RISC-V tracing

What is different?

Nino Vidovic, emBO++

25.03.2023 – 27.03.2023

Who am I?

Connect on LinkedIn

- Nino Vidovic
  - Software Engineer at SEGGER Microcontroller GmbH
  - Product owner of J-Trace Pro product group
  - Coordinator of SEGGER Educational Partnership Program
  - Expert on software development and analysis tools

Talk Content

- Introduction
- State of the art (Arm)
- RISC-V comparison
- Component spotlight
- Summary
- Conclusion

Introduction - What is tracing?

Tracing (software)

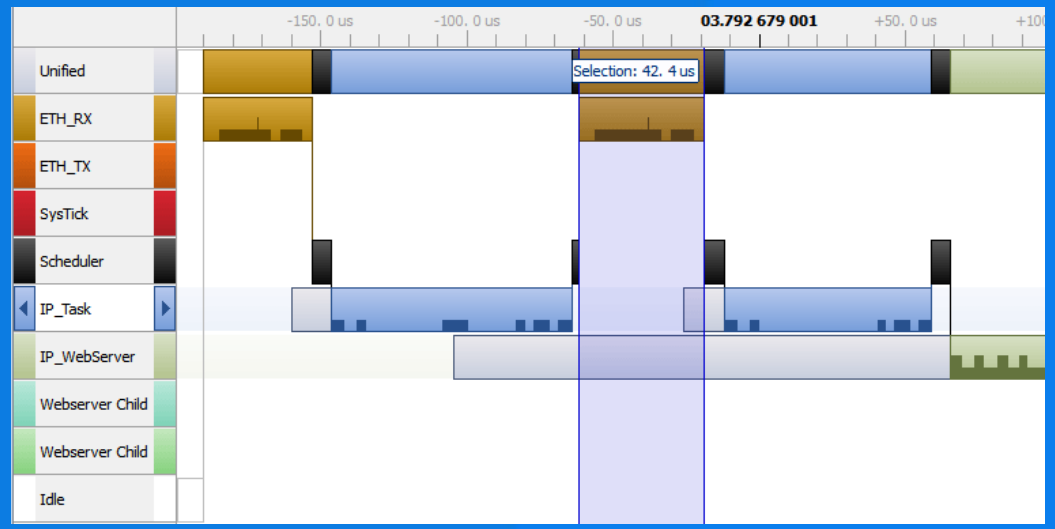
Article Talk

From Wikipedia, the free encyclopedia

In software engineering, **tracing** involves a specialized use of logging to record information about a program's execution. This information is typically used by programmers for debugging purposes, and additionally, depending on the type and detail of information contained in a trace log, by experienced system administrators or technical-support personnel and by software monitoring tools to diagnose common problems with software.[1] Tracing is a cross-cutting concern.

Introduction - Trace types

- Software-tracing (intrusive)
  - Eventlogging e.g. printf
  - Code instrumentation e.g. SystemView
- Hardware-tracing (non-intrusive)
  - On-Chip trace buffer
  - Pin tracing



State of the art - How Arm does it

- Arm Coresight trace components
  - Trace encoder
  - Trace buffer
  - Trace funnel
  - Trace port
- Hardware interface
  - Debug interface
    - Sufficient for buffer tracing
  - Trace pins
    - Needed for “unlimited” streaming trace

VTref	1 • • 2	SWDIO/TMS
GND	3 • • 4	SWCLK/TCK
GND	5 • • 6	SWO/TDO
- - -	7 • 8	TDI
NC	9 • • 10	nRESET
5V-Supply	11 • • 12	TRACECLK
5V-Supply	13 • • 14	TRACEDATA[0]
GND	15 • • 16	TRACEDATA[1]
GND	17 • • 18	TRACEDATA[2]
GND	19 • • 20	TRACEDATA[3]

MIPI 20 compatible

State of the art - How Arm does it

- Trace buffers

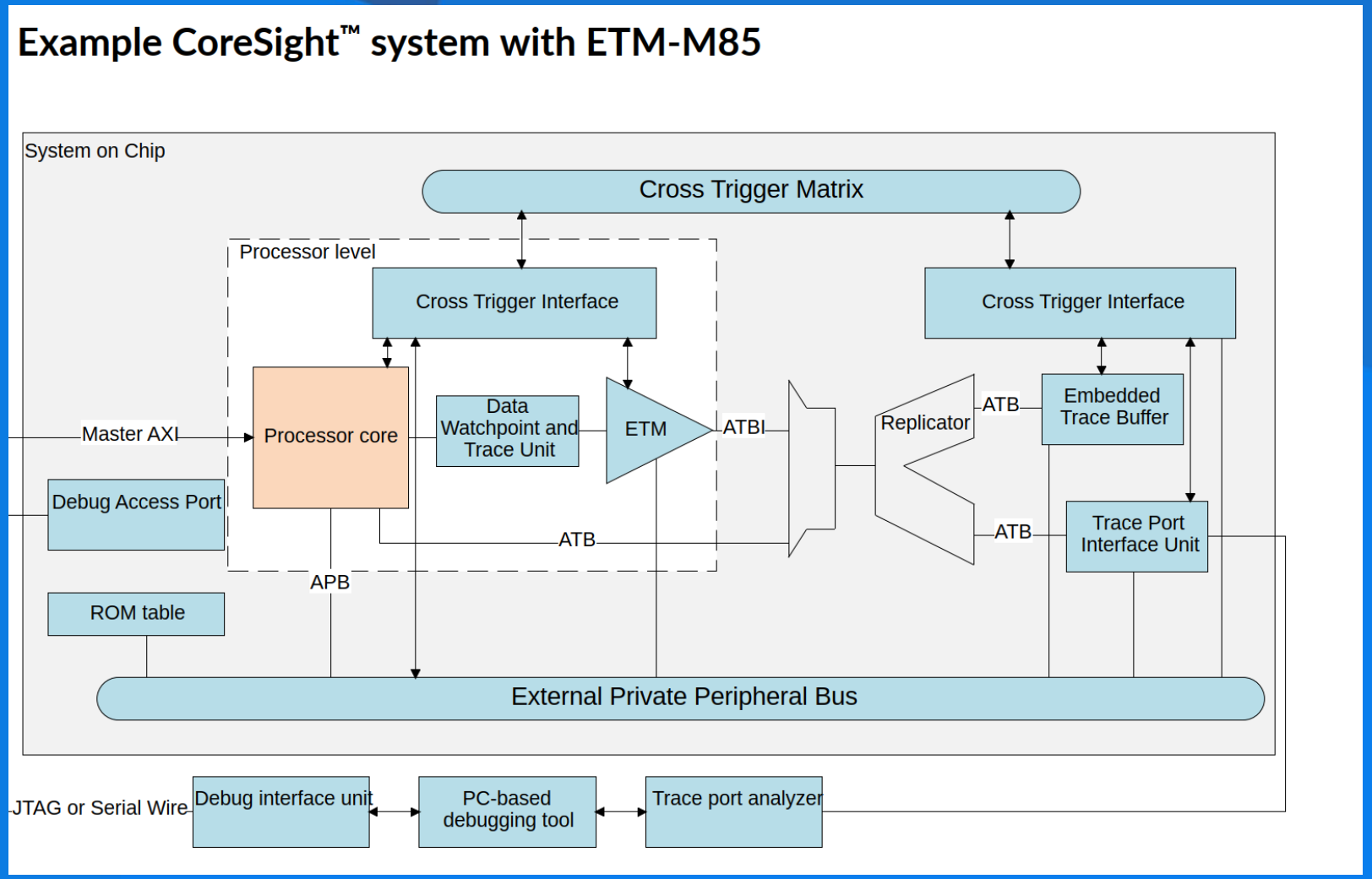
Micro Trace Buffer	MTB	Cortex-M0+ (M33)
Embedded Trace Buffer	ETB	Cortex-M3/M4/M7/M33/A/R
Trace Memory Controller	TMC	Cortex-M4/M7/M85

- Trace Encoders

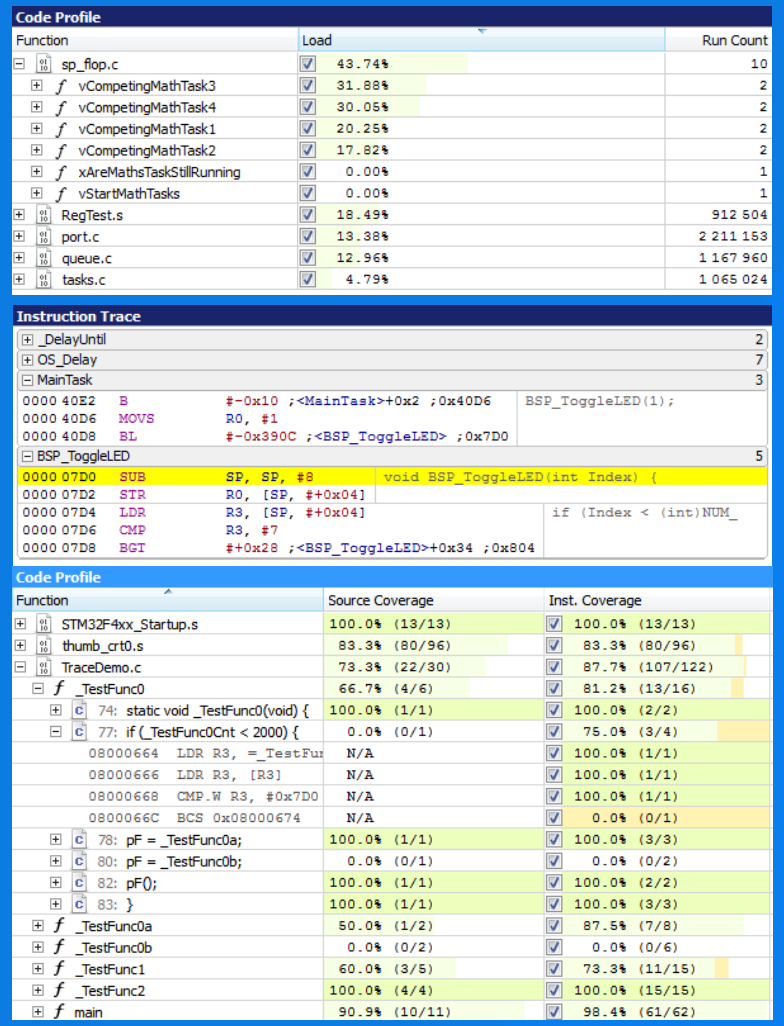
Embedded Trace Macrocell	ETM	Cortex-M/R
Program Trace Macrocell	PTM	Cortex-A
Instrumentation Trace Macrocell	ITM	Cortex-M

- Trace funnel (CSTF)
- Trace port interface unit (TPIU)

State of the art - How Arm does it



State of the art - Trace Encoder data quality



Code Profile

Function	Load	Run Count
sp_flop.c	43.74%	10
vCompetingMathTask3	31.88%	2
vCompetingMathTask4	30.05%	2
vCompetingMathTask1	20.25%	2
vCompetingMathTask2	17.82%	2
xAreMathsTaskStillRunning	0.00%	1
vStartMathTasks	0.00%	1
RegTest.s	18.49%	912 504
port.c	13.38%	2 211 153
queue.c	12.96%	1 167 960
tasks.c	4.79%	1 065 024

Instruction Trace

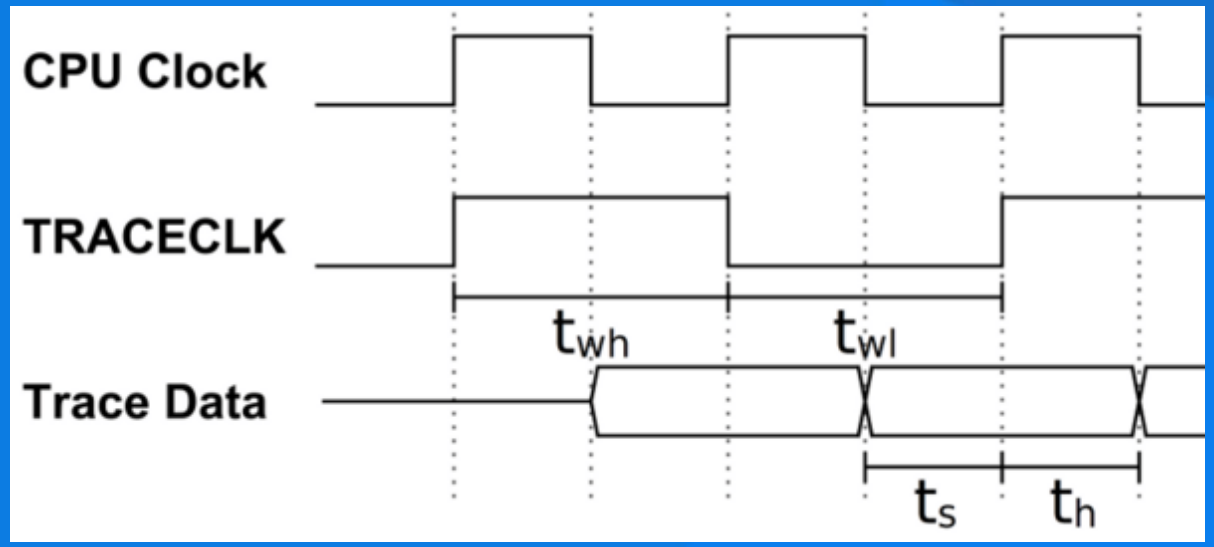
_DelayUntil			2
OS_Delay			7
MainTask			3
0000 40E2	B	#-0x10 ;<MainTask>+0x2 ;0x40D6	BSP_ToggleLED(1);
0000 40D6	MOVS	R0, #1	
0000 40D8	BL	#-0x390C ;<BSP_ToggleLED> ;0x7D0	
BSP_ToggleLED			5
0000 07D0	SUB	SP, SP, #8	void BSP_ToggleLED(int Index) {
0000 07D2	STR	R0, [SP, #+0x04]	
0000 07D4	LDR	R3, [SP, #+0x04]	if (Index < (int)NUM_
0000 07D6	CMP	R3, #7	
0000 07D8	BGT	#+0x28 ;<BSP_ToggleLED>+0x34 ;0x804	

Code Profile

Function	Source Coverage	Inst. Coverage
STM32F4xx_Startup.s	100.0% (13/13)	100.0% (13/13)
thumb_crt0.s	83.3% (80/96)	83.3% (80/96)
TraceDemo.c	73.3% (22/30)	87.7% (107/122)
_TestFunc0	66.7% (4/6)	81.2% (13/16)
74: static void _TestFunc0(void) {	100.0% (1/1)	100.0% (2/2)
77: if (_TestFunc0Cnt < 2000) {	0.0% (0/1)	75.0% (3/4)
08000664 LDR R3, =_TestFu	N/A	100.0% (1/1)
08000666 LDR R3, [R3]	N/A	100.0% (1/1)
08000668 CMP.W R3, #0x7D0	N/A	100.0% (1/1)
0800066C BCS 0x08000674	N/A	0.0% (0/1)
78: pF = _TestFunc0a;	100.0% (1/1)	100.0% (3/3)
80: pF = _TestFunc0b;	0.0% (0/1)	0.0% (0/2)
82: pF();	100.0% (1/1)	100.0% (2/2)
83: }	100.0% (1/1)	100.0% (3/3)
_TestFunc0a	50.0% (1/2)	87.5% (7/8)
_TestFunc0b	0.0% (0/2)	0.0% (0/6)
_TestFunc1	60.0% (3/5)	73.3% (11/15)
_TestFunc2	100.0% (4/4)	100.0% (15/15)
main	90.9% (10/11)	98.4% (61/62)

- ITM
  - Event based PC log (e.g. exceptions)
  - Periodic PC samples
- MTB (Cortex-M0+ “exclusive”)
  - Gapless information about code execution
  - No compression
  - Non sequential program flow is output
- ETMv3, PTM
  - Added compression
  - All non inferable branches are output + sync
- ETMv4
  - Same as ETMv3 + additional compression

State of the art - Timing requirements



Signal Name	Description	Value
twl	TRACECLK LOW pulse width	Min. 2 ns
twh	TRACECLK HIGH pulse width	Min. 2 ns
tr/tf	Clock and data rise/fall time	Max. 3 ns
ts	Data setup time	Min. 3 ns
th	Data hold time	Min. 2 ns

- Double data rate
- Max. 100 MHz Trace clock
- J-Trace Pro, up to 200 MHz

RISC-V Trace Plan

- Based on 9. February 2023 version of tg-nexus-trace git repo

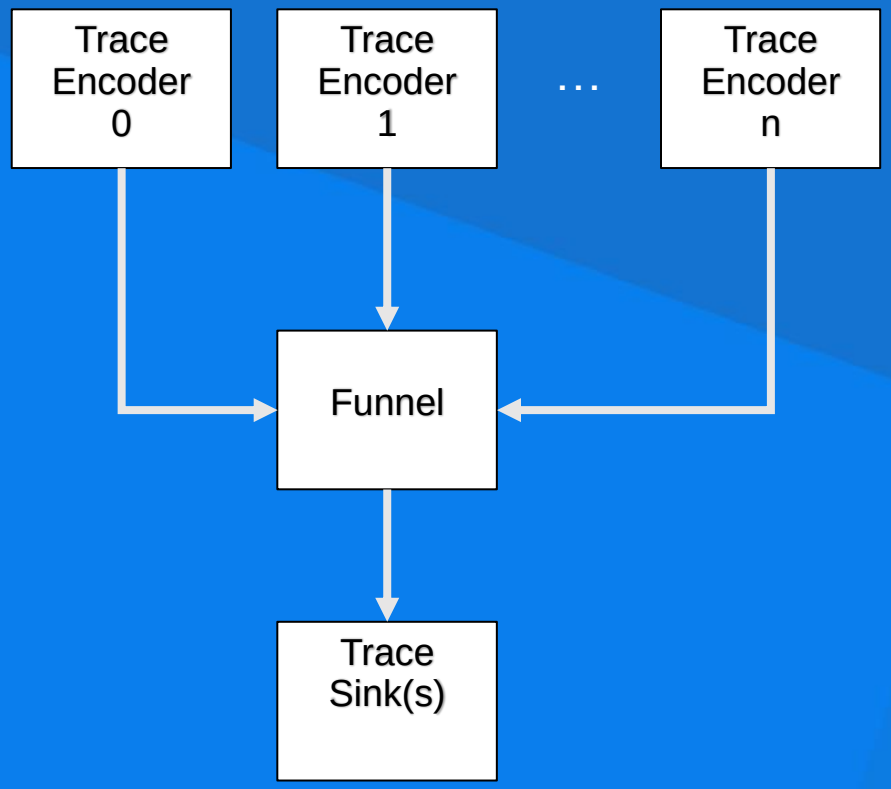
- Plan
  - Same pin interface as Arm
  - Nexus trace compatible pin protocol
  - Two trace encoder variants
    - Nexus Trace (SiFive)
    - Efficient Trace (Siemens)
  - Better compression than Arm

Nexus Standard

- Nexus/IEEE-ISTO 5001-2003 defines a modern debugging interface for embedded systems
- Nexus Trace is a substandard
  - Program trace
  - Data trace
  - Ownership trace
- Nexus specification defines 4 implementation classes
- Arm and RISC-V defines Class 2 as default
- Defines message protocol

RISC-V Nexus Trace components

- Trace Encoder
  - Branch Trace Messages (BTM)
  - History Trace Messages (HTM)
- Trace sinks
  - SRAM
  - System memory
  - PIB (trace pins)
  - Funnel
- Trace funnel

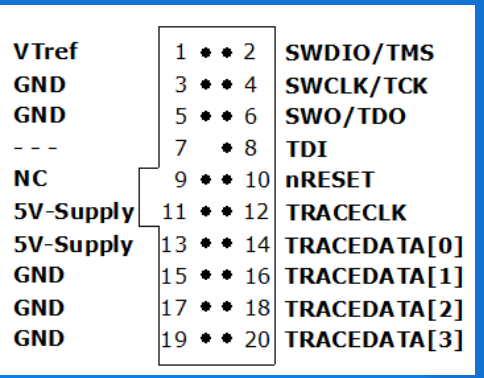


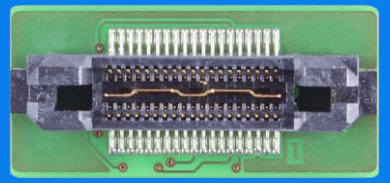
Trace component comparison

RISC-V Nexus Trace	Arm Coresight Trace
Trace Encoder BTM mode	ETMv3
Trace Encoder HTM mode	ETMv4
SRAM trace sink	ETB/TMC
System memory trace sink	TMC
PIB trace sink	TPIU

Trace Connector

- Trace connectors are identical
- MIPI20 defines up to 4 data pins + 1 trace clock
  - Minimum implementation 1 data pin
  - Superset of standard debug connector
- Mictor-38 defines up to
  - 16 data pins
  - 1 trace clock
  - 4 optional trace pins





Trace Encoder Features

- Instruction tracing
- BTM = standard branch tracing + sync
- HTM = BTM data + compression
- Ownership trace (for e.g. OS context switches)
- Triggers
- Stalling
- Different sync modes
- Optional data trace or timestamps

Funnel Features

- Combines trace messages from different sources
- Input channel priority “implementation defined”
  - Round-robin principle recommended
- Optional timestamp generator
- Support for 4 trace sink types

Nexus message format

- Variable and fixed fields
- Starts with TCODE field
  - Unique package identifier
- 6 bits available for TCODE -> 64 messages available
  - Currently 12 messages in use
- All messages end with optional TSTAMP field

Nexus message format

Example: Direct Branch Message

Bits	Name	Description
6	TCODE	Transfer code = 3
Configurable	SRC	Message source
Variable	I-CNT	Instruction count
Variable, Configurable	TSTAMP	Timestamp

Nexus trace transmission protocol

- Message Start/End Out (MSEO)
  - 1 or 2 bits (N-Trace = 2 bits)
  - MSEO = 00 -> Message Start/Normal transfer
  - MSEO = 01 -> End of variable length field
  - MSEO = 11 -> End of message
- Message Data Out (MDO)
  - At least one bit (N-Trace = 6 bits)
- Message split up into slices
  - 6 + 2 bits per slice = 1 byte aligned

Nexus trace transmission protocol

Example: Direct Branch Message

	MDO						MSEO		
Slice	**5**	**4**	**3**	**2**	**1**	**0**	**1**	**0**	**Notes**
	X	X	X	X	X	X	1	1	Idle or end of previous message
0	T5	T4	T3	T2	T1	T0	0	0	Start of message
1	I3	I2	I1	I0	S1	S0	0	0	Normal transfer
2	0	0	0	0	I5	I4	0	1	End of var field I-CNT
3	P5	P4	P3	P2	P1	P0	0	0	Normal transfer
4	0	0	0	0	P7	P6	1	1	End of message

Tx = TCODE, Sx = SRC, Ix = I-CNT, Px = TSTAMP

Summary

- Nexus compatible substandard
  - Including trace connector
- Very flexible/expandable message format
- Compression level highly configurable
  - Default compression already very good
- Flexible and simple Funnel interface
  - Various trace sink options
- Simplified Nexus transmission protocol

What is good/better?

- Nexus standard compatible
  - Higher chance to be supported by trace probe vendors
- More compact transmission protocol
- Potential for fast analyzers
- Many configuration options for implementers
- Compatible with other third-party trace types
- MIPI20 standard compatible trace pin interface
- Easily expandable messages
- Fewer forced sync packets

What is bad/worse?

- Allows trace message supersets
- Compression sometimes too good for live analysis
- Many configuration options for implementers
- No auto detection for trace components
- Wrongly sampled data harder to detect
- Funnel priority not configurable

Conclusion

- Competitive, open trace standard
- Ratification in process
  - Should be done by end of Q2 2023
- Promising feature set and easily expandable
- Base components comparable to Arm trace
  - In some areas (compression, flexibility) appears to be better
- But still no silicon with trace

RISC-V trace at SEGGER

- Preliminary trace buffer support
  - March 26th 2020
- Product launch of J-Trace Pro for RISC-V
  - October 24th 2022
  - Support for N-Trace BTM
  - N-Trace HTM status, under review
  - Plan to support other popular RISC-V trace encoders

THANK YOU