ADIUVO

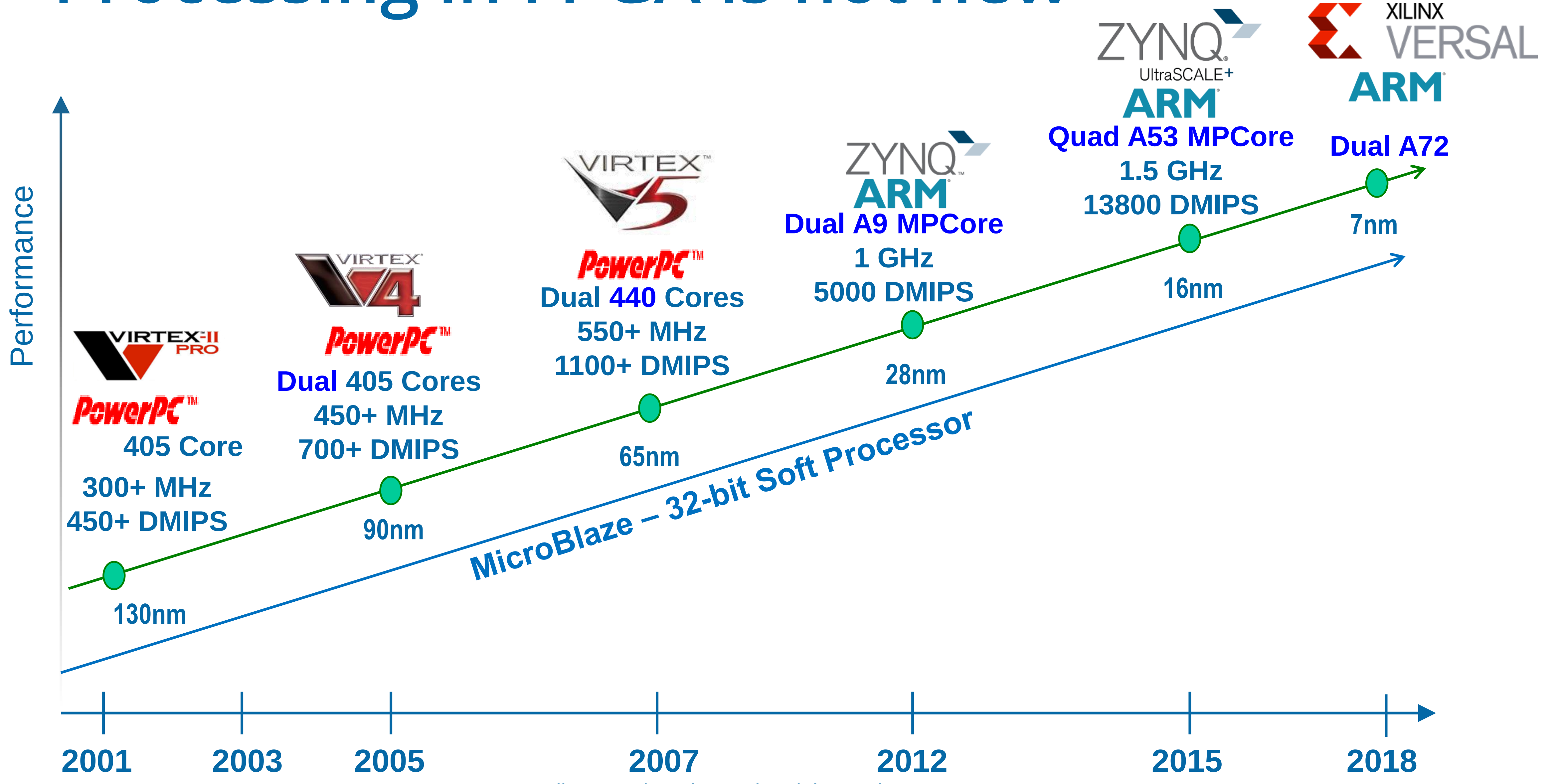
ENGINEERING AND TRAINING, LTD.

Processing In Xilinx Devices

Adam Taylor

Session 1	Session 2	Session 3	Session 4
Introduction to Xilinx FPGA	Processing in Xilinx FPGA	Embedded Linux	Accelerating Solutions

Processing in FPGA is not new

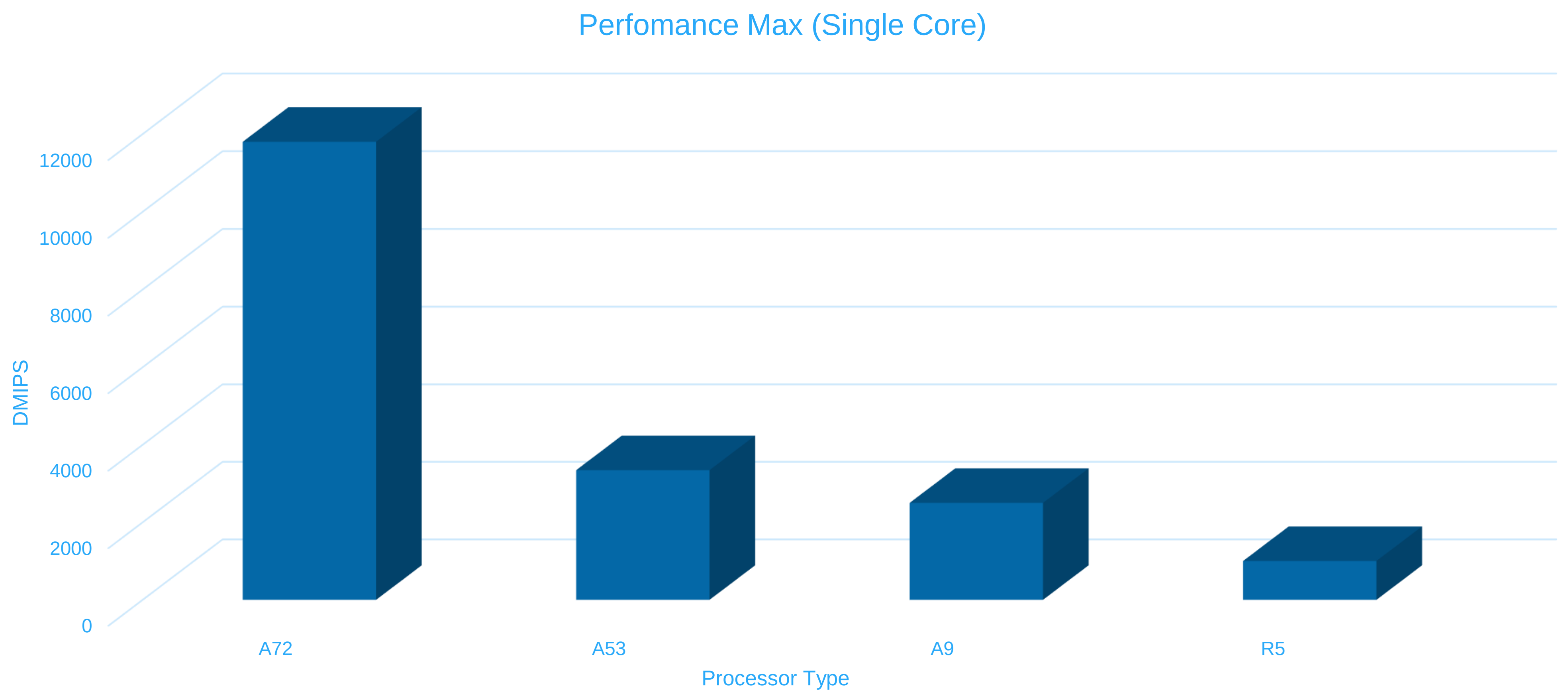


Hard and Soft Processor Cores

Hard Processors In Xilinx

- Cortex-A72 64-bit three way Out of Order Superscalar Application Processor which implements the Armv8-A architecture. Performance wise the Cortex-A72 cores can achieve up to 4.72 DMIPS/ MHz with clock rates up to 2.5 GHz per core.
- Cortex-A53, 64-bit Superscalar Application Processor which implements the ARMv8-A architecture. Performance wise the Cortex-A53 cores can achieve up to 2.24 DMIPS / MHz with clock rates up to 1.5 GHz per core.
- Cortex-A9, 32-bit Superscalar Application Processor which implements the ARMv7-A architecture. Performance wise Cortex-A9 cores can achieve up to 2.5 DMIPS / MHz with clock rates up to 1 GHz per core.
- Cortex-R5 / R5SF, 32- bit processor designed for Real Time Safety Critical Applications which implements the ARMv7-R architecture. Performance wise the Cortex-R5 cores can achieve up to 1.67 DMIPS /MHz with a maximum clock rate of with clock rates up to 600 MHz per core.

Hard Processors In Xilinx



Hard Processors in Xilinx

- The ability to change the entire or partial contents of the programmable logic as the application demands. This enables much easier in the field updates as standards evolve or even allows for different programmable logic designs to be loaded at different parts of the application.
- Power Efficient operation, the processors can be powered down into low power modes and the programmable logic can be powered down. This enables the system to be able to offer solutions which scale power demand with use cases.
- Security the processing system contains all of the necessary infrastructure to provide the confidentiality, integrity and authentication of the application thanks to AES, SHA and RSA algorithms
- Safety, the decoupling of the processing system and programmable logic enables safety solutions to be implemented using diverse approaches which with careful design do not contain a single point of failure.

Soft Processors in Xilinx

- MicroBlaze – 32-bit Reduced Instruction Set Computer (RISC) offered by Xilinx. MicroBlaze is offered in three configurations Microcontroller, Real Time and application offering 1.1 DMIPS/MHz, 1.3 DMIPS /MHz and 1.4 DMIPS/MHz respectively.
- ARM Cortex M1 & M3 32-bit processors based on the Arm Arch V6 and Arm Arch V7 respectively. The Cortex M1 offers 0.8 DMIPS/MHz while the Cortex M3 offers 1.25 DMIPS/MHz
- RISC-V is not actually a processor itself but an Instruction Set Architecture which enables development of open source processors which are compliant with the RISC-V ISA. As such there are several providers of RISC-V cores for implementation in programmable logic. With each implementation providing a different solution. The SiFive E31 RISC-V implementation offers between 2.58 and 1.61 DMIPS/MHz

Soft Processors benefits

Soft processors are tightly coupled with the programmable logic.

- The programmable logic device is the master, it must be configured first to implement the soft-core processor. Once the soft-core processor is implemented within the programmable logic can the processor then boot its application.
- Depending upon the size of the application the soft-core processors application may be contained entirely within Block RAMS provided by the programmable logic. This removes the need for an external non-volatile memory for the SW application.
- As the processor is located within the programmable logic it is not possible to change the contents of the programmable logic at run time. However, it is possible to use partial reconfiguration and reconfigure regions of the programmable logic as required
- Power Management does not have the ability to power down the programmable logic, however techniques such as clock gating and switching to slower clock frequencies for the remaining logic elements.
- While we cannot easily implement a single device implementation which is free from single points of failure. However, we can very easily implement triple modular redundancy soft processor implementations with voting and synchronisation

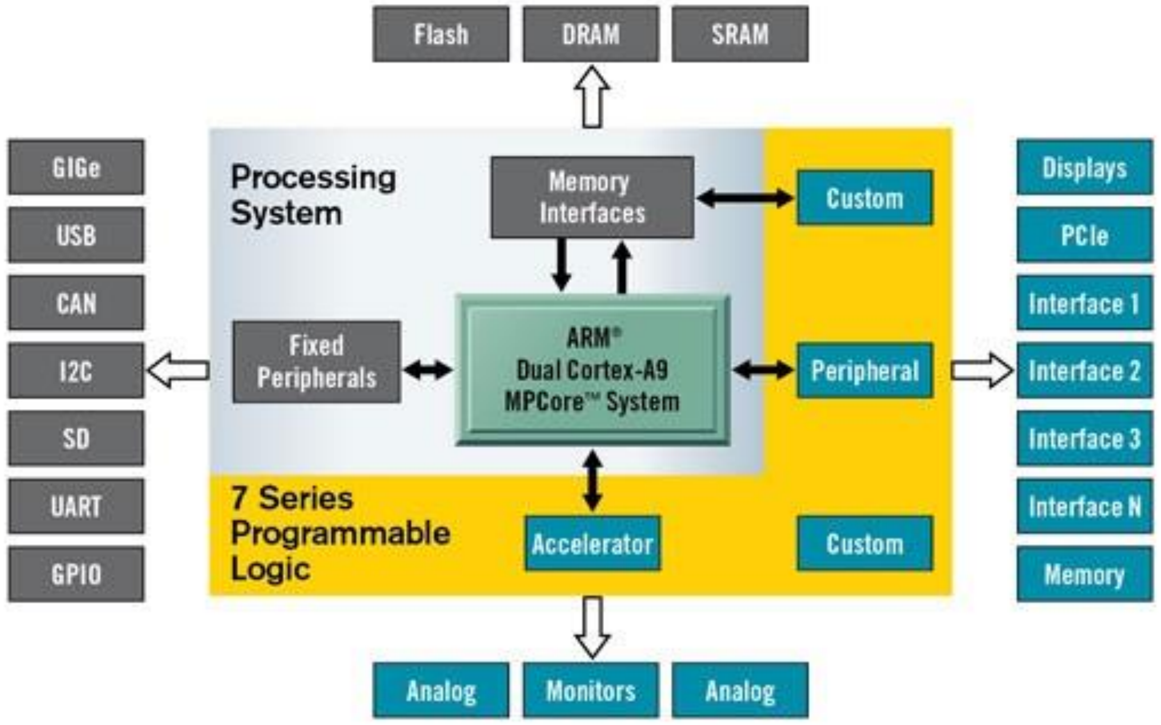
Which One Should I use?

Parameter	Hard Processor	Soft Processor	Comment
Performance	High	Medium to Low	
Impact on Logic Resources	Low	Medium to High	Depends on additional supporting components required
Customize Processor	Medium	High	Hard Processors have limited configurability
Security	High	Medium	Programmable Logic based soft implementations can still encrypt the bit stream.
Power Efficiency	High	Medium	
Portability	Low	High	If Open source is used can be very portable
Ease of Development	Medium	Low	Need to create the processor in the programmable logic first.

Processing Solutions

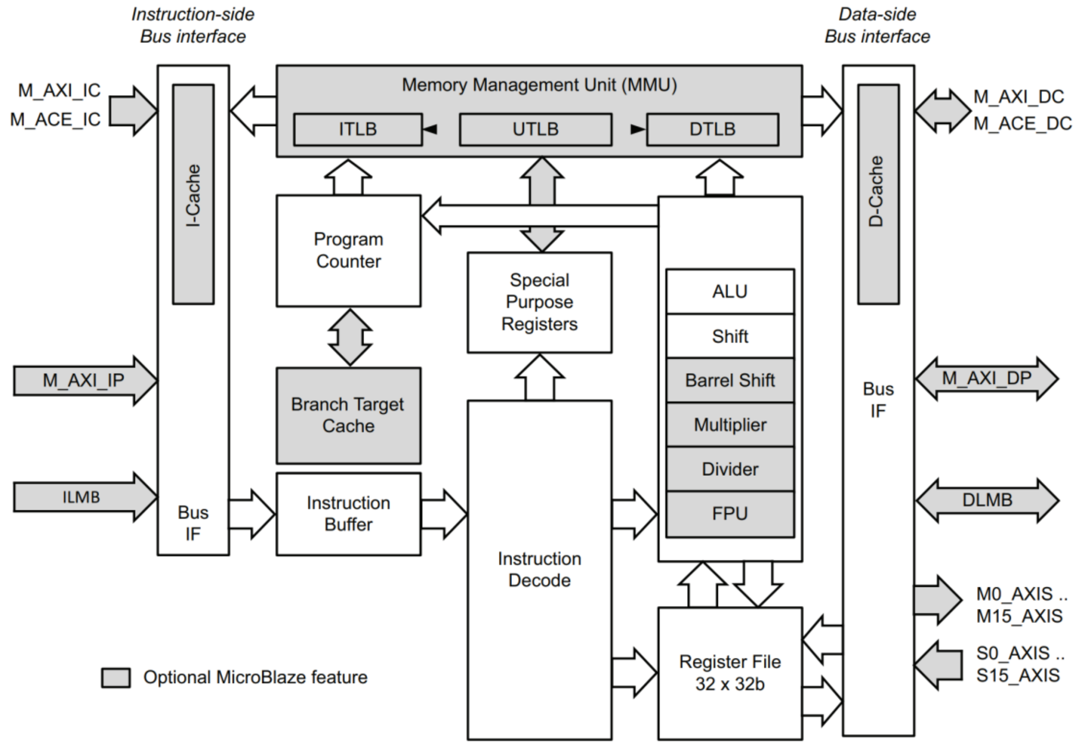
ZYNQ PS In Detail

- Dual Core Arm Cortex A9 processors
- Memory Controller – DDRx and SMC
- Peripherals
  - Gigabit Ethernet, USB, UART, SPI, IIC, CAN, GPIO
- Secure Boot – AES, RSA and SHA



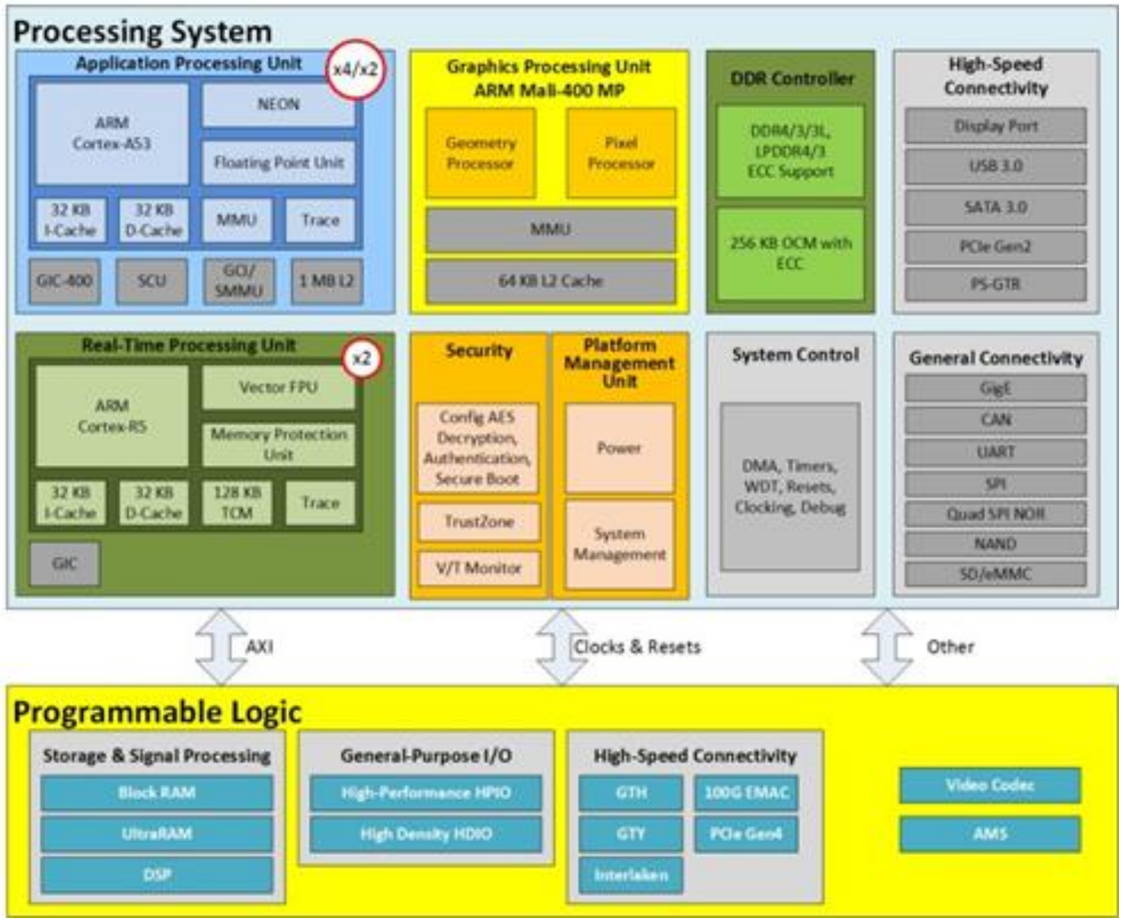
MicroBlaze

- (Mostly) softcore processor
- Harvard Architecture
- Scalable from small micro to running Linux
- Performance scales with configuration options
- Can be implemented in TMR solution (very useful for high rel)



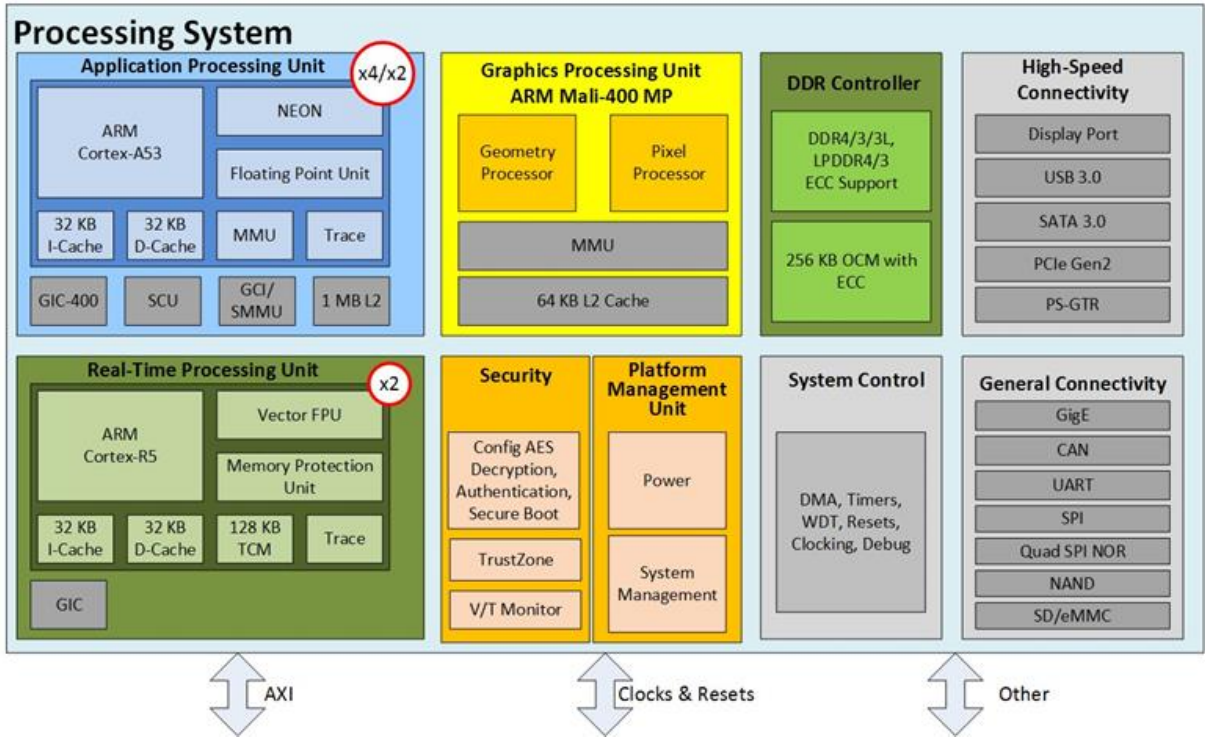
Introduction to Heterogeneous SoC

- Processing System (PS)
  - Boots first
  - Contains processors, fixed peripherals, clocks, memory, and memory controllers
  - Dedicated silicon limited configurability
- Programmable logic (PL)
  - Based on UltraScale architecture logic Details available in other topic clusters and documentation
  - Contains dedicated silicon resources: DSP48e, block RAM, high-speed serial, XADC, PCIe core, etc.
- Interconnects
  - AXI based
  - Clocks and Resets
  - Other



MPSoC /RFSoC PS in detail

- Quad/dual-core ARM Cortex-A53 processor cluster
- Dual-core ARM Cortex-R5 real-time processor cluster ARM Mali-400MP graphics processor
- Memory controllers (DDRx and SMC)
- Peripherals
  - High-speed peripherals: USB 3.0, SATA 3.0, PCIe Gen 2
  - technology, DisplayPort
  - Low-speed peripherals: CAN, UART, I2C
- Security, power management, safety, and reliability



Key Concepts of Processing Systems

Processors in FPGA

Range of elements which make for a more responsive solution in our processing solution.

- Clocking - What clock frequency and what Conversion is needed
- Memory – Internal or external memory required
- On Chip Memory – Application can execute from here normally limited size
- Cache – Closely coupled memory used to increase performance by caching external RAM data
- Interfacing – How do we interface / add peripherals to our processing system

Processors in FPGA

- On Chip Memory – Memory connected to the processors from which applications can be executed.
    - Limited in Size but maybe all that is required for some applications
    - Can be used for security e.g. Trust Zone or Secure Boot
- Cache – The cache structure will vary depending upon the complexity of the programmable SoC.
    - Each processor will have its own small level one (L1) instruction and data cache.
    - Processors will be connected to a larger level two cache (L2) for sharing data between processors.
    - More advanced options include Cache Coherent Interconnects (CCI) are provided these allow the design elements within the PL to be Cache Coherent

Processors in FPGAs

- Interfaces – The programmable SoC will be required to interface with several different commonly used interface standards.
  - Range from SPI, UART and I2C designed to interface with other devices and sensors to more complex higher speed links such as Gigabit Ethernet and PCIe.
  - Typically, a programmable SoC will provide IO bank (s) which can be connected to one of the several supported IO standards. This enables the Programmable SoC processing system to be optimal in its use of IO standards, unused IO standards do not need to be routed out allowing maximum flexibility.

Operating systems

The simplest applications will use a bare metal development.

Bare-metal developments contain no operating system. Xilinx supplies several Application Processing Interfaces (API’s) and libraries which can be used to interact with the elements in both the processor and connected peripherals.

This board support package will need to be regenerated each time the design is updated to include or remove new elements.

Operating System

For more complex applications an operating system is used, the type of OS selected will depend upon if a real time response is required or not.

If a real time response is required, then an operating system such as FreeRTOS or Micrium OS/III may be required.

If a real time response is not required but the application requires working with high level frameworks and networking, then embedded Linux is always a popular OS choice

Operating System

What differentiates a real time operating system from one which is not?

An RTOS is deterministic. That means the response of the system will meet a defined deadline

- Hard RTOS – Missing a deadline is classified as a system failure.
- Firm RTOS – Occasionally missing a deadline is acceptable and is not classified as a failure.
- Soft RTOS – Missing a deadline simply reduces the usefulness of the results.

Multiprocessing and Hypervisors

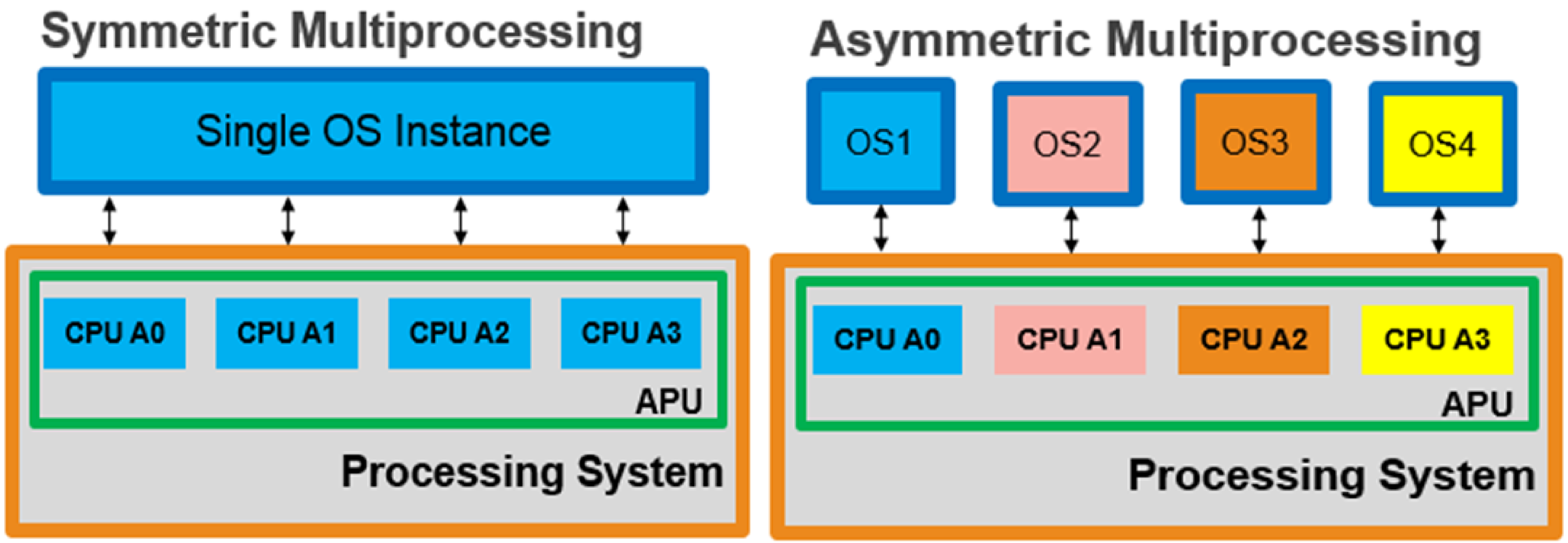
Multiprocessing and Hypervisors

Deploying multi processors is called multiprocessing

At the highest-level Multiprocessing is broken down into two distinct groups depending upon how the processors and the operating systems are combined.

1. Symmetric Multi-Processing - In a SMP application, all processors and resources are under the control of a single operating system.
2. Asymmetric Multi-Processing – In an AMP application, multiple operating systems are used on different processors cores. All operating systems share the same physical system memory.

Multiprocessing and Hypervisors



Multiprocessing and Hypervisors

Using an AMP approach allows the programmable SoC or FPGA to get the best of both worlds combining real time operating systems and application operating systems such as Linux.

The choice of AMP or SMP is not dependent upon if the SoC is homogeneous or heterogeneous.

Multiprocessing and Hypervisors

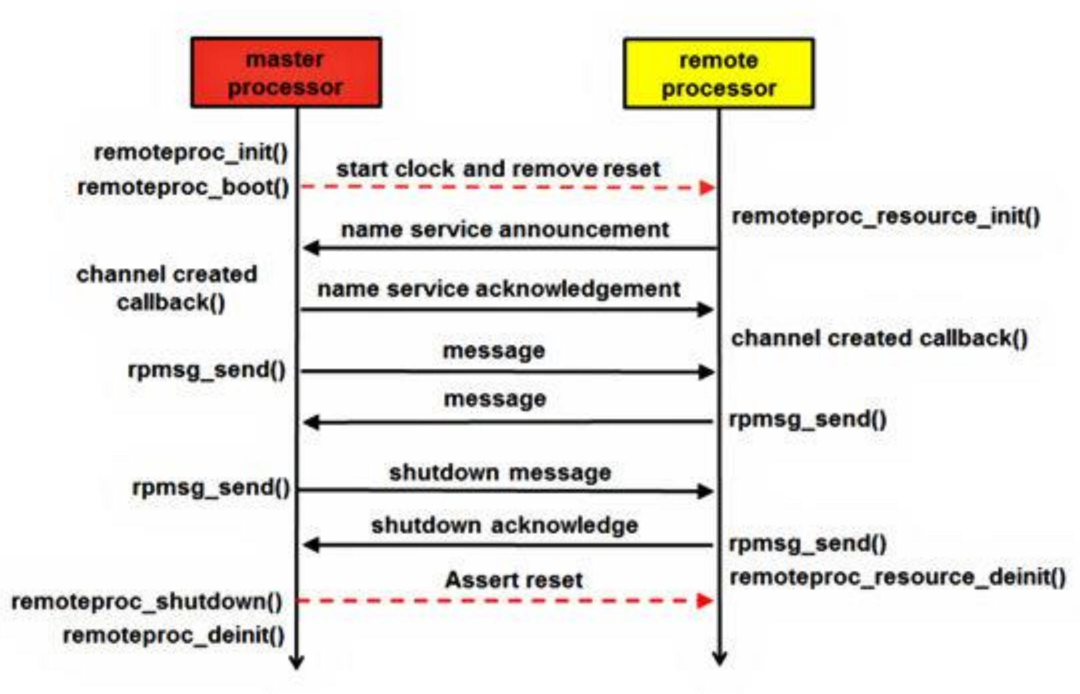
How do we communicate between different cores, running different OS?

- OpenAMP
- OpenAMP provides the framework for different operating systems to communication and manage lifecycles as required
- When OpenAMP is used within a programmable SoC a typical deployment would be a Linux operating system managing the lifecycle and establishing the communication channels with other cores running different operating systems for example bare-metal or FreeRTOS

OpenAMP Overview

Key elements of the OpenAMP framework which enable this are

- VirtIO – Virtualization, which allows communication with the network and device drivers
- RemoteProc – This is the API that controls the remote processor. It starts and stops the execution, allocates resources, and creates the virtIO devices. This API performs what is often termed the Life Cycle Management (LCM) of the remote processor
- RPMesg – The API that allows inter-process communication between the master and remote processor



Is OpenAMP always suitable ?

If we use an OpenAMP approach on a programmable SoC each operating system has potential access to the entire system resources and memory.

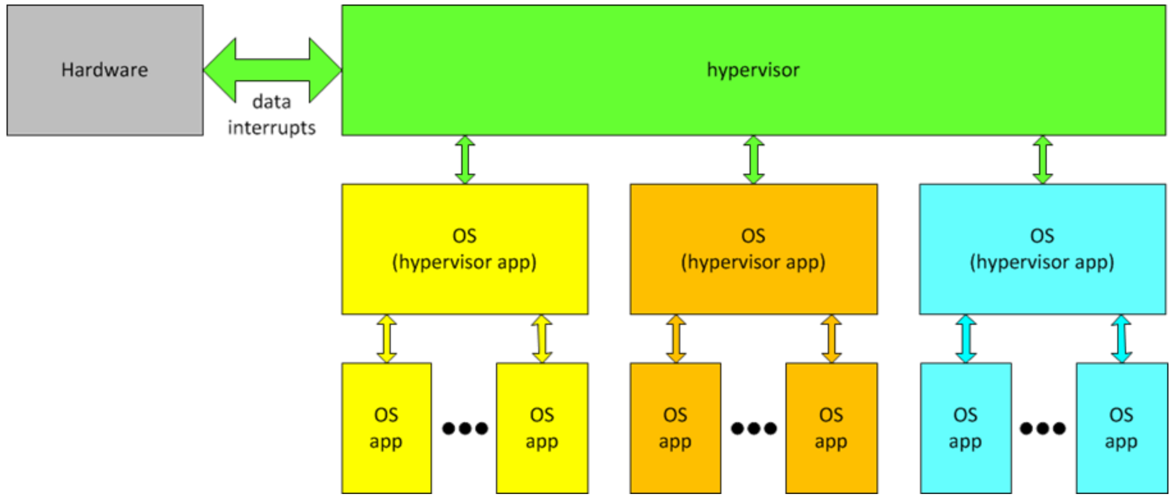
For some applications which are safety or security critical this is may not be acceptable, as such we need a different approach.

In this instance a hypervisor is often used.

Hypervisor

A hypervisor sits between the processor cores and several different virtualised operating systems.

Due to how a hypervisor functions each operating system running on the hypervisor believes it is the only operating system running on the hardware

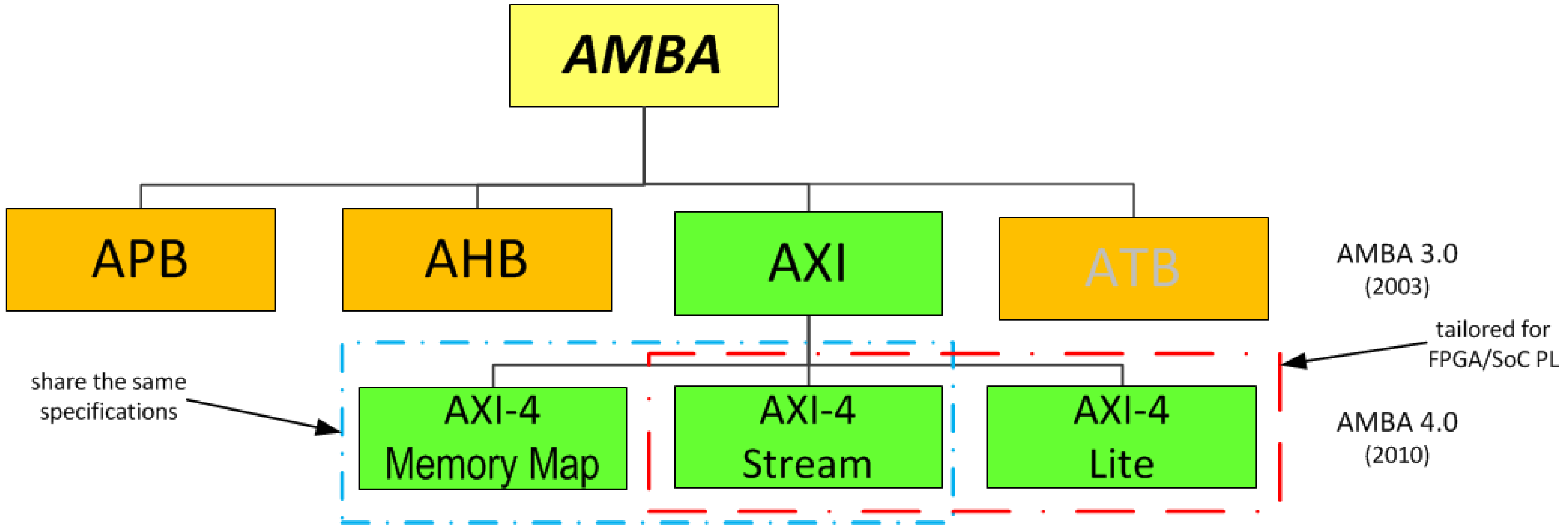


Hypervisor

Using a hypervisor enables the developer to allocate resources and enable access to system resources only as required. This provides for increased robustness and security in the end application.

Arm eXtensible Interface

AXI is Part of AMBA: Advanced Microcontroller Bus Architecture



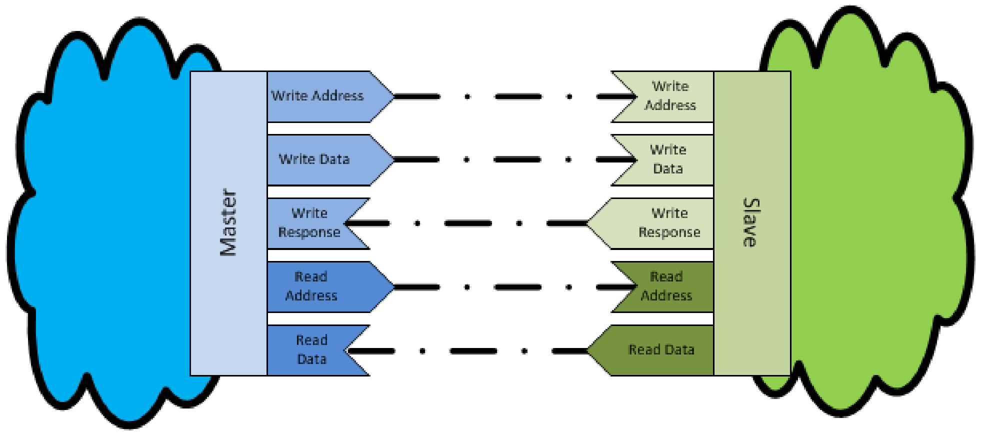
Interface	Features
Memory Map / Full	Traditional address/data burst (single address, multiple data)
Streaming	Data only, streaming
Lite	Traditional address/data—no burst (single address, single data only)

What is AXI

AXI is Advanced eXtensible Interface
- Separate address, control, and data phases
- Support for unaligned data transfers
- Burst-based transactions
- Issuing of multiple outstanding addresses
- Easy to close timing with addition of register stages

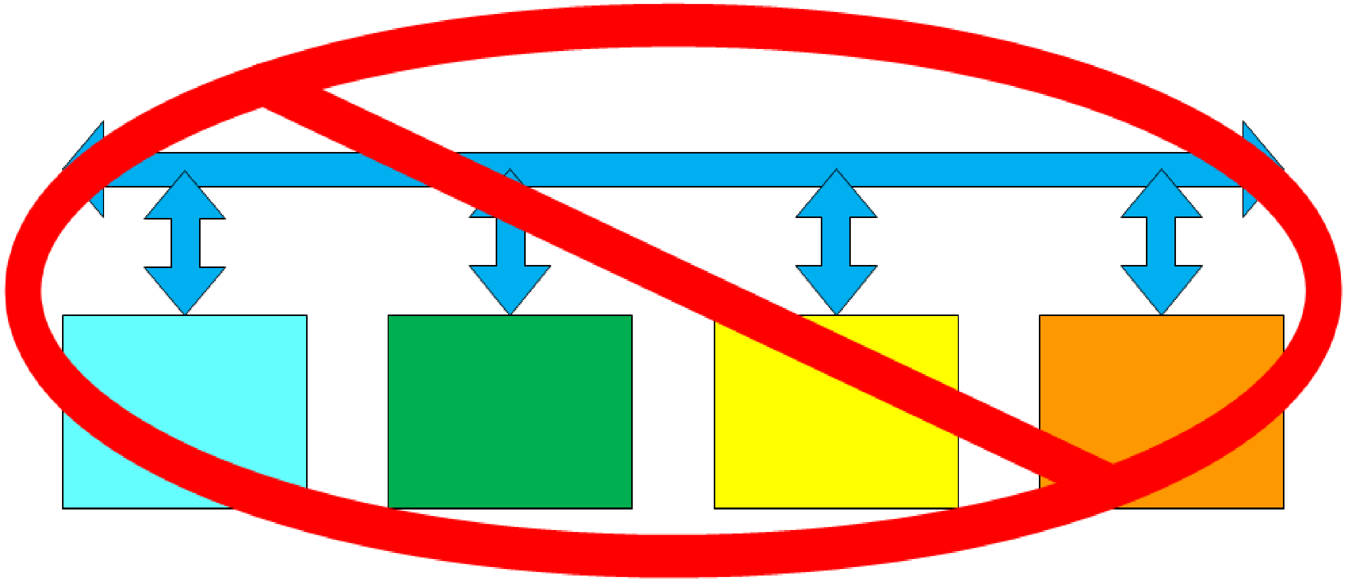
Interconnect and protocol specification
- Describes how two devices connect

The AXI specification describes an interface on a piece of IP
- It does not specify how systems of IP will be connected



What is AXI Not

- AXI is not a bus
  - Buses can be multi-tap/drop; AXI is only point-to-point
- AXI is not an electrical specification
  - No discussion of voltages or other physical requirements

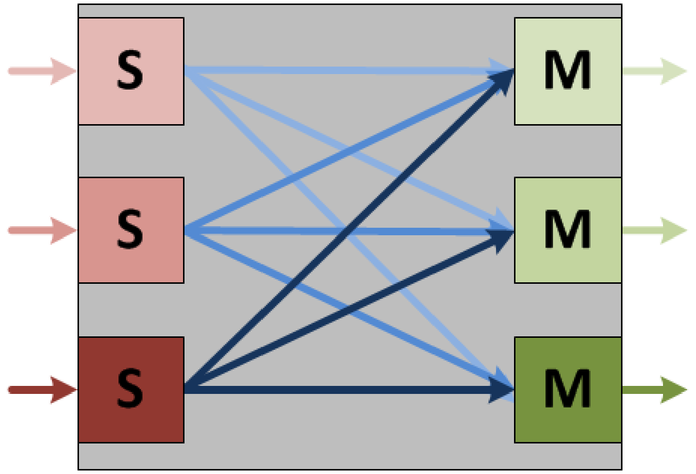


AXI Connections

AXI is a point-to-point protocol

– One master port *talks* to only one slave port

This does not mean that one master device can talk to multiple slave devices, or vice versa



Special IP is used to augment AXI to support a full hierarchy

– Crossbar: connects one or more similar AXI full masters to one or more slaves

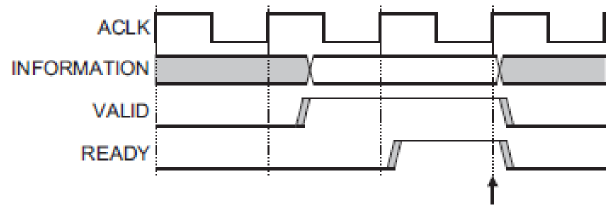
– Interconnect: similar to a crossbar, but provides for different clocks and resets

AXI Channel Handshake Basics

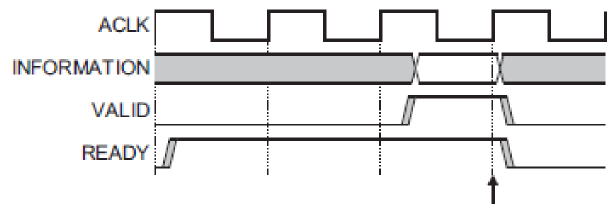
AXI uses a valid/ready handshake acknowledge

Each channel has its own valid/ready
- Address (read/write)
- Data (read/write)
- Response (write only)

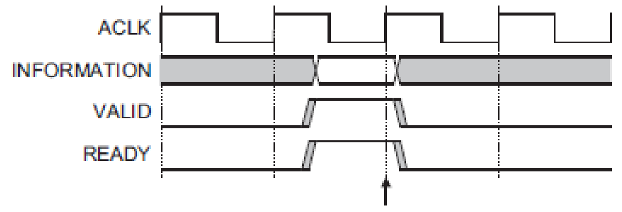
Flexible signaling functionality
- Inserting wait states
- Always ready
- Same cycle acknowledge



Inserting Wait States



Always Ready



Same Cycle Acknowledge

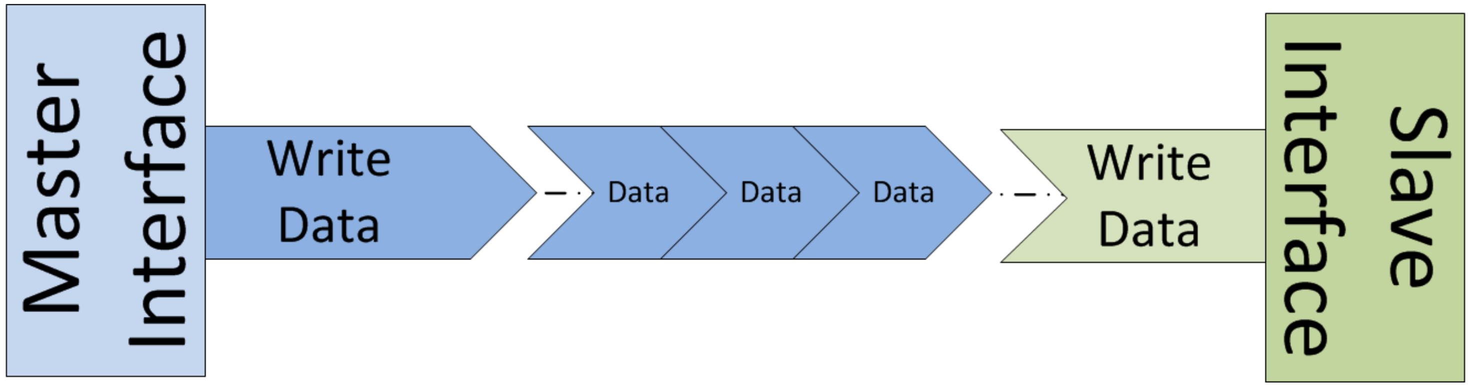
AXI Stream

Data moves in a single direction, from a master to a slave

- Simplex communication: master-to-slave
- Dual direction would be dual simplex; individual channels, for example

Data only

- No address channel
- Minimum control
- Unlimited burst length
- Sideband signals optionally used to condition data
- Contents of data stream interpreted by slave



MicroBlaze

MicroBlaze

- Xilinx Softcore processor
  - » 32-bit / 64-bit RISC Microprocessor
- Highly Configurable – implemented in logic fabric
- Three pre-configured solutions provided by Xilinx
  - » Microcontroller – Suitable for BareMetal developments
  - » Real Time – Intended for real time applications using a RTOS e.g. FreeRTOS
  - » Application – Embedded Linux (PetaLinux)

MicroBlaze Detail

AXI Interfaces enable connection to range of interfaces

- SPI / I2C
- Interrupt Controllers
- DDR
- Ethernet
- Lockstep and TMR support

Performance depends on implementation & Fabric of course

- Microcontroller 1.04 DMIPS/MHz
- Realtime & Application 1.31 DMIPS/MHz
- Spartan-7 Fmax Microcontroller 178MHz, Real Time 155 MHz, Application 120 MHz
- Kintex US Fmax Microcontroller 393 MHz, Real Time 280 MHz, Application 242 MHz

MicroBlaze Development

MicroBlaze Defined using Vivado

- » Instantiate and configure the MicroBlaze
- » Add in peripherals to interface with the external world
- » If necessary, we can implement HLS / RTL IP cores for specialist accelerations
- » Merge Bitfiles – if external memory is not used
- » Simulation of the design – V important for TMR configuration
- » Implement hardware level debugging – ILA and VIO

Software created using Vitis

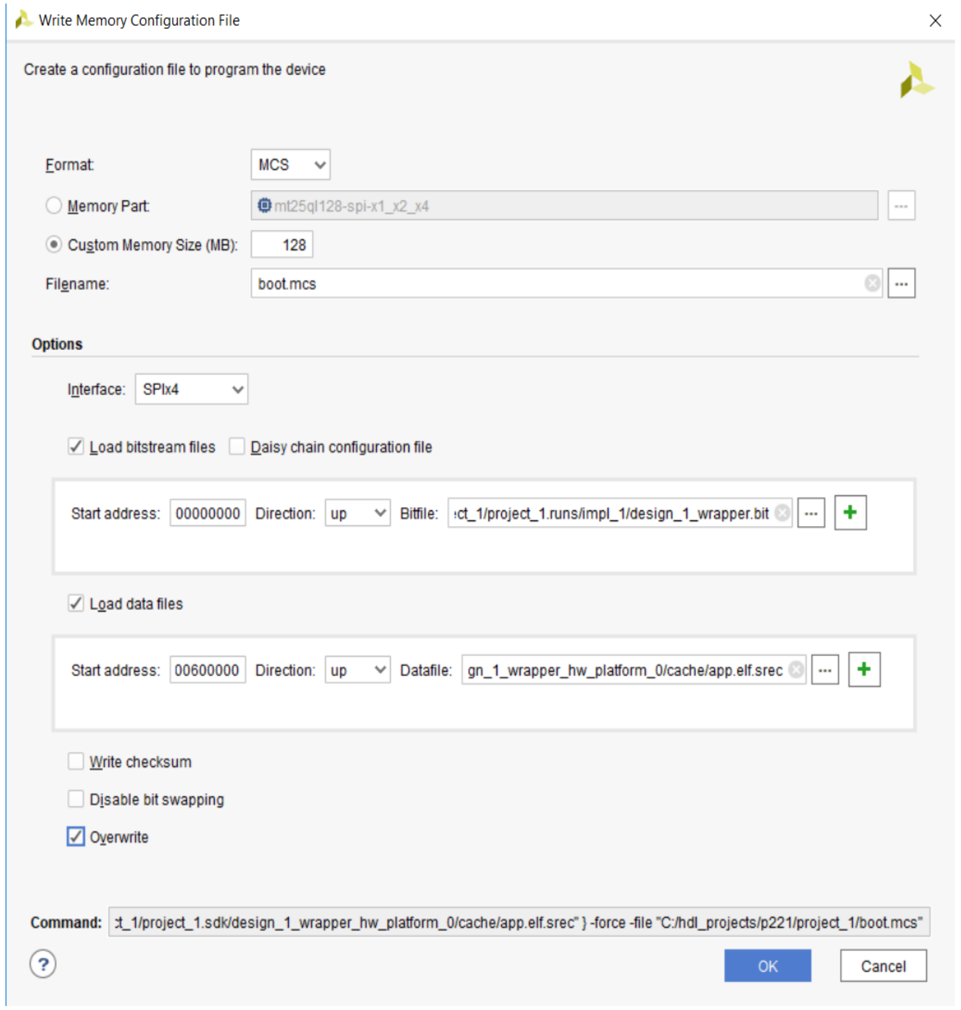
- » Application software development
- » Board Support Package creation – Pulls in API / Drivers for all IP cores in Vivado
- » Boot File creation
- » SW Debug – Memory, Breakpoints, Watch points, etc.

MicroBlaze Boot

Boot Options

- Merge the ELF file with the BlockRAM in tightly coupled memories
- External memory – Contains application
  - » Merge Srec-Boot Loader
- Vitis will create the Srec Boot Loader
- Vivado used to program merge Srec Boot Loader and program the QSPI

Possible to store both the FPGA image and SW application in the same QSPI



Create a configuration file to program the device

Format: MCS

Memory Part: mt25ql128-spi-x1_x2_x4

[✓ Custom Memory Size (MB): 128]

Filename: boot.mcs

Options

Interface: SPIx4

[✓ Load bitstream files] Daisy chain configuration file

Start address: 00000000 Direction: up Bitfile: :ct_1/project_1.runs/impl_1/design_1_wrapper.bit

[✓ Load data files]

Start address: 00600000 Direction: up Datafile: gn_1_wrapper_hw_platform_0/cache/app.elf.srec

Write checksum

Disable bit swapping

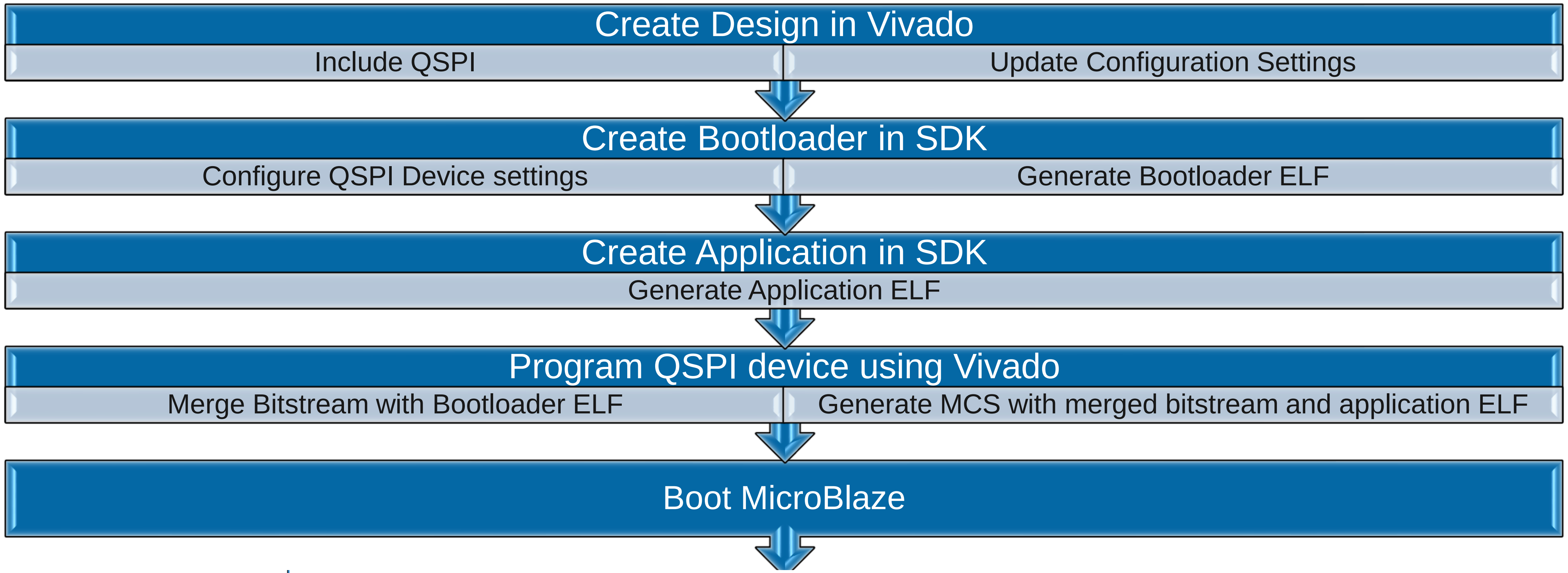
[✓ Overwrite]

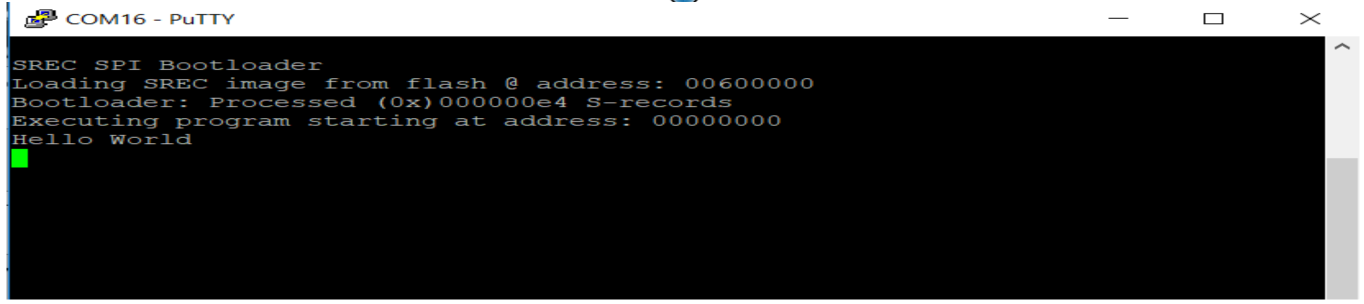
Command:

```
:t_1/project_1.sdk/design_1_wrapper_hw_platform_0/cache/app.elf.srec" } -force -file "C:/hdl_projects/p221/project_1/boot.mcs"
```

ADIUVO
ENGINEERING AND TRAINING, LTD.

MicroBlaze External Boot Flow


Create Design in Vivado
Include QSPI
Update Configuration Settings
Create Bootloader in SDK
Configure QSPI Device settings
Generate Bootloader ELF
Create Application in SDK
Generate Application ELF
Program QSPI device using Vivado
Merge Bitstream with Bootloader ELF
Generate MCS with merged bitstream and application ELF
Boot MicroBlaze



```
SREC SPI Bootloader
Loading SREC image from flash @ address: 00600000
Bootloader: Processed (0x)000000e4 S-records
Executing program starting at address: 00000000
Hello World
```

Introduction to Arm Cortex-M1/M3

ADIUVO
ENGINEERING AND TRAINING, LTD.

Cortex-M1 & M3 Architecture

Both Cortex-M1 and M3 architectures offer:

- 32 bit processor
- Arm Arch V6 (M1)
- Arm Arch V7 (M3)
- Designed for low power, low foot print, low interrupt latency
- Communication with peripherals
- 32 Bit hardware multiplier if desired
- Debug Access Port – JTAG / Serial Wire access

Forward Binary Compatibility

- M1 core implements a subset of the M3 instruction set and features

Optional Features

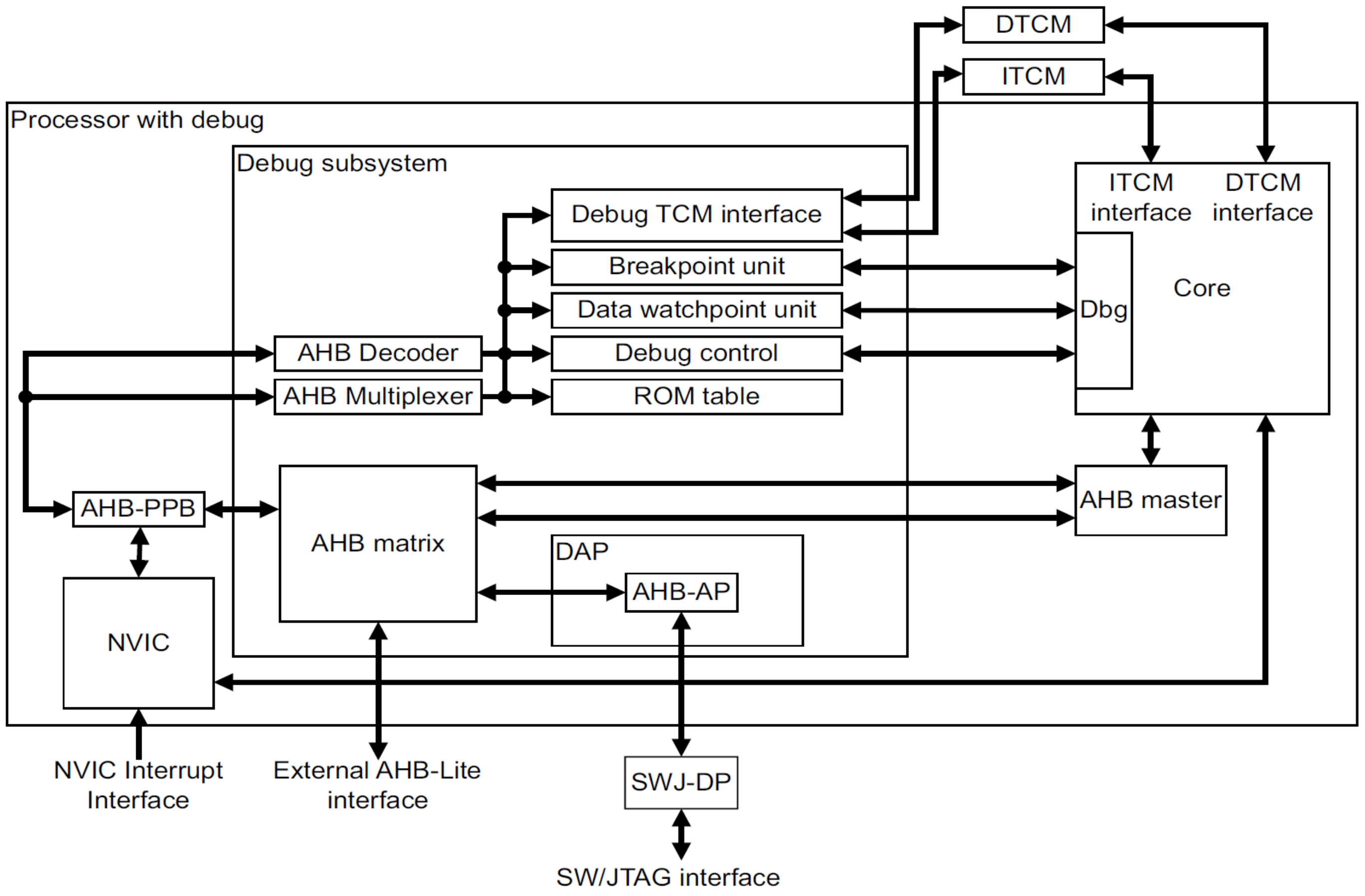
M1 Options

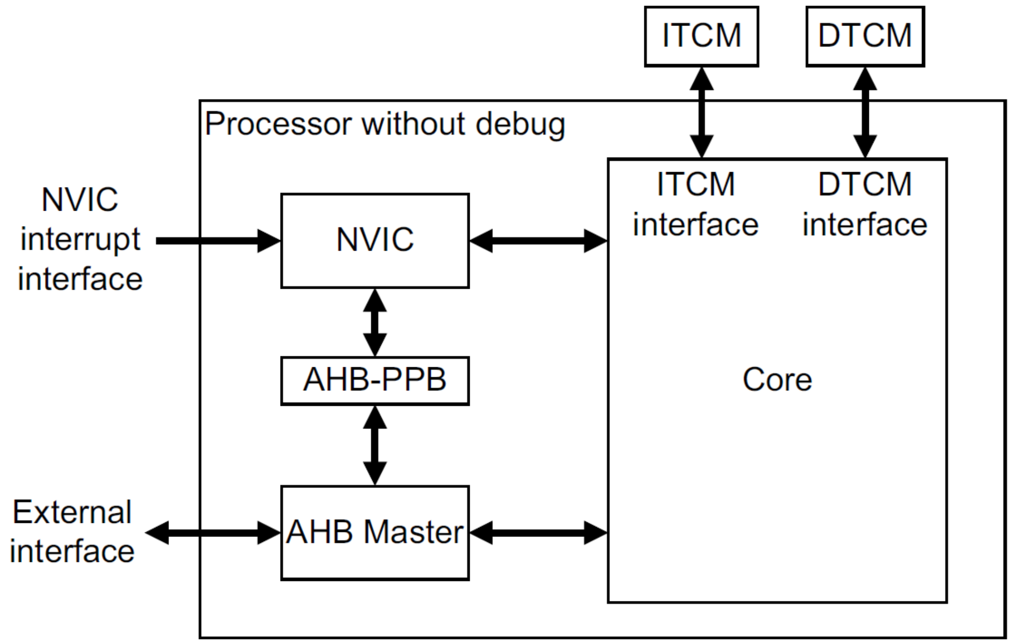
- Debug access to all memory and registers in the system, including the processor register bank when the core is halted
- Debug Access Port (DAP)
- BreakPoint Unit (BPU) for implementing breakpoints
- Data Watchpoint (DW) unit for implementing watchpoints

M2 Options

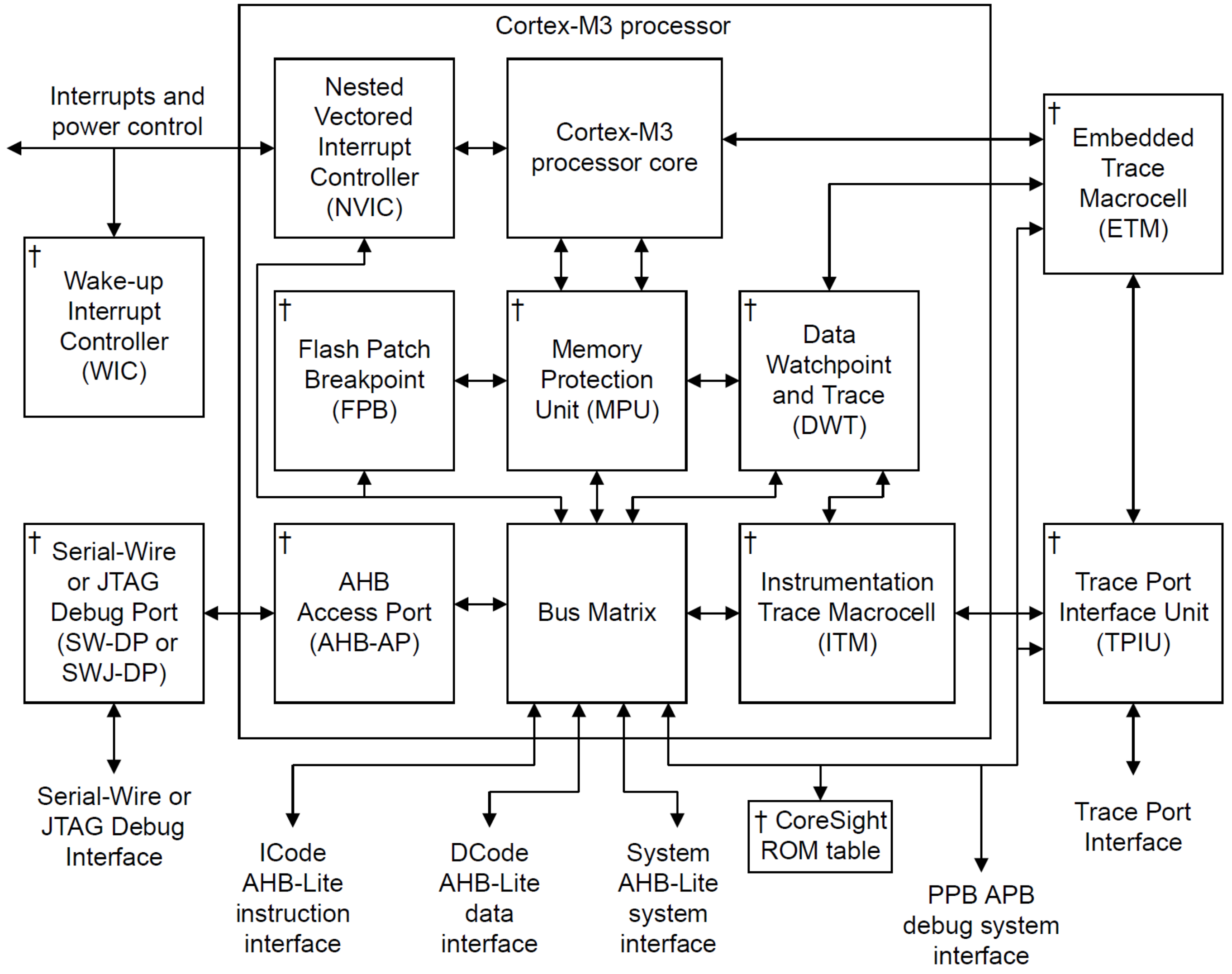
- Memory Protection Unit (MPU).
- Flash Patch and Breakpoint (FPB).
- Data Watchpoint and Trace Unit (DWT).
- Instrumentation Trace Macrocell Unit (ITM).
- Embedded Trace Macrocell (ETM).
- Advanced High-performance Bus Access Port (AHB-AP).
- AHB Trace Macrocell interface (HTM interface).
- Trace Port Interface Unit (TPIU).
- Wake-up Interrupt Controller (WIC).
- Debug Port Debug Port AHB-AP interface.
- Constant AHB control.

M1 Block Diagram



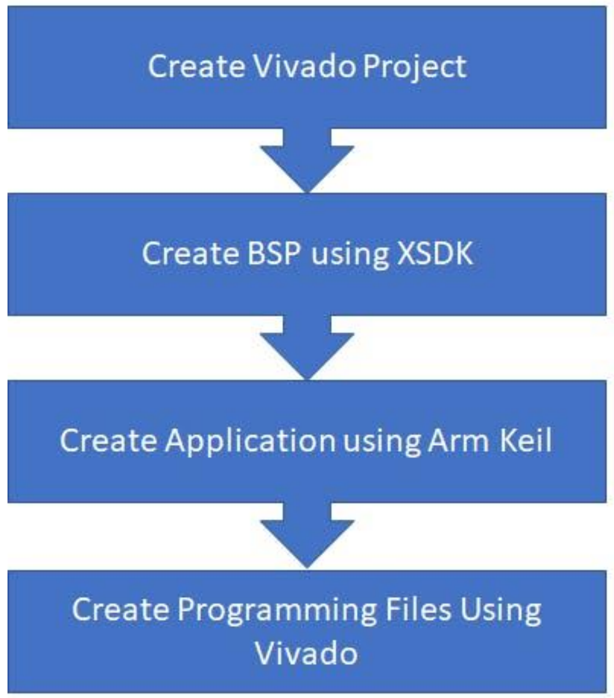


M3 Block Diagram


Cortex-M3 processor
Interrupts and power control
Nested Vectored Interrupt Controller (NVIC)
Cortex-M3 processor core
† Embedded Trace Macrocell (ETM)
† Wake-up Interrupt Controller (WIC)
† Flash Patch Breakpoint (FPB)
† Memory Protection Unit (MPU)
† Data Watchpoint and Trace (DWT)
† Serial-Wire or JTAG Debug Port (SW-DP or SWJ-DP)
† AHB Access Port (AHB-AP)
Bus Matrix
† Instrumentation Trace Macrocell (ITM)
† Trace Port Interface Unit (TPIU)
Serial-Wire or JTAG Debug Interface
ICode AHB-Lite instruction interface
DCode AHB-Lite data interface
System AHB-Lite system interface
† CoreSight ROM table
PPB APB debug system interface
Trace Port Interface

Development Flow

Creating a Cortex-M1 or M3 implementation requires the following development flow:


Create Vivado Project
Create BSP using XSDK
Create Application using Arm Keil
Create Programming Files Using Vivado

Using Vivado we create the FPGA design, including the Cortex M3 Processor and connected peripherals

Once the hardware is created, we export the hardware definition file. This hardware definition file is used by Xilinx SDK to create the Board Support Package. This provides the API and drivers for the processor configuration.

Using Arm Keil, we use the BSP and create the application itself.

The ELF file from Arm Keil is merged with the FPGA configuration file using Vivado. This creates a BIT file which can be used to program the FPGA over JTAG and a MCS file which can be loaded into the configuration memory

ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com

adam@adiuvoengineering.com