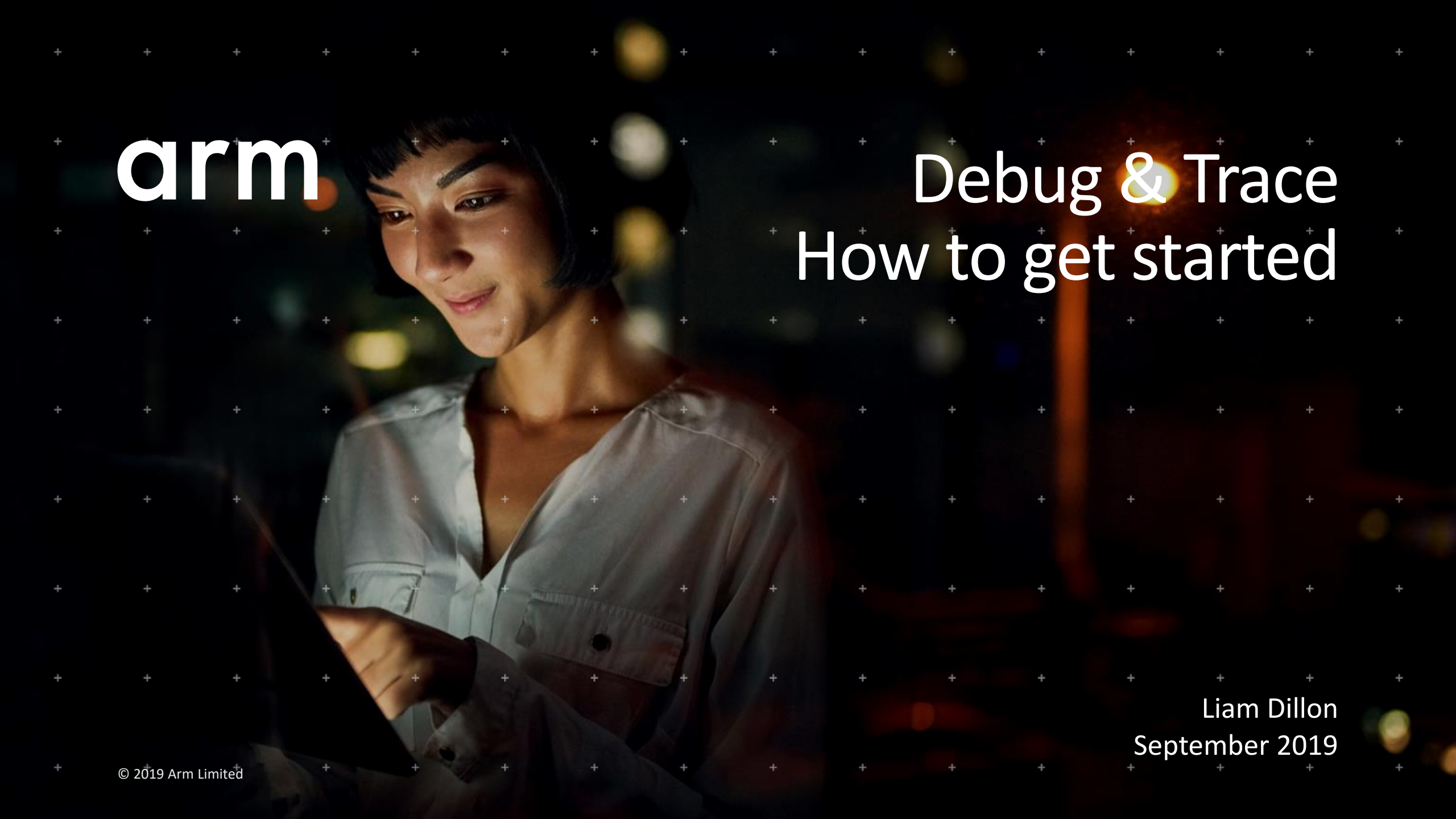


arm

Debug & Trace
How to get started

Liam Dillon

September 2019

Why bother with debug and trace?

- SoC/FPGA bring up
  - Faster bring up by extracting hardware information
- Supports Software debugging
  - Improves reliability
  - Allows hardware/software co-development via debugger in simulation/emulation
- System Optimization
  - Performance profiling
  - Benchmarking
  - Performance optimization

How do you get started?

1. Evaluate your Options
   - Perform a SoC Architecture review and use case analysis
2. Capture Requirements
   - Map requirements to debug components
3. Design your system
   - Configure your components
4. Verification
   - Work your way up from IP integration to use cases

1. Evaluate Options

- The key is to think ahead to support your debug and trace use cases
  - What ARM architecture are you using, consider appropriate IP
  - Understand the processor debug and trace capabilities (RTM)
- Hardware tracing
  - Thinking about adding ETM
  - Perform trace bandwidth calculations
    - Consider trade-offs of bandwidth, capture devices, silicon area & pin requirements
- Software tracing
  - Debugger connect to CPU or interconnect?
  - Self hosted debug
    - Allows “flight recorder” mode where trace data is capture during execution
    - Supports remote debugging
    - Enables Performance analyses using trace

2. Requirement Capture - what do I need?

Specify the debug and trace functionality needed :Cortex A57.A53 big.LITTLE example

Debug and Trace Options	System Requirements	Debug Component Mapping
Architectural Requirements	Cortex-A57/53: ARMv8, System Controller: ARMv7M	APB Control bus for Cortex-A57/53, DAPBUS Control bus for Cortex-M3
Hardware Tracing	Cortex-A57/53 (ETMs): 32 bit trace bus per CPU, Cortex-M3: two (ITM & ETM) 8 bit trace buses	ETMs, ATB Funnel, Replicator, Downsizer, Upsizer, TF & ETR(TMC), Asynchronous ATB bridges, Synchronous ATB bridges
Software Tracing	64 Bit access support, Individual CPU cores, GPU, Debugger and other systems masters to generate instrumentation. Instrumentation of power-control signals using the HWEVENTS interface	STM-500
Multiple Trace Sources Correlation	Debug timestamp distribution toCortex-A57/53 and STM-500	Timestamp components Timestamp (TS) generator, TS Interpolator Narrow Timestamp Asynchronous Bridges, TS Encoder/Decoder
Debug Access to system	Direct access to system memory and peripherals via the AXI interconnects. Provide access to “physical memory”	AXI-AP to connect debugger to NIC-400 interconnect
Cross-communication of debug events	CTM channels for Cortex-A57/53 clusters, CTI for Cortex-M3 events, CTI for CoreSight component trigger events (STM, ETF, ETR & TPIU). Interconnectivity of all triggers using CTM channels	CTIs, CTMs, Event Bridges for asynchronous crossing of events across clock and power domains
big.LITTLE debug –Multiple Power Domains	Cortex-A57 & Cortex-A53 in big.LITTLE configuration. Support for cross-halting via cross-trigger interfaces	Asynchronous bridges –ATB, APB and Event bridges CTI & CTM
Self-Hosted Debug	Use ETR and its AXI port to route trace data to system memory via the fabric. Provide system masters to access CoreSight components via the interconnect. Also, add control for debug authentication signals	ETR Debug APB bus Debug authentication interface

arm

3. Design – How do I build it?

- Break it down into 4 subsystems
  - Debug
    - Decide access to debug and trace components and external debugger access point
    - Configure interconnect and add async and sync bridges for clock/power domain crossing.
  - Trace
    - Once you know what TRACE functionality you need, configure the TMC
    - Configure the ATB upsizers, downsizers and funnels where needed
    - Add async and sync bridges for clock/power domain crossings.
  - Cross trigger
    - Configure individual trigger connectivity for CTI & CTM
  - Timestamp
    - Balance the timestamp distribution network across the SoC
- Integration
  - Once configured, each component has an IP-XACT description that can be using by a stitching tool

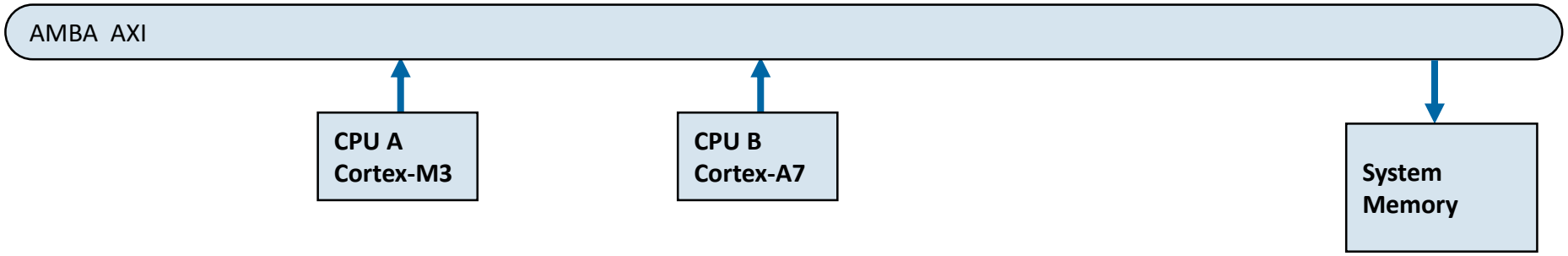
4. Verification –How to make sure it works?

- Take a system wide view
- Use the tools : CoreSight SoC provides a powerful and flexible environment
- Typical flow would follow
  - Connectivity and Integration testing
  - Key functional feature checks
  - Advanced functional feature checks
  - Scenario based verification

Arm SoC-400

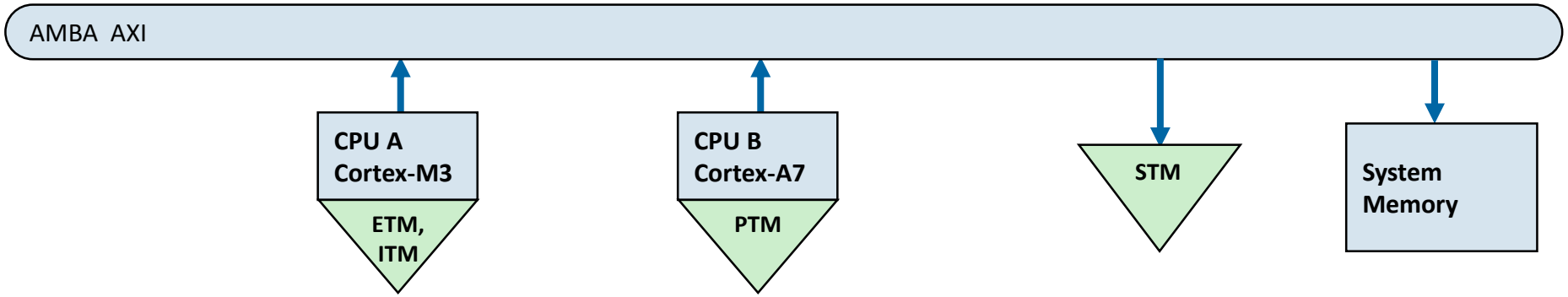
- Provides a catalogue of configurable IP for your debug solution
  - Includes IP-XACT description, scripts and verification IP
  - Comprehensive documentation
  - Fully compatible with all ARM CPUs
- Support from embedded Cortex-M designs to large Cortex-A systems.
  - Supported by 25+ development tool vendors including of course armDS
  - CoreSight SoC gives you on chip visibility, real-time analysis and minimises risk

Example Debug/Trace Subsystem using CoreSight SoC-400



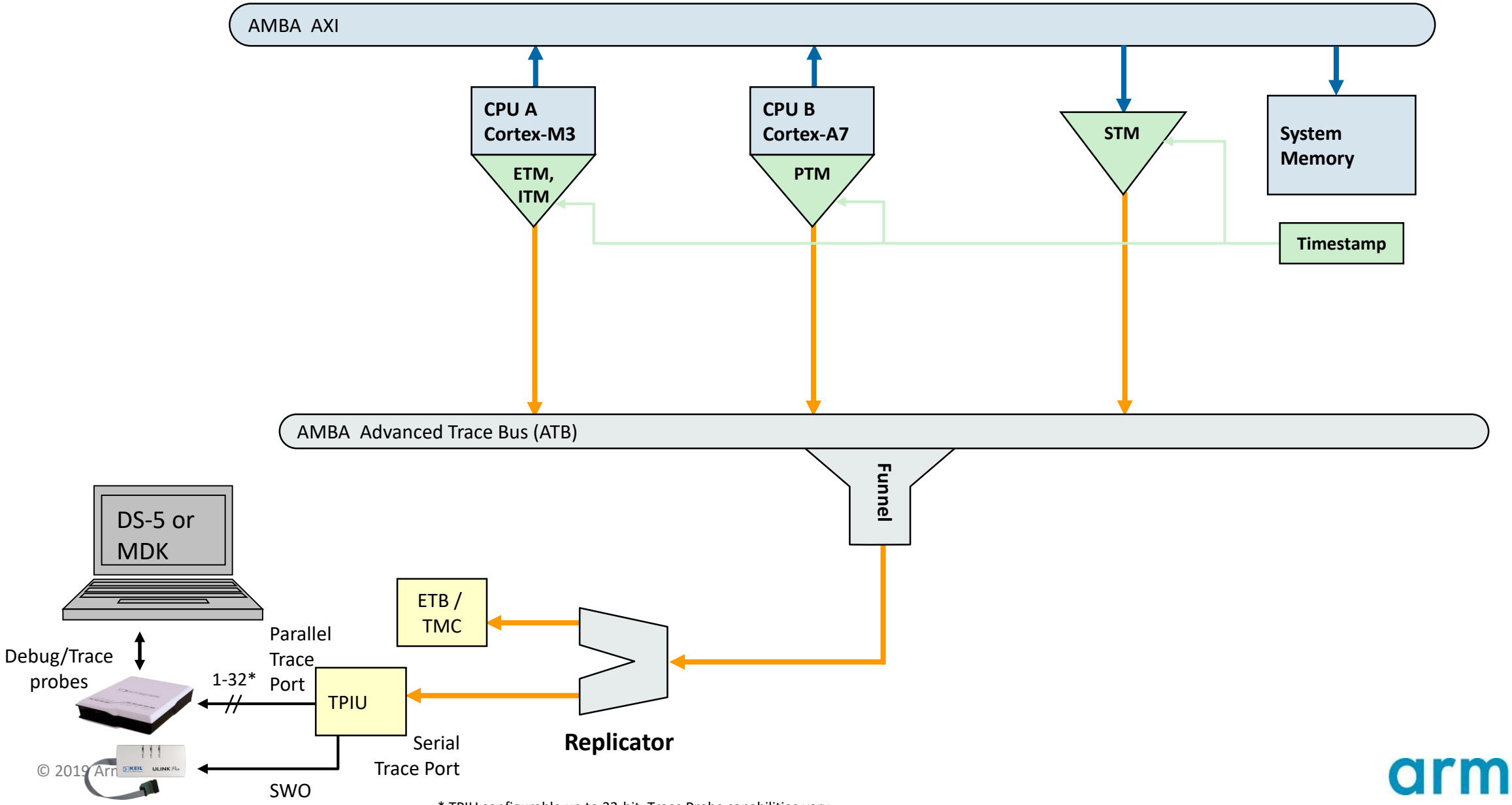
arm

Example Debug/Trace Subsystem using CoreSight SoC-400



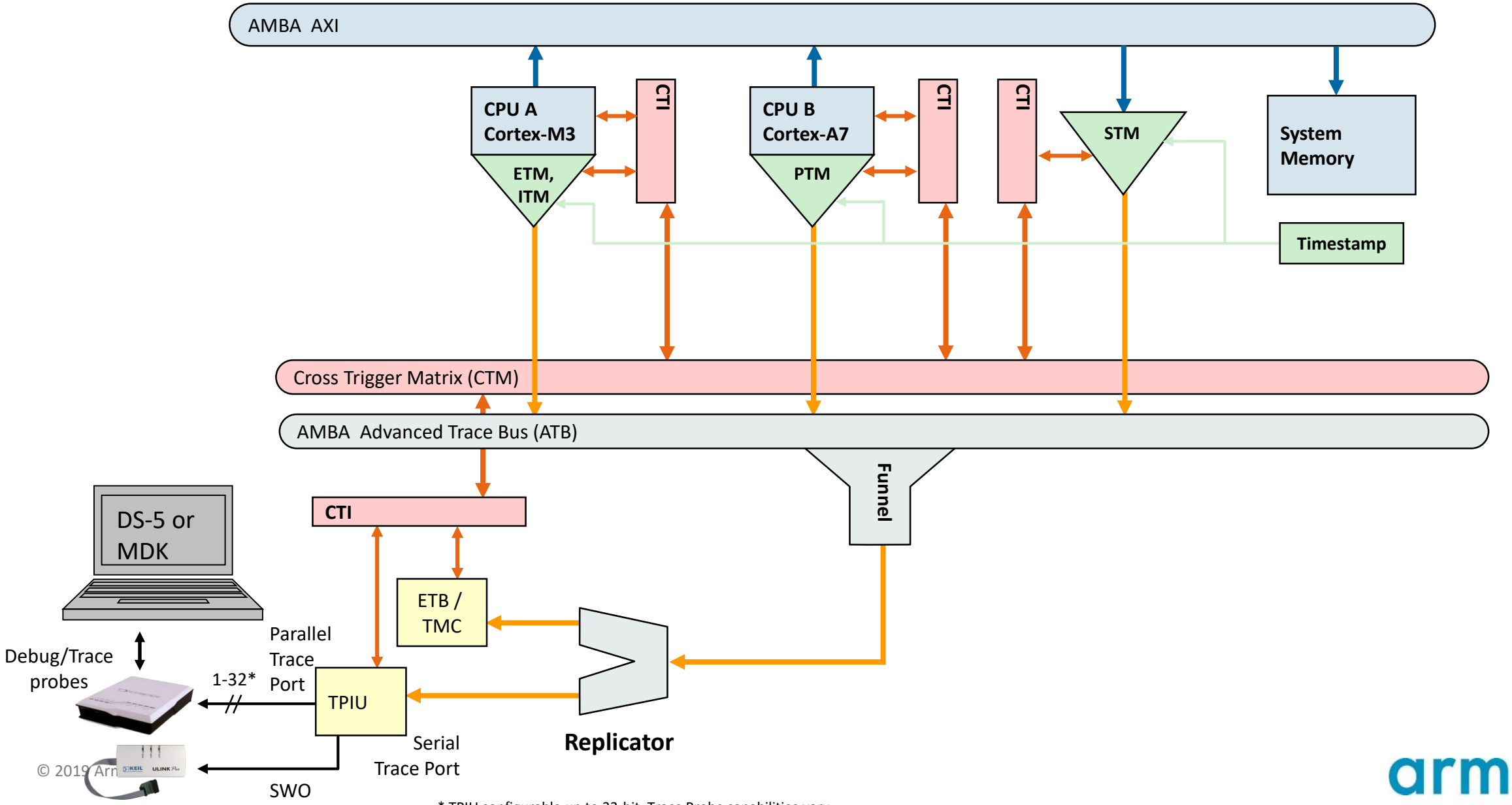
arm

Example Debug/Trace Subsystem using CoreSight SoC-400



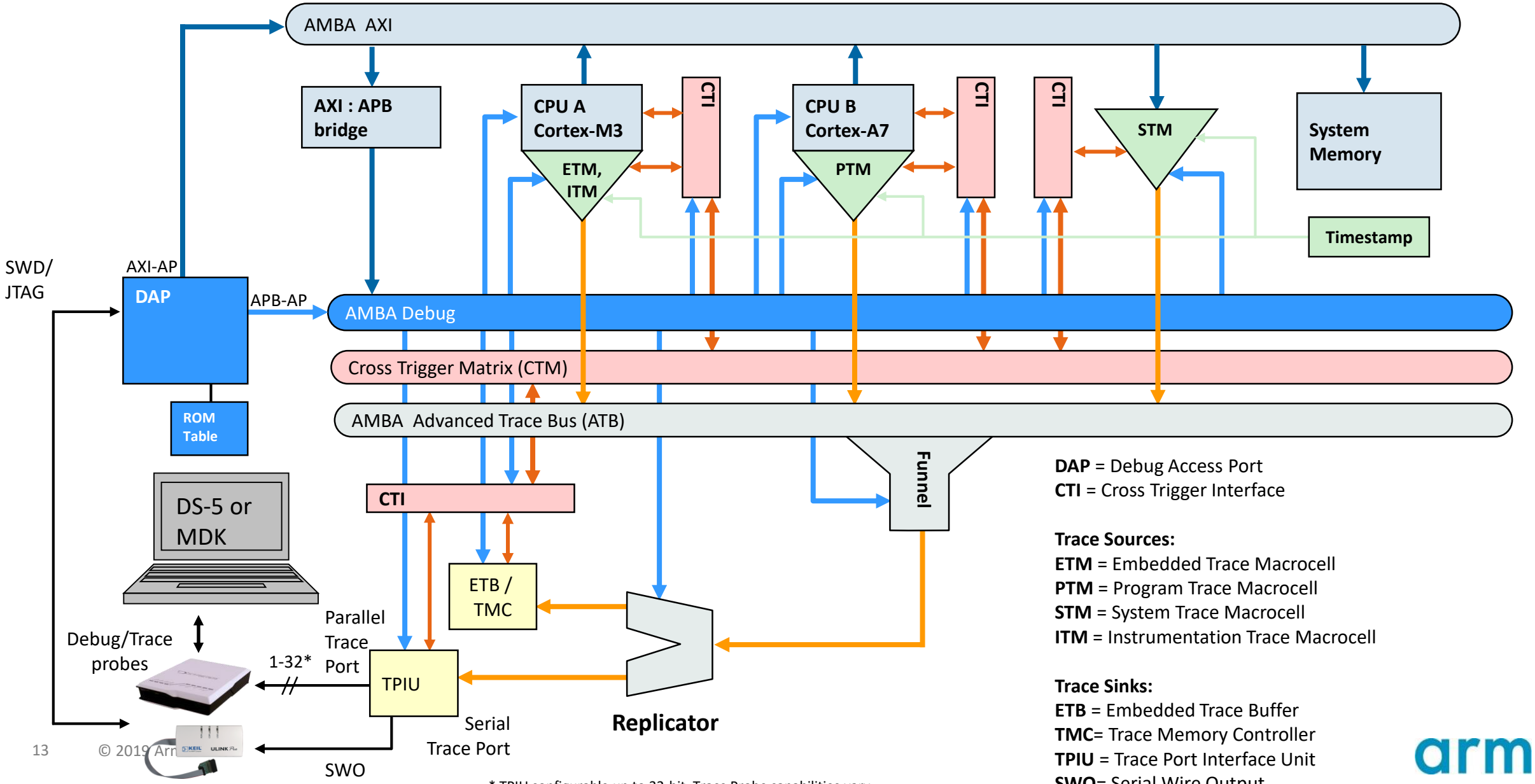
* TPIU configurable up to 32-bit. Trace Probe capabilities vary

Example Debug/Trace Subsystem using CoreSight SoC-400



* TPIU configurable up to 32-bit. Trace Probe capabilities vary

Example Debug/Trace Subsystem using CoreSight SoC-400



DAP = Debug Access Port
CTI = Cross Trigger Interface

Trace Sources:
ETM = Embedded Trace Macrocell
PTM = Program Trace Macrocell
STM = System Trace Macrocell
ITM = Instrumentation Trace Macrocell

Trace Sinks:
ETB = Embedded Trace Buffer
TMC= Trace Memory Controller
TPIU = Trace Port Interface Unit
SWO= Serial Wire Output

* TPIU configurable up to 32-bit. Trace Probe capabilities vary

Further Information

- Introduction to ARM CoreSight SoC-400 video
- CoreSight SoC-400 Whitepaper
- CoreSight SoC-400 TRM
- Arm CoreSight webpage

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה