



Armv8-M Mainline Debug

Learning objectives

At the end of this module you will be able to:

- Establish your debug, trace and profiling requirements and capabilities
- Differentiate which actions in debugging are invasive or non-invasive
- Describe optional Armv8-M Mainline debug & trace components: FPB, DWT, ITM, MTB, ETM and TPIU
- Detect different debug events that have halted a processor
- Distinguish between semihosting calls and standard breakpoints
- Describe profiling features: DWT & Armv8.1-M Performance Monitor Unit (PMU)

Agenda

- **Introduction to Armv8-M Debug & Trace**
- Debugging with Breakpoints, Watchpoints, Semihosting & Vector Catch
- Tracing execution with the DWT, ITM, MTB, ETM
- Profiling with the DWT & PMU

Basic debug requirements

Run Control

- Single step through code
- Set breakpoints on instructions
- Set watchpoints on data accesses

State Control

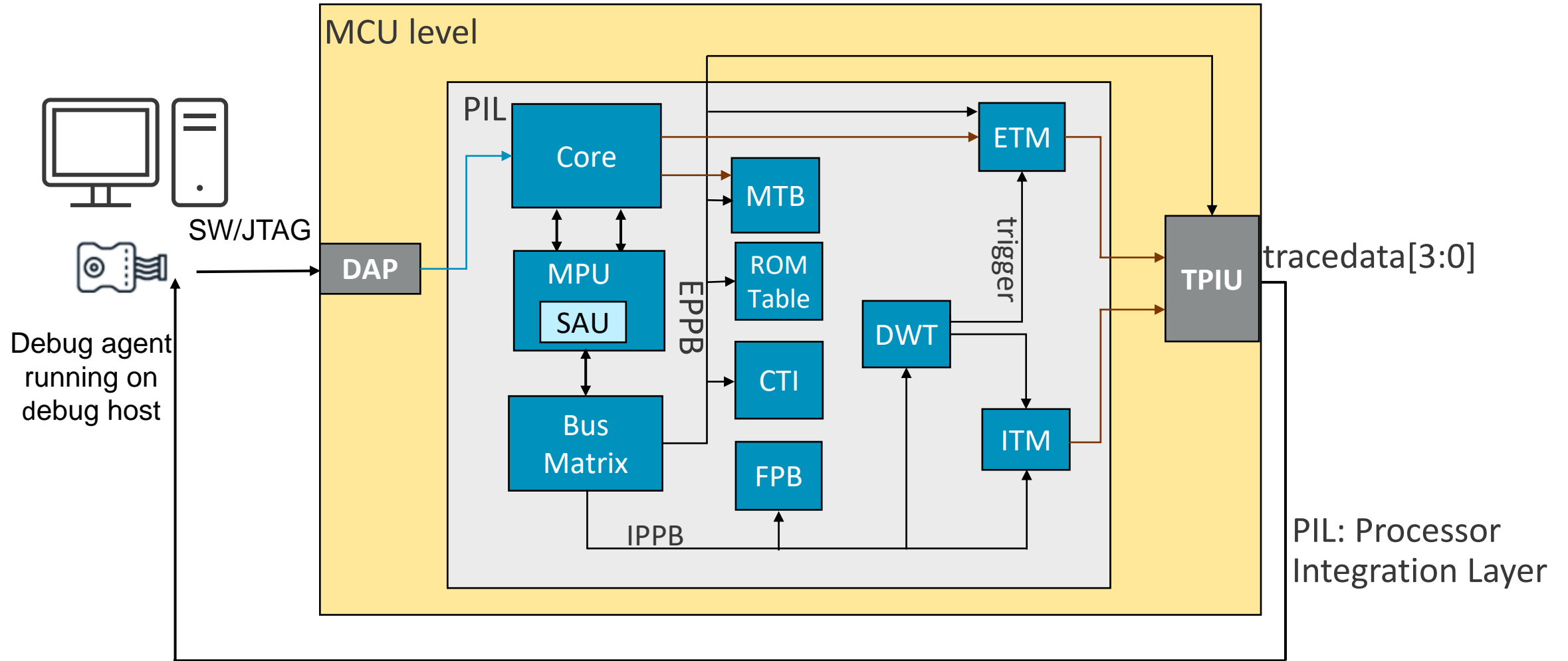
- Processor state
 - Read and write register values
- System state
 - Access to system memory
 - Access to peripherals

Execution History

- Execution trace information



Armv8-M Mainline debug components



Invasive debug vs. non-invasive debug

Invasive Debug

Control the processor

- Halt the core
- Single step through code
- Run code

Examine & alter processor registers

- While processor is halted (in Debug state)

Examine & alter memory and memory-mapped registers

- While processor is running or halted
- Using same memory view as processor

Intrusive to program execution

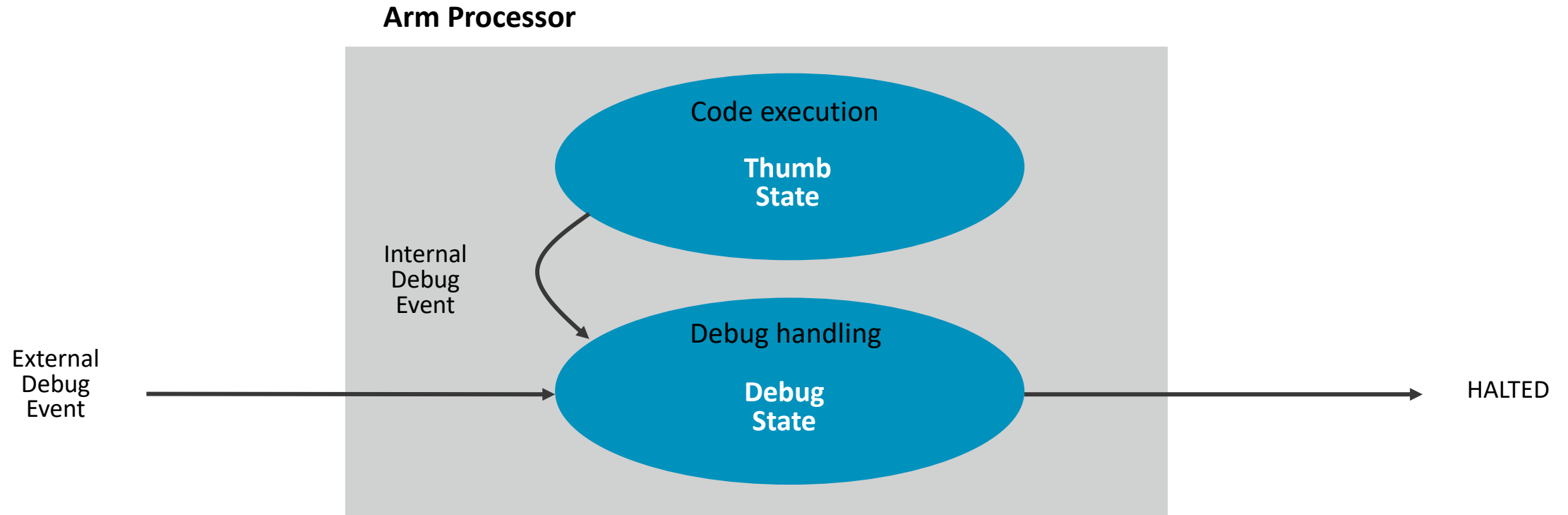
Non-invasive Debug

Observing the behavior of the processor

- Not halting the processor
- Can be non-intrusive to program execution
- But not all “non-invasive” debug is non-intrusive

Non-invasive debug provided via optional MTB, ITM, ETM, and PMU

Debug state - halting debug



If halting debug is enabled in the processor (DHCSR) and a qualifying debug event occurs, the processor enters Debug state

When in Debug state the processor is stopped

- No instructions are executed / No interrupts are serviced

The processor is controlled through external debug interface

Debug events for halting debug

Traditional start/stop debug

- Core executes and then halts in Debug mode

Debug Fault Status Register (DFSR) identifies the type of debug event

- PMU Sticky flag indicating a PMU counter has overflowed
- EXTERNAL EDBGRRQ input asserted from other SoC component
- VCATCH Vector Catch triggered
- DWTTRAP Data access to address matching a watchpoint
- BKPT **BKPT** instruction executed or a breakpoint match in FPB
- HALTED Halt request from debugger (or stepping in debug)

The alternative to halting debug is exception-based self-hosted debug

Self-hosted debug

The processor can be configured to take a DebugMonitor exception instead of halting

- Configured to target either Secure or Non-secure state
- The user must provide the DebugMonitor handler (CMSIS-Core provides a DebugMonitor handler)

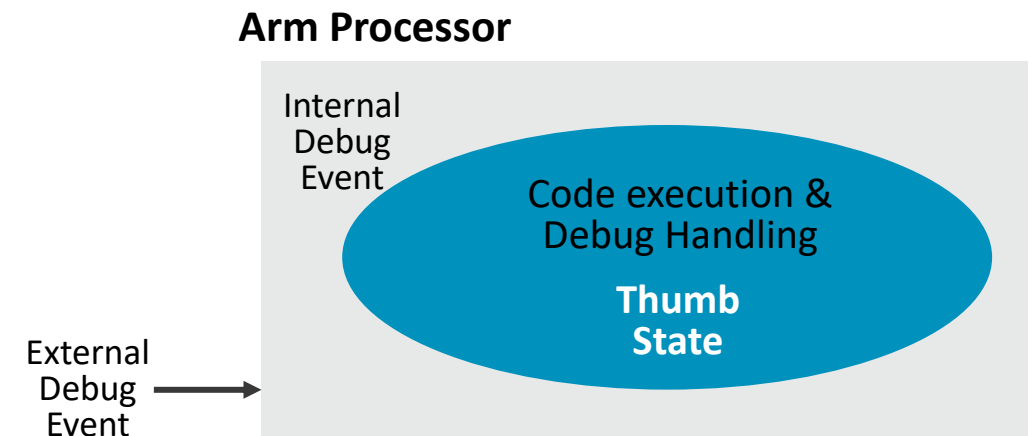
Enabled via the Debug Exception and Monitor Control Register (DEMCR)

Suitable for debugging systems with hard real-time requirements, e.g., motor/disk control

DebugMonitor may be used to handle PMU overflow

Always implemented with the Armv8-M Main Extension

- Never implemented in Armv8-M Baseline implementations



Agenda

- Introduction to Armv8-M Debug & Trace
- **Debugging with Breakpoints, Watchpoints, Semihosting & Vector Catch**
- Tracing execution with the DWT, ITM, MTB, ETM
- Profiling with the DWT & PMU

Vector catch

Mechanism traps selected exceptions

- Core halts when exception is asserted
- No DWT / FPB resources utilized
- Suitable for early software development
- Used by debugger to download applications at reset
- Selection made through debugger

Following exceptions may be trapped

- Reset
- HardFault
- UsageFault
- BusFault
- SecureFault

Note - cannot catch interrupts this way

- Unlike other Arm and Cortex-A/R cores
- Use breakpoint in interrupt handler

FPB – Flash Patch and Breakpoint unit

There are two types of breakpoints:

- Software breakpoint - implemented with the BKPT instruction
 - Debugger replaces original instruction with BKPT for software breakpoint
- Hardware breakpoint - implemented with the FPB comparators

The FPB supports setting breakpoints on instruction fetches

- Intended to be used to set breakpoints on non-volatile code memory
- Debug event occurs when the processor attempts to execute that instruction

The number of implemented instruction address comparators is IMPLEMENTATION DEFINED

- Cortex-M55 can be configured to support 0, 4, or 8 breakpoints

Armv8-M does not support the patching capability from earlier architecture versions

- The name is historical, referencing a patching capability available in earlier architectures

Breakpoints versus watchpoints

Breakpoints are used to debug instructions

- Debug event occurs when instruction fetched from a specific address is executed

Watchpoints are used to debug data accesses

- Debug event occurs when data is accessed at a specific address



Registers are provided to configure breakpoints and watchpoints

Number of available breakpoints and watchpoints is configured during hardware implementation

DWT watchpoints

DWT provides

- Comparator support for data address (watchpoint) and instruction address (PC watchpoint)
- Can break on [data value && data address] match (x1)

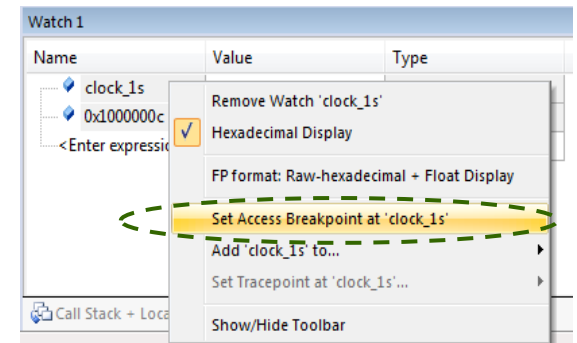
PC watchpoint functionality is used to set watchpoint for range of addresses

- PC watchpoint is aligned to a fetch, not an execution

Four DWT registers may be used by a debugger for halting debug

- Control register (DWT_CTRL)
- Comparator registers (DWT_COMPn)
- VMask registers (DWT_MASKn) – added in Armv8.1-M
- Functions registers (DWT_FUNCTIONn)

Comparator 0 also supports cycle count comparisons (more later)



Semihosting

Library code runs on Arm target, but low-level I/O handled by host

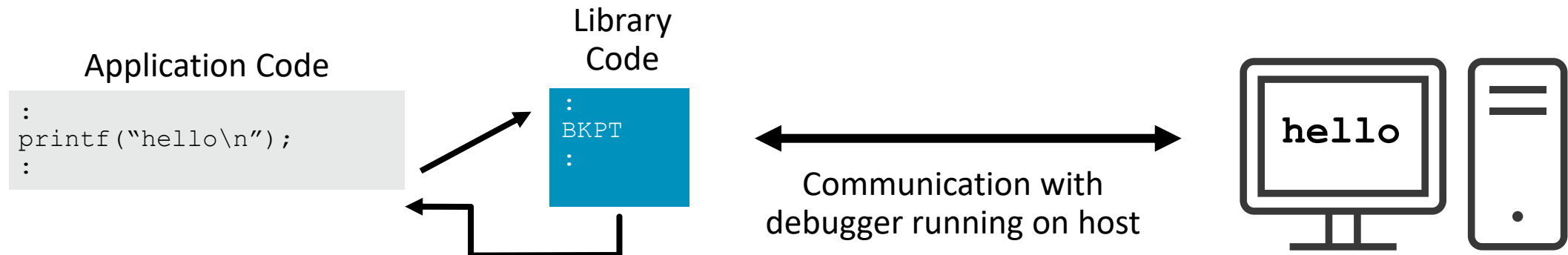
Host access initiated by BKPT instruction

- **BKPT 0xAB** reserved for semihosting
- Cortex-A/R profiles use the SVC of HLT instructions
 - Simpler than traditional semihosting on other cores
 - No need to modify your SVC handler

Interface supported by some tools, e.g., Arm Development Studio

- Some debug tools, e.g., Keil µVision Debugger, do not support semihosting

Debug tools must be connected to provide this functionality



Agenda

- Introduction to Armv8-M Debug & Trace
- Debugging with Breakpoints, Watchpoints, Semihosting & Vector Catch
- **Tracing execution with the DWT, ITM, MTB, ETM**
- Profiling with the DWT & PMU

Trace: Introduction

Trace is the process of capturing data that illustrates how the components are executing and performing

- Mostly program trace (instruction trace) or application trace (instrumentation trace)
- Generated by a trace source, and flows to a trace sink to be analyzed later

Instruction trace

- Information about the instruction execution of the processor
- Generated by the ETM (Embedded Trace Macrocell) or MTB (Micro Trace Buffer)

Instrumentation trace

- Information about the OS, application, and system events
- Software or hardware can generate trace events
- Events are captured and emitted by the ITM (Instrumentation Trace Macrocell)

MTB – Micro Trace Buffer

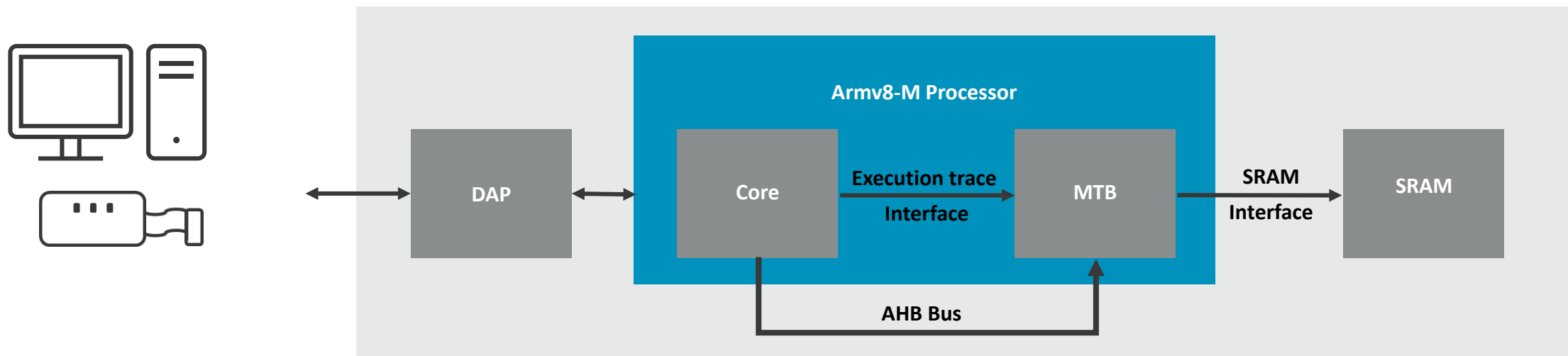
Provides a simple execution trace capability for Armv8-M

- Optional component
- Stores details of branches in execution in on-chip RAM
- Very small area and power reduction features

MTB SRAM can be used for both trace and general use by the processor

- SRAM size is configurable at implementation time
- The position and size of the trace buffer in this SRAM is configurable by software

CoreSight compliant



Instrumentation Trace Macrocell (ITM)

Generates and outputs trace packets

Packet types (in priority order):

- Software trace
 - Software can write directly to ITM stimulus registers, causing packets to be emitted
 - Similar to using `printf()` to debug a C program
- Hardware trace
 - Packets are generated by the DWT and emitted by the ITM
- Timestamps
 - Must be enabled in ITM Trace Control Register
 - Local timestamp (differential) value generated from 21-bit internal counter
 - Counter clocked from either core clock or TPIU clock
 - Global timestamp (absolute) value generated from either a 48-bit or 64-bit external counter
 - If implemented, Armv8-M with the Main extension provides a 64-bit global timestamp
 - Packet generation
 - When any other trace packet is generated (which resets timestamp counter)
 - When timestamp counter overflows

ITM control and setup

ITM Control Register

- Master ITM enable bit
- Control for DWT and timestamps packet generation
- Timestamp prescaler (no pre-scaling, 4, 16 or 64)

ITM Trace Enable Register

- 32-bit register used to enable each stimulus port (1 bit / port)
- Example: 0xF0000000 only enables stimulus ports 28-31

ITM Trace Privilege Register

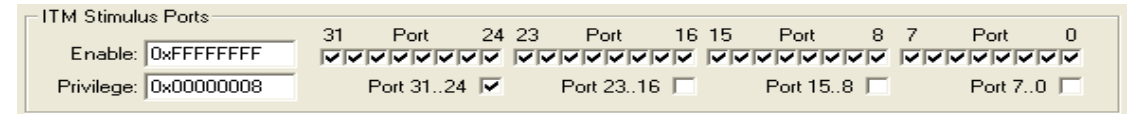
- Bit = 1 means privileged write access only
- Bit = 0 means user write access allowed
- Permission is controlled on an 8-port basis
 - bit[0] controls stimulus ports [7:0]
 - bit[1] controls stimulus ports [15:8]
 - bit[2] controls stimulus ports [23:16]
 - bit[3] controls stimulus ports [31:24]

ITM Stimulus Ports		31	Port	24	23	Port	16	15	Port	8	7	Port	0								
Enable:	0xFFFFFFFF	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>								
Privilege:	0x00000008	Port 31..24				<input checked="" type="checkbox"/>	Port 23..16				<input type="checkbox"/>	Port 15..8				<input type="checkbox"/>	Port 7..0				<input type="checkbox"/>

ITM Stimulus Port registers

Up to 256 ITM stimulus port registers (typical implementations provide 32)

- 0xE000_0000: ITM Stimulus Port 0
- 0xE000_0004: ITM Stimulus Port 1
- :
- 0xE000_0078: ITM Stimulus Port 30
- 0xE000_007C: ITM Stimulus Port 31



31	Port	24	23	Port	16	15	Port	8	7	Port	0				
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
Port 31..24 <input checked="" type="checkbox"/>				Port 23..16 <input type="checkbox"/>				Port 15..8 <input type="checkbox"/>				Port 7..0 <input type="checkbox"/>			

Access to the port registers takes two clocks (always zero wait state)

A read of any port register returns FIFO status

- 0 = Full
- 1 = Not Full

A write generates a software trace packet for the ITM FIFO

- A write into a full FIFO will result in the packet being lost

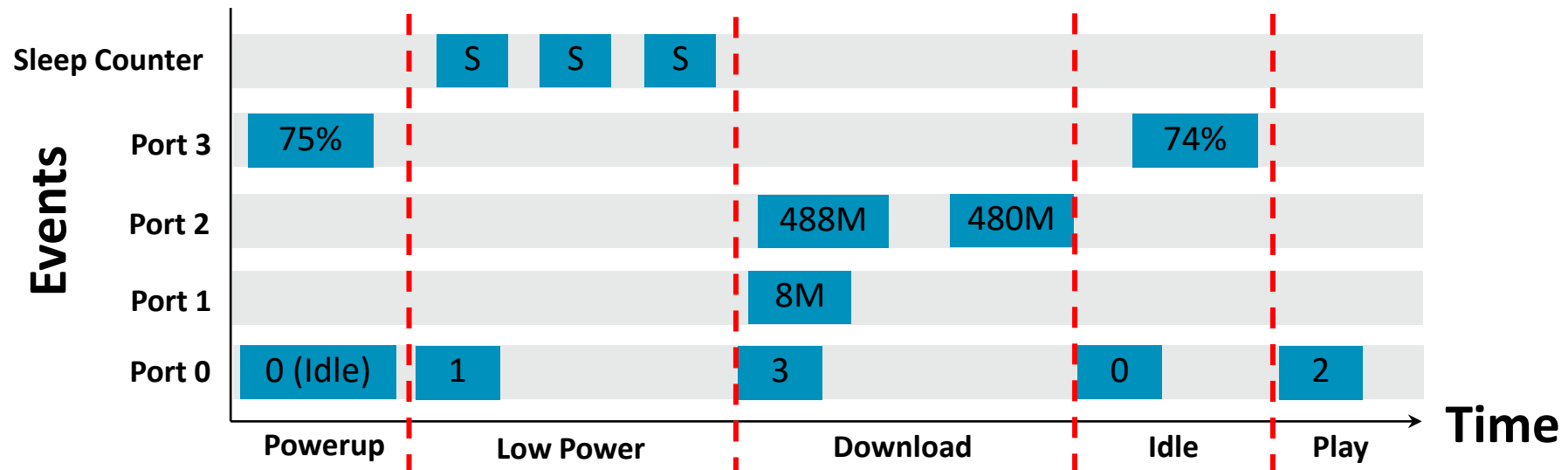
ITM example – MP3 Player Control Processor

ITM Packet Generation

- Port 0 – output new state (IDLE=0, LOW POWER=1, PLAY=2, DOWNLOAD=3)
- Port 1 – output size of new file in MB (on download)
- Port 2 – output remaining memory in MB (on download, after download)
- Port 3 – output battery level (every 5 minutes)

DWT Packet Generation

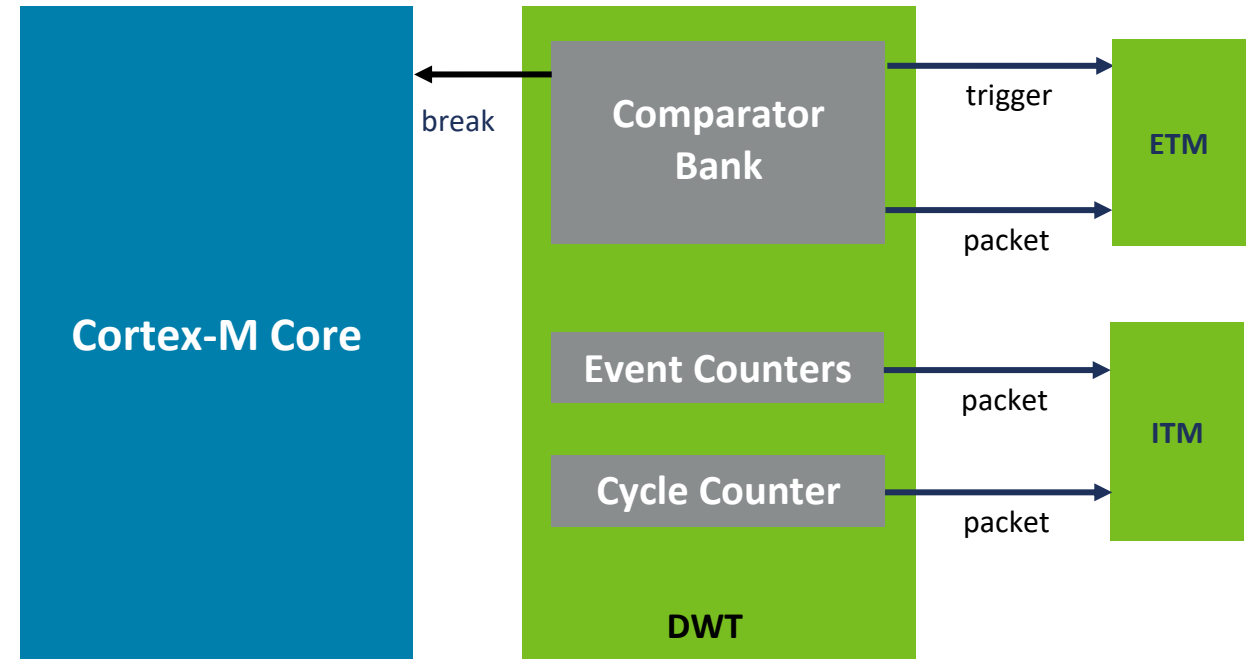
- Enable Sleep Counter (generates S packet)



DWT – trace capture

The DWT is also useful trace capture

- Packets generated for selected events of interest
 - Address matching
 - Periodic PC sampling
 - Data value
 - Exception entry exit and return
 - Event counters
- Packets sent through ITM
 - Must have debug tools connected to view the output
 - Packets are architecturally defined
- Generate trace trigger for ETM



Embedded Trace Macrocell (ETM)

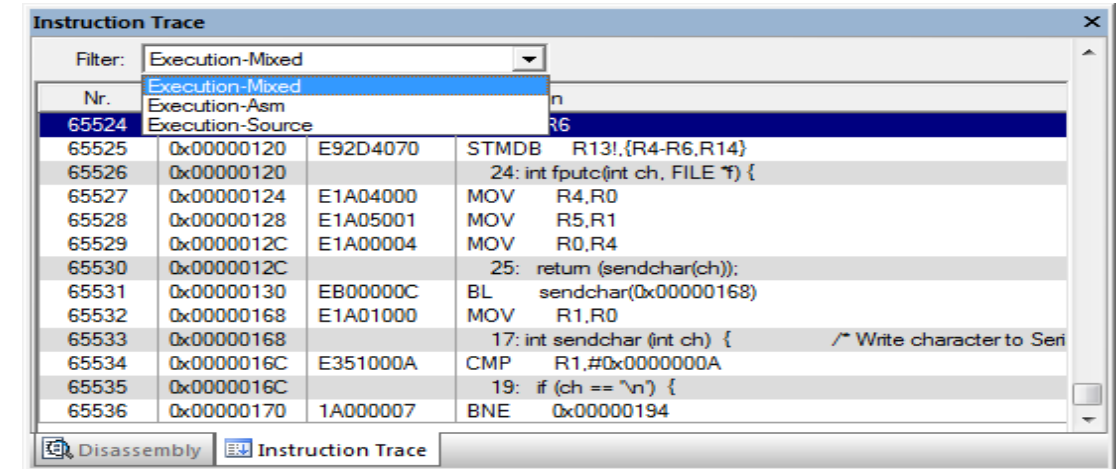
Optional non-invasive debug component

- Instruction trace only (no data trace – use DWT)

ETM hardware monitors activity of processor

Trace allows:

- Historical debug of sequences leading up to events of interest
 - e.g. System crash on peripheral access during overnight testing
- Debug of events in real-time systems where the target cannot be halted
 - Hard Disk drives, Engine Management
- Visibility of accesses inside a SoC
 - To internal memories/peripherals
- Software profiling and code coverage



Nr.	Address	Disassembly	Source
65524	0x00000120	E92D4070	STMDB R13!,{R4-R6,R14}
65525	0x00000120	E92D4070	24: int fputc(int ch, FILE *f) {
65526	0x00000120	E92D4070	
65527	0x00000124	E1A04000	MOV R4,R0
65528	0x00000128	E1A05001	MOV R5,R1
65529	0x0000012C	E1A00004	MOV R0,R4
65530	0x0000012C	E1A00004	25: return (sendchar(ch));
65531	0x00000130	EB00000C	BL sendchar(0x00000168)
65532	0x00000168	E1A01000	MOV R1,R0
65533	0x00000168	E1A01000	17: int sendchar (int ch) { /* Write character to Seri
65534	0x0000016C	E351000A	CMP R1,#0x0000000A
65535	0x0000016C	E351000A	19: if (ch == '\n') {
65536	0x00000170	1A000007	BNE 0x00000194

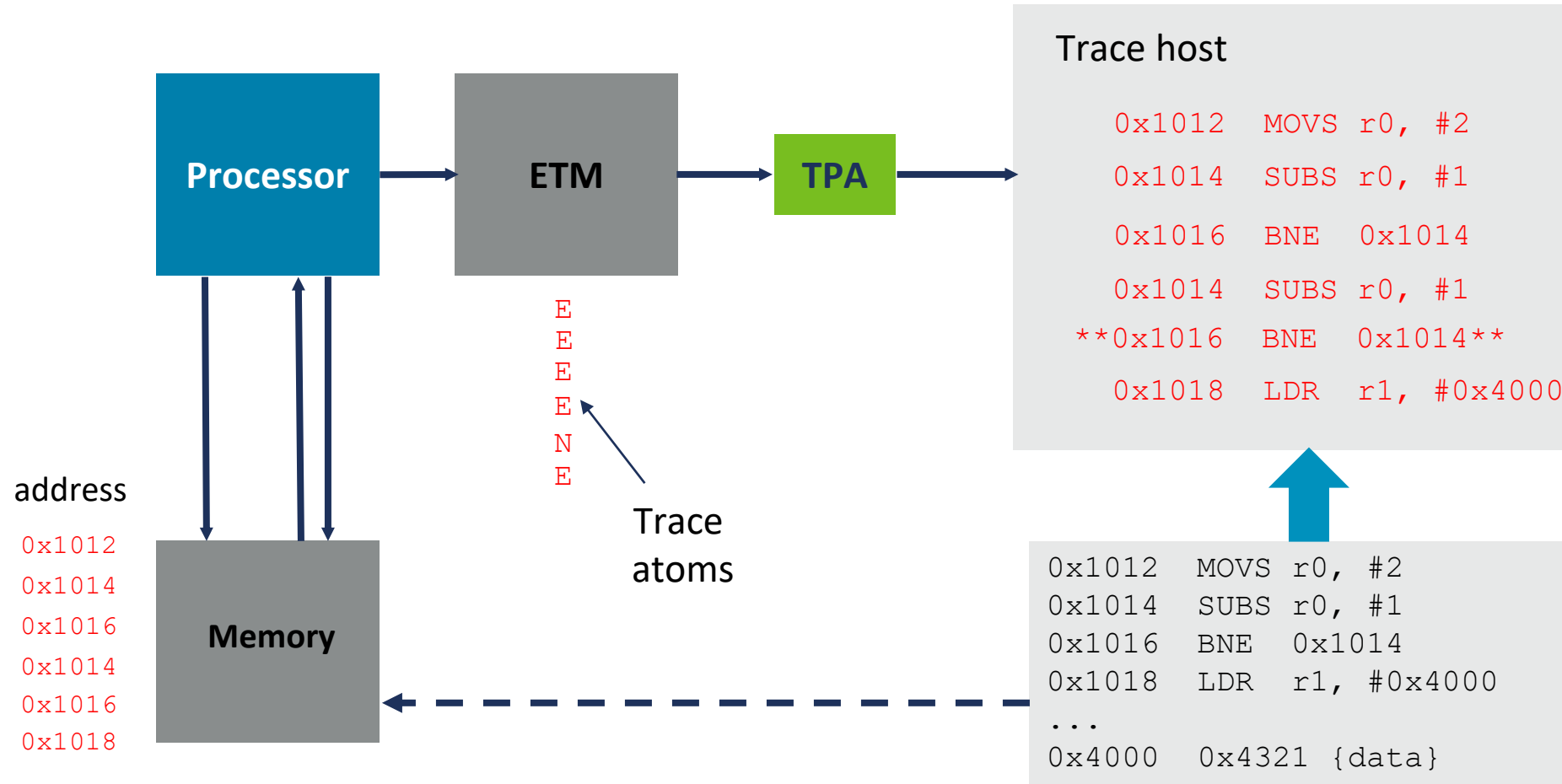
Core Clock: 96.000000 MHz

☒ Trace Enable

☒ UnlimitedTrace

☒ ETM Trace Enable

Basic ETM instruction trace operation

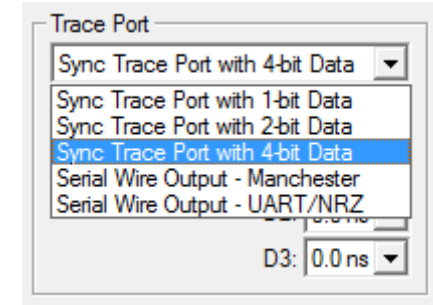


Trace Port Interface Unit (TPIU)

The TPIU formats and serializes data from ETM and/or ITM

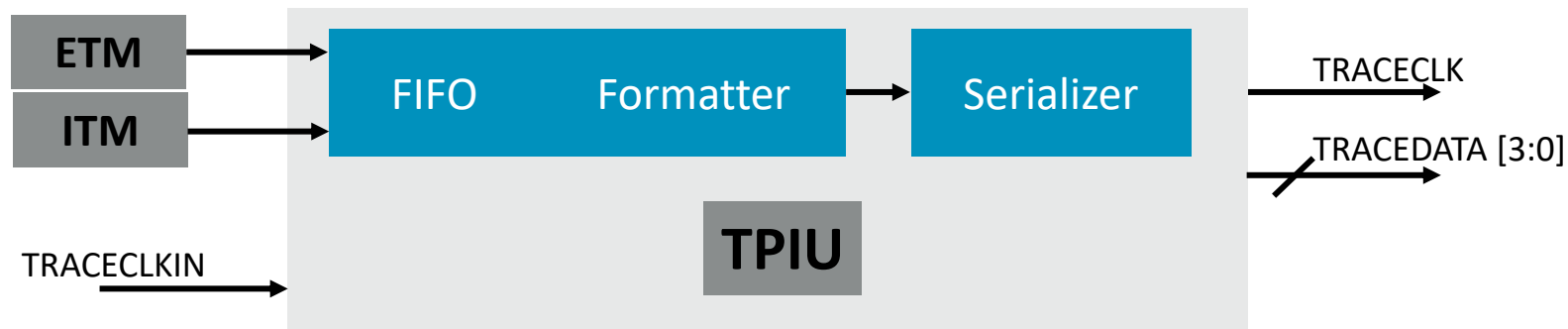
Trace data clocked out asynchronous to core clock

- TRACECLK derived from TRACECLKIN



ETM packets sent over trace port (TRACECLK and TRACEDATA [n:0])

- The supported trace port size can be 1-bit, 2-bit and 4-bit, typical implementations use 4-bit wide trace ports (shown below)
- Data decompressed with a conventional trace port analyzer



Trace ports and bandwidth

The ETM can produce data rates from near zero up to several bytes per instruction

- Use with care – overflowing data will be discarded

Instruction trace alone requires 2-3 pins for TRACEDATA

- When TRACECLKIN running at core clock speed

Using 4 pins for TRACEDATA

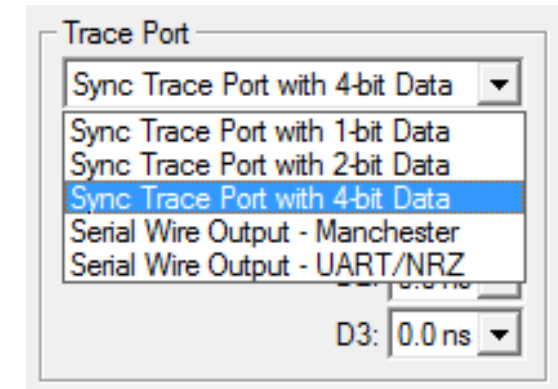
- Provides enough bandwidth for instruction trace

Scale the bandwidth according to HCLK/TRACECLKIN ratio

- May affect the required number of device pins

Silicon vendor must tie off MAXPORTSIZE input on chip

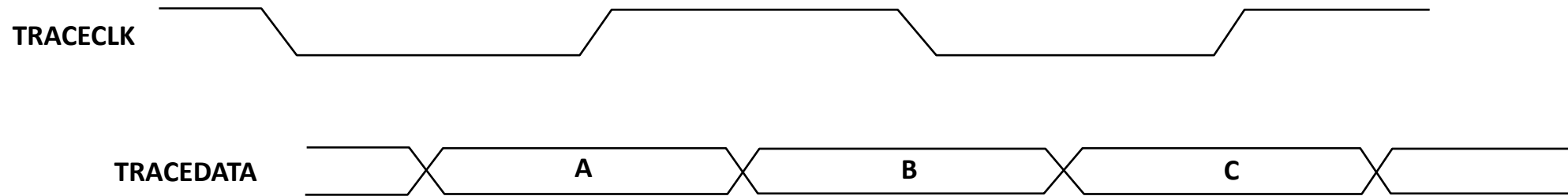
- This signal shows the maximum trace port size that can be configured, e.g., on Cortex-M55, this is 4 bits
- The ETM System Configuration Register (ETMSCR) indicates to the debugger the maximum support port size



Trace clock considerations

TRACECLKIN is divided by 2 to generate TRACECLK output

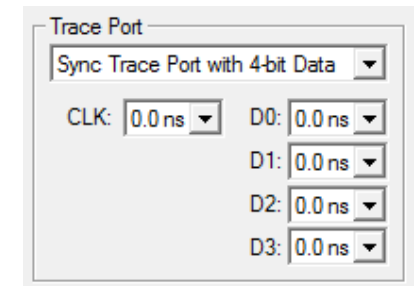
- TRACECLKIN is normally generated on-chip
- TRACEDATA is sampled on both edges
- TRACECLK frequency profile is similar to TRACEDATA



Some tools may expect TRACECLK edges to be centered in TRACEDATA

- Provide on-chip delay or use NegEdge of TRACECLKIN for the divider

Some tools may provide delay compensation



Agenda

Introduction to Armv8-M Debug & Trace

Debugging with Breakpoints, Watchpoints, Semihosting & Vector Catch

Tracing execution with the DWT, ITM, MTB, ETM

Profiling with the DWT & PMU

DWT – Performance profiling

The DWT contains counters for:

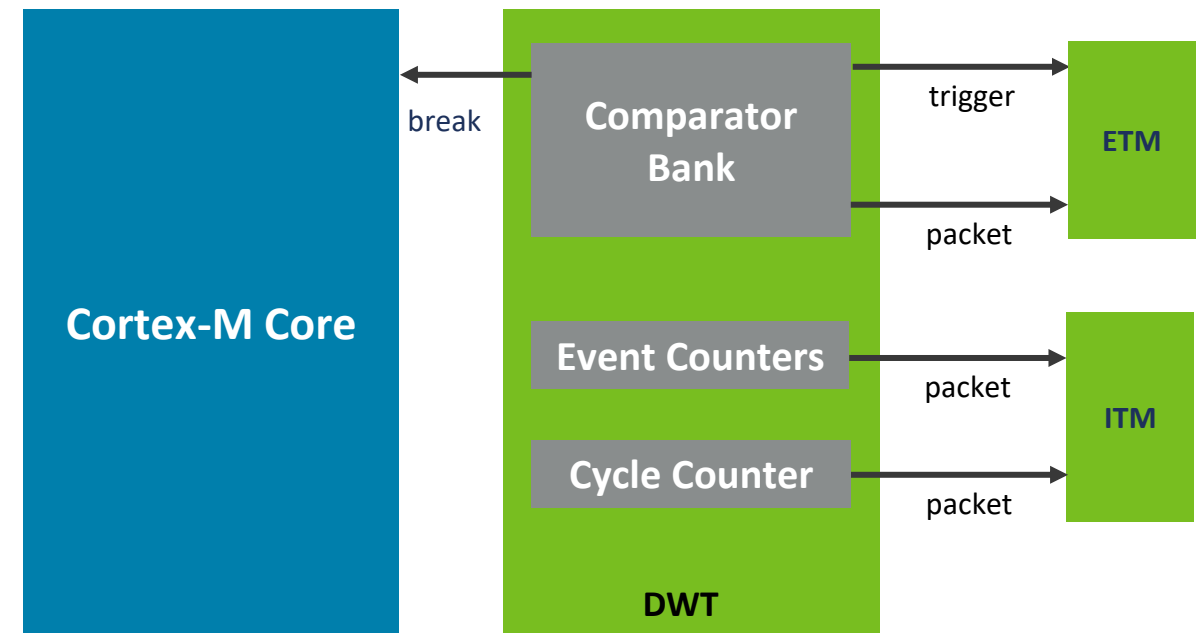
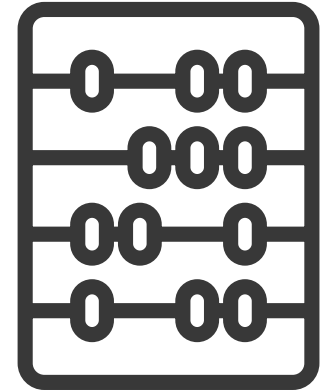
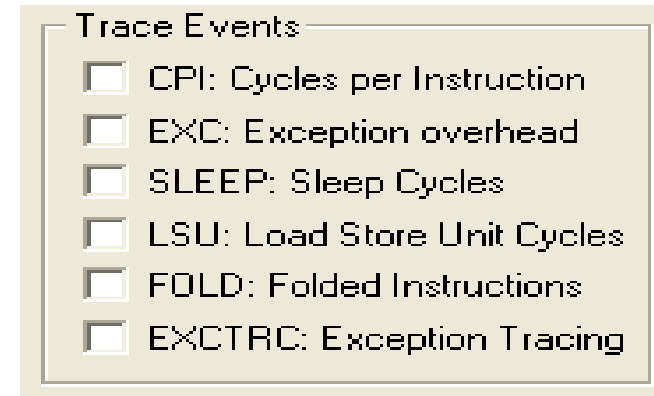
- Cycles (CYCCNT)
- Folded Instructions (FOLDCNT)
- Additional cycles required to execute all load or store instructions (LSUCNT)
- Processor sleep cycles (SLEEPcnt)
- Additional cycles required to execute multi-cycle instructions and instruction fetch stalls (CPICNT)
- Cycles spent in exception processing (EXCCNT)

ITM event packet generated only on counter overflow

- Counters cannot be read by the application code

No limit to the number of counters in use (0 – 5)

- Counters enabled in **DWT Control Register (DWT_CTRL)**



PMU – Performance Monitoring Unit

Armv8.1-M Architecture
only

Implements the Armv8.1-M Performance Monitoring Unit (PMU)

- Enables software to collect event statistics information and system debug to use it for performance analysis

Supports 31 16-bit event counters and one 32-bit cycle counter

- Each event counter can count one event from an event list
- Chain function allows cascading two 16-bit counters into one 32-bit counter

Cortex-M55 implements eight 16-bit counters

- The PMU is configured if the DWT is included in the processor

Operation the of the PMU counters and DWT profiling counters is mutually exclusive

PMU – interrupt / trace

Armv8.1-M Architecture
only

Counters can be configured to generate an interrupt upon overflow

- DEMCR.MON_PEND is set to 1 to make a Debug Monitor exception pended with DFSR.PMU set to 1
- The associated overflow bit programmed by PMU_OVSSET and PMU_OVSCLR indicates which counter triggered the exception

The PMU can export over trace if PMU_TYPE.TRO is RAO

- If PMU_CTRL.TRO is set, whenever the lower 8 bits of the counters overflow the PMU issues an event counter packet with the appropriate counter flag set to 1

References

Arm Developer

- Armv8-M Architecture Reference Manual
- CoreSight Architecture Specification (ARM IHI 0029)
- Arm Debug Interface v5 Architecture Specification (ARM IHI 0031)
- CoreSight Component Technical Reference Manual (ARM DDI 0314)
- ETM Architecture Specifications

Arm Development Studio (ArmDS)

Keil Microcontroller Development Kit (MDK-ARM)

ULINKPro User's Guide

- “Trace Data Not Synchronized”
http://www.keil.com/support/man/docs/ulinkpro/ulinkpro_desyncTraceData.htm

arm

Thank You

Danke

Gracias

Grazie

谢谢

ありがとう

Asante

Merci

감사합니다

धन्यवाद

Kiitos

شكراً

ধন্যবাদ

תודה