TN0132
Technical note

STM32 Serial Wire Viewer and ETM capabilities with EWARM 5.40 and MDK-ARM 3.70

Introduction

This document presents Serial Wire Viewer (SWV) and Embedded Trace Macrocell (ETM) capabilities with these toolchains in various configurations:

- RVMDK 3.70 (RealView ® Microcontroller Development Kit from Keil™)
- EWARM 5.40 (Embedded Workbench® for ARM® from IAR Systems)

The STM32 provides a 4-bit ETM port as well as a Serial Wire Viewer port.

In general, the ETM is used to find problems using heavy duty trace debugging such as looking for difficult bugs, while the SWV is used to provide a low cost method of obtaining information from inside the MCU using ARM CoreSight™ technology.

1 SWV feature capabilities

1.1 Introduction

Serial Wire Viewer is the ability of the ARM™ core to send real-time trace information out via a single wire port called the Serial Wire Output (SWO). The trace information is in several familiar formats such as:

- Instrumentation Trace Macrocell (ITM) for application driven trace source that supports printf style debugging.
- Data Watchpoint and Trace (DWT) for variable monitoring and PC-sampling, which can in turn be used to periodically output the PC (sampled) or various CPU internal counters and to obtain profiling information from the target:
  - Program Counter sampling.
  - Data read and write cycles.
  - Variable and peripheral values.
  - Event counters.
  - Exception entry and return.
- Timestamps and CPU cycles are emitted relative to packets.

Our focus is on the analysis of the serial wire port's output information, in particular configurations, and to highlight its capabilities of providing information and data at the high speed that the STM32 runs at.

1.2 Context

This manual comes with a *.zip* file containing the subdirectories and files that make up the core of application examples.

These application examples are configured at 72 MHz (maximum frequency of the STM32 MCU) and highlight the following SWV features:

- Data access
- Interrupt
- Program counter sampling
- Printf

Each application example's folder contains:

- *inc* subfolder containing the example header files
- *src* subfolder containing the example source files
- *project* subfolder containing two projects that compile the example files:
  - *EWARMv5* containing the project for the EWARM toolchain
  - *ARM-MDK* containing the project for the ARM-MDK toolchain

These examples are tested in the following hardware and software conditions:

- SW/HW toolchain: EWARM 5.40/JLINK v6 and ARM-MDK 3.70/ULINK2
- Target board: STM3210E-Eval Rev.A
- Office PC Pentium® 4 CPU 3.20 GHz, 504 MB of RAM, SP2
- SW clock autodetected

1.3 Program counter (PC) sample

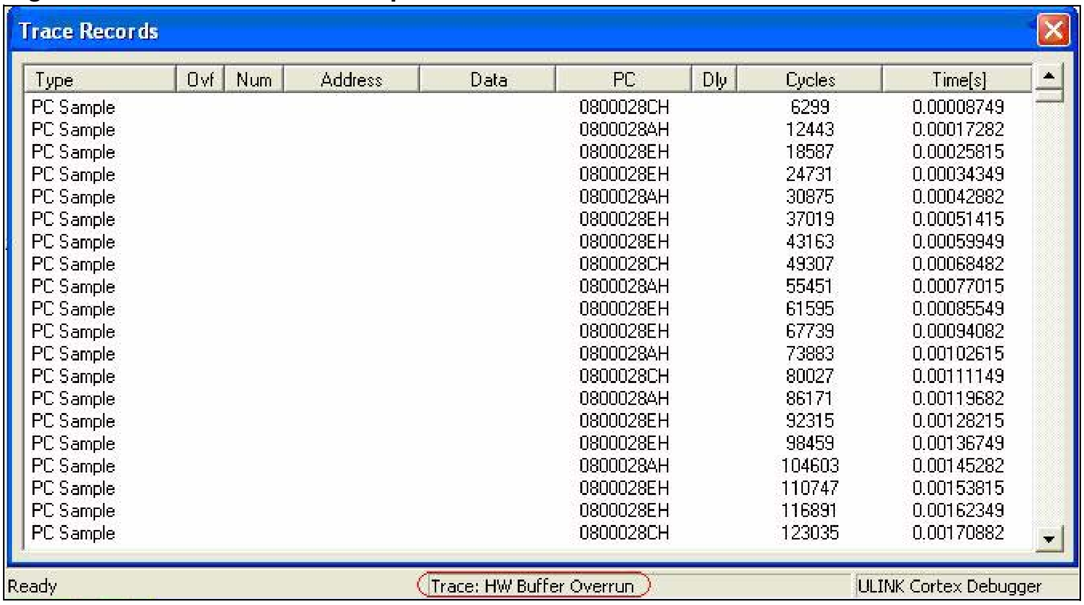
The display of program counter values is useful for program flow change, profile analysis and determining where the CPU might be caught in an infinite loop. Profile analysis gives a helpful indication where the CPU is spending its time.

Nevertheless in some conditions, the SWO and toochains are not capable of providing every program counter value because of the high speed that the STM32 runs at. *Section 1.3.1* and *Section 1.3.2* illustrate the limitations detected with both ARM-MDK and EWARM toolchains with particular configurations.

1.3.1 ARM-MDK / ULINK2 toolchain example

Running the PC Sampling example using the ARM-MDK toolchain highlights that if the PC sampling prescaler is equal to 7*1024 (10044 samples per second), a hardware buffer overrun occurs. This is due to the fact that the USB cannot accept data at the speed ULINK2 is sending it (see *Figure 1*).

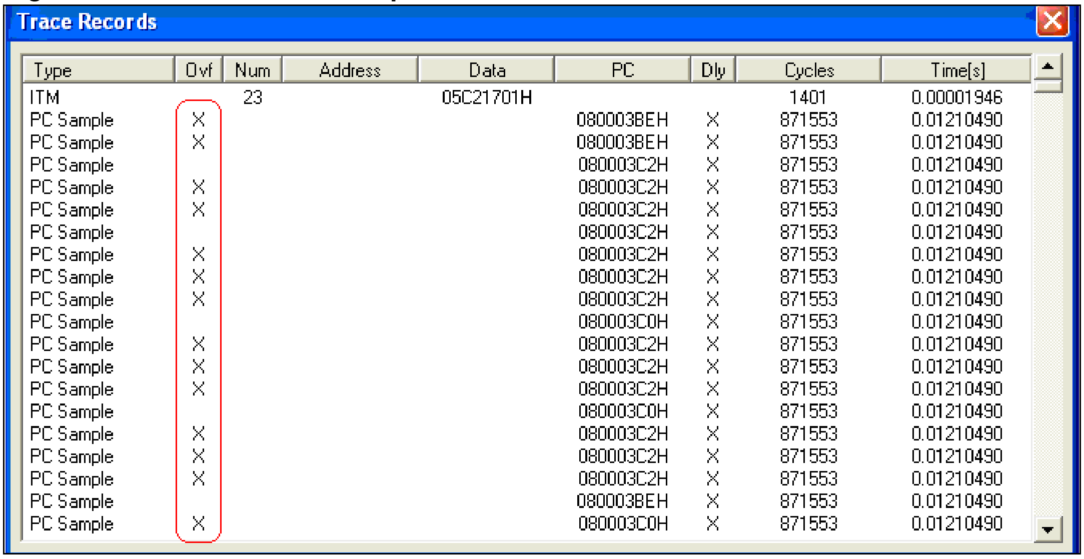
Figure 1. ARM-MDK PC samples: Hardware buffer overrun



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample					0800028CH		6299	0.00008749
PC Sample					0800028AH		12443	0.00017282
PC Sample					0800028EH		18587	0.00025815
PC Sample					0800028EH		24731	0.00034349
PC Sample					0800028AH		30875	0.00042882
PC Sample					0800028EH		37019	0.00051415
PC Sample					0800028EH		43163	0.00059949
PC Sample					0800028CH		49307	0.00068482
PC Sample					0800028AH		55451	0.00077015
PC Sample					0800028EH		61595	0.00085549
PC Sample					0800028EH		67739	0.00094082
PC Sample					0800028AH		73883	0.00102615
PC Sample					0800028CH		80027	0.00111149
PC Sample					0800028AH		86171	0.00119682
PC Sample					0800028EH		92315	0.00128215
PC Sample					0800028EH		98459	0.00136749
PC Sample					0800028AH		104603	0.00145282
PC Sample					0800028EH		110747	0.00153815
PC Sample					0800028EH		116891	0.00162349
PC Sample					0800028CH		123035	0.00170882

By decreasing the prescaler value until 3*1024 (23437 samples per second), an overflow occurs due to the fact that the SWO communication channel is not fast enough to handle that much data. Consequently, an overflow is detected as illustrated by *Figure 2*.

Figure 2. ARM-MDK PC samples: Overflow



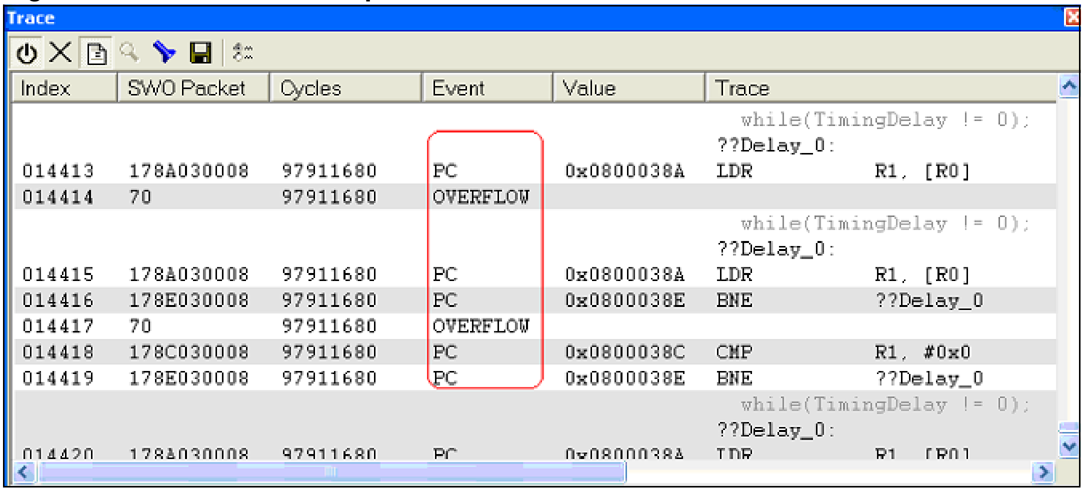
Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		23		05C21701H			1401	0.00001946
PC Sample	X				080003BEH	X	871553	0.01210490
PC Sample	X				080003BEH	X	871553	0.01210490
PC Sample					080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample					080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample					080003C0H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample					080003C0H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample	X				080003C2H	X	871553	0.01210490
PC Sample					080003BEH	X	871553	0.01210490
PC Sample	X				080003C0H	X	871553	0.01210490

1.3.2 EWARM / J-Link toolchain example

Running the PC Sampling example using the EWARM toolchain highlights that when the `Rate` (the number of samples per second) is set to 86538, the SWO communication channel is not fast enough to handle that much data. Consequently, an Overflow is detected, illustrated by *Figure 3*.

Figure 3. EWARM PC Samples: Overflow



Trace

Index	SWO Packet	Cycles	Event	Value	Trace
					while(TimingDelay != 0);
					??Delay_0:
014413	178A030008	97911680	PC	0x0800038A	LDR R1, [R0]
014414	70	97911680	OVERFLOW		
					while(TimingDelay != 0);
					??Delay_0:
014415	178A030008	97911680	PC	0x0800038A	LDR R1, [R0]
014416	178E030008	97911680	PC	0x0800038E	BNE ??Delay_0
014417	70	97911680	OVERFLOW		
014418	178C030008	97911680	PC	0x0800038C	CMP R1, #0x0
014419	178E030008	97911680	PC	0x0800038E	BNE ??Delay_0
					while(TimingDelay != 0);
					??Delay_0:
014420	178A030008	97911680	PC	0x0800038A	LDR R1, [R0]

1.4 Read and write data frames

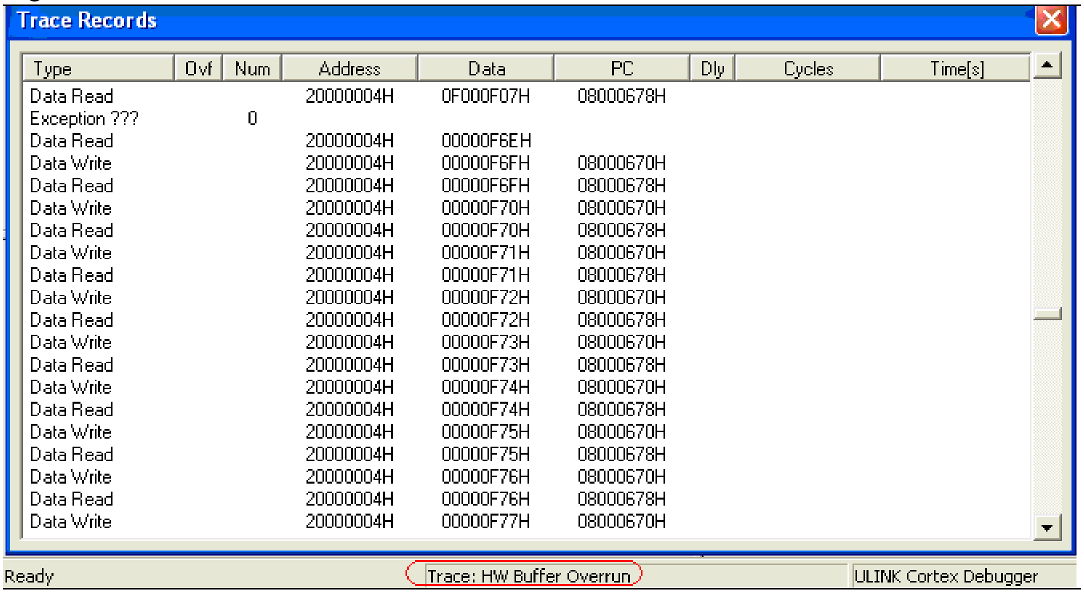
Read and write data frames can be displayed giving the address of the responsible instruction, the data value transferred, the data address and timestamps in both core cycles and seconds. *Figure 4* shows a series of data reads and writes showing these attributes.

1.4.1 ARM-MDK / ULINK2 toolchain example

Running the Data Access example using the ARM-MDK toolchain highlights that if a delay of less than 310 µs is inserted before incrementing the `j` variable, a hardware buffer overrun occurs. This is due to the fact that the USB cannot accept data access at the speed the ULINK2 is sending it (see *Figure 4*).

By decreasing the delay until 155 µs, an overflow occurs because the SWO communication channel is not fast enough to handle that much data access, for example, some values are not displayed in the trace record window (see *Figure 5*). To avoid these two problems in the Data Access example, the user should insert a delay greater than 310 µs.

Figure 4. ARM-MDK data read access: Hardware buffer overrun

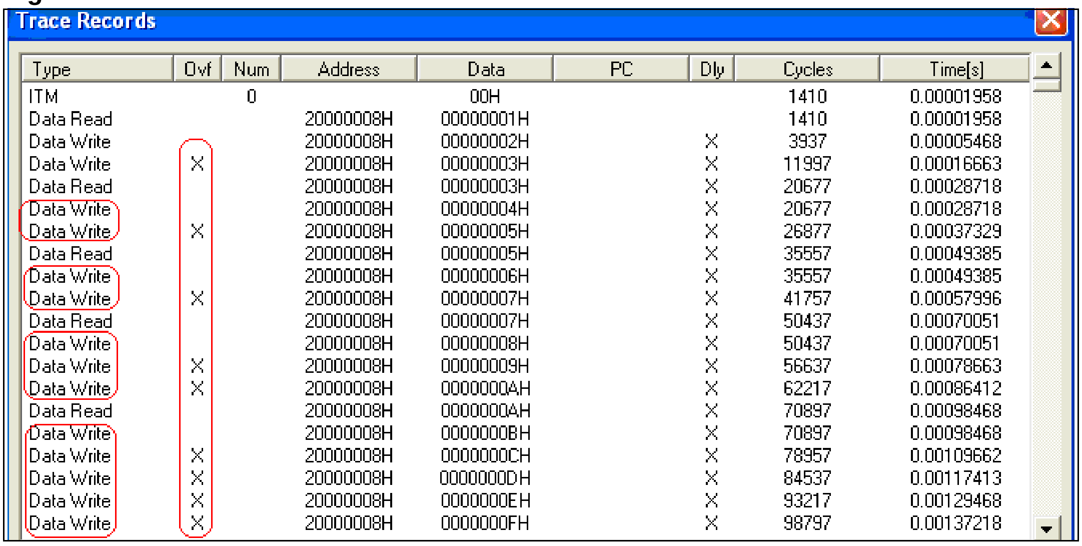


Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Read			20000004H	0F000F07H	08000678H			
Exception ???		0						
Data Read			20000004H	00000F6EH				
Data Write			20000004H	00000F6FH	08000670H			
Data Read			20000004H	00000F6FH	08000678H			
Data Write			20000004H	00000F70H	08000670H			
Data Read			20000004H	00000F70H	08000678H			
Data Write			20000004H	00000F71H	08000670H			
Data Read			20000004H	00000F71H	08000678H			
Data Write			20000004H	00000F72H	08000670H			
Data Read			20000004H	00000F72H	08000678H			
Data Write			20000004H	00000F73H	08000670H			
Data Read			20000004H	00000F73H	08000678H			
Data Write			20000004H	00000F74H	08000670H			
Data Read			20000004H	00000F74H	08000678H			
Data Write			20000004H	00000F75H	08000670H			
Data Read			20000004H	00000F75H	08000678H			
Data Write			20000004H	00000F76H	08000670H			
Data Read			20000004H	00000F76H	08000678H			
Data Write			20000004H	00000F77H	08000670H			

Ready | Trace: HW Buffer Overrun | ULINK Cortex Debugger

Figure 5. ARM-MDK data read access: Overflow



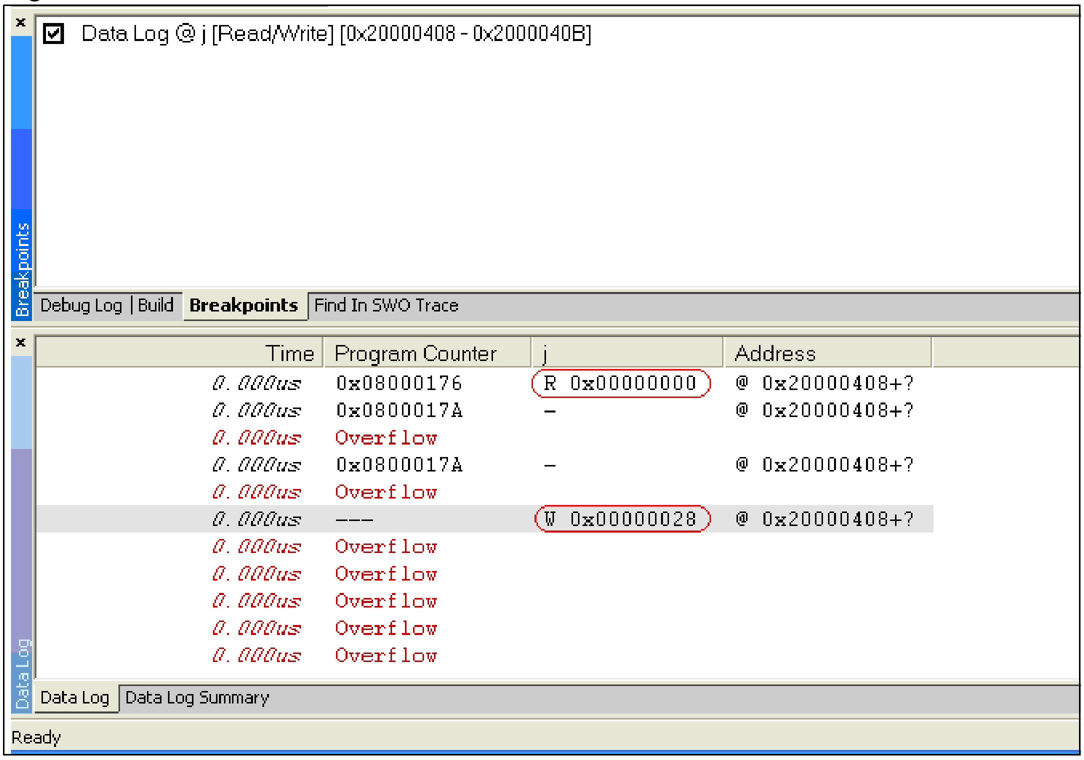
Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
ITM		0		00H			1410	0.00001958
Data Read			20000008H	00000001H			1410	0.00001958
Data Write			20000008H	00000002H		X	3937	0.00005468
Data Write	X		20000008H	00000003H		X	11997	0.00016663
Data Read			20000008H	00000003H		X	20677	0.00028718
Data Write			20000008H	00000004H		X	20677	0.00028718
Data Write	X		20000008H	00000005H		X	26877	0.00037329
Data Read			20000008H	00000005H		X	35557	0.00049385
Data Write			20000008H	00000006H		X	35557	0.00049385
Data Write	X		20000008H	00000007H		X	41757	0.00057996
Data Read			20000008H	00000007H		X	50437	0.00070051
Data Write			20000008H	00000008H		X	50437	0.00070051
Data Write	X		20000008H	00000009H		X	56637	0.00078663
Data Write	X		20000008H	0000000AH		X	62217	0.00086412
Data Read			20000008H	0000000AH		X	70897	0.00098468
Data Write			20000008H	0000000BH		X	70897	0.00098468
Data Write	X		20000008H	0000000CH		X	78957	0.00109662
Data Write	X		20000008H	0000000DH		X	84537	0.00117413
Data Write	X		20000008H	0000000EH		X	93217	0.00129468
Data Write	X		20000008H	0000000FH		X	98797	0.00137218

1.4.2 EWARM / J-Link toolchain example

Similar behavior is detected with the EWARM toolchain. In fact, to be able to perform a read access followed by a write access on all `j` values in the Data Log window, a delay of about 38 µs must be inserted before incrementing `j`. If not, only the first and last values are detected (see *Figure 6*).

Figure 6. EWARM data read access : Overflow



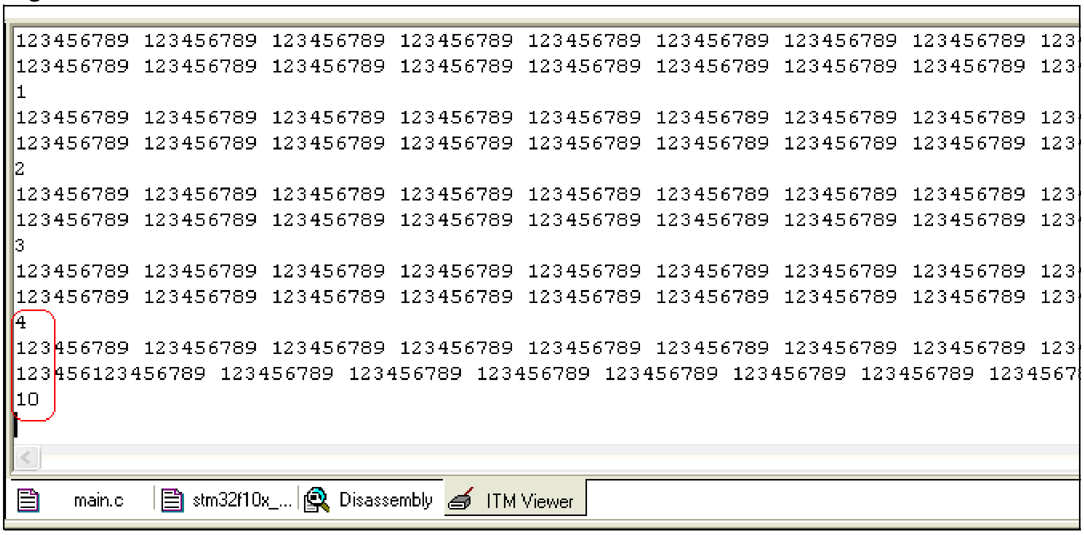
1.5 Printf

The Printf software example writes some data to a specific ITM address and CoreSight automatically sends this data to the SWO port. This data can be displayed on the **Serial Wire Viewer** window. This method is marginally intrusive to the user program and referred to as printf “debugging”.

1.5.1 ARM-MDK / ULINK2 toolchain example

When running the example using the ARM-MDK toolchain, an overload is detected if we send simultaneously 10 * 202 through the ITM port 0. Data is skipped from iterations 5 to 9 (see *Figure 7*). This is not an SWO related limitation but is due to the fact that the USB cannot accept data access at the speed the ULINK2 is sending it.

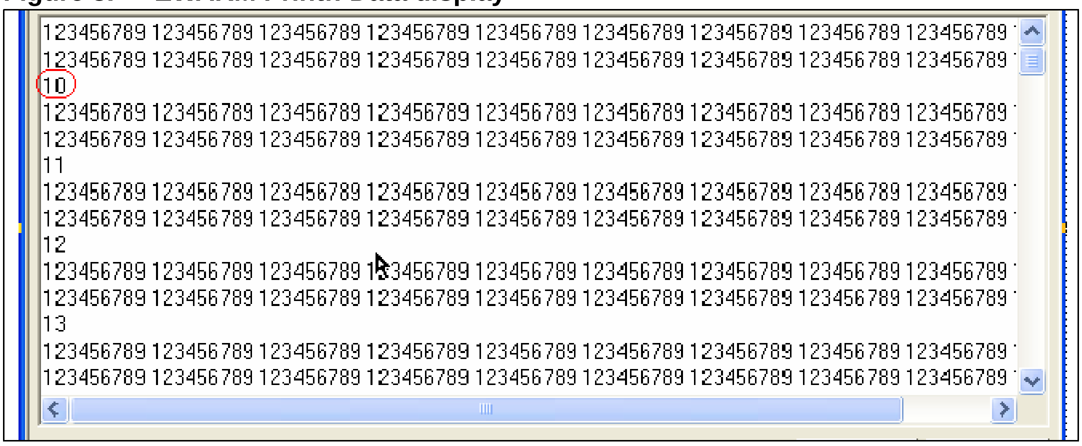
Figure 7. ARM-MDK Printf: Hardware buffer overrun



1.5.2 EWARM / J-Link toolchain example

With the EWARM toolchain, all values sent via the SWO are captured by EWARM (no overload is detected). The **Trace record** window can display the latest fifty lines of submitted data (see *Figure 8*). The user can consult all submitted data stored in the log file.

Figure 8. EWARM Printf: Data display



1.6 Exception trace dialog

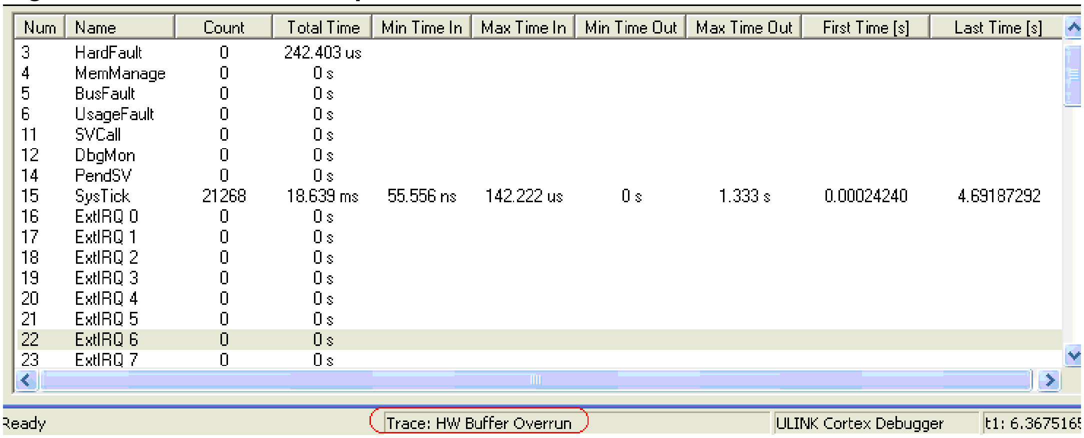
The interrupt example aims to determine the number of times that the interrupt was entered using 2 different methods:

1. Using the interrupt window: the SWV captures Systick exceptions' return and exit. These are timestamped and the exception number is then displayed.
2. Using a variable `Tick` incremented in the Systick interrupt handler.

1.6.1 ARM-MDK / ULINK2 toolchain example

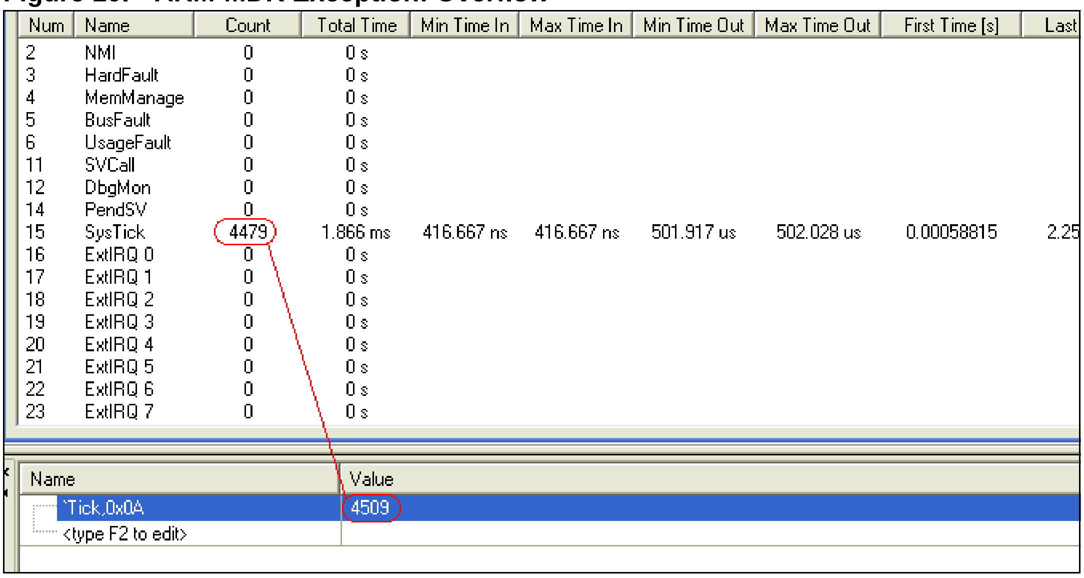
After running the example using the ARM-MDK toolchain, it is easy to see that the `Tick` variable's value and the Systick exception value in the output window are similar only when the interrupt periodicity is greater than or equal to 430 µs (see *Figure 9*). Otherwise some interrupts are missed due to the hardware buffer overrun. By decreasing the interrupt periodicity value until 160 µs, an Overflow is also detected (see *Figure 10*). To avoid these two problems in the Exception example, the user in this case should insert a delay of greater than 430 µs.

Figure 9. ARM-MDK Exception: Hardware buffer overrun



Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
3	HardFault	0	242.403 us						
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	21268	18.639 ms	55.556 ns	142.222 us	0 s	1.333 s	0.00024240	4.69187292
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 10. ARM-MDK Exception: Overflow



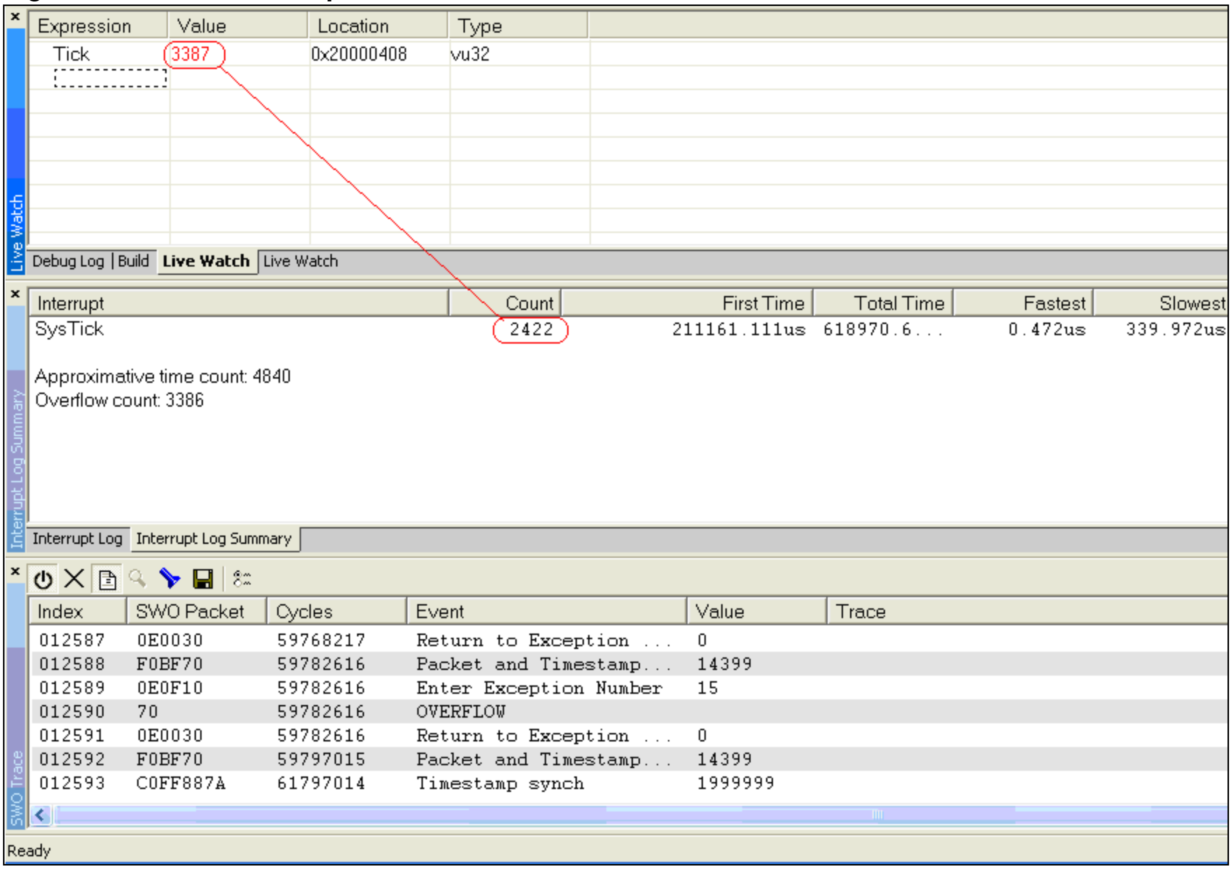
Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last
2	NMI	0	0 s						
3	HardFault	0	0 s						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	0	0 s						
12	DbgMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	4479	1.866 ms	416.667 ns	416.667 ns	501.917 us	502.028 us	0.00058815	2.25
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						

Name	Value
`Tick,0x0A	4509
<type F2 to edit>	

1.6.2 EWARM / J-Link toolchain example

Similar behavior is detected with the EWARM toolchain. In fact, a delay of 17.7 µs should be inserted to guarantee that the SysTick exception value displayed in the **Exception trace** window is the same as the `Tick` variable. If this condition is not verified, some exceptions are missed, as illustrated by *Figure 11*.

Figure 11. EWARM Exception: Overflow



2 ETM feature capabilities

2.1 Introduction

An embedded trace macrocell (ETM) is a real-time trace module providing program flow tracing.

For the STM32, the ETM unit provides a high bandwidth instruction trace over a dedicated 4-bit high-speed trace bus using a special hardware probe such as an IAR J-Trace for a Cortex-M3 or Signum JTAGjet-Trace.

This section describes the ETM features implemented by:

- ARM-MDK/JTAGjet-Trace
- EWARM/J-Trace CM3

The focus is on its ability to provide program flow information at the high speed that the STM32 runs at.

2.2 Context

This user guide comes with a zip file containing the subdirectories and files that makes up the core of application examples.

These application examples are configured at 72 MHz (maximum frequency of the STM32 MCU) and highlight the following trace features:

- Instruction timing
- Data tracing
- Function profiler

Each application example's folder contains:

- *inc* subfolder containing the example header files
- *src* subfolder containing the example source files
- *project* subfolder containing two projects that compile the example files:
  - *EWARMv5* containing the project for the EWARM toolchain
  - *ARM-MDK* containing the project for the ARM-MDK toolchain

These examples are tested in the following hardware and software conditions:

- SW/HW toolchain: EWARM 5.40/J-Trace CM3 and ARM-MDK 3.70/JTAGjet-Trace
- Target board: STM3210E-Eval Rev.A
- Office PC Pentium® 4 CPU 3.20 GHz, 504 MB of RAM, SP2

These examples use the following ETM options:

- Stall processor on FIFO full
- Trace buffer size: 0x400000
- Trace port mode: Normal, half-rate clocking.

2.3 Instruction timing

The ETM allows reconstruction of program execution which is useful for debugging and especially for detecting rare bugs source.

J-Trace CM3 and JTAGjet-Trace have a 4 MB buffer for trace data storage. Since for Cortex-M3 one trace frame corresponds to approximately 1 byte, the buffer overflows after approximately 4 million trace frames.

The instruction timing may be inferred from the timestamp of the transmitted frame.

2.3.1 EWARM / J-Trace toolchain example

This feature is not supported in EWARM 5.40/J-Trace CM3.

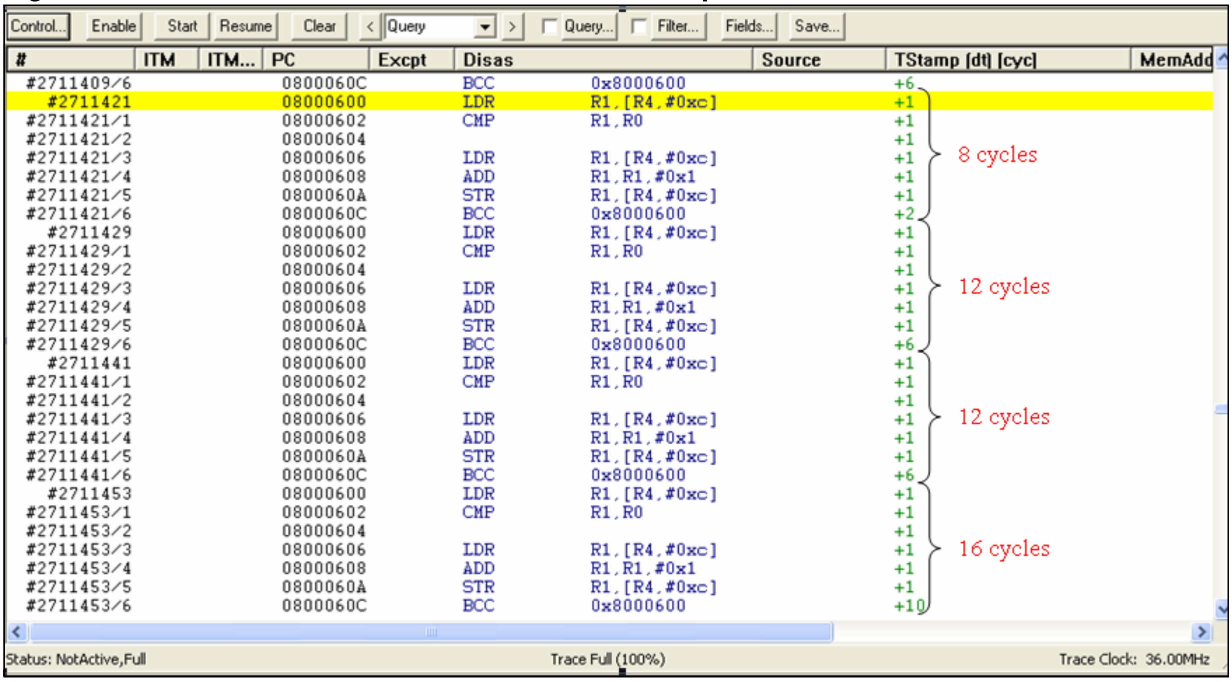
2.3.2 ARM-MDK / JTAGjet-Trace toolchain example

The trace window contains the timestamp `TStamp` field. It shows useful information on the traced code such as the execution time (in cycles) for each instruction.

After running the Instruction Timing example, where every physical sample contains the same 6 instructions, logically, the same execution time should be found for each sample. However, this is not the case, see *Figure 12*.

Generally, the timestamp cannot be relied upon for very short times, but the error of +/- 16 cycles has no real effect once longer times are being observed.

Figure 12. ETM Trace window in ARM-MDK: Timestamps



2.4 Data tracing

Data tracing is a useful feature that allows the debugger to trace variable values or addresses during program execution in addition to the program flow. The STM32 ETM does not have data tracing capability. Nevertheless, data values and/or addresses can be traced by controlling the DWT and ITM on-chip modules.

2.4.1 EWARM / J-Trace toolchain example

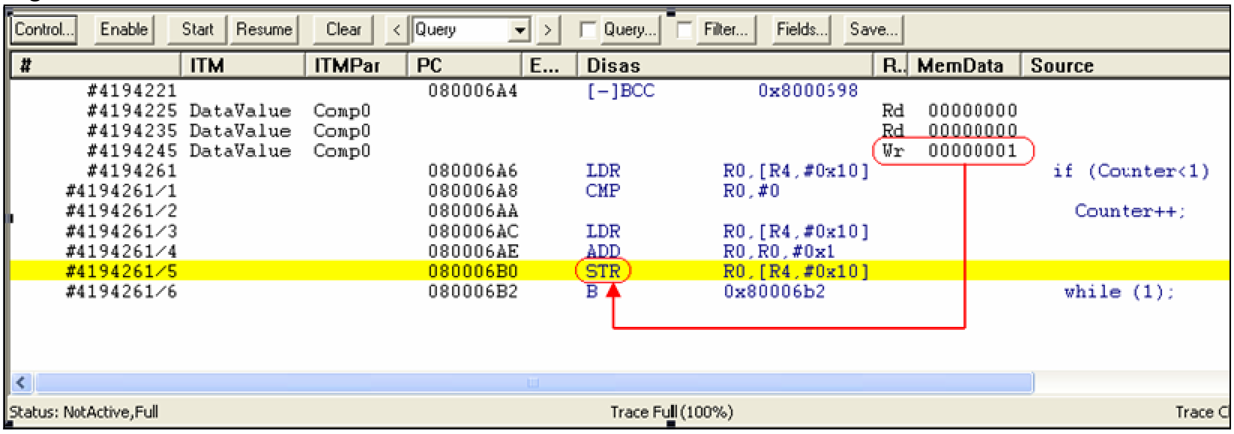
This feature is not supported in EWARM 5.40/J-Trace CM3.

2.4.2 ARM-MDK / JTAGjet-Trace toolchain example

Using JTAGjet, it is possible to insert ITM trace data in the trace stream since it shares the same trace port with the ETM.

In the Data Tracing example, the ITM unit is configured to send data values when accessing the `Counter` variable. Meanwhile, the two trace sources (ITM and ETM) may get out of phase due to FIFO latency. The analysis of the code in the ETM trace window, see *Figure 13*, shows that the STR instruction is displayed after an ITM write record.

Figure 13. ETM Trace window in ARM-MDK



2.5 Function profiler

The function profiler helps find the functions where most time is spent during application execution, and the number of calls of each function.

The following example aims to determine the number of times that a function was called using 2 different ways:

1. Using the function profiler.
2. Using a variable `Tick` incremented in the function interrupt handler

2.5.1 ARM-MDK / JTAGjet-Trace toolchain example

This feature is not supported in ARM-MDK 3.70/JTAGjet-Trace.

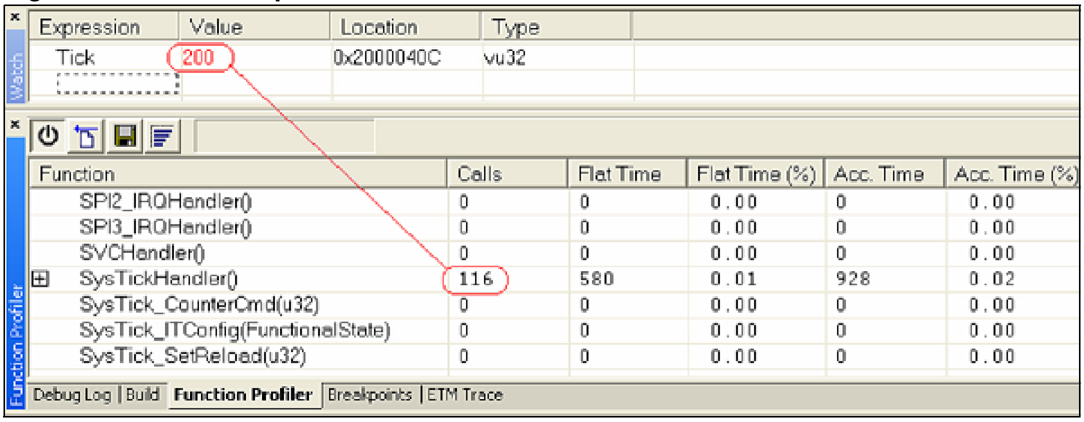
2.5.2 EWARM / J-Trace toolchain example

Using EWARM 5.40, the function profiler is available using SWV with J-Link and ETM trace with J-Trace. If the ETM is used, it shows also the number of calls of each function in addition to the time spent inside it.

Figure 14 shows the watch window with the `Tick` variable which is incremented every `SysTickHandler()` entry, and the function profiler window of the Function Profiler example. `SysTickHandler()` is the handler of the systick exception which occurs periodically (every 1 ms).

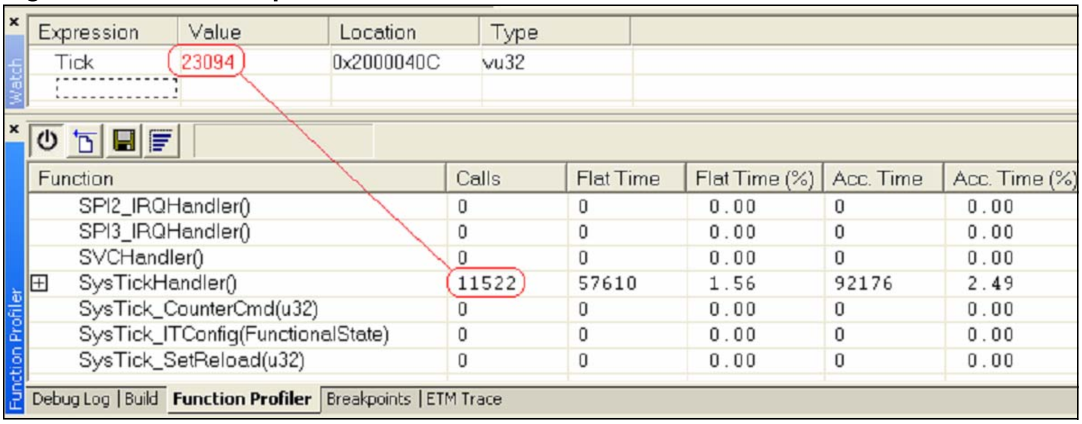
As illustrated by *Figure 14*, the number of `SysTickHandler()` calls is missed. This is due to the fact that the function profiler is based on the last ETM trace data collected, which does not contain all the function entries.

Figure 14. Function profiler window in EWARM



If the SysTick exception occurs more frequently (every 10 µs for example), the difference is be more noticeable as shown in *Figure 15*.

Figure 15. Function profiler window in EWARM



To avoid this discordance, the user should use a pre-filtered trace using the `Trace start` and `Trace stop` breakpoints to limit the amount of trace data.

3 Conclusion

Regarding the SWV feature, clearly the single serial wire port is not able to provide all trace information, because of the high speed of the STM32. Nevertheless, a statistical sampling indication of performance analysis is possible with both EWARM and ARM-MDK toolchains which is sufficiently adequate for profile analysis for the SWV feature.

Concerning the ETM feature, the buffer size of the trace capture device limits the instruction trace capability, nonetheless the STM32 ETM is able to provide an efficient debug and instruction trace especially when it is combined with the ITM trace to obtain a complete debug solution.

4 Revision history

Table 1. Document revision history

Date	Revision	Changes
27-Jan-2010	1	Initial release.