Joint ICTP-IAEA School on
Systems-on-Chip based on
FPGA for Scientific Instrumentation
and Reconfigurable Computing

SoPC Architecture & Design Methodology

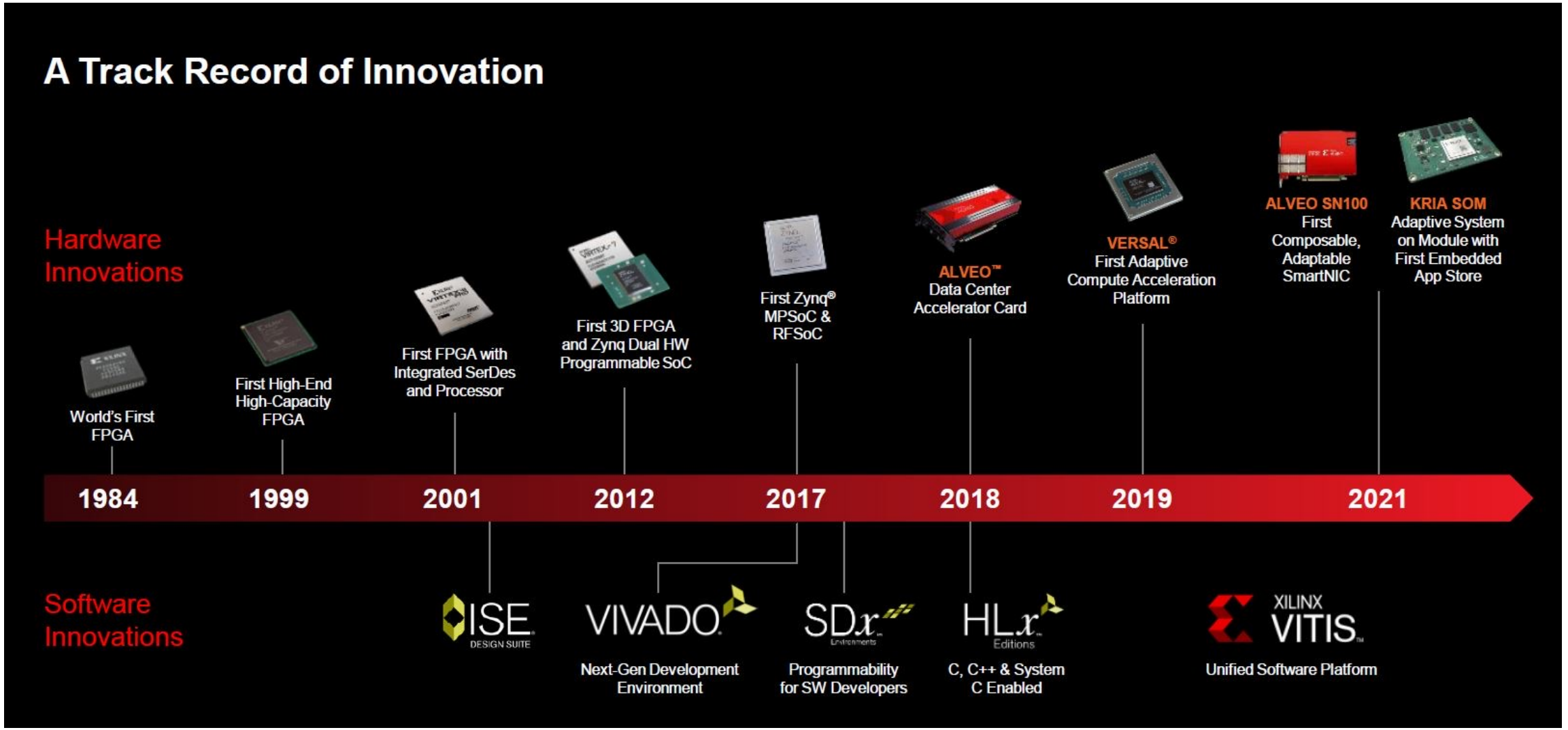
Cristian Sisterna

Universidad Nacional San Juan

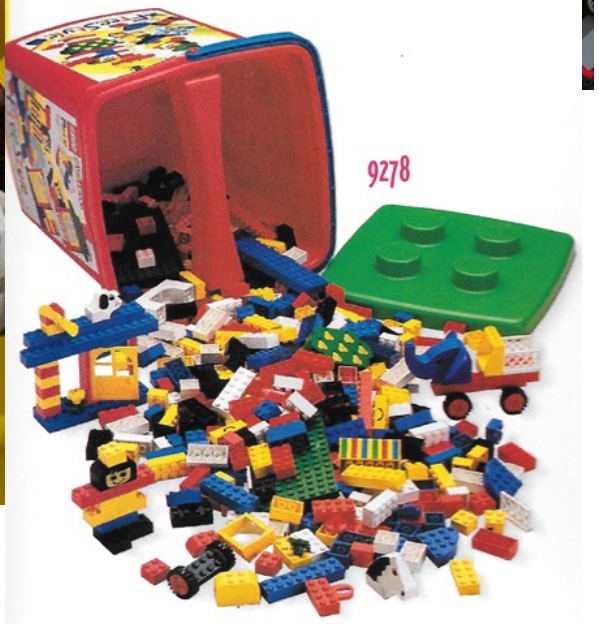
Argentina

FPGA Architecture

FPGA – SoPC Evolution

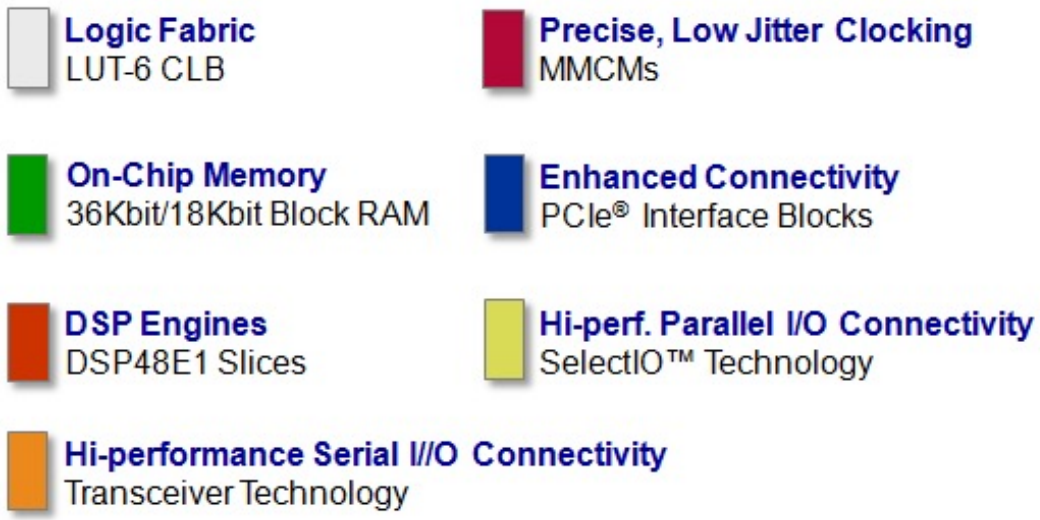


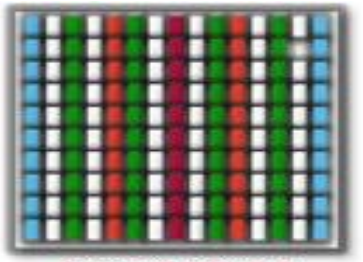
FPGA ~ Lego Bricks



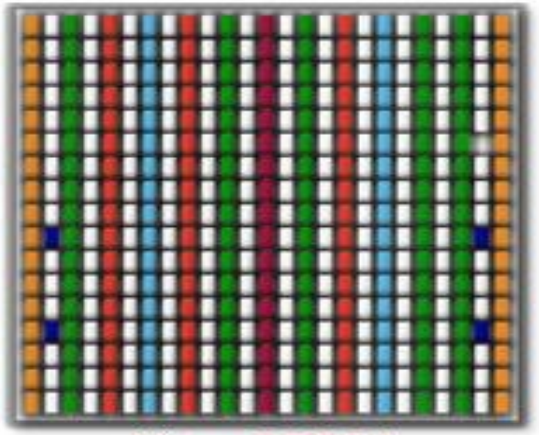
AMD 7-Series Architecture – Common Elements

- Common elements enable easy IP reuse for quick design portability across all 7-series families
  - Design scalability from low-cost to high-performance
  - Quickest time to market




Artix-7 FPGA

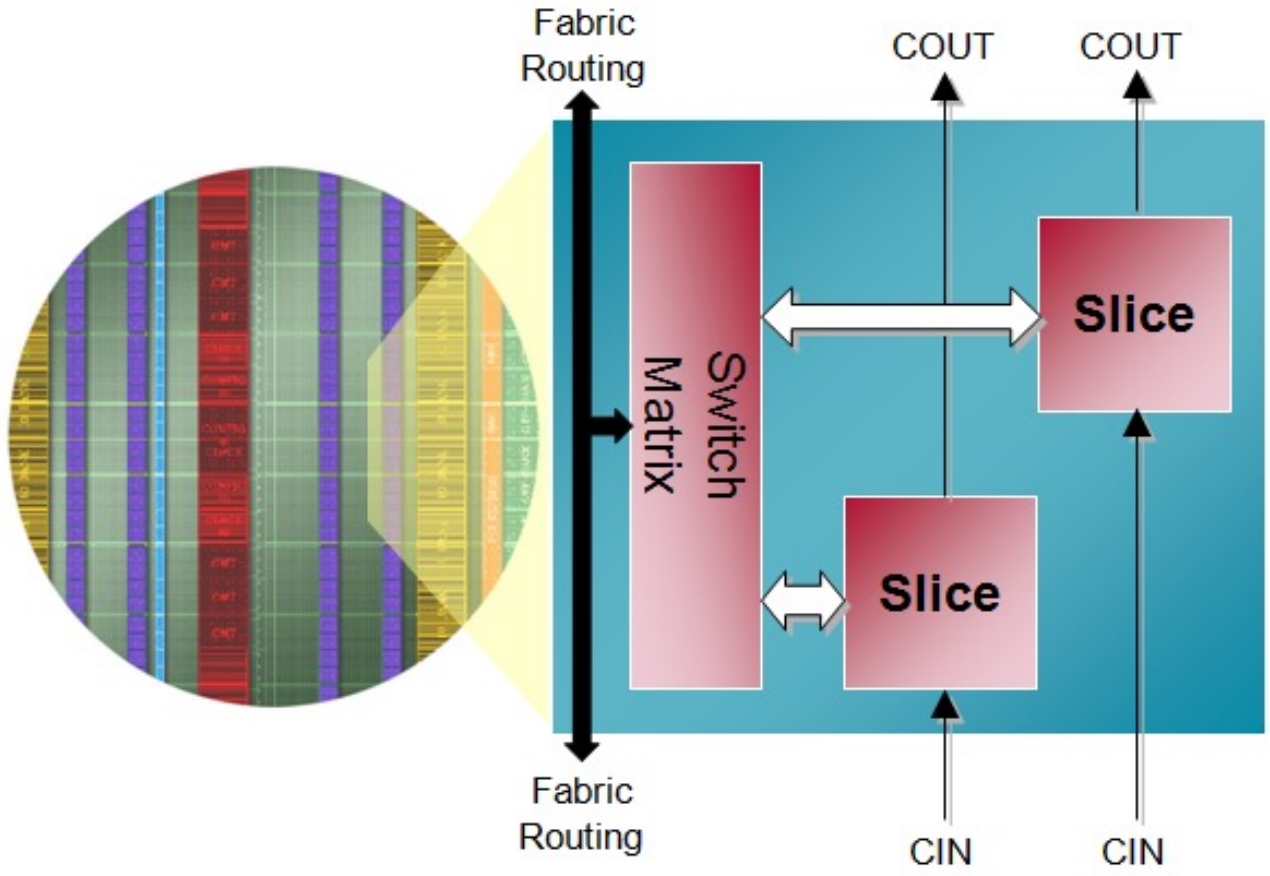

Kintex-7 FPGA


Virtex-7 FPGA

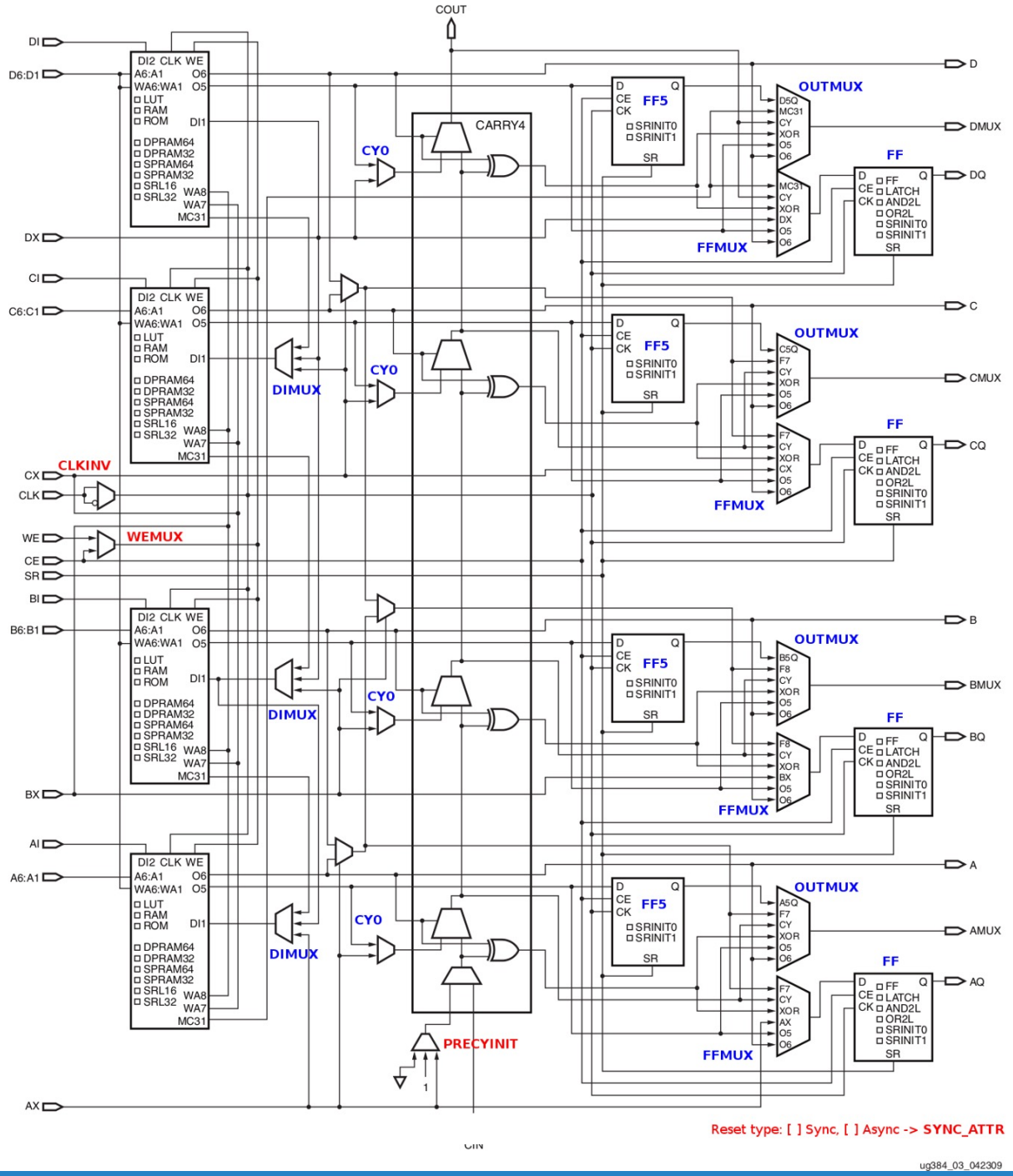
Logic Resources - CLB

Each **CLB** contains two slices:

- Primary resource for design
  - Combinatorial functions
  - Flip-flops
- Connected to switch matrix for routing to other FPGA resources
  - Carry chain runs vertically in a column from one slice to the one above

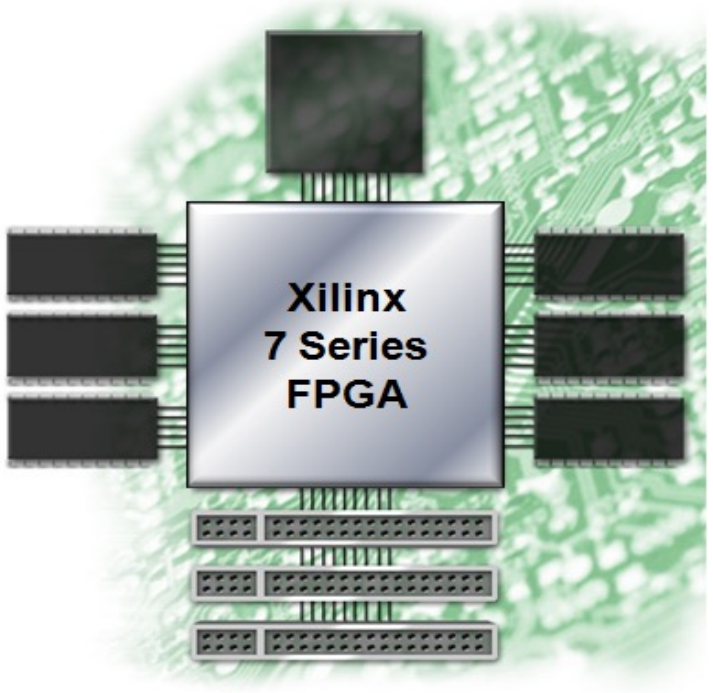


AMD-FPGA SLICE



7-Series FPGA - Inputs/Outputs

- Wide range of voltages
  - 1.2V to 3.3V operation
- Many different I/O standards
  - Single ended and differential
  - Referenced inputs
  - 3-state support
- Very high performance
  - Up to 1600 Mbps LVDS
  - Up to 1866 Mbps single-ended for DDR3
- Easy interfacing to standard memories
  - Hardware support for QDRII+ and DDR3
- Digitally controlled impedance
- Low power

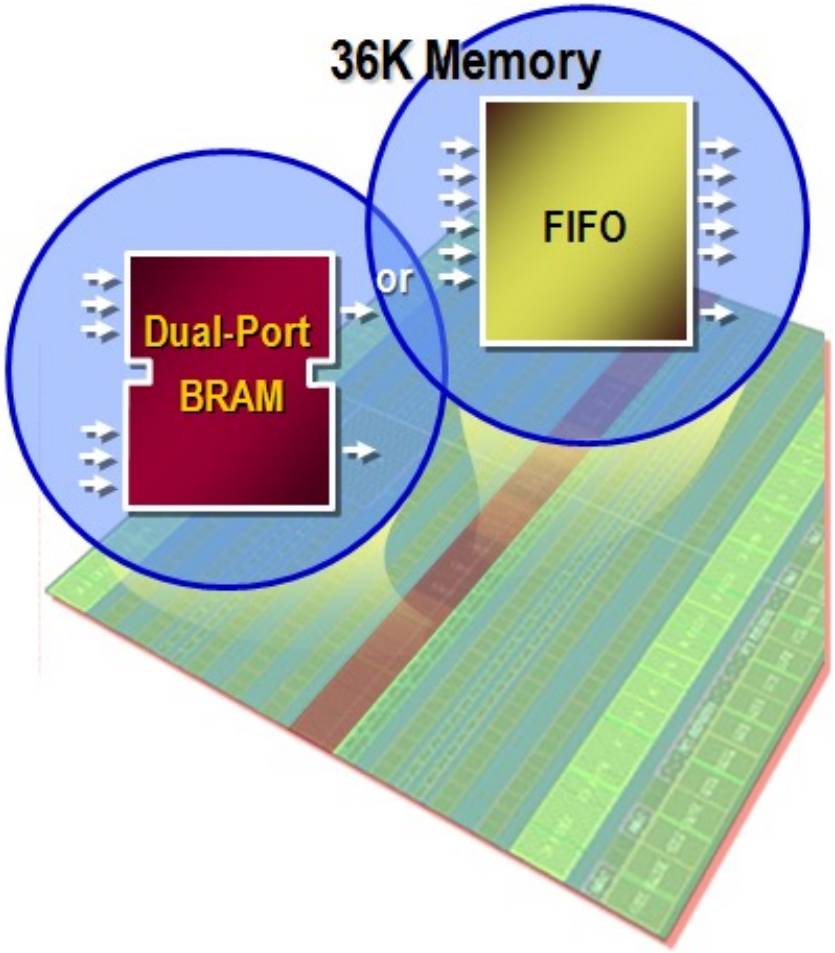


Most Common I/Os Standards Available

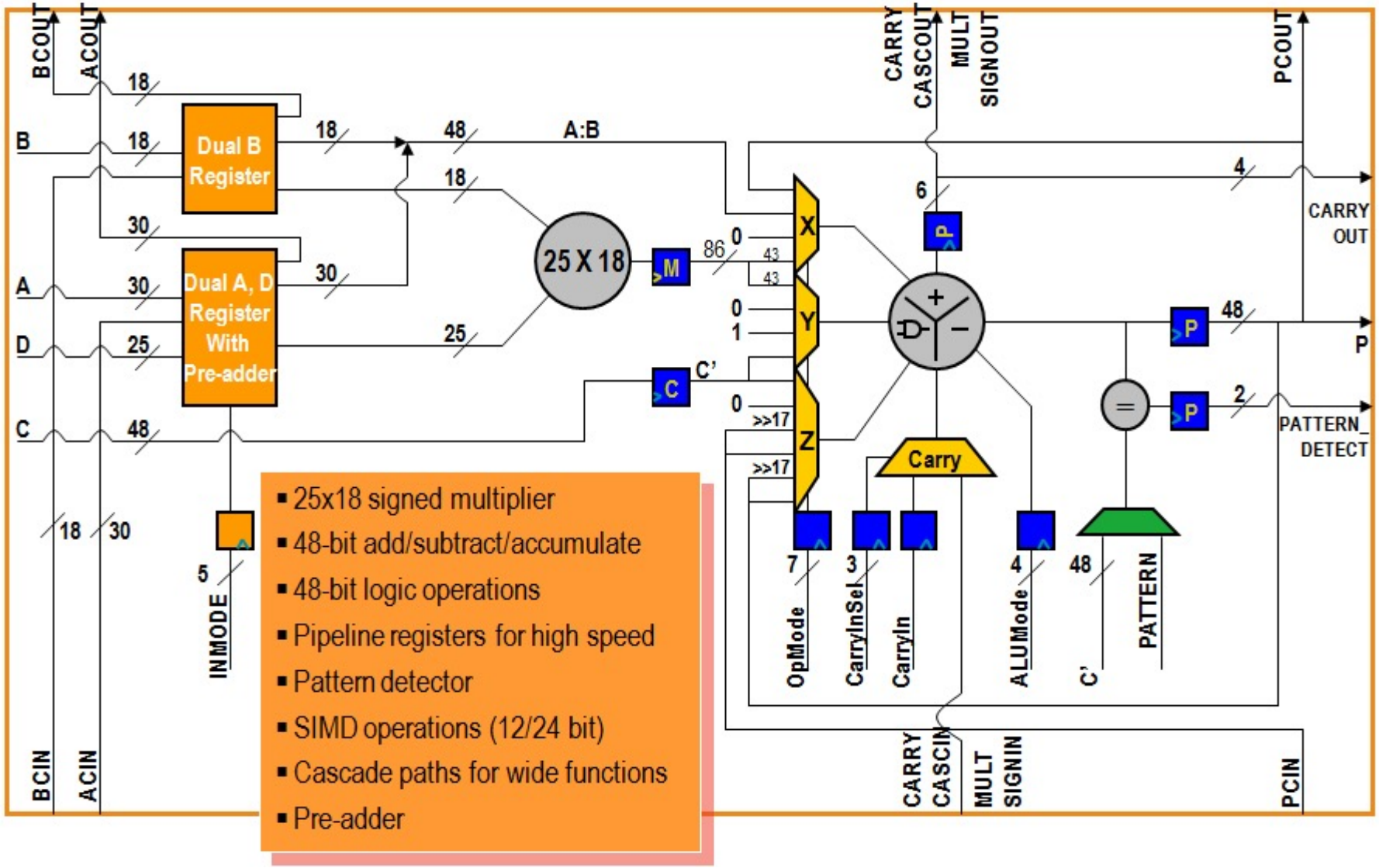
Standard	VCCO	Vref	Application
LVTTL	3.3	na	Single ended General Purpose
LVCMOS25, -18, -15	2.5, 1.8, 1.5	na	
LVDS33, 25, 18	3.3, 2.5, 1.8	na	Low Voltage Differential
PCI 33/66 MHz, 3.3V	3.3	na	PCI Bus
GTL / GTL+	na	0.80	Back-Plane
HSTL-I, HSTL-III	1.5	0.75	SRAM
SSTL3-I, -II, SSTL2-I	3.3	0.90, 1.5	SDRAM

7-Series Block RAM and FIFO

- All members of the 7-series families have the same Block RAM/FIFO
- Fully synchronous operation
  - All operations are synchronous; all outputs are latched
- Optional internal pipeline register for higher frequency operation
- Two independent ports access common data
  - Individual address, clock, write enable, clock enable
  - Independent data widths for each port
- Multiple configuration options
  - True dual-port, simple dual-port, single-port
- Integrated control for fast and efficient FIFOs

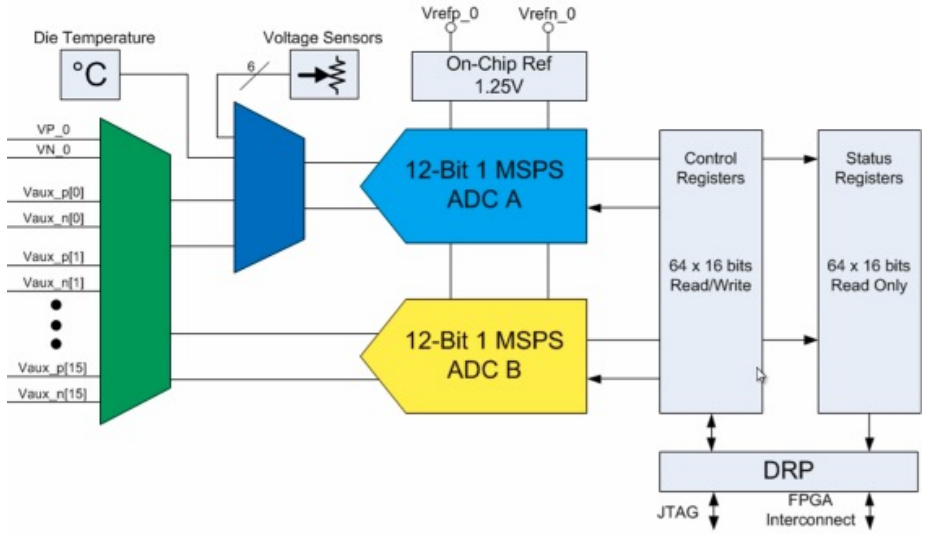


7- Series DSP48E1 Slice



XADC and Analog Mixed Signals (AMS)

- XADC is a high quality and flexible analog interface new to the 7-series
  - Dual 12-bit 1Msps ADCs, on-chip sensors, 17 flexible analog inputs, and track & holds with programmable signal conditioning
  - 1V input range (unipolar, bipolar and differential)
  - 12-bit resolution conversion
  - Built in digital gain and offset calibration
  - On-chip thermal and Voltage sensors
  - Sample rate of 1 MSPS



- Analog Mixed Signal (AMS)
  - Using the FPGA programmable logic to customize the XADC and replace other external analog functions; for example, linearization, calibration, filtering, and DC balancing to improve data conversion resolution

https://henryomd.blogspot.com/2015/06/bare-metal-code-to-read-adc-on-zynq.html

7-Series FPGA Families

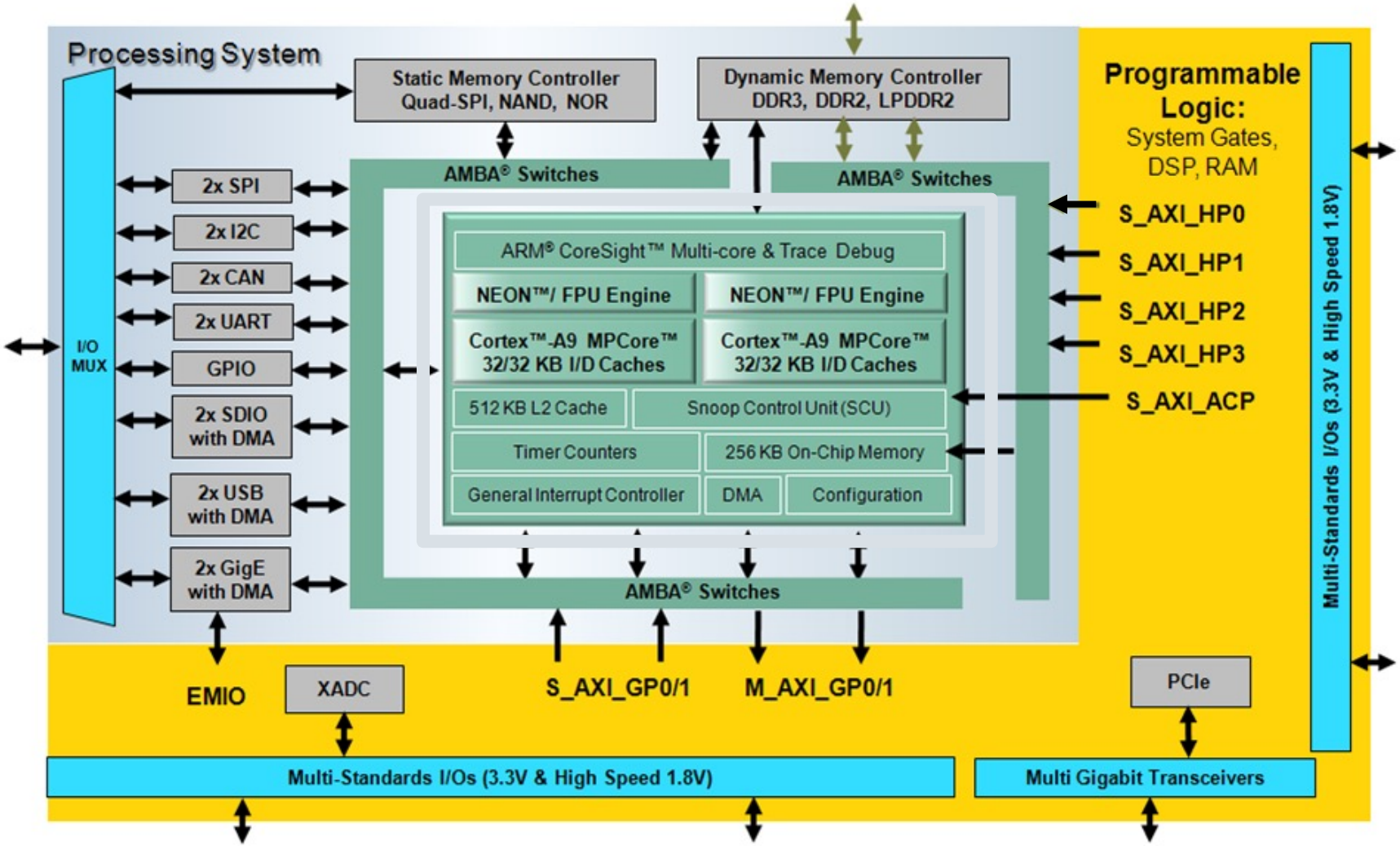
Maximum Capability	ARTIX-7 Lowest Power and Cost	KINTEX-7 Industry's Best Price/Performance	VIRTEX-7 Industry's Highest System Performance	ZYNQ Extensible Processing Platform
Logic Cells	20K – 355K	70K – 480K	285K – 2,000K	30K – 350K
Block RAM	12 Mb	34 Mb	65 Mb	240KB – 2180KB
DSP Slices	40 – 700	240 – 1,920	700 – 3,960	80 – 900
Peak DSP Perf.	504 GMACS	2,450 GMACs	5,053 GMACS	1080 GMACS
Transceivers	4	32	88	16
Transceiver Performance	3.75Gbps	6.6Gbps and 12.5Gbps	12.5Gbps, 13.1Gbps and 28Gbps	6.6Gbps and 12.5Gbps
Memory Performance	1066Mbps	1866Mbps	1866Mbps	1333Mbps
I/O Pins	450	500	1,200	372
I/O Voltages	3.3V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below

Zynq-SoPC Architecture

Zynq-7000 Main Features

- **Complete ARM®-based processing system**
  - Application Processor Unit (APU)
    - Dual ARM Cortex™-A9 processors
    - Caches and support blocks
  - Fully integrated memory controllers
  - I/O peripherals
- **Tightly integrated programmable logic**
  - Used to extend the processing system
  - Scalable density and performance
- **Flexible array of I/O**
  - Wide range of external multi-standard I/O
  - High-performance integrated serial transceivers
  - Analog-to-digital converter inputs

Zynq SoPC Block Diagram



PS Main Components

- **Application processing unit (APU)**
- **I/O peripherals (IOP)**
  - Multiplexed I/O (MIO), extended multiplexed I/O (EMIO)
- **Memory interfaces**
- **PS interconnect**
- **DMA**
- **Timers**
  - Public and private
- **General interrupt controller (GIC)**
- **On-chip memory (OCM): RAM**
- **Debug controller: CoreSight**

PL Main Components

- Configurable logic blocks (CLB)
  - 6-input look-up tables (LUTs)
  - Memory capability within the LUT
  - Register and shift register functionality
- 36 Kb BRAM
- DSP48E1 Slice
- Clock management
- Configurable I/Os
- High Speed Serial Transceivers
- Integrated interface for PCI Express

Zynq	FPGA Based Fabric
7z010, 7z015, 7z020	Artix
7z030, 7z035, 7z045, 7z100	Kintex

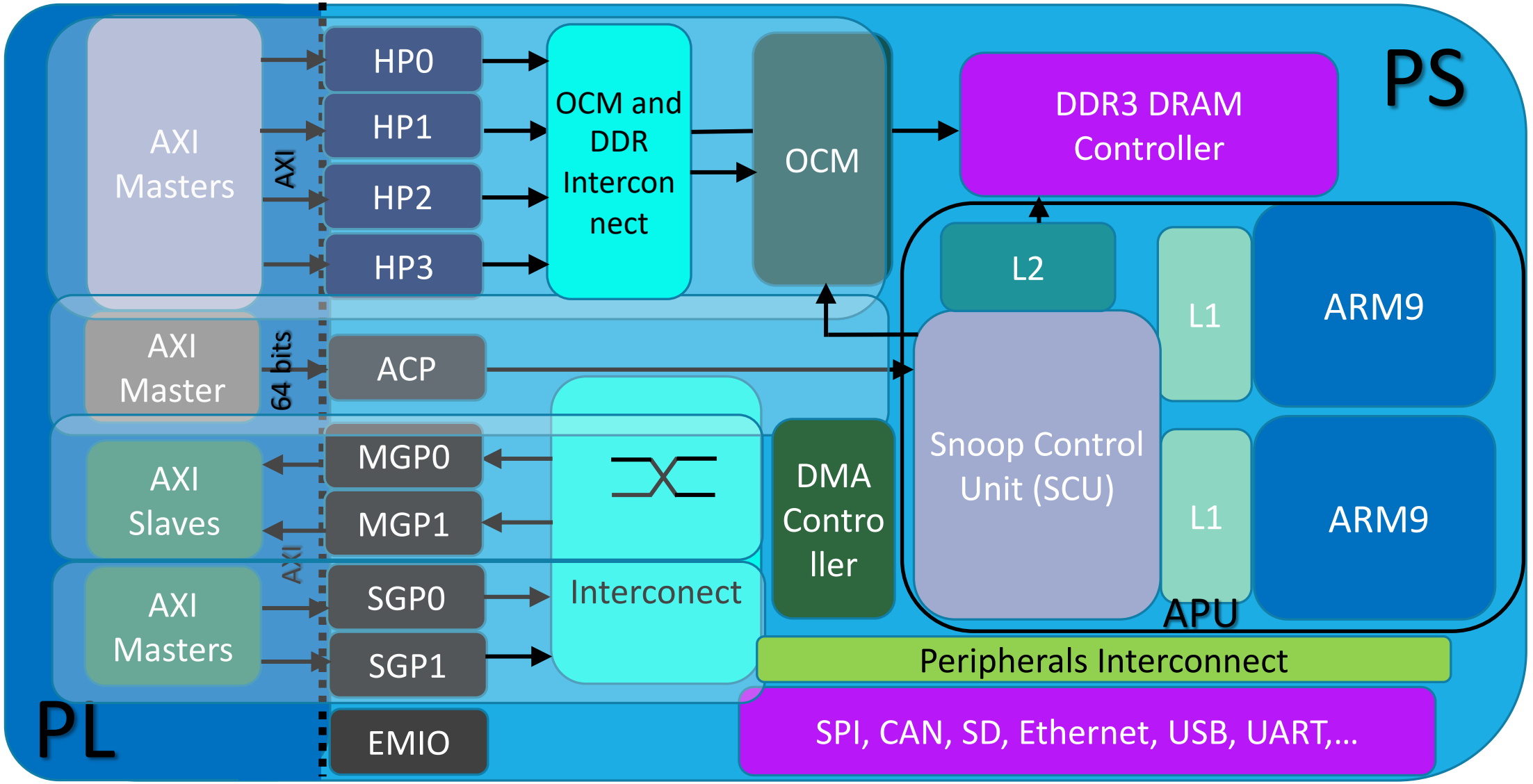
PS-PL Interface

- **AXI high-performance slave ports (HP0-HP3)**
  - Configurable 32-bit or 64-bit data width
  - Access to OCM and DDR only
  - Conversion to processing system clock domain
  - AXI FIFO Interface (AFI) are FIFOs (1KB) to smooth large data transfers
- **AXI general-purpose ports (GP0-GP1)**
  - Two masters from PS to PL
  - Two slaves from PL to PS
  - 32-bit data width
  - Conversion and sync to processing system clock domain

PS-PL Interface

- **One 64-bit accelerator coherence port (ACP) AXI slave interface to CPU memory**
- **DMA, interrupts, events signals**
  - Processor event bus for signaling event information to the CPU
  - PL peripheral IP interrupts to the PS general interrupt controller (GIC)
  - Four DMA channel RDY/ACK signals
- **Extended multiplexed I/O (EMIO) allows PS peripheral ports access to PL logic and device I/O pins**
- **Clock and resets**
  - Four PS clock outputs to the PL with enable control
  - Four PS reset outputs to the PL
- **Configuration and miscellaneous**

Zynq Architecture



PS-PL AXI Interfaces

Interface Name	Interface Description	Master	Slave
M_AXI_GP0	General Purpose (AXI_GP)	PS	PL
M_AXI_GP1		PS	PL

PS-PL Interface Performance

Method	Benefits	Drawbacks	Usage	Performance
PL AXI-HP DMA	• Highest throughput • Multiple interfaces • Command/Data FIFO	• OC/DDR access only • Complex PL Master design	• HP DMA for large datasets	1.200 MB/s (per interface)
PL AXI-ACP DMA	• - Highest throughput - Lowest latency - Optional cache coherency	• Large burst might cause cache trashing • Shares CPU interconnect bandwidth • Complex PL Master design	• HP DMA for smaller coherent datasets • Medium granularity CPU offload	1.200 MB/s
PL AXI-GP DMA	• Medium throughput	• More complex PL Master design	• PL to PS control functions • PS I/O Peripheral access	600 MB/s
CPU Programmed I/O	• Simple Sw • Least PL resources • Simple PL Slave	• Lowest throughput	• Control functions	< 25 MB/s

PS-PL Miscellaneous Signals

- PL Clocks and Resets
- PL Interrupts to PS
- IOP Interrupts to/from the PL
- Events
- Idle AXI, DDR, ARB, SRAM Interrupt
- DMA Controller
- EMIO Signals

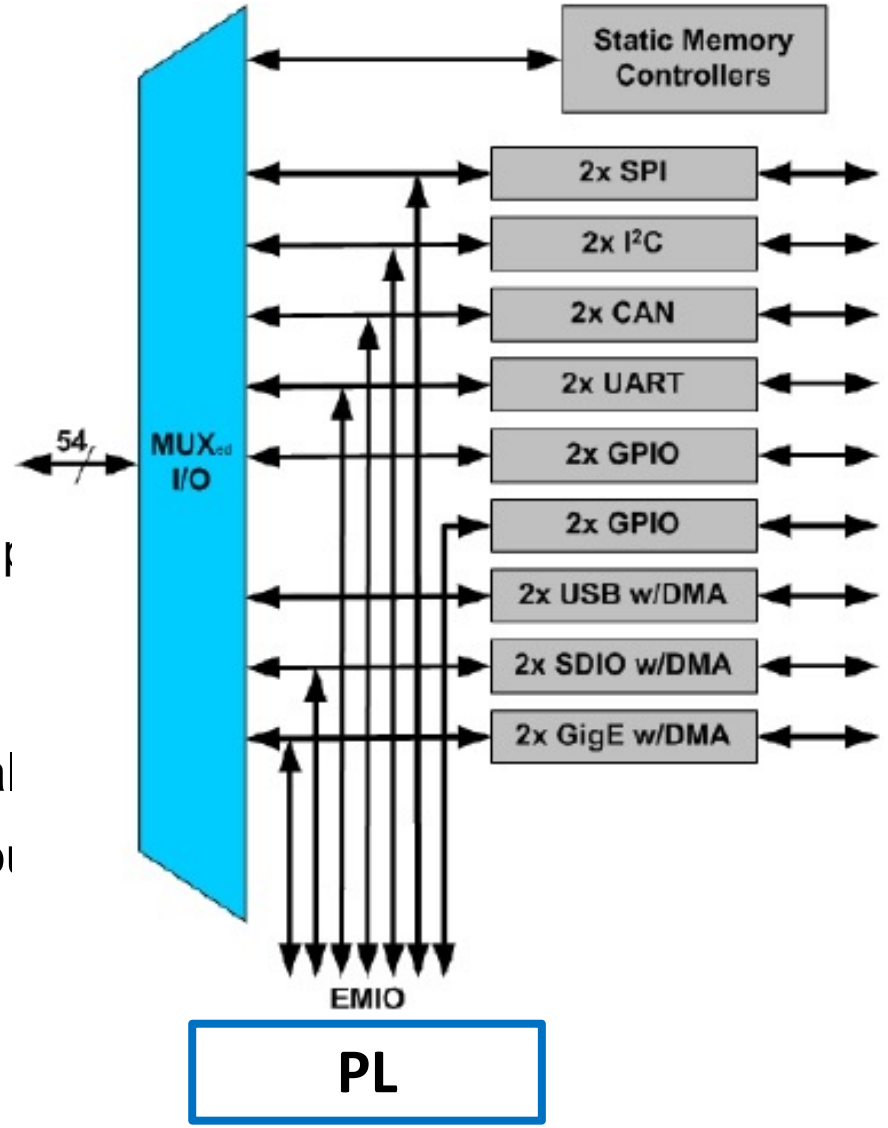
PS Peripherals and Connections

PS Available Peripherals

PS Peripheral	Description
SPI (x2)	Serial Peripheral Interface, either master or slave mode.
I2C (x2)	I^2C bus. Supports master and slave modes.
CAN (x2)	Controller Area Network. CAN 2.0A and CAN 2.0B.
UART (x2)	Universal Asynchronous Receiver Transmitter. One UART is used for terminal connections to a host PC.
GPIO	General purpose Input/Output. There are four GPIO banks, each of 32 bits.
SD (x2)	SD card memory controller
USB (x2)	Universal Serial Bus. Host or OTG (in-the-go) modes.
GigE (x2)	Gigabit Ethernet MAC peripheral, supports 10Mbps, 100MBps and 1Gbps.

Multiplexed I/O – Internal / External

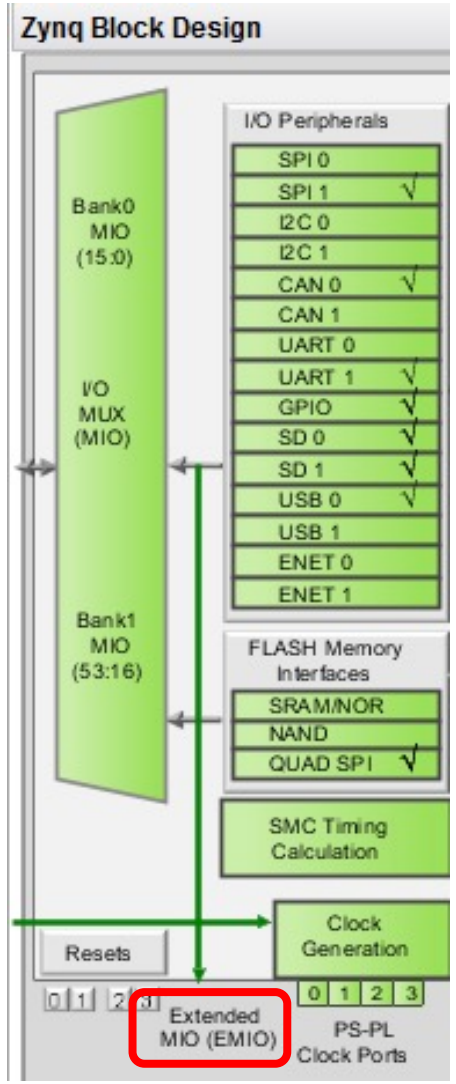
- **Multiplexed input/output (MIO)**
  - Multiplexed output of peripheral and static memories
  - Two I/O banks; each selectable: 1.8V, 2.5V, or 3.3V
  - Configured using configuration
  - Dedicated pins are used
    - User constraints (LOC) should not be present
      - The BitGen process will throw errors if LOC constraints are p
- **Extended MIO**
  - Enables use of the SelectIO™ interface with PS peripheral
  - User constraints must be present for the signals brought ou to the SelectIO pins
    - The BitGen process will throw errors if LOC constraints are not present



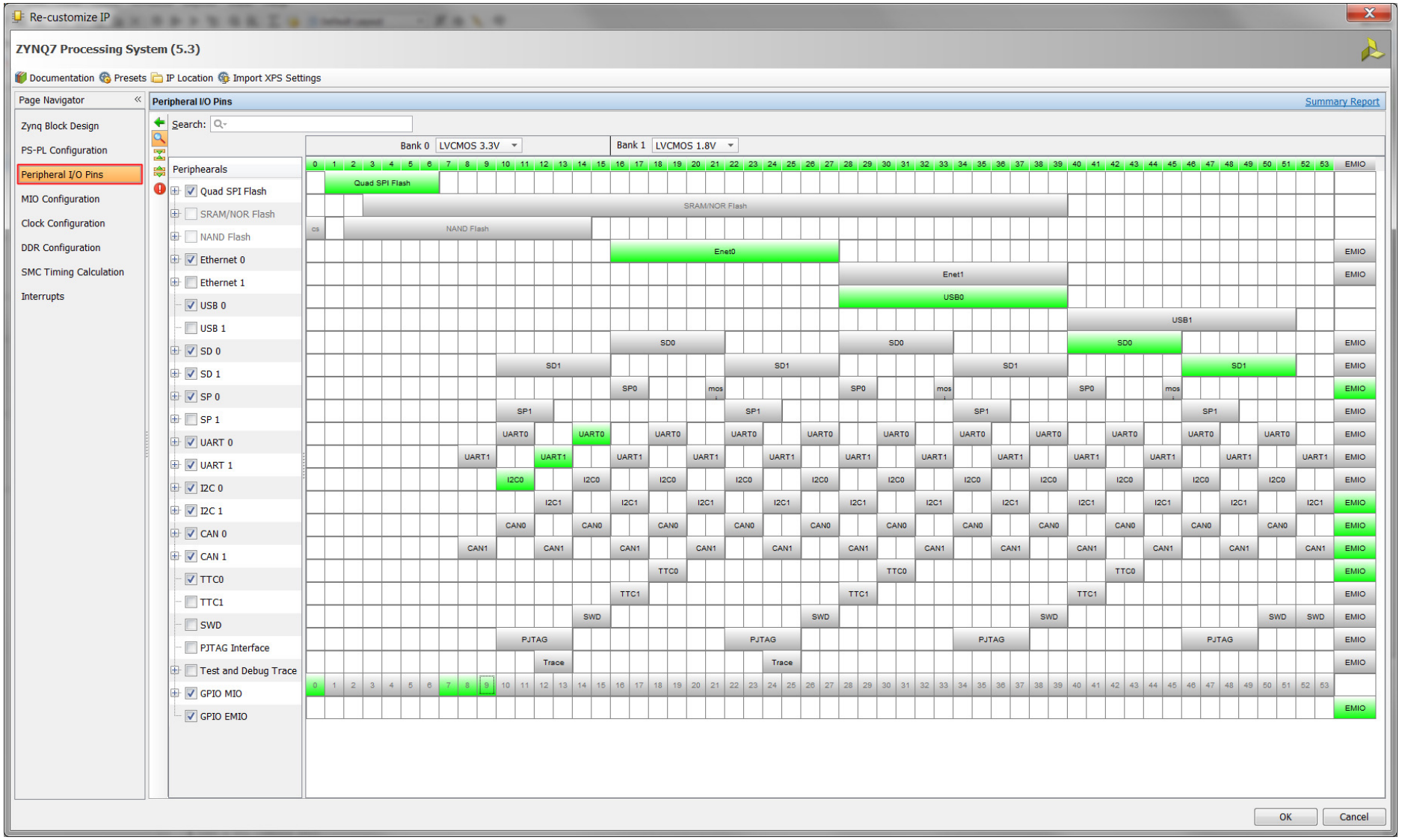
Extended Multiplexed I/O (EMIO)

Extended interface to PL I/O peripheral ports

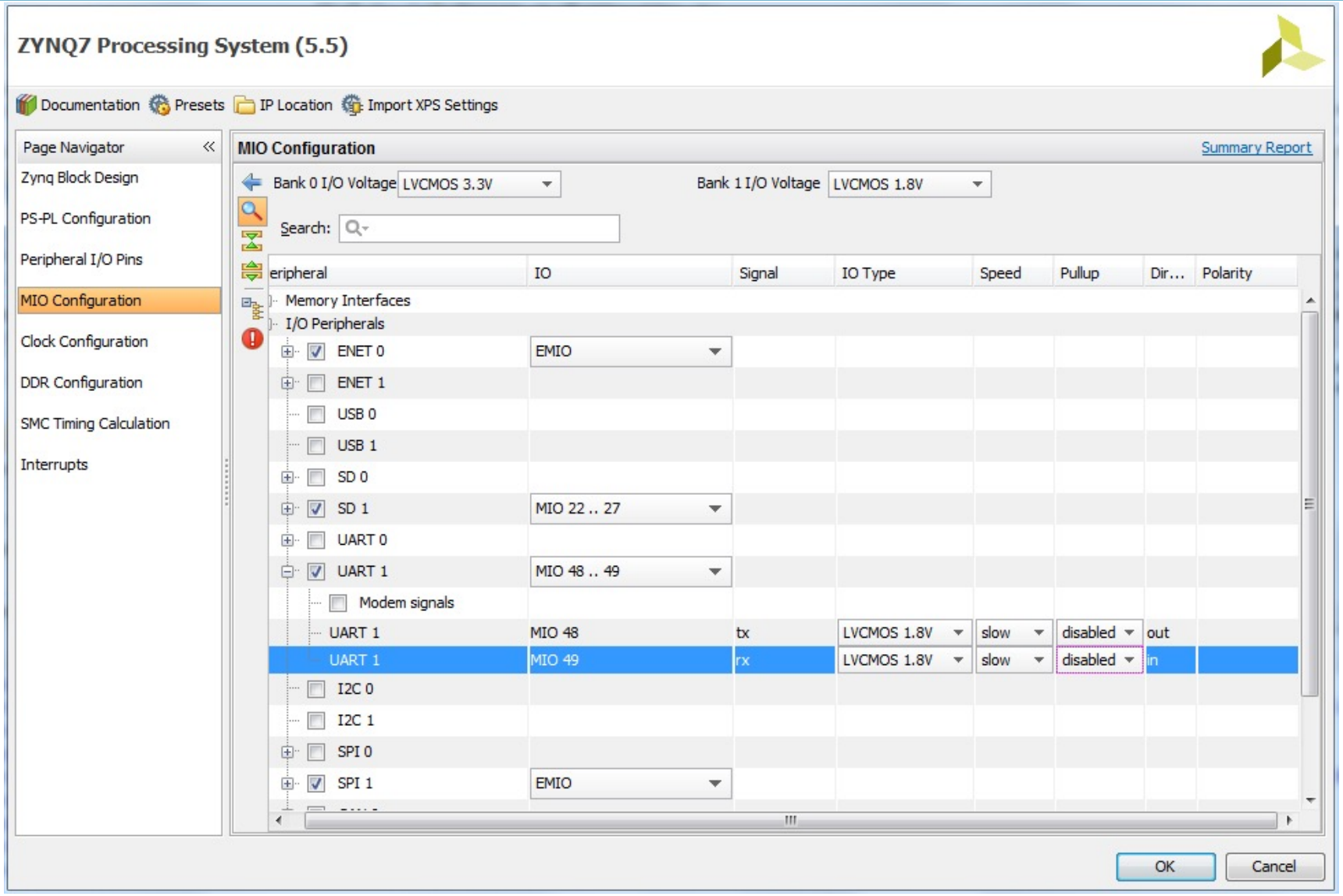
- EMIO: Peripheral port to PL
- Alternative to using MIO
- Mandatory for some peripheral ports
- Facilitates
  - Connection to peripheral in programmable logic
  - Use of general I/O pins to supplement MIO pin usage
  - Allows additional signals for many of the peripherals
  - Alleviates competition for MIO pin usage



Multiplexed I/O (MIO)

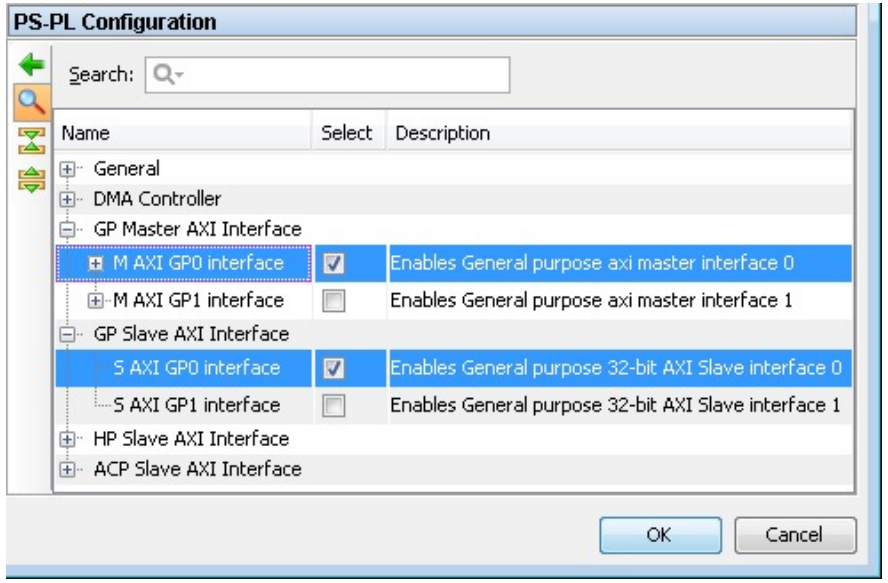


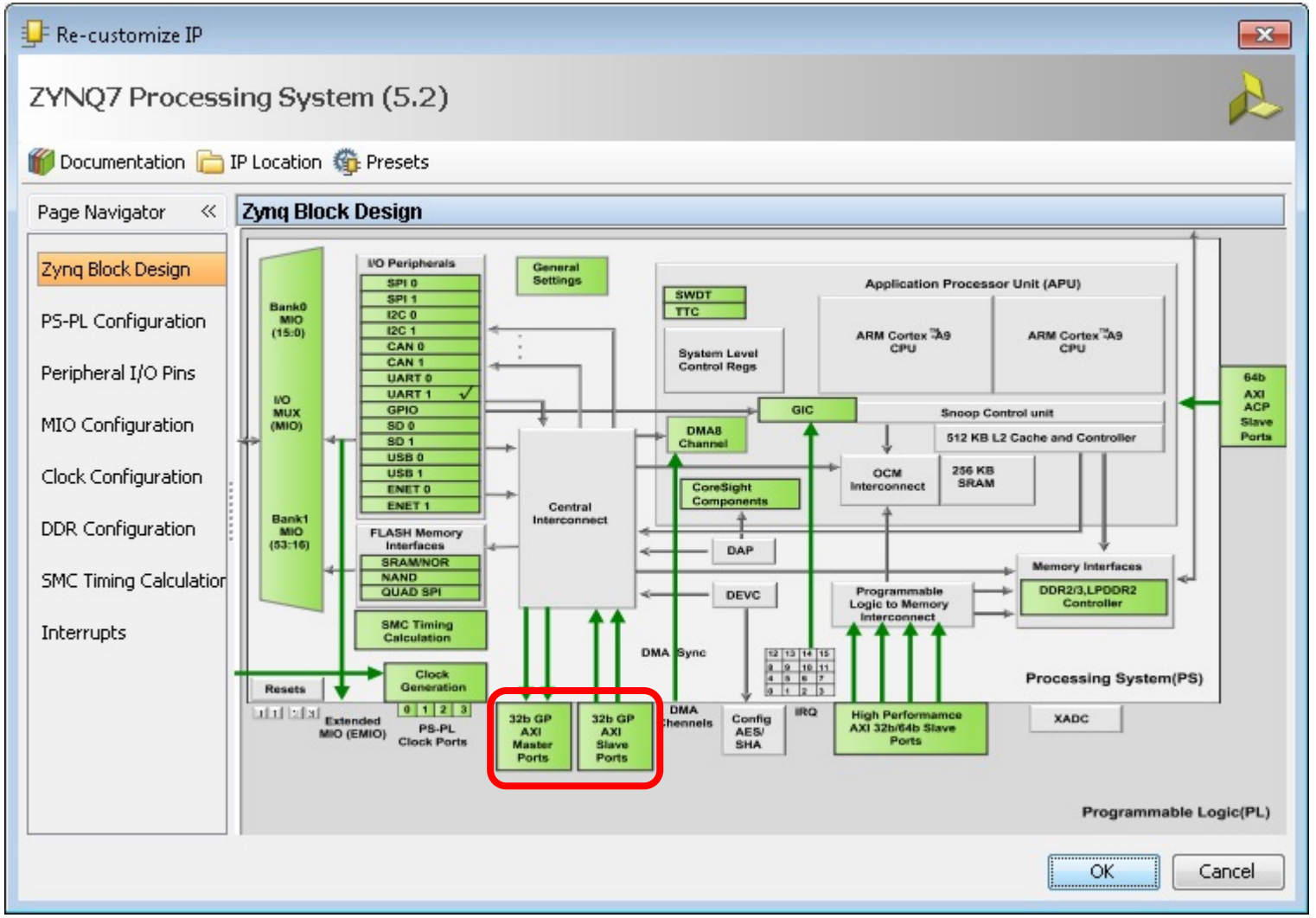
MIO Port Configuration



GP0/1 Ports Configuration for PS-PL Interface

- Click on the menu or green GP Blocks to configure

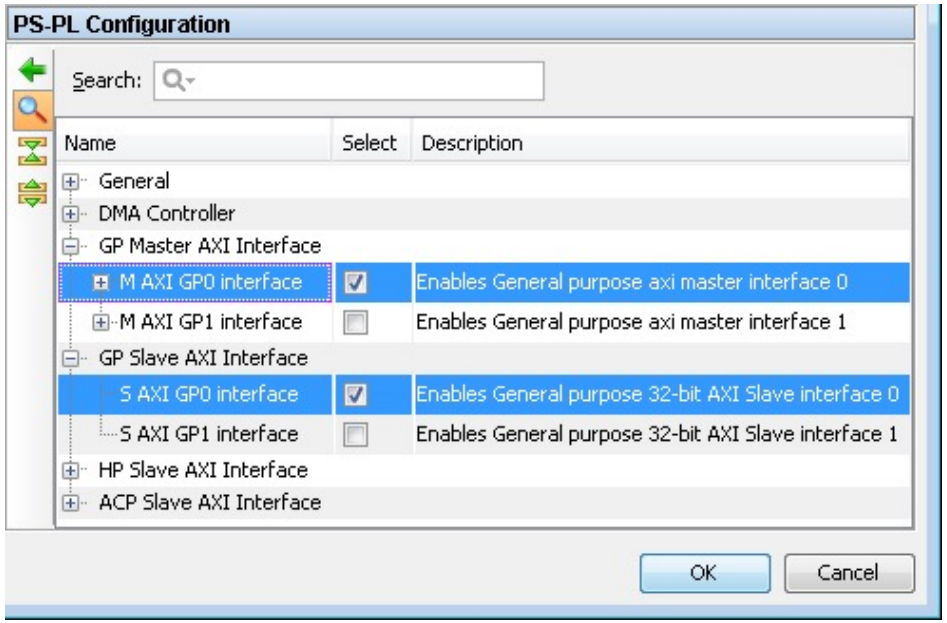




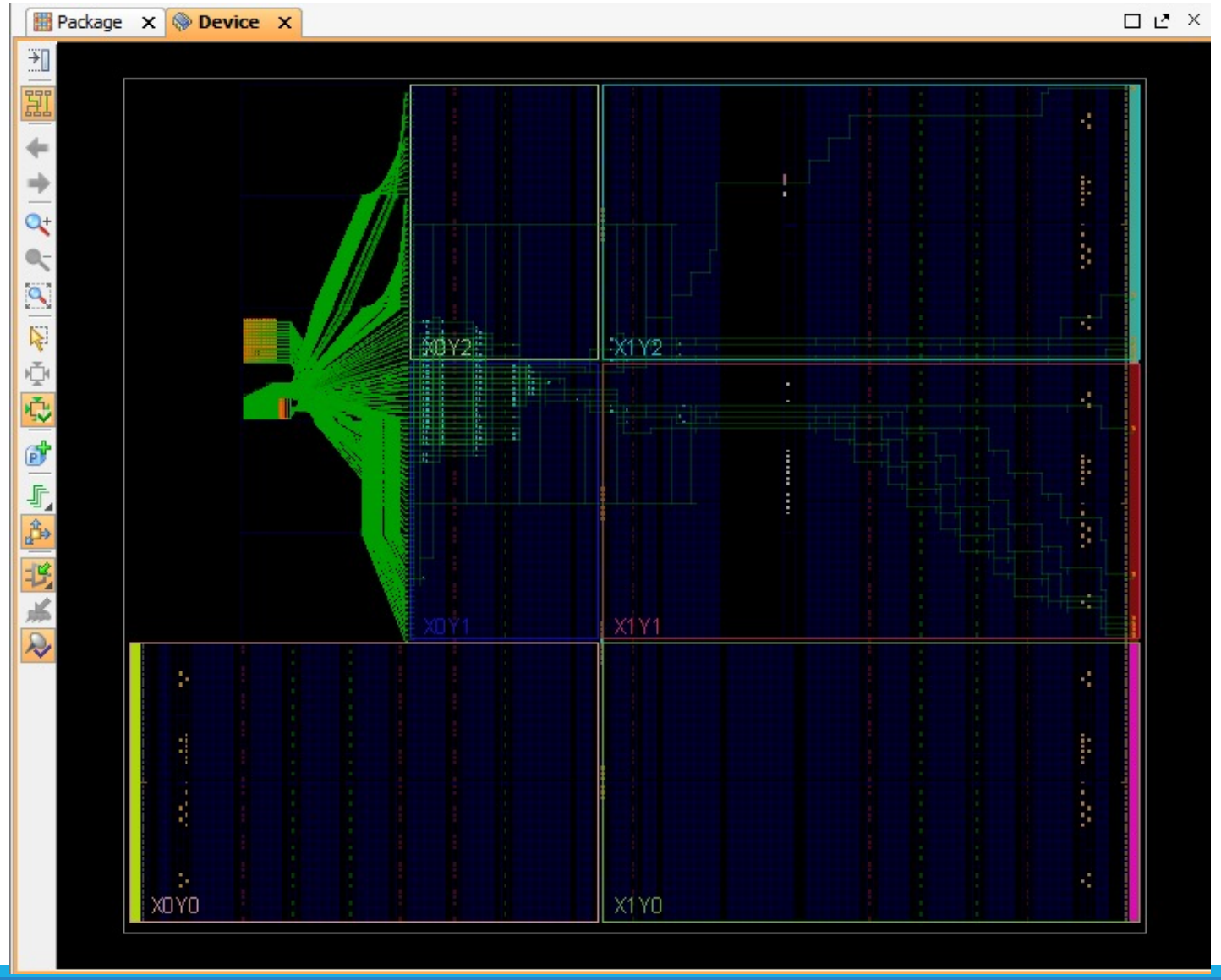
GP 0/1 Ports

- By default, GP Slave and Master ports are disabled

- Enable GP Master and/or Slave ports depending on whether a slave or a master peripheral is going to be added in PL



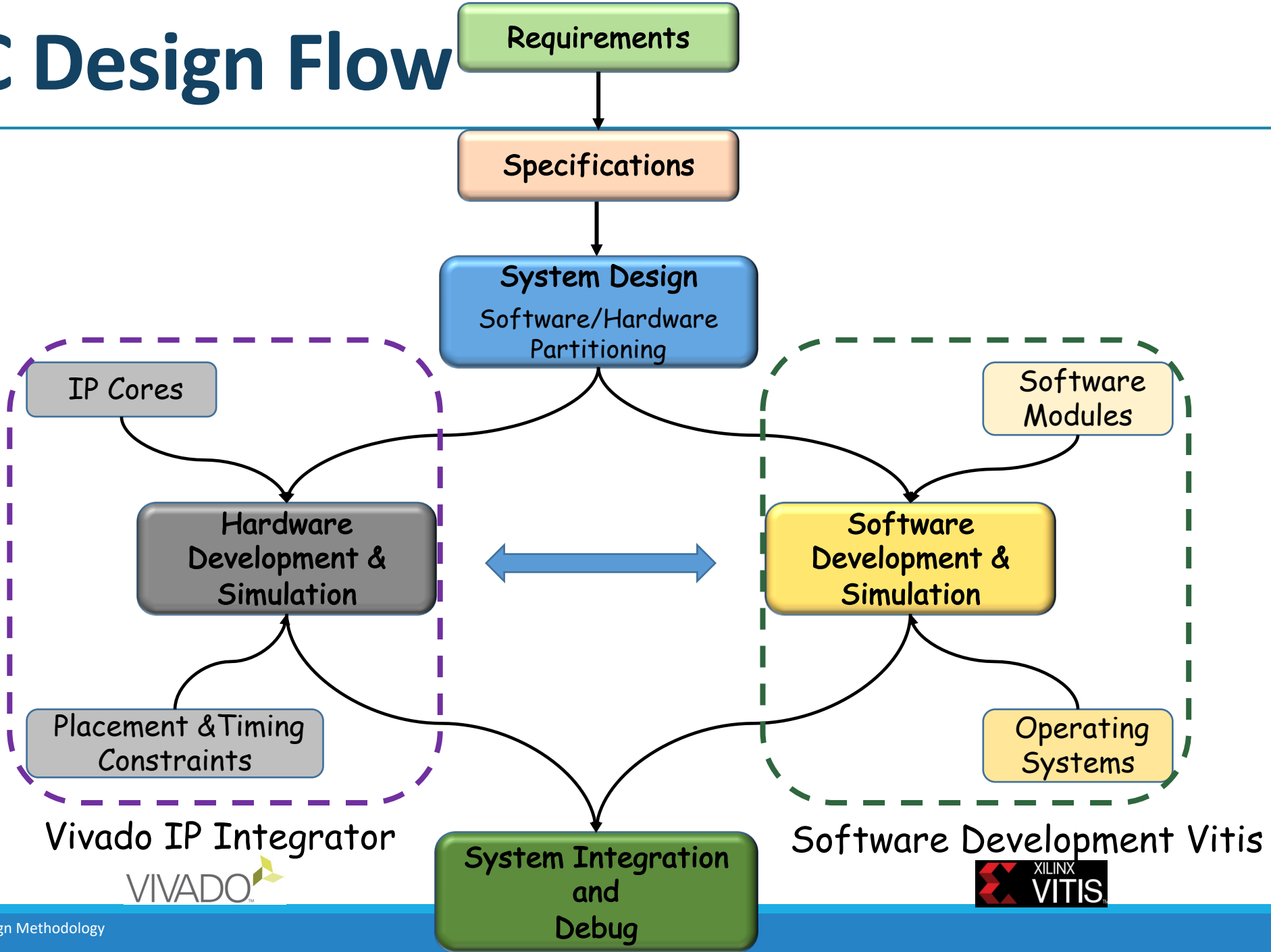
Zynq – Internal Device View



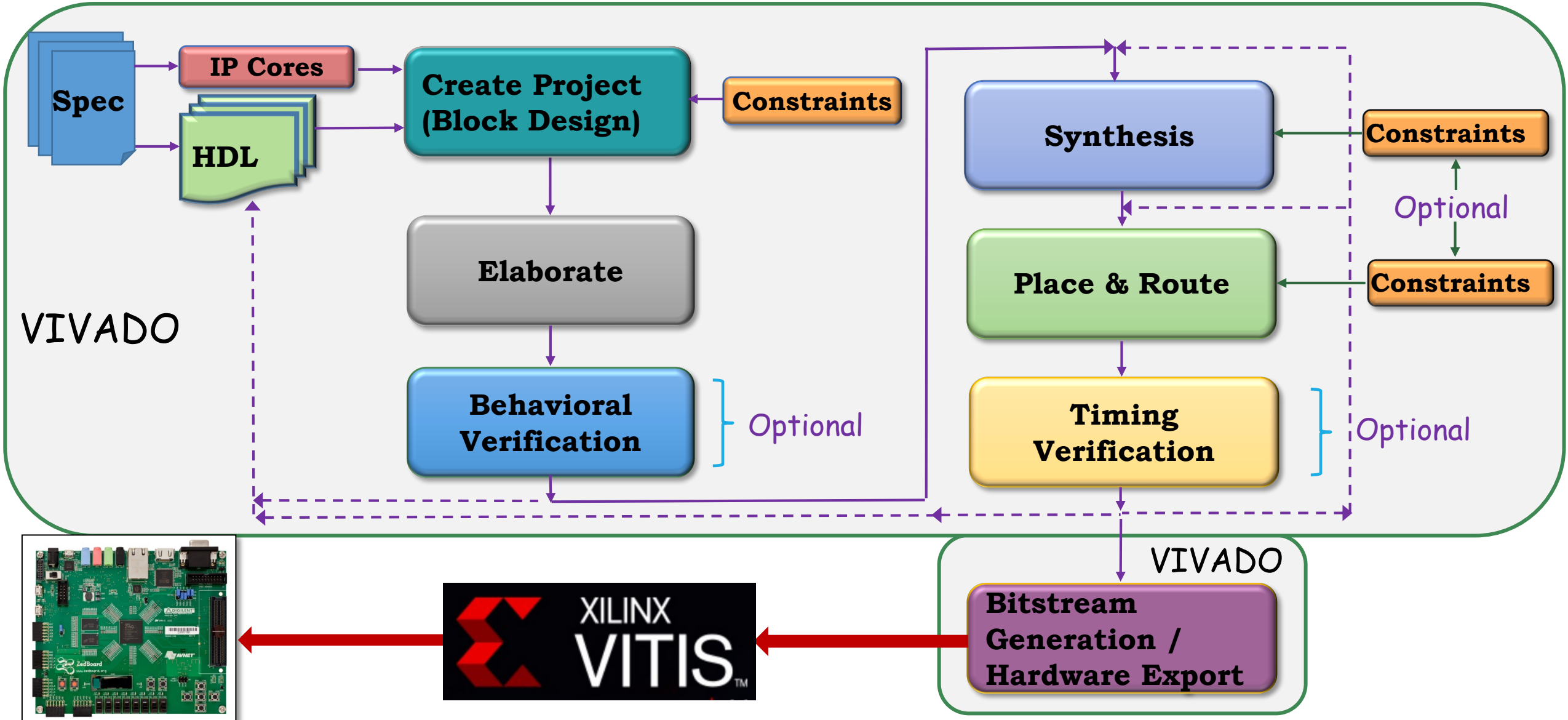
Zynq – Package View

SoPC Design Methodology

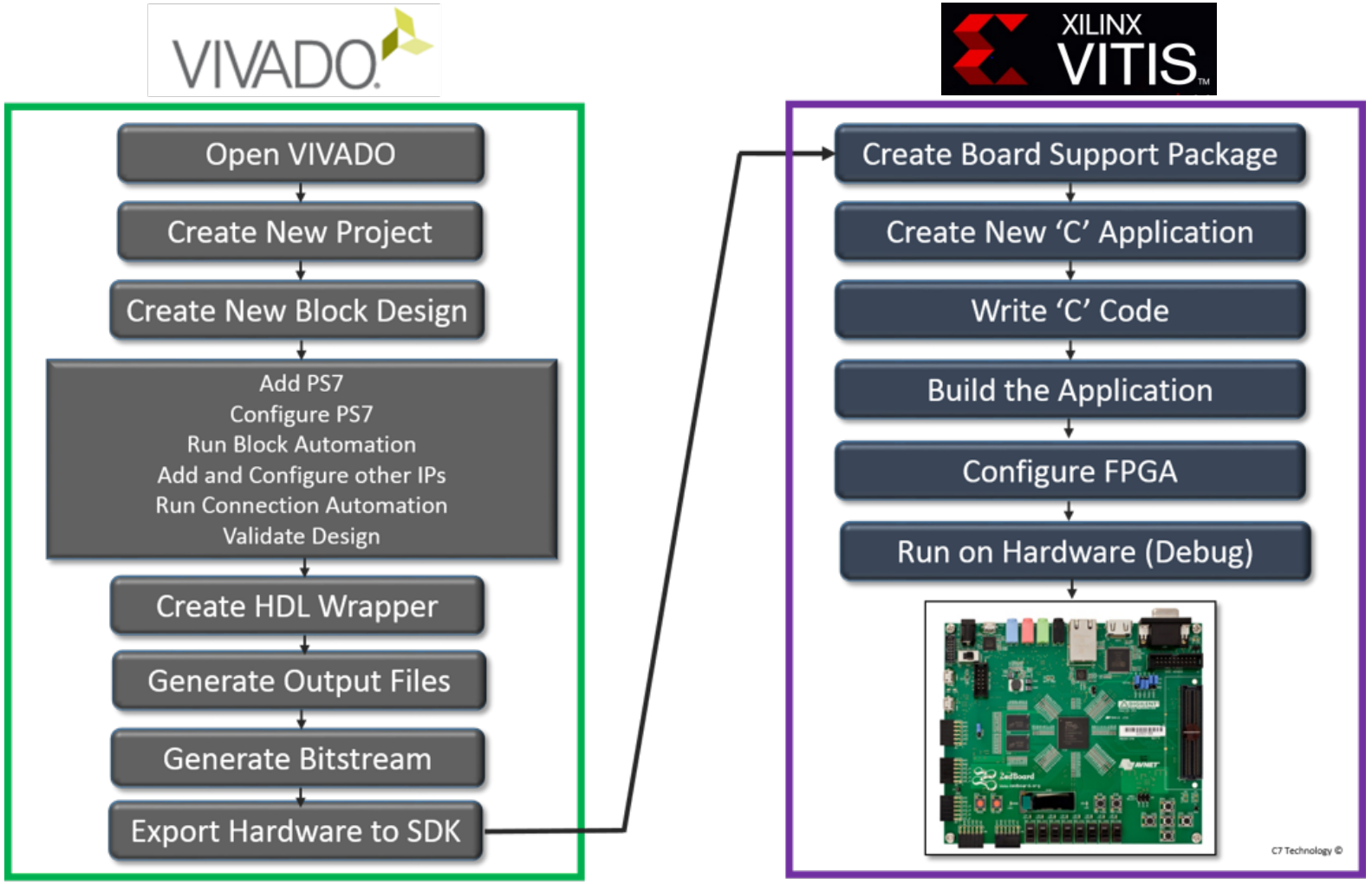
SoPC Design Flow



Embedded System Design – Vivado Flow



Embedded System Design – Vivado-Vitis Design Flow

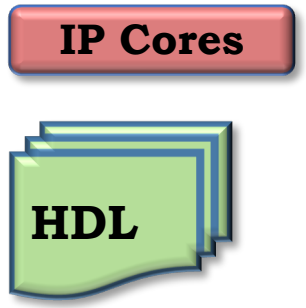


Design Specification

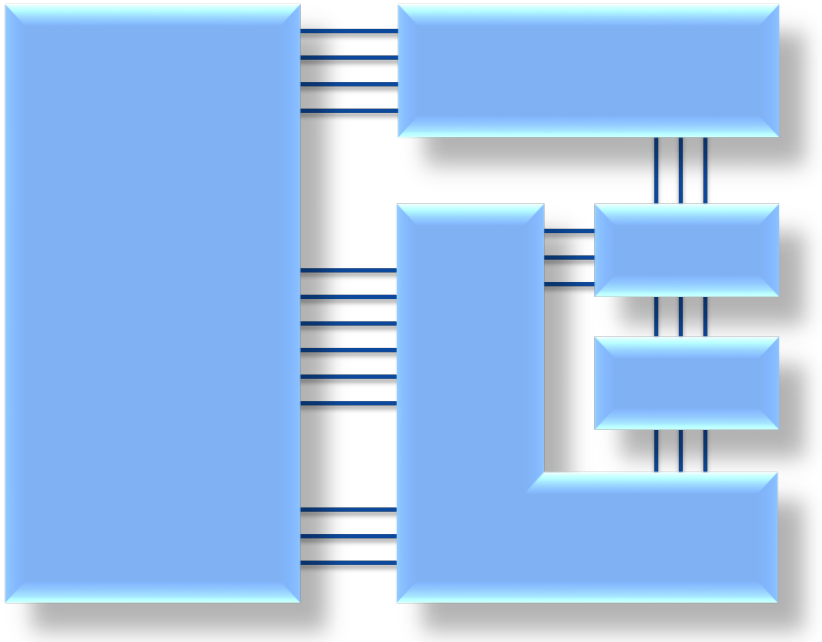
- ✓ Written specifications for the design to be done
- ✓ Spec can specify:
  - ✓ Functionality
  - ✓ Timing
  - ✓ Interfaces
  - ✓ Power

Design Partition

- ✓ Divide and conquer strategy
- ✓ Complex design is progressively partitioned into smaller and simpler functional units. This is known as
- ✓ **Top-Down Design or Hierarchical Design**



- ✓ Behavioral model for each functional unit are written
  - ✓ It has its own synthesis results
  - ✓ It has it own functional test bench
  - ✓ Some cases it has its own place and route and timing constraints



HDL Design

- **Behavioral modeling** describes the functionality of a component (design), that is, what the component will do
  - A behavioral prototype of a component can be quicly created
  - Its functionlatiy verified
  - Synthezised, optimized and mapped, to a specific technology

- **Structural Modeling** connect components to create a specific functionality.
  - Architectural partitioning forms a structural model, but the functional components are modeled behaviorally

Functional Verification – Block Level

- In general, a design should be partitioned along functional lines into smaller functional units, each having a common clock domain, and each of which is to be verified separately.

Behavioral Verification

✓ The verification process is threefold:

- ✓ Development of a test plan
- ✓ Development of test-benches
- ✓ Execution of the simulations

Functional Verification – Block/System Level (1)

- Development of a **test plan:** Specify ***what*** functional features are to be tested and ***how*** they are to be tested
  - For example, the test plan might specify that an exhaustive simulation of its behavior will verify the instruction set of an ALU.
  - Test plans for sequential machines must be more elaborate to ensure a high level of confidence in the design.
  - A test plan identifies the stimulus generators, response monitors, and the "gold" response against which the model will be tested.
  - ***Your grade, and your company's future, will depend on the care that you take in developing and executing your test plan.***

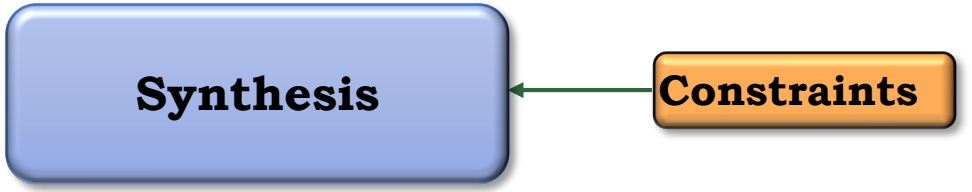
Functional Verification – Block/System Level (2)

- Test bench development
  - A Test Bench is a VHDL module in which the Unit Under Test (UUT) has to bee instantiated together with pattern generators that are to be applied to the inputs of the component during simulation.
    - Note: If a design is formed as an architecture of multiple modules, each must be verified separately, beginning with the lowest level of the design hierarchy, then the integrated design must be tested to verify that the modules interact correctly
- Test Bench execution
  - The test bench is ***exercised*** according to the test plan, and the response is verified against the original specification for the design

Design Integration – System Level Verification

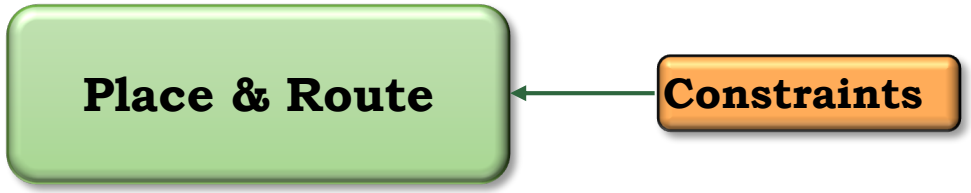
- After each of the functional sub components has been verified to have correct functionality, the architecture must be integrated and verified to have the correct overall functionality
  - A separate test plan for the system is developed at the beginning of this step.
  - This requires development of a separate testbench whose stimulus generators exercise the input/output functionality of the top-level module, monitor port and bus activity across module boundaries, and observe state activity in any embedded state machines.
  - This step in the design flow is crucial and must be executed thoroughly to ensure that the design that is being signed off for synthesis is correct.

Synthesis – Constraints – Post-Synthesis Simulation



- ✓ A synthesis tool is used to translate from ‘software (VHDL)’ to ‘hardware’: logic gates, flip-flops, memory, etc.
- ✓ A synthesis tool removes redundant logic and seeks to satisfy the requirements regarding the area of the logic needed to implement the functionality and the performance (speed) specifications
- ✓ Post-Synthesis simulation is, in general, **optional**, but it is advisable in case of using specific synthesis attributes.

Place & Route



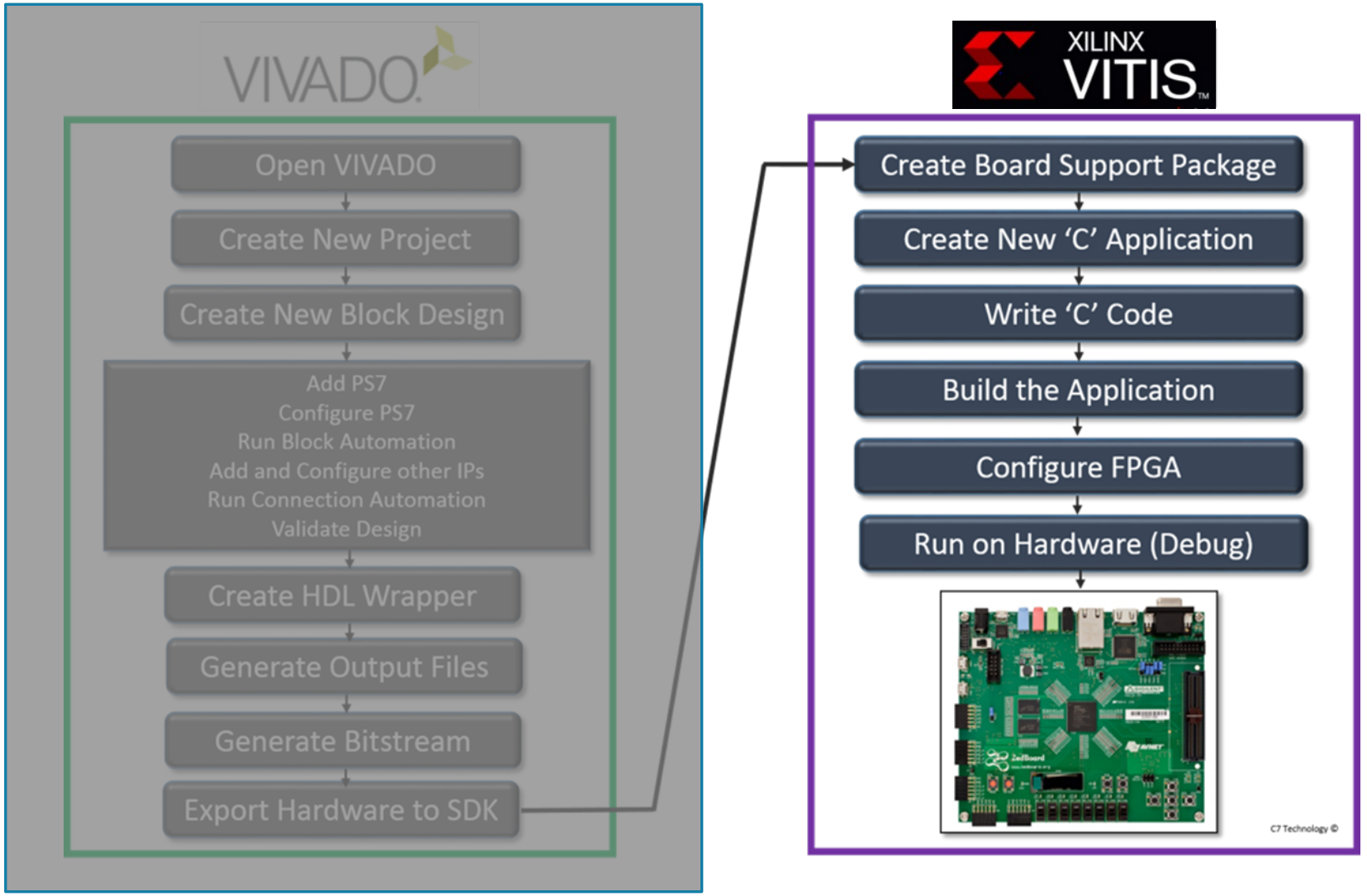
- ✓ The logic generated by the synthesis tool is a netlist, commonly known as EDIF netlist, that is take by the Place and Route tool to scatter the logic in the FPGA's resources.
- ✓ P&R tool has different effort levels, which can be used in case the final result does not meet the needed requirements.
- ✓ In complex design:
  - ✓ P&R could take several hours to accomplish its task.
  - ✓ Some floorplanning may be needed (constraints).

Post Place & Route Simulation

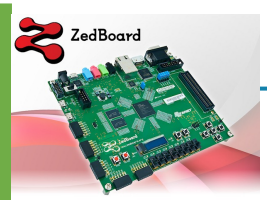
Timing Verification — Optional

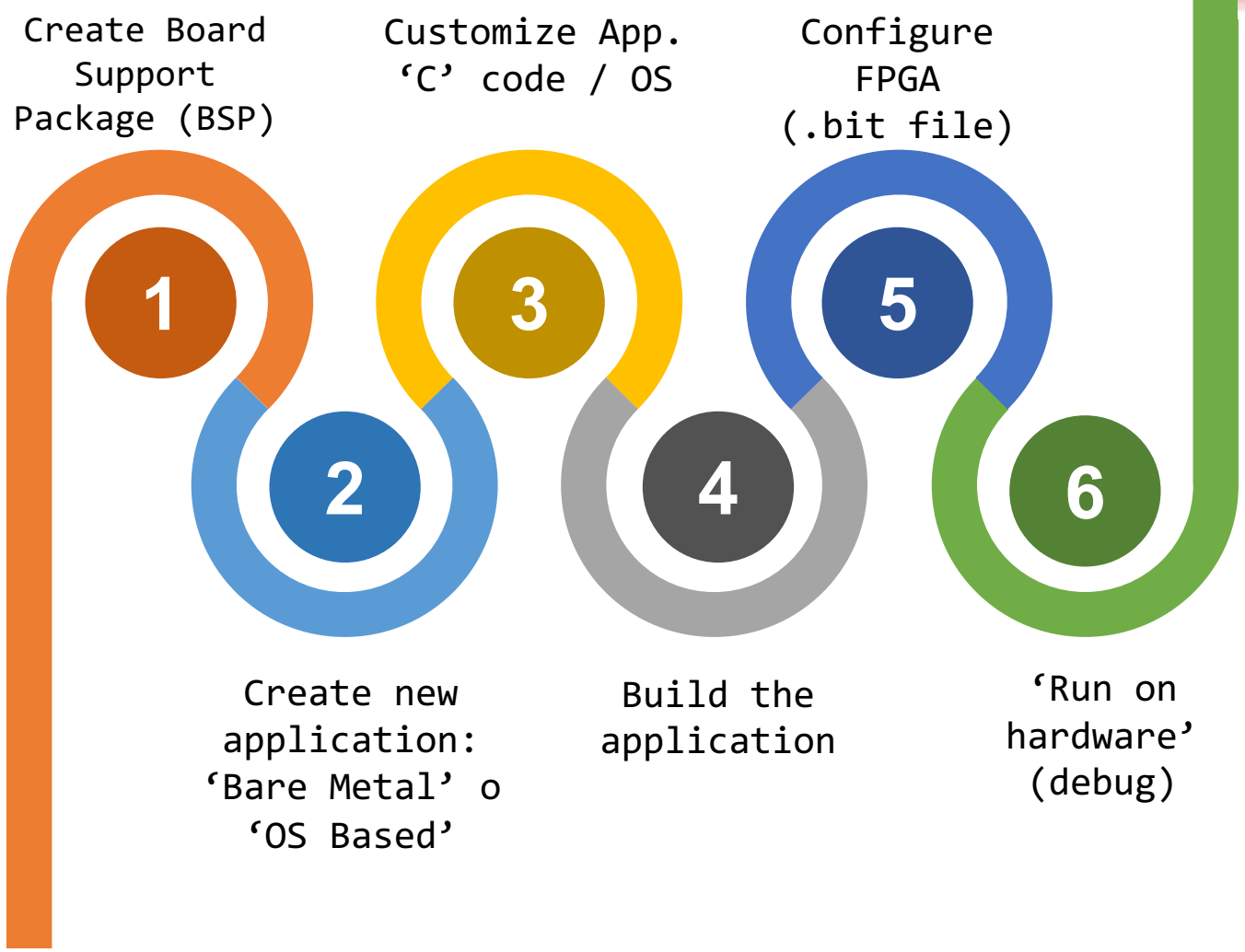
- ✓ The simulation test (test bench) not only test the logical/behavioural functionality but also the timing of the whole system:
  - ✓ Routing and logic delay are taking into consideration when executing this simulation
  - ✓ Each delay is well know after the P&R
  - ✓ Hold-time and Set-up time violations can be find out in this simulation as well as any glitch

Vitis Flow Design



Vitis Design Flow





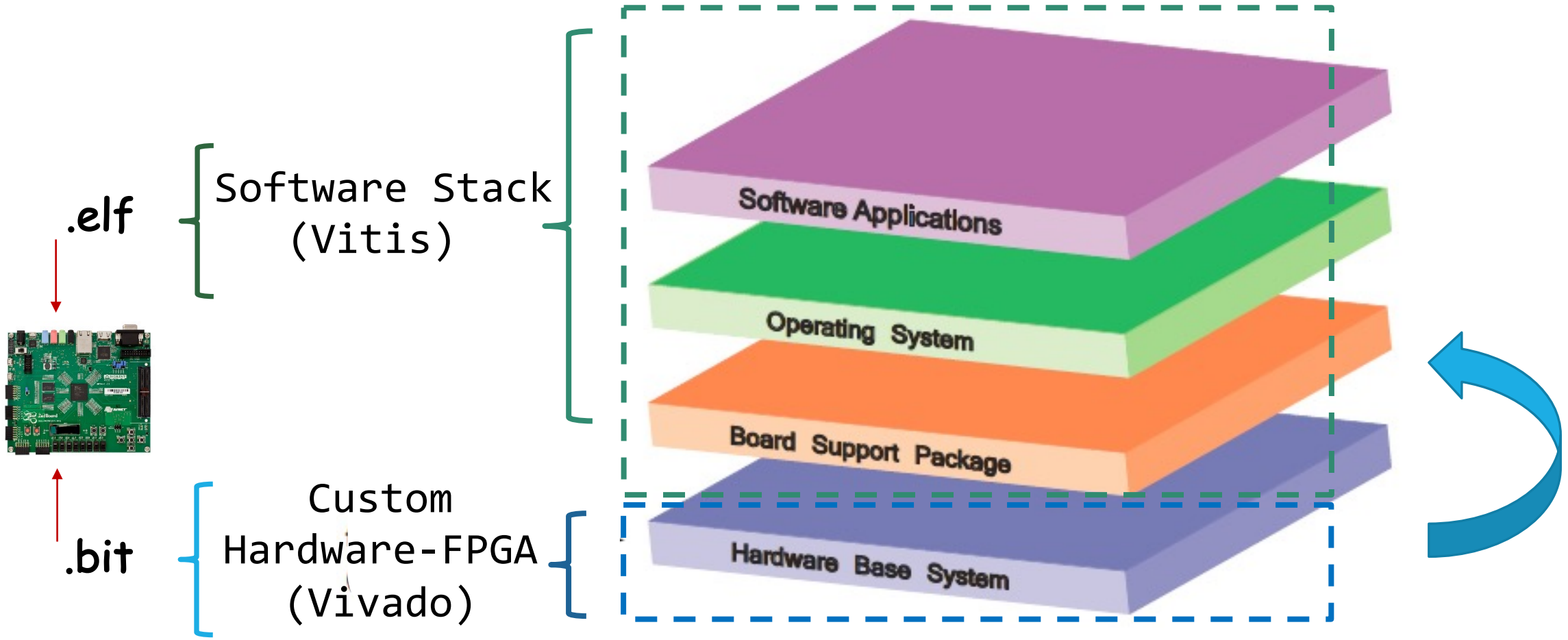
Vitis Design Flow

- Vitis provides a development environment identical for stand-alone (‘C’ *bare metal*) and Linux based developments.
- The Vivado hardware is exported to Vitis:
  - Detail the different hardware components.
  - Specify the memory locations assigned to the different components of the system.
  - Links to peripheral datasheets.
- Vitis generates Board Support Package (BSP): provides specific support code for a given system
  - Describes what libraries are used, how stdin/stdout are mapped, etc.
  - Provides device drivers for the generated hardware
  - Abstract Hardware from Software.
  - More than one BSP can exist per hardware system generated in Vivado.

Vitis - Running the App. - .elf .bit Download

- First of all, configure the PL part of the Zynq, using the .bit file
- Download the .elf file to the PS (processor) part of the Zynq.
- Execute the C/Linux application
- Check the result into either:
  - Terminal
  - Console
  - 3rd party terminal emulator

Vitis Design Flow - .elf .bit Download



Basics of TCL in Vivado

Tool Command Language

TCL , is *an interpreted programming language* with variables, procedures , and control structures, to interface to *a variety of design tools and to design data.*

It has been an industry standard language since early 90s’

AMD-Xilinx adopted TCL for the Vivado Design Suite

Tool Command Language (cont)

TCL in Vivado enables the designer to:

- Create a project
- Target a SoPC device/board
- Create a block design
- Include IP Cores
- Configure PS, IP Cores, etc.
- Run synthesis
- Run implementation
- Modify P&R options
- Customize reports
- Program SoPC

Tool Command Language (cont)

The **Vivado** tools write a journal file called ***vivado.jou*** into the directory from which **Vivado** was launched. The journal is a record of the **Tcl** commands run during the session.

Thus, they can be used as a starting point to create a new **Tcl** script.

Tool Command Language (cont)

```
#-----------------------------------------------------------
# Vivado v2018.3.1 (64-bit)
# SW Build 2489853 on Tue Mar 26 04:18:30 MDT 2019
# IP Build 2486929 on Tue Mar 26 06:44:21 MDT 2019
# Start of session at: Wed May 22 20:07:21 2019
# Process ID: 19219
# Current directory: /cris_projects
# Command line: vivado
# Log file: /cris_projects/vivado.log
# Journal file: /cris_projects/vivado.jou
#-----------------------------------------------------------
start_gui
create_project project_1 /cris_projects/ZedBoard/borrar/hw -part
xc7z020clg484-1
set_property board_part em.avnet.com:zed:part0:1.4
[current_project]
set_property target_language VHDL [current_project]
create_bd_design "design_1"
update_compile_order -fileset sources_1
startgroup
create_bd_cell -type ip -vlnv
xilinx.com:ip:processing_system7:5.5 processing_system7_0
endgroup
apply_bd_automation -rule
xilinx.com:bd_rule:processing_system7 -config {make_external
"FIXED_IO, DDR" apply_board_preset "1" Master "Disable" Slave
"Disable" }  [get_bd_cells processing_system7_0]
generate_target all [get_files
/cris_projects/ZedBoard/borrar/hw/project_1.srcs/sources_
1/bd/design_1/design_1.bd]
startgroup
```

How to run a provided .tcl script

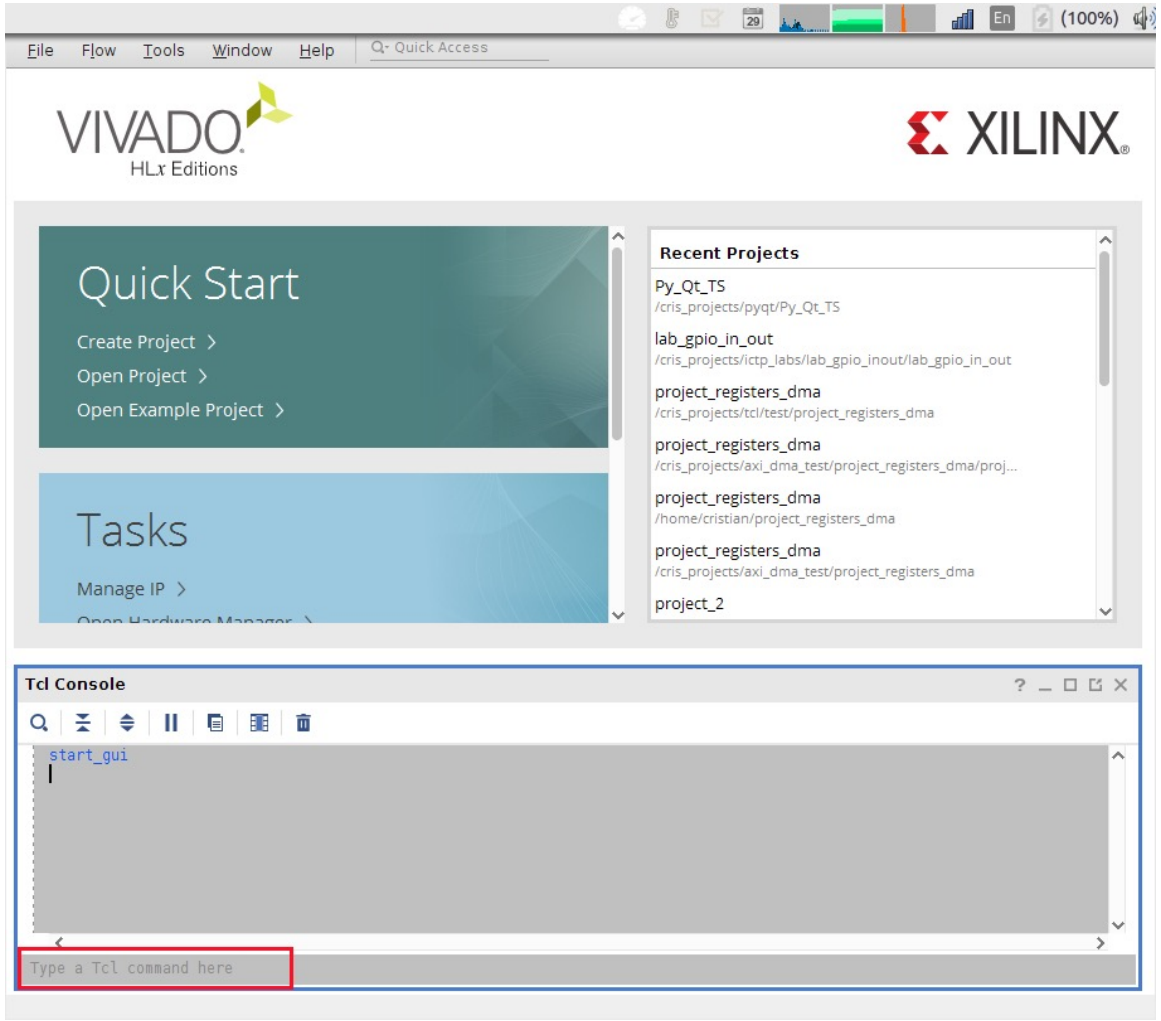
- Methot 1: Through Vivado TCL console
- Method 2: Through Command Line

Method 1: Run .tcl in Vivado TCL Console

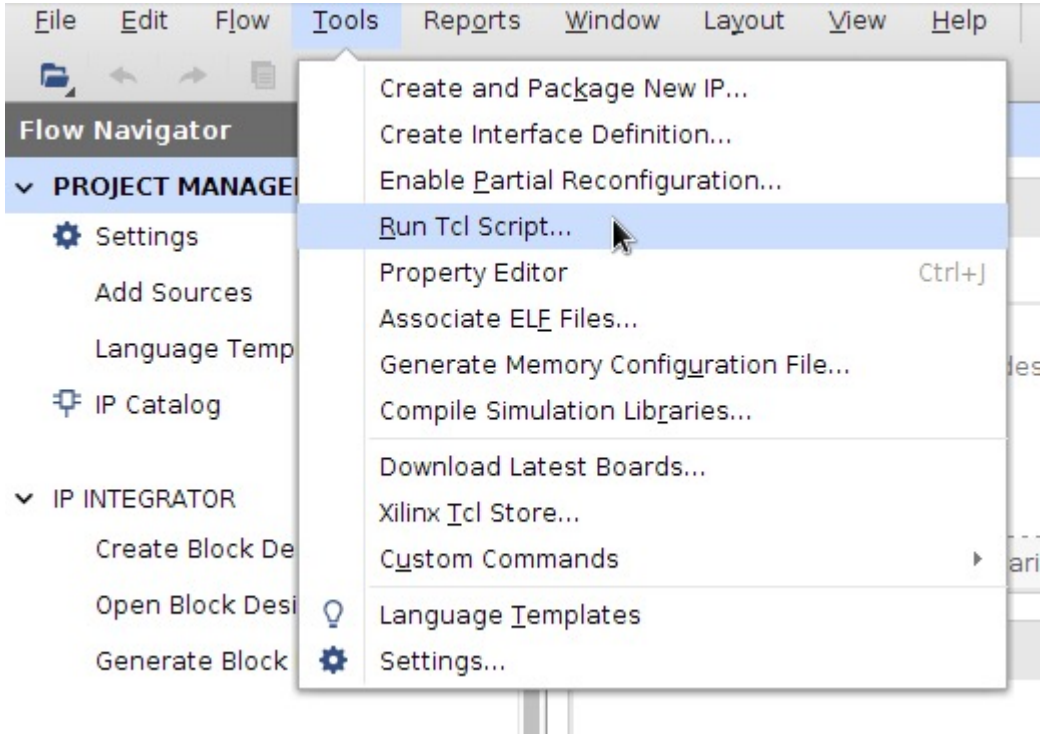
1. Start *Vivado Design Suite*. You can see a *tcl* console on the left bottom of *Vivado Design Suite*
2. Click on the title *'type a tcl command here'* (button left of the screen)
3. Go to the folder location where the tcl script resides (use 'cd', 'pwd')
4. Once the directory has been changed, you can use the *'ls'* command to list the files in the current directory. Check that the .tcl is in there.
5. Run the .tcl script by using the following command:

   source <filename>.tcl
6. The processes defined in the .tcl file will be executed. It could take sometimes to execute a .tcl file (depending on the defined processes)

Vivado TCL Console



Vivado TCL Option in the GUI



Method 2: Run .tcl through Command Line W10/11

1. In W10 you can start the Vivado TCL Shell by doing:

   Start-> All apps->Vivado 20xx.x Tcl Shell

2. A small command line window should come up
3. Go to the folder location where the tcl script resides (use ‘cd’, ‘pwd’)
4. Once the directory has been changed, you can use the *‘dir’* command to list the files in the current directory. Check that the **.tcl** is in there.
5. Run the **.tcl** script by using the following command:

   source <filename>.tcl

6. The processes defined in the .tcl file will be executed. It could take some times to execute a .tcl file (depending on the defined processes)

Run .tcl in Linux

1. Make sure TCL interpreter is installed:

```
$whereis tclsh
tclsh: /usr/bin/tclsh /usr/bin/tclsh8.4 /usr/share/man/man1/tclsh.1.gz
```

2. In case you don't have the tcl interpreter installed, do the following:

```
$ sudo apt-get install tcl8.4
```

Note: if you have already installed Vivado, the Tcl interpreter should be installed

3. Execute TCL script:

```
$ tclsh helloworld.tcl
```

(or)

```
$ chmod u+x helloworld.tcl
$ ./helloworld.tcl
```

Is there any Need to Learn TCL ?

It is purely based on your objectives.

If you want *to automate some basic processes* in creating design , **it is the best choice** as we can export a ***tcl*** script to another computer *and create an exact replica of the project* with same configurations, ip integrations in just a single execution.

Xilinx TCL Docs

Vivado Design Suite TCL Command Reference Guide

Vivado Design Suite User Guide - Using TCL Scripting

TCL Tutorial (up to Chapter 14 for Vivado appl)

Apendix

PS I/O Peripherals

I2C

- I^2C bus specification version 2
- Programmable to use normal (7-bit) or extended (10-bit) addressing
- Programmable rates: fast mode (400 kbit/s) , standard (10 0kbits/s), and low (10 kbits/s)
  - Rates higher than 400 kbits/sec are not supported
- Programmable as either a master or slave interface
- Capable of clock synchronization and bus arbitration
- Fully programmable slave response address
- Reversible FIFO operation supported
- 16-byte buffer size
- Slave monitor mode when set up as master
- I^2C bus hold for slow host service
- Slave timeout detection with programmable period
- Transfer status interrupts and flags

CAN

- Up to 24-MHz CAN_REF clock as system clock
- 64 message-deep receiver and transmitter buffer
- Full CAN 2.0B compliant; conforms to ISO 11898-1
- Maximum baud rate of 1 Mb/s
- Four message filters required for buffer mode
- Listen-only mode for test and debug
- External PHY I/O
- “Wake-on-message”
- Time-stamping for receive messages
- TX and RX FIFO watermarking
- Exception: no power-down mode

SD-SDIO

- Support for version 2.0 of SD Specification
- Full-speed (4 MB/s) and low-speed (2 MB/s) support
  - Low-speed clock (400 KHz) used until bandwidth negotiated
- 1-bit and 4-bit data interface support
- Host mode support only
- Built-in DMA controller
- Full-speed clock (0-50 MHz) with maximum throughput at 25 MB/s
- 1 KB data FIFO interface
- Support for MMC 3.31 card at 52 MHz
- Support for memory, I/O, and combo cards
- Support for power control modes and interrupts

SPI

- Master or slave SPI mode
- Four wire bus: MOSI, MISO, SCK, nSS
- Supports up to three slave select lines
- Supports multi-master environment
- Identifies an error condition if more than one master detected
- Software can poll for status or function as interrupt-driven device
- Programmable interrupt generation
- 50-MHz maximum external SPI clock rate
- Selectable master clock reference
- Integrated 128-byte deep read and write FIFOs
- Full-duplex operation offers simultaneous receive and transmit

UART

- Two UARTs
- Programmable baud rate generator
- 64-byte receive and transmit FIFOs
- 6, 7, or 8 data bits and 1, 1.5, or 2 stop bits
- Odd, even, space, mark, or no parity with parity, framing, and overrun error detection
- "Line break" generation and detection
- Normal, automatic echo, local loopback, and remote loopback channel modes
- Interrupts generation
- Support 8 Mb/s maximum baud rate with additional reference clock or up to 1.5 Mb/s with a 100-MHz peripheral bus clock
- Modem control signals: CTS, RTS, DSR, DTR, RI, and DCD (through EMIO)
- Simple UART: only two pins (TX and RX through MIO)

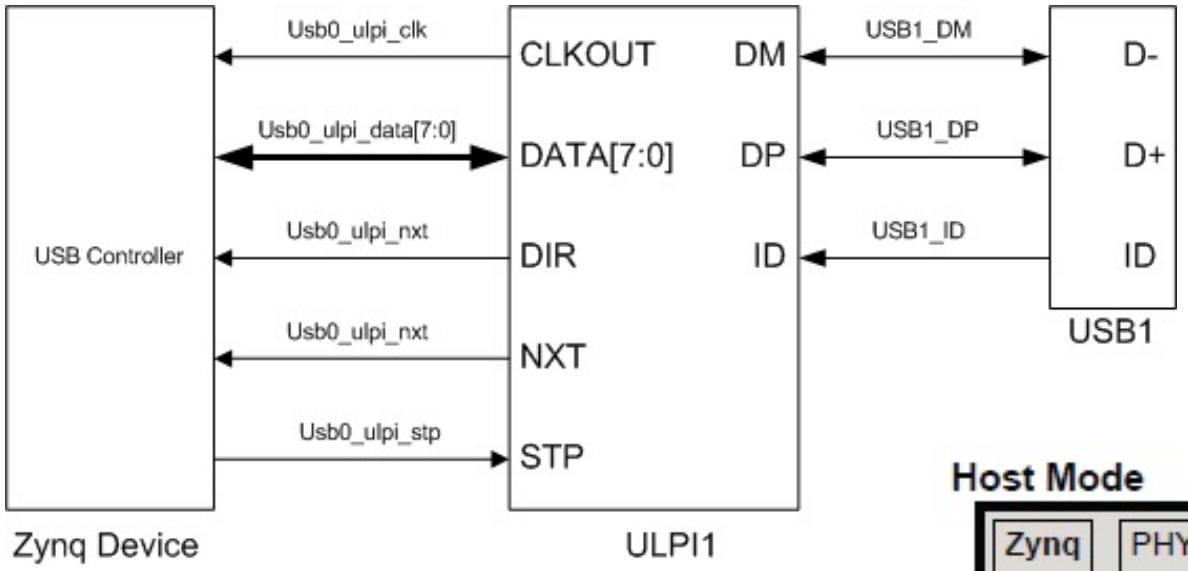
USB

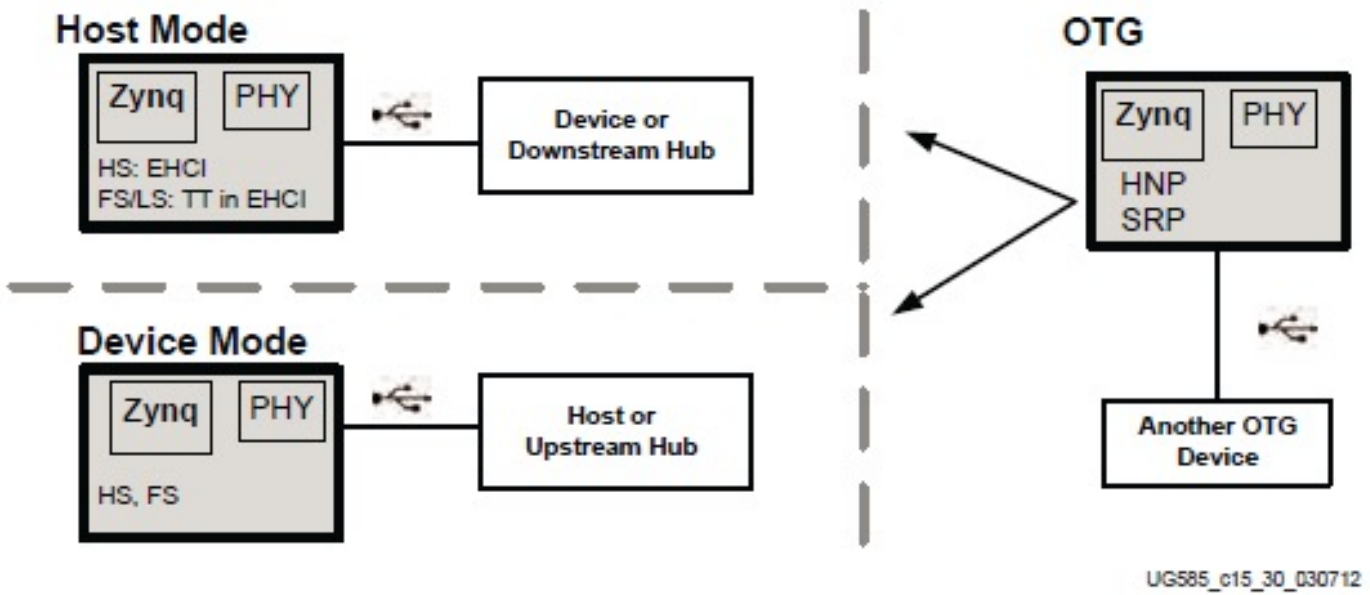
- Two USB 2.0 hardened IP peripherals per Zynq device
  - Each independently controlled and configured
- Supported interfaces
  - High-speed USB 2.0: 480 Mbit/s
  - Full-speed USB 1.1: 12 Mbit/s
  - Low-speed USB 1.0: 1.5 Mbit/s
  - Communication starts at USB 2.0 speed and drops until sync is achieved
- Each block can be configured as host, device, or on-the-go (OTG)
- 8-bit ULPI interface
- All four transfer types supported: isochronous, interrupt, bulk, and control
- Supports up to 12 endpoints per USB block in the Zynq device
  - Running in host mode
- Source-code drivers

USB 2.0 OTG

- Control and configuration registers for each USB block
- Software-ready with standalone and OS linux source-code delivered drivers
- EHCI compliant host registers
- USB host controller registers and data structures compliant to Intel EHCI specifications
- Internal DMA
- Must use the MIO pins

USB 2.0 Usage Example

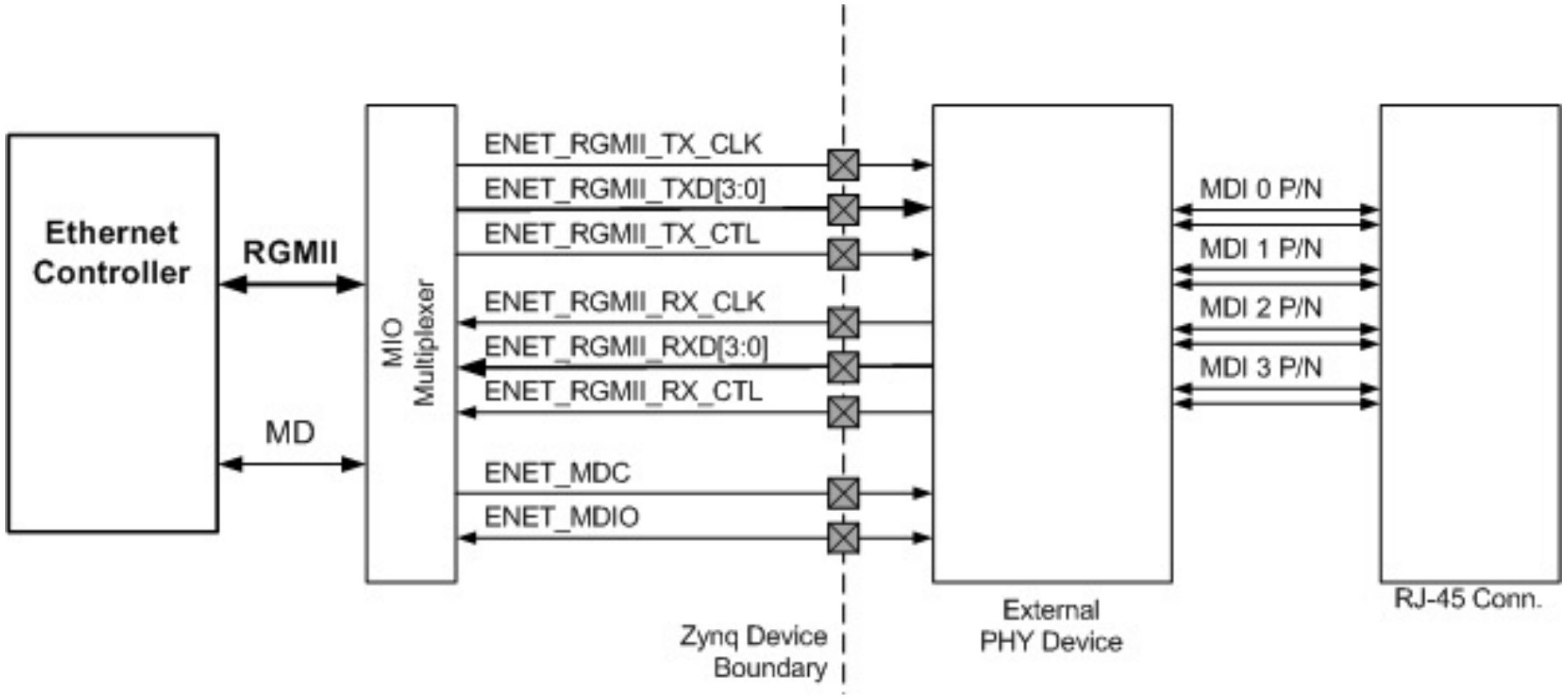




Gigabit Ethernet Controller

- Tri-mode Ethernet MAC (10/100/1G) with native GMII interface
- IEEE1588 rev 2.0
  - Time stamp support
  - 1 us resolution
- IEEE802.3
- RGMII v2.0 (HSTL) interface to MIO pins
  - Need MIO set at 1.8V to support RGMII speed
  - Need to use large bank of MIO pins for two Ethernets
- MII/GMII/SGMII/RGMII ver1.3 (LVCMOS) and ver2.0 (HSTL) interface available through EMIO (programmable logic I/O)
- TX/RX checksum offload for TCP and UDP
- Internal DMA and wake on LAN

Gigabit Ethernet Controller

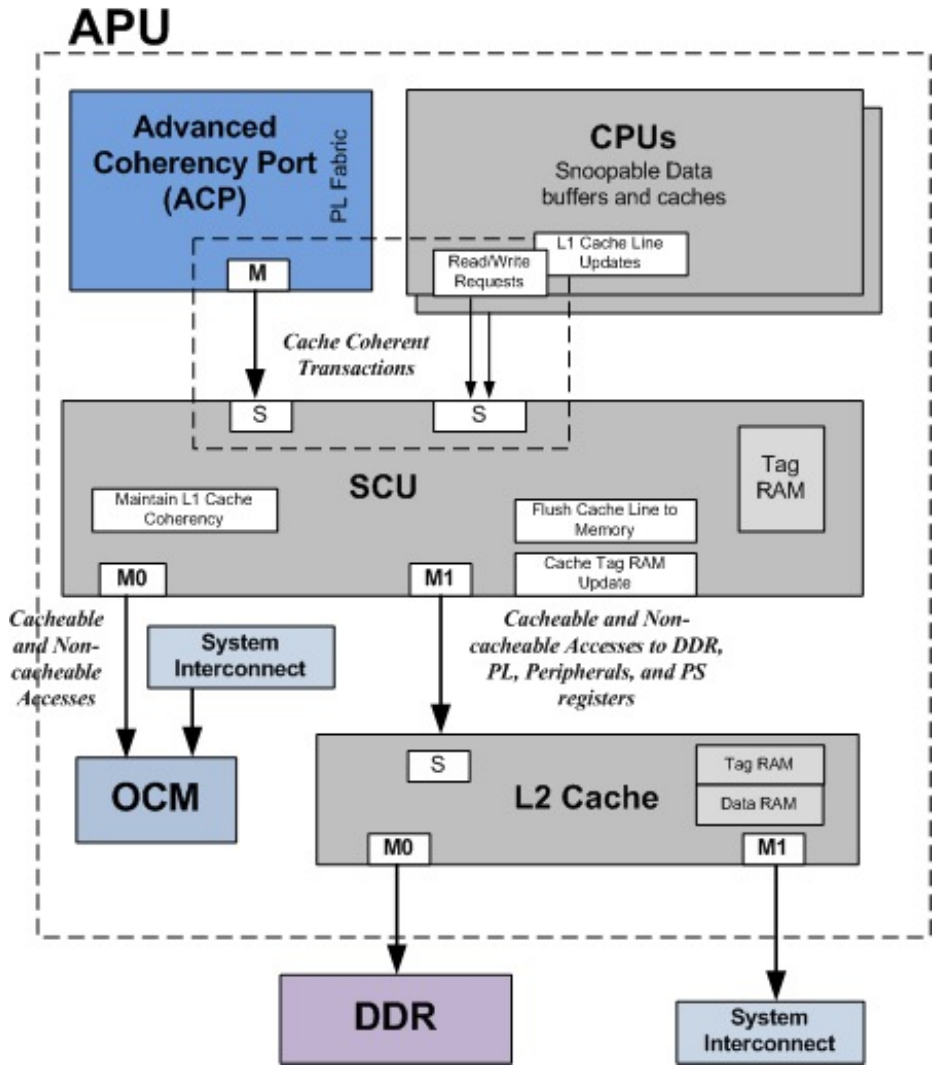


Application Processor Unit (APU)

ARM Processor Architecture

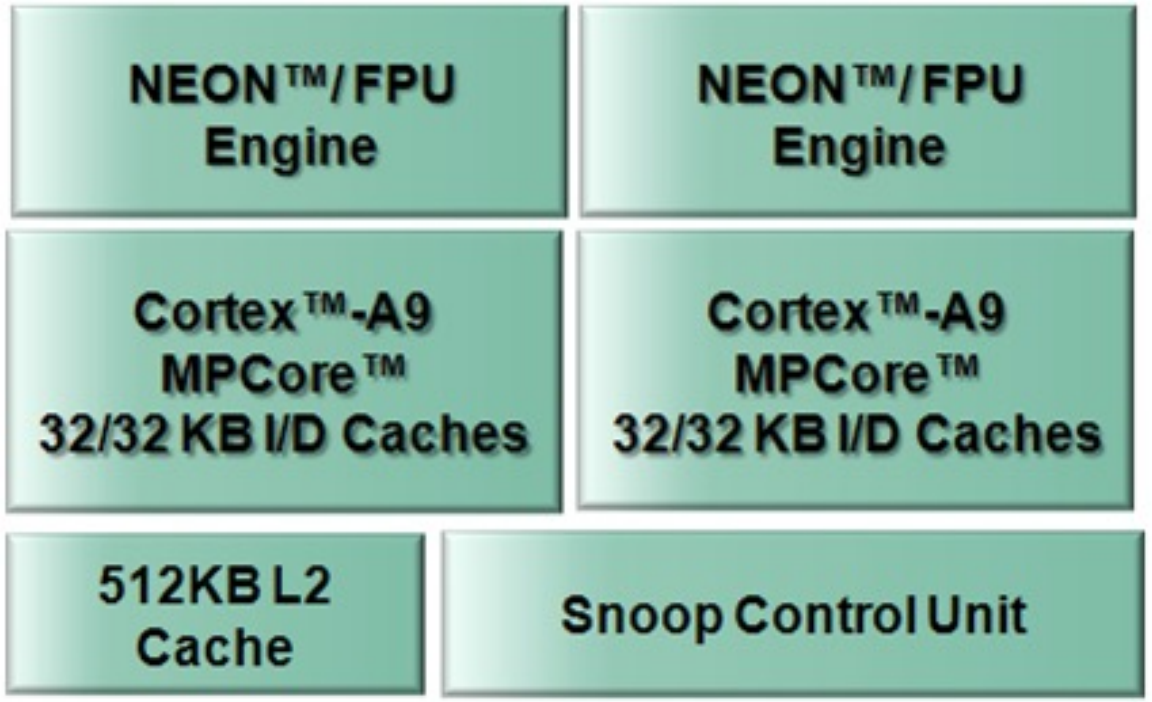
- **ARM Cortex-A9 processor implements the ARMv7-A architecture**
  - ARMv7 is the ARM Instruction Set Architecture (ISA)
    - Thumb instructions: 16 bits; Thumb-2 instructions: 32 bits
    - NEON: ARM's Single Instruction Multiple Data (SIMD) instructions
  - ARMv7-A: Application set that includes support for a Memory Management Unit (MMU)
  - ARMv7-R: Real-time set that includes support for a Memory Protection Unit (MPU)
  - ARMv7-M: Microcontroller set that is the smallest set
- **ARM Advanced Microcontroller Bus Architecture (AMBA®) protocol**
  - AXI3: Third-generation ARM interface
  - AXI4: Adding to the existing AXI definition (extended bursts, subsets)
- **Cortex is the new family of processors**
  - ARM family is older generation; Cortex is current; MMUs in Cortex processors and MPUs in ARM

APU


APU
Advanced Coherency Port (ACP)
PL Fabric
M
CPUs
Snoopable Data buffers and caches
L1 Cache Line Updates
Read/Write Requests
Cache Coherent Transactions
S
S
SCU
Tag RAM
Maintain L1 Cache Coherency
Flush Cache Line to Memory
Cache Tag RAM Update
M0
M1
Cacheable and Non-cacheable Accesses
System Interconnect
Cacheable and Non-cacheable Accesses to DDR, PL, Peripherals, and PS registers
OCM
S
L2 Cache
Tag RAM
Data RAM
M0
M1
DDR
System Interconnect

APU Components

- Dual ARM® Cortex™-A9 MPCore with NEON extensions
  - Up to 800-MHz operation
  - 2.5 DMIPS/MHz per core
  - Separate 32KB instruction and data caches
- Snoop Control Unit (SCU)
  - L1 cache snoop control
    - Accelerator coherency port
- Level 2 cache and controller
  - Shared 512 KB cache with parity



APU Sub-Components

- General interrupt controller (GIC)
- On-chip memory (OCM): RAM and boot ROM
- Central DMA (eight channels)
- Device configuration (DEVCFG)
- Private watchdog timer and timer for each CPU
- System watchdog and triple timer counters shared between CPUs
- ARM CoreSight debug technology

APU Address Map

- All registers for both CPUs are grouped into two contiguous 4KB pages
  - Accessed through a dedicated internal bus
- Fixed at 0xF8F0_0000 with a register block size of 8 KB
  - Each CPU uses an offset into this base address

0x0000-0x00FC	SCU registers
0x0100-0x01FF	Interrupt controller interface
0x0200-0x02FF	Global timer
0x0600-0x06FF	Private timers and watchdog timers
0x1000-0x1FFF	Interrupt distributor

NEON Main Features

- NEON is the ARM codename for the vector processing unit
  - Provides multimedia and signal processing support
- FPU is the floating-point unit extension to NEON
  - Both NEON and FPU share a single set of registers
- NEON technology is a wide single instruction, multiple data (SIMD) parallel and co-processing architecture
  - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide)
  - Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, or 32-bit float

L1 Cache Features

- Separate instruction and data caches for each processor
- Caches are four-way, set associative and are write-back
- Non-lockable
- Eight words cache length
- On a cache miss, critical word first filling of the cache is performed followed by the next word in sequence

L2 Cache Features

- 512K bytes of RAM built into the SCU
  - Latency of 25 CPU cycles
  - Unified instruction and data cache
- Fixed, 256-bit (32 words) cache line size
- Support for per-master way lockdown between multiple CPUs
- Eight-way, set associative
- Two AXI interfaces
  - One to DDR controller
  - One to programmable logic master (to peripherals)

On-Chip Memory (OCM)

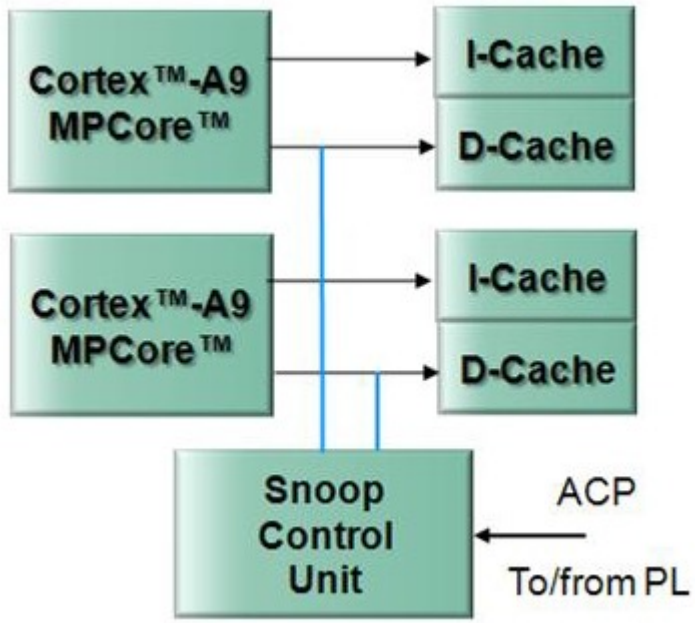
- The on-chip memory (OCM) module contains 256 KB of RAM and 128 KB of ROM (BootROM).
- It supports two 64-bit AXI slave interface ports, one dedicated for CPU/ACP access via the APU snoop control unit (SCU), and the other shared by all other bus masters within the processing system (PS) and programmable logic (PL).
- The BootROM memory is used exclusively by the boot process and is not visible to the user.

Snoop Control Unit (SCU)

- Shares and arbitrates functions between the two processor cores
  - Data cache coherency between the processors
  - Initiates L2 AXI memory access
  - Arbitrates between the processors requesting L2 accesses
  - Manages ACP accesses
  - A second master port with programmable address filtering between OCM and L2 memory support

Cache Coherency Using SCU

- High-performance, cache-to-cache transfers
- Snoop each CPU and cache each interface independently
- Coherency protocol is MESI
  - M: Cache line has been modified
  - E: Cache line is held exclusively
  - S: Cache line is shared with another CPU
  - I: Cache line is invalidated
- Uses Accelerator Coherence Port (ACP) to allow coherency to be extended to PL



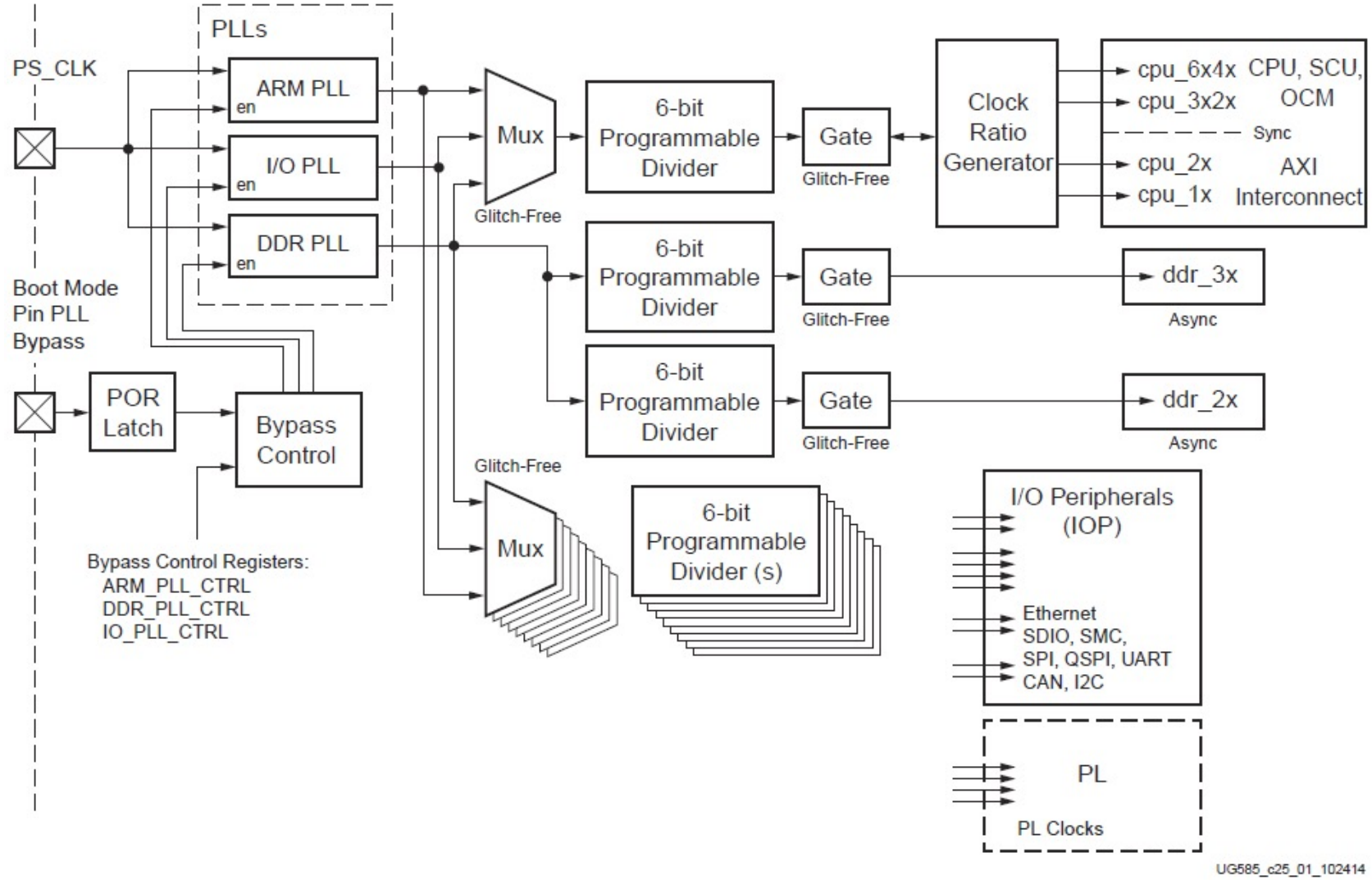
System Level Control Register (SLCR)

- A set of special registers in the APU used to configure the PS
  - Power and clock management
  - Reset control
  - MIO/EMIO management
- Accessible through software
  - Standalone BSP support

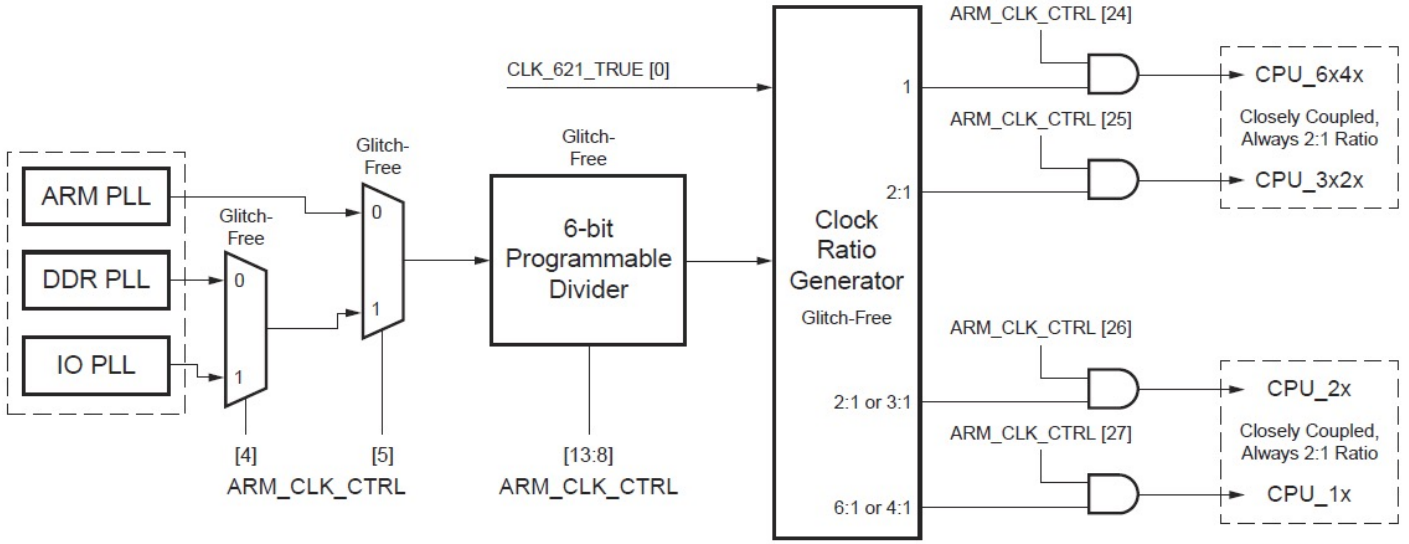
SLCR Categories	
System clock and reset control/status registers	TrustZone control register
APU control registers	SoC debug control registers
DMA initialization registers	MIO/IOP control/status registers
DDR control registers	Miscellaneous control registers
PL reset registers	RAM and ROM control registers

Zynq Clocks

System Clocks

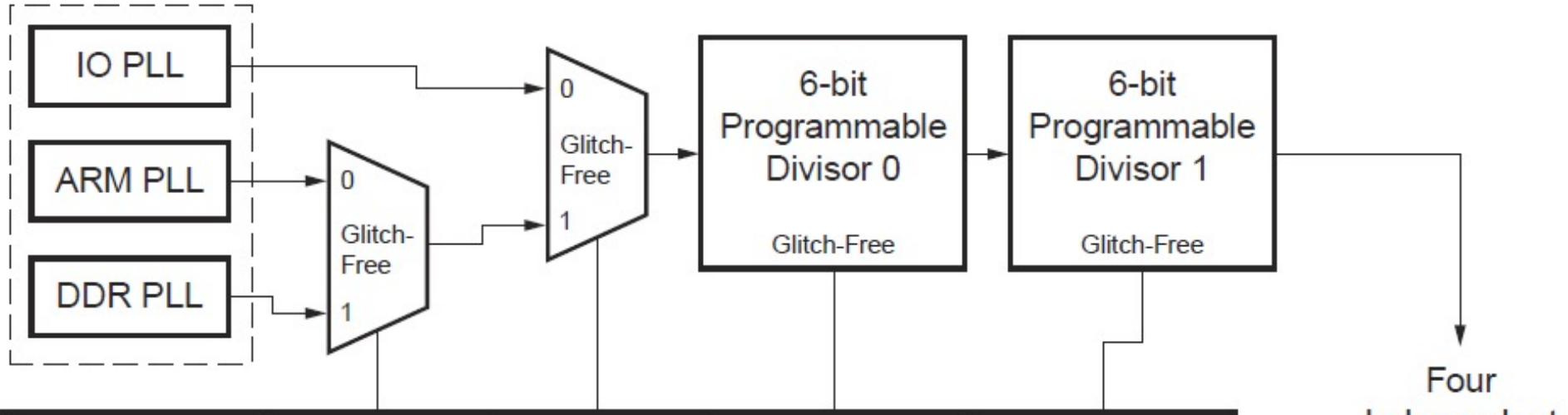


CPU Clock



CPU Clock	6:2:1	4:2:1	Clock Domain Modules
CPU_6x4x	800 MHz (6 times faster than CPU_1x)	600 MHz (4 times faster than CPU_1x)	CPU clock freq, SCU, OCM arbitrer, NEON and L2 Cache
CPU_3x2x	400 MHz (3 times faster than CPU_1x)	300 MHz (2 times faster than CPU_1x)	APU Timers
CPU_2x	266MHz (2 times faster than CPU_1x)	300 MHz (2 times faster than CPU_1x)	IOP, central interconnect, master interconeect, slave interconnect and OCM RAM
CPU_1x	133 MHz	150 MHz	IOP, AHB and APB interface busses

PL Clocks



PL FCLK Clock	Control Register	Mux Ctrl Field	Mux Ctrl Field	Divider 0 Ctrl Field	Divider 1 Ctrl Field	
PL FCLK 0	FPGA0_CLK_CTRL	SRCSEL, 4	SRCSEL, 5	DIVISOR 0, 13:8	DIVISOR 1, 25:20	→ FCLKCLK0
PL FCLK 1	FPGA1_CLK_CTRL	SRCSEL, 4	SRCSEL, 5	DIVISOR 0, 13:8	DIVISOR 1, 25:20	→ FCLKCLK1
PL FCLK 2	FPGA2_CLK_CTRL	SRCSEL, 4	SRCSEL, 5	DIVISOR 0, 13:8	DIVISOR 1, 25:20	→ FCLKCLK2
PL FCLK 3	FPGA3_CLK_CTRL	SRCSEL, 4	SRCSEL, 5	DIVISOR 0, 13:8	DIVISOR 1, 25:20	→ FCLKCLK3

UG585_c25_10_041612