown here is an example using:
`coverage \main\main details`

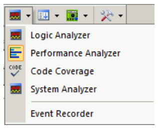
If the log command is run, this will be saved/appended to the specified file.

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

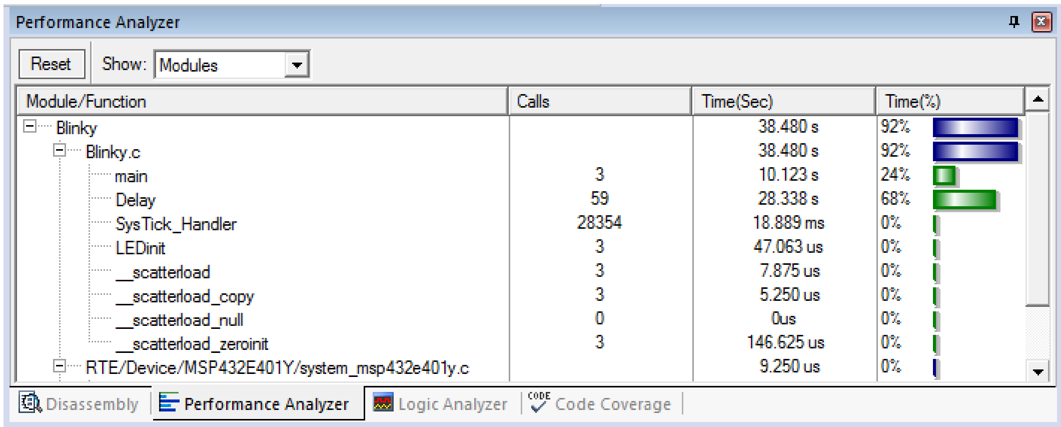
29) Performance Analysis (PA):

Performance Analysis tells you how much time was spent in each function. It is useful to optimize your code for speed.

Keil provides Performance Analysis with the µVision 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.



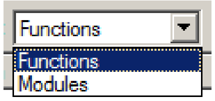
1. Use the same setup as used with Code Coverage. Have the program running.
2. Select View/Analysis Windows/Performance Analysis or select Performance Analyzer from here:
3. A window similar to the one below will open up.
4. Expand some of the module names as shown below.
5. Click RESET in the PA window. Reset
6. This resets the PA window. You will see the PA refresh in real-time from the ETM bugger in real-time.
7. Shown is the number of calls and percentage of total time in this short run from RESET to the beginning of main().
8. There is no need to stop the processor to collect the data. No code stubs are needed in your source files.



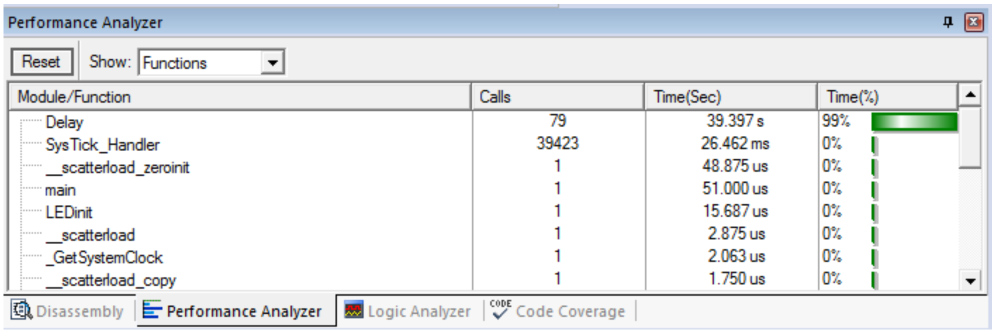
Show: Modules

Module/Function	Calls	Time(Sec)	Time(%)
Blinky		38.480 s	92%
Blinky.c		38.480 s	92%
main	3	10.123 s	24%
Delay	59	28.338 s	68%
SysTick_Handler	28354	18.889 ms	0%
LEDinit	3	47.063 us	0%
__scatterload	3	7.875 us	0%
__scatterload_copy	3	5.250 us	0%
__scatterload_null	0	0us	0%
__scatterload_zeroinit	3	146.625 us	0%
RTE/Device/MSP432E401Y/system_msp432e401y.c		9.250 us	0%

TIP: Double click on any Function or Module name and this will be highlighted in the Disassembly or source windows.



9. Select Functions from the pull down box as shown here and notice the difference.
10. Note the different data set displayed.
11. Note the display changes in real-time while the program is running.
12. Click on the RESET icon RST but the processor will stay at the initial PC and will not run to main().
13. Type **g, main** in the Command window to run the program to the start of main. You can see any initialization that took place.
14. Click on the PA RESET icon. Reset Watch as new data is displayed in the PA window.
15. When you are done, STOP and exit Debug mode .

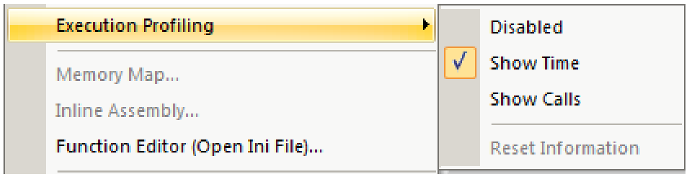


Show: Functions

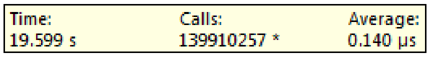
Module/Function	Calls	Time(Sec)	Time(%)
Delay	79	39.397 s	99%
SysTick_Handler	39423	26.462 ms	0%
__scatterload_zeroinit	1	48.875 us	0%
main	1	51.000 us	0%
LEDinit	1	15.687 us	0%
__scatterload	1	2.875 us	0%
_GetSystemClock	1	2.063 us	0%
__scatterload_copy	1	1.750 us	0%

30) Execution Profiling:

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

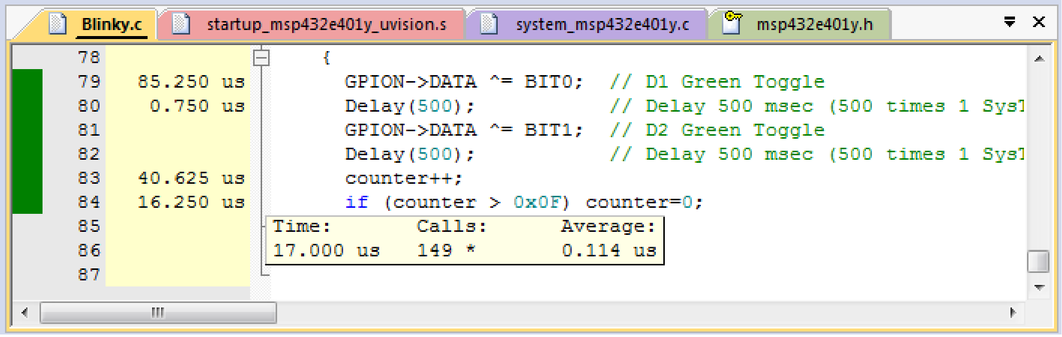


1. Enter Debug mode.
2. Select Debug/Execution Profiling/Show Time.
3. Click on RUN.
4. In the left margin of the Disassembly and C source windows will display various time values.
5. The times will start to fill up as shown below:
6. Click inside the yellow margin of main.c to refresh it.
7. This is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



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

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

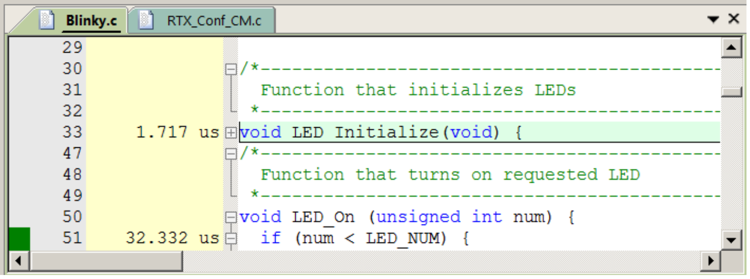


```
                 {
85.250 us          GPION->DATA ^= BIT0;   // D1 Green Toggle
 0.750 us          Delay(500);            // Delay 500 msec (500 times 1 SysT
                   GPION->DATA ^= BIT1;   // D2 Green Toggle
                   Delay(500);            // Delay 500 msec (500 times 1 SysT
40.625 us          counter++;
16.250 us          if (counter > 0x0F) counter=0;
                Time:       Calls:      Average:
                17.000 us   149 *       0.114 us
```

-Outlining:

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

1) Click in the square at the beginning of the function LEDinit near line 12.
2) The C source in the function LED_Init is now collapsed into one line.
3) Any times are added together.
4) This can be useful to hide sections of code to simplify the window you are reading.
5) Click on the + to expand it.
6) Stop the program
7) Exit Debug mode .



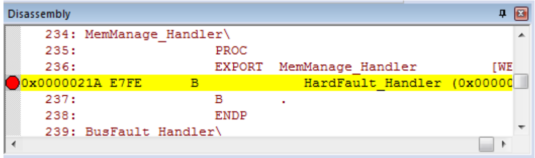
```
             ⊟/*-----------------------------------------------
               Function that initializes LEDs
              *-----------------------------------------------
  1.717 us   ⊞void LED_Initialize(void) {
             ⊟/*-----------------------------------------------
               Function that turns on requested LED
              *-----------------------------------------------
             ⊟void LED_On (unsigned int num) {
 32.332 us   ⊟  if (num < LED_NUM) {
```

For more information see: www.keil.com/support/man/docs/uv4/uv4_ui_outline.htm

31) In-the-Weeds Example: (your addresses might not be the same as those shown below)

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 thread switching plus unexpected and spurious events. Having a recording of the program flow is useful especially when a problem occurs and the consequences are not immediately visible. Another problem is detecting race conditions and determining how to fix them. ETM trace handles these problems and others easily and it is easy to use.

If a Hard Fault occurs in our example, the CPU will end up at 0x021A as shown in the Disassembly window below. This is the Hard Fault handler. This is a branch to itself and will run this instruction forever. The trace buffer will save millions of the same branch instructions. This is not very useful. The addresses used might be different in your project.

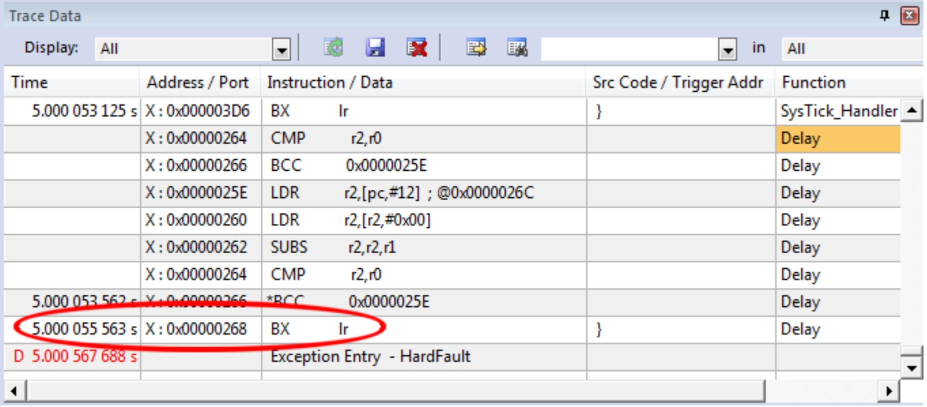


```
   234: MemManage_Handler\
   235:                 PROC
   236:                 EXPORT  MemManage_Handler          [WE
0x0000021A E7FE      B               HardFault_Handler (0x00000
   237:                 B       .
   238:                 ENDP
   239: BusFault Handler\
```

The Hard Fault handler exception vector is found in the file startup_msp432e401y_uvision.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 HardFault_Handler and halt processing.

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

1. Use the ETM example from the previous exercise, enter Debug mode.
2. Locate the Hard Fault near address 0x0021A in Disassembly or near line 232 in startup_msp432e401y_uvision.s.
3. Set a breakpoint at this point. A red circle will appear.
4. In the Command window enter: **g, Delay** and press ENTER. This will put the PC at the start of this function. Delay returns with a BX lr instruction; near address 0x0268 which we will use to create a Hard Fault with lr = 0.
5. The assembly and sources in the Disassembly window do not always match up and this is caused by anomalies in ELF/DWARF specification. In general, scroll downwards in this window to provide the best match.
6. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
7. In the Register window, double-click on the R14 (LR) register and set it to zero. This will cause a Hard Fault when the processor places lr = 0 into the PC and tries to execute the non-existent instruction at memory location 0x00.
8. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch instruction.
9. In the Trace Data window, you will find the Delay function with the BX instruction at the end. When the function tried to return, the bogus value of lr caused a Hard Fault.
10. The B instruction at the Hard Fault vector was not executed because Arm CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.
11. Click on Single Step. You will now see the Hard Fault branch as shown in the bottom screen:



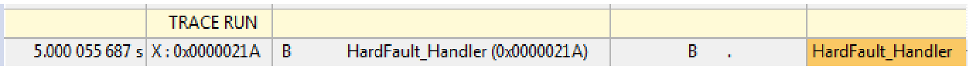
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
5.000 053 125 s	X : 0x000003D6	BX lr	}	SysTick_Handler
	X : 0x00000264	CMP r2,r0		Delay
	X : 0x00000266	BCC 0x0000025E		Delay
	X : 0x0000025E	LDR r2,[pc,#12] ; @0x0000026C		Delay
	X : 0x00000260	LDR r2,[r2,#0x00]		Delay
	X : 0x00000262	SUBS r2,r2,r1		Delay
	X : 0x00000264	CMP r2,r0		Delay
5.000 053 562 s	X : 0x00000266	*BCC 0x0000025E		Delay
5.000 055 563 s	X : 0x00000268	BX lr	}	Delay
D 5.000 567 688 s		Exception Entry - HardFault		

This example clearly shows how quickly ETM trace can help debug program flow bugs.

Exception Entry: Note the Exception Entry at the bottom of the Trace data. This entry is part of SWV. The timestamps of ETM and SWV are not the same. They cannot be correlated together.

TIP: Instead of setting a breakpoint on the Hard Fault vector, you could also right-click on it and select Insert Tracepoint at line 232… and select TraceHalt. When Hard Fault is reached, trace collection will halt but the program will keep executing for testing and Hard fault handlers.

12. Exit Debug mode.



	TRACE RUN			
5.000 055 687 s	X : 0x0000021A	B HardFault_Handler (0x0000021A)	B .	HardFault_Handler

32) Serial Wire Viewer and ETM Trace Summary:

We have several debug systems implemented in Kinetis Cortex-M4 devices:

1. SWV and ITM data output on the SWO pin located on the JTAG/SWD 10 pin CoreSight debug connector. The 20 pin connector adds ETM trace.
2. ITM is a printf type viewer. ASCII characters are displayed in the Debug printf Viewer in µVision.
3. Non-intrusive Memory Reads and Writes in/out the JTAG/SWD ports (DAP).
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.
5. ETM provides a record of all instructions executed. ETM also provides Code Coverage and Performance Analysis.

These features are completely controlled through µVision via a ULINK2 or ULINK*pro*.

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

SWV Trace 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) or RTOS thread switches.

Usually, these techniques allow finding bugs without stopping the program. These are the types of problems that can be found with a quality trace: Some of these items need ETM trace.

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes). How I did I get to this Fault vector ?
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs). ***How did I get here ?***
4) A corrupted stack.
5) Out of bounds data. Uninitialized variables and arrays.
6) Stack overflows. What causes the stack to grow bigger than it should ?
7) **Runaway programs:** your program has gone off into the weeds and you need to know what instruction caused this. ***This is probably the most important use of trace.***
8) Communication protocol and timing issues. System timing problems.
9) Trace adds significant power to debugging efforts. Tells you where the program has been.
10) Weeks or months can be replaced by minutes.
11) Especially where the bug occurs a long time before any consequences are seen.
12) Or where the state of the system disappears with a change in scope(s).
13) Plus - don't have to stop the program to test conditions. Crucial to some applications.
14) A recorded history of the program execution ***in the order it happened.*** Source and Disassembly is as it was written.
15) Trace can often find nasty problems very quickly.
16) Profile Analysis and Code Coverage is provided. Available only with ETM trace.

What kind of data can the Serial Wire Viewer display ?

1. Global variables.
2. Static variables.
3. Structures.
4. Can see Peripheral registers – just read or write to them. The same is true for memory locations.
5. Can see executed instructions. SWV only samples them. Use ETM to capture all instructions executed.
6. CPU counters. Folded instructions, extra cycles and interrupt overhead.

What Kind of Data the Serial Wire Viewer can't display…

1. Can't see local variables. (just make them global or static).
2. Can't see register operations. PC Samples records some of the instructions but not the data values.
3. SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU. SWV and ETM can only see CPU actions. If using a ULINK*pro* and RTX, DMA exceptions will display in the Event Viewer.

33) Creating your own MDK 5 project from scratch:

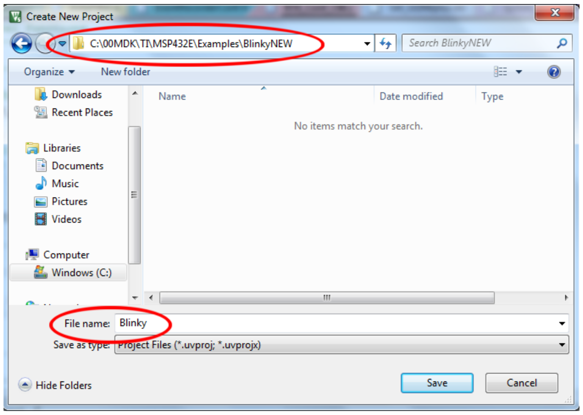
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run a bare Blinky example. It will have a simple incrementing counter to monitor. 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 MSP432E Software Pack for your processor:

1. Start µVision and leave it in Edit mode. Do not enter Debug mode. Any project must be loaded to start the process.
2. **Pack Installer:** The MSP432E Software Pack 3.2.2 or later must be installed. This has already been done on page 5.

Create a new Directory and a New Project:

3. In the main µVision menu, select Project/New/ µVision Project… Create New Project opens.
4. In this window, shown here, go to the folder C:\00MDK\TI\MSP432E\Examples\
5. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
6. Double click on BlinkyNew to open it or highlight it and select Open.
7. In the File name: box, enter Blinky. Click on Save.
8. This creates the MDK 4 project Blinky.uvproj.
9. The Select Device for Target…opens:


Create New Project

C:\00MDK\TI\MSP432E\Examples\BlinkyNEW

No items match your search.

File name: Blinky

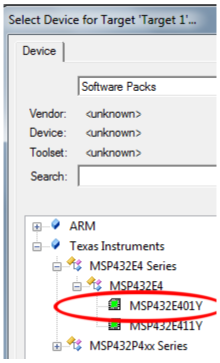
Save as type: Project Files (*.uvproj; *.uvprojx)

Select the Device you are using:

1. Expand Texas Instruments and then select MSP432E401Y as shown here:

TIP: Make sure you select the deepest layer device or this will not work correctly.

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


Select Device for Target 'Target 1'...

Vendor: <unknown>
Device: <unknown>
Toolset: <unknown>
Search:

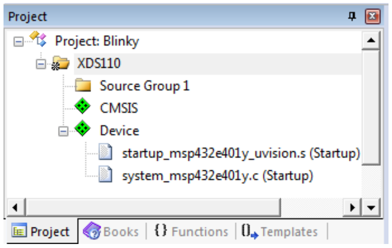
- ARM
- Texas Instruments
  - MSP432E4 Series
    - MSP432E4
      - MSP432E401Y
      - MSP432E411Y
  - MSP432P4xx Series

Select the CMSIS components you want:

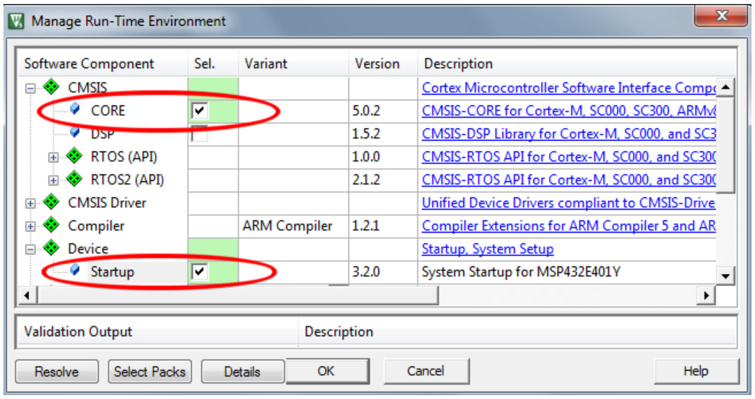
1. Expand CMSIS and Device. Select CORE and Startup as shown below. They will be highlighted in Green indicating there are no other files needed. Click OK to close.
2. Click on File/Save All or select the Save All icon:
3. The project Blinky.uvproj will now be changed to Blinky.uvprojx. (MDK 4 ➡ MDK 5)
4. You now have a new project list as shown on the bottom left below: The CMSIS files you selected have been automatically entered and configured into your project for your selected processor.
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 XDS110 and press Enter. The Select Target name will also change.

What has happened to this point:

You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files and select your debug adapter. The Software Pack has pre-configured many settings for your convenience.


Project

- Project: Blinky
  - XDS110
    - Source Group 1
    - CMSIS
    - Device
      - startup_msp432e401y_uvision.s (Startup)
      - system_msp432e401y.c (Startup)

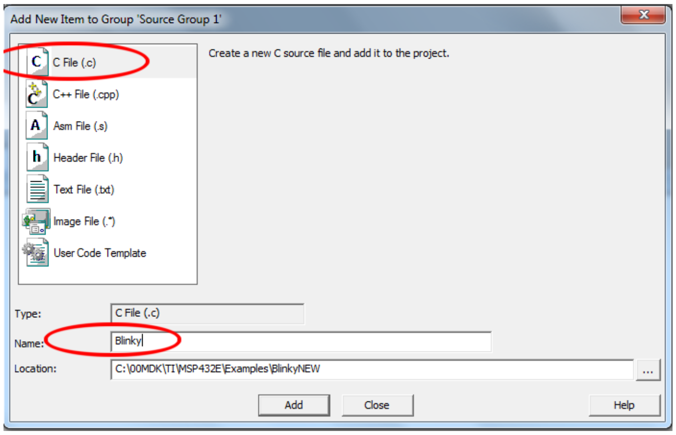

Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Compc
CORE	☑		5.0.2	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8
DSP	□		1.5.2	CMSIS-DSP Library for Cortex-M, SC000, and SC3
RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC30(
RTOS2 (API)			2.1.2	CMSIS-RTOS API for Cortex-M, SC000, and SC30(
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Drive
Compiler		ARM Compiler	1.2.1	Compiler Extensions for ARM Compiler 5 and AR
Device				Startup, System Setup
Startup	☑		3.2.0	System Startup for MSP432E401Y

Validation Output	Description

Create a blank C Source File:

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



Create a new C source file and add it to the project.

Type: C File (.c)

Name: Blinky

Location: C:\00MDK\TI\MSP432E\Examples\BlinkyNEW

Add Some Code to Blinky.c:

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

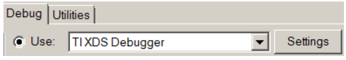
```
#include "msp.h"
unsigned int counter = 0;
/*---------------------------------------------
  MAIN function
 *---------------------------------------------*/
int main (void) {

  while(1) {
     counter++;
         if (counter > 0x0F) counter = 0;
}
}           //make sure you add a CR Carriage Return or Enter after the last parentheses.
```

TIP: You can also add existing source files: Add Existing Files to Group 'Source Files'...

Configure the Target XDS110: ***Please complete these instructions carefully to prevent unusual problems…***

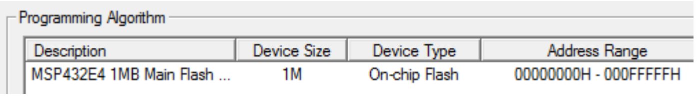
1. Select the Target Options icon . Select the **Target** tab. Note the Flash and RAM addresses are already entered.
2. Select Use MicroLIB to optimize for smaller code size. An error will be generated if you cannot use this feature.
3. Select the **Linker** tab. Select Use Memory Layout from Target Dialog.
4. Click on the **Debug** tab. Select the debugger you are using in the Use: box:
5. Connect your LaunchPad to your PC USB port. If using an external adapter, connect it now. Set the J101 jumpers.



● Use: TI XDS Debugger

TIP: You can select the debug adapter you are using at this point such as a ULINK2, ULINK*plus*, ULINK*pro* or a J-Link. If Flash programming fails, you can try reducing Max Clock: to 1 MHz or so.

6. Select Settings: box on the right side as shown above.
7. Set SW as shown here: Port SW If your LaunchPad board is connected to your PC, you should now see a valid Device Name in the SW Device box. If you do not, you **must** correct this before continuing.
8. Click on the **Flash Download** tab. In RAM for Algorithm box, for Size: enter 0xF00: Size: 0xF000
9. Confirm the correct Flash algorithms are present: Shown here are the correct ones for the MSP432E401 board:
10. Click on OK twice to return to the main menu.
11. Click on File/Save All or
12. Build the files. There will be no errors or warnings if all was entered correctly. If there are, please fix them !



Programming Algorithm

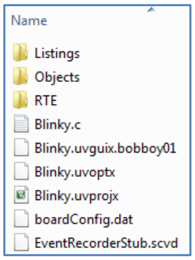
Description	Device Size	Device Type	Address Range
MSP432E4 1MB Main Flash ...	1M	On-chip Flash	00000000H - 000FFFFFH

The Next Step ? First we will do a summary of what we have done so far and then ….

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

What we have so far ?

1. An MDK 5 project (Blinky) has been created in C:\00MDK\TI\MSP432\Examples\BlinkyNEW\
2. The folders have been created similar as shown here:
3. RTE contains the CMSIS-Core startup and system files.
4. The Software Pack has pre-configured many items in this new project for your convenience.

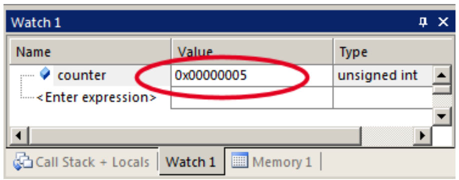


Name

- Listings
- Objects
- RTE
- Blinky.c
- Blinky.uvguix.bobboy01
- Blinky.uvoptx
- Blinky.uvprojx
- boardConfig.dat
- EventRecorderStub.scvd

Running Your Program:

1. Enter Debug mode. The Flash will be programmed. A progress bar is displayed in the Command window.
2. Click on the RUN icon. Note: you stop the program with the STOP icon.
3. Right click on counter in Blinky.c and select Add 'counter' to … and select Watch 1.
4. counter should be updating as shown here:
5. 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.
6. You should now be able to add your own source code to create a meaningful project.



Watch 1

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

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 skip some sequential values that you know must exist.

What else can we do ?

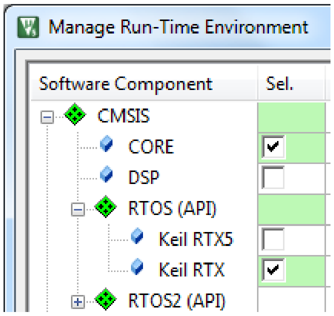
5. You can create new source files using the Add New Item window. See the top of the previous page.
6. You can add existing source files by right clicking on a Group name and selecting Add Existing Files.
7. You can easily add TI MSP432 example files to your project.
8. If you use RTX or Keil Middleware, source and template files are provided in the Add New window.

34) Adding RTX to your MDK 5 project:

The MDK Software Packs contain the code needed to add RTX to your project. RTX is CMSIS-RTOS compliant.

Configuring RTX is easy in MDK 5. These steps use the preceding Blinky example you constructed.
µVision also supports FreeRTOS, Micrium or any CMSIS-RTOS compliant RTOS.

1. Using the same example from the preceding pages, Stop the program and Exit Debug mode.
2. Open the Manage Run-Time Environment window:
3. Expand all the elements as shown here:
4. Select Keil RTX as shown and click OK.
5. Appropriate RTX files will be added to your project. See the Project window.
6. In Blinky.c, on the first line, right click and select Insert '# include file'. Select "cmsis_os.h". This will be added as the first line in Blinky.c.

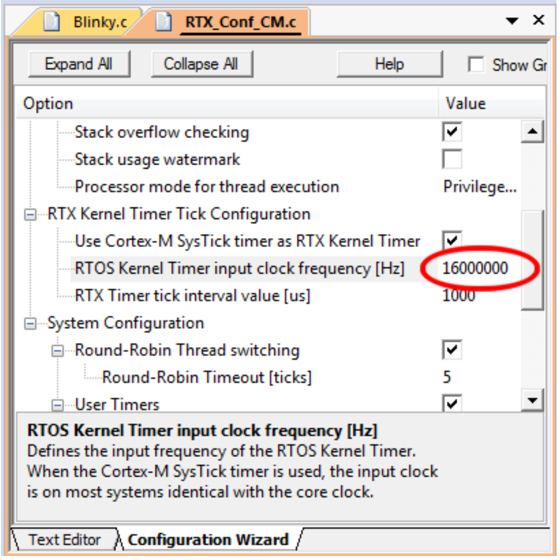


Manage Run-Time Environment

Software Component	Sel.
CMSIS	
CORE	☑
DSP	□
RTOS (API)	
Keil RTX5	□
Keil RTX	☑
RTOS2 (API)	

Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.
3. Select the Configuration Wizard tab at the bottom of this window: Select Expand All.
4. The window is displayed here:
5. Select Use Cortex-M SysTick Timer as RTX Kernel Timer.
6. Set Timer clock value: to 48000000 as shown: (48 MHz)
7. Use the defaults for the other settings.



Option	Value
Stack overflow checking	☑
Stack usage watermark	□
Processor mode for thread execution	Privilege...
RTX Kernel Timer Tick Configuration	
Use Cortex-M SysTick timer as RTX Kernel Timer	☑
RTOS Kernel Timer input clock frequency [Hz]	16000000
RTX Timer tick interval value [us]	1000
System Configuration	
Round-Robin Thread switching	☑
Round-Robin Timeout [ticks]	5
User Timers	☑

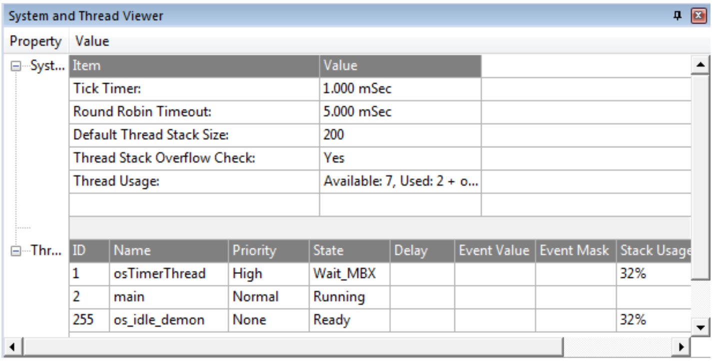
RTOS Kernel Timer input clock frequency [Hz]
Defines the input frequency of the RTOS Kernel Timer.
When the Cortex-M SysTick timer is used, the input clock is on most systems identical with the core clock.

Build and Run Your RTX Program:

1. Click on File/Save All or
2. Build the files. There will be no errors or warnings.
3. Enter Debug mode: Click on the RUN icon.
4. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
5. You can see three 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.
6. Stop the program and Exit Debug mode.

What you have to do now:

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project.
2. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/gsg/. It has very useful information on implementing RTX.
3. You can use the Event Viewer to examine your threads graphically if you have a ULINK2, ULINK*plus*, ULINK*pro* or a J-Link. Event Viewer uses SWV which is not yet supported by XDS110.



System and Thread Viewer

Property | Value

Syst...

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: 2 + o...

Thr...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				32%
2	main	Normal	Running				
255	os_idle_demon	None	Ready				32%

TIP: The Configuration Wizard is a scripting language as shown in the Text Editor as comments starting such as a </h> or <i>.
See www.keil.com/support/docs/2735.htm for instructions on how to add this feature to your own source code.

35) Adding a Thread to your RTX_Blinky:

We will create and activate a thread. We will add an additional variable counter2 that will be incremented by this new thread.

1. In Blinky.c, add this line near line 4: unsigned int counter2 = 0;

```
5   unsigned int counter2 = 0;
```

Create the Thread job1:

2. Add this code before main():
   This will be the new thread named job1.

   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.

```
7     void job1 (void  const *argument) {
8     for (;;) {
9               counter2++;
10                if (counter2 > 0x0F) counter2=0;
11               osDelay(500);
12      }
13    }
```

Add osDelay to main():

3. In the main() function, Add this line just after the if statement near line 20:

```
20   osDelay(500);
```

Define and Create the Thread:

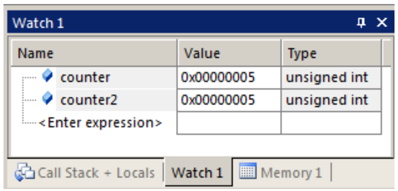
1. Add this line near line 15 just before main():

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

2. Create the thread job1 near line 19 just before the while(1) loop:

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

3. Click on File/Save All or
4. Build the files. There will be no errors or warnings. If there are, please fix them before continuing.
5. Enter Debug mode: Click on the RUN icon.
6. Right click on counter2 in Blinky.c and select Add counter2 to … and select Watch 1.
7. 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.



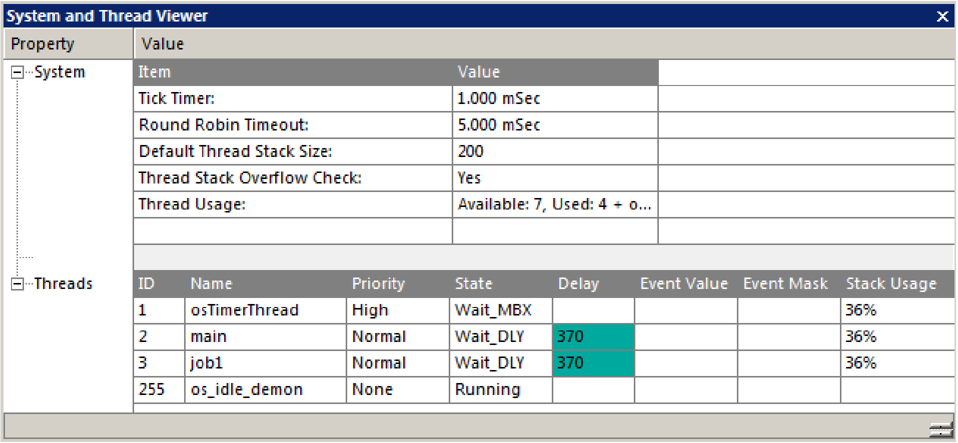
Name	Value	Type
counter	0x00000005	unsigned int
counter2	0x00000005	unsigned int
<Enter expression>		

TIP: osDelay() is a function provided by RTX and is triggered by the SysTick timer.

8. Open the System and Thread Viewer by selecting Debug/OS Support.
9. Note that job1 has now been added as a thread as shown below:
10. Note os_idle_demon is nearly always labelled as Running. This is because the program spends most of its time here.
11. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
12. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
13. There are many attributes of RTX you can add. RTX help files are located here depending on your CMSIS version: C:/Keil_v5/ARM/Pack/ARM/CMSIS/*x.x.x*/CMSIS/Documentation/RTX/html/index.html.
14. **Remove any breakpoints you have created.**

On the next page, we will demonstrate the Event Viewer. For this you must use a Keil ULINK2, ULINK*plus*, ULINK*pro* or a Segger J-Link.

.



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				

Demonstrating the Event Viewer (EV): Requires a Keil ULINK2/ME, ULINK*pro* or a J-Link.

The Event Viewer displays threads running in a graphical format. It is possible to make timing measurements.

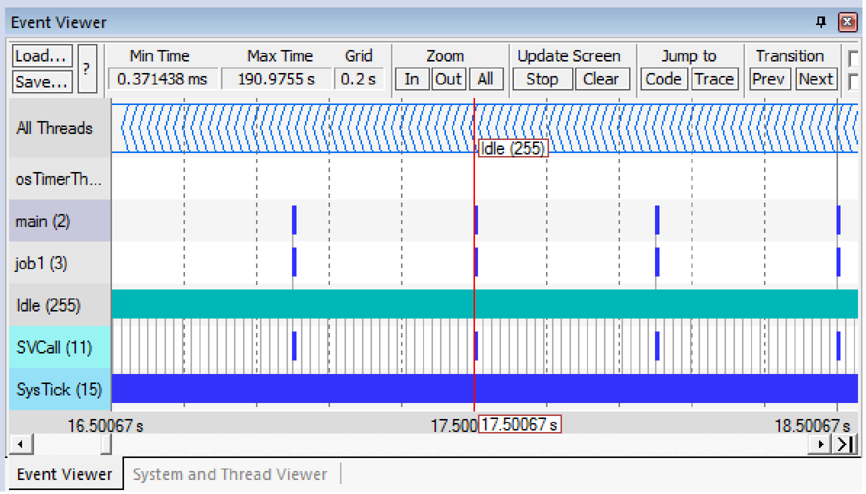
The Event Viewer uses Serial Wire Viewer (SWV). The XDS110 does not support SWV. In order to do the exercise on this page, you must use any Keil ULINK or a J-Link. A ULINK*pro* is best as it handles SWV data the fastest.

Connect and Configure SWV:

1. Connect a ULINK2, ULINK*plus*, ULINK*pro* or J-Link to J11.
2. Connect a USB cable to the LaunchPad to supply power to the board.
3. Select the Target Options icon . Select the Debug tab. Select the debugger you are using:
4. Click the Debug tab. Select the debugger you are using in the Use: box: This is for ULINK2: ULINK2/ME Cortex Debugger
5. Click on Settings on the right side of the target Options window. The Cortex-M Target Driver Setup window opens.
6. Set SW as shown here: Port SW JTAG does not support SWV. If your LaunchPad board is connected to your PC, you should now see a valid IDCODE and Device Name in the SW Device box. If you do not, you **must** correct this before continuing.
7. Set Max. Clock: to 2 MHz. If this value is too high, Flash programming might not work reliably resulting in an error.
8. Select the Trace tab. Select Core Clock: to 16 (16 MHz). Select Trace Enable.
9. Unselect EXCTRC to minimize SWO overflows. ITM bit 31 must be selected as Event Viewer uses this port.
10. Click OK twice to close the Target configuration windows.
11. Click on File/Save All or SWV is now configured.

Run Blinky and Open Event Viewer:

1. Enter Debug mode: Click on the RUN icon.
2. The program should be running as shown in the System and Thread Viewer as shown on the previous page. Viewing this window gives a good visual indication RTX is configured and running properly. EV displays timing graphically.
3. Select Debug/OS Support and select Event Viewer.
4. This window will open and if SWV is correctly configured, the threads will be visible.
5. Use the In and/or Out buttons to select an appropriate horizontal scale.



Min Time 0.371438 ms
Max Time 190.9755 s
Grid 0.2 s

6. Note the program spends most of its time in Idle (os demon). This can be modified in your program.
7. Note the interrupts SVCall and SysTick visible at the bottom. This is with ULINK*plus* or a ULINK*pro*. If your program has other interrupts active, they will also be displayed here.

This is the end of the examples for this tutorial. Thank you !

36) TroubleShooter:

Here are some issues that arose during the creating and testing of this document.

1. Something doesn't work as claimed: usually a step was inadvertently missed. This is very common. Go slower.
2. No XDS110 found: Make sure µVision can see C:\ti\. This contains the XDS110 driver. Make sure the firmware in the Launchpad board is current. Try reloading it. Check to see if there is a newer version.
3. ULINKplus: RDDI-DAP Error when attempt to enter Debug mode. uVision can't communicate with the target via ULINKplus.. Cycle the power of both the board and the ULINKplus.
4. No Serial Wire Viewer: Nearly always the Core Clock: is wrong. This must be set to the exact CPU frequency with a ULINK2, ULINK*plus* or J-Link. ULINK*pro* uses the clock embedded in the SWO or ETM signals. ETM and SWO will work but erroneous timestamps will result.
5. No SWV or can't connect: Sometimes the debug adapter does not completely RESET the CoreSight components by design. Cycle the board power. This is more common when switching between debug adapters.
6. Serial Wire Viewer erratic or Logic Analyzer or Event Viewer distorted: This usually means you have selected too many attributes and SWO pin is overloaded. Unselect those you do not need or use a ULINK*plus* or a ULINK*pro*. If problem is in the Logic Analyzer because your variables change too fast, you might have to poll your variable in your program.
7. SWV or connection troubles: Cycle the board power. This ensures a clean restart in case something was set inside the processor. Especially when switching between debug adapters.
8. No ULINK found: If a ULINK is connected: this means you have selected the wrong ULINK in Target Options.
9. No SWD Connection: Your ULINK or J-Link is connected backwards to the 5 pin CoreSight connector or the Debug switch is in the wrong position.
10. Flash Programming failed: This means your compilation failed because of an error that will be listed. The last .axf file will be deleted and no new one created therefore there is no executable .axf file to download. It can also mean the Max Clock in the Debug tab in the Target Options is set too high. For ULINK2 try 2 MHz.
11. Make sure the RAM Start: address is set to 0x2000_0000. Try increasing the Size: attribute.
12. Program halts unexpectedly: Perhaps you failed to delete breakpoints or Watchpoints after using them ?

ETM Traceport Pins:

See for connectors: www.keil.com/coresight/coresight-connectors

These signals must be provided using a custom adapter for ETM trace operation with a ULINKpro or J-Trace.

Refer to the TI datasheet for your particular processor and package.

ETM Signal	MSP432E ETM Port	MSP432E Pin	CoreSight 20 Pin
TRCLK	PF3	45	12
TRD0	PF2	44	14
TRD1	PF1	43	16
TRD2	PF0	42	18
TRD3	PF4	46	20

37) Document Resources: See www.keil.com/TI

Books:

1. ***NEW!*** **Getting Started with 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. µVision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab. Or 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
9. **Introduction to the MSP432 Microcontroller** by Jonathan Valvano
10. **Real-Time Interfacing to the MSP432 Microcontroller** by Jonathan Valvano
11. **MOOC:** Massive Open Online Class: University of Texas: http://users.ece.utexas.edu/~valvano/

Application Notes:

12. ***NEW!*** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
13. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
14. CAN Primer: Controller Area Network: www.keil.com/appnotes/files/apnt_247.pdf
15. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
16. Porting mbed Project to Keil MDK™ 4 www.keil.com/appnotes/docs/apnt_207.asp
17. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
18. GNU tools (GCC) for use with µVision https://launchpad.net/gcc-arm-embedded
19. RTX CMSIS-RTOS RTX Documents www.keil.com/pack/doc/CMSIS/RTX/html
20. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
21. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
22. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
23. Sending ITM printf to external Windows applications: http://www.keil.com/appnotes/docs/apnt_240.asp
24. ***New*** Using from Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/docs/apnt_209.asp
25. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
26. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/docs/apnt_209.asp
27. ***New*** Migrating from Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
28. ***New*** ARMv8-M Architecture Technical Overview https://community.arm.com/docs/DOC-10896
29. ***NEW!*** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp

Useful ARM Websites:

1. CMSIS Standards: https://github.com/ARM-software/CMSIS_5 and www.arm.com/cmsis/
2. CMSIS Documentation: www.keil.com/pack/doc/CMSIS/General/html
3. ARM and Keil Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content
4. ARM University Program**:** www.arm.com/university. Email: university@arm.com
5. mbed™: http://mbed.org

A blank page for future enhancements.

38) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- **MDK-Lite™** (Evaluation version) 32K Code and Data Limit - $0
- *New* **MDK-ARM-CM™** For all Cortex-M series processors only – unlimited code limit.
- *New* **MDK-Plus™** MiddleWare Level 1. ARM7™, ARM9™, Cortex-M, SecureCore®.
- *New* **MDK-Professional™** MiddleWare Level 2. For complete details: www.keil.com/mdk5/version520.

Middleware includes Network, USB, Graphics and File System. www.keil.com/mdk5/middleware/

USB-JTAG adapters (for Flash programming too)

- **ULINK2** - *(ULINK2 and ME - SWV only – no ETM)*
- **ULINK-ME** – sold only with a board by Keil or OEM. Electrically equivalent to a ULINK2.
- *New* **ULINK*plus*** - Cortex-Mx High performance SWV & power measurement.
- **ULINK*pro*** - SWV & ETM trace. Fast Flash programming speed. Also works with DS-5.
- **ULINK*pro* D** - SWV & no ETM. Fast Flash programming speed. Also works with DS-5.

- ***For special promotional or quantity pricing and offers, please contact Keil Sales.***
- ***In USA, Canada and South America:*** **1-800-348-8051** ***sales.us@keil.com***
- ***Europe, Asia, Middle East, Africa:*** **+49 89/456040-20** ***sales.intl@keil.com***

Keil RTX RTOS is now provided as a component of CMSIS 5: ***New*** https://github.com/ARM-software/CMSIS_5

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

Keil provides free DSP libraries for the Cortex-M0, Cortex-M3, Cortex-M4 and Cortex-M7.

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 quite useful.

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

Call Keil Sales for special university pricing.

See www.keil.com/TI for more information regarding TI support.

For Linux, Android, various RTOS and no OS support (bare metal) on TI Cortex-A processors, please see DS-5: www.arm.com/ds5.

KEIL Tools by ARM
Development Tools
Getting Started
Creating Applications with µVision®4
For 8-bit, 16-bit, and 32-bit Microcontrollers
ARM
Cx51
C166

For more information:

Sales In Americas: sales.us@keil.com or 800-348-8051. **Europe/Asia:** sales.intl@keil.com +49 89/456040-20

Global Inside Sales Contact Point: Inside-Sales@arm.com **Keil World Wide Distributors:** www.keil.com/distis/

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 Texas Instrument specific information: See www.keil.com/TI

CMSIS: www.arm.com/cmsis **CMSIS Version 5:** https://github.com/ARM-software/CMSIS_5

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™ Intelligent Processors by ARM®

arm KEIL