LAUTERBACH
DEVELOPMENT TOOLS

Arm Application Note for MXC Chips

MANUAL

Arm Application Note for MXC Chips

Version 06-Jun-2024

NEXUS Preprocessors

Different NEXUS preprocessors exists that have different capabilities and require different settings. The NEXUS Preprocessor revision can be evaluated in PRACTICE scripts by the PRACTICE function A.PROBEREVISION().

```
if A.PROBEREVISION()>=7.&&A.PROBEREVISION()<16.
(
     A.THRESHOLD VCC
     SYStem.Option.SAMPLE +2
)
```

Following table informs about the existing NEXUS preprocessor revisions:

Revision	Features	Order Number
	Nexus Adapter for Argon+	LA-7606
5.	Initial revision can handle 3.3 V signals	
6.	New input buffers that can handle 1.8 V to 2.8 V signals	
7.	MCKO PLL • support DDR mode by the option SYStem.Option.HalfRate • support variable sample point by the option SYStem.Option.Sample • allows to bypass the PLL by internal jumper if MCKO signal • frequency too low or PLL can't follow MCKO signal frequency shift Input buffer with variable threshold can detect if VTREF falls below certain level • new command Analyzer.THRESHOLD to set up the threshold for input buffers	

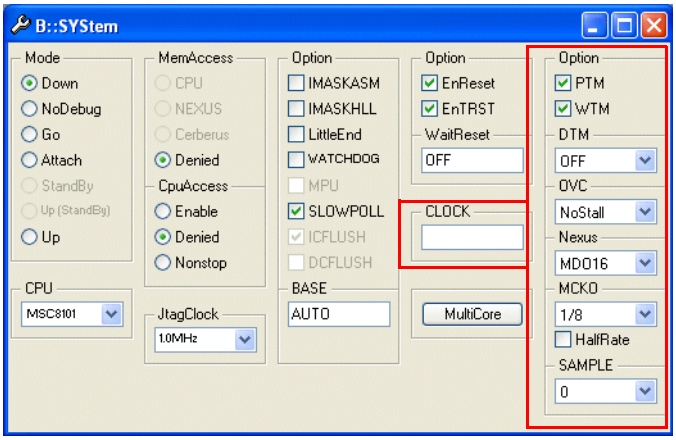
	StarCore AUTOFOCUS II	LA-7999/ LA-7999A
16.	AutoFocus II Preprocessor • variable sample point per signal • graphical analysis of data eyes • data / clock threshold	

Basic NEXUS Handling

This chapter explains the settings that define how the NEXUS probe records data.

Settings of the SYStem Window

If a NEXUS Trace is attached a column with setting dedicated to NEXUS will appear in the SYStem window.



Commands to set up a filter on the target side to specify which message types are produced

- With SYStem.Option.WTM set to ON Watchpoint Trace Messages will be produced. Watchpoint Messages will be generated at on-chip breakpoints with action TraceTrigger, BusTrigger or BusCount.
- SYStem.Option.DTM specify which type of Data Trace Messages (read access, write access, both, none) will be generated. The messages can be filtered by the TraceData On-Chip Break action. Data Trace Messages can be suppressed if the SYStem.Option.OVC is set.

Port Bandwidth related options

- If the target internal trace buffer is full Program Trace Messages will get lost and further analysis show wrong results indicating a FIFO FULL error. SYStem.Option.OVC defines the behavior in case the Trace Buffer content reached a critical amount. The options are to stall the core to have additional time to release messages or to suppress Data Trace Messages. Watchpoint Trace Messages are not suppressed as well as Vendor Defined Messages (e.g. DPU counter values)
- Another opportunity to avoid FIFO FULL errors is to use a high MCKO ratio, but the settings depends also to electrical conditions.
- SYStem.Option.NEXUS allows to specify the amount of used data lines. The current probe can handle 8-Bit and 16-Bit wide NEXUS ports. If possible, use the 16-Bit wide port to avoid FIFO FULLs.

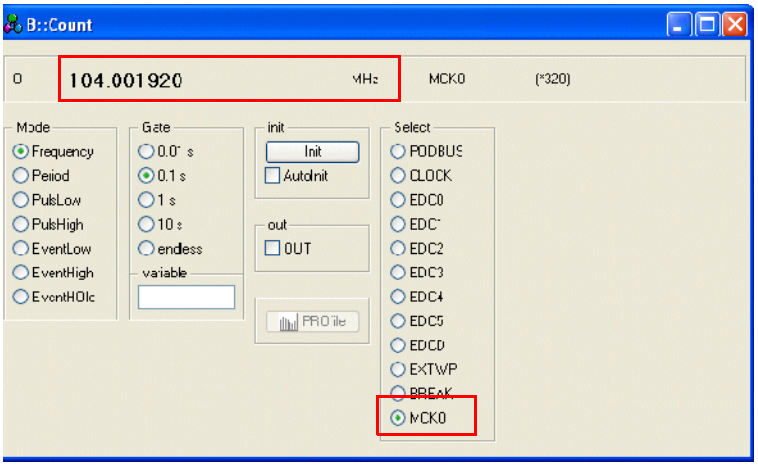
Signal related options

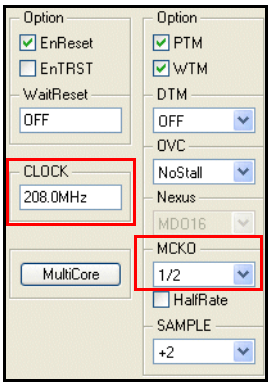
- Signal related options are necessary to avoid FLOW ERRORs that results of bad signal quality. With FLOW ERRORs any further analysis may show wrong results but indication the type of error.
- The MCKO setting defines how much data is transferred through the MDO data lines in ratio to the StarCore core clock. Since the settings does not allow fine tuning it would be necessary to fine-tune the core clock to reach optimum results.
- If supported by the NEXUS probe SYStem.Option.HalfRate can reduce the MCKO clock by two while the data bandwidth remains at the same level. The setting should be used if the bandwidth of the MCKO line is too low.
- If the NEXUS probe supports it, the SYStem.Option.Sample option will allow to move the data sample point relative to the MCKO signal. The StarCore AUTOFOCUS II probe handles the sample by the Analyzer.ShowFocus command.
- Probes with Revision 7. support HalfRate and Sample, but have only a limited support to dynamic core frequencies since they use a PLL internally which can be bypassed through a jumper inside the probe. Also if the core frequency is constant the PLL needs to be set up by the setting SYStem.Clock.
- Another important setting is Trace.THRESHOLD, that defines the comparator voltage. This setting has influence to the signal sample timing and a right setting can reduce FLOW ERRORs. The threshold should be set to 1/2 of the signal voltage level.

In case of Flow Errors a first look to the MCKO signal can help to set up the NEXUS probe. The Count window displays a frequency counter of the MCKO signal line. If all settings are correct the frequency will be

$f_{MCKO\_Display}$ = SYStem.Clock * SYStem.Option.MCKO

for all possible MCKO settings.





The real MCKO frequency at the MCKO line depends to the HalfRate option, too. If HalfRate is true, the MCKO frequency will be divided by two. In this case the value in the Count window will not change since it is multiplied by the internal PLL by two automatically.

Minimum Frequencies

Probes with internal MCKO PLL show a frequency below the minimum PLL range when the real core is below the **SYStem.Clock** value or the target is not connected to the probe. The PLL range will be selected depending to the **SYStem.Clock** and **SYStem.Option.MCKO** value:

1. 24.0 MHz < (SYStem.Clock * SYStem.Option.MCKO) < 50.0 MHz
2. 50.0 MHz <= (SYStem.Clock * SYStem.Option.MCKO) < 100.0 MHz
3. 100.0 MHz <= (SYStem.Clock * SYStem.Option.MCKO) < 200.0 MHz

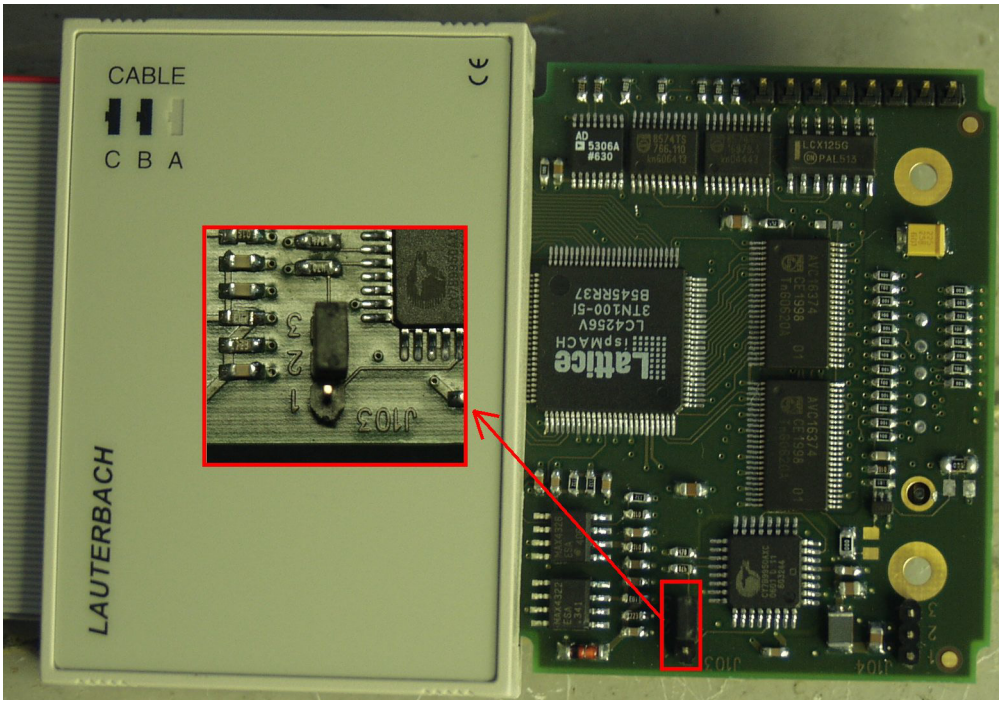
PLL Range	Showed MCKO Frequency if the probe is not connected or MCKO signal cannot be sampled
1	around 12 MHz
2	around 24 MHz
3	around 49 MHz
probe without PLL	0 … 2 MHz

Disable/Enable Probe PLL for MCKO

In certain cases (DVFS etc.) the MCKO PLL does not work correctly and must be disabled.

To disable/enable the PLL on the probe, one has to open the plastic box of the probe (there is no need to open the PowerTrace unit) . The box part apart the blue ribon cables is the right part.

After the PCB is visible, there is a jumper which allows to bypass the PLL. Refer to the picture below.



Pos 3-2 : PLL is active and not bypassed
Pos 1-2 : PLL is inactive and bypassed.

If the PLL is bypassed, all settings for the PLL are not effective

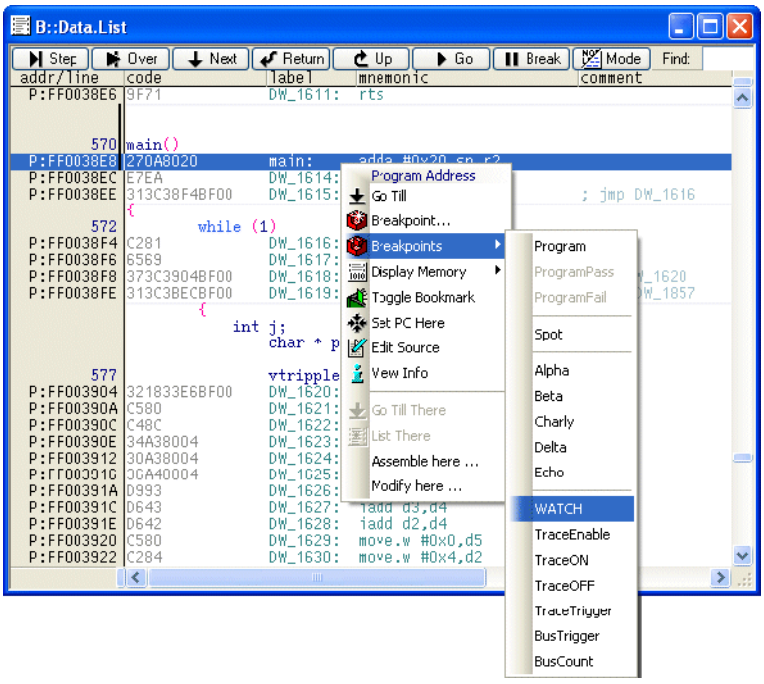
The command Analyzer.TestFocus can test the reliability of a setup by a pin stress test. Therefore it loads a small test program into the Program RAM, executes it and compares the output with the expected pattern. A search algorithm started by the command Analyzer.AutoFocus uses Analyzer.TestFocus to evaluate different parameters sets. Analyzer.AutoFocus will set up the following values:

- detects if VTREF is greater zero (probe connected to target)
- setup Analyzer.THreshold to 1/2 VTREF
- detects if there is any MCKO signal (I/O-MUX settings are correct for MCKO)
- measure MCKO clock and detects if PLL is bypassed, finally setup SYStem.Clock
- Set HalfRate to ON if PLL is not bypassed to get better MCKO signal
- tries to find fastest reliable MCKO clock setting by increasing MCKO clock step by step until test fails
- finds the best Analyzer.THreshold value by using the mean of the reliable Threshold minimum and Threshold maximum
- finds the best SYStem.Option.Sample by the widest range of reliable Threshold minimum and Threshold maximum

If a probe does not support certain features, these features are not set by the algorithm.

NEXUS dedicated On-chip Breakpoint Trigger actions

The OCE/EOnce trigger unit can be used to control the trace message generator or to produce Watchpoint Messages which can trigger actions within the debug tools. The actions can be selected in the Break dialog or just in the context menu in a **Data.List** window:

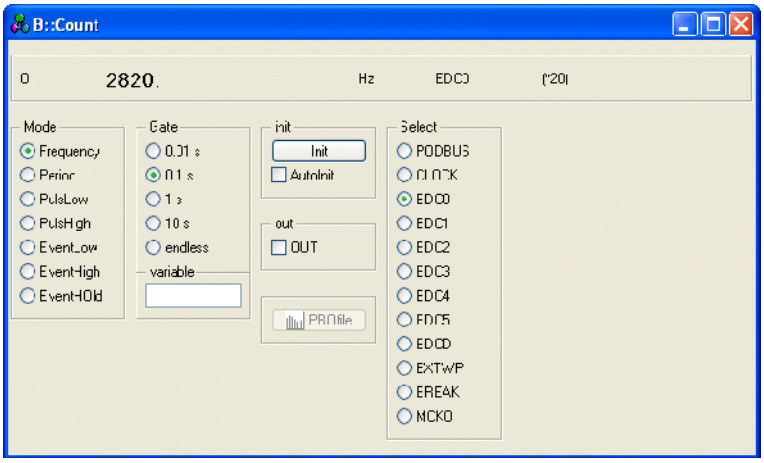


Examples:

```
b.s main /TRACEON              ; Enabled program Trace at main

b.s sieve /TRACEOFF            ; Disabled program Trace at sieve

b.s flags /TRACEDATA           ; Set up a filter for Data Trace (only NEXUS
                               ; withDTM option set to on)

b.s main /TRACETRIGGER         ; Set Watchpoint message or/and EVTO pin to
                               ; generate Trigger

b.s main /BUSTRIGGER           ; Set Watchpoint message or/and EVTO pin to
                               ; generate a trigger pulse on the PodBus

b.s main /BUSCOUNT             ; Set Watchpoint message or/and EVTO pin to
                               ; allow frequency counter feature
```

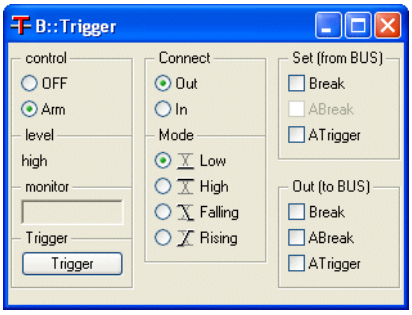
BusCount Watchpoints

BusCount Watchpoints can be enabled by the /BUSCOUNT action of an On-Chip breakpoints. Once a BusCount Watchpoint is set, the debugger will set up the Count window with the right OCE/EOnce event detection unit enabled. BusCount Watchpoints are not supported by StarCore AUTOFOCUS II, yet.



BusTrigger Watchpoints

BusTrigger Watchpoints can be set by the /BusTrigger action in the Break dialog. If a BusTrigger Watchpoint is detected a trigger pulse will be generated to the PodBus. Every debugger box with the connected through a system wide PodBus. Actions derived from the PodBus can be defined in the Trigger window. BusTrigger Watchpoints are not supported by StarCore AUTOFOCUS II, yet.



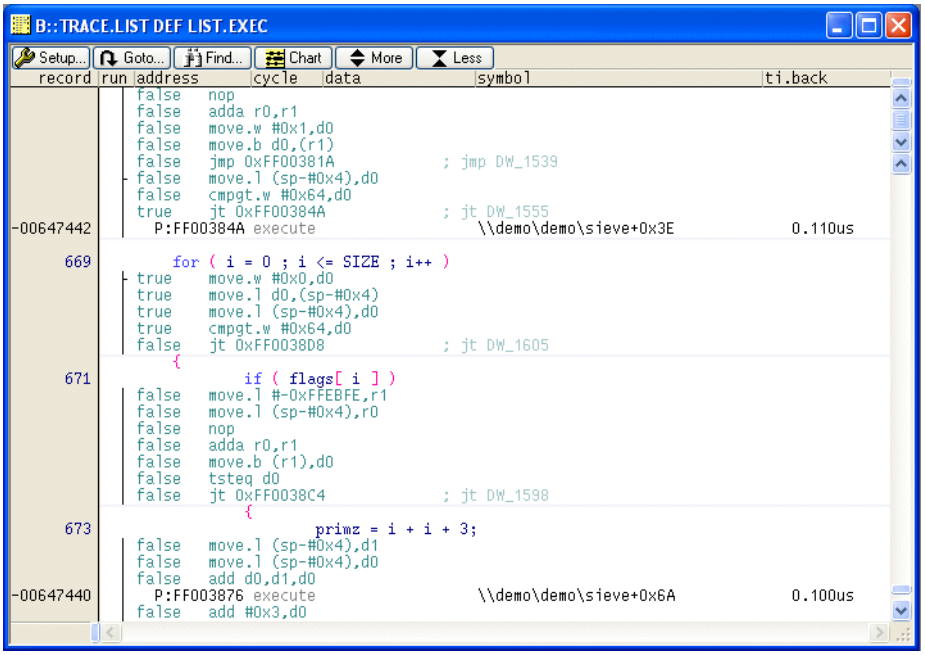
The command **Trigger.Set.Break ON** would stop the StarCore if a PodBus trigger pulse is detected. The **Set.ATrigger** option will generate a Trace Trigger when a PodBus Trigger occurs.

TRACETRIGGER

The Trace Trigger can be used to stop the record of trace messages. The Trace.TDelay option can delay the Analyzer Break relative to the amount of messages stored in the host trace buffer. The Trace Trigger is not supported by StarCore AUTOFOCUS II, yet.

Display of the T-Bit in the Trace.List Window

The option LIST.EXEC or the command TRACE.LIST will show an additional column indicating the T-Bit for every instruction. The command **TRACE.LIST DEF LIST.EXEC** would show the following window:



Setup Analysis

To provide information about the used Operating System the matching OS Awareness must be configured:

```
Task.Config quadros.t32
```

The trace record should contain any write access to the current task ID (Magic) caused by the Kernel. This can be set up by an on-chip breakpoint which selects the right range:

```
Break.Set task.config(magic) /ONCHIP /TRACEDATA
```

Operating System Kernels can include code where a general function analysis can not be done, because there are code sections where the program flow does not follow a call - return pattern. Many advanced analysis and statistic functions are based a complete function analysis of the trace, that's why the debugger must be aware of kernel entry and exit code sections by setting *kentry* and *kexit* markers:

```
; set a kentry Marker to all tvec* functions
sYmbol.ForEach "sYmbol.NEW.MARKER kentry *" tvec*

; set a kentry Marker to all tvec* functions
sYmbol.ForEach "sYmbol.NEW.MARKER kentry *" ivec*

; set a kexit marker to all RTE instructions
Data.Find symbol.range(RTXCProlog) %w 0x9f73
WHILE FOUND()
(
     sYmbol.NEW.MARKER kexit track.address()
     Data.Find
)
```

If the setup is wrong the function analysis based windows will indicate an OVERFLOW error, because they can't reconstruct the function nesting.

Command Analyzer.STATistic.Tree shows the runtime statistics of the function nesting sorted by tasks:

- A diamond indicates a jump into a trap.
- An arrow indicates a jump to an interrupt.

Command Analyzer.STATistic.Task shows a complete overview about the task runtime.

- A task gets active when it is switched inside the kernel by a write to the Task ID variable

Command Analyzer.STATistic.TaskFunc shows the runtime of the functions sorted by tasks.

Command Analyzer.STATistic.TASKKERNEL is analog to Analyzer.Statistic.Task, but the task is switched when the kexit marker is passed. The time between kentry and kexit is summarized to the entry (kernel).

Command Analyzer.CHART.TASKKERNEL shows a timing diagram of the active task sorted by task, kernel or root.

CTS based analysis windows will pop up faster once CTS is activated:

```
CTS.UseFinalMemory OFF
CTS.UseFinalContext OFF
CTS.INCREMENTAT OFF
CTS.ON
CTS.STATistic.TREE
CTS.STATistic.Func
CTS.STATistic.TASKKERNEL
CTS.PROFILECHART.TASKKERNEL
CTS.CHART.TASKKERNEL
CTS.List
```

Export DPU Counters

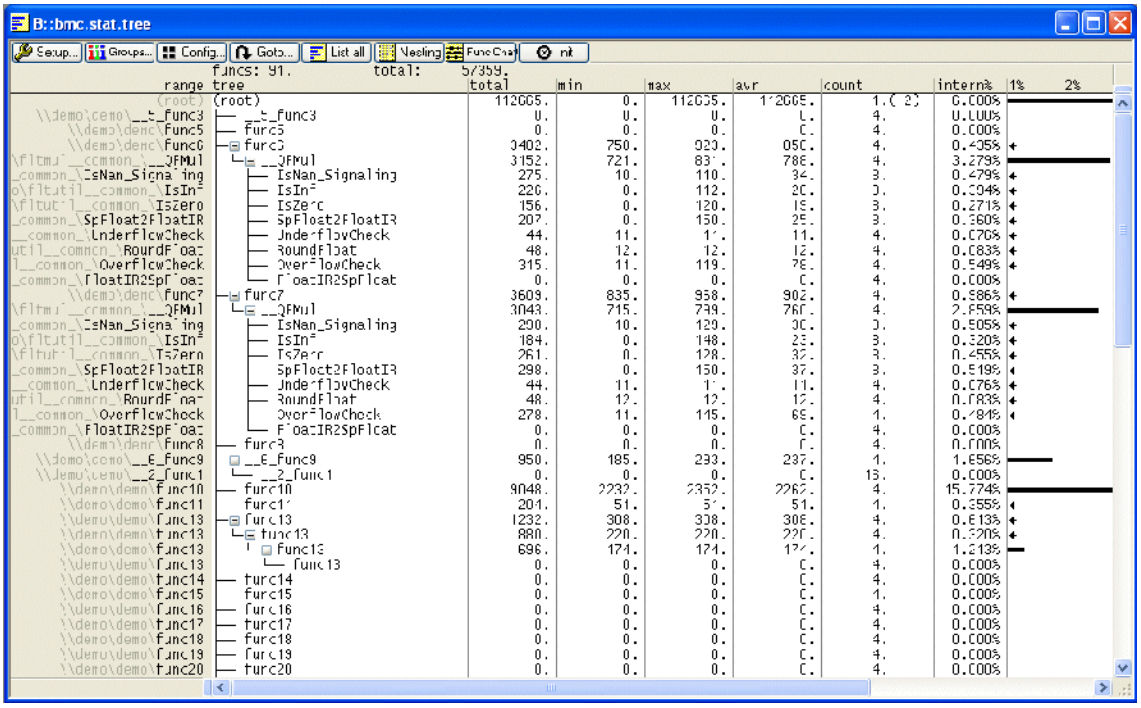
The command **Analyzer.Export** can export a binary file containing the DPU counter values, too. The format of a data record containing a DPU counter set is:

```
0x80 0x00 0x00 0x00 0x00 0x00 0x00 <block> <c0> <c1> <c2> <ts>
with:
     <block>: 0x01 for DPUA and 0x02 for DPUB
     <cn>: counter, 32bit, little endian
     <ts>: timestamp, 64bit, little endian
```

Display Settings

- The command **Trace.List DEFault DPUA0 DPUA1 DPUA2 DPUB0 DPUB1 DPUB2** shows the flow trace and displays the counter value derived from the vendor trace message. The counters will appear between the HLL lines after every function call and return statement. To see the six counters click on **More** and scroll to a function entry or exit. With the command **Trace.List DEFault NEXUS** the original NEXUS messages can be displayed. Use the **More** button in this case, too.
- The command **BMC.SELect** selects one of the six DPU counters for further analysis.

- The **BMC.STATistic.TREE ALL** analysis shows how many counter ticks a function contributed.



The value behind **funcs** show how many different function participate to the analysis. Total shows how many counter ticks the counter was increased while the whole record.

Meaning of Columns	
range	The full symbol path of the function.
tree	The nested functions derived from the trace record.
total	Counter ticks contributed by the function and its sub functions accumulated over the total runtime.
min	The minimum counter ticks contributed by a single call of the function.
max	The maximum counter ticks. Analog to **min**.
avr	The average counter ticks. Analog to **min**.
count	Amount of functions calls. The negative value in brackets identifies how many return statements are missing for a complete analysis. If the trace is stopped at a certain time at least all functions in the call stack should have such a value.
internal	Counter ticks contributed by the function without its sub functions accumulated over the total runtime.
iavr	The average counter ticks contributed by a single call of the function without counting the sub functions.
external	Counter ticks contributed by the function without its internal code accumulated over the total runtime.
eavr	The average counter ticks contributed by a single call of the function without counting the internal code.
own	no meaning here
oavr	no meaning here
maxintr	no meaning here
maxtask	no meaning here