ADIUVO

ENGINEERING AND TRAINING, LTD.

Introduction to FPGA

Adam Taylor

Xilinx Tools and Frameworks

VitisAI

Enables the implementations of machine learning inference, using Tensor Flow, Caffe and PyTorch. *Requires a SoC/RFSoC/Alveo

Vitis

Embedded and accelerated SW development. Used to develop software solutions for MicroBlaze, Arm R5, A9, A53 and A72.

Petalinux

Embedded Linux solutions

PYNQ

Python framework for rapid prototyping on SoC/RFSoC/Alveo

Vitis / Vivado HLS

High Level Synthesis tool supporting C/C++/OpenCL

Vivado

Design Capture and implementation for the base platform

ADIUVO
ENGINEERING AND TRAINING, LTD.

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

ADIUVO
ENGINEERING AND TRAINING, LTD.

What is an FPGA

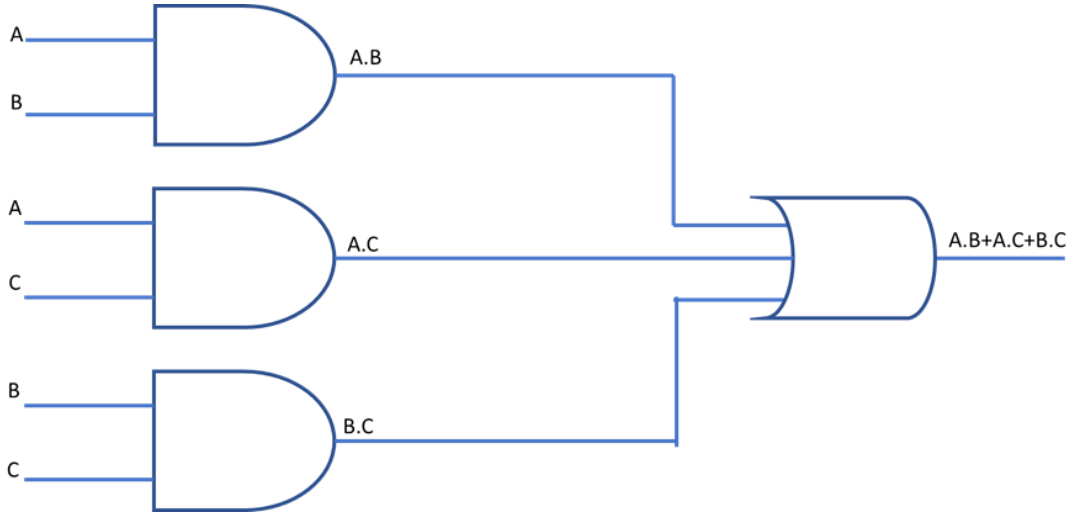
A field-programmable gate array is an integrated circuit designed to be configured by a designer after manufacturing.

Field-Programmable – Means the user can program it in the field

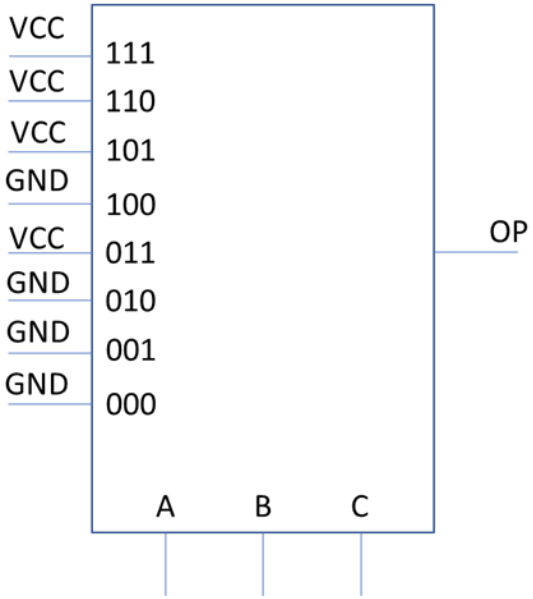
ADIUVO
ENGINEERING AND TRAINING, LTD.

FPGA Architecture

What is programmable logic



Input A	Input B	Input C	OP
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

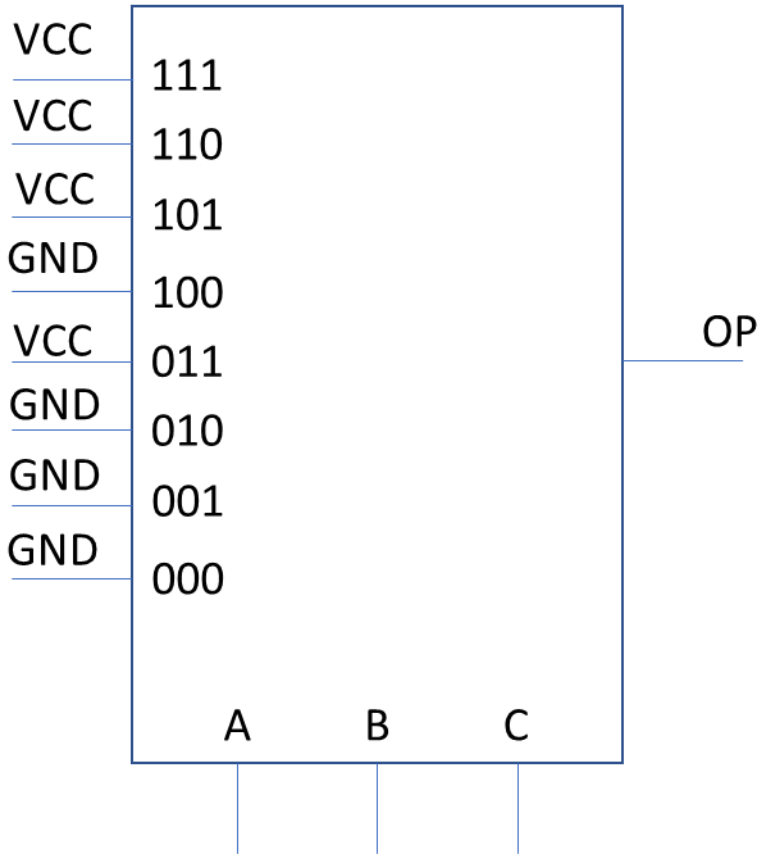


Look Up Table

- The key to programmable logic is how logic equations are implemented in the device.
- It is of course very difficult for the device designers to include a range of AND, OR gates etc. as the number of each gate type will vary for each application
- Device manufacturers addressed this challenge in a very smart manner. In place of discrete gates, they used they use a several input look up table (LUT) which is programmed to implement the combinatorial logic equations.

Look Up Table

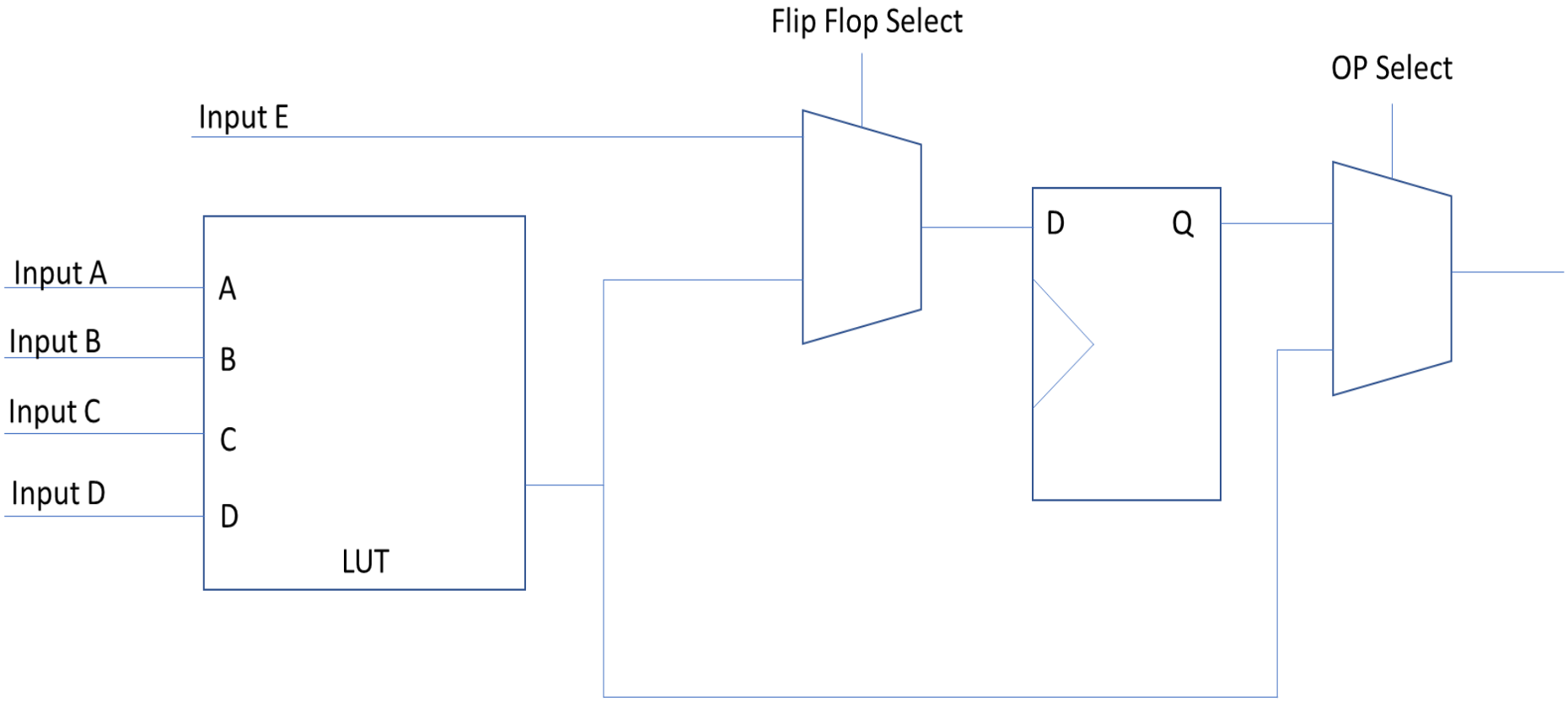
Input A	Input B	Input C	OP
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Configurable Logic Block

- LUT allows implementation of Combinatorial Circuit
- BUT we design synchronous circuits what about the FF
- flip flop to act as storage for the combinatorial output such that we can implement sequential structures
- This combination of a LUT and Flip Flop is often called a Configurable Logic Block (CLB) and a programmable logic device will consist of many thousands of these CLBs. To provide the most flexibility additional multiplexers will be used to support a wide range of CLB configurations.

Configurable Logic Block

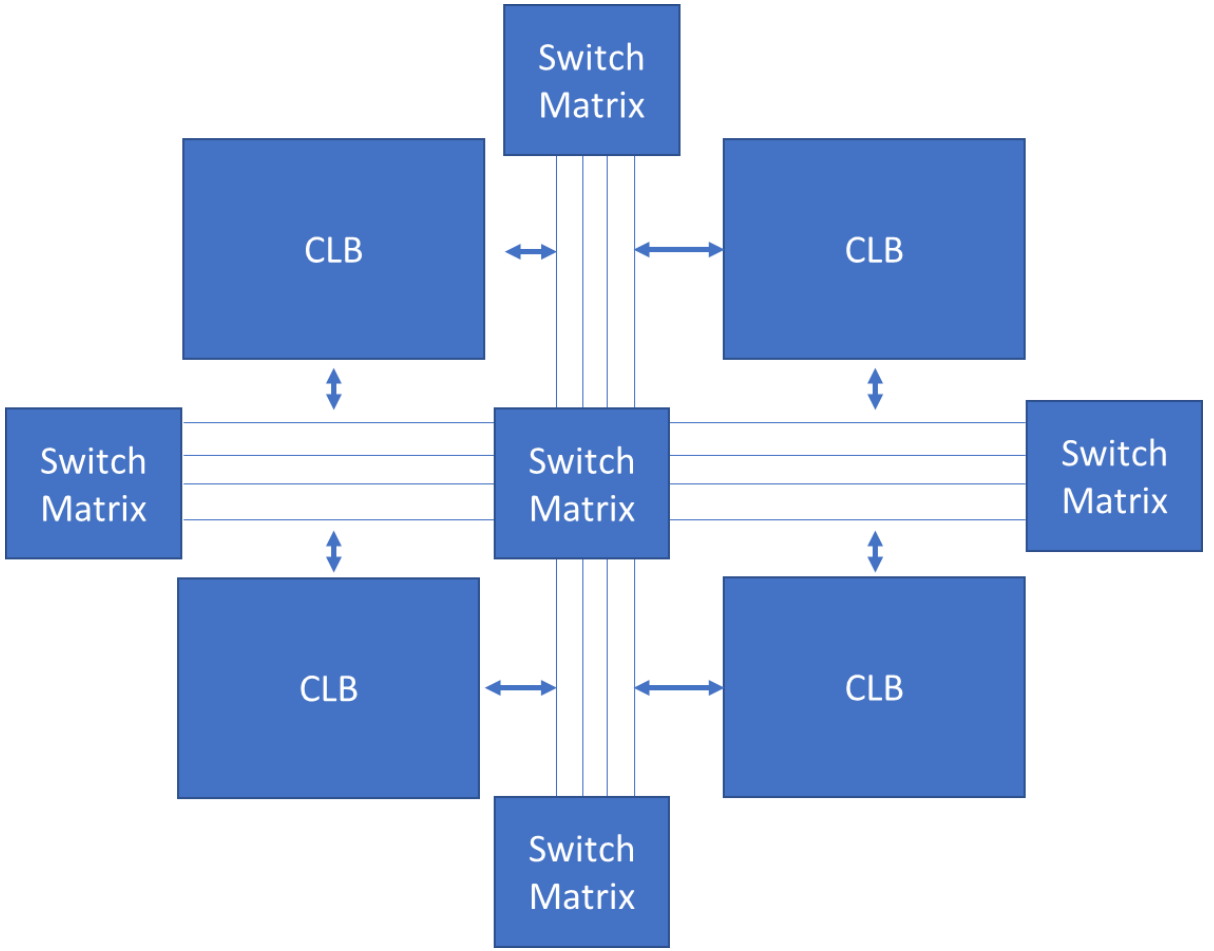


Interconnect Routing

A collection of wires and programmable switches.

These are responsible for connecting CLBs and other building blocks within the FPGA.

These are also called routing channels.



ADIUVO
ENGINEERING AND TRAINING, LTD.

Tools

- Requirements Capture tool
- Editors – Ideal with ability to LINT and check structural issues e.g HDL Creator
- Synthesis Tool – Third Party or Vendor Supplied
- Implementation – Vendor supplied – some open source for lattice
- Simulation – Third Party or Vendor
- Source Control – Subversion, GIT
- Configuration control – often called PLM tool
- Specialist e.g. Fault Injection

Vivado Overview

- Foundation of all design and higher-level tools is Vivado
- Vivado enables us to capture designs using VHDL or Verilog
- Large IP library to accelerate our designs
- Integrates Vivado / Vitis HLS IP cores

- ✓ Simulate designs using Vivado Simulator
- ✓ Synthesize, place and route the design
- ✓ Generate power estimations
- ✓ Create Xilinx Support Architecture

Vivado Overview

FPGA Implementation Flow

Synthesis – Translates the HDL design into a series of logic equations which are then mapped onto the resources available in the target FPGA.

Synthesis – Place – The logic resources determined by the synthesis tool are placed at available locations within the target device.

Routing – The placed logic resources in the design are interconnected using routing and switch matrixes to implement the final application.

Bit File – The generation of the final programming file for the target FPGA.

We can control the flow implementation settings by using Constraints (XDC file) and implementation Strategies

Xilinx Devices

ADIUVO
ENGINEERING AND TRAINING, LTD.

Xilinx Offer a range of devices

FPGA – Spartan, Artix, Kintex, Virtex – Seven / UltraScale / UltraScale+

Heterogeneous SoC – Zynq, Zynq MPSoC, RFSoC

Adaptive Compute Acceleration Platform– Versal

Accelerator Card - Alveo

FPGA

Heterogeneous SOC

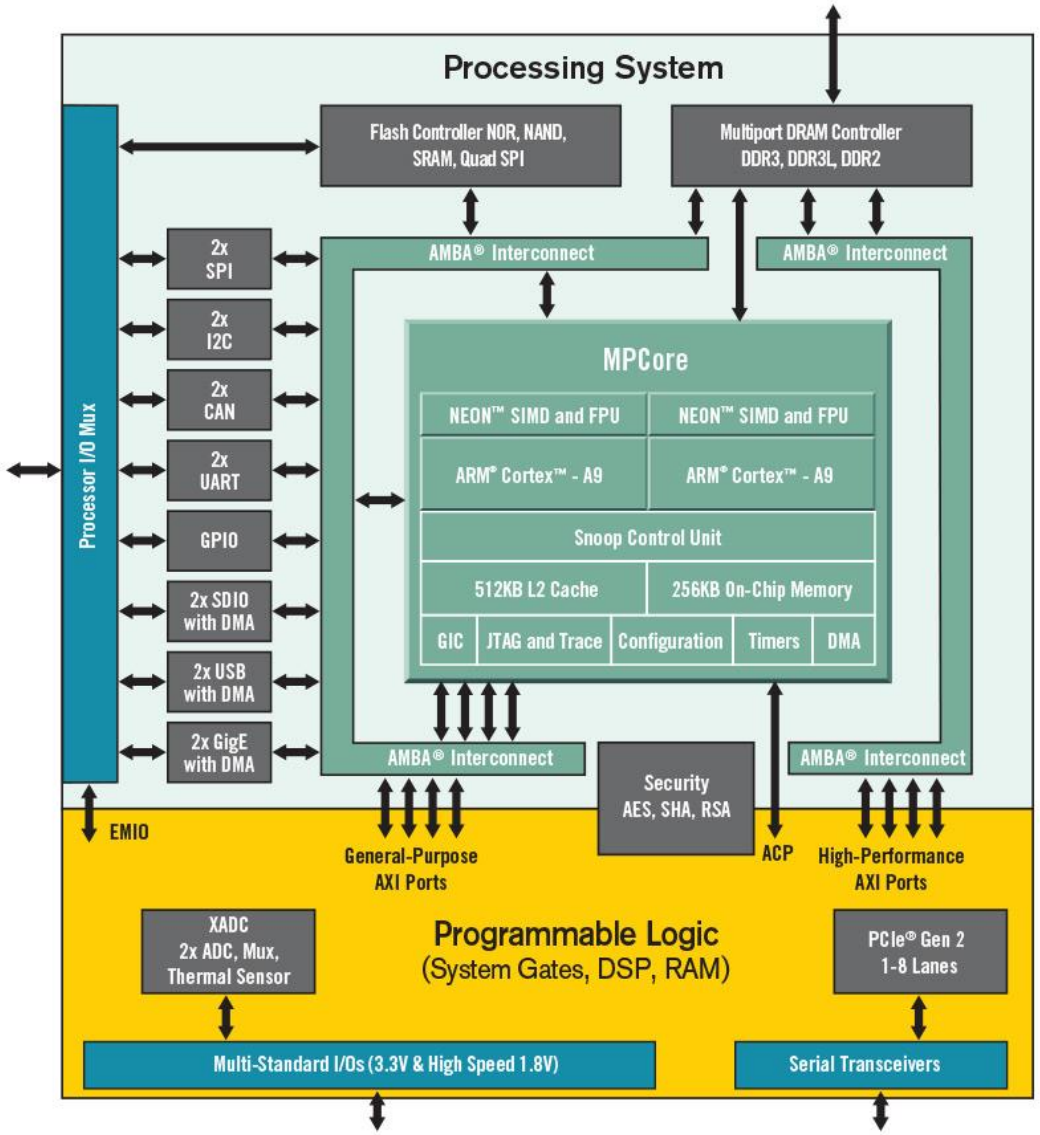
Combine diverse processing elements with programmable logic

Processing elements are silicon implementations

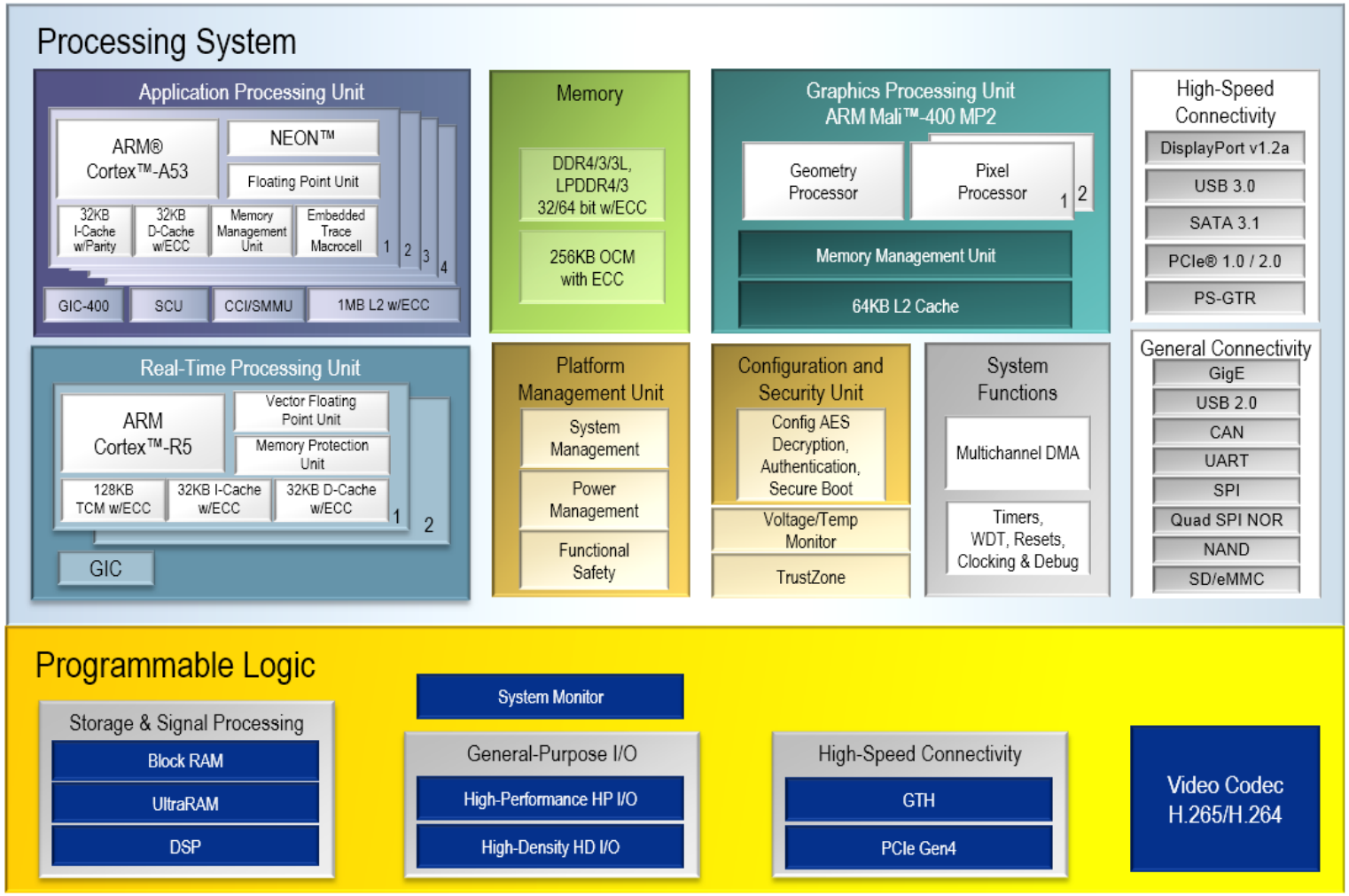
Processing element is the master – it boots just like any other processor

Enables highly optimized solutions be implemented using processor and logic to exploit natural strengths

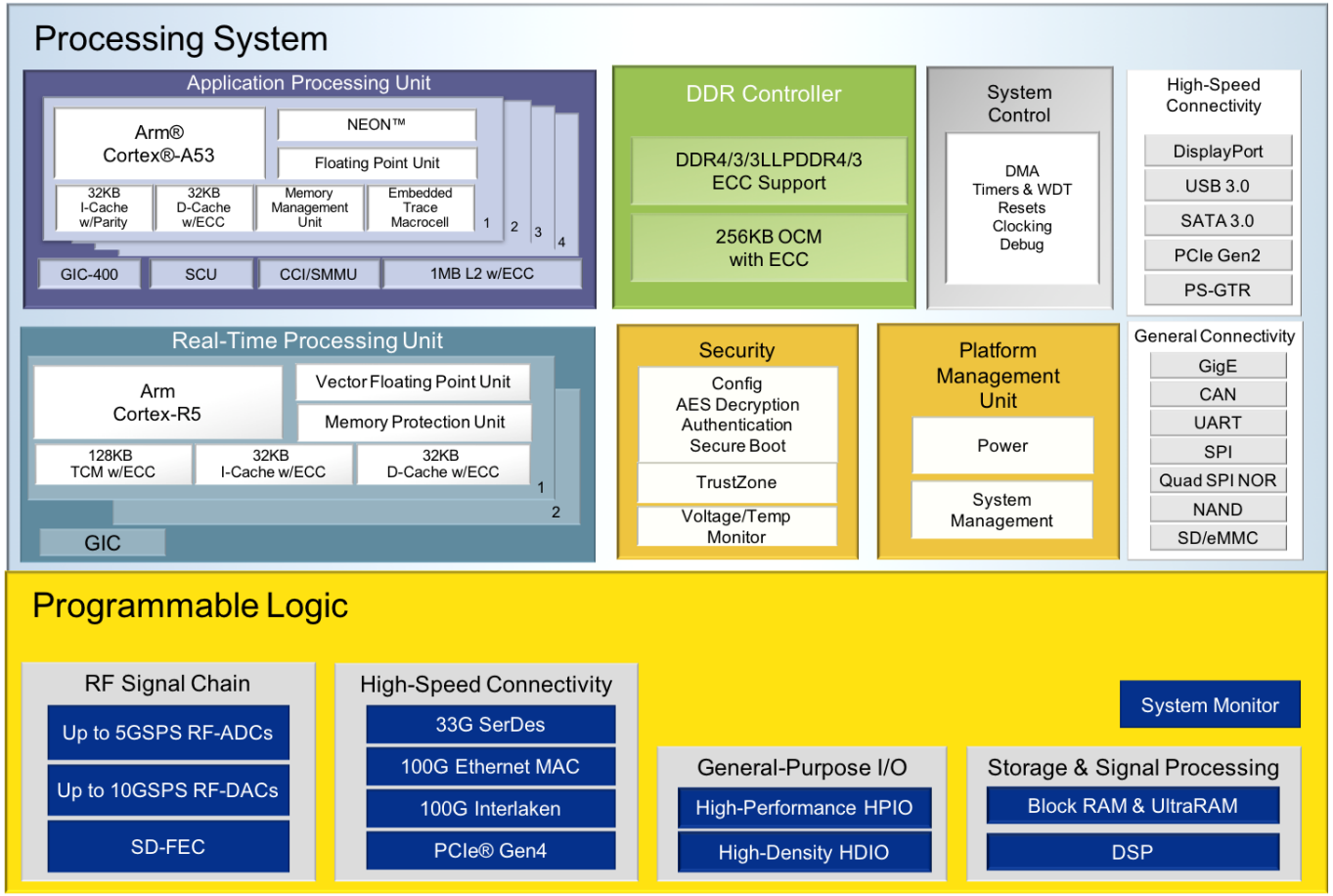
Zynq



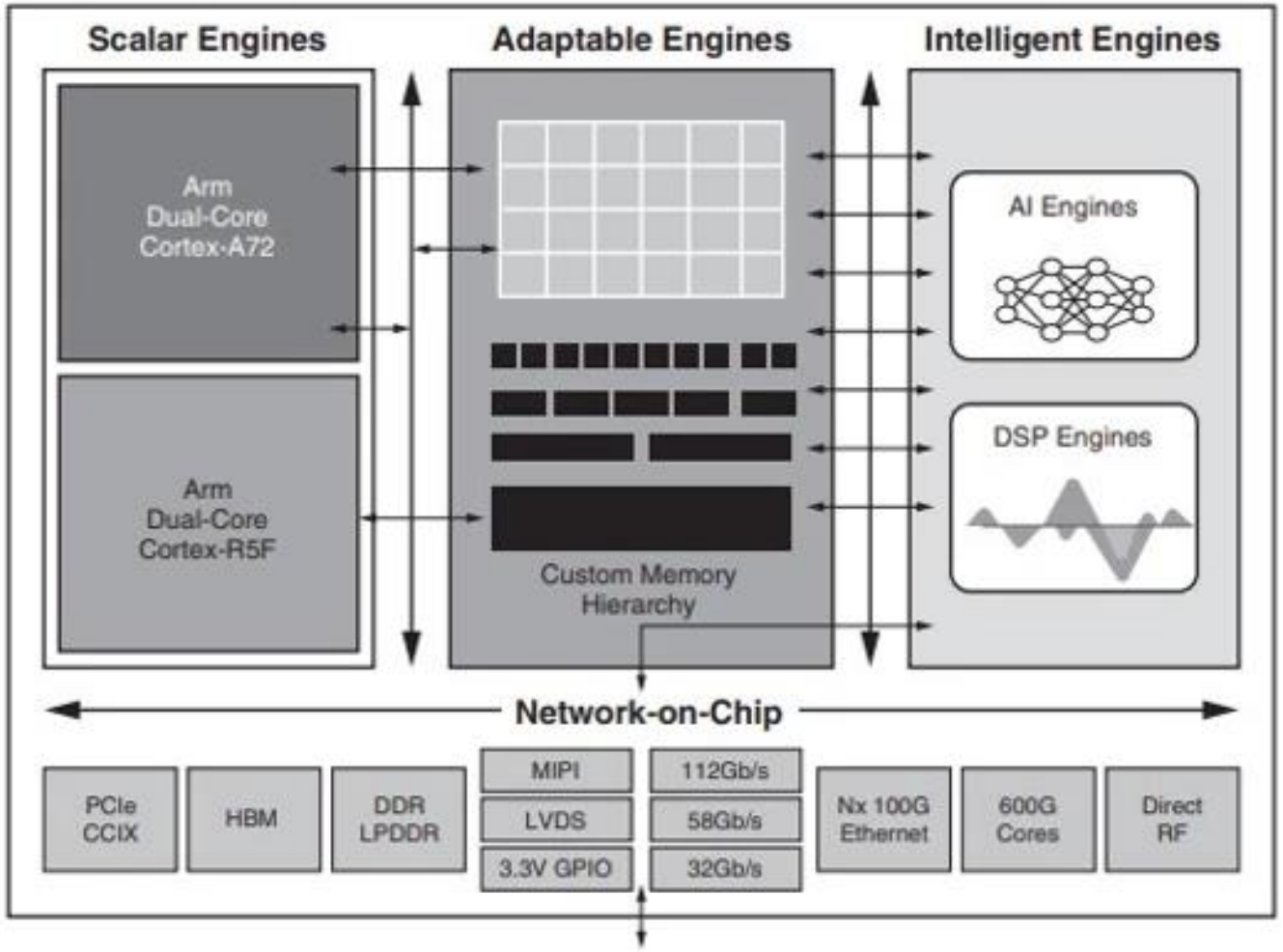
Zynq MPSoC



Zynq RFSoC

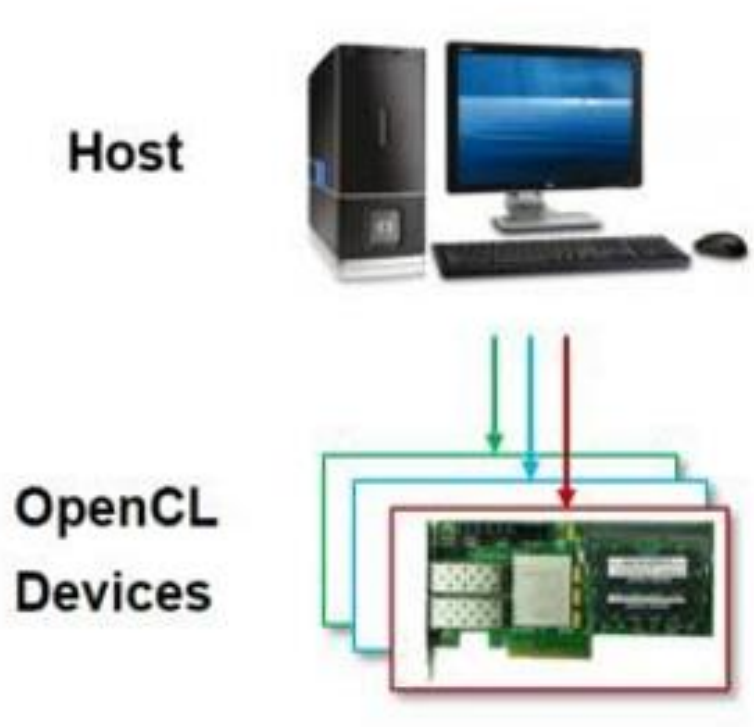


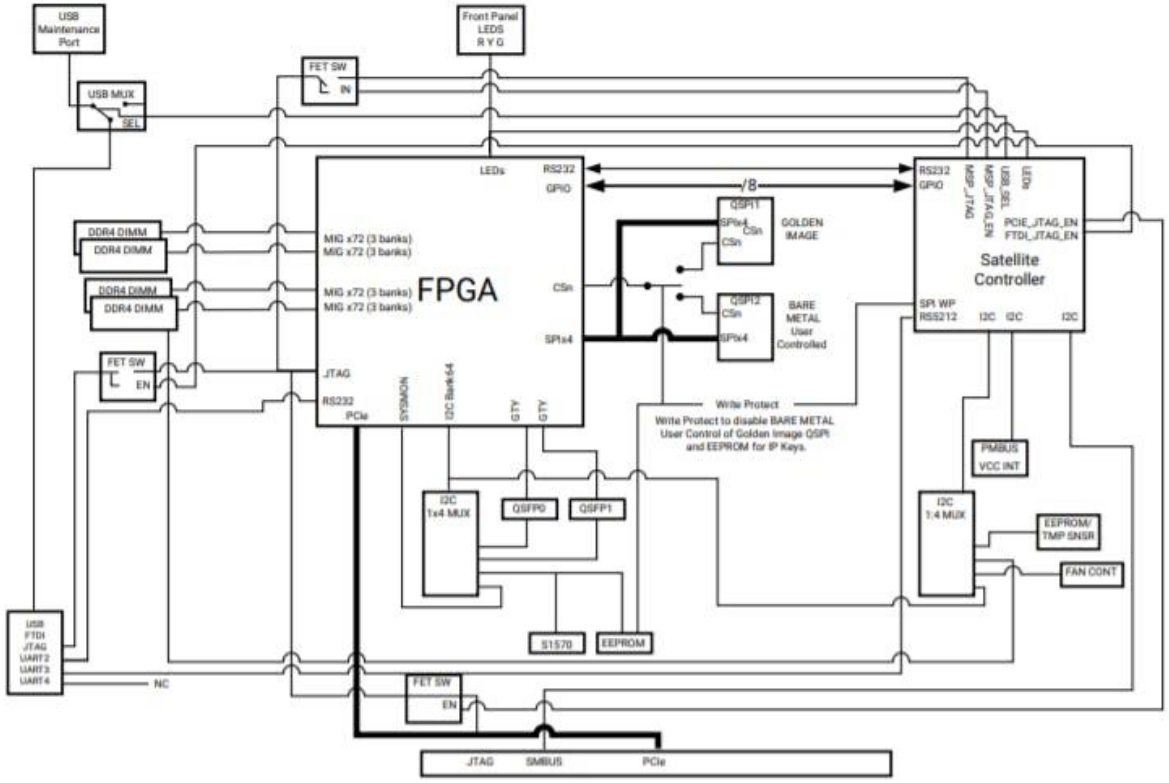
Versal



Alveo

Accelerator card based around OpenCL





Clocking

ADIUVO
ENGINEERING AND TRAINING, LTD.

Clocking

FPGA Designs are Synchronous!

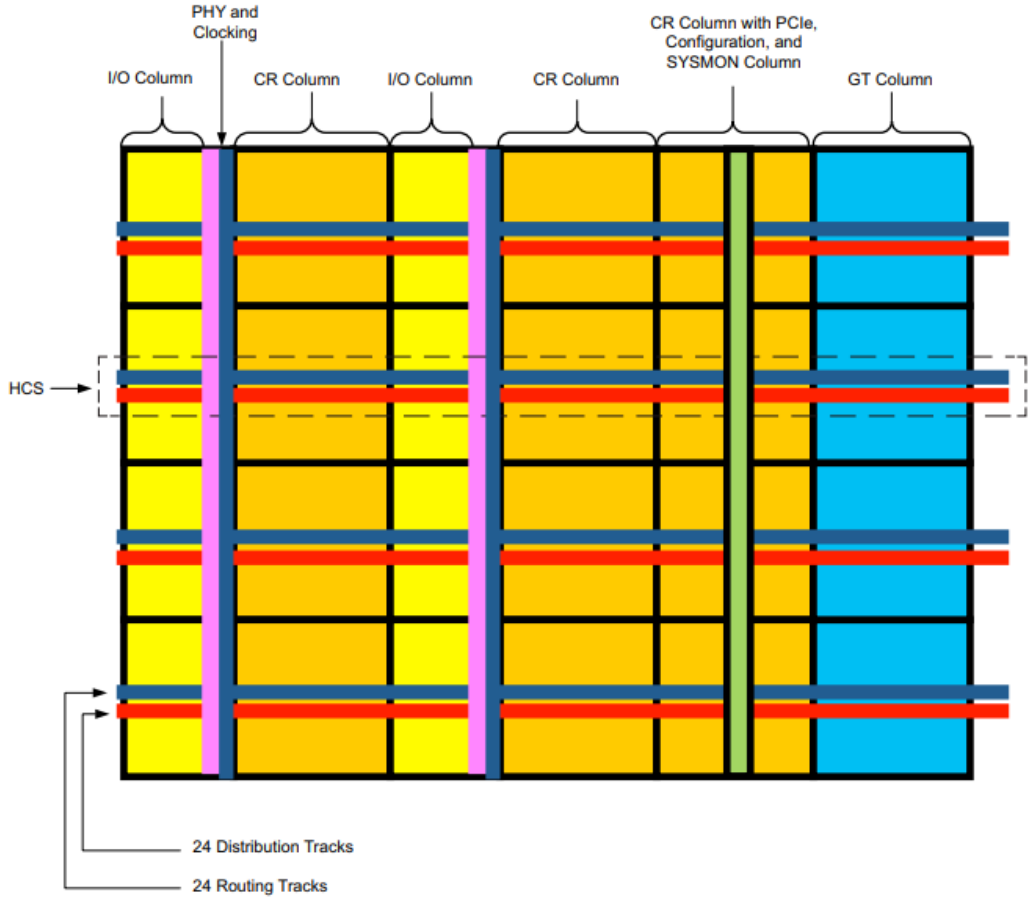
Clocks are high fan out, as such dedicated pins are required to be used.

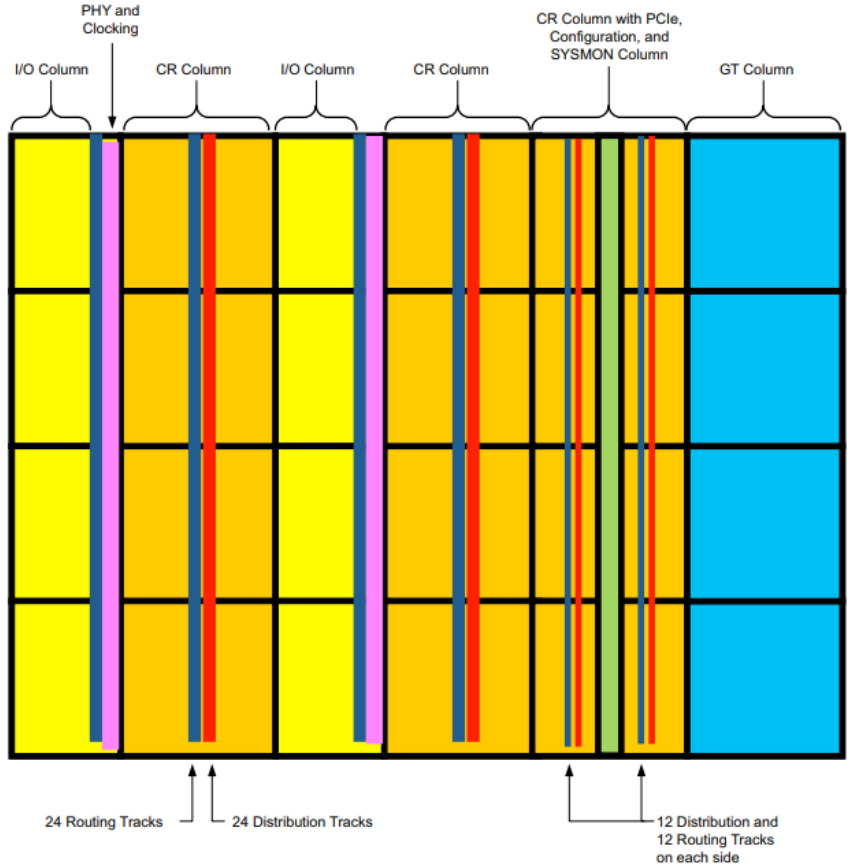
FPGA typically will have multiple clock regions

Clock pins are places in IO banks

- UltraScale / UltraScale+ GC pins or global clock pins
- Seven Series CC and GC pins – GC global clocks CC restricted to close CR
- Special Clock pins e.g. Byte-Lane Clocks (DBC and QBC) typical in memory applications

Vertical and Horizontal Clocks



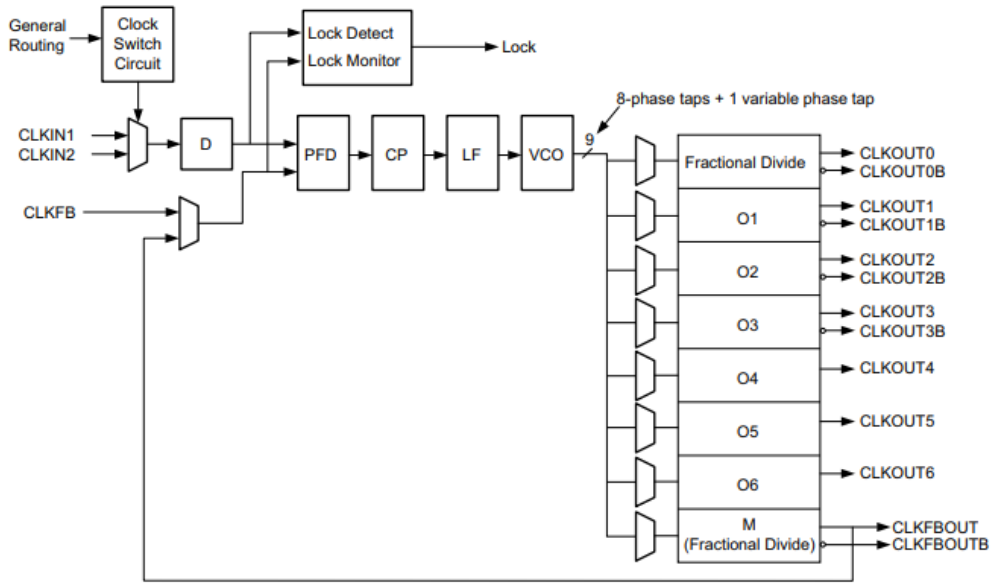


Clocking Resources

FPGA have a range of clock resources to simply solutions

1. Clock Buffers e.g. bufgctl

2. Clock Management Tiles
   - » Mixed Mode Clock Manager (MMCM) & 2 Phase Locked Loops



Clocking Resources

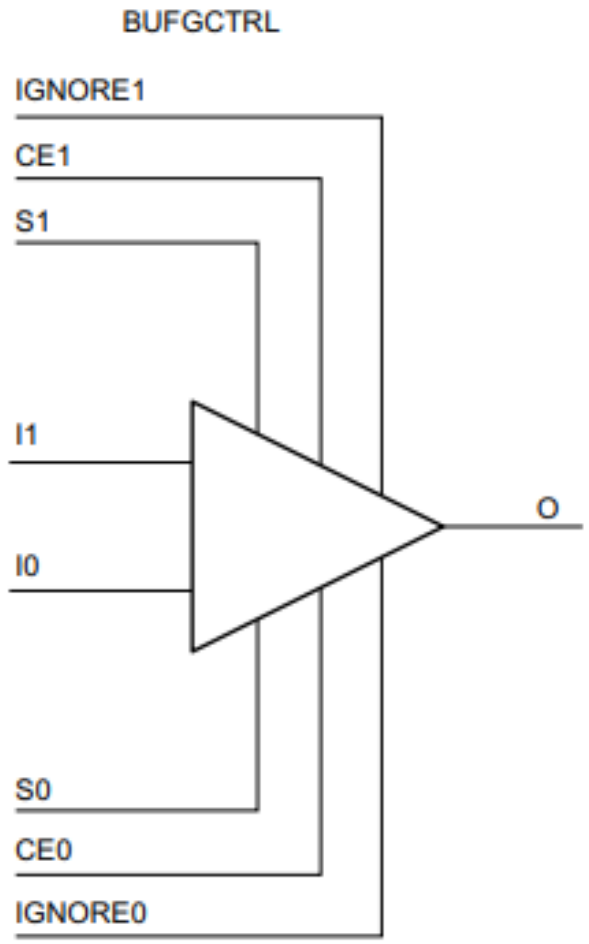
Bufgctl - drive the routing and distribution resources across the entire device.

Can switch between 2 clocks seamlessly.

Basis of many structures including

Bufgce_1, Bufgmux etc.

Read Seven Series, UltraScale clocking guides



What is a Clock Domain?

Same clock domain if:

- Same source and integer multiple frequencies

Different clock domain if:

- Different source
  - **Even if frequency specification is the same!**
  - **All specifications have error bars**
- Not integer multiple frequencies, even if same source

By default, Vivado assumes all clocks are in the same domain – Unless you tell it otherwise!

Not telling it clock relationships can make a significantly longer implementation times as Vivado tries to close timing which is impossible

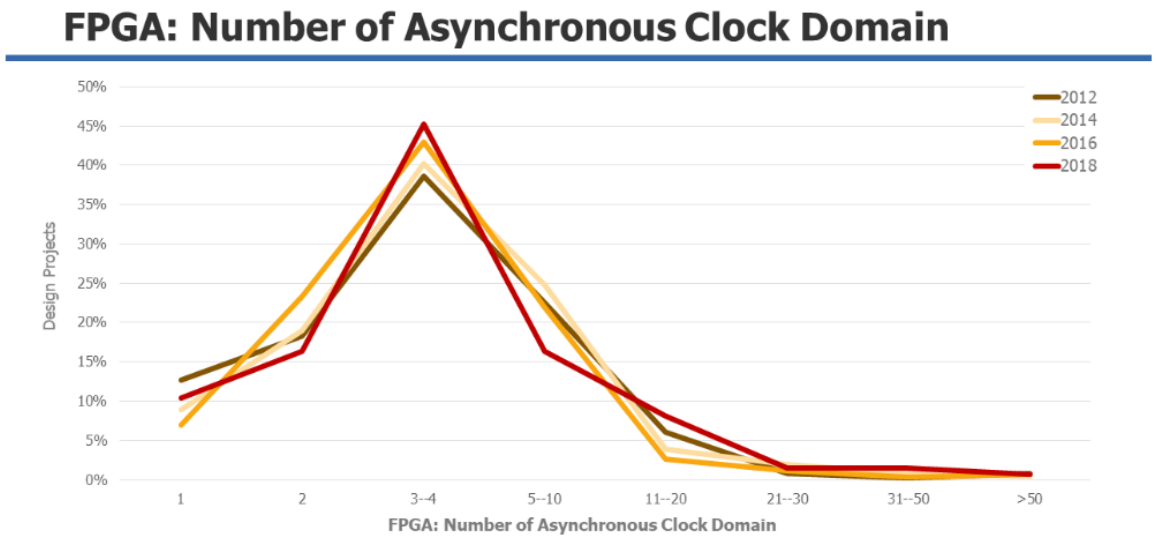
Clock Domain Crossing

ADIUVO
ENGINEERING AND TRAINING, LTD.

Clock Domain Crossing

Ideal solution uses one clock and the entire design is synchronous.

BUT!



Modern devices have multiple clocks to address different clock domains e.g. ADC / DAC clocks, source synchronous interfaces.

Brings with it the need to transfer data, and signals safely and reliability between the clock domains

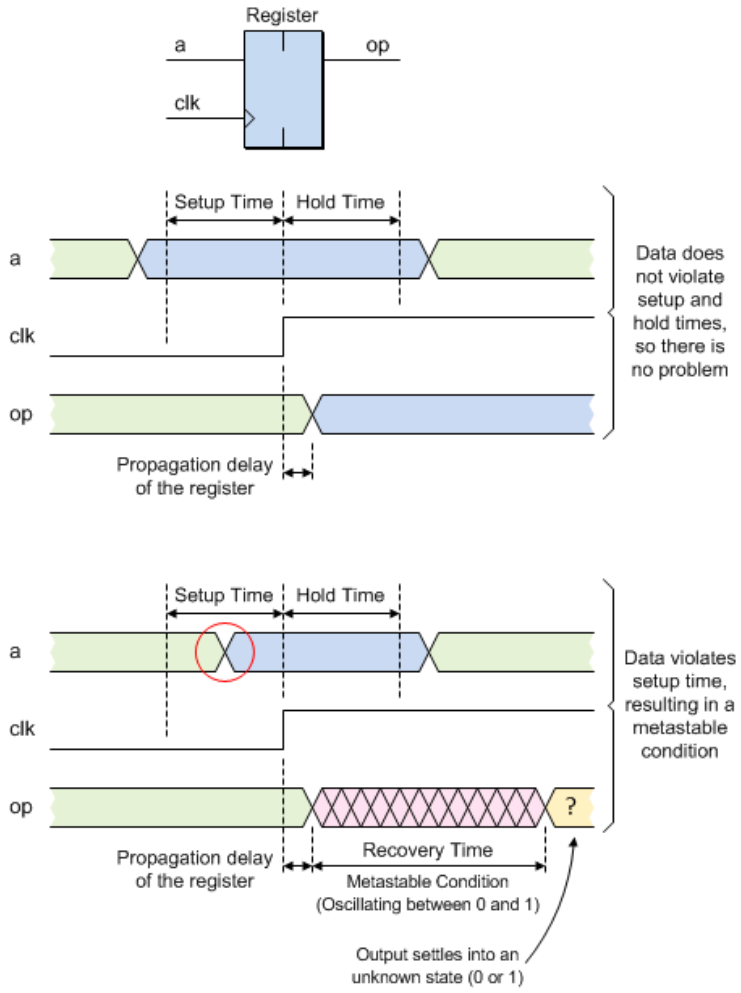
Metastability

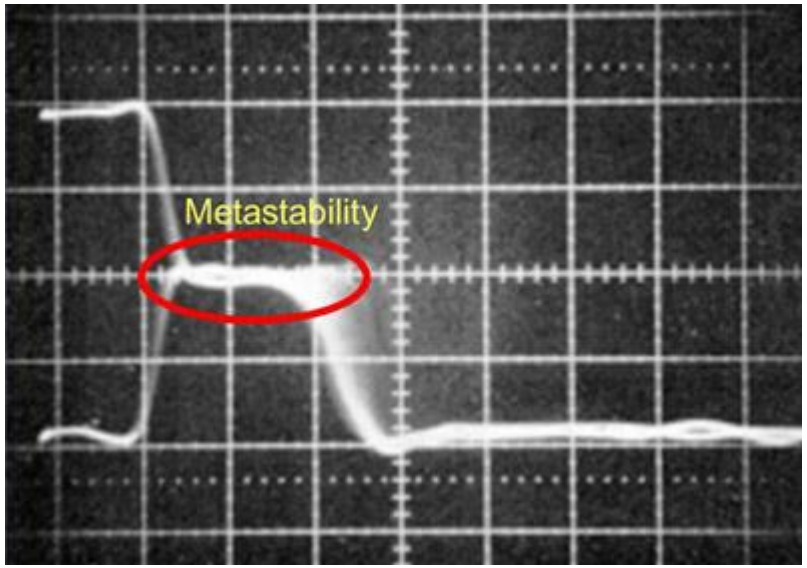
One issue which can arise with incorrect domain crossing is metastability

This can lead to corruption of data or incorrect behaviour

Occurs when a flip flops set up or hold time is violated

Metastability





Clock Domain Crossing

Several Techniques which can be used depending upon what needs to be transferred

- Two stage synchroniser – Ideal for single bit data
- Grey Code Synchroniser – Encodes data bus in grey code and transfer between domains – Ideal for counters as input to be converted to grey code can only decrement / increment by one from previous value
- Hand shake synchroniser – Transfers data bus between two clock domains using handshake signals
- Pulse synchroniser – transfer pulse from one clock domain to another
- Asynchronous FIFO – transfers data from one domain to another, useful for high throughput / burst transfers

Clock Domain Crossing

To support reset functionality across clock domains we may need the following synchronisers structures.

- Asynchronous Reset Synchroniser – enables asynchronous assertion and synchronous de-assertion.
- Synchronous Reset Synchroniser – synchronises a synchronous reset to another clock domain.

How many synchronisers do I need ?

Standard two stage synchroniser assumes the first flip flop, will recover from metastability before the signal is clocked into the second flip flop.

How do we know this is the case and what if we need more stages

$$MTBF = \frac{e^{t_r/\tau}}{T_o \times f \times a} \times k\left(\frac{e^{t_r/\tau}}{T_o \times f}\right)$$

t_r = The speed at which the event is being resolved (clock period – setup time)

T_o = Time window during which the register is susceptible to going metastable.

τ = Settling time

a = Frequency of the asynchronous signal.

f = Frequency of the clock (as stated in Part 1, this should be faster than "a")

k = The number of additional stages (so k=1 for a two-stage synchroniser)

Note that τ and T_o are both process constants.

CDC in Xilinx

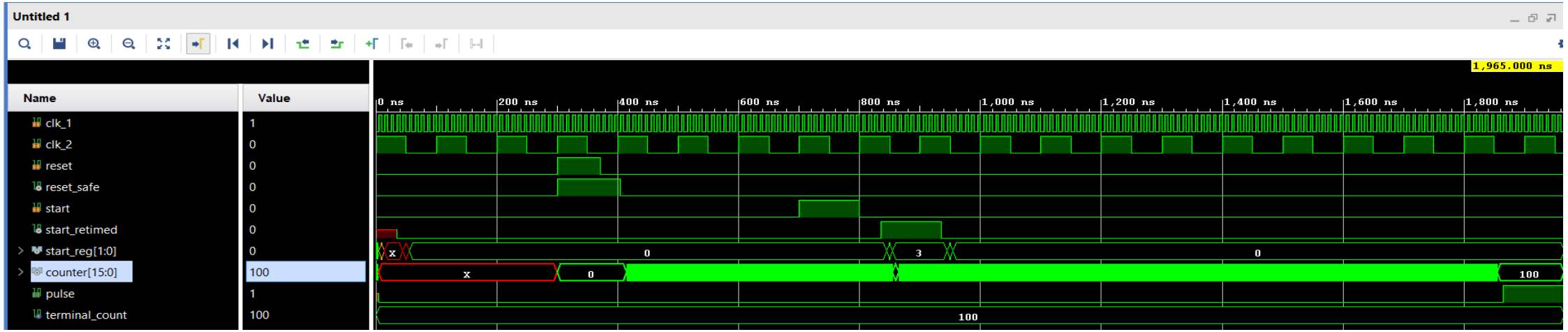
Xilinx Parameterised Macros (XPM) – provide CDC structures

Use registers optimised for CDC in the fabric

- » Registers located close together and have small set up and hold windows

Described within UG953 Libraries Guide

Described within UG 974 UltraScale Architecture Libraries



CDC Design Analysis tools

Detecting all CDC issues can be a challenge in large designs

- Are all IP IO on the correct clock domain
- Very easy to associate signal with wrong domain e.g. FIFO empty and WR clock

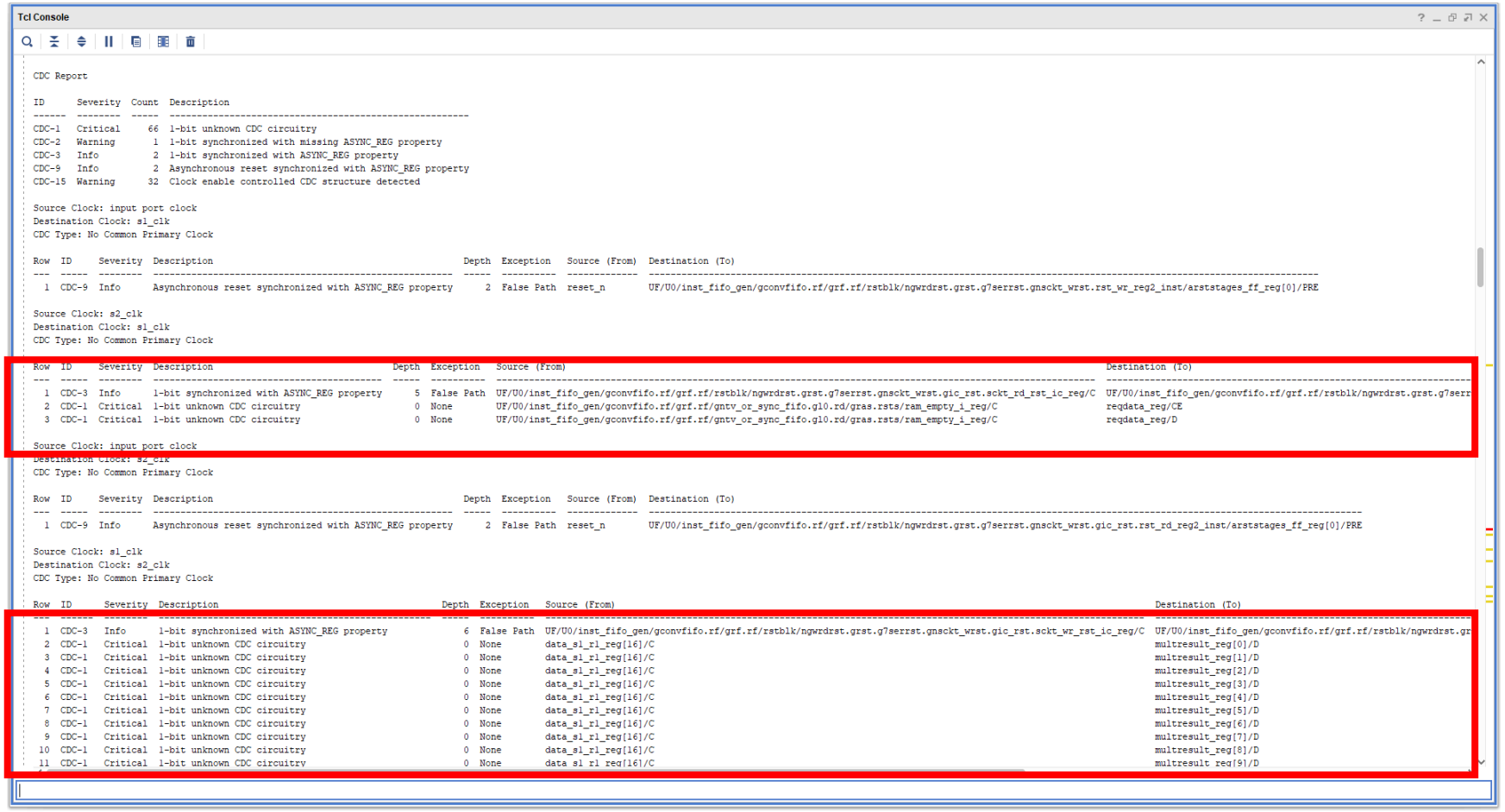
CDC issues can be very difficult to find changing on each start up and may be intermittent

Can be hard to find in simulation – Timing simulation required, takes a long time simulate the system

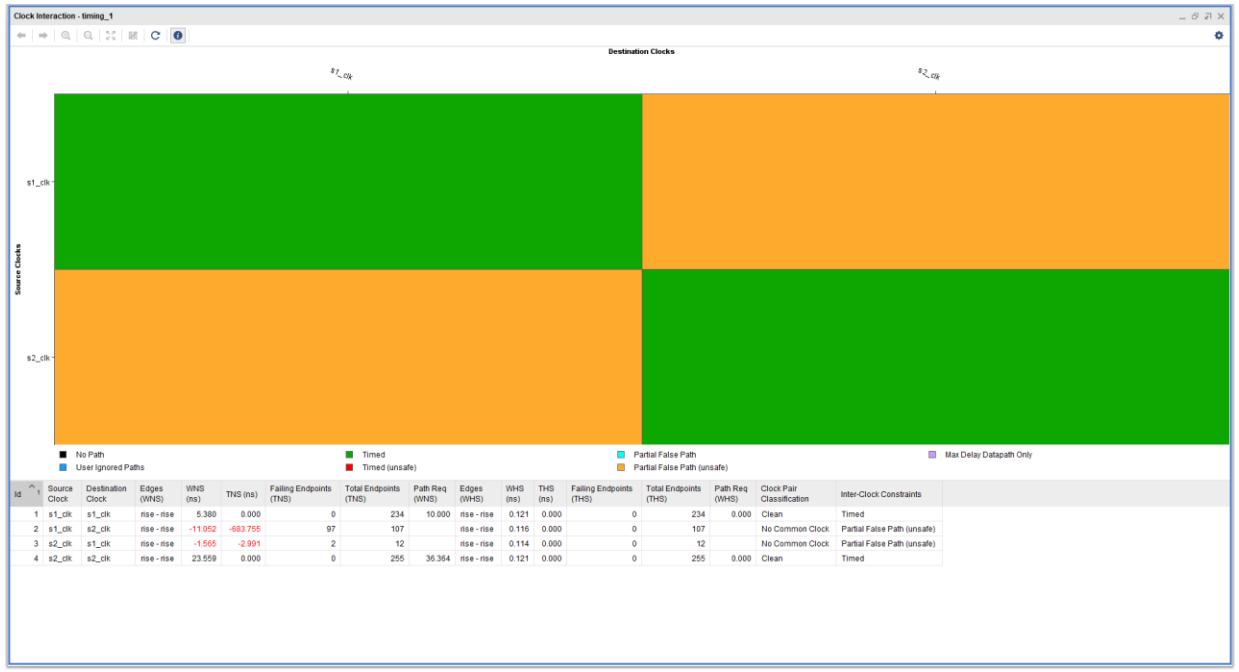
Static analysis tools are better suited to find the CDC issues.

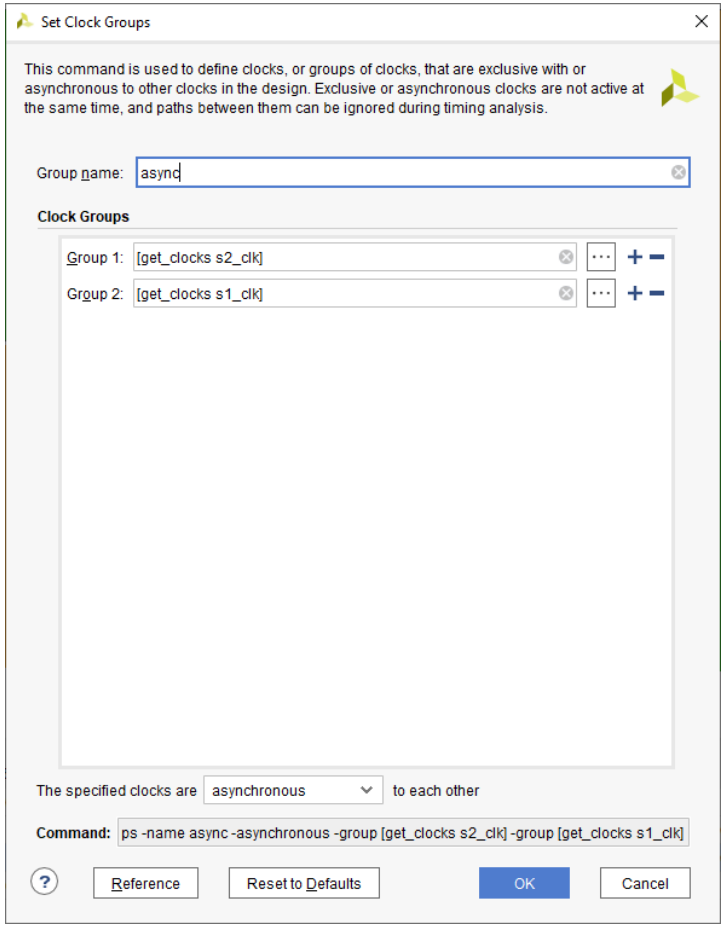
Vivado CDC Report

Following Synthesis – in TCL window run the command report_cdc



Vivado Timing Analysis – Clock Interaction



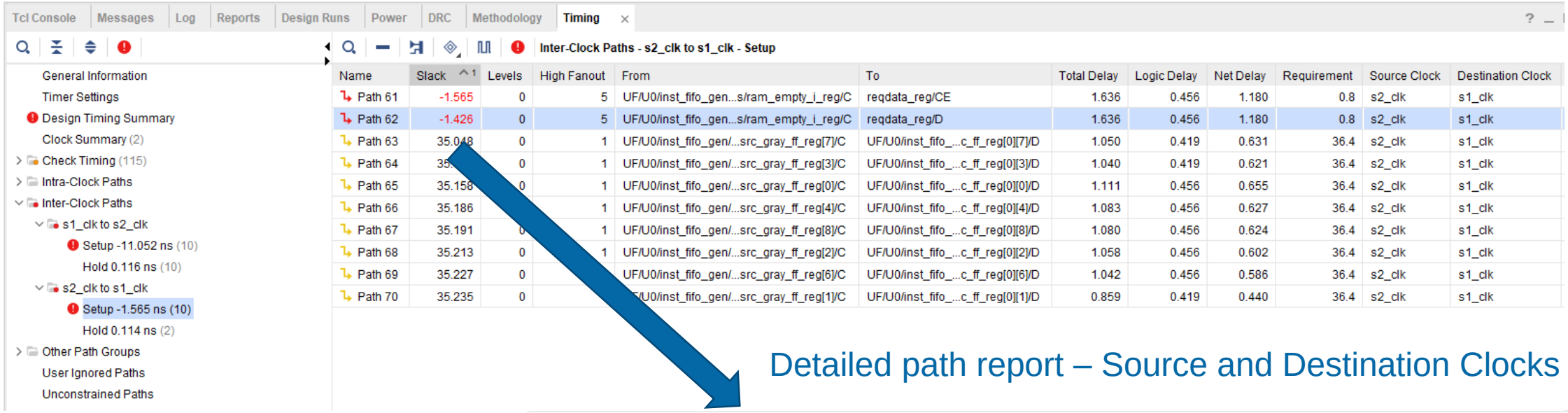


Generated From Flow Navigator when implementation is open

Yellow show unrelated clock – Indicates CDC issues as well

Means we need to define constraints

Vivado Timing Analysis – Inter clock Issues



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 61	-1.565	0	5	UF/U0/inst_fifo_gen...s/ram_empty_i_reg/C	reqdata_reg/CE	1.636	0.456	1.180	0.8	s2_clk	s1_clk
Path 62	-1.426	0	5	UF/U0/inst_fifo_gen...s/ram_empty_i_reg/C	reqdata_reg/D	1.636	0.456	1.180	0.8	s2_clk	s1_clk
Path 63	35.048	0	1	UF/U0/inst_fifo_gen/...src_gray_ff_reg[7]/C	UF/U0/inst_fifo_...c_ff_reg[0][7]/D	1.050	0.419	0.631	36.4	s2_clk	s1_clk
Path 64	35.	0	1	UF/U0/inst_fifo_gen/...src_gray_ff_reg[3]/C	UF/U0/inst_fifo_...c_ff_reg[0][3]/D	1.040	0.419	0.621	36.4	s2_clk	s1_clk
Path 65	35.158	0	1	UF/U0/inst_fifo_gen/...src_gray_ff_reg[0]/C	UF/U0/inst_fifo_...c_ff_reg[0][0]/D	1.111	0.456	0.655	36.4	s2_clk	s1_clk
Path 66	35.186	0	1	UF/U0/inst_fifo_gen/...src_gray_ff_reg[4]/C	UF/U0/inst_fifo_...c_ff_reg[0][4]/D	1.083	0.456	0.627	36.4	s2_clk	s1_clk
Path 67	35.191	0	1	UF/U0/inst_fifo_gen/...src_gray_ff_reg[8]/C	UF/U0/inst_fifo_...c_ff_reg[0][8]/D	1.080	0.456	0.624	36.4	s2_clk	s1_clk
Path 68	35.213	0	1	UF/U0/inst_fifo_gen/...src_gray_ff_reg[2]/C	UF/U0/inst_fifo_...c_ff_reg[0][2]/D	1.058	0.456	0.602	36.4	s2_clk	s1_clk
Path 69	35.227	0		UF/U0/inst_fifo_gen/...src_gray_ff_reg[6]/C	UF/U0/inst_fifo_...c_ff_reg[0][6]/D	1.042	0.456	0.586	36.4	s2_clk	s1_clk
Path 70	35.235	0		/U0/inst_fifo_gen/...src_gray_ff_reg[1]/C	UF/U0/inst_fifo_...c_ff_reg[0][1]/D	0.859	0.419	0.440	36.4	s2_clk	s1_clk

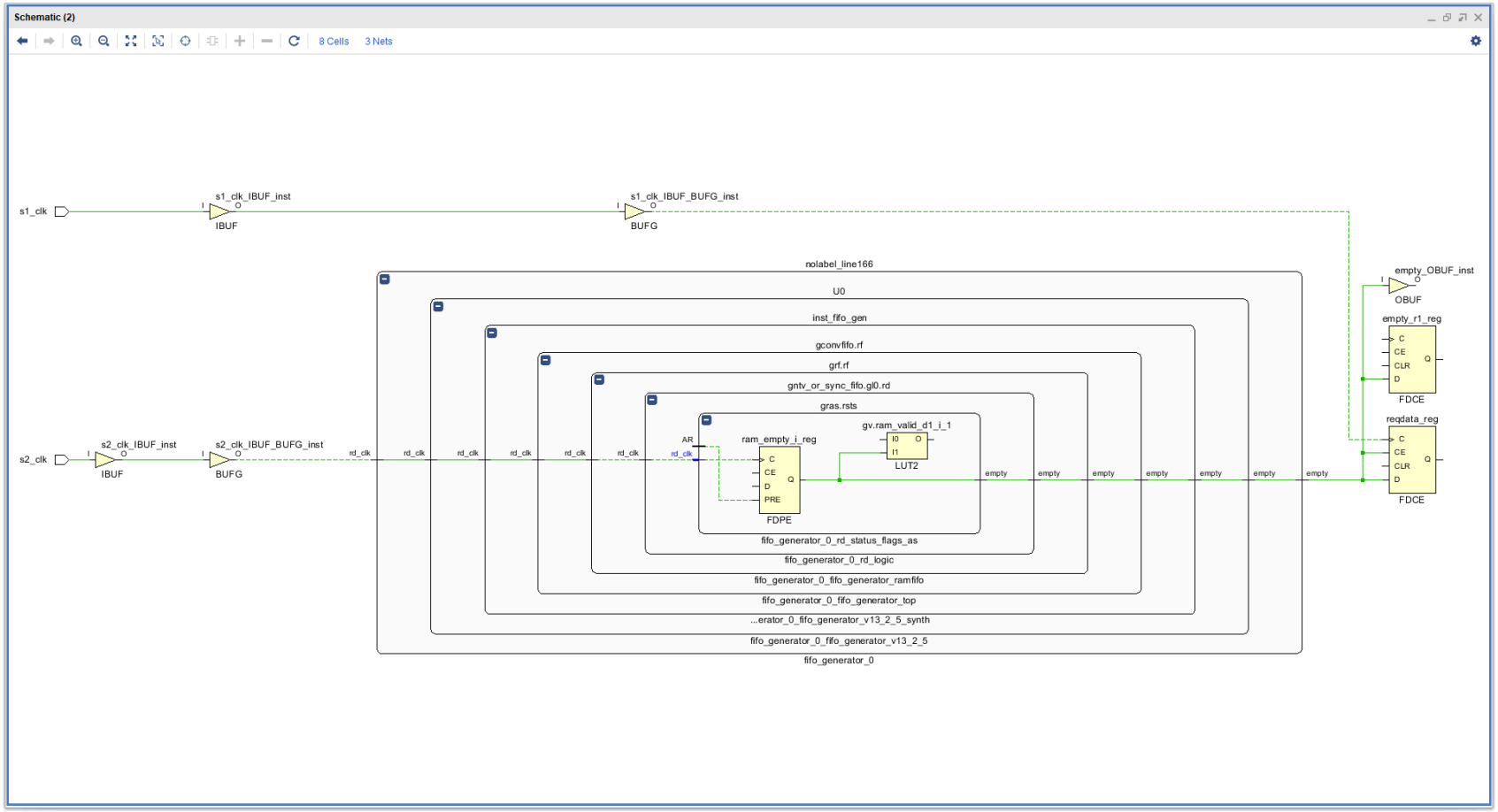
Detailed path report – Source and Destination Clocks different

Summary	
Name	Path 62
Slack	-1.426ns
Source	UF/U0/inst_fifo_gen/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.gl0.rd/gras.rsts/ram_empty_i_reg/C (rising edge-triggered cell FDPE clocked by s2_clk {rise@0.000ns fall@18.182ns period=36.364ns})
Destination	reqdata_reg/D (rising edge-triggered cell FDCE clocked by s1_clk {rise@0.000ns fall@5.000ns period=10.000ns})
Path Group	s1_clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	0.812ns (s1_clk rise@9710.000ns - s2_clk rise@9709.188ns)
Data Path Delay	1.636ns (logic 0.456ns (27.876%) route 1.180ns (72.124%))
Logic Levels	0
Clock Path Skew	-0.537ns
Clock Uncertainty	0.035ns
Clock Dom...Crossing	Inter clock paths are considered valid unless explicitly excluded by timing constraints such as set_clock_groups or set_false_path.

If you forget to report_cdc following synthesis. CDC issues will be apparent in the timing report if we have not correctly addressed the constraints

Vivado Flow - Schematic

Engineer can use information provided in the text report to navigate to the schematic to understand where CDC might be.



Block RAM

ADIUVO
ENGINEERING AND TRAINING, LTD.

BRAM

Used for storage of the Data in the FPGA, more efficient than using registers.

BRAM can be used as Single / Dual Port – Great for clock conversion

BRAM can be used to implement FIFOs – Great for Clock Domain Crossing

What might we store in the BRAMs

- Filter Coefficients
- Image Lines
- Signal Data
- System Configuration Data

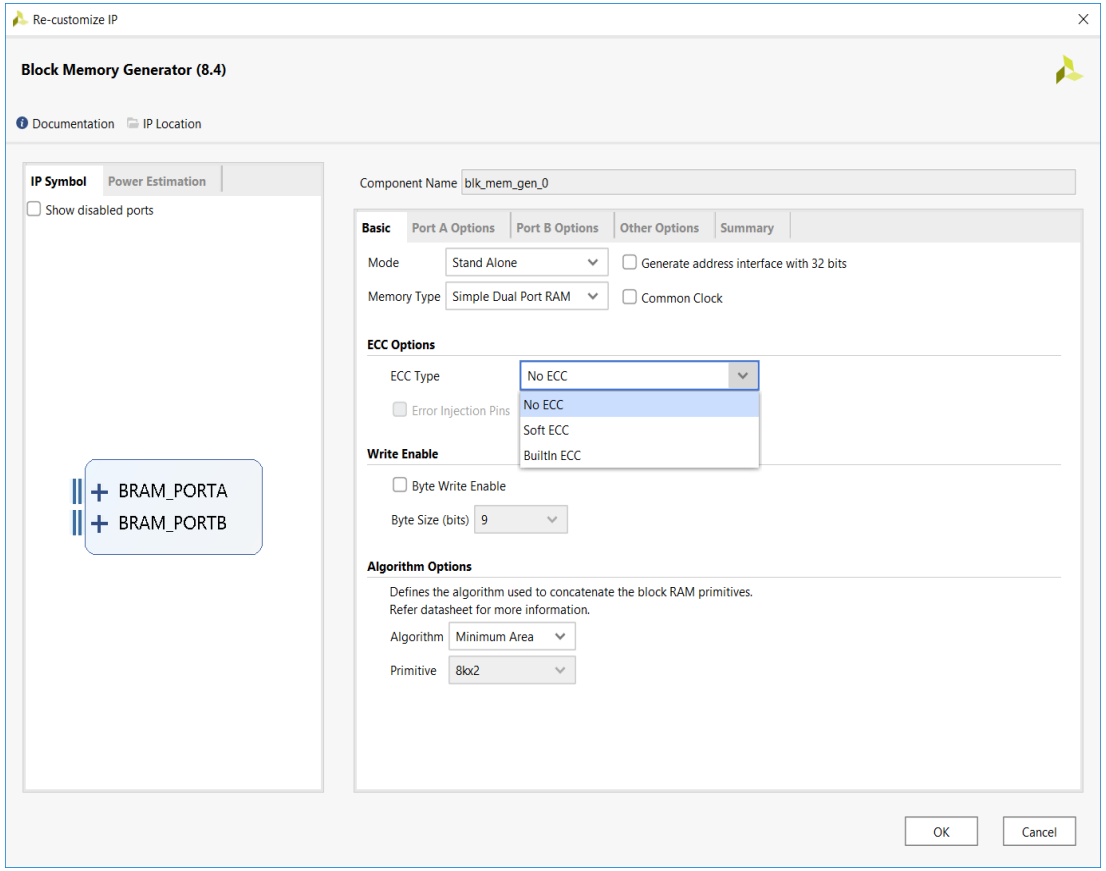
Block RAM

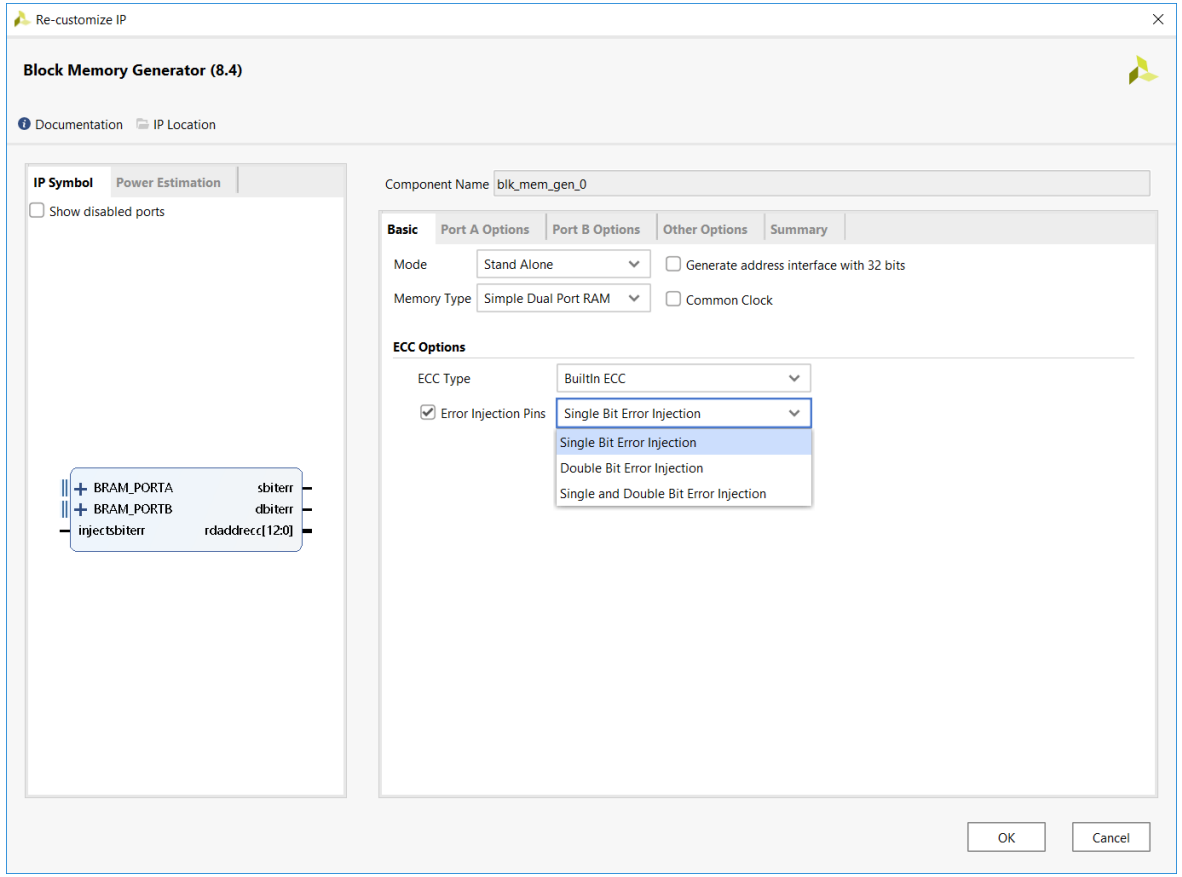
- RAMS can be protected by error protection codes
- Depending upon the device these are hard coded into the RAMS and transparent to the user or require user implementation
- Regardless of how they are implemented it is a good idea to use a scrubbing algorithm which will read back contents out of the memory periodically and ensure correction of any errors to prevent a build up of errors
- If these RAMS are being used to store the configuration of the system it is a good idea to have a copy of this configuration on the ground payload control system to ease recovery in the worse case

BRAM in Xilinx

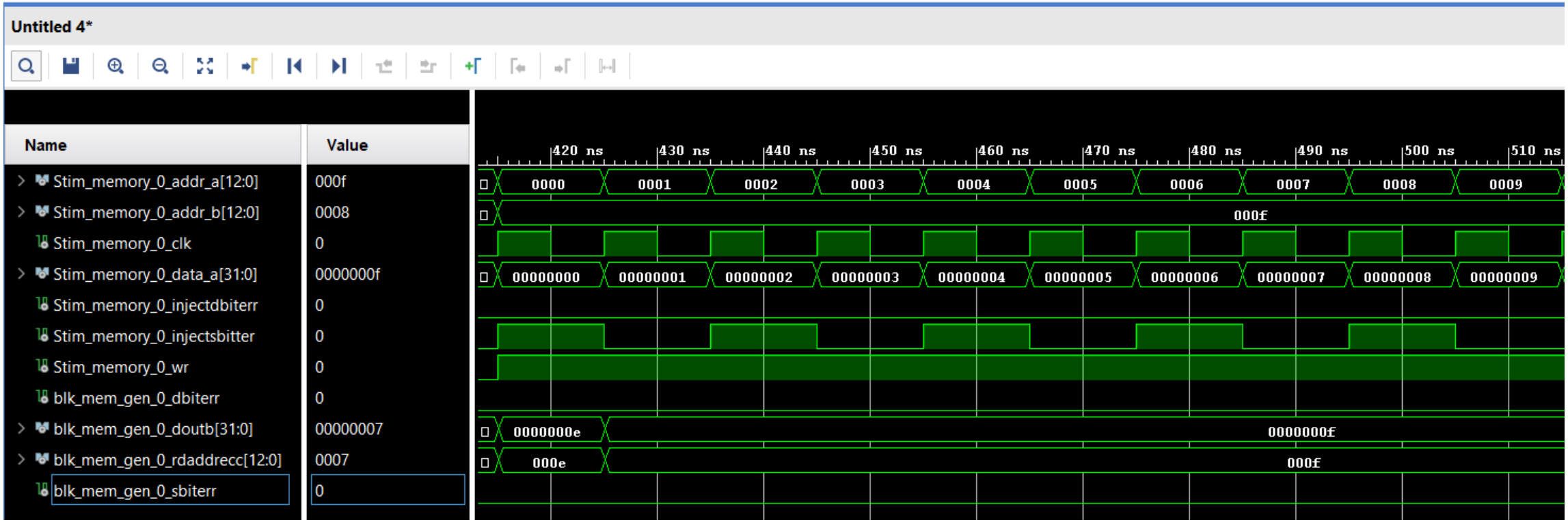
- Inherent support for ECC
- Cannot initialise BRAM with a COE file it ECC is used
- BRAM 64 bit data width and greater Hard Hamming Implementation
- BRAM less than 64 bit data width soft hamming implementation
- Can optimise for performance / power

ECC In Block Memory Generator

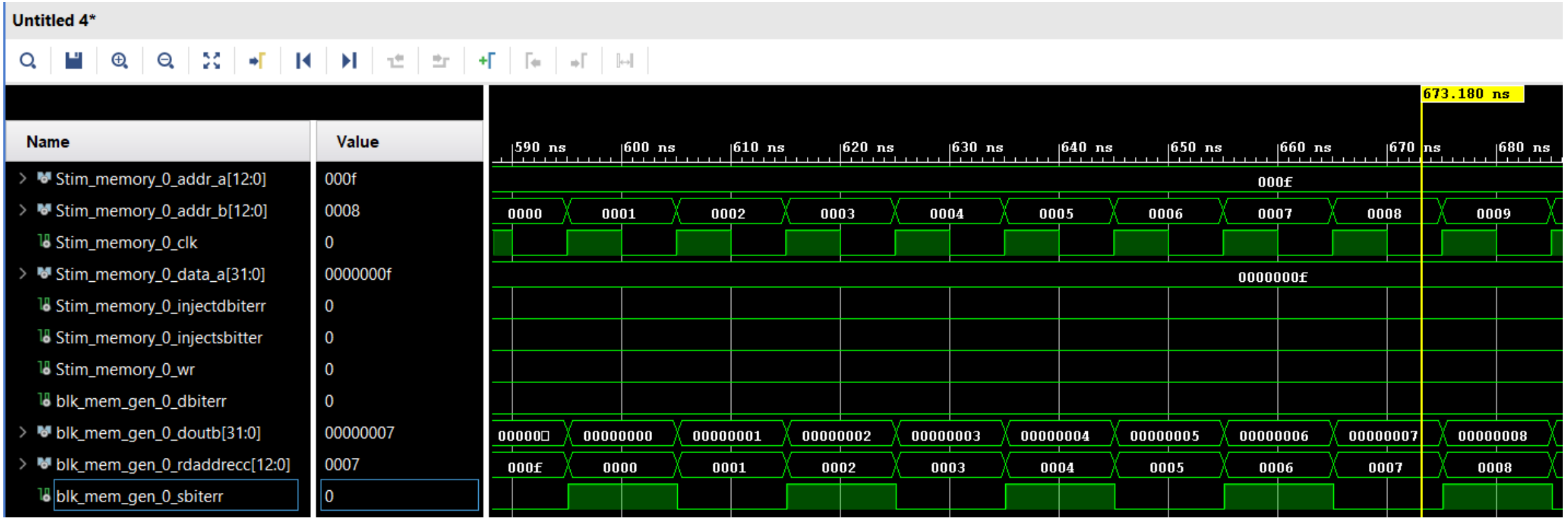




Injection of Error in to BRAM

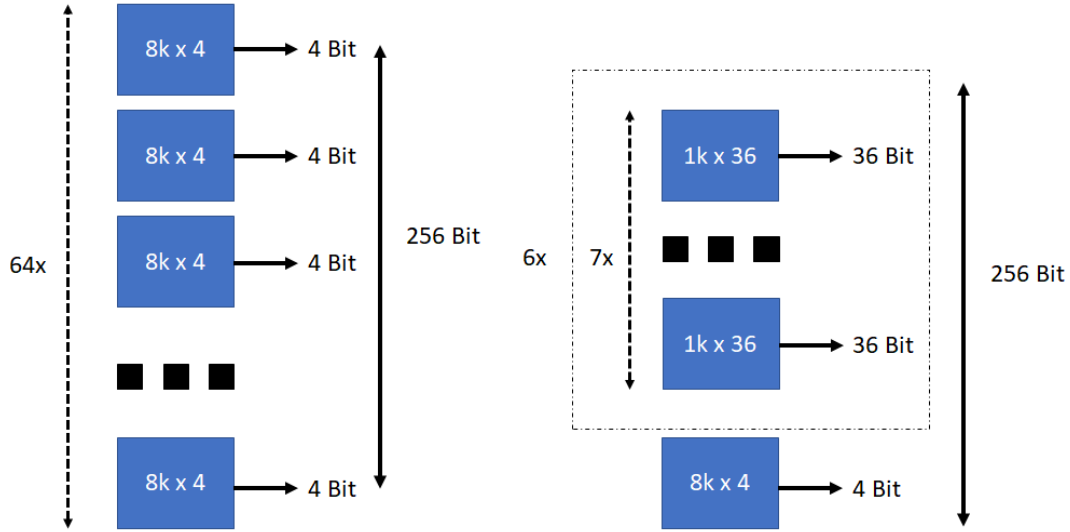


Read out of Error



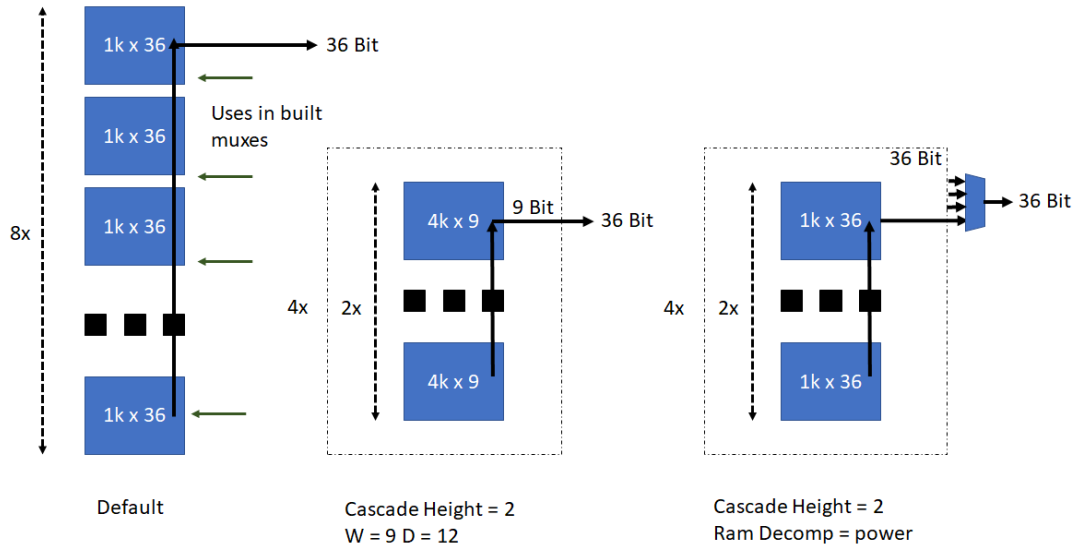
BRAM Optimisations

- Optimise BRAM for power / performance
- 6K by 256 RAM implemented using a 64 BRAMS configured as 8K by 4 bits
- 7 BRAMS configured as 1K by 36 repeated 6 times + one 8k x4 using 43 BRAMS
- RAM_decomposition constraint can be used for second structure



BRAM Optimisations

- Can also use RAM cascade_height constraint
- Not just a choice between power and performance
- Also possible to combine the cascade_height and ram_decomposition



DSP48

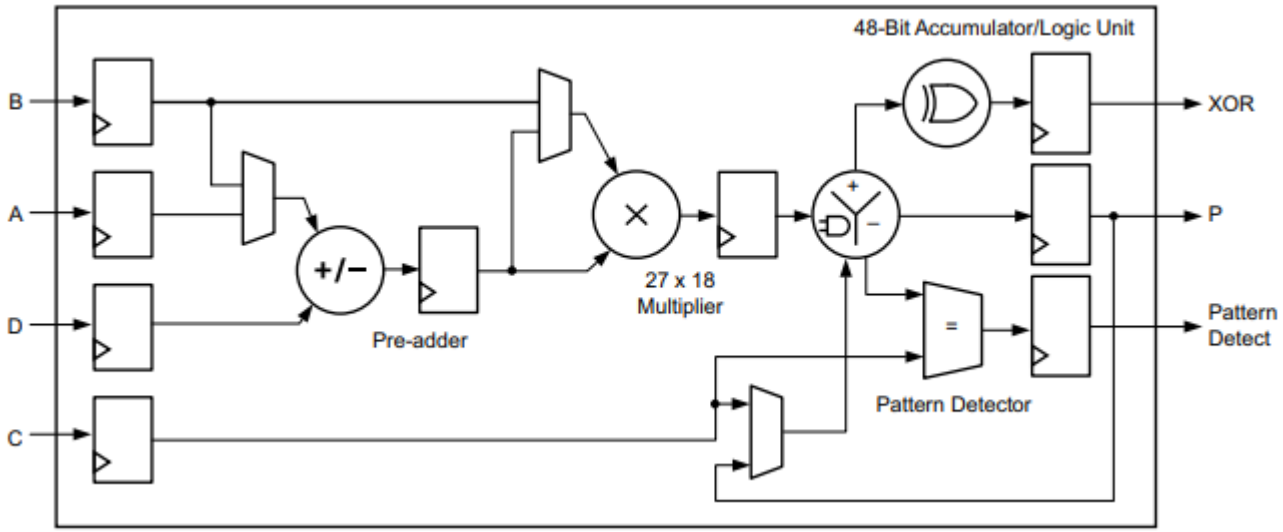
ADIUVO
ENGINEERING AND TRAINING, LTD.

DSP

Mathematical structures such as add / subtract / Multiply / Division can be implemented in FPGA using LUT and Flip Flops

- Not efficient for high performance - Better to have dedicated resources in the FPGA

Enter the DSP48



DSP

Applications

- Fixed- and floating-point Fast Fourier Transform (FFT) functions
- Systolic FIR filters
- MultiRate FIR filters
- CIC filters
- Wide real/complex multipliers/accumulators

Multi Giga Bit Transcievers

ADIUVO
ENGINEERING AND TRAINING, LTD.

Multi Giga Bit Transceivers

Xilinx FPGA implement high speed interfaces such as SATA, USB3, PCIe, Display port and Chip to Chip communications using MGTs

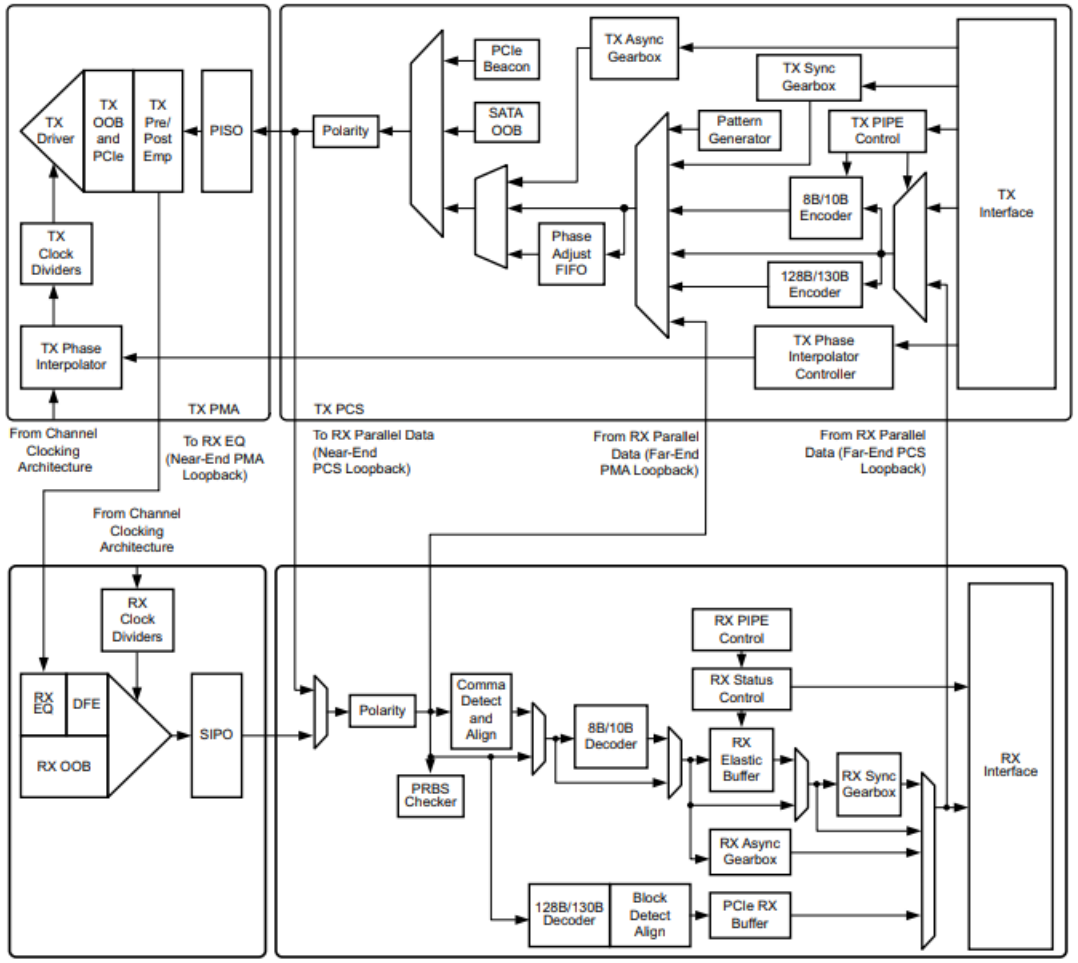
Special dedicated transceivers in the device – Normally in Quads with refence clock as well

Differential signaling and use Current Mode Logic

PMA Physical Media Attachment

PCS Physical Coding Sublayer

MGT Internals



Bandwidths Supported

Device	Type	Max Bandwidth (Gbs)
Versal ACAP	GTY / GTM	32.75 / 58.0
Veral Premium	GTY / GTM	32.75 / 112.0
Virtex US+	GTY / GTM	32.75 / 58.0
Kintex US+	GTH / GTY	16.3 / 32.75
Virtex US	GTH / GTY	16.3 / 30.5
Kintex US	GTH	16.3
Virtex 7	GTX / GTH / GTZ	12.5 / 13.1 / 28.05
Kintex 7	GTX	12.5
Artix 7	GTP	6.6
Zynq MPSoC	GTR / GTH / GTY	6.0 / 16.3 / 32.75
Zynq	GTX	12.5

Constraints

ADIUVO
ENGINEERING AND TRAINING, LTD.

Constraints

Help us instruct and guide the implementation tool

Multiple different types of constraints

- Timing constraints— The timing relationships required for correct operation
- Timing Exceptions— Define any exceptions to those constraints e.g. Multi Cycle Paths or False Paths
- Implementation constraints—Constraints used in the design's placement and routing e.g. IO location / type / location in the device

Clock Constraints

Vivado has several different types of clock constraints

- Primary Clocks – those that enter through an I/O pin
- Generated Clocks – those which are generated automatically via an internal PLL or by the design (for instance, dividing a clock by two with a flip-flop). With generated clocks, one describes how the master clock (either a prime or other generated clock) modifies the waveform
- Virtual Clocks – These are not attached to anything within the design netlist but can be used for I/O timing

Clock Constraints

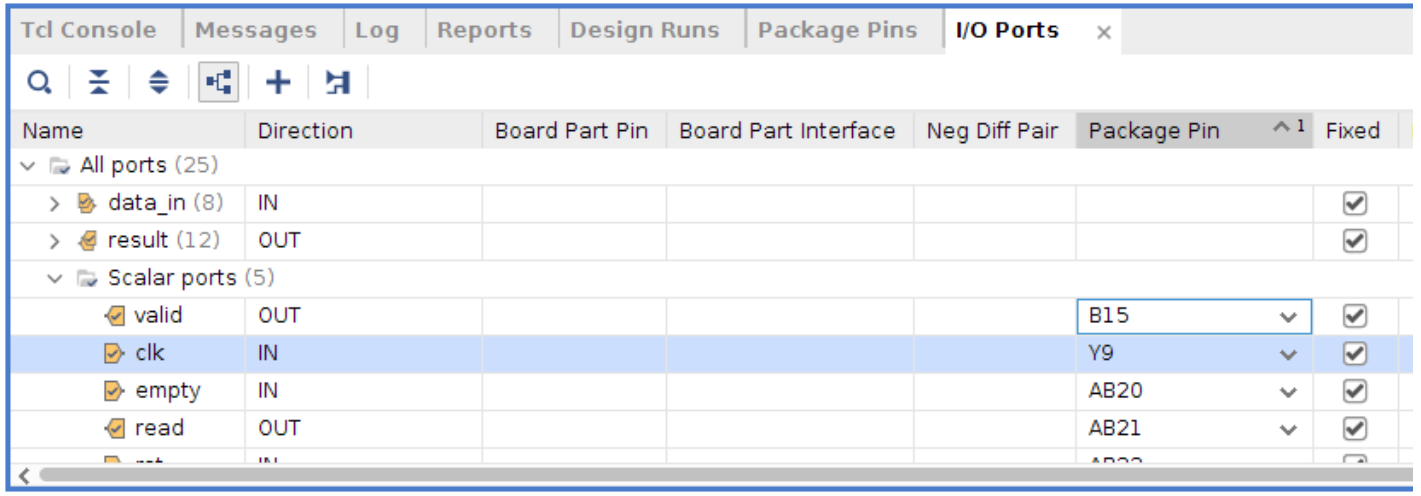
Vivado has three different types of clocks

- Synchronous Clocks – Synchronous clocks have a predictable timing/phase relationship, which is normally the case for a primary clock and its generated clocks because they share a common root clock and will therefore have a common period.
- Asynchronous Clocks – Asynchronous clocks have no predictable timing/phase relationship, which is normally the case for different primary clocks (and the clocks generated from these primary clocks). Asynchronous clocks have different roots.
- Unexpandable Clocks – Two clocks are unexpandable if a common period cannot be determined over 1000 clock cycles. If a common clock period cannot be established, then Vivado uses the worst case set-up relationship over the 1000 cycles. However, there is no guarantee that this relationship truly represents the actual worst case. That estimate is just the best that Vivado can do with the information provided.

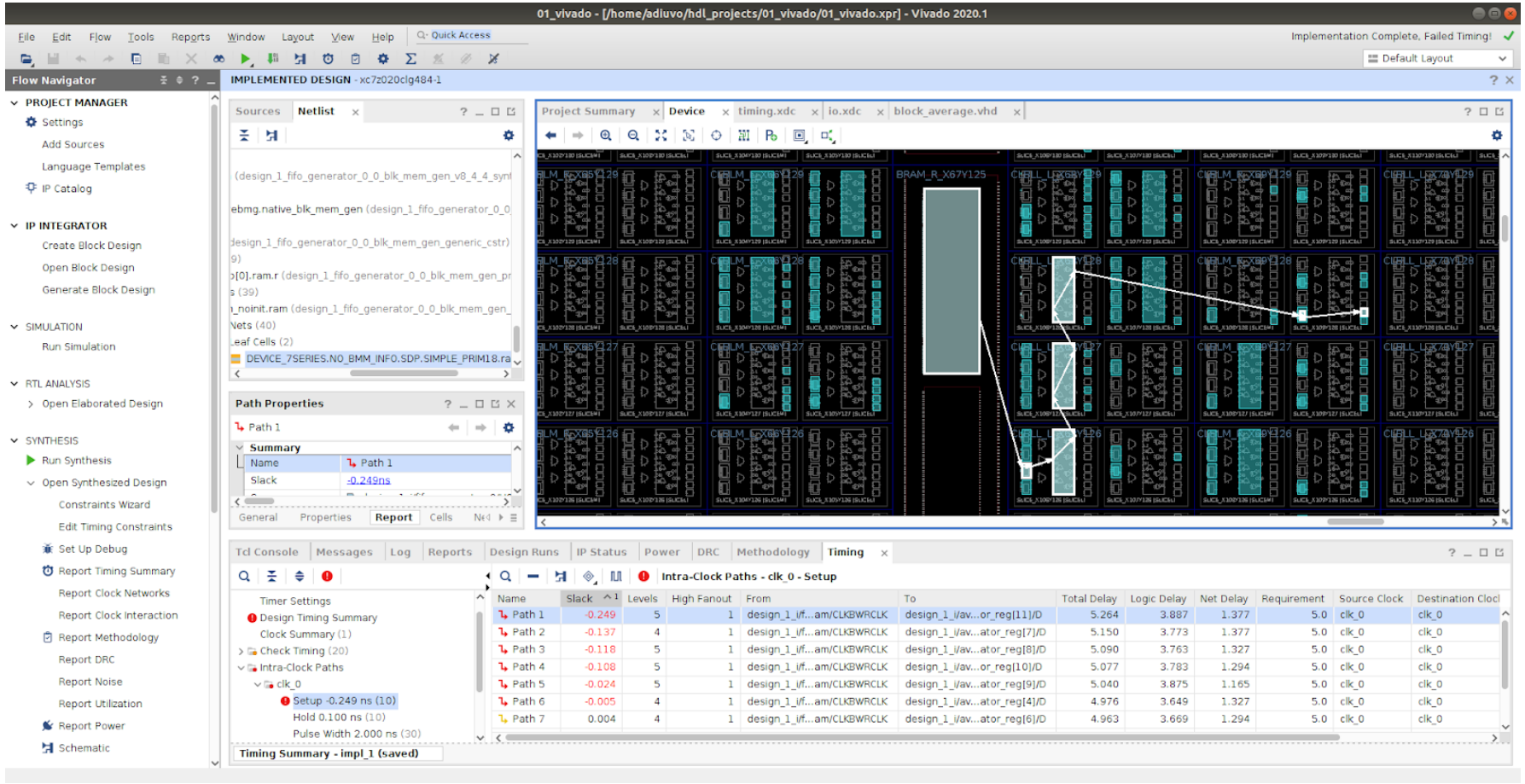
Placement Constraints

Vivado has a range of constraints we can use

- Placement Constraints – define cell location
- Routing Constraints – define signal routing
- I/O Constraints – define I/O location and I/O parameters
- Configuration Constraints – define configuration methods



Placement Constraints



Device Selection

ADIUVO
ENGINEERING AND TRAINING, LTD.

Benefit of FPGA

FPGA's offer several benefits to the system designer

- Flexibility of Design – performance, upgrades
- Reduction in NRE and Cost.
- Reliability – One Device as opposed to lots if using discrete devices - increased reliability with reduced solder connections.
- Time to market can be reduced.
- Maintainability – ability with some FPGA to update in the field.

Device Selection - SRAM, OTP or FLASH ?

- SRAM: The FPGA program is stored in an external memory and loaded into the FPGA each time it is powered.
- FLASH: The FLASH architecture of the FPGA also contains the program; no external memory device is needed.
- One Time Programmable (OTP): The FPGA is applied by blowing fuses in the device. Once programmed, it cannot be modified.

Device Selection

- SRAM
  - Higher Performance (+)
  - Higher Capacity (+)
  - Needs configuration at power up (-)
  - Higher Power (-)
  - Susceptible to Configuration Corruption (-) – but there are mitigation schemes
- OTP
  - Live at power up (+)
  - Cannot be updated / fixed design (-)
  - Lower Power (+)
  - Lowest Performance (-)
- Flash based
  - Live at power up (+)
  - No Configuration Corruption (+)
  - Updateable in the field (+)
  - Middling Performance (-)

Device Selection

Typical use cases

- SRAM
  - Software defined radio, image processing, satellite communications, EW, AI etc.
- OTP
  - Control and communication, security functions e.g. crypto
- Flash
  - Control & Communication, Crypo, image processing, SDR

Vivado IP Integrator

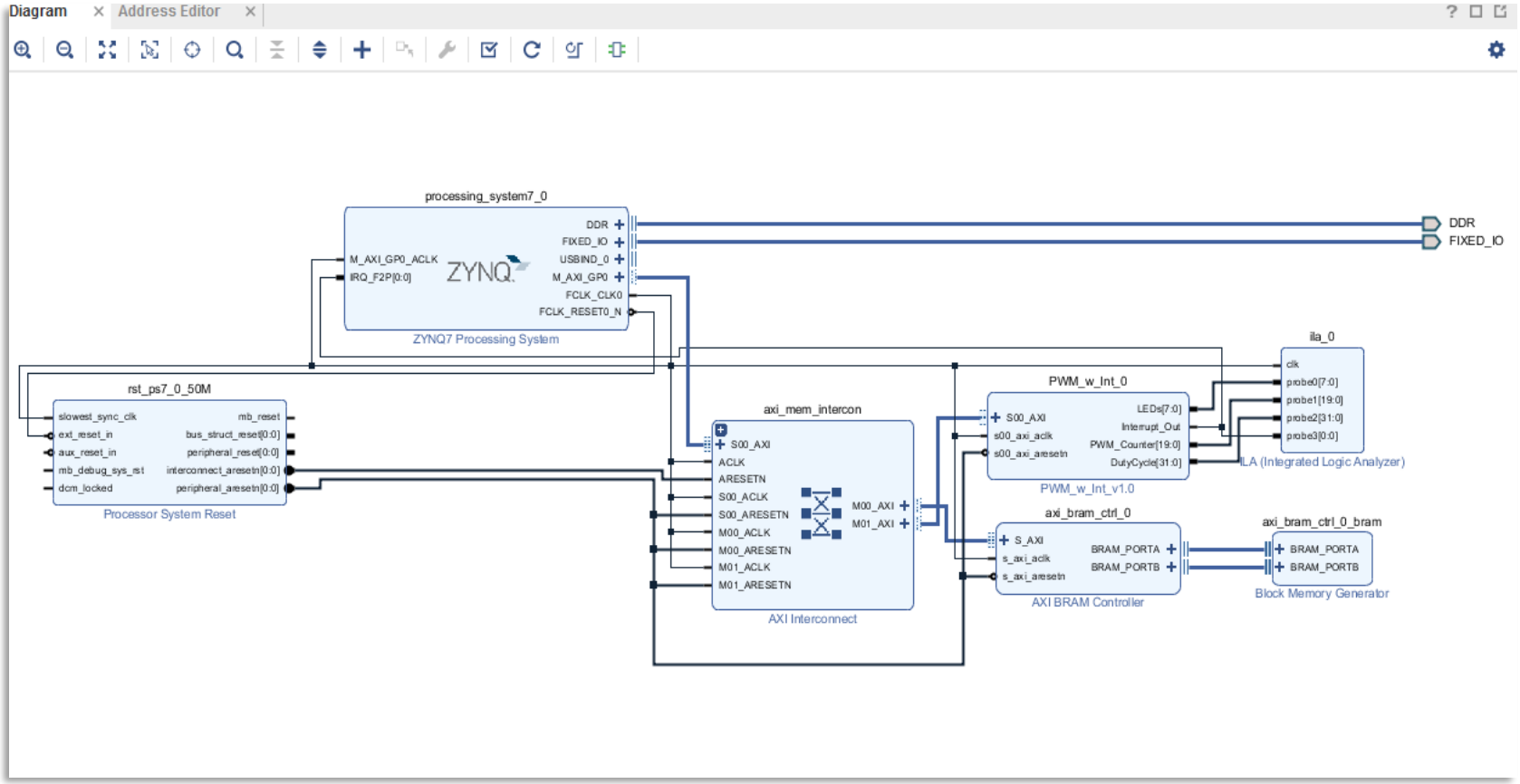
ADIUVO
ENGINEERING AND TRAINING, LTD.

Vivado IP Integrator (IPI)

Create system-level designs

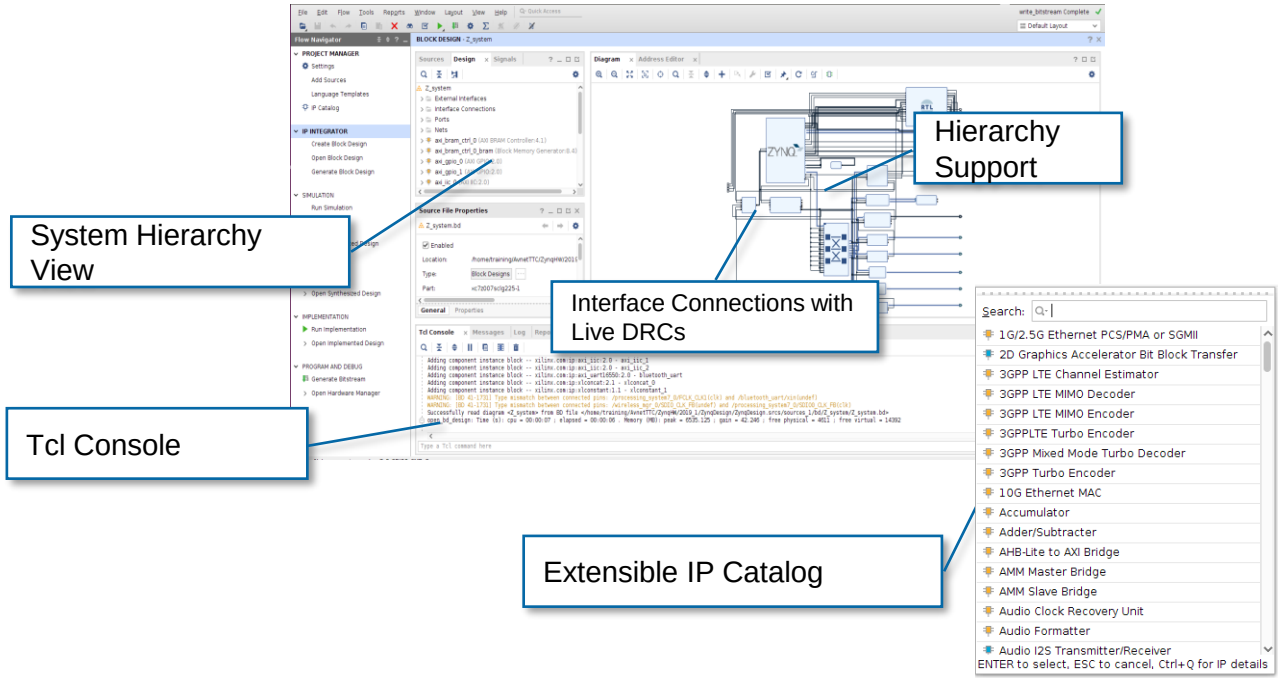
- Instantiate and interconnect IP cores

- IP-centric design flow
  - Plug-and-play IP
  - Vast IP catalog
- Accelerates
  - Integration
  - Productivity
- Example applications
  - Embedded
  - DSP
  - Video
  - Analog
  - Networking



Vivado IP Integrator: Intelligent IP Integration

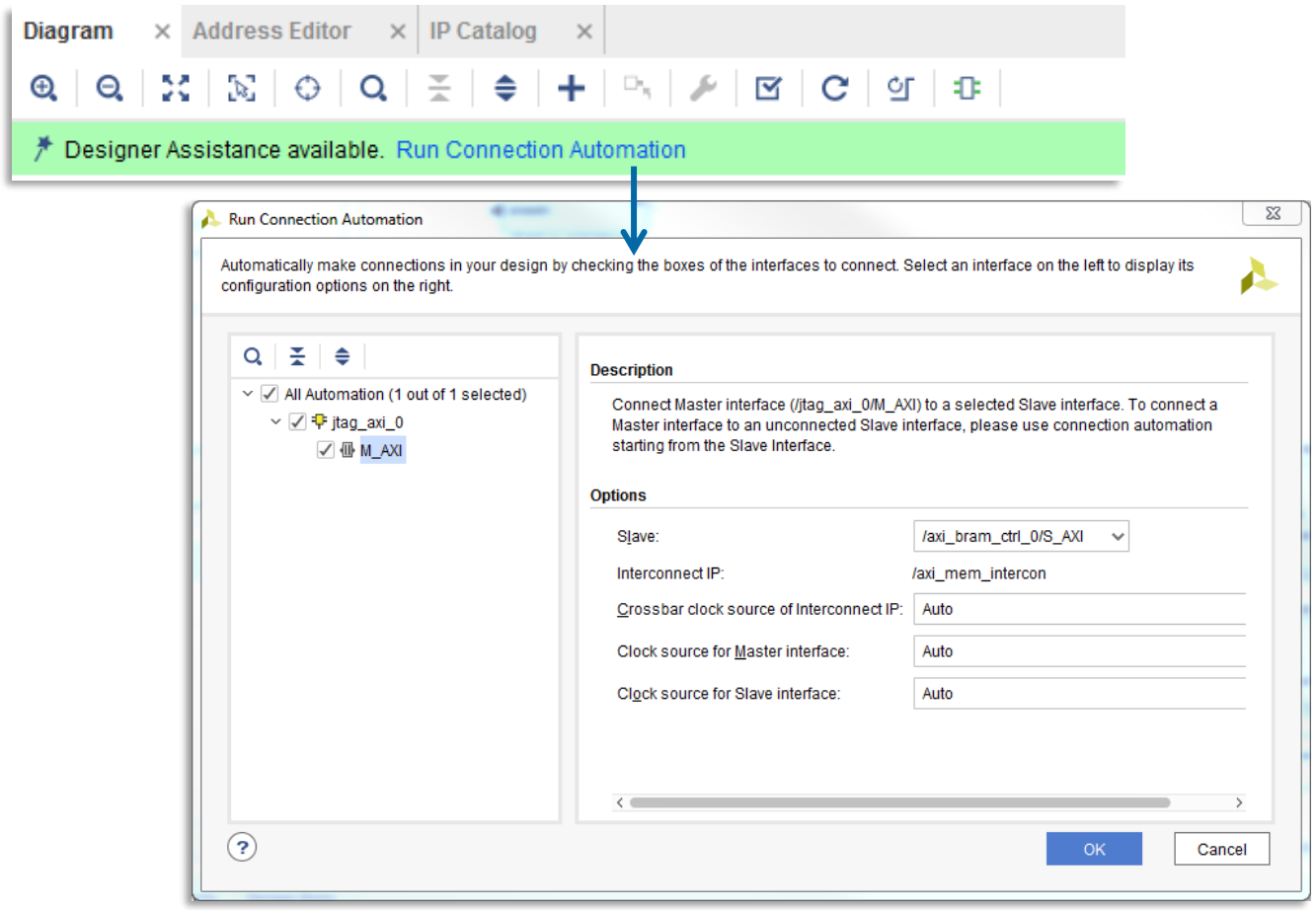
- Automated IP subsystems
- Block automation for rapid design creation
- One click IP customization
- Board aware
- Support all 7 Series FPGAs and Zynq SoCs
- Built-in presets, accelerating design creation



Vivado IP Integrator: Intelligent IP Integration

Correct-by-construction

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

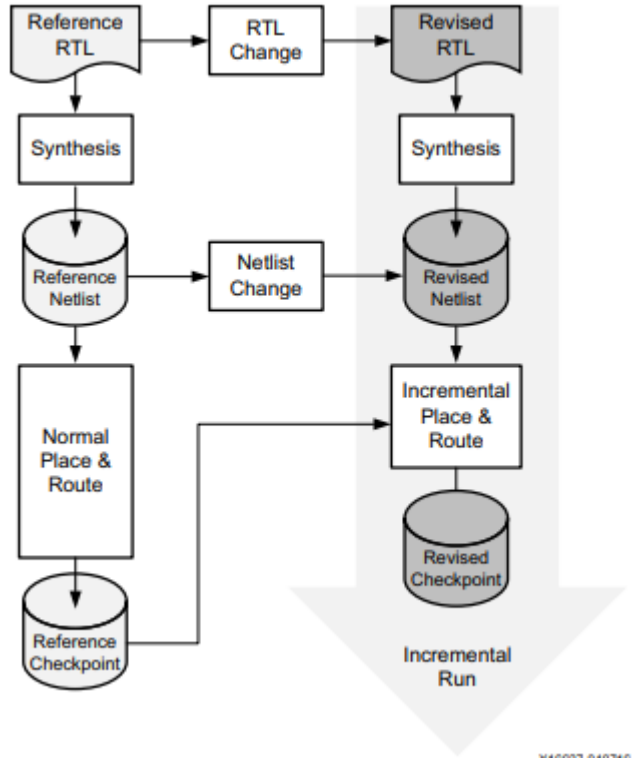


Reducing Run Time

ADIUVO
ENGINEERING AND TRAINING, LTD.

Design Check Points

- Vivado uses a physical design database to store placement and routing information
- Design checkpoint files (.dcp) allow you to save and restore this physical database at key points in the design flow



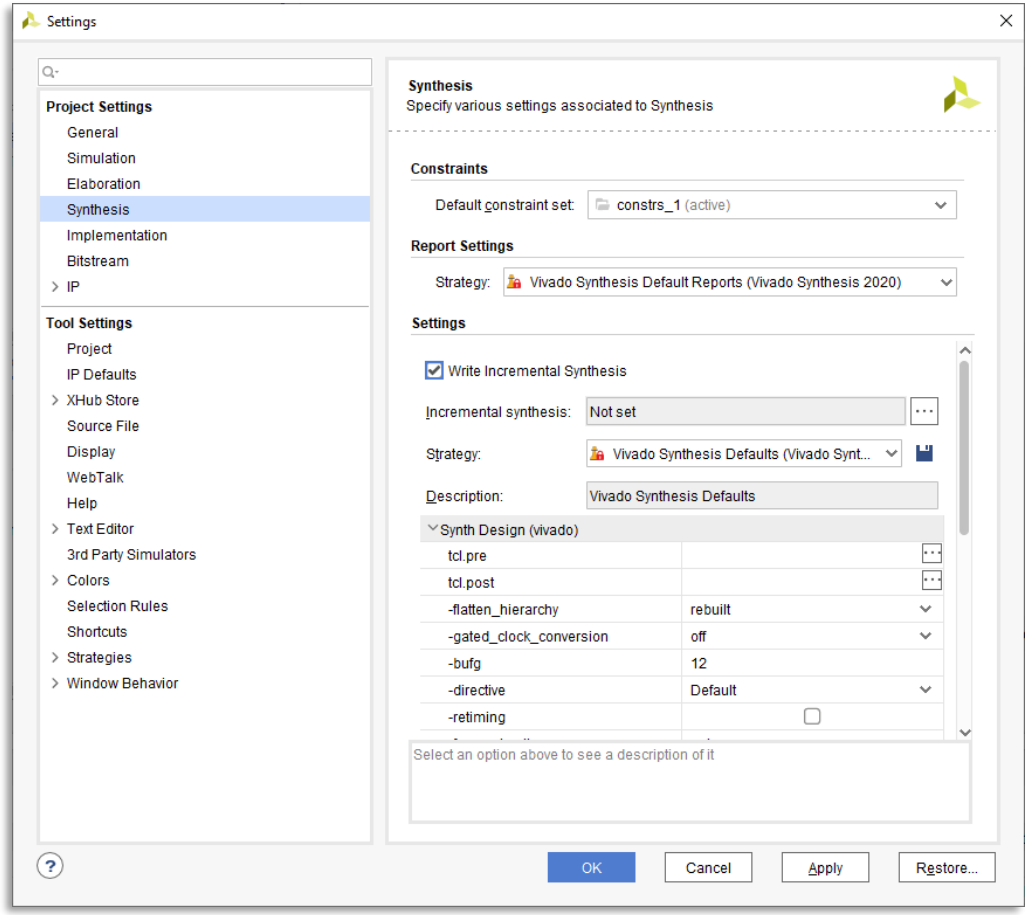
Vivado – Reducing Compile Time

- Implementing a FPGA design can take a considerable time > 1 Hour
- Iterating the design can therefore be an issue, there are several options which can reduce the implementation time both in synthesis and place and route

Synthesis	
Global	Performs a traditional top-down synthesis of the entire design. Selecting this option takes the longest time because you need to re-run the entire synthesis every time you make a change.
Out of Context Per IP	Runs synthesis and creates a Design Check Point (DCP) for every individual IP block within your design. These check points are then collected into a black-box at the top-level implementation. Using this option means that only the blocks you change need to be re-synthesized, which saves time. OOC-IP also creates an IP customization file (XCI) for each IP block, allowing for customization and OOC XDC files. OOC-IP is the default setting for synthesis within Vivado. This option applies to all IP within the block diagram.
Out of Context Per Block Diagram	Like the OOC-IP option however this option allows you to define the entire block diagram as OOC

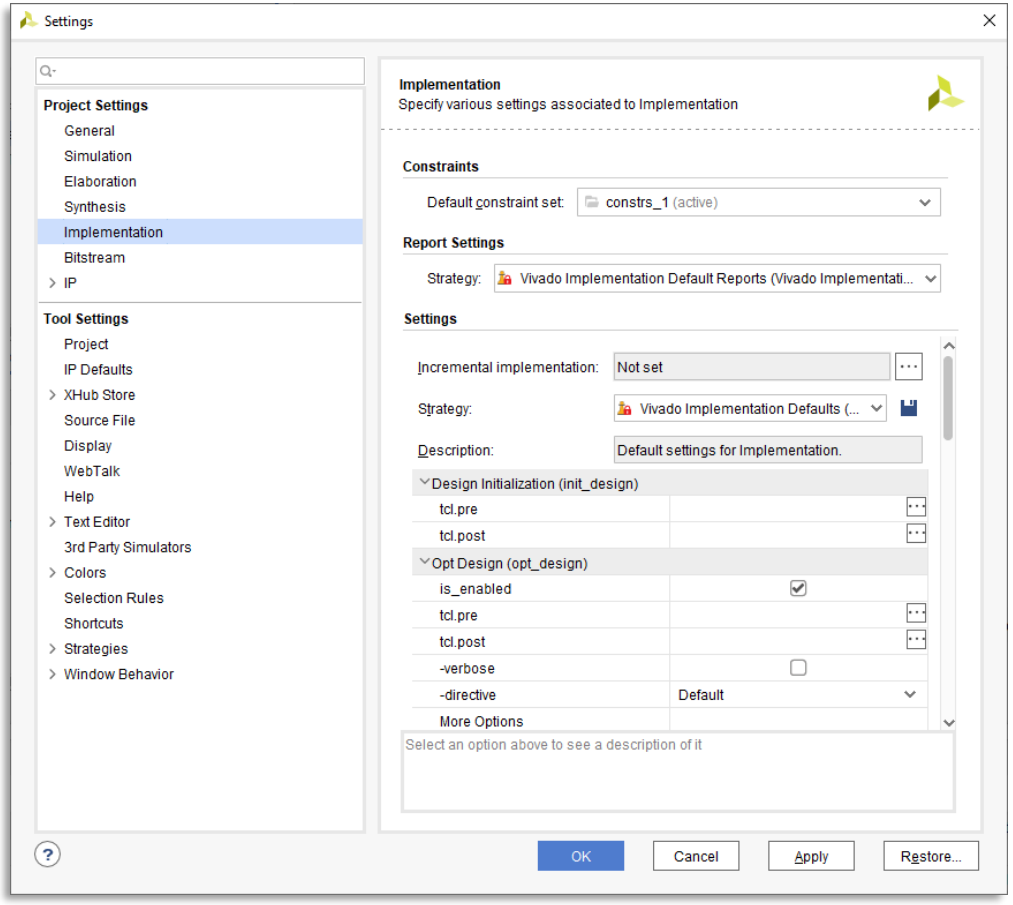
Vivado – Reducing Compile Time

- Incremental synthesis can also be used when the changes are small
- Write out incremental synthesis to the post synthesis design check point
- Selecting incremental synthesis then provides two options
  - Automatically use previous DCP
  - Use a defined DCP



Vivado – Reducing Compile Time

- Incremental implementation allows use of Design Check Point as the starting point
- Preserves QoR predictability by reusing prior placement and routing from a reference design
- Speeds up place and route run time or attempts last mile timing closure



Strategies and Reports

ADIUVO
ENGINEERING AND TRAINING, LTD.

Vivado – Strategies

- Strategies are a defined set of Vivado implementation feature options that control the implementation results.
- These strategies can be used to explore:
  - Timing Performance (e.g., Performance_Explore)
  - Congestion - Strategies to reduce routing congestion in areas of the design
  - Area - Optimise for area
  - Power – Optimise for power
  - Quick Flow – Reduced implementation time

We should always of course try to achieve timing closure

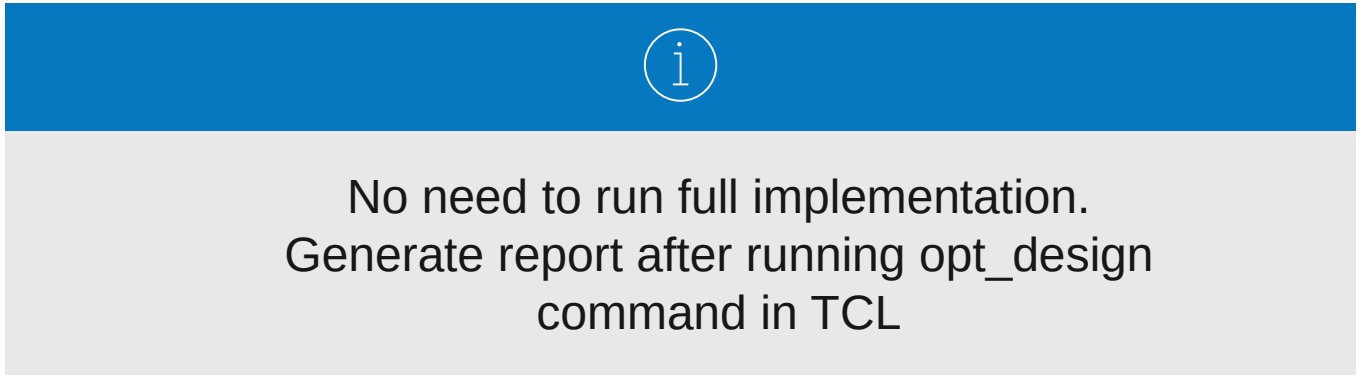
Vivado – Reports

- Vivado provides several reports which can be used to help focus in on performance issues in the design:
  - Design Analysis Report – Provides information on design timing, congestion, and complexity of design
  - Quality of Result Report – Provides overall design assessment and methodology check – QOR can also make suggestions to fix issues in the design.
- Both are very useful to achieve timing closure of the design

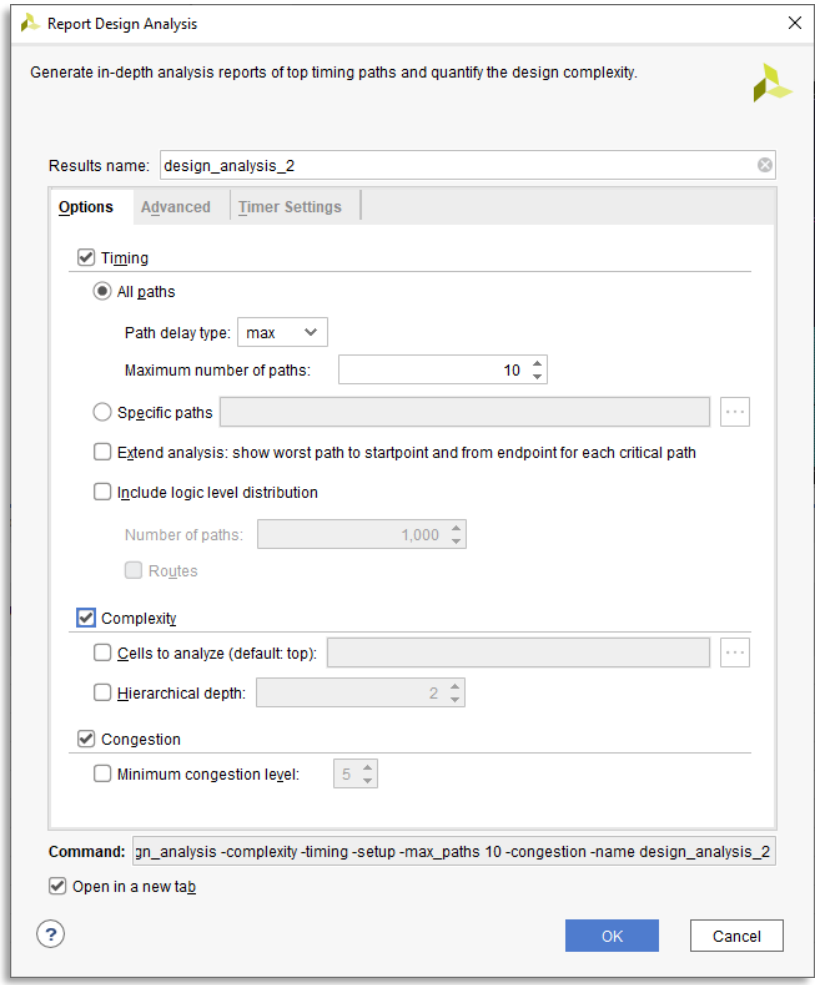
Vivado – Design Analysis Report

Design Analysis Report provides information on:

- Timing – Provides information on the timing and physical characteristics of timing paths.
- Complexity – Provides information on routing complexity and LUT distribution.
- Congestion – Provides information on routing congestion

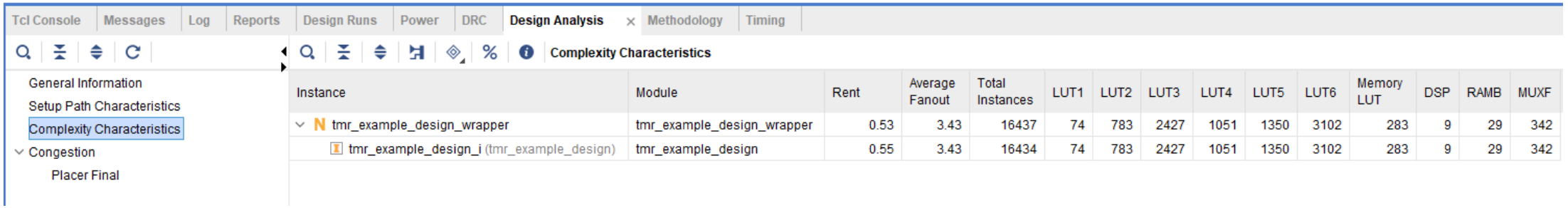


No need to run full implementation.
Generate report after running opt_design command in TCL



Vivado – Design Analysis Report

- Along with timing information, DAR can provide information on design complexity including indicating design risk for implementation
- Low Risk Rent Analysis <0.65 and Fan Out <4
- High Risk Rent Analysis >0.65 <0.85 and Fan Out >4 % <5 – May be difficult to place without congestion



Instance	Module	Rent	Average Fanout	Total Instances	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	Memory LUT	DSP	RAMB	MUXF
N tmr_example_design_wrapper	tmr_example_design_wrapper	0.53	3.43	16437	74	783	2427	1051	1350	3102	283	9	29	342
tmr_example_design_i (tmr_example_design)	tmr_example_design	0.55	3.43	16434	74	783	2427	1051	1350	3102	283	9	29	342

Vivado – Quality of Result

- Quality of Result Assessment (QoRA) and Quality of Result Suggestions (QoRS) since both provide information that can be used to achieve timing closure
- Like Design Analysis Report – Run initially after doing the Opt

1. Overall Assessment Summary

QoR Assessment Score	2 - Implementation may complete. Timing will not meet
Report Methodology Severity	Critical warnings
ML Strategy Compatible	Yes
Incremental Compatible	No
Next Recommended Flow Stage	Review methodology warnings and fix or waive them

3. Methodology Check Details

ID	Description	Criticality	Number of Violations
TIMING-6	No common primary clock between related clocks	Critical Warning	2
TIMING-7	No common node between related clocks	Critical Warning	2
TIMING-8	No common period between related clocks	Critical Warning	2
TIMING-9	Unknown CDC Logic	Warning	1

Vivado – Quality of Results

SCORE	MEANING	CORRECTIVE ACTION
1	Design will not implement	Redesign RTL / HLS modules
2	Design will implement timing problems	Review constraints & RTL HLS
3	Design Runs have a small chance of success	Use QoR suggestions, review clocking, ML strategies
4	Design should meet timing if directives used	Use QoR suggestions, ML strategies
5	Design will implement without timing issues	Run Implementation

Vivado – Quality of Results

```
Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------
| Tool Version : Vivado v.2020.1 (win64) Build 2902540 Wed May 27 19:54:49 MDT 2020
| Date         : Sat Aug 29 14:13:22 2020
| Host         : DESKTOP-L30MJC1 running 64-bit major release  (build 9200)
| Command      : report_qor_suggestions -file QOR_Suggestions.txt
| Design       : design_1_wrapper
| Device       : xczu9eg
| Design State : Fully Placed
| ML Models    : v2019.2.0
------------------------------------------------------------------------------------

Report QoR Suggestions

Table of Contents
-----------------
1. QoR Suggestions Report Summary
2. ML Strategies
3. QoR Suggestions - XDC

1. QoR Suggestions Report Summary
---------------------------------

+-------------+-----------+-----------+-------------+---------------+-----------+--------------------+-----------------------------------------------------------------------------------------------------+-------------+
|     Name    |     Id    |   Status  | Generated At | Applicable For | Automatic | Incremental Friendly |                                                Description                                          |    Source   |
+-------------+-----------+-----------+-------------+---------------+-----------+--------------------+-----------------------------------------------------------------------------------------------------+-------------+
| RQS_XDC-3-1 | RQS_XDC-3 | Generated | place_design | synth_design   | No        | No                 | Tight constraints for given unsafe paths. Fix unsafe paths by amending the design or                | Current Run |
|             |           |           |             |               |           |                    | adding false path, datapath only, or clock group constraints.                                       |             |
+-------------+-----------+-----------+-------------+---------------+-----------+--------------------+-----------------------------------------------------------------------------------------------------+-------------+
* By default the number of failing paths is limited to 100. Use the -max_paths options to override.
** The design checks report may change until design is completely implemented/routed


2. ML Strategies
----------------

+---+----+---------+---------+
| # | Id | Command | Options |
+---+----+---------+---------+
* ML Strategies are available only in default/explore at successfully routed design.
```

Verification

ADIUVO
ENGINEERING AND TRAINING, LTD.

Verification

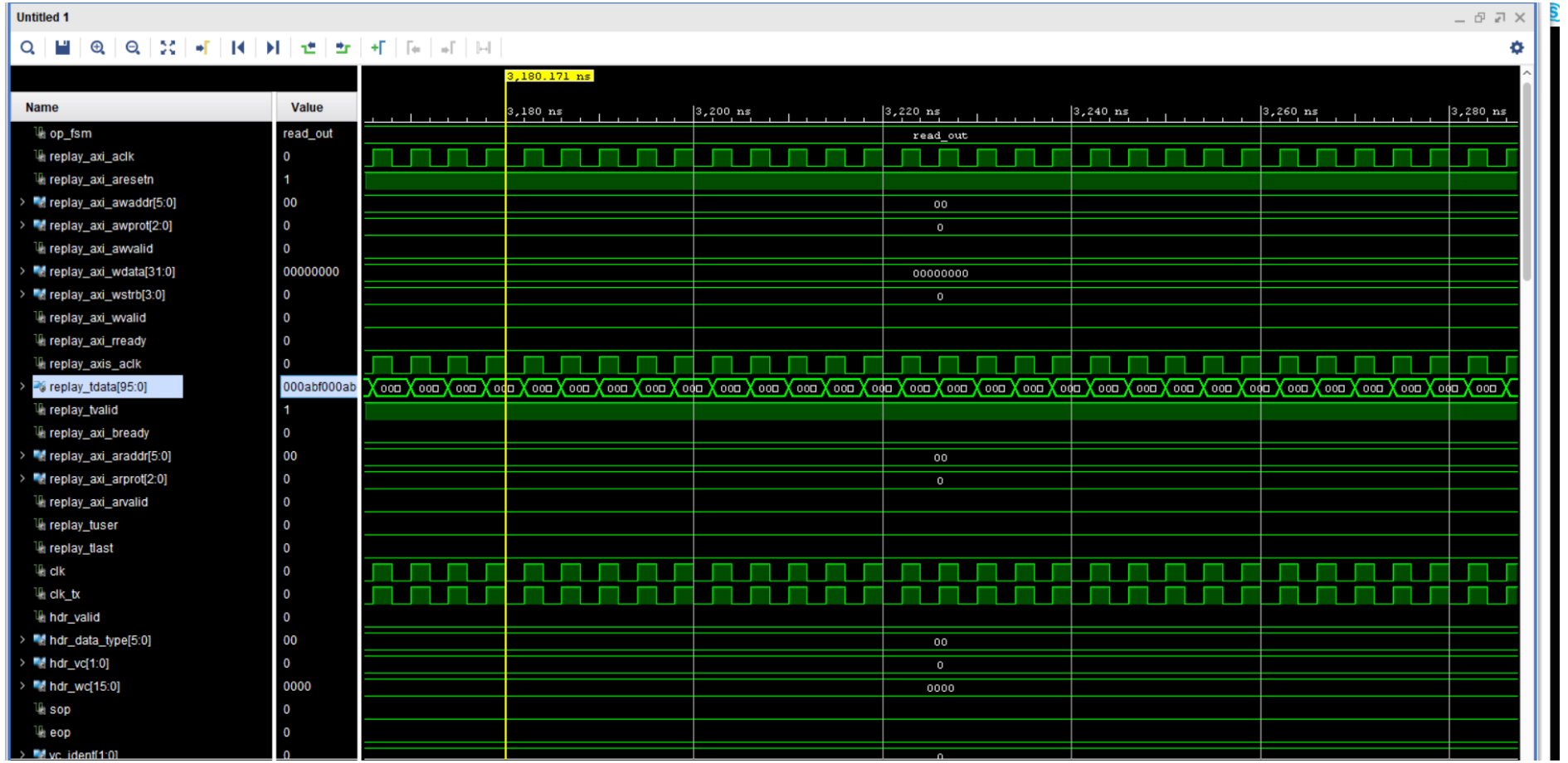
Before we can deploy an FPGA based solution we must be sure it functions as expected, across all use cases. This is where verification and validation come in, for those unfamiliar with the terms

- Verification – Does the FPGA function in line with the specification
- Validation – Is the specification correct i.e. does it address the needs of the use case.

While validation takes place mostly within the higher levels of system engineering domain, verification takes place at the FPGA level. Depending upon the needs of the applications verification can range from being very complex and time consuming to simply confirming the expected behaviour.

At the heart of verification is simulation and the simulator.

Simulation



Simulation

Verification can be used at several different levels

- Functional Simulation only – This check if the design is functionally correct
- Functional Simulation & Code Coverage – This checks that along with the functional correctness of the design that, all of the code within the design has been tested.
- Gate Level Simulation – This verifies the functionality of the design when back annotated with timing information from the final implemented design, this can take a considerable time to perform.

Verification What else can we do?

- Static Timing Analysis – This analyses the final design to ensure the timing performance of the module is achieved.

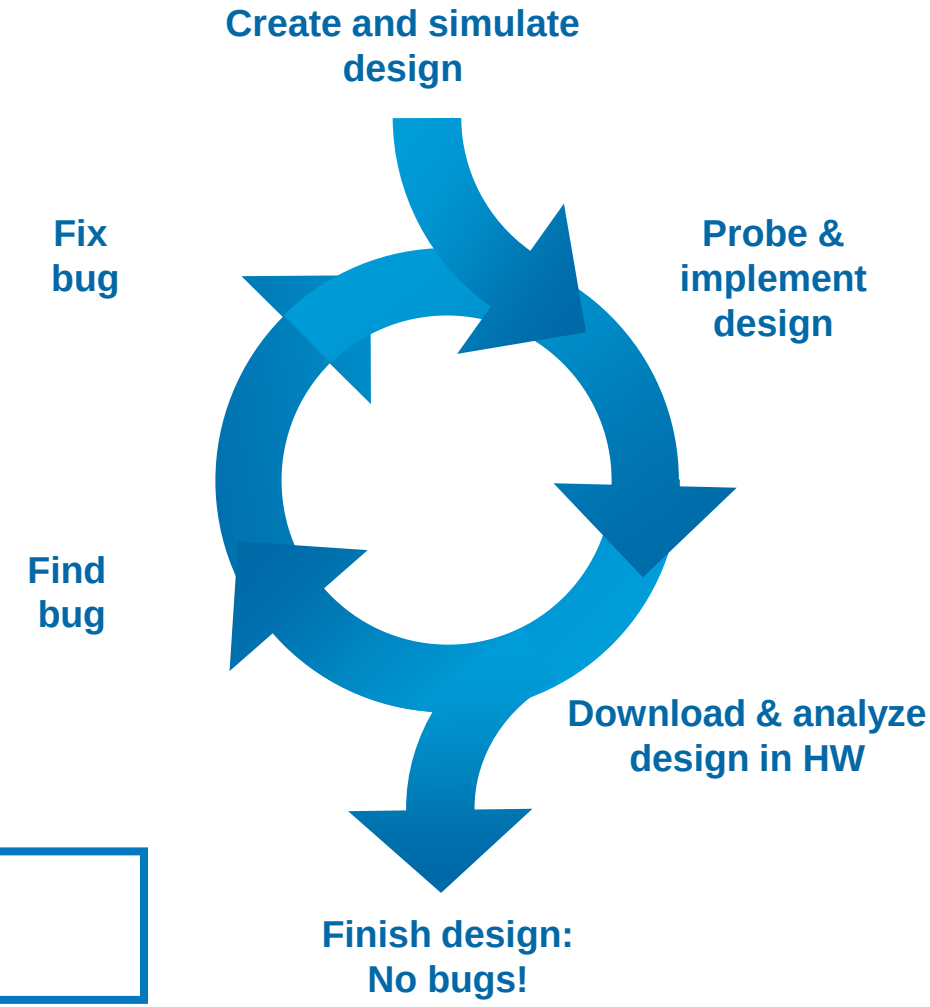
- Formal Equivalence Checking - This is used to check the equivalence of netlists against RTL files

Debugging

ADIUVO
ENGINEERING AND TRAINING, LTD.

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
  - Probe: Adding/modify debug probes
  - Implement: Compile design w/probes
  - Analyze: Look for bugs using probes
  - Fix: Fix any bugs, repeat as necessary



The reconfigurable nature of FPGAs enables the iterative debug process

Vivado Logic Debug IP

ILA

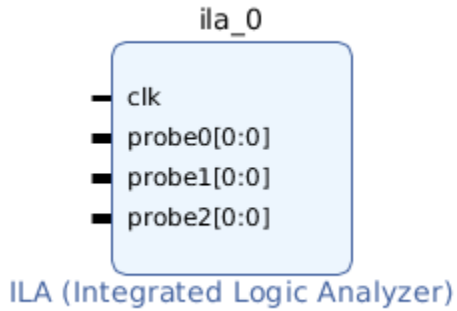
- Integrated Logic Analyzer debug IP core
- Netlist insertion support
- HDL instantiation support
- Added support for advanced trigger and capture features

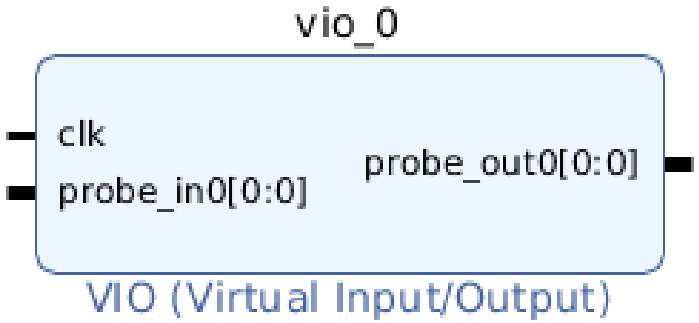
VIO

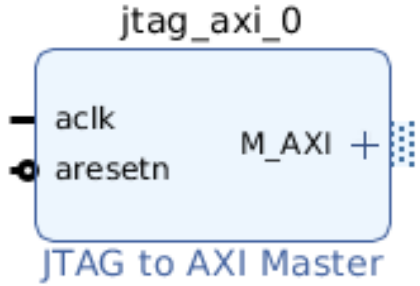
- Vivado native Virtual Input / Output
- HDL instantiation support

JTAG to AXI

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



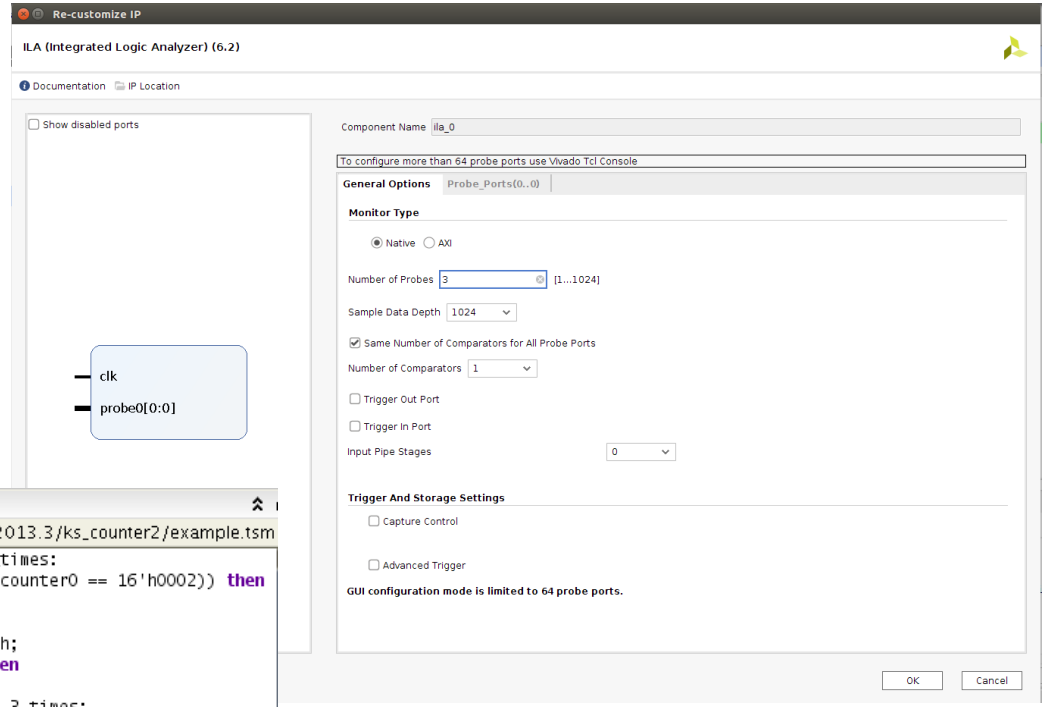


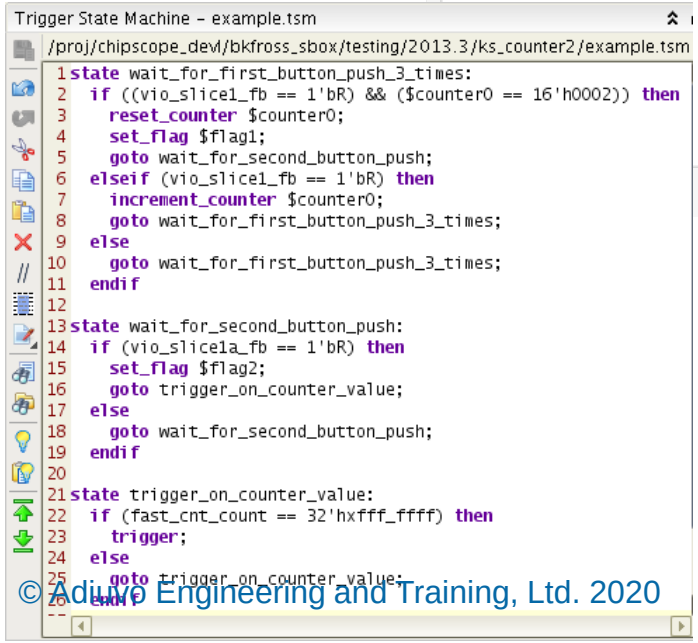


ILA Advanced Trigger Features

Advanced trigger state machine:

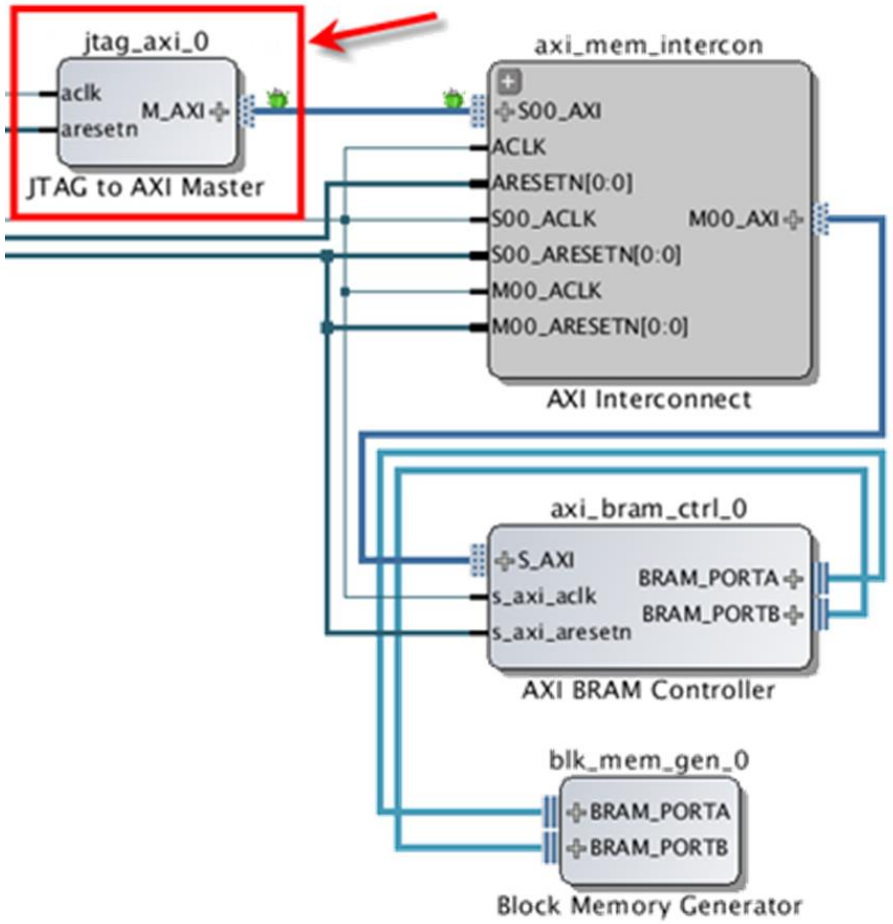
- Fully programmable at run-time
- Up to 16 states
- 3-way branching per state
- Up to four comparators per PROBE input
- Four programmable counters (reset, increment, conditional compare)
- Four programmable flags (set, clear, monitor in GUI)





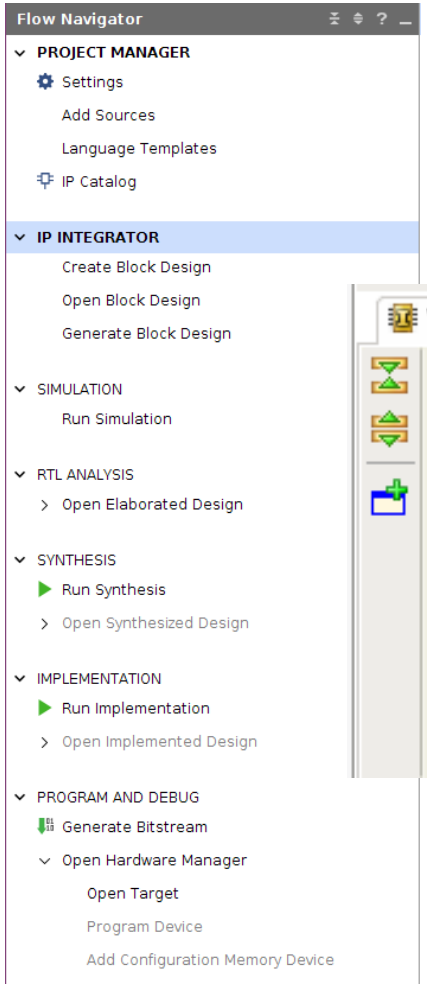
JTAG to AXI Master

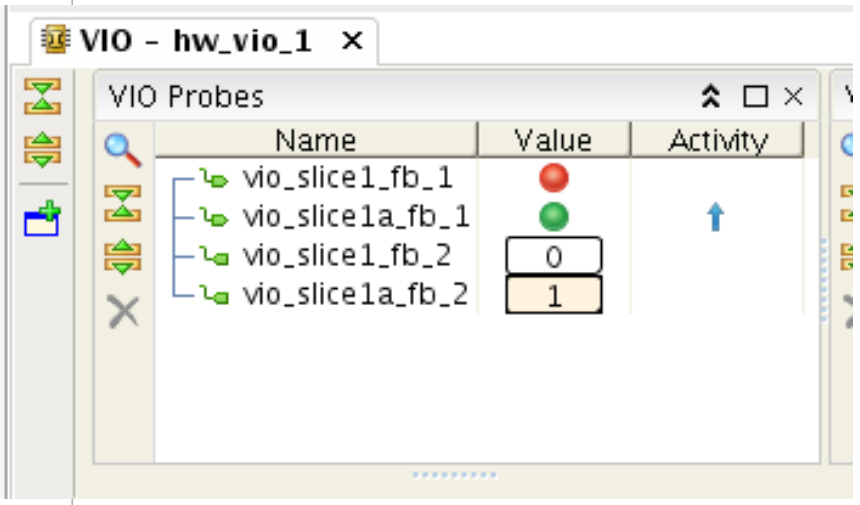
- Interact with AXI-based system **without** writing microprocessor code
- Connects to AXI or AXI-Lite interfaces
- Can be used in both IP Integrator block designs and HDL designs
- Vivado run-time Tcl commands to create and run AXI transactions
  - help *hw_axi*

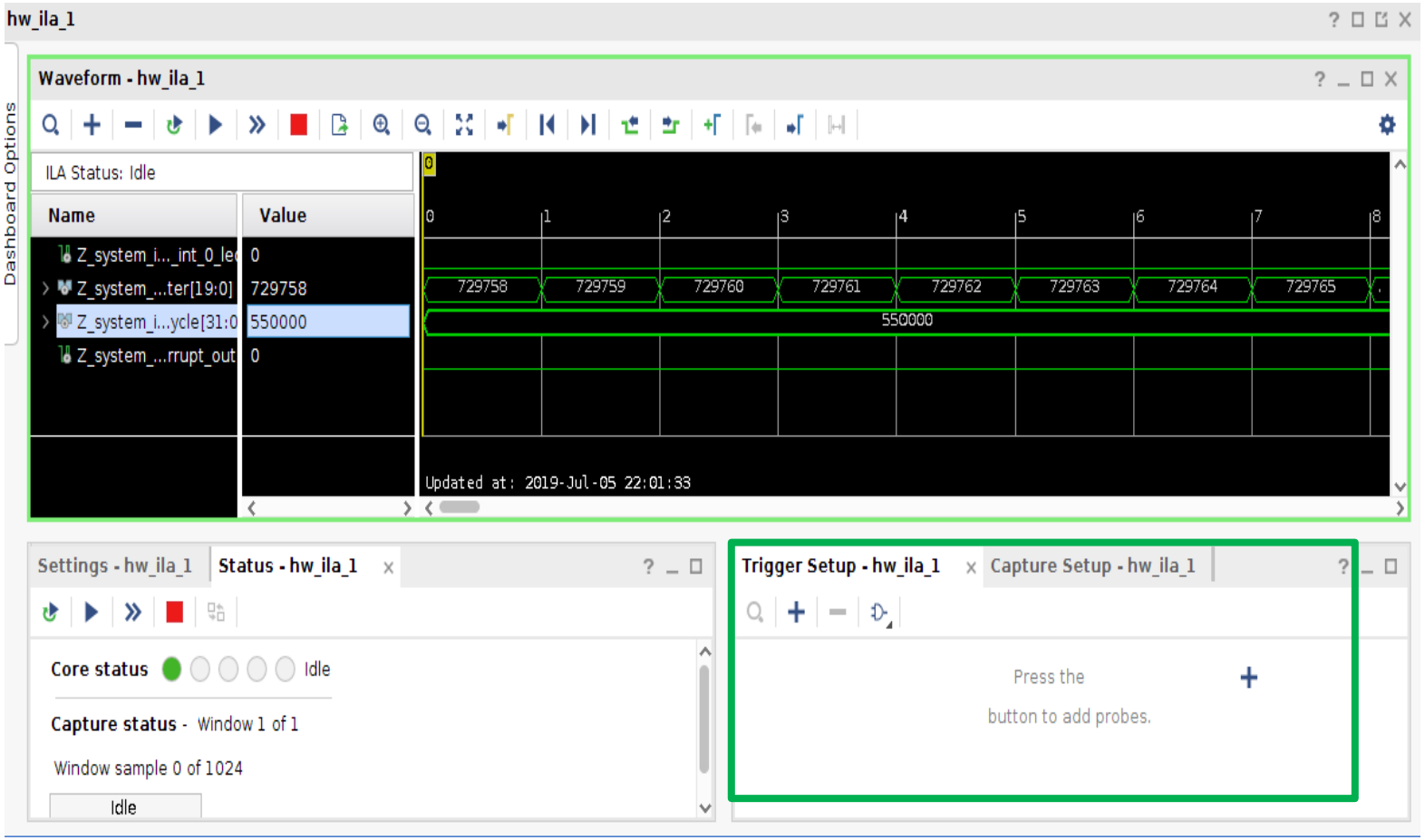


Hardware Manager & GUI Ease of Use

- 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







ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com

adam@adiuvoengineering.com