WHITE PAPER

Multi-core microcontroller design with Cortex-M processors and CoreSight SoC

Joseph Yiu, ARM
Ian Johnson, ARM

January 2013

Abstract:

While the majority of Cortex™-M processor-based microcontrollers are single core designs, some new microcontrollers with multiple Cortex-M processors are also available. As multi-core designs are increasing in popularity and complexity, the methodologies of designing multi-core Cortex-M systems are becoming more critical. This paper introduces typical system level designs for multi-core Cortex-M microcontrollers and some of the various factors that need to be considered when designing the memory system, together with low power support and additional hardware to allow multi-core systems to work effectively. This paper also covers example debug subsystems for multi-core designs, the features typically needed in multiple core debug systems, such as debug event communication; and the way in which the Processor Integration Layer in ARM' CoreSight™ SoC debug and trace technology supports the required debug functionality.

1 Introduction

Traditional multi-core designs focus on application processors running a full-feature OS such as Linux, iOS or Windows. Recent developments mean there are now more microcontrollers available on the market with multiple processors. For example, the NXP LPC4300 contains an ARM Cortex-M4 and a Cortex-M0 processor, and the Freescale Vybrid contains a Cortex-M4 and a Cortex-A5 processor. This illustrates that there is a need for multi-processor designs in certain microcontroller application areas.

While Cortex-M processors have a number of features that allow the design to work in multi-core systems, the design process can be challenging. A key factor is that there is no "typical" system level design or reference platform as there is for application processors running an OS with SMP (Symmetric Multi Processing) support. In the microcontroller world different applications have different system level requirements, and as a result each system level design can look very different.

The key challenges of developing multi-core Cortex-M microcontroller products include three areas:

- Memory and bus system design
- Low power strategy

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

- Debug and trace subsystem

This paper focuses on hardware features of multi-core design. There are, of course, also some other aspects particularly debug tools and software: for example, how to visualize and make it easier for programmers to understand the interactions between various program codes running on different processors, and identify race conditions.

2 Memory and bus system design

2.1 ROM

One of the first things to consider when designing the memory and peripheral bus systems is which parts of the system will be shared and which parts of the system are not to be shared. If none of the resources are to be shared, then there is no contention issue and each processor can run at full speed (or as fast as the memory system allows). However, in many cases some of the memories and peripherals will be shared.

Since the existing Cortex-M processors don't include a cache memory controller, the processors need to access the program memory fairly frequently. If the program memory is shared, the memory system will have to arbitrate and prioritize the accesses (figure 1). However, the situation is not as bad as it may first seem. This is because:

- The processor does not fetch instructions every clock cycle. In many applications, most of the instructions executed by the processor are 16-bit instructions; and so on each instruction fetch (32-bit) up to 2 instructions can be obtained. As a result, almost half of the time the processor bus interface is idle.

- The processor has a small instruction buffer. For Cortex-M3 and Cortex-M4 processors, the instruction buffer is three words. So a small delay in the instruction fetch does not cause too much of a performance penalty.

For a dual-core design, the accesses to the shared program memory from both processors could end up interleaving each other most of the time and therefore the memory sharing might only have a small impact on the performance:

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

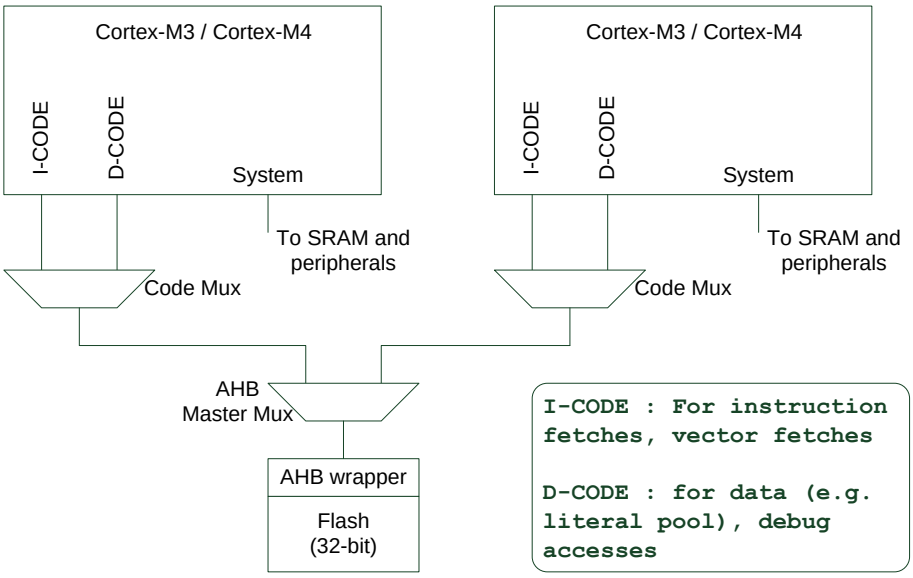


Figure 1: Simple program memory sharing between two Cortex-M series processors

With a simple program memory sharing arrangement as shown in figure 1, the performance of the two processors each running Dhrystone 2.1 is about 78% of the full performance (without program memory sharing). This is tested with running 200 iterations of Dhrystone 2.1 compiled with ARM DS-5™ version 5.12, in Verilog RTL simulation.

To improve the performance further, we can modify the bus arrangement to give literal data accesses (from D-CODE) higher priority (figure 2). Since the processor might already have buffered up a couple of instructions in the instruction buffer, the stalling of instruction fetch does not always stall the program execution, whereas stalling of literal data access does.

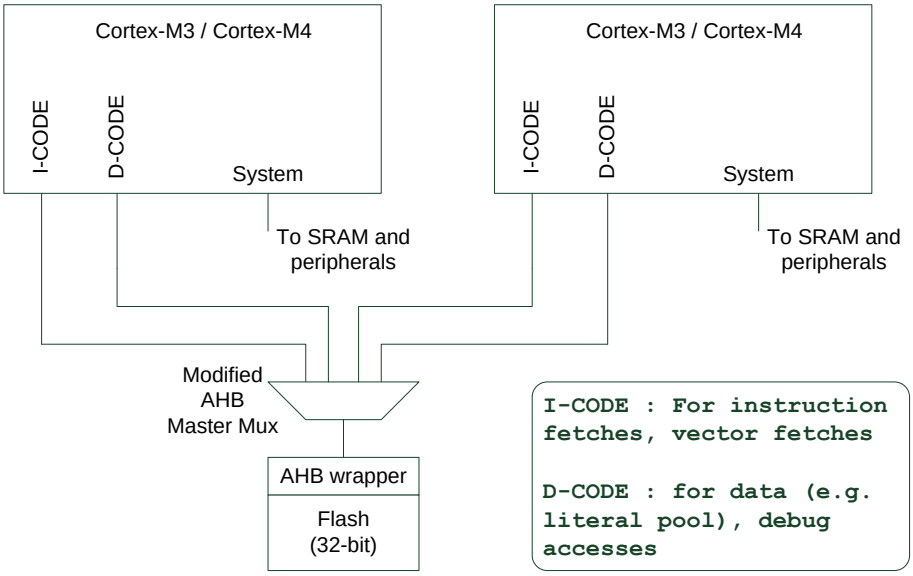


Figure 2: Modified program memory sharing to give literal data read higher priority

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

With this arrangement, the performance of the two processors simultaneously running Dhrystone is about 78.9% of the full performance, slightly better than the previous setup. The improvement depends on how often literal data in flash/ROM are accessed and how often the literal accesses happen simultaneously.

We can also improve the performance by converting the program memory to 64-bit or even 128-bit and adding a small prefetch buffer that fetches the instructions with 64-bits or 128-bits accesses (figure 3). By doing this, the number of instruction fetches going to the program memory is reduced and this can reduce the amount of memory access conflicts.

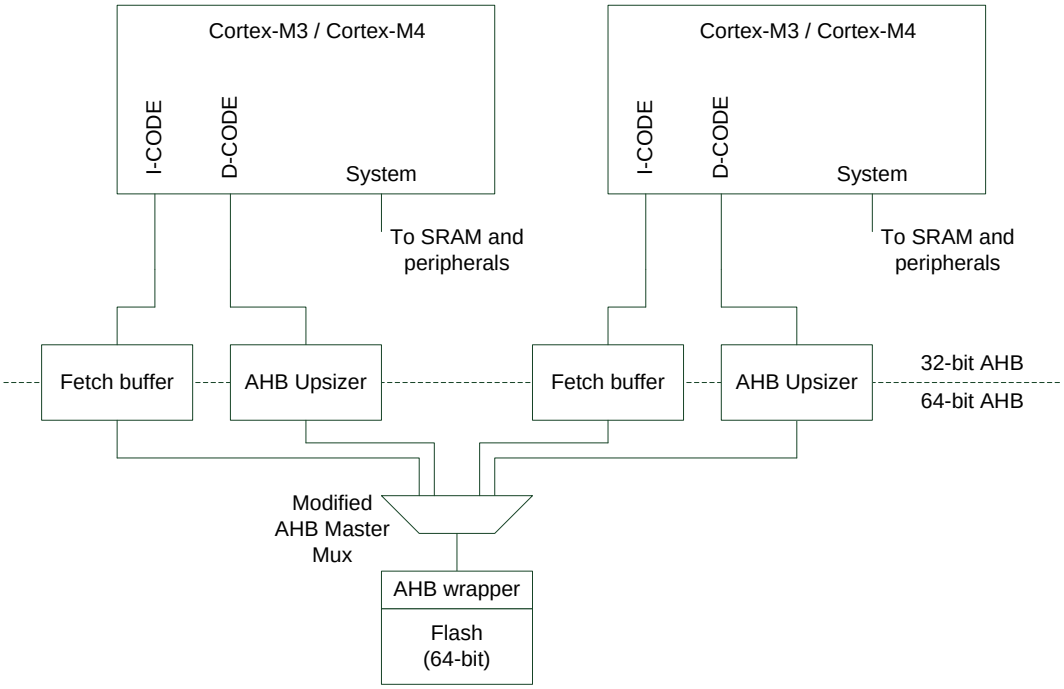


Figure 3: Use of instruction fetch buffers to reduce access conflicts

With the design as shown in figure 3 using 64-bit program memory, the performance of the two processors is 92% of the theoretical maximum performance.

If more processors are added, there are various choices

- Having multiple private SRAM blocks and copying the program image from program ROM to SRAM
- Adding a system level cache
- Using wider flash memories

2.2 RAM

The situation for SRAM is a bit different. If two processors are sharing a single SRAM, you need to make sure that the data memories (including stack and heap memories) used by the two processors

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

are in different address ranges (figure 4). Otherwise the data accesses from each processor could corrupt each other and cause failures.

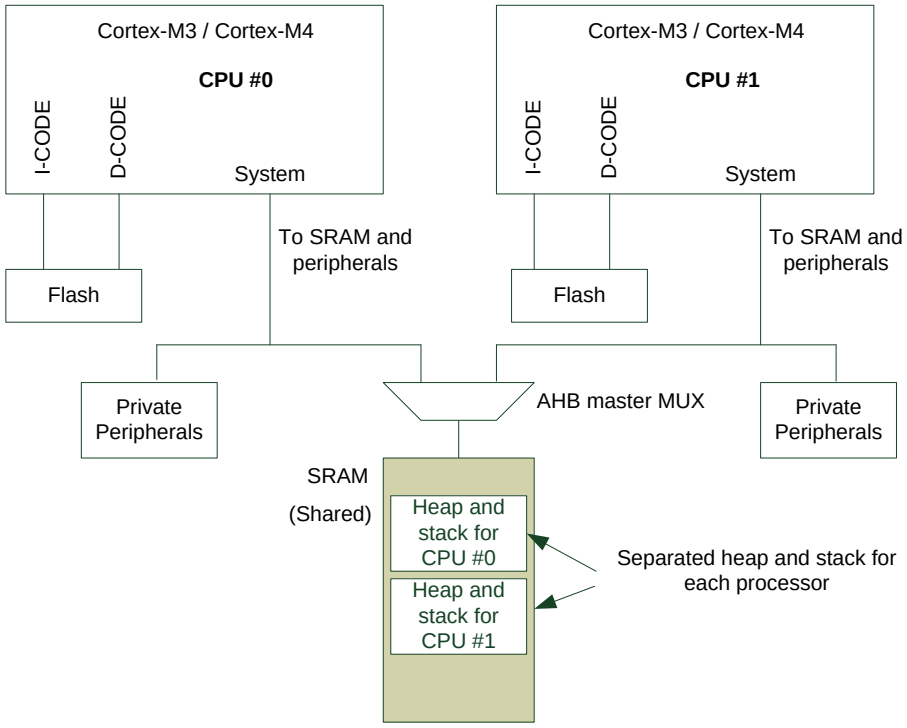


Figure 4: Stack and Heap memory areas of each processor must not overlap in shared SRAM

Typically, you can use some tricks in the system design and in system startup code, so that the stack and heap memories of each core are initialized to different ranges. For example, a hardware peripheral register can be designed to return the bus master value (HMASTER) of the AHB bus, and this value can be used during stack and heap initialization to select different stack and heap space for each core. (figure 5)

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

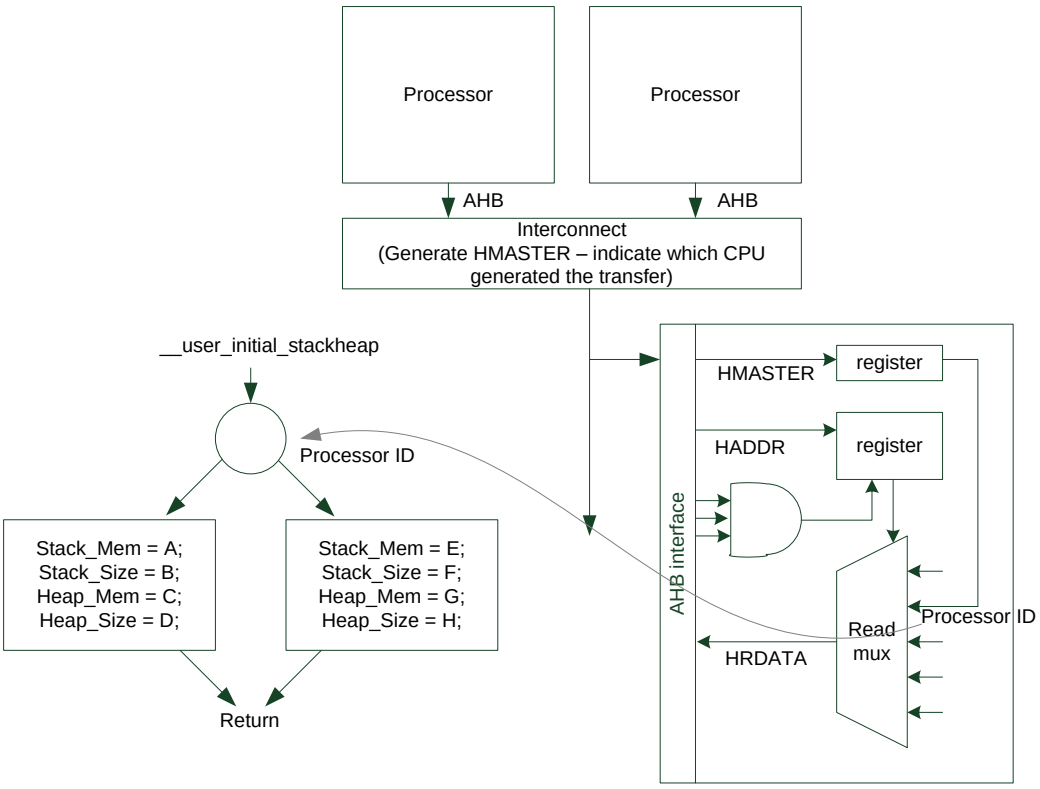


Figure 5: A processor ID register design

During the interrupt entry sequence, the Cortex-M processor needs to push a number of registers onto the stack. If the stack memory is placed on the shared SRAM, it can increase the interrupt latency. For example, if both processors receive interrupt requests simultaneously and if the bus system arbitrates the accesses using a round robin scheme, the interrupt latency for both processors can potentially be increased to almost twice of the interrupt latency in a single core system.

To reduce the latency, we could have two SRAM blocks and design the bus system so that the SRAM blocks can be shared, but also allow concurrent accesses (figure 6).

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

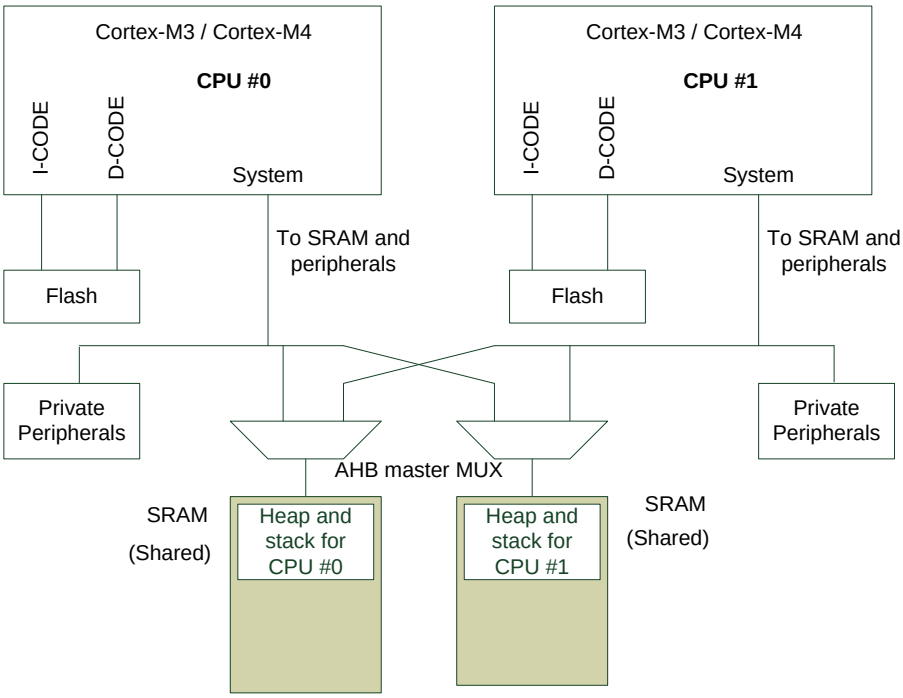
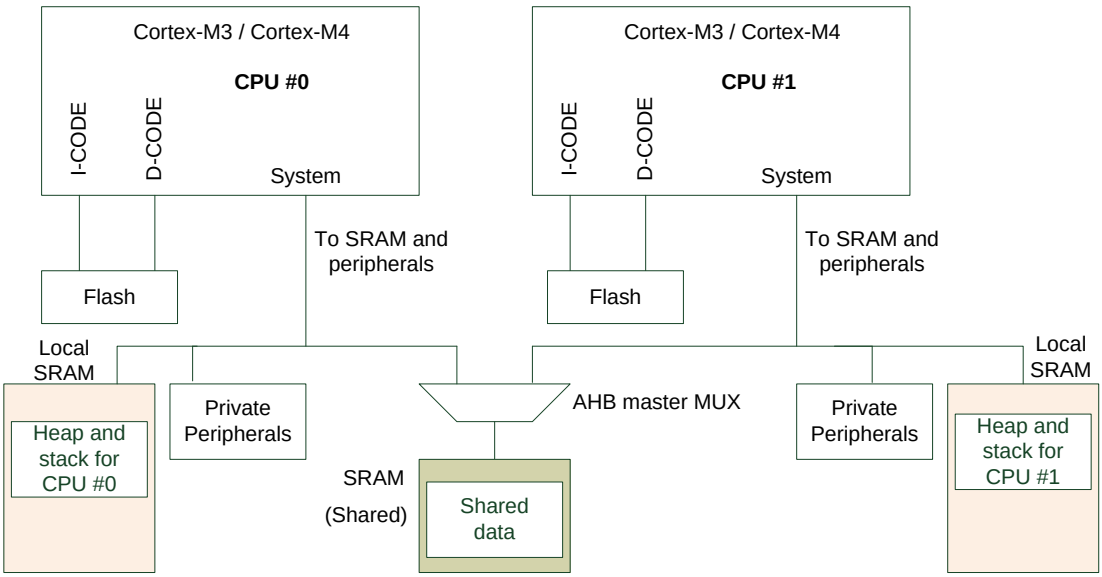


Figure 6: Dual core system with two SRAM block to allow concurrent SRAM accesses

The memory arrangement in figure 6 works for applications where the two processors use different parts of the SRAM regions. But if the two processors need to use an identical program image, the address locations of global variables and static variables will be identical between the two processors and the contentions caused by accessing these data can potentially lead to system failure.

A simpler solution is to have local SRAM for each processor (figure 7), and have a separate shared SRAM for shared data. This increases the silicon size and power usage, but helps software development as well as maintains low interrupt latency.



Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

Figure 7: Separated local SRAM for each processor

Having separate SRAM, also allows you to have separate vector tables in local SRAM for each processor. There could be a small access delay for vector fetches while stacking is taking place, which can also happen if you have the vector table in a shared program ROM memory.

Typically, in a system with many cores, each processor subsystem is likely to have its own private SRAM block (figure 8). This SRAM block is used for data (including stack and heap) as well as program code. In this way we can reduce the traffic on the shared memory as much as possible. Accesses to shared data memory and shared peripherals will still need to go through the shared bus system.

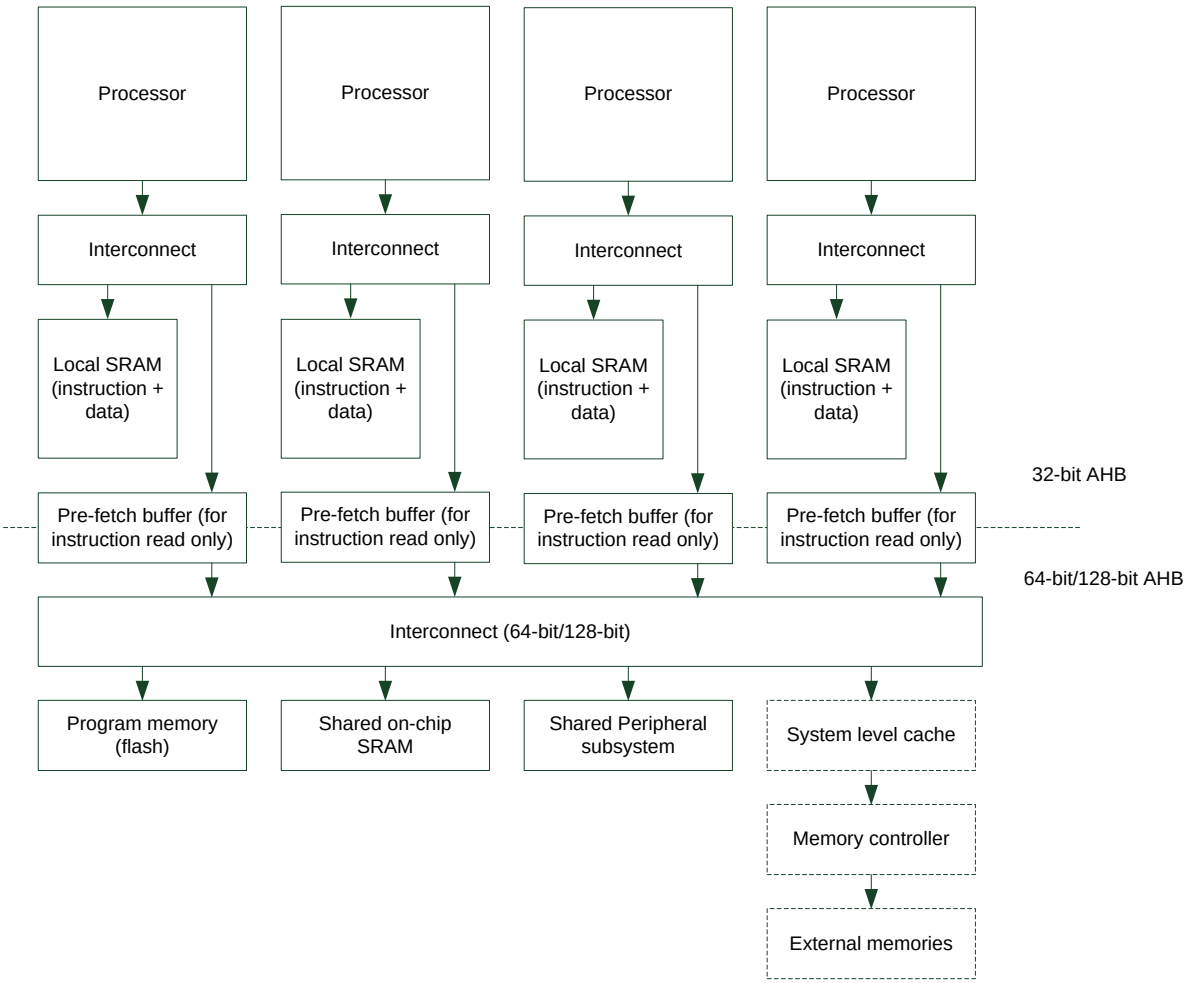


Figure 8: Example memory system design for a quad-core system

One of the potential requirements is the support of DMA for the local SRAM. In such a system the DMA bus system needs to be designed very carefully. For example, a poorly designed bus system could lead to bus deadlock in some corner cases (figure 9).

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

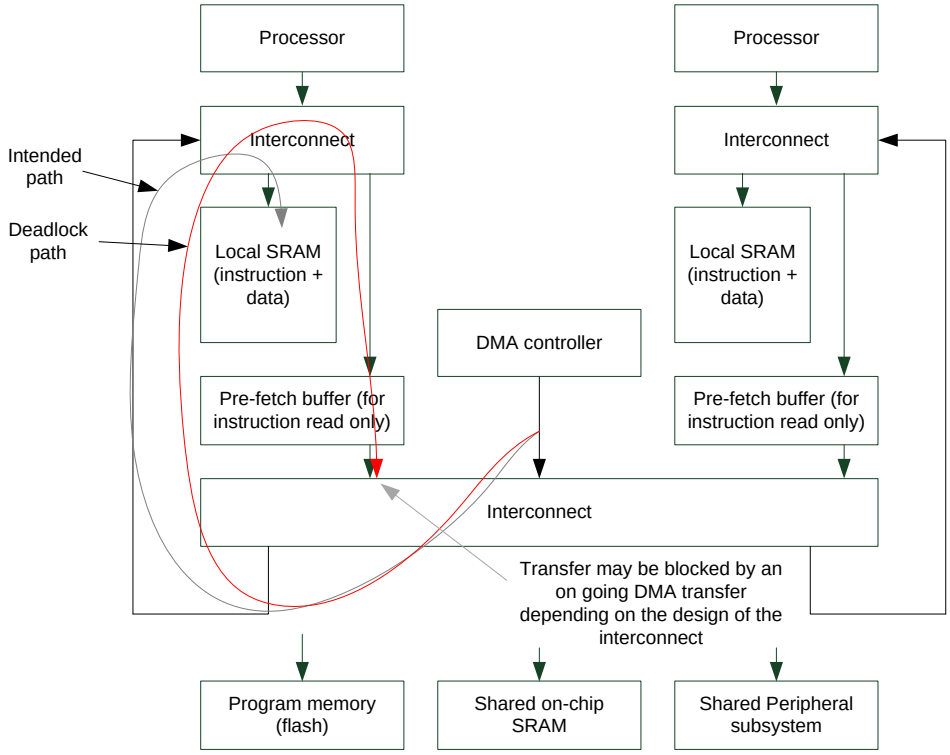


Figure 9: Bus dead lock can occur if the DMA system is poorly designed

We can design the system slightly differently to avoid the deadlock problem (figure 10).

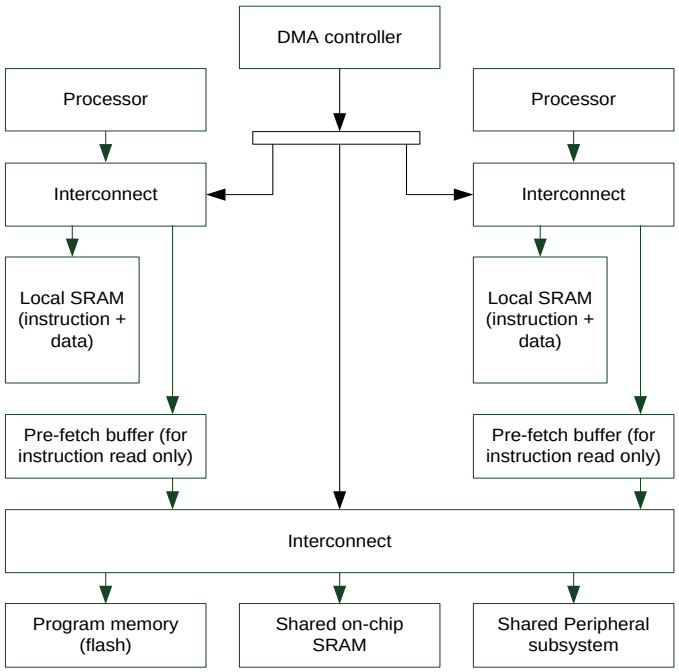


Figure 10: An alternate bus system design that avoids potential deadlock problem

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

3 Low power strategies

One of the advantages of having multiple processors is that you can put some of them into sleep mode depending on processing task load status. Potentially, you can further reduce power by powering down some of the processors. However, there may be a number of considerations when designing a multi-core Cortex-M system for low power:

- Shared system bus needs to be clocked even if just one processor is running.
- DMA controller(s) might generate bus accesses when the processors are asleep.
- Debugger accesses to the memory system go through the processor internal bus interconnect (bus matrix). So if a debugger is connected, it is necessary to keep the processor's system bus clock running even if the processor is in sleep mode.
- Some debuggers might rely on the CoreSight architecture for topology detection. So at system start up it is preferable to have all of the processors alive (but optionally in sleep mode) instead of powered down. In this way the debugger can still read the ID registers of each processor core and its debug components.

When designing a multi-core system, the clock of the bus system can be partitioned and can use clock gating to reduce power consumption. Each partition is likely to have a gated clock and potentially a free running clock signal (e.g. for interrupt detection) (figure 11).

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

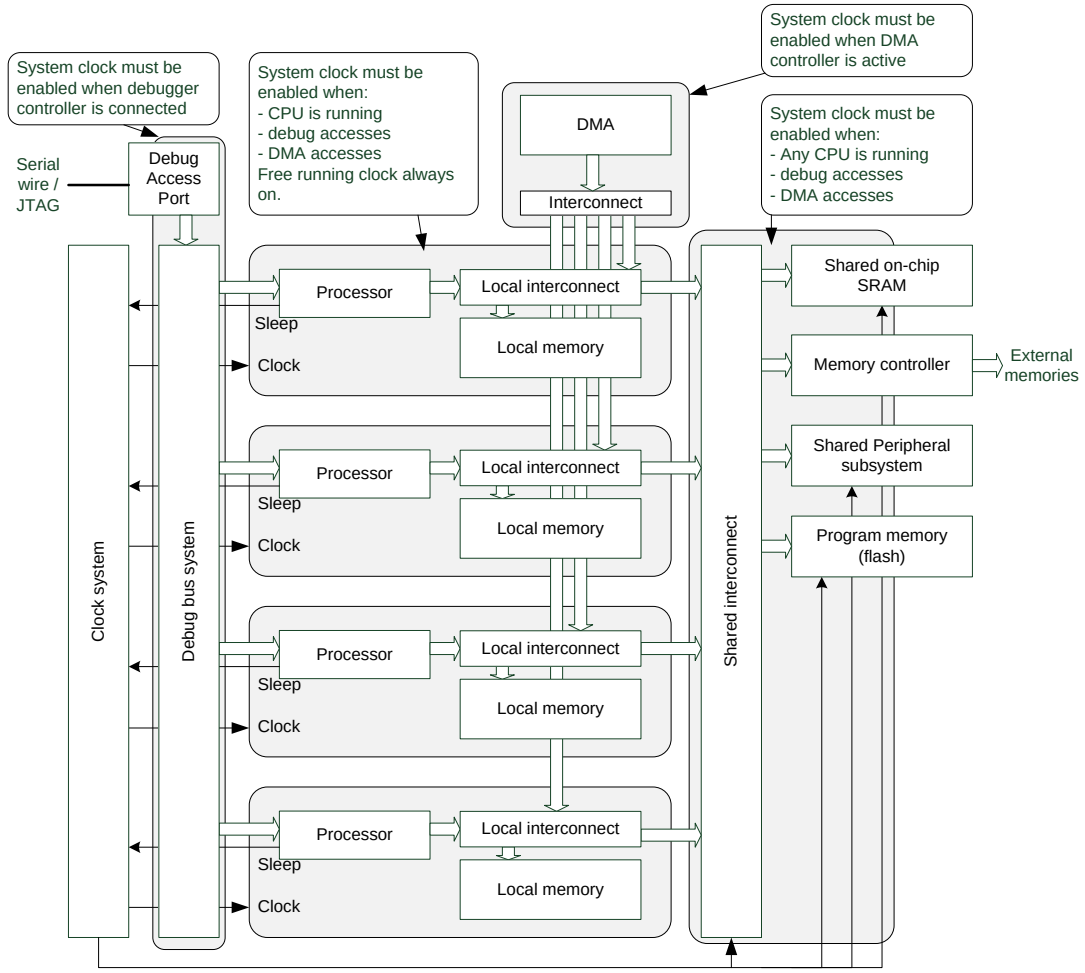


Figure 11: Clock partitioning in a multi-core design

In many applications, the system can boot up with just one processor, and then turn on other processors as required. Since the debugger might need visibility of all processors when making the connection, it could cause problems if the unused processors are powered down.

To solve this issue, all the processors can boot up, read the processor ID register (figure 5) and then enter sleep if the processor determines that it is not currently needed. It is possible to power down any unused processors later on using software control to in order to save power. For example, a master processor can read a hardware register to determine if a debugger is connected, and if there is no debugger connection, power down the unused processors. Alternatively, we can leave the unused processors in sleep mode and wake them up from sleep if needed.

The trace bus system can be powered down by default to save power, and powered up as soon as one of the processors enables its trace system. The Cortex-M3 and Cortex-M4 processors have a TRCENA output pin (Trace Enable) and this reflects the TRCENA bit in the Debug Exception and Monitor Control Register (DEMCR).

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

Once the trace system is powered up, it should remain powered up even when all the processors are in sleep mode because the trace system can still generate periodic synchronization packets for maintaining trace port synchronization, and sleep profiling event packets. In addition, there could be data that remains in the data buffers of the trace components. If the trace subsystem is powered down, the remaining data and the configuration settings of the trace components would be lost.

4 Debug and Trace system design

The Cortex-M processors are designed to support the CoreSight Debug architecture. The debug and trace systems are very scalable (from single processor to multi-core designs). By default, the Cortex-M processors are shipped with an example integration layer for single processor system. For example, figure 12 shows the Cortex-M3 and Cortex-M4 integration layer.

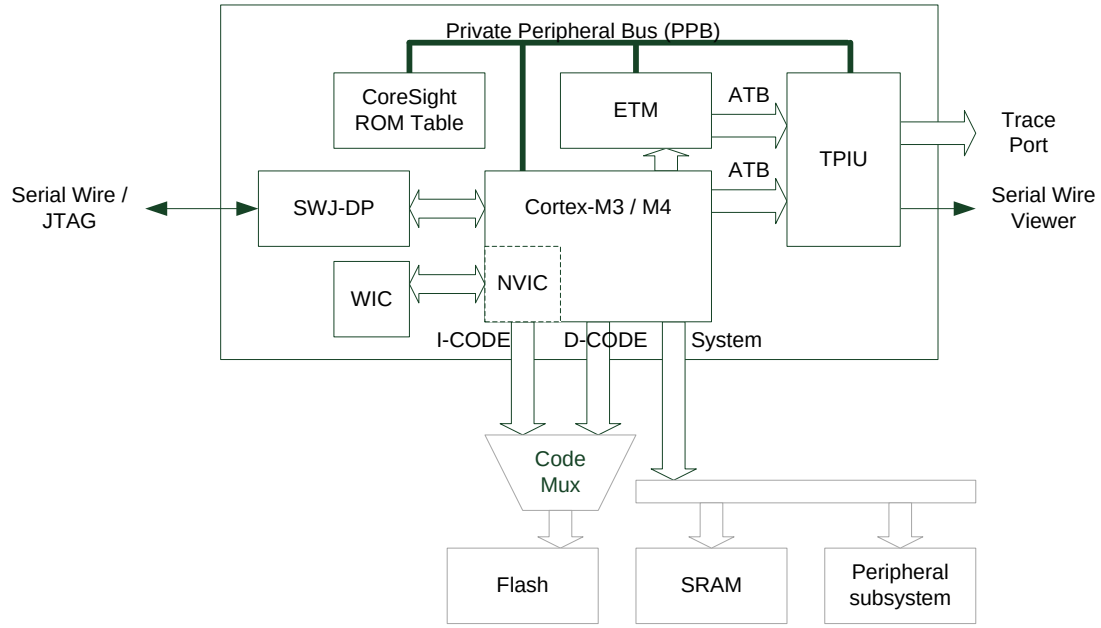
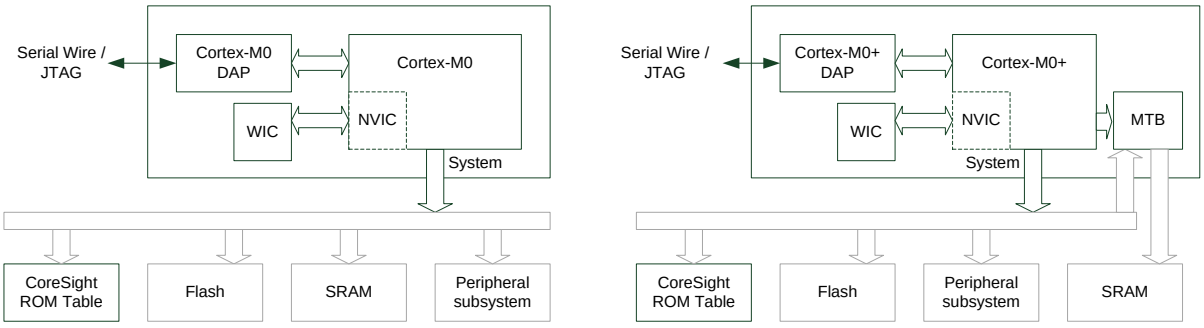


Figure 12: Cortex-M3 and Cortex-M4 Integration layer for single core system

The Cortex-M0 and Cortex-M0+ processors have similar example integration layer designs in their deliverables (figure 13). Some of the details are different, for example, Cortex-M0 and Cortex-M0+ processors do not have a trace bus interface (ATB ports), and the debug interface design for Cortex-M0 and Cortex-M0+ has a smaller DAP (Debug Access Port) which only supports one debug protocol.



Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

Figure 13: Cortex-M0 and Cortex-M0+ Integration layer for single core system

In a multi-core system based on Cortex-M processors, the debug and trace system may look like the design as shown in figure 14.

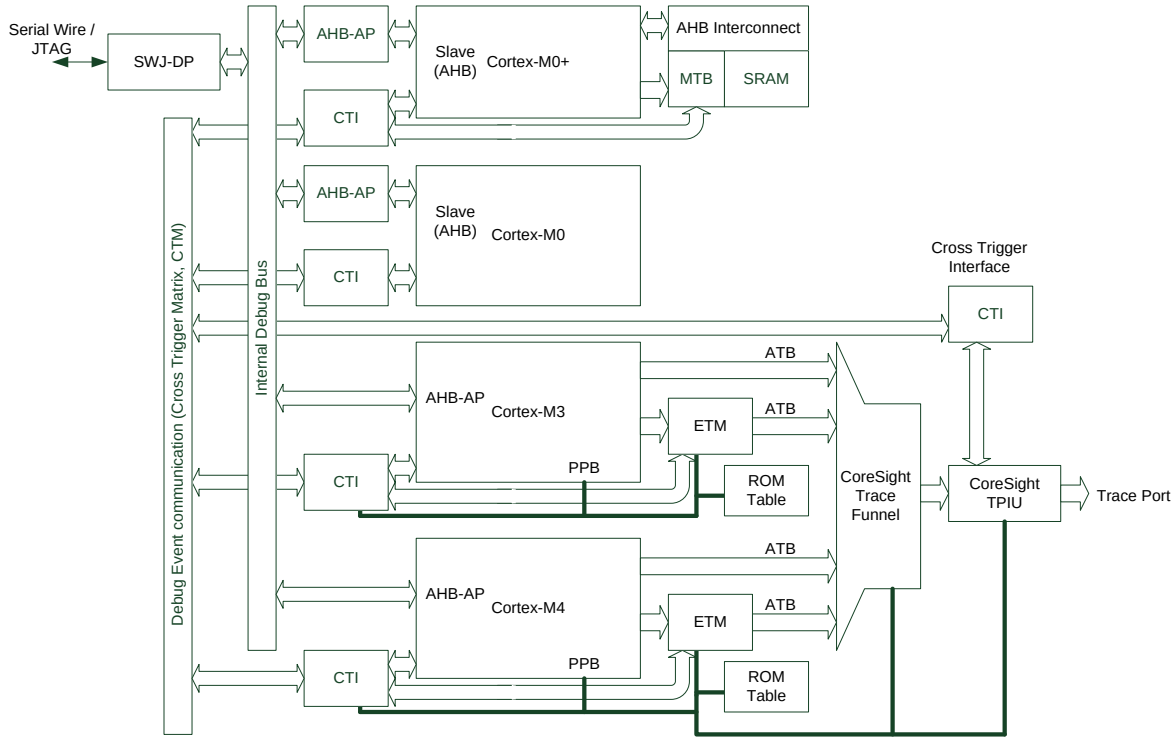


Figure 14: A basic debug system with multiple Cortex-M processors

Figure 14 shows a number of CoreSight components:

- CTI (Cross Trigger Interface) – a programmable event mapping logic block for debug event transfers.
- CTM (Cross Trigger Matrix) – a configurable unit for distribution of debug events
- AHB-AP (AHB Access Port) – a device to convert commands from a debugger into AHB transfers.
- CoreSight Trace Funnel – a programmable device to merge multiple trace sources into a single trace bus.
- CoreSight Trace Port Interface Unit – a Trace port interface design to support high trace bandwidth (up to 32-bit Trace Port width compared to 4-bit on the Cortex-M3/M4 TPIU)

Using the CTI and CTM, the debugger can synchronize debug events between various processors in the system. For example:

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

- When a debug event such as a breakpoint occurs on one processor, the event can propagate to other processors and halt all of the processors at the same time.
- The debugger can restart all processors at the same time.
- A trigger event occurring on one debug component can propagate to all processor systems to insert trace packets in the ETM / MTB traces.

By using such an arrangement, the various processors in the system can share:

- A single debug connection
- A single trace port interface
- A single programmable debug event communication network

The trace system can be enhanced with additional components such as an ATB upsizer to improve ATB bandwidth utilization, and an ATB async bridges to support trace from sources running at different clock speeds. You can also configure the system to control some of the CoreSight trace components with a separate internal debug bus instead of using the Cortex-M processors' PPB, as shown in figure 15.

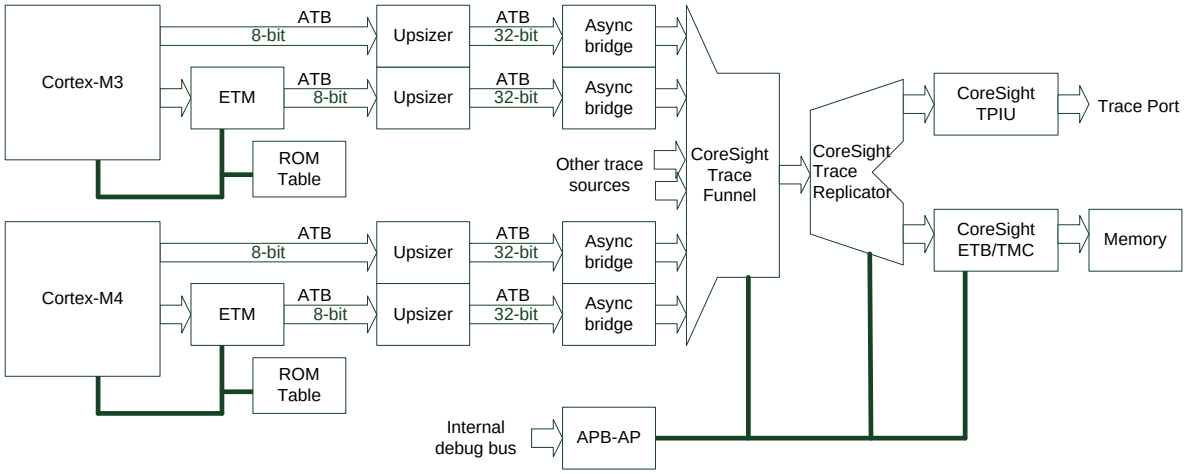


Figure 15: Trace system enhanced to support additional trace sources from different clock or power domains.

Such multi-core debug technology has been available for a number of years now. However, designers need to modify the example integration layers to create such a system. The changes include:

- Remove the SWJ-DP or Cortex-M0/M0+ DAP from the integration layer and expose the debug bus interface to the top level.
- Remove the TPIU from the Cortex-M3/M4 integration and expose the ATB interface to the top level.

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

- Add the CTI module and additional glue logic between the CTI and debug interface signals on the processor and the ETM.
- Adjust the ROM table entries to reflect the new debug component configurations.
- Adjust the clock and reset handling to enable the design to be used in a multi-processor design.

These changes incur considerable design and verification work and also require the designer to have in-depth understanding of the processor and ETM debug interfaces. To make it easier for SoC designers to design such systems, ARM provides a product called CoreSight SoC.

CoreSight SoC is a design kit that contains:

- A wide collection of configurable debug infrastructure components with IP-XACT descriptions (e.g. for use with the AMBA Designer environment).
- Tools that support automatic generation of the debug and trace system integration and their testbenches.
- Scripts to aid system design (e.g. simulation with vectors for integration testing, scripts for synthesis).

As well as offering components, CoreSight SoC provides a full system design approach/solution. The CoreSight SoC packages can be used with ARM AMBA Designer, a GUI based design environment (figure 16) that allows debug and trace systems for complex SoC projects to be built quickly and easily.

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

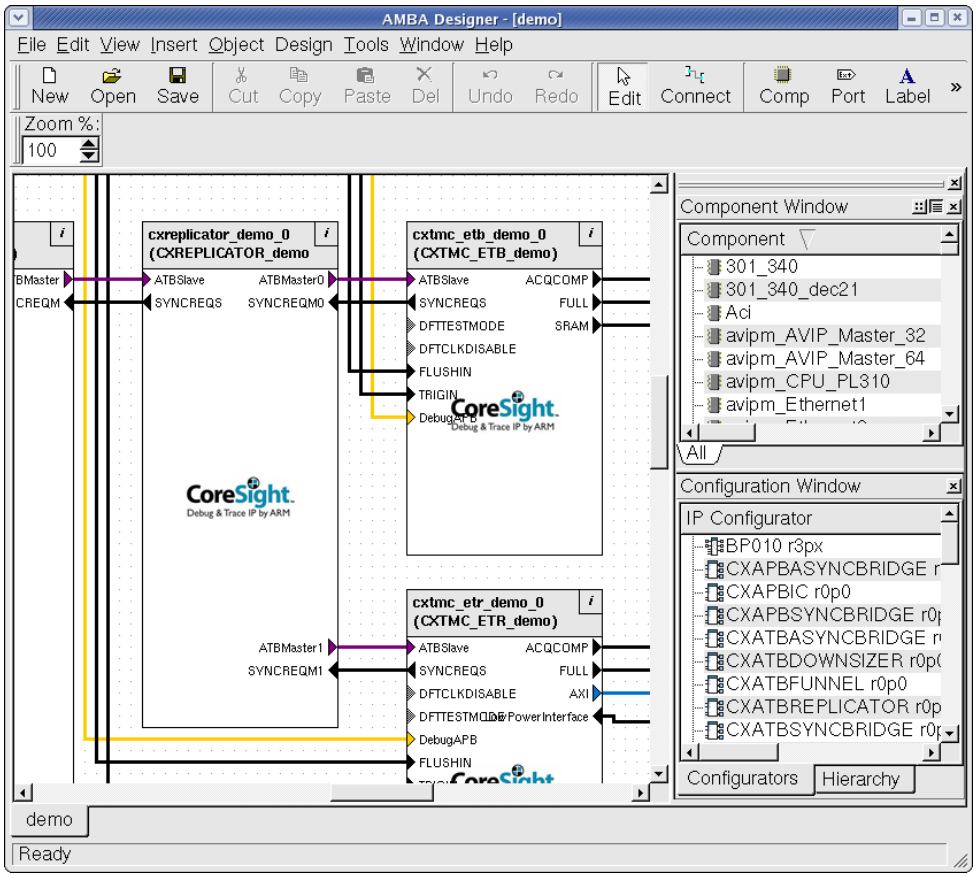


Figure 16: AMBA Designer

The CoreSight SoC product is complemented by a number of library packages that support a wide range of ARM processors:

Library package	Supported processors
CoreSight SoC LIB-400A	Cortex-A processors (e.g. Cortex-A8, A9)
CoreSight SoC LIB-400R	Cortex-R processors (e.g. Cortex-R4, R5)
CoreSight SoC LIB-400M	Cortex-M processors (e.g. Cortex-M0, M3, M4)

Table 1. CoreSight SoC library packages

Each CoreSight SoC LIB packages contains:

- Configurable Processor Integration Layer (PIL) for multi-core design.
- IP-XACT models of the PILs (e.g. for use with AMBA Designer environment).

Note that some of the newer ARM processors already include CoreSight SoC support files in the deliverables.

Using PIL designs that are pre-validated by ARM enables fast and easy integration of multi-processor designs within AMBA Designer as well as the generation of required RTL for the debug and trace system, testbenches and tests. (figure 17 and figure 18)

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

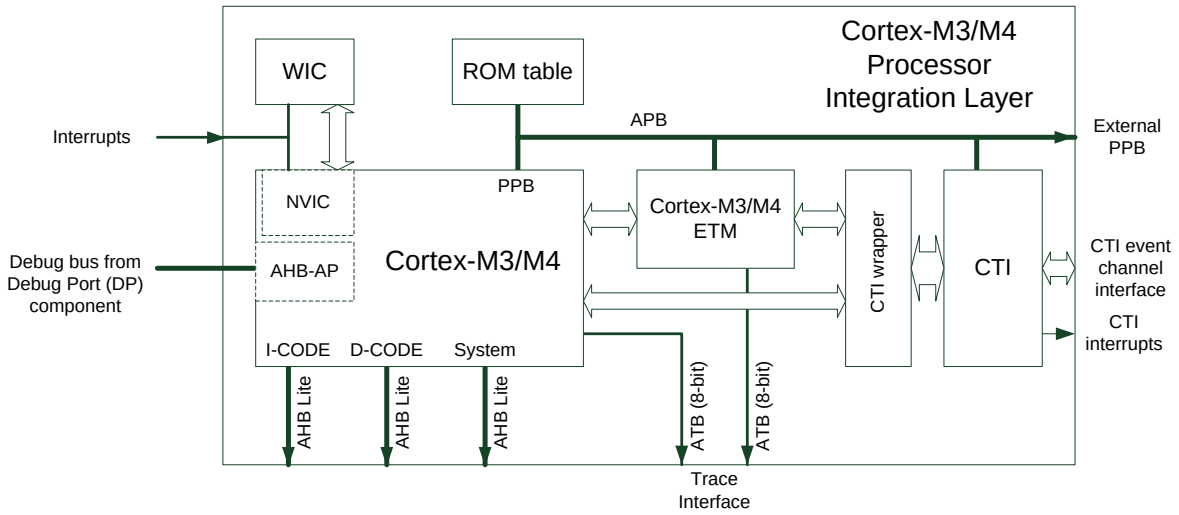


Figure 17: Cortex-M3 and Cortex-M4 PIL design in CoreSight SoC LIB-400M.

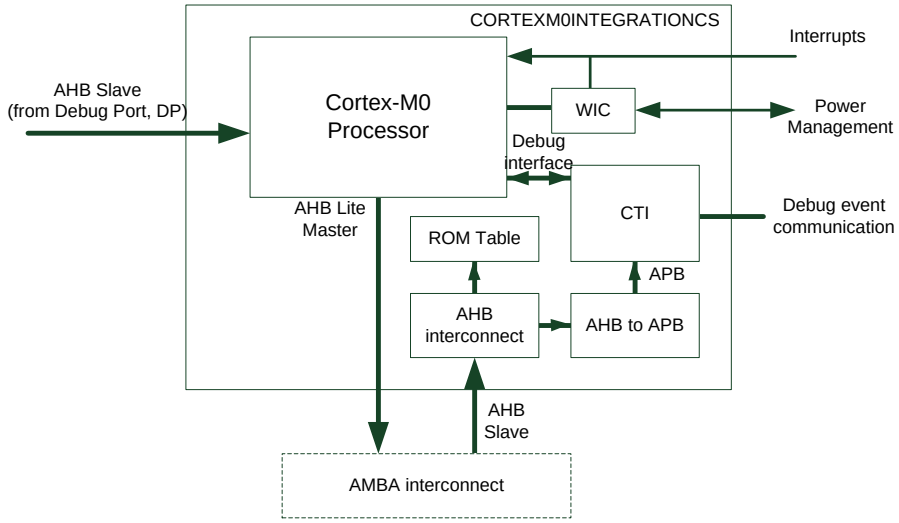


Figure 18: Cortex-M0 PIL Design in CoreSight SoC LIB-400M

AMBA Designer is not mandatory when designing a system with CoreSight SoC, it is possible to use the Verilog RTL of the PIL and CoreSight Debug components directly.

5 Summary

While the majority of Cortex-M based products are single processor designs at the moment, Cortex-M processors can also be found in a wide range of multi-processor designs. Unlike application processors, these multi-core Cortex-M systems do not use SMP (Symmetric Multi-Processing). Typically, they are AMP(Asymmetric Multi-Processing) based, and in many cases, some of the processors can be put into sleep mode or even powered down when they are not required. As a result, multi-clock domains and sometimes multiple power domains are required.

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged

Sharing of program memory is common for some multi-core processor designs. The impact to the performance caused by the shared program ROM access is a concern for some chip designers. From experiments carried out the performance drop can be around 22% with a simple memory system design, and with some simple program data buffering this can be reduced to just about 8%.

While multi-core designs often allow some of the processors to be put into sleep or even powered down, these low power features can cause complexity in the system level design as well as in the debug infrastructure. Therefore chip designers must consider various aspects of operating scenarios to ensure that their designs can operate correctly and allow program code to be debugged easily.

As multi-core designs are increasing in popularity and complexity, the methodologies of designing multi-core Cortex-M systems are becoming more critical. Manual coding of the system is possible, but can be very time consuming, and the design and verification requires in-depth knowledge of the CoreSight architecture and processor IPs. CoreSight SoC was developed to accelerate the design process and utilize reference processor subsystems called PIL (Processor Integration Layers). CoreSight SoC also includes various tools which work alongside AMBA Designer. These assist silicon designers in creating complex SoC with multiple processors including Cortex-A, Cortex-R and Cortex-M. In addition, the design environment also allows designers to test their designs quickly using automatic testbench generation.

Copyright © 2013 ARM Limited. All rights reserved.
The ARM logo is a registered trademark of ARM Ltd.
All other trademarks are the property of their respective owners and are acknowledged