

Hardware trace for system visibility

Al Grant
al.grant@arm.com
BSDCam



The visibility problem

- What's going on?
- Kernel activity
- Userspace activity
- Interrupts
- Power management (DVFS, throttling, hotplug)
- Microarchitectural events (cache, prefetch...)
- Use of system resources (e.g. power)

Software trace

- programmer-inserted (printf debugging)
 - problem: no standard language mechanism
- compiler-inserted
 - e.g. `gcc -pg`
- post-link
 - DynInst
- dynamic
 - probes
- hybrids
 - predefined tracepoints, post-defined payload
 - e.g. `dtrace`

Other visibility mechanisms

- PC / PMU sampling
 - e.g. ARM Streamline
 - limited visibility; might not work if interrupts disabled
- Architectural virtualization
 - gem5, ARM Fast Models (FVP), QEMU
 - limited timing accuracy
- Hardware emulation
 - FPGA, simulation, h/w emulator
 - excellent accuracy (up to DDR & I/O) and visibility
 - expensive and/or slow, difficult to interact
- Dynamic binary instrumentation
 - Pin, DynamoRIO: <https://github.com/DynamoRIO/dynamorio>
 - mostly for userspace, loses timing accuracy



The DR. is in.

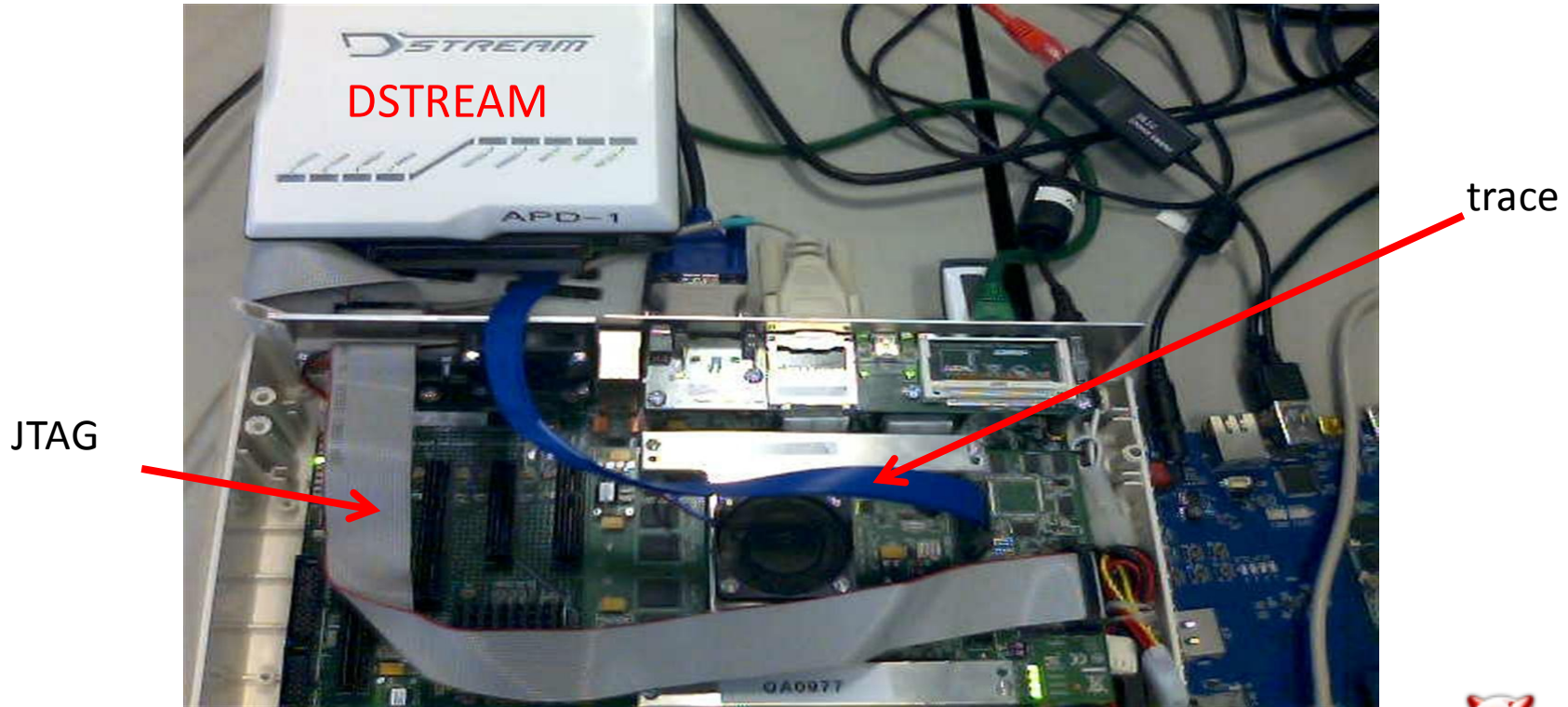


System visibility - compared

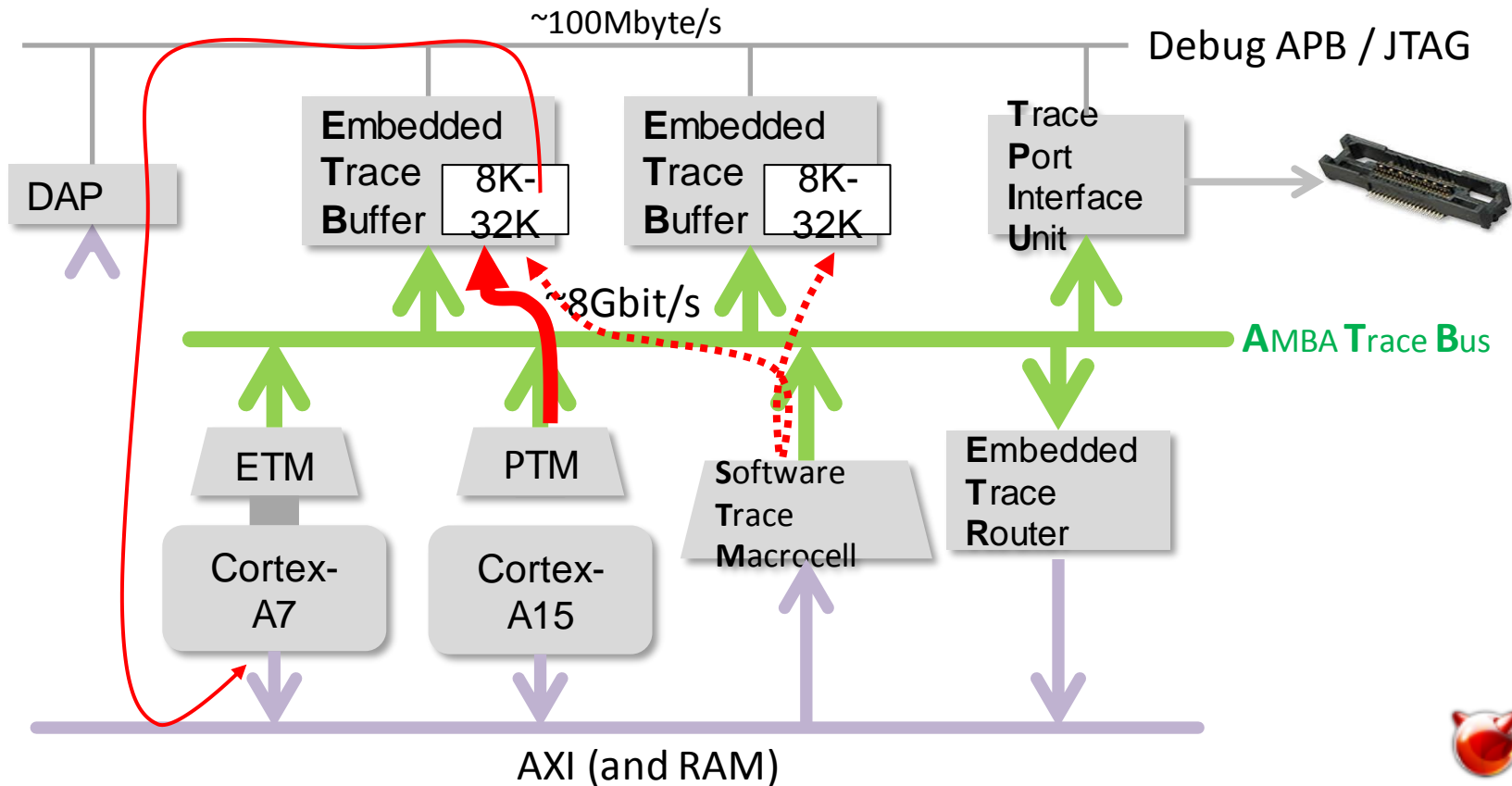
	\$ bn. cyc	Availability	Speed	Accuracy	Obs'ability
emulation	\$200	early	0.1MHz	high	high
simulation	\$20	early	1KHz	high	high
"fast RTL"	\$2	?	0.01MHz	high	medium
FPGA	10c	medium	10MHz	high	medium
gem5	2c	early(ish)	1MHz	medium	medium
fast models	0.02c	early	100MHz	low/medium	medium
silicon + dtrace	0.001c	late	1GHz	medium	medium

what can we do about this?

CoreSight – system visibility for ARM



CoreSight trace basics



Trace sources

- Program trace (CoreSight ETM/PTM)
 - branches (“waypoints”) + cycle counts
 - exceptions
 - data trace / instruction trace on embedded CPUs only
- Software trace (CoreSight STM/ITM)
 - messages under program control – conforms to MIPI STPv2
 - messages generated by writing to stimulus area in memory
- Hardware event trace (also CoreSight STM)
 - selected signals, as chosen by SoC designer
 - typically includes CPU interrupt lines
- Common global timestamp for all trace sources

Program trace

- Deep performance analysis
 - deep analysis of where time is spent in specific routines
 - cycle-accurate timing between branches
 - works for interrupt-disabled code
 - various filtering/selection features
 - user only, kernel only, address range filtering
 - non-invasive*
- Profiling
 - sample small fragments of processor trace
 - similar to Intel's Last Branch Record
- Post-mortem analysis (FMEA)
 - set up trace in circular buffer
 - stop trace on kernel crash or other interesting event

System call – two cores

```
163 ] (cpu2) --> mark_held_locks
166 ] (cpu2) @c0055f70:TMB e92d4ff0 PUSH {r4-r11,l,r}
175 ] (cpu2) @c0055f74:TMB 00004682 MOV r10,r0
176 ] (cpu2) @c0055f76:TMB f8d034b0 LDR r3,[r0,#0x4b0]
176 ] (cpu2) @c0055f7a:TMB 0000b083 SUB sp,sp,#0xc
178 ] (cpu2) @c0055f7c:TMB 00002b00 CMP r3,#0
178 ] (cpu2) @c0055f7e:TMB 0000dd4c BLE {pc}+0x9c; 0xc005601a
178 ] (cpu2) @c0055f7c:BR c005601a
178 ] (cpu2) --> mark_held_locks+0xaa
179 ] (cpu2) @c005601a:TMB 00002001 MOVs r0,#1
179 ] (cpu2) @c005601c:TMB 0000b003 ADD sp,sp,#0xc
182 ] (cpu2) @c005601e:TMB e8bd8ff0 POP {r4-r11,pc}
186 ] (cpu2) @c0056022:BR c0056096
186 ] (cpu2) --> trace_hardirqs_on_caller+0x62
187 ] (cpu2) @c0056096:TMB 0000b158 CBZ r0,{pc}+0x1a; 0xc00560b0
187 ] (cpu2) BR not taken
188 ] (cpu2) @c0056098:TMB f8d5349c LDR r3,[r5,#0x49c]
190 ] (cpu2) @c005609c:TMB 0000b973 CBNZ r3,{pc}+0x20; 0xc00560bc
190 ] (cpu2) @c005609a:BR c00560bc
190 ] (cpu2) --> trace_hardirqs_on_caller+0x88
198 ] (cpu2) @c00560bc:TMB 00004639 MOV r1,r7
198 ] (cpu2) @c00560be:TMB 00004628 MOV r0,r5
199 ] (cpu2) @c00560c0:TMB f7ffff56 BL {pc}-0x150; 0xc0055f70
199 ] (cpu2) @c00560c0:BR c0055f70
```

core #1: 33 cycles

```
350 ] (cpu1) --> mark_held_locks
350+ ] (cpu1) @c0055f70:TMB e92d4ff0 PUSH {r4-r11,l,r}
350+ ] (cpu1) @c0055f74:TMB 00004682 MOV r10,r0
350+ ] (cpu1) @c0055f76:TMB f8d034b0 LDR r3,[r0,#0x4b0]
350+ ] (cpu1) @c0055f7a:TMB 0000b083 SUB sp,sp,#0xc
350+ ] (cpu1) @c0055f7c:TMB 00002b00 CMP r3,#0
350+ ] (cpu1) @c0055f7e:TMB 0000dd4c BLE {pc}+0x9c; 0xc005601a
351 ] (cpu1) @c0055f7c:BR c005601a
351 ] (cpu1) --> mark_held_locks+0xaa
351+ ] (cpu1) @c005601a:TMB 00002001 MOVs r0,#1
351+ ] (cpu1) @c005601c:TMB 0000b003 ADD sp,sp,#0xc
351+ ] (cpu1) @c005601e:TMB e8bd8ff0 POP {r4-r11,pc}
353 ] (cpu1) @c0056022:BR c0056096
353 ] (cpu1) --> trace_hardirqs_on_caller+0x62
353+ ] (cpu1) @c0056096:TMB 0000b158 CBZ r0,{pc}+0x1a; 0xc00560b0
354 ] (cpu1) BR not taken
354+ ] (cpu1) @c0056098:TMB f8d5349c LDR r3,[r5,#0x49c]
354+ ] (cpu1) @c005609c:TMB 0000b973 CBNZ r3,{pc}+0x20; 0xc00560bc
355 ] (cpu1) @c005609a:BR c00560bc
355 ] (cpu1) --> trace_hardirqs_on_caller+0x88
355+ ] (cpu1) @c00560bc:TMB 00004639 MOV r1,r7
355+ ] (cpu1) @c00560be:TMB 00004628 MOV r0,r5
355+ ] (cpu1) @c00560c0:TMB f7ffff56 BL {pc}-0x150; 0xc0055f70
356 ] (cpu1) @c00560c0:BR c0055f70
```

core #2: 6 cycles (waypoints only)

System call – two cores

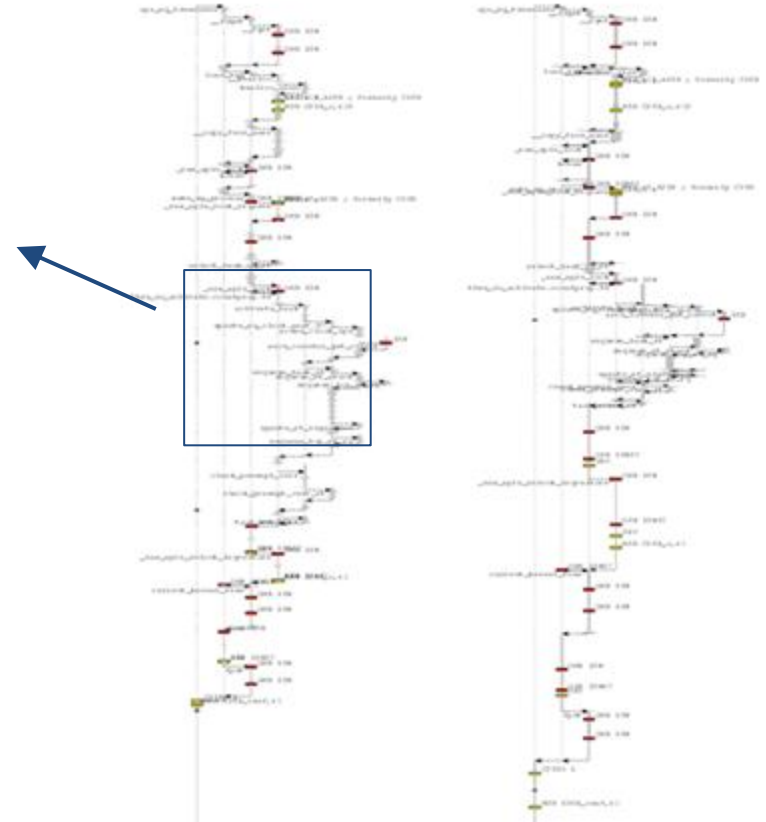
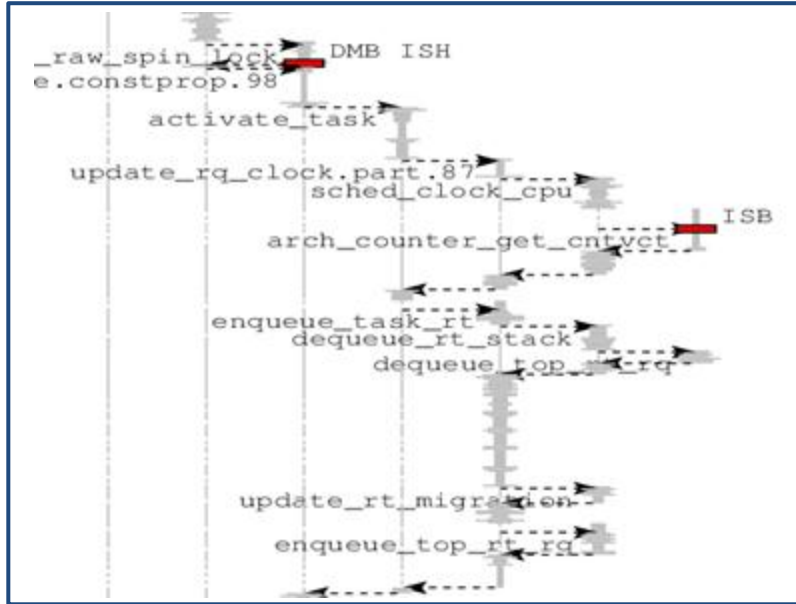
```
31 ] (cpu2)--> vector_swi
32 ] (cpu2) @c000cce0:TMB 0000b092 SUB sp,sp,#0x48
35 ] (cpu2) @c000cce2:TMB e88d1fff STM sp,{r0-r12}
44 ] (cpu2) @c000cce6:TMB 000046e8 MOV r8,sp
45 ] (cpu2) @c000cce8:TMB f3ef8a00 MRS r10,APSR ; formerly CPSR
47 ] (cpu2) @c000ccec:TMB f08a0a0c EOR r10,r10,#0xc
50 ] (cpu2) @c000ccf0:TMB f38a8100 MSR CPSR_c,r10
55 ] (cpu2) @c000ccf4:TMB f8c8d034 STR sp,[r8,#0x34]
56 ] (cpu2) @c000ccf8:TMB f8c8e038 STR lr,[r8,#0x38]
56 ] (cpu2) @c000ccfc:TMB f08a0a0c EOR r10,r10,#0xc
59 ] (cpu2) @c000cd00:TMB f38a8100 MSR CPSR_c,r10
64 ] (cpu2) @c000cd04:TMB f3ff8800 MRS r8,SPSR
65 ] (cpu2) @c000cd08:TMB f8cde03c STR lr,[sp,#0x3c]
66 ] (cpu2) @c000cd0c:TMB f8cd8040 STR r8,[sp,#0x40]
67 ] (cpu2) @c000cd10:TMB 00009011 STR r0,[sp,#0x44]
71 ] (cpu2) @c000cd12:TMB f8dfc0ac LDR r12,{pc}+0xae ; 0xc000cdc0
76 ] (cpu2) @c000cd16:TMB f8dcc000 LDR r12,[r12,#0]
77 ] (cpu2) @c000cd1a:TMB ee01cf10 MCR p15,#0x0,r12,c1,c0,#0
83 ] (cpu2) @c000cd1e:TMB e92d500f PUSH {r0-r3,r12,lr}
86 ] (cpu2) @c000cd22:TMB f049fa09 BL {pc}+0x49416 ; 0xc0056138
86 ] (cpu2) @c000cd22:BR c0056138
86 ] (cpu2)--> trace_hardirqs_on
```

core #1: 55 cycles

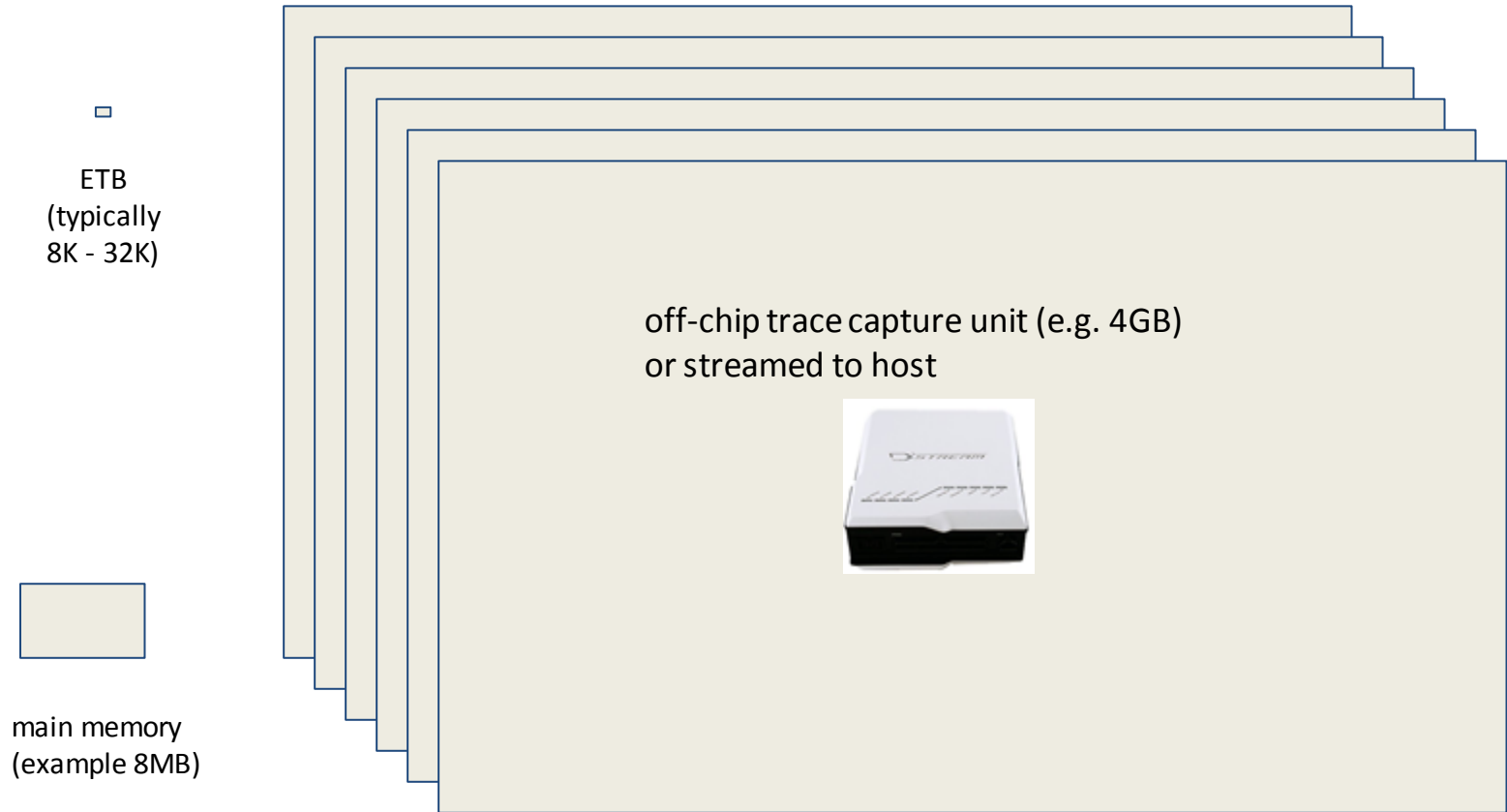
```
96 ] (cpu1)--> vector_swi
96+] (cpu1) @c000cce0:TMB 0000b092 SUB sp,sp,#0x48
96+] (cpu1) @c000cce2:TMB e88d1fff STM sp,{r0-r12}
96+] (cpu1) @c000cce6:TMB 000046e8 MOV r8,sp
96+] (cpu1) @c000cce8:TMB f3ef8a00 MRS r10,APSR ; formerly CPSR
96+] (cpu1) @c000ccec:TMB f08a0a0c EOR r10,r10,#0xc
96+] (cpu1) @c000ccf0:TMB f38a8100 MSR CPSR_c,r10
96+] (cpu1) @c000ccf4:TMB f8c8d034 STR sp,[r8,#0x34]
96+] (cpu1) @c000ccf8:TMB f8c8e038 STR lr,[r8,#0x38]
96+] (cpu1) @c000ccfc:TMB f08a0a0c EOR r10,r10,#0xc
96+] (cpu1) @c000cd00:TMB f38a8100 MSR CPSR_c,r10
96+] (cpu1) @c000cd04:TMB f3ff8800 MRS r8,SPSR
96+] (cpu1) @c000cd08:TMB f8cde03c STR lr,[sp,#0x3c]
96+] (cpu1) @c000cd0c:TMB f8cd8040 STR r8,[sp,#0x40]
96+] (cpu1) @c000cd10:TMB 00009011 STR r0,[sp,#0x44]
96+] (cpu1) @c000cd12:TMB f8dfc0ac LDR r12,{pc}+0xae ; 0xc000cdc0
96+] (cpu1) @c000cd16:TMB f8dcc000 LDR r12,[r12,#0]
96+] (cpu1) @c000cd1a:TMB ee01cf10 MCR p15,#0x0,r12,c1,c0,#0
96+] (cpu1) @c000cd1e:TMB e92d500f PUSH {r0-r3,r12,lr}
96+] (cpu1) @c000cd22:TMB f049fa09 BL {pc}+0x49416 ; 0xc0056138
299 ] (cpu1) @c000cd22:BR c0056138
299 ] (cpu1)--> trace_hardirqs_on
```

core #2: 203 cycles!

Kernel critical path – two cores



Where is trace captured?



How do I use it?

- Configure the CoreSight devices
 - JTAG
 - memory-mapped I/O
- Retrieve the trace
 - stream from external trace port
 - JTAG to embedded trace buffer
 - memory-mapped I/O to embedded trace buffer (slow)
 - trace to memory (where available)
- Decode the trace
 - need to know configuration
 - need program images
 - may need timestamped sideband data – see later
- Integrate all the above into useful tools e.g. profilers, debuggers

CoreSight device programming

- Linux kernel drivers
 - basic framework in 4.1
 - ARMv8 support being added in 4.2, 4.3
- CoreSight Access Library (CSAL)
 - available in DS-5 example code
 - <http://ds.arm.com/downloads/>
- Need to know CoreSight topology for SoC
 - discoverable by investigation (but not at boot time)
 - described in vendor documentation (sometimes)
 - Linux Device Tree
 - SoC-specific recipes may be needed (clocking, power etc.)

Decoding ETM/PTM trace

- You will need
 - the trace stream
 - details of how the trace sources were configured
 - images for kernel and program(s)
 - a decoder, e.g. <https://github.com/Linaro/OpenCSD/tree/arm-dev>
- For dynamically changing address spaces you will also need sideband info
 - timestamped events detailing when the address space mapping changes
 - e.g. loading dynamic libraries
 - context switch (unless CONTEXTIDR used)
 - could capture this through kernel trace, into ring buffer or CoreSight STM
- Trace decode will be integrated into Linux ‘perf’ tools
 - will be aligned with Intel’s Processor Trace support

STM – Software Trace Macrocell

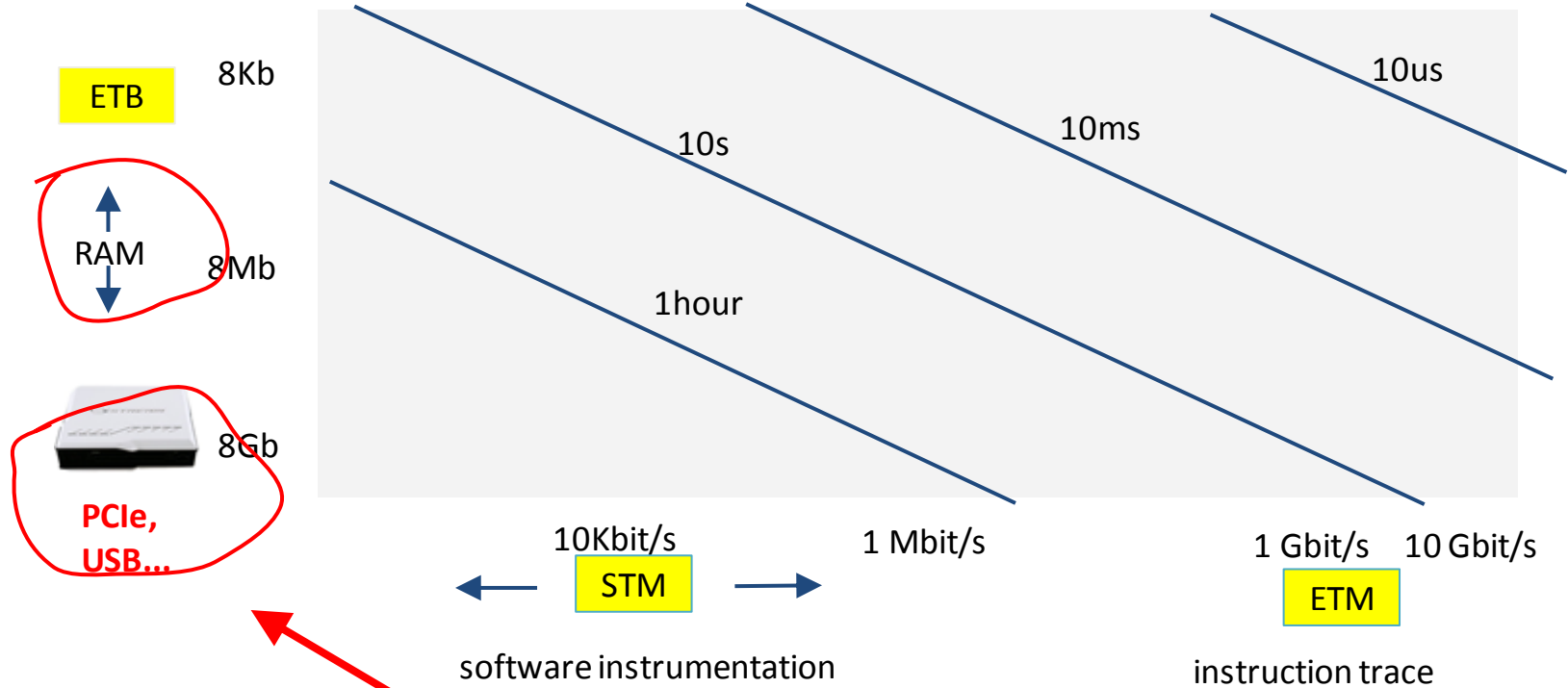
- Injects software-generated messages into the trace stream
- Messages generated by writing to stimulus port area
- Stimulus port area likely to be relatively fast
 - faster than CoreSight device programming registers
- Messages can be timestamped with CoreSight timestamp
- Messages can be blocking or non-blocking
- Message id and options determined by port address
- Stimulus port area can be mapped page by page
 - pages can be mapped into userspace
- Overall cost: generating and writing message data
 - tens of cycles
- No d-cache pollution or inter-core effects!
- Use cases: printf debugging, alternative to ring buffer
- CoreSight STM and Intel's Software Trace Hub are very similar
- Linux now has a generic STM framework covering both

For more information...

- ARM CoreSight:
 - <http://www.arm.com/products/system-ip/debug-trace/> (product info, aimed at SoC designers)
 - <http://infocenter.arm.com/help/index.jsp> (technical documentation)
 - <http://ds.arm.com/> (debugging tools)
- Linaro blog:
 - <https://www.linaro.org/blog/core-dump/coresight-initial-steps-supporting-hw-assisted-tracing-linux-arm-socs/>
- Generic STM driver:
 - <https://lwn.net/Articles/650245/>
- Thank you!

BACKUP

Trace bit rate and buffer size



CoreSight trace is becoming more practical and accessible!