NXP AN5413 Cookbook Rev. 1 for S32K14x: Examples Ported to Arm® Keil® MDK Toolkit.

Winter 2017 V 1.0 bob.boys@arm.com www.keil.com/NXP

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_304.asp

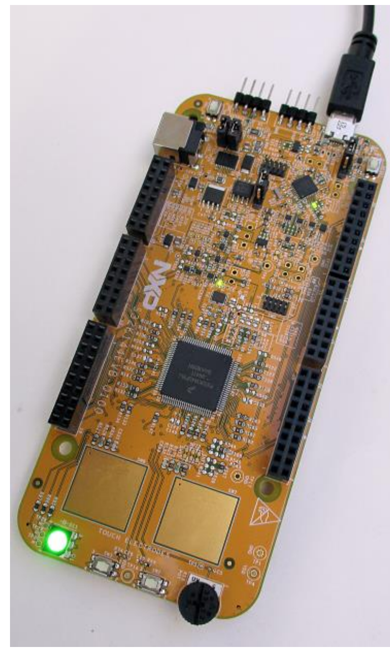
A Lab for the S32K-144 EvalBoard is here: www.keil.com/appnotes/docs/apnt_299.asp

A lab for the S32K-148 EVAL using ETM: www.keil.com/appnotes/docs/apnt_305.asp

Introduction:

This document explains the debugging features provided by the NXP S32K-144 processor using the Cookbook examples. These examples are provided ported to Keil µVision® using Arm Compiler 5. It is intended for those who want to follow the material in the NXP Cookbook with Keil MDK toolkit. The Cookbook Rev. 1 is located here: www.nxp.com/assets/documents/data/en/application-notes/AN5413.pdf

These examples can be used with Keil MDK-Lite™ which is a free evaluation version that limits code size to 32 Kbytes. The addition of a valid license number will turn it into an unrestricted commercial version. Contact Keil Sales for licensing questions.



General Notes:

1. MDK 5.24a, S32K Pack 1.0.0 and OpenSDA in CMSIS-DAP mode was used.
2. The S32K144 EVB board was used. This board is pictured here:
3. The examples have not been ported to S32K148 EVB yet. See the chart in the Cookbook for a list of the board peripheral ports changes.

Debug Adapters:

This document uses OpenSDA in CMSIS-DAP mode. For exercises using Serial Wire Viewer (SWV) any Keil ULINK™ or a J-Link must be used. A notice in red is given.

1. OpenSDA is an on-board adapter. It has CMSIS-DAP and P&E modes. CMSIS-DAP is preferred. ULINK2, ULINK*plus*, ULINK*pro* and J-Link are external tools. If you want to see the SWV features, use any ULINK or a J-Link.
2. Debug Adapters that can be used: OpenSDA CMSIS-DAP is used by default.
    a. OpenSDA P&E mode: Basic stop and go debugging.
    b. OpenSDA CMSIS-DAP mode: Provides DAP live window updates.
    c. Keil ULINK2: adds Serial Wire Viewer (SWV)
    d. Keil ULINK*plus*: SWV and adds Power Measurement. (coming soon)
    e. Keil ULINK*pro*: adds ETM trace S32K-148 only. Fast SWV.
    f. Segger J-Link: Adds SWV.
3. Each example project has Target Options for each of these debug adapters.
4. For useful CoreSight™ definitions, see page 22.

CoreSight™ Debug Technology Summary:

These technologies are used to provide information and interaction with the S32K processor family. Refer to CoreSight Definitions on page 22 for detailed information.

1. **Basic CoreSight:** 6 hardware breakpoints and 1Watchpoint. These can be set/unset while the program is running.
2. **Debug Access Port (DAP):** JTAG or SWD modes available. Provides data updated while your program is running. Used by Watch, Memory, Peripherals and RTOS awareness windows. Works with OpenSDA CMSIS-DAP, any Keil ULINK™ and J-Link and OpenSDA P&E. DAP and CMSIS-DAP are not the same. See page 22 for definitions.
3. **Serial Wire Viewer (SWV):** Non-intrusive data trace. Displays interrupts, variables in graphical Logic Analyzer, data writes and CPU counters. printf display using no UART (printf is slightly intrusive). Needs a Keil ULINK2, ULINK*plus*, ULINK*pro* or a J-Link. SWV updates windows while the program is running.
4. **ETM Instruction Trace:** S32K-148 and ULINK*pro* only. Provides a record of all instructions in the order they were executed. ETM is useful for debugging program flow problems and program crashes. ETM also supplies Code Coverage, Performance Analysis and Execution Profiler.

Keil Software and Software Packs:

Hello Cookbook Example Program:

The Rest of the Cookbook Examples:

Other Useful Information:

1) Keil MDK Software Download and Installation:

1. Download MDK 5.24a Lite or later from the Keil website. www.keil.com/mdk5/install 

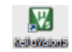
2. Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil_v5
3. We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
4. If you install MDK into a different folder, you will have to adjust for the folder location differences.
5. You do not need a debug adapter: just the S32K board, a USB cable and MDK installed on your PC.
6. For the exercises using SWV, you need a Keil ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* or a J-Link.
7. You do not need a Keil MDK license for this tutorial. All examples will compile within the 32 K limit.

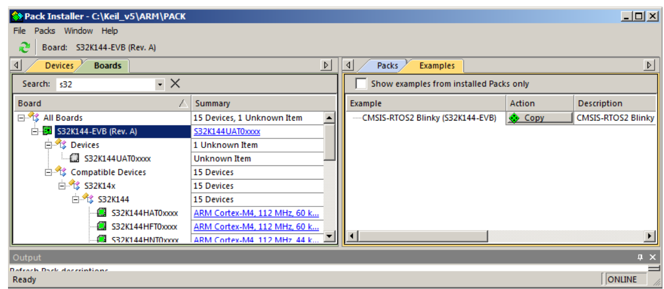
2) µVision S32K Software Pack Download and Install Process:

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

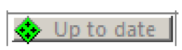
TIP: The left hand pane filters the selections displayed on the right pane. You can start with either Devices or Boards.

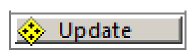
Start µVision and open Pack Installer:

1. Connect your computer to the internet. This is needed to download the Software Packs. Start µVision: 
2. Open the Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.
3. This window opens up: Select the Devices tab:
4. Note "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet before continuing.
5. If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or  to refresh once you have connected to the Internet.



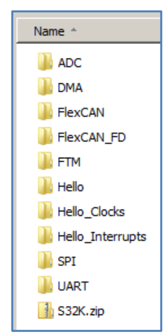
Install The S32K Software Pack:

1. In the Devices tab, select NXP and then S32K Series as shown above: The devices supported are displayed.
2. Select Keil::S32_SDK_DFP.1.0.0.pack and click Install. This Pack will download and install in the MDK files. This download can take several minutes.
3. Its status is indicated by the "Up to date" icon: 

4. Update means there is an updated Software Pack available for download. 

3) Install the Keil Cookbook Examples:

1. Obtain the Cookbook examples from here: www.keil.com/appnotes/docs/apnt_304.asp
2. These files will be in a .zip format. Create the folder: C:\00MDK\Boards\NXP\S32K\
3. Extract the zip archive into the folders you just created.
4. The folder structure will then look like this:



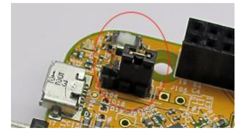
4) Configuring OpenSDA in CMSIS-DAP Mode:

If you are using any Keil ULINK, J-Link or OpenSDA P&E as your debug adapter: you can skip this page:
This document will use NXP OpenSDA in CMSIS-DAP mode as an CMSIS-DAP debug adapter. This will replace the P&E debugger that comes pre-installed on a new S32K board. Target connection by µVision will be via a standard USB cable connected to USB connector J7. The on-board Kinetis K20 processor U8 acts as the CMSIS-DAP on-board debug adapter.

Program the K20 with the CMSIS-DAP application file CMSIS-DAP.S19:

1) Locate the file CMSIS-DAP.S19:

1. CMSIS-DAP.S19 is located in www.keil.com/appnotes/docs/apnt_299. It is also in every project in the \OpenSDA folder. You will now copy this file into the S32K board USB device.



2) Put the S32K Board into Bootloader: Mode:

1. Set jumper J104 to position 1-2.
2. Hold RESET button SW5 on the board down and connect a USB cable to J7 as shown below:
3. When you hear the USB dual-tone, release the RESET button to enter bootloader mode.
4. The **S32K board** will act as a USB mass storage device called BOOTLOADER connected to your PC. Open this USB device with Windows Explorer.

3) Copy CMSIS-DAP.S19 into the S32K Board:

1. Copy and paste or drag and drop CMSIS-DAP.S19 into this Bootloader USB device.

4) Exit Bootloader Mode:

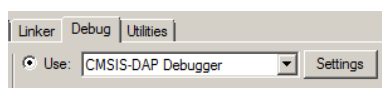
1. Set jumper J104 to back to position 2-3.
2. Cycle the power to the S32K board while ***not*** holding RESET button down.
3. The **S32K** board is now ready to connect to µVision.

TIP: This application will remain in the U8 K20 Flash each time the board power is cycled with RESET not pressed. The next time board is powered with RESET held on, it will be erased. CMSIS-DAP.S19 is the CMSIS application in the Motorola S record format that loads and runs on the K20 OpenSDA processor.

TIP: If you must later re-program CMSIS-DAP.S19 and it still does not work with µVision: check that Port: is set to SW and not JTAG. See the **TIP:** below.

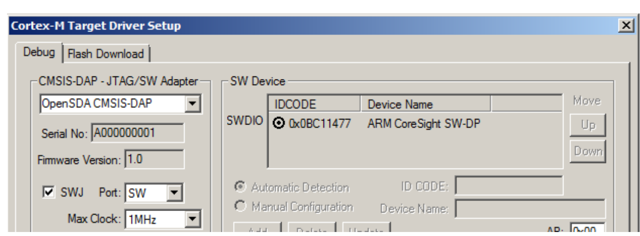
5) Testing The OpenSDA CMSIS-DAP Connection: (Optional Exercise)

1. Start µVision if it is not already running. Select Project/Open Project.
2. Select the Hello project C:\00MDK\Boards\NXP\S32K\Hello\Hello.uvprojx. (or any other project you choose)
3. Select Target Options or ALT-F7 and select the Debug tab:
4. Select CMSIS-DAP Debugger as shown here:



5. Click on Settings: and the window below opens up: Select SW in the Port box as shown below. An IDCODE and Device name will then be displayed indicating connection to the CoreSight DAP. This means CMSIS-DAP OpenSDA is working. You can continue with the tutorial. Click on OK twice to return to the µVision main menu.
6. If nothing or an error is displayed in this SW Device box, this ***must*** be corrected before you can continue.
7. Select File/Save All or .

TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window. CMSIS-DAP will not work with JTAG selected, only SW. But this is a useful way to refresh the SW setting.

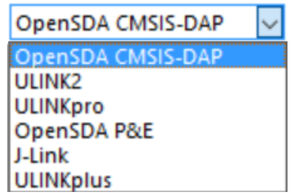


TIP: You can use this method to test the operation of other debug adapters.

6) *Hello* example program:

Now we will connect a Keil MDK development system using the S32K board. This page will use the OpenSDA CMSIS-DAP debug adapter but you can select others.

1. Connect a USB cable between your PC and the S32K board J7 USB connector.
2. If you are using an external debug adapter, connect it to J14 SWD. Power the board to USB J7. J107 position 2-3.
3. Green LED D3 (power) will light. If not, check J107. 2-3 for USB power. 1-2 is for external power 12 volts to J16.
4. Start µVision by clicking on its desktop icon.
5. Select Project/Open Project.
6. Open the file: C:\00MDK\Boards\NXP\S32K\Hello\Hello.uvprojx.
7. Choose your debug adapter accordingly:



8. Compile the source files by clicking on the Rebuild icon. .
9. Enter Debug mode by clicking on the Debug icon. The Flash memory will be programmed. Progress will be indicated in the Output Window or in the P&E window. Select OK if the Evaluation Mode box appears.

TIP: If the Flash programs with P&E but does not enter Debug mode, select Debug mode again:

10. Click on the RUN icon.
11. Press the button SW2. It is beside the potentiometer R13.

The blue LED will come one when SW2 is pressed.

Now you know how to compile a program, program it into the S32K processor Flash, run it and stop it !

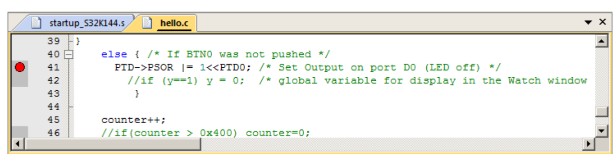
Note: The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

TIP: When you need to, but not right now, you stop the program with the STOP icon.

7) Hardware Breakpoints:

The S32K14x Cortex®-M4 has six hardware breakpoints that can be set or unset on the fly while the program is running and when using a CMSIS-DAP or any Keil ULINK or a J-Link debug adapter. The Cortex-M0+ has two hardware breakpoints. **You must stop the program to set/unset breakpoints with P&E.**

1. With Hello running, in the hello.c window, locate the line PTD->PSOR |= 1<<PTD0; which is near line 41.
2. Note the darker grey blocks on the left in hello.c. This grey block indicates assembly instructions are present.
3. Click on the grey block opposite PTD->PSOR |= 1<<PTD0;
4. A red circle will appear and the program will presently stop as indicated by a yellow arrow. Remember to restart the program if using P&E.
5. Note the breakpoint is displayed in both the Disassembly and source windows: You can set/unset them in either window.
6. Leave the hardware breakpoint active for the next exercise.



TIP: Select Debug/Breakpoints or Ctrl-B and the Breakpoint window opens. You can manage Breakpoints and Watchpoints in this window. You can temporarily disable/enable them. Close this window if you opened it.

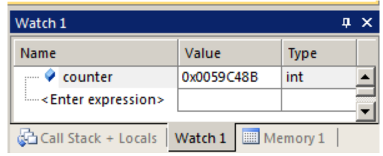
TIP: If you set too many breakpoints, µVision will warn you.

TIP: Arm hardware breakpoints do **not** execute the instruction they are set to and land on. Arm CoreSight hardware breakpoints are no-skid. This is a rather important feature for effective debugging.

TIP: µVision uses only hardware breakpoints in Flash memory. Soft or Flash breakpoints are not used. These breakpoints have the disadvantage of substituting instructions and potentially modifying the program behavior.

8) Watch Window:

1. In hello.c, right click on counter near line 45 and select Add counter to … and select Watch 1. Watch 1 will open and counter will be displayed as shown here:



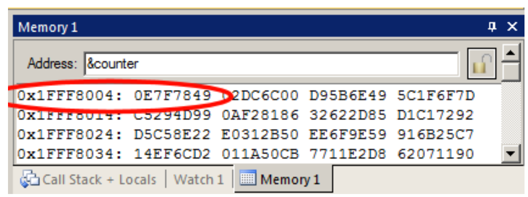
2. Every time you click on the RUN icon, counter will increment.
3. Remove the breakpoint by clicking on it.
4. Click on the RUN icon. Watch 1 will display <cannot evaluate>.
5. counter is a local variable declared at the start of the main() function near line 17. Watch 1 can only display locals when they are in scope and the program stopped. Locals are normally stored in a CPU register Rn.
6. Stop the program and leave Debug mode.
7. Add static in front of the declaration for counter in hello.c near line 17: static int counter = 0;

TIP: You can also make counter a global variable or part of a structure or union.

8. Compile the source files.
9. Enter Debug mode and click RUN.
10. Note counter now increments while your program is running.

9) Memory Window:

1. In hello.c, right click on counter near line 45 and select Add counter to … and select Memory 1. Memory 1 will open and counter will be displayed:
2. Note the value of counter is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to but this not what we want to see at this time.
3. Add an ampersand "&" in front of the variable name and press Enter. counter will be displayed in Memory 1.
4. The physical address in this case is 0x1FFF_8004.
5. Right click in the Memory window and select Unsigned/Int.
6. The data contents of counter is displayed as shown here:



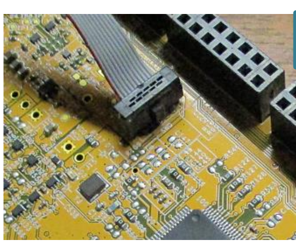
Modify Memory:

7. Right click on the changing memory location and select Modify Memory at 0x1FFF_8004. Enter 0 in the dialog box and press Enter. You might get a different memory address depending on the compilation and other settings.
8. Note the data was set to zero and started incrementing over.
9. Stop the program

TIP: You can change the value in a Watch window but only of the data is changing very slowly or the program is stopped.

10) Single-Stepping:

1. With Blinky.c in focus (Blinky.c tab is underlined), click on the Step In icon or F11 a few times: You will see the program counter jumps a C line at a time. The yellow arrow indicates the next C line 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 one assembly instruction at a time.



Connecting a Keil ULINK:

Keil ULINK*pro* with NXP S32K144 EVB

11) CPU Clock Speed:

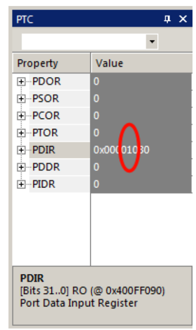
The CMSIS file system_S32K144.c contains some clock source code. Included is a global variable SystemCoreClock that can be used to display the CPU speed. In some programs, SystemCoreClockUpdate() must be called first.

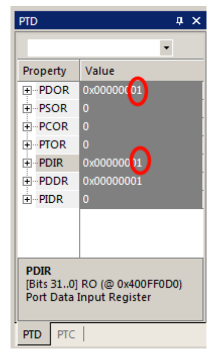
1. Stop the program Exit Debug mode
2. In hello.c, near line 19, uncomment SystemCoreClockUpdate():
3. Compile the source files . Enter Debug mode and click RUN .
4. In Watch 1, double click on <Enter expression> and enter SystemCoreClock.
5. SystemCoreClock will be displayed in Watch 1. Right click on its name and unselect Hexadecimal Display.
6. 48000000 (48 MHz) will be displayed.

TIP: SystemCoreClock is a calculated, not a measured value. A bug could give you an erroneous reading. For a method to determine the core clock using SWV, see www.keil.com/appnotes/docs/apnt_297.asp.

12) Peripheral Display - System Viewer:

Peripheral views are provided inside µVision. They update while your program runs and many registers can be modified.





Observe Button SW2 (it is connected to PTC):

1. Select Peripherals/System Viewer/GPIO/PTC. This port connects to SW2.
2. Press SW2 and note PDIR changes from 0x30 to 0x1030 and back.

Observe the Blue LED (it is connected to PTD):

1. Select Peripherals/System Viewer/GPIO/PTD. PTD connects to the blue LED.
2. Note two tabs appear at the bottom of these windows for easy selection.
3. When you press the SW2 button, PDOR and PIOR goes from 0x01 to 0x00.

Putting a Peripheral Register into Watch 1:

1. Some registers can be displayed in a Watch window.
2. Note when you select a register its description and address appear at the bottom of the window.
3. Using the address of PTD PDOR, enter this into Watch 1: ***((unsigned long *)0x400FF0C0)**
4. This value will change as you press SW2. Watch and Memory windows are updated periodically.

Changing a Register:

1. Stop the program You can do this while the program runs but in this case the variable is overwritten too quickly.
2. In either the Watch 1 or PTD, change the value from 0x01 to zero. The blue LED will come on.

TIP: You must be careful what registers you modify in case it causes unintended consequences that can be difficult to find.

13) Display a Global Variable:

1. In hello.c, uncomment the global variable declaration **uint32_t LED = 1;** found near line 4.
2. In hello.c, uncomment the statement **if (LED == 0) LED = 1;** near line 38.
3. Uncomment the statement **if (LED == 1) LED = 0;** near line 42.
4. Select File/Save All or . Build the source files . Enter Debug mode and click RUN .
5. Right click on LED and select Add LED to … and select Watch 1. LED will be displayed in Watch 1.
6. As you press the SW2 button, LED will change value.
7. When you are finished, Stop the program and leave Debug mode .

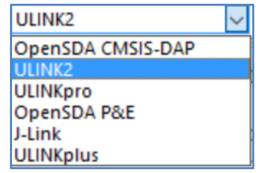
What we have at this point:

Everything on the last two pages uses the DAP Read/Write mechanism. This is nearly always non-intrusive. These features work with OpenSDA, Keil ULINK2, ULINK-ME, ULINK*plus*, ULINK*pro* and J-link. With OpenSDA in P&E mode, you must stop the program to configure and update the displayed variables. Do not confuse DAP with CMSIS-DAP.

14) Serial Wire Viewer (SWV): **This needs any Keil ULINK, or a J-Link.** Skip the next three pages if you do not have one of these debug adapters. Serial Wire Viewer is an important and useful debug tool.

SWV Configuration:

1. Attach any ULINK or a J-Link to the 10 pin CoreSight SWD connector. Power the board with a USB cable.
2. Choose the debug adapter you are using. This for the Keil ULINK2: ➡



3. Select Options for Target or ALT-F7 and select the Debug tab.
4. Your debugger must be displayed beside Use:.
5. Select Settings: Settings on the right side of this window.
6. Confirm Port: is set to SW and SWJ box is enabled for SWD operation. SWV will not work with JTAG.
7. Click on the Trace tab. The window below is displayed.
8. In Core Clock: enter 48 MHz. Select Trace Enable. This value ***must*** be set correctly to your CPU speed.

TIP: To find Core Clock: frequency: Enter the global variable SystemCoreClock in a Watch window and run the program.

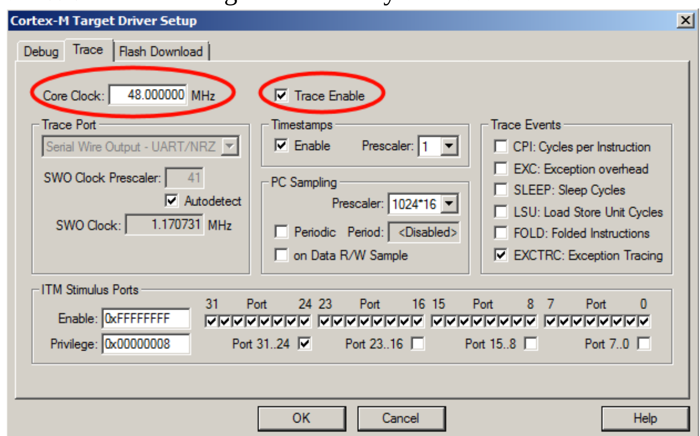
9. Click on OK twice to return to the main µVision menu. SWV is now configured and ready to use.

Notes:

1. With a ULINK*pro*, select SWO Manchester.
2. Core Clock: must be accurately set for ULINK2, ULINK-ME and J-Link. ULINK*pro* uses Core Clock only to determine timing values.

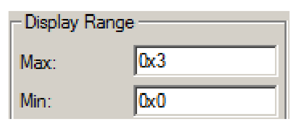
Problems: The most probable cause of SWV not working is an incorrect Core Clock: value. See www.keil.com/appnotes/docs/apnt_297.asp

The only valid ITM frames in the trace window are ITM 0 and ITM 31. If you see any other values, this nearly always means the Core Clock: value is incorrect.



15) Logic Analyzer (LA):

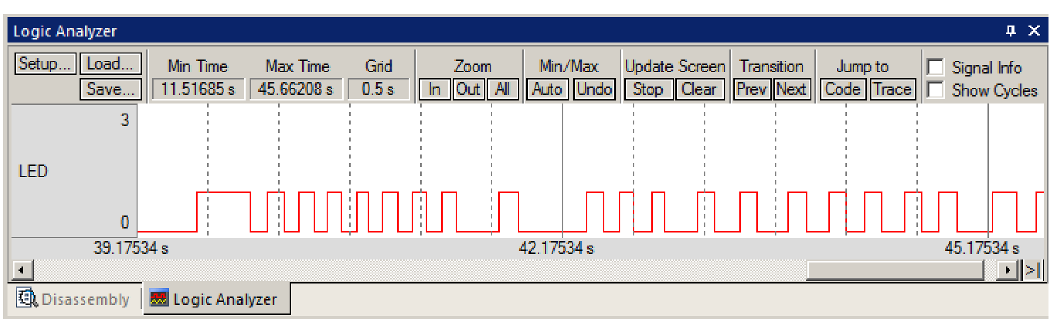
1. Build the source files . Enter Debug mode and click RUN .
2. Right click on LED in Hello.c near line 4 and select Add LED to … and select Logic Analyzer. LED will now be displayed in the LA as shown below.
3. In the Logic Analyzer (LA), select the Setup button.
4. In Display Range:, enter 0x3 for Max: and 0 for Min:. ➡



5. Click on Close.
6. Using the Zoom box, adjust to obtain a Grid: of about 0.5 seconds or another suitable value.
7. Press SW2 multiple times and a waveform similar to the one below will appear:

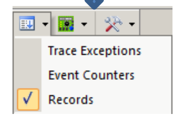
TIP: You can put up to four variables in the Logic Analyzer. They cannot be local or automatic variables as these are not visible unless in scope. Global, static, structures or anything in memory can be displayed.

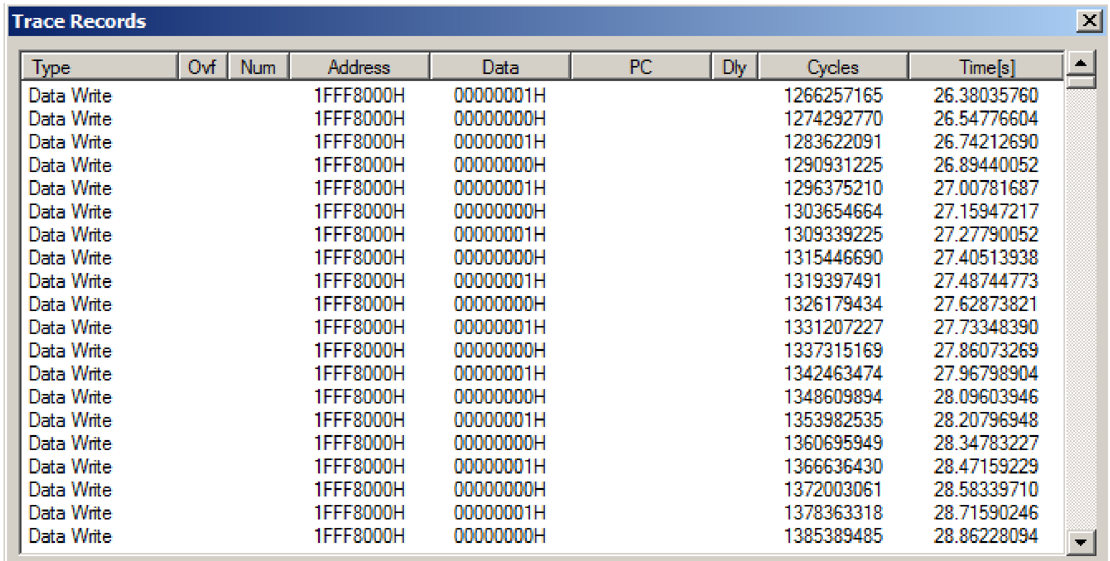
TIP: These are event created. Each variable entered creates an event and data comes out the SWO pin.



16) Trace Records: This needs any Keil ULINK, or a J-Link.

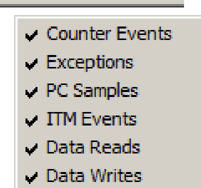
1. Open Trace Records window by clicking on the small arrow beside the Trace icon and select Records:
2. The Trace Records window shown below opens:
3. Note the Data Write frames displayed. These are the writes to the variable LED.
4. This is caused by the LED entry you made in the Logic Analyzer.
5. The first line: The value 0x01was written to the variable at address ox1FFF8000 at the cycles/times.
6. If On Data R/W Sample is selected in the trace configuration window (shown on previous page), the address of the instruction causing this write will be displayed in the PC column.
7. Double-click anywhere in Trace Records and it will be cleared and will start to fill up again if the program is running.




Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			1FFF8000H	00000001H			1266257165	26.38035760
Data Write			1FFF8000H	00000000H			1274292770	26.54776604
Data Write			1FFF8000H	00000001H			1283622091	26.74212690
Data Write			1FFF8000H	00000000H			1290931225	26.89440052
Data Write			1FFF8000H	00000001H			1296375210	27.00781687
Data Write			1FFF8000H	00000000H			1303654664	27.15947217
Data Write			1FFF8000H	00000001H			1309339225	27.27790052
Data Write			1FFF8000H	00000000H			1315446690	27.40513938
Data Write			1FFF8000H	00000001H			1319397491	27.48744773
Data Write			1FFF8000H	00000000H			1326179434	27.62873821
Data Write			1FFF8000H	00000001H			1331207227	27.73348390
Data Write			1FFF8000H	00000000H			1337315169	27.86073269
Data Write			1FFF8000H	00000001H			1342463474	27.96798904
Data Write			1FFF8000H	00000000H			1348609894	28.09603946
Data Write			1FFF8000H	00000001H			1353982535	28.20796948
Data Write			1FFF8000H	00000000H			1360695949	28.34783227
Data Write			1FFF8000H	00000001H			1366636430	28.47159229
Data Write			1FFF8000H	00000000H			1372003061	28.58339710
Data Write			1FFF8000H	00000001H			1378363318	28.71590246
Data Write			1FFF8000H	00000000H			1385389485	28.86228094

8. Right click in the Trace Records and see the types of frames you can filter out.



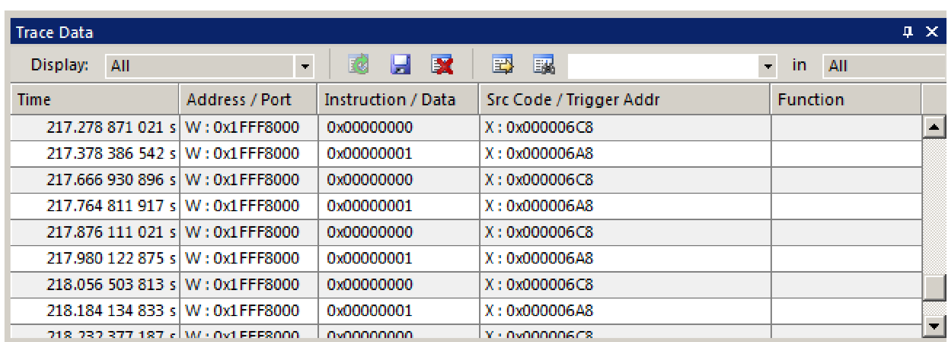
TIP: These Data writes are created by each event created when a write occurs in this case, to the variable LED. If there are too many writes or other types of frames, the SWO port (is 1 bit) is easily overloaded. This will be seen a "x" in the Ovf and Dly columns. A solution is to sample the frames to another variable and display it or use a ULINK*pro*. *Manchester is better and 4 bit Trace Port is even better. Only the S32K-148 has ETM trace and the corresponding 4 bit Trace Port.*

ULINK*pro* Trace Data Window:

You can see the same data displayed, In this case, the instruction causing the write at 0x06C8 is displayed.

.

9. When you are finished, Stop the program and leave Debug mode .


Trace Data

Display: All in All

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
217.278 871 021 s	W : 0x1FFF8000	0x00000000	X : 0x000006C8	
217.378 386 542 s	W : 0x1FFF8000	0x00000001	X : 0x000006A8	
217.666 930 896 s	W : 0x1FFF8000	0x00000000	X : 0x000006C8	
217.764 811 917 s	W : 0x1FFF8000	0x00000001	X : 0x000006A8	
217.876 111 021 s	W : 0x1FFF8000	0x00000000	X : 0x000006C8	
217.980 122 875 s	W : 0x1FFF8000	0x00000001	X : 0x000006A8	
218.056 503 813 s	W : 0x1FFF8000	0x00000000	X : 0x000006C8	
218.184 134 833 s	W : 0x1FFF8000	0x00000001	X : 0x000006A8	

Notes:

1. With a ULINK*pro* or a J-Link, the Trace Records is called Data Trace and will look different (ULINK*pro* is shown above) and the program must be stopped to update it.
2. If you double click on a ULINK*pro* frame, this will be highlighted in the Disassembly and source windows.
3. J-Link does not currently display Data Write frames.

17a) printf using SWV ITM 0: This needs any Keil ULINK, or a J-Link.

uVision® has an easy way to display printf statements in the Debug printf window. No UART is needed. It is possible to use a getf function but this will not be demonstrated here. See the S32K144 lab for details.

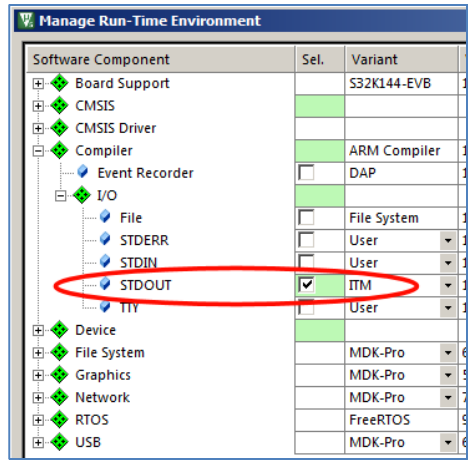
ITM = Instrumentation Trace Macrocell.

Configuration: µVision must not be in Debug mode.

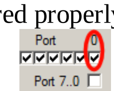
1. In hello.c, near line 3, uncomment #include <stdio.h>.
2. Near line 37, uncomment this statement: printf ("LED_On\n ");

Configure Event Recorder: Use this method if using SWV. Use DAP if using OpenSDA CMSIS-DAP.

3. Click on the Manage Run-Time Environment (MRTE) icon: The window below opens:
4. Expand Compiler/I/O and select STDOUT and ITM as shown:
5. Verify all blocks are green and not yellow or red.
6. Click OK.
7. This action adds the file retarget_io.c to the project.

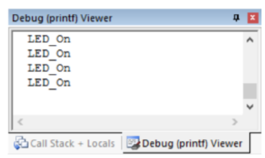


TIP: Serial Wire Viewer (SWV) must be configured properly. ITM Port 0 must be enabled. It is set by default and is shown here:



Build and RUN:

1. Build the source files . Enter Debug mode and click RUN .
2. Open View/Serial Windows and select Debug (printf) Viewer.
3. Each time you press SW2, the printf statement writes to the Debug printf Viewer as shown here:

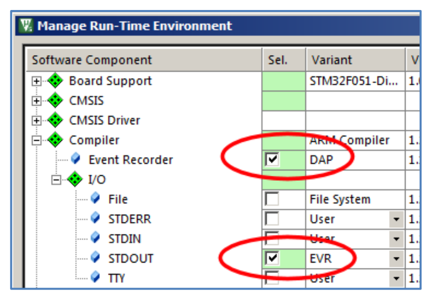


17b) printf using Event Recorder & DAP: This works with OpenSDA in CMSIS-DAP mode.

1. Use this method if using OpenSDA in CMSIS-DAP mode. This also works for Arm Cortex-M0 and Cortex-M0+.

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, EventRecorder.c and EventRecorderConf.h will be added to your project under the Compiler group in the Project window.
8. Right click near the top of Blinky.c, and select Insert "#include" and select #include "EventRecorder.h".
9. At the beginning of the main() function, add this line: EventRecorderInitialize (EventRecordAll, 1);
10. Follow the **Build and RUN** instructions as shown above.



Event Recorder Notes: You can use Event Recorder to annotate your own source code. This feature uses DAP reads therefore will work on processors without Serial Wire Viewer such as Cortex-M0. You can use CMSIS-DAP, any Keil ULINK or any J-Link with Event Recorder. See www.keil.com/support/man/docs/uv4/uv4_db_dbg_evr.htm

TIP: If you ever see orange or red blocs in the MRTE, try clinking the Resolve button. If it can, µVision will attempt to find the required files.

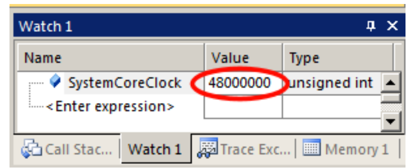
18) Hello_Clocks: This works with all debug adapters.

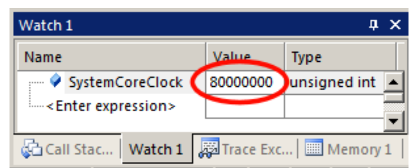
This project is the same as the previous Hello with clocks_and_modes.c added to change the clock frequency.

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\Hello_Clocks\Hello.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here.
4. In hello_clocks.c near line 43, uncomment SystemCoreClockUpdate();
5. This runs a function in system_S32K144.c to do some clock configuration.
6. Compile the source files by clicking on the Rebuild icon. .
7. Enter Debug mode The Flash memory will be programmed. Progress will be indicated in the Output Window or in the P&E window. Select OK if the Evaluation Mode box appears.

TIP: If the Flash programs with P&E but does not enter debug mode, select Debug mode again:

8. Enter SystemCoreClock in Watch 1.
9. Right click on SystemCoreClock and unselect Hexadecimal Display.
10. 48 MHz will display in Watch 1.
11. Click RUN.
12. Watch 1 will change to 80 MHz when clocks_and_modes.c is executed.
13. The blue LED will blink by itself as it is now controlled by Timer LPIT0.



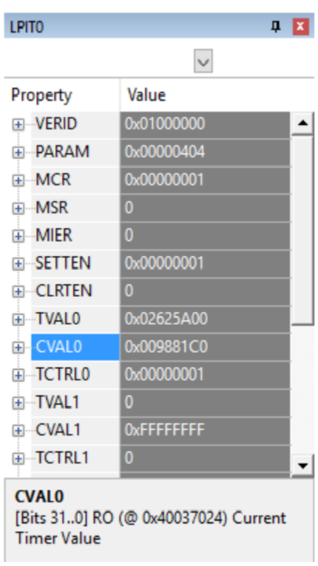


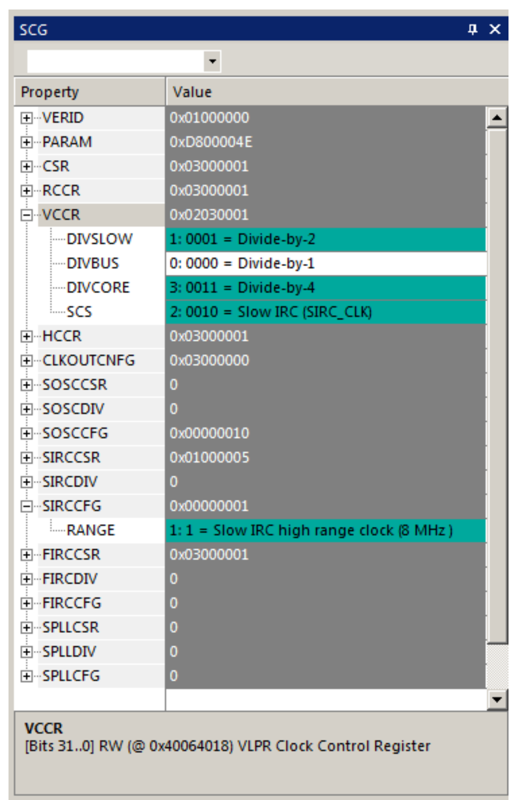
SCG Clock Peripheral:

1. The NXP clock registers can be displayed in µVision.
2. Open Peripherals/System Viewer/SCG and the window below right opens up:
3. It is possible to modify some of these values if allowed by the S32K.

LPIT0 Timer Peripheral:

1. Open Peripherals/System Viewer/LPIT0 and the window below left opens:
2. Click on CVAL0 and note its physical address is displayed.
3. Note Current Timer Value is updating while the program is running.





19) Hello_Interrupts: This works with all debug adapters.

This project is the same as Hello_Clocks with an interrupt added to blink the blue LED.

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\Hello_Interrupts\hello_interrupts.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here.
4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode. The Flash memory will be programmed. Progress will be indicated in the Output Window or in the P&E window. Select OK if the Evaluation Mode box appears.
6. Click RUN.
7. The blue LED will come on but it probably will not blink as expected.
8. The issue is the interrupt routine exits before the interrupt flag is cleared. This results in a second interrupt.
9. Set a breakpoint in the function LPIT0_Ch0_IRQHandler which is found near line 69.
10. Click RUN. and the blue LED will now toggle each time you click RUN.
11. Remove the breakpoint.
12. See www.keil.com/support/docs/3928.htm for an explanation.

Fix this Issue:

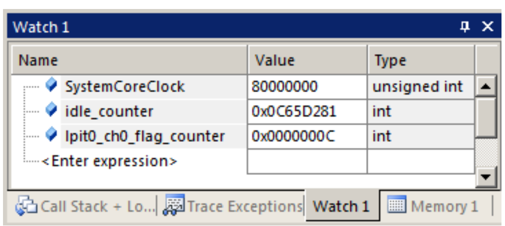
1. In hello_interrupts.c, there are several possible solutions:
2. Uncomment one of these.
3. If you use x = PTD->PTOR, also uncomment unsigned int x; near line 70.
4. Build the source files . Enter Debug mode and click RUN .
5. The blue LED will now blink.`
6. For rationale on which solution to use, review the Keil answer 3928 above.

```
LPIT0->MSR |= LPIT_MSR_TIF0_MASK;
//PTD->PTOR = PTD->PTOR;    // sel
//LPIT0->MSR=0;
//x = PTD->PTOR;    // also uncomm
//__dsb(0);
```

Displaying program timings with two global variables:

There are two global variables declared near line 13 and 14 in hello_interrupts.c. They are idle_counter and lpit0_ch0_flag_counter. These display certain events in this example.

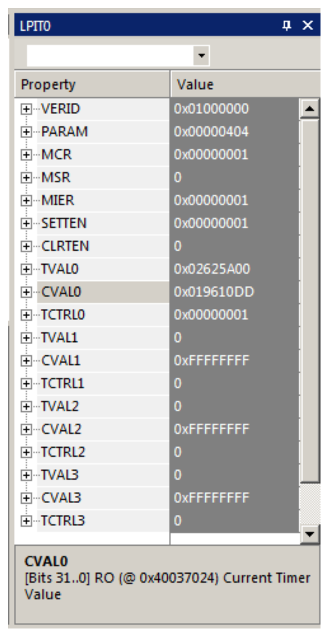
1. Right click on these and add them to Watch 1.
2. They will be displayed and update while the program runs in Watch 1 shown below:



3. Refer to the Cookbook for detailed explanations of this example.

Displaying LPIT0 Peripheral Registers:

1. Open Peripherals/System Viewer/LPIT0.
2. The registers values will be updating (if appropriate) as the program runs.
3. It is possible to modify some of these values if allowed by the S32K.
4. CVAL0 is the only register changing while the program is running.



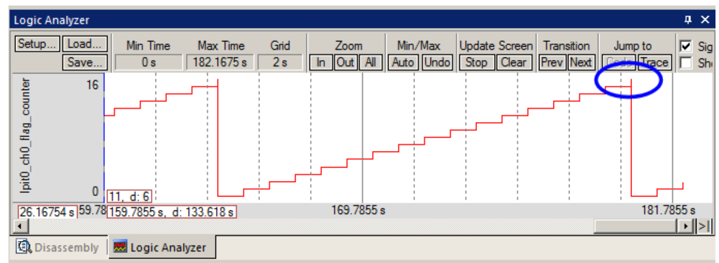
Displaying lpit0_ch0_flag_counter in the Logic Analyzer (LA): SWV is used: any ULINK or J-Link.

SWV is preconfigured in this program for ULINK2, ULINK*pro* and J-Link with a Core Clock: set to 80 MHz.

1. Stop the program and leave Debug mode .
2. In hello_interrupt.c,, near line 72, uncomment **if (lpit0_ch0_flag_counter > 0xF) lpit0_ch0_flag_counter = 0;**
3. Build the source files . Enter Debug mode and click RUN .
4. Right click on **lpit0_ch0_flag_counter** and add it to the Logic Analyzer.
5. Each time the interrupt routine is executed a step is displayed in the LA. The spike (blue circle) results from the test.
6. The time measured with Cursor and Signal Info is 1 second per step.
7. This is very useful to display and measure variables not visible with an outside instrument such as a scope.

Set lpit0_ch0_flag_counter to Zero:

1. In Watch 1, double-click on **lpit0_ch0_flag_counter** and set this to 0x00.
2. This will be displayed in the LA



Displaying Exceptions (including Interrupts): SWV is used: any ULINK or J-Link is needed.

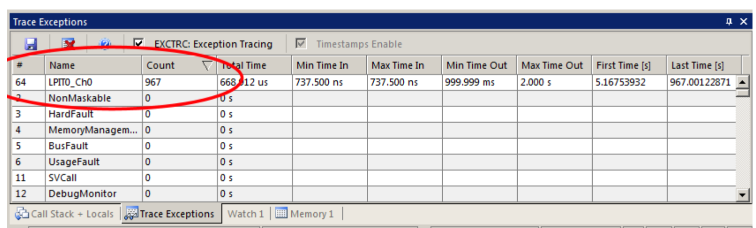
There are two windows used to display interrupts and their timings. If you are using Keil RTX and a ULINK*pro*, it is possible to display when and for how long handler routines happen. This is in the Event Viewer in a manner similar to the LA.

Exceptions include interrupts in Arm documentation.

Trace Exceptions Window:

1. Click on the Trace Exceptions window tab or select View/Trace/Trace Exceptions or select:

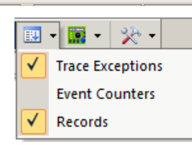
2. The Trace exceptions window opens:
3. Click in the Count column header to bring active exceptions to the top.
4. LPIT0_CH0 is displayed with various data.



#	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
64	LPIT0_Ch0	967	668.[illegible]12 us	737.500 ns	737.500 ns	999.999 ms	2.000 s	5.16753932	967.00122871
2	NonMaskable	0	0 s						
3	HardFault	0	0 s						
4	MemoryManagem...	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	0	0 s						
12	DebugMonitor	0	0 s						

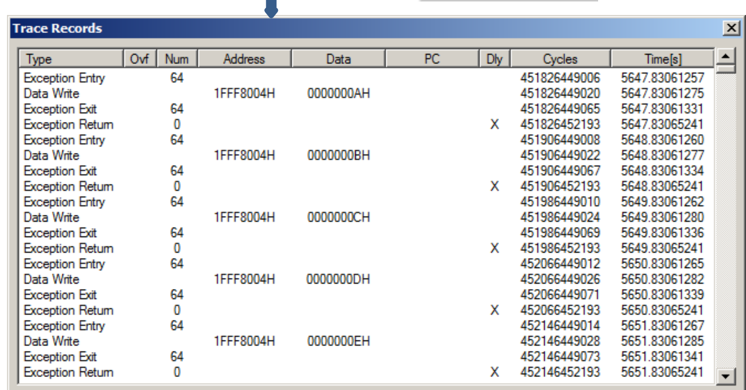
Trace Records Window:

1. Open the Trace Records window by selecting select View/Trace/Records or select:



2. Trace Records will display Interrupt 64 with Data Writes from the LA.
3. Right click anywhere and note how to filter various frames out.

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



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		64					451826449006	5647.83061257
Data Write			1FFF8004H	0000000AH			451826449020	5647.83061275
Exception Exit		64					451826449065	5647.83061331
Exception Return		0				X	451826452193	5647.83065241
Exception Entry		64					451906449008	5648.83061260
Data Write			1FFF8004H	0000000BH			451906449022	5648.83061277
Exception Exit		64					451906449067	5648.83061334
Exception Return		0				X	451906452193	5648.83065241
Exception Entry		64					451986449010	5649.83061262
Data Write			1FFF8004H	0000000CH			451986449024	5649.83061280
Exception Exit		64					451986449069	5649.83061336
Exception Return		0				X	451986452193	5649.83065241
Exception Entry		64					452066449012	5650.83061265
Data Write			1FFF8004H	0000000DH			452066449026	5650.83061282
Exception Exit		64					452066449071	5650.83061339
Exception Return		0				X	452066452193	5650.83065241
Exception Entry		64					452146449014	5651.83061267
Data Write			1FFF8004H	0000000EH			452146449028	5651.83061285
Exception Exit		64					452146449073	5651.83061341
Exception Return		0				X	452146452193	5651.83065241

TIP: ULINK*pro*, ULINK*plus* and J-Link have different trace windows from the ULINK2 Trace Records.

Unlike ULINK2, you must stop the program to update ULINK*pro* and J-Link Trace Data windows.

20) DMA: This works with all debug adapters.

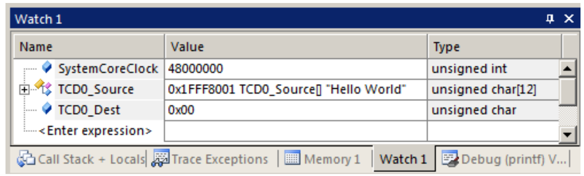
This project uses DMA to move a string from one memory location to another serially. This will be displayed in Watch .

Open the DMA Project, Build and Program the Flash:

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\DMA\DMA.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here.
4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode. The Flash memory will be programmed. Progress will be indicated in the Output Window or in the P&E window. Select OK if the Evaluation Mode box appears.
6. Open the Project window.
7. Double-click on DMA.c to open it. You can also select File/Open and navigate to the src folder.

Configure Watch 1:

1. DMA.c contains a array TCD0_Source[] and a variable TCD0_Dest near line 12. We will add them to Watch 1.
2. Right click on TCD0_Dest and add it into Watch 1. Right click on TCD0_Source and add it into Watch 1.
3. You can expand TCD0_Source to see all of its elements.



RUN the program:

1. Set a hardware breakpoint in main.c near line 42:
   DMA->SSRT = 0;
2. Click RUN.
3. Each time you click RUN, another letter is transferred from TCD0_Source to TCD0_Dest by the DMA controller.
4. This is visible in Watch 1.
5. When the transfer is complete, the program will be in the endless while (1) {} near line 50.
6. Click RESET to start the demo over.

21) Timed I/O (FTM): This works with all debug adapters.

Important TIP: A wire needs to be connected on J2 between pins 8 and 2 or 4. A resister such as 1kΩ can be used.

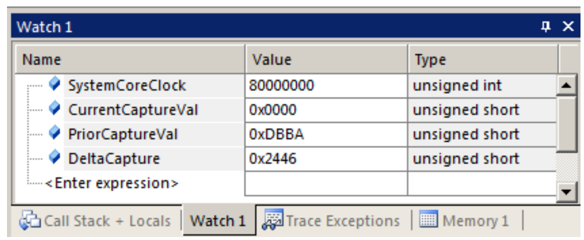
Open project, Build and Program Flash:

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\FTM\FTM.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here.
4. Compile the source files by clicking on the Rebuild icon. Enter Debug mode.

Variables Display Watch 1:

There are three global variables declared in FTM.c: CurrentCaptureVal, PriorCaptureVal and DeltaCapture. These will be displayed in Watch 1 at their initial values of 2. SystemCoreClock will be set to 80 MHz.

1. Select View/Watch Windows and select Watch 1. The three variables are already entered. If not, do this now.
2. Click RUN . The tri-colour LED will blink.
3. The values in Watch 1 will change.
4. Note DeltaCapture changes briefly every few seconds.



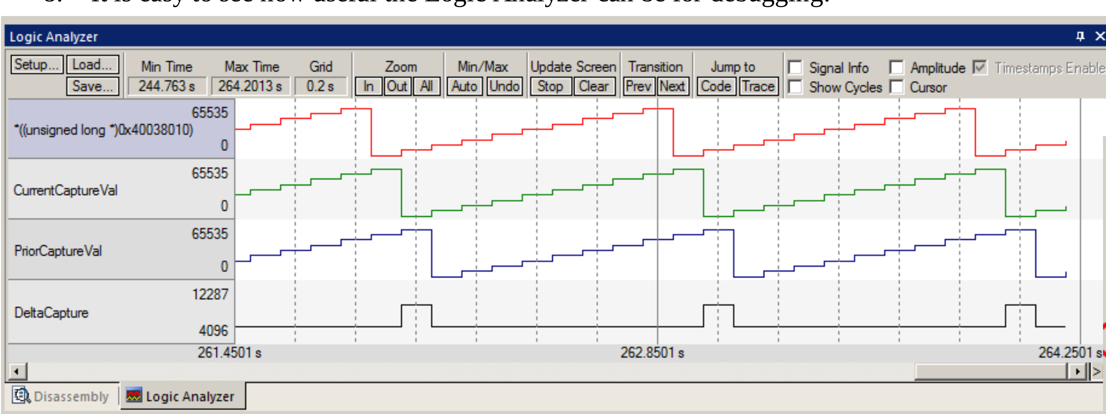
FTM Registers Display:

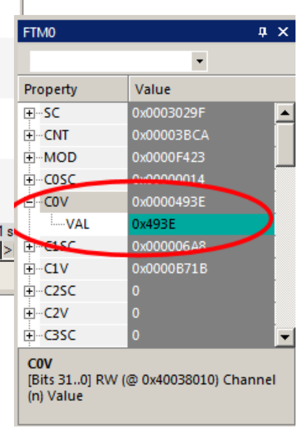
1. Select Peripherals/System Viewer/FTM0.
2. The FTM0 window will open. Various registers will update while the program is running.

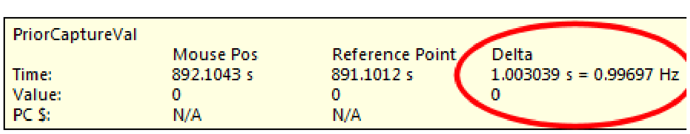
TIP: Memory operation fails displayed in the Command window are from reads or writes not permitted in FTM0.

Variables Display Logic Analyzer: SWV is used: any ULINK or J-Link is needed.

1. Stop the program and leave Debug mode .
2. Select your debug adapter in the Select target box. ULINK2 is used here. ULINK2
3. Build the source files . Enter Debug mode and click RUN .
4. The three variables mentioned above are displayed plus a third: *((unsigned long *)0x40038010).
5. In the LA, select Setup and highlight *((unsigned long *)0x40038010). Set Max: to 0xFFFF. Click Close.
6. This is the COV register address in the FTM0. This addressed is displayed at the bottom of the FTM0 window when COV is selected. It can them be added to the LA in the format provided. See below right:
7. Using Cursor and Signal Info, each step is 0.10 seconds. Period is 1 second. See below left:
8. It is easy to see how useful the Logic Analyzer can be for debugging.







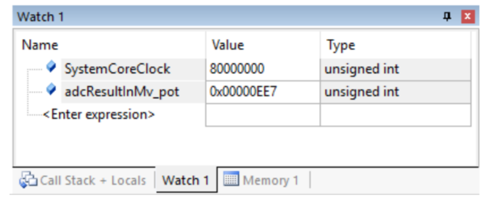
22) ADC – SW Trigger: This works with all debug adapters.

Open project, Build and Program Flash:

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\ADC\ADC.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here.
4. Compile the source files by clicking on the Rebuild icon.
5. Enter Debug mode. The Flash memory will be programmed. Progress will be indicated in the Output Window.
6. Click RUN.
7. As you rotate the pot R13, the ADC reads the pot position and the LEDs will sequence accordingly.

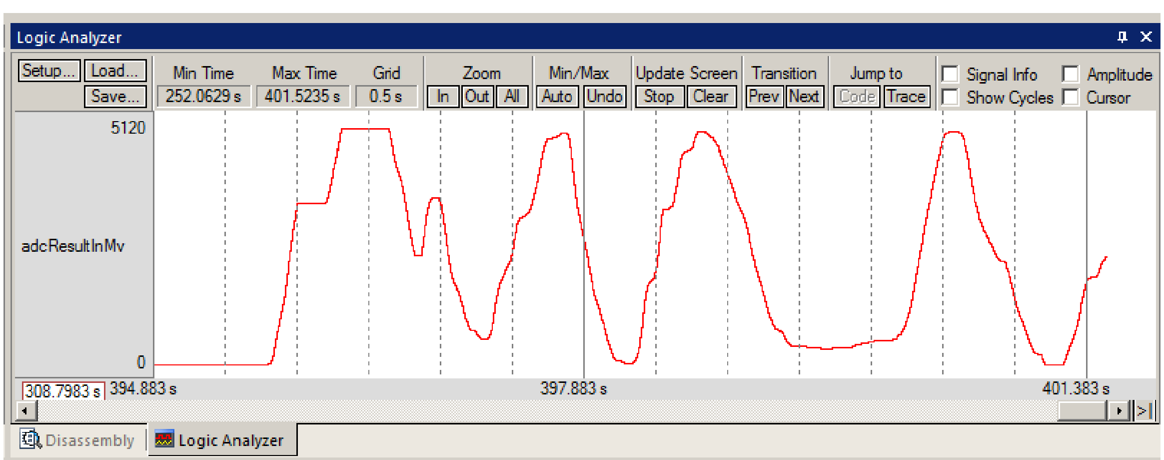
Reading the ADC Output:

1. adcResultInMv_pot is a global variable in main.c near line 18 that contains the value of the pot as read by the ADC.
2. Right click on the variable adcResultInMv_pot and enter it in Watch 1.
3. adcResultInMv_pot will update while the program runs. Vary the pot R13.



Display adcResultInMv_pot in the Logic Analyzer: SWV is used: any ULINK or J-Link needed.

1. Stop the program. Exit Debug mode. SWV is preconfigured for 80 MHz in this project.
2. Select your debug adapter in the Select target box. ULINK2 is used here. ULINK2
3. Near line 52 in main.c, uncomment this line: **for (int x = 0; x < 9000; x++);**
4. This slows the program down. There are so many Data Writes created that the SWO port is severely overloaded.
5. Build the source files. Enter Debug mode and click RUN.
6. Right click on adcResultInMv_pot and add it to Logic Analyzer (LA).
7. Click setup in the LA and set the Display Range: Max: to 0x1400. Click Close.
8. The LA will be displaying adcResultInMv_pot in real-time as shown below:



23) UART: **This works only with OpenSDA P&E mode.**

This example uses OpenSDA in P&E mode because it has a serial USB converter. Other debug adapters can be used if you are not interested in the USB serial port. Consider using the printf method described on the next page.

The P&E firmware for this mode is contained in C:\00MDK\Boards\NXP\S32K\UART\OpenSDA\. Program it into the S32K board. Directions are in the S32K-144 board lab on page 7: www.keil.com/appnotes/docs/apnt_299.asp

Open project, Build and Program Flash:

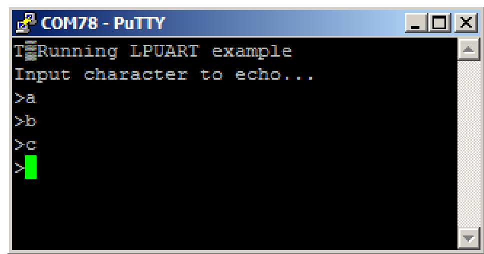
1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\UART\UART.uvprojx.
3. Choose the OpenSDA P&E debug adapter.
4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode. The Flash memory will be programmed. Progress will be indicated in the Output Window.

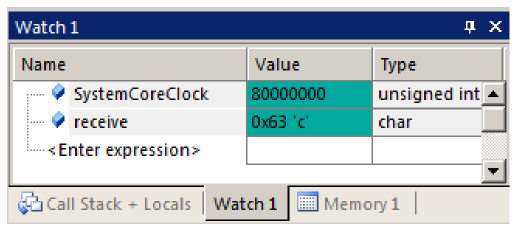
Open A COM Port on Your PC:

1. Configure a communication utility such as PuTTy to the appropriate Windows COM port. 9600 baud.

Modify Source and Configure Watch 1:

1. The declaration of the variable receive in LPUART.c has been changed to static: static char receive;
2. Right click on static variable receive near line 61 in LPUART.c and add it to Watch 1.
3. Click RUN .
4. The terminal program will display the text as shown here:
5. Type in any letter and it will be echoed.
6. Stop the program.
7. Watch 1 will contain the last letter you typed as shown:





µVision Communication Software:

In the Mange Run-Time Environment (MRTE) utility contains several methods of communications I/O as shown below:

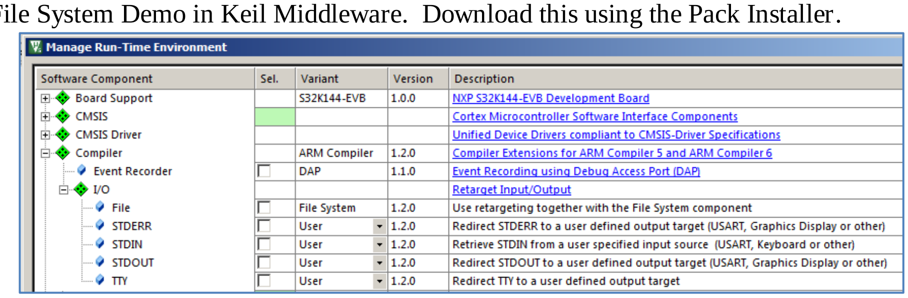
We have already seen this working in **printf using SWV ITM 0:** on page 9.

Obtaining a character typed into the Debug printf Viewer window from your keyboard:

It is possible for your program to input characters from a keyboard with the function ITM_ReceiveChar in core.CM4.h.

This is documented here: www.keil.com/pack/doc/CMSIS/Core/html/group__ITM__Debug__gr.html

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



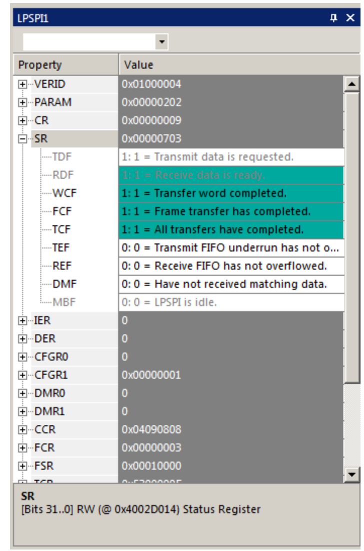
24) SPI: This works with all debug adapters.

Note: The S32K board must be powered with 12 volts to J16 for this exercise. Jumper J107 must be set to position 1-2 (towards the S32K chip) as shown in this photograph:

Open the Project, Build and Program the Flash:

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\SPI\SPI.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here.
4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode. The Flash memory will be programmed. Progress will be indicated in the Output Window.

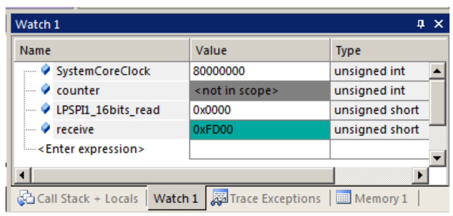
Open System Viewer LPSPI1 and Configure Watch 1 Window:



1. Select Peripherals/System Viewer/LPSPI1. This window opens:
2. We will put some interesting variables into Watch 1:
   a. counter (local main.c/main() near line 33)
   b. LSPI1_16bits_read (global in main.c near line 16)
   c. receive (local in LPSPI.c near line 67)
3. Right click on each variable and add it to Watch 1 as shown below:

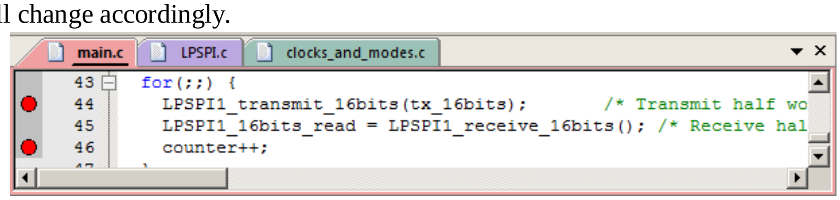
Add Breakpoints:

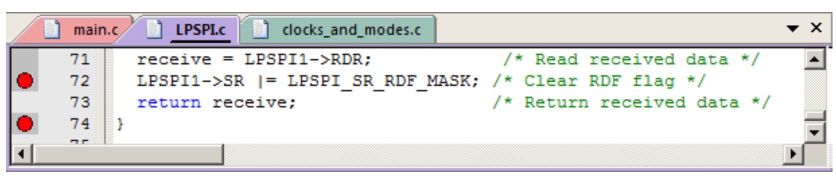
1. Add 4 breakpoints to main.c and LPSPI.c as shown below:



Run the Program:

1. Click RUN .
2. Each time you click RUN, variables will change accordingly.
3. Note the SR register changing in the LPSPI1 window.
4. The program is running correctly when receive contains 0xFDEF.
5. When you are finished, Stop the program . Exit Debug mode.





25) CAN General Information:

CAN Primer: www.keil.com/appnotes/docs/apnt_247.asp

S32K CAN Demo using two boards:

The projects CAN and CAN_FD described in the NXP Cookbook are designed to operate using two S32K boards. Each CAN_HI and each CAN_LO are connected together on J13.

A CAN test tool can be used in place of the second S32K board.

1. One board, designated as NODE_A sends a message with an ID of 0x555 on the CAN bus.
2. This is defined by #define NODE_A in FlexCAN.h. If NODE_A is not defined, then that board becomes NODE_B.
3. The second board, with NODE_A not defined, will then send a message of ID 0x511 on the CAN bus.
4. When NODE_A sees this message with an ID of 0x511, it sends another identical message with 0x555.
5. When 1,000 messages have been sent, the green LED is toggled. Then the process is repeated forever.
6. This screen shows the start sequence of the CAN messages displayed on a CAN analyzer:

Receive Overruns: 0 Errors: 0

No	Time (abs)	State	ID (hex)	D...	Data (hex)	ASCII
1	00:00:00.102		555	8	A5 11 22 33 44 55 66 77	.."3DUfw
2	00:00:14.210	S	511	7	57 6F 72 6B 73 20 21	Works !
3	00:00:14.211		555	8	A5 11 22 33 44 55 66 77	.."3DUfw

If no second board is connected:

If CAN_HI and CAN_LO are not connected and the program is running in NODE_A:

1. The CAN controller will send out the first message 0x555 forever since it never receives the ACK signal from another board or test tool.
2. This CAN signal can be viewed with an oscilloscope connected to CAN_HI and/or CAN_LO. Remember these signals are a differential pair.

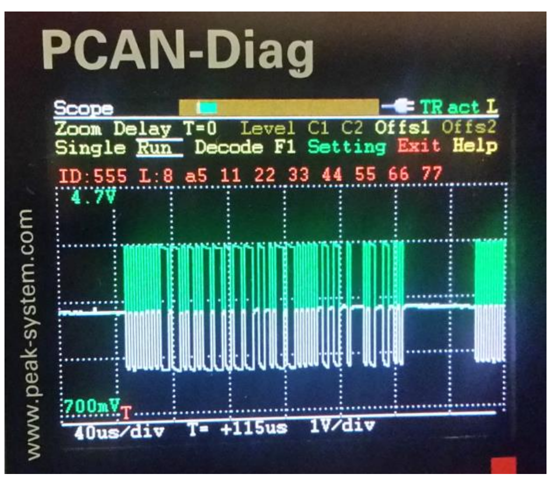
With a CAN test tool: A test tool was used in this document.

With a test tool you can display the CAN messages and insert a CAN message on the bus as either NODE_A or NODE_B.

Problems Getting CAN to work:

1. CAN_HI and CAN_LO are reversed.
2. The S32K board is not powered by 12 volts. The green LED D3 in the middle of the board must light. Check J107.
3. J107 is in the wrong position. For an external power supply, it must be in position 1-2.

Picture of CAN0:



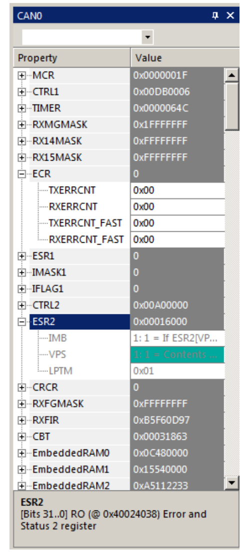
26) CAN: **This works with all debug adapters.**

1. The S32K board must be powered with 12 volts to J16 for this exercise. Jumper J107 must be set to position 1-2 (towards the S32K) as shown in this photograph:
2. Do not connect another CAN node to the S32K board yet.
3. If using OpenSDA connect your PC to the USB connector.
4. If you are using an external debug adapter connect it to J14 SWD connector.

Open project, Build and Program Flash:

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\FlexCAN\FlexCAN.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here. OpenSDA CMSIS-DAP
4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode. The Flash memory will be programmed.

Open System Viewer CAN0:

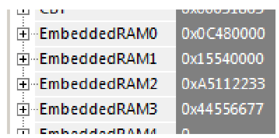


1. Select Peripherals/System Viewer/CAN0. This window opens:
2. Click RUN . CAN0 fills in as it is configured.
3. The program is sending a CAN message ID = 555 Data = A5 11 22 33 44 55 66 77
4. This message is sent endlessly since there is no other node to set the ACK bit. When an ACK bit is received, the program will wait for a message ID

Receive Overruns: 0 Errors: 0

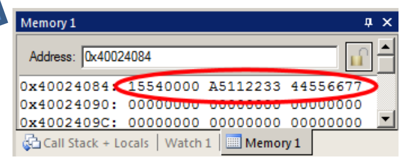
No	Time (abs)	State	ID (hex)	D...	Data (hex)	ASCII
12,188	00:00:03.370		555	8	A5 11 22 33 44 55 66 77	.."3DUfw
12,189	00:00:03.370		555	8	A5 11 22 33 44 55 66 77	.."3DUfw

5. In CAN0, this information is located in EmbeddedRAM1 through 3:
6. Note RAM1 0x1554 shifted right 2 bits = 0x555.
7. Highlight EmbeddedRAM1 to determine its physical address.
8. Enter this address (0x40024084) into Memory 1. Set to Unsigned Long.



Configure Watch 1 Window:

1. The variable rx_msg_count has been made static. It is located in main.c near line 31. This is to enable it to be always in scope.
2. Right click on rx_msg_count and add it to Watch 1.



Connect another S32K-EVAL board or a CAN test tool:

1. By default, this program sets the S32K board to Node A as in FlexCAN.h: **#define NODE_A**
2. Connect a S32K board or Test Tool to J13 CAN_HI and CAN_LO. RUN the program(s).
3. The S32K EVAL board will output a CAN frame with ID of 0x555.
4. If the Test Tool sends a CAN frame with ID 0x511, the S32K board with Node_A, A responds with an ID of 0x555:
5. rx_msg_count is incremented by 1. When 1,000 frames are received, the green LED toggles.
6. Stop the program . Exit Debug mode.

Receive Overruns: 0 Errors: 0

No	Time (abs)	State	ID (hex)	D...	Data (hex)	ASCII
1	00:00:00.102		555	8	A5 11 22 33 44 55 66 77	.."3DUfw
2	00:00:14.210	S	511	7	57 6F 72 6B 73 20 21	Works !
3	00:00:14.211		555	8	A5 11 22 33 44 55 66 77	.."3DUfw

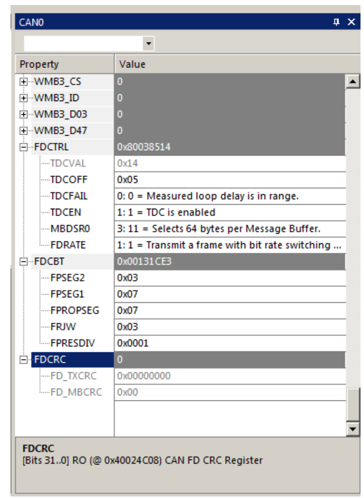
27) CAN FD: This works with all debug adapters.

1. The S32K board must be powered with 12 volts to J16 for this exercise. Jumper J107 must be set to position 1-2 (towards the S32K) as shown in this photograph:
2. Do not connect another CAN node to the S32K board yet.
3. If using OpenSDA connect your PC to the USB connector.
4. If you are using an external debug adapter connect it to J14 SWD connector.

Open project, Build and Program Flash:

1. Select Project/Open Project.
2. Open the file: C:\00MDK\Boards\NXP\S32K\FlexCAN_FD\FlexCAN_FD.uvprojx.
3. Choose your debug adapter accordingly: OpenSDA CMSIS-DAP is used here. OpenSDA CMSIS-DAP
4. Compile the source files by clicking on the Rebuild icon. .
5. Enter Debug mode. The Flash memory will be programmed.

Open System Viewer CAN0:



1. Select Peripherals/System Viewer/CAN0. This window opens: The CAN_FD registers are at the bottom as shown here:
2. Click RUN . CAN0 fills in as it is configured.
3. The program is sending a CAN message ID = 555 Data = A5 11 22 33 44 55 66 77
4. This message is sent endlessly since there is no other node to set the ACK bit. When an ACK bit is received, the program will wait for a message ID.

Connect another S32K-EVAL board or a CAN test tool:

1. By default, this program sets the S32K board to Node A as in FlexCAN.h: **#define NODE_A**
2. Connect a S32K board or Test Tool to J13 CAN_HI and CAN_LO. RUN the program(s).
3. The S32K EVAL board will output a CAN frame with ID of 0x555.
4. If the Test Tool sends a CAN frame with ID 0x511, the S32K board with Node_A, A responds with an ID of 0x555:
5. rx_msg_count is incremented by 1. When 1,000 frames are received, the green LED toggles.
6. Stop the program . Exit Debug mode.

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

CoreSight Debug Technology Summary:

1. **Basic CoreSight:** 6 hardware breakpoints, Watchpoints and program run/stop. These can be set/unset while the program is running.
2. **Debug Access Port (DAP):** JTAG or SWD modes available. Provides data updated while your program is running. Used by Watch, Memory, Peripherals and RTX System windows. Works with OpenSDA CMSIS-DAP, any ULINK and J-Link and OpenSDA P&E.
3. **Serial Wire Viewer (SWV):** Non-intrusive data trace. Displays interrupts, variables in graphical Logic Analyzer, data writes and CPU counters. printf display using no UART (printf is slightly intrusive). ULINK2, ULINK*pro* and J-Link. SWV updates while the program is running.
4. **ETM Instruction Trace:** S32K-148 and ULINK*pro* only. Provides a record of all instructions executed in their order. ETM is useful for debugging program flow problems and program crashes. ETM also supplies Code Coverage, Performance Analysis and Execution Profiler.

CoreSight Definitions:

Cortex-M0 and Cortex-M0+ may have only features 2) and 4) plus 11), 12) and 13) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 can have all features listed implemented. MTB is normally found on Cortex-M0+. It is possible some processors have all features except ETM Instruction trace and the trace port. Consult your specific datasheet.

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 TDO shares the same pin as SWO. The SWV data normally comes out the SWO pin or 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 accesses which provide on-the-fly memory accesses without the need for processor core intervention. µVision uses the DAP to update Memory, Watch, Peripheral and RTOS kernel awareness windows while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed.
   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 trace capability providing display of reads, writes, exceptions, PC Samples and printf.
6. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
7. **Trace Port:** A 4 bit port that ULINK*pro* uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by µVision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations. µVision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINK*pro* provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis. ETM is output on the Trace Port or accessible in the ETB (ETB has no Code Coverage or Performance Analysis).
10. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK*pro*. ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal user RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. Cortex-M3/M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 usually have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs. The CPU is halted before the instruction is executed.
13. **Watchpoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. There also referred to as Access Breaks in Keil documentation.

29) Document Resources: See www.keil.com/NXP

Books:

1. ***NEW!*** **Getting Started with MDK 5:** Obtain this free book here: www.keil.com/mdk5/
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. **The Definitive Guide to the Arm Cortex-M0/M0+** by Joseph Yiu. Search the web for retailers.
5. **The Definitive Guide to the Arm Cortex-M3/M4** by Joseph Yiu. Search the web for retailers.
6. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
7. MOOC: Massive Open Online Class: University of Texas: http://users.ece.utexas.edu/~valvano/

Application Notes:

1. ***NEW!*** Arm Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/safety
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer using Keil MCB170: www.keil.com/appnotes/files/apnt_247.pdf
4. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. GNU tools (GCC) for use with µVision https://launchpad.net/gcc-arm-embedded
8. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
9. Cortex-M Processors for Beginners: http://community.arm.com/docs/DOC-8587
10. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
11. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
12. FlexMemory configuration using MDK www.keil.com/appnotes/files/apnt220.pdf
13. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
14. ***NEW!*** Migrating Cortex-M3/M4 to Cortex-M7 processors: www.keil.com/appnotes/docs/apnt_270.asp
15. ***NEW!*** ARMv8-M Architecture Technical Overview www.keil.com/appnotes/files/apnt_291.pdf
16. ***NEW!*** Determining Cortex-M CPU Frequency using SWV www.keil.com/appnotes/docs/apnt_297.asp

Keil Tutorials for NXP Boards: www.keil.com/NXP

1. KL25Z Freedom www.keil.com/appnotes/docs/apnt_232.asp
2. K20D50M Freedom Board www.keil.com/appnotes/docs/apnt_243.asp
3. Kinetis K60N512 Tower www.keil.com/appnotes/docs/apnt_239.asp
4. Kinetis K60D100M Tower www.keil.com/appnotes/docs/apnt_249.asp
5. Kinetis FRDM-K64F Freedom www.keil.com/appnotes/docs/apnt_287.asp
6. Kinetis K64F120M Tower www.keil.com/appnotes/docs/apnt_288.asp
7. S32K-144 EVB Tutorial: www.keil.com/appnotes/docs/apnt_299.asp
8. S32K-148 EVB using ETM: www.keil.com/appnotes/docs/apnt_305.asp

Useful Arm Websites:

1. ***NEW!*** CMSIS 5 Standards: https://github.com/ARM-software/CMSIS_5 and www.keil.com/cmsis/
2. Forums: www.keil.com/forum http://community.arm.com/groups/tools/content https://developer.arm.com/
3. Arm University Program**:** www.arm.com/university. Email: university@arm.com
4. mbed™: http://mbed.org

30) Keil Products and contact information: See www.keil.com/NXP

Keil Microcontroller Development Kit (MDK-ARM™) for NXP processors:

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

For the latest MDK details see: www.keil.com/mdk5/selector/

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

USB-JTAG/SWD Debug Adapter (for Flash programming too)

- **ULINK2** - *(ULINK2 and ME - SWV only – no ETM)* **ULINK-ME** is equivalent to a ULINK2.
- *New* **ULINK*plus***- Cortex-Mx High performance SWV & power measurement.
- **ULINK*pro*** - Cortex-Mx SWV & ETM instruction trace. Code Coverage and Performance Analysis.
- **ULINK*pro* D** - Cortex-Mx SWV no ETM trace ULINK*pro* also works with Arm DS-5.

You can use OpenSDA on the S32K board. For Serial Wire Viewer (SWV), a ULINK2, ULINK-ME or a J-Link is needed. For ETM support, a ULINK*pro* is needed. OS-JTAG or OpenSDA do not support either SWV or ETM.

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

For the Arm University program: go to www.arm.com/university Email: university@arm.com

All software products include Technical Support and Updates for 1 year. This can easily be renewed.

Keil RTX™ Real Time Operating System

- RTX is provided free as part of Keil MDK.
- No royalties are required. It has a BSD or Apache 2.0 license.
- RTX source code is included with all versions of MDK.
- Kernel Awareness visibility windows are integral to µVision.
- https://github.com/ARM-software/CMSIS_5

For the entire Keil catalog see www.keil.com or contact Keil or your local distributor. For NXP support: www.keil.com/NXP

For Linux, Android, bare metal (no OS) and other OS support on NXP i.MX and Vybrid series processors please see DS-5 and DS-MDK at www.arm.com/ds5/ and www.keil.com/ds-mdk.

Getting Started with DS-MDK: www.keil.com/mdk5/ds-mdk/install/

For more information:

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

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

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

Forums: www.keil.com/forum http://community.arm.com/groups/tools/content https://developer.arm.com/

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

Cortex Intelligent Processors by ARM®

arm KEIL