Getting Started with Red Trace

Red Suite 4

November 10, 2011

Table of Contents

1 Overview

1.1 Red Trace and Serial Wire Viewer

Red Trace is the collective name for the technology incorporated into Red Suite and associated debug probes to access the Serial Wire Viewer (SWV) functionality provided by Cortex-M3/M4 based systems. SWV offers levels of visibility into your debug target never before possible without expensive external trace capture boxes. It also requires only one extra pin on top of the standard Serial Wire Debug (SWD) connection.

Notes

- Use of Red Trace requires connection to the target system using a compatible debug probe. This includes Code Red's Red Probe, Red Probe+, and TI's ICDI, but not LPC-Link or the RDB-Link of an RDB1768v2 board.
- If you are using Red Suite with an evaluation license, then you will only be able to invoke a single Red Trace run per debug session.

1.2 Red Trace Views

Red Trace presents target information collected from a Cortex-M3/M4 based system in a number of different trace views within the Red Suite IDE

Each trace view is presented within the **Debug Perspective** or the **Develop Perspective** by default.

Trace views which are not required may be closed to simplify the user interface. They may be re-opened using the "Window->Show View->Other…" menu item, as per Figure 1.

The following trace views are provided :

Profile

- Provides a statistical profile of application activity.

Interrupt Statistics

- Provides counts and timing information for interrupt handlers.

Interrupt Trace

- Provides a time-based graph of interrupts being entered and exited, including nesting.

Data Watch

- Provides the ability to monitor (and update) any memory location in real-time, without stopping the processor.

Host Strings

- Provide a very low overhead means of displaying diagnostic messages as your code is running

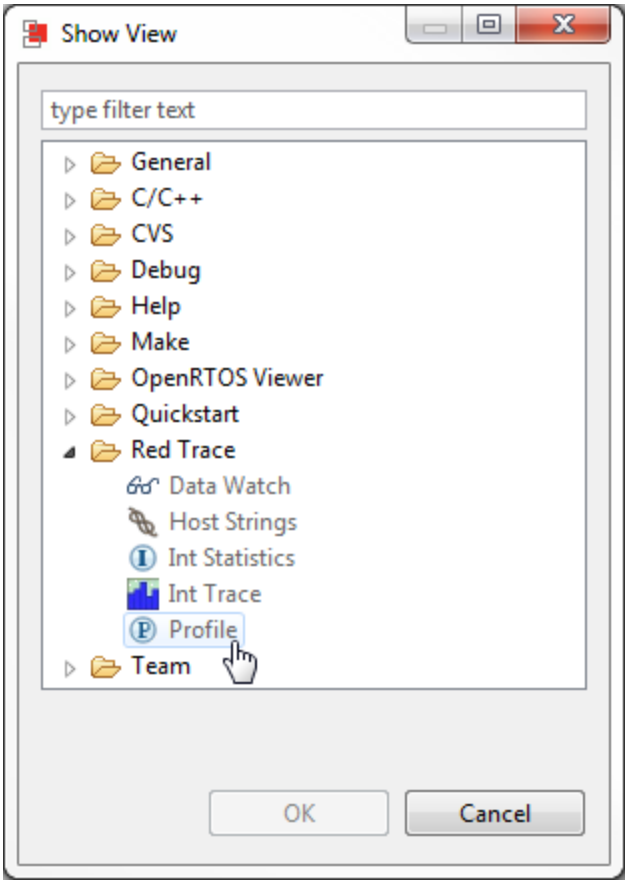


Figure 1 - Reopening a view

All views, except for Host Strings, are completely non-intrusive. They do not require any changes to the application, nor any special build options; they function on completely standard applications.

Each trace view provides a set of toolbar buttons which are used to control the collection and presentation of trace information. Starting data collection within one trace view will result in data collection for other views being suspended. The data presentation area of each trace view is enabled only when data collection for the view is active.

2 Configuration

2.1 Starting Red Trace

To use Red Trace, you must be debugging an application on a Cortex-M3/M4 based MCU, connected via a supported debug probe

You may start Red Trace at any time whilst debugging your program. The program does NOT have to be stopped at a breakpoint. Before the collection of data commences, Red Trace will prompt for settings related to your target processor, as per Figure 2.

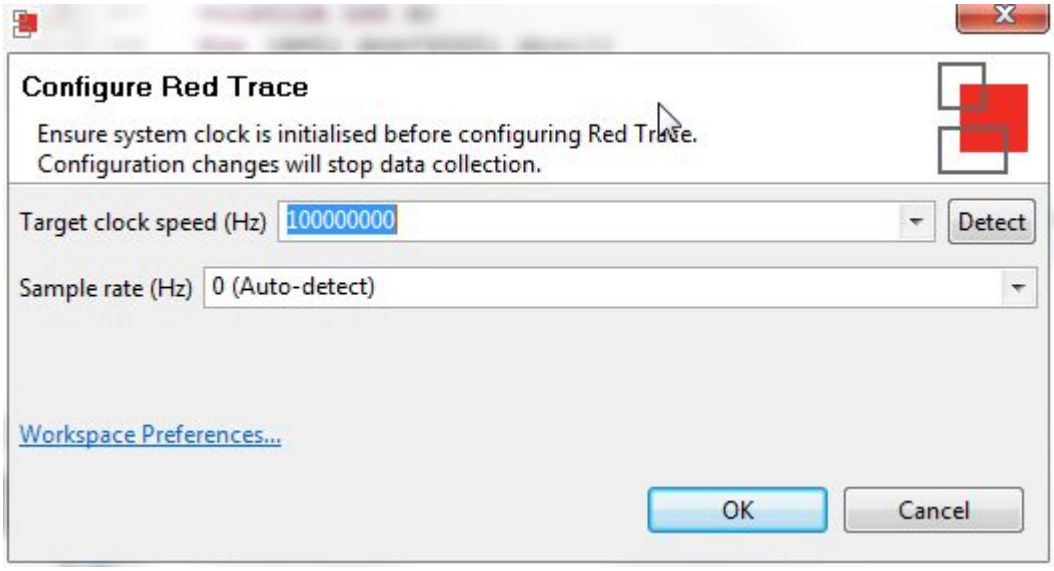


Figure 2 - Configure Red Trace

Target Clock Speed

The **Detect** button will detect the *current* clock speed of the target processor. Due to the way the Trace data is transferred by the target processor, setting the correct clock speed is essential to determine the correct baud rate for the data transfer. If the clock speed setting does not match the actual clock speed of the processor, data will be lost, and the trace data will be meaningless.

Care needs to be taken when to ensure that when using the Detect button, the application has set the clock speed for normal operation, otherwise the Detect button will provide an inappropriate value. Clock set up code may be executed by the startup code run before the breakpoint at the start of main() is hit. For example in NXP projects which use CMSIS – Cortex Microcontroller Software Interface Standard – this will be done by the startup code calling the SystemInit() function. However some projects – for instance the TI/Luminary StellarisWare examples - clock setup is typically done within the main() function itself.

Sample rate

This is the frequency of data collection (sampling) from the target. You can normally leave this set to the default Auto-detect, which will provide a sample rate of approximately 50kHz, depending on the target clock speed. Available sample rates are calculated from the clock speed and can be seen in the dropdown.

2.2 Start Trace

Once you are debugging your application, press the **Start Trace** button to begin the collection of trace data and enable the updating (refresh) of the view at regular intervals. Once trace has been started, updates of the view may be paused by pressing the button again (now reading **Stop Trace**). Collection of data will continue while refreshing of the view is paused.

2.3 Refresh 

Press the Refresh button to start the collection of trace data (if trace had not already been started) and to perform a single update (refresh) of the view. The collection of data will continue. You may switch between continuous refresh (using the **Start Trace** button) and individual refreshes of the view in order to inspect the collected data in more detail.

2.4 Settings

If it becomes necessary to change the system clock speed or the data sampling rate during a debugging session, use the Settings button to notify Red Trace. Any changes will stop data collection.

2.5 Reset Trace

Press the Reset Trace button to reset any cumulative data presented in the view. The collection of data will continue.

2.6 Save Trace

Press the Save Trace button to save the contents of the trace view to a file in CSV format. This can be useful for offline analysis of the data in a spreadsheet, for example.

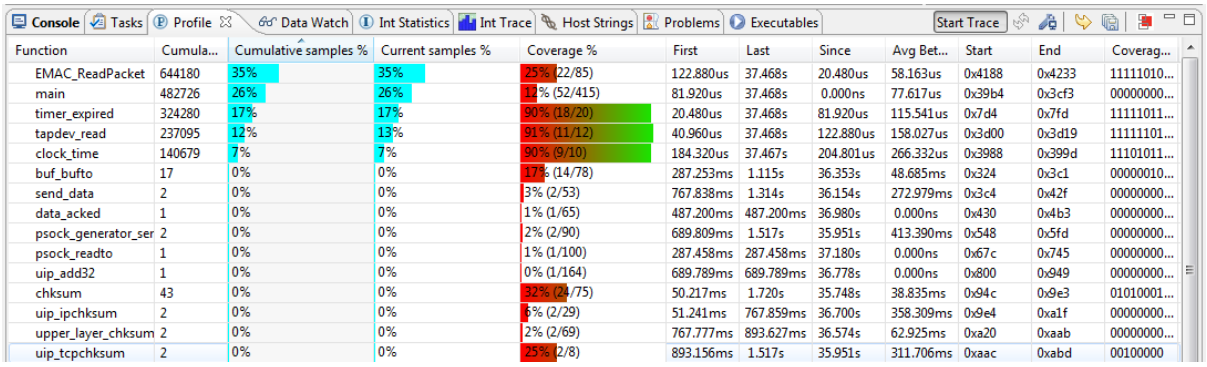
3 Profile Trace

3.1 Overview

Profile tracing provides a statistical profile of application activity. This works by sampling the program counter (PC) at the configured sample rate (typically around 50 kHz). It is completely non-intrusive to the application – it does not affect the performance in any way. As profile tracing provides a *statistical* profile of the application, more accurate results can be achieved by profiling for as long as possible. Profile tracing can be useful for identifying application behavior such as code hotspots.

3.2 Profile view

The Profile view gives a profile of the code as it is running, providing a breakdown of time spent in different functions. An example screenshot is shown below:



Function	Cumula...	Cumulative samples %	Current samples %	Coverage %	First	Last	Since	Avg Bet...	Start	End	Coverag...
EMAC_ReadPacket	644180	35%	35%	25% (22/85)	122.880us	37.468s	20.480us	58.163us	0x4188	0x4233	11111010...
main	482726	26%	26%	12% (52/415)	81.920us	37.468s	0.000ns	77.617us	0x39b4	0x3cf3	00000000...
timer_expired	324280	17%	17%	90% (18/20)	20.480us	37.468s	81.920us	115.541us	0x7d4	0x7fd	11111011...
tapdev_read	237095	12%	13%	91% (11/12)	40.960us	37.468s	122.880us	158.027us	0x3d00	0x3d19	11111101...
clock_time	140679	7%	7%	90% (9/10)	184.320us	37.467s	204.801us	266.332us	0x3988	0x399d	11101011...
buf_bufto	17	0%	0%	17% (14/78)	287.253ms	1.115s	36.353s	48.685ms	0x324	0x3c1	00000010...
send_data	2	0%	0%	3% (2/53)	767.838ms	1.314s	36.154s	272.979ms	0x3c4	0x42f	00000000...
data_acked	1	0%	0%	1% (1/65)	487.200ms	487.200ms	36.980s	0.000ns	0x430	0x4b3	00000000...
psock_generator_ser	2	0%	0%	2% (2/90)	689.809ms	1.517s	35.951s	413.390ms	0x548	0x5fd	00000000...
psock_readto	1	0%	0%	1% (1/100)	287.458ms	287.458ms	37.180s	0.000ns	0x67c	0x745	00000000...
uip_add32	1	0%	0%	0% (1/164)	689.789ms	689.789ms	36.778s	0.000ns	0x800	0x949	00000000...
chksum	43	0%	0%	32% (24/75)	50.217ms	1.720s	35.748s	38.835ms	0x94c	0x9e3	01010001...
uip_ipchksum	2	0%	0%	6% (2/29)	51.241ms	767.859ms	36.700s	358.309ms	0x9e4	0xa1f	00000000...
upper_layer_chksum	2	0%	0%	2% (2/69)	767.777ms	893.627ms	36.574s	62.925ms	0xa20	0xaab	00000000...
uip_tcpchksum	2	0%	0%	25% (2/8)	893.156ms	1.517s	35.951s	311.706ms	0xaac	0xabd	00100000

Figure 3 - Profile View

- **Cumulative samples**: This is the total number of PC samples that have occurred while executing the particular function.
- **Cumulative samples (%)**: This is the same as above, but displayed as a percentage of total PC samples collected.

- **Current samples (%)**: Number of samples in this function in the last data collection (refresh period)
- **Coverage (%)**: Number of instructions in the function that have been seen to have been executed.
- **First**: This is the first time (relative to the start time of tracing) that the function was sampled.
- **Last**: This is the last time (relative to the start time of tracing) that the function was sampled.
- **Since**: It is this long since you last saw this function. (current - last)
- **Avg Between**: This is the average time between executions of this function.

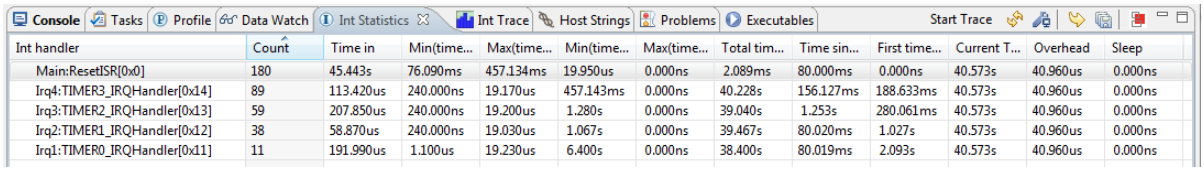
4 Interrupts

4.1 Overview

Interrupt tracing provides information on the interrupt performance of your application. This can be used to determine time spent in interrupt handlers and to help optimize their performance.

4.2 Interrupt Statistics view

The Interrupt Statistics view displays counts and timing information for interrupt service handlers. An example screenshot is shown below:



Int handler	Count	Time in	Min(time...	Max(time...	Min(time...	Max(time...	Total tim...	Time sin...	First time...	Current T...	Overhead	Sleep
Main:ResetISR[0x0]	180	45.443s	76.090ms	457.134ms	19.950us	0.000ns	2.089ms	80.000ms	0.000ns	40.573s	40.960us	0.000ns
Irq4:TIMER3_IRQHandler[0x14]	89	113.420us	240.000ns	19.170us	457.143ms	0.000ns	40.228s	156.127ms	188.633ms	40.573s	40.960us	0.000ns
Irq3:TIMER2_IRQHandler[0x13]	59	207.850us	240.000ns	19.200us	1.280s	0.000ns	39.040s	1.253s	280.061ms	40.573s	40.960us	0.000ns
Irq2:TIMER1_IRQHandler[0x12]	38	58.870us	240.000ns	19.030us	1.067s	0.000ns	39.467s	80.020ms	1.027s	40.573s	40.960us	0.000ns
Irq1:TIMER0_IRQHandler[0x11]	11	191.990us	1.100us	19.230us	6.400s	0.000ns	38.400s	80.019ms	2.093s	40.573s	40.960us	0.000ns

Figure 4 - Interrupt Statistics View

Information displayed includes:

- **Count**: The number of times the interrupt routine has been entered so far.
- **Time In**: The total time spent in the interrupt routine so far.
- **Min (time in)**: Minimum time spent in the interrupt routine for a single invocation.
- **Max (time in)**: The Maximium time spent in a single invocation of the routine.
- **Min (time between)**: The minimum time between invocations of the interrupt.
- **Max (time between)**: The maximum time between invocations of the interrupt.

- **Total time between**: The total time spent outside of this interrupt routine.
- **Time since last**: The time elapsed since the last time in this interrupt routine.
- **First time run**: The time (relative to the start of trace) that this interrupt routine was first run.
- **Current time**: The elapsed time since trace start.
- **Overhead**: The cumulative overhead time elapsed entering and exiting all handlers
- **Sleep**: Time the processor spent sleeping.

4.3 Interrupt Trace view

The Interrupt Trace view provides a time-based graph of interrupts and exceptions and shows their nesting and when the exception is dismissed. This gives a visual representation of time in interrupt service routines and the transitions between them. An example screenshot is shown below:

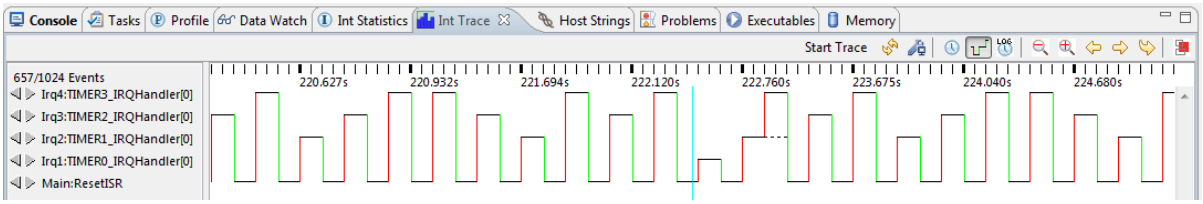


Figure 5 - Interrupt Trace View

Each interrupt handler (interrupt service routine) is listed in the left hand panel, and horizontally is the time axis. In the waveform, entry into an interrupt routine is indicated with a red vertical line and exit is indicated by a green vertical line.

The buttons in the top right corner deserve further explanation and are enlarged in Figure 6.

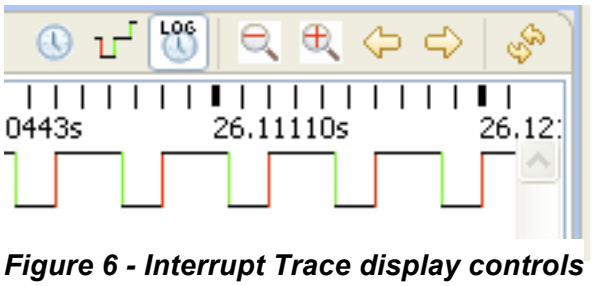


Figure 6 - Interrupt Trace display controls

- Clicking on the clock icon puts the display into a linear time mode.
- The next button puts the display into an event view. In this view time is no longer linear on the x-axis but this can be very useful for showing sparse events that are spread out in time at seemingly unrelated intervals.
- The 'LOG' button puts the display into a log-time view which has the effect of compressing time to show more information with less loss of detail.
- '+' and '-' are used to 'zoom' the display in the time-axis.
- The left and right buttons are used to scroll the time display.
- The refresh button literally shows a capture of another set of samples in the view.

The panel on the left of the Interrupt Trace view lists the interrupt service routines in the application being debugged, as per Figure 7.

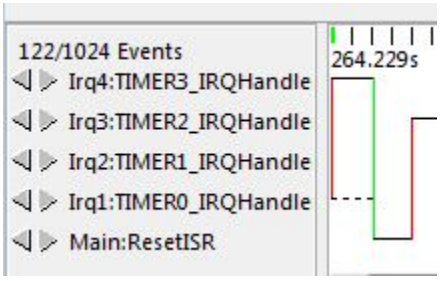


Figure 7 - Interrupt handlers

The arrows to the left of the names of the service routines allow you to move the centre of the waveform display to the next transition on that event. This is particularly useful with sparse events.

5 Data Watch Trace

5.1 Overview

This view provides the ability to monitor (and update) any memory location in real-time, without stopping the processor. In addition up to 4 memory locations can also be traced, allowing all access to be captured. Information that can be collected includes whether data is read or written, the value that is accessed and the PC of the instruction causing the access.

This information can be used to help identify ‘rogue’ memory accesses, monitor and analyse memory accesses, or to profile data accesses.

Real-time memory access is also available, allowing any memory location to be read or written without stopping the processor. This can be useful in real-time applications where stopping the processor is not possible, but you wish to view or modify in-memory parameters. Any number of memory locations may be accessed in this way and modified by simply typing a new value into a cell in the Data Watch Trace view.

5.2 Data Watch view

The view is split into 2 sections – with the “item display” on the left and the trace display on the right, as per Figure 8.

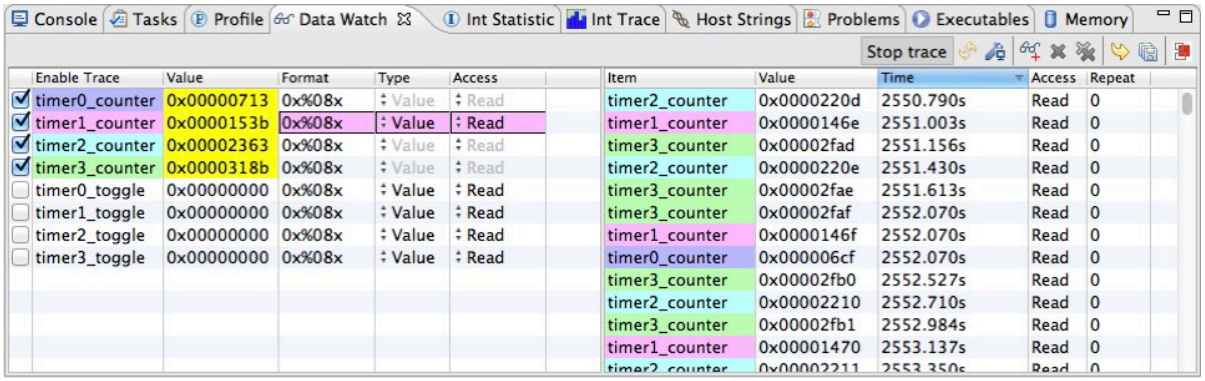


Figure 8 - Data Watch View

Use the Add Data Watch Items button  to display a dialog to allow the memory locations that will be presented to be chosen, as per Figure 9.

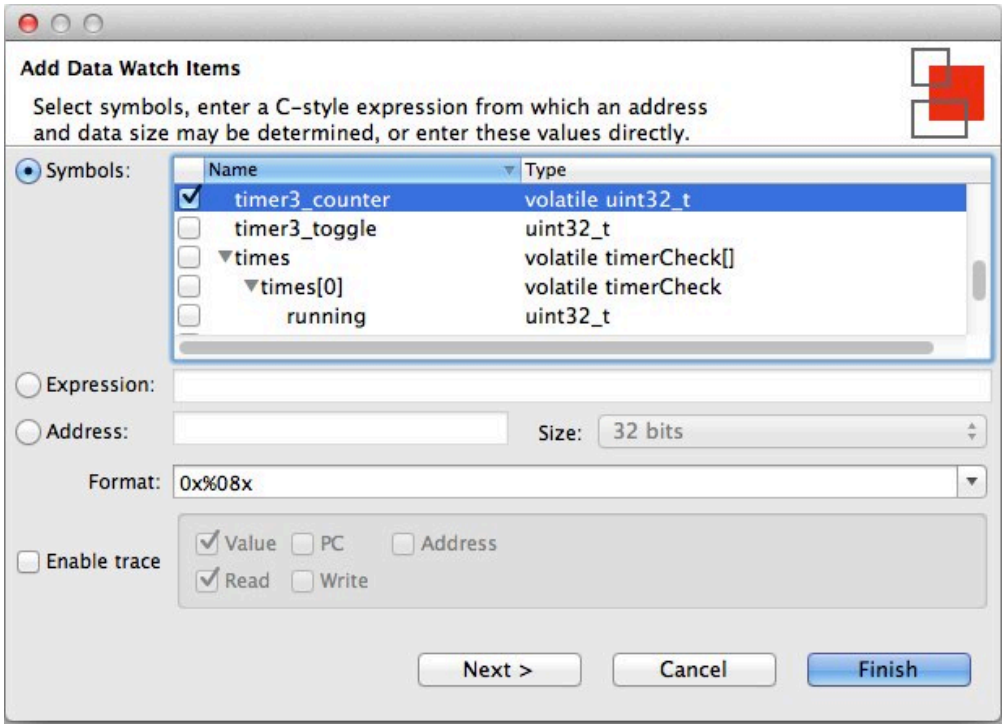


Figure 9 - Add Data Watch Items

These locations may be specified by selecting global variables from a list; by entering a C expression; or by entering an address and data size directly. Trace may be enabled for each new item. If trace is not enabled, the value of the data item will be read from memory.

The format drop down box provides several format strings to choose from for displaying an items value. The format string can be customized in this box, as well as in the item display.

With trace enabled, the options for tracing the item's value, the PC of the instruction accessing the variable, or its address can be set. Additionally, the option to trace reads, writes or both can be set when adding a variable. These settings can be subsequently updated in the item display.

Note that it is not possible to add some kinds of variables when the target is running. Suspending the execution of the target with the  button before adding these variable will overcome this limitation.

Pressing **Finish** adds the current data watch item to the item display and returns to the data watch view. Pressing **Next** adds the current data watch item, and displays the dialog to allow another item to be added.

5.2.1 Item Display

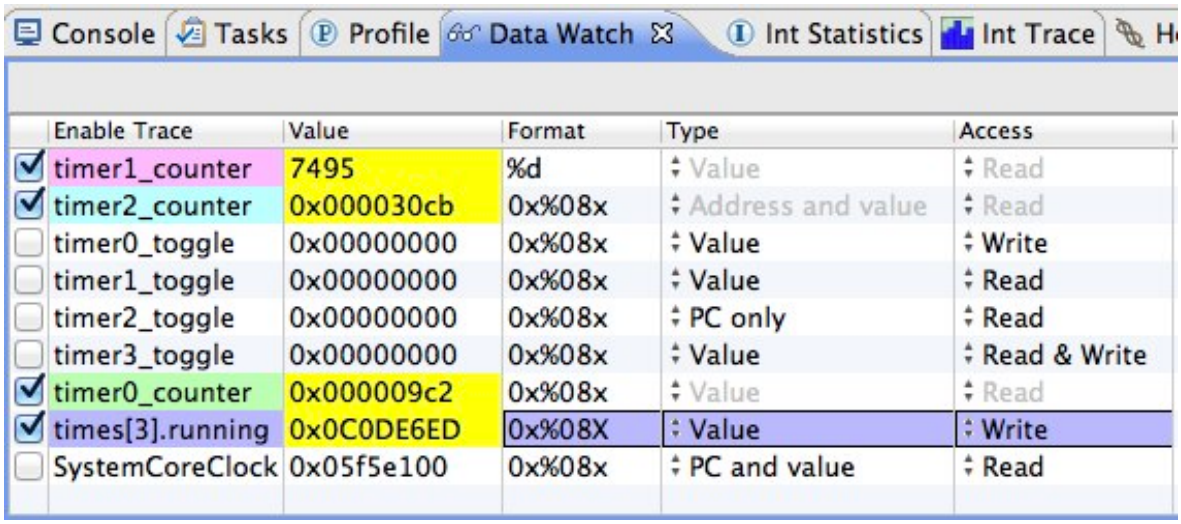


Figure 10 - Data Watch Item Display

As shown in

Figure 10, the item display lists the data watch items that have been added. The following information is presented:

- **Enable Trace** – tracing of this item may be enabled or disabled using the checkbox. A maximum of 4 items may be traced at one time. Each traced item is given a color code, so that it may be picked out easily on the trace display (see Figure 11).
- **Value** – shows the current value of the item and may be edited to write a new value to the target. If the current value has changed since the last update, then it will be shown highlighted (in yellow).
- **Format** – shows the printf-style expression used to format the value and may be edited
- **Type** – shows the trace type, which may be edited while trace is disabled:
    - **Value** – just the trace the value transferred to/from memory
    - **PC only** – just trace the PC of the instruction making the memory access

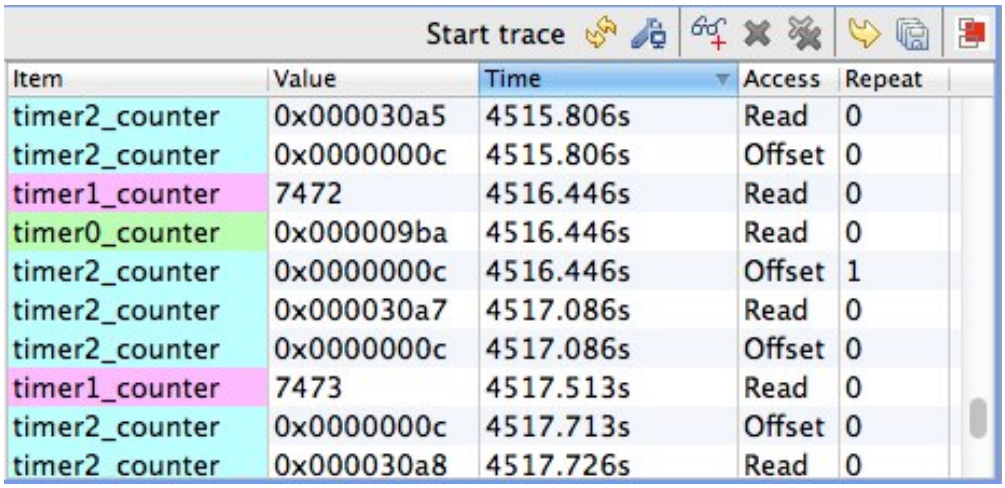
    - **PC and value** – trace the value and the PC
    - **Address** – trace the address of memory accessed
    - **Address and value** – trace the address and value

- **Access** – shows the access type, which may be edited while trace is disabled:
    - **Write** – just trace writes to the memory location
    - **Read** – just trace reads to the memory location
    - **Read & Write** – trace both reads and writes

The variables in the item display persist between debug sessions. They are saved when the session ends and automatically re-added when the trace is started. If a variable is removed from the code between debug sessions the user is alerted that the variable cannot be restored.

5.2.2 Trace Display

The trace display shows the traced values of the memory locations. Each location being traced is given a particular color code, to allow all access to it to be easily picked out.



Item	Value	Time	Access	Repeat
timer2_counter	0x000030a5	4515.806s	Read	0
timer2_counter	0x0000000c	4515.806s	Offset	0
timer1_counter	7472	4516.446s	Read	0
timer0_counter	0x000009ba	4516.446s	Read	0
timer2_counter	0x0000000c	4516.446s	Offset	1
timer2_counter	0x000030a7	4517.086s	Read	0
timer2_counter	0x0000000c	4517.086s	Offset	0
timer1_counter	7473	4517.513s	Read	0
timer2_counter	0x0000000c	4517.713s	Offset	0
timer2_counter	0x000030a8	4517.726s	Read	0

Figure 11 - Data Watch trace display

6 Host Strings (ITM)

6.1 Overview

Host Strings use the Instrumentation Trace Macrocell (ITM) within the Serial Wire Viewer (SWV) functionality provided by Cortex-M3/M4 based systems to display diagnostic messages as your code is running.

The overhead of using Host Strings is much lower than the traditional method of generating diagnostic messages using printf (either through semihosting or retargeted to use a serial port), as the “intelligence” for Host Strings is contained within Red Trace rather than in application code running on the target. The overhead is so low that the instrumentation can remain in your target application code and the generation of the debug messages can simply be globally turned off by default in the ITM trace enable register and then turned on again when the debugger is attached – giving very elegant in-system diagnostic capabilities.

The Host Strings mechanism uses a very low overhead store instruction added to your code or your OS kernel to cause an appropriate diagnostic string to be displayed within the Red Suite IDE. Time stamps are captured along with the instrumentation events.

6.2 Defining Host Strings

Host Strings are defined within a Host Strings file. To create a Host Strings file for a project use the New File wizard by right-clicking on the project within the Project Explorer view and selecting “New->Host Strings File” from the pop-up menu. The Host Strings file must be located immediately under the project folder and must be named **hoststrings.xml**.

Host Strings files are associated with the Host Strings Editor and the new file is opened within this editor automatically when the New File wizard finishes, as per Figure 12

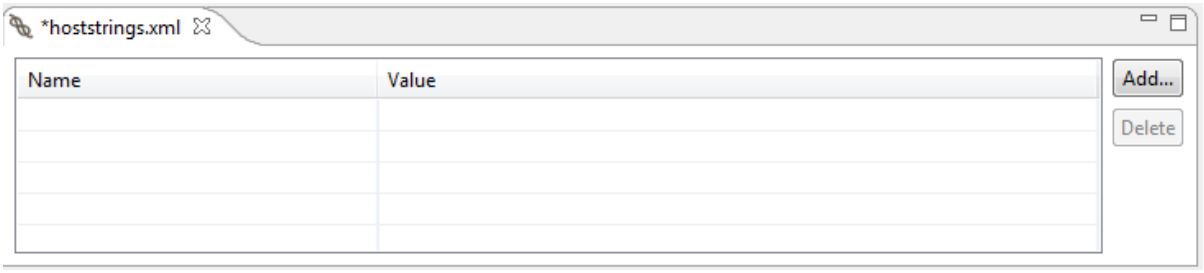


Figure 12 - blank Host Strings Editor

Each host string describes a message to be presented within the Host Strings view within Red Suite, plus the formatting information for a single parameter which is sent from the target for presentation within the message. Click on the “Add...” button within the editor to add a new host string, as per Figure 13.

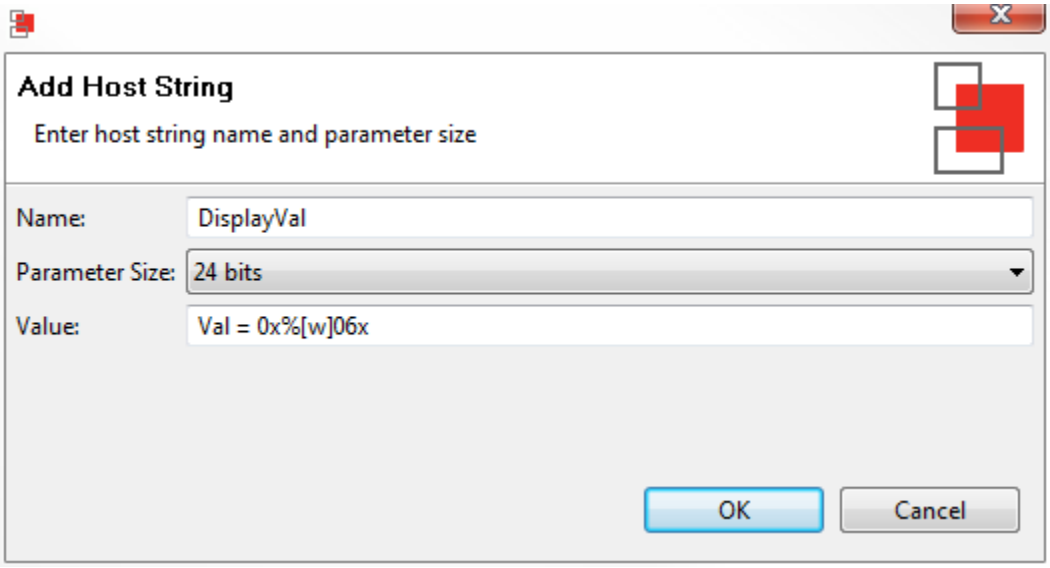


Figure 13 - adding a new Host String

Name

- The name of the Host Strings Macro that will be invoked by the application code running on the target. Note that this name will be automatically prefixed with “HS_” when the hoststrings.h header file is generated by the Host Strings process during project build. Note that the host string name is subject to the naming restrictions for C pre-processor macros.

Parameter Size

- An optional parameter can be passed as part of a call to a Host Strings Macro. This can be of size 8bit, 16bit or 24bit or specified as “None” if no parameter is required. Note that the Host Strings mechanism does not support the passing of full 32 bit quantities as a parameter – such parameters will be truncated to 24 bits by the macro.

Value

- String that will be displayed in the Host Strings view within Red Suite when the Host Strings Macro is executed on the target.
- By default, when you add a new Host String, the value will be set to the name of the Host String, plus the parameter in hexadecimal format. However this value can be edited as required.
- The format of the parameter is **`%[inf]ctrl,`** where
  `inf` = “`b`”, “`h`” or “`w`” for byte, half, word (truncated to 24-bits).
  `ctrl` = “`u`” or “`d`” (for decimal), “`x`” or “`X`” (for hexadecimal), or “`b`” (for binary).
  The `ctrl` specifier can also be prefixed by a number to specify the minimum number of characters to be printed and whether this should be zero-padded (similar to printf).
- Host strings providing incorrect formatting information are marked with a warning triangle within the editor.

Figure 14 shows an example set of Host Strings within the Host Strings Editor….

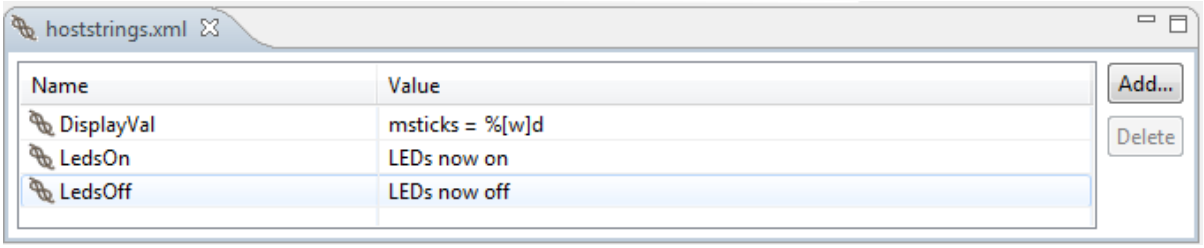


Figure 14 – populated Host Strings Editor

- **DisplayVal** will display the string “msticks = ”, followed by the parameter in decimal format (as specified by the %[w]d).
- **LedsOn** will display the string “LEDs now on”
- **LedsOff** will display the string “LEDs now off”

Note that the value of existing host strings may be edited by clicking on the message string within the editor.

6.3 Building the Host Strings macros

When a Host Strings file is created using the New File wizard, the management of host strings for the associated project is enabled automatically. A header file named **hoststrings.h** is generated using a Host Strings Builder as part of the project build. This behavior may be enabled or disabled for any project by right-clicking on the project within the Project Explorer view and selecting “Managed Host Strings” from the pop-up menu. The header file contains the host string macros for use within your code

6.4 Instrumenting your code

Project code may be instrumented with Host Strings by referencing the Host Strings header file using :

```
#include "../hoststrings.h"
```

Note that the above assumes that the source code for your project is located in a subdirectory immediately below the root directory of your project (where the Host String files are located). If your project is more complex, then it may be better to add "`${ProjDirPath}`" to your project include paths using the menu:

```
Project -> Properties -> C/C++ Build -> Settings ->
      MCU C Compiler -> Includes
```

(repeating for both Debug and Release Build variants) and then simply use:

```
#include "hoststrings.h"
```

Host string messages may then be triggered within your code by adding calls to the pre-processor macros defined within this header file. For example:

```
HS_DisplayVal(msticks);
```

6.5 Host Strings view

The Host Strings view presents the message associated with a host string each time the associated host string macro is called within your code. The message includes the formatted value of the parameter passed into the macro.

Figure 15 shows example Host Strings output, as generated by running the "`RDB1768cmsis2_HostStringsDemo`" project, which can be found in the RDB1768cmsis2.zip example bundle provided with Red Suite 4.

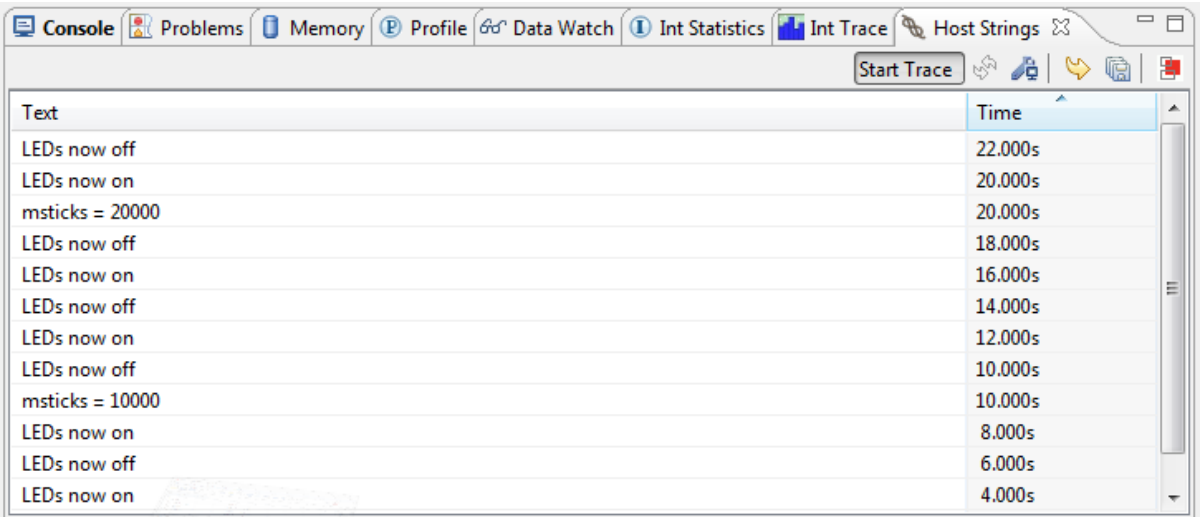


Figure 15 - Host Strings output