LAUTERBACH
DEVELOPMENT TOOLS

Application Note Benchmark Counter RH850

Release 09.2023

MANUAL

Application Note Benchmark Counter RH850

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .....

ICD In-Circuit Debugger .....

Processor Architecture Manuals .....

RH850 .....

RH850 Application Notes .....

Application Note Benchmark Counter RH850 ..... **1**

Introduction **3**

Intended Audience 3

Prerequisites 3

Related Documents 3

Measuring Runtimes **4**

Measuring a Single Function 4

Breaking when a Function Takes Longer than a Specified Time 7

Measuring Multiple Functions 9

Measuring a Task or Thread 12

Multi-Core Considerations **14**

Application Note Benchmark Counter RH850

Version 10-Oct-2023

Introduction

The Renesas RH850 family of devices all provide on-chip performance counters. These can be programmed to provide very accurate runtime measurements of a single block of code.

TRACE32 uses the **BMC** (Bench Mark Counter) command group to program and control these on-chip performance counters. The intent of this application note is to demonstrate the use of the RH850 performance counters with TRACE32.

Intended Audience

Developers, who want to be able to:

- Accurately profile a block of code or single function.
- Accurately measure the runtimes of several functions.
- Be able to halt the target when a block of code exceeds a certain execution time.

All this can be done with a TRACE32 debugger, no trace capability is required.

Prerequisites

It is assumed that TRACE32 has been correctly configured for the target and the symbols for the application being debugged are loaded into TRACE32. The reader should also be familiar with developing embedded systems in C or C++ and have a basic understanding of JTAG debugging embedded targets.

- The CPU core clock must not be altered during the sampling period. The core must not be allowed to go into low power where clocks are temporarily disabled/suspended.
- When measuring tasks or threads, the TRACE32 OS Awareness for the target OS must be correctly configured.

Related Documents

- General setup for RH850 debuggers: **"RH850 Debugger and Trace"** (debugger_rh850.pdf)

Measuring Runtimes

In this section:

- Measuring a Single Function
- Breaking when a function takes longer than a specified time
- Measuring Multiple Functions
- Measuring a Task or Thread

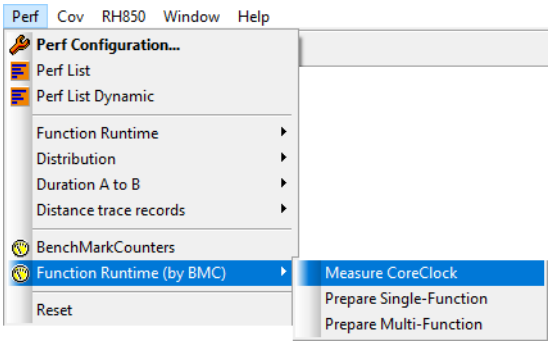
Measuring a Single Function

A single function or contiguous block of code may be marked for runtime measurement. The instructions below will show how to do this.

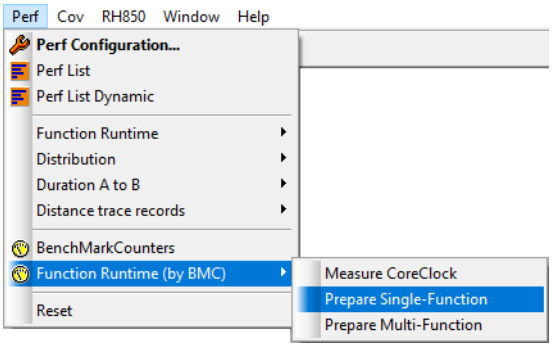
To measure a single function:

1. Measure the CPU core clock frequency.

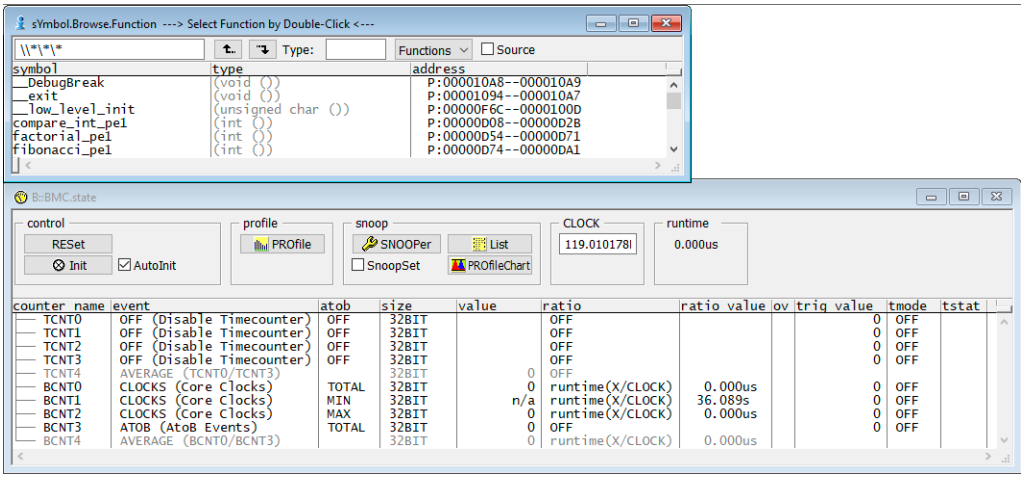
   The menu item shown in the diagram below will do this.



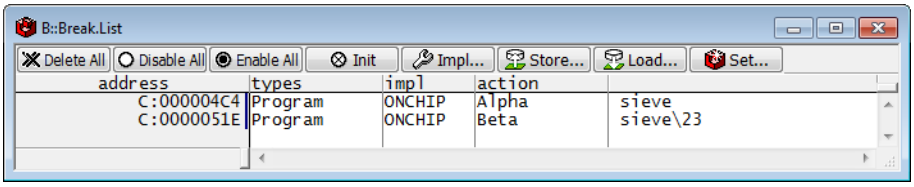
2. Select the function to be measured by using the menu item shown here.



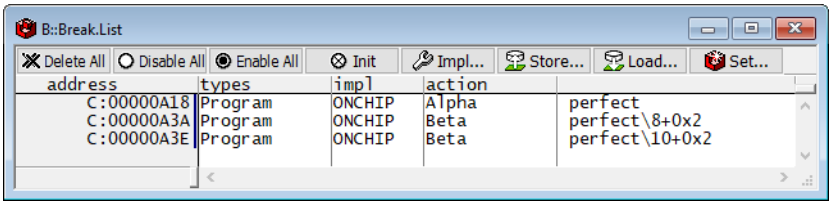
   This will open a list of all functions that have been loaded into the TRACE32 symbol database and a window showing the state of the Benchmarking Counters (**BMC.state**). It should look like the image below.



3. Double-click a function name to set the entry and exit markers.

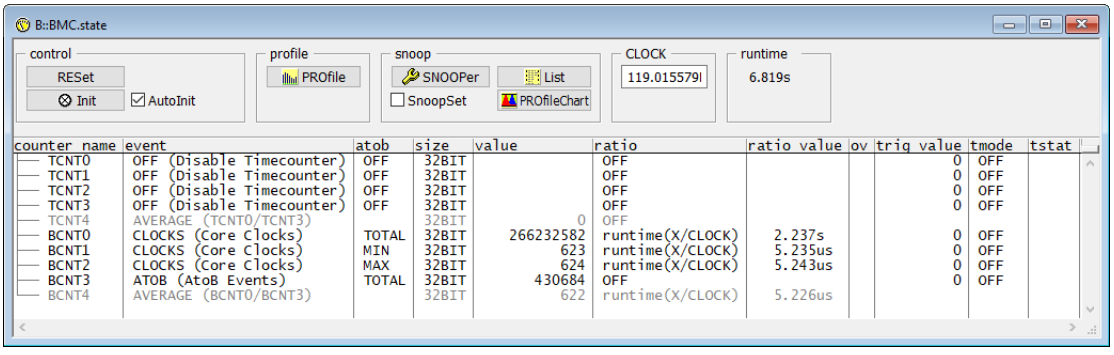


These will be displayed in a **Break.List** window and look something like the image above. The starting point is marked with an Alpha breakpoint and the ending point is marked with a Beta breakpoint. A single starting point can be paired with up to 7 ending points; ideal if a function has more than one return point. In the example below, function `perfect()` has two exit points and the automated marking process has detected these and marked them both.



4. Start the target running.

The **Benchmark Counter** window is updated about twice a second and looks like this.



Be aware the measured times include the execution times of all subfunction calls and interrupt

requests. The data is interpreted as:

Row	Value	Ratio Value
BCNT0	Total number of measured CPU cycles	Time for the measured CPU clock cycles
BCNT1	Minimum number of clock cycles measured for this function.	Minimum time for the selected function.
BCNT2	The maximum number of clock cycles for the selected function.	The maximum time for the selected function.
BCNT3	Number of events (times the end marker has been counted).	N/A
BCNT4	Mean number of cycles for the selected function.	The mean execution time for the selected function.

Breaking when a Function Takes Longer than a Specified Time

It is possible to cause the target to halt when a particular piece of code takes longer than a specified time to execute. If this is combined with the program flow trace then the sequence of events leading up to a missed deadline can be examined. For information about program flow trace, see **"NEXUS On-chip Trace"** in RH850 Debugger and Trace, page 36 (debugger_rh850.pdf).

In this example, the function `perfect_pe1()` will be used. It has two exit points which provides us with two distinct runtime groupings. The initial setup is the same as for measuring a single function.

The counters will only trigger if a cycle count exceeds a maximum value. This can be converted into a time value using the formula below:

```
No. Cycles / CPU clock (Hz) = time
```

To break when a function takes longer than a specified time:

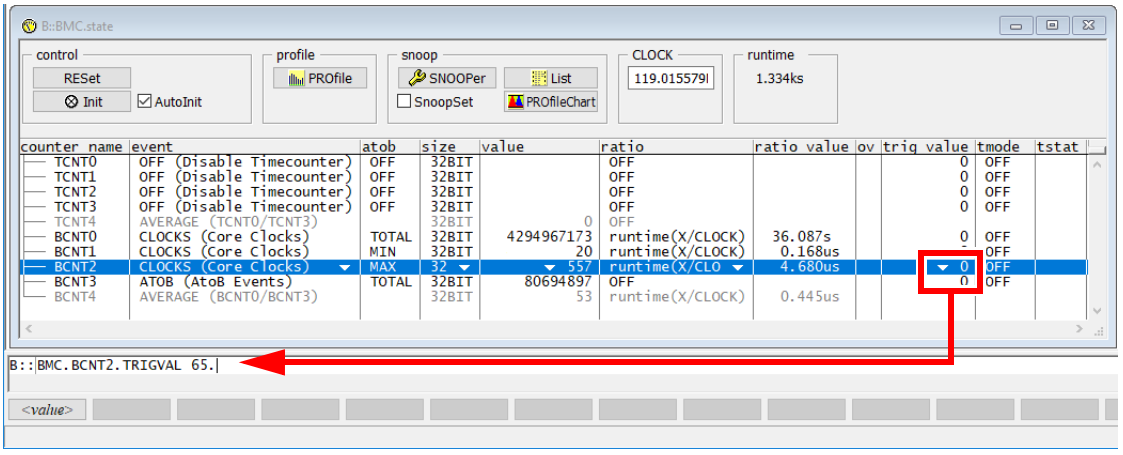
1. Calculate the number of cycles using the formula above:

   ```
   n = 0x54e-6s * 119015579 Hz

     = 64.268 cycles
   ```

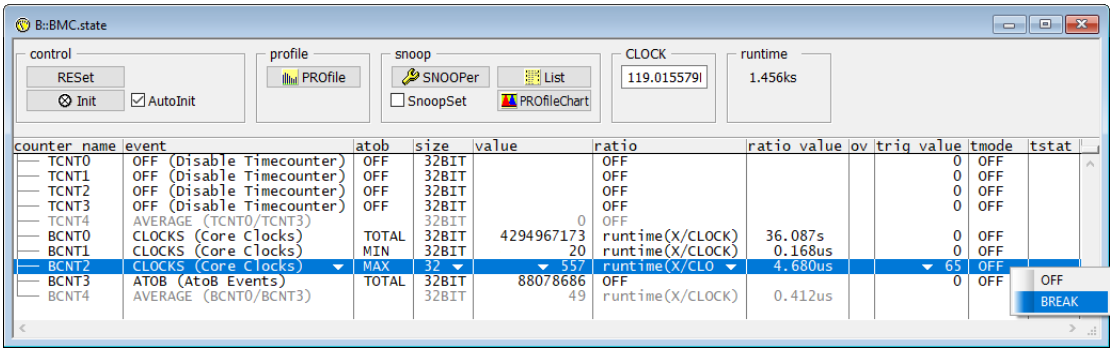
   Use a value of 65.

2. Double-click the intersection of the BCNT2 (max) row and the **trig value** column.



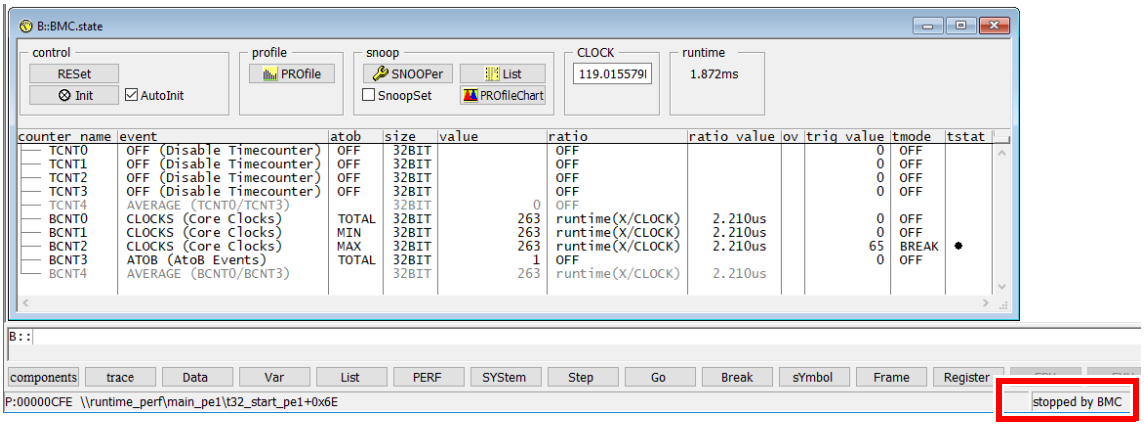
The Trigger register only supports 32-bit values. It is not possible to enter a larger value; the top bits will be quietly discarded.

3. Enter the required value on the TRACE32 command line. Note the final "." to indicate to TRACE32 that we want this to be treated as a decimal number.



4. Right-click **tmode** column on the same row and change from OFF to BREAK. This is shown in the image above.

5. Start the target. If the time between two marked points (Alpha and Beta) exceeds the number of cycles, the target will be halted. TRACE32 status line changes to read "stopped by BMC" and **BMC.state** window shows which counter caused the break.



There is a small latency when using this feature. The range has been measured between 5 and 9 clock cycles using the example provided, although it is dependent upon the complexity of the application code at that point. The event must be detected by the core's counter logic and then transferred to the core-break logic and the halt event must then be inserted into the core's instruction pipeline. When the event reaches the core, it will halt.

If the target has been configured to provide trace data, the tracing will also stop being sampled at that point, allowing the user to look backwards through the trace buffer in order to determine the cause of the missed deadline.

Measuring Multiple Functions

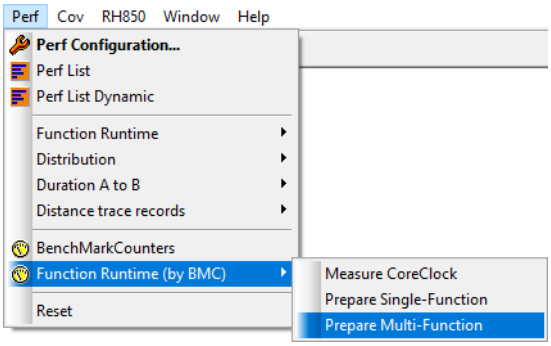
The runtime values of multiple functions can be measured by using the Benchmark Counters. To do this, a list of functions to measure is generated and each is sampled in turn for a user specified period. When all of the samples have been collected, a table is updated showing min, max and mean runtimes for each function in the list.

Prerequisite:

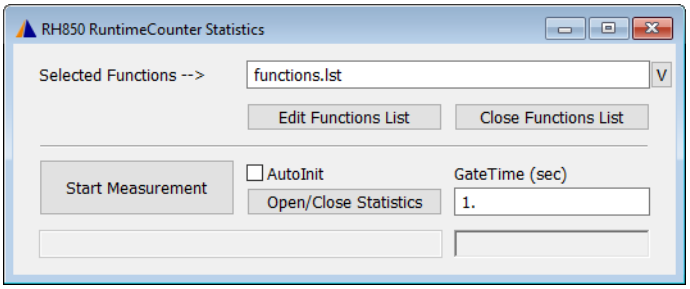
- To start, the CPU clock must be measured. Instructions for how to do this can be found here **"Measuring a Single Function"**, page 4.

To measure multiple functions:

1. Select Prepare Multi-Function from the BMC perf menu.



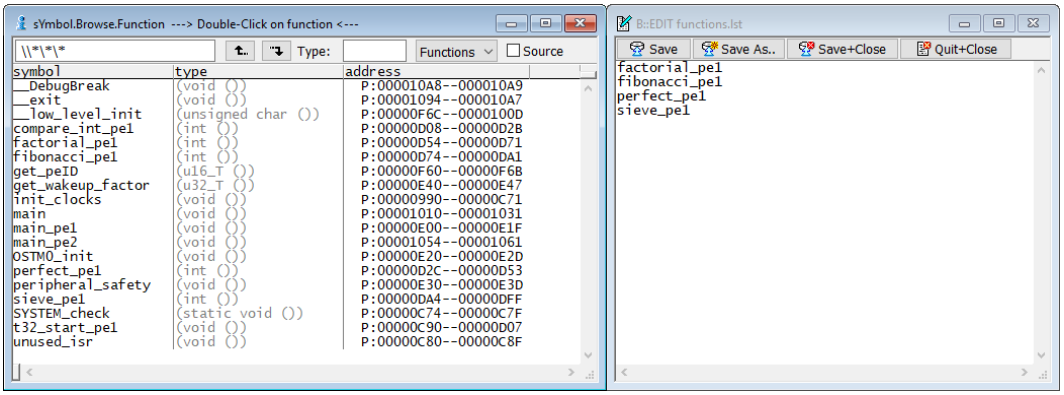
This will open the main control window which looks like the image below.



2. Enter the GateTime (seconds) you want. The GateTime value determines the length of time of the sampling period for each function on the list.

3. A file with a previously prepared list of functions to be profiled can be loaded by clicking the **V** button. The file is a plain text file with the functions placed one per line with a final return at the end of the file. Alternatively, a new file can be prepared by clicking the **Edit Functions List** button.

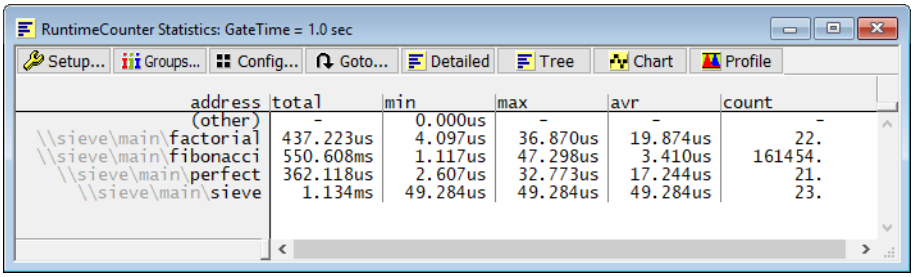
This will open a symbol browser window and an edit window. Functions can be added to the list in the editor by double-clicking them in the browse window or by manually typing them. See example below. (Remember to save the list when finished.)



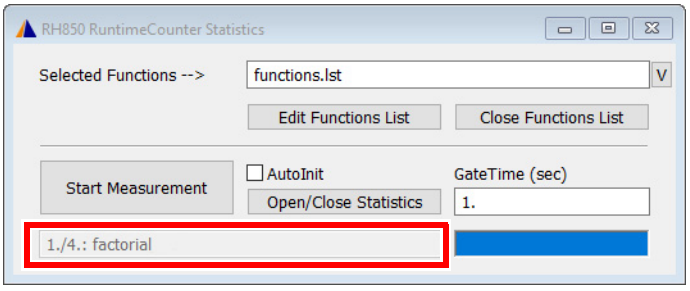
4. Selecting the **AutoInit** check box will clear any existing results before starting the current sample run(s). If the **AutoInit** check box is not checked the new data will be added to any existing data for functions that have already been sampled.

5. The current statistics can be viewed by clicking the **Open/Close Statistics** button. If this window is not visible when a test is started, it will automatically be opened.

6. To start the measurements, click the **Start Measurement** button. The statistics window will open and be filled as data for each new function becomes available. It will look something like this.



As the measurements for each function are performed, the main control window will be updated to indicate which function is currently being measured.



7. Export measurement results in CSV format.

```
;select destination of export
PRinTer.select ClipBoard CSV     ; select clipboard or
PRinTer.select <my_file>.csv CSV  ; select a file

; print measurement results to selected destination
WinPrint.BTrace.STATistic.Func %TimeFixed Total MIN MAX AVeRage
Count
```

Measuring a Task or Thread

The runtimes of a task or thread can be measured in a similar fashion. In this section the word task is used but could apply to task, process or thread; whatever makes sense for the chosen OS.

Examples of OS configuration PRACTICE scripts (*.cmm) can be found under `~~/demo/rh850/kernel`.

NOTE:
- The OS Awareness for TRACE32 must be correctly configured for your chosen OS.
- Ensure that the OS does not change the CPU clock frequency or allow the core to go into low power/sleep modes.

Prerequisite:

- To start, the CPU clock must be measured. Instructions for how to do this can be found here **"Measuring a Single Function"**, page 4.

To measure a task or thread:

1. Open the Perf->Function Runtime (by BMC)->Prepare Single-Function menu item and close the symbol browse window; we will not be selecting a function from the list but need the underlying setup to be completed before we can proceed.

2. Now we need to mark the starting point for the counter by an Alpha breakpoint and the stopping point by a Beta breakpoint.

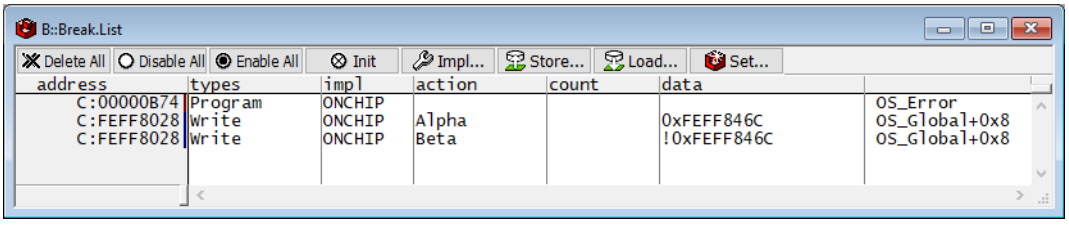
   Our starting point is the point of time where the kernel writes the identifier for our task to the variable that contains the identifier of the currently running task. Our stopping point is the point of time where the kernel writes another identifier to this variables.

   This is done with the help of the following TRACE32 functions:

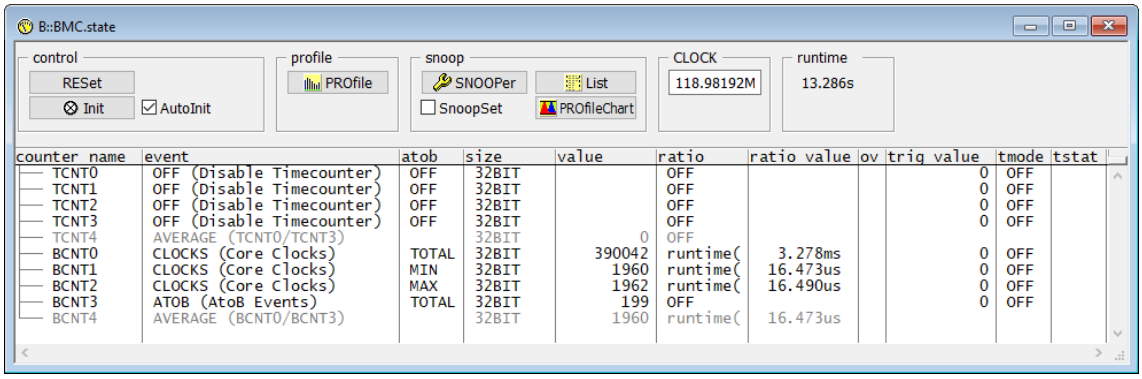
TASK.CONFIG(magic)	Returns the address of the location that contains the currently running task.
TASK.MAGIC("<task>")	Returns the identifier (magic number) of the specified task name.

```
Break.Set TASK.CONFIG(magic) /WRITE /DATA TASK.MAGIC("task") /Alpha
Break.Set TASK.CONFIG(magic) /WRITE /DATA !TASK.MAGIC("task") /Beta
```

and the results will look like this.



3. Start the target executing and review the results in the **BMC.state** window. It should look like this:

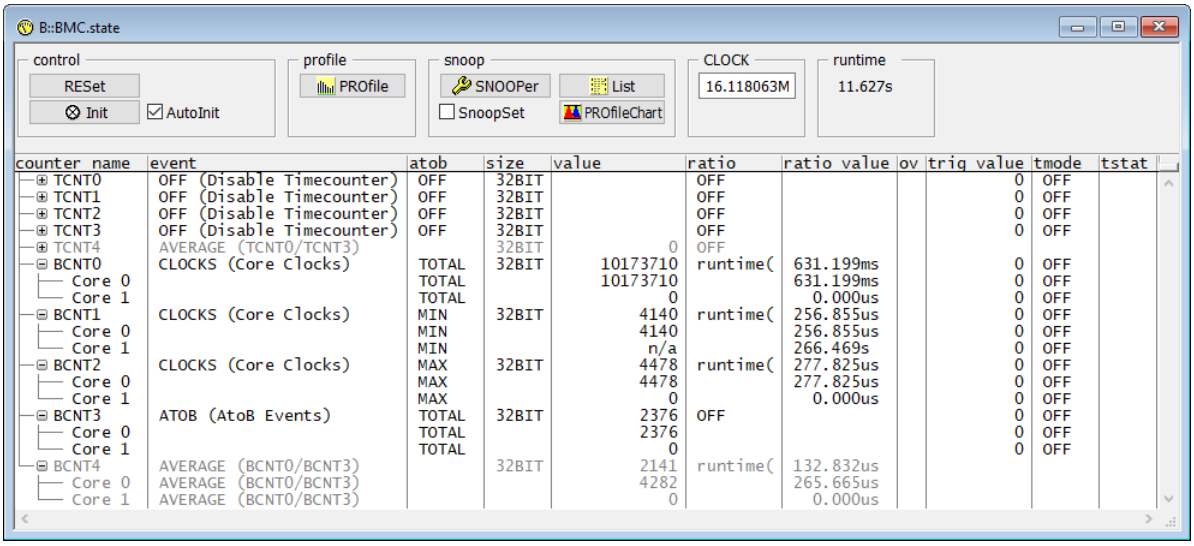


counter_name	event	atob	size	value	ratio	ratio value	ov	trig value	tmode	tstat
TCNT0	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
TCNT1	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
TCNT2	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
TCNT3	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
TCNT4	AVERAGE (TCNT0/TCNT3)		32BIT	0	OFF					
BCNT0	CLOCKS (Core Clocks)	TOTAL	32BIT	390042	runtime(	3.278ms		0	OFF	
BCNT1	CLOCKS (Core Clocks)	MIN	32BIT	1960	runtime(	16.473us		0	OFF	
BCNT2	CLOCKS (Core Clocks)	MAX	32BIT	1962	runtime(	16.490us		0	OFF	
BCNT3	ATOB (AtoB Events)	TOTAL	32BIT	199	OFF			0	OFF	
BCNT4	AVERAGE (BCNT0/BCNT3)		32BIT	1960	runtime(	16.473us				

Multi-Core Considerations

If the target is configured for Asymmetric Multi-Processing (AMP) then each core will be controlled by a unique instance of TRACE32. An example of this type of setup can be found under `~~/demo/rh850/hardware/rh850-f1h-emu-adapter/multicore_amp`. Configured like this, each core has its own set of Benchmarking counters and each instance of TRACE32 will behave exactly as described in the examples above.

If the target is configured for Symmetric Multi-Processing (SMP), where tasks or threads are scheduled across all available processor cores by a single operating system kernel, then a single instance of TRACE32 will control all cores. An example of this can be found under `~~/demo/rh850/hardware/rh850-f1h-emu-adapter/multicore_smp`. In this configuration the benchmark counters for all cores will be shown in one window. The times are displayed for the code running across all cores and can then be opened to show the times for any particular core in the array.



counter_name	event	atob	size	value	ratio	ratio value	ov	trig value	tmode	tstat
⊞ TCNT0	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
⊞ TCNT1	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
⊞ TCNT2	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
⊞ TCNT3	OFF (Disable Timecounter)	OFF	32BIT		OFF			0	OFF	
⊞ TCNT4	AVERAGE (TCNT0/TCNT3)		32BIT	0	OFF					
⊟ BCNT0	CLOCKS (Core Clocks)	TOTAL	32BIT	10173710	runtime(	631.199ms		0	OFF	
Core 0		TOTAL		10173710		631.199ms		0	OFF	
Core 1		TOTAL		0		0.000us		0	OFF	
⊟ BCNT1	CLOCKS (Core Clocks)	MIN	32BIT	4140	runtime(	256.855us		0	OFF	
Core 0		MIN		4140		256.855us		0	OFF	
Core 1		MIN		n/a		266.469s		0	OFF	
⊟ BCNT2	CLOCKS (Core Clocks)	MAX	32BIT	4478	runtime(	277.825us		0	OFF	
Core 0		MAX		4478		277.825us		0	OFF	
Core 1		MAX		0		0.000us		0	OFF	
⊟ BCNT3	ATOB (AtoB Events)	TOTAL	32BIT	2376	OFF			0	OFF	
Core 0		TOTAL		2376				0	OFF	
Core 1		TOTAL		0				0	OFF	
⊟ BCNT4	AVERAGE (BCNT0/BCNT3)		32BIT	2141	runtime(	132.832us				
Core 0	AVERAGE (BCNT0/BCNT3)			4282		265.665us				
Core 1	AVERAGE (BCNT0/BCNT3)			0		0.000us				

For each BCNTx row:

- Top entry shows cumulative total for all cores.
  - Except BCNT4 which shows the mean of all cores
- Core *<n>* shows the minimum, maximum, mean and total runtimes for the selected code/function when it was running on this core only.