

### Introduction:

This note describes the process of operating ARM® Keil™ MDK toolkit featuring μVision® and Microsemi's (Actel) new SmartFusion family which contains an embedded ARM® Cortex™-M3 processor. SmartFusion implements ARM Serial Wire Viewer (SWV) debug technology that is available on Cortex-Mx processors and supported by Keil. This note describes how to get all the components of this important technology working with μVision. **This article needs MDK® 4.20 or later.**

### Keil μVision:

μVision is the IDE and is a component of the Keil MDK™ toolkit development system. Keil supports all 8051, ARM7, Cortex-M1 and Cortex-M3 processors as used in various Microsemi (Actel) products.

MDK-ARM™ components include the μVision 4 IDE, ARM Realview® compiler, assembler and Keil RTX™ RTOS.

The Keil ULINK® JTAG/SWD adapter family includes the ULINK2, ULINK-ME and the ULINK*pro*.

RL-ARM™ contains the sources for RTX plus a TCP/IP stack a FLASH file system, USB and CAN drivers. At this time, RTX, TCP/IP and Flash file support is available. USB and CAN support is not yet completed.

### Why Use Keil MDK ?

MDK provides these features particularly suited for Microsemi FPGAs:

1. μVision IDE with Integrated Debugger, Flash programmer and the RealView ARM compiler.
2. Support for ARM7, Cortex-M1 and Cortex-M3. Nothing more to buy. 8051 support is separate and uses μVision.
3. A full feature RTOS is included with MDK: RTX by Keil. No royalty payments are required.
4. Serial Wire Viewer trace capability is included for SmartFusion Cortex-M3.
5. RTX Kernel Awareness window. It is updated in real-time and uses no system resources.
6. Choice of USB adapters: ULINK2, ULINK-ME, ULINK*pro* and Segger J-Link (black version).
7. Kernel Awareness for Keil RTX, CMX, Quadros and Micrium. All RTOSs will compile with MDK.
8. Keil Technical Support is included for one year. This helps you get your project completed faster.

### Serial Wire Viewer:

**Serial Wire Viewer** (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM, CPU counters and a timestamp. This information comes from the ARM CoreSight™ debug module integrated into the SmartFusion Cortex-M3. SWV does not steal any CPU cycles, is non-intrusive and requires no stubs in your source code.

**This document details these features:**

1. Serial Wire Viewer (SWV).
2. Real-time Read and Write to memory locations for Watch, Memory and RTX Tasks windows.
3. Breakpoints and Watchpoints (Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTOS – RTX.
5. Keil TCP/IP stack – is one component of RL-ARM. An http server example is described on page 28.

### Keil Contact Information: [www.keil.com](http://www.keil.com)

#### USA: North and South America:

Keil, An ARM Company  
Plano, Texas  
800-348-8051 (Toll Free)  
[sales.us@keil.com](mailto:sales.us@keil.com)  
[support.us@keil.com](mailto:support.us@keil.com)

#### Europe and Asia:

Keil, An ARM Company  
Grasbrunn, Germany  
+49 89/456040-20  
[sales.intl@keil.com](mailto:sales.intl@keil.com)  
[support.intl@keil.com](mailto:support.intl@keil.com)

<b>Part 1: Introduction and Getting Ready:</b>	<b>3</b>
1) Serial Wire Summary:	3
2) What the trace is useful for:	3
3) What kind of data can the Serial Wire Viewer display ?	3
4) Hardware: The SmartFusion boards and debug adapters:	4
5) Software: Keil MDK 4.20:	5
6) Connecting the Hardware:	5
<b>Part 2: Tracing Examples and Useful Information:</b>	<b>6</b>
1) The Keil Blinky example:	6
<b>Part 3: Configuring the Serial Wire Trace:</b>	<b>6</b>
1) Debug Ports: JTAG, SWD, SWJ and SWO. A short tutorial:	6
2) Selecting SWD or JTAG ports:	7
3) Configuring the Trace:	8
4) Trace Configuration Fields:	9
5) Program STOP Control:	11
6) Debug Information Display:	11
<b>Part 4: The Rest of the Tracing Examples:</b>	<b>12</b>
1) Creating a Global Variable (to use in our examples):	12
2) Watch window:	13
3) Memory Windows:	13
4) Logic Analyzer:	14
5) Data Reads and Writes in the Trace Records window:	15
6) Watchpoint Example (Access Breaks):	16
7) Breakpoints (Execution Breaks):	17
8) Exceptions: Tracing:	18
9) PC Samples: Tracing:	19
10) Debug (printf) Viewer:	20
11) Opening and using RTX Kernel Awareness: The RTX Viewer:	21
12) Logic Analyzer Window: View variables real-time in a graphical format:	23
<b>Part 5: ...more Useful Information</b>	<b>24</b>
1) Creating a new $\mu$ Vision project from scratch:	24
2) Running Blinky from RAM:	25
3) SmartFusion eNVM (Flash) Programming:	27
4) TCP/IP: Keil http demonstration:	28
5) ARM JTAG/SWD adapter connector schematics and the New ULINK-ME:	29
6) Acronym List:	30
7) Overloading the SWO pin:	30
8) Top Six Reasons why you can't get SWV working:	31
9) Programming the A2F-EVAL-KITBoard (F2 device) (FPGA Flash)	32

## 1) Serial Wire Summary:

We have three basic debug systems as implemented in SmartFusion Cortex-M3 devices:

1. SWV and ITM data output on the SWO pin located on the standard JTAG debug connector.
2. ITM is a printf type viewer. ASCII characters are displayed in the Debug printf Viewer in  $\mu$ Vision.
3. Memory Reads and Writes in/out the JTAG/SWD ports. The Memory and Watch windows use this technology.
4. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.

These are all completely controlled through  $\mu$ Vision via a ULINK.

## 2) Usefulness of the 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 task 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:

- 1) Pointer problems.
- 2) Illegal instructions and data aborts (such as misaligned writes).
- 3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs)
- 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. Needs ETM to be most useful.*
- 8) Communication protocol and timing issues. System timing problems.
- 9) Profile analysis and code coverage. Available only with ETM trace.

## 3) 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.
- 6) CPU counters. Folded instructions, extra cycles and interrupt overhead.

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

- 7) Can't see local variables. (just make them global or static).
- 8) Can't see register to register operations. PC Samples records some of the instructions but not the data values.
- 9) SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU and SWV can only see CPU actions.

## 4) Hardware:

We will examine two SmartFusion boards and will focus on the smaller one: the A2F-EVAL-KIT.

**A2F-DEV-KIT:** See Figure 1. It is connected to a ULINK-ME in the upper right corner. This is referred by Microsemi as the SmartFusion **D**evelopment Kit.

This kit is populated with a SmartFusion A2F500M3G device. This board was once available with an A2F200 device but this option is no longer available. The A2F500 Serial Wire Viewer works at full speed. The A2F200 SWV runs at 98 KHz.

**A2F-EVAL-KIT:** See Figure 2. It is shown with a Keil ULINK2 USB to JTAG/SWD adapter. This is referred by Microsemi as the SmartFusion **E**valuation Kit. This kit is populated with an AF200M3G. The A2F200 device has the Serial Wire Viewer (SWV) speed limited to 98 KHz.

**Debug Adapters:** A Keil ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link (black case, version 6 or later) can be used. ULINK2 and ULINK-ME are functionally the same. The ULINK*pro* is the same as the other ULINKs but faster flash programming and with the addition of ARM ETM trace support. SmartFusion does not support ETM.

**TIP:** Earlier versions of ULINK-ME provide 3.3 volts power to a target board through JTAG connector pin 1. You might want to remove this by cutting the pin 1 lead or remove the diode on the ME that is connected to pin 1. SmartFusion boards have complex power supplies that can be confused when 3.3 volts is applied before 5 volts. ULINK2 is not affected.

**Blinky Alert:** You do not need a FPGA design programmed into SmartFusion parts to program and run the Cortex-M3 processor with  $\mu$ Vision. You do need the FPGA programmed in order to have the LEDs blink as used by the Keil examples. The file Keil\_SmartFusion\_tutorial\_PF.zip available from the Actel website contains the design file mss\_101.stp. This file is also attached to this pdf file. Click on the paper clip icon in the bottom left hand corner.

If the LEDs do not light and the Blinky program is running correctly, program mss\_101.stp into the SmartFusion FPGA.

Instructions to program mss\_101.stp into the FPGA Flash (eNVM) using the Microsemi program FlashPro are in Section 9) on page 32 on this document.

FlashPro is available stand-alone or as part of the Libero IDE FPGA development software. Both are free from the Actel.com website.

mss\_101.stp and instructions are also available from [www.keil.com](http://www.keil.com) in the location where this document is stored.

**TIP:** You must change two jumpers to program the FPGA Flash. Remember to switch these jumpers back in order to connect successfully with a ULINK. These jumpers are JP7 and JP10 in the EVAL board. This is a common error.

On the DEV-KIT, switch SW9 must be set to "S/W Debug ON", JP& set to RVI and JP6 to V1P5\_EXT in order for ULINK to work with the examples in this lab. The other jumpers on both boards must be in their default positions.



Figure 1: DEV-KIT with Keil ULINK-ME



Figure 2: EVAL-KIT with Keil ULINK2

## 5) Software:

You must use Keil MDK 4.20 or later to get all the required SmartFusion files. The evaluation version (MDK-Lite) is available free from [www.keil.com](http://www.keil.com). A license is not needed. You can compile code up to 32K. The Keil examples are all less than this.

Please install MDK into the default directory of C:\Keil to easier follow the directories in this note. Normally, you install MDK anywhere. You can also install multiple copies and they will not interfere with each other.

Keil provides two examples: Blinky and one using the RTOS: RTX\_Blinky. These are found in C:\Keil\ARM\Boards\Actel\SmartFusion. A TCP/IP example is available on [www.keil.com](http://www.keil.com). See page 28 for details.

## 6) Connecting the Hardware:

**A2F-EVAL-KIT:** the Small Blue Board: See Figure 3.

1. Connect the debugging adapter to J3 RVI Header, the standard JTAG adapter as shown.
2. JP7 and JP10 jumpers set to position 2-3. They are labeled RVI and M3 respectively.
3. JP6 2-3, J6 installed, J5 and J8 are not installed.
4. Connect a USB cable to power the board to J14. Some LEDs will illuminate.

**A2F-DEV-KIT:** the Big Green Board: See Figure 4.

1. Connect the ULINK to J3, the standard JTAG adapter as shown in Figure 4.
2. SW9 to S/W Debug ON. (the right most position)
3. JP7 set to 2-3 RVI. (the other position is to program the FPGA Flash)
4. JP6 must be in position 2-3 V1P5\_EXT.
5. JP5 both jumpers to C1 – Single Prog. See Figure 4 for a close-up of these jumpers.
6. Power the board with the Microsemi provided 5 volt AC adapter. If you see a Found New Hardware Wizard message: click Cancel to ignore it as we will not be using the on-board USB port.

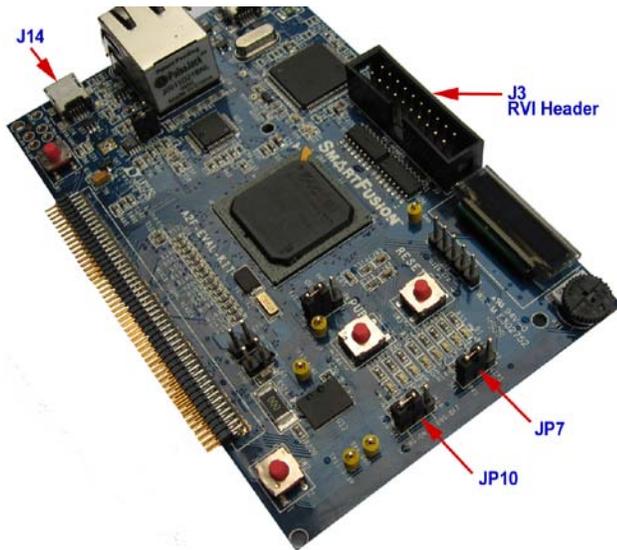


Figure 3: EVAL-KIT Jumper Settings for Keil ULINK operation.

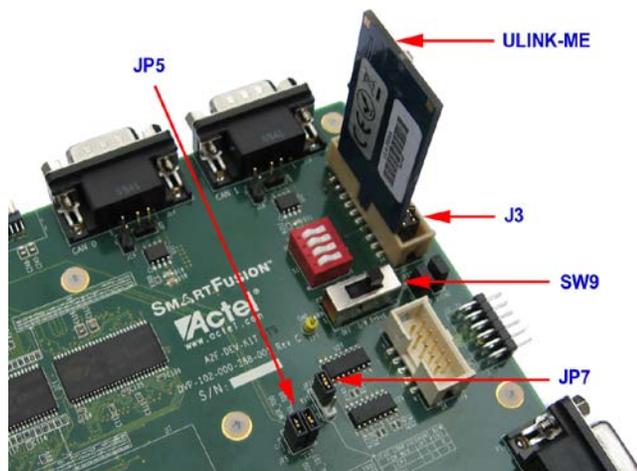


Figure 4: DEV-KIT Jumpers Settings for ULINK operation.

**Software Update:** MDK 4.20 and later contains several enhancements. The first are updated CMSIS compliant header files from Microsemi. The other is a FLASH programming algorithm to move the example files from the area of 0x6000\_0000 to 0x0. This is more consistent with typical Cortex-M3 programming. Programs must be run at less than 0x2000\_0000 in order for the hardware breakpoints to work reliably. You should use MDK 4.20 or later !

MDK-Lite is available free on [www.keil.com/arm](http://www.keil.com/arm) and click on the Download icon. No license is required.

RTX RTOS is not configured for SmartFusion in MDK 4.20. It is compatible with MDK 4.13a. RTX will be reconfigured to work with SmartFusion after 4.20. Contact Keil Technical Support for the status of this project. You can access the solution to use RTX with MDK 4.20 at [www.keil.com/support/docs/3551.htm](http://www.keil.com/support/docs/3551.htm). If you do not use RTX, there are no problems.

## 1) Blinky example program using Microsemi SmartFusion

Now we will run the Keil MDK development system using a SmartFusion evaluation board.

You do not need anything programmed in the FPGA Flash to run Cortex-M3 programs.

1. Connect the hardware as in Figure 1 or 2. Use a ULINK2 or ULINK-ME.
2. Start  $\mu$ Vision by clicking on its desktop icon. 
1. Select Project/Open Project from the main menu.
2. Open the file C:\Keil\ARM\Boards\Actel\SmartFusion\Blinky\Blinky.uvproj.
3. On DEV-KIT change the "6" in this line in Blinky.c to a 0, 1, 2 or 3. `const unsigned long led_mask[] = { 1 << 6 };`
4. Compile the source files by clicking on the Rebuild icon. 
5. Program the SmartFusion eNVM flash by clicking on the Load icon:  Progress is indicated in bottom left corner.
6. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode notice appears.
7. Click on the RUN icon to start the Blinky program.  **Blinky is now running !**
8. If the FPGA is programmed to connect the GPIO port to the LEDs, at least one LED will blink.
9. Stop the program with the STOP icon. 

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

**LEDs:** No LEDs will blink on the EVAL-KIT or the DEV-KIT when the Blinky program is running unless the FPGA is programmed with a hardware design to connect the GPIO port to the LED. This is the one exception where you need some design programmed into the FPGA. See under 4) Hardware on page 4 for a suitable design.

### Something Extra to do: setting a Hardware Breakpoint (on-the-fly):

1. Run the program again. 
2. Open the Blinky.c tab. It is probably already open.
3. Scroll down to near Line 43: `num += dir;`
4. Note on the left of the line numbers are darker grey blocks. This indicates there is assembly code present and you can set a hardware breakpoint on this line. You can also see these in the Disassembly window.
5. *While the program is still running*, double-click to the left of the 43 and a red block will be created. This is a hardware breakpoint. SmartFusion has 6 but  $\mu$ Vision uses one or two and sometimes more.
6. The cyan arrow is where the source and disassembly windows are matching. You set this with your mouse.
7. Note that the program will soon stop at line 43. The yellow arrow is the current program counter position. This will be the next instruction executed when the CPU is started again.

```

41  for (;;) {                               /* Loop forever */
42  /* Calculate 'num': 0,1,...,LED_NUM-1,LED_NUM-1,...,1,0,0,... */
43  num += dir;
44  if (num == LED_NUM) { dir = -1; num = LED_NUM-1; }

```

**TIP:** You can set and unset breakpoints on-the-fly. No need to stop the CPU. This is an important feature.

8. Double-click on the red box to remove the breakpoint.

## 1) Debug Ports: JTAG, SWD, SWJ and SWO:

**JTAG:** The JTAG port is used to access the internal CoreSight debug module from the outside world and uses from 4 to 5 pins. A USB to JTAG adapter such as the ULINK2 connects the debugger software to the target chip.

**Serial Wire Debug (SWD):** SWD is a two wire alternative to the JTAG port. SWD has a bi-directional signal plus a clock. It shares the JTAG signals TMS and TCK respectively on the JTAG connector. SWD can do everything JTAG can except for boundary scan. You must use SWD to use the Serial Wire Viewer. It is called SW in  $\mu$ Vision.

**SWJ:** This switches between JTAG and SWD mode. SWJ must be enabled to select either JTAG and SWD.

**SWO:** Serial Wire Output. This is a single pin high speed output that provides the Serial Wire Viewer trace data. It is easily overloaded with excessive amounts of data. SWO shares a pin with the JTAG signal TDO.

**JTAG and SWD:** The example above uses the SmartFusion JTAG port to communicate with the ARM CoreSight debugging module. To use Serial Wire Viewer, we will need to use the Serial Wire Debug (SWD) port. SWD can do everything JTAG can except Boundary Scan. See the next page to activate it.

**TIP:** SmartFusion eval boards have a 510 ohm resistor from the nTRST pin (# 3) on the JTAG connector to ground that might cause a Debug adapter to not function with JTAG selected. This mostly affects the ULINK $pro$ . Either remove this resistor or put a 330 ohm between nTRST (pin 3) and VTref (pin 1). Alternatively, just use SWD. This is the easiest fix.

## 2) Selecting SWD or JTAG:

You must use Single Wire Debug (SWD) mode to use Serial Wire Viewer. This is configured in  $\mu$ Vision as shown in Figure 5. You must first have a ULINK connected to a Cortex-M3 target.

- 1) If necessary, stop the CPU  and exit debug mode. 
- 2) Click on the Target Options icon  or select “Project/Options for Target” or pressing Alt+F7.
- 3) Click on the Debug tab.
- 4) Click Settings beside the USE: ULINK Cortex Debugger box to open Figure 5.

The box labeled SW Device (or labeled JTAG Device Chain if JTAG is selected) in Figure 5 will display the CoreSight devices detected in the processor or an error if none are found.

- 5) Select SWJ and SW in the Port: box. Confirm SW-DP is shown as in Figure 5. You must see SW-DP before you continue to Step 6.

Figure 5 shows that the ULINK is now successfully been connected to the CoreSight SWD port.

**Urgent TIP:** If you don't see either SW-DP or JTAG-DP in this window and get an error message or device not found – stop right now and fix this. This section *must* work before you can continue any debugging. **Make sure the jumpers are set correctly** and the board is properly powered. See **TRAP** below. Try re-powering the board.

**TRAP:** Top reason people can't get SW and/or JTAG working is a wrong setting of the jumpers JP7 and JP9 (on EVAL board) and JP5, JP6, JP7 and SW9 on the larger DEV board. This is because these must be switched to enable FPGA programming. By default they are set to FPGA programming and not for standard SWD or JTAG operation.

- 6) Click on OK twice to return.

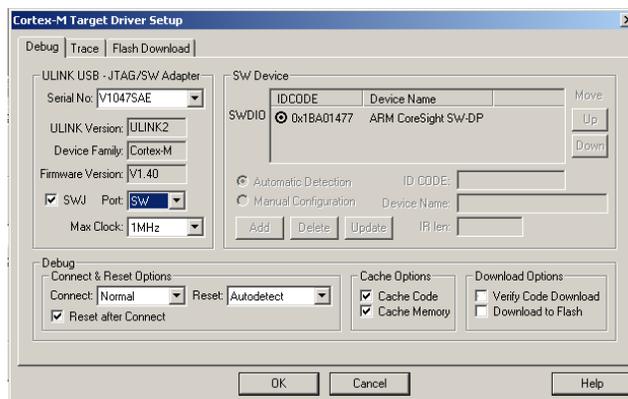


Figure 5: Selecting SWD Mode

**TIP:** To refresh this window switch between JTAG and SW in the Port: drop down menu.

**TIP:** The box “Verify Code Download” must be unchecked for SmartFusion. Verify Code is done inside the programmer.

### 3) Configuring the Trace:

In order to see the Serial Wire Viewer features, the Trace must be configured as in Figure 6.

- 1) If necessary, stop the CPU  and exit debug mode. 
- 2) Click on the Target Options icon  or select “Project/Options for Target”.
- 3) Click on the Debug tab.
- 4) Click Settings beside the USE: ULINK Cortex Debugger box. Confirm SWJ and SW are selected.
- 5) Click on the Trace tab to open Figure 6.
- 6) For A2F200 parts set Core Clock to 0.098 MHz. For the A2F500, set Core Clock to 50 MHz.
- 7) Select Trace Enable and EXCTRC.
- 8) Set the Timestamps Prescaler to 1 to lessen data overruns.
- 9) Unselect Periodic and on Data R/W Sample. Window will look like Figure 6.
- 10) Click on OK twice to return to the main screen.
- 11) The Serial Wire Viewer is now activated.
- 12) Click on File/Save All to save these settings.

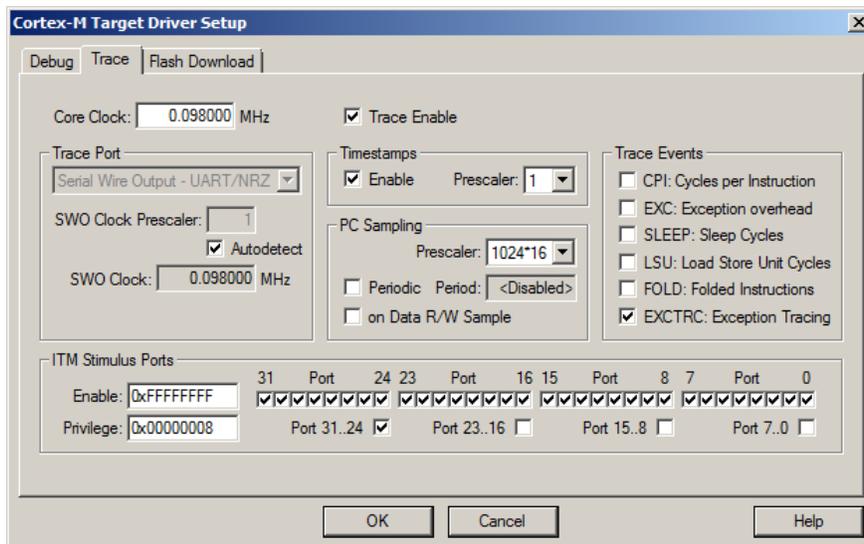


Figure 6: Trace Configuration window.

**TIP:** How to open this window again to modify trace selections:

- 1) **In Edit mode:** The Target Options icon as already described. 
- 2) **In Debug Mode:** Select Debug/Debug Settings from the main menu.

You select various SWV elements in Figure 6. It is important to note that it is easy to overload the single wire Serial Wire Output (SWO) pin. This is more pronounced with the A2F200 than the A2F500 device. The general rule is to activate only those features you need.

**TIP:** You can only enable/disable trace from Edit mode. You can change the rest while in either mode.

The next page describes in detail the various fields of the Trace Configuration window.

Two pages after that describes some of the attributes of Serial Wire Viewer and the Debug Access Port (DAP) data read and write operations.

### Testing SWV:

Run the program and open the Trace Records window. It should look similar to Figure 21 on page 18. If it is not working correctly, usually you get either nothing or spurious frames. If you see ITM frames with numbers other than 0 or 31, or other frames that do not make sense, the Core Clock speed is probably wrong. This test assumes you have SYSTICK running.

## 4) Trace Configuration Fields:

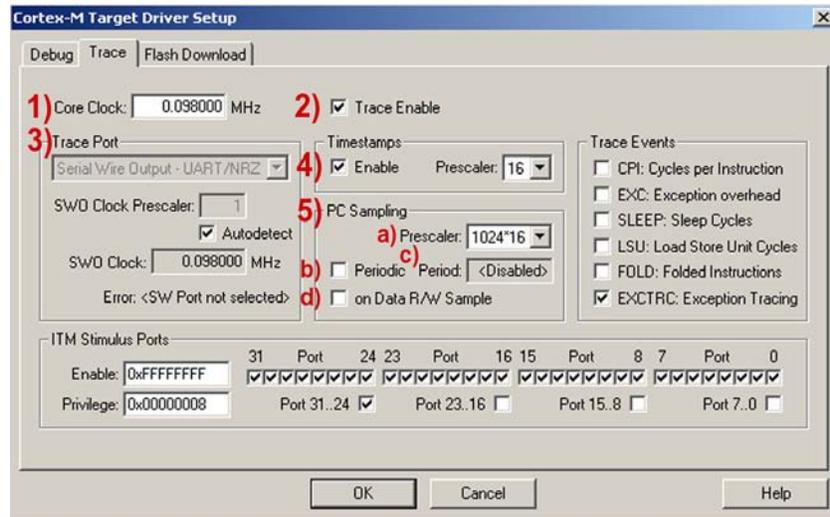


Figure 7: Trace Configuration window Fields 1) through 5).

- 1) **Core Clock:** The CPU clock speed for SWV. SWO Clock signal is derived from and is a ratio of the Core Clock. Current ratio for SmartFusion A2F200 is CPU/1024. 100 MHz/1024 = 98 KHz. For the A2F500 the ratio is 100/2 = 50 MHz. If you change the speed the CPU is running at, these values will obviously change.
- 2) **Trace Enable:** Enables SWV and ITM which is essentially everything on this window except Trace Port. This does not affect the DAP Watch and Memory window display updates.
- 3) **Trace Port:** Selects the SWO trace output UART or Manchester protocol.
  - a. **Serial Wire Output – UART/NRZ:** ULINK2 and ULINK-ME must use this. UART/NRZ encoding is not supported by ULINKpro. An error will result when you enter debug mode with ULINKpro.
  - b. **Serial Wire Output – Manchester:** Use Manchester encoding. ULINKpro only.
- 4) **Timestamps:** Enables timestamps and selects a Prescaler. 1 is the default. Selecting a higher value can, but not always lessen SWO overloads. Completely disabling the timestamps can lessen data overruns but can disable other SWV features. It is worth a try if you are having overload problems.
- 5) **PC Sampling:** Samples the program counter and displays in the Trace Records window.
  - a. **Prescaler** 1024\*16 (the default) means every 16,384<sup>th</sup> PC is displayed. The rest are lost.
  - b. **Periodic:** Enables PC Sampling.
  - c. **Period:** Automatically derived from Prescaler and Core Clock settings.
  - d. **On Data R/W Sample:** Displays the address of the instruction that made a data read or write of a variable entered in the Logic Analyzer in the Trace Records window. This is not connected with PC Sampling.

**TIP:** It is important to ensure the Serial Wire Output (SWO) pin is not overloaded.  $\mu$ Vision will alert you when an overflow occurs with a “X” in the Trace Records window or with a “D” or a “O” in the ULINKpro Instruction Trace window.  $\mu$ Vision easily recovers from these overflows and immediately continues displaying the next available trace frame. Dropped frames are somewhat the normal situation especially with many data reads and/or writes.

**TIP:** ULINKpro can process SWV information much faster than the ULINK2 or ULINK-ME can. This results in fewer dropped frames especially with high data transfer rates out the SWO pin. Data overruns are often associated with a fast stream of data reads and writes which are set in the Logic Analyzer. Minimize these issues by displaying only the information you really need.

The A2F500 can output SWV frames much faster than the A2F200 therefore much less susceptible to overloading and dropping frames at high data rates.

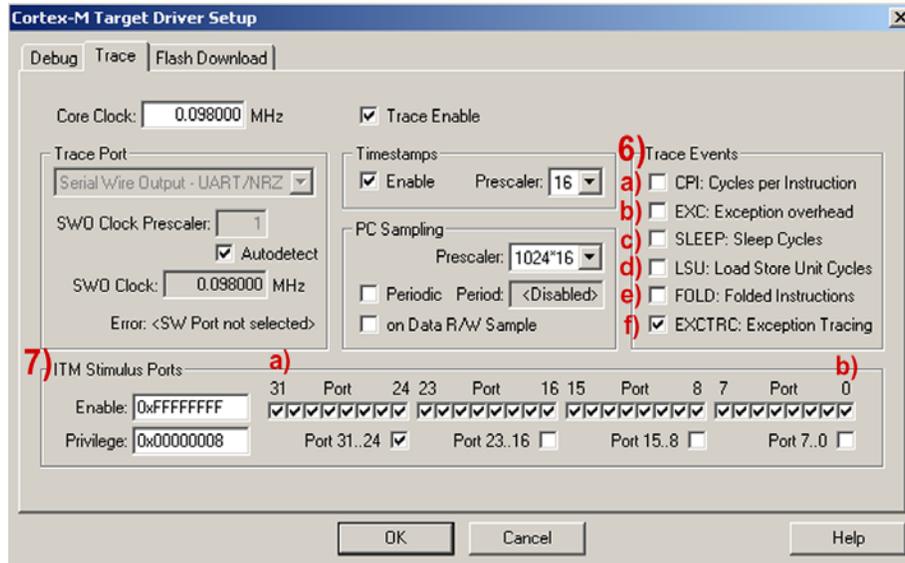


Figure 8: Trace Configuration window Fields 6) through 8)

- 6) **Trace Events:** Enables various CPU counters. All except EXCTRC are 8 bit counters. Each counter is cumulative and an event is created when this counter overflows every 256 cycles. These values are displayed in the Counter window. The event created when a counter wraps around is displayed in the Trace Records window or the Instruction Trace window (*ULINKpro*). Event Counters are updated using the DAP and not SWV. These events are memory mapped and can be read by your program or the  $\mu$ Vision Memory window.
- a. **CPI: Cycles per Instruction:** The cumulative number of extra cycles used by each instruction beyond the first one plus any instruction fetch stalls.
  - b. **Fold:** Cumulative number of folded instructions. This will result from a predicted branch instruction removed and flushed from the pipeline giving a zero cycle execution time.
  - c. **Sleep:** Cumulative number of cycles the CPU is in sleep mode. Uses FCLK for timing.
  - d. **EXC:** Cumulative cycles CPU spent in exception overhead not including total time spent processing the exception code. Includes stack operations and returns.
  - e. **LSU:** Cumulative number of cycles spent in load/store operations beyond the first cycle.
  - f. **EXCTRC:** Exception Trace. This is different than the other items in this section. This enables the display of exceptions in the Instruction Trace and Exception windows. It is not a counter. This is a very useful feature and is often used in debugging.
- 7) **ITM Stimulus Ports:** Enables the thirty two 32 bit registers used to output data in a *printf* type statement to  $\mu$ Vision. Port 0 is used for the Debug (*printf*) Viewer, Port 31 is used for the Keil RTX real-time kernel awareness window. Only Ports 0 and 31 are currently implemented in  $\mu$ Vision and should normally be checked. Ports 1 through 30 are not currently implemented and are Don't Care.
- a. **Port 31:** Enables the ITM port used for the RTX Viewer.
  - b. **Port 0:** Enables the ITM port used for the Debug (*printf*) Viewer. A small amount of instrumentation code is needed in your project. See 10) Debug (*printf*) Viewer in Part 4 for information on using this feature.

## 5) Program STOP Control:

**Breakpoints:** SmartFusion has six hardware breakpoints. The  $\mu$ Vision debugger will use one, perhaps two for its operation.  $\mu$ Vision will alert you if you attempt to use more breakpoints than are available. Breakpoints can be set on-the-fly while the program is running as long as the program is located at addresses 0x2000 0000 or lower. For addresses greater than this, you must stop the program to set a hardware breakpoint.

**Watchpoints:** Sometimes referred to as Access Breaks in Keil documents. These can be thought of conditional breakpoints. Four are available in SmartFusion processors. They can't be set on the fly. You must stop the program execution.

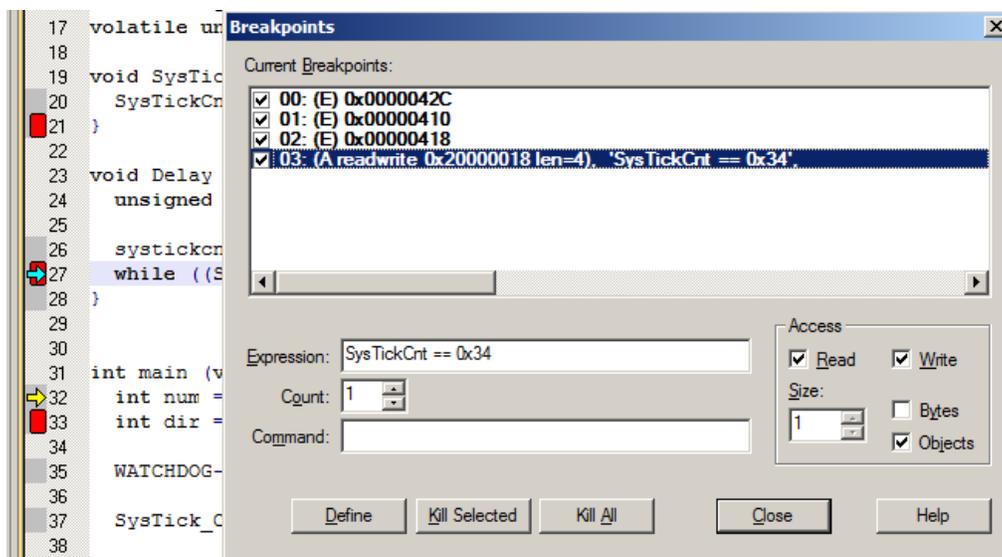
**TIP:** See Watchpoints and Breakpoints in Part 4.

**TIP:** Breakpoints and Watchpoints are different – they originate in different parts of CoreSight. For instance, killing all breakpoints inside  $\mu$ Vision does not kill the Watchpoints.

Breakpoints can be set by clicking in the left column where assembly instructions are indicated as shown below. Or they can be entered in the Breakpoints window. Watchpoints (also known as Access Points) must be entered in the Breakpoints window.

This window is opened only in Debug mode by selected Debug/Breakpoints or pressing Ctrl-B.

Shown below are three breakpoints and one watchpoint. The three breakpoints are shown as red blocks in the source window on the side. You can temporarily unselect entries in this window.



## 6) Debug Information Display:

When we speak of “trace” we can refer to Serial Wire Viewer (SWV) or ETM. Both produce historical records concerning the execution of the program. SmartFusion does not support ETM.

**a) Serial Wire Viewer (SWV):** SWV is a term used to describe some debugging features of CoreSight. Specifically, this refers to the real-time display of PC Samples, Exceptions (including interrupts), data reads and writes, ITM, CPU counters and a timestamp. These features will be described presently. Most of the SWV information comes out the SWO pin. The “Read and Write to Memory Locations” described below can also be thought of as part of SWV. However, this data comes out of the processor via the SWD or JTAG DAP port rather than the SWO pin. The Event Counters also come out the DAP.

**b) Read and Write to Memory Locations:**  $\mu$ Vision has the ability to read and write memory locations on-the-fly and without stealing CPU cycles by using the Debug Access Port (DAP). The Cortex-M3 is a Harvard architecture: this means it has separate code and data address busses. While the CPU is fetching instructions at full speed from the code space,  $\mu$ Vision can access the data space via JTAG or SWD. This feature is active even while the CPU is halted.

This feature is used to display data in the Watch and Memory windows as well as the RTX awareness window. Not only can values of memory locations be displayed and updated in real-time: you can also insert values while the program is running. The only instance real-time will be violated is in the unlikely event the CPU accesses a memory location the exact same time as  $\mu$ Vision does.

## 1) Creating a Global Variable:

We will add a Global Variable to use in our demonstrations:

1. Open the source file Blinky.c by clicking on its tab.
2. Create a global variable `count` near line 18 as in Figure 9.  
`int count = 0;`
3. Add the line `count++;` near line 48.
4. If in Debug mode, exit it. 
5. Compile the project.  You must get no errors or warnings. See the bottom left corner for the Build Output window.
6. Program the Flash.  It will Verify OK.
7. Enter Debug mode. 
8. Click on RUN. 
9. Leave this running for the Watchpoint example on the next page.

```
16 const unsigned long led_mask[] = { 1 <<
17 volatile unsigned long SysTickCnt;
18 int count=0;
19 void SysTick_Handler (void) {
20     SysTickCnt++;
21 }
22
23 void Delay (unsigned long tick) {
24     unsigned long systickcnt;
25
26     systickcnt = SysTickCnt;
27     while ((SysTickCnt - systickcnt) < t;
28 }
29
30
31 int main (void) {
32     int num = -1;
33     int dir = 1;
34
35     WATCHDOG->WDOGENABLE = 0x4C6E55FA;
36
37     SysTick_Config(SystemCoreClock/1000);
38
39     GPIO->GPIO_6_CFG = 5;
40
41     for (;;) {
42         /* Calculate 'num': 0,1,...,LED_NUM
43         num += dir;
44         if (num == LED_NUM) { dir = -1; nur
45         else if (num < 0) { dir = 1; nur
46         GPIO->GPIO_OUT &= ~led_mask[num];
47         Delay(500);
48         count++;
49         GPIO->GPIO_OUT |= led_mask[num];
50         Delay(500);
51     }
```

Figure 9: Variable count .

**TIP:** The changes you made can be saved in the File menu by selecting Save All. It will also be saved if you close the source file, change to a different project or shut  $\mu$ Vision down.

**TIP:** You can edit a source file while in Debug or Edit mode. You must be in Edit mode to compile it.

**TIP:** The Serial Wire Viewer is unable to display local variables and DMA transfers. It can see static, volatile and global variables, structures and peripheral read and write operations. See the Appendix for details.

**TIP:** To see a local variable simply make it static: `static int variable_name;`

## 2) Watch Windows: Updated in Real-time !

Blinky should still be running from the adding of the global variable count on the previous page.

1. Open a Watch window if not already open: select View/Watch Windows/Watch1.
2. In the Watch window, double-click or press F2 and enter `count`. Press Enter.
3. You can do this while the program is running. The CPU is not slowed down or stopped.
4. You can also Drag 'n Drop a variable into various windows.
5. The value of count will be displayed and updated in real-time without stealing any CPU cycles.
6. Double-click on the value of count and enter 0 and press Enter. See Figure 10.
7. 0 will be inserted into the memory where count is stored and will continue incrementing.

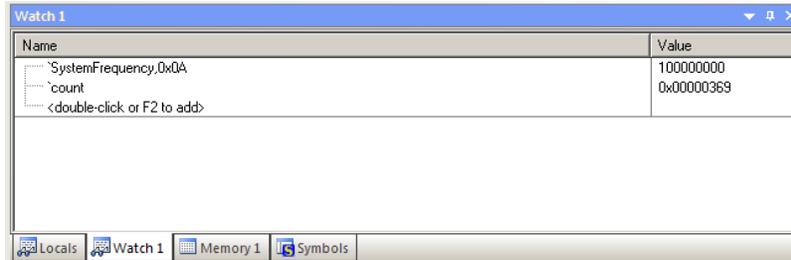


Figure 10: Watch Window

**TIP:** To Drag 'n Drop into a tab that is not active such as the Logic Analyzer, Watch or Memory windows, pick up the variable by blocking it, click and hold, and move it over the tab you want to open; when the tab opens, move your mouse into the window and release the variable.

**TIP:** Periodic Windows Update (in the View menu) must be activated else the windows are updated only when you stop.

## 3) Memory Windows: also are updated in Real-time !

1. Click on the Memory tab to open it.
2. Enter count in the space provided. The program can be running. See Figure 11.
3. Note that as count increments, the address it points to changes. This is useful for work with pointers.
4. Add an ampersand (&) in front of count. i.e. `&count`
5. Note the memory address changes to 0x2000 0010 in this case.
6. This is the physical address where the variable `count` is stored.
7. Note you can right click to vary the format the data is displayed. You can also insert a new value.

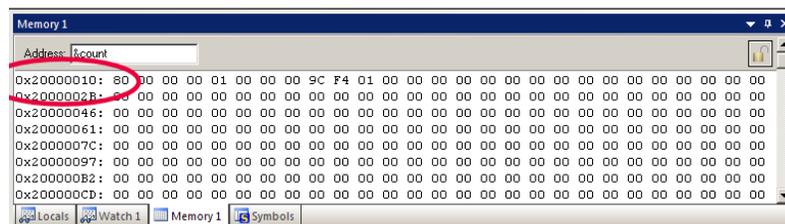


Figure 11: Memory Window

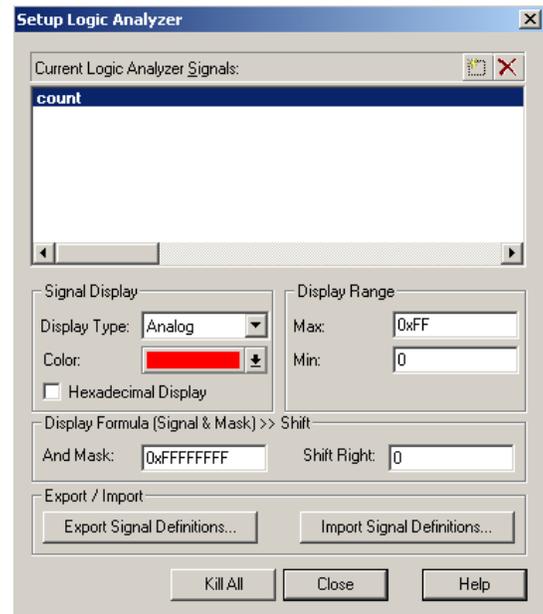
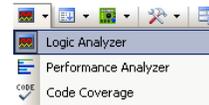
**How It Works: Read and Write to Memory Locations:**  $\mu$ Vision has the ability to read and write memory locations on-the-fly and without stealing CPU cycles by using the Debug Access Port (DAP). The Cortex-M3 is a Harvard architecture: this means it has separate code and data address busses. While the CPU is fetching instructions at full speed from the code space,  $\mu$ Vision can access the data space via JTAG or SWD. This feature is active even while the CPU is halted. SWV is not used. It can be disabled.

This feature is used to display data in the Watch and Memory windows, Event Counters as well as the RTX Awareness window. Not only can values of memory locations be displayed and updated in real-time: you can also insert values while the program is running. The only instance real-time will be violated is in the unlikely event the CPU accesses a memory location the exact same time as  $\mu$ Vision does.

#### 4) Logic Analyzer Window: Updated in Real-time !

The Logic Analyzer (LA) displays up to four variables in a graphical format. When a variable is placed in the LA, it is also displayed in the Trace Records window.

1.  $\mu$ Vision must be in Debug mode. The program can be running.
2. Open View/Analyzer Windows/Logic Analyzer or by the drop down menu if the LA is not already displayed.
3. Select the LA tab and select Setup in the LA window. Figure 12 will open up.
4. Click on the Insert icon  in Figure 12, or press the PC keyboard INSERT key.
5. Enter our global variable `count`. You can also drag and drop the variable into the LA window.
6. Select the Display Range max = 0xFF and Min = 0. Click on Close.
7. Click on RUN if not already running. LA window will be updating as in Figure 13.
8. If necessary, click on Zoom: Out for an appropriate X axis. Try 2 or 5 seconds or for the A2F200, much higher.
9. Add `count` to the Watch window if it is not already there.
10. In the Watch window: change the value of count to various values to change LA values.



#### Figure 13 Explanation:

A: count is incrementing. Note Range = 100K sec.

B: count is manually set to zero in Watch window.

C: count is manually set to 0xF0 in the Watch window.

D: count is manually set to zero in the Watch window.

Figure 12: Logic Analyzer Setup window.

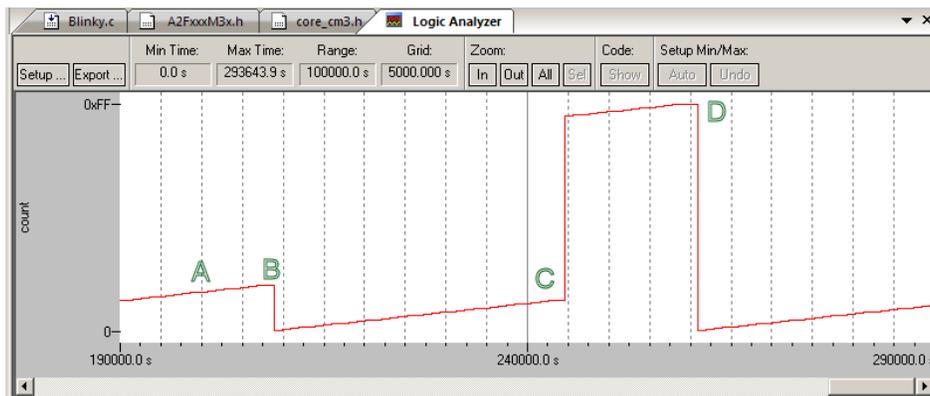


Figure 13: Logic Analyzer windows displaying global variable count.

**Crucial TIP:** If the Logic Analyzer data does not increment but the variable does in the Watch or Memory windows, this can mean the SmartFusion SWO is overloaded. Sometimes the LA will update only by clicking on the scroll bar or stopping the program. Disable some of the SWV functions such as Exceptions (EXCTRC). Disabling Timestamps often stops the LA. Try reducing the Range: to 10 seconds or so as a high range can stop the LA update process.

**TIP:** You can enter up to four variables in the LA window. Try entering three more instances of the variable count.

**TIP:** The A2F500 works better with SWV than does the A2F200 and its slower SWO pin.

### Logic Analyzer Notes:

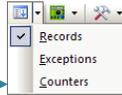
1. The Logic Analyzer displays up to four variables in a graphical format.
2. SWV is used to obtain the data displayed. SWV must be configured and it comes out the Serial Wire Output pin.
3. You will not be able to enter variables if the SWV Trace configuration is not enabled and configured properly. The Core Clock needs to be accurately set.
4. Each variable will have its read and write operations displayed in the Trace Records window.
5. You must enter a variable name. Raw memory addresses are allowed. See the TIP: below.
6. You might have to fully qualify a variable. An example is \Blinky\AD\_dbg.
7. Number 1 reason you can't enter a variable is Trace not properly set. Test SWV using PC Samples.
8. You can drag and drop variables into the LA. Variables must be global, static or a structure.
9. You may need to modify the range to greater than 5 seconds to clearly display most variables.
10. The LA is updated at a fixed rate of approximately 500 msec.

**TIP:** Raw addresses can also be entered into the Logic Analyzer.

An example is: \*((unsigned long \*)0x20000000)

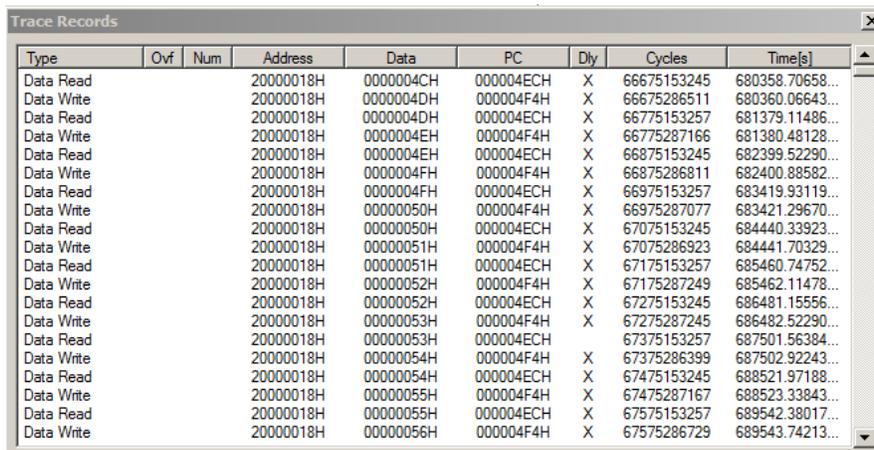
### 5) Data Reads and Writes in the Trace Records for variable count:

When a variable is entered into the Logic Analyzer, the associated data reads and/or writes are displayed in the Trace Records window. This is the method to enter reads and writes in the Trace Records window.

1. Select Debug/Debug Settings and the Trace tab. Unselect EXCTRC and PC Samples.
2. Click on OK twice to return to the main menu.
3. Click on RUN.
4. Open the Trace Records window with View/Trace/Records or the pull-down menu: 
5. Figure 14 opens up. Data Reads and Writes to variable count (at 0x2000\_0010 in this case) are displayed.
6. Note the data values are incrementing. Double-click in the Trace Records window to clear it.
7. Open Debug/Debug Settings and select the Trace tab.
8. Select on Data R/W sample and click on OK twice. This will add the PC column.
9. Click on RUN. 

#### Figure 14 Explanation:

- 1) Data Read to count address 0x2000\_0018 the data value 4C by the instruction at 0x0000\_04EC.
  - 2) Data Write to count address 0x2000\_0018 the data value 4D by the instruction at 0x0000\_04F4.
- The time these events occurred is timed by both CPU cycles and in seconds (Time(s)).  
The X in the DLY column indicates the timestamp is delayed from its true value because of overflow.



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Read			20000018H	0000004CH	000004ECH	X	66675153245	680358.70658...
Data Write			20000018H	0000004DH	000004F4H	X	66675286511	680360.06643...
Data Read			20000018H	0000004DH	000004ECH	X	66775153257	681379.11486...
Data Write			20000018H	0000004EH	000004F4H	X	66775287166	681380.48128...
Data Read			20000018H	0000004EH	000004ECH	X	66875153245	682399.52290...
Data Write			20000018H	0000004FH	000004F4H	X	66875286811	682400.88582...
Data Read			20000018H	0000004FH	000004ECH	X	66975153257	683419.93119...
Data Write			20000018H	00000050H	000004F4H	X	66975287077	683421.29670...
Data Read			20000018H	00000050H	000004ECH	X	67075153245	684440.33923...
Data Write			20000018H	00000051H	000004F4H	X	67075286923	684441.70329...
Data Read			20000018H	00000051H	000004ECH	X	67175153257	685460.74752...
Data Write			20000018H	00000052H	000004F4H	X	67175287249	685462.11478...
Data Read			20000018H	00000052H	000004ECH	X	67275153245	686481.15556...
Data Write			20000018H	00000053H	000004F4H	X	67275287245	686482.52290...
Data Read			20000018H	00000053H	000004ECH	X	67375153257	687501.56384...
Data Write			20000018H	00000054H	000004F4H	X	67375286399	687502.92243...
Data Read			20000018H	00000054H	000004ECH	X	67475153245	688521.97188...
Data Write			20000018H	00000055H	000004F4H	X	67475287167	688523.33843...
Data Read			20000018H	00000055H	000004ECH	X	67575153257	689542.38017...
Data Write			20000018H	00000056H	000004F4H	X	67575286729	689543.74213...

Figure 14: Trace Records displaying read and write operations to variable count.

## 6) Watchpoints: (also known as Access Points)

This is an excellent opportunity to show how the Watchpoints work in Keil  $\mu$ Vision.

Watchpoints are also known as Access Breaks and are indicated by an (A in Figure 16. They are set in the Breakpoints Window. Watchpoints can be thought of as conditional breakpoints when compared to an Execution breakpoint. Execution breakpoints stop the CPU when a specified instruction is fetched (but not executed). Watchpoints stop the CPU when a specified data access occurs and any specified expression becomes true. There might be some Program Counter skid. This is useful for locating read and write operations you do not expect.

SmartFusion has 4 Watchpoints. Watchpoints must be configured in Debug mode and with the CPU halted.

**TIP:** You can set only one data comparison Watchpoint as described below in a Cortex-M3 processor.

### How to set a Watchpoint:

1. Stop the program and remain in Debug mode.
2. Open the Breakpoint window through View/Breakpoints or Ctrl-B. Figure 15 opens up.
3. Watchpoints are created in the bottom half of Figure 15 and are defined and listed in the top half.
4. Enter the count in the bottom half of the Breakpoints window: Figure 15. Click Read and Write.
5. Figure 15 says the program will be halted if the value of 0x22 is written to or read *once* from the variable count. (Count = 1) The other entries are the default settings.
6. Click on Define and the Watchpoint moves to the upper half as in Figure 16. Click on Close.
7. Set count to less than 0x22 in the Watch window if necessary or the Watchpoint will never activate.
8. The Trace Records window must be open. Double-click on it to clear it.
9. Click on RUN and wait for the program to stop. 
10. Scroll to the bottom of the Trace Records window and the data write is displayed as in Figure 17.
11. Delete (kill) this Watchpoint else it will interfere with your other examples.

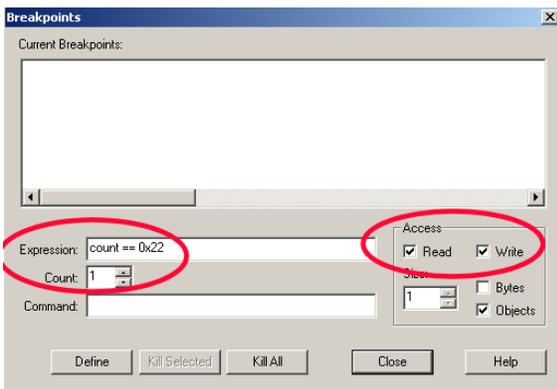
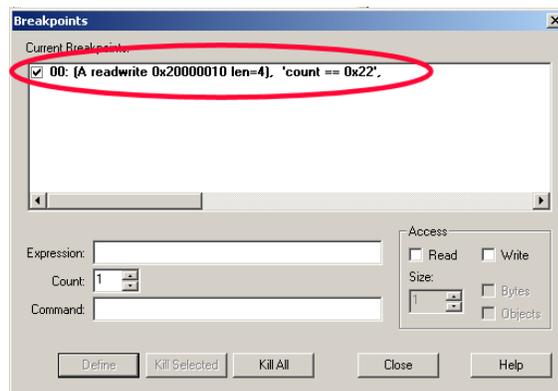


Figure 15: Setting up count



12. Remove all variables from the Logic Analyzer.

Figure 16: count Defined

**TIP:** In the definition of a Watchpoint, legal operatives are &, &&, <, >, <=, >=, == and !=.

Type	Opf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Read			20000010H	00000018H	600003E8H	×	12675006160	129336.79755...
Data Write			20000010H	00000019H	600003EEH	×	12675139137	129338.15445...
Data Read			20000010H	00000019H	600003E8H	×	12775005159	130357.19550...
Data Write			20000010H	0000001AH	600003EEH	×	12775138209	130358.55315...
Data Read			20000010H	0000001AH	600003E8H	×	12875004160	131377.59346...
Data Write			20000010H	0000001BH	600003EEH	×	12875137545	131378.95454...
Data Read			20000010H	0000001BH	600003E8H	×	12975003159	132397.99141...
Data Write			20000010H	0000001CH	600003EEH	×	12975137073	132399.35788...
Data Read			20000010H	0000001CH	600003E8H	×	13075002160	133418.38938...
Data Write			20000010H	0000001DH	600003EEH	×	13075135261	133419.74756...
Data Read			20000010H	0000001DH	600003E8H	×	13175001159	134438.78733...
Data Write			20000010H	0000001EH	600003EEH	×	13175134659	134440.14958...
Data Read			20000010H	0000001EH	600003E8H	×	13275000160	135459.18530...
Data Write			20000010H	0000001FH	600003EEH	×	13275133226	135460.54312...
Data Read			20000010H	0000001FH	600003E8H	×	13374999159	136479.58325...
Data Write			20000010H	00000020H	600003EEH	×	13375132790	136480.94683...
Data Read			20000010H	00000020H	600003E8H	×	13474998160	137499.98122...
Data Write			20000010H	00000021H	600003EEH	×	13475131841	137501.34531...
Data Read			20000010H	00000021H	600003E8H	×	13574997159	138520.37917...
Data Write			20000010H	00000022H	600003EEH	×	13575131030	138521.74520...

Figure 17: Trace Records with data write of 0x22 to variable count.

## 7) Breakpoints: (Execution Breakpoints)

These breakpoints can be set on the fly while the program is running: just double-click in the left margin of a source window. When the PC points to this instruction, the CPU will be stopped before that instruction is executed.

These are called Execution Breakpoints indicated by the (E) in Figure 19. They are set on instructions in the code space. Never on Data. See Watchpoints for operations on data accesses.

### Setting Breakpoints:

There are two ways to set a breakpoint (BP):

1. Double-click on the left margin in a source or disassembly file: even while the program is running. Figure 18 shows the set breakpoint by the red box. This must be set on a gray box which indicates there is an assembly instruction present. Line 45 has no assembly instructions while 46 does.
2. The Breakpoints window. See Figure 19. Open it with Debug/Breakpoints... or Ctrl-B. It is accessible only in Debug mode and when the program is stopped. Watchpoints (Access Breaks) are also entered in the Breakpoints window.

#### To create an entry:

1. Enter the appropriate data in the window.
2. Click on Define.
3. Click on Close.

#### To modify an entry:

1. Double-click on the entry. It will be brought down into the bottom area.
2. Make your modifications.
3. Click on Define to enter it.
4. You will have to delete the original or disable it by un-checking the box.

#### Notes:

1. Hardware BPs are not eligible for all options shown in Figure 19 although Count is available.
2. You can disable a BP by un-checking the box in Figure 19 in the Current Breakpoints area.
3.  $\mu$ Vision will allow only the available BPs to be set.

**TIP:** This is the same window used to create and modify Watchpoints.

**TIP:** If an expression points to code space: an execution or hardware breakpoint is created.

**TIP:** If an expression points to a data address, a Watchpoint or Access Break is created.

**TIP:** If you attempt to enter more Breakpoints or Watchpoints than the device has:  $\mu$ Vision will warn you, usually when you attempt to execute your program..

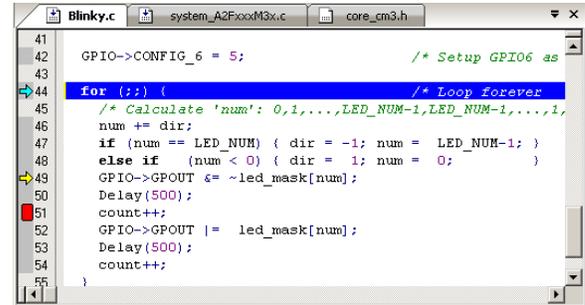


Figure 18: Breakpoint set on Line 51

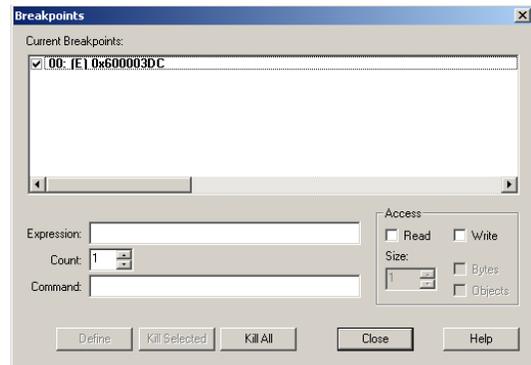


Figure 19: Breakpoints window

## 8) Exception Tracing:

Serial Wire Viewer displays exceptions and interrupts in real-time without stealing any CPU cycles or needing instrumentation code. Note that ARM interrupts are a subset of exceptions.

The Keil examples often have the SysTick timer running. We will use this exception as an example.

1. Assume Blinky.c is loaded and  $\mu$ Vision is in debug mode and the CPU is stopped.
2. Open the Trace Configuration window by clicking on Debug/Debug Settings.
3. Enable EXCTRC: Unselect Periodic and On R/W Sampling. Click on OK twice to return.
4. Click on RUN to start the program.
5. Open the Exception Trace window and ensure the Trace Record window is still open.
6. The Exception Trace window will display the SysTick timer in Figure 18. Double-click to reset.
7. Scroll up and down and you can see the listing of all potential exceptions available in the NVIC.



Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	0	0 s						
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	4928	4.555 s	1.139 s	1.139 s	1.967 s	3294 s	147168.9130...	155367.8952...
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						

Figure 20: SysTick Timer.

1. Right click on the Trace Records window and filter out Data Read and Write frames. Exceptions will be left as in Figure 21.
2. Note the X in the Ovf column. Some frames were lost due to overloading the SWO pin.
3. Remove TimeStamp in the Trace Configuration window. Return and run the program and overflows will be gone. But so will the timestamps.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Exception Entry		15				X	1430400087	14595.91925510
Exception Return	X	0				X	1430400087	14595.91925510
Exception Return	X	0				X	1430400087	14595.91925510
Exception Return	X	0				X	1430491376	14596.85077551
Exception Return	X	0				X	1430572522	14597.67879592
Exception Entry		15				X	1430684095	14598.81729592
Exception Return	X	0				X	1430684095	14598.81729592
Exception Entry		15				X	1430795669	14599.95580612
Exception Return	X	0				X	1430795669	14599.95580612
Exception Return	X	0				X	1430876814	14600.78381633
Exception Entry		15				X	1430988394	14601.92238776
Exception Return	X	0				X	1430988394	14601.92238776
Exception Entry		15				X	1431099972	14603.06093878
Exception Return	X	0				X	1431099972	14603.06093878
Exception Return	X	0				X	1431181118	14603.88895918
Exception Entry		15				X	1431292692	14605.02746939
Exception Return	X	0				X	1431292692	14605.02746939
Exception Return	X	0				X	1431373834	14605.85544898
Exception Entry		15				X	1431485409	14606.99396939
Exception Return	X	0				X	1431485409	14606.99396939

Figure 21: Exceptions in the Trace Records window.

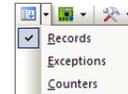
### Explanation of Exception frames:

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned including any Cortex-M3 tail-chaining.

## 9) PC Samples Tracing:

µVision can display a sampling of the program counter values in the Trace Records window. PC Samples is configured in the Trace Records window. This window is opened with View/Trace/Records or the drop down menu shown here:

1. Open the Trace Configuration window by clicking on Debug/Debug Settings.
2. Enable PC Samples by checking Periodic in the PC Sampling area.
3. Disable EXCTRC: and the TimeStamps.
4. Click on OK twice to return.
5. Click on RUN to start the program.
6. Open the Trace Records window. PC Samples will be displayed as shown in Figure 22.



The values displayed in the PC column gives you an indication where the program is spending its time.

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample	X				60000336H		3034007362	31936.91960000
PC Sample	X				6000033EH		3034023746	31937.09206316
PC Sample	X				6000033EH		3034040130	31937.26452632
PC Sample	X				6000033EH		3034056514	31937.43698947
PC Sample	X				6000033EH		3034072898	31937.60945263
PC Sample	X				6000033EH		3034089282	31937.78191579
PC Sample	X				60000338H		3034105666	31937.95437895
PC Sample	X				60000338H		3034122050	31938.12684211
PC Sample	X				60000336H		3034138434	31938.29930526
PC Sample	X				6000033EH		3034154818	31938.47176842
PC Sample	X				6000033EH		3034171202	31938.64423158
PC Sample	X				6000033EH		3034187586	31938.81669474
PC Sample	X				6000033EH		3034203970	31938.98915789
PC Sample	X				60000338H		3034220354	31939.16162105
PC Sample	X				60000338H		3034236738	31939.33408421
PC Sample	X				60000336H		3034253122	31939.50654737
PC Sample	X				6000033EH		3034269506	31939.67901053
PC Sample	X				6000033EH		3034285890	31939.85147368
PC Sample	X				6000033EH		3034302274	31940.02393684
PC Sample	X				6000033EH		3034318658	31940.19640000

Figure 22: PC Samples from the ULINK2 or ULINK-ME

**TIP:** PC Samples are very liable to overload the SWO pin because of the relatively low 98 KHz speed of the current A2F200 silicon. The A2F500 displays PC Samples much easier. All other functions, including the Logic Analyzer, must be turned off in order for the PC Samples to update with the A2F200. Otherwise, the program must be stopped to display the PC Samples. Note the X in the Ovf column indicates many PC Sample frames were lost. This is still useful information.

The A2F500 SmartFusion can display PC Samples without any overloading conditions. It is still a good idea to limit those SWV features selected to only those you actually need to minimize overloading and subsequently lost frames.

**ULINK<sub>pro</sub> Instruction Trace Window:** The ULINK<sub>pro</sub> provides the Instruction Trace window shown in Figure 23. Figure 21 shows two Exceptions, a Data Write and seven PC Samples. Note disassembled instructions are shown and if available, source code will also be displayed. If you double-click on a PC Sample you will be taken to that instruction in the disassembly and/or source window. The ULINK<sub>pro</sub> can process SWV data very fast and therefore is much less susceptible to data overruns.

#	Type	Flag	Num	PC	Opcode	Instruction	Source Code	Address	Data	Cycles	Time[s]
30474	Exception Return		0							1756069016	24.38984744
30475	Exception Entry		34							1756069159	24.38984943
30476	Exception Exit		34							1756069219	24.38985026
30477	Exception Return		0							1756069228	24.38985039
30478	Data Write	D						0x20000000	0x04B1	1756069292	24.38985128
30479	PC Sample			0x080003CE	481C	LDR r0,[pc,#112];@0x080004...				1756074728	24.38992678
30480	PC Sample			0x080003D0	8800	LDRH r0,[r0,#0x00]				1756091112	24.39015433
30481	PC Sample			0x080003D4	D004	BEQ 0x080003E0				1756107496	24.39038189
30482	PC Sample			0x080003E2	4819	LDR r0,[pc,#100];@0x080004...				1756123880	24.39060944
30483	PC Sample			0x080003E4	7800	LDRB r0,[r0,#0x00]				1756140264	24.39083700
30484	PC Sample			0x080003E6	B130	CBZ r0,0x080003F6				1756156648	24.39106456
30485	PC Sample			0x080003F6	E7EA	B 0x080003CE				1756173032	24.39129211

Figure 23: Instruction Trace window from ULINK<sub>pro</sub>.

## 10) Debug (printf) Viewer:

ITM Stimulus Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into the  $\mu$ Vision Debug (printf) Viewer for display. You write ASCII values to Port 0.

1. Stop the program and exit debug mode if necessary.
2. Add this code to Blinky.c. A good place is right after the place where you declared count variable: (the following code is a pointer to ITM Port 0)
 

```
#define ITM_Port8(n) (*(volatile unsigned char*)(0xE0000000+4*n))
```
3. Near line 51 enter these three lines: Just after the line `count++;` that you entered is a good place.
 

```
ITM_Port8(0) = 0x35;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
```
4. Rebuild the source files, program the Flash memory and enter Debug mode.
5. Open Debug/Debug Settings and select the Trace tab. Confirm ITM Port 0 is selected, unselect Periodic, EXCTRC, on Data R/W Sample and TimeStamps Enable as shown in Figure 24.

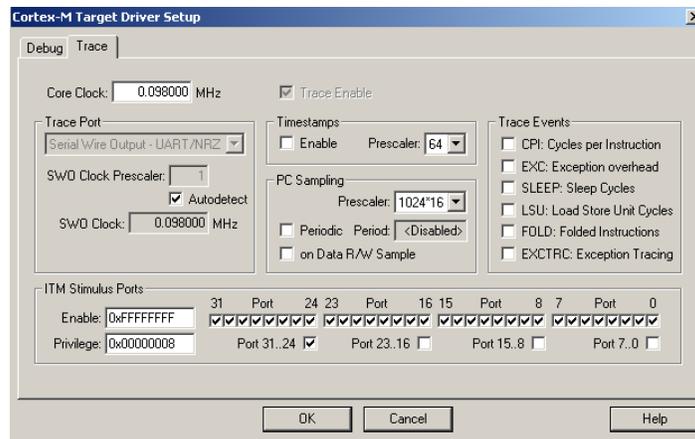


Figure 24: Trace Configuration for printf

6. Click on View/Serial Windows and select Debug (printf) Viewer.
7. Make sure Periodic Update is enabled or the window will only update when you stop the program.
8. Click on RUN.
9. In the Debug (printf) Viewer you will see the value “5” appear every few seconds as shown in Figure 25. You could add more ASCII characters to format your own messages.

### Trace Records

These writes to the ITM will be displayed in the Trace Records window.

### ITM Conclusion

The writes to ITM Stimulus Port 0 are intrusive and are usually one cycle. It takes zero additional CPU cycles to get the data displayed in the Debug (printf) Viewer.



Figure 25: ITM writes of "5" with LF and CR

**TIP:** ITM\_SendChar is a Keil supplied function you can use to send characters to the Debug Viewer. It is found in the header `core_CM3.h` in the directory `C:\Keil\ARM\RV31\INC`.

## 11) Opening and Using RTX Kernel Awareness: The RTX Viewer

Keil has its own full feature RTOS called RTX. It is provided as part of Keil MDK full tool suite for no charge and can have up to 255 tasks. No royalty payments are required.

If the RTX source code is required, this is included in the Keil RL-ARM™ Real-Time Library which also includes USB, CAN, TCP/IP networking and a Flash File system.  $\mu$ Vision will work with any RTOS.

Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a memory area by the RTOS. Keil provides Task Awareness windows for RTX.

$\mu$ Vision's RTX Viewer is a kernel awareness feature with live update. This uses both the Serial Wire Viewer ITM Stimulus Port and the DAP (Debug Access Port).

RTX Viewer windows are available while in Debug mode by selecting Debug/OS Support. See Figure 23.

An example using RTX is available for SmartFusion.

RTX is easy to configure with SmartFusion. This integration is made easy by the Cortex-M3 architecture.

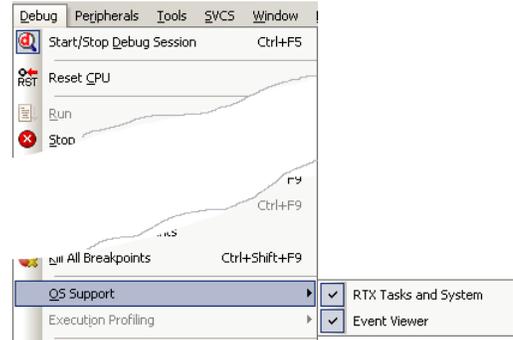


Figure 26: Opening the RTX Viewer

### Configuring $\mu$ Vision to activate RTX Kernel Awareness:

The RTX Viewer has two components: the RTX Tasks and System window which uses the DAP for access and the Event Viewer which uses SWV (Serial Wire Viewer) through the ITM Stimulus Port # 31. Table 1 outlines the mandatory configuration items.

Table 1:  $\mu$ Vision configuration items needed for RTX Viewer

Window\setting	Enable RTX	Periodic Update	SWD	SWV	ITM
<b>RTX Tasks &amp; System</b>	Yes	Yes	Yes	No	No
<b>Event Viewer</b>	Yes	No	Yes	Yes	# 31

#### Notes for Table 1:

- Enable RTX:** Click on the Options for Target icon  and select the Target tab. In the Operating System dialog box, select RTX Kernel – Viewer.
- Periodic Update:** while in Debug mode, select View/Update Periodic Window. On by default.
- SWD:** Serial Wire Debug: Options for Target select Debug tab. SWD is needed for SWV
- SWV:** Serial Wire Viewer. Options for Target select Debug tab and then Settings. The following items are mandatory: everything else is optional.
  - Trace Enable must be checked and a valid entry for Core Clock: 0.098 MHz for SmartFusion A2F200 silicon (date code 0952) and 50 MHz for A2F500 devices when the CPU is running at 100 MHz.
  - ITM Stimulus # 31.

**NOTE:** RTX RTOS is not configured for SmartFusion in MDK 4.20. It is compatible with MDK 4.13a. RTX will be reconfigured to work with SmartFusion after 4.20. You can access the solution to use RTX with MDK 4.20 at [www.keil.com/support/docs/3551.htm](http://www.keil.com/support/docs/3551.htm).

If you do not use RTX, there are no problems. Contact Keil Technical Support for the status of this issue.

### Open RTX\_Blinky and configure the RTX Viewer:

- Open the RTX\_Blinky project from C:\Keil\ARM\Boards\Actel\SmartFusion\RTX\_Blinky\Blinky.uvproj.
- Ensure the SWV is configured in the Trace Configuration window. Use SW and not JTAG. Enable ITM Port 31.
- Compile  and load the project into the Flash . Enter Debug mode. 
- Click on the RUN icon to run RTX\_Blinky. 
- Four LEDs will blink sequentially.

**TIP:** If RTX\_Blinky does not run and when you stop it the PC is on the Hard Fault vector, see the **NOTE:** above.

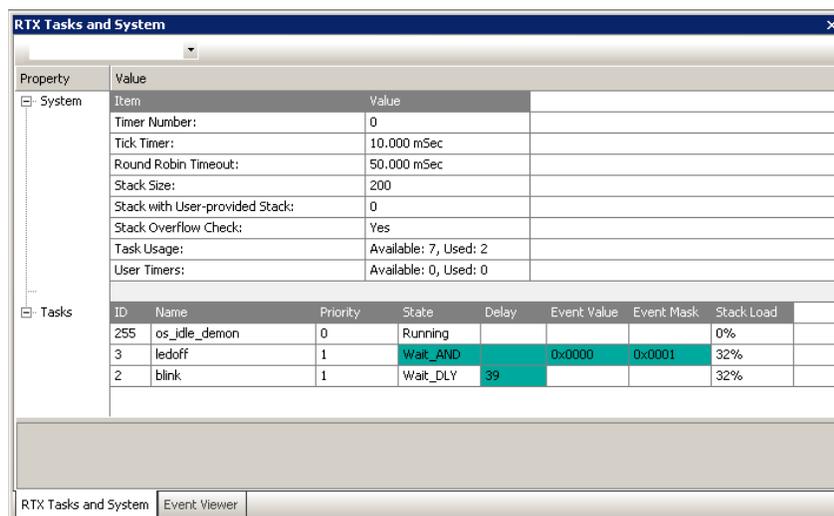
## Open the RTX Viewer Windows:

1. Open Debug/OS Support and select RTX Tasks and System and the window in Figure 27 opens up. RTOS visibility is updated in real-time using the same CoreSight DAP technology as used to update and modify the Watch and Memory windows.
2. Open Debug/OS Support and select Event Viewer and the window in Figure 28 opens up.
3. Note the tabs at the bottom left to switch between the two windows.

### Notes:

1. The Event Viewer is updated with the ITM Stimulus Port # 31: is part of the Serial Wire Viewer.
2. If the RTX Tasks and System window is updated only when you stop the program: check View/Update Periodic Window This setting will not affect the Event Viewer.
3. If the Event Viewer shows no data: click on the ALL icon and adjust the range with + and -. Make sure the Serial Wire Viewer is configured properly.

**TIP:** To view the exception SysTick, enable EXTRC in the Trace Configuration window. Select View/Trace and select the Exception window. SysTick will also be displayed in the Instruction Trace window.



Property	Value						
System							
Item	Value						
Timer Number:	0						
Tick Timer:	10.000 mSec						
Round Robin Timeout:	50.000 mSec						
Stack Size:	200						
Stack with User-provided Stack:	0						
Stack Overflow Check:	Yes						
Task Usage:	Available: 7, Used: 2						
User Timers:	Available: 0, Used: 0						
Tasks							
ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
3	ledoff	1	Wait_AND		0x0000	0x0001	32%
2	blink	1	Wait_DLY	39			32%

Figure 27: RTX Viewer: RTX Tasks and System window.

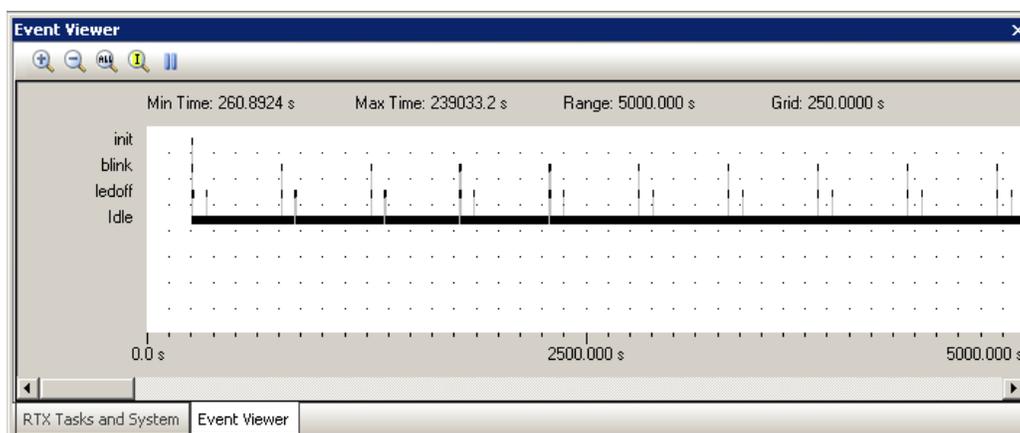


Figure 28: RTX Viewer: Event Viewer window.

**TIP:** The RTX Viewer gets its data from the DAP read/write port. This is not SWV. The Event Viewer gets its data from SWV therefore SWV must be configured. Remember A2F200 Core Clock: = 0.098 MHz and A2F500 = 50 MHz.

## 12) Logic Analyzer Window: View variables real-time in a graphical format:

µVision has a graphical Logic Analyzer window. Up to four variables can be displayed in real-time using the Serial Wire Viewer in the SmartFusion. RTX\_Blinky uses four tasks to create the waveforms. We will graph these four waveforms.

1. Close the RTX Viewer windows. Stop the program and exit debug mode.
2. Add 4 global variables `unsigned int phasea` through `unsigned int phased` to Blinky.c as shown here:
3. Add 2 lines to each of the four tasks Task1 through Task4 in Blinky.c as shown below: `phasea=1`; and `phasea=0`; :the first two lines are shown added at lines 053 and 056 (just after LED\_On and LED\_Off function calls). For each of the four tasks, add the corresponding variable assignment statements `phasea`, `phaseb`, `phasec` and `phased`.
4. We do this because in this simple program there are not enough suitable variables to connect to the Logic Analyzer.

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

```

016 OS_TID t_phaseC;
017 OS_TID t_phaseD;
018 OS_TID t_clock;
019
020 unsigned int phasea;
021 unsigned int phaseb;
022 unsigned int phasec;
023 unsigned int phased;
024
025 #define LED_A 0x01
026 #define LED_B 0x02
    
```

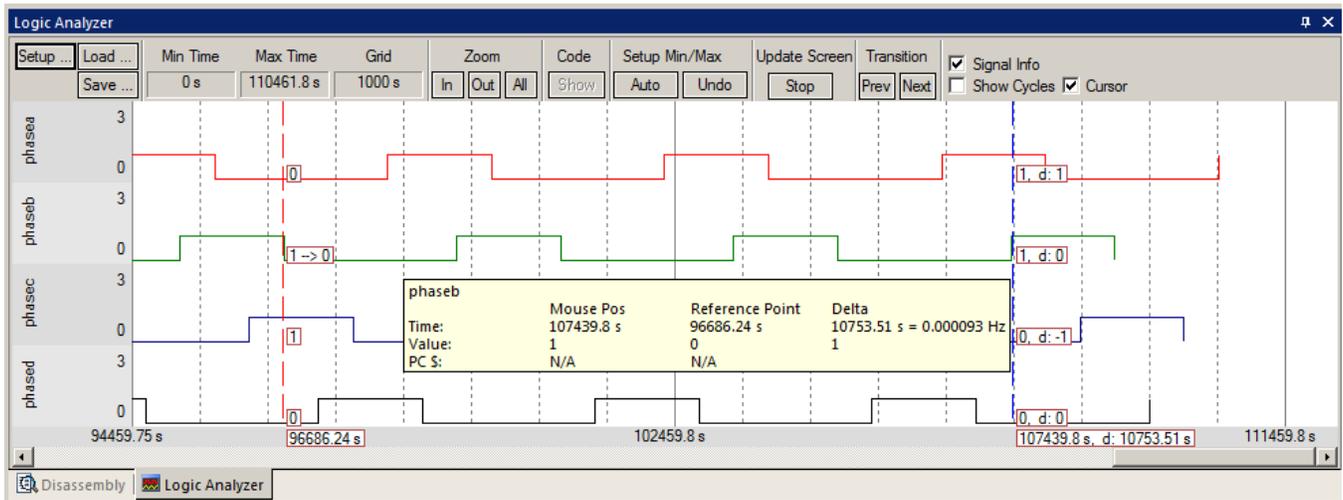
5. Rebuild the project. Program the Flash .
6. Enter debug mode .
7. You can run the program at this point. .
8. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar.

```

046 /*
047      *      Task 1 'phaseA': Pha:
048      *
049  _task void phaseA (void) {
050      for (;;) {
051          os_evt_wait_and (0x0001, 0);
052          LED_On (LED_A);
053          phasea=1;
054          signal_func (t_phaseB);
055          LED_Off(LED_A);
056          phasea=0;
057      }
058  }
    
```

### Enter the Variables into the Logic Analyzer:

9. Click on the Blinky.c tab. Block `phasea`, click, hold and drag up to the Logic Analyzer tab (don't let go yet!)
10. When it opens, bring the mouse down anywhere into the Logic Analyzer window and release.
11. Repeat for `phaseb`, `phasec` and `phased`. These variables will be listed on the left side of the LA window as shown. Now we have to adjust the scaling.
12. Click on the Setup icon and click on each of the four variables and set Max. in the Display Range: to 0x3.
13. Click on Close to go back to the LA window.
14. Using the OUT and In buttons set the range to 1000 seconds. (or 2 sec for the A2F500).
15. You will see the following waveforms appear. Click to mark a place See below. Select Signal Info and Cursor. Hover over one of the waveforms and get timing and other information as shown in the inserted box labeled phaseb:



**TIP:** You can also enter these variables into the Watch and Memory windows to display and change them in real-time.

## 1) Creating a new project: Using the Blinky source files:

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. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run the Blinky example as usual. You can use this process to create any new project from your own source files created with  $\mu$ Vision's editor or any other editor.

### Create a new project called Mytest:

1. With  $\mu$ Vision running and not in debug mode, select Project/New  $\mu$ Vision Project.
2. In the window Create New Project go to the folder C:\Keil\ARM\Boards\Actel\SmartFusion.
3. Right click and create a new folder by selecting New/Folder. I named this new folder FAE.
4. Double-click on the newly created folder "FAE" to enter this folder as is shown below.
5. Name your project. I called mine Mytest. You can choose your own name.
6. Click on Save.
7. "Select Device for Target 1" window opens up.
8. This is the Keil Device Database<sup>®</sup> which lists all the devices Keil supports (plus some secret ones).
9. Locate the Actel directory, open it and select A2F200M3F. Note the device features are displayed
10. Click on OK.
11. A window opens up asking if you want to insert the default SmartFusion startup code to your project. Click on "Yes". This will save you a great deal of time.
12. In the Project Workspace in the upper left hand of  $\mu$ Vision, open up the folders by clicking on the "+" beside each folder.
13. We have now created a project called Mytest and the target hardware called Target 1 with one source file startup\_A2FxxxM3.s.
14. Click once (carefully) on the name "Target 1" (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose Smart as shown in Figure 30. Press Enter to accept this. Note the Target selector also changes to Smart. Click on the + to open up the directory structure if necessary. You can create many target hardware configurations including a simulator and easily select them.

### Select the source files:

1. Using MS Explore (right click on Windows Start icon), copy blinky.c, core\_cm3.c and system\_A2FxxxM3x.c from C:\Keil\ARM\Boards\Actel\SmartFusion\Blinky to the Keil\SmartFusion\FAE folder.
2. In the Project Workspace in the upper left hand of  $\mu$ Vision, right-click on "Smart" and select "Add Group". Name this new group "Source Files" and press Enter.
3. Right-click on "Source Files" and select **Add files to Group "Source Files"**.
4. Select the file Blinky.c, core\_cm3.c and system\_A2FxxxM3.c and click on Add (once) and then Close. These will show up in the Project Workspace when you click on the + beside Source Files..
5. Select Options For Target and select the Debug tab. Make sure ULINK Cortex Debugger is selected. Select this by checking the circle just to the left of the word "Use:".
6. At this point you could build this project. Configure the ULINK for SW and you can program the flash and run it on your SmartFusion board.

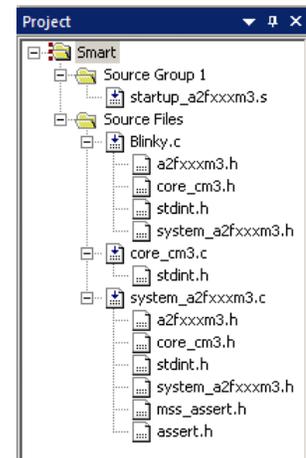


Figure 29: Project Menu

## 2) Running Blinky from RAM:

It is possible to run your program in the SmartFusion Cortex-M3 RAM rather than Flash.

$\mu$ Vision makes it easy to create different target settings. We will create one using RAM in the Blinky project.

There are six simple steps to accomplish this:

### 1) Create a New Target Configuration in $\mu$ Vision:

1. Open the project Blinky.uvproj.
2. Open Project from the main menu and select Manage and then select Components, Environment, Books...
3. In the Project Targets box, click on the Insert icon. 
4. Enter the name of your target (I chose RAM) and press the Enter key. Click on OK.
5. In the main menu, click on the Select target window and select your RAM target: 
6. You can now configure the target RAM which will be different from A2F200M3F Flash.

### 2) Create an initialization file to configure SmartFusion when Debug mode is entered:

7. We will use  $\mu$ Vision to create an initialization file to configure SmartFusion to use RAM.
8. Select File/New and a blank text window will be created. Enter the following text:

```
FUNC void Setup (void) {
    _WDWORD(0x40006010, 0x4C6E55FA); // Disable watchdog
    SP = _RDWORD(0x60080000); // Setup Stack Pointer
    PC = _RDWORD(0x60080004); // Setup Program Counter
    _WDWORD(0xE00ED08, 0x20000000); // Setup Vector Table Offset Register
}
LOAD %L INCREMENTAL // Download
Setup(); // Setup for Running
g, main
```

9. Select File/Save As and enter ram.ini and press the Enter key.

### 3) Enter the memory addresses into $\mu$ Vision:

10. Open the Target Options menu.  The Target tab will be selected as in Figure 31.
11. Read/Only Memory Areas configures the program space. Read/Write Memory Areas allocates data.
12. We will divide up the SmartFusion RAM (64K in this case) between the program and data spaces.
13. RAM in the SmartFusion we are using starts at 0x2000\_0000 and ends at 0x2001 0000. (64 K bytes)
14. Enter ROM Start 0x2000 0000 and size 0x8000 as shown in Figure 31. Enter RAM Start 0x2000\_8000 and size 8000 as shown in Figure 31. Do not click on OK yet.



Figure 30: Configuring Memory Area (partial screen)

15. 32K of RAM is now available for your program code and 32 K of RAM for data. You can use other values. You must make sure there is enough RAM for your program and variables to fit into.

#### 4) Create a directory for the compiled files to be stored:

16. Now we will create a directory for the compiled files. Select the Output tab.
17. Click on Select Folder for Objects... box. Select Create New Folder icon.  New Folder is entered.
18. Rename it RAM and press Enter.
19. Double-click on this new folder to add it in the Path box. Click on OK. Your compiled files will go here.

#### 5) Select ram.ini to be executed when Debug mode is entered:

20. Select the Debug tab. Options for Target should still be open. Use  if you closed it. 
21. In the Initialization File: box, enter ram.ini as shown in Figure 32. You can use the Browse icon.
22. ram.ini will be executed every time you enter Debug mode.
23. Unselect Load Application at Startup: This makes sure your program isn't loaded twice. Run to main() will be grayed out. This function will be done by the last command in the ram.ini file.
24. Click on Utilities and uncheck Update Target before Debugging to turn off automatic Flash programming when entering Debug mode.
25. Click OK. Select File/Save All.

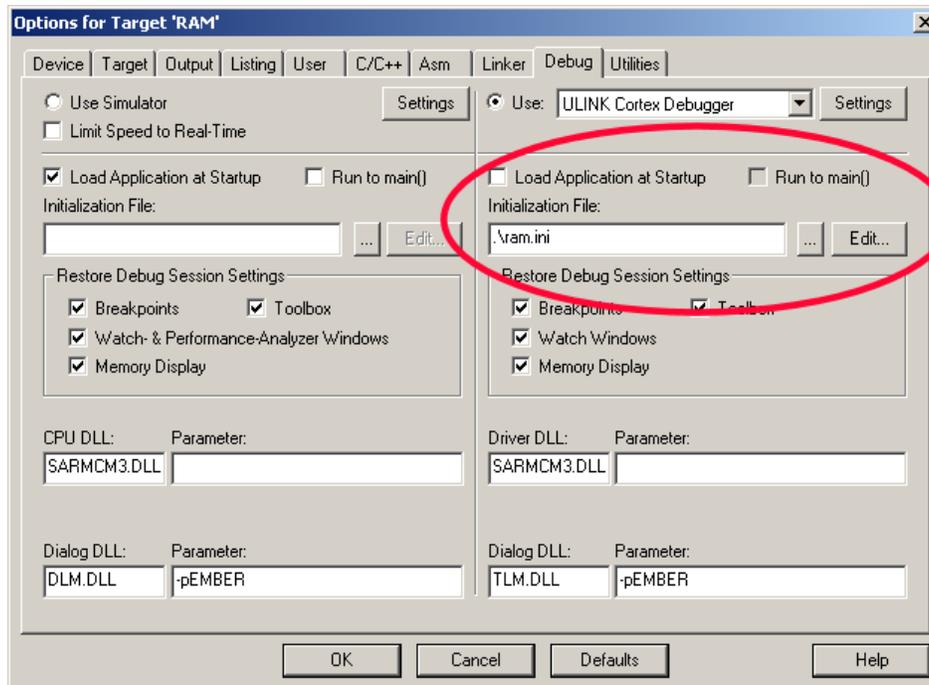


Figure 31: Selecting ram.ini

#### 6) Compile and RUN the program:

26. Rebuild and enter Debug mode. Do not use Load. Load is only for programming Flash.
27. Note the disassembly window will show an address in the area of 0x2000\_0000.
28. This is the Blinky program located in RAM.
29. Breakpoints set will be software breakpoints and can be as many as you want. You cannot set them on-the-fly as with hardware breakpoints. You must stop the program to set/unset them.

### 3) SmartFusion eNVM Programming: (for the Cortex-M3)

µVision provides an integral FLASH programming feature. The source files are in C:\Keil\ARM\FIash\A2FxxxM3

#### Configuring Flash Programming:

- 1) Click on the Target Options icon.  You must be in Edit and not in Debug mode.
- 2) Click on the Utilities tab. The rear window of Figure 33 opens up.
- 3) Click on Settings. The front window of Figure 33 opens up.
- 4) Under description is the Flash programming algorithm. Figure 33 is correct.
- 5) If this field is blank, you will receive an error when you try to program the SmartFusion eNVM.
- 6) If it is blank or you need to make a change, click on Add. Select the appropriate algorithm and select OK.
- 7) Click on OK twice to return to the main menu.

**TIP:** To program the flash when entering Debug mode, select Update Target before Debugging as shown in the rear Figure 33. If not selected, you will need to program it manually with the Load icon.

**TIP:** The setting Verify Code Download in the Target Options window/Settings: must be deselected for SmartFusion eNVM programming. Verify is now performed by the programming algorithm. If Verify Code Download is enabled, an error will result because it will test the Actel bootloader code rather than the user flash.

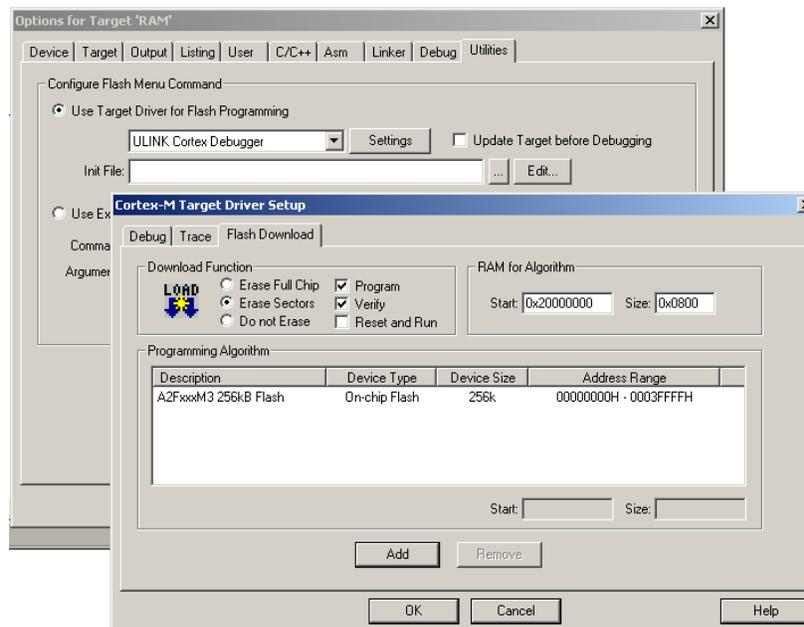


Figure 32: Flash Programming Configuration

**SmartFusion BootLoader:** SmartFusion contains a bootloader in its eNVM (Flash). At RESET this bootloader runs first and then control is passed to the user program typically located at 0x0.

With Blinky loaded and in Debug mode, click on the RESET icon. 

Note the addresses in the disassembly window are in the area of 0x6000\_0000. Probably at 0x6008\_051C which is the start of the bootloader code.

Run and stop the program. Now the addresses are in the user code in the area of 0x0. This is because the Flash has now been remapped by the bootloader.

#### 4) TCP/IP: Keil http demonstration:

Keil has ported its TCP/IP stack to the SmartFusion Ethernet peripheral. This TCP/IP is part of the Keil RL-ARM software suite. Contact Keil sales for more information on RL-ARM.

A web server (http) demonstration is available on: [www.keil.com/download/docs/404.asp](http://www.keil.com/download/docs/404.asp)

Instructions are included as well as a pre-compiled sample executable. (Http\_demo.axf).

Do not attempt to rebuild the project unless you have RL-ARM installed. Http\_demo.axf will be deleted and no new version will be created. You must then reload Http\_demo.axf from the example files again.

You can load this axf file into the evaluation version of  $\mu$ Vision and run and debug the http program.

A http demon will be installed on the SmartFusion and you can connect to it with your favourite browser as shown:



The screenshot shows a web page titled "Keil Embedded WEB Server Example for Actel". The page features the Keil logo (An ARM Company) and the Actel logo (POWER MATTERS). Below the logos, there is a navigation menu with links: [ [Network](#) | [System](#) | [LED](#) | [Button](#) | [Language](#) | [Statistics](#) ]. The page also contains text explaining that the web pages are served by the Web server which is part of TCPnet in the Real-Time Library. It mentions that the example is developed using the RealView Microcontroller Development Kit and the Real-Time Library. For additional information about Keil products, it directs users to visit [www.keil.com](http://www.keil.com). The footer of the page includes the copyright notice: Copyright © 2004-2010 KEIL - An ARM Company. All rights reserved.

## 5) ARM JTAG/SWD/SWO Adapter Connector Schematics:

For the complete description of CoreSight and the new Hi-Density debug connector search [www.arm.com](http://www.arm.com) for this document: DDI0314F\_coresight\_component\_trm.pdf. Many customers are now using this new, smaller connector.

Hi-Density connector part numbers can be found here: [www.samtec.com/ftp/pub/pdf/ftsh\\_mt.pdf](http://www.samtec.com/ftp/pub/pdf/ftsh_mt.pdf).

The 10 pin part number that Keil uses in its boards is Samtec FTSH-105. A 20 pin is available: FTSH-110.

**TIP:** TRSTN resets the JTAG module inside the processor. Most better debug adapters do not need it.

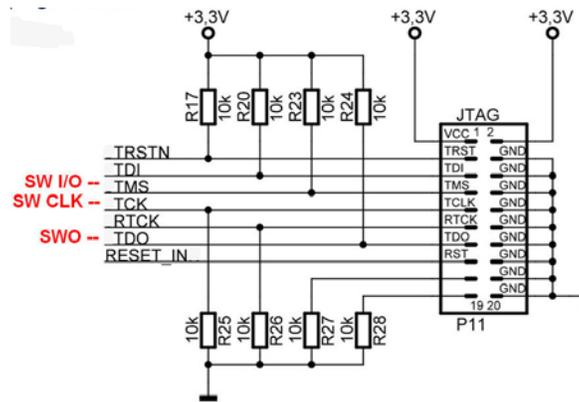


Figure 33: Legacy JTAG Connector showing location of the Serial Wire signals. Top view.

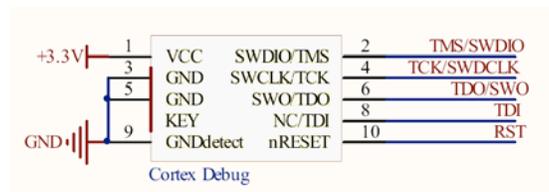


Figure 34: 10 pin Hi-Density JTAG/SWD/SWO connector. Top view.

### The new ULINK-ME:

The current ULINK-ME does not supply 3.3 volts to a target board through JTAG connector pin 1. Older versions of the ME did. You can remove this feature by cutting the pin 1 lead or remove the diode on the ME. This diode is on the bottom right corner of the ME beside the JTAG connector. This will not affect the EVAL-KIT. The DEV-KIT is affected more with its complicated power supplies. The ULINK2 or ULINK*pro* are not affected as neither provides power to a target.

The new ULINK-ME also incorporates a connector and cable as shown in Figure 35 for the new compact 10 pin JTAG/SWD connector. It does not supply the 3.3 volts to pin 1. It is shipping now.

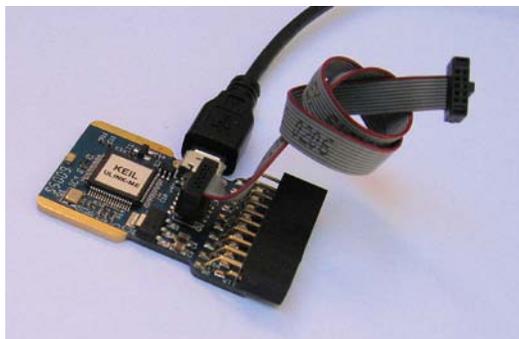


Figure 35: ULINK-ME with the new 10 pin connector and cable.

## 6) Acronym List

<b>BP</b>	Break Point
<b>DAP</b>	Debug Access Port
<b>eNVM</b>	SmartFusion Flash - embedded nonvolatile memory
<b>ETM</b>	Embedded Trace Macrocell
<b>GPIO</b>	General Purpose Input/Output
<b>ITM</b>	Instrumentation Trace Macrocell
<b>SWD</b>	Serial Wire Debug
<b>SWV</b>	Serial Wire Viewer
<b>SWO</b>	Serial Wire Output
<b>TPIU</b>	Trace Port Interface Unit
<b>WP</b>	Watch Points. They are also called Access Breaks.

**Macrocell:** A macrocell is a pre-built “black box” of gates used by the chip designer to add a function or peripheral to a processor. Examples are ETM, ITM and memory. Macrocells can also be thought of as “building blocks”. This is a term used by chip designers and not end users.

**Hardware Design:** This is Microsemi’s terminology for the FPGA program. It is similar to a microcontroller project.

## 7) Overloading the SWO pin:

The Serial Wire Output pin (SWO) is just that – a single bit high speed line that is expected to handle all the data sent to it. A Cortex-M3 can read and write data much faster than the SWO can process.

Overloads have an X in the Ovf or Dly (overflow or Delay) column in the Trace Records window: but not always. Sometimes the frames have corrupt, missing or have wrong information and this is easy to spot.

ITM frames with IDs other than 0 or 31 are a sure sign of either overloading or an incorrect Core Clock setting.

### Here are some hints on reducing traffic to minimize SWO overruns:

1. Reduce the number of variables in the Logic Analyzer.
2. Reduce the number of items selected in the Trace Configuration window: These include:
  - a. PC Sampling. Turn this off or increase the Prescaler to reduce the sampling rate.
  - b. On Data R/W Sample.
  - c. EXCTRC: Exception Tracing
  - d. Any Trace Events. (the Counters)
  - e. ITM Stimulus Ports 31 and 0. (Viewer and RTX Awareness respectively)
3. Set the Timestamp Prescaler to 1.
4. Turn the timestamps off. Sometimes this will turn the trace off though. But it is worth a try.
5. Lower the rate of reads, writes and exception calls in your software.
6. Turn off the SYSTICK timer in your software if you are not using it.
7. Reduce the Range setting in the Logic Analyzer. Sometimes a very large number overloads  $\mu$ Vision.
8. The General Rule is: activate only those SWV features you need.
9. Using a faster computer can help in some cases. SWV can create a great deal of information that must be processed quickly. This is especially true where the Logic Analyzer window is concerned.
10. We will be making continuous improvements to  $\mu$ Vision to provide you with a better SWV experience.

**TIP:** If you are seriously overloading the SWO, there will usually be at least some data coming out.  $\mu$ Vision recovers easily and painlessly from data overruns. You are made aware when frames are lost.

## 8) Top Seven Reasons why you can't get SWV working:

Symptoms can range from an inability to enter a known variable in the Logic Analyzer, corrupted trace frames or no information at all.

1. Core Clock is wrong. This is the number 1 reason. The SmartFusion A2F500 has a higher SWO speed than the current 98 KHz. Normally, the SWO speed is derived from the Core Clock speed. For the next version of SmartFusion devices, you would enter 50 MHz or whatever your clock speed is in Core Clock: and the SWO speed will be calculated from that.  
With A2F200 silicon code date 0952 calculate Core Clock speed as (CPU Speed/1024). 100 MHz/1024 = 98 KHz. With A2F500 devices running at a CPU speed of 100 MHz, set Core Clock to 50 MHz.
2. Periodic Update is not enabled. Your windows update only when the processor is stopped.
3. Trace Enable is not checked.
4. Using JTAG instead of SWD.  $\mu$ Vision will complain about this if you try it. SWV needs SWD (or SW) selected.
5. SWD speed too high. Try a slower speed. This is not usually the problem.
6. You have selected too many SWV items and it is extremely seriously overloaded. This effect often manifests itself when you stop the program and the Trace Records window is updated.
7. Certain portions of  $\mu$ Vision in MDK are in the process of being updated. This is particularly true of the Logic Analyzer and ULINK $pro$  features. With the current limited bandwidth of the SmartFusion SWO pin, a few Serial Wire Viewer features might not work consistently as described. These will be updated as soon as possible.

### For more information:

Contact Keil or your local representative as listed on [www.keil.com/distis](http://www.keil.com/distis)

Obtain the Keil ULINK $pro$  User Manual for information on Serial Wire Viewer.

Visit [www.keil.com](http://www.keil.com) for the latest version of this document.

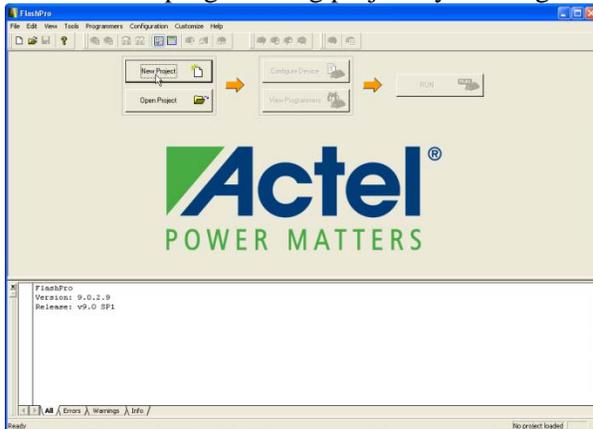
For SmartFusion: [www.actel.com/products/SmartFusion](http://www.actel.com/products/SmartFusion)

For additions, comments or corrections to this document: contact Robert Boys [bob.boys@arm.com](mailto:bob.boys@arm.com)

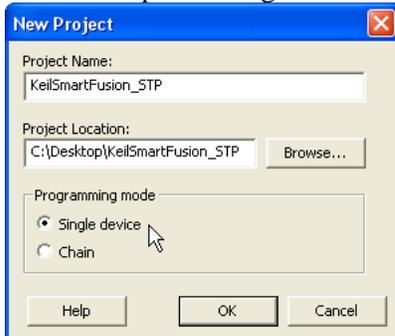


## 10) Programming the A2F-EVAL-KITBoard (F2 device) (FPGA Flash eNVM)

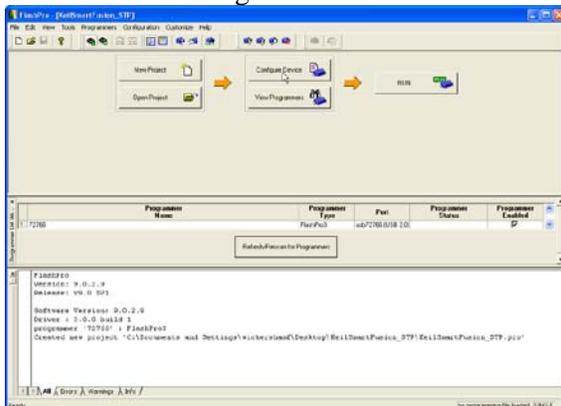
1. This is to program the FPGA eNVM (Flash) to connect SmartFusion to the board LEDs in order for the Keil examples to work correctly. Do not confuse this with programming the SmartFusion Cortex-M3 Flash. They are different.
2. Ensure that jumpers JP7 and JP10 are populated in the 1-2 position and that the 2 USB cables are connected to the board.
3. Open the FlashPro Programming Software. FlashPro is available stand-alone or as part of the Libero IDE FPGA development software. Both are free from the Actel.com website.
4. Create a new programming project by clicking the New Project button



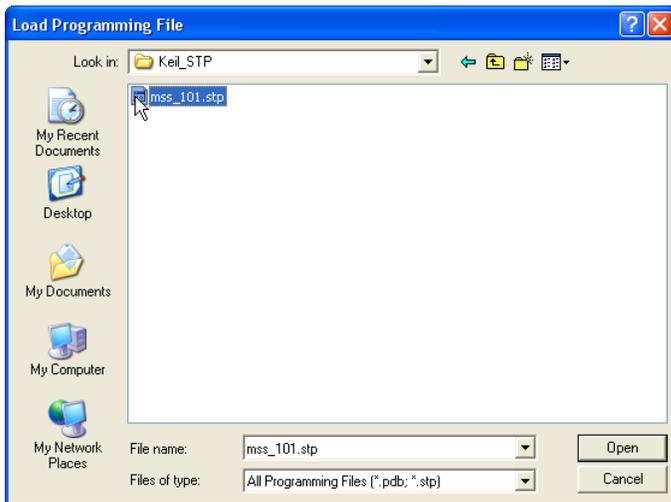
5. Select the option “Single Device” when choosing the programming mode



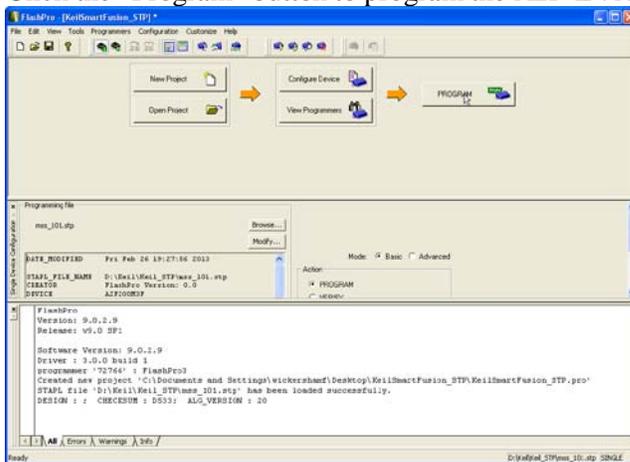
6. Click on the “Configure Device” button. This will open the “Load Programming File” window.



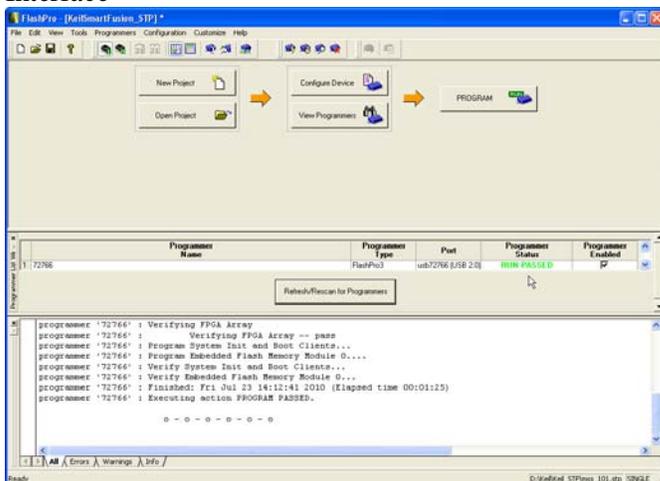
7. Browse the PC file system to find the SmartFusion Eval Kit mss\_101.stp file. This file is attached to this PDF or is available on [www.actel.com](http://www.actel.com) or [www.keil.com](http://www.keil.com).



8. Click the “Program” button to program the A2F-EVAL-KIT SmartFusion device.



9. Allow FlashPro to program the Eval board until the “Run Passed” text can be seen on the FlashPro Interface



## 8) Keil Products:

### Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Professional (Includes RL-ARM) Promotion with ULINK<sub>pro</sub> until September 30, 2011 - \$9,995
- MDK-Standard with included RTX RTOS - \$4,895
- MDK-Basic - 256K Compiler Limit, No debug Limit, RTX included - \$2,695
- MDK-Lite (Evaluation version) RTX included \$0

All versions include RTX source code.

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

Call Keil Sales for special university pricing.

For the ARM University program: go to [www.arm.com](http://www.arm.com) and search for university.

All products include Technical Support for 1 year. This can be renewed.

### Keil Real Time Library (RL-ARM™)

- Included with MDK-Professional.
- TCP/IP stack for SmartFusion. Flash file system. No CAN or USB.

### USB-JTAG adapter (for Flash programming too)

- ULINK2 - \$395 (*ULINK2 and ME - SWV only – no ETM*)
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINK<sub>pro</sub> - \$1,395 – Cortex-Mx SWV & ETM trace

**Note: USA prices.** Contact [sales.intl@keil.com](mailto:sales.intl@keil.com) for pricing in other countries.

Prices are for reference only and are subject to change without notice.

For the entire Keil catalog see [www.keil.com](http://www.keil.com) or contact Keil or your local distributor.



---

### For more information:

**Keil Sales** In USA: [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. Outside the US: [sales.intl@keil.com](mailto:sales.intl@keil.com)

**Keil Technical Support** in USA: [support.us@keil.com](mailto:support.us@keil.com) or 800-348-8051. Outside the US: [support.intl@keil.com](mailto:support.intl@keil.com).

For comments or corrections please email [bob.boys@arm.com](mailto:bob.boys@arm.com).

For the latest version of this document, contact the author, Keil Technical support or [www.keil.com](http://www.keil.com).

---

