



White paper

**System analysis and real-time tracing
with Serial Wire Viewer (SWV)**

COPYRIGHT NOTICE

© Copyright 2011 Atollic AB. All rights reserved. No part of this document may be reproduced or distributed without the prior written consent of Atollic AB.

TRADEMARK

Atollic, Atollic TrueSTUDIO, Atollic TrueINSPECTOR, Atollic TrueVERIFIER and Atollic TrueANALYZER and the Atollic logotype are trademarks or registered trademarks owned by Atollic. ECLIPSE™ is a registered trademark of the Eclipse foundation. ARM and Cortex are trademarks or registered trademarks of ARM Ltd. All other product names are trademarks or registered trademarks of their respective owners.

DISCLAIMER

The information in this document is subject to change without notice and does not represent a commitment of Atollic AB. The information contained in this document is assumed to be accurate, but Atollic assumes no responsibility for any errors or omissions. In no event shall Atollic AB, its employees, its contractors, or the authors of this document be liable for any type of damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

DOCUMENT IDENTIFICATION

ASW-WPSWV December 2011

REVISION

First version

Atollic AB
Science Park
Gjuterigatan 7
SE- 553 18 Jönköping
Sweden

+46 (0) 36 19 60 50

E-mail: sales@atollic.com

Web: www.atollic.com

Atollic Inc
115 Route 46
Building F, Suite 1000
Mountain Lakes, NJ 07046-1668
USA

+1 (973) 784 0047 (Voice)
+1 (877) 218 9117 (Toll Free)
+1 (973) 794 0075 (Fax)

E-mail: sales.usa@atollic.com

Web: www.atollic.com

Contents

Abstract	1
Introduction.....	2
Serial Wire Viewer overview	3
Serial Wire Debug (SWD)	3
Serial Wire Output (SWO)	3
Serial Wire Viewer (SWV).....	3
Instrumentation Trace Macro cell (ITM).....	4
System analysis and real-time tracing.....	5
SWV Trace configuration.....	5
SWV Trace log and graphical timeline chart	7
SWV Data Trace and graphical timeline chart	8
SWV Exception Trace Log and graphical timeline chart.....	10
SWV ITM graphical timeline chart	10
The ITM Console.....	11
SWV Statistical profiling.....	12
Summary.....	13

Tables

No table of figures entries found.

ABSTRACT

ARM® Cortex® microcontrollers continue to push the price/performance ratio to unprecedented levels. In addition, the inclusion of a subset of the CoreSight debug architecture in these device families improves debugging capabilities substantially without incurring excessive cost. It is now possible to have greater visibility into the dynamics of complex real-time embedded applications than ever before. This visibility is extremely useful not only in the increasingly complex applications typically found in today's products, but in applications that cannot be halted for the debugging process.

This white paper outlines how the Serial Wire viewer and associated technologies which are part of the CoreSight architecture can be used for advanced debugging while embedded applications execute at full speed.

INTRODUCTION

Finding bugs in software is a difficult and time consuming process. Debugging is usually more difficult in many embedded applications for several reasons. Execution timing is often critical, and results can be skewed by the inclusion of instrumentation code. In cases of motor, actuator or other control applications, it may not be possible, or particularly revealing to stop the device under test for debugging operations. In many applications, it is desirable to examine resources as the application executes in order to preserve synchronization of interrupts as well as internal or external events.

The Serial Wire Viewer (SWV) technology from ARM, found in Cortex-M microcontroller product families, provides real-time tracing and memory access with little or no processor overhead while the application executes at full speed. This is made possible by the internal architecture of the devices, and does not require software support such as low priority daemons or monitor code that must be linked with the application. The Instrumentation Trace Macro cell (ITM) is a minimally intrusive way to transmit data via one of the 32 ITM ports as the application executes. This enables printf() style debugging without a physical UART, USB, Ethernet or other type of communication channel.

This white paper gives a brief background to the Serial Wire Viewer (SWV) and Instrumentation Trace Macro cell (ITM) technology. It also outlines how advanced debuggers can exploit the SWV and ITM technology to deliver powerful debug capabilities to developers. Many embedded developers continue to use “tried and true” methods of debugging such as blinking LED and printf() outputs. When the ease of use, information content and the cost efficiency of modern debugging technology is considered, it is well worth a developer’s while to devote the relatively small amount of effort to becoming familiar with these methodologies as the payback in time savings in debugging and testing will be considerable.

SERIAL WIRE VIEWER OVERVIEW

To use system analysis and real-time tracing in compatible ARM processors, a number of different technologies interact; Serial Wire Viewer (SWV), Serial Wire Debug (SWD) and Serial Wire Output (SWO). Their respective roles will be explained below.

SERIAL WIRE DEBUG (SWD)

Serial Wire Debug (SWD) is a debug port similar to JTAG, and provides the same debug capabilities (run, stop on breakpoints, single-step) but with fewer pins. It replaces the JTAG connector with a 2-pin interface (one clock pin and one bi-directional data pin). The SWD port itself does not provide for real-time tracing.

SERIAL WIRE OUTPUT (SWO)

The Serial Wire Output (SWO) pin can be used in combination with SWD and is used by the processor to emit real-time trace data, thus extending the two SWD pins with a third pin. The combination of the two SWD pins and the SWO pin enables Serial Wire Viewer (SWV) real-time tracing in compatible ARM processors.

SERIAL WIRE VIEWER (SWV)

Serial Wire Viewer is a real-time trace technology that uses the Serial Wire Debugger (SWD) port and the Serial Wire Output (SWO) pin. Serial Wire Viewer provides advanced system analysis and real-time tracing without the need to halt the processor to extract certain types of debug information.

Serial Wire Viewer (SWV) can provide the following types of target information:

- Periodic samples of program counter (PC) values
- Event notification on memory accesses (such as reading/writing C variables)
- Event notification on exception entry and exit
- Event counters
- Timestamp and CPU cycle information

Based on this trace data, modern debuggers can provide developers with advanced debugger capabilities.

The developer can configure SWV in many different ways, to make it emit various combinations of information, e.g. values of memory locations, events, PC values etc. As more types of trace data are enabled for transmission, the more trace packages are needed to be sent from the CPU to the PC debugger. It is possible in certain circumstances that all of the data may not be received by the front end debugging software on the PC.

There are several reasons for this. The target CPU running at full speed can exceed bandwidth on the one-pin SWO output. In addition, trace data makes its way to the SWO via internal buffers which are limited in depth. If the CPU generates more trace packages than the SWO pin can transmit, there will be resulting packet loss. This is normal.

The reason is that Cortex-M microcontrollers are designed to meet aggressive cost targets. The resource limitation is a byproduct of the tradeoff between resources and cost, and is to some extent, “by design”. While this imposes minor limitations on what can be done, it is also true that SWV and ITM, used with care, are capable of providing very powerful debug information that in some cases are otherwise unobtainable information without very expensive debug hardware.

INSTRUMENTATION TRACE MACRO CELL (ITM)

The Instrumentation Trace Macro cell (ITM) enables applications to write arbitrary data to the SWO pin, which can then be interpreted and visualized in the debugger in various ways. For example, ITM can be used to redirect printf() output to a console view in the debugger.

This is highly desirable and useful for several reasons. First, the overhead is very low as a memory location can be transferred in a single clock cycle. Second, there is no requirement for an on-board resource such as a UART or USB connection, as the data is transmitted using the debug probe hardware. Third, the ITM port has 32 channels, and by writing different types of data to different ITM channels, the debugger can interpret or visualize the data on various channels differently.

Since writing a byte to the ITM port only takes one clock cycle, applications can thus keep the ITM trace instrumentation when the product ships. It is then not needed to rebuild and retest a product build with instrumentation removed. Leaving this kind of instrumentation in the shipped code also provides the capability of connecting a debugger to a live system in-the-field for detailed analysis in deployed products.

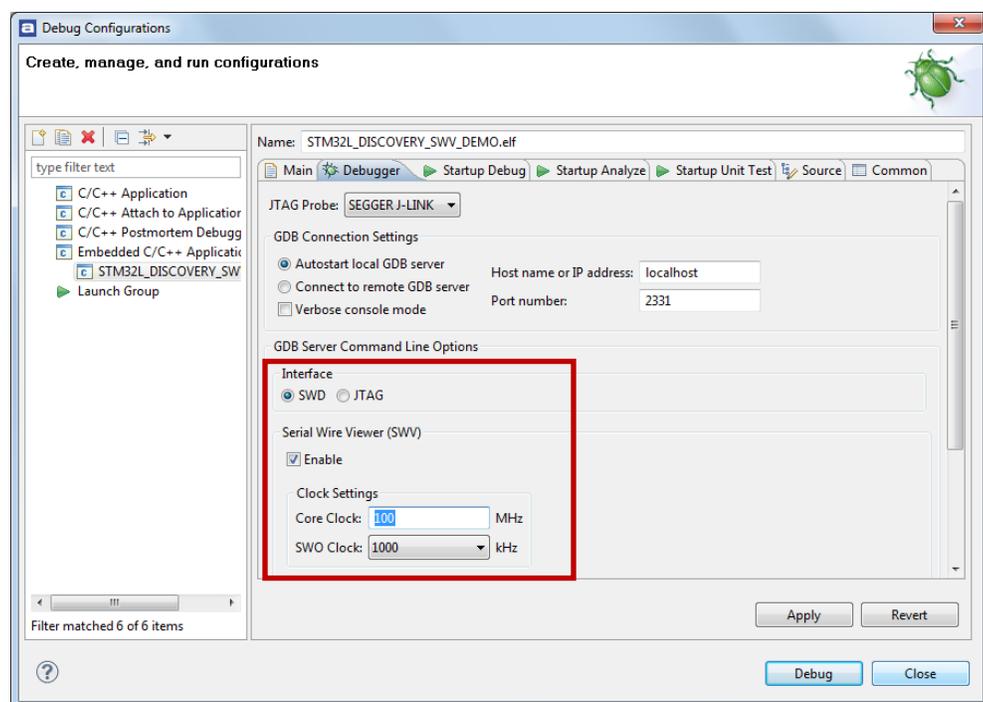
SYSTEM ANALYSIS AND REAL-TIME TRACING

Modern debuggers can exploit Serial Wire Viewer (SWV) to provide a number of advanced debugger capabilities to embedded developers. This section outlines some of the Serial Wire Viewer debugger features found in **Atollic TrueSTUDIO®** (compatible ARM targets only).

SWV TRACE CONFIGURATION

Serial Wire Viewer (SWV) can be enabled in the **Atollic TrueSTUDIO®** debug configuration dialog. To enable SWV, enable it by selecting the corresponding checkbox and select the CPU core clock frequency and desired SWO clock speed.

All JTAG probes do not support SWV. For example the first generation of ST-LINK does not, but the ST-LINK/V2 does. Older Segger J-Link do not, but current models do. For J-Link, a debugger shipped after about mid-2009 will support SWV.

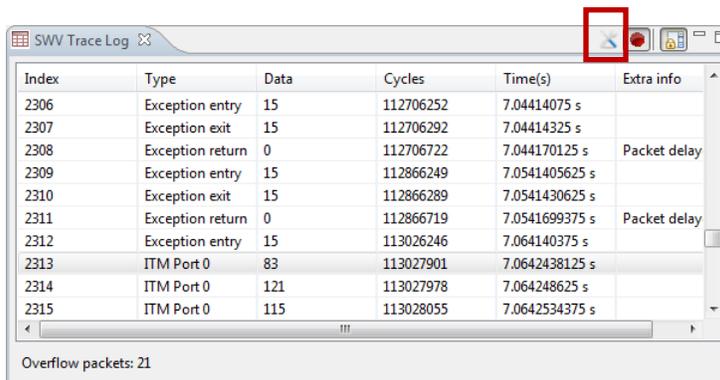


To make SWV work, it is important to make sure SWD mode is selected, SWV is enabled, and that the CPU core clock speed and SWO clock is configured correctly.

It is particularly important to note the SWD/JTAG radio buttons. Sometimes older projects provided by semiconductor manufacturers have JTAG set by default, and overlooking this is a sure way to disable SWV.

Once SWV has been enabled, a debug session can be started and execution runs to the first breakpoint, typically in the power-on-reset interrupt handler or the first line in the main() function.

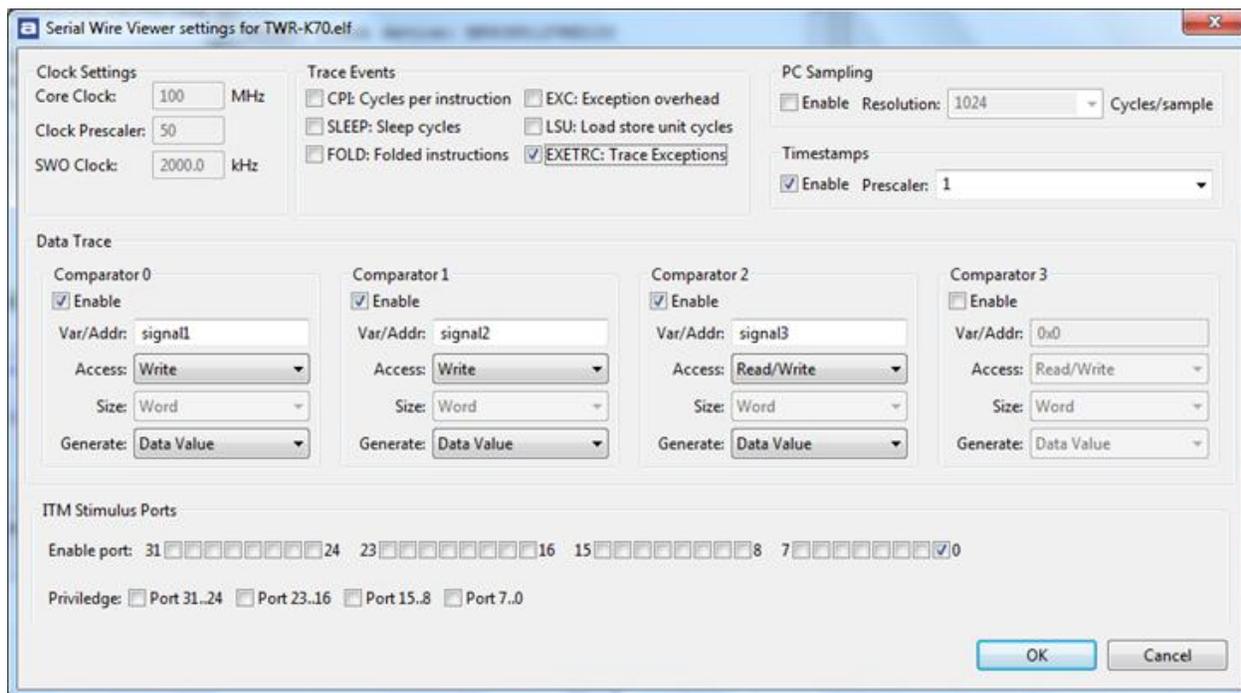
Configuration of the events that are to be captured is configured in the SWV Trace configuration dialog box. This dialog box is opened by clicking on the “hammer and wrench” toolbar button in any of the SWV docking views.



Index	Type	Data	Cycles	Time(s)	Extra info
2306	Exception entry	15	112706252	7.04414075 s	
2307	Exception exit	15	112706292	7.04414325 s	
2308	Exception return	0	112706722	7.044170125 s	Packet delay
2309	Exception entry	15	112866249	7.0541405625 s	
2310	Exception exit	15	112866289	7.0541430625 s	
2311	Exception return	0	112866719	7.0541699375 s	Packet delay
2312	Exception entry	15	113026246	7.064140375 s	
2313	ITM Port 0	83	113027901	7.0642438125 s	
2314	ITM Port 0	121	113027978	7.064248625 s	
2315	ITM Port 0	115	113028055	7.0642534375 s	

Overflow packets: 21

Once the SWV configuration dialog box is opened, the user is presented with a number of SWV trace settings:



Serial Wire Viewer settings for TWR-K70.elf

Clock Settings
Core Clock: 100 MHz
Clock Prescaler: 50
SWO Clock: 2000.0 kHz

Trace Events
 CPI: Cycles per instruction
 EXC: Exception overhead
 SLEEP: Sleep cycles
 LSU: Load store unit cycles
 FOLD: Folded instructions
 EXETRC: Trace Exceptions

PC Sampling
 Enable Resolution: 1024 Cycles/sample

Timestamps
 Enable Prescaler: 1

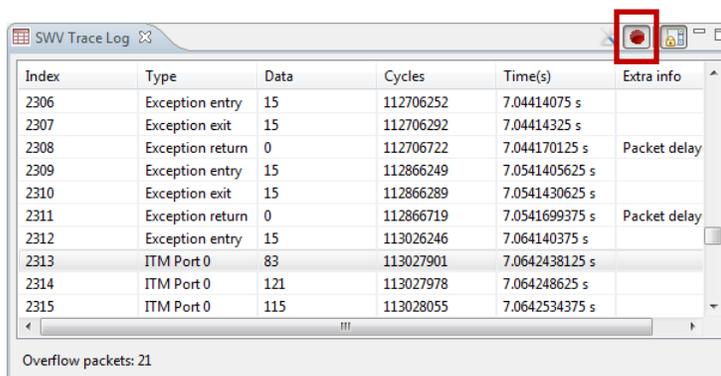
Data Trace

Comparator 0 <input checked="" type="checkbox"/> Enable Var/Addr: signal1 Access: Write Size: Word Generate: Data Value	Comparator 1 <input checked="" type="checkbox"/> Enable Var/Addr: signal2 Access: Write Size: Word Generate: Data Value	Comparator 2 <input checked="" type="checkbox"/> Enable Var/Addr: signal3 Access: Read/Write Size: Word Generate: Data Value	Comparator 3 <input type="checkbox"/> Enable Var/Addr: 0x0 Access: Read/Write Size: Word Generate: Data Value
---	---	--	---

ITM Stimulus Ports
Enable port: 31 24 23 16 15 8 7 0
Privilege: Port 31..24 Port 23..16 Port 15..8 Port 7..0

OK Cancel

When the desired SWV event packages have been selected, the trace record button must be activated to enable recording of trace records. This is done by clicking on the red “record” button. Seeing its display changed is the way it is shown to be activated.



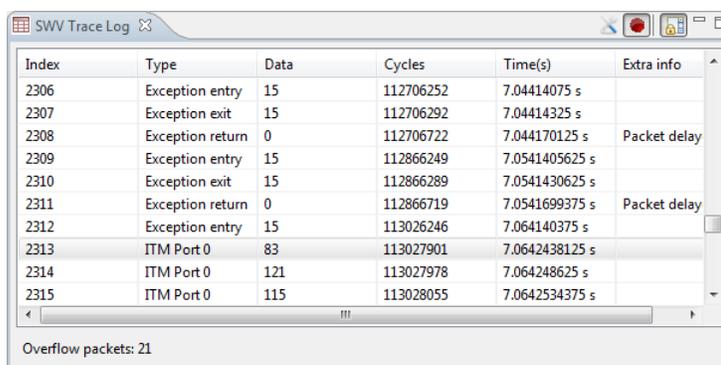
Index	Type	Data	Cycles	Time(s)	Extra info
2306	Exception entry	15	112706252	7.04414075 s	
2307	Exception exit	15	112706292	7.04414325 s	
2308	Exception return	0	112706722	7.044170125 s	Packet delay
2309	Exception entry	15	112866249	7.0541405625 s	
2310	Exception exit	15	112866289	7.0541430625 s	
2311	Exception return	0	112866719	7.0541699375 s	Packet delay
2312	Exception entry	15	113026246	7.064140375 s	
2313	ITM Port 0	83	113027901	7.0642438125 s	
2314	ITM Port 0	121	113027978	7.064248625 s	
2315	ITM Port 0	115	113028055	7.0642534375 s	

Overflow packets: 21

With serial wire viewer tracing now enabled and the desired event notification types configured, trace recording is activated. As the application executes, the debugger will receive SWV trace records from the processor via the debug probe, and the debugger software will update its trace views in real-time as long as target execution continues.

SWV TRACE LOG AND GRAPHICAL TIMELINE CHART

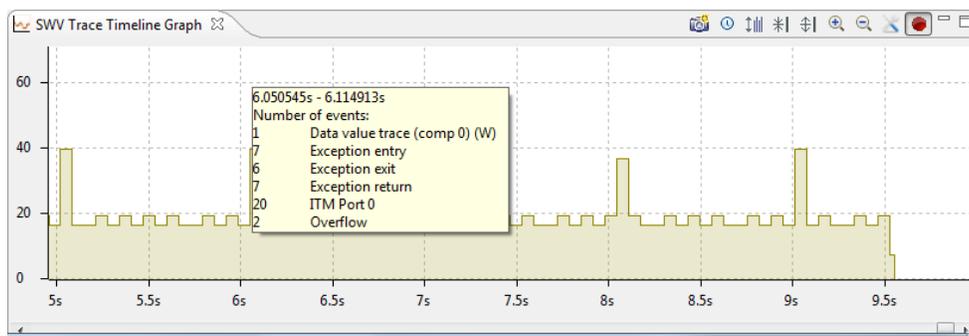
The Serial Wire Viewer master view is the **SWV Trace Log**, and the accompanying graphical timeline chart. The **SWV Trace Log** lists all trace records that has been recorded during the current debug session.



Index	Type	Data	Cycles	Time(s)	Extra info
2306	Exception entry	15	112706252	7.04414075 s	
2307	Exception exit	15	112706292	7.04414325 s	
2308	Exception return	0	112706722	7.044170125 s	Packet delay
2309	Exception entry	15	112866249	7.0541405625 s	
2310	Exception exit	15	112866289	7.0541430625 s	
2311	Exception return	0	112866719	7.0541699375 s	Packet delay
2312	Exception entry	15	113026246	7.064140375 s	
2313	ITM Port 0	83	113027901	7.0642438125 s	
2314	ITM Port 0	121	113027978	7.064248625 s	
2315	ITM Port 0	115	113028055	7.0642534375 s	

Overflow packets: 21

The **SWV Trace Log** provides detailed information on the recorded trace records. If a quick overview of the tracing results is desired, the graphical **SWV Trace Timeline Graph** view is better, as it visualize the trace data over time in an easy to understand manner.



Both the **SWV Trace Log** and the **SWV Trace Timeline Graph** views are updated in real time, and the graph even has smooth scrolling with various zoom capabilities. The graph view provides features for switching between time and clock cycle mode, and has a feature that saves a screenshot of the graph to a file.

SWV DATA TRACE AND GRAPHICAL TIMELINE CHART

The **SWV Data Trace Log**, and its accompanying graphical timeline chart, provides information about current values of memory locations and C variables, and all memory accesses that have been recorded during the current debug session.

The **SWV Data Trace Log** display real-time updated values for selected C variables or memory locations, including different 8/16/32-bit integer and floating point data types. By selecting a monitored data object in the **Watch** pane, the memory access list for that data object is listed in the **History** pane with detailed information as well.

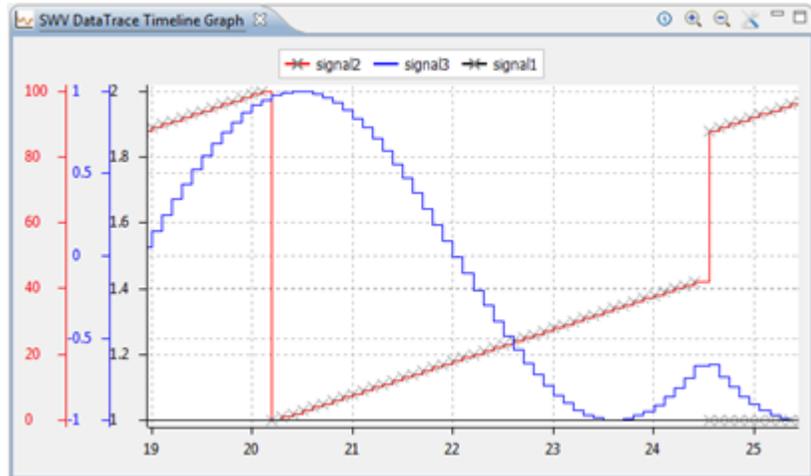
The figure is a screenshot of the 'SWV Data Trace' window. It is divided into two main sections: 'Watch' and 'History (signal3)'. The 'Watch' section contains a table with three rows of monitored variables. The 'History' section contains a table listing memory access events for the selected variable 'signal3'.

Comp	Name	Value
0	signal1	1
1	signal2	39
2	signal3	0.34995136

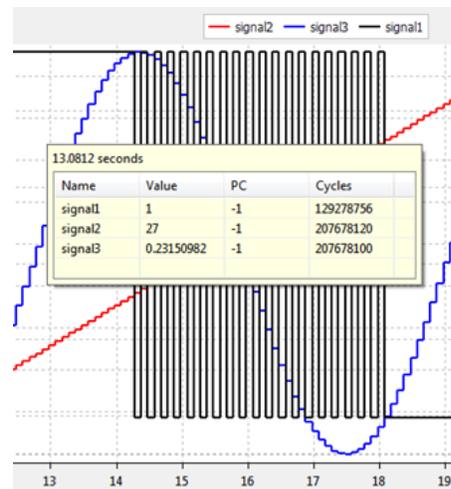
Access	Value	PC	Cycles
WRITE	0.91580963	0xd1700	2895493005
WRITE	0.871114	0xd1700	2905492998
WRITE	0.81776625	0xd1700	2915492931
WRITE	0.75622165	0xd1700	2925492888
WRITE	0.68712115	0xd1700	2935492533
WRITE	0.61115515	0xd1700	2945492498
WRITE	0.5290827	0xd1700	2955492471
WRITE	0.4417238	0xd1700	2965492435
WRITE	0.34995136	0xd1700	2975492408

If the SWV tracing is configured to emit program counter (PC) values for every memory access, a double click in the memory access history list automatically open the source code file and line number that caused that specific memory access.

Like the **SWV Trace Log**, the **SWV Data Trace Log** has an accompanying graphical **SWV DataTrace Timeline Graph** view. This graphical chart provides unprecedented overview of how different variable values change over time during execution, and how they relate to each other.



By moving the mouse over a specific part of the diagram, a tooltip displays additional information, like timestamp and the exact values of all monitored C variables or memory addresses at that time position.

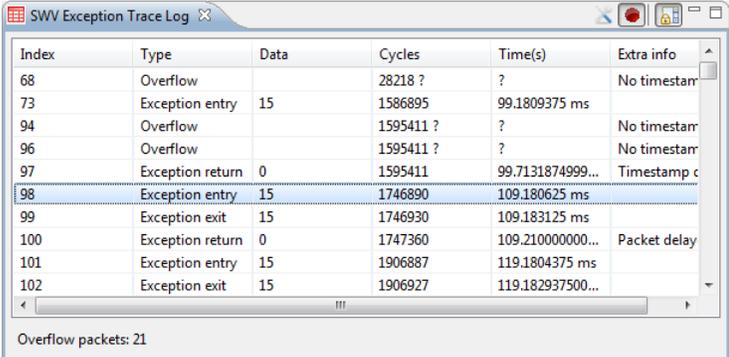


Both the **SWV Data Trace Log** and the **SWV DataTrace Timeline Graph** views are updated in real time during execution, and the graph even has smooth scrolling with various zoom capabilities. The graph view provides features for switching between time and clock cycle mode, and has a feature that saves a screenshot of the graph to a file.

Note: The **SWV DataTrace Timeline Graph** view is not shipping at the time of publishing this document. It will be released in **Atollic TrueSTUDIO for ARM v2.3**, with release date scheduled for 15th of December 2011.

SWV EXCEPTION TRACE LOG AND GRAPHICAL TIMELINE CHART

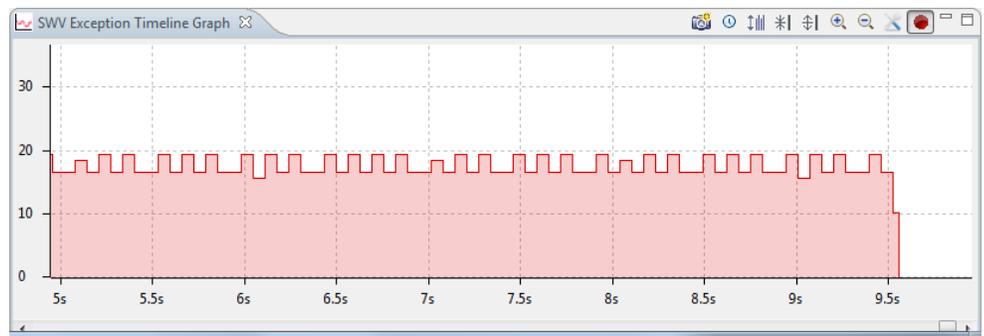
The **SWV Exception Trace Log**, and its accompanying graphical timeline chart, provides information about the interrupt and exception events that have been recorded during the current debug session.



Index	Type	Data	Cycles	Time(s)	Extra info
68	Overflow		28218 ?	?	No timestam
73	Exception entry	15	1586895	99.1809375 ms	
94	Overflow		1595411 ?	?	No timestam
96	Overflow		1595411 ?	?	No timestam
97	Exception return	0	1595411	99.7131874999...	Timestamp c
98	Exception entry	15	1746890	109.180625 ms	
99	Exception exit	15	1746930	109.183125 ms	
100	Exception return	0	1747360	109.210000000...	Packet delay
101	Exception entry	15	1906887	119.1804375 ms	
102	Exception exit	15	1906927	119.182937500...	

Overflow packets: 21

Like the **SWV Trace Log**, the **SWV Exception Trace Log** has an accompanying graphical **SWV Exception Timeline Graph** view.

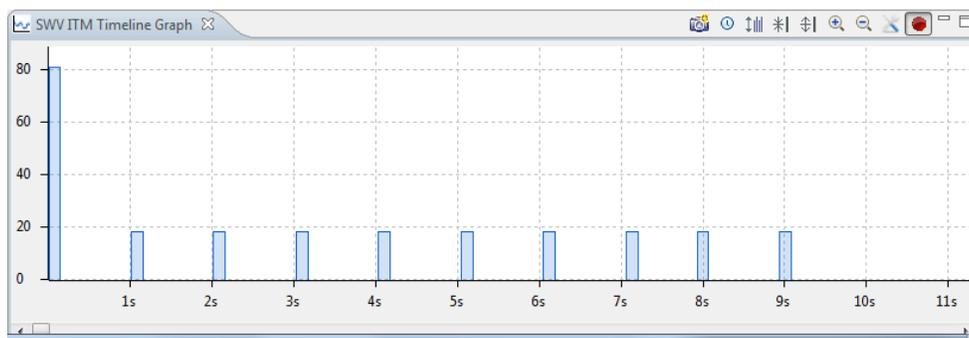


By moving the mouse over a specific part of the diagram, a tooltip displays additional information dependent on zoom level. If the diagram is zoomed out, summary information is displayed in the tooltip. If the diagram is zoomed in to a certain level, the tooltip displays detailed information about the trace record(s) under the mouse.

SWV ITM GRAPHICAL TIMELINE CHART

The **SWV ITM Timeline Graph** provides graphical information about the ITM trace output from the target application, such as trace records from redirected printf() output.

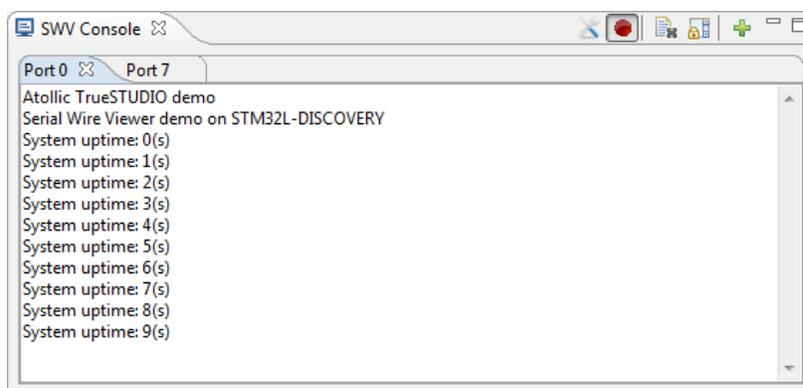
This debugger view can be quite useful. If the application writes a data byte to a certain ITM channel before and after a code section is executed, this gives direct information about when execution of that code section started and stopped. It is also possible to subtract the timestamp of the two ITM packages to obtain a precise measurement of its exact execution time. This is very useful for timing critical sections of code.



By moving the mouse over a specific part of the diagram, a tooltip displays additional information dependent on zoom level. If the diagram is zoomed out, summary information is displayed in the tooltip. If the diagram is zoomed in to a certain level, the tooltip displays detailed information about the trace record(s) under the mouse.

THE ITM CONSOLE

The **ITM Console** is specifically designed to print human readable text output from the target application, typically by the means of a `printf()` that redirects its output to an ITM channel.



The **ITM Console** prints data from the ITM channel 0 by default, but by clicking on the **Add channel** toolbar button, additional tabs with output from other ITM channels can be created. The application can then be instrumented to let different subsystems write trace strings to different channels.

For example, the main application can write trace strings to ITM channel 0, while a Flash file system or TCP/IP stack might be instrumented to write trace messages to the ITM channel 7.

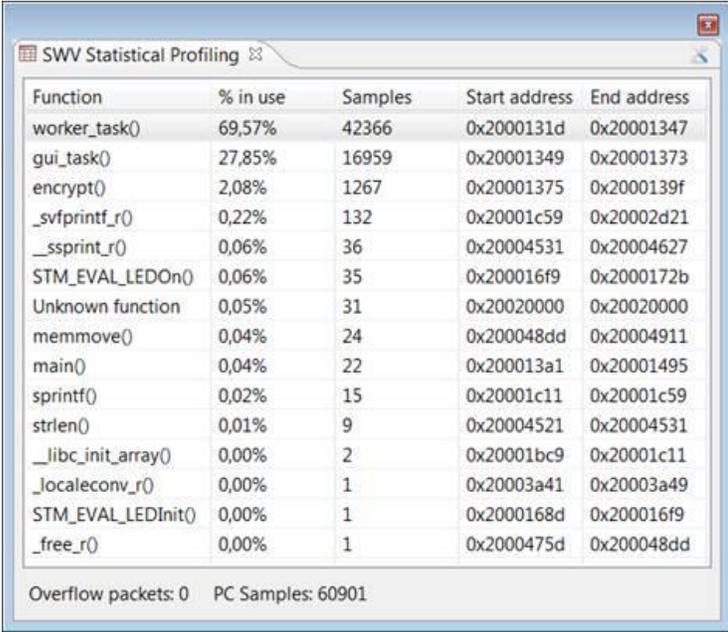
Not only can their output be individually turned on or off in the **SWV Configuration** dialog, but the output end up in unique console windows too.

SWV STATISTICAL PROFILING

The **SWV Statistical Profiling** view uses the Serial Wire Viewer program counter (PC) sampling, to provide statistical samples of where the CPU executes code at periodic snapshots. This enables the debugger to work out in which C-functions execution is made most often, thus enabling an execution time profiling feature.

As the program counter values are only received at periodic snapshots, the profiling is built on statistical trends, rather than exact and 100% accurate information. It is for example possible that execution of a function (that may be running frequently) is completely missed if the PC samples by random happens to be taken at snapshot times where the processor is executing other code.

Over time however, this risk becomes smaller and smaller, and so the statistical profiling feature still is very valuable in understanding where in your program execution spend most time.



Function	% in use	Samples	Start address	End address
worker_task()	69,57%	42366	0x2000131d	0x20001347
gui_task()	27,85%	16959	0x20001349	0x20001373
encrypt()	2,08%	1267	0x20001375	0x2000139f
_svfprintf_r()	0,22%	132	0x20001c59	0x20002d21
__ssprint_r()	0,06%	36	0x20004531	0x20004627
STM_EVAL_LEDOn()	0,06%	35	0x200016f9	0x2000172b
Unknown function	0,05%	31	0x20020000	0x20020000
memmove()	0,04%	24	0x200048dd	0x20004911
main()	0,04%	22	0x200013a1	0x20001495
sprintf()	0,02%	15	0x20001c11	0x20001c59
strlen()	0,01%	9	0x20004521	0x20004531
__libc_init_array()	0,00%	2	0x20001bc9	0x20001c11
__localeconv_r()	0,00%	1	0x20003a41	0x20003a49
STM_EVAL_LEDInit()	0,00%	1	0x2000168d	0x200016f9
_free_r()	0,00%	1	0x2000475d	0x200048dd

Overflow packets: 0 PC Samples: 60901

Note: The **SWV Statistical Profiling** view is not shipping at the time of publishing this document. It will be released in **Atollic TrueSTUDIO for ARM v2.3**, with release date scheduled for 15th of December 2011.

SUMMARY

Many of the hardware components of electronics products are becoming less expensive as time goes on. This trend is actually driving up software development and testing costs, as the adoption of cheaper hardware enables more product features that must be supported by software. This means more bugs to find, and less time to find them. The information provided by SWV and ITM can be valuable tools in a developer's repertoire when used properly.

Being able to visualize the time evolution of specific variables and other events as the application executes can provide the developer valuable insights into topics such as:

- Whether or not control algorithms are functioning properly
- Whether or not memory locations are being corrupted inadvertently
- Whether or not pointers are behaving as expected
- Locating sections of code that require optimization
- Locating specific lines of code that are causing memory corruption
- Determining whether or not interrupts are firing as expected

The problems listed above, as well as others, are frequently the source of "million dollar bugs," so named because of the expense in time, money, and potential loss of company reputation associated with these bugs. There are certainly further debugging strategies that can be implemented based on the information provided by SWV and ITM. These are only limited by the developer's imagination.

Atollic TrueSTUDIO® provides embedded developers with advanced features for powerful debugging using Serial Wire Viewer technology, while at the same time simplifying their work and shortening development and testing time.

Atollic provides a family of well integrated tools for professional embedded systems development and debugging, static source code analysis, test automation and test quality measurement.

More information about Atollic, **Atollic TrueSTUDIO®**, **Atollic TrueINSPECTOR®**, **Atollic TrueANALYZER®** and **Atollic TrueVERIFIER™** products is available here:

www.atollic.com

www.atollic.com/truestudio

www.atollic.com/trueinspector

www.atollic.com/trueanalyzer

www.atollic.com/trueverifier