WHITE PAPER

CoreSight Technical Introduction

A quickstart for designers

August 2013

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.
The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means "ARM or any of its subsidiaries as appropriate".

Confidentiality Status
This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Contents

About CoreSight

CoreSight technology from ARM provides solutions for Debug and Trace of complex SoC designs. This document introduces the concepts which will help you to work with CoreSight. CoreSight licensees should refer to the CoreSight System Designers Guide for more in-depth information.

Debug : This refers to features to observe or modify the state of parts of the design. Features used for debug include the ability to read and modify register values of processors and peripherals. Debug also includes the use of complex triggering and monitoring resources. Debug frequently involves halting execution once a failure has been observed, and collecting state information retrospectively to investigate the problem.

Trace: CoreSight provides features which allow for continuous collection of system information for later off-line analysis. Execution trace generation macrocells exist for use with processors, software can be instrumented with dedicated trace generation, and some peripherals can generate performance monitoring trace streams.

Trace and Debug are used together at all stages in the design flow from initial platform bringup, through software development and optimization, and even to in-field debug or failure analysis.

Historically, the following methods of debugging an ARM processor based SoC exist:

Conventional JTAG debug ('external' debug)

> This is invasive debug with the processor halted using:
>
> • Breakpoints and watchpoints to halt the processor on specific activity.
>
> • A debug connection to examine and modify registers and memory, and provide single-step execution.

Conventional monitor debug ('self-hosted' debug)

> This is invasive debug with the processor running using a debug monitor that resides in memory.

Trace

> This is non-invasive debug with the processor running at full speed using:
>
> • A collection of information on instruction execution and data transfers.
>
> • Delivery off-chip in real-time, or capture in on-chip memory.
>
> • Tools to merge data with source code on a development workstation for future analysis.

CoreSight technology addresses the requirement for a multi-processor debug and trace solution with high bandwidth for entire systems beyond the processor, despite ever increasing SoC complexity and clock speeds. Efficient use of pins made available for debug is crucial.

CoreSight provides:

• A library of modular components and interconnects.

• Architected discovery and identification methods to allow for flexible system design and easy inclusion of differentiated debug/trace functions.

• A standard implementation of the ARM Debug Interface for debug tools to work with.

Elements of a CoreSight design

The CoreSight architecture introduces a number of key concepts which together enable complex systems to be designed. Standardized programming models and feature discovery registers allow debug tools to be largely generic with minimal dependence on the feature set of an individual SoC.

Debug Access Port

The Debug Access Port (DAP) is present on any SoC which presents a physical port to be connected to external debug tools. The DAP is an implementation of the standardized ARM Debug Interface, and provides a bridge between a reliable low pin count interface and on-chip memory mapped peripherals. See page 7 for more details of the DAP. Transactions generated by the DAP are referred to as External Debugger Accesses.
The DAP provides (amongst other things) architected top level control for debug domain power control, and fast code download direct to system memory.
CoreSight components implement memory mapped interfaces, but the DAP can also act as a bridge to an on-chip JTAG scan chain where necessary for legacy components. This gives increased flexibility and power savings when working with multiple clock and power domains on the SoC.

Self Hosted Debug

Most processors have direct access to their own debug resources by using dedicated instructions. In addition, it is common for most processors on a SoC to have access to some or all of the remaining debug components. Exact details vary, but there is typically a region in the system memory map which is multiplexed with external accesses to the debug components. Self hosted debug is typically managed by debug monitor software running on either the target processor or a second processor in the SoC.
Access control mechanisms are provided to permit interworking between an external debugger and self-hosted debug such that the external debugger does not need to be aware of the actions of the debug monitor.
Save and Restore sequences can be used by on-chip software to maintain the debug state across power-down cycles, and provide the illusion to the external debugger that the SoC remains powered on. This is particularly important for debug of battery powered devices where infrequent events are being monitored.

Discovery using ROM Tables

All CoreSight systems will include at least one ROM table. This serves the purpose of both uniquely identifying the SoC to an external debugger, and allowing discovery of all of the debug components in a system. Discovery relies on the use of identification registers at architected positions in the memory map of every debug component. All CoreSight components use this standard. This permits discovery sequences of identify at least a sub-set of the feature-set without detailed knowledge of every component. For both external debug, and self-hosted debug, there is a pointer to the address of the top-level ROM table from that debug agent. The ROM table provides a list of address offsets which can be used to locate the next level of component. Components can be ROM tables again, or individual components. Provided the system complies with the rule that each component is only referenced once in the ROM tables and there are no loops, it is possible to identify all the debug components which are accessible to each debug agent.

Processor debug and monitoring features

The exact features vary between processor design, and can also vary from one implementation of a processor to another. Processors typically provide a halting debug mode (where architectural state can be observed) and single step execution. Also common are breakpoint units and Performance Monitoring Units (PMU). CoreSight provides an Embedded Cross Trigger mechanism to synchronize or distribute debug requests and profiling information across the SoC.

Cross Triggering

CoreSight Embedded Cross Trigger (ECT) functionality provides modules for connecting and routing arbitrary signals for use by debug tools. Wherever there are signals to sample or drive, a Cross Trigger Interface (CTI) is used to control the selection of which signals are of interest. Most systems will implement a CTI per processor, and at least one CTI for system level components. The CTIs in the system are interconnected using a Cross Trigger Matrix (CTM) which distributes any selected input events across the SoC to every CTI. Each CTI is programmed to use these distributed events to drive local control signals.

For processors and ETM trace units, the event connections to the CTI are standardized (although this does vary from processor to processor, as described in the processor documentation). Typical connections are listed below.

Source	Destination	Example use case
Trace logic External Outputs (4 bits)	CTI Trigger inputs	Trace logic resources to trigger trace capture or debug
Trace logic External Outputs (2 bits)	PMU inputs	PMU counters to extend trace logic counters
PMU Events (~30 bits)	Trace logic External inputs	Filter trace based on processor events such as cache miss
PMU overflow	CTI Trigger inputs	Forward PMU counter overflow to interrupt controller or other clusters
Processor Debug Restart	CTI Trigger input	Synchronized debug restart across clusters (supporting halt and restart)
Trace Buffer Full	CTI Trigger input	Halt processor on trace buffer full
CTI Trigger Output	Processor interrupt input	Cause interrupt based on input to CTI or other CTI in system
CTI Trigger Output	Processor Debug Halt Request	Enter debug state based on input to CTI or other CTI in system
CTI Trigger Output	Trace Port Trigger request	Indicate trace trigger to trace capture device

Table 1 - Cross Trigger Connections

Trace Sources

CoreSight technology provides a standard infrastructure for the transmission and capture of trace data (presented as arbitrary streams of bytes). This allows for optimum sharing of common resources. Various trace sources are available:

Processor Trace Units

Processor debug is implemented by Embedded Trace Macrocells (ETM trace unit) or Program Trace Macrocells (PTM trace unit) depending on the target processor. Each ETM trace unit or PTM trace unit is specific to the processor it is designed for.

The feature set varies depending on the use cases anticipated for the different processors, but all CoreSight ETM and PTM trace units which use an AMBA Trace Bus (ATB) output can be combined in a system. Trace units might support the following:

- Processor execution trace in varying degrees of detail
- Resource logic, often useful as an extension to processor performance monitoring resources
- Filtering logic to reduce the amount of non-interesting data which is captured

A common feature of trace units is efficient compression and encoding, relying on a copy of the executed code for decompression. Using halting debug, it is possible to extract the code image from program memory.

Instrumentation Trace Units

The instrumentation trace and system trace units provide the ability for running software to be instrumented with messaging (either by the programmer, or through a tool flow). This is more intrusive than using processor trace, but provides information at a higher level. The instrumentation trace macrocells are typically mapped into system memory. Tightly coupled Instrumentation Trace Macrocells (ITM) exist for some processors, the System Trace Macrocell (STM) is a more generic version which can be used in any system.

Trace (ATB) interconnect

One advantage of using a standard trace bus protocol is that a small set of modular components can be used to generate sophisticated trace infrastructure. These components include bridges for timing closure, clock and power domain crossing, replicators and funnels which can be used to combine data streams, and buffer components. Upsizers and downsizers are used to convert busses of varying data width. A key feature of the AMBA Trace Bus (ATB) is that the trace source identification is passed with the data, permitting cycle by cycle interleaving of trace data from different sources.

CoreSight trace interconnects provide the following features:

- Backpressure to stall a trace source based on the ability of downstream infrastructure to collect data
- Flushing of any data stored in intermediate buffer components through the interconnect
- Transfer of byte orientated data, agnostic to the underlying data protocol
- Synchronisation request distribution

Trace Sinks

A trace sink is the final CoreSight component in a trace interconnect. A system can have more than one trace sink, configured to collect overlapping or distinct sets of trace data. Trace sinks can stream data off chip, provide a dedicated buffer, or route trace data into shared system memory. These different solutions cover a wide range of latency and bandwidth capabilities.

Processor Trace Architectures

The ETM and PTM trace units are trace sources that monitor ARM processors. Each ETM trace unit and PTM trace unit is associated with certain processor lines, and each ETM and PTM implementation conforms to certain ETM and PTM architectures. The architecture consists of a generic programmers model and a trace protocol.

ETMv1, ETMv2

> The earliest ETM architectures, representing internal processor pipeline status in a cycle by cycle basis. No longer in common use.

ETMv3

> Major revision to earlier protocols, implementing a byte-based packet protocol and the first ETM protocol to support CoreSight. Supports instruction by instruction execution and data transfer trace, depending on the processor.

PFTv1

> Derived from ETMv3, providing only trace of branch execution and exceptions. Supported by Cortex-A9, Cortex-A12 and Cortex-A15

ETMv4

> A major revision of the earlier protocols, supporting advanced processor architectures. Includes the instruction execution trace style of PFTv1, and optionally ETMv3 style data trace capabilities. Supported by Cortex-R7, Cortex-A53 and Cortex-A57.

Within a CoreSight system, any processor trace units supporting ETMv3, PFTv1 or ETMv4 architectures can operate in combination.
Most processor trace units provide a single ATB output bus (either 8 bit for the Cortex-M variants, or 32 bit). This carries both instruction trace, and data trace if supported. Some R-class processor trace units are unusual in providing a 32 bit ATB interface for instruction trace and a 64 bit ATB interface for data trace. This reflects the high cost of implementing data trace for a high performance processor, and also the need within some real-time application segments to support high-quality data trace capture.

Debug access and DAP topology

Traditional SoC debug used a JTAG interface to connect to a TAP controller in the processor. Where multiple processors are present, the JTAG scan chain would cascade the TAP controller of each processor, possibly through multiple clock and power domains.
Access to system memory would be achieved by halting the processor and downloading instructions while halted to cause the processor to perform the necessary memory accesses.
The DAP introduced by the CoreSight architecture moves the primary point of connection away from the individual processor, and implements a bridge between the external protocol and various different on-chip protocols. This provides a flexible and scalable solution where this bridge point can remain powered and responsive irrespective of the activity of individual processors.

Figure 1 shows a view of the components which are visible in the debug memory mapped space with their discovery registers. Registers provide identification and address offset details. Remember that the DAP will be multiplexed with accesses from the main system interconnect too.
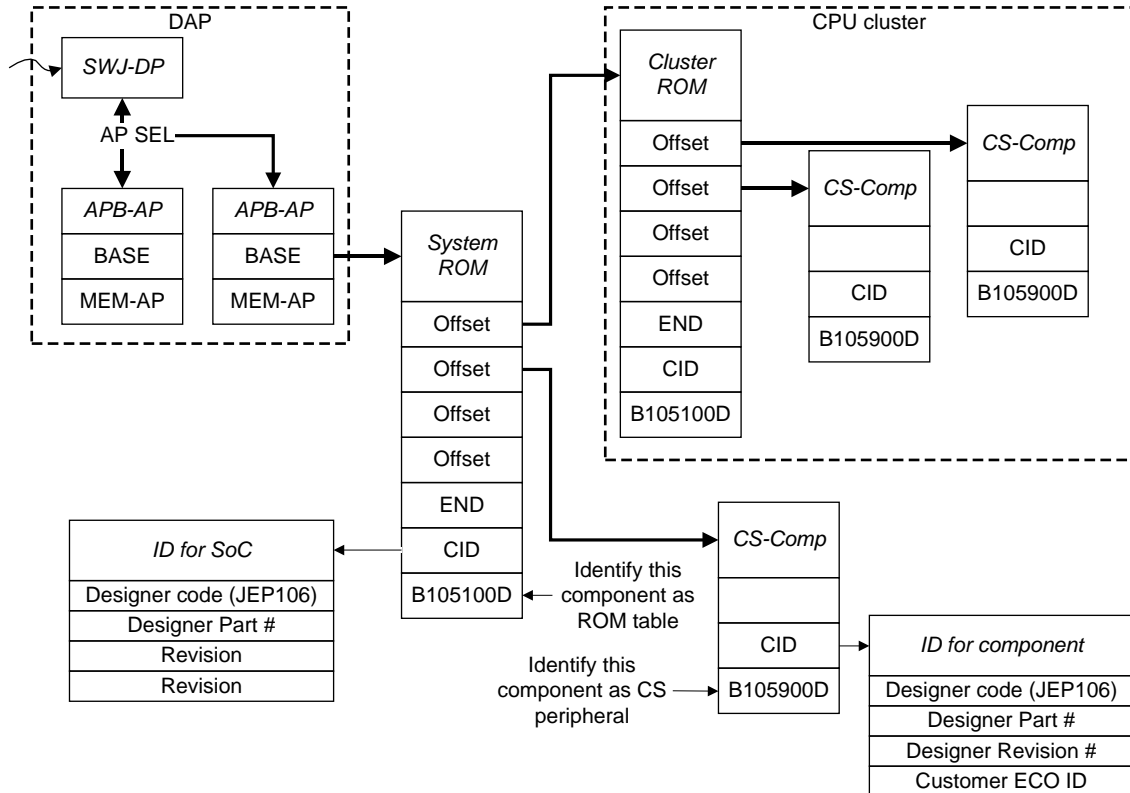
Figure 1 – Debug address map discoverability

Debug Port

Every DAP requires a Debug Port (DP). This is the master device, and implements the external interface. Debug ports supporting both JTAG and optimized 2-pin Serial Wire interface can be licensed from ARM. The debug port provides:

- always-on connection for the debugger
- debug fault and status reporting
- power and reset request interface

Debug port accesses from the external debugger are performed as 32 bit (word) read or write transactions, targeting either DP registers, or Access Port (AP) registers.
Multiple Debug ports (usually in multiple packages) can be addressed from a single external debug agent using:

- daisy chained JTAG scan chain
- star topology JTAG scan chain
- multi-drop serial wire

Access Port

Each DAP contains between 1 and 256 Access Ports (APs). The APs are controlled by the DP in response to external commands. Most APs implement a master port which interfaces to an on-chip standard bus interface. Memory APs exist for memory-mapped interfaces such as APB, AHB and AXI

interconnects. A JTAG-AP can be used to interface the DAP to a traditional JTAG TAP controller. Customized access ports can also provide a simple interface to dedicated chip-level debug logic.

Memory APs provide the following features:

- Target address register
- Read or write to target address
- Bus error reporting
- Transaction in progress status
- Address incrementor (to accelerate block read/write operations)
- Access control mechanisms
- Information about connected debug components
- Perform access appearing as system master, or external debug agent.

DAP address space

Any individual memory mapped address in system memory might require several accesses to enable the correct path, and requires more than simply the target address in the on-chip memory map:

- **DP Identifier**: The debug agent might support concurrent access to more than one DAP.
- **AP Select:** The target AP must be selected by writing to a register in the DP.
- **TAR Select:** The target address must be set by writing to a register in the AP. Each AP can have a unique view of some or all of the memory mapped components in the target system.
- **Data Access**: Once all the addresses necessary for a DAP access to the system are set, a request to the AP can initiate the on-chip access as either a read or a write.
- **Read Data retrieval:** Although the on-chip access will now proceed, the debugger must perform another access to the DAP in order to retrieve the data value. This need not result in a second on-chip access.

When an access fails for some reason, the debugger is able to identify the failure. Usually the debugger can re-try the access and recover from simple errors on the interface.

Debug memory map views

Both externally hosted debug agents and on-chip debug agents (for example a debug monitor) require access to debug components. Within CoreSight, these debug components are provided on a dedicated bus, the debug APB. This ensures a clear separation between system memory space and debug memory space. An exception is the Cortex-M processors where a shared AHB interconnect supports both system memory and debug access as an area-reduction trade-off.

An on-chip agent must first navigate the system memory bus before being multiplexed with the DAP initiated transactions on the Debug APB. This provides two memory mapped views, one from the external debugger and one from the on-chip agent. Both views share access to the debug components using the same address offsets within the mapped regions. The system view of the debug APB will typically have a non-zero base address whilst the external debugger view uses a base address of zero.

The upper address bit (PADDRDBG31) is only accessible from the external debugger and serves as an access control mechanism.

Debug memory discovery and ROM table entries

Every CoreSight component with an APB memory map occupies one or more 4kB blocks of memory. Within this block, CoreSight defines the content of some discovery registers. See the TRM for each individual component for specific details. The discovery pointer structure is shown in Figure 1 on page 8, some examples of the individual registers are shown in Table 2 on page 10.

Name/Offset	Example Values	Description
DEVTYPE 0xFCC	0x00000016: Processor Performance monitor 0x00000013: Processor Trace unit	Only used by CoreSight debug components. Can classify unknown 'new' components
PID4 0xFD0	0x04 : 4kB component, ARM	Size of address block, and part of designer ID
PID3,PID2,PID1,PID0 0xFE0-0xFEC	0x004BB906 : ARM CTI rev4 0x003BB912 : ARM TPIU rev 3	Unique part identifier consisting of Designer (via JEP106 code) 3 digit part allocated by designer Part revision Part ECO identifier Part modified
CID3,CID2,CID1,CID0 0xFF0-0xFFC	0xB105900D : CoreSight Debug 0xB105100D : CoreSight ROM Table	Component identifier, indicates if the CoreSight layout is used. Other values might be used by ARM PrimeCells and other components.

Table 2 – Example CoreSight discovery registers

At least one ROM table component must be present as a slave to any AP which contains debug components. This will be the APB-AP, or AHB-AP in the case of a Cortex-M system. Each ROM table contains a list of address offsets which can be used to locate component base addresses. These components can themselves be ROM tables, but each physical component or ROM table must appear only once in the expanded list of pointers.

The AP contains a base address register which must point to the master ROM table for that bus. Typically, this will occupy the lowest 4k block of the address space. The ROM table is a CoreSight component, and contains standardized identification registers. It also contains an identifier for the SoC as a whole which can be used by debug agents to look-up against a database of known devices. This lookup can provide information about SoC specific features.

Typically the ROM table hierarchy will match the design hierarchy of modules containing debug APB. In this way, larger systems can be constructed from sub-systems and clusters. As a result, the debug APB is often sparsely populated.

Typical CoreSight systems

The systems shown here demonstrate the most basic configurations of a CoreSight system. More complex systems might involve clusters of processors, multiple clock domains, etc.

Single processor debug

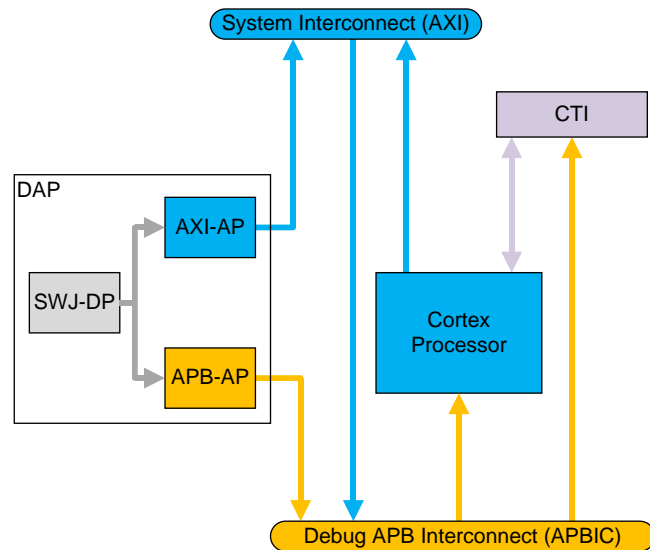Figure 2 shows CoreSight debug in a single processor system.

Figure 2 Single processor with Debug APB access

This configuration provides no trace capabilities. The DAP shown here is configured with a combined Serial Wire and JTAG external interface, and APB internal debug access. The Debug APB connects using an APB-Interconnect to configure the CTI and access the processor. The CTI supports triggering of the processor from a designated resource, and enables connection to additional triggering resources if this example is integrated into a larger system.

Single source trace

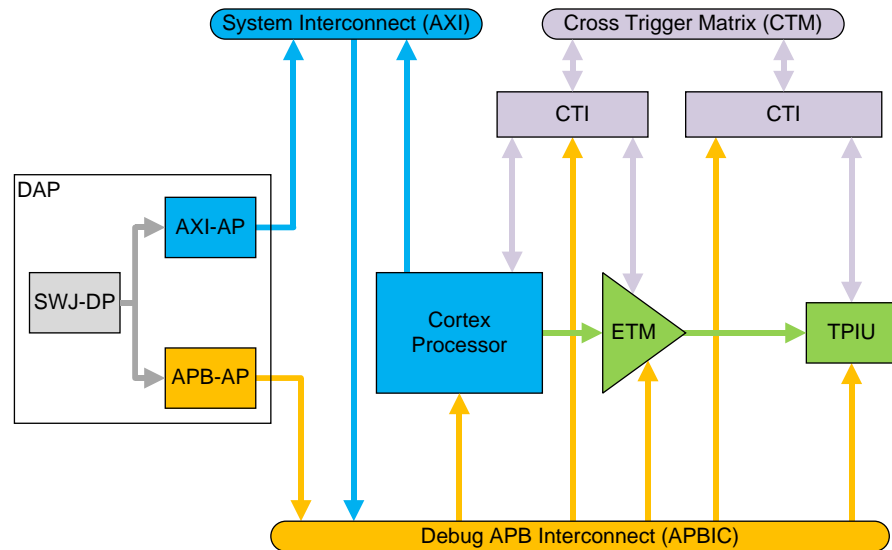Figure 3 shows a single processor trace using the CoreSight infrastructure.

Figure 3 - Single source trace with the TPIU

The CoreSight-compliant ETM trace unit outputs trace directly to a TPIU for direct output of trace off-chip. You can extend this system to add a CoreSight ETB and replicator to provide on-chip storage of trace data.

Multi source trace in a single processor system

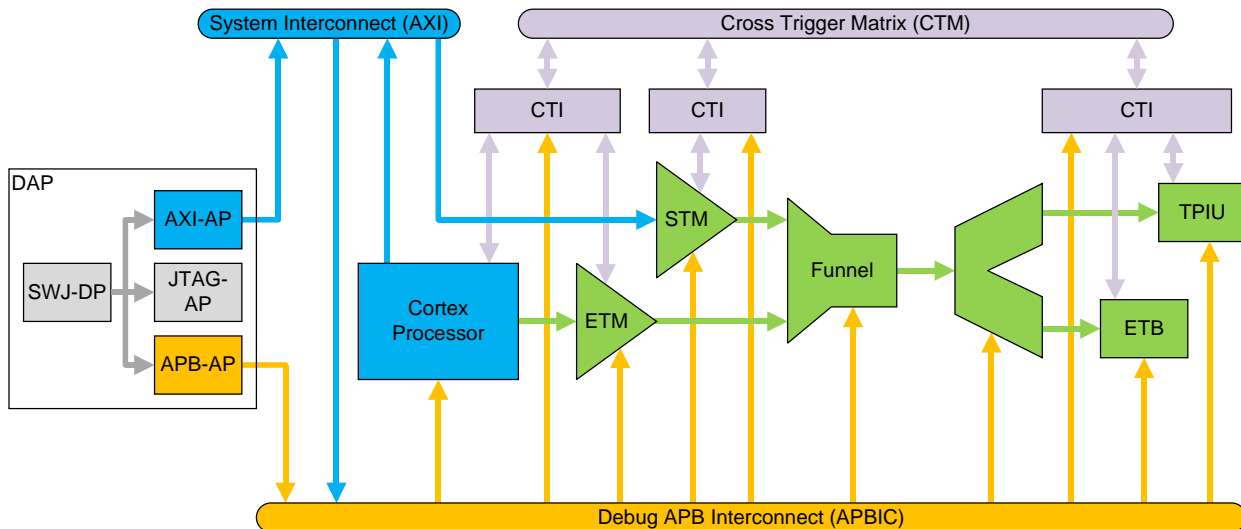Figure 4 shows full trace capabilities in a single processor system.

Figure 4 - Full CoreSight trace with single processor

The ETM trace unit provides processor instruction and data tracing, and the STM provides instrumentation trace. The trace funnel combines trace from all sources into a single trace stream. This is then either:

- Replicated to provide on-chip storage using the CoreSight ETB (limited capacity)
- Output off chip using the TPIU (limited bandwidth)

You can program components using the DAP and operate cross-triggering using the CTM and CTIs.

When multiple trace sources are active in the system, each source must be configured with a unique trace source ID, and every trace sink must have trace formatting enabled. One function of the trace formatter is to embed the trace IDs in the final data stream. When only one trace source is active, the trace sink can be used in bypass mode which can be more efficient in some scenarios.

System topology restrictions

The CoreSight architecture includes some rules which restrict the system topology. These rules allow for system-agnostic debug tool design and topology detection. Violating the topology rules might also result in deadlock or livelock conditions.
Some rules relate to the debug memory map, which is limited to any path from external interface to peripheral only crossing 3 levels of protocol addressing (external interface, subset of debug interconnect, address within interconnect) and this addressing not having any replication or aliasing. Restrictions on the trace bus require no duplication or re-use of any trace ID which reaches any other trace component, or feed any trace source back in a feedback loop.

Trace capture

The trace that CoreSight trace sources generate must be captured by one or more *Trace Capture Devices* (TCDs). The following common forms of TCD exist:

- On-chip trace buffer.
- Off-chip logic analyzer.
- Off-chip dedicated Trace Port Analyzer

Logic analyzers are expensive and are less well supported by development tools, but can often capture trace at higher speeds than is possible with a *Trace Port Analyzer* (TPA). Most developers capture trace using a TPA or on-chip trace buffer.

The CoreSight ETB and Embedded Trace Router (ETR) are ATB slaves and connect to the CoreSight system directly to enable capture of trace data on-chip. A TPA, or logic analyzer, must connect to the pins of a trace port that a TPIU drives.

Many systems implement either one ETB or one TPIU. However, it is possible to implement multiple trace sink components using a CoreSight Replicator.

Figure 5 on page 14 shows a system that implements an ETB and a TPIU connected to a TPA.
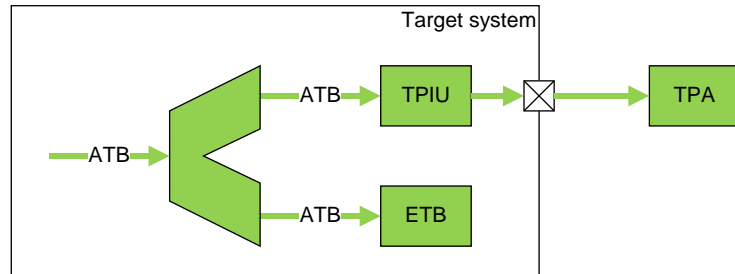
Figure 5 - Example system with ETB and TPIU

5.1.1 Operation of a TCD

A TCD has a large circular buffer at its center. Trace is written into this buffer as it is generated. Trace capture does not stop when the buffer becomes full, but instead overwrites old trace.

A TCD is sensitive to two special signals, that the ETB or TPIU generate:
- Trigger.
- Trace disabled.

A TPIU indicates these signals to a TCD as follows:

- Using the optional **TRACECTL** top level pin. This is the easiest way for a TCD to detect this information, but requires a dedicated pin when trace is in use.

- Using the CoreSight formatter protocol. This requires a TCD that can extract this information from the formatter protocol, and results in a trace port that is one pin smaller. There is a protocol overhead cost (at least 6%), but this is offset by freeing up one more pin. The formatting protocol also permits the use of more than one enabled trace source at a time.

Trigger

The trigger is an input to the trace sink, and an output from a CTI. If there is more than one trace sink, each can receive a different condition as its trigger. Most trace sources, for example an ETM trace unit or *AHB Trace Macrocell* (HTM), can output a signal to use as a trigger. Usually, the CTIs are configured to send a trigger to all trace sinks when any trace source signals its trigger condition.

When a trigger is detected, the TCD counts a programmable number of trace records before it stops trace capture. After this point, it ignores any more trace. By setting the appropriate number of programmable trace records, you can select a window of trace to capture around the trigger condition. Figure 6 shows this context.
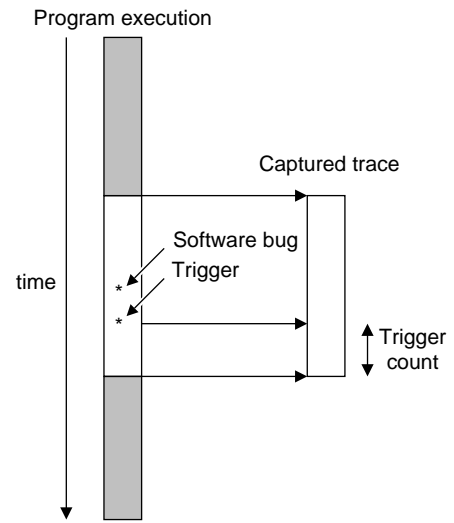
Figure 6 - Use of the trigger to set a trace window

You can configure the trigger to output when the system detects a bug. The window of trace indicates the behavior of the system before and after the bug occurred.

You can use the trigger count in the following ways:

*   Set the trigger count to a small value. This gives a window of trace mostly before the trigger occurred, capturing the software bug under investigation.

*   Set the trigger count to a value slightly smaller than the size of the buffer. This gives a window of trace mostly after the trigger occurred.

*   Set the trigger count to roughly half the size of the buffer. This gives a window of trace before and after the trigger occurred.

When trace capture has stopped, the development tools download the trace from the TCD.

Trace disabled

Trace disabled indicates to the TCD that there is no trace to capture. It ensures that the values of the trace port pins are only captured when trace data is available. The formatting protocol can also indicate that there is no data to be captured by using a specific sequence, but again this requires on the TCD being able to perform some analysis of the stream before it is captured.

Streaming trace capture

Usually, the ETB, ETR, or TPIU wait until there is sufficient trace to use all the pins of the trace port before any trace is captured in the on-chip memory or output over the trace port. For example, if only one byte of trace is available in a system that implements a 16-bit trace port, no trace is output until a second byte of trace is available. In addition, when the formatting protocol is in use, a full block of 16 bytes must be captured before the data can be fully decompressed. This complicates the task of designing a trace capture system where data must be continuously streamed and analyzed in near real time. Different approaches to this problem can be used depending on the system requirements, and are unlikely to detract from the user experience when streaming trace is expected.

Trace Capture capacity

A trace capture system is likely to be one of the limiting factors determining how much trace can be generated. The resources dedicated to trace capture are likely to be limited, and it is important to ensure that the typical use-cases can be supported with a low enough level of data loss. Although CoreSight is designed with graceful degradation in the case that more trace is generated than can be captured, this should not be relied on. Careful use of filtering will result in more useful trace being captured than relying too much on the overflow/recovery behavior.

The demands of a trace source can vary greatly, an ETM trace unit might produce between 1 bit per instruction for instruction only trace, or over 30 bits per instruction when tracing instructions and data. Even if the data to be traced can be filtered, this might not help much for short-term bursts of data so an on chip trace FIFO can help. For more complex trace systems, this becomes more of a cost-effective solution as the resource added is shared between more of the trace logic. The user can select which trace source needs most bandwidth, but still enable a smaller amount of trace from several other sources, or use the other sources as triggering resources.

Trace Synchronization

Most trace sources use complex protocols which rely not only on identifying the correct packet boundaries in the protocol, but also initializing the various decompression schemes. When the trace capture formatter protocol is in use (as is necessary for simultaneous capture from more than one source), the formatter protocol requires synchronization too.

A TPA will typically capture trace into a circular buffer. This means that if capture is stopped once the buffer has wrapped round, some early trace will have been lost. In order to decompress the trace stream, the tools must search the buffer until a synchronization point can be detected. Any trace which was captured but is before the synchronization point must be discarded (usually the synchronization cannot be extended backwards). Since it is inefficient to synchronize each trace stream too frequently, most trace sources allow for software programming of the synchronization points.

Depending on the quantity of trace being captured, it might be necessary to change the synchronization period. When capturing into a small buffer, more frequent synchronization results in a higher proportion of the captured trace being usable (but more use of the buffer for non-useful trace)

In systems where several trace sources are active together, the synchronization of each source is independent. Some trace sources support the use of a distributed synchronization request to be generated from the TCD. This ensures that all trace sources initiate their synchronization sequences at the same time.

Timestamps

Many trace sources can embed global (SoC level) timestamps in their trace stream. These can be used to correlate activity between different traces sources, particularly when the trace data might be captured in different TPAs, or subject to delays as a result of protocol or buffering.

Timestamps are typically a 64 bit count, derived from an always on domain with a frequency of at least 10 MHz. The timestamp distribution mechanism uses a narrow bus to distribute this count value, and an interpolation mechanism to generate corresponding count values at higher resolutions where the count needs to be used. This provides a trade-off where the ordering between events in a well designed system can be determined, at least to the accuracy of any communication between the CPUs originating the events. Timestamps can also be used for performance measurement, as an alternative to the more precise but more bandwidth intensive cycle counts which some trace sources can insert.