



TRACE32[®] GTM Debug and Trace



1. TRACE32 brief introduction
2. GTM Debug
3. GTM Trace
4. Our role with GTM

AGENDA



AGENDA

1. **TRACE32 brief introduction**
2. **GTM Debug**
 1. **Debug prerequisites**
 1. Hardware interface
 2. define access path
 3. define debug registers
 2. **Debug functions**
 1. Go/Break/Step
 2. Breakpoints
 3. High-level language debugging
 4. Inline Assembler
 5. Var
 6. Call back
 7. Peripheral View of GTM registers
 8. Peripheral View of Memory
 9. GTM v4.1 new features
3. **GTM Trace**
 1. **Trace prerequisites**
 1. Hardware interface
 2. on-chip or off-chip trace
 3. define trace encoder
 4. define trace protocol
 5. define trace path
 6. define trace triggers
 7. how to identify the trace from main core or GTM
 2. **Trace functions**
 1. Multi Channel Sequencer (MCS)
 2. Advanced Routing Unit (ARU)
 3. I/O Channels (TIM, TOM, ATOM and TIO)
 4. Digital PLL Module (DPLL)
 5. Sensor Pattern Evaluation (SPE)
 6. Time Base Unit (TBU)
4. **Our role with GTM**



TRACE32 introduction

TRACE32 introduction

Lauterbach GmbH



1979



Company founded by
Lothar Lauterbach

1983



TRACE80 market launch

2009



New headquarter in
Hoehenkirchen/Germany
(near Munich)

2013



Global reach by
10 branch offices and
20+ distribution partners

Today



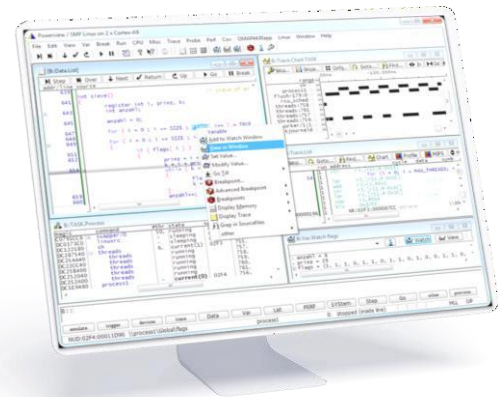
Well-established
international company
with 130+ employees



TRACE32 introduction

TRACE32® Modular System

PowerView, feature-rich debug and trace software solution



TRACE32® MODULAR SYSTEM



PowerDebug (x50)
Base Module

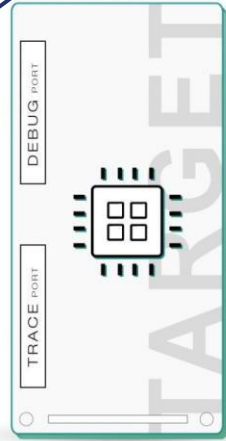
PowerTrace
Off-chip Trace Extension



Debug Probe



Trace Probe



Platform-specific Debug Probes



PowerDebug debug accelerators (base modules)



PowerTrace high-speed trace extensions (parallel and serial)



Trace Probes, i.e. passive cables, trace preprocessors (accelerators) and adapters



GTM 4.x China debug & trace open-source group





GTM Debug

Debug prerequisites

- Hardware interface
- define access path
- define debug registers

Debug functions

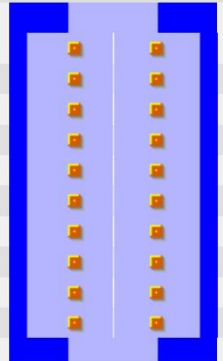
- Go/Break/Step
- Breakpoints
- High-level language debugging
- Inline Assembler
- Var
- Call back
- Peripheral View of GTM registers
- Peripheral View of Memory
- GTM v4.1 new features

GTM Debug

Debug prerequisites

> Hardware interface

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	VSUPPLY (not used)
TRST-	3	4	GND
TDI	5	6	GND
TMS TMSC SWDIO	7	8	GND
TCK TCKC SWCLK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND
RESET-	15	16	GND
DBGRQ	17	18	GND
DBGACK	19	20	GND

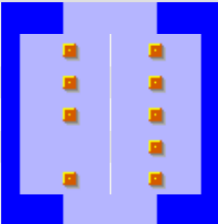


Target connector, top-view

> JTAG-20

- > 2.54mm

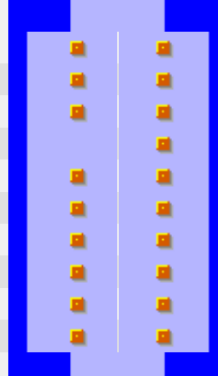
Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS TMSC SWDIO
GND	3	4	TCK TCKC SWCLK
GND	5	6	TDO - SWO
GND (KEY)	7	8	TDI
GND	9	10	RESET-



> MIPI-10

- > 1.27 mm

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS TMSC SWDIO
GND	3	4	TCK TCKC SWCLK
GND	5	6	TDO - SWO
GND (KEY)	7	8	TDI
GND	9	10	RESET-
GND	11	12	TRC CLK
GND	13	14	TRC DATA[0]
GND	15	16	TRC DATA[1]
GND	17	18	TRC DATA[2]
GND	19	20	TRC DATA[3]



> MIPI-20T

- > 1.27 mm
- > Use JTAG signals only



GTM Debug in China

Debug prerequisites

- > define access path
 - > Arm CoreSight system access path
 - > Romtable
 - > JTAG chain access path

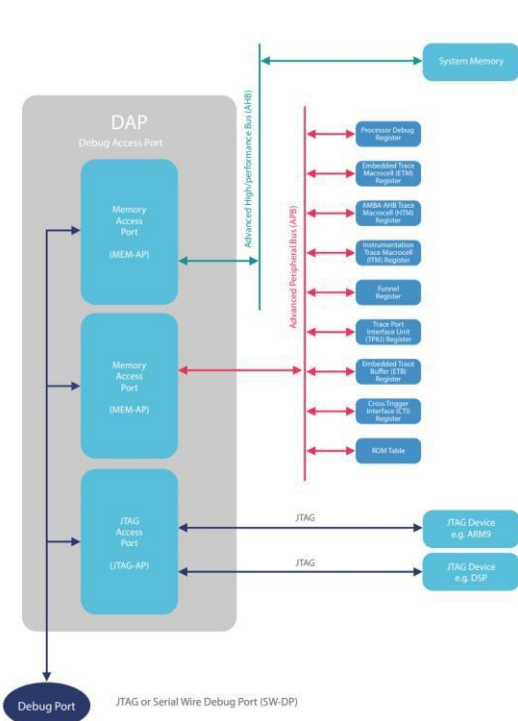
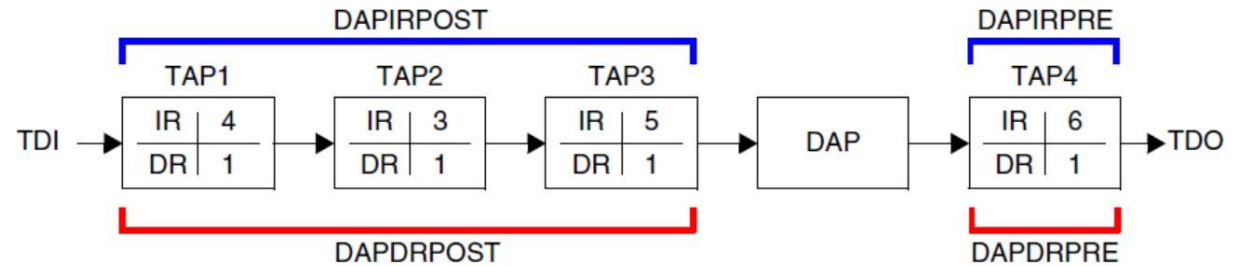


Figure: Debug Access Port (DAP) for SoC-400

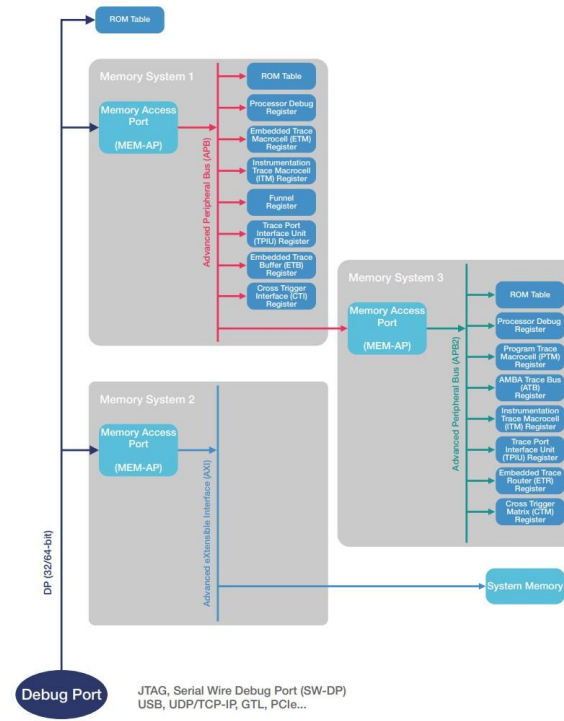
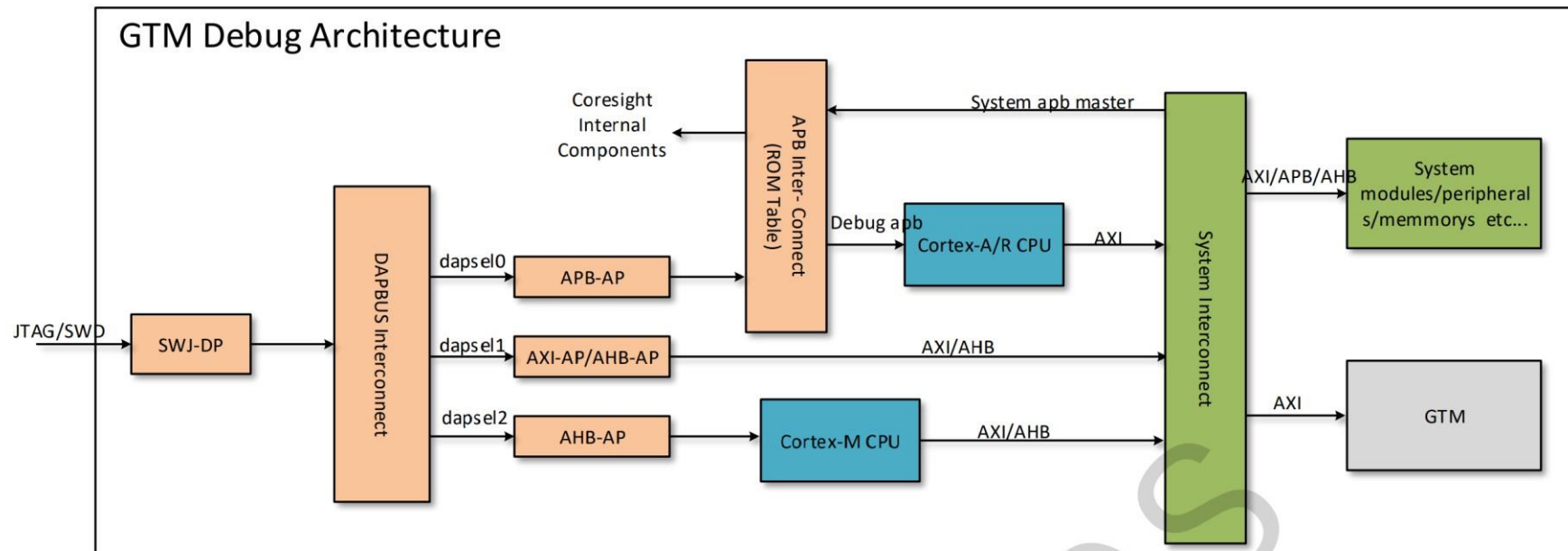


Figure: Debug Access Port (DAP) for SoC-600

GTM Debug Architecture diagram

- GTM IP Registers can be accessed via the DAP AXI-AP and other main core (such as arm)
- GTM Wrapper can be accessed via the APB-AP or AHB-AP or AXI-AP
- GTM CTI can be accessed via the DAP APB-AP





GTM Debug

Debug prerequisites

- define debug registers
 - The necessary debug control register
 - Optional register
 - Chinese customer define DBG register

```
GTM TOP-Level Configuration Registers
GTM_REV          31531586  DEV_CODE2      3          DEV_CODE1      1
                  MAJOR      03          MINOR          01
                  STEP       B6          No reset
GTM_RST          00000000  RST            0          No reset
GTM_CTRL        00000001  TO_VAL         0
GTM_AEI_ADDR_XPT 00000000  TO_WIRO        Read        TO_MODE        observe
GTM_AEI_STA_XPT 0009FE1C  TO_WIRO        Read        TO_ADDR        000000
GTM_IRQ_NOTIFY   00000000  AEIM_USP_BE    No interrupt AEIM_IM_ADDR    No interr
                  AEI_USP_BE    No interrupt  AEI_IM_ADDR
                  AEI_TO_XPT    Not occurred  AEIM_IM_ADDR_IRQ_EN Disabled
GTM_IRQ_EN       00000000  AEIM_USP_BE_IRQ_EN Disabled    AEIM_IM_ADDR_IRQ_EN Disabled
                  AEI_USP_BE_IRQ_EN Disabled
                  AEI_TO_XPT_IRQ_EN Disabled
GTM_IRQ_FORCINT  XXXXXXXX  TRG_AEIM_USP_BE XXXXXXXXXXXX TRG_AEIM_IM_ADDR XXXXXXXX
                  TRG_AEI_USP_BE XXXXXXXXXXXX TRG_AEI_IM_ADDR  XXXXXXXX
                  TRG_AEI_TO_XPT XXXXXXXXXXXX
GTM_IRQ_MODE     00000000  IRQ_MODE       Level
GTM_EIRQ_EN      00000180  AEIM_USP_BE_EIRQ_EN Disabled    AEIM_IM_ADDR_EIRQ_EN Disabled
                  AEI_USP_BE_EIRQ_EN Disabled
                  AEI_TO_XPT_EIRQ_EN Disabled
GTM_BRIDGE_MODE  02001001  BUFF_DPT       02          BRG_RST         No reset
                  BUFF_OVL     No overflow   MODE_UP_PGR     No update
                  BRG_MODE     Async_bridge
GTM_BRIDGE_PTR1  00100401  RSP_TRAN_RDY   00          FBC              01
                  TRAN_IN_PGR  1
GTM_BRIDGE_PTR2  00000001  TRAN_IN_PGR2   1
GTM_HW_CONF      084F022E  IRQ_MODE_SINGLE_PULSE Available
                  IRQ_MODE_LEVEL Available
                  RAM_INIT_RST  not initialized
                  ATOM_TRIG_CHAIN 1
                  SYNC_INPUT_REG Yes
GTM_MCS_AEM_DIS  00000000  DIS_CLS11     Enabled    DIS_CLS10      Enabled
                  DIS_CLS8       Enabled    DIS_CLS7       Enabled
                  DIS_CLS5     Enabled    DIS_CLS4       Enabled
                  DIS_CLS2     Enabled    DIS_CLS1       Enabled
GTM_TIM0_AUX_IN_SRC 00000000  SRC_CH_7 TOM0 ch 7 SRC_CH_6 TOM0 ch 6 SRC_CH_5 TOM0 ch 5
                  SRC_CH_4 TOM0 ch 4 SRC_CH_3 TOM0 ch 3 SRC_CH_2 TOM0 ch 2
                  SRC_CH_1 TOM0 ch 1 SRC_CH_0 TOM0 ch 0
GTM_TIM1_AUX_IN_SRC 00000000  SRC_CH_7 TOM0 ch 15 SRC_CH_6 TOM0 ch 14 SRC_CH_5 TOM0 ch 13
                  SRC_CH_4 TOM0 ch 12 SRC_CH_3 TOM0 ch 11 SRC_CH_2 TOM0 ch 10
                  SRC_CH_1 TOM0 ch 9 SRC_CH_0 TOM0 ch 8
GTM_TIM2_AUX_IN_SRC 00000000  SRC_CH_7 TOM1 ch 7 SRC_CH_6 TOM1 ch 6 SRC_CH_5 TOM1 ch 5
                  SRC_CH_4 TOM1 ch 4 SRC_CH_3 TOM1 ch 3 SRC_CH_2 TOM1 ch 2
```




GTM Wrapper Register

- Sample from XHSC
 - The CTI support up to 32 Trigger Inputs
 - if one customer's SoC has only 4 MCSx the CTI0 Trigger Input5—Input12 can only be reserved.

Symbol ↵	Register Name ↵
GTMD_GCR↵	GTM Debug Global Control Register
GTMD_HCR↵	GTM Debug Halt Control Register↵
GTMD_HSR↵	GTM Debug Halt Status Register↵
GTMC_MCSBPEN↵	MCS Break Point Enable Register↵

CTI0 Trigger Input	Connected to
Trigger Input0	GTM halted
Trigger Input1	MCS0 HBP EVENT
Trigger Input2	MCS1 HBP EVENT
Trigger Input3	MCS2 HBP EVENT
Trigger Input4	MCS3 HBP EVENT
Trigger Input5	Reserved
Trigger Input6	Reserved
Trigger Input7	Reserved
Trigger Input8	Reserved
Trigger Input9	Reserved
Trigger Input10	Reserved
Trigger Input11	Reserved
Trigger Input12	Reserved
CTI0 Trigger Output	Connected to
Trigger Output0	GTM Debug Request
Trigger Output1	GTM Restart
Trigger Output2	
Trigger Output3	



GTM Debug

Debug Functions

- Go/Break/Step
- Multi core Debug
- SYNC

The image displays two instances of the TRACE32 PowerView debugger. The left instance is for GTM (GTM: GTM) and the right instance is for TriCore 1 (Power Debug USB 3 @). Both windows show a code editor with assembly and C code, a register window, and a control flow graph.

Left Window (GTM: GTM):

- Code editor shows assembly instructions for Core 0, including `addl r4,0x4C00`, `andl r4,0xFFFF`, `movl r6,0xFFFF`, and `wurmx r4,tbu_ts1`.
- Register window shows `uioutput = outputchain[i];`.
- Control flow graph shows a `do { for (i = 0; i <= SIZE; i++) { while (true); } }` loop.

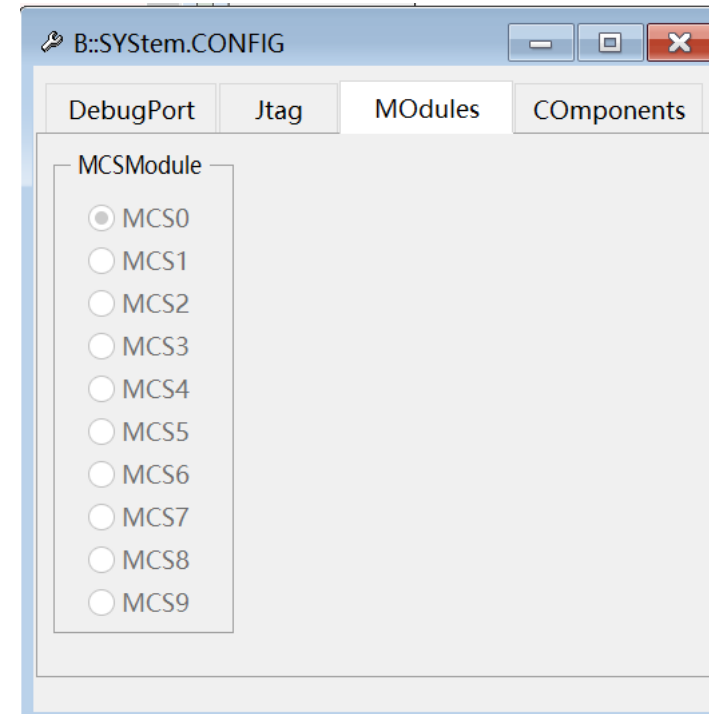
Right Window (TriCore 1):

- Code editor shows C code for a loop: `for (i = 0 ; i <= SIZE ; i++) { if (flags0[i]) { primz = i + i + 3; k = i + primz; while (k <= SIZE) { flags0[k] = FALSE; k += primz; } anzahl++; }`
- Register window shows `flags1[i + i]`.
- Control flow graph shows a nested loop structure.

Both windows have a status bar at the bottom showing the current state of the debug session, such as `stopped` or `stopped (inside line)`.

GTM Debug

- MCS Module select
 - SYStem.CONFIG.MCSModule MCS
 - Support AMP Multi MCS



GTM Debug

Debug Functions

➤ Breakpoints

The screenshot displays the TRACE32 PowerView interface with several panels:

- Code View:** Shows assembly code with comments. A breakpoint is set at address P:00000304. The code includes functions like `void __channel(0) channel_0(void)` and `void __channel(1) channel_1(void)`.
- Registers:** Shows the state of registers R0 through R7. R0 contains 00010021, and PC is 02FC.
- Memory Dump:** Shows a dump of memory starting at address 0, with values like 01234567.
- Breakpoint List:** Shows a list of breakpoints with columns for Setup, Delete, Disable, Enable, Init, Store, Load, and Set. A breakpoint is listed at address P:00000304, type Program, method ONCHIP, and is currently enabled.





GTM Debug

Debug Functions

- High-level language debugging

```
B::Data.List /CORE 0
Step Over Diverge Return Up Go Break Mode Find:
addr/line source
34 int
main( void )
{
37     unsigned int delay;
    STRG = SYNCHSTART;
    do {
39         for (int i = 0; i < 20; i++ ){
40             delay = TBU_TS1;
41             delay += DELAYSPEED;
42             delay &= 0xffff;
43             __wurmx(&TBU_TS1, delay, 0xFFFF);
44             uiOutput = outputchain[i];
46         } while (true);
47     }

51 int
mainChannel1( void )
{
55     unsigned int tmpvalueLED0, tmpvalueLED1;
    __aru_t valueLED0, valueLED1;
    __wurmx( & STRG, SYNCHSTART, 1 );
56     tmpvalueLED0 = 0;
57     tmpvalueLED1 = 0;
58     valueLED0.data_l = 0x100;
59     valueLED1.data_l = 0x100;
60     valueLED0.acb = 0x0;
61     valueLED1.acb = 0x0;
62     while (true){
65         valueLED0.data_h = tmpvalueLED0;
        valueLED1.data_h = tmpvalueLED1;
66         __awr(0,valueLED0);
        __awr(1,valueLED1);
67         if (uiOutput & 0x1){
68             tmpvalueLED0 = 0x0ff;
69         } else if (tmpvalueLED0 > 0x0){
70             tmpvalueLED0 -= 0x1;

```



GTM Debug

Debug Functions

> Inline Assembler

```
[B:\Data.List /CORE 0]
┌─ Step ─┐ ┌─ Over ─┐ ┌─ Diverge ─┐ ┌─ Return ─┐ ┌─ Up ─┐ ┌─ Go ─┐ ┌─ Break ─┐ ┌─ Mode ─┐ ┌─ Find: ─┐
└────────┘ └────────┘ └────────┘ └────────┘ └────────┘ └────────┘ └────────┘ └────────┘ └────────┘
addr/line code label mnemonic comment
P:00000310 E0040000 ret
P:00000314 17000394 channel_0:movl r7,0x394
P:00000318 E00302C0 call 0x2C0 ; main
P:0000031C A2800000 mov r2,sta
P:00000320 42FFFFFFE andl r2,0xFFFFFE
P:00000324 A8200000 mov sta,r2
P:00000328 170003D4 channel_1:movl r7,0x3D4
P:0000032C E0030020 call 0x20 ; mainChannel1
P:00000330 A2800000 mov r2,sta
P:00000334 42FFFFFFE andl r2,0xFFFFFE
P:00000338 A8200000 mov sta,r2
P:0000033C 17000414 channel_2:movl r7,0x414
P:00000340 E00300C8 call 0xC8 ; mainChannel2
P:00000344 A2800000 mov r2,sta
P:00000348 42FFFFFFE andl r2,0xFFFFFE
P:0000034C A8200000 mov sta,r2
P:00000350 17000454 channel_3:movl r7,0x454
P:00000354 E0030170 call 0x170 ; mainChannel3
P:00000358 A2800000 mov r2,sta
P:0000035C 42FFFFFFE andl r2,0xFFFFFE
P:00000360 A8200000 mov sta,r2
P:00000364 17000494 channel_4:movl r7,0x494
P:00000368 E0030218 call 0x218 ; mainChannel4
P:0000036C A2800000 mov r2,sta
P:00000370 42FFFFFFE andl r2,0xFFFFFE
P:00000374 A8200000 mov sta,r2
P:00000378 17000394 __START: movl r7,0x394
P:0000037C 12000000 movl r2,0x0
P:00000380 13000000 movl r3,0x0
P:00000384 E00302C0 call 0x2C0 ; main
P:00000388 E000038C jmp 0x38C ; _Exit
P:0000038C A2800000 _Exit: movl r2,sta
P:00000390 42FFFFFFE andl r2,0xFFFFFE
P:00000394 A8200000 mov sta,r2
P:00000398 00000388 _lc_ub_s.:nopl
P:0000039C 00000000 nopl
P:000003A0 00000000 nopl
P:000003A4 00000000 nopl
P:000003A8 00000000 nopl
```

GTM Debug

Debug Functions

- Variable View (run time access)
 - SYStem.option Dualport on

The screenshot shows two windows from the GTM debug environment. The top window, titled "B::sYmbol.Browse.sYmbol", displays a list of symbols and their types. The bottom window, titled "B::Var.Watch", shows the current values of variables.

symbol	type
.vector.4	
.vector.5_loop	
.vector.6_loop	
.vector.7_loop	
__START	
__Exit	(void ())
__lc_ub_stack_0	
__lc_ub_stack_1	
__lc_ub_stack_2	
__lc_ub_stack_3	
__lc_ub_stack_4	
__lc_ub_stack_main	
channel_0	(void ())
channel_1	(void ())
channel_2	(void ())
channel_3	(void ())
channel_4	(void ())
exit	
main	(int ())
mainChannel1	(int ())
mainChannel2	(int ())
mainChannel3	(int ())
mainChannel4	(int ())
outputchain	(unsigned int [20])
uiOutput	(unsigned int)

Variable	Value
outputchain	(0, 1, 2, 4, 8, 16, 32, 64, 128, 0, 0, 128, ...)
uiOutput	2



GTM Debug

Debug Functions

> Call back

The screenshot displays the TRACE32 PowerView for GTM: GTM interface. It shows four windows:

- Top Left:** Source code for `B:\List.auto /core 0`. The `mainChannel1` function is visible, with a `while (true)` loop containing a `for` loop. Line 43 is highlighted.
- Top Right:** Source code for `[B:\List.auto /core 1]`. The `mainChannel1` function is visible, with a `while (true)` loop. Line 65 is highlighted.
- Bottom Left:** Call stack for `B:\Frame.view /Locals /Caller /core 0`. It shows `main()` with `delay = 64533` and `i = 8`.
- Bottom Right:** Call stack for `B:\Frame.view /Locals /Caller /core 1`. It shows `mainChannel1()` with `tmpvalueLEDO = 0` and `tmpvalueLED1 = 0`.

The status bar at the bottom indicates the current location: `P:0000004C \\\sieve_mcs00\sieve_mcs00\mainChannel1+0x2C (1 more locations)`. The status is `1 stopped (inside line)`.



GTM Debug

Debug Functions

- Peripheral View of GTM registers

B:PER
GTM TOP-Level Configuration Registers
ARU (Advanced Routing Unit)
Broadcast Module
First In First Out Module
AEI to FIFO Data Interface
FIFO to ARU Unit
CMU (Clock Management Unit)
TBU (Time Base Unit)
CCM (Cluster Configuration Module)
TIM (Timer Input Module)
TOM (Timer Output Module)
ATOM (ARU-connected Timer Output Module)
CDTM (Cluster Dead Time Module)
MCS (Multi Channel Sequencer)
MCFG (Memory Configuration)
MAP (TIM0 Input Mapping Module)
DPLL (Digital PLL Module)
DPLL RAM (DPLL RAM Region)
SPE (Sensor Pattern Evaluation)
ICM (Interrupt Concentrator Module)
CMP (Output Compare Unit)
MON (Monitor Unit)
GTM Implementation



GTM Debug

Debug Functions

> Peripheral View of Memory

The screenshot displays the TRACE32 PowerView for GTM: GTM interface. The main window shows the peripheral view of memory, with a tree structure on the left. The tree includes 'GTM_TOP-Level Configuration Registers' and 'ARU (Advanced Routing Unit)'. A context menu is open over the 'ARU' tree, showing options like 'Display Memory', 'View', 'Dump', 'Indirect List', 'Indirect View', 'Indirect Dump', 'Track List', 'Track View', and 'Track Dump'. The 'Dump' option is selected. Below the tree, a frame window shows the 'main()' function with a delay of 64533 and a loop counter 'i = 8'. The bottom status bar indicates 'FPI:F0100288 ARU Access Register Lower Data Word'.

The dump window shows the following data:

address	#Find...	Modify...	Long	E	Track	Hex	Ascii
FPI:F0100280	000001FE	00000000	00000000	00000077		E S N N N N N N N N N N N N N N	ERUUUUUUUUUUUUUUUUUUUUUUUUUU
FPI:F0100290	00000000	00000100	00000078	00000000		N N N N N N N N N N N N N N N N	S S S S S S S S S S S S S S S S
FPI:F01002A0	00000100	00000003	00000000	00000000		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F01002B0	00000000	0000007F	????????	00000000		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F01002C0	00000000	00000000	00000000	00000000		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F01002D0	00000000	00000000	00000000	00000000		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F01002E0	00000000	00000000	00000000	00000000		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F01002F0	????????	????????	????????	006A006A		????????	????????
FPI:F0100300	00C00003	0000FFFF	0000FFFF	00000100		E N G N E F N N E F N N N S N N	X U U U F F U F F U U U U U U
FPI:F0100310	00000000	00000000	00000000	00000000		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F0100320	00000000	00000000	00000000	00000001		N N N N N N N N N N N N N N N N	U U U U U U U U U U U U U U U U
FPI:F0100330	00000001	00000001	00000001	00000001		S S S S S S S S S S S S S S S S	S S S S S S S S S S S S S S S S
FPI:F0100340	00000001	00000000	00000000	00000000		S S S S S S S S S S S S S S S S	H U U U U U U U U U U U U U U U
FPI:F0100350	????????	????????	????????	????????		????????	????????
FPI:F0100360	????????	????????	????????	????????		????????	????????
FPI:F0100370	????????	????????	????????	????????		????????	????????
FPI:F0100380	????????	????????	????????	????????		????????	????????
FPI:F0100390	????????	????????	????????	????????		????????	????????
FPI:F01003A0	????????	????????	????????	????????		????????	????????
FPI:F01003B0	????????	????????	????????	????????		????????	????????
FPI:F01003C0	????????	????????	????????	????????		????????	????????
FPI:F01003D0	????????	????????	????????	????????		????????	????????
FPI:F01003E0	????????	????????	????????	????????		????????	????????
FPI:F01003F0	????????	????????	????????	????????		????????	????????
FPI:F0100400	000001FE	00000000	000001FE	00000000		E S N N N N N N N N N N N N N N	ERUUUUUUUUUUUUUUUUUUUUUUUUUU



GTM Debug

Debug Functions

- GTM v4.1 new features
 - New Hardware Breakpoint Unit
 - Two Breakpoints (break before make) for every MCS Module
 - Used for Single Stepping or Breakpoints
 - Breakpoints on Program Address or Range
 - Breakpoints on Data Address or Range; Read or Write Access
 - Breakpoint on Data Address or Range and Data Value
 - Channel Selection possible



Virtual target support

- COSEDA virtual target has an adaptor via the MCD interface to the virtual GTM

MCD

The Multi-Core Debug API (MCD) is an interface between software development tools and simulated or real systems with multi-core SoCs.

Lauterbach, together with other renowned companies, defined this interface. Start-ups to market leaders are using this API to connect their virtual or emulated target to our PowerView GUI.

[Read more](#)

GDB

Utilizing the Remote Serial Protocol (RSP), TRACE32 can connect to any GDB server implementation. Beyond Linux application and kernel debugging, TRACE32 PowerView can also control many simulator products like QEMU.

Vendor specific Interfaces (CADI, IRIS, ARCINT, etc.)

TRACE32 is the market-leading tool. TRACE32 supports vendor-specific interfaces of processor models and simulators from most providers. You will have your choice of vendors and models.

GTL

The Generic Transactor Library (GTL) is an API designed by Lauterbach. It connects the transactors of an emulation system to TRACE32's complete debug and trace software stack. The API provides interfaces for trace recording, bus level, and signal level transactors. This allows you to make trade-offs between detail and execution speed.

[Read More](#)

Live Demo

➤ Demo show with XHSC board



The screenshot displays the TRACE32 PowerView interface for ARM 1. It is divided into several panes:

- Top Left:** A list of instructions with columns for Step, Over, Diverge, Return, Up, Go, Break, Mode, and Find. The code is from `mcs00.c`.
- Top Right:** A list of instructions with columns for Step, Over, Diverge, Return, Up, Go, Break, Mode, and Find. The code is from `mainchannell.c`.
- Bottom Left:** A pane for `B::System.state` showing Mode (Up), MemAccess (DAP), Option (IMASKASM, IMASKHLL, DUALPORT, PERSTOP), DebugPort (MCS1), Jtag, and Modules (XC27X, 10.0MHz).
- Bottom Right:** A pane for `B::System.CONFIG` showing Mode (Up), MemAccess (DAP), Option (MACHINESPA, DUALPORT), Option DisMode (AUTO), and reset (RESetOut).

The status bar at the bottom indicates the system is `running`.





GTM Trace

Trace prerequisites

- Hardware interface
- on-chip or off-chip trace
- define trace encoder
- define trace protocol
- define trace path
- define trace triggers

how to identify the trace from main core or GTM

Trace functions

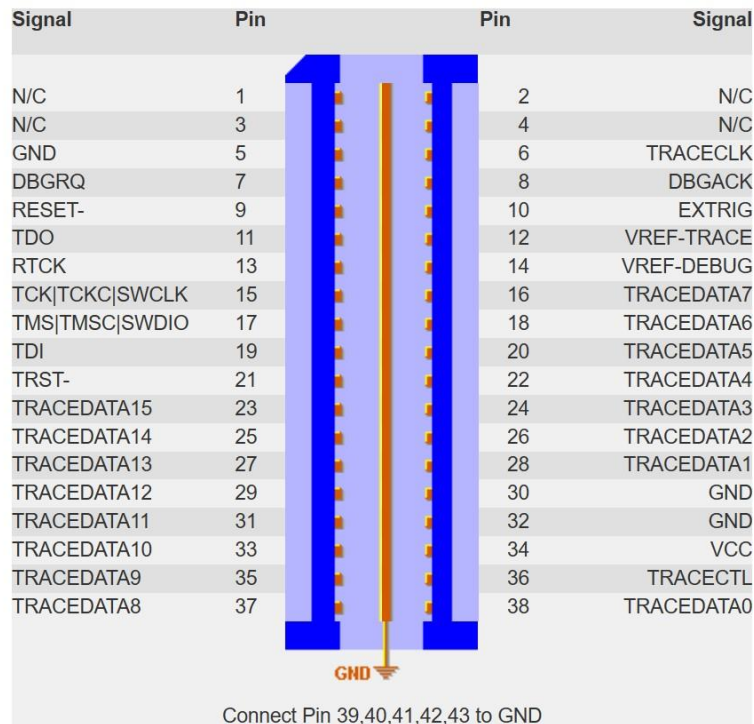
- Multi Channel Sequencer (MCS)
- Advanced Routing Unit (ARU)
- I/O Channels (TIM, TOM, ATOM and TIO)
- Digital PLL Module (DPLL)
- Sensor Pattern Evaluation (SPE)
- Time Base Unit (TBU)

GTM Trace

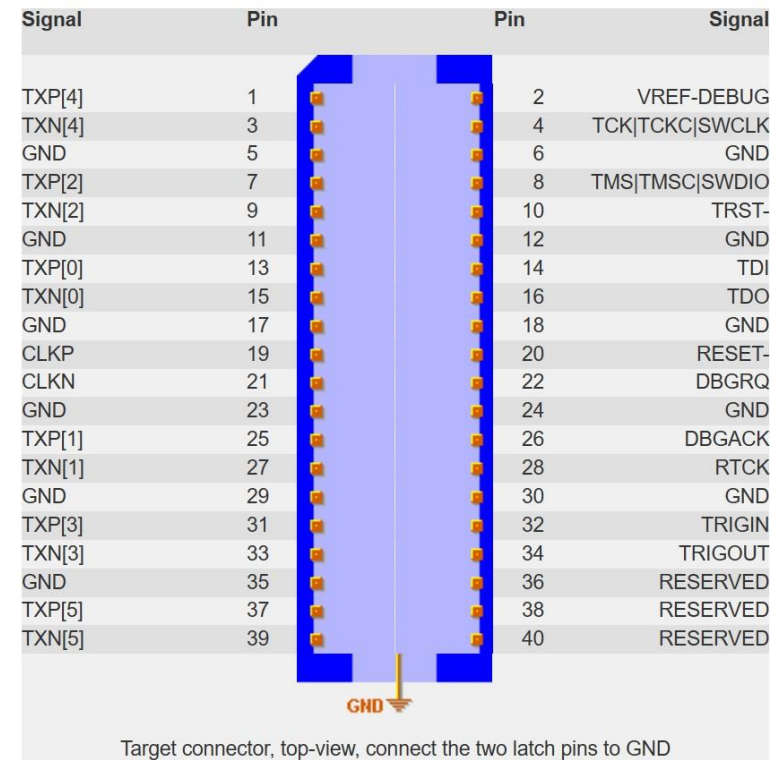
Trace prerequisites

> Hardware interface

Connector 1



■ Pinout of 40-pin Samtec Connector

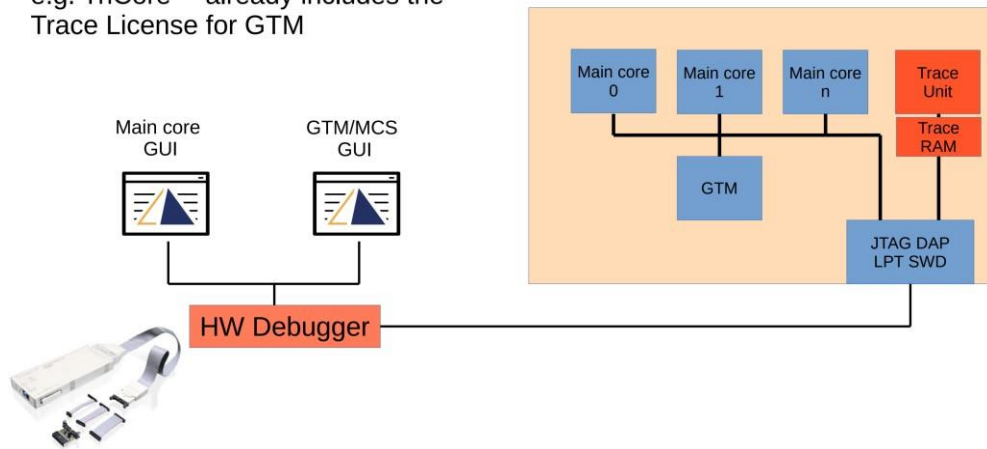


GTM Trace

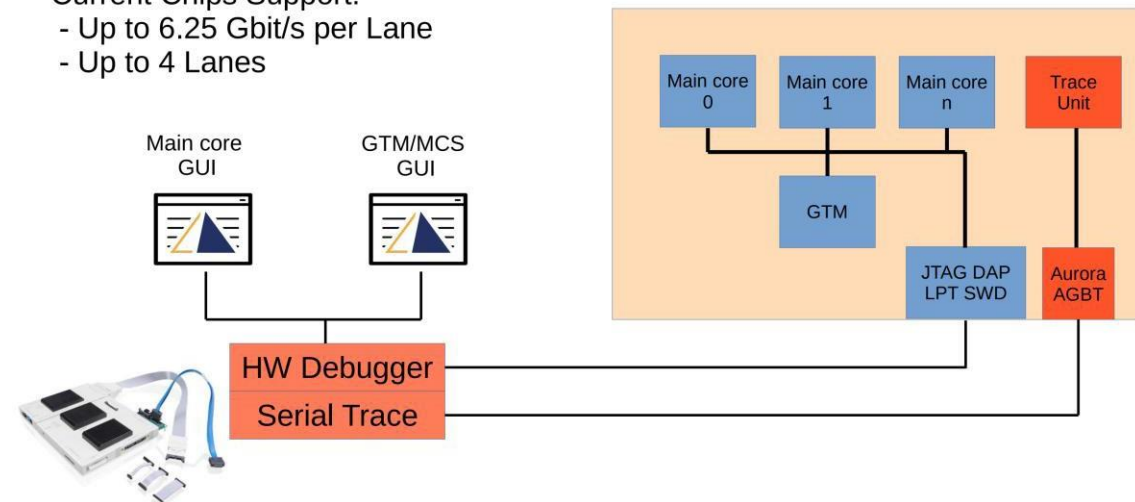
Trace prerequisites

- > on-chip or off-chip trace

The Trace License for the main core e.g. TriCore™ already includes the Trace License for GTM



Aurora GigaBit Trace
Current Chips Support:
- Up to 6.25 Gbit/s per Lane
- Up to 4 Lanes

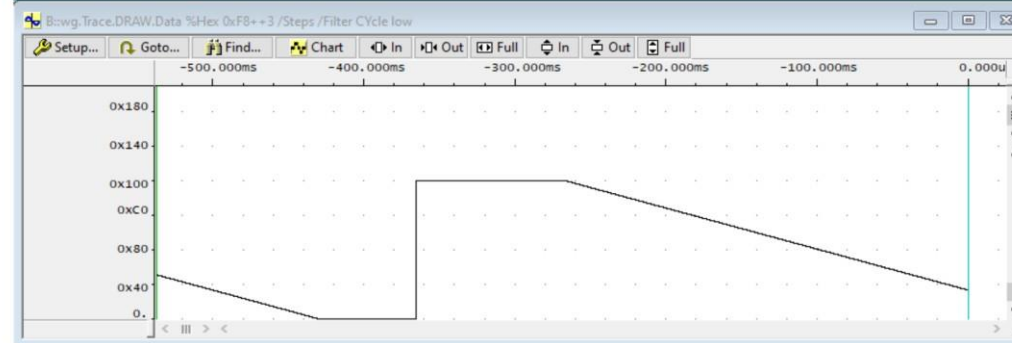
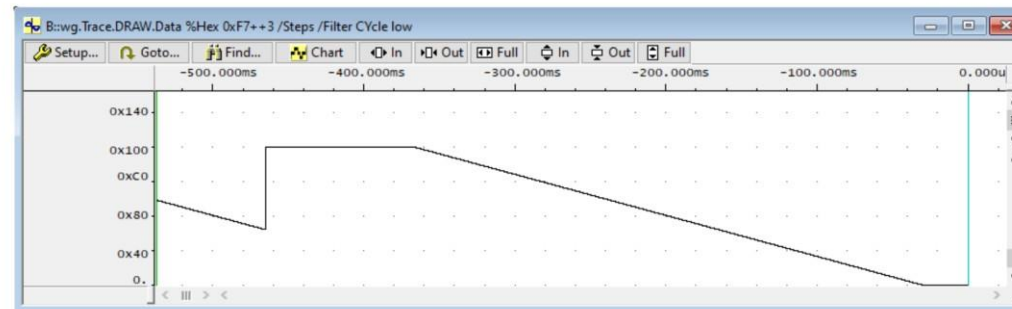


GTM Trace

Trace Functions

- Advanced Routing Unit (ARU)

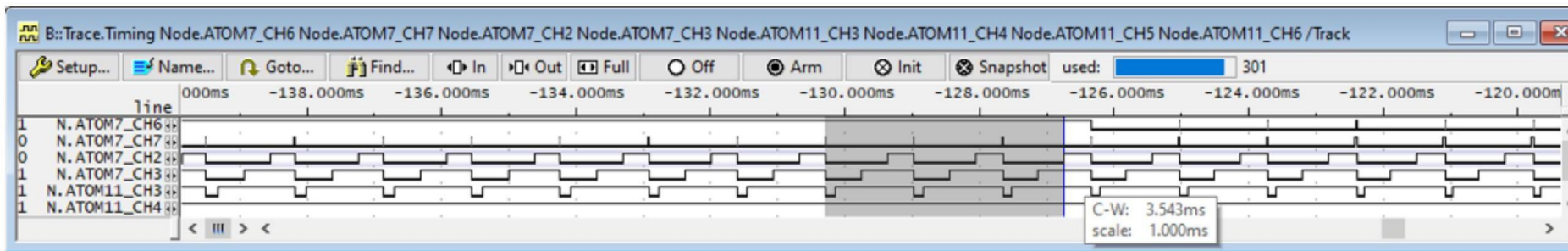
record	run	address	cycle	data	symbol	ti.back
-001140		ARU:000000F7	low	0000000C		0.020us
-001120		ARU:000000F8	high	00000100		2.580us
-001102		ARU:000000F8	low	00000090		0.020us
-001055		ARU:000000F7	high	00000100		1.313ms
-001036		ARU:000000F7	low	000000D8		0.020us
-001016		ARU:000000F8	high	00000100		2.580us
-000998		ARU:000000F8	low	0000008F		0.020us
-000951		ARU:000000F7	high	00000100		1.313ms
-000932		ARU:000000F7	low	000000DA		0.020us
-000912		ARU:000000F8	high	00000100		2.580us
-000894		ARU:000000F8	low	0000008E		0.020us
-000847		ARU:000000F7	high	00000100		1.313ms
-000828		ARU:000000F7	low	000000D9		0.020us
-000808		ARU:000000F8	high	00000100		2.580us
-000790		ARU:000000F8	low	0000008D		0.020us
-000743		ARU:000000F7	high	00000100		1.313ms
-000721		ARU:000000F7	low	000000D8		0.020us
-000702		ARU:000000F8	high	00000100		2.580us
-000683		ARU:000000F8	low	0000008C		0.020us
-000636		ARU:000000F7	high	00000100		1.313ms
-000617		ARU:000000F7	low	000000D7		0.020us
-000598		ARU:000000F8	high	00000100		2.580us
-000579		ARU:000000F8	low	00000088		0.020us
-000531		ARU:000000F7	high	00000100		1.313ms
-000512		ARU:000000F7	low	000000D6		0.020us
-000492		ARU:000000F8	high	00000100		2.580us
-000474		ARU:000000F8	low	0000008A		0.020us
-000427		ARU:000000F7	high	00000100		1.313ms
-000408		ARU:000000F7	low	000000D5		0.020us
-000388		ARU:000000F8	high	00000100		2.580us
-000370		ARU:000000F8	low	00000089		0.020us
-000323		ARU:000000F7	high	00000100		1.313ms
-000304		ARU:000000F7	low	000000D4		0.020us
-000284		ARU:000000F8	high	00000100		2.580us
-000266		ARU:000000F8	low	00000088		0.020us
-000219		ARU:000000F7	high	00000100		1.313ms
-000200		ARU:000000F7	low	000000D3		0.020us
-000180		ARU:000000F8	high	00000100		2.580us
-000156		ARU:000000F8	low	00000087		0.020us
-000105		ARU:000000F7	high	00000100		1.313ms
-000083		ARU:000000F7	low	000000D2		0.020us
-000064		ARU:000000F8	high	00000100		2.580us
-000045		ARU:000000F8	low	00000086		0.020us



GTM Trace

Trace Functions

- I/O Channels (TIM, TOM, ATOM and TIO)



Logic analyzer measurement window for N.ATOM7_CH2. The window displays statistical data for a selected signal.

Measurement	avr	min	max	bits & jitter
time	853.240us	832.680us	873.800us	
time	460.030us	436.900us	483.160us	
period	1.316ms	1.316ms	1.316ms	438.552us
frequency	759.Hz	759.Hz	759.Hz	2.28KHz
duty cycle	65:35	63:37	67:33	7.618%
bitstream				

Logic analyzer like measurements

MCS Trace flow

➤ MCS Ptrace record

The screenshot displays the TRACE32 PowerView for GTM: GTM interface, showing the MCS Ptrace record. The main window is divided into several panes:

- B:DataList /CORE 0:** Disassembler window showing assembly code. The code includes a loop for (int i = 0; i < 20; i++) and a while (true) loop. The code is as follows:

```
41 mov r4,tbu_ts1 ; delay, tbu_ts1
42 delay += DELAYSPEED; ; delay, 19456
43 addl r4,0x4C00 ; delay, 65535
44 andl r4,0xFFFF ; delay, 65535
45 __wurmx(&TBU_TS1, delay, 0xFFFF);
46 movl r6,0xFFFF ; delay, tbu_ts1
47 wurmx r4,tbu_ts1 ; delay, tbu_ts1
48 uioutput = outputchain[i];
49 mov r4,r5 ; delay, i
50 shl r4,2 ; i, 2
51 mrdi r4,r4,0x4D8 ; i, 1, 1240
52 mwr r4,0x528 ; r4, uioutput
53 do {
54 for (int i = 0; i < 20; i++) {
55 addl r5,0x1 ; i, 1
56 atsl r5,0x14 ; i, 20
57 jbs sta, cy, 0x2D4
58 } while (true);
59 jmp 0x2CC
60 mrdi r6,r7,0x0
61 subl r7,0x4
62 ret
180 void __channel(0) channel_0( void )
181 {
182 channel_0:movl r7,0x394
183 main();
184 call 0x2C0 ; main
185 mov r2,sta
186 andl r2,0xFFFFFE
187 mov sta,r2
```
- B:TraceList /Track:** Trace list window showing the execution flow. The trace records include:

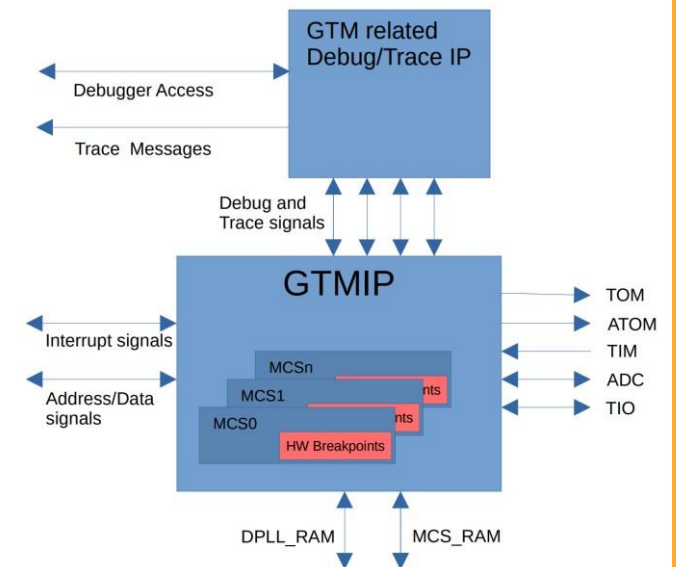
```
+00192081 ARU:00000078 low 00000000 ..\sieve_mcs00\mainchannel1+0x58 0.020
+00192101 P:000002E8 ptrace ..ve_mcs00\sieve_mcs00\main+0x28 100.003
44 0 uioutput = outputchain[i];
+00192105 0 mov r4,r5 ; delay, i
+00192108 0 shl r4,2 ; i, 2
+00192108 0 P:000002F0 ptrace ..ve_mcs00\sieve_mcs00\main+0x30 0.040
+00192112 0 mrdi r4,r4,0x4D8 ; i, 1, 1240
+00192118 0 P:000004E0 rd-data ..00\sieve_mcs00\outputchain+0x8 0.160
+00192118 0 P:000002F4 ptrace ..ve_mcs00\sieve_mcs00\main+0x34 0.040
+00192124 0 mwr r4,0x528 ; r4, uioutput
+00192124 0 D:00000528 wr-data ..eve_mcs00\sieve_mcs00\uioutput 0.160
+00192130 0 P:000002F8 ptrace ..ve_mcs00\sieve_mcs00\main+0x38 0.040
0 do {
39 0 for (int i = 0; i < 20; i++) {
+00192136 0 addl r5,0x1 ; i, 1
+00192140 0 P:000002FC ptrace ..ve_mcs00\sieve_mcs00\main+0x3C 0.040
+00192140 0 atsl r5,0x14 ; i, 20
+00192144 0 jbs sta, cy, 0x2D4 ..ve_mcs00\sieve_mcs00\main+0x40 0.040
+00192144 0 P:000002D4 ptrace ..ve_mcs00\sieve_mcs00\main+0x14 0.160
40 0 delay = TBU_TS1;
+00192149 0 mov r4,tbu_ts1 ..ve_mcs00\sieve_mcs00\main+0x18 0.040
+00192149 0 P:000002D8 ptrace ..ve_mcs00\sieve_mcs00\main+0x18 0.040
41 0 delay += DELAYSPEED; ; delay, 19456
+00192152 0 addl r4,0x4C00 ; delay, 19456
+00192152 0 P:000002DC ptrace ..ve_mcs00\sieve_mcs00\main+0x1C 0.040
42 0 delay &= 0xFFFF; ; delay, 65535
+00192156 0 andl r4,0xFFFF ; delay, 65535
+00192156 0 P:000002E0 ptrace ..ve_mcs00\sieve_mcs00\main+0x20 0.040
43 0 __wurmx(&TBU_TS1, delay, 0xFFFF);
+00192160 0 movl r6,0xFFFF ..ve_mcs00\sieve_mcs00\main+0x24 0.040
+00192160 0 P:000002E4 ptrace ..ve_mcs00\sieve_mcs00\main+0x24 0.040
0 wurmx r4,tbu_ts1 ; delay, tbu_ts1
+00192942 ARU:00000077 high 00000100 ..\sieve_mcs00\mainchannel1+0x57 1.311
+00192950 ARU:00000077 low 000000FF ..\sieve_mcs00\mainchannel1+0x57 0.020
+00192965 ARU:00000078 high 00000100 ..\sieve_mcs00\mainchannel1+0x58 5.140
+00192976 ARU:00000078 low 00000000 ..\sieve_mcs00\mainchannel1+0x58 0.020
```
- B:Register /CORE 0:** Register window showing the state of registers. The PC register is highlighted at 02E4.
- B:BreakList:** Break list window showing a break point at address P:00000304, type Program, method ONCHIP, and location main\12.

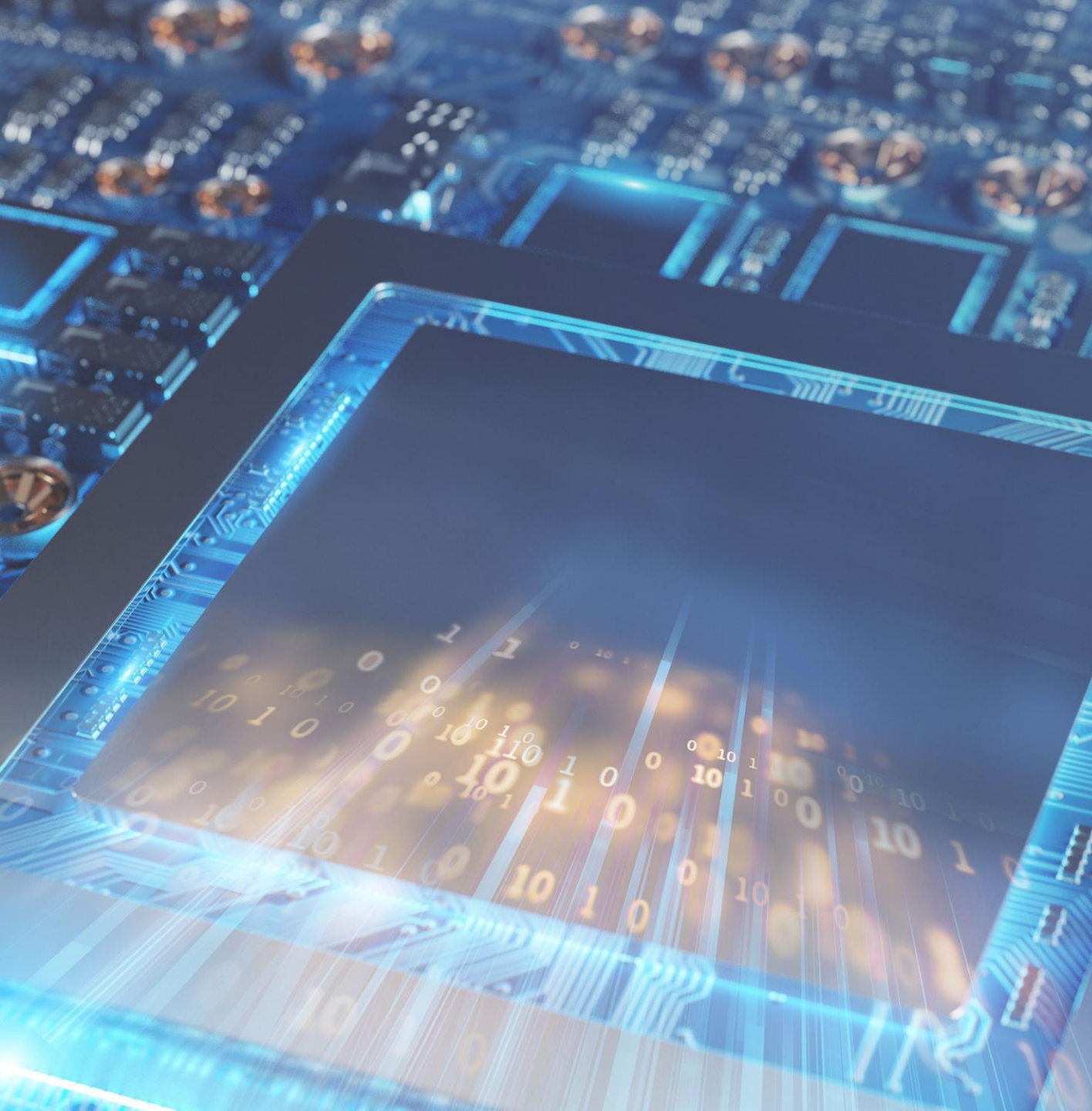


Our role with GTM

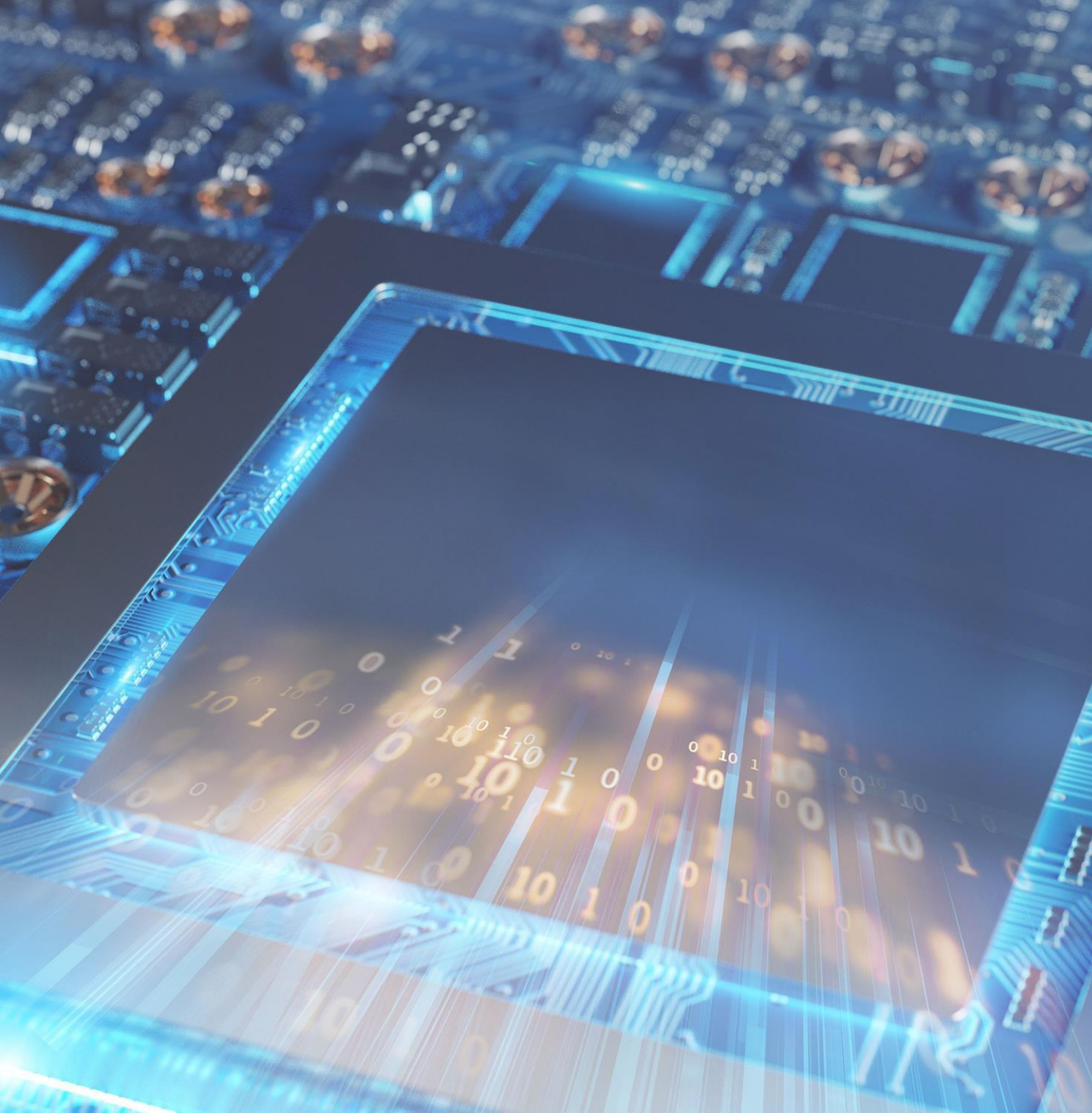
Our role with GTM

- We are honored to work with Bosch and relevant local semiconductor companies to define or standardize a way to debug and trace GTM. Hope our existing know-how will help the users to improve their development efficiency.
- We don't know which signals the GTMIP provides.
- We are really only a user of the Debug Interface.
- We are not allowed to be the initiator.
- We can comment on proposals and give recommendations.





QUESTIONS?



THANK YOU!