Developing Zynq AP SoC Hardware

With Xilinx
Vivado HL WebPACK

AVNET
Reach Further™

Course Objectives

Understand the Zynq-7000 All Programmable SoC development flow with Vivado's IP Integrator

Introduce the single ARM Cortex™-A9 Processors Cores

- Explore Robust AXI Peripheral Set

Utilize the Xilinx embedded systems tools to

- Design a Zynq AP SoC System
- Add Xilinx IP as well as custom IP
- Run Software Applications to test IP
- Debug an Embedded System

Agenda

Zynq Overview
Xilinx Embedded Tool Flow
- Lab 1

Zynq Processor Overview
- Lab 2

Peripherals, Peripherals and more Peripherals!
- Lab 3

The Power of TCL
- Lab 4

Lunch

Merging the Processing Subsystem (PS) and Programmable Logic (PL)
- Lab 5

Zynq PS DMA Controller
- Lab 6

Creating Custom IP
- Lab 7

Vivado's Hardware Manager
- Lab 8

Tcl Scripting
- Lab9

What's Next

AVNET

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

The First All Programmable SoC

Leading Customers

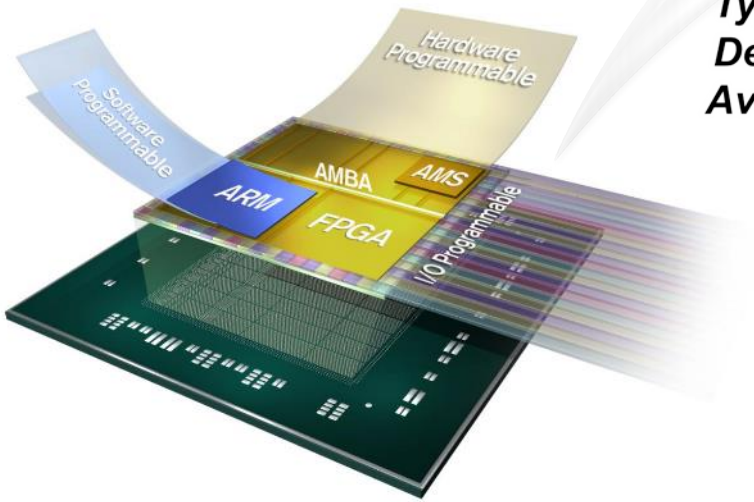
CISCO
SAMSUNG
ZTE
NATIONAL INSTRUMENTS
SIEMENS
Alcatel·Lucent
BOSCH
Invented for life
Raytheon
ERICSSON

60+ Types of Dev Kits Available

15,000+ Dev Kits Sold

200+ Ecosystem Partners

3200+ Design Wins



Industry Awards

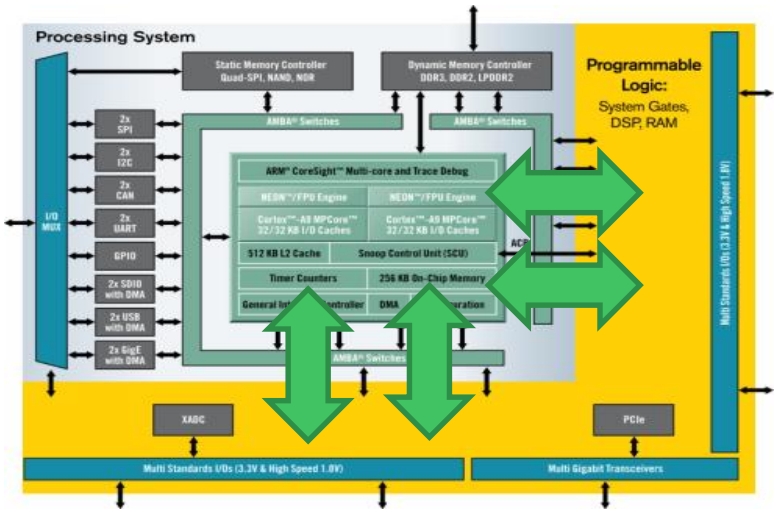
Electronic Products 2012 Product of the Year Award
The Linley Group 2012 Analysts' Choice
ELECTRON D'OR
Microprocessor Report Analysts' Choice Zynq
ACE AWARDS EE Times EDN
IET AWARDS The Institution of Engineering and Technology
Elektra2011 European Electronics Industry Awards WINNER
EDN China Innovation Award
Best of Show Award ESC 2010 San Jose

AVNET

Increased System Performance

Increasing SW Processing Performance with:

- Programmable Logic
- Massive DSP processing
- High throughput AXI
  - Over 3000 PS to PL direct connections
- High performance I/Os
- Gigabit transceivers



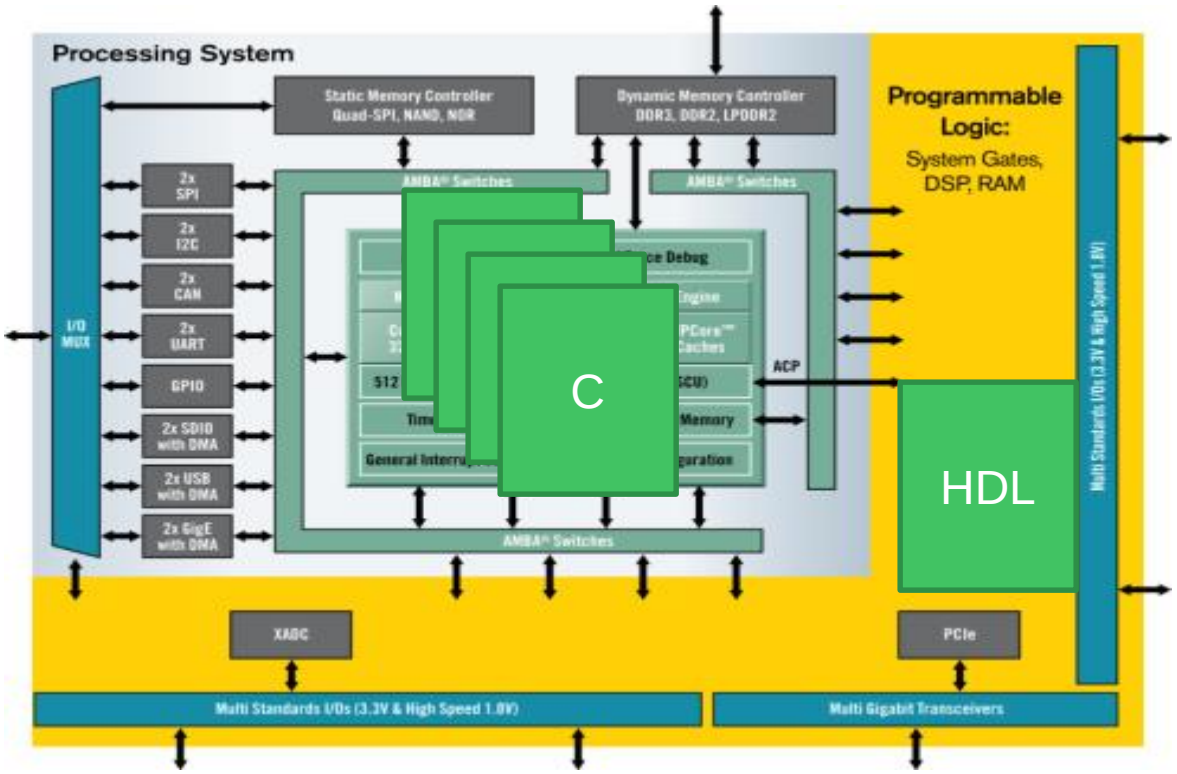
Elements	Performance (up to)
Processors (each)	1 GHz
PL Fabric/ DSP Fmax	741 MHz
DSP (aggregate)	1080 GMACs
Transceivers (each)	12.5Gbps

AVNET

Increased System Performance

Optimized & Simplified HW/SW Partitioning

- HW acceleration enables scaling SW performance to address many applications
- Low latency interfacing for efficient co-processor implementation and high throughput data transfers



AVNET

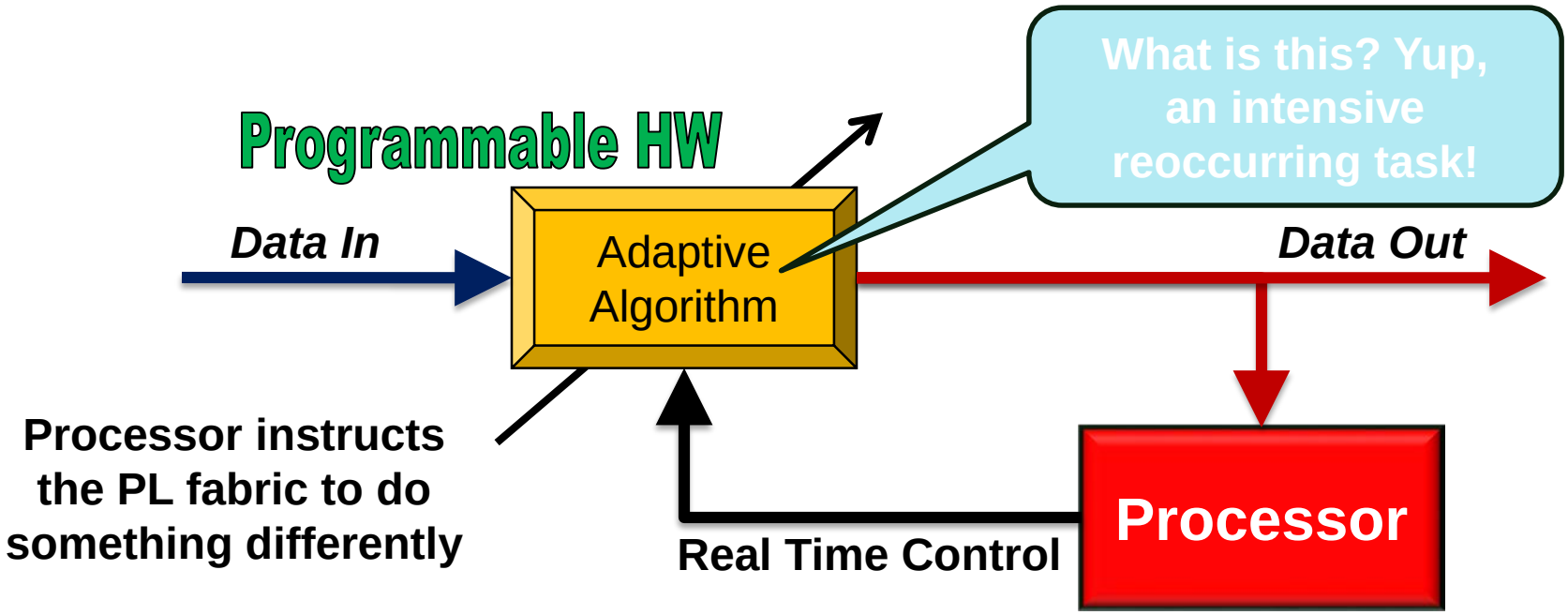
More Than Just Parallel Processing

HW Runs Heavy Data Path Algorithms

- Take advantage of HW parallelism
- Maintain high bandwidth throughput (High Frequency)

SW Controls and Updates Elements of Algorithms

- Easier to process complex input criteria
- Typical lower data rate then actual data path (Off-load HW)

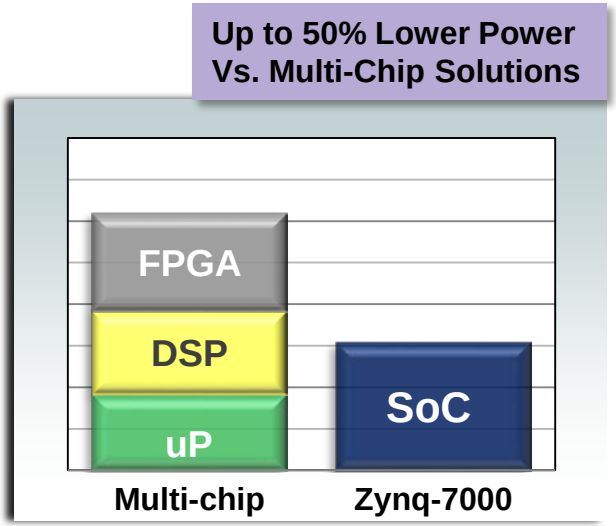


SDR, Video, Encryption, Data Processing

Total Power Reduction

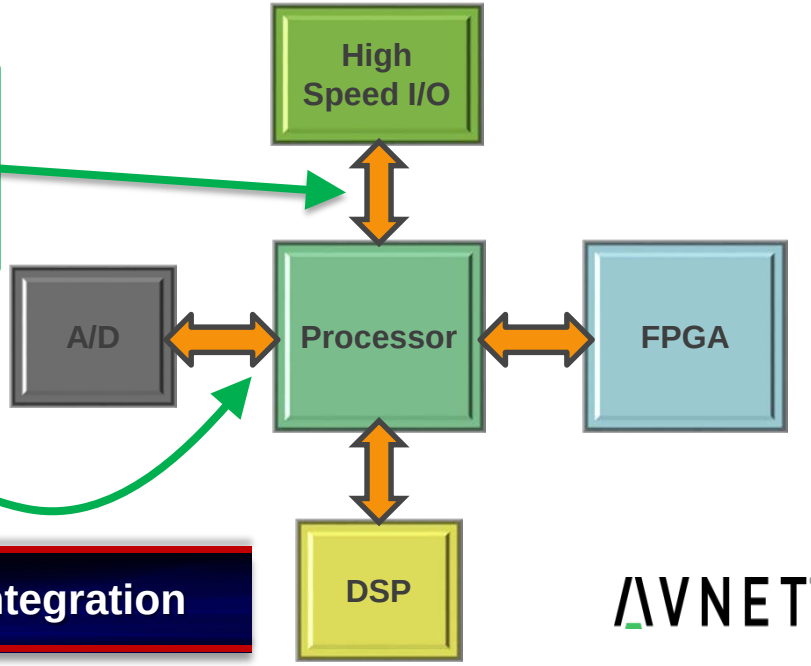
Integration = Power Reduction
- Static Power
- I/O Power

Flexible/Tunable Power Envelope
- Adjusted frequencies
- Clock gating
- ARM low Power States



Significant Power Dissipation in I/Os

EMI Reduced



Power reduction at the system level through integration

AVNET

Extensive OS, Middleware & Stack Ecosystem

Middleware and Stacks

mentor embedded
The MathWorks
QNX
iVeia
ENEA
Microsoft
Micrium
Green Hills SOFTWARE
WIND RIVER
timesys
logicBRICKS Designed by XYLON
expresslogic
Discretix trust • embedded
eSOL
montavista

Operating Systems

THREADX
T-Kernel
NUCLEUS
ENEA Linux and OSE
RTXC RTOS
WIND RIVER Linux and VxWorks
QNX
INTEGRITY
XILINX
freeRTOS
ecos
Microsoft Windows Embedded
android
µC/OS-II The Real-Time Kernel
Open Source Linux & Commercial Linux

Hypervisors

Open Kernel Labs
WIND RIVER
sierraware
open virtualization
Green Hills SOFTWARE
SYSGO EMBEDDING INNOVATIONS

Zynq-7000 Device Portfolio Summary

Scalable platform offers easy migration between devices

		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz[1]			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support[2]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[2]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports Primary Interfaces & Interrupts Only	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM	1.8Mb	2.5Mb	3.8Mb	2.1Mb	3.3Mb	4.9Mb	9.3Mb	17.6Mb	19.1Mb	26.5Mb
	(# 36Kb Blocks)	(50)	(72)	(107)	(60)	(95)	(140)	(265)	(500)	(545)	(755)
	DSP Slices	60	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades: Commercial		-1		-1			-1			-1
	Speed Grades: Extended		-2		-2,-3			-2,-3			-2
	Speed Grades: Industrial		-1, -2		-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

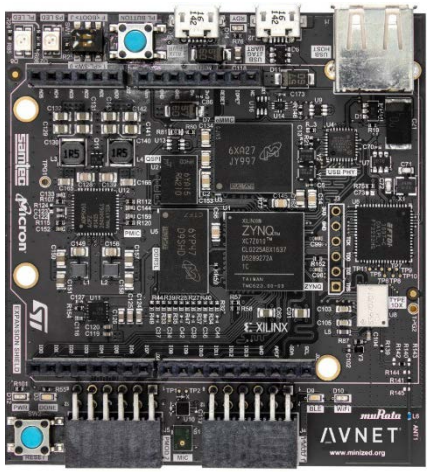
MiniZed

AVNET

Zynq Prototyping and Production Solutions

We've got you covered!

ZedBoard
MicroZed
PicoZed
MiniZed
ZC702
ZC706
Zynq MMP
Zynq Mini-ITX
EMBV
SVDK
IDK



AVNET

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

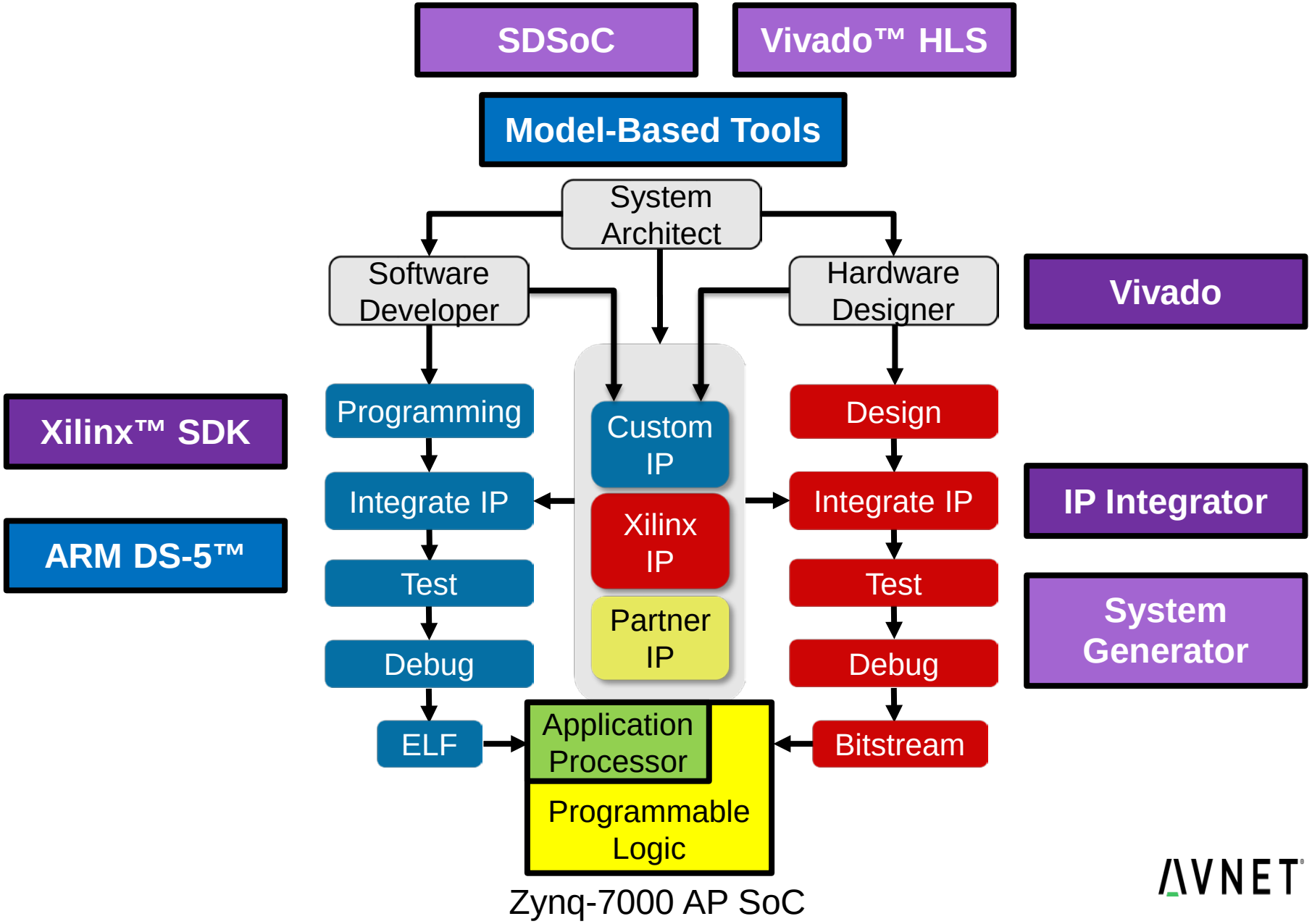
Xilinx Embedded Tool Flow

Vivado Design Suite WebPACK Edition

- **FREE!**
- Supports **seven** Zynq devices: (XC7Z007S – XC7Z7030)

Vivado Design Suite - HLx Edition Features	Vivado HL Design Edition	Vivado HL System Edition	Vivado Lab Edition	Vivado HL WebPACK (Device Limited)	Free 30-day Evaluation
Accelerating Implementation					
Synthesis and Place and Route	✓	✓		✓	✓
Partial Reconfiguration	✓	✓		Can be purchased as an option	✓
Accelerating Verification					
Vivado Simulator	✓	✓		✓	✓
Vivado Device Programmer	✓	✓	✓	✓	✓
Vivado Logic Analyzer	✓	✓	✓	✓	✓
Vivado Serial I/O Analyzer	✓	✓	✓	✓	✓
Debug IP (ILA/VIO/IBERT)	✓	✓		✓	✓
Accelerating High Level Design					
Vivado High-Level Synthesis	✓	✓		✓	✓
Vivado IP Integrator	✓	✓		✓	✓
System Generator for DSP		✓			✓
	$2995 NL $3595 FL	$3495 NL $4295 FL	Free	Free	Free

Zynq-7000 AP SoC Embedded Design Flow

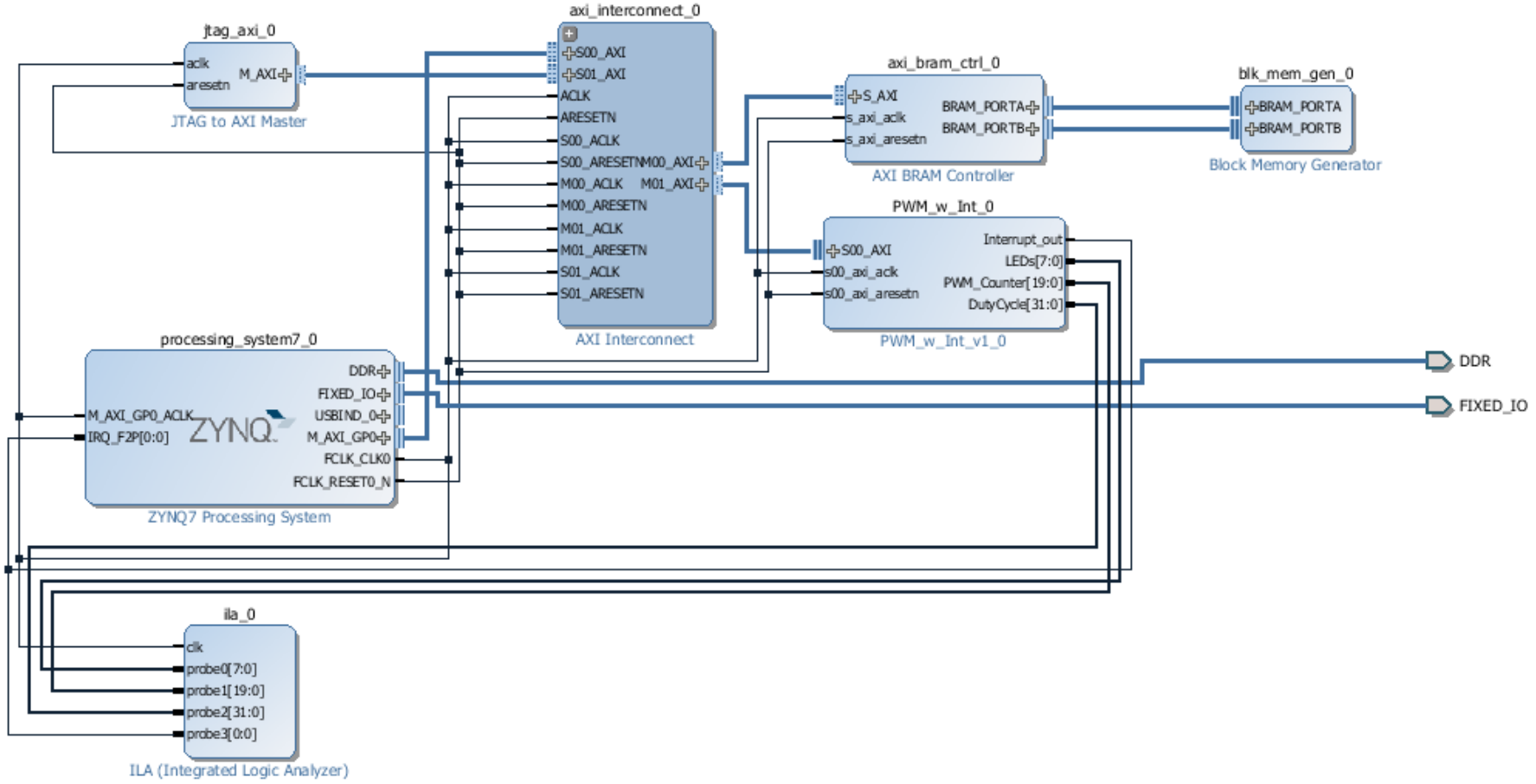


AVNET

Introducing Vivado IP Integrator (IPI)

You were doing schematics 20 years ago, and we're doing them again!

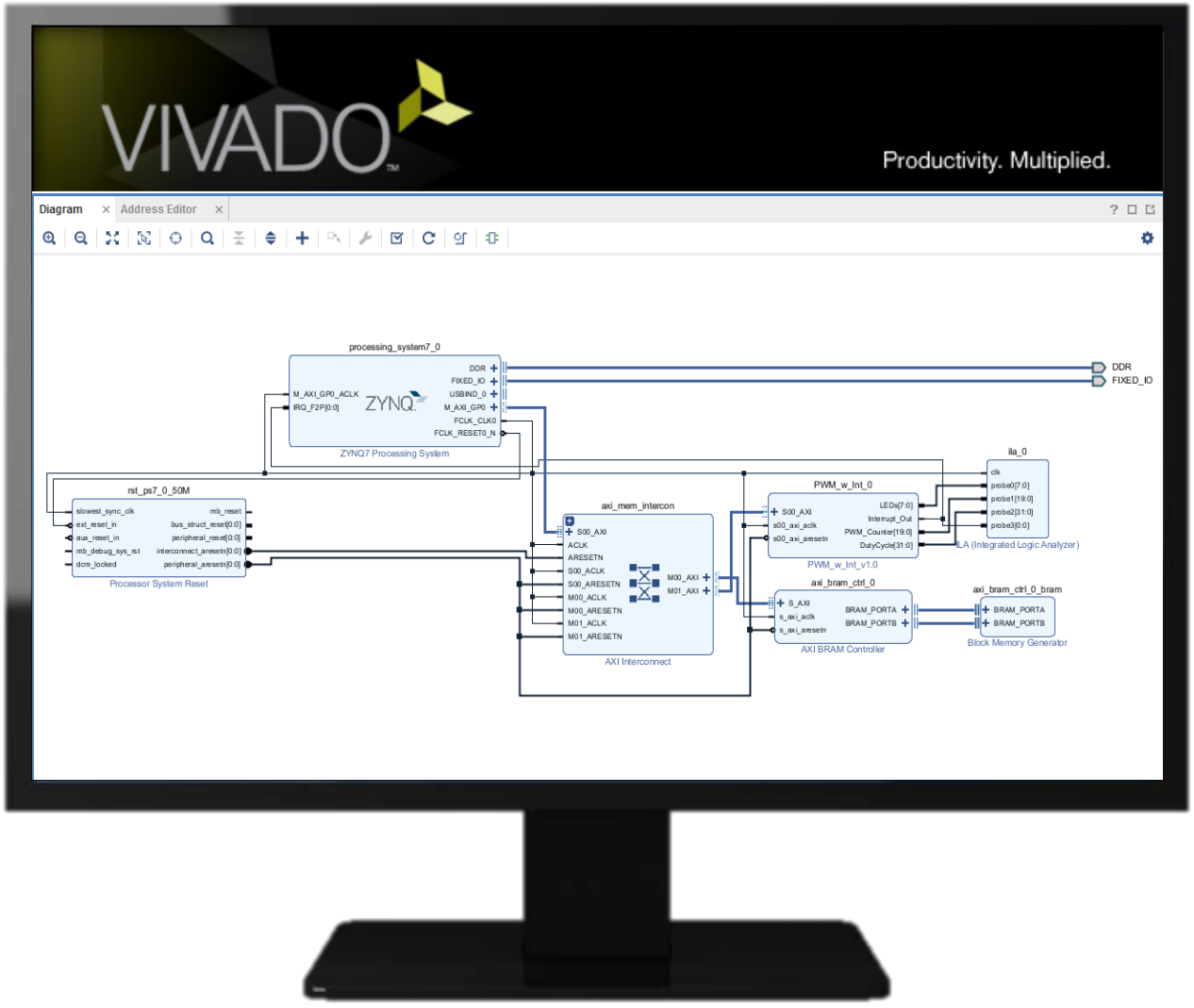
- Why? We thought designs were complex then… But now…



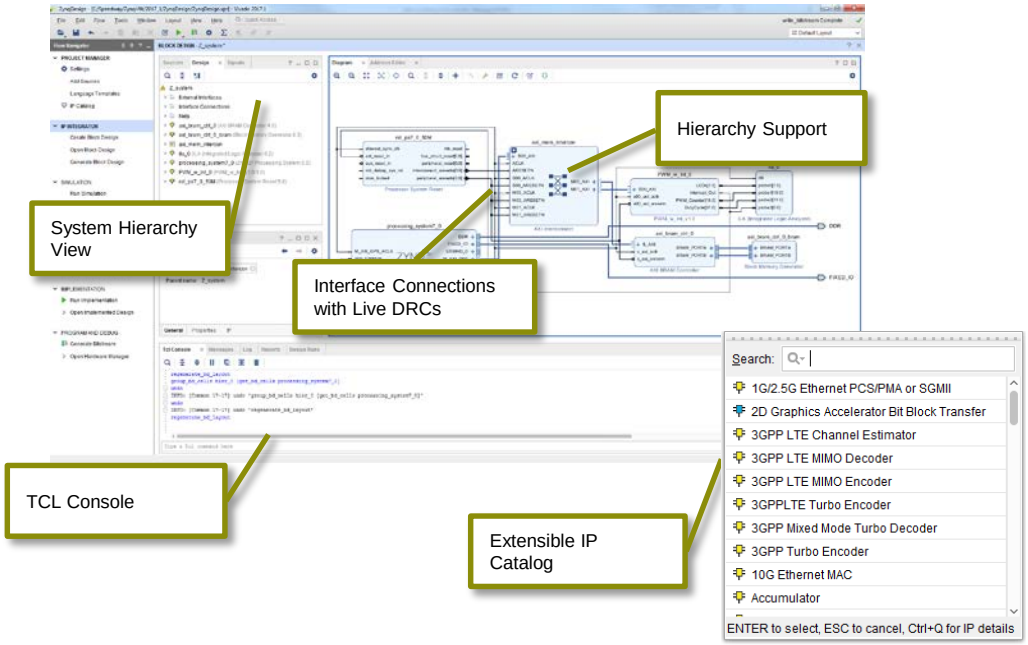
Introducing Vivado IP Integrator

Enabling an IP Centric Design Flow

- Plug-and-play IP
- Vast IP catalog
- Accelerates
  - Integration
  - Productivity
- Smarter systems
  - Embedded
  - DSP
  - Video
  - Analog
  - Networking



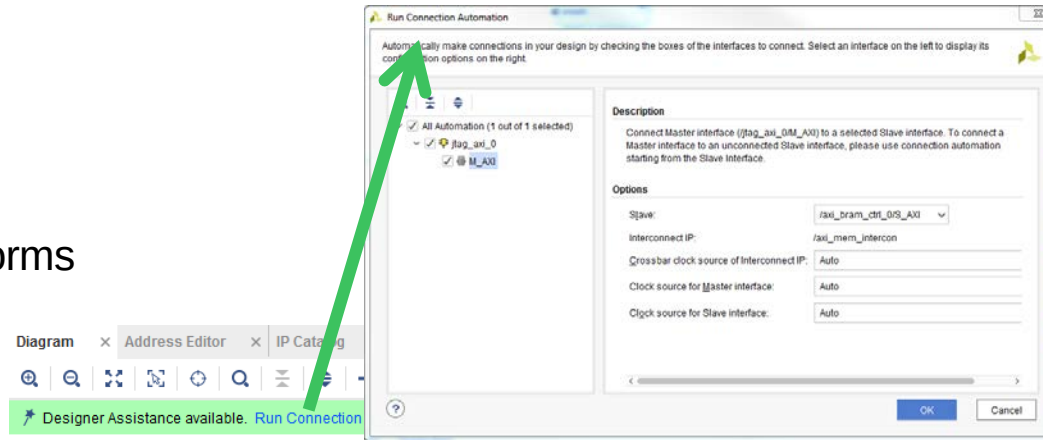
Vivado IP Integrator: Intelligent IP Integration



Correct-by-construction

- Interface level connections
- Extensible IP repository
- Real-time DRCs and parameter propagation / resolution
- Designer Assistance

- **Automated IP Subsystems**
  - Block automation for rapid design creation
  - One click IP customization
- **Board Aware**
  - Support all 7 series and Zynq Platforms



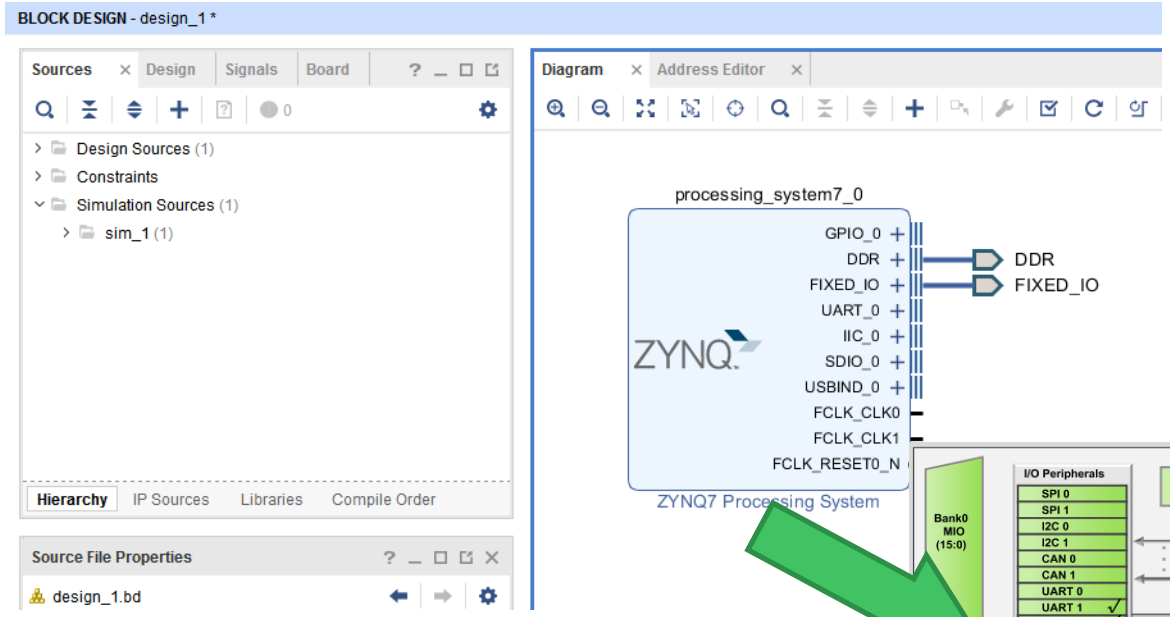
Agenda

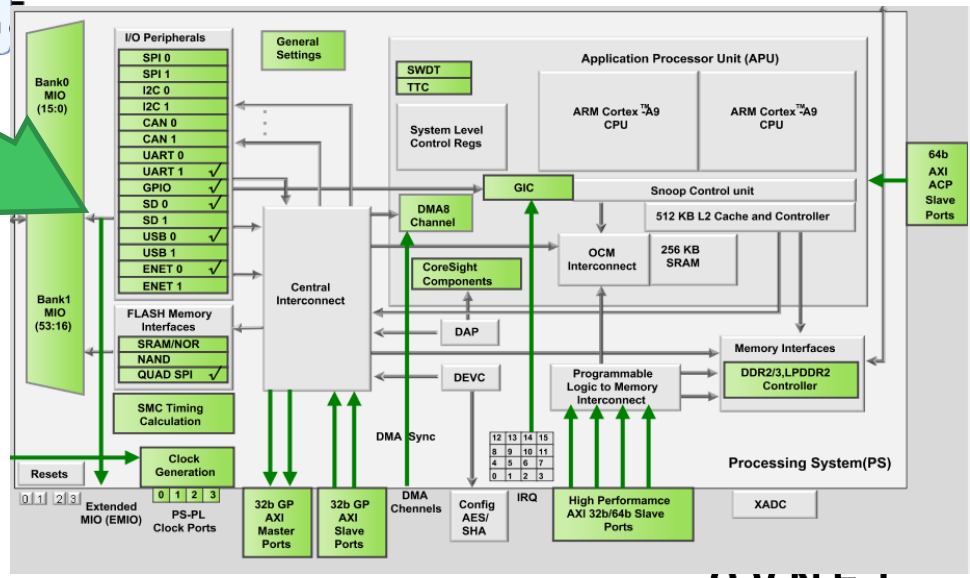
- Zynq Overview
- Xilinx Embedded Tool Flow
  - **Lab 1 – Building a Basic Zynq Design with Vivado IP Integrator**
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

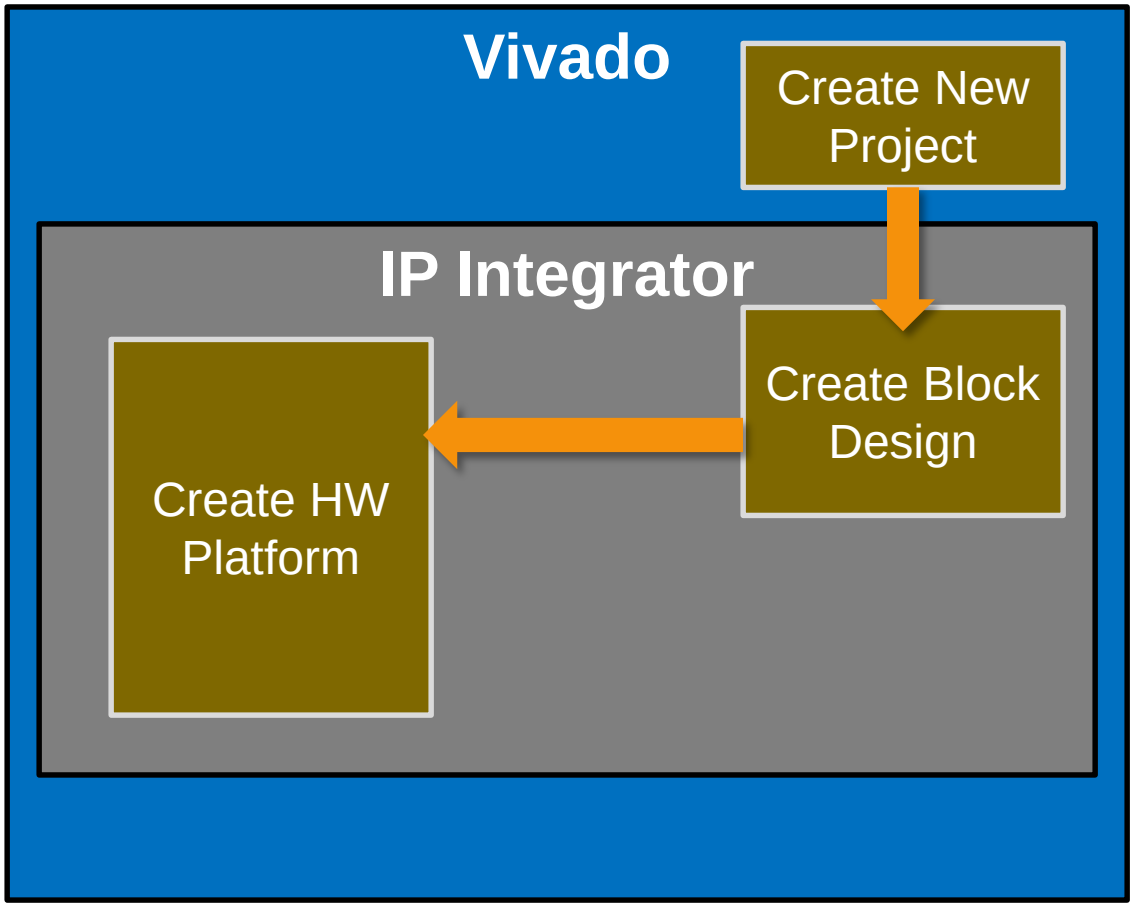
Getting Started: Adding an Embedded Source

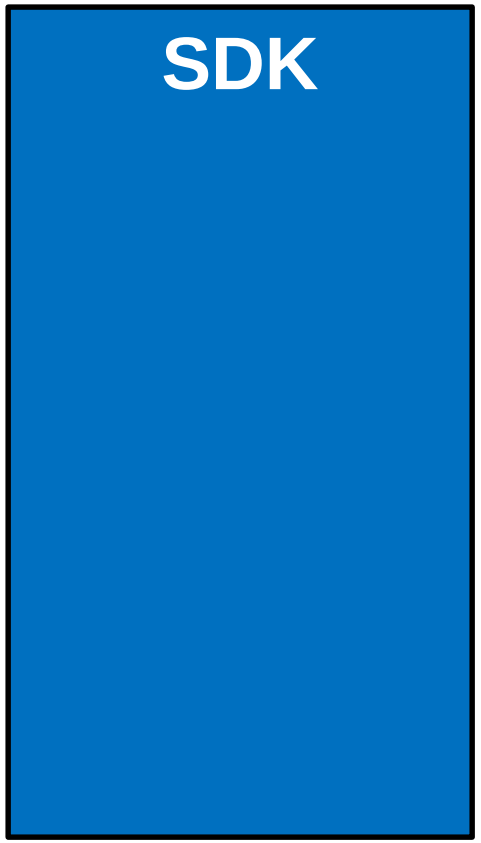
1. Create Block Design in IP Integrator
2. Customize Zynq Processor





Lab 1 – Create Basic System

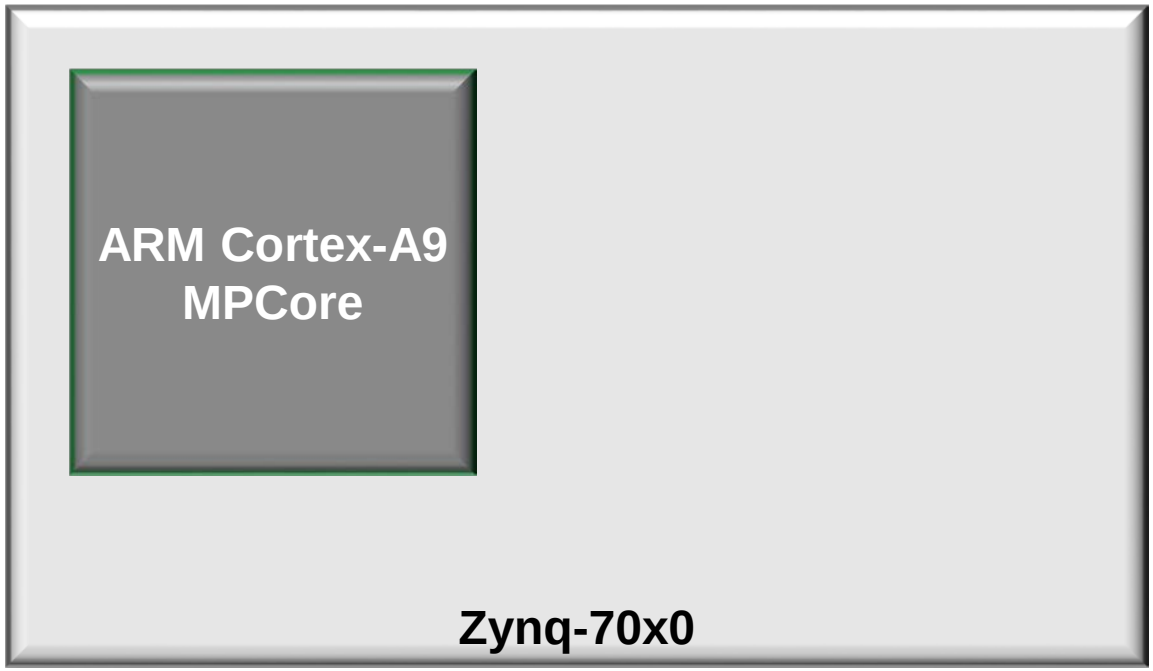




Lab 1 – Create a Basic System

Create Vivado project
Create a block design
Add an embedded source

C:/Speedway/ZynqHW/2017_1



AVNET

Questions

What device is selected? What is the Top Module Name?
- MiniZed: xc7z007sclg225
- The top module name has not been defined yet. But when created, will appear under *Design Sources*.

What are the Vivado Synthesis and Implementation Strategies?
- Both are set to Default

What other information is available in the Project Summary?
- Synthesis and Implementation status, DRC violations, timing results, device utilization statistics and power information.

List some of configurable components in the Zynq PS?
- I/O Peripherals, General Settings, Flash Memory Interfaces, Clock Generation, AXI Ports, DDR Memory Controller, and more.

The I/O Peripherals are connected to external I/O through the I/O Mux (MIO), how many I/O are available in the MIO?
- 54 Total (except in CLG225 package). 16 in MIO Bank 0 and 38 in MIO Bank 1

Agenda

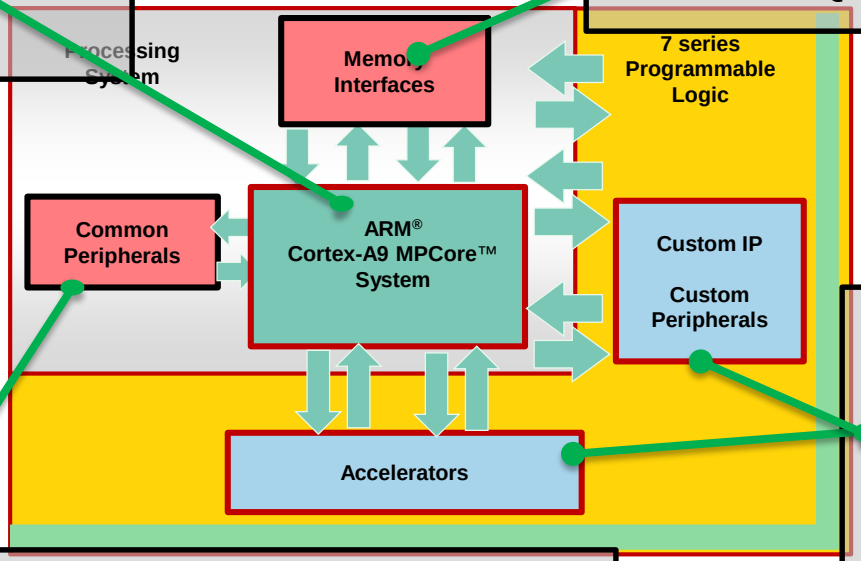
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Complete ARM-Based System

Processing Core Complex
- Dual or Single ARM Cortex A9
- NEON Extensions
- FPU
- Up to 1GHz

High Bandwidth Memory
- 256KB OCM
- 32KB L1 / 512KB L2 Cache
- Memory Controllers
  - DDR3/L, DDR2, LPDDR2
  - 2xQSPI, NOR, NAND

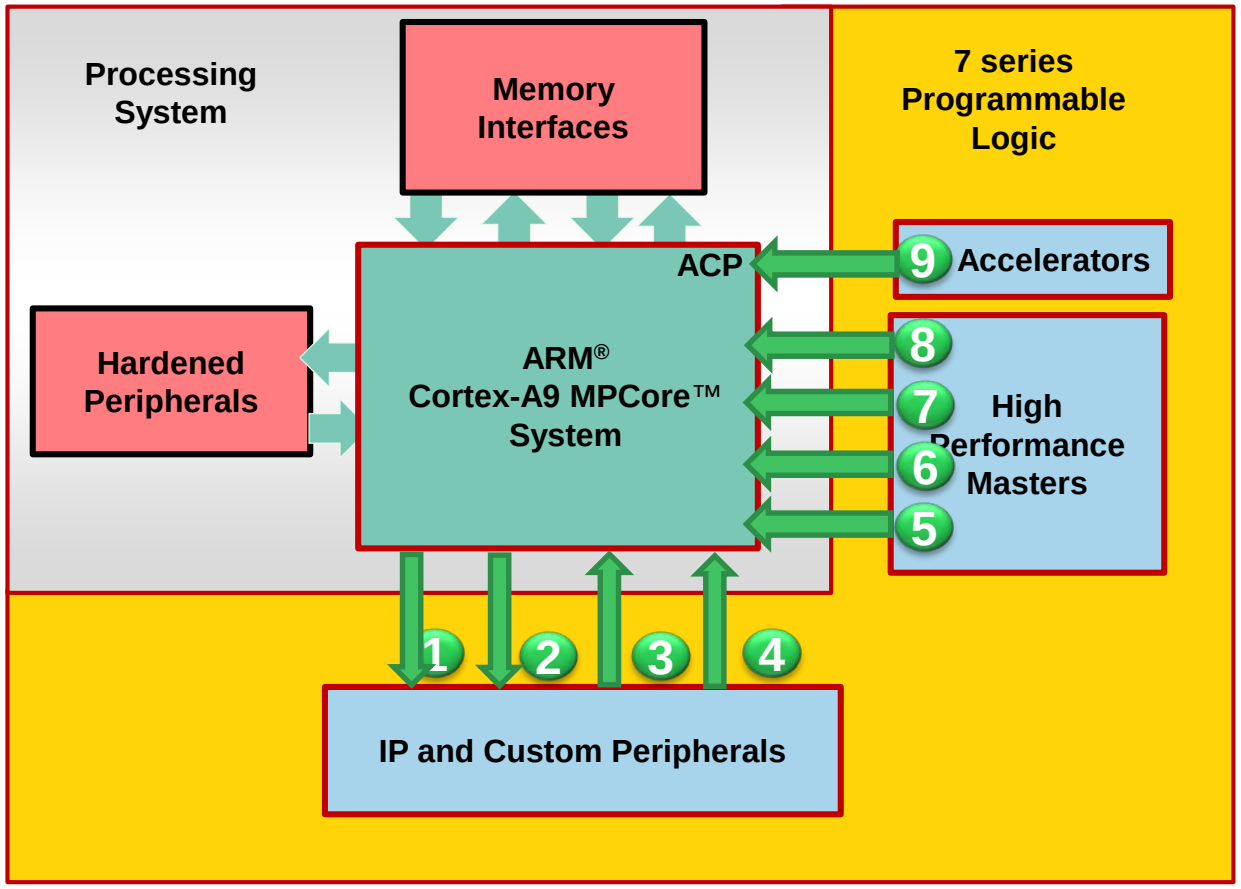


AMBA AXI
- High Bandwidth between PS-PL
- ACP Port
  - Hardware Acceleration

Integrated Memory Mapped Peripherals
- 2x USB 2.0 OTG w/DMA (High Speed 480Mbps)
- 2x TEMAC w/DMA
- 2x SD/SDIO w/DMA
- 2x UART, 2x CAN, 2xI2C, 2x SPI, GPIO

AVNET

Zynq-7000 AP SoC Architecture Overview



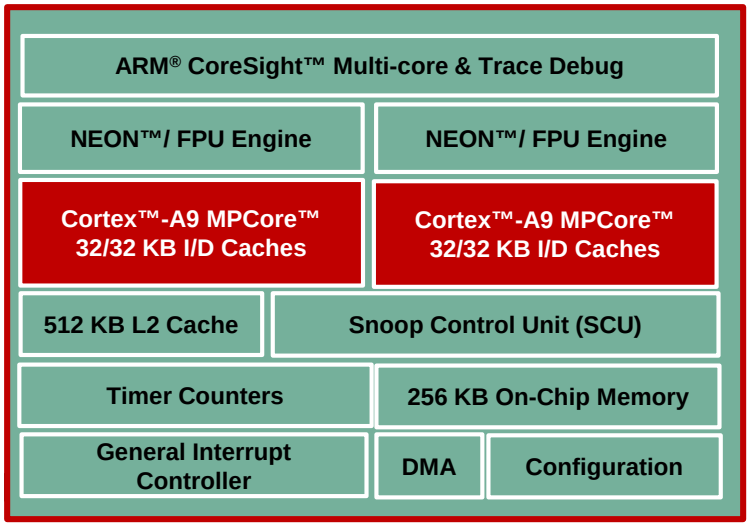
9 Independent PS-to-PL Interface
~100Gbps of Bandwidth

AVNET

ARM Cortex-A9 Processor and L1 Caches

ARM Cortex-A9 processor key features

- 2.5 DMIPS/MHz or 1128 CoreMark™ per core (@667MHz)
- Up to 1GHz operation
- Harvard architecture, independent
  - 64-bit data
  - 64-bit instruction
- Little endian support for both instruction and data



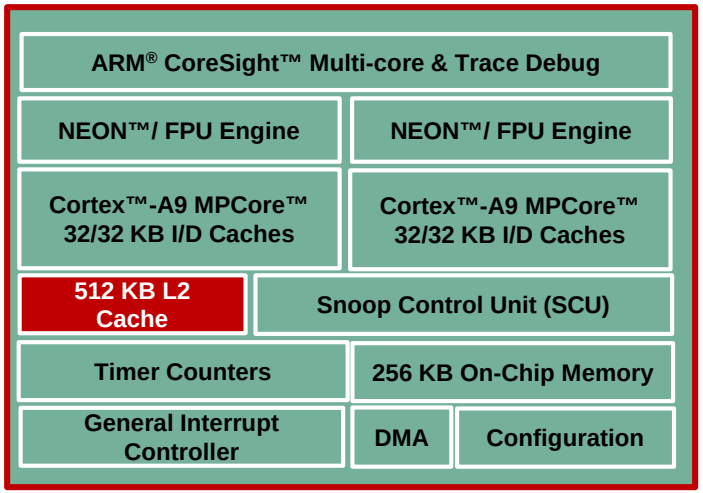
ARM Cortex-A9 processor L1 cache key featu

- 32KB Instruction and 32KB Data cache
- The cache line length is eight words (32 bytes)
- All L1 caches support parity
- 4-way set associative, write-back

AVNET

L2 Cache Controller

High performance L2 cache controller
- 512KB of cache
- Supports lockdown by line for routines that might need low latency
- 8-way set associative
- Parity support
- Write-through and write-back support



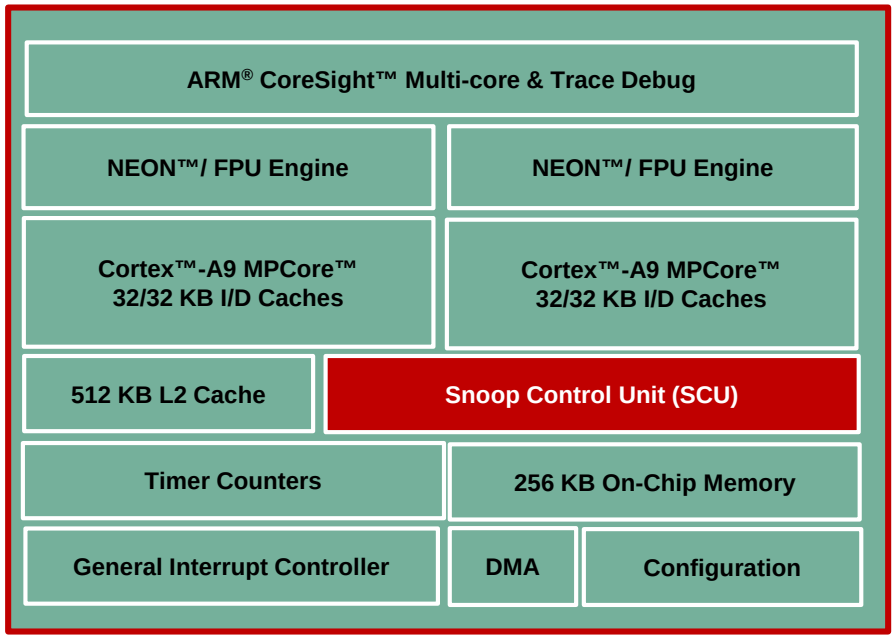
L2 cache controller AXI interfaces
- One AXI master (M) interface for the DDR controller
- One AXI master interface for all slave devices in the PL and PS
- One AXI slave (S) interface for the Snoop Control Unit

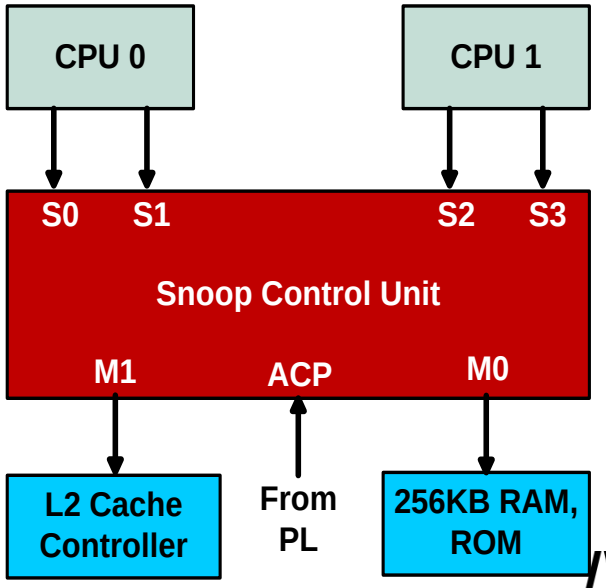
AVNET

Snoop Control Unit (SCU)

The Snoop Control Unit (SCU) connects two Cortex-A9 processors to the system memory/peripherals via AXI interfaces in Zynq-7000 AP SoC

- Provides L1 data cache coherency between the Cortex-A9 processors and the Accelerator Coherence Port (ACP)
- Arbitrates between Cortex-A9 processors requesting L2 cache accesses
- Provides access to the on-chip ROM and RAM
- Manages Accelerator Coherence Port (ACP) accesses





Arrow shows direction of the Master

AVNET

Cortex-A9 NEON™ Processing

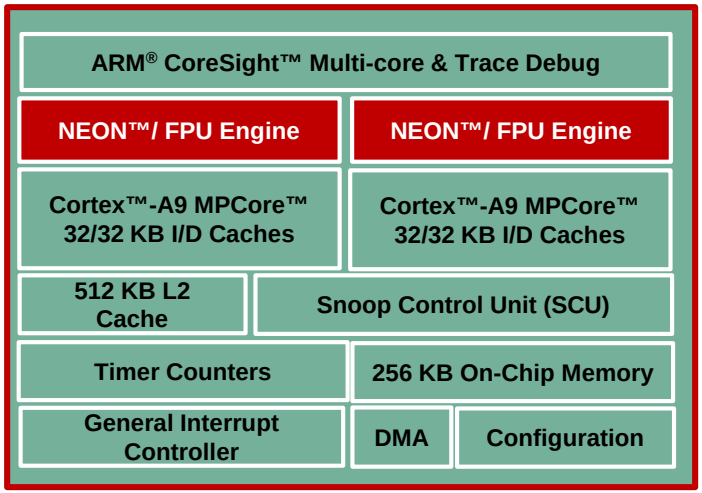
A co-processor that extends the ARM instruction set targeted at audio, video, 3-D graphics, image and speech processing applications

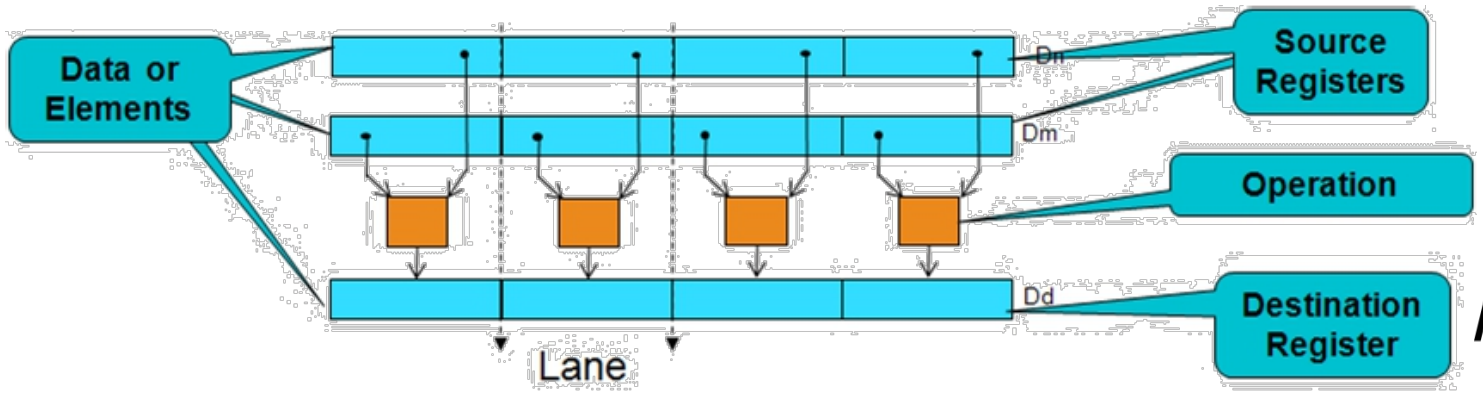
NEON SIMD instruction supports multiple
- 8, 16, 32, and 64-bit integer data
- 32-bit floating-point data (via **FPU**)

Instructions need fewer cycles than ARM
- NEON typically half

Disabled at reset, enabled via software

Requires Linaro or ARM compiler, or specialized Library
- http://www.xilinx.com/support/documentation/application_notes/xapp1206-boost-sw-performance-zynq7soc-w-neon.pdf

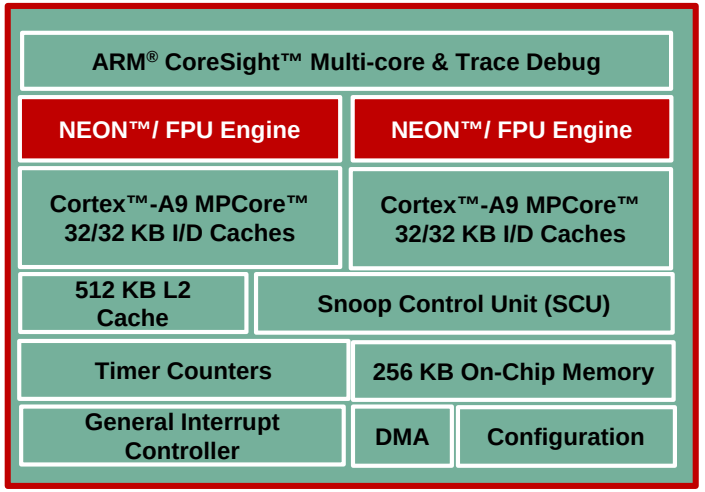




AVNET

FPU Engine

- Extension to NEON sub-processor (not independent component)
- High performance single/double precision floating-point operations
  - VFP v3 FPU (no vector feature)
  - New half-precision conversions (FP16)
- Compliant with IEEE-754 standard with noted exceptions
  - Refer to the UG585 for more information
- Register set shared with NEON
- 2 MFLOPS/MHz performance

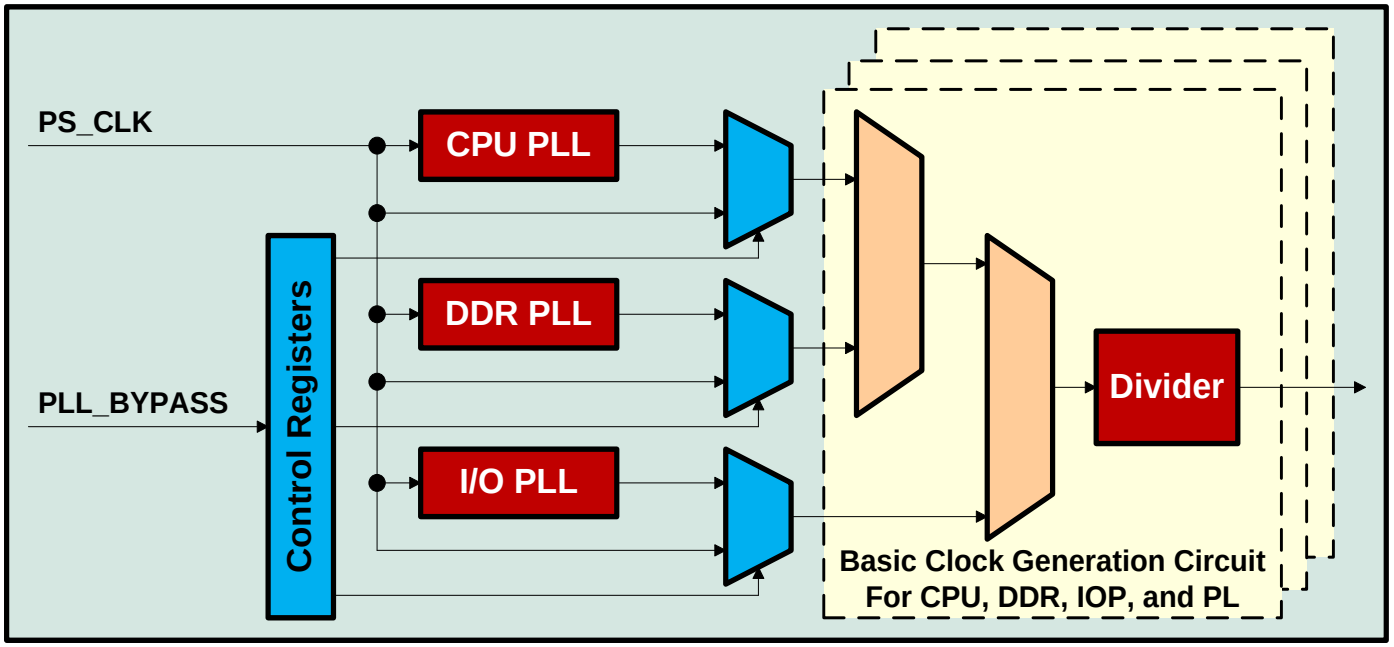


AVNET

Zynq-7000 AP SoC Clock Generation Module

Generates clocks for the CPU, DDR, PL, and all common peripherals

- PS_CLK is the PS reference clock input and is required to be 30-60 MHz
  - A 33.333 MHz reference frequency is recommended
- Each clock generation circuit has a PLL source selection multiplexer followed by a programmable divider
- Four general-purpose clock outputs are generated for the PL
- Xilinx tools provide means to set the clock frequencies

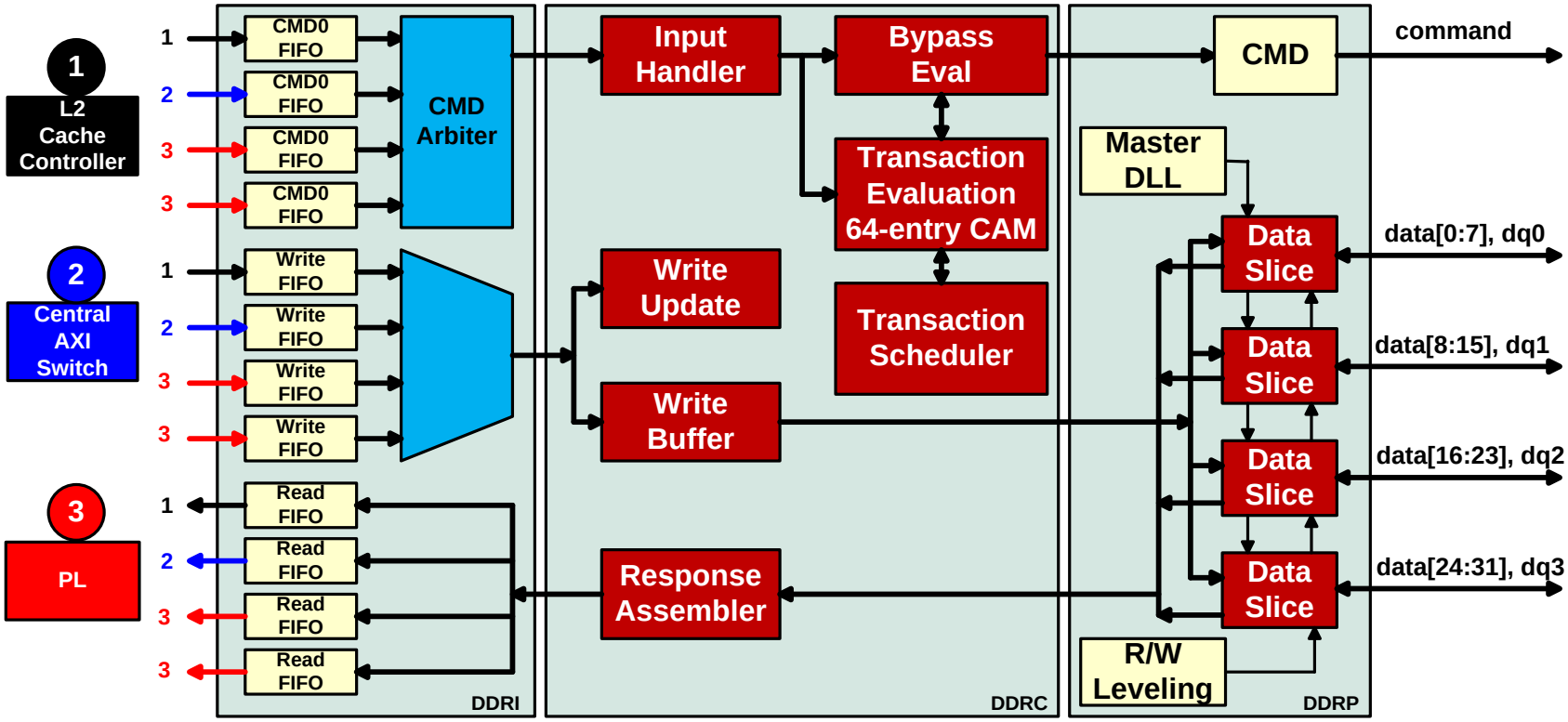


AVNET

DDR Memory Interface

DDR memory interface features consist of

- Support for LPDDR2, DDR2, and DDR3/L (400, 400, 533MHz in a -1 device)
- Configurable 16-bit and 32-bit external DDR data bus width
- ECC with one bit error correction and two bit error detection (16-bit only)
- One external chip select, no DIMM or parity support



Checkpoint!

What is the cost of the tools to develop a Zynq design?

Nothing, Tools are FREE for 7007S-7030 devices

How many AXI interfaces exist between the PS and PL?

Nine – 4 GP, 4 HP and 1 ACP

What is the max CPU frequency?

1 GHz

What is the recommended input clock frequency?

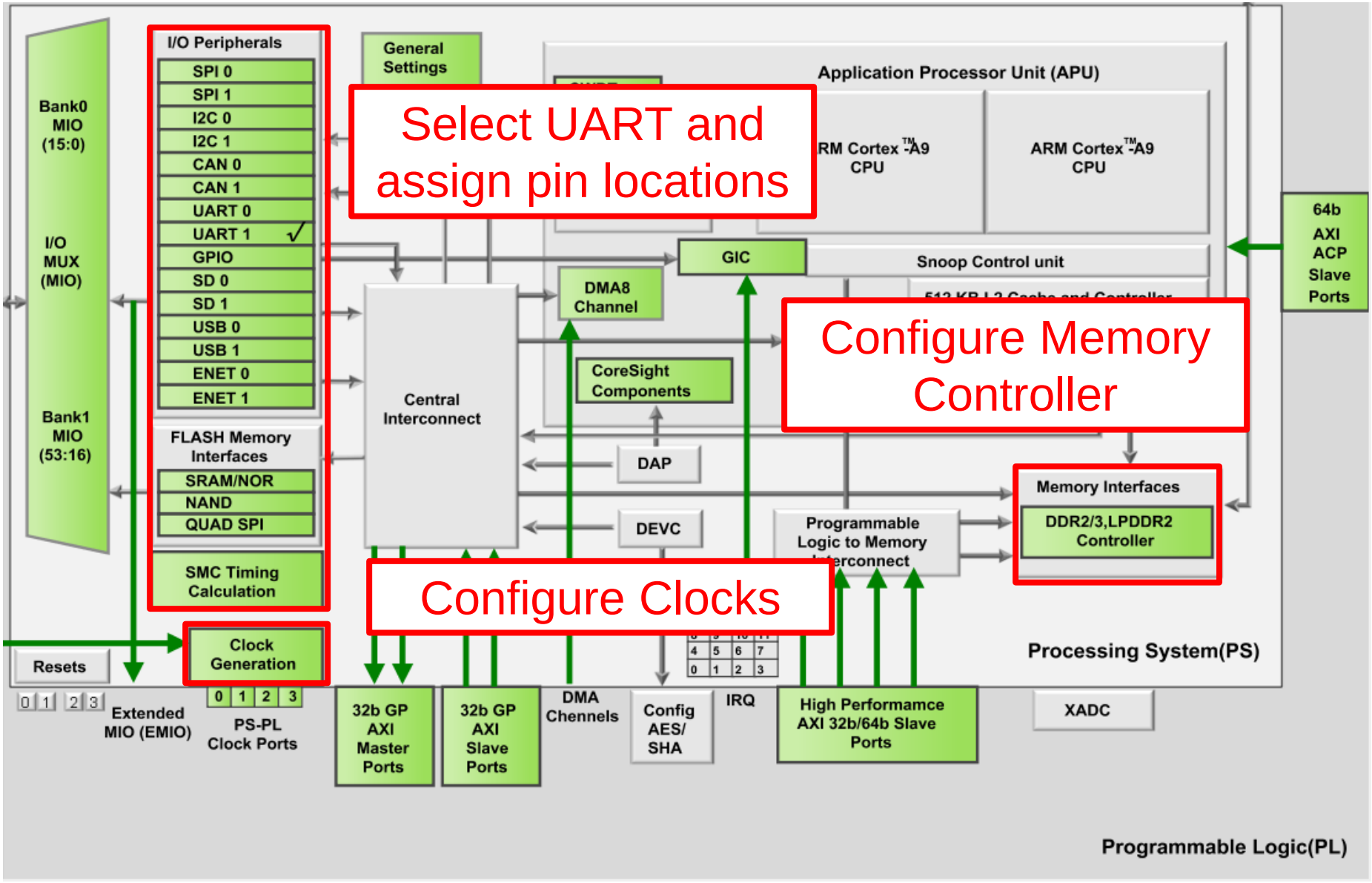
33.333 MHz

Agenda

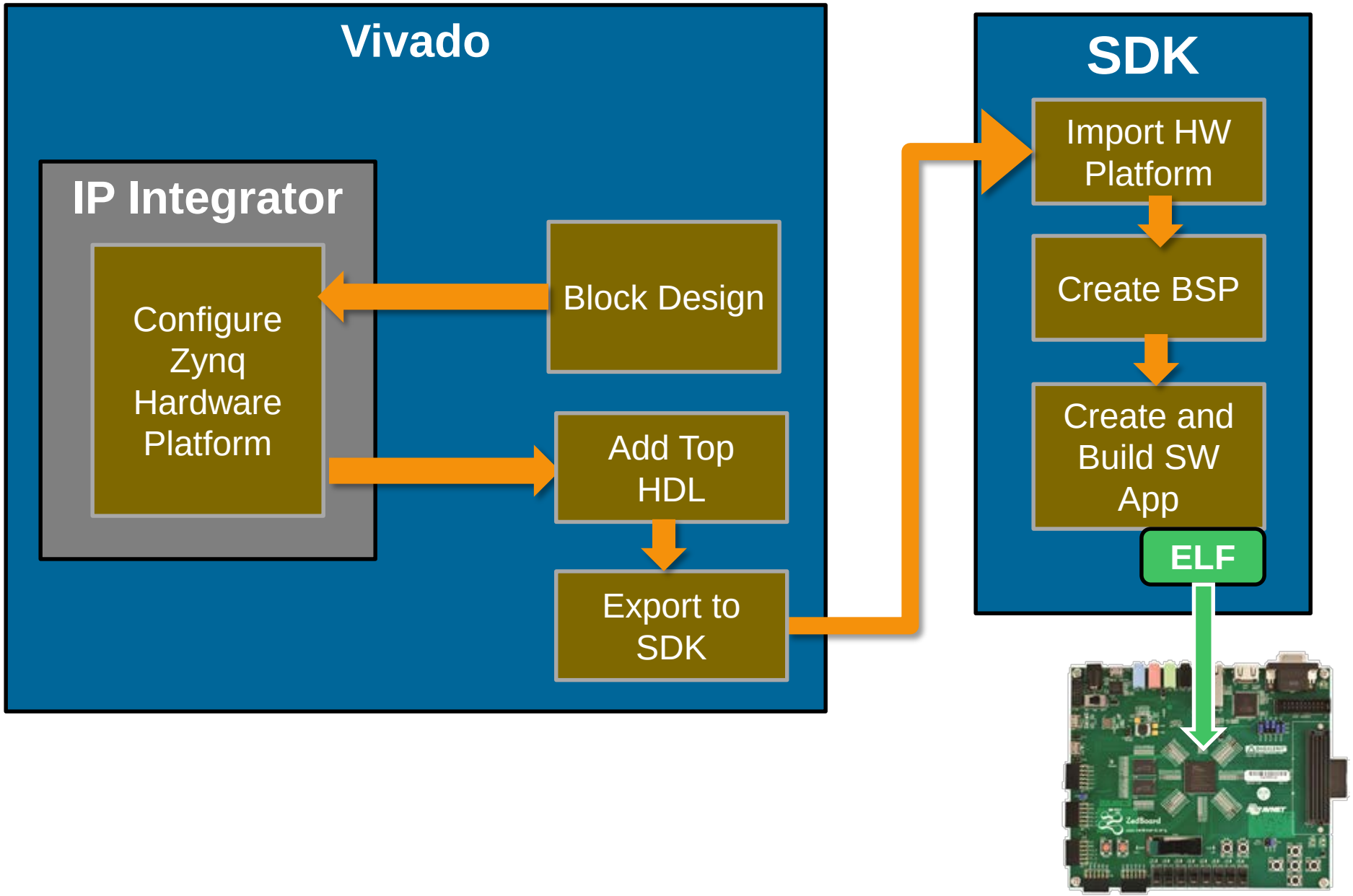
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - **Lab 2 – Detailed Processor Configuration**
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

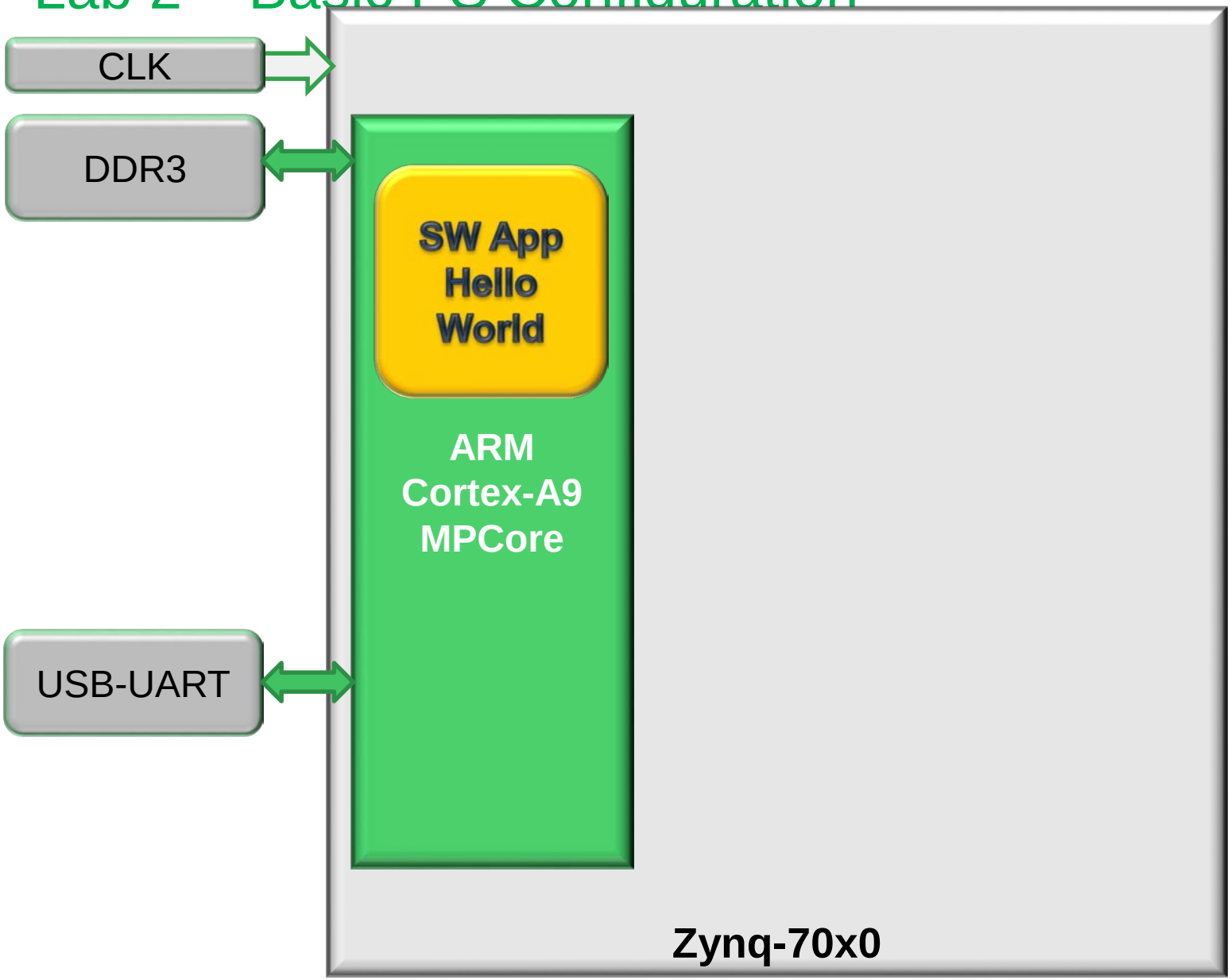
Lab 2 - PS Configuration



Lab 2 – Basic PS Configuration



Lab 2 – Basic PS Configuration



AVNET

Questions

What do you think is the purpose of EMIO?

- EMIO provide PL and PL I/O access to PS peripherals.

Why are the Peripherals not listed alphabetically in the I/O Peripherals Configuration tool?

- The peripherals are listed in order of priority. Peripherals on the top of the list generally need to be connected first and less MIO options.

Extra Credit: If the Modem Signals are used with one of the UART peripherals, where must they be mapped?

- EMIO

Where can the DDR interface speed be set?

- In the Clock Configuration window or the DDR Configuration window.

Where did Vivado get the Memory Part Configuration Settings? Where would you get them for a custom part?

- Vivado has a preset library of memory part details. With each release of Vivado, Xilinx adds new memory devices to this library.
- If you use a custom part, these values must be extracted from the specific DDR datasheet.

Questions

What is the maximum speed the DDR3 interface can run at?
Extra Credit: What is the slowest?

- 533MHz. Slowest? It depends on the DDR3 memory device. Even though Vivado lists the slowest speed as 10MHz, the DDR3 memory device's internal PLL may not be able to run that slow. When a known memory device is selected, Vivado prevents users from setting a speed to slow. In this case, 303MHz.

Does the Hello World C source include Zynq initialization?

- No

How did the Zynq get initialized in this Hello World experiment?

- When you created the Run Configuration, there was a Device Initialization tab. Here there is a field to point to an initialization TCL file. This is set by default to the ps7_init.tcl file that was created as part of the Export to SDK. Inside this TCL, you will find a number of XMD commands that initialize all the registers exactly how you specified in XPS.

Agenda

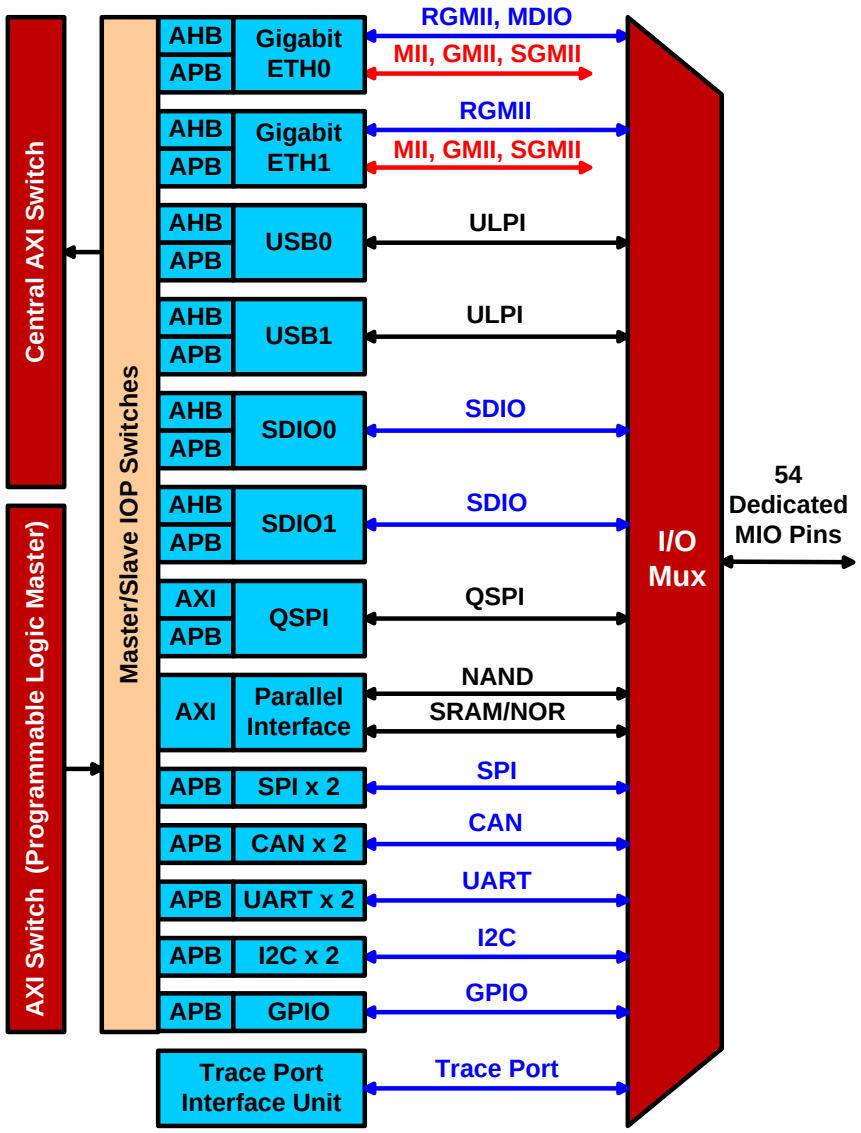
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

Zynq-7000 AP SoC Common Peripherals

The PS common peripherals

- Gigabit Ethernet w/DMA (2 ports)
- USB 2.0 w/DMA (2 ports)
- SD/SDIO w/DMA (2 ports)
- CAN (2 ports)
- I2C (2 ports)
- UART (2 ports)
- SPI (2 ports)
- NAND (x8/x16)
- NOR/SRAM (x8)
- Quad SPI
- Up to 118 GPIO ports (54 MIO + 64 EMIO)

Peripheral I/O can be routed to the PS MIO or PL Extended MIO (EMIO)



AVNET

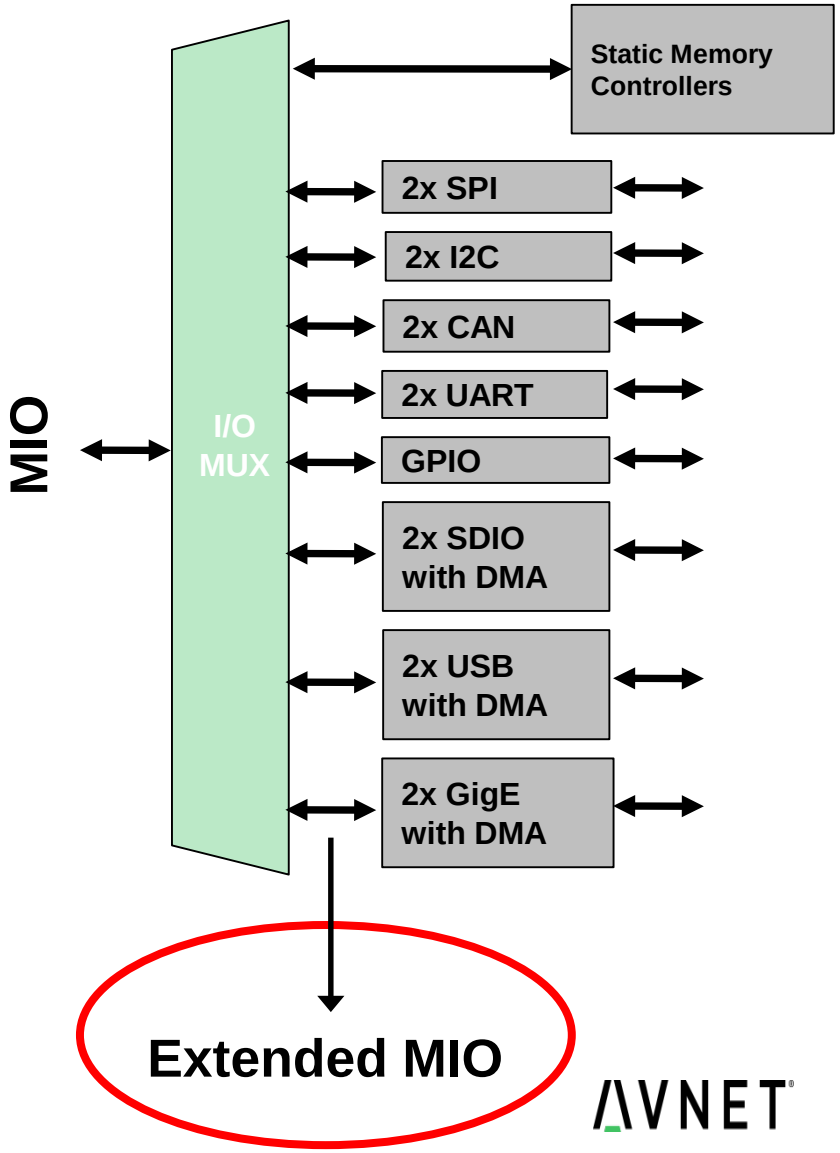
Zynq-7000 AP SoC Common Peripherals I/O Pins

Peripheral I/O connected to MIO only
- USB0 and USB1 ULPI
- QSPI
- NAND Flash
- SRAM/NOR Flash
- ETH0 or ETH1 via RGMII

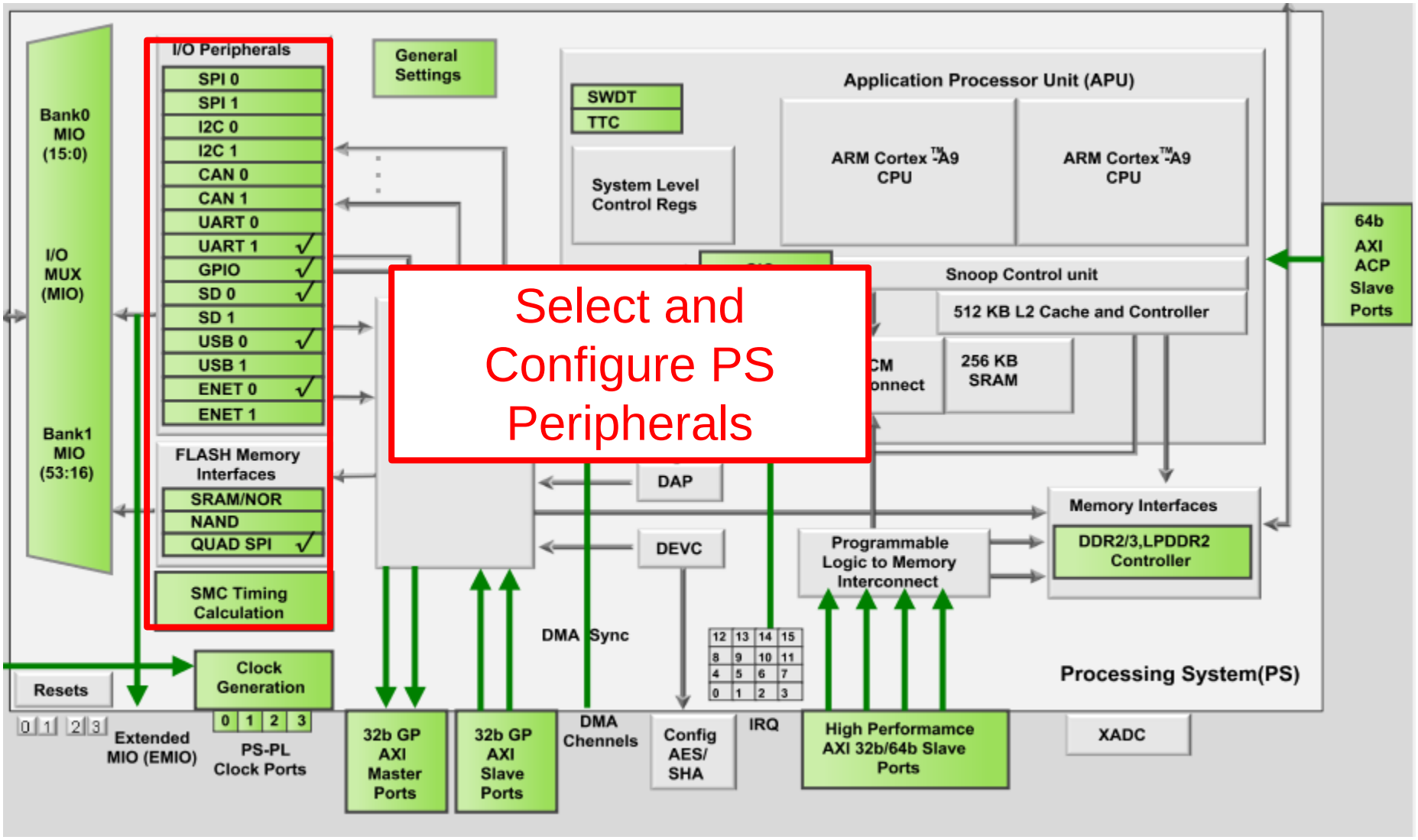
Peripheral I/O connected to MIO or EMIO
- SDIO0 and SDIO1
- CAN
- I2C
- UART
- SPI
- GPIO
- Trace Port
- Processor JTAG

Peripheral I/O connected to EMIO only
- ETH0 and ETH1 MII, GMII, SGMII
  - RGMII with PL-shim
- UART Modem Signals

MIO[8:7] are output only



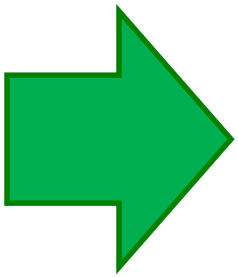
PS MIO Configuration

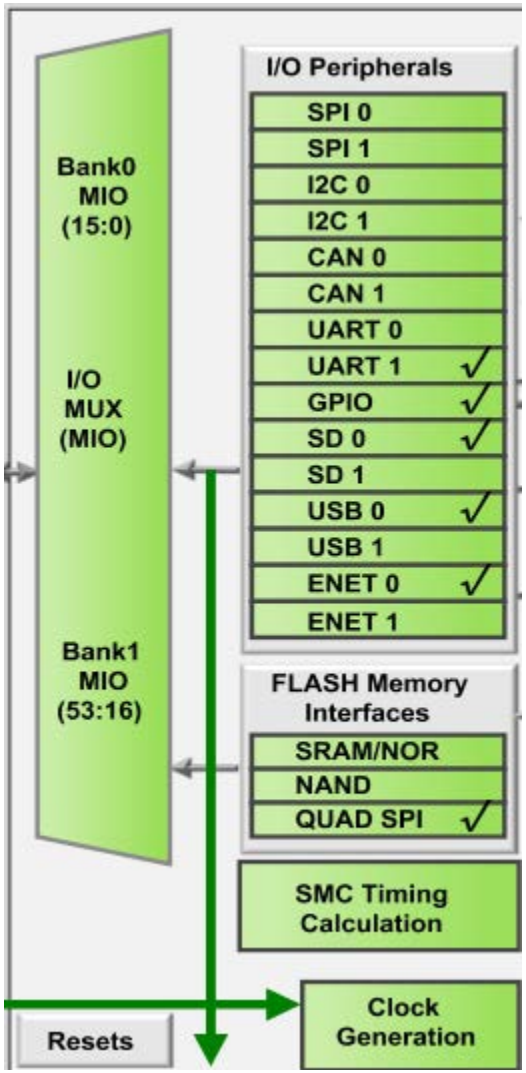


AVNET

Select Peripherals

Peripheral	IO
Memory Interfaces	
☑ Quad SPI Flash	MIO 1 .. 6
☐ SRAM/NOR Flash	
☐ NAND Flash	
I/O Peripherals	
☑ ENET 0	MIO 16 .. 27
☐ ENET 1	
☑ USB 0	MIO 28 .. 39
☐ USB 1	
☑ SD 0	MIO 40 .. 45
☐ SD 1	
☐ UART 0	
☑ UART 1	MIO 48 .. 49
☐ I2C 0	
☐ I2C 1	
☐ SPI 0	
☐ SPI 1	
☐ CAN 0	
☐ CAN 1	
GPIO	





AVNET

Zynq Address Map

Address Range	CPUs & ACP	AXI_HP	Other Bus Masters	Description
0000_0000 to 000F_FFFF	DDR/OCM	OCM	OCM	SCU and OCM
0010_0000 to 3FFF_FFFF	DDR	DDR	DDR	DDR, All interconnect masters
4000_0000 to 7FFF_FFFF	PL		PL	General Purpose Port #0 to the PL, M_AXI_GP0
8000_0000 to BFFF_FFFF	PL		PL	General Purpose Port #1 to the PL, M_AXI_GP1
E000_0000 to E02F_FFFF	IOP		IOP	MIO Peripheral registers
E100_0000 to E5FF_FFFF	SMC		SMC	Static Memory Controller (NAND, NOR, SRAM)
F800_0000 to F800_0BFF	SLCR		SLCR	System Level Control Registers
F800_1000 to F880_FFFF	PS		PS	PS System registers
F890_0000 to F8F0_2FFF	CPU			CPU Private Registers
FC00_0000 to FDFF_FFFF	QSPI		QSPI	Quad-SPI linear address for linear mode
FFFC_0000 to FFFF_FFFF	OCM	OCM	OCM	OCM Memory (Mapped High)

AVNET

Zynq Address Map

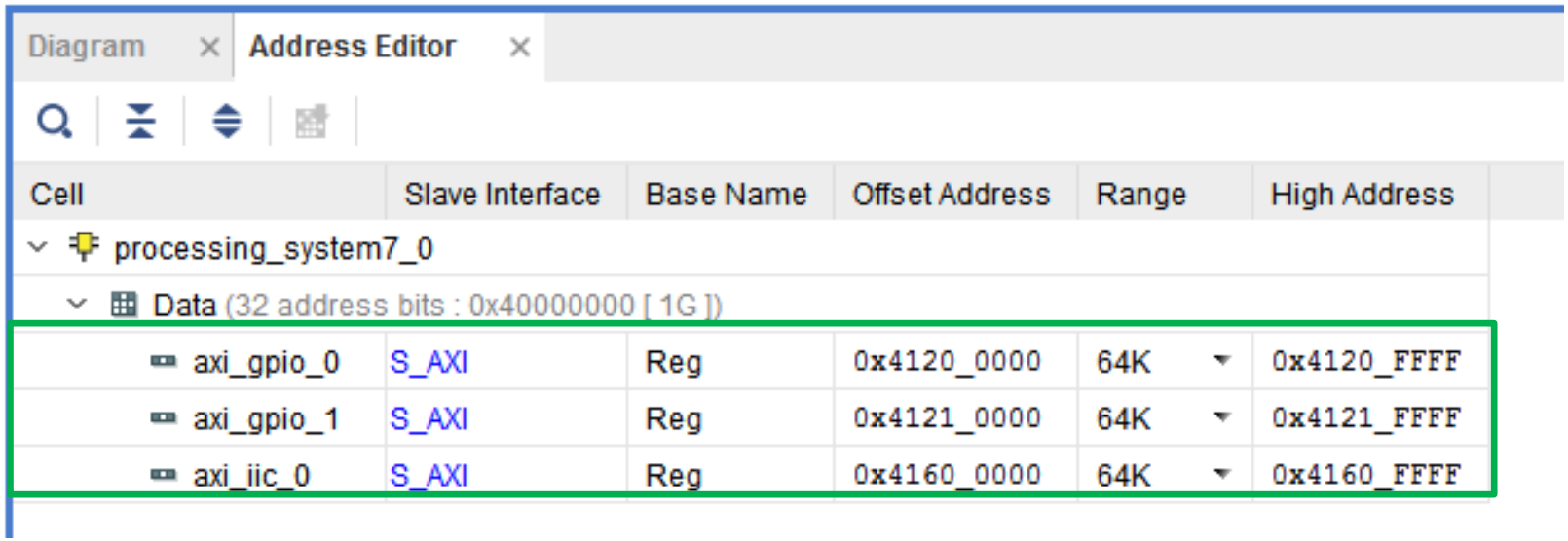
Address Range	CPUs & ACP	AXI_HP	Other Bus Masters	Description
0000_0000 to 0				
0010_0000 to 3				
4000_0000 to 7				
8000_0000 to B				
E000_0000 to E				
E100_0000 to E				
F800_0000 to F				
F800_1000 to F				
F890_0000 to F				
FC00_0000 to F				
FFFC_0000 to F				

Address Base	MIO Peripheral
E000_0000, E000_1000	UART Controllers 0, 1
E000_2000, E000_3000	USB Controllers 0, 1
E000_4000, E000_5000	I2C Controllers 0, 1
E000_6000, E000_7000	SPI Controllers 0, 1
E000_8000, E000_9000	CAN Controllers 0, 1
E000_A000	GPIO Controller
E000_B000, E000_C000	Ethernet Controllers 0, 1
E000_D000	Quad-SPI Controller
E000_E000	Static Memory Controller (SMC)
E010_0000, E010_1000	SDIO Controllers 0, 1

AVNET

Address Mapping Sub-system View

Vivado automatically assigns addresses for PL-based peripherals



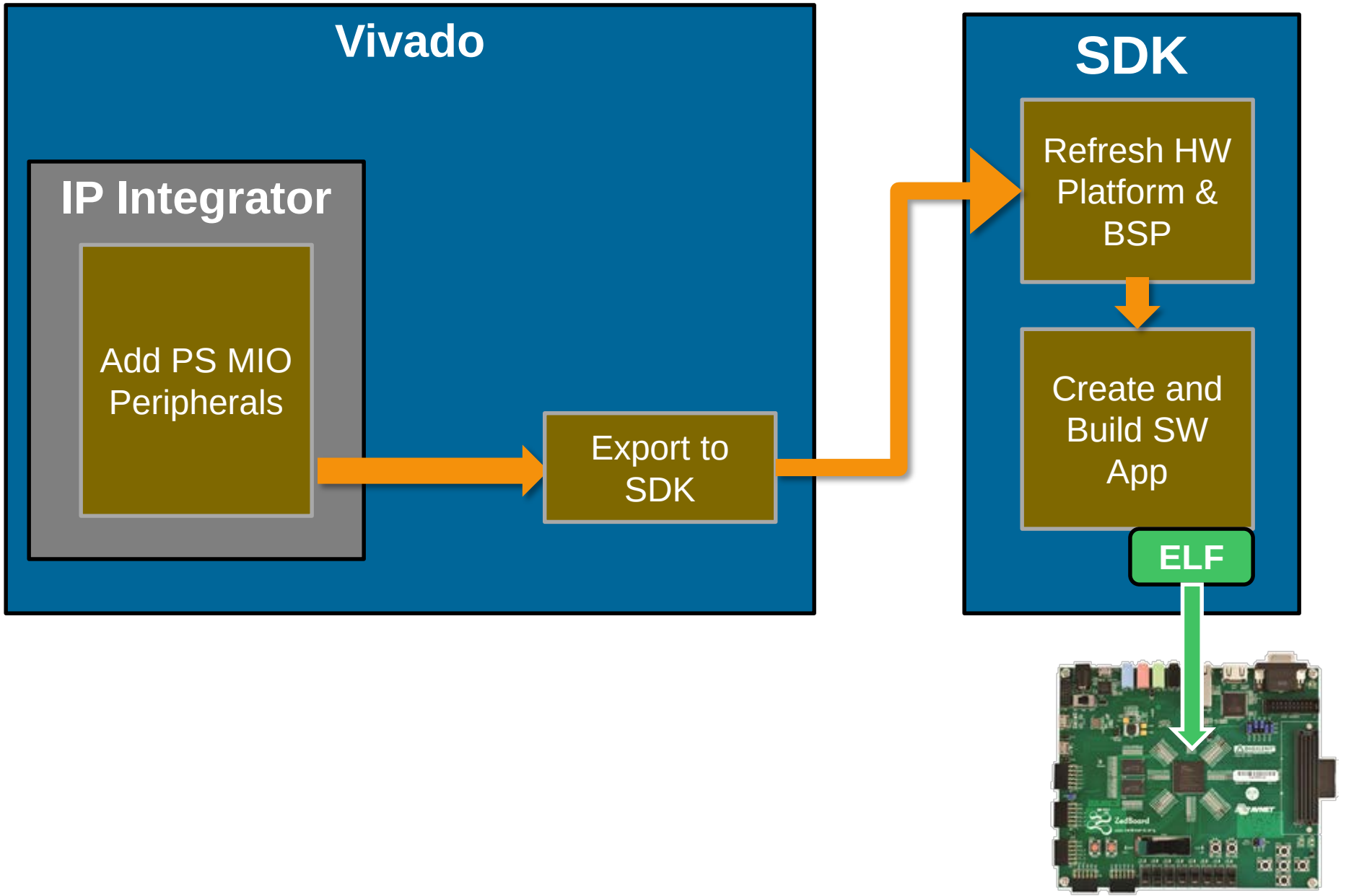
AVNET

Agenda

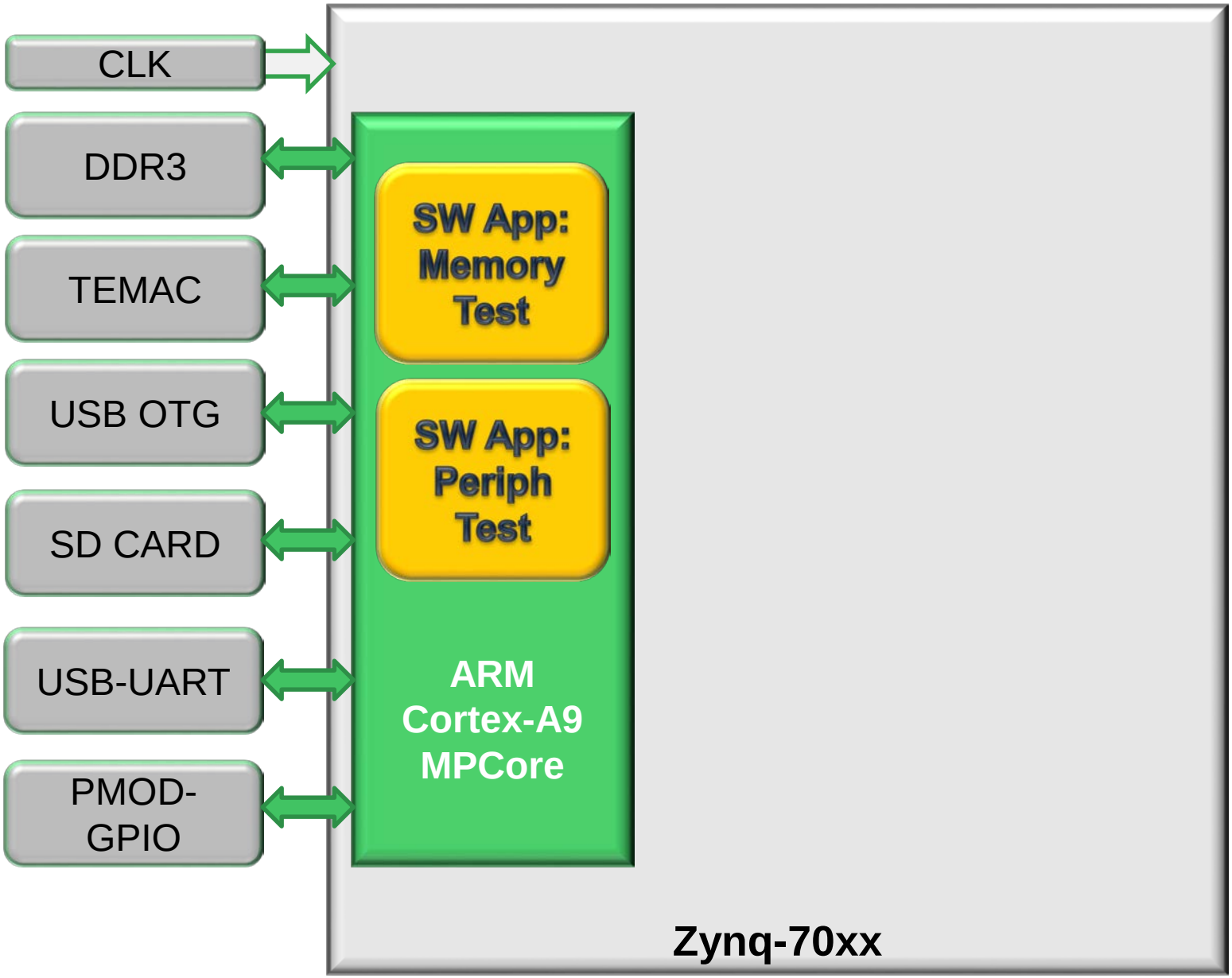
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1 – Building a Basic Zynq Design with Vivado IP Integrator
- Zynq Processor Overview
  - Lab 2 – Detailed Processor Configuration
- Peripherals, Peripherals and more Peripherals!
  - **Lab 3 – Adding MIO Peripherals**
- The Power of TCL
  - Lab 4 – TCL Overview
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5 – Adding an AXI PL Peripheral
- Zynq PS DMA Controller
  - Lab 6 – Utilizing PS DMA
- Creating Custom IP
  - Lab 7 – Adding Custom IP to the Vivado IP Catalog
- Vivado's Hardware Manager
  - Lab 8 – Hardware Debugging Zynq Designs
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Lab 3 – Add PS MIO Peripherals



Lab 3 – Add MIO Peripheral & Create Test App



AVNET

Questions

Why is EMIO not an option for connecting USB 0?

- Due to the ULPI interface standard for the USB peripherals, timing could not be met by going through the EMIO. Therefore, USB peripherals MUST be connected to MIO or they are lost.

Which other peripherals would you expect not to have EMIO options? Why?

- The 3 Flash devices. All other peripherals can be mapped to EMIO. The Flash cannot be mapped to EMIO because we depend on the Flash to be the boot device.

What is the Power pin for on the SD peripheral? (Hint: use the TRM)

- This is a control pin output to enable/disable power to the SD Card slot.

12 peripherals can be mapped to PS Pmod. Find one that cannot?

- SP0 and Trace (clk, ctl, and d[3:0]; d[15:4] would have to go to EMIO). These peripherals can be mapped to the PS Pmod: Dual Quad, SPI, SD 1, UART 0, I2C 0, I2C 1, SPI 1, CAN 0, CAN 1, Watchdog, PJTAG, GPIO.

Name one scenario where having independent access to PJTAG would be useful.

- Hardware/software simultaneous debug. The standard JTAG port is used for ChipScope while the PS Pmod PJTAG is used with an ARM DS-5 or DSTREAM for processor debug.

Questions

What is special about MIO[7] and MIO[8]?

- They are output only.

Notice that the FCLK_CLK1 Actual Frequency doesn't match the Requested Frequency. Why is this?

- The PLLs have a limited number of M/N ratios to generate the various frequencies. With the ENET set to 1000 Mbps, we have a hard requirement for a 125 MHz output clock. Starting with a 33.3333 MHz reference clock, the PLL gets set to 1 GHz, then divided by 8 to generate 125 MHz. With the PLL set to 1 GHz, you are limited now by available dividers. When 150 MHz is requested, the closest divider is 7. 1 GHz / 7 = 142.857 MHz.

Look again at Figure 20 – Memory Test Results. What is the Base Address for the memory test?

- 0x00100000

Why doesn't the DDR start at address 0x0?

- This is where OCM resides. You can remap DDR to reside at address 0x0, but this is typically done during a 2nd-stage bootloader. See Section 29.4.1 of the TRM.

AVNET

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

What is Tcl?

History

- Invented by John Ousterhout at UC Berkeley in 1988
  - Intended to unify command languages in EDA software
  - Provides generic facilities such as variables, control structures and procedures
  - Embeddable and extensible
- Grew popular while improved by Sun & Scriptics/Ajula in the 1990s
- Currently Open Source from ActiveState (www.activestate.com)

Adopted by the “big guys”

- Synopsys moved from dcsh to dctcl 10 years ago
- Mentor: ModelSim/Precision/Calibre/Eldo…
- Cadence: Virtuoso/Encounter…
- Even the “other” guys use it!!

Introduction to General Tcl

- Interactive and Interpreted Language
  - No compilation – send commands to interpreter
  - Everything can be treated as a string
- Very Simple, or very complex as you need
  - Loosely typed
  - Full featured language
  - Variables, control, error handling, file IO, etc
- Main utilization
  - Rapid prototyping
  - Scripted applications

Vivado Tcl Defined

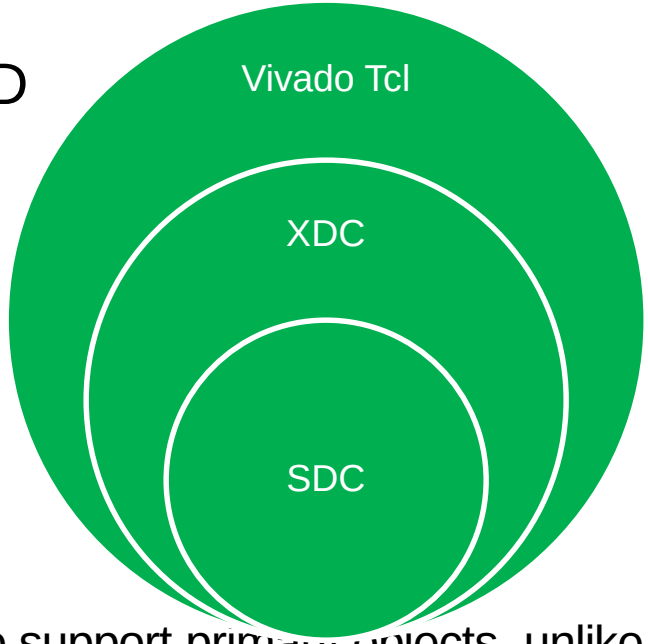
SDC - Synopsys Design Constraints

XDC - Xilinx Design Constraints

- Tcl Equivalent to UCF/PCF/XCF/BCD

Vivado Tcl

- SDC + XDC
- Xilinx commands
  - Project, compilation, report,…
- Xilinx objects
  - Netlist, device, timing, project,…
- General Tcl (8.5)
  - List-related commands are overloaded to support primary objects, unlike Synopsys and the competition
  - Primary objects can directly be used with general commands, unlike Synopsys and the competition



Benefits of XDC and Vivado Tcl

Why you should care

- Future constraint language from Synthesis to P&R
  - Vivado-only for sign-off static timing analysis (STA)
- Powerful debug and analysis
  - Fast custom timing reports
  - What-if Analysis with incremental STA
  - Extendable
- Industry standard Tool Control
  - Synplify, Precision, and all ASIC Synthesis/P&R 3rd Party EDA tools use same interface
- Cross-platform Scripting (Linux and Windows)
  - Just beware those DOS backslashes! “\”
    - Use “/” or {}

Invoking Vivado via Tcl

Vivado IDE startup:

- *vivado -source setup.tcl*
- Type commands directly in Tcl Console
- Tools → Run Tcl Script

Batch Mode – No IDE:

- *vivado -mode batch -source my.tcl*

Interactive Shell – No IDE:

- *vivado -mode tcl*
- Can start the IDE at any time via **`start_gui`** command

Vivado Command Categories Examples

Project - Project Management
- create_project
- add_files
- import_files
- import_ip

Object - Object Access
- get_cells
- get_property
- set_property

Report
- report_timing
- report_power
- report_drc

Project - Runs, Synthesis and Implementation Control

create_run
launch_runs
wait_on_run

FileIO

read_verilog
read_xdc
read_csv
write_verilog

GUIControl

start_gui
stop_gui

AVNET

Tcl References

Tcl Built-in Command Reference:

- http://www.tcl.tk/man/tcl8.5/TclCmd/contents.htm

General Tcl Tutorial

- http://www.tcl.tk/man/tcl/tutorial/tcltutorial.htm

Books (Search on Amazon.com):

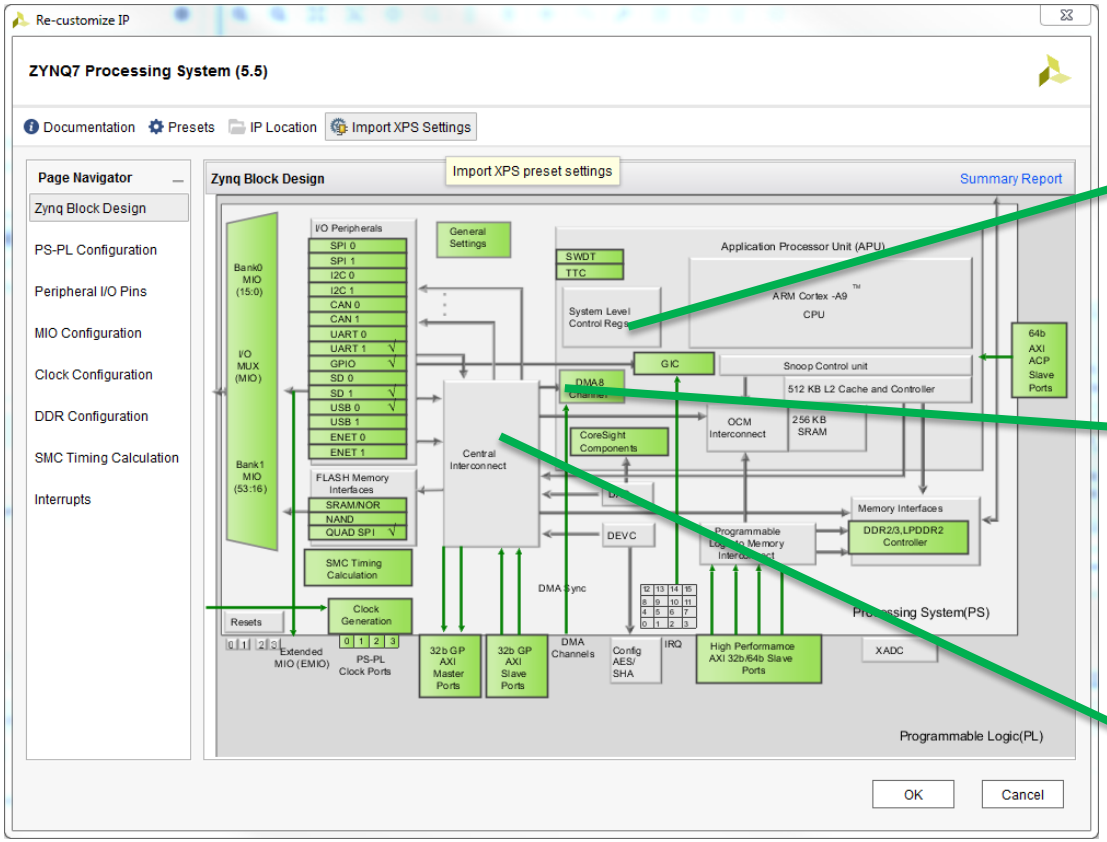
- Brent Welch:
  - http://www.amazon.com/Practical-Programming-Tcl-Tk-4th/dp/0130385603/ref=sr_1_2?ie=UTF8&s=books&qid=1271867857&sr=8-2
- John Ousterhout:
  - http://www.amazon.com/Tcl-Toolkit-2nd-John-Ousterhout/dp/032133633X/ref=sr_1_1?ie=UTF8&s=books&qid=1271867857&sr=8-1

Files Created by Vivado Block Design

Block design saved as a .bd file

Everything created in a block can be recreated by a TCL script…

- More later!



.BD

XML

PS7_INIT

XML File

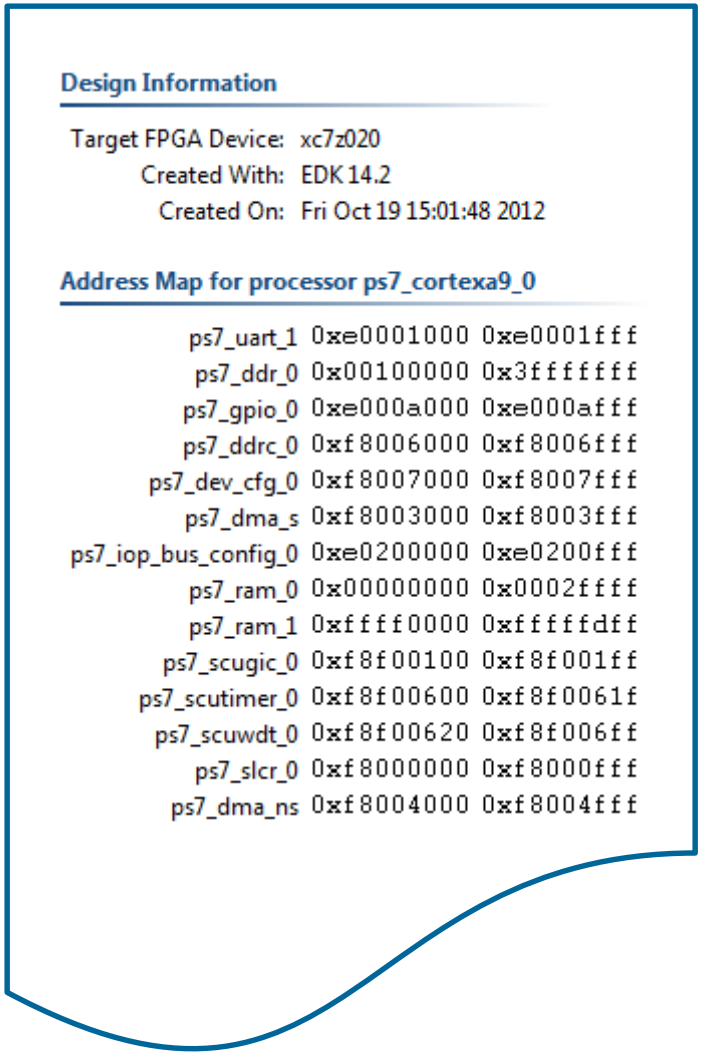
Hardware platform description for FSBL and BSP generation

Automatically created by Vivado, Exported to SDK

Text File, but do not edit, use IP Integrator, or TCL commands

Contains

- Processor instantiation
- Peripheral instances and addresses
  - Similar to ISE's MHS, but in XML format



Design Information

Target FPGA Device: xc7z020
Created With: EDK 14.2
Created On: Fri Oct 19 15:01:48 2012

Address Map for processor ps7_cortexa9_0

ps7_uart_1	0xe0001000	0xe0001fff
ps7_ddr_0	0x00100000	0x3fffffff
ps7_gpio_0	0xe000a000	0xe000afff
ps7_ddrc_0	0xf8006000	0xf8006fff
ps7_dev_cfg_0	0xf8007000	0xf8007fff
ps7_dma_s	0xf8003000	0xf8003fff
ps7_iop_bus_config_0	0xe0200000	0xe0200fff
ps7_ram_0	0x00000000	0x0002ffff
ps7_ram_1	0xffff0000	0xffffdfff
ps7_scugic_0	0xf8f00100	0xf8f001ff
ps7_scutimer_0	0xf8f00600	0xf8f0061f
ps7_scuwdt_0	0xf8f00620	0xf8f006ff
ps7_slcr_0	0xf8000000	0xf8000fff
ps7_dma_ns	0xf8004000	0xf8004fff

AVNET

Processor Initialization – PS7_Init

PS7_Init file describes processor configuration

- DDR Controller Specifications
- PLL Configuration
- JTAG Modes
- Zynq Peripheral Configuration

C, TCL and HTML files created

ps7_pll_init_data

ARM PLL INIT

ARM PLL INIT

- Register : ARM_PLL_CFG @ 0XF8000110

Bitfield	Bits	Mask	Value	Shifted Value
PLL_RES	7:4	f0	2	20
PLL_CP	11:8	f00	2	200
LOCK_CNT	21:12	3ff000	fa	fa000
ARM_PLL_CFG @ 0XF8000110		3ffff0		fa220

DDR Memory information

DDR Controller Configuration

Parameter	Value
Enable DDR	1
Memory Type	DDR 3
Memory Part	MT41J128M16 HA-15E
DRAM bus width	32 Bit
ECC	Disabled
BURST Length (lppdr only)	8
Internal Vref	1
Operating Frequency (MHz)	533.333333
HIGH temperature	Normal (0-85)

Memory Part Configuration

Parameter	Value
DRAM IC bus width	16 Bits
DRAM Device Capacity	2048 MBits
Speed Bin	DDR3_1066F
BANK Address Count	3
ROW Address Count	14
COLUMN Address Count	10
CAS Latency	7
CAS Write Latency	6
RAS to CAS Delay	7
RECHARGE Time	7
tRC (ns)	49.5
tRASmin (ns)	36.0
tFAW	45.0
ADDITIVE Latency	0

BD File

Exists in Vivado

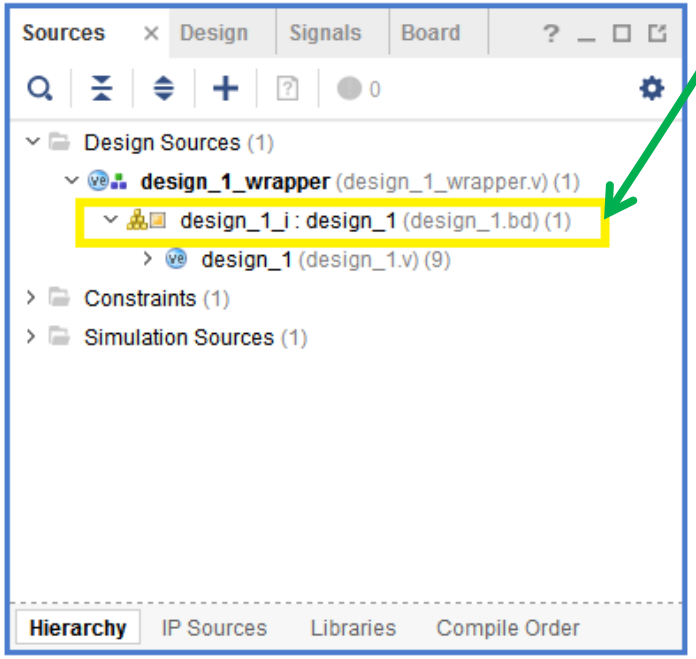
The BD file is the block design file for IP Integrator

Takes on the name of the project, *<project>.bd*

- In the labs, Z_system.bd is used

Contains and controls

- Details that make up the block design
  - XML format
- Previously, MHS file
- DON'T EDIT outside IPI



AVNET

Checkpoint!

What tool is used to create and define the Zynq Processors?
Files?

IP Integrator
PS7_Init, XML, BD

What tool is used to create Zynq Software?

SDK – Software

AVNET

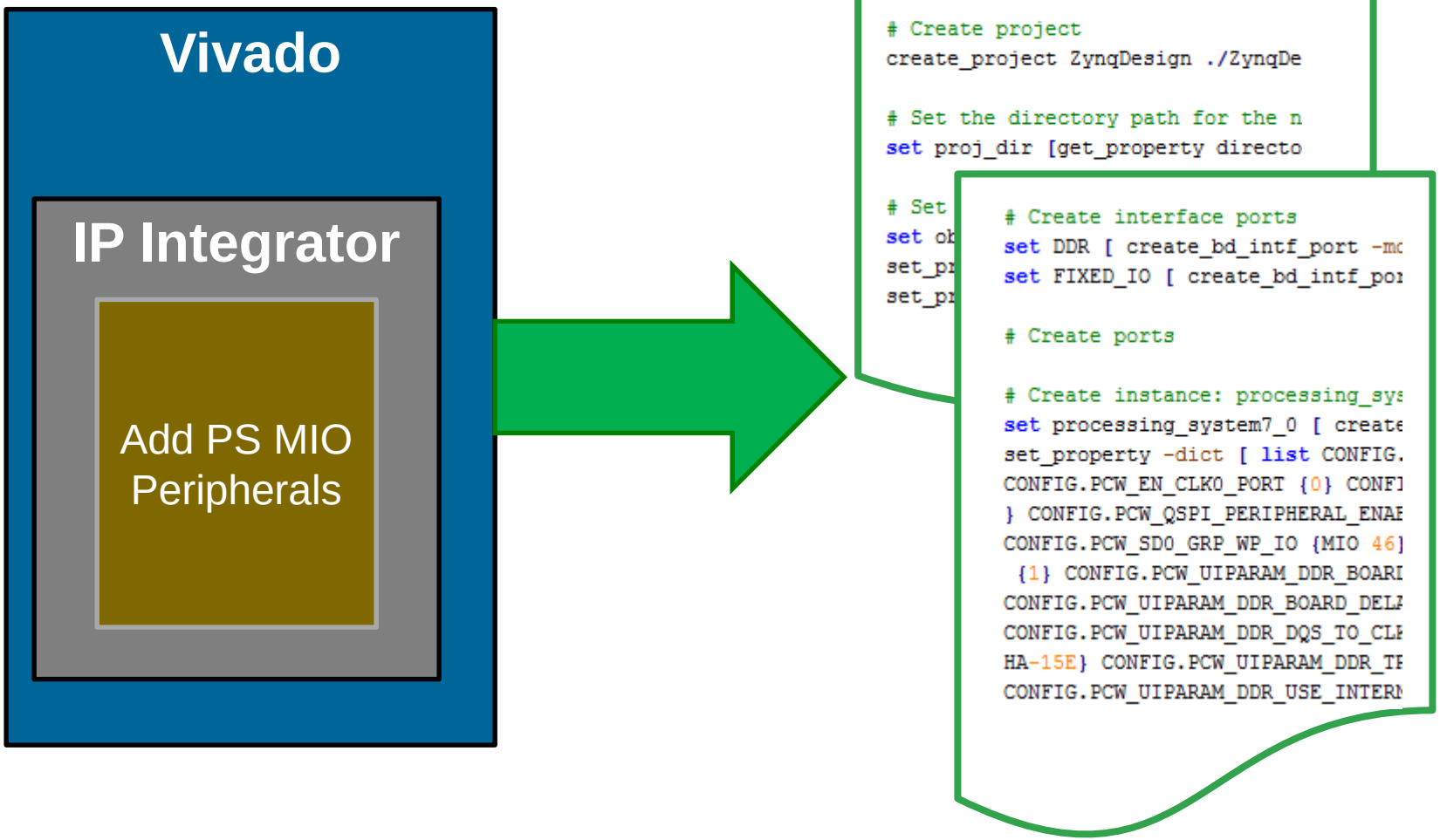
Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - **Lab 4 – Using TCL with Vivado Embedded Designs**
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Lab 4 – TCL Scripting

Create TCL scripts that rebuild project



AVNET

Questions

What is the project name?

- # CHANGE DESIGN NAME HERE
  set design_name Z_system

How many interface ports do you see? What are they?

- 2 – DDR & FIXED_IO

What is the MIO Bank1 Voltage set to?

- CONFIG.PCW_PRESET_BANK1_VOLTAGE {LVCMOS 1.8V}

In what directory will the project be created?

- This is a tough question. The first TCL command in this script sets the original project location, but that is only used to import source files. The second TCL command creates the project and does it in whatever directory the TCL script was sourced from. So before running this script, make sure you are in the directory you want the project created.

What part is selected?

- set_property "part" "xc7z010clg400-1" $obj

Does the TCL file load the block design?

- Yes it does:
  # Import local files from the original project
  set files [list \
  "C:/Speedway/ZynqHW/2017_1/ZynqDesign/ZynqDesign.srcs/sources_1/bd/Z_system/Z_system.bd"\
- But it gets this source from our original project directory. So if run on another PC, the project will not get recreated correctly.

Does the TCL file reload the synthesis and implementation settings?

- Yes, see Create 'synth_1' run and Create 'impl_1' run

AVNET

Questions

In the project_setup.tcl script, what files does the script need to recreate the project?

- ZynqDesign/ZynqDesign.srcs/sources_1/bd/Z_system/Z_system.bd"\
  ZynqDesign/ZynqDesign.srcs/sources_1/bd/Z_system/hdl/Z_system_wrapper.v"\

Does the Zynq IP block look the same? Are all I/O peripherals mapped the same?

- Yes, it is the same

Could both of the TCL files used to recreate this project be combined into one TCL file?

- Yes, they could simply be combined, but the project_setup.tcl would need to remove importing the block design source. Otherwise when the basic_design.tcl is run, it will already see a block design with the same name and quit running.

What source files were needed to recreate this entire project? In other words, what do you need to archive for revision control at this point?

- None, only the two TCL scripts are required. Per the above question, one script could be created that performs the entire project creation, block design creation, and Zynq IP Customization. That's quite powerful. Thus only the TCL file would need to be checked into a revision control system.

AVNET

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

Interfacing the PL to the PS

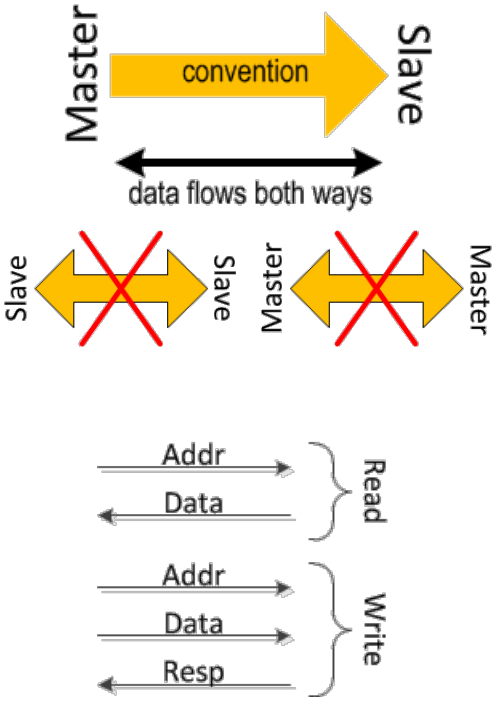
- **Interfacing the PL to the PS**
- Interconnect in the PS
- What is AXI?
- AXI Master Interfaces
- AXI Slave Interfaces
- AXI ACP Interface
- Going to Programmable Logic
- Summary

Interfacing to Programmable Logic

- **Most PS-PL interfacing is accomplished with nine AXI ports**
  - Two general-purpose master ports
  - Two general-purpose slave ports
  - Four high-performance slave ports
  - One accelerator coherence port (ACP) slave port
- **AXI is a point-to-point protocol**
  - A single master can connect to multiple slave by using a switch
  - An understanding of PS interconnect (switches) operation is necessary for proper design use of the various ports
- **Other diagnostic, clock, and interrupt signals complete the PS-PL interface**

PL and PS AXI Port Terminology

- **AXI ports come in two flavors (or genders)**
  - Master: M_AXI_GP0 – transaction initiator
  - Slave: S_AXI_GP0 – transaction target
- **Like ports cannot connect together**
  - Master ports only connect to Slave ports
  - Slaves only connect to a Master port
- **All PL or PS AXI ports are AXI-Full (AXI-Memory)**
  - All five AXI channels are implemented
    - Read: Address, Data; Write: Address, Data, Response
  - Full duplex



AMBA AXI 3.0

- **The PS-PL AXI ports are based on the AMBA® AXI 3.0 version of the protocol**
- **Xilinx IP is committed to the AMBA AXI 4.0 version of the protocol**
- **The two versions are very similar**
  - Burst data size: 16 bytes vs. 256 bytes
  - Additional signals: advanced usage
  - Other minor differences
- **Xilinx tools automatically build a logic shim interface as needed to adapt between the two**

Interconnect in the PS

- Interfacing the PL to the PS
- **Interconnect in the PS**
- What is AXI?
- AXI Master Interfaces
- AXI Slave Interfaces
- AXI ACP Interface
- Going to Programmable Logic
- Summary

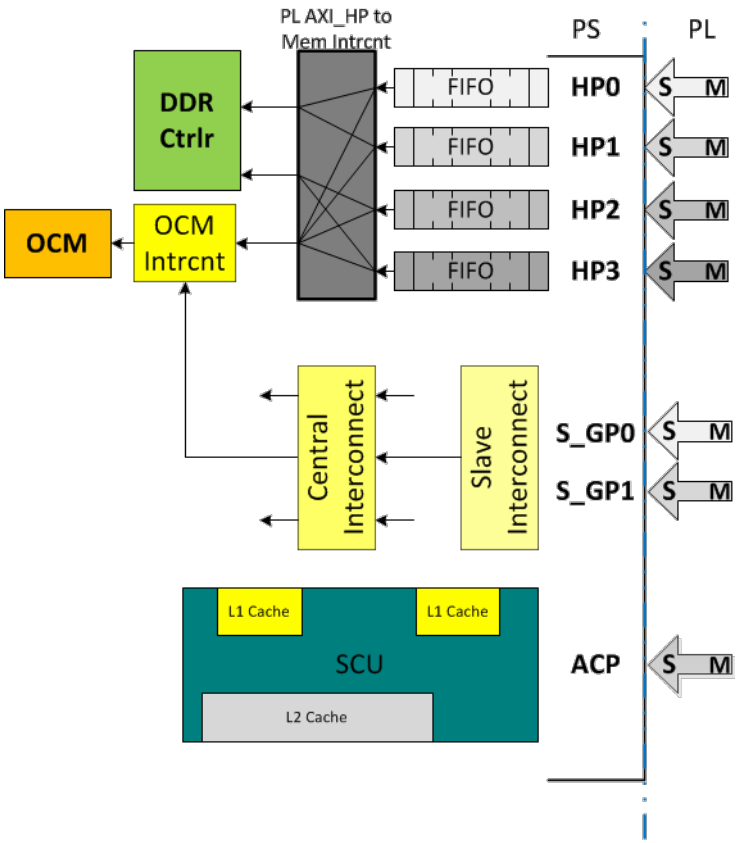
A Picture is Worth a Thousand Words

- **Understanding the PS interconnect facilitates a high-performance Zynq architecture design**
- **The PS has six interconnect structures to facilitate data movement**
- **Sometimes paths may need to make their way through multiple switches**
- **Trade-offs exist between**
  - PS peripheral access
  - Performance
  - PL AXI port access and loading
- **(Pictures are on the following slides)**

AVNET

Simplified Connections

- **Elements within the PS can often be accessed by more than one AXI port**
  - Some paths are faster than others
- **Choosing the proper port can make a big difference in performance**
  - Path latency
  - Port bandwidth



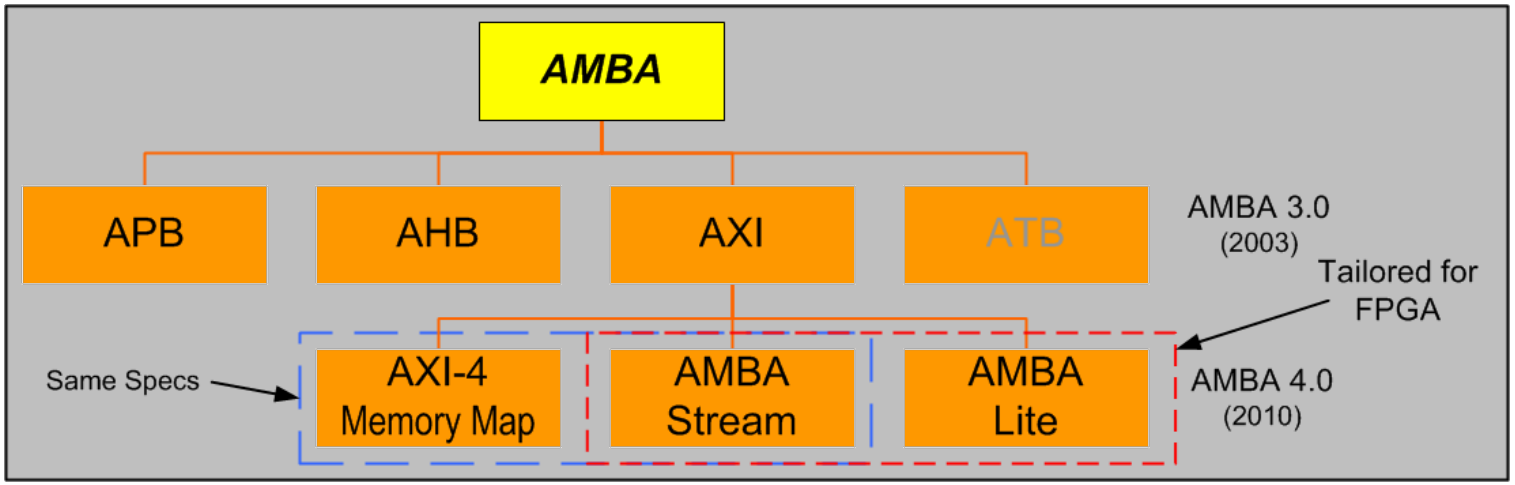
What are the Differences Between Slave HPx, GPx, and ACP?

- **The four S_AXI_HPx ports can only access the PS OCM and DMC**
  - Purpose is low-latency PL to memory access
  - FIFOs built into interface allow for streaming
- **The two S_AXI_GPx ports can access most PS peripherals, including memory**
  - Purpose is programmable logic access to PS IOP and other PS slaves
  - Much slower access to OCM and DDRx memory
- **Single S_AXI_ACP can access all PS memory and peripherals**
  - Purpose is programmable logic co-processor memory interface
    - Performance advantage through cache hits
  - Lower latency then HPx for cache misses

What is AXI?

- Interfacing the PL to the PS
- Interconnect in the PS
- **What is AXI?**
- AXI Master Interfaces
- AXI Slave Interfaces
- AXI ACP Interface
- Going to Programmable Logic
- Summary

AXI is Part of AMBA: Advanced Microcontroller Bus Architecture



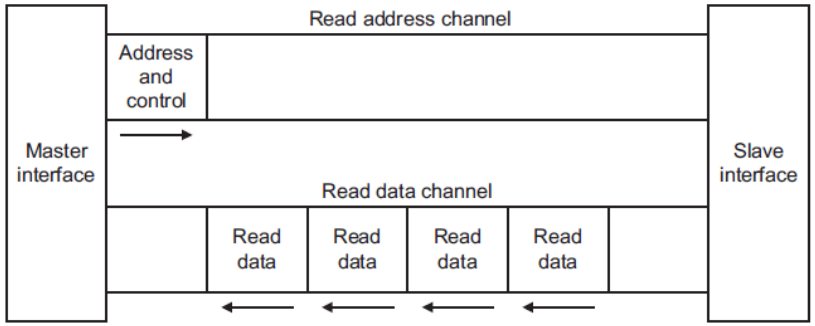
Interface	Features	Similar To
Memory Map / Full	Traditional address/data burst (single address, multiple data)	PLB v46, PCI
Streaming	Data only, burst	Local Link/DSP interfaces/FIFO/FSL
Lite	Traditional address/data—no burst (single address, single data only)	PLB v46 single OPB

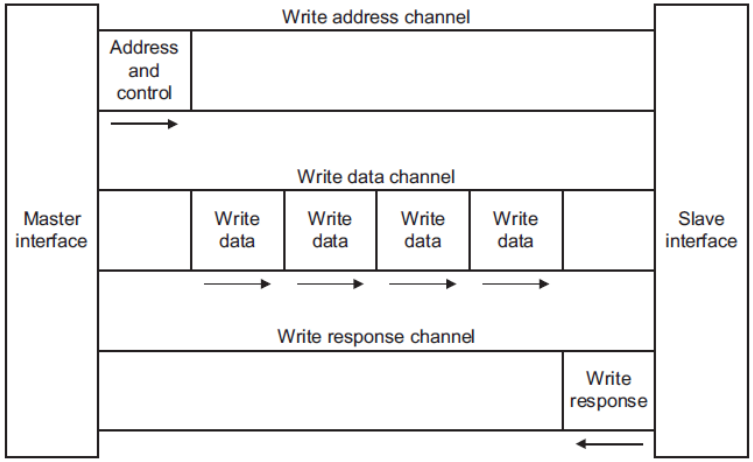
AVNET

AXI Memory Mapped Protocol

Satisfies: Processor -> Interconnect -> Peripheral use case

AXI4 Interface comprises 5 channels:

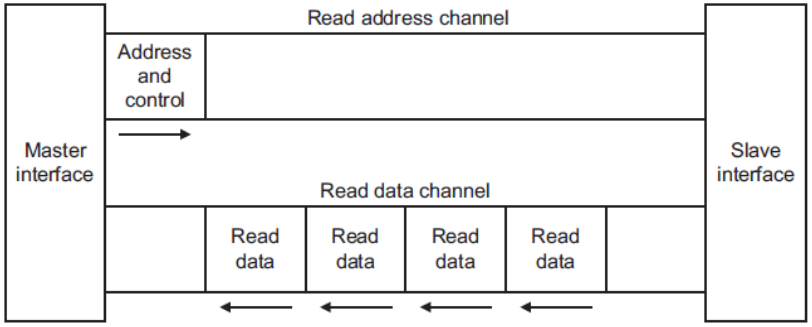




- 2 Read channels
  - Address
  - Read data and read response (combined in one channel)
- 3 Write channels
  - Address
  - Write Data
  - Write Response

AVNET

Overview of each of the 5 channels (A1.3)



Read Address

- ARVALID
- ARREADY

Handshake

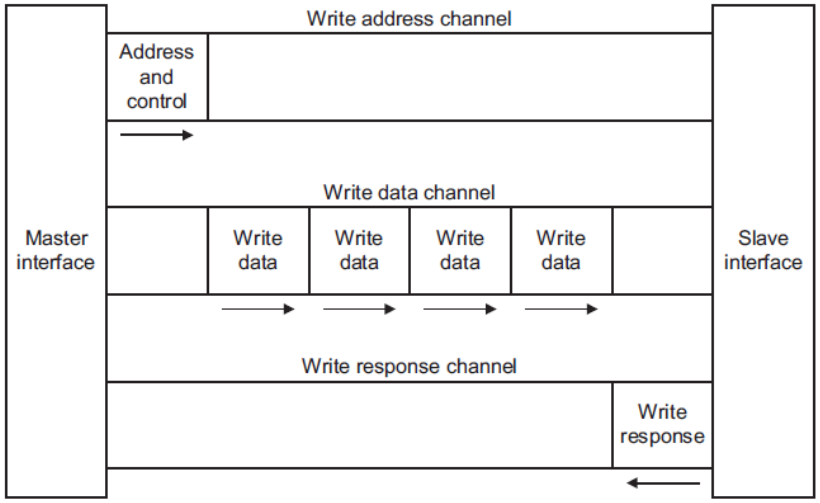
- ARID[m:0]
- ARADDR[a:0]
- ARLEN[7:0]
- ARSIZE[2:0]
- ARBURST[1:0]
- ARPROT[2:0]
- ARLOCK
- ARCACHE[3:0]
- ARREGION[3:0]
- ARQOS[3:0]

Read Data

- RVALID
- RREADY

Handshake

- RID[m:0]
- RDATA[n-1:0]
- RRESP[1:0]
- RLAST



Write Address

- AWVALID
- AWREADY

Handshake

- AWID[m:0]
- AWADDR[a:0]
- AWLEN[7:0]
- AWSIZE[2:0]
- AWPROT[2:0]
- AWBURST[1:0]
- AWLOCK
- AWCACHE[3:0]
- AWREGION[3:0]
- AWQOS[3:0]

Write Data

- WVALID
- WREADY

Handshake

- WDATA[n-1:0]
- WSTRB[n/8-1:0]
- WLAST

Write Response

- BVALID
- BREADY

Handshake

- BID[m:0]
- BRESP[1:0]

AVNET

AXI Channel handshake basics

All AXI channels follow READY/VALID handshake rules

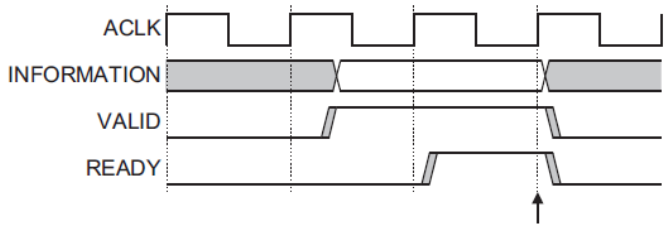
Once valid is asserted, it can't be deasserted until the corresponding ready has been received.

Once valid is asserted, other signals driven by initiator must not change

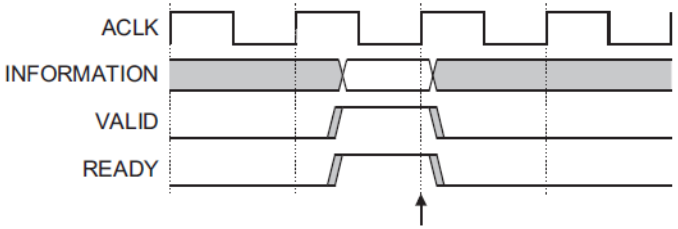
READY can be withheld

Valid must NEVER be withheld (system may deadlock)

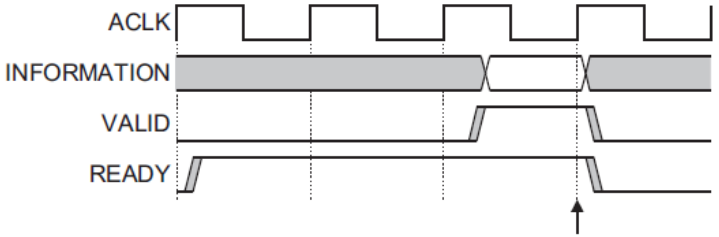
No combinatorial paths between input and output signals of the same interface



VALID before READY



VALID co-incident with READY



READY before VALID
("Pre-asserted READY")
Master must **never** wait for READY

AVNET

AXI4, AXI4-Lite

AXI4-Lite is a simplified version of AXI4 interface
Intended for peripheral control/register interfaces
Restricted data widths (32 or 64 bit)
No bursts, no IDs, no exclusive access.

Read Address
- ARVALID
- ARREADY
- ARID[m:0]
- ARADDR[a:0]
- ARLEN[7:0]
- ARSIZE[2:0]
- ARBURST[1:0]
- ARPROT[2:0]
- ARLOCK
- ARCACHE[3:0]
- ARREGION[3:0]
- ARQOS[3:0]

Write Address
- AWVALID
- AWREADY
- AWID[m:0]
- AWADDR[a:0]
- AWLEN[7:0]
- AWSIZE[2:0]
- AWBURST[1:0]
- AWPROT[2:0]
- AWLOCK
- AWCACHE[3:0]
- AWREGION[3:0]
- AWQOS[3:0]

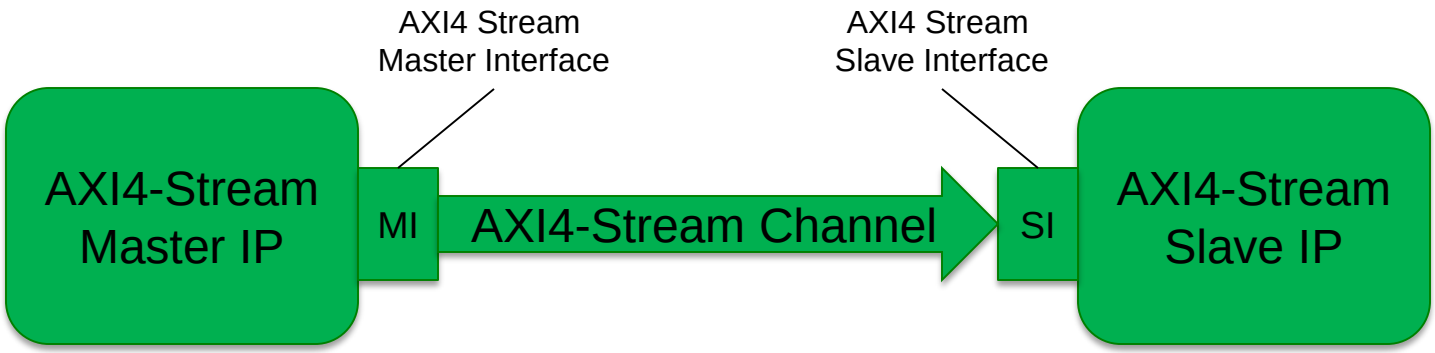
Read Data
- RVALID
- RREADY
- RID[m:0]
- RDATA[n-1:0]
- RRESP[1:0]
- RLAST

Write Data
- WVALID
- WREADY
- WDATA[n-1:0]
- WSTRB[n/8-1:0]
- WLAST

Write Response
- BVALID
- BREADY
- BID[m:0]
- BRESP[1:0]

AVNET

AXI-Stream



AXI4-Stream is a unidirectional, point-to-point interface standard for exchanging continuous streams of data

Similar to the write data channel of memory mapped AXI4

Stream interfaces are flexible

Most of the signals are optional so IPs can ‘opt-in’ to only those signaling features that are actually necessary for their application

AVNET

Documentation

- ***Xilinx AXI Reference Guide* (UG761)**
  - AXI for Xilinx system development
  - AXI support in Xilinx tools and IP AXI
  - feature adoption in Xilinx FPGAs
  - Migrating to Xilinx AXI protocols

- **ARM specification**
  - AMBA AXI protocol version 2.0
  - AMBA 4 AXI4-Stream protocol version 1.0
  - infocenter.arm.com/help/topic/com.arm.doc.set.amba

AXI Master Interfaces

- Interfacing the PL to the PS
- Interconnect in the PS
- What is AXI?
- **AXI Master Interfaces**
- AXI Slave Interfaces
- AXI ACP Interface
- Going to Programmable Logic
- Summary

General-Purpose Master Ports

- **Two identical 32-bit AXI master ports**
  - M_AXI_GP0
  - M_AXI_GP1
- **Provides initiator access from the PS-to-PL target in programmable logic**
- **Attaches to a slave AXI port in programmable logic**
  - PL AXI interconnect
  - PL slave
    - Your peripheral built in programmable logic
    - IP Integrator interface or other third-party-based IP
- **Why two ports?**
  - Each port is capable of driving a number of peripherals using an AXI switch
  - Multiple ports enable the designer to distribute bandwidth

General-Purpose Master AXI Port Features

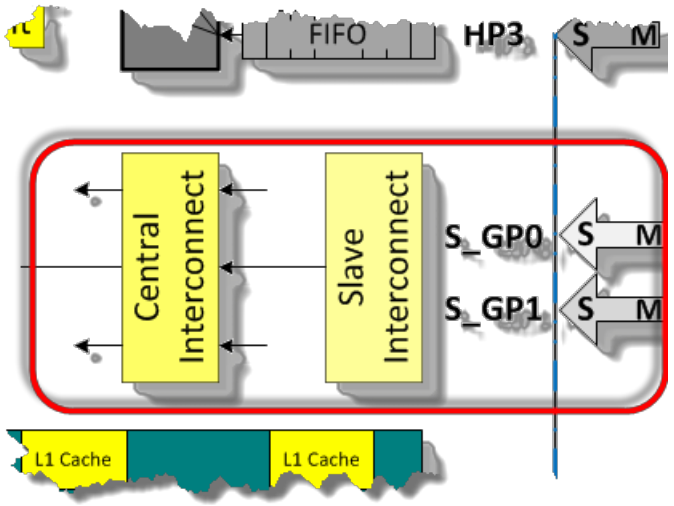
- **Each master is allocated a 1GB address space**
  - Addresses start at 0x40000000 and 0x80000000, respectively
  - Only low-order 30 bits of AXI address are meaningful
- **PS masters can be**
  - Cortex-A9 processors via the L2 cache controller
  - USB controller (2)
  - Ethernet controller (2)
  - SD/SDIO controllers (2)
  - DMAC
  - Debug access port
- **Mostly used for CPU and I/O peripheral block data movement** *to* **programmable logic**

AXI Slave Interfaces

- Interfacing the PL to the PS
- Interconnect in the PS
- What is AXI?
- AXI Master Interfaces
- **AXI Slave Interfaces**
- AXI ACP Interface
- Going to Programmable Logic
- Summary

General-Purpose Slave Ports

- **Two identical 32-bit AXI slave ports**
  - S_AXI_GP0
  - S_AXI_GP1
- **Provides initiator access from PL master to PS target**
- **Attaches to master AXI port in PL**
  - PL AXI interconnect
  - PL master
    - MicroBlaze processor
    - Your master built in programmable logic
    - Other third-party-based IP



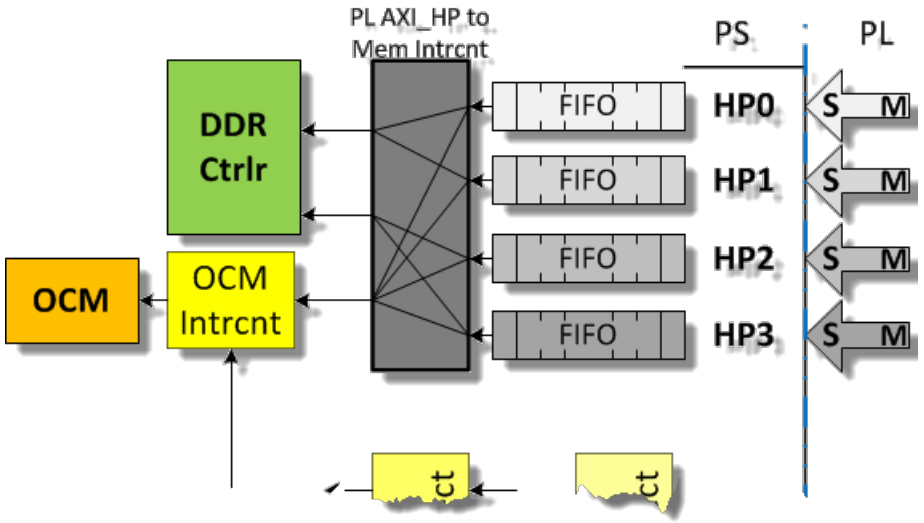
AVNET

General-Purpose Slave AXI Port Features

- **Allows PL masters to access PS slaves**
- **PS slaves can be**
  - DDRx controller
  - On-chip memory (OCM)
  - IOP peripheral
  - Device configuration controller (DEVC)
  - Debug access port (DAP)
- **Mostly used for PL masters to access IOP, OCM RAM, and DDRx (simple access)**

High-Performance Slave Ports

- **Four identical AXI slave ports**
  - S_AXI_HP0
  - S_AXI_HP1
  - S_AXI_HP2
  - S_AXI_HP3



- **Provides initiator access from PL master to PS memory**

- **Attaches to master AXI port in PL**
  - DMA controller
  - MicroBlaze processor
  - Custom master built in programmable logic
  - Other third-party-based IP

AVNET

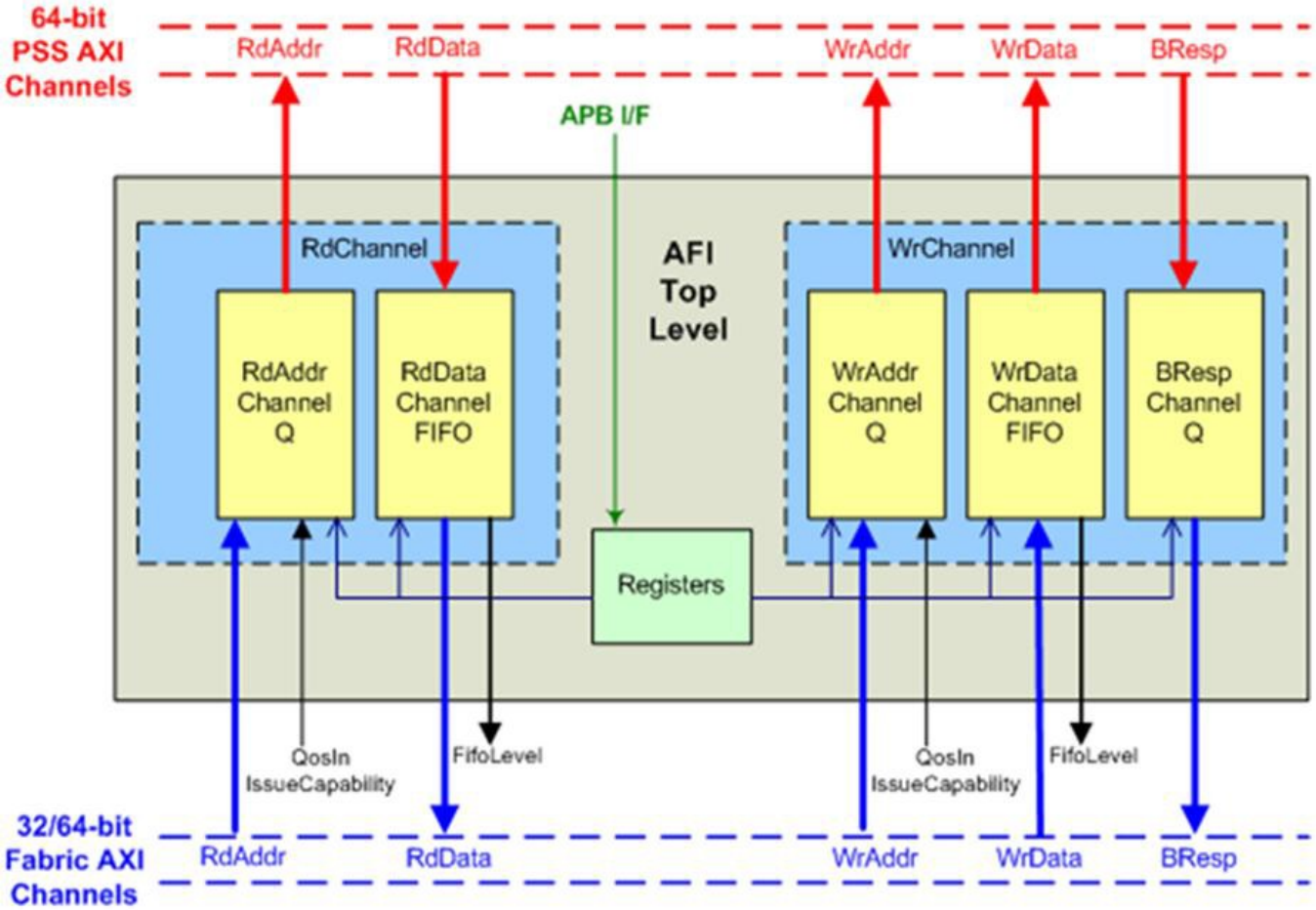
High-Performance Slave Port Features (1)

- **Four 64-bit/32-bit configurable FIFO-based AXI slave interfaces (AFI)**
- **Asynchronous crossing between the PL clock domain and PS clock domain**
- **QoS supported from the programmable logic ports**

High-Performance Slave Port Features (2)

- **FIFOs smooth out of “long-latency” events**
  - 1 KB (128 by 64 bit) data FIFOs
  - Both read and write paths
- **Provides low latency access to DDR and OCM**

AXI FIFO Interface (AFI) Diagram



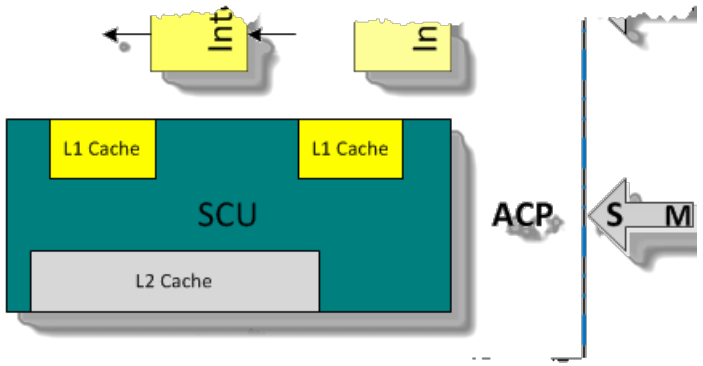
AVNET

AXI ACP Interface

- Interfacing the PL to the PS
- Interconnect in the PS
- What is AXI?
- AXI Master Interfaces
- AXI Slave Interfaces
- **AXI ACP Interface**
- Going to Programmable Logic
- Summary

Accelerator Coherence Port (ACP) Slave Port

- **One accelerator coherence port (ACP)**
  - S_AXI_ACP
- **Provides initiator access from PL master to PS target**
  - Via L1 and L2 cache
- **Attaches to master AXI port in PL**
  - Typically a PL-based co-processor
  - Also could be (extreme design scenarios)
    - MicroBlaze processor
    - Custom master built in programmable logic
    - Other third-party-based IP



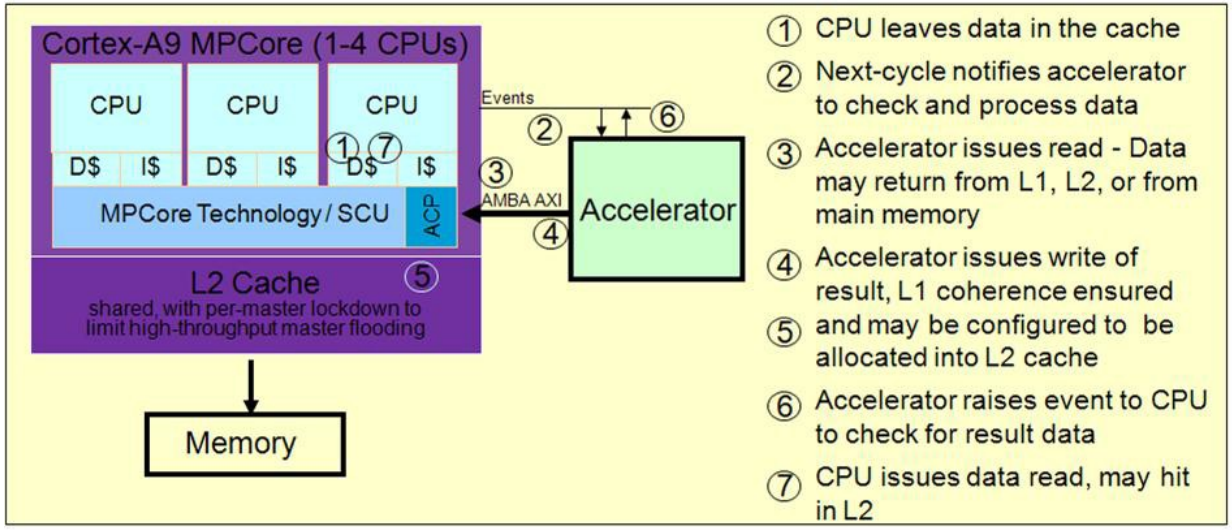
AVNET

Accelerator Coherence Port (ACP) Slave Port Features

- **64-bit AXI slave port from PL into PS**
- **Direct connection to the snoop control unit (SCU)**
- **Cache coherent with L1 and L2 caches**
- **PL co-processor tightly coupled with PL event ports**
  - Performance relies on PL co-processor access cache hits
  - Software and co-processor access both use cached memory accesses
    - Software program coordinated with co-processor design
    - Preferably L1 cache hit over L2
    - Direct access to memory (cache miss) slower than access by HPx slave port

Enhanced Accelerator SoC Integration

- **ARM MPCore: accelerator coherence port (ACP)**
  - Sharing benefits of the ARM MPCore optimized coherency design
  - Accelerators gain access to CPU cache hierarchy
  - Compatible with standard un-cached peripherals and accelerators
  - Use of event signals optional



AVNET

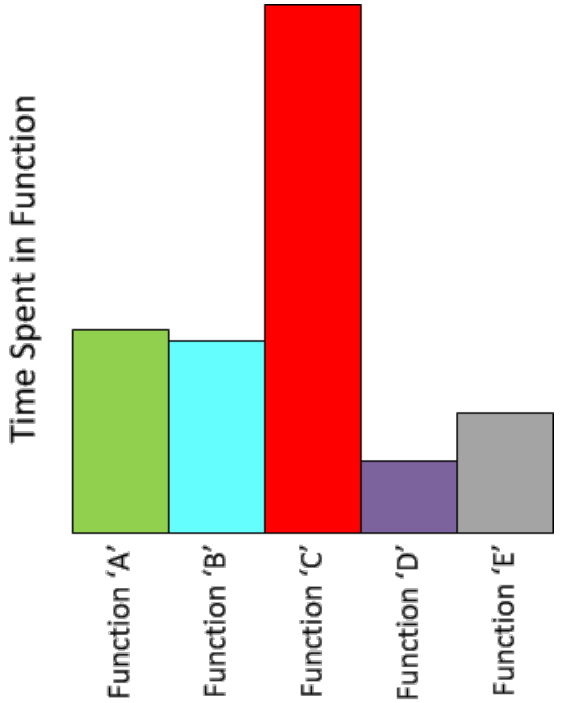
Going to Programmable Logic

- Interfacing the PL to the PS
- Interconnect in the PS
- What is AXI?
  - AXI Master Interfaces
- AXI Slave Interfaces
- AXI ACP Interface
- **Going to Programmable Logic**
- Summary

AVNET

Which Functions Move to Programmable Logic

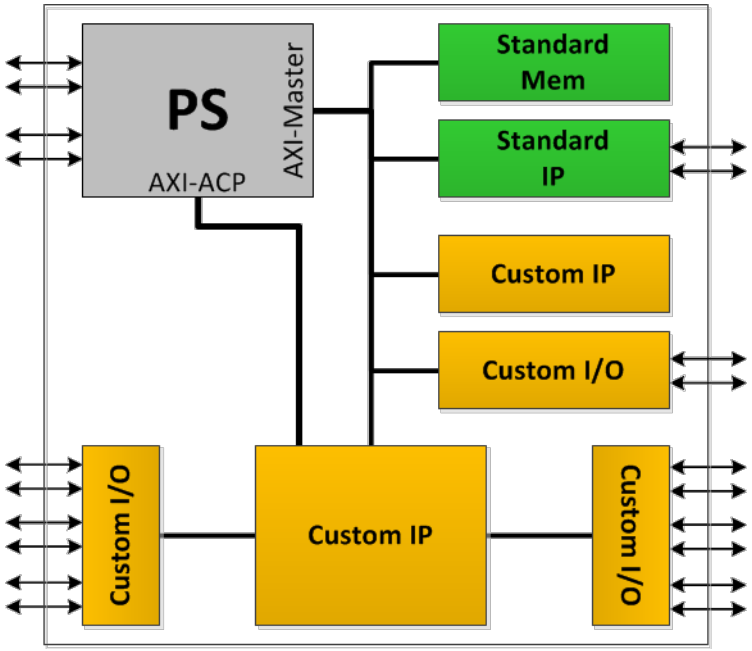
- **Slow software tasks can be accelerated by taking them to hardware**
  - After thoroughly profiling
  - Good candidates are where the software spends most of its time
    - Usually one or two functions stand out



AVNET

ACP vs. Other Ports

- **The accelerator coherency port is not the only way to accelerate a design**
  - Some designs benefit from using the HP ports rather than the ACP
  - Many video designs fall into this category
- **Many accelerator designs use the ACP or HP ports for moving data, and the M_AXI_GP for reading/writing registers, and interrupts for signaling completion**



AVNET

AXI Bandwidths

Raw Performance – Single Channel, Single Direction

Type	Max Bandwidth	Connects To
M/S_AXI_GPx (32bit)	600 MB/s	Masters connect to PL Slaves tie the PL to many of the internal resources within the PS (IOP, OCM, etc.)
S_AXI_HPx (64bit)	1200 MB/s	DDRx controller, OCM-RAM
S_AXI_ACP (64bit)	1200 MB/s	SCU (L1, L2 caches, DDRx indirectly)

Compare to Peripheral Bandwidth

Type	Max Bandwidth	Type	Max Bandwidth
DDR Controller (32bits)	4264 MB/s	SD (4bits)	25 MB/s
OCM (64bits)	1779 MB/s	USB (8bits)	60 MB/s

AVNET

Summary

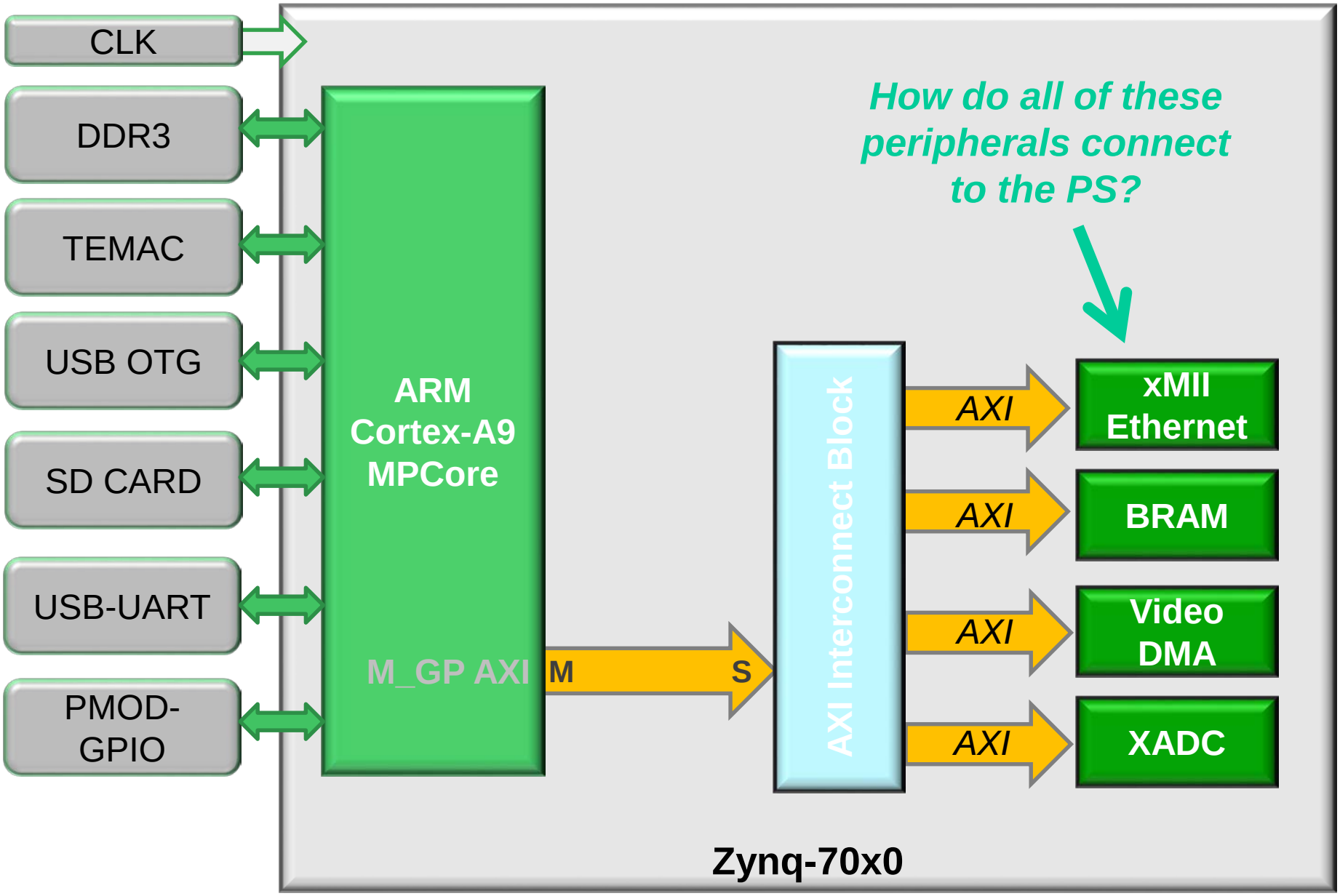
- Interfacing the PL to the PS
- Interconnect in the PS
- AXI Master Interfaces
- AXI Slave Interfaces
- AXI ACP Interface
- **Summary**

AVNET

Summary

- **Nine AXI ports are used to move data between the PS and the PL**
  - Two general-purpose masters (M_AXI_GPx)
  - Two general-purpose slaves (S_AXI_GPx)
  - Four high-performance slaves for DDRx access (S_AXI_HPx)
  - One slave for DDRx access via processor cache (S_AXI_ACP)
- **Six PS interconnects provide pathways for PS components and PS/PL AXI ports**

Adding Soft, PL IP Peripherals



AXI Interconnect Block

Connects Multiple Master / Slave Pairs

Up to 16 masters and 16 slaves per interconnect

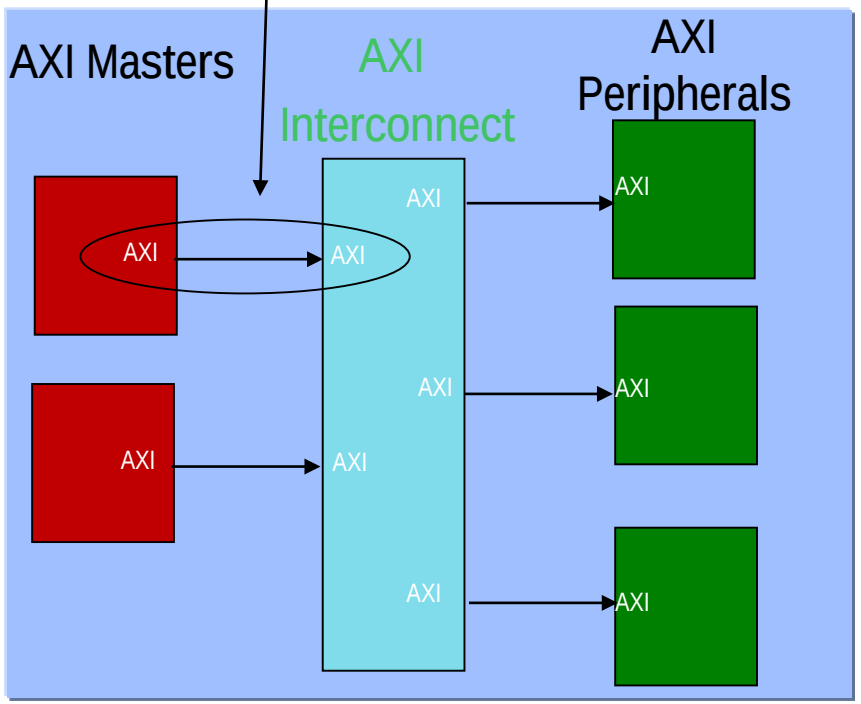
32 to 256 bit data widths per endpoint

Built-in AXI protocol conversion

Each master / slave pair has own clock domain

Pipeline registers per channel to boost timing

AXI defines a point to point, master/slave interface



AVNET

Checkpoint!

What is the estimated throughput of a High Performance DMA AXI interface?

64 bits (8 bytes) * 150 MHz = 1200 Mbytes/s

How many interfaces can an AXI Interconnect Block support?

16 Masters and 16 Slaves

How many slave peripherals can the AXI GP ports support?

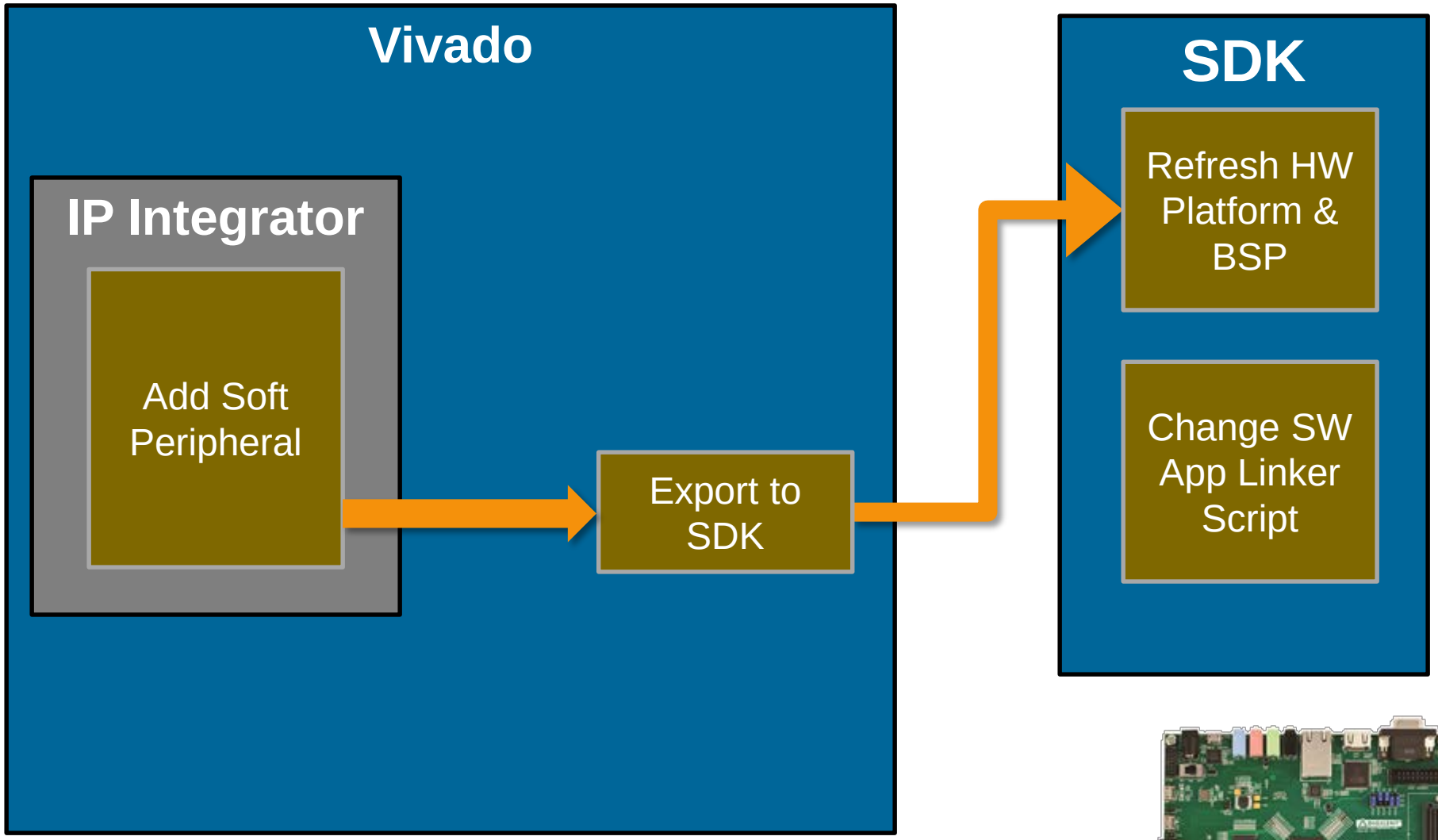
32 as there are two AXI GP Master ports

Agenda

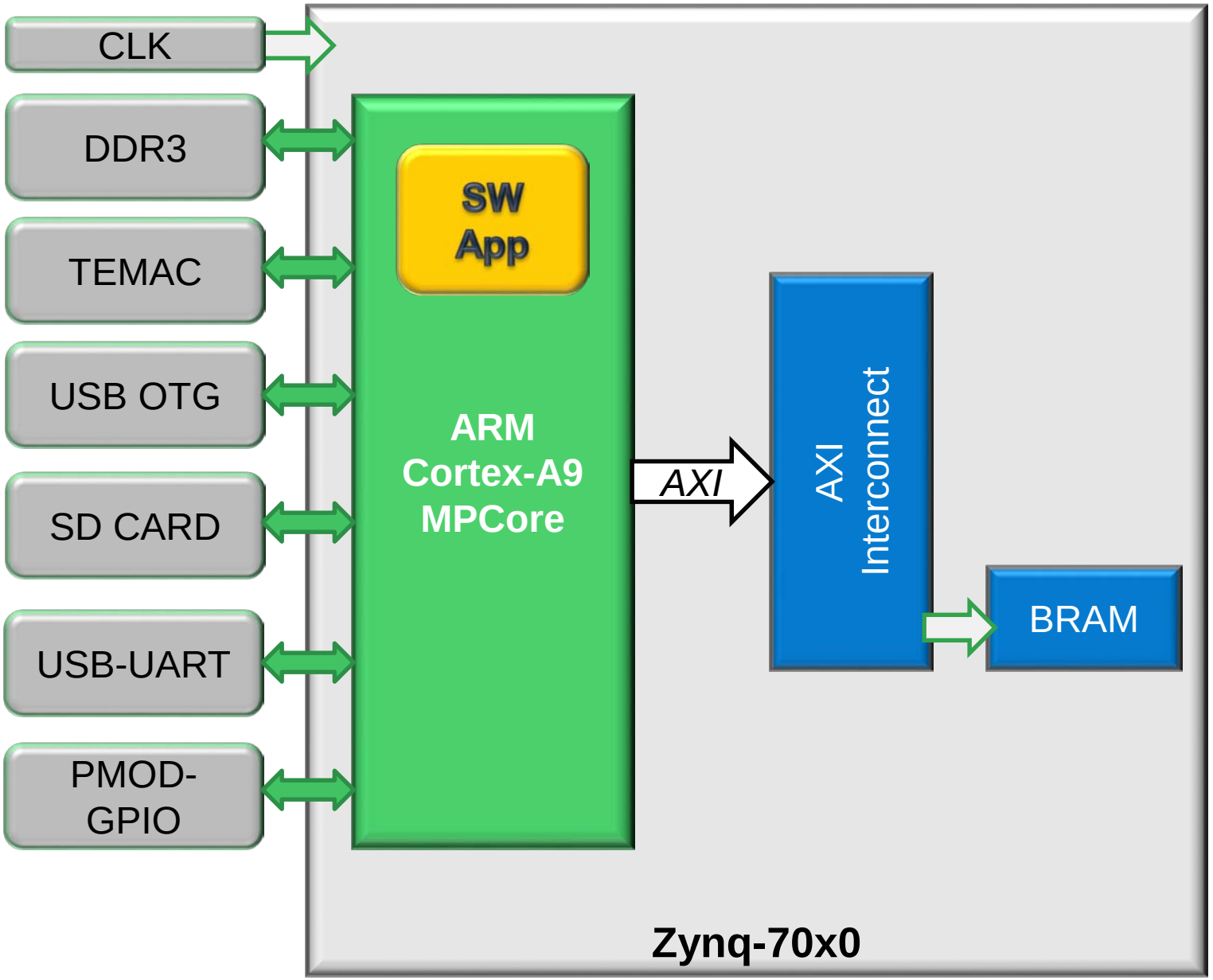
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - **Lab 5 – Adding an AXI PL Peripheral**
- Zynq PS DMA Controller
  - Lab 6 –
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Lab 5 – Add PL Peripherals



Lab 5 – Add PL Peripheral



AVNET

Questions

How many BRAM's are consumed by the Block Memory Generator?

- 2

What is the base address of the BRAM? Why is it mapped here?

- 0x40000000, because this is starting address space for M_AXI_GP0 (refer to the Zynq All Programmable SoC User Guide).

If more IP peripherals were connected, where would they connect?

- Add Master AXI ports to the AXI Interconnect Block for additional slaves to connect into.

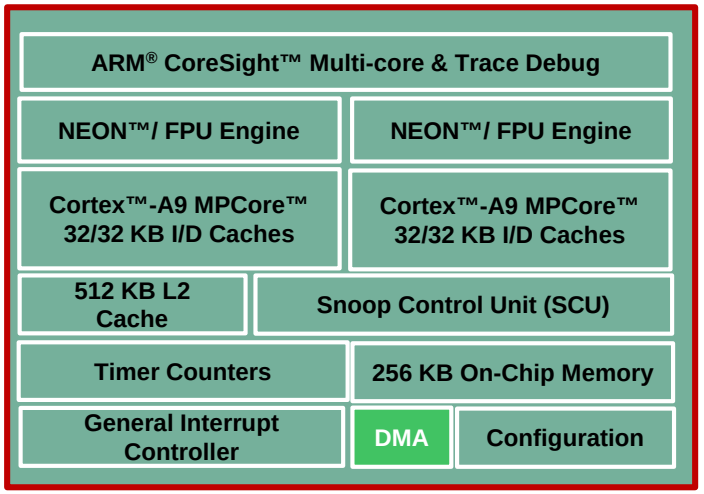
Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5 –
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

PS DMA Controller

Direct Memory Access
    Controllers can manage data transactions within PS
Meanwhile CPU's can keep working out of cache
Works with PL slaves only



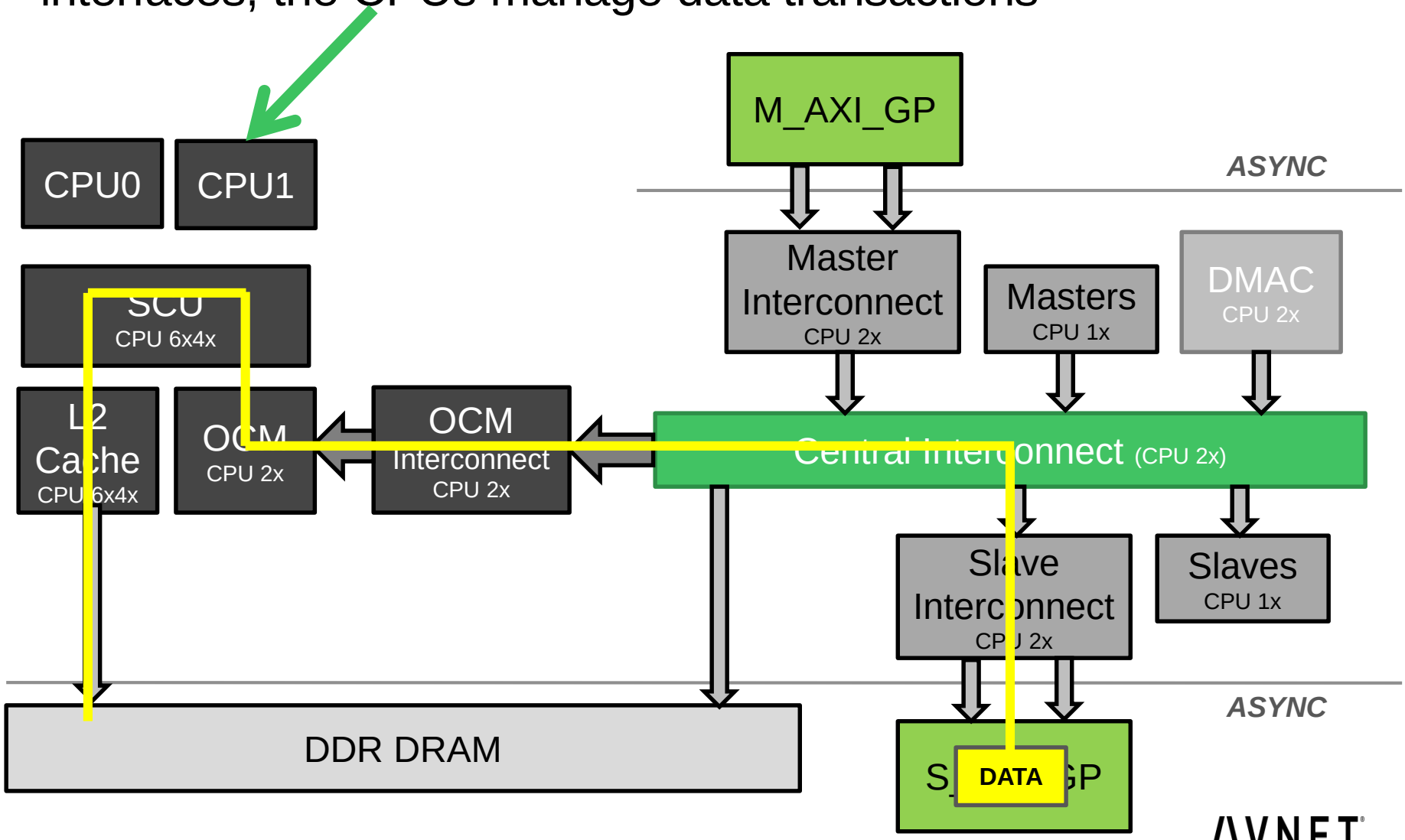
Flow

- CPU's setup DMA to handle data transaction
- CPU returns to work as normal
- CPU waits for interrupt from DMAC indicating transaction complete

AVNET

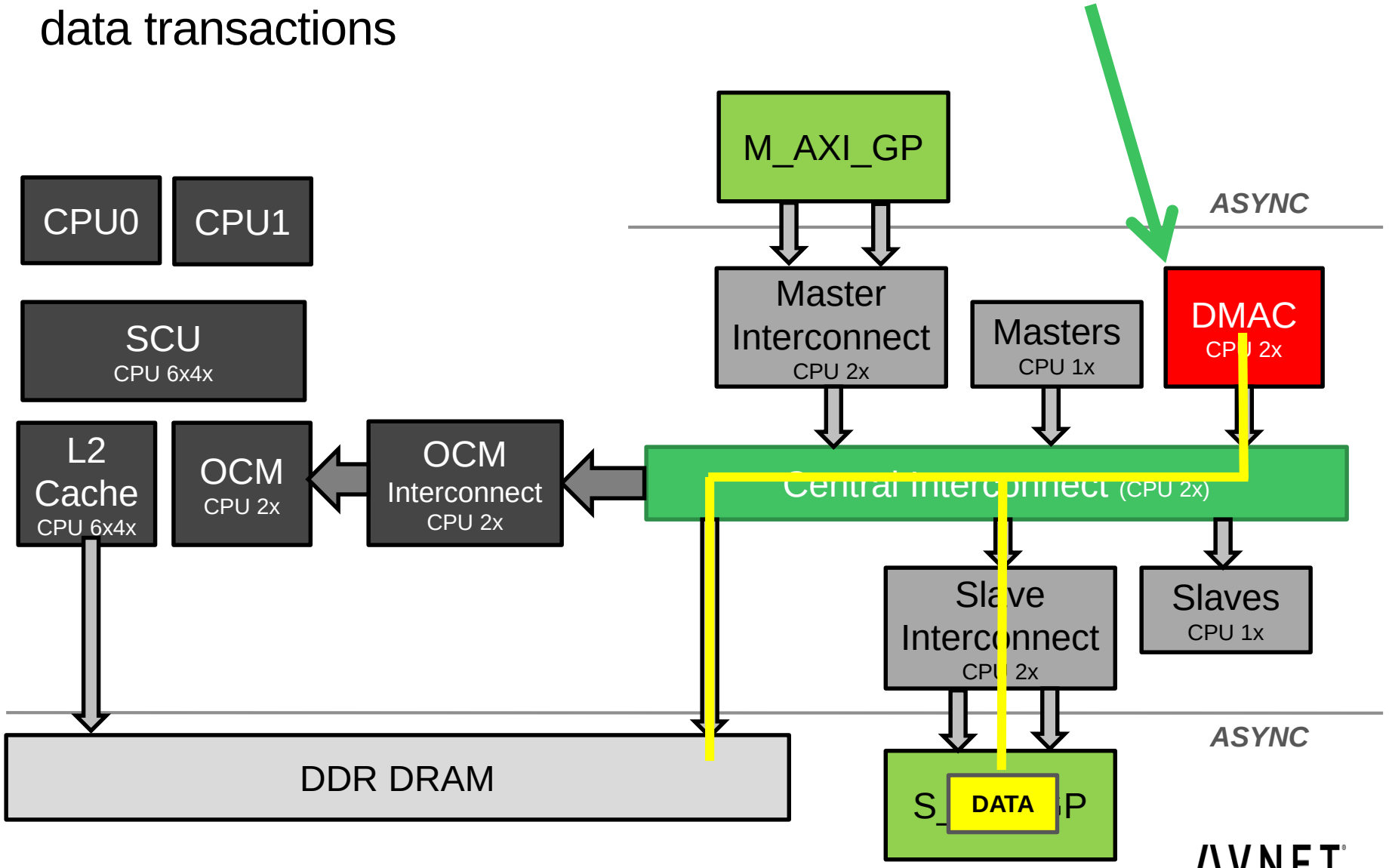
PS-PL Data Transfers

When data passes between the PS and PL via AXI GP interfaces, the CPUs manage data transactions



PS-PL DMA Data Transfers

When utilizing a DMA Controller (DMAC), the DMAC manages data transactions



PS DMAC R/W Options

The PS DMAC can move data between:

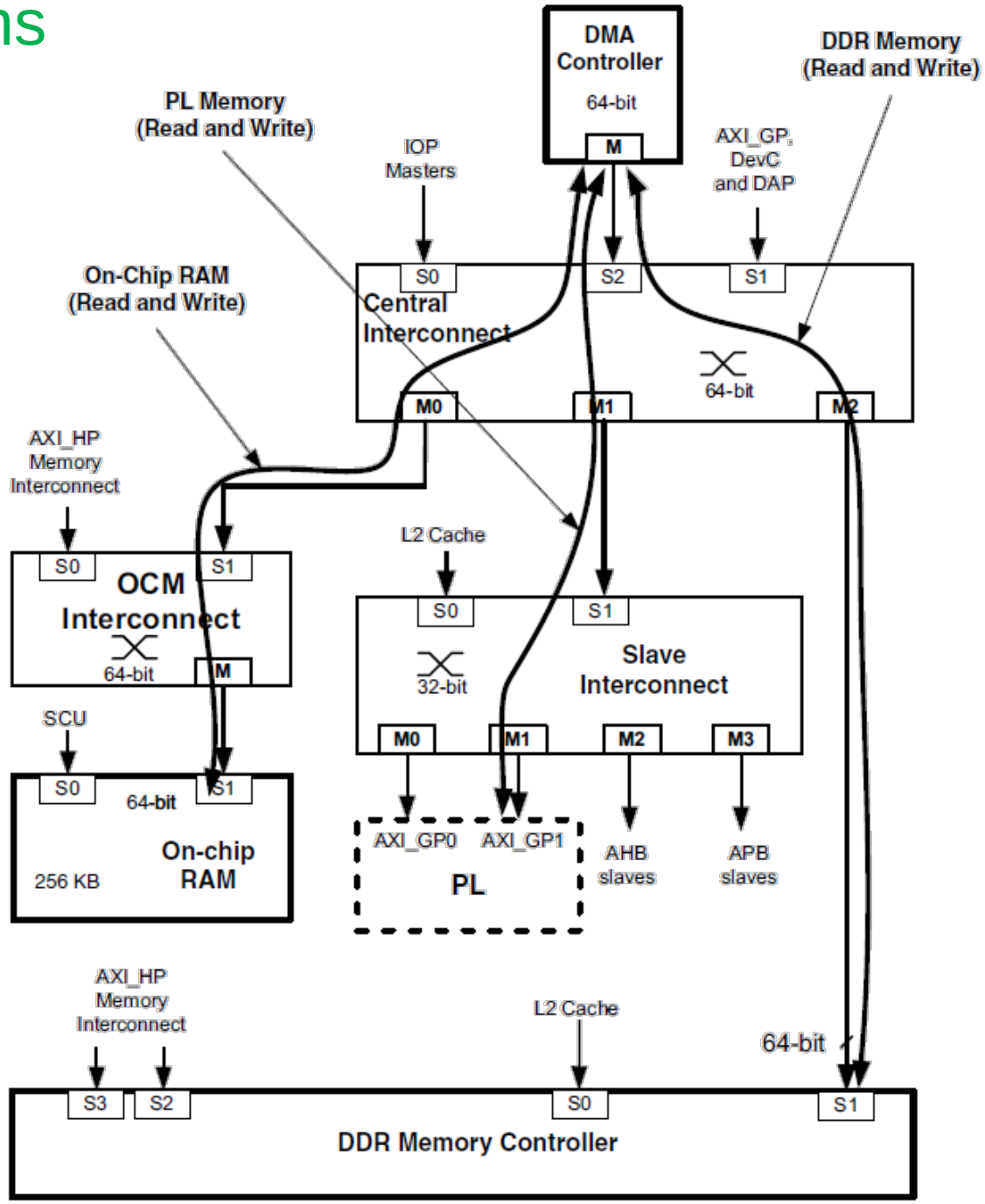
- On-chip Memory
- DDR Memory
- Slave PL Peripherals

Each path can be read or write

Common Transactions:

- Memory to Memory
- DDR Memory to/from PL Peripheral

Scatter Gather support

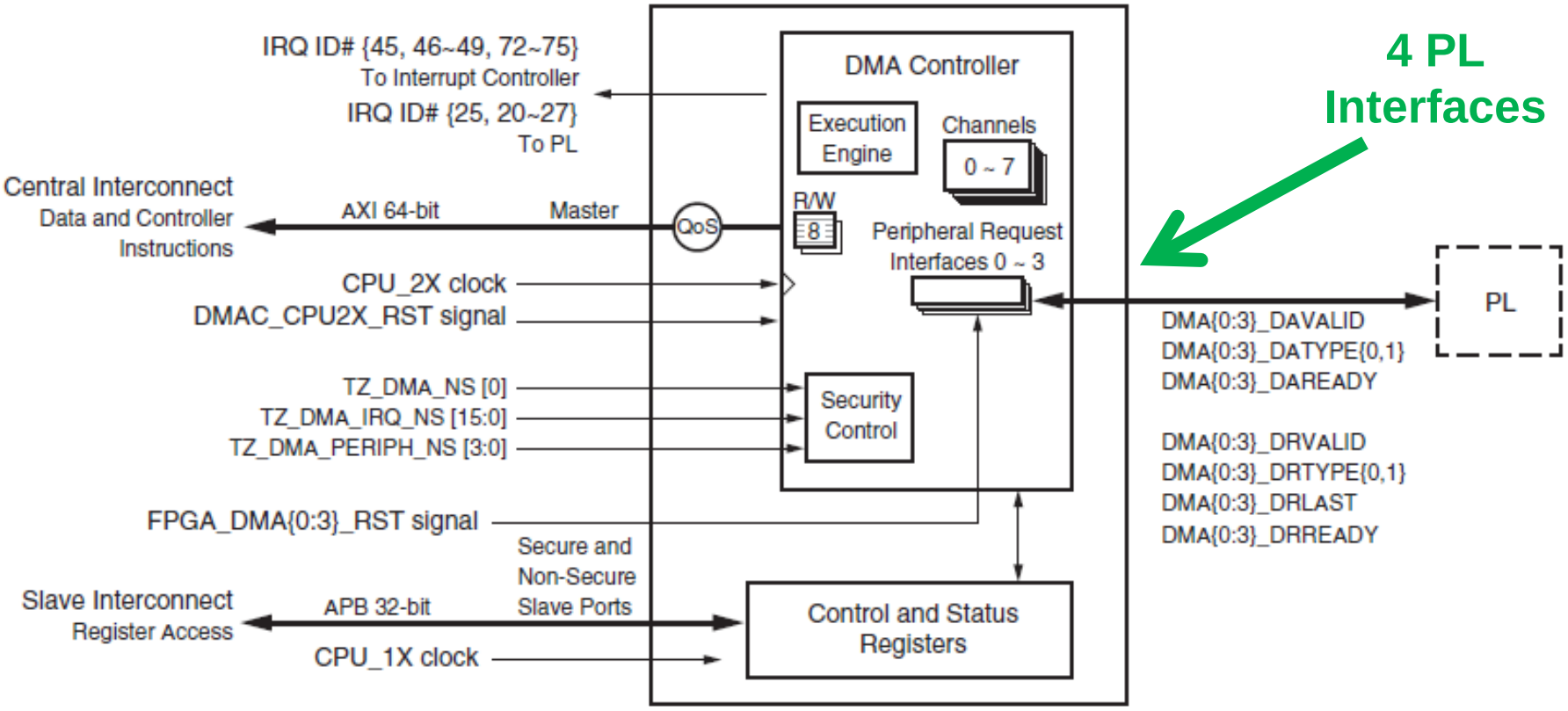


DMA Controller System Viewpoint

Eight concurrent DMA channels
- Four for the programmable logic
- Four for the processing system

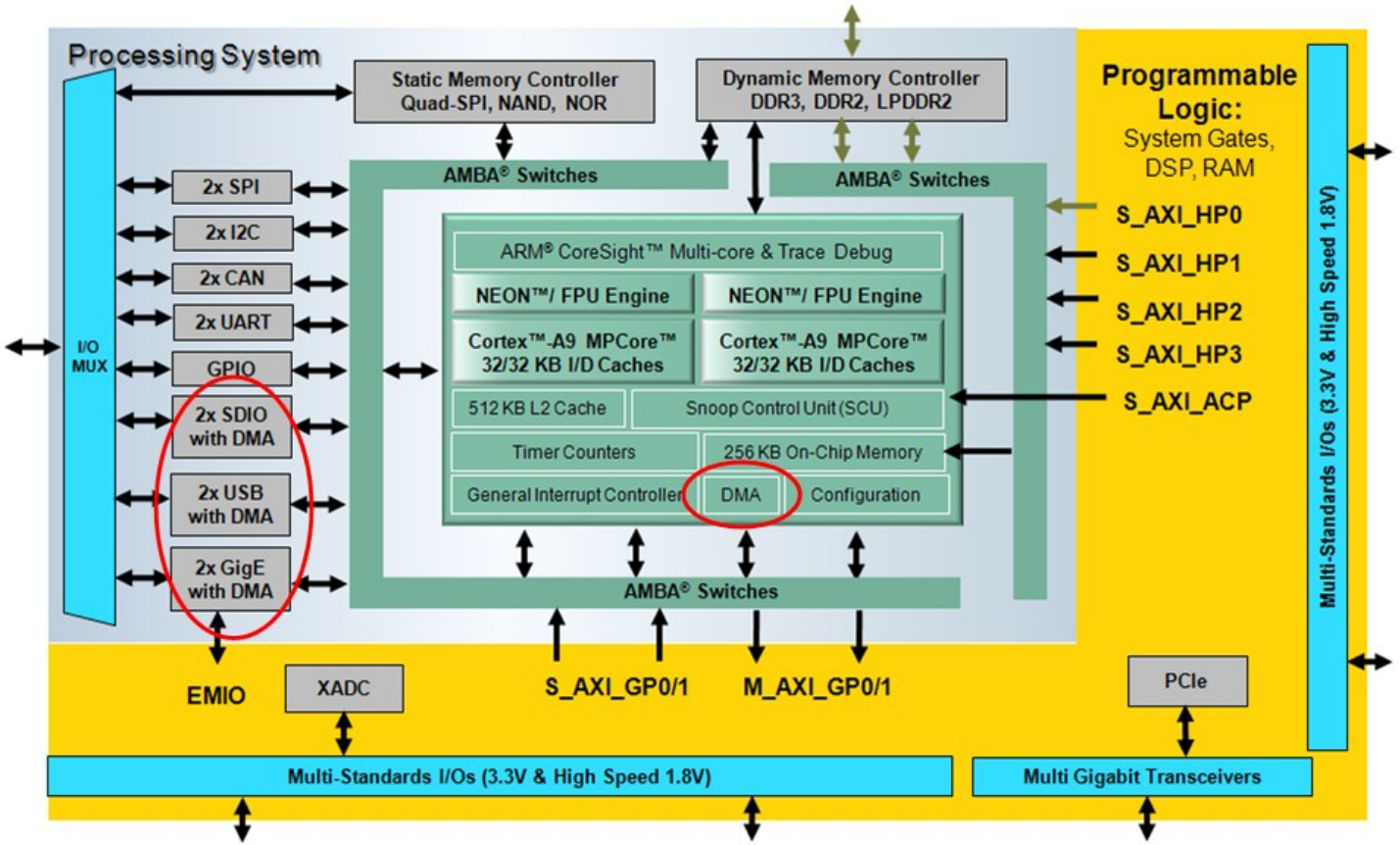
Eight interrupt lines

- **64 deep Multi-channel FIFO**
- **Support for both 32-bit and 64-bit transfers**



Additional PS DMA Controllers

DMA is intrinsically supported in USB, Ethernet, and SD-SDIO



DMA Instruction Execution Engine

Contains an instruction processing block that enables it to process program code that controls a DMA transfer

Maintains a separate state machine for each thread

Channel arbitration

- Round-robin scheme to service the active DMA channels
- Services the DMA manager prior of servicing the next DMA channel
- Changes to the arbitration process are not supported

Channel prioritization

- Responds to all active DMA channels with equal priority
- Changes to the priority of a DMA channel over any other DMA channels are not supported

Programming the PS DMA

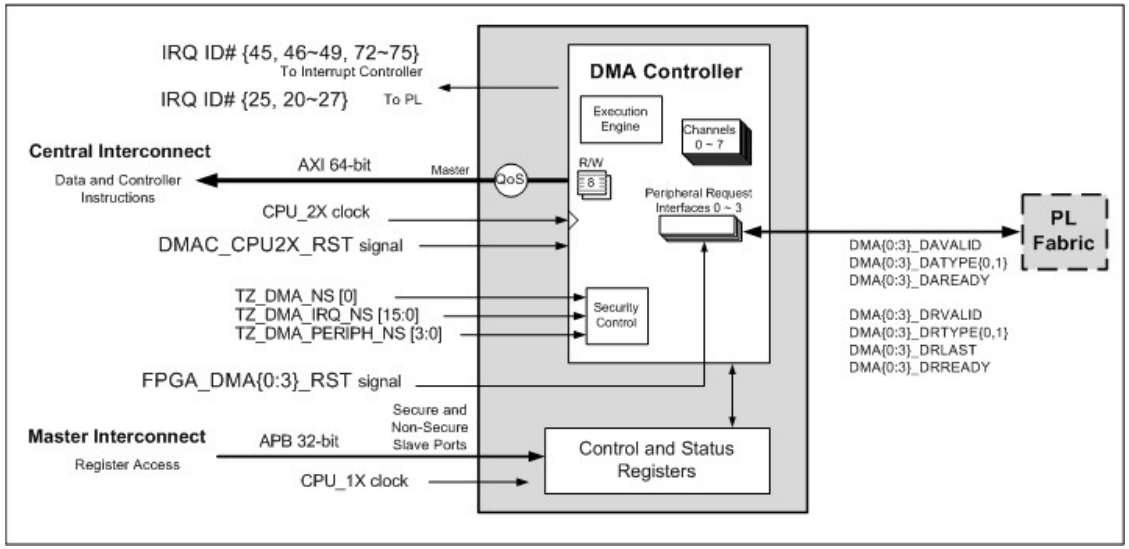
DMAs require a fair bit of programming

Similar to an assembly language

- Variable-length instructions 1-6 bytes
- Separate program counter for each channel

Documented in the Zynq-7000 All Programmable SoC Technical Reference Manual (UG585)

- 67 pages



Programming Guide for DMA Controller

Startup

- Example: Start-up Controller
  1. Configure Clocks
  2. Configure Security State
  3. Reset the Controller
  4. Create Interrupt Service Routine
  5. Execute DMA Transfers

Execute a DMA Transfer

1. Write Microcode into Memory for DMA Transfer
   a. Create a program for the DMA channel.
   b. Store the program in a region of system memory.
2. Start the DMA Channel Thread

Create ISR

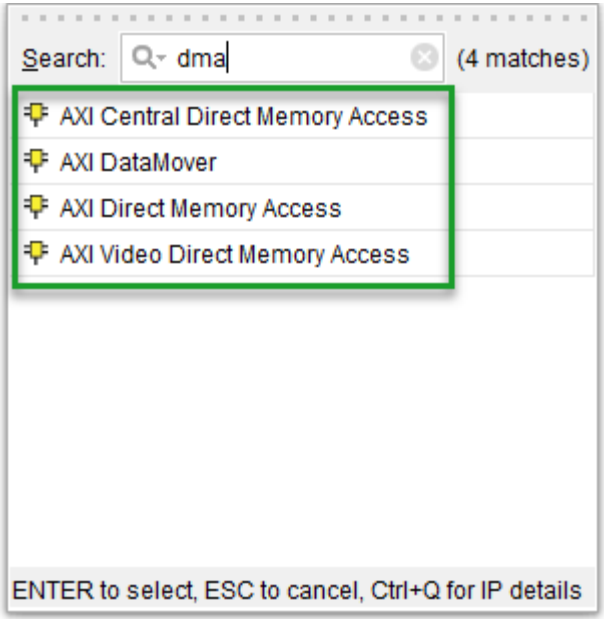
See UG585 for more details

AVNET

PL AXI Direct Memory Access

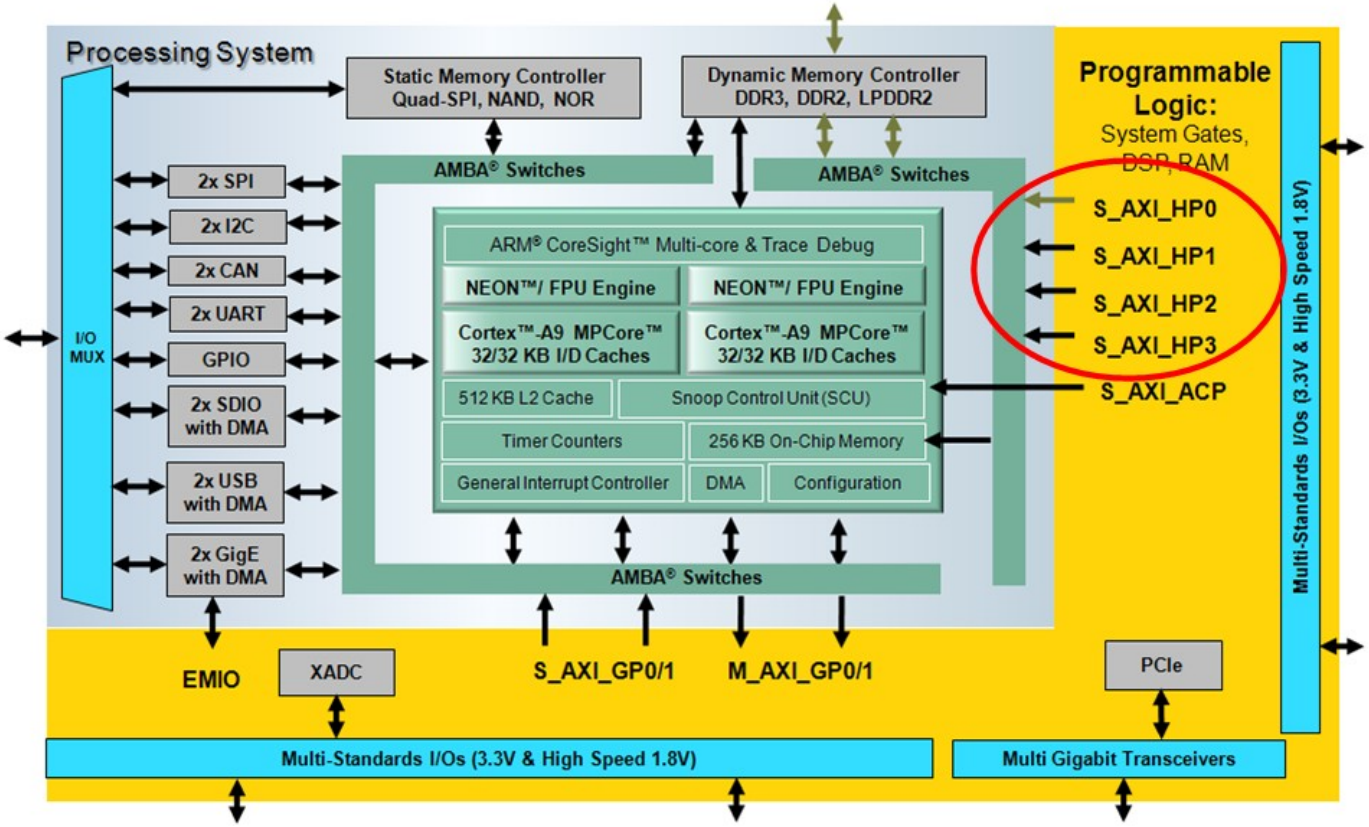
Four PL-based DMACs are available

- AXI Central DMA (CDMA)
  - Memory-to-memory operations
- AXI DMA
  - High Bandwidth DMA
  - Memory to/from AXI Stream-type peripherals
- AXI DataMover
  - FIFO Memory Mapped to Streaming
  - Streaming AXI interface alternative to traditional DMA, no Scatter Gather
- AXI Video DMA
  - Optimized for streaming video application to/from memory



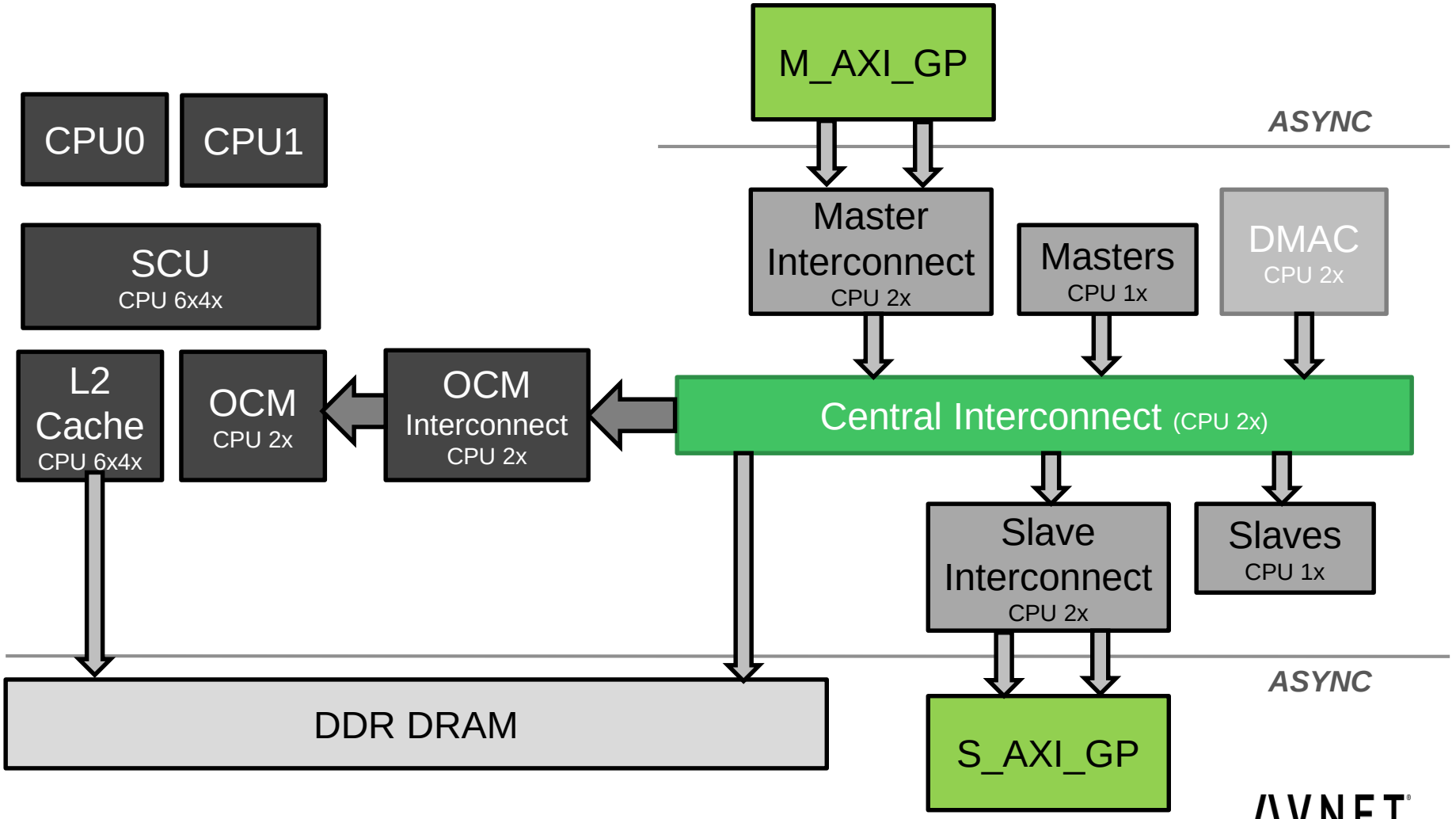
Support for HP PL DMA

PL-based DMA engines are used to transfer data from PL Masters to DDR and OCM



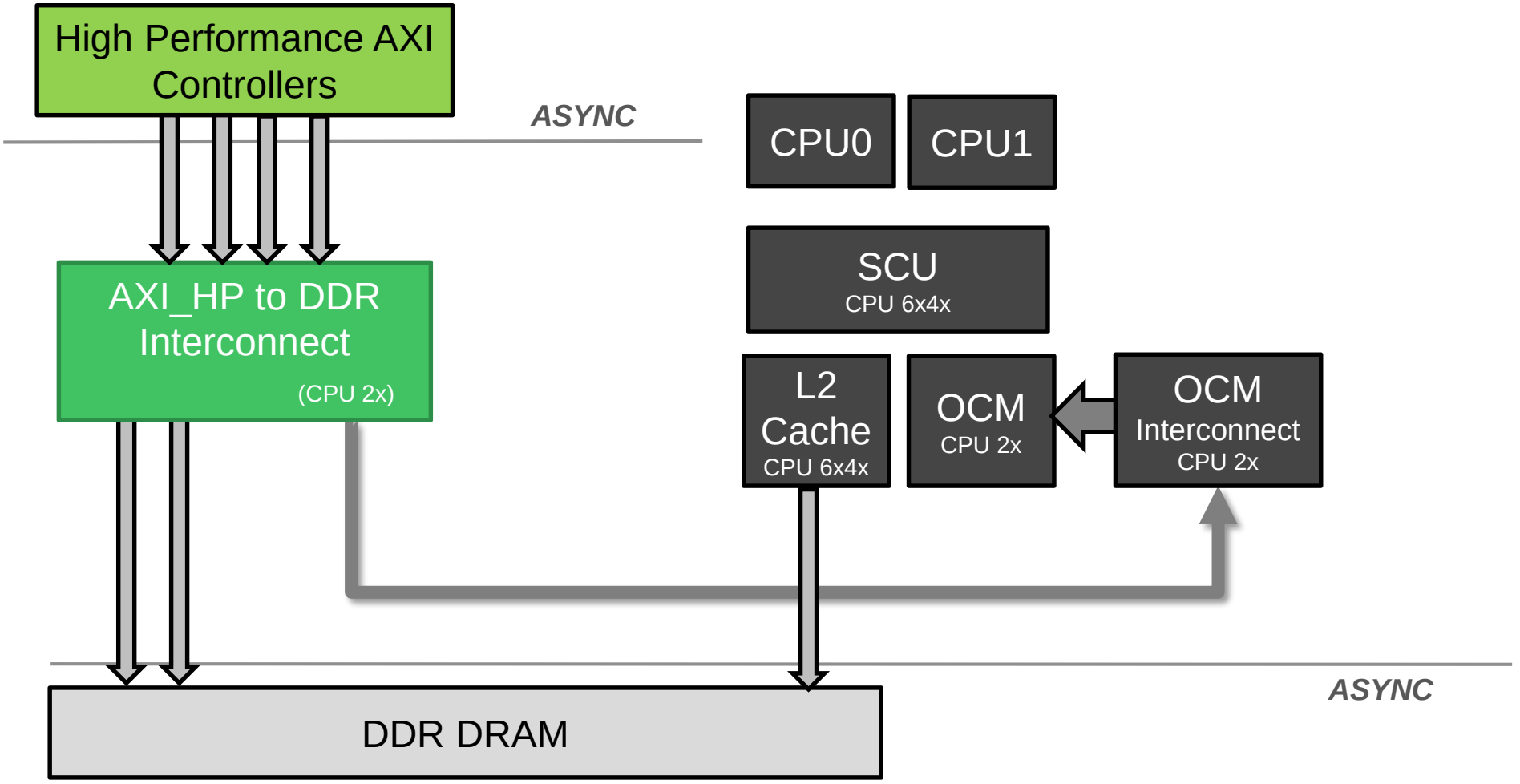
HP PS-PL Data Transfers

When data passes between the PS and PL via AXI HP interfaces, data can pass directly to Memory



HP PS-PL Data Transfers

When data passes between the PS and PL via AXI HP interfaces, data can pass directly to Memory



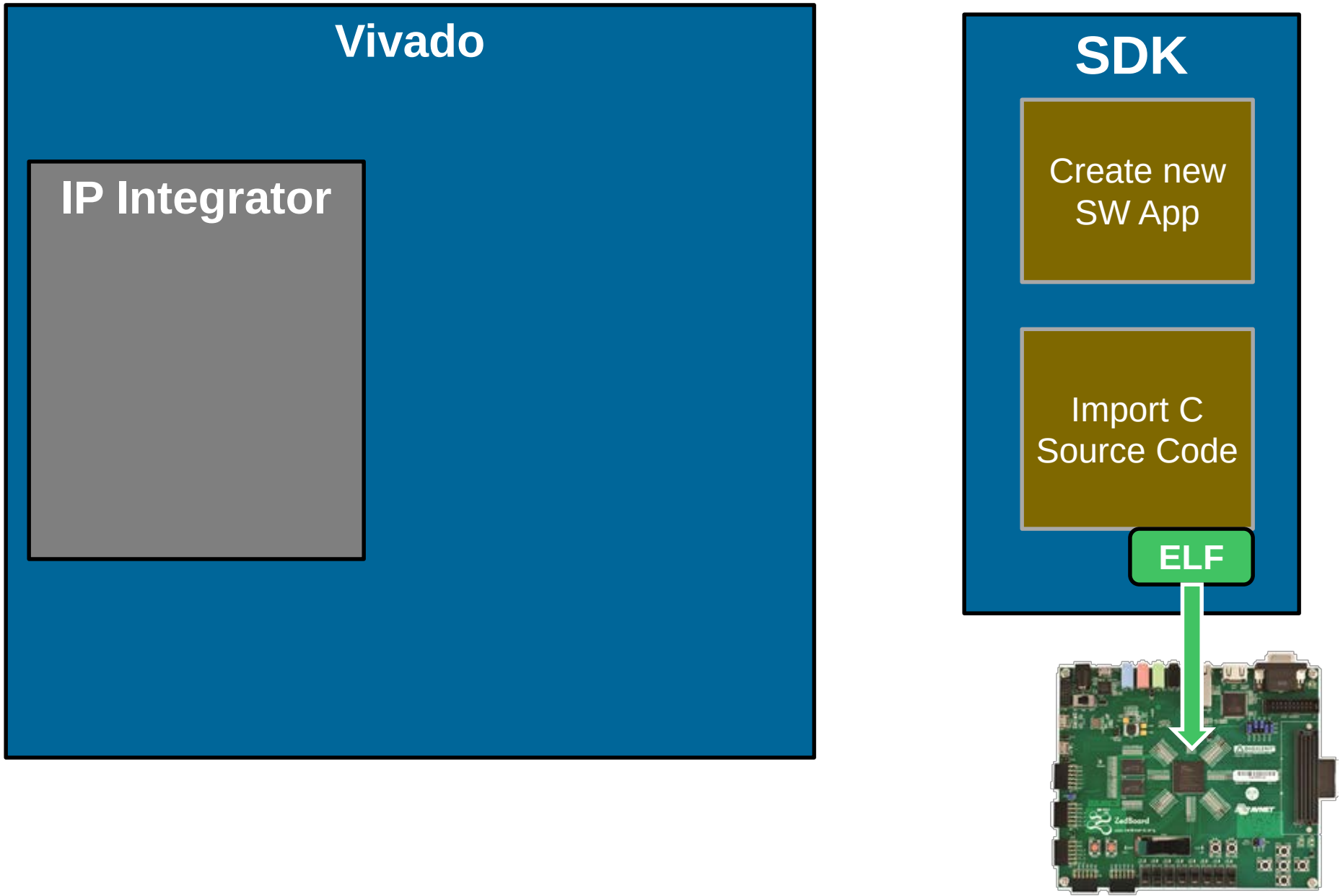
AVNET

Agenda

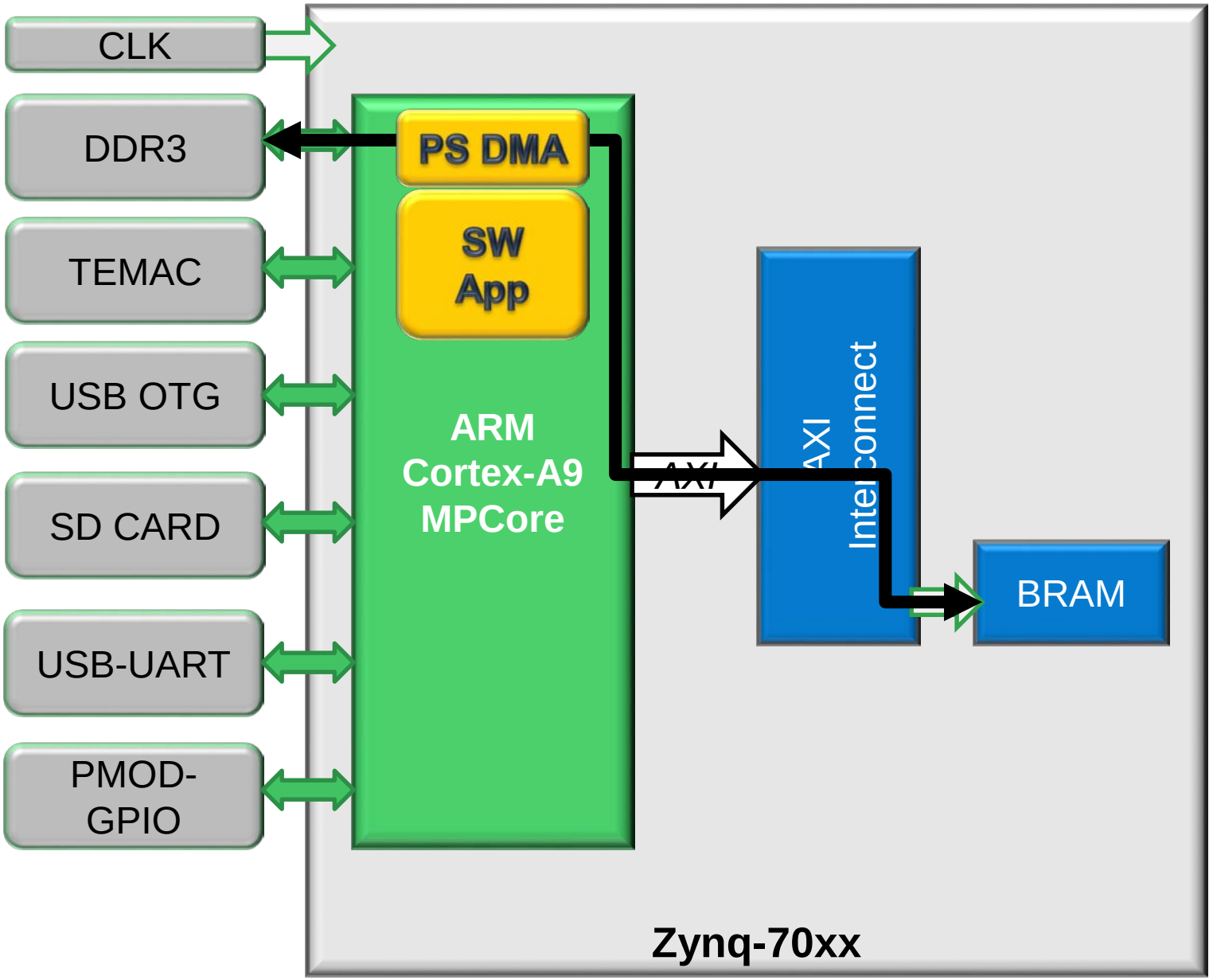
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - **Lab 6 – Improving Data flow between PL and PS utilizing PS DMA**
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Lab 6 – Access PS DMA Controller



Lab 6 – Access PS DMA



AVNET

Questions

Was a new BSP required for this application?

- No. We could have used the BSP created for Hello_World_BRAM in the past lab. That BSP included drivers for the PL BRAM. However, it's not necessarily bad practice to generate a new BSP for each new HW platform.

What is the performance increase when transferring 8192 bytes from BRAM to DDR3? Try again from DDR3 to DDR3? BRAM to BRAM?

- 11x, 3x, 20x (Numbers may vary)

What is the performance increase when transferring 256 bytes from BRAM to DDR3? Try again from DDR3 to DDR3? BRAM to BRAM?

- 10x, 1x, 18x (Numbers may vary)

Agenda

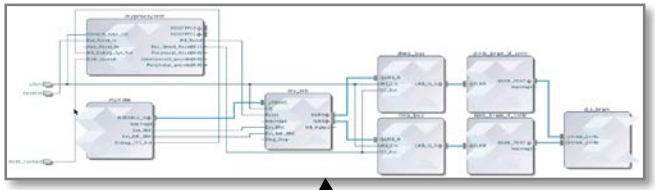
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

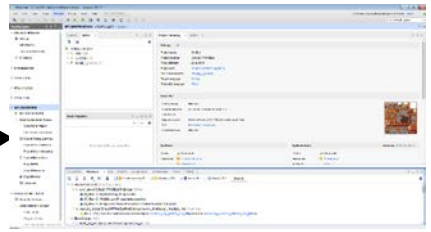
Expanding the IP Catalog

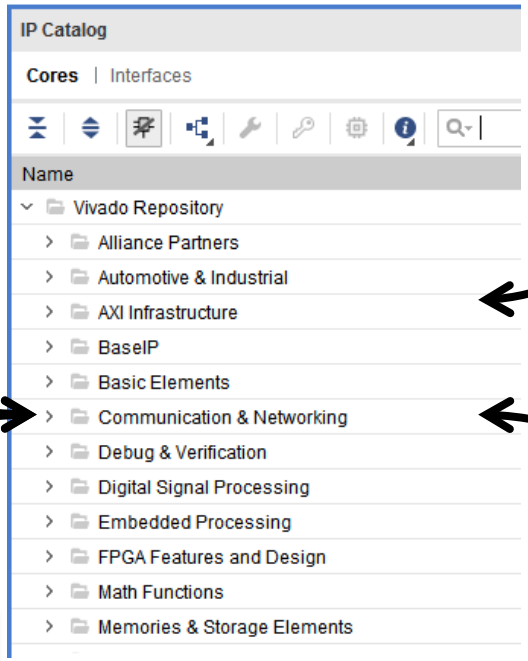
Wouldn't it be nice to have your own IP in the Vivado IPI Catalog?

Vivado IP Integrator

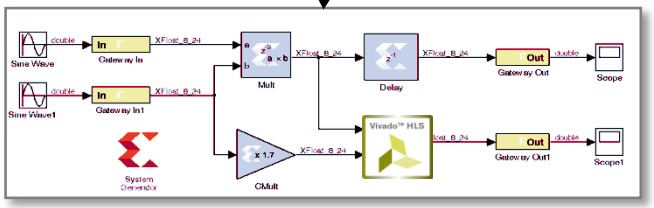


Vivado RTL Integration





Vivado IP Catalog



System Generator for DSP

Vivado HLS

AVNET

Vivado IP Overview

Vivado has a new and unified catalog with all IP in one place

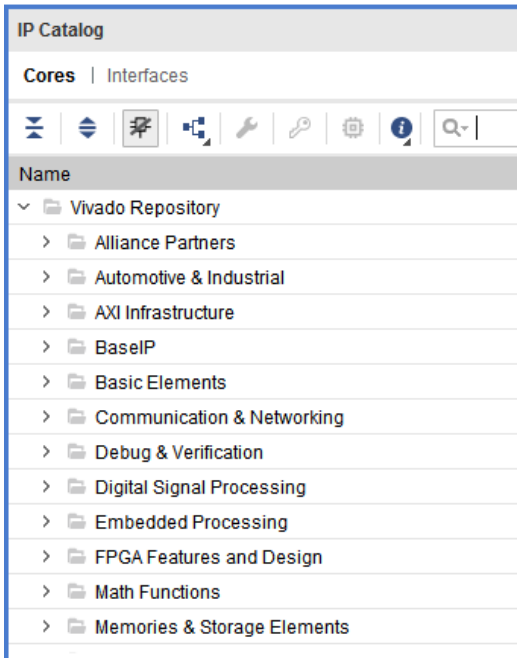
- The Vivado IP catalog uses IP-XACT to store metadata

IP can be added to the catalog by users (IP Packager)

All IP deliver IEEE P1735 simulation models as an output product

- Xilinxcorelib only required for support of legacy/non-updated IP cores

Streamlined versioning of Xilinx IP



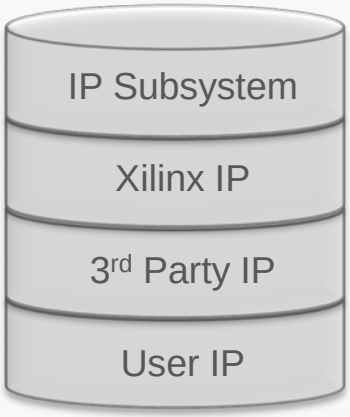
AVNET

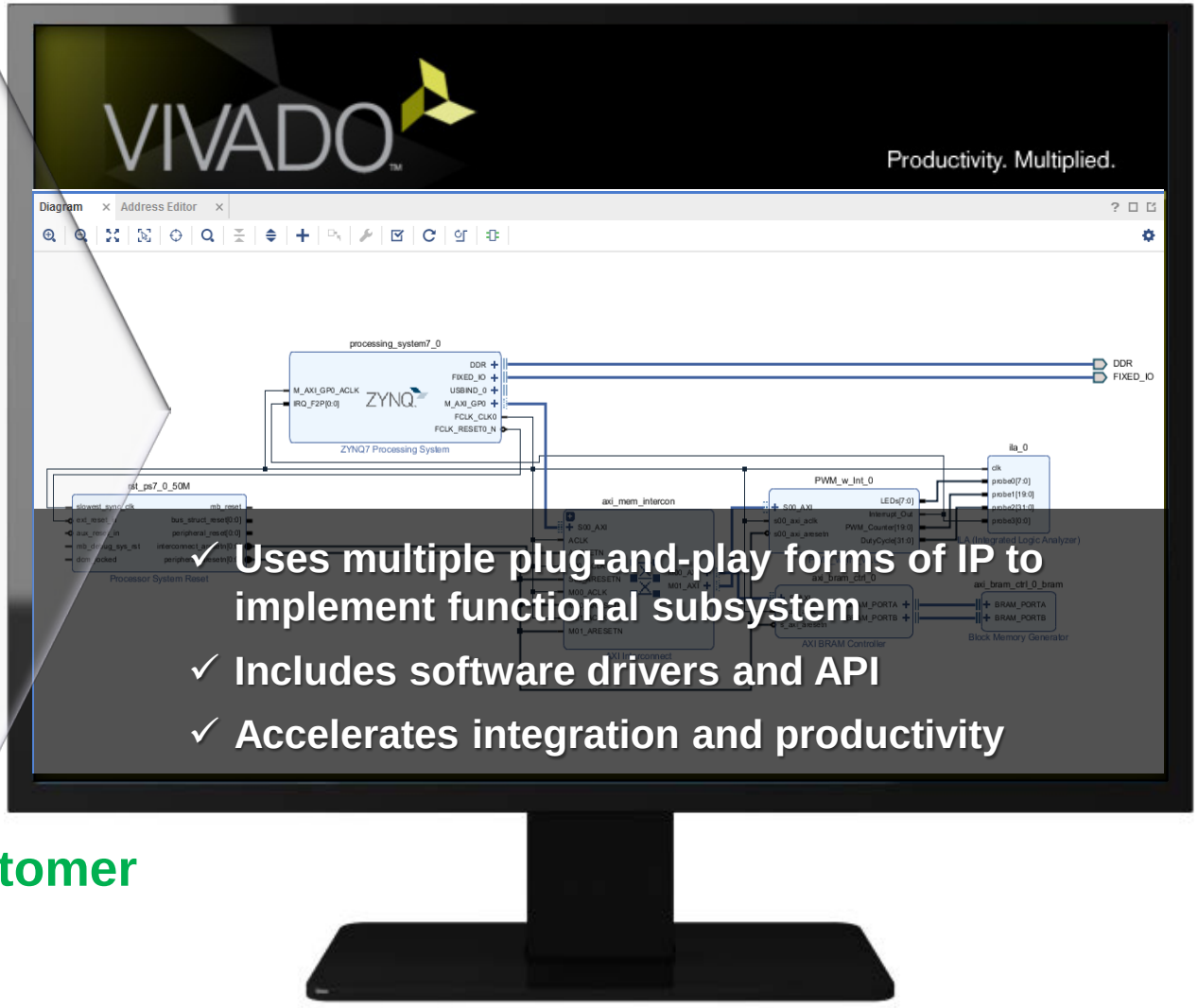
IP Integrator – IP Packager
Enabling Reuse and Delivering Fully Functional IP Subsystems

IP Packager

- Source (C, RTL, IP)
- Simulation models
- Documentation
- Example Designs
- Test bench

Standardized IP-XACT

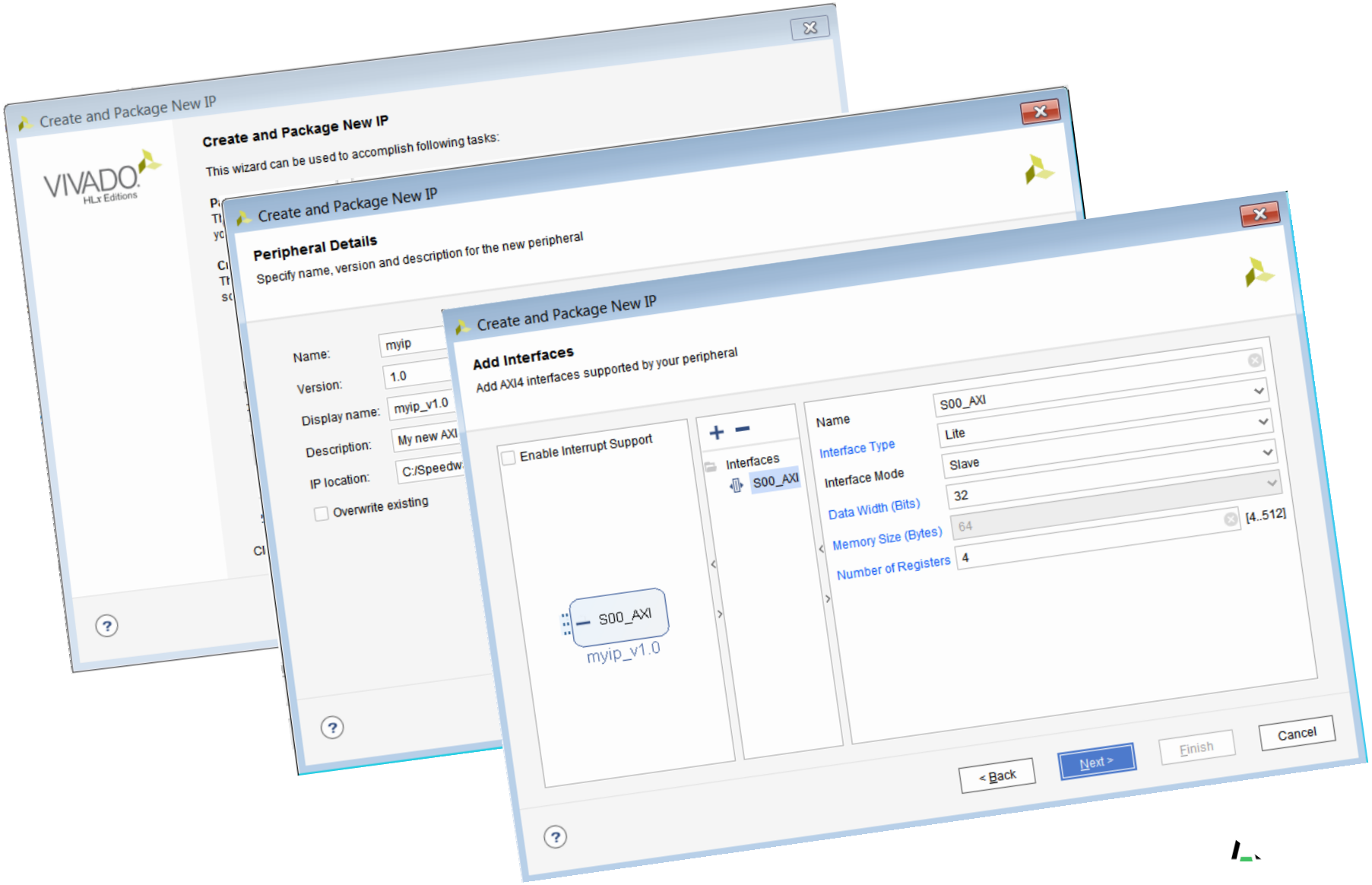




Thousands of IPI customer designs generated

Creating New IP with IP Packager

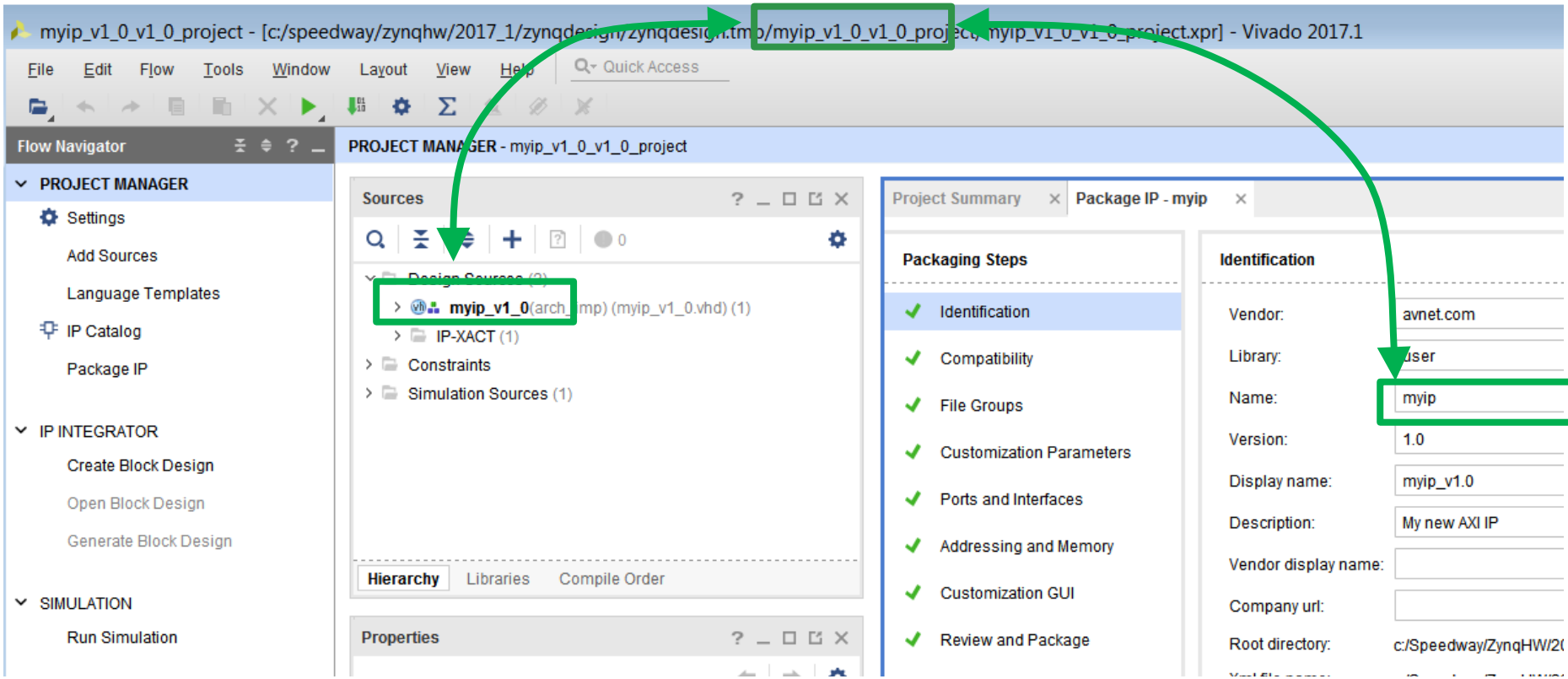
Vivado provides wizard to create and package IP



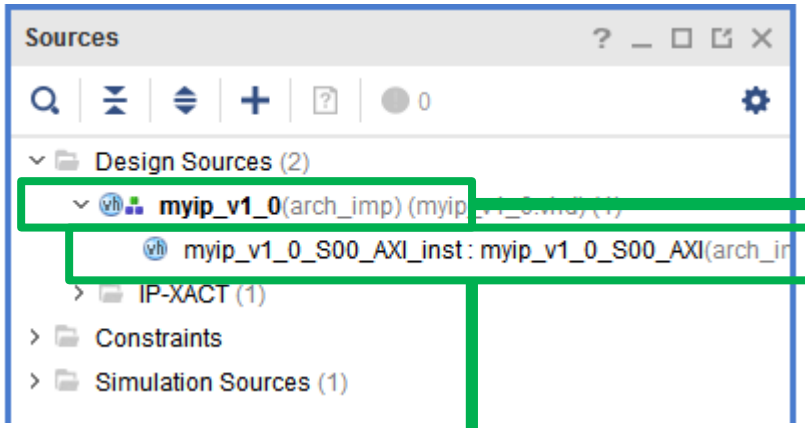
IP Packager creates a new Vivado Project

Inherits name of the IP, **edit_<ip_name>_v1_0.xpr**

- **Appends version number**



IP Packager Creates HDL Sources



Top-Level HDL Source

```
module NewIP_v1_0 #
(
    // Users to add ports here
    // Ports of Axi Slave Bus Interface S00_AXI
    input wire  s00_axi_aclk,
    input wire  s00_axi_aresetn,
    input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_awaddr,
    input wire [2 : 0] s00_axi_awprot,
    input wire  s00_axi_awvalid,
    output wire  s00_axi_awready,
    input wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_wdata,
    input wire [(C_S00_AXI_DATA_WIDTH/8)-1 : 0] s00_axi_wstrb,
    input wire  s00_axi_wvalid,
    output wire  s00_axi_wready,
    output wire [1 : 0] s00_axi_bresp,
    output wire  s00_axi_bvalid,
// Instantiation of Axi Bus Interface S00_AXI
    NewIP_v1_0_S00_AXI # (
        .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
        .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
    ) NewIP_v1_0_S00_AXI_inst (
        .S_AXI_ACLK(s00_axi_aclk),
        .S_AXI_ARESETN(s00_axi_aresetn),
        .S_AXI_AWADDR(s00_axi_awaddr),
        .S_AXI_AWPROT(s00_axi_awprot),
        .S_AXI_AWVALID(s00_axi_awvalid),
        .S_AXI_AWREADY(s00_axi_awready),
        .S_AXI_WDATA(s00_axi_wdata),
        .S_AXI_WSTRB(s00_axi_wstrb),
        .S_AXI_WVALID(s00_axi_wvalid),
        .S_AXI_WREADY(s00_axi_wready),
        .S_AXI_BRESP(s00_axi_bresp),
        .S_AXI_BVALID(s00_axi_bvalid),
        .S_AXI_BREADY(s00_axi_bready),
        .S_AXI_ARADDR(s00_axi_araddr),
        .S_AXI_ARPROT(s00_axi_arprot),
        .S_AXI_ARVALID(s00_axi_arvalid),
        .S_AXI_ARREADY(s00_axi_arready),
        .S_AXI_RDATA(s00_axi_rdata),
        .S_AXI_RRESP(s00_axi_rresp),
        .S_AXI_RVALID(s00_axi_rvalid),
```

Slave AXI-4-Lite Interface

```
module NewIP_v1_0_S00_AXI #
(
    // Users to add parameters here

    // User parameters ends
    // Do not modify the parameters beyond this line

    // Width of S_AXI data bus
    parameter integer C_S_AXI_DATA_WIDTH    = 32,
    // Width of S_AXI address bus
    parameter integer C_S_AXI_ADDR_WIDTH    = 4
)
(
    // Users to add ports here

    // User ports ends
    // Do not modify the ports beyond this line
```

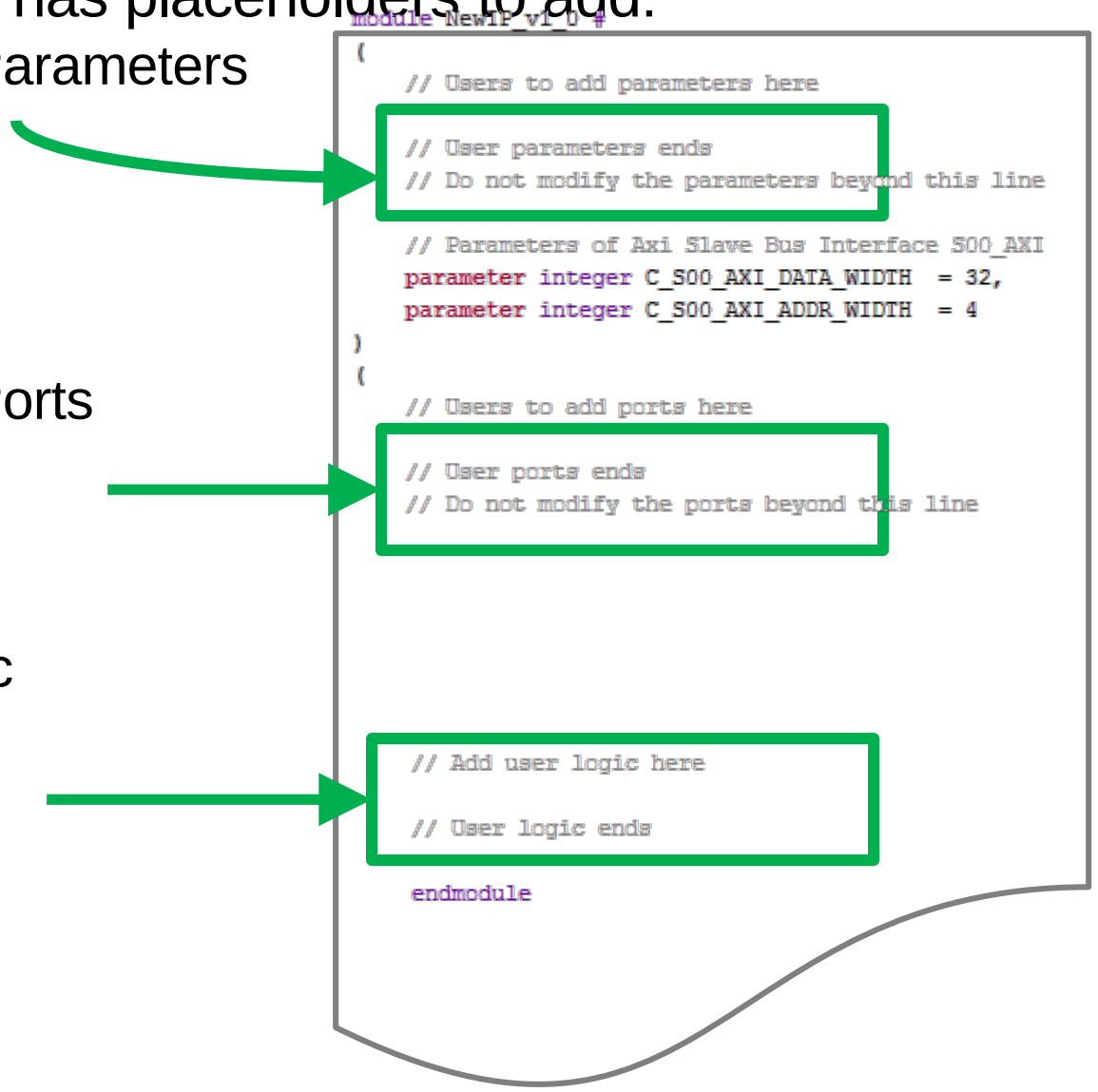
Instantiation

AVNET

Top-Level HDL

Top-Level HDL has placeholders to add:

- External Parameters
- External Ports
- User Logic



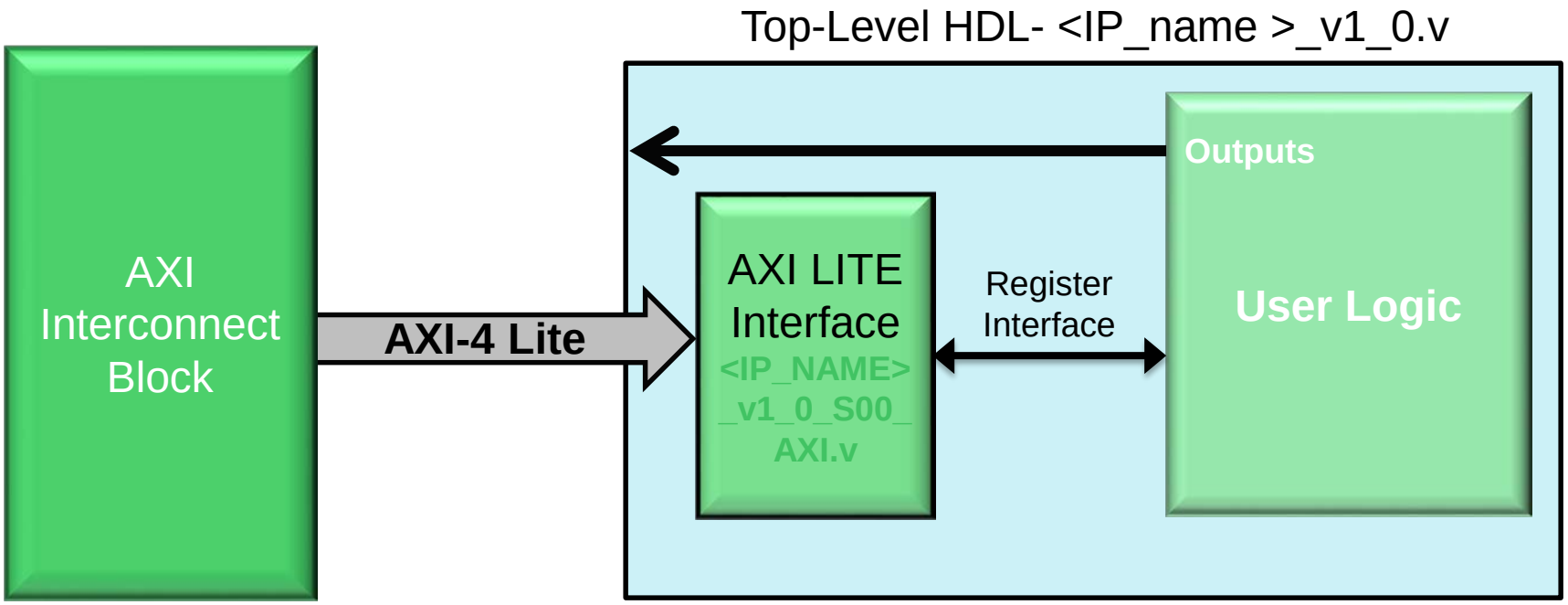
AVNET

Block Diagram View of Project

<IP_NAME>_v1_0_S00_AXI.v is wrapper file containing AXI interface

- Abstracts AXI interface to simple registers
- Requires editing when exporting external ports

User_logic is where custom HDL code goes

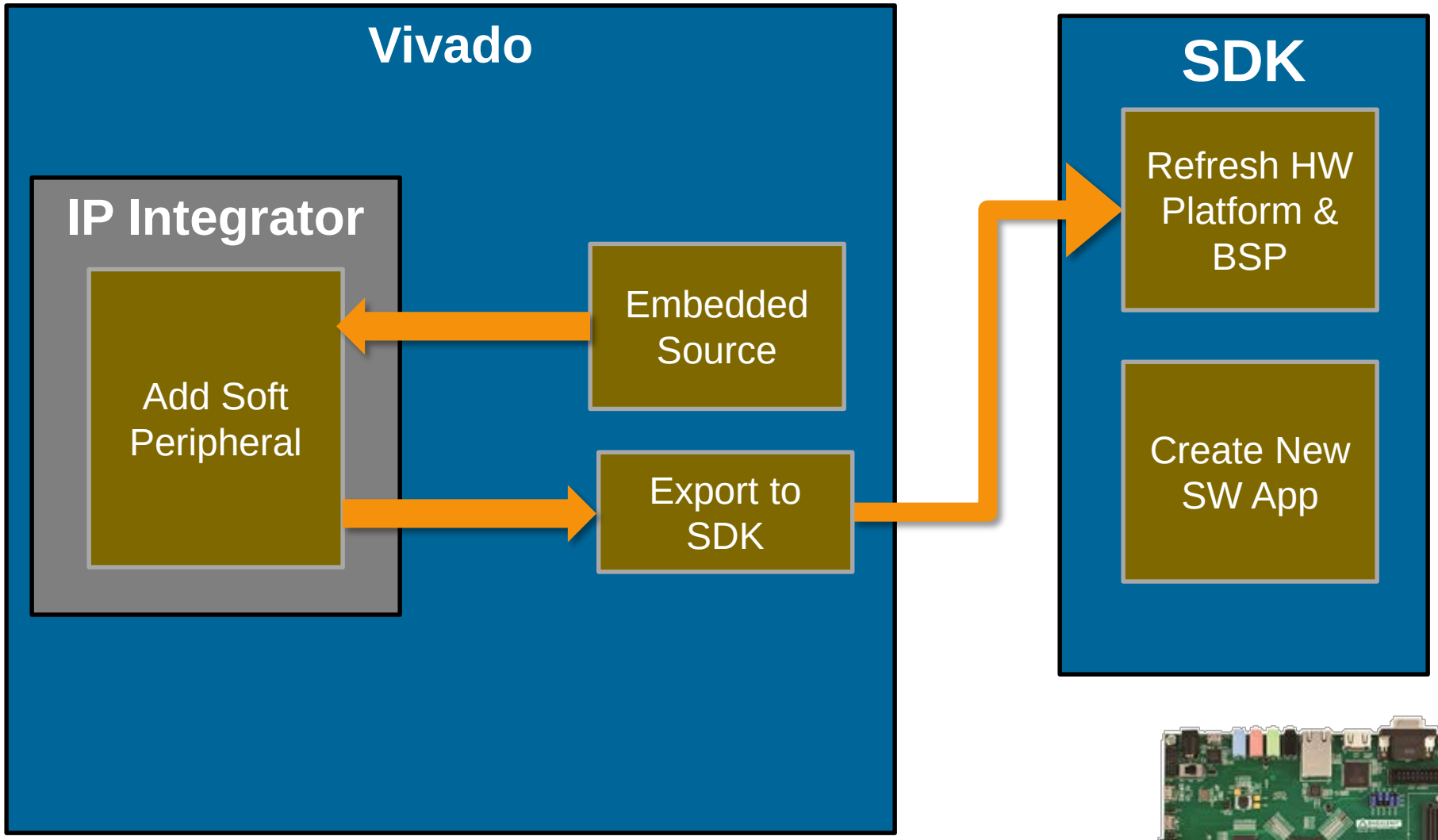


Agenda

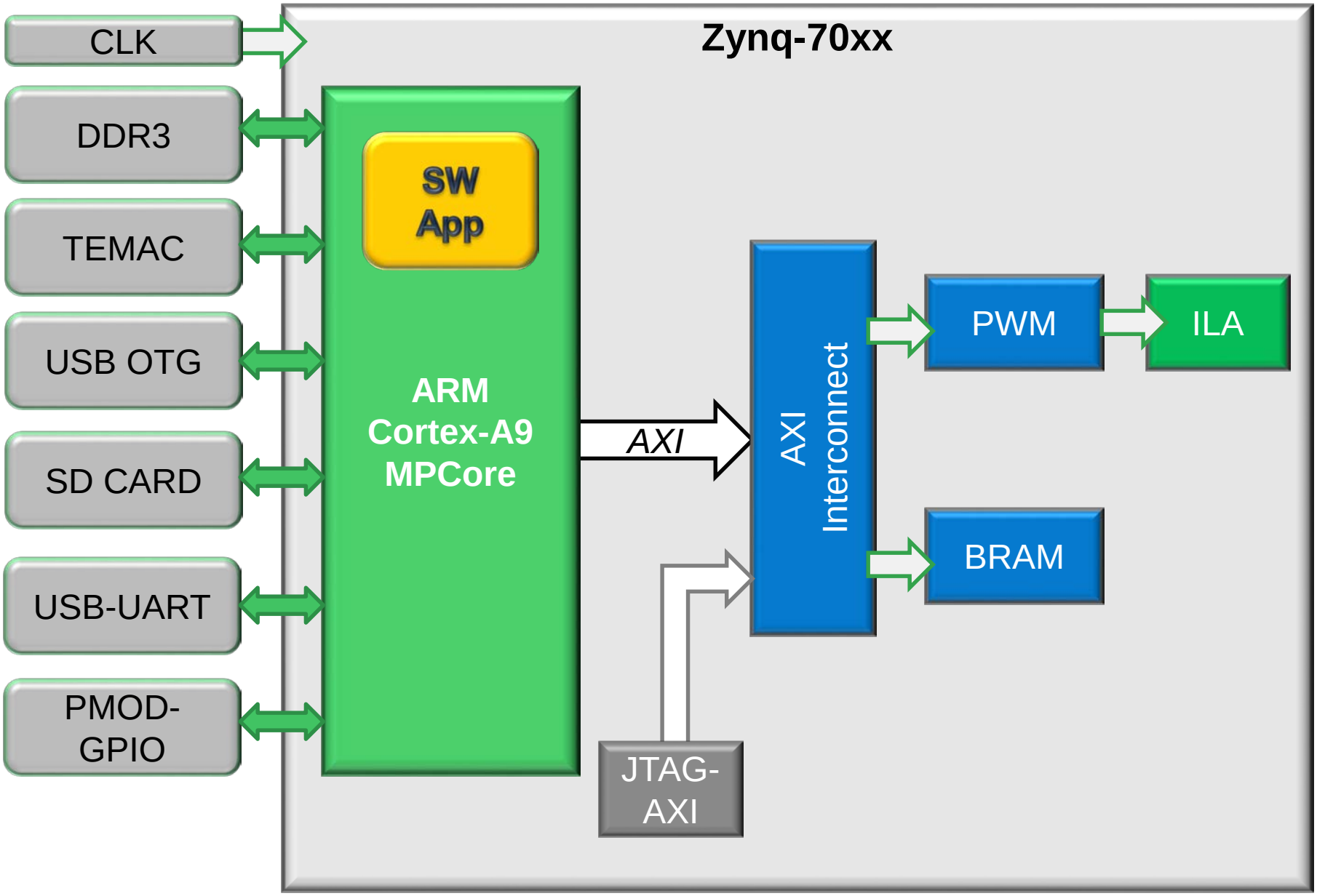
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - **Lab 7 – Adding Custom IP to the Vivado IP Catalog**
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

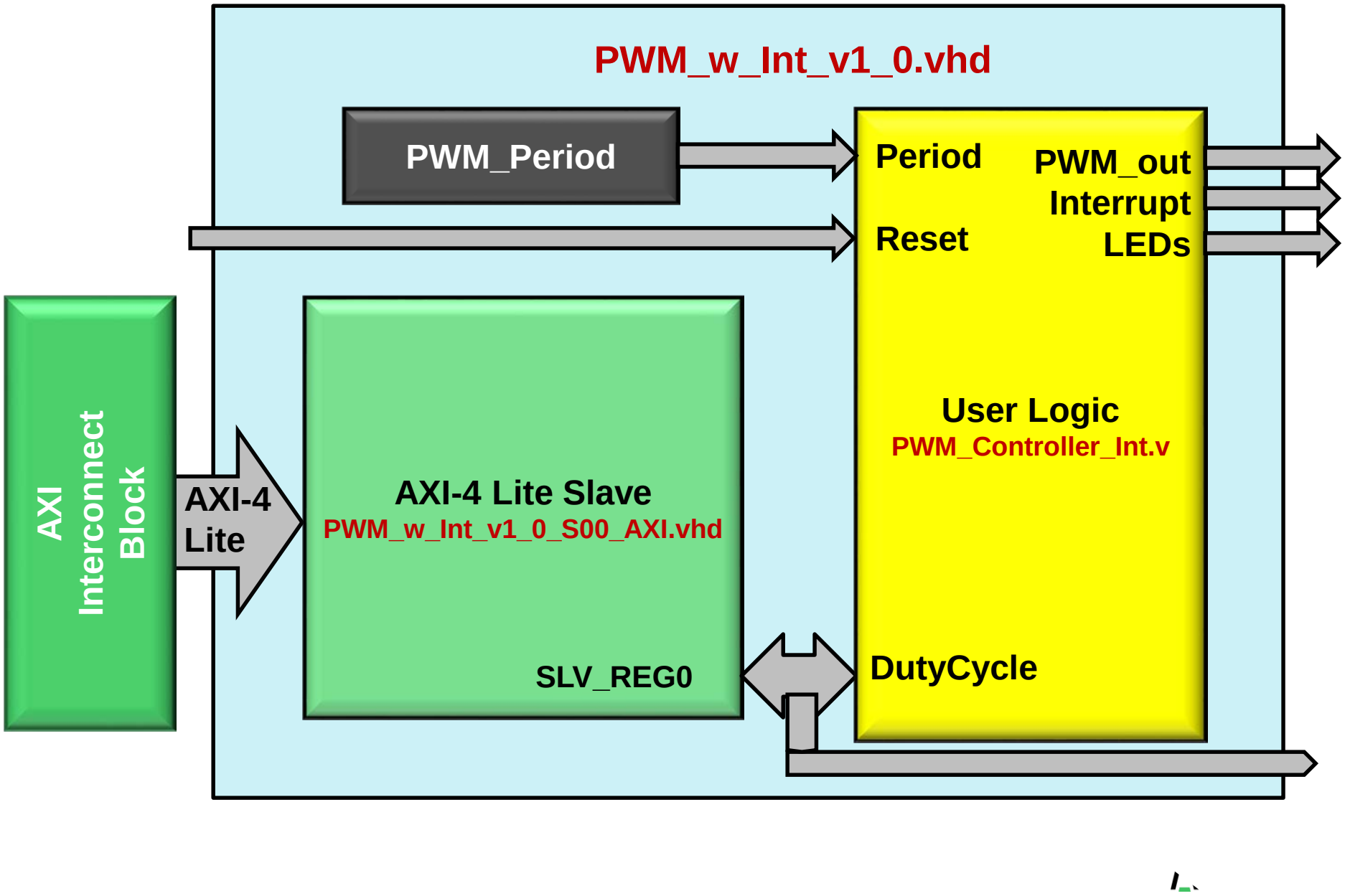
Lab 7 – Add Custom PL Peripheral



Lab 7 – Add Custom PL Peripheral



IP Core Finished



Questions

Where is the IP project located?

- In the same directory as our current project:
  C:/Speedway/ZynqHW/2017_1/ZynqDesign

How many AXI Masters and Slaves can be added to the AXI Interconnect?

- *16 and 16*

How many interrupts can be generated from fabric resources?

- *8 per M_AXI_GP = 16*
- *Plus nFIQ & nIRQ per core = 4*
- *20 Total*

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

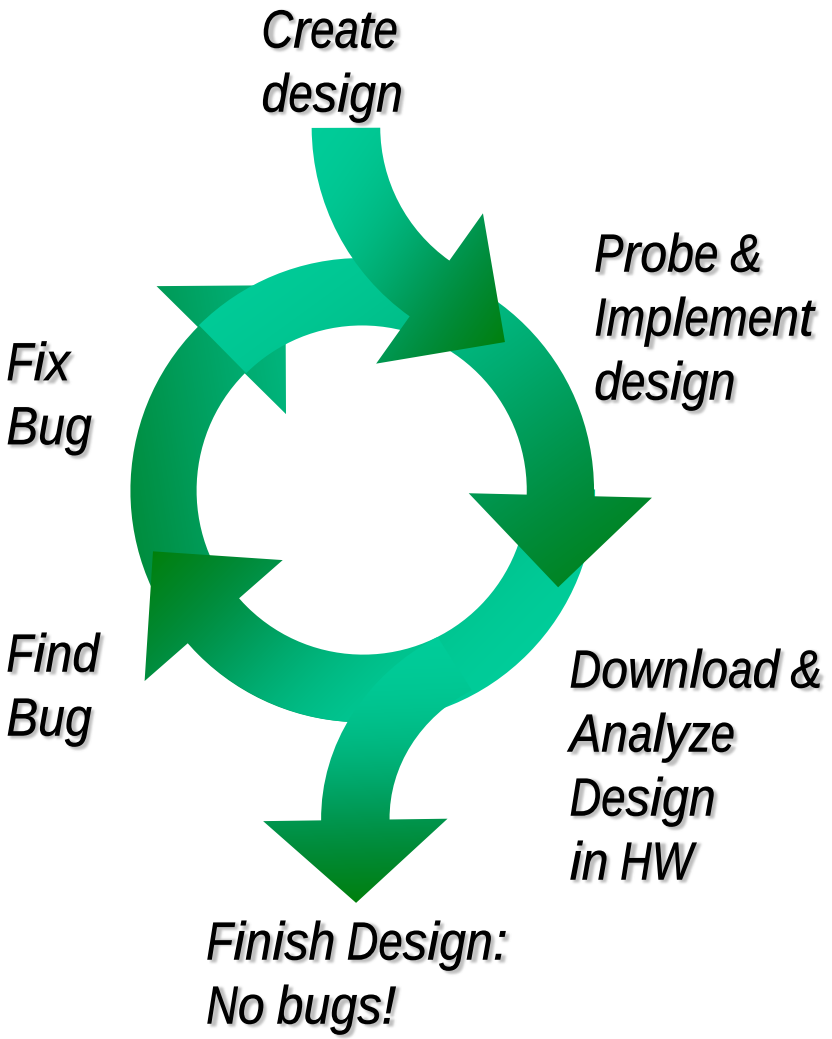
Recommended Debug Methodology

Engineers are trained to solve problems logically
- Break a problem into smaller parts
- Simplify by reducing variables & variation
- Make a prediction, verify the results
- Plan how and where to debug early in the design cycles

FPGA Design is an Iterative Process

Debugging is an iterative process

1) Probe: Adding/modify debug probes
2) Implement: Compile design w/probes
3) Analyze: Look for bugs using probes
4) Fix: Fix any bugs, repeat as necessary



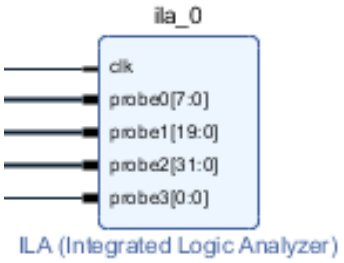
The reconfigurable nature of FPGAs enables the iterative debug process

AVNET

Vivado Logic Debug IP

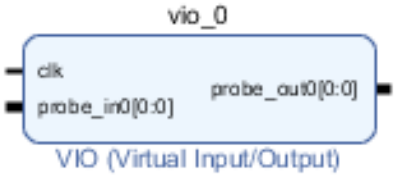
ILA 6.2

- Integrated Logic Analyzer debug IP core
- Netlist insertion support
- HDL instantiation support
- Added support for advanced trigger and capture features



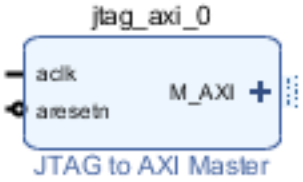
VIO 3.0

- Vivado native Virtual Input / Output
- HDL instantiation support



JTAG2AXI 1.2

- JTAG to AXI Master debug IP core
- HDL instantiation support
- IP Integrator block designs and HDL designs

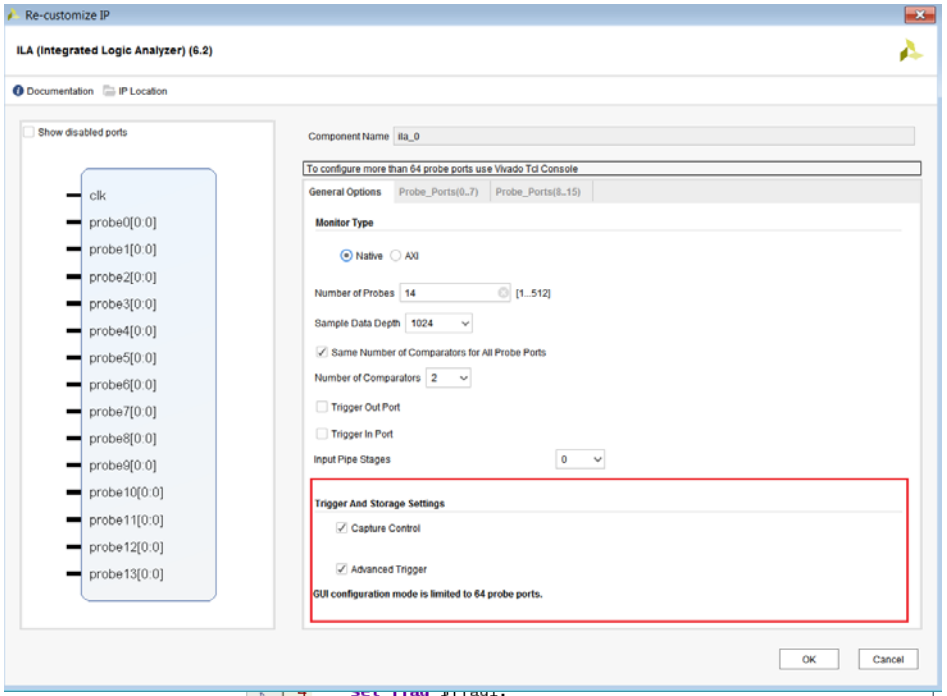


AVNET

ILA 6.2 Advanced Trigger Features

Advanced trigger state machine:

- Fully programmable at run-time
- Up to 16 states
- 3-way branching per state
- Up to four comparators per PROBE input
- Four programmable counters (reset, increment, conditional compare)
- Four programmable flags (set, clear, monitor in GUI)



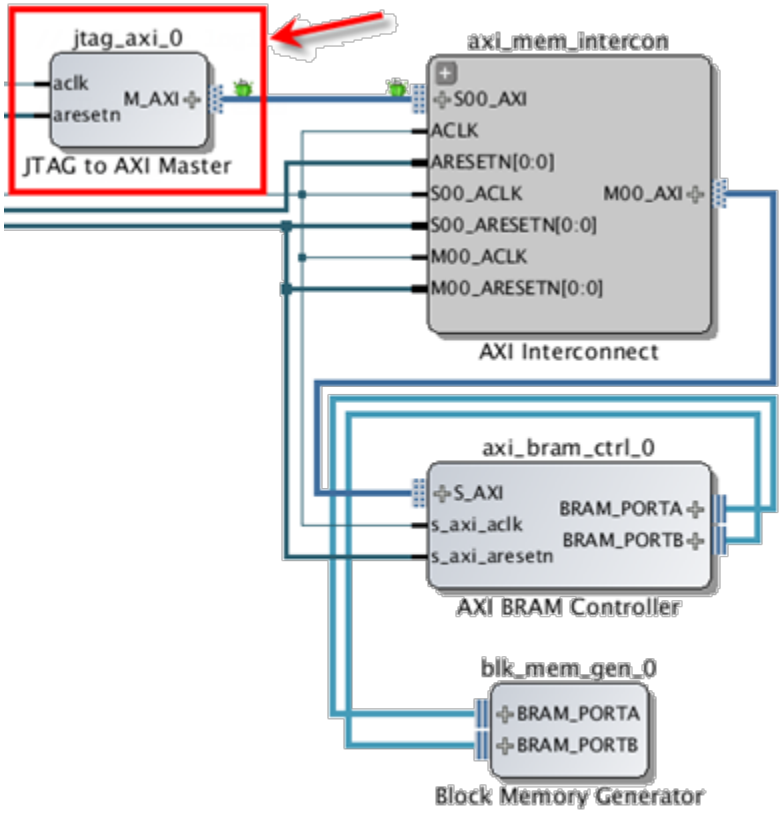

```
    goto wait_for_second_button_push;
  elseif (vio_slice1_fb == 1'bR) then
    increment_counter $counter0;
    goto wait_for_first_button_push_3_times;
  else
    goto wait_for_first_button_push_3_times;
  endif

state wait_for_second_button_push:
  if (vio_slice1a_fb == 1'bR) then
    set_flag $flag2;
    goto trigger_on_counter_value;
  else
    goto wait_for_second_button_push;
  endif

state trigger_on_counter_value:
  if (fast_cnt_count == 32'hfff_ffff) then
    trigger;
  else
    goto trigger_on_counter_value;
  endif
```

JTAG to AXI Master

- Interact with AXI-based system
  - without writing microprocessor code
- Connects to AXI or AXI-Lite interfaces
- Can be used in both IP Integrator block designs and HDL designs
- Vivado run-time Tcl commands to create and run AXI transactions
  - help *hw_axi*



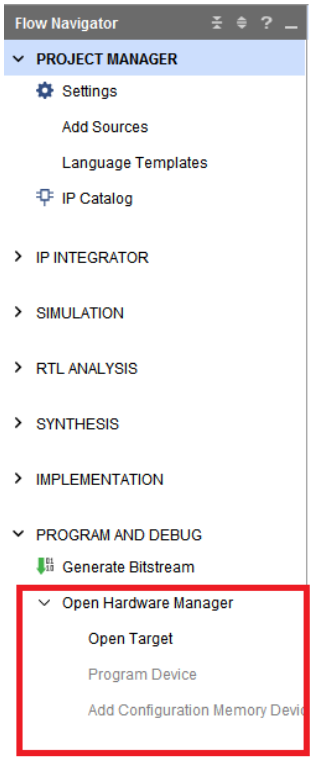
AVNET

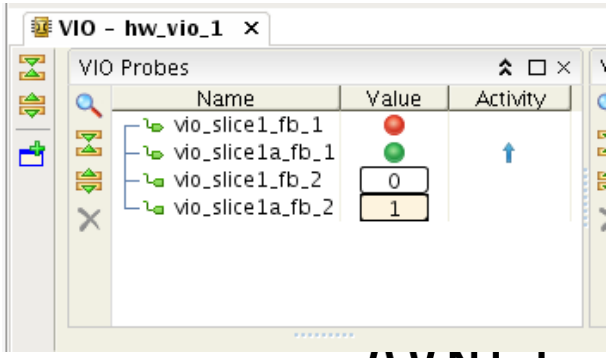
Hardware Manager & GUI Ease of Use

Hardware Manager and related tasks in Flow Navigator improve flow and highlight next steps
- Not another tool that needs to be opened

Simplified Debug Probes window for ease of finding probe information

ILA and VIO “dashboards” in GUI provide better organization and customization

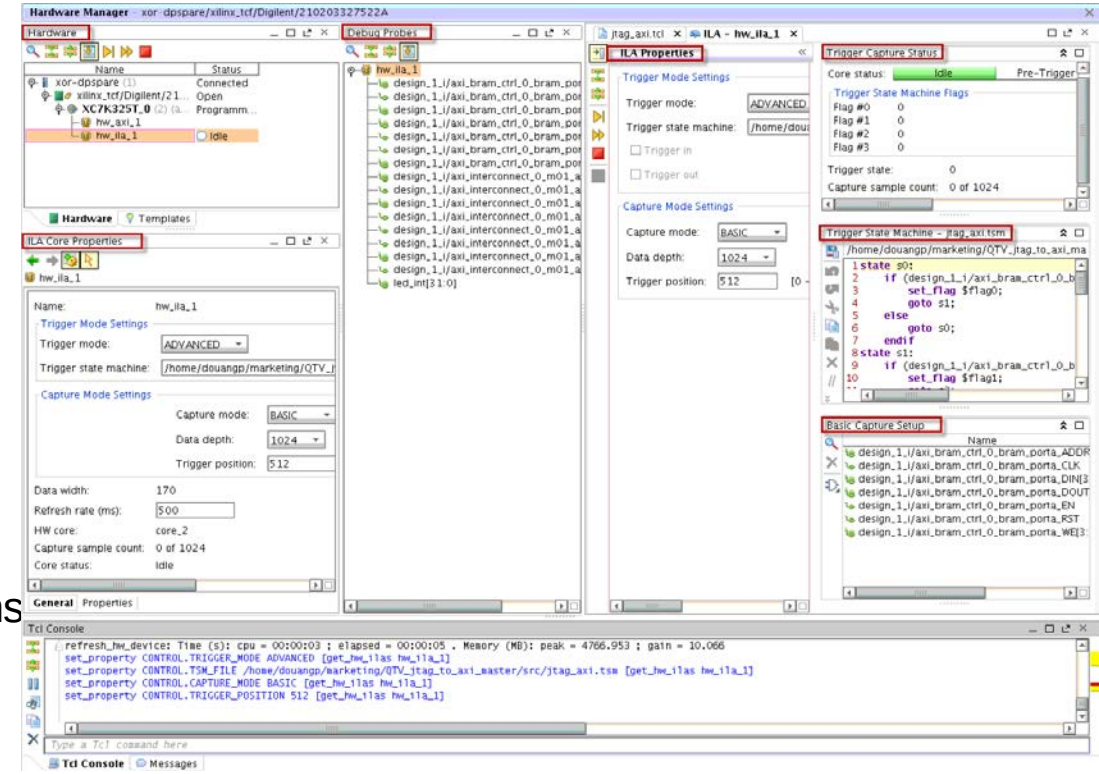




Benefits of Vivado Logic Debug

Key Benefits Summary

- Runtime software interface for interacting with debug IP cores
- Robust Core insertion flow
- Advanced trigger and capture control
  - Detect complex trigger events
  - Control what data to capture by ILA core
- Added runtime access interface for JTAG to AXI Master IP
  - Provides quick read and write transactions to an AXI-based system
- Hardware Manager & GUI Ease of Use
- Complete Tcl scripting of run-time functions



UG936 – Vivado Design Suite Tutorial
Programming and Debugging

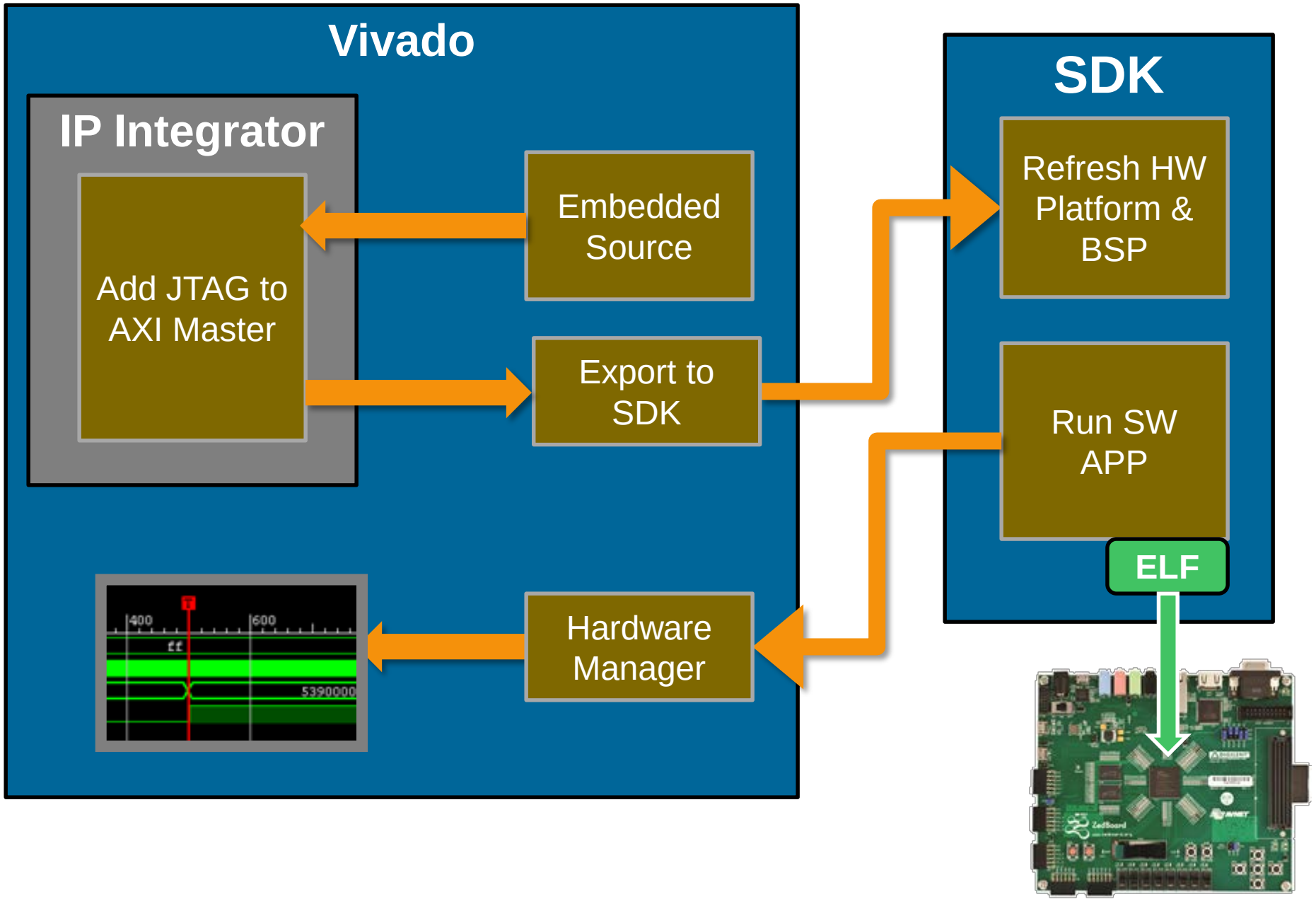
AVNET

Agenda

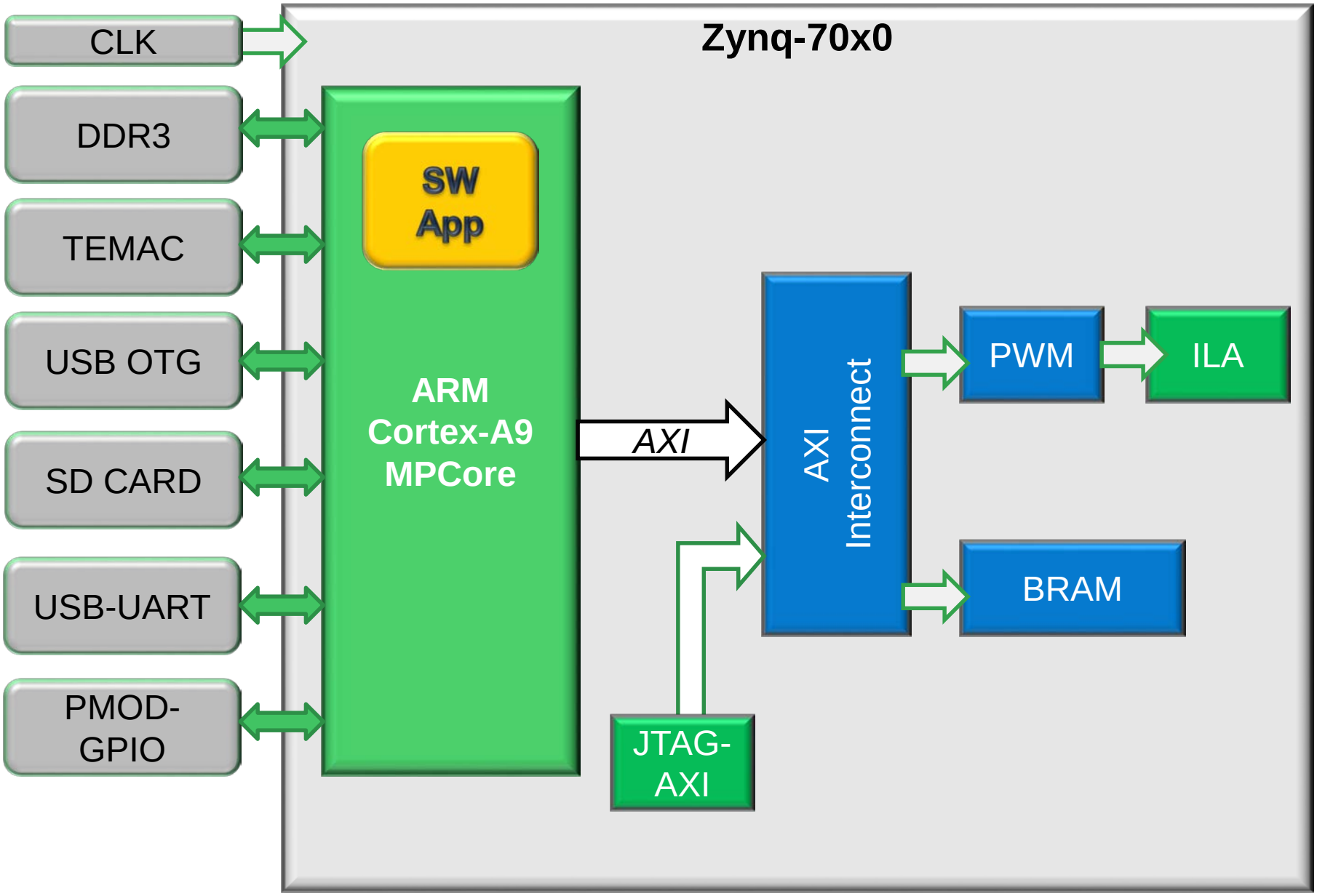
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1 ator
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - **Lab 8 – Hardware Debugging Zynq Designs**
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Lab 8 – Add PL Peripherals



Lab 8 – Add JTAG to AXI Master



Questions

What function is called when the Interrupt is detected?

- PWMIsr

```
/* Connect the interrupt handler that will be called when an
* interrupt occurs for the device. */
result = XScuGic_Connect(IntcInstancePtr,
INTC_PWM_INTERRUPT_ID,
(Xil_ExceptionHandler) PWMIsr, 0);
```

What does the PWMIsr function do?

- Resets the Brightness value and writes it out.

In the PWMIsr function, what does it do to the brightness value?

- Zero

What is the INTC_PWM_INTERRUPT_ID? Where did this number come from?

- *This is defined as:*
  PAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR
- *This is assigned to 91 in xparameters.*h?

```
/* Definitions for Fabric interrupts connected to ps7_scugic_0 */
#define XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR 91
```

AVNET

Questions

Can you detect how long it takes the processor to handle the interrupt?

- Technically yes, but not with this design. Why? Because our capture depth is only 1024 clock cycles, and the trigger does not reset in that time. If the capture depth was increased then it could be captured.

How would you delete one of the AXI transaction commands?

- delete_hw_axi_txn [get_hw_axi_txns write_txn]

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

Advantages of Scripting your Vivado Project

Allows for ease of sharing your hardware project

- Eliminates computer specific repositories

Accomplishes hour long rebuilds in minutes

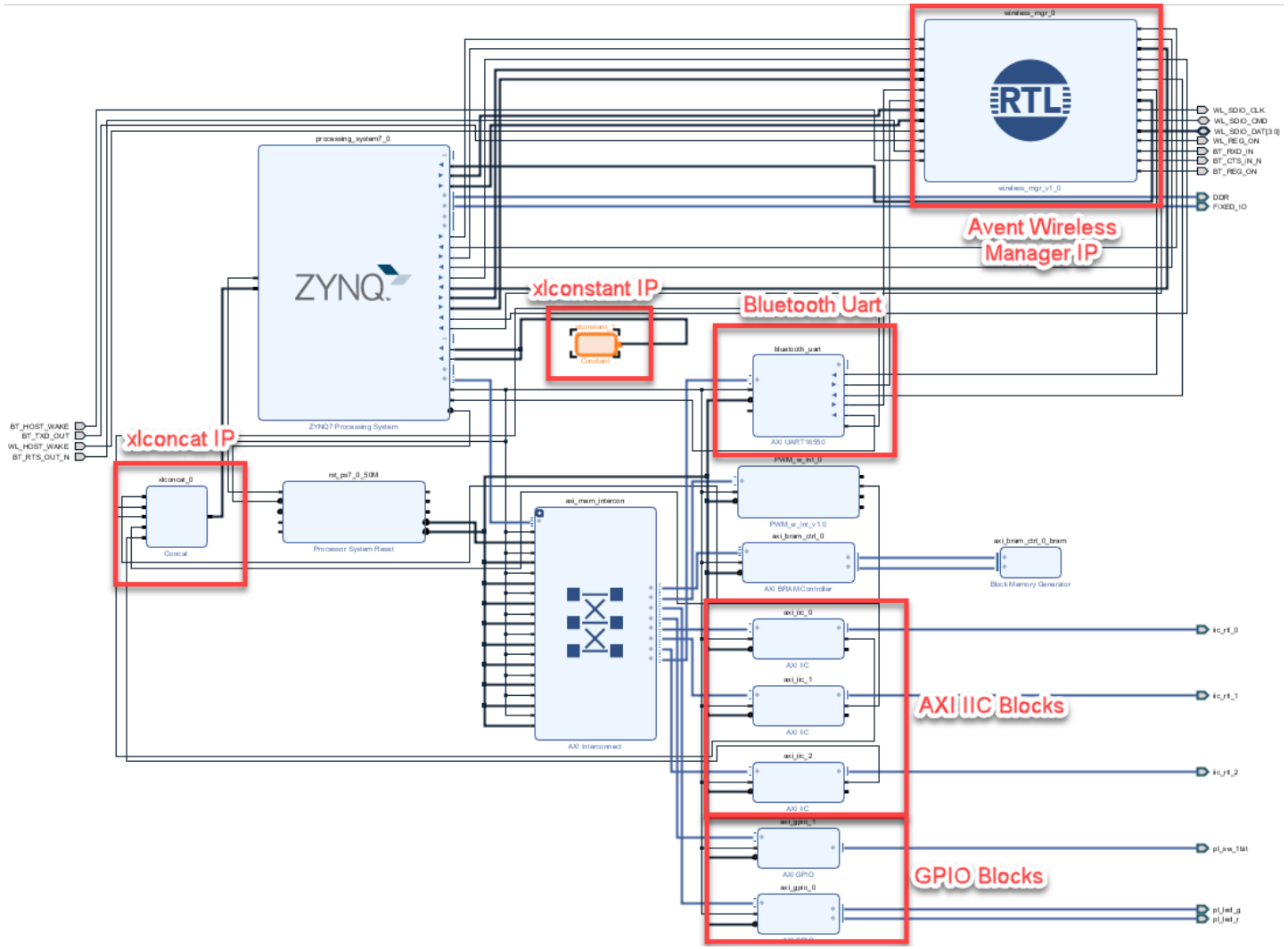
- All IP and interfaces are added customized and connected in minutes
- Allows you to customize your finished design easily

Agenda

- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9 – The Power of Scripting using Tcl
- What's Next

AVNET

Completed Zynq Hardware Design



AVNET

Agenda

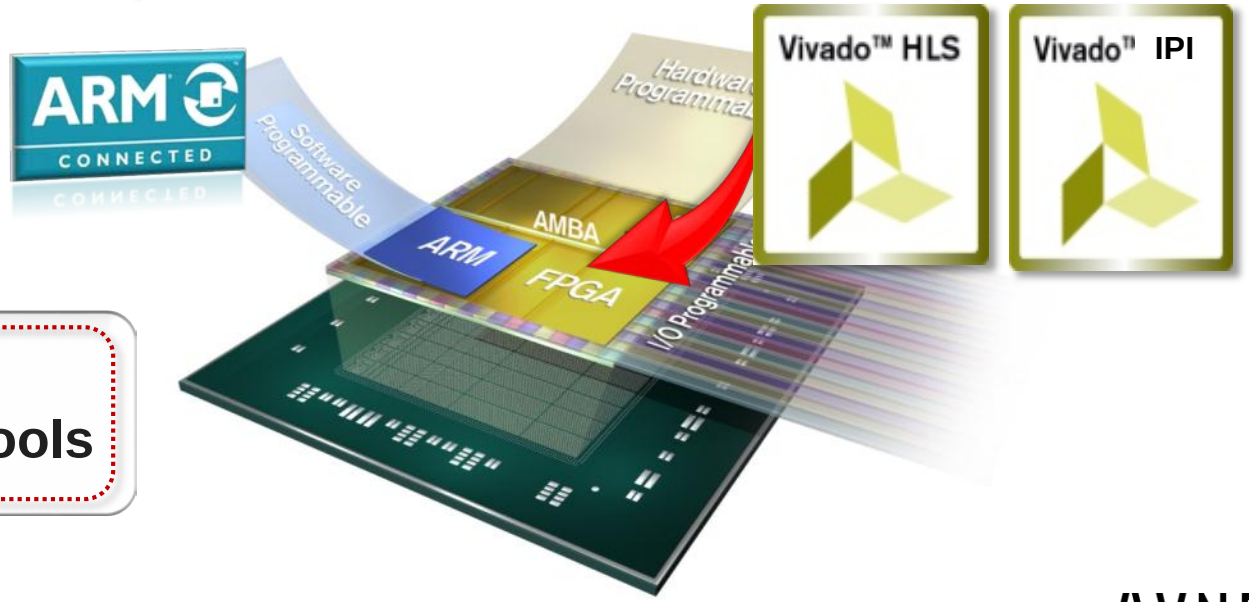
- Zynq Overview
- Xilinx Embedded Tool Flow
  - Lab 1
- Zynq Processor Overview
  - Lab 2
- Peripherals, Peripherals and more Peripherals!
  - Lab 3
- The Power of TCL
  - Lab 4
- Merging the Processing Subsystem (PS) and Programmable Logic (PL)
  - Lab 5
- Zynq PS DMA Controller
  - Lab 6
- Creating Custom IP
  - Lab 7
- Vivado's Hardware Manager
  - Lab 8
- Tcl Scripting
  - Lab 9
- What's Next

AVNET

All Programmable Abstractions & Automation

Abstraction

Automation



Next Generation
High-level Design Tools

AVNET

Widest Selection of Software Tools

Software Development Tools

XILINX SDK

ARM

YOKOGAWA

LAUTERBACH DEVELOPMENT TOOLS

KMC Kyoto Microcomputer Co.,Ltd.

mentor embedded

Microsoft Visual Studio

abatron

Computex

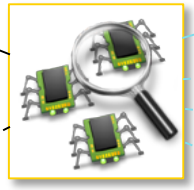
No Cost

Commercial Offerings

Heterogeneous Multi-Core Debugger

Cortex-A9

Cortex-A9



MicroBlaze 32

MicroBlaze 32

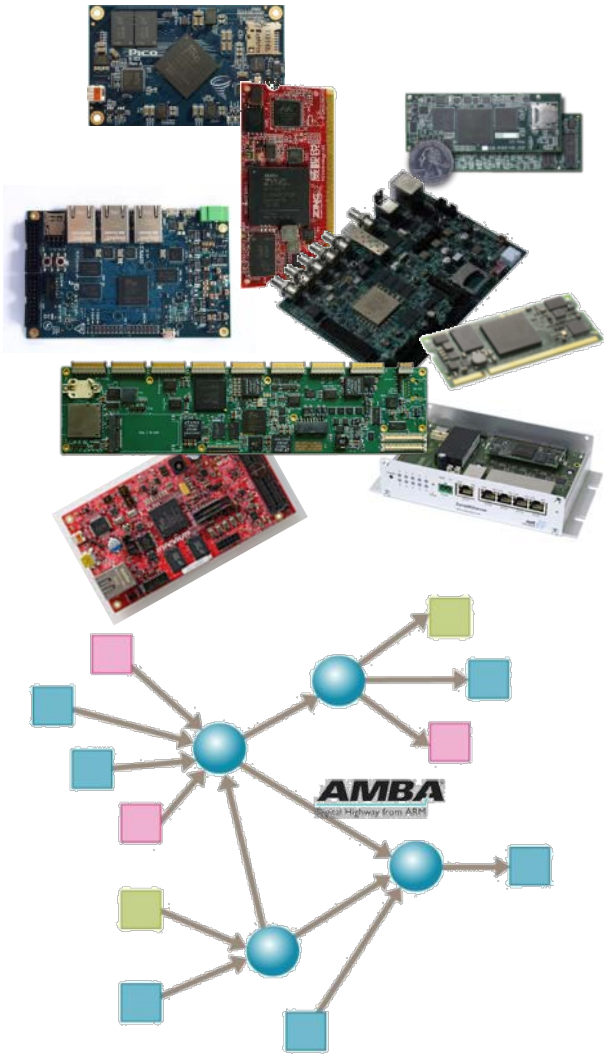
MicroBlaze 32

MicroBlaze 32

Scalable offering from Xilinx and partners to best fit your needs

VNET

Largest Portfolio of IP, Design Kits and Ref. Designs



Widest Selection of Development Platforms and SOMs

- Over 20 boards and SOM from Xilinx and partners
- Embedded, intelligent control, video signal processing
- 100+ FMC daughter cards

Ready to Use Segment Solutions

- Boards, IPs, reference designs, app notes, OS and drivers
- Automotive, A&D, broadcast, consumer, industrial, medical, wired and wireless

Large Selection of AXI-based IPs

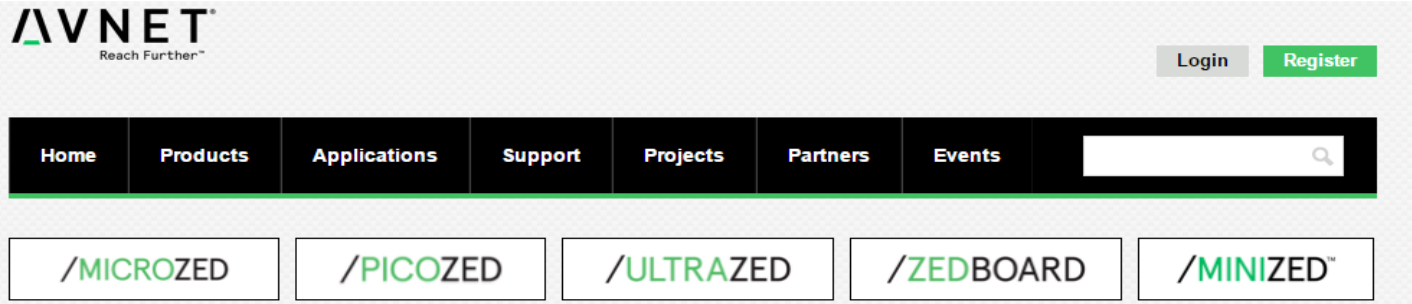
- Xilinx IPs standardized on AXI
- Large availability of Xilinx and Partner IPs
- Enhanced portability between FPGAs and SoCs

AVNET

Getting Help and Support

Visit ZedBoard.org or MiniZed.org

http://www.zedboard.org | http://www.MiniZed.org



For Xilinx technical support, you may contact your local Avnet/Silica FAE or Xilinx Online Technical Support at www.support.xilinx.com. On this site you will also find the following resources for assistance:

- Software, IP, and Documentation Updates
- Access to Technical Support Web Tools
- Searchable Answer Database with Over 4,000 Solutions
- User Forums
- Training - Select instructor-led classes and recorded e-learning options

Contact Avnet Support for any questions regarding ZedBoard reference designs, kit hardware, or if you are interested in designing any of the kit devices into your next design

AVNET

Course Objectives

Understand the Zynq-7000 All Programmable SoC development flow with Vivado's IP Integrator

Introduce the new Extensible Single ARM Cortex™-A9 Processors Cores

- Explore Robust AXI Peripheral Set

Utilize the Xilinx embedded systems tools to

- Design a Zynq AP SoC System
- Add Xilinx IP as well as custom IP
- Run Software Applications to test IP
- Debug an Embedded System