

# **Training Cortex-M Tracing**

Release 09.2024



TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Training	
Training Arm ETM	
Training Cortex-M Tracing	1
History	4
Cortex-M Trace	4
Connectors	7
Basic Trace Configuration	8
Trace Buffer Management	10
MTB Program Flow Trace	13
ETM Program Flow Trace	14
ETM Configuration	14
Trace Capture	16
ETM Stream Mode	17
Displaying the Results	18
Trace Searching	21
Trace Filtering	24
Tracing Certain Events	25
Tracing Between Two Points	27
Graphical Navigation	28
Analyzing the Results	30
Function Runtime	31
Distribution	36
Duration A to B	38
Distance Trace Records	39
Trace and Groups	40
Grouping by Modules	41
Grouping by Address Range	43
Timing	45
Trace Based Code Coverage	48
Trace Based Debugging	49
Off-line Analysis	53
Data Watchpoint and Trace Unit	55
PC Sampler	56
Data Trace	58

Task/Thread switch Tracing	61
Interrupt Trace	63
ETM Trigger	65
Instrumentation Trace Macrocell	66
Software Generated Trace	67
Using ITM for printf style output	70
Time Stamping	72
Stream Mode	73
Pipe Mode	74

Version 05-Oct-2024

# History

29-Sep-23 Table of optional trace blocks for member of the Cortex-M family updated.

06-Feb-18 Initial version of the manual.

## **Cortex-M Trace**

This document should be read in conjunction with "Training Arm CoreSight ETM Tracing" (training\_arm\_etm.pdf) which shows how to drive and interrogate the TRACE32 trace subsystems. Many of the views and types of data collected will be similar to those shown in that document. It is assumed that the reader will be familiar with the basic trace concepts, allowing this document to focus on Cortex-M specific features.

Many of the debug components of Cortex-M based designs are optional IP blocks that may or may not have been included in the design of the chip being debugged. Always check the chosen chips' documentation. Trace components may include:

## Micro Trace Buffer (MTB)

This allows program flow data to be saved to internal SRAM. The data can be read via the JTAG or Serial Wire Debug (SWD) interface. The amount of SRAM used and the location of the buffer are software configurable. The size of the SRAM buffer limits the amount of program flow trace that can be captured. Trace writes to SRAM take priority over system writes to the AHB-Lite interface, with one or more wait states being inserted into AHB-Lite accesses if a trace write occurs simultaneously. This may affect the run-time performance of the application being traced under high bus load situations. This optional IP block may not be present in all devices.

## Embedded Trace Macrocell (ETM)

This unit allows program flow data to be fed to the TPIU. There, it will be formatted for eventual delivery to off-chip trace tools. The amount of trace data that can be captured is equal to the size of the buffer in the external tools or, in the case of streaming, the size of the host PCs hard drive. This feature is an optional IP block and may not be present in all devices.

## Instrumentation Trace Macrocell (ITM)

This unit provides three main features: software instrumented trace; integration of trace packets from the DWT into the trace stream; timestamp generation for ITM and DWT trace packets. It is an optional feature and may not be present in all devices.

## Data Watchpoint and Trace (DWT)

This optional component provides a number of trace like features: Interrupt trace; Data Trace; ETM Trigger; PC Sampler and Trigger. Each of these will be covered in more detail in a later section of this document. The output from the DWT is fed into the ITM for formatting and inclusion into the trace stream.

## Trace Port Interface Unit (TPIU)

This block is required to route the trace to its final destination: off-chip or on-chip. The TPIU can aggregate trace from multiple sources into a single stream, allowing for tracing of multi-core designs. Two off-chip modes are supported: Serial Wire Viewer (SWV) or Serial Wire Output (SWO); and parallel trace. SWV or SWO is a single bit wide trace port designed for low speed trace – usually from the ITM/DWT – where this option is selected the TPIU will silently 'drop' ETM packets and not transmit them. Where parallel trace is selected all packets from all sources will be transmitted. The parallel port can be 1 to 4 bits wide and may be clocked independently from the CPU.

The table below outlines the optional trace blocks that are available for members of the Cortex-M family.

Armv6-M	
Cortex-M0	No trace options.
Cortex-M0+	Optional: MTB Optional: Limited DWT may be included that supports data breakpoints and PCSampler.
Cortex-M1	No trace options.
Armv7-M	
Cortex-M3	Optional: ITM, DWT, ETMv3.
Cortex-M4	Optional: ITM, DWT, ETMv3.
Cortex-M7	Optional: ITM, DWT, ETMv4 (optional with full data trace).
Armv8-M	
Cortex-M23	Optional: DWT. Optional: ETMv3 or MTB. The two are mutually exclusive.
Cortex-M33	Optional: ITM, DWT, ETMv4, MTB. Designs can support both MTB and ETM in the same device.
Cortex-M35P	Optional: ITM, DWT, ETMv4, MTB. Designs can support both MTB and ETM in the same device.
Cortex-M55	Optional: ITM, DWT, ETMv4.
Cortex-M85	Optional: ITM, DWT, ETMv4.
STAR	Optional: ITM, DWT, ETMv4.

A graphical overview of the Cortex-M (>= M23) debug components can be seen in the block diagram below.



This is the block diagram for Cortex-M0+ devices.



## Connectors

Two connectors are commonly used with Cortex-M based systems. The diagram below shows the pinouts and which features are supported by each.



It is imperative to ensure that your development board has the correct connector for the debug features you wish to use. Other connectors are possible.

The trace system is configured using the **Trace.state** window, which can be accessed via the command line or from the **Trace** menu.

Tra	ce	Perf	Cov	STM32F10x
	Co	nfigura	ation	
B	CT	S Settir	ngs	
	ETI	VI Setti	ngs	
	Trig	gger Di	alog	
	List	:		•
<del></del>	Tin	ning		•
N	Ch	art		•
	ITN	1		+
B	TPI	U setti	ngs	
P	Sav	e trace	data .	
2	Loa	d refe	rence o	lata
	Res	et		

🔑 B::Trace.state			- • ×
METHOD			
○ Analyzer	nalyzer Onchip OART	OLOGGER OSNOOPer OFI OHAnalyzer OIntegrator OP	DX OLA robe OIProbe
state	used ACCES	SS TDelay	
○ DISable	auto	~ <b>0</b> .	TrOnchip
OFF	0.	0% ~	🔑 TPIU
○ Arm	- SIZE CLOCH	к —	🔑 ITM
⊖ trigger	268435456.	TSELect	🛞 BMC
⊖ break		BusA	
○ SPY	Mode THres	hold	
	Fifo 1.65	∽ ⊤Out ───	
commands —	○ Stack	BusA	
RESet	○ Leash Tes	tFocus	
⊗ Init	◯ STREAM	ItoFocus	
SnapShot	○ PIPE		
📰 List	ORTS		
AutoArm			
AutoInit			
SelfArm			

Most of the features in this window are explained under "**ETM Setup**" in Training Arm CoreSight ETM Tracing, page 6 (training\_arm\_etm.pdf). Here, we will concentrate on those that affect Cortex-M based systems.

The generated trace data can be stored on-chip in a memory buffer or it can be streamed off-chip to a set of trace tools. On-chip trace storage is often referred to as MTB or Embedded Trace Buffer (ETB). Whilst off-chip trace storage is often referred to as ETM. These are historical and largely irrelevant as the same trace data can be routed to either storage medium.

For off-chip trace (ETM, ITM), select **Trace.METHOD CAnalayzer** or **Trace.METHOD Analyzer**. If the tools do not support one of these modes, it will be grayed out.

For on-chip trace (MTB), select Trace.METHOD Onchip.

For off-chip trace, the size will be automatically filled based upon the amount of storage for trace data there is in the chip or the tools. This can be over-ridden for off-chip trace if a smaller amount of data is required, simply type the new value into the field or use the command **Trace.SIZE** *<n>*. For on-chip trace (MTB), the size must be specified.

The Mode setting affects the way that the trace buffer is managed.

FIFO	The trace data is stored in the buffer memory in the TRACE32 trace hardware. When the buffer fills, earlier records are over-written with the new data. When the trace sampling stops, the maximum amount of trace history is available.
Stack	The trace data is stored in the buffer memory in the TRACE32 trace hardware. When the buffer is almost full, sampling stops but the target is left running.
Leash	The trace data is stored in the buffer memory in the TRACE32 trace hardware. When the buffer is almost full, trace sampling stops and the target is halted.
STREAM	The buffer memory in the TRACE32 trace hardware is used as a large FIFO and the trace data is streamed to a file on the host system's hard drive. The amount of trace captured is limited by either the size of the hard disk partition, a user specified amount or (2^64)-1 frames.
PIPE	The buffer memory in the TRACE32 trace hardware is used as a large FIFO and the trace data is streamed to an application which reads a pipe or FIFO on the host system. The trace stream must be processed in real-time; no storage is performed. The trace stream can be effectively unbounded.

The **Trace.CLOCK** needs to be set to the core clock frequency.

The **Trace.TDelay** setting is used to position a trigger point somewhere other than the start (STACK of LEASH mode) or end (FIFO mode) of the buffer. It takes either a percentage or a number of records to capture after the trigger condition is reached. The trigger condition is a breakpoint of type TraceTrigger. This does not stop the core but merely triggers the trace system; of course the action of the trigger may also be to stop the core. More information can be found here ("**Trace Buffer Management**", page 10.

The **Trace.TSELect** and **Trace.TOut** settings allow for an external trigger to be used. This can be the BusA signal which is present on the PODBUS as well as the Trigger pin on the outside the TRACE32 debug hardware. From left to right: Trigger pin, USB connector, (optional) Ethernet connector, power connector.



The Trigger pin is controlled via the Trigger Bus Window (**TrBus**). Selecting **<trace>.TSELect BusA** will use the Trigger pin as the trace trigger event. Selecting **<trace>.TOut BusA** will trigger the pin when the trace system triggers or stops tracing.

The trace buffer acts as storage for all of the information generated by the core. When the target stops, the records in the trace buffer allow the user to look back into the past. It is impossible to look back beyond the start of the trace buffer.



If the sampling is halted before the buffer fills a smaller amount of data is collected.

Once the buffer has filled, the time window moves forwards to accommodate new samples. When the trigger occurs, the trace stops sampling. If more than a buffer's worth of data had been generated some of it would have been lost.



To capture more data:

- Use tools with a larger trace buffer. Eventually, this option will become exhausted.
- Stream the trace data to your local hard drive. More information can be found "ETM Stream Mode", page 17
- Stream the trace data to a local application for real-time processing. The trace data is not stored by TRACE32. More information can be found here: "Pipe Mode", page 74.
- Use the on chip features to filter the generated trace data to view only events of interest. More information can be found here: "Trace Filtering", page 24.

The DWT can generate a Trigger packet which causes the trace tools to trigger on an event. The **Trace.TDelay** option controls how the tools will react. It holds a portion of the trace buffer in reserve to be filled up only after the trigger event occurs. Setting the value to 50% will place the trigger event in the middle of the trace buffer, giving the user an equal amount of trace buffer dedicated to events before the trigger and events after the trigger.



Setting the point to 10% will 'reserve' 10% of the buffer to store events that occurred after the trigger.



When the trigger event occurs, the trace state changes to "trigger", as seen in the image below.

🔑 B::Trace.state				- • ×
METHOD				
○ Analyzer	nalyzer 🔿 Onchip	○ ART ○ LOGG ○ HAnaly	ER OSNOOPer OF yzer OIntegrator OP	Probe OLA
- state	used	ACCESS	TDelay	
○ DISable		auto 🗸 🗸	120795954.	TrOnchip
OFF	26018096.		90% ~	🔑 TPIU
○Arm	- SIZE	CLOCK		🔑 ETM
<ul> <li>trigger</li> </ul>	134217728.	72.0MHz	TSELect	🖉 ITM
⊖ break			BusA	🛞 BMC
○ SPY	Mode	THreshold		
	Fifo	1.65 ~	- TOut	
commands	◯ Stack		BusA	
RESet	◯ Leash	TestFocus		
🛇 Init	○ STREAM	💥 AutoFocus		
SnapShot				
🔣 List	ORTS			
AutoArm				
AutoInit				
SelfArm				

When the amount of trace that corresponds to the **Trace.TDelay** setting has been captured the trace will stop sampling and the mode will change to break.

The Trigger point can be located in the **Trace.List** window and adding the Time.TRIGGER column shows all trace events timed relative to the trigger event: events after have a positive time index; events before have a negative time index.

B::Trace.Lis	st def ti.	trigger									
🔑 Setup	<b>∩</b> Go	to 👘 Find	Chart	📕 Profile	MIPS	More	Less				
recor	rd  rur	n address	cycle	data	symbol			ti	.back	ti.trigger	
2	52	}							0.160us	-0.440us	^
12	22	if(Ste	p != HALT)						0.033us	-0.407us	~
12	25	۲ //Ch Adju	ange the f stPulse(va	requency ls[Zone].	of the pu base + ((	lse (adc_val -	vals[Zone	e].offs	0.095us	-0.311us	); ^
		/// Arguments // Arguments // Returns // Notes //////////// static void	: INT16 : None : Chan <u>c</u> //////// AdjustPuls	50 - the r ges the ti ///////// ;e(INT160	new value ming valu ///////// newval)	to put in ues for Ti	the ARR r mer1 drivi	egister ing the m	10tor ////////////////////////////////////		
14	46	{							0.371us	0.060us	
		IIM_IImeBa	seinitiype	PDet IIM_	InmeBases	structure;					
14	49	if(newval	> 0x2000)	//Make	we son't	break the	timer		0.024us	0.084us	
19	51	TIM_Time	BaseStruct	:Init <mark>(&amp;</mark> TIM	I_TimeBase	Structure	);		0.038us	0.122us	
	<	* Function N * Descriptio * Input	ame : TIN n : Fil : - 1	I_TimeBase  ]s each T IM_TimeBa	StructIni IM_TimeBa IseInitStr	t seInitStr uct : poi	uct member nter to a	with it TIM_Time	s defaul BaseInit	t value. TypeDef	<b>*</b>

The Micro Trace Buffer (MTB) provides basic program flow trace capabilities for cores with limited resources. It is not designed to compete with ETM or PTM. The trace data is stored in a user configurable area of RAM at runtime. External debug tools can be used to start or stop the trace. The size and location of the RAM storage is configurable in software, allowing the resources to be reclaimed from a development build when they are no longer needed. The MTB can be programmed by the debug tools to cause the processor to enter a halt state when the buffer becomes full.

TRACE32 treats the MTB as an on chip trace buffer. The On Chip family of commands apply and **Trace.METHOD Onchip** should be set.

TRACE32 needs to know the base address and size of the buffer. This is done with the commands:

- Onchip.TBADDRESS <address>
- Trace.SIZE <size>

The MTB is usually placed in a separate memory region in the linker control file. This makes it easy for TRACE32 to locate the base address with something like this.

```
Trace.TBADDRESS ADDRESS.OFFSET(sYmbol.SECADDRESS(.mtb))
Trace.SIZE 64.
```

A list of all sections can be obtained with the command **sYmbol.List.SECtion** and my look like this.

address         path/section         acc           D:1FFFE0001FFFE3FF         \sieve_ram_thumb_v6m\.isr_vector         R           P:1FFFE400200008FF         \sieve_ram_thumb_v6m\.text         R-X           D:200008F020000A37         \sieve_ram_thumb_v6m\.rodata         R	init physical
D:1FFFE0001FFFE3FF \\sieve_ram_thumb_v6m\.isr_vector R P:1FFFE400200008EF \\sieve_ram_thumb_v6m\.text R-X D:200008F020000A37 \\sieve_ram_thumb_v6m\.rodata R	L- 🔨
D:20000A36-2000049F \\sieve_ram_thumb_v6m\.data RW- D:20000A402000140B \\sieve_ram_thumb_v6m\.bss RW- D:2000140C2000160F \\sieve_ram_thumb_v6m\.stack RW- D:200016402000167F \\sieve_ram_thumb_v6m\.mtb RW-	L- L-    

Here, the base address of the .mtb section can be clearly seen.

There is no timing information available for the program flow trace; no statistical or analytical operations may be performed.

When most people think of trace, what they mean is ETM. This is a program flow trace that is generated by the Cortex-M core. This section should be read in conjunction with ."Training Arm CoreSight ETM Tracing" (training\_arm\_etm.pdf) which provides a more general overview of ETM and the kinds of analysis that can be performed on the captured data.

## **ETM Configuration**

ETM trace data requires a fully functional TPIU to be able to pass the data to the final destination (off-chip or on-chip). The ETM features are configured in the ETM window. It can be accessed from the Trace menu, from the Trace Configuration window or by using the command **ETM**.



#### It looks like this.



Next, the TPIU may need to be configured. This can be accessed via:

- Selecting TPIU settings from the Trace menu
- Clicking the TPIU button in the Trace.state window
- Using the command TPIU.state

The **TPIU.PortSize** value can be changed to 1bit wide, 2bits wide, 4bits wide or Serial Wire Viewer (SWV). The default value of **TPIU.SyncPeriod** is every 1024 packets. This can be changed by the user and configures how frequently synchronisation packets will be emitted by the target.

B::TPIU.state		
tpiu	PortSize 4 ~ PortMode Continuous ~	SyncPeriod —
commands RESet CLEAR Register Drace	SWVPrescaler 1.	

NOTE:	<ul> <li>Some devices lack enough pins to bring out all of the possible signals on the chip. When this occurs, the user will need to ensure that the correct pin muxing is enabled to route the TPIU signals from the chip to the debug header on the board. Check your chipset documentation! Other things to check are: <ul> <li>Your application does not use a set of chip features that prevent the correct multiplexing of the trace port pins.</li> <li>Any chosen RTOS does not re-multiplex the trace port pins as part of its setup/boot procedures.</li> <li>Any third party libraries or drivers do not change the trace port pin multi-</li> </ul> </li> </ul>
	plexing.
	• Some devices multiplex the trace port pins with GPIO. Ensure the pin direction is correctly set. Also, check the maximum possible speed of the GPIO pins used; the trace will not emit data faster than this.

## To capture Program Flow trace using ETM:

- 1. Set **ETM.ON**. Via command or GUI.
- 2. Set Trace.CLOCK.
- Set either Trace.Method Analyzer or Trace.Method CAnalyzer. Unsupported variants will be grayed out.
- 4. Set Trace.OFF.
- 5. Set **Trace.AutoArm ON**. This will cause the sampling to start and stop in synchronisation with the target.
- 6. Set **Trace.AutoInit ON**. This will clear the existing buffer before sampling new data.
- 7. Set the desired **Trace.Mode** (Stack, FIFO or Leash). More information can be found here "**Trace Buffer Management**", page 10.
- 8. Start the target running. It can be halted manually or via a breakpoint.

Details on how to display the results can be found here "**Displaying the Results**", page 18 and details on analyzing the trace data can be found here "**Analyzing the Results**", page 30.

To capture trace data for extended periods of time may require storing more data than can be held in the buffer in the tools. For example, the µTrace's (MicroTrace) 128Mbyte buffer holds around 20-30seconds of trace of a bare metal demo program running at 66MHz on a Cortex-M3.

Trace data can be streamed to a local file system for analysis.

This requires a 64bit host OS and a 64bit version of TRACE32. By default, the stream file is stored to the temporary directory specified in the configuration file (usually ~~/config.t32). This can be found with the command:

```
PRINT OS.PresentTemporaryDirectory()
```

A new stream file can be set with Trace.STREAMFILE <file>

Set the Trace mode to **STREAM** and the used bar switches from white to yellow as seen in the picture, below left.



ℬ B::Trace		
METHOD		
○ Analyzer	nalyzer 🔿 Onchip	OART
state	used	ACCE
○ DISable		auto
OFF	1851347600.	
Arm	- SIZE	
○ trigger		66.0
Obreak		
⊖ SPY	Mode	THree
	○ Fifo	1.65
commands	◯ Stack	
RESet	OLeash	Tes
🛇 Init	STREAM	💥 Ai
SnapShot		

The used bar is now used to show the fill level of the tools' internal buffer. The buffer is now used as a large FIFO to smooth out any peaks in the trace flow before streaming to the host PC. The number underneath is the number of trace frames captured. The image (above, right) shows the state after around 5 and a half minutes of trace capture.

When TRACE32 is closed any stream files are automatically deleted from the host file system. Captured trace data can be saved using **CAnalyzer.STREAMSAVE** *<file>* so that it remains on the local file system after TRACE32 has closed. When TRACE32 is re-started, the saved data can be re-loaded with **CAnalyzer.STREAMLOAD** *<file>*. The data is saved in a raw format and it is not expected that users will be able to interpret this. More information about working with loaded trace files can be found here "**Off-line Analysis**", page 53.

Once the trace data has been captured a list of all events can be obtained by using the **Trace.List** command or by clicking the List button in the Trace configuration window. The results will look like this.

🔢 B::Trace.List											×	
🔑 Setup 👔	🕽 Got	to 🧃	Find	Chart	📕 Profile	MIPS	More	Less				
record	run	addre	ss	cycle	data	symbol			t	i.back		
263		e { novs str b	lse // Cloc SPI1_CL r5,0x r6,#0 r6,[r 0x800	<pre>:k change K_XOR(); (800597C) x20 ·5] 058F8</pre>	;	r5,??DataT ??SPI1_Tra	able6_4	15			< > <	
265 -0000000019		tst r beq 0 T:08		Data ask & _Dat 5904 GFC ptrace	a)	; _Mask,_Data ; ??SPI1_TranserByte_16 drv_spi1\SPI1_TranserByte+0xCC						
267	267   ] ( mo st ↓ b	ldr movs str ↓ b		SPI1_ r5,0x r6,#0 r6,[r 0x800	MOSI_H(); (8005978 )x80 (5] (590A	;	r5,??DataT ??SPI1_Tra	able6_3	L7			
273 -000000018	BRK	ldr ldrb ldr	} for(vol r5,0» r5,[r r6,0» T:080059	atile Int 8005970 5] 8005974 00E	32U i = ; ;	_ <b>SPI_Dev[_</b> r5,??DataT r6,??DataT —drv_sp	SPI1_Devic able6_1 able6_2 i1\SPI1_Tr	<b>ce].Delay;</b> ranserByte	i; i)	); -0.000us	~	
	<u> </u>									2	1.1	

The columns shown are the default values and others can be added. See Trace.List for more information.

Along the top of the window is a series of buttons that give access to other features.

Setup Opens the Trace configuration window.

Goto Opens the Goto window which allows the user to jump to points of interest in the trace listing. For more details on filtering "Trace Filtering", page 24 and searching "Trace Searching", page 21.

Find Opens the search dialog and allows the user to search for events within the trace buffer.

Chart

Displays a graphical view of functions on the y-axis and time on the x-axis. Bear in mind that: trace captured in FIFO mode will all have a negative time index; if a Trigger has been used trace captured before the trigger event will have a negative time index.



More information on driving the Chart windows can be found here "Graphical Navigation", page 28.

Profile

Shows a graphical representation of cpu usage by function over time. Small time slices are used to calculate the percentage of cpu time used by each function that occurs during the slice. See "Graphical Navigation", page 28 for more information on interacting with this window. Each function is represented by a colored block. Clicking on the colored block opens a pop-up which displays more information about the item.



## MIPS

Provides a graphical representation of how many instructions per second (over a given time slice) were used executing each function. "Graphical Navigation", page 28.



Clicking the **More** or **Less** buttons will add or remove certain items from the display. In four steps, it moves between showing all CPU cycles, including dummies, to just showing the HLL code.

The Trace Goto dialog looks like this and can be accessed by clicking the "Goto" button on any trace view window.

🔒 Trace Goto			
- Record / Time ,	Goto		
Previous	First	Trigger	Zero
Next	Last	Ref	Track Cancel

It provides a convenient way of jumping to certain points within the Trace.List window.

A user entered record number or time index can be entered. If a bookmark has been created this can be located here too. More information about **BookMarks** can be found by following this link.

Jump to the First captured trace record.

Jump to the **Trigger** event. More information on the trigger can be found here: "**Trace Buffer Management**", page 10.

Jump to time index of **zero**. This is the first event in Stack or Leash mode and the last event in FIFO mode. Right-clicking any trace window allows a user created zero point to be defined. This will be used as the zero point in all future calculations. The command **Trace.ZERO** can also be used to set the marker.

Jump to the Last captured trace event.

Jump to a user created reference point. Right-clicking any trace display window will allow a reference point to be manually created. The command **Trace.REF** may also be used to set the reference marker.

Selecting **Track** will jump to the last position that the user placed a cursor at in any trace window.

Any windows that are opened with the /Track option will also jump to the selected point.

It is possible to search the contents of the trace using the Find window which can be accessed by clicking the Find... button on the **Trace.List** window and looks like this.

🏥 Trace Find			_	
○ Expert	Oycle	⊖ Group	○ Changes ○ Signal	⊖ Up ● Down
- address / ex	pression —			
			~	🔒 🗌 HLL
Cycle	→ Data			~
Find Next	Find First Find	d Here Find	All Clear	Cancel

For example, to find all occurrences of function iTIM4ISR, you would enter:

🎒 Trace Find			_	
○ Expert	Oycle	⊖ Group	○ Changes ○ Signal	⊖ Up ● Down
– address / exp	pression			
iTIM4ISR			~	1 🗌 HLL
Cycle				~
Find Next	Find First Fin	d Here Find	All Clear	Cancel

And click the Find All button. A list of all entries to the function *iTIM4ISR* are shown. The *ti.back* column shows the time between calls to this function. More information on trace timing can be found here: "Timing", page 45.

🛐 B::Trace.FindA	AII , A	ddress iTIM4ISR				
188	rur	n address	cycle	data	symbol	ti.back
-0022301590		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	~
-0022199878	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	22.336ms
-0022083082	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021966687	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms 🎽
-0021849260	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms \land
-0021733745	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021617738	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021500697	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021383706	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021266625		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021151348		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0021034636	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0020918935	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0020719046	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0020490852	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0020262717	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0020146369	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0020029482	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019913733	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019797819	1 1	T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019680910		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019564511		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019447039		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019331444		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms
-0019215272		T:08007DDC	ptrace		\\DDC2016\app_adc\iTIM4ISR	25.486ms 🗸
	<					> . <u>.</u> :

Clicking any item in this window will cause the **Trace.List** window to jump to the same point within the trace buffer.

Cortex-M ETM trace is program flow trace only; there are no data trace items. These can be injected via the DWT/ITM. More information can be found here: "Data Watchpoint and Trace Unit", page 55. To search for all task switches (writes to TASK.CONFIG(magic)) use:

🏥 Trace Find			_	
○ Expert	Ocycle	⊖ Group	○ Changes ○ Signal	⊖ Up
– address / ex	pression			
task.config(I	magic)		~	🔒 🗌 HLL
Cycle	Data			
Write	~   [			$\sim$
Find Next	Find First Fin	d Here <b>Find</b>	I All Clear	Cancel

Then click **Find All**. A list of task switches will be displayed. The values in the **ti.back** column show how long each task or thread was running for before it was switched.

🛐 B::Trace.FindA	AII , A	ddress task.config(magic	) CYcle Write	2			×
286	run	n address	cycle	data	symbol	ti.back	
-0000026356		D:2000265C	wr-long	00000000	\\DDC2016\Global\OSTCBCur		~
-0000026343	11	D:2000265C	wr-long	00000000	\\DDC2016\Global\OSTCBCur	1.243ms	
-0000026144	11	D:2000265C	wr-long	20000BA8	\\DDC2016\Global\OSTCBCur	5.788ms	
-0000026037	11	D:2000265C	wr-long	20000C04	\\DDC2016\Global\OSTCBCur	31.180us	~
-0000025920	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	19.960us	^
-0000025896	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.427ms	
-0000025736	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.240us	_
-0000025713	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.469ms	
-0000025547	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.220us	
-0000025523	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.469ms	
-0000025357	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.780us	
-0000025334	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.468ms	
-0000025175	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.160us	
-0000025152	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.469ms	
-0000024988	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.220us	
-0000024964	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.469ms	
-0000024801	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.160us	
-0000024778	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.469ms	
-0000024616	11	D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.680us	
-0000024593	11	D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.468ms	
-0000024434		D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.280us	
-0000024410		D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.469ms	
-0000024248		D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.780us	
-0000024216		D:2000265C	wr-long	20000B4C	\\DDC2016\Global\OSTCBCur	25.468ms	
-0000024053		D:2000265C	wr-long	20000C60	\\DDC2016\Global\OSTCBCur	17.160us	×
	< .					>	

More information on tracing task or thread switches can be found here: "Task/Thread switch Tracing", page 61.

The Find window is a wrapper for the command **Trace.FindAll**. Follow the link for a detailed explanation.

Events can be filtered by using the DWT. The DWT filtering takes place on the chip and is non-intrusive to the target's runtime performance. Filtering to capture only those events of interest can relieve pressure on the internal FIFO in the chip and extend the length of time for which data can be captured.

The DWT comparators can be conveniently programmed by using the TRACE32 breakpoint features. Several event filter types are available and can be assigned by using the **Break.Set** command or dialog.

🔯 B::Break.Set			– 🗆 X
- address / expre	ssion		✓ 1 □ HLL
• type • Program O ReadWrite O Read O Write O default	options EXclude NOMARK DATA	Temporary DISable DISableHIT	implementation auto action stop Spot Alpha beto
Ok	Add	Delete	Charly Delta Echo TraceEnable TraceData TraceON TraceOFF TraceTrigger

The last five entries on the drop-down menu will affect how trace is generated.

TraceEnable	When a breakpoint of this type is set, a match will generate a trace packet for this event only. All other ETM trace generation is suspended.
TraceData	A match to this event will cause a DWT data trace packet to be emitted.
TraceON	ETM trace will be switched on when this event matches.
TraceOFF	ETM trace will no longer be generated when this event matches.
TraceTrigger	When this event matches, a Trigger packet will be emitted.

By using a matching pair of TraceON and TraceOFF markers it is possible to restrict the trace to only an area of interest and nothing else. This will extend the length of time for which meaningful data can be captured at the expense of not capturing all events.

The Trigger packet causes the trace tools to trigger on an event. The **Trace.TDelay** option in the Trace configuration window controls how the tools will react. It holds a portion of the trace buffer in reserve to be filled up only after the trigger event occurs. Setting the value to 50% will place the trigger event in the middle of the trace buffer, giving the user an equal amount of trace buffer dedicated to events before the trigger and events after the trigger.

For example, the time taken between an Interrupt Service Routine executing and the task designed to respond to that interrupt being scheduled.

When using Filters like this it is recommended to switch the timing mode to be cycle accurate (ETM.TImeMode CycleAccurate. Some early Cortex-M cores do not support a cycle accurate timing model. In these cases, set Trace.PortFilter ON.

Set a breakpoint on the ISR and the wake-up part of the task of type **/TraceEnable**. The list in the picture below shows an example of marking two events.

😫 B::Break.List								-
X Delete All O Disable Al	I 🖲 Enable All	⊗ Init	م∂ Impl	Sto	re	😤 Load	🙆 Set	
address t	ypes	impl	action					
T:08007D2A P T:08007DDE P	rogram rogram	ONCHIP ONCHIP	TraceEna TraceEna	ble ble	tADC iTIN	C\84 M4ISR\3		^
								$\sim$
	C						>	

Listing the contents of the trace buffer with Trace.List shows only the filtered events.

📰 B::Trace.List		
🔑 Setup 🖡	🕽 Goto 🎒 Find 🕂 Chart 🛛 Profile 🎴 MIPS 🔷 More 🗶 Less	
record	run address cycle data symbol	ti.back
-0000000697	TRACE ENABLE T:08007D2C ptrace \\DDC2016\app_adc\tADC+0x11C ldr r0,0x8007DF8 ; r0,??DataTable1_1 TRACE ENABLE	10.240us
-000000688	T:08007DDE ptrace \\DDC2016\app_adc\iTIM4ISR+0x2 INT8U err;	25.486ms ^
171	err = OSSemPost(sTIM4);  dr r0,0x8007E00 ; r0,??DataTable1_3  TRACE FNARE	
-0000000678	T:08007D2C ptrace         \\DDC2016\app_adc\tADC+0x11C           ldr         r0,0x8007DF8         ; r0,??DataTable1_1	25.476ms
-0000000671	T:08007DDE ptrace \\DDC2016\app_adc\iTIM4ISR+0x2 INT8U err;	0.180us
171	err = OSSemPost(sTIM4);  dr r0,0x8007E00 ; r0,??DataTable1_3	
-0000000663	T:08007D2C ptrace         \\DDC2016\app_adc\tADC+0x11C           ldr         r0,0x8007DF8         ; r0,??DataTable1_1	10.220us
-000000654	T:08007DDE ptrace \\DDC2016\app_adc\iTIM4ISR+0x2	25.486ms
	INT8U err;	<b>∀</b> >

To show a histogram of distances between two points use the command **Trace.STATistic.AddressDURation**. For example:

To measure the time between the last instruction of the ISR and the first instruction of the task that wakes to deal with the ISR. All of the results are between 10.150 us and 10.350 us.

F	E B::trace.stat.AddressDURation iTIM4ISR\3 tADC\84									×	
	🖉 Setup 🛛 🏨	Chart 🏮 Z	oom 🏹 🛛	Zoom	Full						
		samples: total:	97. 2.447s	avr: in:	10.316us 1.001ms	min: out:	10.114us 2.446s	max: ratio:	19.965us 0.040%		
	up to	count	ratio	1%	2%	5%	10%	20%	50%	100	
<	9.600us	0.	0.000%								~
	9.700us	0.	0.000%								
	9.800us	0.	0.000%								
	9.900us	0.	0.000%								
	10.000us	0.	0.000%								
	10.100us	31	31 958%								
	10.300us	65.	67.010%								
	10.400us	0.	0.000%								
	10.500us	0.	0.000%								
	10.600us	0.	0.000%								
	10.700us	0.	0.000%								
	10.800us	0.	0.000%								
	10.900us	0.	0.000%								
	11.000us	0.	0.000%								
	11.100us		0.000%								
	11.20003	1.	1.030%								~
F		< 1.	1.050/0	T.						1	
1	_										

## Trace.STATistic.AddressDURation iTIM4ISR\3 tADC\84

A pair of breakpoints can be used: a breakpoint of type **/TRACEON** to start tracing at a particular event; and a breakpoint of type **/TRACEOFF** to stop tracing at this event. For example: To trace the main loop of a task (tMotor), set a pair of breakpoints to mark the on and off positions.

B::Break.List	:							×
💥 Delete All 🔇	🔾 Disable All	Enable All	⊗ Init	Impl	Store	🔀 Load	诊 Set	
address		types	impl	action				
T:(	080073DE	Program	ONCHIF	P Trace0	N ti	Motor\33		$\wedge$
T:(	080074E0	Program	ONCHIF	P Trace0	FF t	Motor\82+0	)x24	
		1		1	I		\ \	
1		×					/	.::

Run the target to collect the trace data and then view the results. As we can see from the image below, more has been captured than just the task we were interested in.

A B::Trace.Chart.sYm	bol									
🔑 Setup 👖 Grou	ips	Config.	A	Goto	🔒 Go	to	👘 Find.	🕩 1	n ⊧⊡∙O	ut 🖸 Full
		3	.500s	-	-3.000	s	-2.50	0 s	-2.000	s
	addre	ess 🗰								
	(othe	er) 🚯								🗛
DC2016\app_moto	r\tMot	tor 🐽				ЦШЦІ	U) ( I ( I ( I ( I			
2016\os_mbox\OS	MboxPe	end 👀				ļļII I I Į I	IIIIIIIIII			
6\Global\OS_CPU	_SR_Sa	ave 🗤 📃								
s_core\OS_Event	TaskWa	ait 🐠	<u>, III, III</u>	<u>İIII (IIII (</u>		IİIIIİI	IIIIIII		Ü I I Ü I I Ü I I Ü I I	1
DC2016	_Kesto	bret	in nim			ninnin				ni mini mi
DC2016 \os_core \	Cobodi	leu ()		<u> </u>		IIIIIIII	<u>11,111,111,</u> 111,		Ü I I I I I I I I I I I I I I I I I I I	
\ppc2016\clobal	\osc+v	Cw III				ninnin				ninninni
al\OS COLL Dends	VHandl	lor IIII								
PIL PendSVHandle	r nosa	veil								
6\os_cpu_c\0STa	skSwHo	ook iiiiii				ii i i i i i i i i i i i i i i i i i i				
016\app lcd\App	TaskL	CD	ί Τ΄ Τ΄ Τ΄		1111	ПТТТ.	11 111		ÌIÌIÌ	
2016\Global\ a	eabi i	i2f o					- · · · · · í		( ) ( ) ( )	
C2016\Global\	iar_ui	i2f 🕡		i i i		i i	i i í		i i i	· · ·
2016\Global\a	eabi_f	2d 🐽	· · I	i li i	· · · ·	ĽĽ	i i fi	· · · ·	i ni i i i	i ii
016\Global\ae	abi_do	liv		1 II I		1			. ( .	i i i
2016\Global\a	eabi_c	12f 🚯		1 I I						
rv_glcd\GLCD_Te	xtSetF	os 🐽		I.,		1.	J			L
6\drv_glcd\GLCD	_SetFo	ont 🐽	L II		, I, I, I	ĻLL,	II.II	I, L	JILI.	1,1 1,1 1,
\\DDC2016\Globa	\prir	itt 🐽		<u>,    ,    ,</u>		<u>,      </u>				1,1,1,1,1,
2016\Global\_Pr	INTTFL									
UDC2016/GIODAI	\_Mbtc	DWC 🕦								
		<	>	<						<b>&gt;</b> :

By marking a start and stop point, all code executed between these two points will be sampled. This includes interrupts, sub-functions, task switches, etc.

If the application is running as a small bare metal loop then this approach works very well. If the application has an RTOS or scheduler then more data may be collected than anticipated.

Setting a ranged breakpoint does not help as the DWT will generate a trace event for each branch within the range. This will result in a lot of FIFO overflows. To view only the code within the task of interest it is better to use a group (see "Trace and Groups", page 40) and filter on that afterwards. If the amount of trace is too large to fit in the tools' internal buffer, consider using Stream mode: "ETM Stream Mode", page 17.

All graphical views of the trace data have some common navigation features.

Zoom/pan using the In/Out/Full buttons. Some can zoom in two dimensions, others in only one.

B::Trace.PROfileCha	rt.sYmbol								
🔑 Setup 👖 Group	os 🔡 Config	🔒 Goto	👘 Find	. ∎ •⊡• In	Out D Full	‡ In	🛓 Out	🗘 Full	Fine
10.000us	(other)		iTIM4I	7D	OSSomP	oct	_	OS_CPU	SR_Sav
	0s	-4.000s		- 3	3.000s		-2.00 <u>0</u> s		
ratio									

Click and drag to select an area. Click within the selected area to zoom that to full screen.



B::Trace.Cha	art.sYmbol														-		×
🔑 Setup	iii Groups	E Confi	g	<b>∩</b> Goto	)	<b>P</b>	oto	Ĥ	Find	• <b>⊡•</b> In	▶ <b>□•</b> Out	🖸 Full					
			-	4.1009	5		-4.	050s		-4.00	)0s	- 3	.950s		-3.9	900s	
	add	ress 🗰															
	(ot	her) 🚯			.												^
DC2016\app	_adc\iTIM	4ISR 🔸			1.												
DC2016\os_	sem\OSSem	iPost 🐽			1.			1.									
6\Global\0	S_CPU_SR_	Save	IIIII		III II III		IIII II II II II II		I II II II II II II II II II II II II I			ļu u iļu ļu į		U, III III II, III III II		u III IŲ III III IIĮ	
os_core\05	_EventTas	kRdy 🐽												Į .			
core\OS_Ev	entTaskRe	move							l					ļ			į.
Tobal (OS_C	PU_SK_Res	tore			III II III			I II II,II II I	I, II II II II II II II II II II II II I			ļu u iļu ļu ļ	l III II III	ų III III II U		li li li li li li li	ļ.
DC2016\05_	core\0S_S	cnea 🗈															L.
016\05_C0r	e \US_Sche	anew ()			u u u u u			I II II,II II I				ļu u iļu ļu		U U U U U U U U		u nu ni nu ni	ų.
TTM Close	TODAT (USC								· ·					÷ .			ŀ
	Pandevuan								· ·					÷ •			ł
	penusvnan andlon no				1.1				· ·								ŀ
6) os cou c	And ter _no	Hook			1.1				· ·	· ·				4			ŀ
DC2016\os		Dendu	1		1.1									4 .			t i
$1 \times 10000000000000000000000000000000000$	ann adc	+ ADC IN	1		1.1				2.1					1 .	· · ·		r
DC Softwar	estartCon	vCmd III			1.1					· ·				1 .	· · ·		r i
x adc\ADC	GetElanSt	atus	11						2	· · ·				1 1	· · ·		ſ
ADC GetCo	nversionV	alue	1		1					1				1			V
			<	> <										12111111111111111111111111111111111111		>	
1			-														

Click and drag to time a region.

H B::Trace.Chart.sYmbol									×
Setup iii Groups II Config	Goto	🔒 Goto	🛐 Find	<b>∙⊡• I</b> n	•⊡• Out	🖸 Full			
	-4.00	6268000s	-4	.006260	5000s		-4.00	6264000s	
address 👀	I								
(other)									. 🔥
DC2016\app_adc\1TIM4ISR									
DC2016\os_sem\OSSemPost						·			
6\Global\OS_CPU_SR_Save						. 💻			
os_core\OS_EventTaskRdy@									
core OS_Event LaskRemove 0		_							
Tobal \OS_CPU_SR_Restore						_			
DC2016\os_core\OS_Sched									
DDG2016) clabal) OSCHWWW						C-W: /	2.509US		
VDC2016 (GIODAI (OSCEXSWA)						scale:	1.000us		
allos CDU Dendsyllandler									
BU DendSyllandlen necesse									
6) of course COSTack Sullack									
DC2016\os_som\OSSomBond									
\\DDC2016\app.adc\+ADC	· · ·								
DC SoftwareStartConvCmd									- <b>-</b>
v adc/ADC CetElagStatus									
ADC GetConversionValue									· 🗸
Abc_decconversionvarue						11.			
	~ `								·

Click and use the mouse scroll wheel: scroll up to zoom in and scroll down to zoom out.

Double-click but don't release the second mouse button press. Whilst holding the button, move the mouse up to zoom in and down to zoom out. Move left and right to scroll through the window along the x-axis.

The Trace.Chart.sYmbol window can be ordered along the y-axis. Drag the row to the required position.

🕂 B::Trace.Chart.sYmbol					_	×		B::Trace.C	hart.sYmbol						
Setup jiiGroups II Config	Goto	🔒 Goto	Find	<b>∙⊡• I</b> n	•Out 🗈	Full		🔑 Setup	iii Groups	Config.	. 🔒 Goto	🔒 Goto	Find	In PD+C	Dut 🖸 Full
	-4.00	6268000s	-	4.0062	66000s						-4.0	06268000s	-	4.00626600	00s
address 💀						_			ad	dress 💀				1	
(other)						^	-	\\DDC201	6\app_ado	\tADC 🚯	_				. ^
DC2016\app_adc\111M41SR								DC 2016)	(0)	ther)					
6\clobal\OS_CPU_SP_Savo							▼	DC2016\ap	p_adc\III	mPost W					
os core\OS EventTaskRdv								6\Global\		Savell					
core\OS EventTaskRemove								os core\0	S EventTa	skRdv	10				
lobal OS_CPU_SR_Restore								core\OS_E	ventTaskR	emove					
DC2016\os_core\OS_Sched				/				lobal\05_	CPU_SR_Re	store					
016\os_core\OS_SchedNew								DC2016\os	_core\ <b>OS</b> _	Sched 🚯					_
\DDC2016\Global\ <b>OSCtxSw</b>								016\os_co	re\OS_Sch	iedNew 🕩					
M IIM_ClearIlPendingBit								VDDC2016	Global\OS	CtxSwill					
BU Rends/Handler, nosave									PondSVH3	ngBit @					
6\os_cpu_c\0STaskSwHook								PU PendSV	Handler r	Ind ref 12					
DC2016\os_sem\OSSemPend								6\os cpu	c\0STaskS	wHook 🕷					
\\DDC2016\app_adc\ <b>tADC</b>								DC2016\os	sem\0SSe	mPend 💀					
DC_SoftwareStartConvCmd								DC_Softwa	reStartCo	nvCmd 🚯					
x_adc\ADC_GetFlagStatus					]  [	~		x_adc\ADC	_GetFlagS	itatus 🚯					×
<	> <				2	► lat					> <				>

Adding the **/Track** option to any window showing trace data allows it to snap to a cursor placed in any other trace display window.

B::Trace.List /Trac	k	22	3 🛃 B:: Trace. Chart. sYmbol
🖉 Setup 🔉 G	oto 👘 Find 🕂 Chart 📕 Profile 📕 MIPS 🗢 More 🗶 Less		Setup iii Groups II Config Q Goto Q Goto iii Find II In III Out III Fu
record ru	n address cycle data symbol t	i	-4.006268000s -4.006266000s
729	Course a real of the second se	^	address
7.7.1	/* Check the parameters */		\\DDC2016\app_adc\tADC \\
/ 31	assert_param(IS_ADC_ALL_PERIPH(ADCx));	¥	Y DC2016 and add STMC 20
	/* Return the selected ADC conversion value */	^	A DC2016 App_adt (TIPHTISK )
734	return (u16) ADCx->DR:		6\Global\OS_CPU_SR_Save
	ldr r0,[r0,#0x4C] ; ADCx,[r0,#76]		os_core\OS_EventTaskRdy
	uxth r0,r0 ; ADCx,ADCx		core OS_EventTaskRemove
0010515555	bx r14		lobal OS_CPU_SR_Restore
-0018515655	T:0800/D44 ptrace \\DDC2016\app_adc\tADC+0x134	-	UC2016\os_core\US_SchedNewW
114	if (new ADC 1= old ADC)		
	ldr r0.[r13]		m TIM Clear ITPendingBit
	cmp r0,r5 ; r0,old_ADC		al\OS_CPU_PendSVHandler
	beq 0x8007DB2 ; ??tADC_4		PU_PendSVHandler_nosave
-0018515654	T:08007D4C ptrace \\DDC2016\app_adc\tADC+0x13C		6\os_cpu_c\OSTaskSwHook
110	1 (f ( down Tanana ) 2 ( down Tanana ) 2 )		DC2016\05_SemV0SSemVend 0
110	$\frac{1}{1} \frac{1}{1} \frac{1}$	~	v x adc/ADC GetFlanStatus
<	Tur To, oxoo EoT , To, bucutubles_1	>	< > < >

The collected trace data can be analyzed to show a lot of performance information about the code running on the target. This is handled by the **PERFormance** commands and more information can be found by following the link. The features can also be accessed from the **Perf** menu; the highlighted features will be discussed in this manual.



These items work with collected trace data.

The items in this menu deal with how long each function and/or sub-function takes to execute and how many times it is called. A basic display looks like this:

E::Trace.ST	TATistic.FUNC															×
Setup	iii Groups	Confi	ia 🔒 Goto.	🗾 Detailed	E Nestina	🐨 Chart										
•			funcs: 35.	to	tal: 5.0	65										
			ranco. 55.			.05										
		r ange	total	min	max	avr	count	intern%	1%	2%	5%	10%	20%	50%	100	
	(	root)	5.026s	-	5.026s	-	-	<0.001%	+							~
\\demo_	sram\sieve	main	5.026s	-	5.026s	-	1. (0/1)	8.894%				-				
demo_sram	\sieve\ <b>fun</b>	nc_sin	660.788us	660.788us	660.788us	660.788us	1.	0.006%	+							
\demo_sra	m\sieve\ <b>in</b>	nt_sin	325.554us	0.128us	0.697us	0.518us	628.	0.006%	+							
\\demo_s	ram\sieve\	tunc2	198.485ms	2.044us	2./63us	2.508us	/9135.	3.438%								
\\demo_s	ram\sieve\	Tunci	63.609ms	0.01/us	0.236us	0.115us	553446.	1.265%								
\\demo_sr	am\sieve\	uncza	106.252ms	1.100US	1.455US	1.343US	/9135.	2.114%		_						
\\demo_sr	am\sieve\f	unc2b	102 110mc	1.05905	1.20005	1.10005	70125	2.021%		-						
(\ueino_si	linitLinko	dist	355 464mc	4 328uc	4. 640us	4 49205	70135	7 072%		_						
\\demo_si	ram\sieve\	func4	59 626ms	0.492us	1.021us	0.753us	79135	1 186%								
\\demo_si	ram\sieve\	func3	5.478ms	0.033us	0.144us	0.069us	79135.	0.108%	+							
\\demo_s	ram\sieve\	func5	7.148ms	0.034us	0.118us	0.090us	79135.	0.142%	÷.							
\\demo_s	ram\sieve\	func8	299.043ms	3.671us	3.932us	3.779us	79135.	5.950%								
\\demo_s	ram\sieve\	func9	148.041ms	1.775us	1.975us	1.871us	79135.	2.190%								
\\demo_sr	am\sieve\f	unc10	1.508s	18.916us	19.135us	19.052us	79135.	29.998%								
\\demo_sr	am\sieve\f	unc11	27.270ms	0.231us	0.462us	0.345us	79135.	0.542%	+							
\\demo_sr	am\sieve\f	unc13	162.132ms	0.150us	2.166us	1.180us	316538.(0/2)	3.225%		_						
\\demo_sr;	am\sieve\t	unc14	7.191ms	0.060us	0.12/us	0.091us	/9134.	0.143%	<del>•</del>							
\\demo_sr	am\sieve\	unc15	7.780ms	0.048us	0.159us	0.098us	/9134.	0.154%	•							
\\demo_sr	am\sieve\	unc16	7.734ms	0.04305	0.16/US	0.09805	/9134.	0.153%	•							
\\demo_sr	am\sieve\f	unc1/	7.52/ms	0.04005	0.12005	0.09505	79134.	0.145%	5							
\\demo_sr	am\sieve\	uncio	7.300IIIS	0.06/us	0.12405	0.09005	79134.	0.130%	Σ							
\\demo_sr	am\sieve\f	unc20	8 583mc	0.063us	0.1140	0.108us	79134	0.170%	2							
\\demo_sr	am\sieve\f	unc21	8. 320ms	0.070us	0.138us	0.105us	79134	0.165%	2							
\\demo_sr;	am\sieve\f	unc22	8.271ms	0.070us	0.170us	0.104us	79134.	0.164%	÷							
\\demo_sr	am\sieve\f	unc23	9.904ms	0.035us	0.218us	0.125us	79134.	0.197%	÷.							
\\demo_sr	am\sieve\f	unc24	5.024ms	0.028us	0.140us	0.063us	79134.	0.099%	•							
\\demo_sr	am\sieve\f	unc25	4.846ms	0.022us	0.122us	0.061us	79134.	0.096%	+							
\\demo_sr	am\sieve\ <b>f</b>	unc26	4.345ms	0.021us	0.104us	0.055us	79134.	0.086%	+							
\\demo_sr	am\sieve\ <b>f</b>	unc28	25.705ms	0.245us	0.470us	0.325us	79134.	0.511%	+							
\\demo_sr	am\sieve\e	ncode	571.106ms	7.048us	7.338us	7.217us	79134.	6.480%								
\\demo_s	ram\sieve\	subst	245.390ms	0.016us	0.521us	0.310us	/91340.	4.882%								
\\demo_s	ram\sieve\	steve	815.13/ms	10.201us	10.363us	10.301us	/9134.	16.218%								
																~
I			<													2 .::

This shows a list of functions that were sampled during the trace collection period and runtime statistics (including min, max and mean) for each.

Many other columns can be added and the list can be sorted in a number of ways. Clicking the "Config" button gives access to this dialog, which looks like this.

Statistic Config		– 🗆 ×
Sort       Sort visible       available            • OFF           • Global        NAME         GROUP             • Address           • Sort core        TotalRatio             • Symbol           • CoreSeparated        Internal             • TotalRatio           • CoreSeparated        Internal             • TotalRatio           • CoreSeparated        Internal             • TotalRAtio           • MIN        Internal             • TotalMAX           • RatioMAX        Internal             • All Windows           • All Windows	G ->	selected Total MIN MAX AVeRage Count InternalRatio InternalBAR.LOG

Each line in the list has a right-click menu associated with it. This provides access to more detailed analyses.

B::Trace.STATistic.FUNC						
🖉 Setup 👖 Groups 🔡 Config 🔒	🕻 Goto 🛛 🗾 Detail	ed 🗵 Nesting	Chart			
	funcs: 32.	to	otal: 14.86	65s		
	<b>.</b>		1	1		
range	total	m1n 0.016	max 0 F21	avr 0.210	count	intern%
\\demo_sram\sieve\subst	/25.000ms	0.010us	0.521us	0.310us	2340558.(1/0)	4.8//%
(noot)	14 865c	7.02/us	1/ 865c	7.219us	234030.(1/0)	8 807%
\\demo_sram\sieve\sieve	2 410s	10 201us	10 414us	10 29905	234056	16 215%
\\demo_sram\sieve\ <b>func2</b>	588,452ms	Statistic	.765us	2.514us	234056.	3,440%
\\demo_sram\sieve\ <b>func1</b>	188.901ms	List First	. 235us	0.115us	1637893.	1.270%
\\demo_sram\sieve\ <b>func2a</b>	313.085ms	Liet Laet	.455us	1.338us	234056.	2.106%
\\demo_sram\sieve\ <b>func2b</b>	276.292ms		. 327us	1.180us	234056.	1.858%
\\demo_sram\sieve\ <b>func2d</b>	302.647ms		. 397us	1.293us	234056.	2.035%
ram\sieve\initLinkedList	1.051s	🕽 Goto Max	.640us	4.490us	234056.	7.069%
\\demo_sram\sieve\ <b>func4</b>	177.079ms	Bookmark Max	.021us	0.757us	234056.	1.191%
\\demo_sram\sieve\func3	16.449ms		143us	0.0/0us	234056.	0.110%
\\demo_sram\sieve\ <b>func5</b>	21.299ms	E Linkage	.119us	0.091us	234056.	0.143%
\\demo_sram\sieve\ <b>Tunc8</b>	884.002ms	Parents	. 903us	3.///us	234056.	5.940%
\\demo_sram\sieve\ <b>tunc9</b>	457.010ms	Children	. 97 JUS	10.051us	234056 (0/1)	2.10/%
\\demo_sram\sieve\functo	4.439S		135us	19.031us	234030.(0/1)	29.997%
\\domo_snam\sieve\func13	470 372mc	Duration Analysis	166us	1 185uc	036220	3 22/9
\\demo_sram\sieve\func14	21 508ms	Findall Duration	15605	0.00205	234055	0 1/1/%
\\demo_sram\sieve\func15	23 068ms	Distance Analysis	15705	0.09203	234055	0 155%
\\demo_sram\sieve\func16	23.120ms	Findall Distance	.167us	0.099us	234055	0.155%
\\demo_sram\sieve\func17	21.506ms			0.092us	234055.	0.144%
\\demo_sram\sieve\func18	22.673ms	here	• .120us	0.097us	234055.	0.152%
\\demo_sram\sieve\func19	21.535ms	0.058us	0.114us	0.092us	234055.	0.144%
\\demo_sram\sieve\func20	24.873ms	0.063us	0.125us	0.106us	234055.	0.167%
\\demo_sram\sieve\ <b>func21</b>	24.735ms	0.070us	0.133us	0.106us	234055.	0.166%
\\demo_sram\sieve\func22	24.605ms	0.070us	0.133us	0.105us	234055.	0.165%
\\demo_sram\sieve\func23	29.060ms	0.108us	0.165us	0.124us	234055.	0.195%
\\demo_sram\sieve\func24	14.171ms	0.038us	0.140us	0.060us	234055.	0.095%
]	<					

The first section jumps to the first, last or maximum entry in the Trace.List window.

**Linkage** shows an analysis of all places that this function was called from along with runtime information. This example shows all the places in the application where func1() is called from and for each it displays the runtime measurements. It is a convenient method to access **Trace.STATistic.LINKage**.

E B::Trace.ST	AT.LINKage T	:0x200000	86							X
🖉 Setup	iii Groups	E Conf	ig 🔒 Goto	🗾 Detailed	Nesting	Chart				
			funcs: 2.	t	otal: 63.6	09ms				
		range	total	min	max	avr	count	total% 1%	2%	
\\demo_sr	am\sieve\	func2	25.676ms	0.017us	0.212us	0.108us	236906.	40.365%		$\sim$
\\demo_sr	amisiever	(Tune9	37.933005	0.027us	0.25605	0.12005	516540.	59.054%		
			<						>	Ĭ.,
1		_								

**Parents** shows the call tree back to the entry point or root of the application, again with performance information for each function in the tree. This is a convenient way to access **Trace.STATistic.ParentTREE**.

B::Trace.STAT.ParentTREE T:0x20000086		- • 💌
🌽 Setup iii Groups 🔡 Config 📭 Goto 🛒 Detailed 😿 Nesting	🙀 Chart	
funcs: 7. total: 63.0	309ms	
range tree	total min max avr count	tota]% 1% 2%
\\demo_sram\sieve\func1 \\demo_sram\sieve\func1 \\demo_sram\sieve\func2	63.609ms 0.01/us 0.236us 0.115us 553446. 25.676ms 0.017us 0.212us 0.108us 236906.	40.365%
\\demo_sram\sieve\main (root)	25.676ms 0.017us 0.212us 0.108us 236906. 25.676ms 0.017us 0.212us 0.108us 236906.	40.365%
\\demo_sram\sieve\func9	37.933ms 0.027us 0.236us 0.120us 316540.	. 59.634%
(root) (root)	37.933ms 0.027us 0.236us 0.120us 316540.	59.634%
		<u> </u>
<		i. <

**Children** shows the call tree starting at the selected function and traversing downwards through all of the sub-functions with performance information for each node. This menu item provides an easy way to access **Trace.STATistic.ChildTREE**.

B::Trace.STAT.ChildTREE T:0x2000008E			
Setup iii Groups II Config Q G	oto 🗧 Detailed 🗵 Nesting 🛛 🖼 Cha	irt	
funcs: 2	. total: 198.485ms		
range tree	total min ma	ix avr count	intern% 1% 2%
\\demo_sram\sieve\tunc2 = func2 \\demo_sram\sieve\func1 = func2	198.485ms 2.044us 1 25.676ms 0.017us	2.763us 2.508us 79135. 0.212us 0.108us 236906.	87.064%
			· · · · · · · · · · · · · · · · · · ·
<			ي. <

**Duration** shows a histogram of runtimes for the selected function. TRACE32 allocates 16 appropriate bucket sizes ad assigns the runtime values to each of these. These can be over-ridden by the user on the command line by using **Trace.STATistic.FuncDURation**. The zoom buttons and scroll bar can be used to navigate or display more or less details for a specific range.

🗾 B::Trace.STAT.FuncDURation P:0x2000008E						
🔑 Setup 📠 Chart 🔅	Zoom 📮 Zoom 📮 Full					
samples: total:	79135. avr: 2.508us mir 5.026s in: 198.485ms out	1: 2.044us max: 2.763us 2: 4.827s ratio: 3.949%				
up to count	ratio 1% 2% 5%	10% 20% 50% 100				
<ul> <li>2.200us</li> <li>2.250us</li> <li>2.350us</li> <li>2.350us</li> <li>2.350us</li> <li>138</li> <li>2.400us</li> <li>6800</li> <li>2.450us</li> <li>2.550us</li> <li>2.977</li> <li>2.550us</li> <li>2.650us</li> <li>2.400us</li> <li>8477</li> <li>2.650us</li> <li>2.700us</li> <li>3060</li> <li>2.750us</li> <li>463</li> <li>2.850us</li> <li>2.850us</li> <li>2.850us</li> <li>2.950us</li> <li>3.000us</li> <li>463</li> <li>3.000us</li> <li>464</li> </ul>	. 0.338% + 0.001% + 1.752% . 8.600% . 27.771% . 13.882% . 10.710% . 15.560% . 0.585% + 0.015% + 0.000% . 0.000% .					

**Findall Duration** provides a convenient way of automatically searching for all entry and exit points for the selected function. The yellow lines are function entries and the white lines are the corresponding exit. The value in the ti.fore column shows the time for each event, so the top line shows that func2 took 2.20  $\mu$ s to execute and it was another 60.960 $\mu$ s before it was called again.

🛐 B::Trace.FindA	II , Address.CODE P:0x20	00008E  P:0x200000E4 /List Address sYmbol Tlr	n 🗖 🗖 💌
158270	address	symbol	ti.fore
-0028807885	T:200008E	\\demo_sram\sieve\func2	2.200us 🔨
-0028807872	T:200000E4	<pre>\\demo_sram\sieve\func2+0x56</pre>	60.960us
-0028807527	T:200008E	\\demo_sram\sieve\func2	2.080us
-0028807514	T:200000E4	\\demo_sram\sieve\func2+0x56	60.896us 🎽
-0028807164	T:200008E	\\demo_sram\sieve\func2	2.304us 🛆
-0028807151	T:200000E4	\\demo_sram\sieve\func2+0x56	61.120us
-0028806801	T:200008E	\\demo_sram\sieve\func2	2.060us
-0028806788	T:200000E4	\\demo_sram\sieve\func2+0x56	61.140us
-0028806445	T:200008E	\\demo_sram\sieve\func2	2.060us
-0028806432	T:200000E4	\\demo_sram\sieve\func2+0x56	61.120us
-0028806075	T:200008E	\\demo_sram\sieve\func2	2.040us
-0028806062	T:200000E4	\\demo_sram\sieve\func2+0x56	61.000us
-0028805711	T:200008E	\\demo_sram\sieve\func2	2.040us
-0028805698	T:200000E4	\\demo_sram\sieve\func2+0x56	61.160us
-0028805351	T:200008E	\\demo_sram\sieve\func2	2.040us
-0028805338	T:200000E4	\\demo_sram\sieve\func2+0x56	60.896us
-0028804994	T:200008E	\\demo_sram\sieve\func2	2.304us
-0028804979	T:200000E4	\\demo_sram\sieve\func2+0x56	60.896us
-0028804636	T:200008E	\\demo_sram\sieve\tunc2	2.344us
-0028804617	T:200000E4	\\demo_sram\sieve\func2+0x56	60.808us
-0028804270	T:200008E	\\demo_sram\sieve\func2	2.292us
-0028804253	T:200000E4	\\demo_sram\sieve\func2+0x56	60.960us
-0028803903	T:200008E	\\demo_sram\sieve\func2	2.080us
-0028803891	T:200000E4	\\demo_sram\sieve\func2+0x56	61.120us
-0028803545	T:200008E	\\demo_sram\s1eve\func2	2.120us ¥
	<		). (

**Distance Analysis** shows the amount of time that elapsed between one call to a function and the next call to that function. This is a convenient way of accessing the **Trace.STATistic.AddressDIStance** command.

E::Trace.STAT.AddressDIStance P:	0x20000086			-		×
🖉 Setup 📠 Chart 🏼 🏮 Zo	oom 🖕 Zoom	🗣 Full				
samples: total:	553445. avr 5.026s in:	9.080us 5.025s	min: 0.050us out: 768.427us	a max: 45.360us ratio: 99.9849	; 6	
up to count	ratio  1%	2%	5% 10%	20% 50%	100	
<ul> <li>&lt; 0.000us</li> <li>0.395176.</li> <li>10.000us</li> <li>0.15.000us</li> <li>0.20.000us</li> <li>79135.</li> <li>25.000us</li> <li>0.30.000us</li> <li>0.35.000us</li> <li>0.40.000us</li> <li>0.40.000us</li> <li>0.45.000us</li> <li>3893.</li> <li>50.000us</li> <li>0.60.000us</li> <li>0.65.000us</li> <li>0.75.000us</li> <li>0.75.000us</li> <li>0.80.000us</li> <li>0.</li> </ul>	0.000% 71.402% 0.000% 0.000% 14.298% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000% 0.000%					< >
() J <sup>×</sup>						

**Findall Distance** is an easy way to search for all entry points of the selected function. The ti.fore column shows the time between calls to that function.

🛐 B::Trace.FindA	II , Address.CODE P:0x2000008E /List Address sYmbol Time.Back	
79135	address symbol	ti.back 🔤
-0028807885	T:2000008E \\demo_sram\sieve\func2	~
-0028807527	T:2000008E \\demo_sram\sieve\func2	63.160us
-0028807164	T:2000008E \\demo_sram\sieve\func2	62.976us
-0028806801	T:2000008E \\demo_sram\sieve\func2	63.424us 🎽
-0028806445	T:2000008E \\demo_sram\sieve\func2	63.200us 🛆
-0028806075	T:2000008E \\demo_sram\sieve\func2	63.180us
-0028805711	T:2000008E \\demo_sram\sieve\func2	63.040us
-0028805351	T:2000008E \\demo_sram\sieve\func2	63.200us
-0028804994	T:2000008E \\demo_sram\sieve\func2	62.936us
-0028804636	T:2000008E \\demo_sram\sieve\func2	63.200us
-0028804270	T:2000008E \\demo_sram\sieve\func2	63.152us
-0028803903	T:2000008E \\demo_sram\sieve\func2	63.252us
-0028803545	T:2000008E \\demo_sram\sieve\func2	63.200us
-0028803181	T:2000008E \\demo_sram\sieve\func2	63.240us
-0028802810	T:2000008E \\demo_sram\sieve\func2	62.960us
-0028802452	T:2000008E \\demo_sram\sieve\func2	63.088us
-0028802087	T:2000008E \\demo_sram\sieve\func2	63.292us
-0028801718	T:2000008E \\demo_sram\sieve\func2	63.036us
-0028801357	T:2000008E \\demo_sram\sieve\func2	63.364us
-0028800993	T:2000008E \\demo_sram\sieve\func2	63.300us
-0028800626	T:2000008E \\demo_sram\sieve\func2	62.852us
-0028800267	T:2000008E \\demo_sram\sieve\func2	63.288us
-0028799904	T:2000008E \\demo_sram\sieve\func2	62.976us
-0028799541	T:2000008E \\demo_sram\sieve\func2	63.404us
-0028/99183	T:2000008E \\demo_sram\sieve\func2	63.260us 🗸
	<	≥ <sub>ii</sub>

Distribution shows the values that have been assigned to a variable during the sampling period. This requires data trace, which on Cortex-M requires the ITM/DWT to be configured correctly. Data trace packets consume more trace bandwidth than program flow trace, so care should be taken when monitoring data items not to cause overflows in the on chip FIFO of the target.

To monitor a variable, a breakpoint can be used to program the DWT to emit a data trace packet on reads, writes or any access. To monitor writes to HLL variable flags[3], set a breakpoint like this:

Break.Set Var.ADDRESS(flags[3]) /Write /TraceData

The Var.ADDRESS() macro returns the address of the 4th element of array flags. It can be used on any complex variable where a simple symbol table lookup will not find the address. The same breakpoint would be set in the UI like this.

🕲 B::Break.Set			_		$\times$
Start a capture w	ith the selected sett	ings.	<u>2</u> [	HLL	*
type Program ReadWrite Read Write default	options EXclude NoMark	Temporary DISable DISableHIT	a a T	nethod uto ction — TraceDat ¥ adva	ta ~
Ok	Add	Delete		Cance	el

The ITM needs to be set for Data Trace: ITM.DataTrace ON or via the GUI. Start the target to collect trace samples.

If the ETM trace is set to OFF (no ETM trace data generated) then the menu items under the **Perf** menu can be used to analyze the data. If ETM is on (Program flow trace data also generated) then the analysis cannot be performed by using the items under the **Perf** menu. When ETM is detected, the **Perf** will default to using ETM trace for analysis. Instead, use the ITMTrace series of commands. Both commands will be shown in the examples below. The generic Trace command can also be used.

## Trace.STATisitc.DistriB Data

B::Trace.STATistic.D	istriB Data													×
Setup 🔡 Conf	fig 🔒 Goto	🗾 Detailed	l 🔂 Chart	🔼 Profile										
it	tems: 3.	tot	al: 1.61	4s sampl	es: 50820.									
class to	otal	min  r	iax	avr	count	ratio%	1%	2%	5%	10%	20%	50%	100	
(other)	86.600us	86.600us	86.600us	2 05705	0.	0.005%	+							^
data=0x1 data=0x0	1.536s	60.260us	60.540us	60.462us	25410. (1/0)	95.181%								
	ç												>	Ľ.
,														

The image above shows that for 95.181% of the sampling time, the value of flags[3] was 0 and for 4.813% it was a 1. The (other) entry is the time at the start of the trace capture where the value of flags[3] was unknown as no writes had yet been captured.

Clicking the Config button will open a dialog to allow the user to change the sorting order and to add or remove different columns.
Clicking the **Chart** button will show how the values of the variable changed over time in a graphical format.

# Trace.Chart.DistriB Data

A B::Trace.CH	ART.DistriB D	ata							-	
🔑 Setup	iii Groups	E Config	Q Goto	🔒 Goto	Find	<b>∙⊡• In</b>	•⊡• Out	• Full		
		-	1.1090000	00s			-	1.10800000s		
	c	lass	. I							
	data=0x	F248								· · ·
	data=0x	F20C								
	data=0x	:63F4 🗈								
	data=0x	6430								
	data=0x	646C	<b>—</b>							·
	data=0x	64A8								
	data=0x	64E4							· · · <u> </u>	. <b></b> .
	data=0x	6520		<b>-</b>					· · · ·	
	data=0x	CLOB IN							· · · ·	
	data=0x	CED4 III							· · ·	
	data=0x	661000							· · ·	
	data-0x	6640 00			• <u> </u>				· · · · · ·	
	data-0x	6688			· - ·					
	data-0x	6604			· · ·			· · · · ·		
	data-0x	6700			· · ·			· · · <b></b>		
	data=0x	6730						· · · · ·		· · · · ·
	data=0x	6778					1 - I			· · · ·
	data=0x	67B4								
	data=0x	67F0								· · · · · · · · · · · · · · · · · · ·
		<	> <							
1		1.								

This feature uses the **Trace.STATistic.DURation** command to measure the time between two arbitrary points; it is no longer constrained to function entry and the corresponding exit. For instruction timing **Trace.STATistic.AddressDURation** is better.

To show how long a variable contains one value before it switches to another use something like:

Trace.STATistic.DURation /FilterA Data.</width> <value> /FilterB Data.</width> <value>

Where *<width>* can be:

- B byte (8bits), can be split into B0, B1, B2 or B3 to represent a single byte in a 32bit access
- W Word (16bits)
- L long (32bits)
- Q Quad (64bits)
- T Triple (128 bits)

#### To measure the time between a variable changing from a 0 to a 1:

- 1. Set a breakpoint on the variable to generate trace data, for example: Break.Set Var.ADDRESS(flags[3]) /Write /TRACEDATA
- 2. Set ITM.ON
- 3. Set ITM.DataTrace ON
- 4. Collect trace data
- 5. Use Trace.STATistic.DURation /FilterA Data.B 0x00 /FilterB Data.B 0x01 to show the results

E B::Trace.STATis	tic.DURation /Filt	erA Data.B 0x	00 /FilterB	Data.B 0x01						×
🖉 Setup 📗	Chart 🛛 🏮 Z	oom 🍹	Zoom	Full						
	samples: total:	27982. 1.778s	avr: in:	60.424us 1.691s	min: out:	60.200us 86.854ms	max: ratio	60.540us : 95.114%		
up to	count	ratio	1%	2%	5%	10%	20%	50%	100	
<ul> <li>&lt; 60.150us</li> <li>60.200us</li> <li>60.250us</li> <li>60.300us</li> <li>60.350us</li> <li>60.400us</li> <li>60.500us</li> <li>60.600us</li> <li>60.600us</li> <li>60.750us</li> <li>60.850us</li> <li>60.850us</li> <li>60.900us</li> <li>60.950us</li> </ul>	0. 32. 169. 1335. 3059. 4223. 8863. 6465. 3836. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.	$\begin{array}{c} 0.000\%\\ 0.114\%\\ 0.603\%\\ 4.770\%\\ 10.932\%\\ 31.673\%\\ 23.104\%\\ 13.708\%\\ 0.000\%\\ 0.00\%$	÷ +		-					

This allows the user to time between two arbitrary points in the application code. This is done by using the **/Filter** options to the **<trace>.STATistic.DIStance** command.

For example, if it is required to measure the distance between a call to func2() and the subsequent call to func5():

F	B::trace.stat.d	stance /Filter	Address func2	/Filter Addr	ess func5						×
	Setup 👔	Chart	🗘 Zoom	Ž Zoom	Full						
		samples total:	: 1668 1.061	7. avr: 5 in:	63.519us 1.060s	min: out:	63.300us 579.080us	max: ratio:	63.700us 99.945%		
	up to	count	ratio	1%	2%	5%	10%	20%	50%	100	
~	63, 250us 63, 350us 63, 450us 63, 450us 63, 550us 63, 550us 63, 550us 63, 650us 63, 750us 63, 750us 63, 750us 63, 850us 63, 950us 64, 000us 64, 050us	11 99 114 366 500 300 177 90	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	9% ← 9% ← 9% ← 9% / 9% / 2% ← 9% / 9% / 9% / 9% / 9% / 9% / 9% / 9% /		_	_				< > >

## Trace.STATistic.DIStance /Filter Address func2 /Filter Address func5

Filtering can also be performed using data values. Use the Data.<x> filter options instead of Address. For example, to filter on a variable containing the value 0x63f4 and the next time it contains the value 0x6520 use the command:

## Trace.STATistic.DIStance /Filter Data.W 0x63F4 /Filter Data.W 0x6520

B::trace.stat.	distance /Fil	ter Data	.W 0x63	3f4 /Filt	ter Data.W	/ 0x6520						×
Setup	llu Chart	¢ Z	oom	Ž 7	loom	Full						
	sample total:	es:	1.2	37. 34s	avr: in:	30.997ms 1.147s	min: out:	1.270ms 87.432ms	max: ratio	62.121ms : 92.916%		
up to	count		rati	0	1%	2%	5%	10%	20%	50%	100	
0.0000 5.000 10.000 15.000 25.000 30.000 30.000 40.000 45.000 60.000 60.000 75.000 75.000 80.000	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	0. 15. 4. 0. 0. 0. 0. 0. 0. 0. 9. 9. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0	0. 40. 10. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 24. 24. 24. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.	000% 540% 810% 000% 000% 000% 000% 000% 000% 324% 324% 324% 000% 000%					=	-	>	. < >

Trace data can be assigned to logical groups to aid analysis and filtering. Groups can be created with the **GROUP.Create** command and viewed with **GROUP.List**. For an overview of the **GROUP** functionality, refer to the **GROUP** command group.

Groups can be created around:

- Address Ranges
- Functions
- Modules
- Symbols
- Tasks

Grouping only affects the display of captured trace data, not the data itself.

When a group is created, TRACE32 automatically creates a base group called "other". This contains everything that is not a part of any defined group.

Grouping code into logical function blocks makes it easier to reduce the amount of trace data that users need to analyze and allows them to focus on the regions or interactions of interest.

More than one item can be part of a group. In this example two object files will be allocated to a single group. Both sets of code are involved with the motor control functions of the target so it makes sense to group them logically like this.

## GROUP.CreateTASK "Motor\_Tasks" "\app\_motor" "\app\_rpm" /RED

**GROUP.Create** without any arguments will open the creation dialog. The image below shows it filled in to create the same group as the command above. Clicking the blue 'i' button will open a module browser window from which the user can select multiple modules.

iii B::GROUP.Cr	eate		-		$\times$
name ——					
Motor_Tasks					
- addressrange( \app_motor \a	s)			~	0
– options ––––					
🗹 Enable	Hide	Merge	R	ED	$\sim$
Ok	Set	Delete		Cance	el

GROUP.List now shows the new group and the default "other" group.

i	ii B::GROUP.	.List									3
	💥 Reset	O Disable All	Enable All	O Hide All	Shov	v All 😤	Store	🔀 Load	iii Create	🙀 Create Task	
	group				enable	hide	merge	color			
	■ "Motor \ap \ap "other	r_Tasks" p_motor p_rpm r"						RED RED RED			^
					, ř						$\mathbf{v}$
	<									>	

In the GROUP.List window, check the hide column for group "other". Alternatively, use the command:

## **GROUP.HIDE** "other"

iii B::Group.List	- • ×
💥 Reset 🛛 Disable All 💿 Enable All 🔾 Hide All 💿 Show All 😨 Store 💱 Load 🉀 Create	🙀 Create Task
group enable hide merge color	
■ "Motor_Tasks" V RED	~
\app_motor √ RED	
KED	
other y y	~
	>

Now the trace view windows will suppress the display of anything that is marked as hidden.

B::Trace.Chart.sYmbol								
℅ Setup iii Groups	E Config	🔒 Goto	🔒 Goto	🛉 Find	<b>∙⊡• In</b>	•Out D Full		
	-	5.000s	-4.000s	-3.	000s	-2.000s	-1.000s	0.
add	ress 🛛 🔜							
(oth group "oth	ier) 🐽 🔜							^
C2016\app_motor\tMo	tor i							
\app_motor\AdjustPu	lse							
								~
	<	> <						>

With the "other" group hidden, the **Trace.List** window looks like this. Different groups are color coded and the colors can be seen in the bar on the left side of the window. As control passes between groups a line is added to the trace listing to show this. In the example below 75.459us were taken up processing group "other" and before that 1.120us were spent processing in group "Motor\_Tasks".

B::Trace.List									x
🖉 Setup 🖡	Goto 🛉 Find	Chart	🔼 Profile	K MIPS	More	Less	]		
record	run address	cycle	data	symbol				ti.back	
-0023457549	ldrh r0 uxth r7 add r1 ldrh r1 subs r1 adds r2 adds r2 ldr r2 mla r1 T:080 ldr r2 mla r1 grou ∫ b 0x	<pre>,[r0,r8,ls1 ,r7, ,sp,#0x4 ,r1,r8,ls1 # ,[r1,#0x2] ,r7,r1 ,sp,#0x4 ,r2,r8,ls1 # ,r2,r1,r0 06ELA ptrace 0x8007018 800721C p "Other" — p "Motor_Tas 8006D1E</pre>	#0x3]; #0x3 ; #0x3 ; #0x3 ; #0x3 ; #0x3 ; ; #0x3 ; ;	adc_val,ac r1,r1,Zone r1,adc_val r2,r2,Zone DDC201 r0,??DataT ITM_printf ??tMotor_C	dc_val a,lsl #3 l,r1 a,lsl #3 L6\app_mot Table3_4	tor\tMotor	+0x15A	0.000u: - 1.120u: - 75.469u:	S
78	while(DE	F_TRUE)							
80	add r2 movs r1 Idr r0 Idr r0 bI 0x	OSMboxPend(M ,sp,#0x0 ,#0x0 ,0x8007008 ,[r0] 8007082	NotorMbox ;	, O, &err) r0,??DataT OSMboxPend	); / Fable3	//wait for	ever f	or a mess	ig v
	_ <b>`</b>								<b>*</b> :

This example creates two groups: one for the application code and another for the kernel code. Multiple entries can be added to the addressrange(s) field, as shown in the "Application" example below. Whereas, the "kernel" entry only covers a single range.

👬 B::GROUP.Crea	te		-		×		🙀 B::group.create			_		Х
Application						]	Kernel					
addressrange(s)	x08007005 0x08008	1180x0800848D		~	<u>•</u>	]	addressrange(s)	08003A5F			~	1
← options ✓ Enable	Hide	☐ Merge	В	LUE	~	]	← options ✓ Enable	Hide	☐ Merge		RED	~
Ok	Set	Delete		Canc	el		Ok	Set	Delete	[	Cance	el

The groups can be listed using the **GROUP.List** command:

ſ	iii B::GROUP.	.List										×	
	💥 Reset	O Disable All	Enable All	O Hide All	Shov	v All	20	Store	🔀 Load	iii Create	🙀 Create Task		
	group				enable	hid	e	merge	color			Ē	
	■ "App1 \app (\ap = "Kerno (OSI "other	ication" p_motor pp\main) el" EventNameS r"	(iPA10ISR) et)(`??(	\12+0x1) OSTmr_Ta					BLUE BLUE BLUE RED RED				~
	<								1		)	Þ	

The "other" group is clearly visible here. The source view (List) windows now have an added color bar to show which group the code being viewed belongs to.

📕 (B::List tRP	[M							- 0	×
Step	Nover	Diverge	🗸 Return	Ċ Up	► Go	II Break	🔛 Mode	e 😽 1.	. "I
ado	dr/line s	ource							
Đ	45 46 47 48	if(demo ITM_p else if ITM_T	Trace == rintf("RPI (demoTra RACE_D16(	2    demo M: New Clo ce == 1) 2,cur_clk	Frace == ock value );	3 ) %d",cur_c	1k);		î
	50	if (err {	== 05_N0	_ERR)					-
	53	if (o	ld_clk >	cur_clk)	//Coun	ter has wr	apped		
	55	cur. }//if else	_ticks =	(0 - old_o	clk) + cu	r_clk;			
	59	cur. }//els	_ticks = 0 se	cur_clk -	old_clk;				
	62	if (c	ur_ticks :	> 0)	//Can't	divide by	zero so	catch i	t ¥ > .::

A trace listing window will show also show the groups color coded and will indicate where control passes from one group to another.

B::Trace.List									×
🖉 Setup 🔒	Goto 🏥	)Find 🗛 🔿	Chart 🛛 🔼 Prof	ile 📕 MIPS	More	Less			
record	run addre	ss	cycle dat	a symbol				ti.back	
94 95	msr B bx	SR PRIM primask, X LR r14	n <mark>ask, RO</mark> rO						<b>^</b>
-0004219106		T:08002C8A	ptrace	DDC20	16\os_core	e\OSIntExi	t+0x92	0.040us	^
675	} } pop	{r4,pc}							
-0004219102		T:080036B4	ptrace	u_c\0	S_CPU_Sys <sup>-</sup>	FickHandle	er+0x20	0.120us	
-0004219097	<pre>pop</pre>	{r4,pc} T:080058F8	ptrace er"	drv_s	pi1\SPI1_	TranserByt	e+0xC8	0.600us	_
265	↓ tst beq	// Set Dat if (_Mask r2,r1 0x800590	a & _Data) 4	; _Mask,_Da ; ??SPI1_Tr	ta anserByte	16			
271	ldr movs str	<pre>} else {     SPI1_MOS     r5,0x800     r6,#0x80     r6,[r5] }</pre>	SI_L(); 9597C	; r5,??Data	Table6_4				~
	<							>	

One or more groups can be removed from the trace display windows by clicking the "hide" column in the **GROUP.List** window or by using the command **GROUP.HIDE** *<name>*.

i	ii B::GROUP.	List									×
	🗶 Reset	O Disable All	Enable All	O Hide All	Shov	v All 💈	Store	😤 Load	iii Create	🙀 Oreate Task	
	group				enable	hide	merge	color			
	■ "App1 \apj (\aj ■ "Kern (OSI "othe	ication" p_motor pp\main) el" EventNameS r"	(iPA10ISR\ et)(`??0	(12+0x1) )STmr_Ta	<			BLUE BLUE BLUE RED RED			~
					v	•					~
L											2.11

The "other" group is now suppressed from display.

B::Trace.List						x
🔑 Setup 🔃 Goto 🎁 Fi	nd 🕂 Chart 📕 Prot	ile 🛛 🔛 MIPS	🔷 More 🛛 👗 Le	ss		
record run address	cycle data	symbol		ti	. back	
113	CPSID I					^
114	BX IR					
bx	r14					× .
0001336034 T:0	)80059E6 ptrace	6\drv_	spi1\AtomicExcha	nge+0x0A	0.160us	^
gr	oup other				0.020us	
,						
	leaters		. Coo No	ta #3		
118 CP0_SK_F	MSR PRIMASK, RO		; see No	ote #2.		
msr	primask,r0					
119 hy	BX LR					
0001336032 T:0	)80059F4 ptrace	6\drv	spi1\AtomicExcha	nae+0x18	0.280us	
gr	oup "other"			<u> </u>	0.160us	
gr	oup "Kernel" —				3.385us	
		:				
	CPU_	CRITICAL_EXI	т();	/* (	CPU_SR_Re	2
						- 1
	}					
******	**************	*********	******	*********	*******	
					>	×.

# Timing

A very detailed set of analyses can be performed using the **PERF** commands ("**Analyzing the Results**", page 30) but measuring time from a significant event or between two points can be performed using the timing columns in the **Trace.List** window.

The default **Trace.List** window includes the **TIme.Back** column. This is the time taken from the last trace entry to this one. Additional timing columns can be added to any trace listing window, for example:

🧮 B::Trace.List T	IME.BACK TIME.ZE	RO DEFAULT		3
🔑 Setup 📭	🕽 Goto 🏼 🏥 Fi	nd 🔂 Chart	t 🌃 Profile 🌃 MIPS 🗢 More 🗶 Less	
record	ti.back t	i.zero ru	un address cycle data symbol	
298	0.044us	4.647ks	<pre>* Note(s) : 1) Interrupts may or may not be ENABLED during thi */ #if (OS_CPU_HOOKS_EN &gt; 0) &amp;&amp; (OS_TIME_TICK_HOOK_EN &gt; 0) void OSTimeTickHook (void) { #if OS_APP_HOOKS_EN &gt; 0 App_TimeTickHook(); #endif </pre>	~ ~
304	0.044us	4.647ks	<pre>#if OS_TMR_EN &gt; 0     OSTmrCtr++;</pre>	
305	0.220us	4.647ks	if (OSTmrCtr >= (OS_TICKS_PER_SEC / OS_TMR_CFG_TICKS_PER_SEC)	
310	0.176us	4.647ks	#endif } #endif	
863	0.070us	4.647ks	<pre>#if OS_TIME_GET_SET_EN &gt; 0     OS_ENTER_CRITICAL(); /* Upd</pre>	
	٢		<td>•</td>	•

The most commonly used timing options are listed in the table below. Other, less common, options are described in the documentation for **Trace.List**.

Tlme.Back	The time elapsed since the last entry.
TIme.Fore	The time elapsed from this entry until the next entry.
TIme.REF	The time elapsed since a user-defined reference point.
TIme.Zero	The time elapsed since the debug session started or a user-defined Zero point.
TIme.Trigger	The time elapsed since the trace trigger event.

Also, bear in mind that some times can be negative as the item may have occurred before the timing point.

Some marker points can be user-defined or moved by the user. To do this, right-click in any windows that shows a view of the captured trace data and select the appropriate marker from the pop-up menu.

🕂 B::Trace.Chart.sYmbol				
Setup iii Groups II Config	🔒 Goto 🔒 Goto	🛉 Find 🕩 In	▶ Out 🖸 Full	
addr ess 🕅	3.529260000s	-3.529250000s	-3.5	29240000s
(other) DC2016\app_adc\iTIM4ISR	n : : : : : : : : : : : : : : : : : : :			· · · · · · · · ^
DC2016\os_sem\OSSemPost 6\Global\OS_CPU_SR_Save				1111100
core\OS_EventTaskRemove	R+ Set Ref			
DC2016\os_core\OS_SchedNewN	z > Set Zero			
\DDC2016\Global\OSCtxSw	Construction Construction Construction	rk i i i	· · · · · · ·	
al\OS_CPU_PendSVHandler 🚯 PU_PendSVHandler_nosave	Q View			
6\os_cpu_c\OSTaskSwHook	List			<b>I</b> 
DC_SoftwareStartConvCmd	Chart			
X_adc\ADC_GetFlagStatus	First in Statistic	laan ahaa ah	.■	

The two images below show how setting the **TI.Zero** marker in the **Trace.Chart** window will affect the timing in the **Trace.List** window.



After setting the Zero point in the Chart window, the values in the TI.Zero column in the List window change to show the timings relative to the new Zero point.

Different markers are shown in different colors.

Green	Zero marker
Red	User-defined REF marker
Yellow	User-defined bookmarks.

All three can be seen in the picture below.



A blue line represents a user placed cursor and is used for tracking between trace view windows.

Left-clicking anywhere in the Trace.Chart window will create a pop-up which shows the timing information:



С-Т	Current time (selected by cursor) to Trigger point.
C-R	Current time (selected by cursor) to REF marker.
C-Z	Current time (selected by cursor) to Zero point.

The manual "Application Note for Trace-Based Code Coverage" (app\_code\_coverage.pdf) gives a detailed introduction to the trace-based code coverage. However, the manual does not contain details about the architecture-specific setups. Here is an overview of the setups for the ETM.



The following settings are recommended:

ETM.Trace ON	; Enable program flow trace
ETM.TImeMode External	; Enable tool timestamp
ITM.OFF	; Switch the ITM off

Once a set of trace data has been captured it is possible to walk through the history of the recording, often filling in missing details. The Context Tracking System (CTS) is used to perform these functions. CTS is accessed from the Trace menu or via the command **CTS.state**.

B::CTS.state		- • ×
state	progress —	options
OFF		UseSIM
OON		UseVM
	- warnings	UseConst
commands		UseMemory
RESet	– fifofulls –	UseRegister
⊗ Init		UseCACHE
TAKEOVER		UseReadCyde
🐯 PROCESS	Mode	<b>✓</b> UseWriteCycle
👼 List	Full	✓ SmartTrace
🔑 Trace	○ Memory	SELectiveTrace
Northe 🎾	○ CACHE	✓ INCremental

More information about trace based debugging can be found here: "**Trace-based Debugging**" in General Commands Reference Guide C, page 163 (general\_ref\_c.pdf).

Once the mode is set to ON (via the GUI or the command **CTS.ON**), the CTS system analyses the recorded trace in more detail, often filling in missing data accesses and areas where FIFO overflows occurred. This may take a few minutes depending upon the size of the captured trace and the speed of the host PC and available RAM. It will look something like this:

	PuCTS state	•		k += primz;		
	B::CTS.state	progress shown: 1.615% warnings 33719, fifofulls 0, Mode	options UseSIM UseVM UseVM UseConst UseMemory UseRegister UseCACHE UseReadCyde UseWriteCyde SmartTrace	k += primz; ++;		00000
	CACHE		☐ SELECTIVE IT AGE			00000
B::						
compo UT:1FF	FF968 \\rtosder	Data mo_pic_thumb_ii_v7r	Var List n\midi\sieve+0x4C (	PERF         SYStem           CTS)         0x200015C0	Step Go CTS (0.) busy	Bre

When CTS is active, the TRACE32 status bar and control icons in the List window change color to yellow. New buttons appear in the List window tool bar, allowing the user to step forwards or backwards through the code and to run back to a function entry point. Compare the two images below.

📰 B::List									
Step Step	🛃 Diverge	🗸 Return	Ċ Up	► Go	II Break	Mode	😹 t. 🤜	Find:	sieve.c
addr/line	source								
± 389		for (	v9 = 0	; v9 < 3	3 ; v9++	)			^
🗐 B::List									- • •
B::List	🛃 Diverge	🞸 Return	🕐 Ир	📕 Step 🗍	Over 🔨 Entry	📕 Off 🔀	66 <b>t.</b> "J	Find:	sieve.c
B:List	▲ Diverge source	🞸 Return	と Up	K Step 👫	Over 🔨 Entry	Off 🕅	t. "J	Find:	sieve.c
B:List	L Diverge Source	✓ Return	<u>د</u> له vshor	K Step 4	Over hEntry	Off 🎬	trecord)	Find:	sieve.c

Stepping backwards or forwards through the code shown in the List window will cause any register, memory or variable display windows to be updated to show the re-constructed values as they would have been at that point in history.

The right-click menu also has new options added to allow users to go forwards or backwards to a point or run forwards or backwards to an access on a variable.

📰 [B::List]				(B::List)		
📕 Step	📙 Over	🛃 Diverge 🛛 🎸 Return 👘 🕐 Up	p 🛛 📙 Step 🗍 Over 🔩 Entry 📃 Off 🕈	Step	📙 Over 📊	🚣 Diverge 🛛 🎸 Return 🛛 🕐
ad	ldr/line	source		a	ddr/line	source
					738	count = 0;
÷	740	for ( i = 0	; i <= SIZE ; flags[ i-	•	740	for $(i = 0;$
Đ	742		; <= SIZE; i++) {			
	743	Program Address	ags[i]) {	÷	742	for (i = 0; i
	/44	GO Back Thi	prime = i + i + 3;		/43	if (flag
	/45	👱 Go Till	k = i + prime;		/44	prime
÷	/46	👘 Breakpoint	while (k <= SIZE)		/45	k = 10
	/4/	Breakpoints	flags[k] =	±	/46	whi lea
	748	Display Memory	K += prime;		747	6
	750		3 countries		740	, <sup>1</sup> 6
	730	BOOKmark	counc++;		750	}
		Toggle Bookmark			730	Count
		🚓 Set PC Here				, <i>s</i>

Clicking the List button in the **CTS.state** window opens the **CTS.List** window. This shows a nested view of the code. Where possible, arguments and return values have been reconstructed using the current state of the target and the captured trace data. Timing for each node is represented in the ti.back column. Each node can be opened or closed to show more detail. Where a piece of data cannot be reconstructed a series of ??? will appear in all windows that show CTS data.

😽 B::CTS.List				×
🔑 Setup 🥻 CT	'S 🔃 Goto 🎒 Find 📮 TREE 🛛 😽 Chart 🛛 😾 Chart 🔶 More 🛛	Less		
record			ti.back	
				^
	$C = \frac{2}{2}$			
	return = 11266			~
-0062822147	func13+0x56	A ¥	- 0.402us	^
-0062822145	func13+0x56	A <b>v</b> —	- 1.039us	
-0062822143		A <b>v</b> —	- 1.719us	
-0062822140	return = ??? func13+0x56	A <b>v</b>	- 2.532us	
-0062822139 -0062822090	sieve+0x5C		- <b>7.367us</b> - 7.367us	
-0062822090	group "RTOS"freentos\tasks_c\1761		- 0 112us	-
-0062822086	TaskGetTickCount+0x6	Ā ¥ —	- 0.112us	
-0062822083	group "other"		- 0.112us	
-0062822083	func1 midi.c \1	A ¥ —	- 0.222us	
-0062822079	100010101 100010101 100010101	A <b>v</b> —	- 0.222us	
-0062822079 🕀	midi.c \1	* <b>*</b>	- 0.231us	
-0062822076	func1+0x1A	<b>. .</b>	- 0.231us	
-0062822069 🕀		A <b>V</b>	- 0.107us	
-0062822065			- 0.107us	~
<			>	

In any trace display window, the right-click menu has a Set CTS option. Selecting this will cause the current state to jump to that point in history with as much of the context reconstructed as possible. This allows the use of high level tools such as chart windows to locate an issue and then zoom in to see in detail what occurred at that time.



Once CTS is active, all trace analysis features will work with the reconstructed data. This is often better than the captured data as any gaps may be filled in but will be no worse.

A set of trace data can be saved for later, off-line analysis. To save the trace data, use **Trace.SAVE** <*file*>. In most cases only the /ZIP option is useful. Saved trace data files have the extension ".ad".

Trace files can be re-load with Trace.Load <file>.

Normal trace analysis features work exactly as if they were using a recently captured "live" trace. The only difference is that the trace display windows now contain a red "LOAD" mnemonic. These can be seen in the image below.



A more detailed analysis may often be achieved if the contents of the RAM are saved along with the CPU registers at the point where the trace had finished being captured. This can be done like this:

```
Data.SAVE.Binary <file> <address_range>
STOre <file> Register
```

TRACE32 software can also be installed or configured as an instruction set simulator. In this way the saved trace can be analyzed off-line.

## To analyze saved trace data:

- 1. Start TRACE32 in simulator mode, set the CPU type and system mode to UP.
  - SYStem.CPU
  - SYStem.Up
- 2. Load the saved RAM contents.
  - Data.LOAD.Binary <file> <address>
- 3. Load the saved CPU registers
  - DO <save\_reg\_file>
- 4. Load application symbols
  - Data.LOAD.Elf <symbol\_file> /NoCODE
- 5. Load the saved trace file

- Trace.Load <file>.ad

# **Data Watchpoint and Trace Unit**

- The DWT provides some useful additions to the basic program flow trace, although not all will be covered here as we will focus only on those features that provide a trace like output.
- PC Sampler
- Data Trace capability
- Interrupt Trace
- ETM Trigger

The block diagram below shows how the DWT is organized to inject trace packets into the ITM stream for eventual spooling to the TPIU and then to the trace port (ETM or SWV). However, if the TPIU is configured for SWV then it will not transmit ETM packets; only the ITM packets will be transmitted.



Each trace source is given a unique 7-bit Trace ID (ATID) so that the streams can be separated by the debug tools.

For details on time stamping please refer to "Time Stamping", page 72.

This feature is not a true program flow trace in the way that MTB or ETM will allow full capture of program flow. The PC Sampler will periodically sample the Program Counter and emit the address as a trace packet to the ITM. The sampled PC is emitted via the ITM which makes this technique ideal for systems which only support the ITM or Serial Wire Viewer for trace as it can provide some data about program flow. The program flow may not be 100% accurate; items that can complete in less than the selected number of cycles to sample may not show in the data at all. The PC can be sampled every 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 or 32768 clock cycles.

This feature is configured in the **ITM.state** window. There is no separate DWT configuration window as the trace events from the DWT require the ITM block to also be present on the device or they cannot be emitted. Since the two are so tightly linked, the **ITM.state** window allows the user to configure both the ITM and DWT blocks of the Cortex-M device.



Set the frequency of the PCSampler (via the GUI or the **ITM.PCSampler** command) and enter the value of the CPU clock to enable timing of the data.

Run the target to generate and sample the data then use the command Trace.List to show the results.

🔢 B::Trace.List								
🔑 Setup	🔒 Goto	🛉 Find 🚺 Chart	👗 Profile	K MIPS	More	Less		
recor	d run a	address	cycle	data	symbol			ti.back
-0107982	1	T:08003298	fetch		\\DDC2	016\os_	_core\05_TaskIdle+0x18	7.100us
-0107981	6	T:080035EC	fetch		\\DDC2	016\os_	cpu_c\0STaskId]eHook	7.120us
-0107981	0	т:08003544	fetch		\\DDC2	016\Glo	ba1\0S_CPU_SR_Save+0x4	7.120us 🗸
-0107980	5	т:08003292	fetch		\\DDC2	016\os_	_core\05_TaskIdle+0x12	7.100us
-0107979	9	т:08003290	fetch		\\DDC2	016\os_	_core\05_TaskIdle+0x1C	7.120us
-0107979	3	т:08003286	fetch		\\DDC2	016\os_	.core\05_TaskIdle+0x6	7.100us
-0107978	8	т:08003290	fetch		\\DDC2	016\os_	_core\05_TaskIdle+0x10	7.120us
-0107978	3	т:0800354с	fetch		\\DDC2	016\Glo	bal\0S_CPU_SR_Restore+0x4	7.100us
-0107977	5	T:080032A0	fetch		\\DDC2	016\os_	_core\05_TaskId1e+0x20	7.120us
-0107977	0	T:0800328C	fetch		\\DDC2	016\os_	core\0S_TaskIdle+0x0C	7.100us
-0107976	5	T:0800354C	fetch		\\DDC2	016\Glo	bal\OS_CPU_SR_Restore+0x4	7.120us
-0107975	9	T:080032A0	fetch		\\DDC2	016\os_	_core\05_TaskIdle+0x20	7.120us
-0107975	4	T:0800328A	fetch		\\DDC2	016\os_	_core\05_TaskIdle+0x0A	7.100us
-0107974	9	T:08003298	fetch		\\DDC2	016\os_	core\05_TaskId]e+0x18	7.120us
-0107974	3	T:080035EC	fetch		\\DDC2	016\os_	cpu_c\0STaskId1eHook	7.100us
-0107973	8	T:08003544	fetch		\\DDC2	016\Glo	ba1\0S_CPU_SR_Save+0x4	7.120us
-0107973	3	T:08003292	fetch		\\DDC2	016\os_	core\05_TaskIdle+0x12	7.100us
-0107972	7	T:0800329C	fetch		\\DDC2	016\os_	core\05_TaskIdle+0x1C	7.120us
-0107972	2	T:08003286	fetch		\\DDC2	016\os_	core\0S_TaskIdle+0x6	7.100us
-010/9/1	5	T:08003292	fetch		\\DDC2	016\os_	core\05_TaskIdle+0x12	7.120us
-010/9/0	9	T:0800329C	fetch		\\DDC2	016\os_	core\05_TaskIdle+0x1C	/.100us
-010/9/0	4	T:080032A0	fetch		\\DDC2	016\os_	core\05_TaskIdle+0x20	/.120us
-010/969	9	T:0800328C	tetch		\\DDC2	016\os_	core\0S_TaskIdle+0x0C	/.100us
-0107969	3	T:0800354C	tetch		\\DDC2	016\Glo	bal\OS_CPU_SR_Restore+0x4	/.120us
-0107968	8	T:080032A0	fetch		\\DDC2	016\os_	_core\05_TaskId1e+0x20	/.120us v
	<							> a

# Clicking the Chart button or using the command Trace.Chart.sYmbol will show the functions against time and look like this.

N B::Trace.Chart.sYmbol					
	Goto ♀Goto ♠Find ♦P In ♦D•Out	🖸 Full			
	-2.000s	-1.500s	-1.000s	-500.000ms	0.0
address 🛛					· · · · · · · · · · · · · · · · · · ·
\os_cpu_c\OSTaskStkInit					^
DDC2016\os_core\OSStart					A
tms2T10x_gp10\GP10_Init					2 - 2
DC_GetCalibrationStatus					
6) ctm22f10x tim) TIM Cmd f					a a a a a a a a a a a a a a a a a a a
\\ppc2016\app.adc\ <b>tApc</b>		1. 1. 11 - 21 - 21	- 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1	(3) 2 (2) (2) (2)	2 - 2 p - 1
016\os_core\OS_SchedNew/N		1	i hari i	har an traite da har -	<ul> <li>ini in</li> </ul>
016\app_motor\ConfigPWM			,	10, 0 , 1 C, 0, 0, 0	· · · · ·
RCC APB1PerinhResetCmd					1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 -
s core\OS EventTaskWait					
RCC_APB2PeriphClockCmd					
tm32f10x_exti\EXTI_Init					
2f10x_gpio\GPIO_SetBits	"ען המתהקות או עלים אי היא אי אור אי אי אי אי אי אי אי אי אי אי אי אי אי				
v_glcd\GLCD_PowerUpInit					
C2016\drv_glcd\Dly100us 🕦		<b>—</b>			
2016\os_core\OSTimeTick					
lobal OS_CPU_SR_Restore		<u> </u>			
C2016\os_core\OSIntExit					
O GIODAI CPU_SR_Restore					2 2
1/CLCD_SPT_PassivePlack					2
6\dpy_glcd\CLCD_SendCmd4	· · · · · · · · · · · · · · · · · · ·	1 2 2 2 2			d
11\GLCD_SPT_TranserByte					C
dry spil AtomicExchange					1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 -
11\GLCD_SPT_ChipSelect					a a 1
rv spil\SPI1 ChipSelect					2 - 2 - 1
d_11\GLCD_SPI_SendBlock					2 - 2 - 1
<\OS_CPU_SysTickHandler		NAN (ANALASI), KANAN (ANAMAN), MANANJARI	. 10 191010001(10111-11) - 0010 (01-0	ninim in nim i' nüilinininim mi ninimini	(000000)
2016\Global\CPU_SR_Save					
spil\SPI1_ChipDeselect					
os_cpu_c\OSTimeTickHook		inin ilmininin li ilmininin ilminin			I, BUBLIQUE BI
10x_gpio\GPIO_ResetBits					
X_adc ADC_GetF TagStatus					
DC2016\os_sem\OSSemPost		14 HULL - LUC IL.L			CLINE IL
BU BondSyllandlon posavo					a a 4
C2016\os_tmp\OSTmp_Lock		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	'qu		e e e
6\os_task\0STaskSuspend					2 - 2 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 -
os cpu c\OSTaskIdleHook					· · · ·
016\os_core\05_TaskIdle					
al\OS_CPU_PendSVHandler					
core\OS_EventTaskRemove					0 1 1
os_core\OS_EventTaskRdy					0 0 11
C2016\Global\OSIntCtxSw					
\DDC2016\Global\OSCtxSw					· · · · · · · · · · · · · · · · · · ·
J	< > <				ار. <

A D. Trans Chart Mark

One of the features of the DWT is to be able to inject data trace events into the ITM for inclusion into the trace stream. The data events are configured using the TRACE32 breakpoint interface. More information can be found here: "On-chip Breakpoints", page 84.

Set the ITM.DataTrace mode.



Several modes are supported:

OFF	No Data Trace information will be generated.
Address	If the data address matches a DWT comparator, address information about the data access will be emitted as a trace packet.
Data	If the data address matches a DWT comparator, the data value will be emitted as a trace packet.
ON	If the data address matches a DWT comparator, address and data information about the data access will be emitted as a trace packet.
DataPC	If the data address matches a DWT comparator, address and data information about the data access will be emitted along with the address of the instruction that issued the data access.
OnlyPC	If the data address matches a DWT comparator, address of the instruction that performed the data access will be emitted.
CorrelatedData	Produces the same information as DataPC but the streams (ETM & ITM) are merged in TRACE32.

To generate data trace, set a TraceData breakpoint on the value to be traced.

🙆 B::Break.Set			- 🗆 ×
address / expres	ssion ————		✓ I □ HLL
type Program ReadWrite Read Write default	options EXclude NOMARK DATA	Temporary DISable DISableHIT	implementation auto action TraceData V advanced
Ok	Add	Delete	Cancel

Use: Break.Set ADC\_Val /Write /TraceData

Run target to collect data. Show results:

B::Trace.List	t								×
🔑 Setup	ſ↓ Go	to 👸 Find	Chart	🔼 Profile	K MIPS	More	Less		
record	run	n address	cycle	data	symbol			ti.back	
000000264		D:20002	2600 wr-lon	g 00000FFF	\DDC20	16\Global	\ADC_Val	25.486ms	^
000000235		D:20002	2600 wr-lon	g 00000E04	\\DDC20	16\Global\	\ADC_Val	407.773ms	
000000228		D:20002	2600 wr-lon	g 00000C53	\\DDC20	16\Global\	\ADC_Val	25.486ms	
000000221		D:20002	2600 wr-lon	g 00000A49	\\DDC20	16\Global\	\ADC_Val	25.486ms	~
000000213		D:20002	2600 wr-lon	g 00000884	\\DDC20	16\Global\	\ADC_Val	25.486ms	^
0000000206		D:20002	2600 wr-lon	g 00000712	\\DDC20	16\Global\	\ADC_Val	25.486ms	
0000000190		D:20002	2600 wr-]on	g 000005E5	\\DDC20	16\Global	ADC_Va	25.485ms	
0000000183		D:20002	2600 wr-]on	g 00000489	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000176		D:20002	2600 wr-]on	g 0000035F	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000168		D:20002	2600 wr-]on	g 000002A1	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000161	.	D:20002	2600 wr- <u>l</u> on	g 00000218	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000154		D:20002	2600 wr- <u>l</u> on	g 000001C0	\\DDC20	16\Global	\ADC_Va]	25.491ms	
0000000146		D:20002	2600 wr- <u>l</u> on	g 0000014E	\\DDC20	16\Global	\ADC_Va]	25.481ms	
0000000139		D:20002	2600 wr- <u>l</u> on	g 0000011A	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000132		D:20002	2600 wr- <u>l</u> on	g 00000102	\\DDC20	16\Global	\ADC_Va]	25.485ms	
0000000116		D:20002	2600 wr- <u>l</u> on	g 000000F0	\\DDC20	16\Global	\ADC_Val	25.486ms	
0000000109		D:20002	2600 wr- <u>l</u> on	g 000000CF	\\DDC20	16\Global	\ADC_Val	25.486ms	
0000000101	.	D:20002	2600 wr- <u>l</u> on	g 000000AD	\\DDC20	16\Global	\ADC_Val	25.486ms	
0000000094		D:20002	2600 wr-lon	g 00000088	\\DDC20	16\Global	\ADC_Val	25.485ms	
000000087		D:20002	2600 wr-lon	g 0000007B	\\DDC20	16\Global	\ADC_Val	25.486ms	
0000000079		D:20002	2600 wr-lon	g 00000062	\\DDC20	16\Global	\ADC_Val	25.486ms	
000000072		D:20002	2600 wr-]on	g 00000045	\\DDC20	16\Global	ADC_Va	25.486ms	
000000065		D:20002	2600 wr- <u>l</u> on	g 00000036	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000057		D:20002	2600 wr- <u>l</u> on	g 00000011	\\DDC20	16\Global	\ADC_Va]	25.486ms	
0000000042		D:20002	2600 wr-lon	g 00000000	\\DDC20	16\Global\	\ADC_Val	25.486ms	×
	<							>	

The traced data values can be shown as a graph as the change against time. Use the command

Trace.DRAW.Var %DEFault ADC\_Val

#### to show this:

Setup	🔒 Goto	)	<u>iji</u> P	-ind			C	nart			In	•	<b>1</b> 0	ut	٩.	D Fi	ull	¢	) In		ž0	ut	\$	Ful	1																
		00s				- 5	. 0	00s					-4	1.0	000	s				- 3	. 00	00s					- 2	. 00	00s				-	1.	00	0 s					0
			• •		•		-							L .							<u> </u>					-		L.,				• •		.					• •		
	3500.																																								
	3000.																																								
	2500.																																								
	2000.																																								
	1500.																																								
	1000.																																								
	500.	$ \cdot _{\sim}$	~~~	~~	v		$\sim$	ŕv	$\sim$	v	W	ĸ	~	~	Λ.	~	W	M	~	w	ŵ	m	~	,	Ŵ	ý	ŝ	2	św	٨.	,N	$\sim$	W	_	~~	~~	$\sim$	~/	\	~	r٧
	0.																								· ·																

To see where in the code a data access came from:

- 1. Switch ITM.DataTrace DataPC
- 2. Set the breakpoint to ReadWrite
  - Break.Set <var>/ReadWrite /TraceData
- 3. Run the target to collect the trace samples.

The results can be shown filtered by any kind of access: read, write or any. The command **Trace.STATistic.PsYmbol** is used.

## To show all accesses: Trace.STATistic.PsYmbol /Filter sYmbol <var>

Setup iii Groups II Config      G Goto      F Detailed      F Tree      Chart      Profile     Config     Config	
items: 5. total: 2.0985 Samples: 10906.	100
address total min max avr count rations 1% 2% 5% 10% 20% 50% (other) 1.500us 1.500us 1.500us - 0. <0.01% ← √/sieve\initinkedist 302.06ims 81.920us 83.960us 83.075us 3636. 11.195%	100
m_thumb_ii_v7\sieve\main 2.396s 9.420us 650.280us 329.476us 7272. 88.804%	-
	>

#### To show only read accesses: Trace.STATistic.PsYmbol /Filter <var> Cycle READ

E BulTMCAnalyzer.STATistic.PsYmbol /Filter sYmbol ast Cycle READ	- • •
🤌 Setup iii Groups 😫 Config 🔒 Goto ₹ Detailed 🛒 Tree 🙀 Chart 📕 Profile	
items: 2. total: 2.698s samples: 3636.	
address total min max avr count iratio% 1% 2% 5% 10% 2	20% 50% 100
(other) 83,460us 83,460us 83,460us - 0. 0.003%	^
ELTIMO 11_V/STEVE (Main 2.0305 / 41.42005 / 45.00005 / 42.00505 3030. 33.337%)	~
	> .:i

#### To show only write accesses: Trace.STATistic.PsYmbol /Filter <var> Cycle WRITE

E B::ITMCAnalyzer.STATistic.PsYmbol /Filter sYmbol ast Cycle WRITE	_ • •
🤣 Setup 👖 Groups 👪 Config 🔃 Goto 🗾 Detailed 🛒 Tree 🔥 Chart 🔛 Profile	
items: 3. total: 2.698s samples: 7272.	
address total imin imax lavr icount iratio% 1% 2% 5% 10% 20%	50% 100
(other) 1.500us 1.500us 1.500us - 0. <0.001% +	<b>^</b>
Lingthe Lingth L	
	×
	). ji

If your design uses an RTOS, task switches can be traced. If the RTOS is supported by a TRACE32 OS Awareness you can use the address **TASK.CONFIG(magic)**. If your RTOS is not supported by TRACE32 you can use the address of the variable that holds the currently executing thread ID.

Configure ITM.DataTrace Data and set a breakpoint (Break.Set TASK.CONFIG(magic) /Write /TraceData) and run the target to collect trace samples.



A list of all writes (all task switches) can be seen with ITMTrace.List.

B::ITMTrace	e.List							
🔑 Setup	🔒 Goto	👸 Find	Chart	📕 Profile	🔼 MIPS	More	Less	
recor	d run add	ress	cycle	data	symbol			ti.back
-000000183	31	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\OSTCBCur	15.460us 🔺
-000000181	.8	D:200026	5C wr-lon	g 20000B4C	\\DDC20:	16\Global'	\OSTCBCur	25.445ms
-000000172	20	D:200026	5C wr-lon	g 20000BA8	\\DDC20:	16\Global'	\OSTCBCur	24.880us
-000000160	)4	D:200026	5C wr-lon	g 20000C60	\\DDC20:	16\Global'	\OSTCBCur	15.680us 🎽
-000000158	33	D:200026	5C wr-lon	g 20000B4C	\\DDC20:	16\Global'	\OSTCBCur	25.445ms ^
-000000148	32	D:200026	5C wr-lon	g 20000BA8	\\DDC20	16\Global	\0STCBCur	25.440us
-000000137	0	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\0STCBCur	15.560us
-000000135	58	D:200026	5C wr-lon	g 20000B4C	\\DDC20:	16\Global	\0STCBCur	25.445ms
-000000126	52	D:200026	5C wr-lon	g 20000BA8	\\DDC20:	16\Global	\0STCBCur	24.900us
-000000114	9	D:200026	5C wr-lon	g 20000C60	\\DDC20:	16\Global	\0STCBCur	15.440us
-000000113	36	D:200026	5C wr-lon	g 20000B4C	\\DDC20:	16\Global	\0STCBCur	25.445ms
-000000104	0	D:200026	5C wr-lon	g 20000BA8	\\DDC20:	16\Global	\0STCBCur	25.440us
-000000092	23	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\0STCBCur	15.220us
-000000091	0	D:200026	5C wr-lon	g 20000B4C	\\DDC20	16\Global	\0STCBCur	25.445ms
-000000081	4	D:200026	5C wr-lon	g 20000BA8	\\DDC20	16\Global	\0STCBCur	24.880us
-000000070	02	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\0STCBCur	15.340us
-000000069	90   0	D:200026	5C wr-lon	g 20000B4C	\\DDC20	16\Global	\0STCBCur	25.446ms
-000000058	38	D:200026	5C wr-lon	g 20000BA8	\\DDC20	16\Global	\0STCBCur	24.900us
-000000047	6	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\OSTCBCur	15.660us
-000000046	54	D:200026	5C wr-lon	g 20000B4C	\\DDC20	16\Global	\OSTCBCur	25.445ms
-00000036	54	D:200026	5C wr-lon	g 20000BA8	\\DDC20	16\Global	\0STCBCur	24.900us
-000000024	5	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\OSTCBCur	15.560us
-000000023	32	D:200026	5C wr-lon	g 20000B4C	\\DDC20	16\Global	\OSTCBCur	25.445ms
-000000013	32	D:200026	5C wr-lon	g 20000BA8	\\DDC20	16\Global	\OSTCBCur	25.440us
-000000001	.8	D:200026	5C wr-lon	g 20000C60	\\DDC20	16\Global	\OSTCBCur	15.560us 🗸
	<							≥

The ti.back column shows how long each task or thread was running before it was switched out for a new one.

# A more detailed view can be made using: ITMTrace.List List.TASK DEFault

B::ITMCAnalyzer.List List.TASK DEFault	- • •
🔑 Setup 📭 Goto 👘 Find 👬 Chart 🛛 Profile 🖉 MIPS 🖨 More 🗶 Less	
record run address cycle data symbol	ti.back
-0000000923 D:2000265C wr-long 20000C60 \\DDC2016\Global\OSTCBCur	15.220us 🔥
TASK magic = 20000B4C, name = ADC Reader	
-0000000910 D:2000265C wr-long 20000B4C \\DDC2016\Global\OSTCBCur	25.445ms
TASK magic = 20000BA8, name = Motor Ctrl	
-0000000814 D:2000265C wr-long 20000BA8 \DDC2016\Global\OSTCBCur	24.880us ^
IASK magic = 20000C60, name = LCD Disp $$	15 240
	15.340us
TASK magic = 2000084C, name = ADC Reader	25 446mg
TASK majo = 20000848 name = Motor Ctr ] ===	23.440005
-000000588 D D:2000265 wr-long 20000848 \DC2016\Global\05TCBCur	24 900us
TASK magic = 20000C60 name = LCD Disp	21.50005
-0000000476 D:2000265c wr-long 20000c60 \\DDC2016\Global\OSTCBCur	15.660us
TASK magic = 20000B4C, name = ADC Reader	1000000
-0000000464 D:2000265C wr-long 20000B4C \\DDC2016\Global\OSTCBCur	25.445ms
TASK magic = 20000BA8, name = Motor Ctrl	
-0000000364 D:2000265C wr-long 20000BA8 \\DDC2016\Global\OSTCBCur	24.900us
TASK magic = 20000C60, name = LCD Disp	
-0000000245 D:2000265c_wr-long 20000C60 \\DDC2016\Global\OSTCBCur	15.560us
TASK magic = 20000B4C, name = ADC Reader	25.445
-0000000232 D:2000265C wr-long 2000084C \DDC2016\Global\OSTCBCur	25.445ms
IASK magic = 20000BA8, name = Motor Ctrl	25 440
	25.440US
	15 560uc M
Dizobizote wi - Tong zoobcoo (\bbczote (dibbal\ositeBcur	13.300u3 V
	×

# To view how tasks or threads changed over time, use: Trace.Chart.TASK

B::Trace.CHART.TASK											×
🔑 Setup 👔 Groups	🔡 Config	Goto	🗘 Goto 🏼 🏥	Find	<b>∙⊡•</b> In	•⊡• Out	🖸 Full				
0	00s	-5.000s	-4.00	00s	-	-3.000s		-2.000s	-1.000s	(	0.0
r ange 🕕											
(unknown) 🖬 ADC Reader 🖸				Từ từ từ từ từ t	Lininini		núnúnún				III ^
LCD Dispo uC/OS-II Tmr			00000000	0.000							
uC/OS-II Stat	 . <b>.</b>				<b>U</b> ,	· · · ·					
MOLOF CLFT	. (0.0000000000000000000000000000000000	IIII			(ru(ni)ni				(0.0.00000000	Щ	~
	< > <										>

# To view run time information for each task, use Trace.STATistic.TASK

B::Trace.STATistic.	TASK												×
🔑 Setup 🎁 Gro	Setup       iii Groups       Config       Detailed       Nesting       Profile         tasks:       7.       total:       5.760s       1												
range (unknown) ADC Reader LCD Disp uC/OS-II Tmr uC/OS-II Stat	total 0.000us 4.676ms 4.058s 602.159us 7.100us	min 0.000us 17.100us 547.220us 13.660us 7.100us		avr 20.510us 25.048ms 31.693us 7.100us	count 0. 228. 162. 19.	ratio% 19 0.000% 0.081% + 70.443% 0.010% + <0.001% +	6 2%	5%	10%	20%	50%	100	^
uC/OS-II Idle Motor Ctrl	1.696s 1.329ms	653.440us 12.880us	25.475ms 25.340us	19.952ms 15.631us	85. 85.	29.441% 0.023%						>	> 11

The DWT can be programmed to generate trace events for each interrupt entry and exit point.

Select InterruptTrace from the ITM configuration window or use the command ITM.InterruptTrace ON.

To reduce the likelihood of internal trace FIFO overflows, switch off all other trace sources (ProfilingTrace, DataTrace and PCSampler). Run the target to sample the trace data. The results can be displayed using the **Trace.List** command and look like this.

B::ITMTrac	e.List										x
🔑 Setup	🔒 Got	o	Find	Chart	📕 Profile	MIPS	More	Less			
recor	d ru	n a	dress		cycle	data	symbo			ti.back	
-0000590	1				return	000	)			0.040us	^
-0000589	6		т:С	8007960	entry	002	E \\DDC2	2016\app	_adc\iTIM4ISR	988.600us	
-0000589	3		т:С	8007960	exit	002	E \\DDC2	2016\app	_adc\iTIM4ISR	7.340us	~
-0000589	0		т:С	800357A	entry	000	16\G	loba1\05	_CPU_PendSVHandler	0.100us	~
-0000588	7		т:С	800357A	exit	000	16\G	loba1\05	_CPU_PendSVHandler	1.120us	
-0000588	3		т:С	8003694	entry	000	.os_cp	ou_c\0S_	CPU_SysTickHandler	0.100us	
-0000588	0		т:С	8003694	exit	000	a.os_cp	ou_c\0S_	CPU_SysTickHandler	9.680us	
-0000587	7				return	000	) .		-	0.040us	
-0000587	4		т:С	800357A	entry	000	16\G	loba1\05	_CPU_PendSVHandler	15.960us	
-0000587	1		т:С	800357A	exit	000	16\G	loba1\05	_CPU_PendSVHandler	1.100us	
-0000586	7				return	000	0			0.060us	
-0000586	2		т:С	8003694	entry	000	a.os_cp	ou_c\0S_	CPU_SysTickHandler	966.480us	
-0000585	9		т:С	8003694	exit	000	a.os_cp	ou_c\0S_	CPU_SysTickHandler	9.560us	
-0000585	6				return	000			-	0.120us	
-0000585	1		т:С	8003694	entry	000	a.os_cp	ou_c\0S_	CPU_SysTickHandler	990.080us	
-0000584	3		т:С	8003694	exit	000	os_cr	ou_c\0S_	CPU_SysTickHandler	9.560us	
-0000584	0				return	000	o – .		-	0.100us	
-0000583	5		т:С	8003694	entry	000	a.os_cp	ou_c\0S_	CPU_SysTickHandler	990.220us	¥
	<									>	

The ti.back column shows the time from the previous sample to this one, so iTIM4ISR was running for 7.340 us. – this is the difference between the entry marker and the exit marker. Clicking the Chart button (or using the command **Trace.Chart.sYmbol**) will show a graphical representation of interrupt nesting against time and may look like this. The "other" row is any code executing on the target that is not an interrupt service routine.

B::Trace	B::Trace.Chart.sYmbol											
🔑 Setup.	iii Groups	Config	<b>Q</b> Goto	🔒 Goto	👸 Find	<b>∙⊡• I</b> n	•⊡• Out	🖸 Full				
		774	50000s		- 3	.97740	0000s			- 3	.9773	
L	ado	ress		·								
DC2016\a	ot app_adc\iTIM	4ISR 🔂									^	
	PU_PendSVHar	idler 🚯										
C (US_CPC	Systickhai										· · ·	
		<	> <								ي. <	

Adding a breakpoint to generate a data trace packet on task switches will allow us to analyze which tasks or threads were interrupted. Set:

```
ITM.InterruptTrace ON
ITM.DataTrace Data
Break.Set TASK.CONFIG(magic) /Write /TraceData
```

## The results can be displayed with: chart.TASKVSINTERRUPT



# Or with Trace.STATistic.TASKVSINTERRUPT

B::ITMCAnalyzer.STATistic.TASKVSINTERRUPT	BITMCAnalyzer.STATistic TASKVSINTERRUPT												
🔑 Setup 👖 Groups 🚦 Config 🛒 Detaile	🔑 Setup 📅 Groups 🔛 Config 🛒 Detailed 🐺 Nesting 🙌 Chart 🛛 🔛 Profile												
funcs: 11. total: 2.696s													
range	total 1	nin	max	avr	count	intern%	1% 2%	5%	10%	20%	50%	100	
(none)@(unknown)	110 570	1 100	1 100	1 1 2 8	107 (1 (1)	0.000%							^
(none)@ADC Reader	2.668ms	1.100us	2,668ms	- 1.120US	-	0.093%	<b>*</b>						
bal\OS_CPU_PendSVHandler@Motor Ctrl	114.980us	1.040us	1.180us	1.116us	104.(1/1)	0.004%	÷						
(none)@Motor Ctrl	1.618ms	1_0000	1.618ms	1 15700	106 (1/0)	0.055%	+						
(none)@LCD Disp	2.6925	1.000us	2.692s	- 1.13/US	-	98.847%	•					_	
pu_c\OS_CPU_SysTickHandler@LCD Disp	25.890ms	9.540us	11.120us	9.617us	2692.	0.960%	+						
\\DDC2016\app_adc\iTIM4ISR@LCD Disp	778.041us	7.320us	7.460us	7.410us	105.	0.028%	<b>+</b>						
_c\OS_CPU_SysTickHandler@Motor Ctrl	19.220us	9.560us	9.660us	9.610us	2.	<0.001%	4						
													$\sim$
-	<												×

The DWT contains an ETM Trigger capability. This is hidden from the user and is accessed using the /TraceTrigger breakpoint type. More information can be found in:

"Trace Filtering", page 24

"Trace Control by Filter and Trigger" in Training Arm CoreSight ETM Tracing, page 91 (training\_arm\_etm.pdf)

The ITM organizes trace from three main areas:

- 1. Software Generated Trace
- 2. Integrates packets from DWT into the trace stream
- 3. Generates timestamp packets for insertion into the trace stream

If packets arrive simultaneously from more than one of these sources, the ITM is responsible for arbitration and prioritizes them in the order represented by the list above.

The ITM is an optional component and may not be included in all designs (a read to a non-existent ITM register will return 0x00000000). It also requires a TPIU to output any data.

Here is a list of important ITM registers. All registers are fully accessible in Privileged mode and all registers can be read in user mode. If the corresponding bits in **ITM\_TPR** are set then user mode also has write access to **ITM\_STIM[31..0]** and **ITM\_TER**.

0xE0000FF0 - 0xE0000FFC	Component Identification Registers (CID[30])
0xE0000FD0 - 0xE0000FEC	Peripheral Identification Registers (PID[70])
0xE0000FB0	Lock Access Register (ITM_LAR)
0xE0000E80	Trace Control Register (ITM_TCR)
0xE0000E40	Trace Privilege Register (ITM_TPR)
0xE0000E00	Trace Enable Register (ITM_TER)
0xE0000000 - 0xE000007C	Stimulus Port Registers (ITM_STIM[310])

Before the ITM can be used it must be enabled and unlocked. TRACE32 will enable the ITM automatically by setting the **TRCENA** bit in the Debug Exception and Monitor Control Register.

🛷 🕮	per , "Core	Registers (Cort	ex-M3),Debug"				- • ×	
E De	ebug						~	ς
	Core D	ebug						
	DHCSR	00030003	DBGKEY	0	DBG	SKEY	0	
			DBGKEY	0	DBC	JKEY	0	
			S RESET ST/DRCKEV	No reset	· S R	SETTRE ST/DRCK	(EV Not read	11
			DBGKEY	0	DBG	GKEY	0	
			DBGKEY	ō	DBG	KEY	ō	
			S_LOCKUP/DBGKEY	Not runn	ning S_S	SLEEP/DBGKEY	Not sleeping	
			S_HALT/DBGKEY	Halted	S_R	REGRDY/DBGKEY	Available	
			C_SNAPSTALL	Disabled	1 C_M		Not masked	
				Fnabled	С_н	IAL I	Halled	
	DCRSR	XXXXXXXX	REGWNR	Enabred	REG	SEL		
	DCRDR	02000000						
	DEMCR	010007F1	TRCENA Enab	led	MON_REQ	0		
		-	NOT_TERD NOT	penorng	MON_EN	Disabled	VC_HARDERR Enabled	
			VC_INTERK Enab	led	VC_BUSER	RR Enabled	VC_STATERK Enabled	
			VC_CORFRESET Enab	led	VC_NOCFL	INN LINDIEU	VC_MMERK Enabled	
÷	Flash	Patch and	Breakpoint Unit (F	PB)			~	1
<							>	
1								4

Unlocking is done by writing **0xC5ACCE55** to **ITM\_LAR**.

Writing a value to any of the ITM stimulus ports causes a data packet to be generated and fed to the TPIU for inclusion into the trace stream. First, the ITM must be unlocked. An example is shown below.

The Cortex-M CMSIS includes functions for writing data to the ITM but macros are more efficient and reduce the overhead of using the ITM at runtime.

```
static volatile unsigned int *ITM BASE =
                   (volatile unsigned int *)0xE0000000;
#define ITM ENABLE ACCESS { ITM BASE[0x3EC]=0xC5ACCE55; }
#define ITM TRACE PRIV
                           ITM_BASE[0x390]
#define ITM TRACE ENABLE
                           ITM BASE[0x380]
#define ITM TRACE D8( channel , data ) { \
  volatile unsigned int *_ch_=ITM_BASE+(_channel_); \
 while (* ch == 0); \setminus
  (*((volatile unsigned char *)(_ch_)))=(_data_); \
}
#define ITM_TRACE_D16(_channel_,_data_) { \
  volatile unsigned int *_ch_=ITM_BASE+(_channel_); \
 while ( *_ch_ == 0); \
  (*((volatile unsigned short *)(_ch_)))=(_data_); \
}
#define ITM_TRACE_D32(_channel_,_data_) { \
  volatile unsigned int * ch =ITM BASE+( channel ); \
 while ( *_ch_ == 0); \
  *_ch_ = (_data_); \
}
int main()
{
 hardware_setup();
 ITM_ENABLE_ACCESS;
 ITM TRACE PRIV
                          = 0;
  ITM_TRACE_ENABLE
                          = 0 \times FFFFFFF;
```

Data is written to one of the 32 stimulus ports using the ITM\_TRACE\_D\* macros. The Cortex-M makes no assumptions about the contents of each stimulus port and leaves the assignment of them up to the user. For example, one could use each port for a different thread or task within the embedded application.

The ITM can be configured from the main ITM window which can be opened using the menu item, clicking the ITM button on the main trace configuration window or by using the **ITM.state** command. The window looks like this.

B::ITM.state			- • •
itm ○ OFF ④ ON commands RESet ③ CLEAR ☞ Register ④ ITMTrace ④ TPIU Ⅲ List ⑤ BMC	trace	TImeMode External CydePrescaler 1/1 CydeAccurate CLOCK 72.0MHz TimeStampMode 1/8192 TimeStamps TimeStampLOCK	SyncPeriod TraceID 16. TracePriority 2.

## To capture software generated trace:

- 1. Switch ITM.ON
- 2. Set Trace.CLOCK to the CPU clock frequency. More information about timing can be found here:"**Time Stamping**", page 72
- 3. In the main trace configuration window (Trace.state), set
  - METHOD to either Analyzer or CAnalyzer. Only those options which the tools support will be available.
  - Set the state to OFF (Trace.OFF)
  - Set AutoArm ON (Trace.AutoArm ON). This will start and stop trace sampling as the core starts and stops.
  - Set AutoInit ON (Trace.AutoInit ON). This will clear any existing trace data from the buffer before capturing new data.
  - Set Mode to FIFO, Stack or Leash. A discussion of how this affects the trace buffer can be found here: "Trace Buffer Management", page 10.
- 4. Start the target to capture trace data. The AutoArm option will ensure that capture starts and stops with the target.
- 5. The target can be halted manually or via a breakpoint.

B::ITMTrac	e.List							
🖉 Setup	🔒 Goto	Find	Chart	K Profile	🔼 MIPS	More	Less	
recor	d run add	lress	cycle	data	symbol			ti.back
-00000013	33	C:E00000	DOC wr-byt	te 0:	L			11.660us 🔥
-000000012	28	C:E00000	004 wr-wor	-d 0150	2			25.474ms
-000000012	25	C:E00000	00C wr-byt	te O	0			11.660us
-00000012	21	C:E00000	004 wr-wor	-d 01B/	4			25.474ms 🎽
-000000011	L8	C:E00000	00C wr-byt	te 0:	L			11.780us ^
-000000011	L4	C:E00000	004 wr-wor	rd 01D	)			25.475ms
-000000011	11	C:E00000	00C wr-byt	te 0:	L			11.660us
-000000009	94	C:E00000	004 wr-wor	-d 01BI	0			25.474ms
-0000000008	39	C:E00000	00C wr-byt	te 0:	L			11.780us
-0000000008	35	C:E00000	004 wr-wor	-d 01B	E			25.474ms
-000000008	32	C:E00000	00C wr-byt	te 0.	L			11.780us
-00000007	77	C:E00000	004 wr-wor	-d 0211	E			25.474ms
-00000007	74	C:E00000	00C wr-byt	te 01	L			11.660us
-00000007	70	C:E00000	004 wr-wor	d 01B	4			25.474ms
-000000000	57	C:E00000	00C wr-byt	te 01	L			11.760us
-000000006	53	C:E00000	004 wr-wor	-d 01CI	D.			25.474ms
-000000006	50	C:E00000	00C wr-byt	te 0.	L			11.780us
-000000009	0	C:E00000	004 wr-wor	rd 021	3			25.474ms
-000000004	1/	C:E00000	JOC wr-byt	te 0.	L			11./80us
-000000004	13	C:E00000	004 wr-wor	-d 016	5			25.4/5ms
-000000004	10	C:E00000	JOC wr-byt	te O	)			11./80us
-000000000	36	C:E00000	004 wr-wor	rd 01B	2			25.4/4ms
-000000000	52	C:E00000	JUC wr-byt	te 0.	L			11./60us
-000000002	28	C:E00000	JU4 wr-wor	a 01D	2			25.4/4ms
-00000002	25	C:E00000	JUC wr-byt	te 0.	L			11.680us 👻
	_ <							<u>ب</u> <

The individual channels can be split out for display by using the **ITM.PortFilter** command. For example, to view data from just channel 1 use:

ITM.PortFilter 0x01 ITMTrace.FLOWPROCESS ITMTrace.List

To view additional channels, use **ITM.PortFilter** with a binary value for the channels you wish to include. TRACE32 uses 0yXXXXXXXX to indicate a binary value. For example:

ITM.PortFilter 0y0011010 ITMTrace.FLOWPROCESS ITMTrace.List A complete source code example can be found under:

~~/demo/arm/hardware/stm32/stm32f3/custom\_itmprintf

An example itm\_printf() function is shown below.

```
#include <stdarg.h>
#include "itm printf.h"
extern int vsprintf(char *buf, const char *fmt, va_list args);
void ITM printf(const char *format,...)
{
    union {
  char c[100];
  unsigned int i[25];
    } line;
    unsigned int v;
    int i,j,l;
    va_list ap;
    va_start(ap, format);
    l=vsprintf(&(line.c[0]),format,ap);
    1++;
    1++;
    va_end(ap);
    i=0;
    i=0;
    while (i<1)
    {
        v=line.i[j];
        i+=4;
        j++;
        if (i>1)
            v&=(0xFFFFFFFF>>((i-1)*8));
        ITM_TRACE_D32(0,v);
    }
}
```

**NOTE:** The version of vsprintf() provided by your compiler <u>MUST</u> be thread safe and re-entrant. If not, consider protecting it with a mutex or similar.

This function can be called in the application just like a regular printf().

ITM\_printf("ADC: New ADC value %d", new\_ADC);

With ITM set to collect data trace, a standard trace listing will look like this.

🔢 B::itmtrace.list								×
🖉 Setup 🔒	Goto	Find	Chart	🔼 Profile	MIPS	<b>♦</b> More	Less	
record r	un addı	ress	cycle	data	symbol	ti.ba	ack	
-0000000165		C:E0000	000 wr-lor	ng 6E6F5A2	0	0.	560us	~
-0000000160		C:E0000	000 wr-]or	ng 203D206	5	0.	560us	
-0000000155		C:E0000	000 wr-]or	ng 0000003	1	0.	560us	
-0000000147		C:E0000	000 wr-]or	ng 3A544F4	D	77.	780us	Ť
-0000000142		C:E0000	000 wr-]or	ng 77654E2	0	0.	540us	^
-0000000137		C:E0000	000 wr-lor	ng 6C75502	0	0.	440us	
-0000000131		C:E0000	000 wr-lor	ng 3120657	3	0.	560us	
-0000000126		C:E0000	000 wr-lor	ng 3731373	9	0.	560us	
-0000000121		C:E0000	000 wr-lor	ng 0000000	0	0.	560us	
-0000000113		C:E0000	000 wr-lor	ng 3A43444	1	25.	328ms	
-0000000108		C:E0000	000 wr-lor	ng //654E2	0	0.	560us	
-0000000103		C:E0000	000 wr-lor	ng 4344412	0	0.	560us	
-000000009/		C:E0000	000 wr-lor	ng 6C61562	0	0.	560us	
-0000000092		C:E0000	JOU wr-lor	ng 352065/	5	0.	560us	
-000000086		C:E0000	JOU wr-lor	ng 0000303	3	_0.	540us	
-000000078		C:E0000	JOU wr-lor	1g 3A544F4	D	//.	560us	
-00000000/3		C:E0000	JOU wr-Ior	1g 6E6F5A2	0	0.	560us	
-0000000068		C:E0000	JUU wr-Ior	1g 2030206	5	0.	560us	
-0000000062		C:E0000	JOU wr-Ior	ng 0000003	1	_0.	540us	
-0000000055		C:E0000	000 wr-10r	19 5A544F4	0	//.	540us	
-00000000000		C:E0000	000 wr-10r	Ig //654E2	0	0.	540US	
000000044		C : E0000	000 wr - 10r	19 00/0002	2	0.	440us	
-0000000039		C E0000	000 wr 10r	19 3220037	3	0.	F60uc	
-0000000028		C:E0000	000 wr-10r 000 wr-10r	na 0000000	0	0.	560us	~
	<				-		>	

To display the results correctly, use **PrintfTrace** instead.

PrintfTrace.List MESSAGE List.NoDummy

B::printftr	ace.list MES	SAGE list.nodu	mmy				• •
🔑 Setup	🔒 Goto	👘 Find	Chart	🔼 Profile	K MIPS	More	Less
reco	rd messa	ge					
-00000015	59 ADC:	New ADC Va	alue 336				~
-00000015	25 MOT:	Zone =_0					
-00000015	02 MOT:	New Pulse	36937				
-00000014	64 ADC :	New ADC_Va	alue 516				Ţ
-00000014	30 MOT:	Zone = 1					^
-00000014	06 MOT:	New Pulse	23949				
-00000013	67 ADC :	New ADC_Va	alue 442				
-00000013	33 MOT:	Zone = 1	20545				
-00000013	10 MOT:	New Pulse	20545				
-00000012	72 ADC	New ADC Va	alue 428				
-00000012	38 MOT:	Zone = 1	10001				
-00000012		New Purse	19901				
-00000011	AG MOT	Zopo = 0	arue 552				
-00000011	24 MOT	Now Pulso	36685				
-00000010	88 ADC	New ADC V:	100 425				
-00000010	54 MOT	Zone - 1	1100 425				
-00000010	31 MOT	New Pulse	19763				
-00000009	96 ADC :	New ADC Va	alue 432				
-00000009	62 MOT:	Zone = 1					
-00000009	39 MOT:	New Pulse	20085				
-00000008	96 ADC :	New ADC Va	alue 452				
-00000008	62 MOT:	Zone = 1					
-0000008	39 MOT:	New Pulse	21005				~
	<						➤:

To provide any kind of timing analysis the trace decoders need some time stamping information. There are several options, each with its own set of advantages and drawbacks.

B::ITM.state			- • ×
itm OFF OFF ON commands RESet CLEAR Register ITMTrace ITMTrace ITMTrace ISMC	trace	TImeMode External V CydePrescaler 1/1 V CydeAccurate CLOCK 72.0MHz TimeStampMode – 1/8192 V TimeStamps TimeStampGOX	SyncPeriod

ITM.TImeMode	Controls the timing information used by the trace decoders. Timestamps can be off, generated by the ITM or generated by the TRACE32 hardware as each packet is decoded. An option exists to combine internal and external timestamps to allow better correlation with other trace sources.
ITM.CyclePrescaler	This sets the divider from the core clock to generate the timestamp clock for ITM messages. ITM.CLOCK still contains the core clock value.
ITM.CycleAccurate	If this option is enabled, the ITM will insert cycle count packets into the trace stream. Timestamps are based upon the cycle count and the value of the core clock. If this option is not selected, TRACE32 hardware tools will generate timestamps for packets as they are de-queued from the trace interface.
ITM.TimeStampCLOCK	This is the value of the global timestamp clock and is used for multi-core systems.
ITM.TimeStampMode	Determines whether the timestamp clock is derived form the CPU clock or TPIU clock.
ITM.TimeStamps	Enable global timestamp packets.
ITM.CLOCK	This is the frequency of the cpu core clock. It is used in conjunction with CycleAccurate mode to calculate how long between trace events.
ITM.SyncPeriod	Controls how frequently (number of clock cycles) a synch packet is generated. The default is 1024.

In most cases, the default options are the correct ones to choose.
ITM trace data can also be streamed to a file on the local hard drive. The configuration is very similar to that of ETM which is explained here: "ETM Stream Mode", page 17.

Trace data can also be streamed to a shared library for custom processing or handling. An example of this can be found under  $\sim /demo/arm/hardware/xmc/xmc4500/custom_itmprintf$ . The full source code for the DLL or shared library is in the DLL sub-directory, along with make files and instructions on how to build for your host operating system. How to create a build such a DLL or shared library is beyond the scope of this document.

To load a shared object for custom trace processing use the command

## <trace>.CustomTraceLoad "name" <file>

To use, set **Trace.Mode PIPE**. The DLL is responsible for managing its own display of the data that it has processed. The TRACE32 api allows for the creation of a custom command to display the data in text only format in a window inside TRACE32. If a more complex display of the data then this is required it is the responsibility of the DLL to do this. By using stream mode like this, the trace data is not stored but parsed on-the-fly.

The ITM trace data can be streamed to a pipe (Named pipe in Windows, pipe or FIFO in Linux or MacOS). This allows for a host application to react in almost real-time to data being generated by the target. There is no buffer to fill, so theoretically, the trace could be of unlimited duration.

An example of creating a Windows forms application with a named pipe using Visual Studio (tested on Windows 7, 8 and 10 with Visual Studio 2015 and 2017) can be found under ~~/demo/arm/hardware/stm32/stm32f3/custom\_itmprintf.

The script in this directory shows how to configure TRACE32 for streaming ITM data to a pipe.

## To stream ITM data to a named pipe:

- 1. Set Trace.Mode PIPE
- 2. Launch the Host OS application that will create and open the pipe
- 3. Connect TRACE32 to the pipe with Trace.PipeWRITE "<pipe>"
- 4. Start the target

The ITM data is exported from TRACE32 to the pipe in a packet with the following format.

Byte	Meaning
0	The size of the rest of the packet in bytes. This byte does not count towards the total length.
1 - 8	Timestamp in Little Endian format
9	Message type: data = 1 indicates there are 8 bits data = 2 indicates there are 16 bits data = 3 -indicates there are 32 bits
10	ITM Channel ID
11 - 14	Payload in Little Endian format.

Up to 8 pipes can be simultaneously supported by TRACE32. Calling **Trace.PipeWRITE** without any arguments will close all open pipes.