An introduction to MTV

Multi-Core Debug Solution (MCDS) Trace Viewer

About this document

Scope and purpose

This Application Note describes the basic usage of the MCDS (Multi-Core Debug Solution) Trace Viewer (MTV) tool.

Non-intrusive, parallel trace with MCDS is a very powerful tool to analyze and debug a hard real-time system in full operation.

Note: *MTV can only be used with Emulation Devices (ED) and production devices with miniMCDS.*

Intended audience

This Application Note is intended to introduce MTV to first-time users.

Disclaimer

Note: *MTV is a free tool without support, and does not cover the complete MCDS functionality. For a tool with full MCDS support please contact Infineon tool partners.*

Table of contents

1 Getting started

1.1 Installation

MTV is part of the DAS installation. DAS can be downloaded at **http://www.infineon.com/DAS**.

Once DAS is installed, MTV can be found in the start menu or inside the DAS installation directory. For example 'C:/Program Files/DAS/clients/mcds_trace_viewer.exe'.

1.2 First start-up

At MTV start-up the main GUI appears, providing some controls and the trace table. The trace table is of course empty at first start-up, and no content is displayed (Figure 1 item (5)).

If a device is connected to the PC (such as a USB cable to a TriBoard or with a DAP miniWiggler), a standard trace can be immediately generated by just pressing the record button (4).

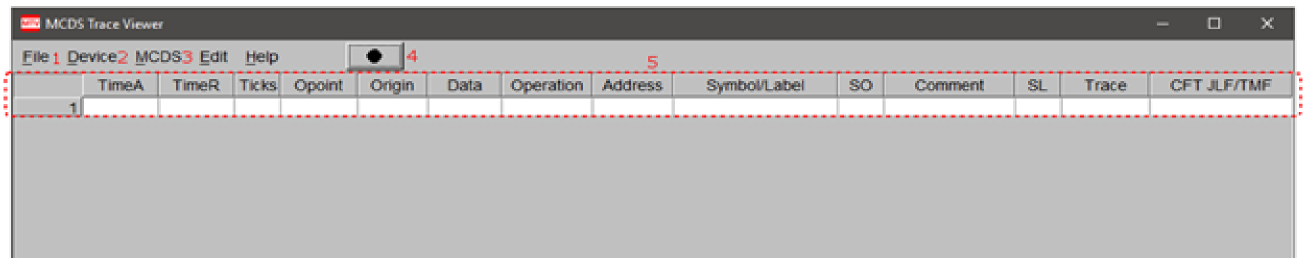


Figure 1 **Main MTV GUI**

The important controls to first take note of are the following:

1. File
   - The file menu is used to select the appropriate Executable-and-Linking-Format (ELF) file for the target HW/SW-System. It is also possible to save the current MCDS configuration or load a previously saved configuration. The trace data can also be saved (see Chapter 4.6).
2. Device
   - Under the device menu, MTV can be connected to a specific device. Please make sure that the device is powered, running and connected with the PC.
3. MCDS
   - The MCDS menu is used to set up the complete MCDS on the target system. For example, observation points, triggers, a trace buffer, and so on.
4. Record
   - The record button is used to start and stop the tracing of the target system.
5. Trace table
   - The trace table displays the trace data. The information is converted to a human readable format and arranged in different columns which will be explained in the following section.

1.3 The trace table

Note: Other controls not listed in this table are described in the next chapter where a first data trace is set up and performed.

Table 1 Description of the trace table columns

Column header	Description
TimeA	The timestamp of the trace in nano seconds. Be aware that the column default width may be too small and hides the first digits (seconds).
TimeR	TimeR displays the accumulation of the Ticks at a given time step.
Ticks	The MCDS clock Ticks between trace messages. Please note that one Tick is equal to two CPU cycles.
Opoint	Displays the Observation Point of the trace data. The observation point is the physical data acquisition point inside the SoC. For example the CPU0, CPU1, SRI bus, and so on.
Origin	The origin of the activity. In most cases this is the same as Opoint. For data trace at busses the Origin column displays the original master.
Data	The Data column displays the data written or read.
Operation	The operation being executed but not on the level of assembler mnemonics for program trace. It displays a more abstract type of the operation. For example; IP CALL, IP RET for program trace or R32, W32, R16, and so on, for data trace.
Address	The address column displays the pointer of the instruction (IP) which is being executed. If the Operation column displays an R/W Operation, the address column displays the address where data is read or written to.
Symbol / Label	The Symbol and Label column refers to the symbols and labels provided by the ELF file. Here you can see for instance which function an instruction belongs to. Note that the function here refers directly to the C-Function. In case that there is no symbol defined for a certain address range in the ELF file, the name of the HW resource is displayed (such as memory or peripheral name).
SO	Symbol Offset (SO) refers to the executed instruction or a data address. Code or data symbols in an ELF file have a start address and a range. SO is the difference to the start address. If there is no symbol defined in the ELF file the offset is displayed for the associated HW resource.
Comment	The Comment column mainly displays the mnemonic of the assembler instruction which is being executed in this time step.
SL	Stack Level (SL) indicates the current position on the stack. The stack is used to save and restore the context when a function is called or returned. Please note that the stack level is assumed as 0 at the beginning of the tracing.
Trace	This column provides some additional information concerning the tracing. For example, CFT for (Compact) Function Trace.
CFT JLF / TMF	This column contains special information for compact function trace. This is out of the scope for this basic application note.

2 First trace

In order to perform the first trace, some settings are required. These basic settings will be described in this section.

2.1 Connect the Device

As the first step, make sure your device is connected properly via a USB-Cable for a TriBoard, or otherwise via a DAP miniWiggler.

After connecting the Device to the PC, open the 'Device' menu at the top and select 'Connect Device…' or press [ALT]+[D]. A window with all the devices which are connected to the PC is shown (Figure 2).

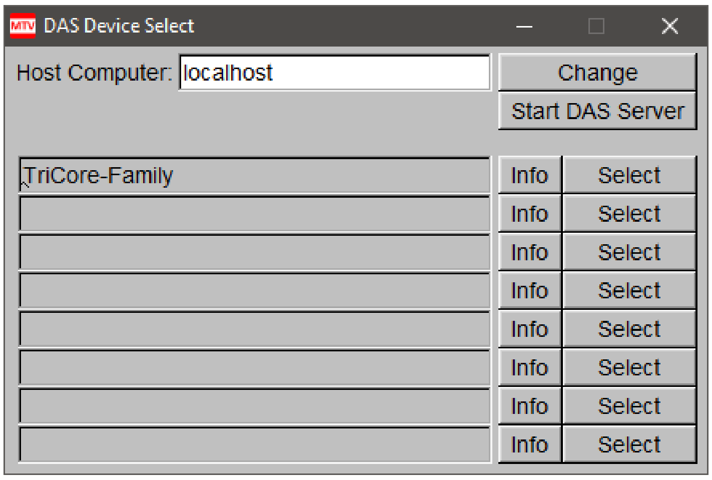
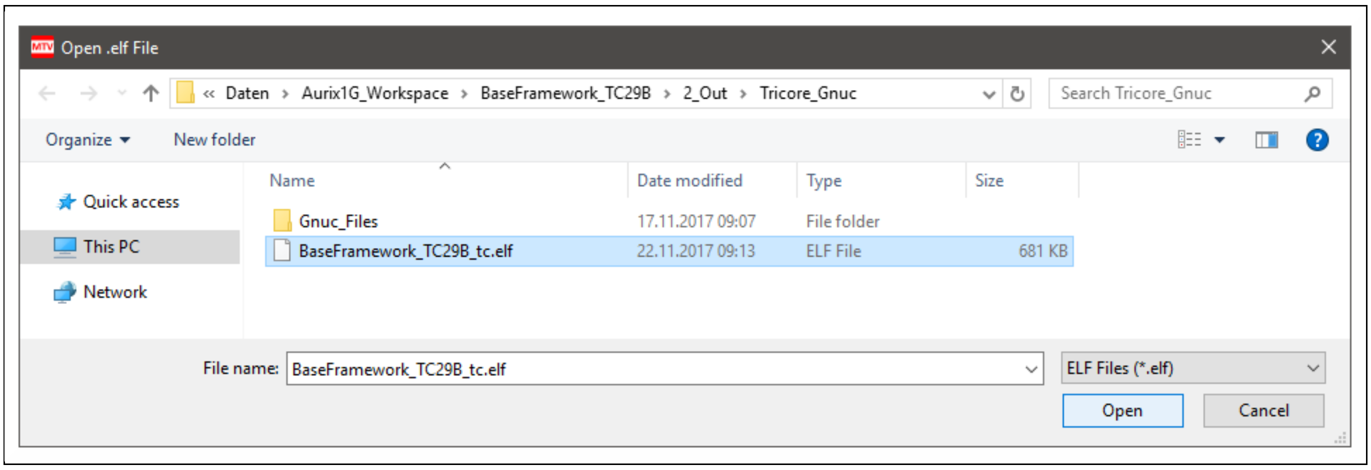


Figure 2 Device selection window

Identify the correct Device and click the 'Select' button to connect to the device. If only one device is connected to the computer it can be automatically connected by just clicking the record button.

2.2 Load the Executable and Linking Format file

To get the full information of the trace data, an ELF file is needed. To load this, simply open the 'File' menu and select 'Open .elf File…' or press [ALT]+[E]. A file browsing dialog is opened. In this dialog search for the ELF file which runs on the device and open it (Figure 3).



Name	Date modified	Type	Size
Gnuc_Files	17.11.2017 09:07	File folder	
BaseFramework_TC29B_tc.elf	22.11.2017 09:13	ELF File	681 KB

File name: BaseFramework_TC29B_tc.elf
ELF Files (*.elf)

Figure 3 ELF file browsing dialog

2.3 'Reset Device First' option

This step is not mandatory but offers a reproducible trace behavior while observing the first trace. With this option, any time a device is being traced, in any state, a reset of the device is performed first and then the tracing is started. This allows us to see the initialization sequence of the device, which is normally being executed before the main function after the device start.

To select this option, simply open the 'Device' menu and click the radio button control named 'Reset Device First' (Figure 4).

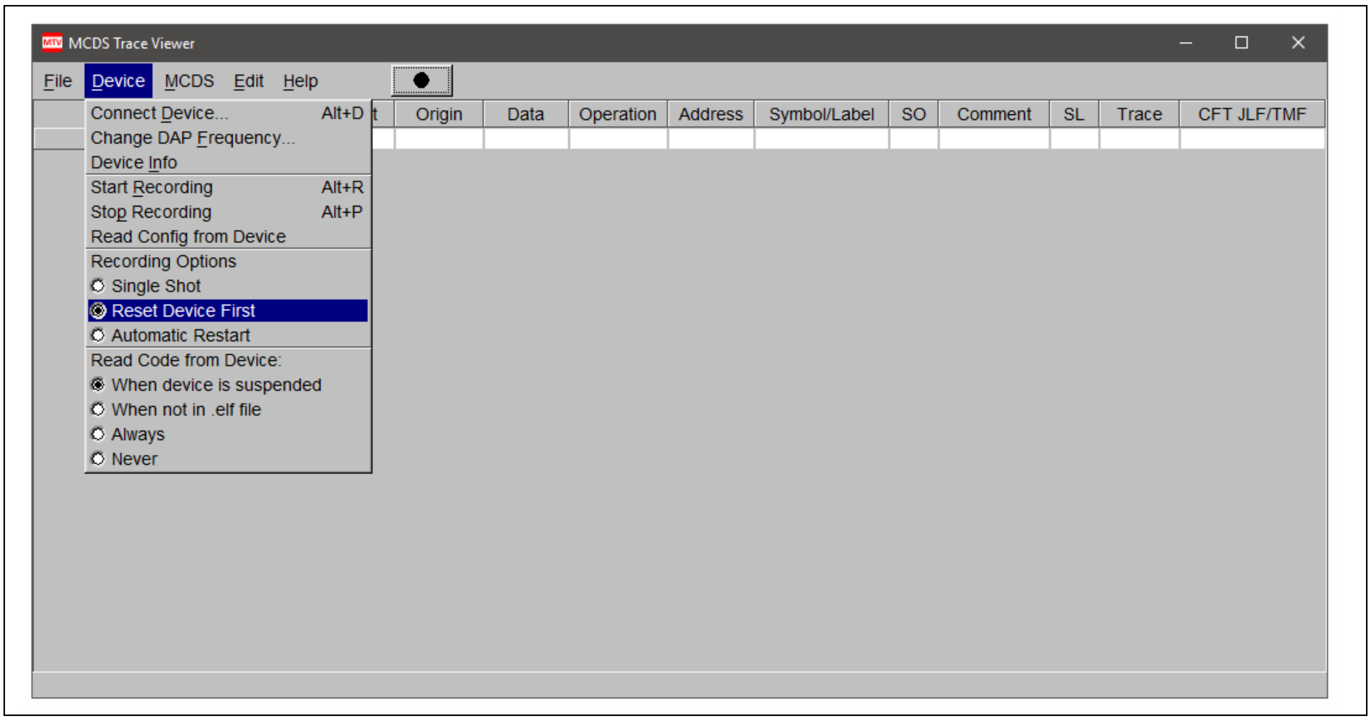


Figure 4 'Reset Device First' option turned on with the appropriate radio button

2.4 Starting the trace

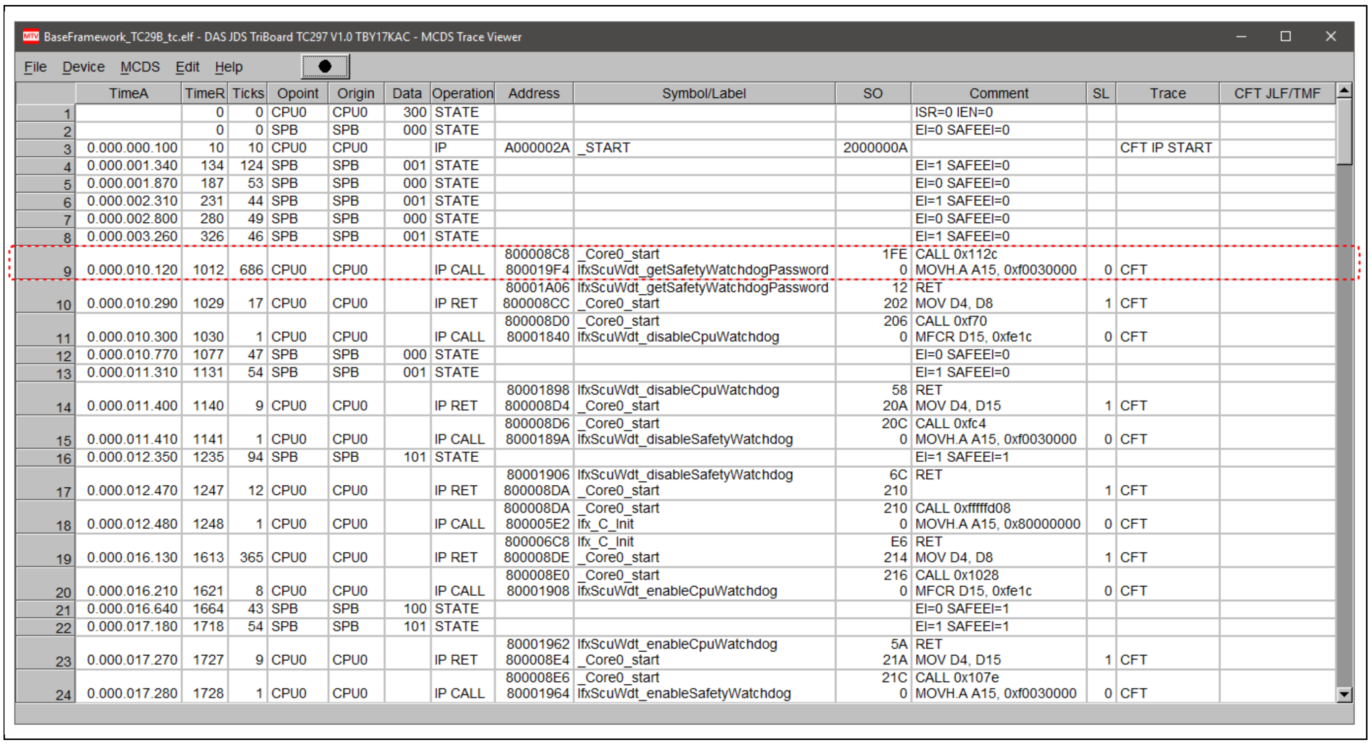
After all the settings are made, we can start our first trace by simply clicking the button at the top:

●

This button can have one of three different colors, all indicating the current status of the trace:

- Black
  - No tracing is currently running.
  - If the button is now pressed, the tracing will be started on the target device.
- Red
  - The MTV / device is currently recording.
  - If the button is now pressed again, the recording is stopped and the collected trace data will be displayed.
- Yellow
  - If a trigger condition is set for tracing, yellow indicates that the trigger condition was already met, but the trace is still running until the circular trace buffer is 'full'. Chapter 3 will cover triggers and go more into detail on this.

After the trace has finished the trace data is displayed (Figure 5).


BaseFramework_TC29B_tc.elf - DAS JDS TriBoard TC297 V1.0 TBY17KAC - MCDS Trace Viewer

	TimeA	TimeR	Ticks	Opoint	Origin	Data	Operation	Address	Symbol/Label	SO	Comment	SL	Trace	CFT JLF/TMF
1		0	0	CPU0	CPU0	300	STATE				ISR=0 IEN=0			
2		0	0	SPB	SPB	000	STATE				EI=0 SAFEEI=0			
3	0.000.000.100	10	10	CPU0	CPU0		IP	A000002A	_START	2000000A			CFT IP START	
4	0.000.001.340	134	124	SPB	SPB	001	STATE				EI=1 SAFEEI=0			
5	0.000.001.870	187	53	SPB	SPB	000	STATE				EI=0 SAFEEI=0			
6	0.000.002.310	231	44	SPB	SPB	001	STATE				EI=1 SAFEEI=0			
7	0.000.002.800	280	49	SPB	SPB	000	STATE				EI=0 SAFEEI=0			
8	0.000.003.260	326	46	SPB	SPB	001	STATE				EI=1 SAFEEI=0			
9	0.000.010.120	1012	686	CPU0	CPU0		IP CALL	800008C8 800019F4	_Core0_start IfxScuWdt_getSafetyWatchdogPassword	1FE 0	CALL 0x112c MOVH.A A15, 0xf0030000	0	CFT	
10	0.000.010.290	1029	17	CPU0	CPU0		IP RET	80001A06 800008CC	IfxScuWdt_getSafetyWatchdogPassword _Core0_start	12 202	RET MOV D4, D8	1	CFT	
11	0.000.010.300	1030	1	CPU0	CPU0		IP CALL	800008D0 80001840	_Core0_start IfxScuWdt_disableCpuWatchdog	206 0	CALL 0xf70 MFCR D15, 0xfe1c	0	CFT	
12	0.000.010.770	1077	47	SPB	SPB	000	STATE				EI=0 SAFEEI=0			
13	0.000.011.310	1131	54	SPB	SPB	001	STATE				EI=1 SAFEEI=0			
14	0.000.011.400	1140	9	CPU0	CPU0		IP RET	80001898 800008D4	IfxScuWdt_disableCpuWatchdog _Core0_start	58 20A	RET MOV D4, D15	1	CFT	
15	0.000.011.410	1141	1	CPU0	CPU0		IP CALL	800008D6 8000189A	_Core0_start IfxScuWdt_disableSafetyWatchdog	20C 0	CALL 0xfc4 MOVH.A A15, 0xf0030000	0	CFT	
16	0.000.012.350	1235	94	SPB	SPB	101	STATE				EI=1 SAFEEI=1			
17	0.000.012.470	1247	12	CPU0	CPU0		IP RET	80001906 800008DA	IfxScuWdt_disableSafetyWatchdog _Core0_start	6C 210	RET	1	CFT	
18	0.000.012.480	1248	1	CPU0	CPU0		IP CALL	800008DA 800005E2	_Core0_start Ifx_C_Init	210 0	CALL 0xfffffd08 MOVH.A A15, 0x80000000	0	CFT	
19	0.000.016.130	1613	365	CPU0	CPU0		IP RET	800006C8 800008DE	Ifx_C_Init _Core0_start	E6 214	RET MOV D4, D8	1	CFT	
20	0.000.016.210	1621	8	CPU0	CPU0		IP CALL	800008E0 80001908	_Core0_start IfxScuWdt_enableCpuWatchdog	216 0	CALL 0x1028 MFCR D15, 0xfe1c	0	CFT	
21	0.000.016.640	1664	43	SPB	SPB	100	STATE				EI=0 SAFEEI=1			
22	0.000.017.180	1718	54	SPB	SPB	101	STATE				EI=1 SAFEEI=1			
23	0.000.017.270	1727	9	CPU0	CPU0		IP RET	80001962 800008E4	IfxScuWdt_enableCpuWatchdog _Core0_start	5A 21A	RET MOV D4, D15	1	CFT	
24	0.000.017.280	1728	1	CPU0	CPU0		IP CALL	800008E6 80001964	_Core0_start IfxScuWdt_enableSafetyWatchdog	21C 0	CALL 0x107e MOVH.A A15, 0xf0030000	0	CFT	

Figure 5 The result of the first trace

Here we can observe the start-up function of the target device.

At the Symbol/Label '_Core0_start' (red dotted area) the first call inside the program creates the first CFT message. This standard configuration trace is on Function level and without data.

The MTV has much more to offer than this basic trace. Therefore the next chapter will cover a more detailed trace setting for a specific purpose.

3 Example: Data Tracing inside an Interrupt

To demonstrate a more detailed trace setup, the following code snippet is used.

Code Listing 1

```
// timer initialization missing here
IFX_INTERRUPT(Stm1_Isr, 0, 9) {
       uint32 stmTicks = (uint32)(2 * IfxStm_getFrequency(&MODULE_STM1));
       IfxStm_increaseCompare(&MODULE_STM1, IfxStm_Comparator_0, stmTicks);
while (__cmpAndSwap(&lock, 1, 0)); // ................. Acquire Lock
for (int pp = 10; pp <= 13; pp++) {
              IfxPort_togglePin(&MODULE_P33, pp);
       }
       lock = 0; // .......................................... Release Lock
}
```

The code snippet implements an Interrupt Service Routine (ISR) for the STM1 of the device.

Inside the interrupt, firstly the compare value is increased by a calculated value. With this compare value the timer fires an interrupt every 100 milliseconds.

At each interrupt, 4 LEDs on the TriBoard are toggled inside the for-loop. Due to another interrupt manipulating the same LEDs, a simple spinlock is implemented in line 4 and 8.

The chosen example goal is to trace the Spinlock-Mechanism inside the interrupt, to see if everything works as expected. This will be used for introducing MTV in more detail.

3.1 Trace setup

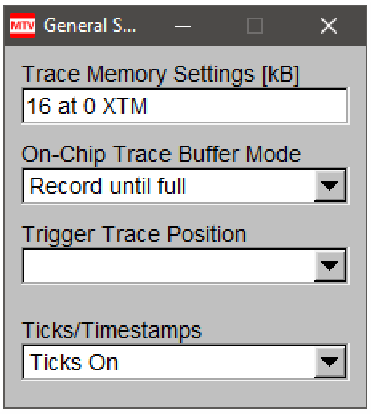
3.1.1 Trace buffer settings

First we have to precisely set-up MTV in order to get the desired trace data. The first steps for the set-up can be adapted from the previous chapter covering the device connection, ELF file loading, and device reset options.

Next, we want to increase the trace buffer size and set it to a circular buffer, which is stopped by a trigger. The settings are opened by clicking the ‘MCDS’ menu and clicking ‘General’ or pressing [ALT]+[G]. The window that appears is shown in Figure 6.

At the top of this window we can set up the trace memory. We want to use one MB of memory, instead of the default size of 16 kB, which is displayed as ‘16 at 0 XTM’. The ‘at 0 XTM’ refers to the memory which is used to store the trace data and at which position (0 means start) the trace data section starts. The definition of the start address is important for TCM (Trace/Common Memory) which can be shared between tracing and application.

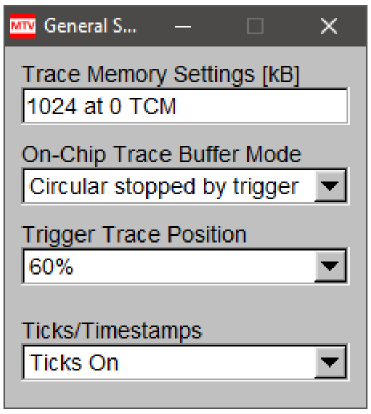
To get 1 MB of trace memory just replace the complete line with ‘1024’ and press [Enter]. MTV now asks if we want to switch to the ‘TCM’ memory, because the ‘XTM’ memory has only a size of 16 kB. Click ‘Yes’.



Trace Memory Settings [kB]: 16 at 0 XTM
On-Chip Trace Buffer Mode: Record until full
Trigger Trace Position:
Ticks/Timestamps: Ticks On

Figure 6 **Trace buffer settings (Default)**

Now we need to set-up the 'On-Chip Trace Buffer Mode'. As default, the trace buffer is filled with trace data until it is full and the tracing is stopped automatically (Record until full). We don't know at which time the interrupt will occur, and therefore we can't be sure to get the desired trace data. It could happen that the timer interrupt of our example is fired after the buffer is full. To address this case we select 'Circular stopped by trigger' (see Figure 7).



Trace Memory Settings [kB]: 1024 at 0 TCM
On-Chip Trace Buffer Mode: Circular stopped by trigger
Trigger Trace Position: 60%
Ticks/Timestamps: Ticks On

Figure 7 **Trace buffer settings (Trigger)**

Now the buffer is filled as a circular buffer (if it is full, the oldest trace data will be overwritten).

It is important now to set a trigger in order to stop the tracing. The trigger is set in another menu and will be covered later.

There are now two settings in this window, the 'Trigger Trace Position' and the 'Ticks/Timestamps'.

The Trigger Position indicates how much data will be traced after the trigger condition has been met. For example, if '40 Bytes before End' is selected here, the MCDS will trace 40 additional bytes of trace data after the trigger condition has been met and the tracing stops. This default setting maximizes the trace data before the trigger condition. In many cases the trigger is on the symptom and the interesting cause is before the trigger. The small overhang of 40 bytes ensures that relevant data around the trigger point is in the trace buffer before the trace is stopped.

A more balanced pre and post-trigger setting is the '60%' option. With this setting 40% of the buffer will be filled with trace data following after the first trigger hit.

The last setting is the setting for the ticks and timestamps of the MCDS. This document will only cover the first three options:

- Ticks Disabled
  - If this setting is used, the MCDS does not give any time or tick information. On the one hand, the trace data needs less memory but on the other there is no information on timing behavior. There is no indicator for the elapsed time. The first three columns of the trace data will remain empty.
    This option is useful to trace a longer time if no timing information is needed.
- Ticks On
  - With this default setting the MCDS shares information about the time behavior of the tracing. If some timing behaviors have to be observed, this is the right setting.
- Ticks and Timestamps
  - This setting is the same as the 'Ticks On' setting but with the difference that the MCDS gives an exact timestamp of the device up-time every 4 kB of trace data. The user can now see that the device is already running for some time and the trace data is aligned to this time.

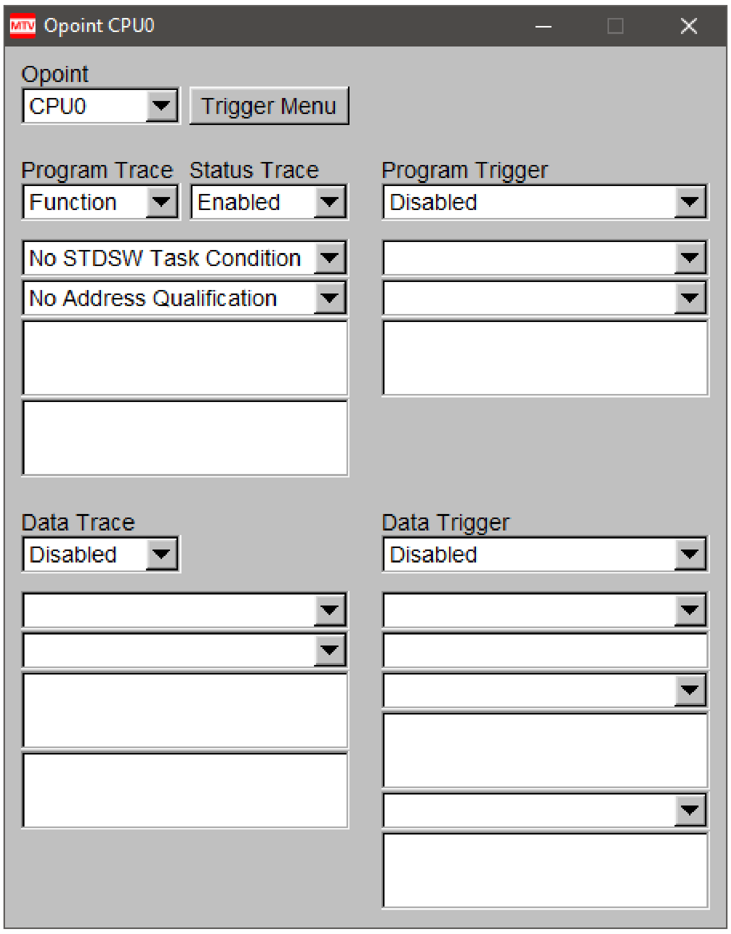
Because we don't want to trace a huge time window, we can leave the setting at the default value 'Ticks On'. If everything is set up correct, the general settings for the trace buffer should be the same as those in Figure 7.

3.1.2 Observation point setting

After the trace buffer has been set up correctly, we have to set up the observation point. Here we can set the trigger condition, the data to be traced, and the memory sections to be observed.

The window can be opened at the 'MCDS' menu by clicking on 'Opoint CPUa' or pressing [CTRL]+[P].

Right after the window has appeared, click the 'Trigger Menu' button at the upper right corner to open the trigger settings for this observation point. The window presented should appear as follows:



Opoint: CPU0
Program Trace: Function
Status Trace: Enabled
Program Trigger: Disabled
No STDSW Task Condition
No Address Qualification
Data Trace: Disabled
Data Trigger: Disabled

Figure 8 Opoint CPU0 (Default)

In this window are 4 sections:

1. Program Trace / Status Trace
2. Program Trigger
3. Data Trace
4. Data Trigger

To achieve our goal, we need to set up (1), (2) and (3) properly. The 'Data Trigger' section will not be covered in this application note.

Beside these 4 sections, the correct CPU should be selected with the 'Opoint' drop down selection. In our case this will be CPU0.

3.1.2.1 Program trace

The first section defines how detailed and at which memory sections the tracing should be performed.

Level of trace detail

For the level of detail there are three settings:

- Function
  - This is the default setting and traces only the calls and returns of functions so that a basic program flow behavior can be observed. This setting generates the minimum trace data.
- Flow
  - This setting keeps track of all instructions being executed. A trace message is generated at points where the program execution is not sequential. With this setting we can see what is going on inside the traced functions.
- Instruction
  - The Instruction setting will generate as much as possible trace messages for the program execution. This adds more details for the execution time of instructions. Note that this option needs much more trace memory than the Flow setting.

We will use the Instruction setting to achieve our goal.

Address qualification

The next important setting is the address qualification. With the address qualification you can specify in which regions of the program memory the device should be traced. There are three options available:

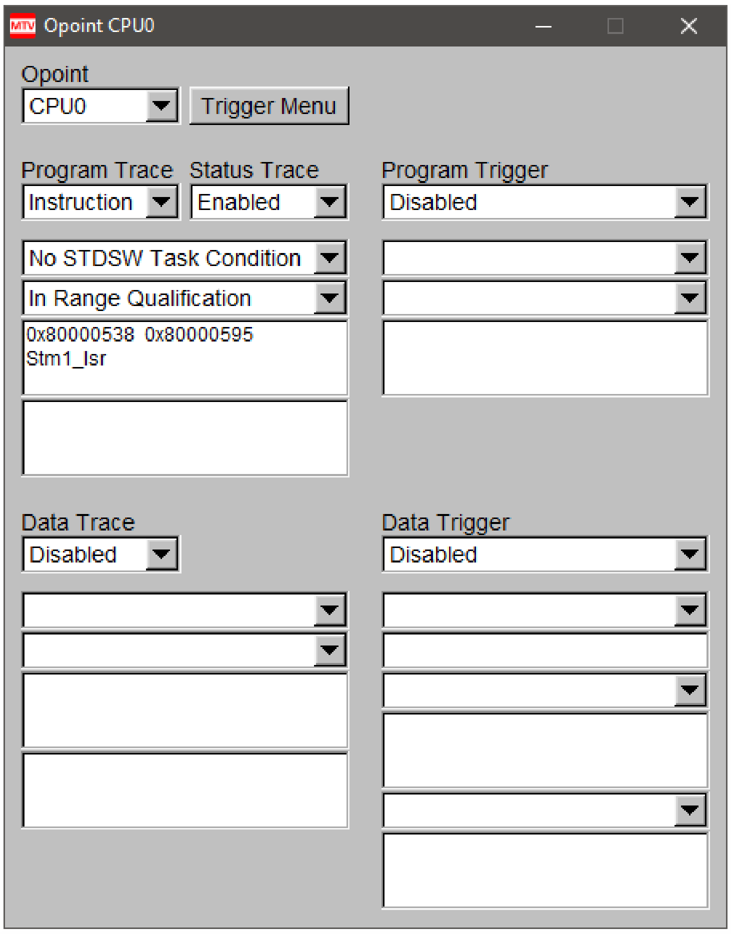
- No Address Qualification
  - With this option the complete program execution is traced.
- In Range Qualification:
  - If the In Range Qualification is used, the device is only traced if the Instruction Pointer (IP) points to an address inside of the configured ranges.
- Out Of Range Qualification
  - This is the same option as the In Range Qualification but in reverse. Here the device will only be traced if the IP is pointing to an address outside of the configured ranges.

To set the range of the address qualification the two text areas below the address qualification setting can be used.

In these text areas only the first line is used to define the address range. The range can be defined by using two hexadecimal addresses.

For instance, if one writes ‘0x8000F000 0x8000F010’ in the first line of one of the text areas, the address range for the qualification is set to ‘0x8000F000’ to ‘0x8000F010’. In the case that an ELF file is loaded, there is an easier way to set the address qualification. Just type the function name to be traced, and de-focus the text area. MTV will then automatically replace the function name with the correct address range. The function name will also be moved to the second line of the text area, showing which function is traced with the address range.

For our goal, we simply write ‘Stm1_Isr’ in the first text area and click on another control (defocus the text area). The MTV will then automatically set the right address range for us, as seen in Figure 9.


Opoint CPU0

Opoint: CPU0
Trigger Menu

Program Trace: Instruction
Status Trace: Enabled
Program Trigger: Disabled

No STDSW Task Condition
In Range Qualification
0x80000538 0x80000595
Stm1_Isr

Data Trace: Disabled
Data Trigger: Disabled

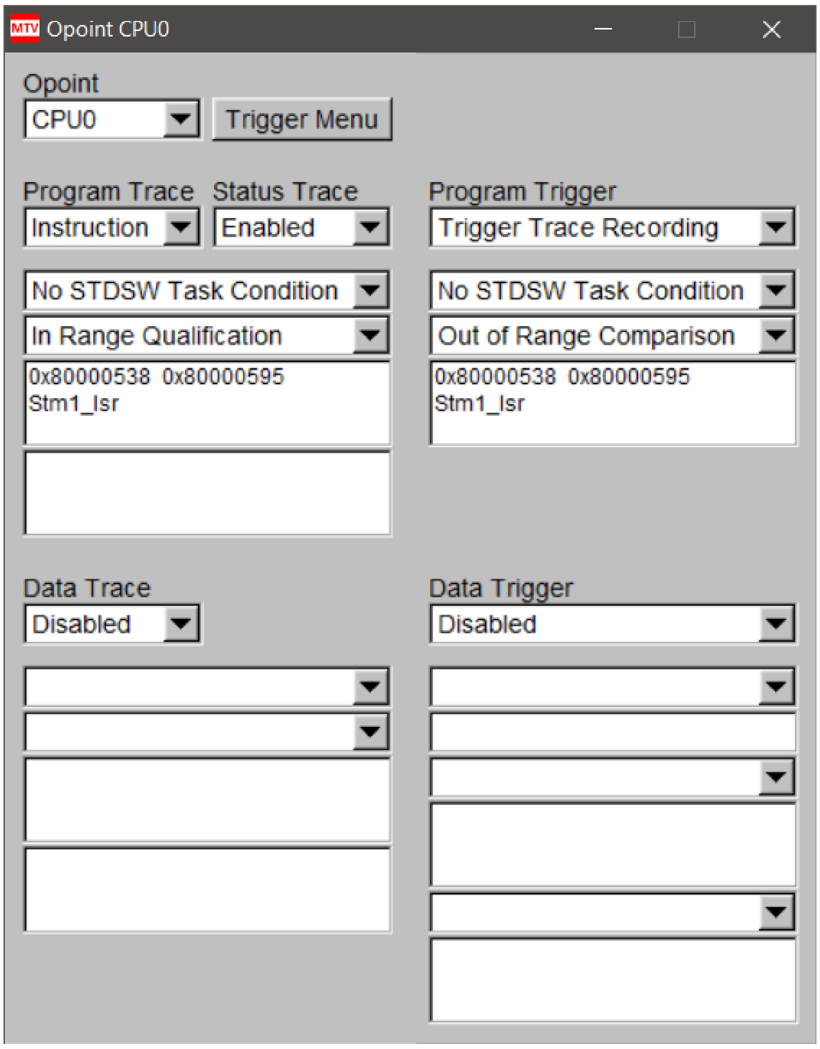
Figure 9 **Opoint CPU0 (Timer interrupt trace)**

3.1.2.2 Program Trigger

We will use the Program trigger to stop the tracing at a certain point. To set up the trigger, select 'Trigger Trace Record' on the top of the Program Trigger section, where 'Disabled' is the default. MTV will display a message regarding the trace buffer mode, if it is still set to 'Record until full' (section 3.1.1). In this case MTV automatically changes the buffer settings for the use of a trigger. Simply close this message.

The second drop down menu can be ignored, because STDSW (STandarD SoftWare) Tasks will not be covered in this application note.

The third drop down menu is again the range qualification. This time the qualification is used to trigger the device to stop tracing. If the IP points to this range, the trigger will be fired. For our goal we use the range of the timer ISR again here. Simply write 'Stm1_Isr' and defocus the text area, as done for the Program Trace section. The result should look like Figure 10.



CPU0
Instruction
Enabled
Trigger Trace Recording
No STDSW Task Condition
In Range Qualification
0x80000538 0x80000595
Stm1_Isr
No STDSW Task Condition
Out of Range Comparison
0x80000538 0x80000595
Stm1_Isr
Disabled
Disabled

Figure 10 Opoint CPU0 (Program Trigger)

3.1.2.3 Data Trace

The last section we have to modify is the Data Trace section, because we want to see which data is read and written. For this purpose select 'All' on the drop down menu at the top of the Data Trace section. The other options are self-explanatory. Use these to see only reads or writes. 'A' refers to 'address' and 'D' to 'data'. For instance, 'A+D Write' will display address and data of all write operations.

There is also a range qualification possible, to see only reads or writes in specific sections. We will not use this for now, because we already set the program trace to the ISR so that we only see reads and writes in this ISR. But we will later use this address qualification to hide some read and write operations inside the traced ISR function that we do not want to observe.

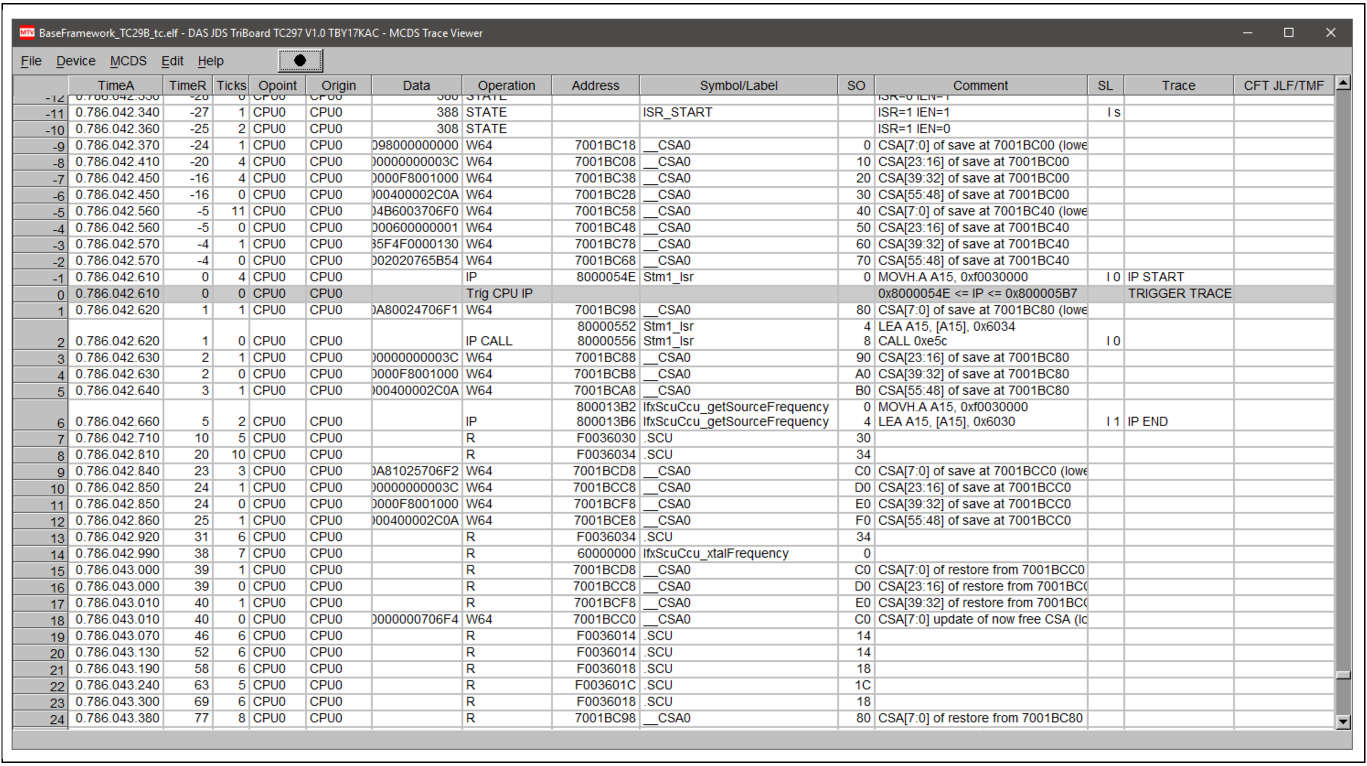
Note: *In AURIX TC2xx only the read address and not the data is available for trace and triggering. In AURIX TC3xx (after TC39x A-step) also the read data is available.*

3.2 Tracing the ISR

Now that we have set everything up, we are able to start the tracing of the timer interrupt by clicking the record button:

●

After the tracing has finished, the trace data view will be displayed. For example:


BaseFramework_TC29B_tc.elf - DAS JDS TriBoard TC297 V1.0 TBY17KAC - MCDS Trace Viewer
File
Device
MCDS
Edit
Help

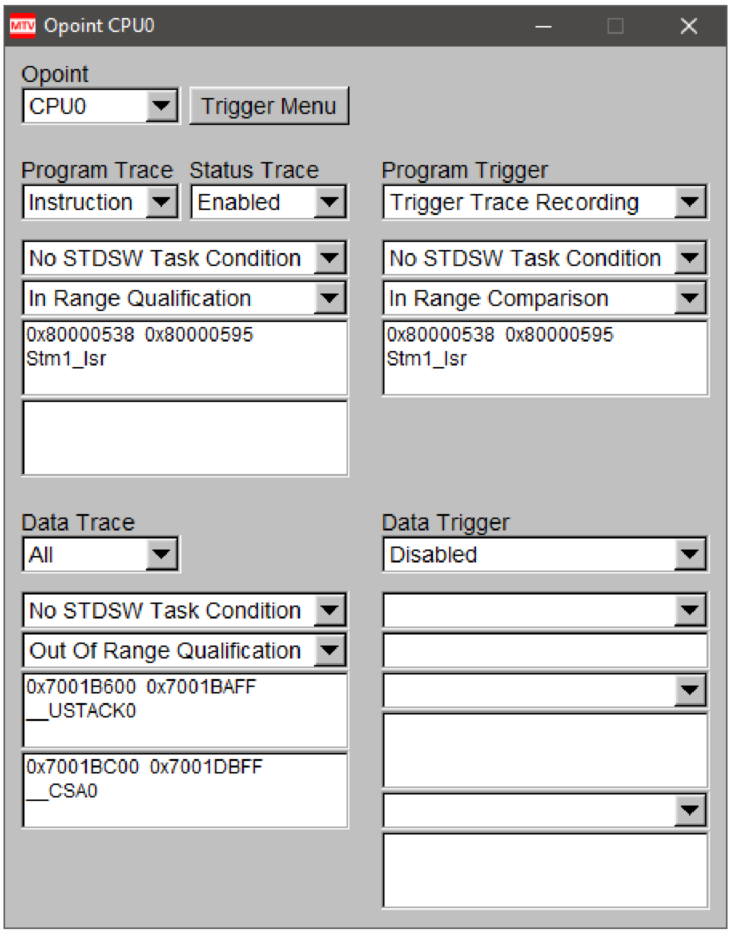
	TimeA	TimeR	Ticks	Opoint	Origin	Data	Operation	Address	Symbol/Label	SO	Comment	SL	Trace	CFT JLF/TMF
-12	0.786.042.330	-28	0	CPU0	CPU0	380	STATE				ISR=0 IEN=1			
-11	0.786.042.340	-27	1	CPU0	CPU0	388	STATE		ISR_START		ISR=1 IEN=1	I s		
-10	0.786.042.360	-25	2	CPU0	CPU0	308	STATE				ISR=1 IEN=0			
-9	0.786.042.370	-24	1	CPU0	CPU0	)98000000000	W64	7001BC18	__CSA0	0	CSA[7:0] of save at 7001BC00 (lowe			
-8	0.786.042.410	-20	4	CPU0	CPU0	)0000000003C	W64	7001BC08	__CSA0	10	CSA[23:16] of save at 7001BC00			
-7	0.786.042.450	-16	4	CPU0	CPU0	)000F8001000	W64	7001BC38	__CSA0	20	CSA[39:32] of save at 7001BC00			
-6	0.786.042.450	-16	0	CPU0	CPU0	)00400002C0A	W64	7001BC28	__CSA0	30	CSA[55:48] of save at 7001BC00			
-5	0.786.042.560	-5	11	CPU0	CPU0	)4B6003706F0	W64	7001BC58	__CSA0	40	CSA[7:0] of save at 7001BC40 (lowe			
-4	0.786.042.560	-5	0	CPU0	CPU0	)00600000001	W64	7001BC48	__CSA0	50	CSA[23:16] of save at 7001BC40			
-3	0.786.042.570	-4	1	CPU0	CPU0	35F4F0000130	W64	7001BC78	__CSA0	60	CSA[39:32] of save at 7001BC40			
-2	0.786.042.570	-4	0	CPU0	CPU0	)02020765B54	W64	7001BC68	__CSA0	70	CSA[55:48] of save at 7001BC40			
-1	0.786.042.610	0	4	CPU0	CPU0		IP	8000054E	Stm1_Isr	0	MOVH.A A15, 0xf0030000	I 0	IP START	
0	0.786.042.610	0	0	CPU0	CPU0		Trig CPU IP				0x8000054E <= IP <= 0x800005B7		TRIGGER TRACE	
1	0.786.042.620	1	1	CPU0	CPU0	)A80024706F1	W64	7001BC98	__CSA0	80	CSA[7:0] of save at 7001BC80 (lowe			
2	0.786.042.620	1	0	CPU0	CPU0		IP CALL	80000552 80000556	Stm1_Isr Stm1_Isr	4 8	LEA A15, [A15], 0x6034 CALL 0xe5c	I 0		
3	0.786.042.630	2	1	CPU0	CPU0	)0000000003C	W64	7001BC88	__CSA0	90	CSA[23:16] of save at 7001BC80			
4	0.786.042.630	2	0	CPU0	CPU0	)000F8001000	W64	7001BCB8	__CSA0	A0	CSA[39:32] of save at 7001BC80			
5	0.786.042.640	3	1	CPU0	CPU0	)00400002C0A	W64	7001BCA8	__CSA0	B0	CSA[55:48] of save at 7001BC80			
6	0.786.042.660	5	2	CPU0	CPU0		IP	800013B2 800013B6	IfxScuCcu_getSourceFrequency IfxScuCcu_getSourceFrequency	0 4	MOVH.A A15, 0xf0030000 LEA A15, [A15], 0x6030	I 1	IP END	
7	0.786.042.710	10	5	CPU0	CPU0		R	F0036030	.SCU	30				
8	0.786.042.810	20	10	CPU0	CPU0		R	F0036034	.SCU	34				
9	0.786.042.840	23	3	CPU0	CPU0	)A81025706F2	W64	7001BCD8	__CSA0	C0	CSA[7:0] of save at 7001BCC0 (lowe			
10	0.786.042.850	24	1	CPU0	CPU0	)0000000003C	W64	7001BCC8	__CSA0	D0	CSA[23:16] of save at 7001BCC0			
11	0.786.042.850	24	0	CPU0	CPU0	)000F8001000	W64	7001BCF8	__CSA0	E0	CSA[39:32] of save at 7001BCC0			
12	0.786.042.860	25	1	CPU0	CPU0	)00400002C0A	W64	7001BCE8	__CSA0	F0	CSA[55:48] of save at 7001BCC0			
13	0.786.042.920	31	6	CPU0	CPU0		R	F0036034	.SCU	34				
14	0.786.042.990	38	7	CPU0	CPU0		R	60000000	IfxScuCcu_xtalFrequency	0				
15	0.786.043.000	39	1	CPU0	CPU0		R	7001BCD8	__CSA0	C0	CSA[7:0] of restore from 7001BCC0			
16	0.786.043.000	39	0	CPU0	CPU0		R	7001BCC8	__CSA0	D0	CSA[23:16] of restore from 7001BC(			
17	0.786.043.010	40	1	CPU0	CPU0		R	7001BCF8	__CSA0	E0	CSA[39:32] of restore from 7001BC(			
18	0.786.043.010	40	0	CPU0	CPU0	)000000706F4	W64	7001BCC0	__CSA0	C0	CSA[7:0] update of now free CSA (lc			
19	0.786.043.070	46	6	CPU0	CPU0		R	F0036014	.SCU	14				
20	0.786.043.130	52	6	CPU0	CPU0		R	F0036014	.SCU	14				
21	0.786.043.190	58	6	CPU0	CPU0		R	F0036018	.SCU	18				
22	0.786.043.240	63	5	CPU0	CPU0		R	F003601C	.SCU	1C				
23	0.786.043.300	69	6	CPU0	CPU0		R	F0036018	.SCU	18				
24	0.786.043.380	77	8	CPU0	CPU0		R	7001BC98	__CSA0	80	CSA[7:0] of restore from 7001BC80			

Figure 11 First ISR tracing

The trigger that stopped the tracing is highlighted in grey by MTV.

We can now observe that there are many read and write operations that don't belong to our goal and that are making the whole trace data less readable. In this case these read and write operations are from the Context-Save-Array '__CSA0' and from the stack '__USTACK0'. To exclude these two we now use the 'Out Of Range Qualification' of the Data Trace section.

Put '__CSA0' in the first text area and '__USTACK0' in the second and change the qualification to 'Out Of Range Qualification'. The trace settings should now look like Figure 12.



Opoint: CPU0 · Trigger Menu

Program Trace: Instruction
Status Trace: Enabled
No STDSW Task Condition
In Range Qualification
0x80000538 0x80000595
Stm1_Isr

Program Trigger: Trigger Trace Recording
No STDSW Task Condition
In Range Comparison
0x80000538 0x80000595
Stm1_Isr

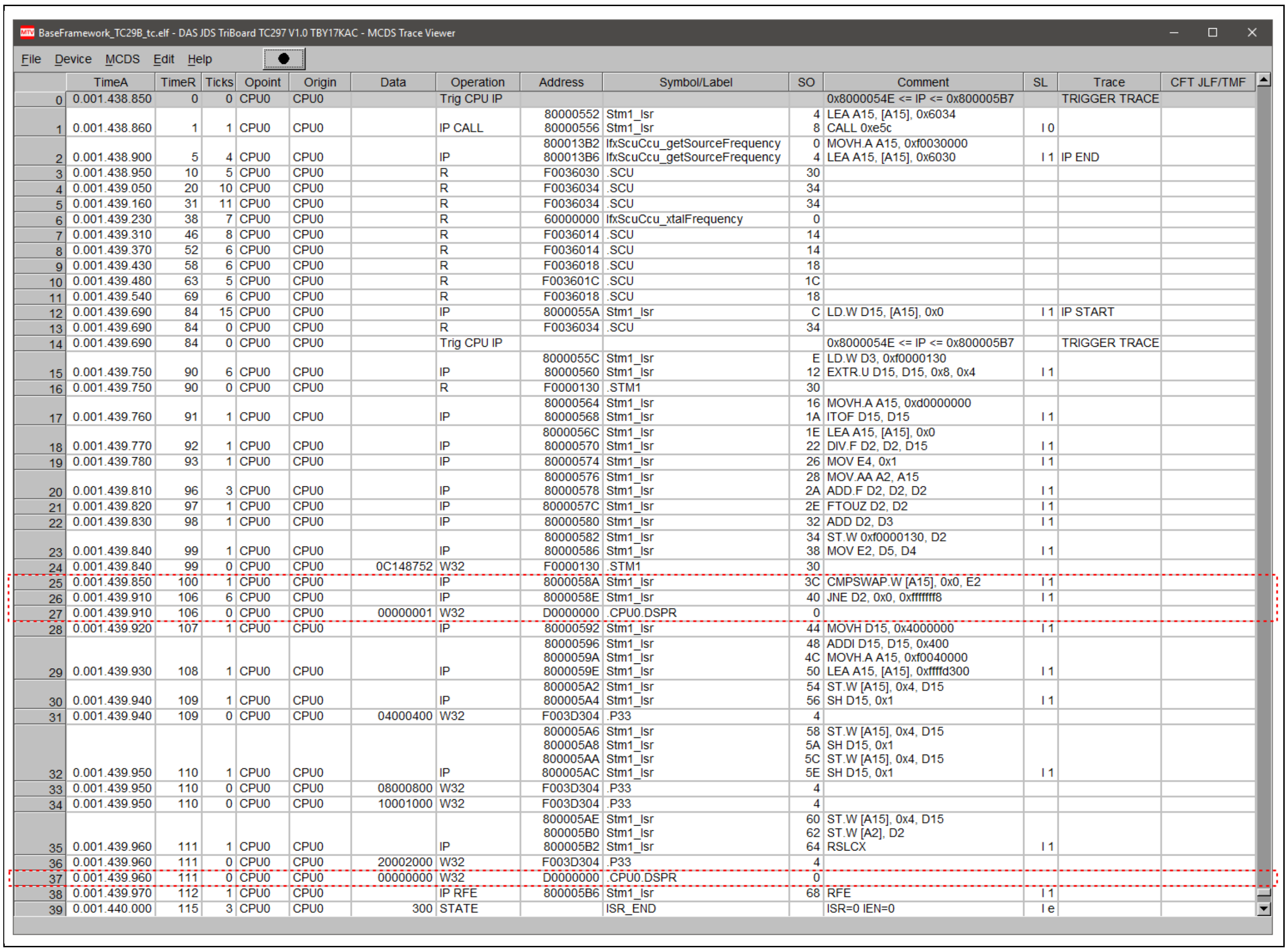
Data Trace: All
No STDSW Task Condition
Out Of Range Qualification
0x7001B600 0x7001BAFF
__USTACK0
0x7001BC00 0x7001DBFF
__CSA0

Data Trigger: Disabled

Figure 12 Opoint CPU0 (Final)

Now after rerunning the tracing we should be able to get the trace data without the read and write operations of the context saving and stack operations, and get a clearer view of the instructions belonging to our code (Figure 13).

Example: Data Tracing inside an Interrupt


BaseFramework_TC29B_tc.elf - DAS JDS TriBoard TC297 V1.0 TBY17KAC - MCDS Trace Viewer
File
Device
MCDS
Edit
Help

	TimeA	TimeR	Ticks	Opoint	Origin	Data	Operation	Address	Symbol/Label	SO	Comment	SL	Trace	CFT JLF/TMF
0	0.001.438.850	0	0	CPU0	CPU0		Trig CPU IP				0x8000054E <= IP <= 0x800005B7		TRIGGER TRACE	
1	0.001.438.860	1	1	CPU0	CPU0		IP CALL	80000552 80000556	Stm1_Isr Stm1_Isr	4 8	LEA A15, [A15], 0x6034 CALL 0xe5c	I 0		
2	0.001.438.900	5	4	CPU0	CPU0		IP	800013B2 800013B6	IfxScuCcu_getSourceFrequency IfxScuCcu_getSourceFrequency	0 4	MOVH.A A15, 0xf0030000 LEA A15, [A15], 0x6030	I 1	IP END	
3	0.001.438.950	10	5	CPU0	CPU0		R	F0036030	.SCU	30				
4	0.001.439.050	20	10	CPU0	CPU0		R	F0036034	.SCU	34				
5	0.001.439.160	31	11	CPU0	CPU0		R	F0036034	.SCU	34				
6	0.001.439.230	38	7	CPU0	CPU0		R	60000000	IfxScuCcu_xtalFrequency	0				
7	0.001.439.310	46	8	CPU0	CPU0		R	F0036014	.SCU	14				
8	0.001.439.370	52	6	CPU0	CPU0		R	F0036014	.SCU	14				
9	0.001.439.430	58	6	CPU0	CPU0		R	F0036018	.SCU	18				
10	0.001.439.480	63	5	CPU0	CPU0		R	F003601C	.SCU	1C				
11	0.001.439.540	69	6	CPU0	CPU0		R	F0036018	.SCU	18				
12	0.001.439.690	84	15	CPU0	CPU0		IP	8000055A	Stm1_Isr	C	LD.W D15, [A15], 0x0	I 1	IP START	
13	0.001.439.690	84	0	CPU0	CPU0		R	F0036034	.SCU	34				
14	0.001.439.690	84	0	CPU0	CPU0		Trig CPU IP				0x8000054E <= IP <= 0x800005B7		TRIGGER TRACE	
15	0.001.439.750	90	6	CPU0	CPU0		IP	8000055C 80000560	Stm1_Isr Stm1_Isr	E 12	LD.W D3, 0xf0000130 EXTR.U D15, D15, 0x8, 0x4	I 1		
16	0.001.439.750	90	0	CPU0	CPU0		R	F0000130	.STM1	30				
17	0.001.439.760	91	1	CPU0	CPU0		IP	80000564 80000568	Stm1_Isr Stm1_Isr	16 1A	MOVH.A A15, 0xd0000000 ITOF D15, D15	I 1		
18	0.001.439.770	92	1	CPU0	CPU0		IP	8000056C 80000570	Stm1_Isr Stm1_Isr	1E 22	LEA A15, [A15], 0x0 DIV.F D2, D2, D15	I 1		
19	0.001.439.780	93	1	CPU0	CPU0		IP	80000574	Stm1_Isr	26	MOV E4, 0x1	I 1		
20	0.001.439.810	96	3	CPU0	CPU0		IP	80000576 80000578	Stm1_Isr Stm1_Isr	28 2A	MOV.AA A2, A15 ADD.F D2, D2, D2	I 1		
21	0.001.439.820	97	1	CPU0	CPU0		IP	8000057C	Stm1_Isr	2E	FTOUZ D2, D2	I 1		
22	0.001.439.830	98	1	CPU0	CPU0		IP	80000580	Stm1_Isr	32	ADD D2, D3	I 1		
23	0.001.439.840	99	1	CPU0	CPU0		IP	80000582 80000586	Stm1_Isr Stm1_Isr	34 38	ST.W 0xf0000130, D2 MOV E2, D5, D4	I 1		
24	0.001.439.840	99	0	CPU0	CPU0	0C148752	W32	F0000130	.STM1	30				
25	0.001.439.850	100	1	CPU0	CPU0		IP	8000058A	Stm1_Isr	3C	CMPSWAP.W [A15], 0x0, E2	I 1		
26	0.001.439.910	106	6	CPU0	CPU0		IP	8000058E	Stm1_Isr	40	JNE D2, 0x0, 0xfffffff8	I 1		
27	0.001.439.910	106	0	CPU0	CPU0	00000001	W32	D0000000	.CPU0.DSPR	0				
28	0.001.439.920	107	1	CPU0	CPU0		IP	80000592	Stm1_Isr	44	MOVH D15, 0x4000000	I 1		
29	0.001.439.930	108	1	CPU0	CPU0		IP	80000596 8000059A 8000059E	Stm1_Isr Stm1_Isr Stm1_Isr	48 4C 50	ADDI D15, D15, 0x400 MOVH.A A15, 0xf0040000 LEA A15, [A15], 0xffffd300	I 1		
30	0.001.439.940	109	1	CPU0	CPU0		IP	800005A2 800005A4	Stm1_Isr Stm1_Isr	54 56	ST.W [A15], 0x4, D15 SH D15, 0x1	I 1		
31	0.001.439.940	109	0	CPU0	CPU0	04000400	W32	F003D304	.P33	4				
32	0.001.439.950	110	1	CPU0	CPU0		IP	800005A6 800005A8 800005AA 800005AC	Stm1_Isr Stm1_Isr Stm1_Isr Stm1_Isr	58 5A 5C 5E	ST.W [A15], 0x4, D15 SH D15, 0x1 ST.W [A15], 0x4, D15 SH D15, 0x1	I 1		
33	0.001.439.950	110	0	CPU0	CPU0	08000800	W32	F003D304	.P33	4				
34	0.001.439.950	110	0	CPU0	CPU0	10001000	W32	F003D304	.P33	4				
35	0.001.439.960	111	1	CPU0	CPU0		IP	800005AE 800005B0 800005B2	Stm1_Isr Stm1_Isr Stm1_Isr	60 62 64	ST.W [A15], 0x4, D15 ST.W [A2], D2 RSLCX	I 1		
36	0.001.439.960	111	0	CPU0	CPU0	20002000	W32	F003D304	.P33	4				
37	0.001.439.960	111	0	CPU0	CPU0	00000000	W32	D0000000	.CPU0.DSPR	0				
38	0.001.439.970	112	1	CPU0	CPU0		IP RFE	800005B6	Stm1_Isr	68	RFE	I 1		
39	0.001.440.000	115	3	CPU0	CPU0	300	STATE		ISR_END		ISR=0 IEN=0	I e		

Figure 13 ISR-Trace without Context Save and Stack operations

The trigger that stopped the tracing is again marked in grey (at the top) by MTV. The area marked with the red dotted border in the figure above displays the instructions representing the spinlock mechanism. We can observe that firstly a Compare-And-Swap instruction is executed, which in this case at first attempt acquires the lock variable.

Right after the Compare-And-Swap instruction a Jump-Not-Equal instruction follows, which would jump back before the Compare-And-Swap instruction to repeat it, as long as the lock variable is acquired by somebody else.

While acquiring the lock variable it is set to 1, which can be seen at the W32 command right after the JNE instruction in line 27. After the critical code section has been executed, the lock variable is set to 0 to release the spinlock mechanism for the other timer interrupt. The release of the lock variable can be seen at the W32 instruction in line 37.

Note: *There can be a certain delay between the data trace messages (here W32) and the associated instructions (here CMPSWAP and ST.W). This is due to the different stages at the CPU pipeline where this information is retrieved.*

4 Common errors and further hints

This section discusses common errors and gives further hints for the correct usage of the MTV.

4.1 No data traced or wrong data traced

4.1.1 Halted CPU

A common error while using MTV is, that the CPU is halted for instance by another debugging tool. In this case the record button stays red and no trace data is generated since even the tick message generation only starts after the first regular trace message.

4.1.2 Volatile programming

While writing and testing software on the SoC, volatile RAM is often used to save limited write cycles of the flash memory. If MTV is used to collect trace data with the option 'Reset Device First', make sure that the software was written to the permanent flash instead of RAM. Otherwise a reset will result in tracing the execution of the old software which is stored in the permanent flash.

4.1.3 Wrong trigger

If a circular buffer is used for tracing, meaning that the trace is in an infinite loop until the first trigger condition is met, MTV can be stuck in an infinite tracing loop. This happens if the trigger is wrong and doesn't hit. In that case the tracing will never stop. Another problem might be that there was no trigger set at all.

4.2 Zero ticks

For certain traces the column displaying the 'Ticks' of the CPU reads zero. At first this might be confusing and looks like an error inside the CPU or the MTV but this is not the case.

There are two main reasons why this can happen.

- There may be two different trace units (Program Trace Unit PTU and Data Trace Unit DTU) or even at different observation points (Core1 and Core2 for example) which are operating independently. Therefore it will happen that two trace messages are generated at the same time. The second trace message is then displayed with zero Ticks.
- A second reason belongs to the reduced clock frequency of the MCDS. As mentioned, one Tick is equal to two CPU cycles, therefore the clock of the MCDS runs with half the frequency of the CPU. Due to this clock difference, it might happen that there are more trace messages observed at a time. If this happens there are for instance two trace messages which in reality have different time stamps, but the same timestamp while being observed by the MCDS. The second trace message then has zero Ticks.

4.3 Negative stack level

The Stack Level indicates the current position on the stack for the function context.

Normally the main function is on Stack Level 0. If now a function is called, the context is stored on the stack and the Stack Level is incremented to 1. After executing the function, a return operation is called which restores the context of the main function from the stack. This results in a decreasing Stack Level.

Take for instance the following source code:

```
void func_2() {
      calc_something();
}
void func_1() {
      func_2();
}
int main(void) {
      func_1();
      return 0;
}
```

This generates the Stack Level history in the following table:

Table 2 Stack level history

main	func_1 called	func_2 called	func_2 return	func_1 return
`SL = 0`	`SL = 1`	`SL = 2`	`SL = 1`	`SL = 0`

If the data tracing is now started without a reset of the device, the Stack Level may be on some higher value because a function f_2 is in execution. In fact the Stack Level observed by the MCDS always starts with 0. If then f_2 returns and the context of the function f_1, which has called f_2, is restored, the Stack Level decreases to -1.

4.4 Unable to connect the device

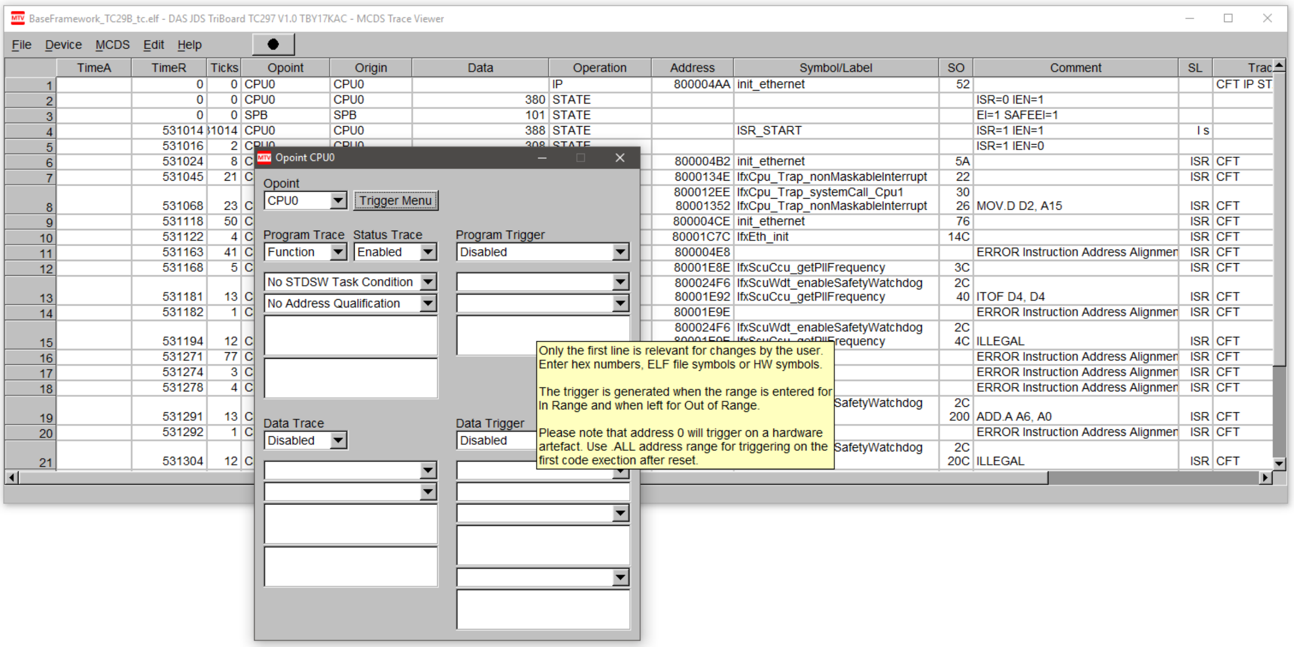
There may be several reasons why MTV cannot connect to the device:

- The device is not plugged in on the PC or the USB cable is damaged.
- The device is not powered or the power supply is damaged.
- Another debugging tools hardware is connected to the device.
- The required DAS USB drivers are not installed correctly.

Note: Please read the DAS Release notes that are included in the DAS installation.

4.5 Tooltips

Most, but not all of the controls in MTV (buttons, text areas, drop down menus, and so on) provide usage-hints as tooltips. These tooltips are displayed, after a brief pause, when you hover the cursor over a control. An example is shown in Figure 14.


BaseFramework_TC29B_tc.elf - DAS JDS TriBoard TC297 V1.0 TBY17KAC - MCDS Trace Viewer
File
Device
MCDS
Edit
Help

	TimeA	TimeR	Ticks	Opoint	Origin	Data	Operation	Address	Symbol/Label	SO	Comment	SL	Trac
1		0	0	CPU0	CPU0		IP	800004AA	init_ethernet	52			CFT IP ST
2		0	0	CPU0	CPU0	380	STATE				ISR=0 IEN=1		
3		0	0	SPB	SPB	101	STATE				EI=1 SAFEEI=1		
4		531014	31014	CPU0	CPU0	388	STATE		ISR_START		ISR=1 IEN=1	I s	
5		531016	2	CPU0	CPU0	308	STATE				ISR=1 IEN=0		
6		531024	8	C				800004B2	init_ethernet	5A		ISR	CFT
7		531045	21	C				8000134E	IfxCpu_Trap_nonMaskableInterrupt	22		ISR	CFT
8		531068	23	C				800012EE 80001352	IfxCpu_Trap_systemCall_Cpu1 IfxCpu_Trap_nonMaskableInterrupt	30 26	MOV.D D2, A15	ISR	CFT
9		531118	50	C				800004CE	init_ethernet	76		ISR	CFT
10		531122	4	C				80001C7C	IfxEth_init	14C		ISR	CFT
11		531163	41	C				800004E8			ERROR Instruction Address Alignmen	ISR	CFT
12		531168	5	C				80001E8E	IfxScuCcu_getPllFrequency	3C		ISR	CFT
13		531181	13	C				800024F6 80001E92	IfxScuWdt_enableSafetyWatchdog IfxScuCcu_getPllFrequency	2C 40	ITOF D4, D4	ISR	CFT
14		531182	1	C				80001E9E			ERROR Instruction Address Alignmen	ISR	CFT
15		531194	12	C				800024F6 [illegible]	IfxScuWdt_enableSafetyWatchdog IfxScuCcu_getPllFrequency	2C 4C	ILLEGAL	ISR	CFT
16		531271	77	C							ERROR Instruction Address Alignmen	ISR	CFT
17		531274	3	C							ERROR Instruction Address Alignmen	ISR	CFT
18		531278	4	C							ERROR Instruction Address Alignmen	ISR	CFT
19		531291	13	C					SafetyWatchdog	2C 200	ADD.A A6, A0	ISR	CFT
20		531292	1	C							ERROR Instruction Address Alignmen	ISR	CFT
21		531304	12	C					SafetyWatchdog	2C 20C	ILLEGAL	ISR	CFT

Opoint CPU0

Opoint: CPU0 — Trigger Menu

Program Trace: Function
Status Trace: Enabled
Program Trigger: Disabled
No STDSW Task Condition
No Address Qualification

Data Trace: Disabled
Data Trigger: Disabled

Only the first line is relevant for changes by the user.
Enter hex numbers, ELF file symbols or HW symbols.

The trigger is generated when the range is entered for In Range and when left for Out of Range.

Please note that address 0 will trigger on a hardware artefact. Use .ALL address range for triggering on the first code exection after reset.

Figure 14 Tooltips inside the MTV

4.6 File handling

Besides loading an ELF file, it is possible to save the current MCDS configuration and load it again later. The trace data can also be saved, which can then be processed by other tools, such as Excel or Matlab for example.

If you intend to post-process the data with Microsoft Excel, MTV already provides an appropriate handler for the copy and paste of trace data. To copy the data open the 'Edit' menu and click 'Select All', after everything is selected (the trace data will be highlighted in grey), again open the 'Edit' menu and click 'Copy'. The same result can be achieved by pressing [Ctrl]+[A] followed by [Ctrl]+[C]. It is also possible to select specific trace lines or ranges with the mouse by using the normal Windows conventions.

After the trace data has been copied to the clipboard, you can paste it directly into an Excel table. The trace data is then automatically parsed into several rows and columns within the Excel sheet.

4.7 Address qualification – Function not found

It is possible to type the function name into the text area for the address qualification. Under some circumstances this will fail. The entered name will then disappear and no address range is inserted by MTV. This happens if the function is existent in the C-Code of the application, but has been removed due to optimizations performed by the compiler.

Take the following code for instance:

```
void func() {
        calc_something();
}
int main(void) {
        func();
}
```

can be optimized to:

```
int main(void) {
        calc_something();
}
```

In this case it is not possible to enter ‘func’ as a range qualifier, only ‘main’ or ‘calc_something’ (except ‘calc_something’ has been removed to further optimizations).

To prevent the compiler of removing some specific function, the ‘volatile’ keyword can be used:

```
volatile void func() {
        calc_something();
}
int main(void) {
        func();
        return 0;
}
```

With this keyword the return type ‘void’ of the function is marked as ‘volatile’ and the compiler is not allowed to remove the function ‘func’, because of the volatile return type.

With this trick the function can be used as a range qualifier, even if the compiler wants to remove it due to optimization reasons.

Obviously an alternative is to turn off the optimization instead of using this solution, but most of the time the compiler optimization is desired.

5 Acronyms

The following table shows the most common acronyms used in the MTV GUI and this document:

Table 3 Acronyms

Acronym	Explanation
A	Address
A+D	Address + Data
CPU	Central Processing Unit
DAP	Device Access Port
DAS	Device Access Server
DMA	Direct Memory Access
ELF	Executable and Linking Format
EMEM	Emulation Memory
GUI	Graphical User Interface
INT	Interrupt
IP	Instruction Pointer
IS	See 'INT TS8_IS' in the manual
LMU	Local bus Memory Unit
MCDS	Multi-Core Debug Solution
OCDS	On-Chip Debug Support
OLDA	Online Data Acquisition
OTGB	OCDS Trigger Bus
PMU	Program Memory Unit
R/W	Read/Write
SMU	Safety Management Unit
SP	Service Provider
SPA	See 'INT TS8_SPA' in the manual
SRN	Service Request Node
SSI	Single Signal Interface
STDSW Task	Standard Software Task
TCM	Trace and Common Memory (part of EMEM)
TS	Trigger Set
XCM	Extended Common Memory (part of EMEM)
XTM	Extra Trace Memory (part of EMEM)

Revision history

Document version	Date of release	Description of changes
V1.0	February 2018	First release

Edition 2018-02-07

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32401