Vivado Boot Camp Day 1 Notes

Resources:  
 User Guides

Product Guides

**Vivado Synthesis Guide 901 –** coding attributes!!!

<https://slideplayer.com/slide/5967461/>

wp380\_Stacked\_Silicon\_Interconnect\_Technology.pdf

<https://forums.xilinx.com/t5/Versal-and-UltraScale/Ultrascale-clock-architecture-issue/td-p/782002>

ug572-ultrascale-clocking.pdf

<http://fpgasite.blogspot.com/2017/05/fpga-internal-tri-state-buses.html>

Use **Language Templates** thru the roof!!!

**Training Overview**:

**CLB**s:

Slices

**LUT**s

**CIB**s (Configurable Interconnect Blocks; Interconnect Fabric).

Memory Resources (UltraRAM vs HBM?):

**LUT**s

**Distributed** **RAM**

**B**lock **RAM**,

**UltraRAM** (288Kb SRAM) ???

High Bandwidth memory (HBM) ???

FIFO

DSP Resources

Interfacing:

I/O

SERDES design,

DDR4 Physical Layer Interfaces.

Clocking Resources:

MMCM (clock manager)

PLL

**Day 1 Overview**:

Architecture.

CLB Resources.  
HDL Coding Techniques.

Memory Resources.

DSP Resources.

**Day 2 Overview**:

IO Resources

Component Mode

Native Mode

Clocking Resources

**Day 3 Overview**:

Clocking Resources.

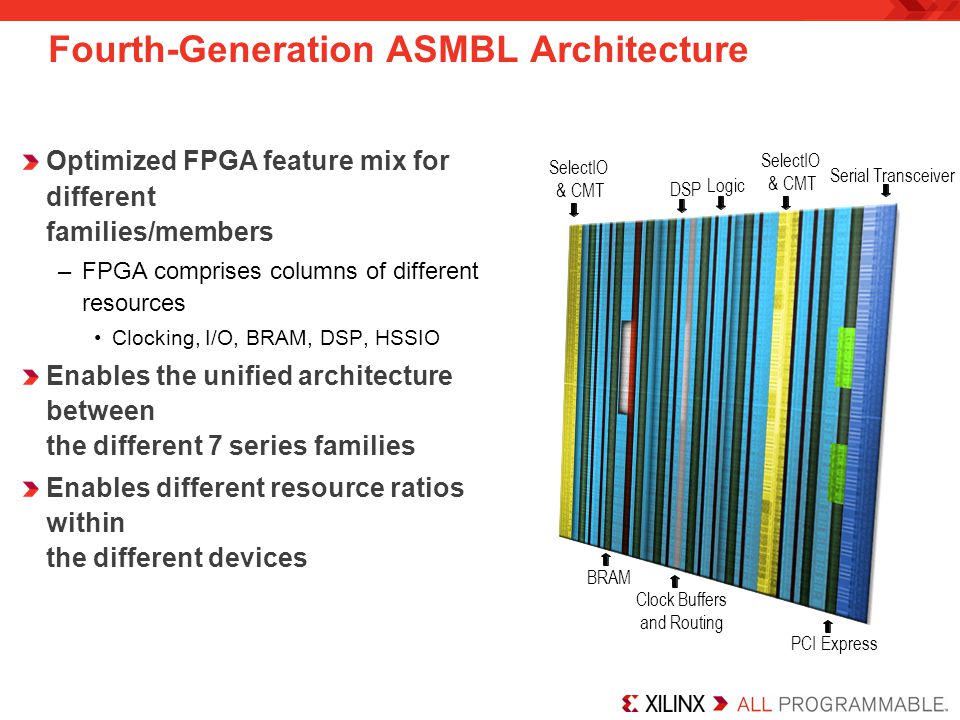
Dedicated Hardware (ADCs and such)

Transceivers.

Migration methodology.

Zynq SoC Architecture Overview.

Starting with the 7-series and up the Fourth Generation ASMBL Architecture (everything is organized in columns) is used:



Logic (CLB) Columns.

DSP Columns.

BRAM Columns and UltraRAM Columns.

Clock Buffers and Routing Columns.

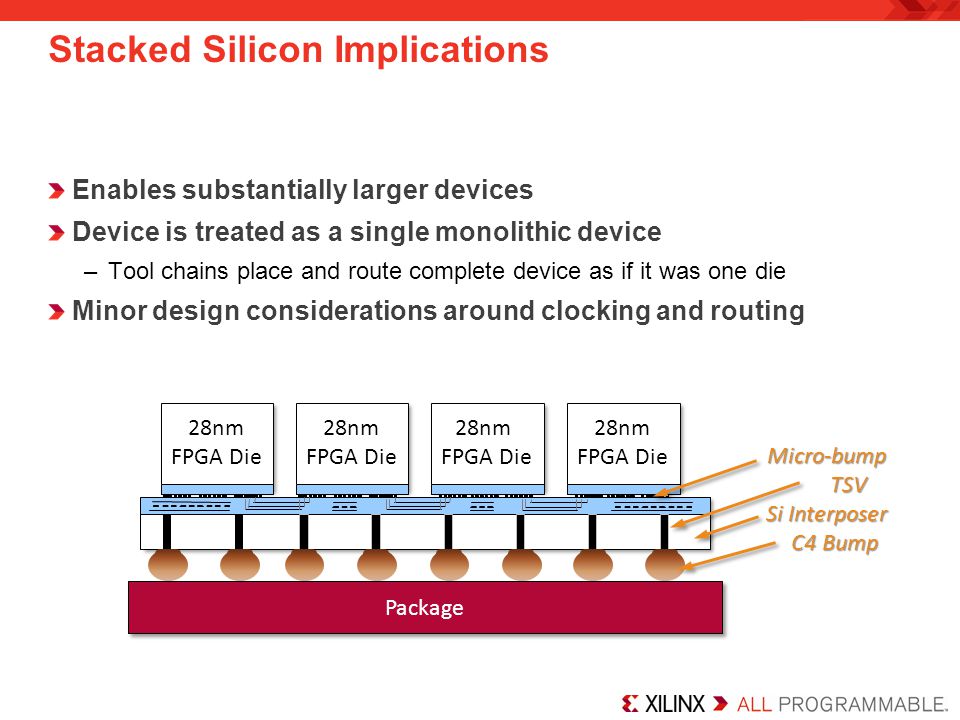
IO and CMT (Clock Management Tiles) Columns.

Serial Transceiver Columns.

PCI Express.

HSSIO

**Stacked** **Silicon** **Interconnect** **Technology** – Stacks multiple **FPGA**s onto a single chip

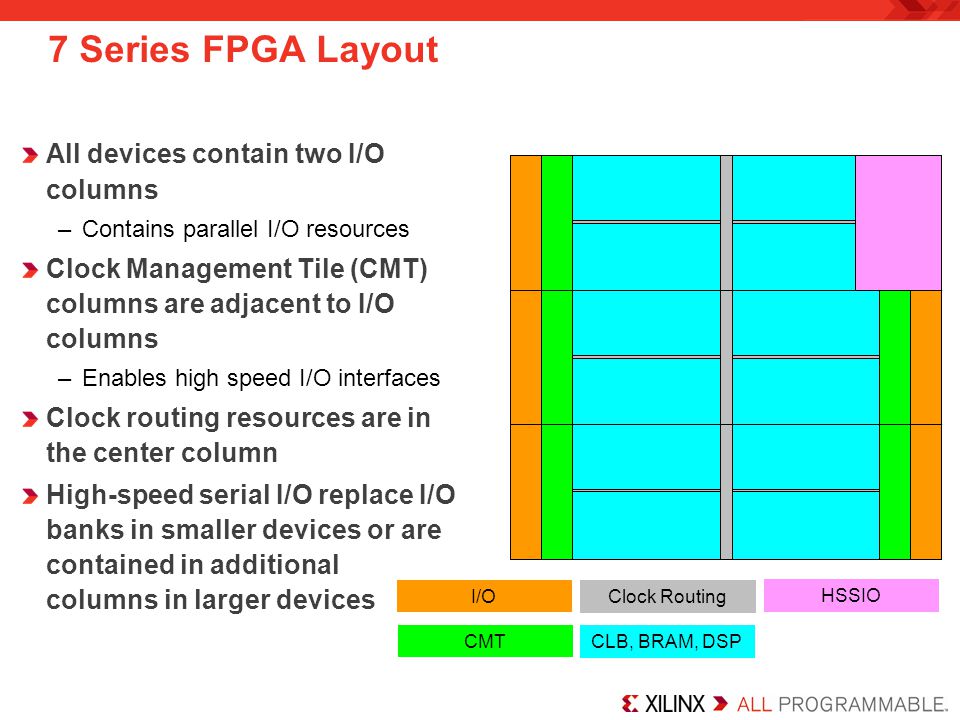


Each FPGA Die subunit is referred to as **Super Logic Region** (**SLR**)

The chip comprised of **SLRs** is referred to as **Silicon Interposer** (**Si Interposer**).

Adjacent **SRL**s are interconnected via **interposers** using Through-Silicn Vias (TSVs).

7-Series SLR layout:



IO columns.

Clock Management Tile (**CMT**) columns:

**MMCM** (clock manager).

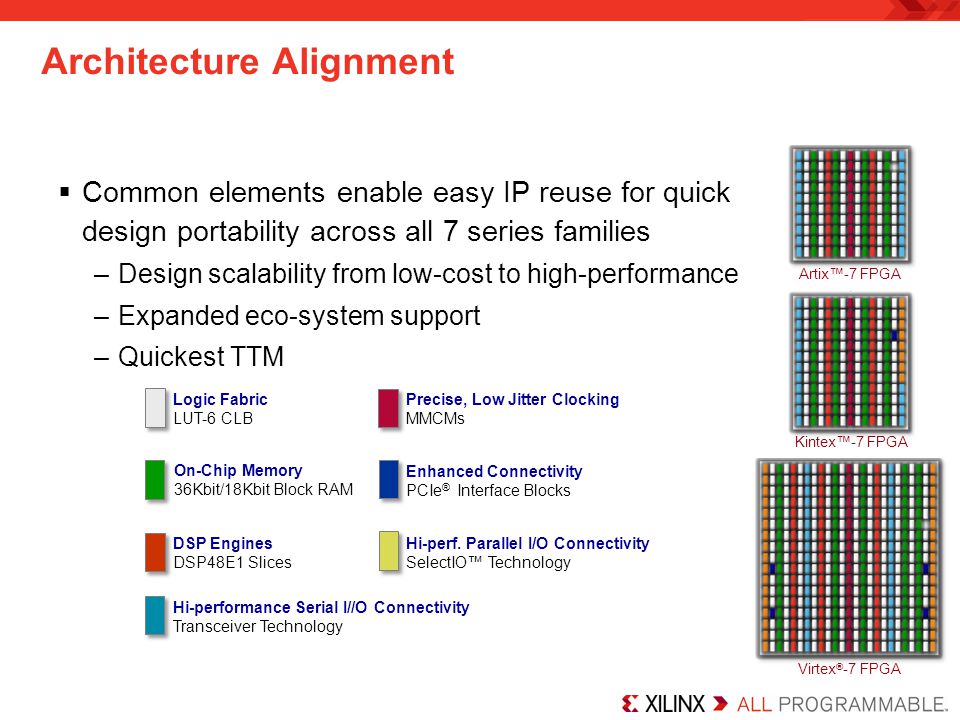
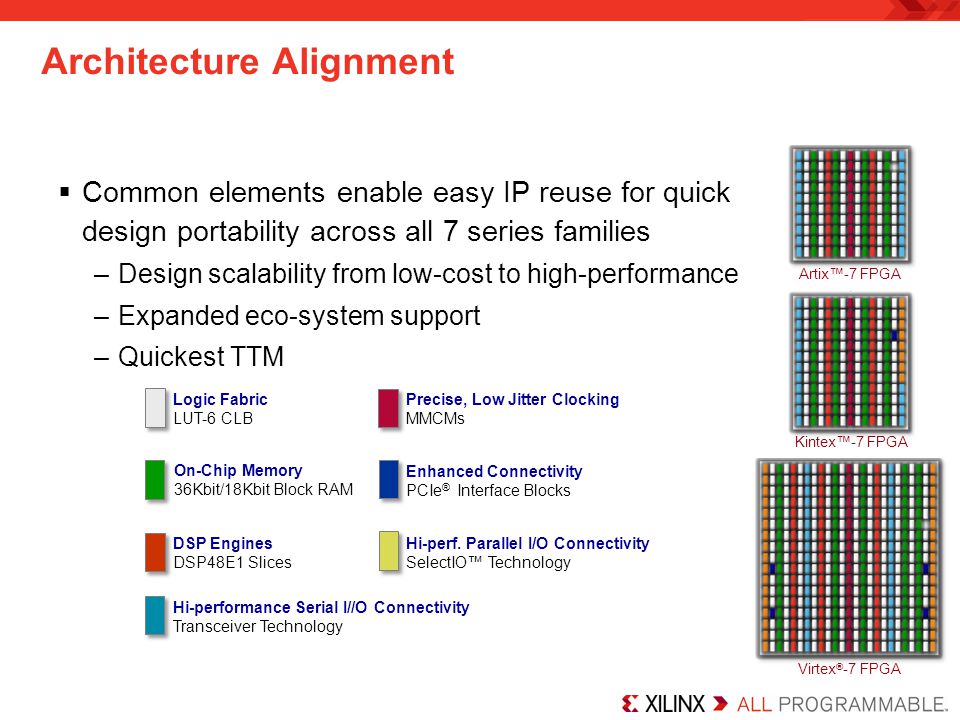
**PLL**.

Clock Routing resources.

**HSSIO** columns (High Speed Source Synchronous Interfaces).

**CLB**, **BRAM**, **DSP** columns.

Architectural Alignment:



Enabled flexible device migration across all 7 series device families by reusing the same components among devices with different layouts.

Clock Regions and IO Banks:



Two types of clocking:

Regional clocking.

Global Clocking.

The boundaries between the Regional and Global clocking is grey area which often overlap.

Clock region aligns with an IO Bank

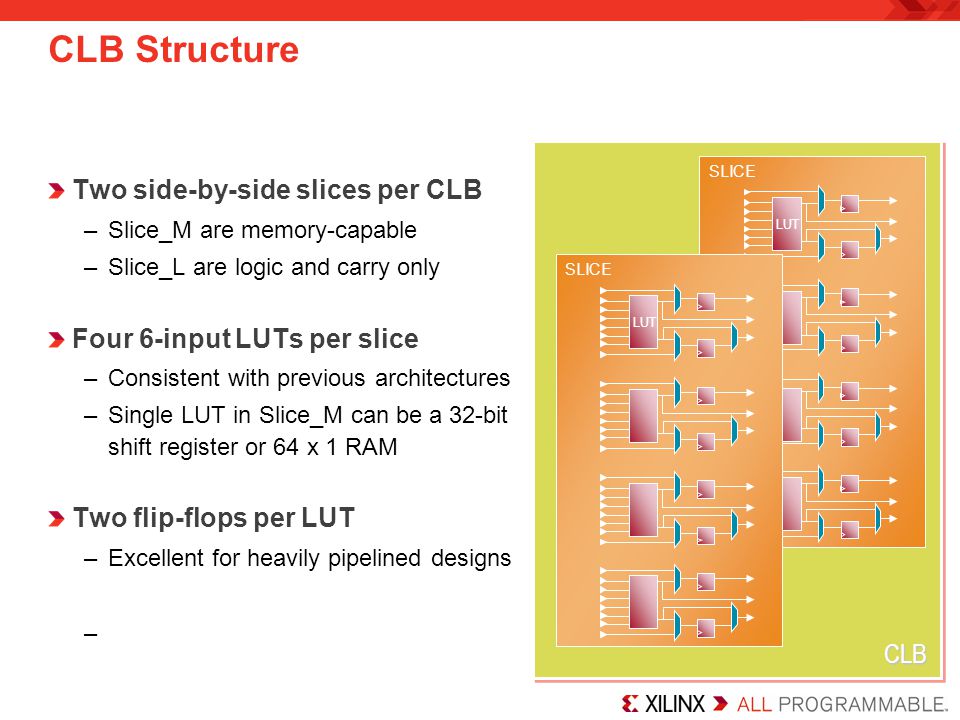
A clock region is about half the chip.

Within a given clock region, you have 50 CLBs and 50 IOBs.

Typical external clock sourcing involves:  
 **Synchronous**: oscillator -> **mmcm** -> multiple clocks with a common clock root.

**Asynchronous**: don’t share the same clock source (handled via fifo, or synchronization circuit).

CLB (Configurable Logic Block) Structure:



CLB contains:

Carry logic.

6-input LUTs.

Registers on the output.

Each CLB contains 2 side by side slices:

A slice contains 4 6-input LUTs.

A slice contains 2 flip-flops per each LUT (8 flip-flops per Slice).

**Slice\_M** (25% of Total Slices; every 4th Slice) – memory capable Slice.

LUTs are user definable as 64x1 RAM/ROM (**Distributed RAM**), or 32-bit shift register.

**Slice\_L** (75% of Total Slices) – Logic and Carry Slice.

Logic functionality only and not to be used as RAM/ROM.

Has carry logic built-in.

Clocking:

Utilizes

CMT (Clock Management Tile; upto 24 CMTs per device) which contains:

MMCM (1 per CMT)

PLL (1 per CMT)

Clock Buffers to support high clock fan out.

Set up for low-skew clock distribution.

Dedicated Hardware:

Transceivers – support various protocols.

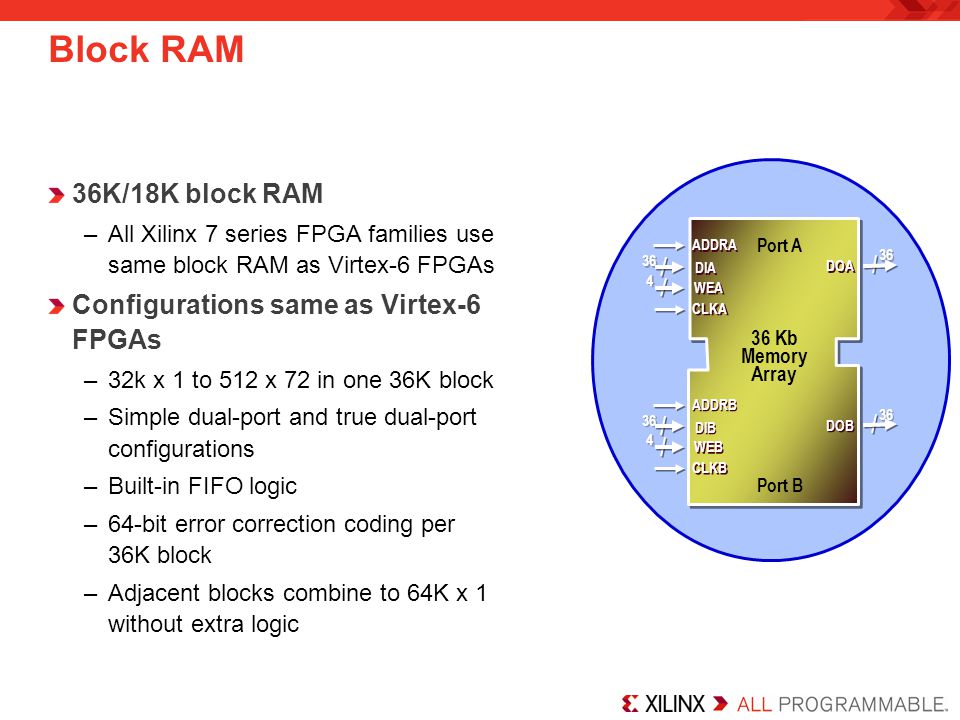
PCIe – various generations and lane widths depending on the FPGA family.

XADC - can be used for internal and external purposes.

**Memory**:

**Distributed RAM** - derived from LUTs of Slices.

**Block RAM:**

****

Large fully functional RAM Cells

Organized into Columns.

Can be size configured as **simple dual-port** RAM or **true** **dual-port** RAM.

Includes hardened built-in FIFO logic implementation.

Includes hardened Built-in **ECC** for 64-bit error correction coding per each 36Kb block.

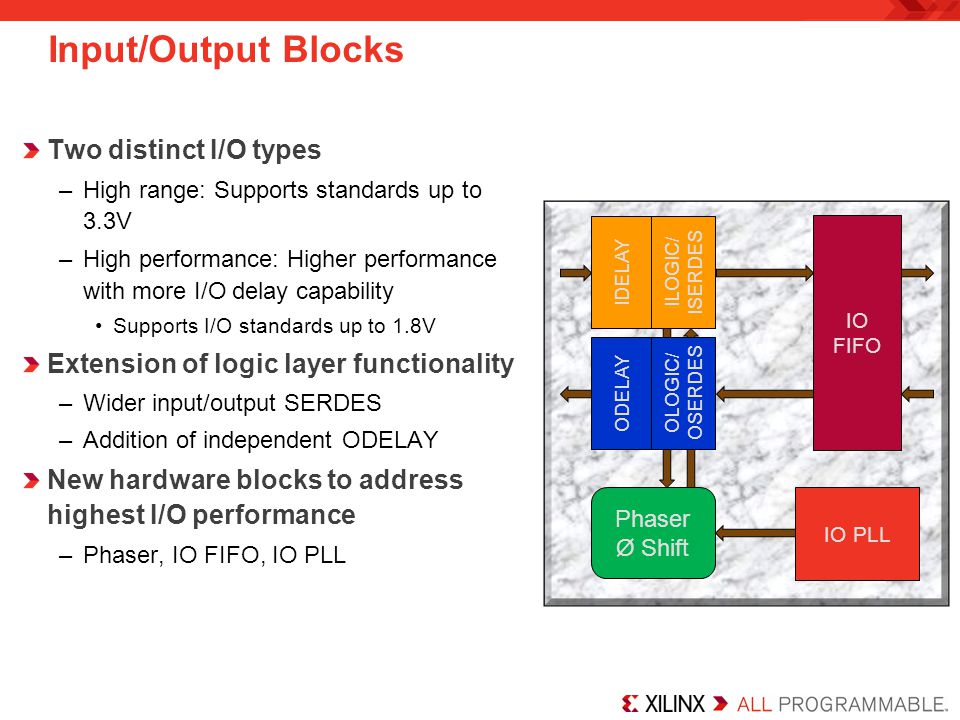
**Single** **bit** **detection + correction**

**Multi bit** **detection**.

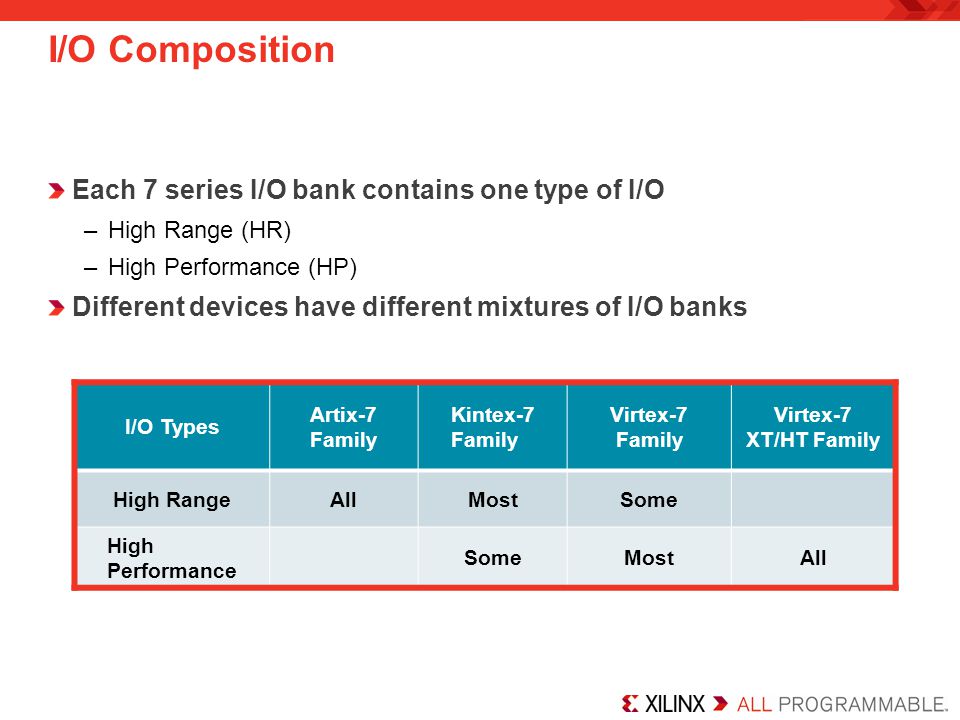
Can be size configured as a **Single 36Kb** **RAM,** or **Two 18Kb** where one is **RAM** and the other one is either **RAM** or **FIFO**.

Each adjacent 36Kb block contains a dedicated cascade logic to

**IO Blocks**:



Two types:



**HR (High** **Range)** **IO**: supports voltages up to **3.3V**

**HP (High** **Performance) IO**: higher performance with more IO delay capability; supports up to **1.8V**

Contain:

Simple Input/Output Buffers.

Registers.

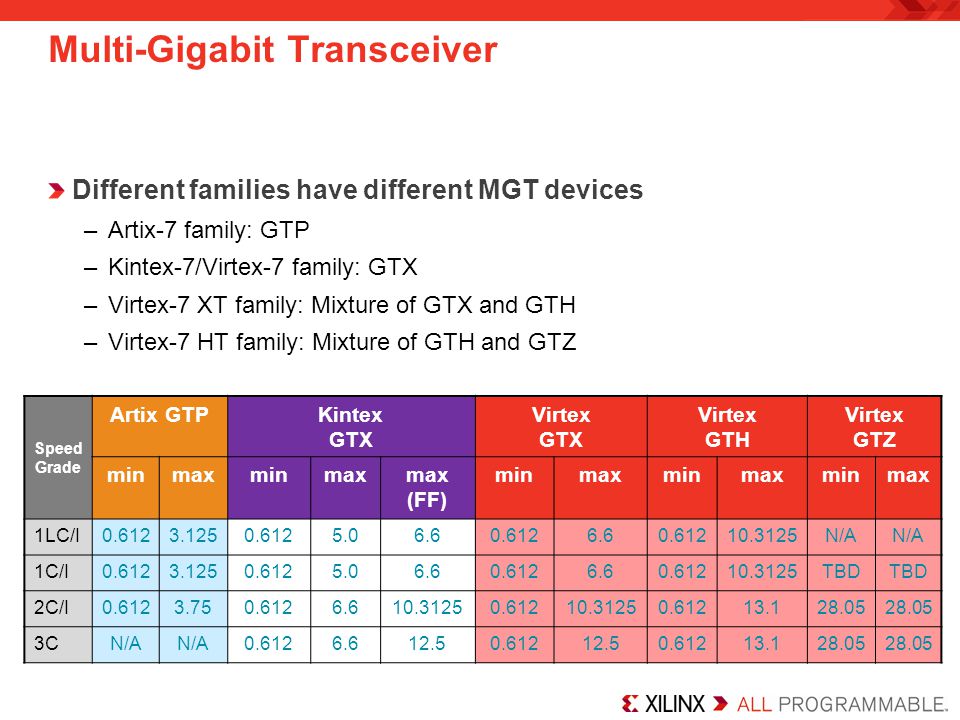
Delay Elements.

PLL.

Phase Shifter.

Variety of Voltage Standard support but with this limitations - bank dependent thus one **REF** and **VCCO** pin per a bank.

**Transceivers**:



**GTP**:

Up to **6.6 Gbps.**

Ultra high volume transceivers.

Wire bond package capable.

**GTX**:

Up to **12.5 Gbps**

Support for most common **10 Gbps** Protocols

**GTH**:

Up to **13.1 Gbps**

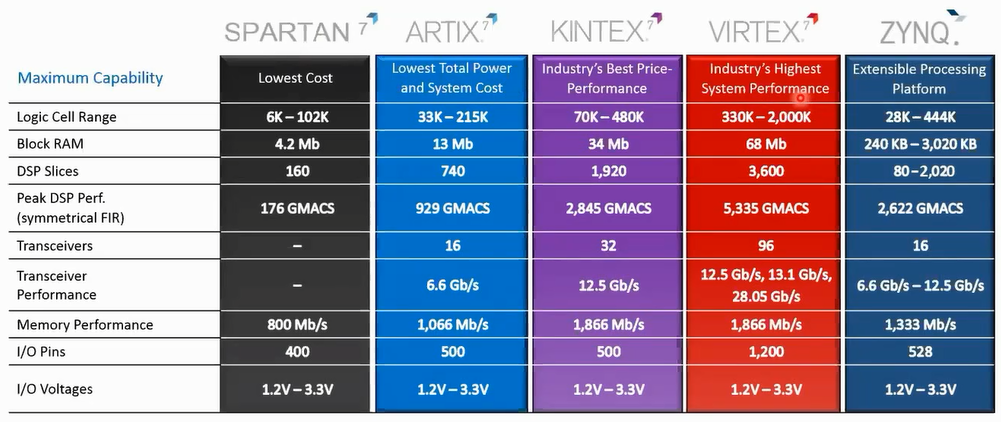
Support **10 Gbps** protocols with high FEC overhead.

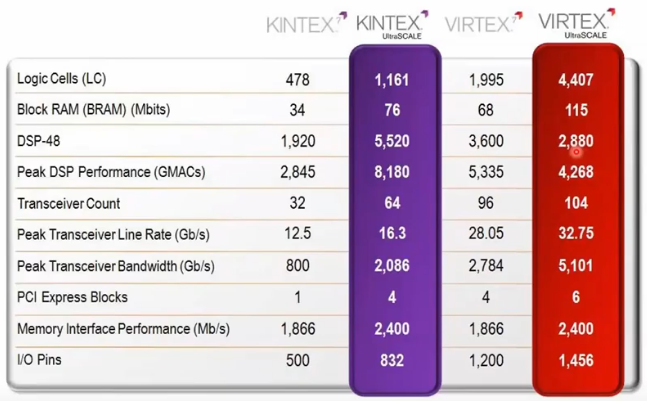
**GTZ**:

Up to **28.05 Gbps**

Enables **100-400 Gbps** communication.

**FPGA Family Comparison**:





**Spartan**:

Lowest price and highest performance-per-watt (power efficiency)

Industrial, automotive, infotainment, motor and motion control

**Artix**:

Lower prices and high performance

Battery powered devices, automotive, commercial digital cameras.

**Kintex**:

Best price/performance ratio.

Wireless and wired communications, medica, broadcast.

**Virtex**:

Highest performance and largest capacity.

High-end wired communication, Test and measurement, Advanced RADAR, High-performance computing.

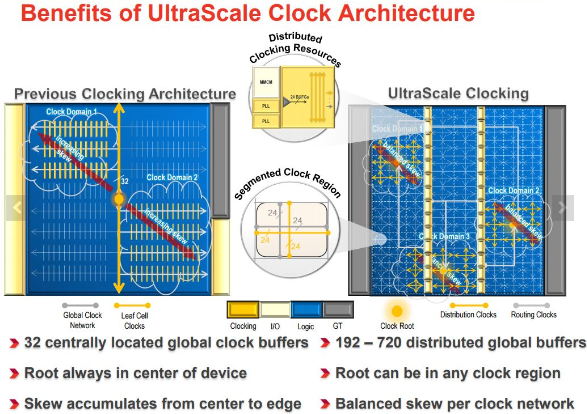
**UltraScale** **Architecture** addresses Interconnect Bottleneck:

**Routing** **delay**: dominates overall delay.

**Clock** **Skew**: consumes more timing margin

**Sub**-**optimal** **CLB** **packing**: reduces performance and utilization.

**UltraScale** **Clock** **Routing** **architecture**:



Previous generations would have clock routed starting in the middle at the single clock root centered in the middle of the chip and fanned out to the rest of the clock region:

UltraScale implements distributed clock network where each clock region has its own clock root in the middle:

**Root** **Clock** can be “instantiated” within each clock region (instead of having a single clock root at the center of the device).

Minimized clock route wire length.

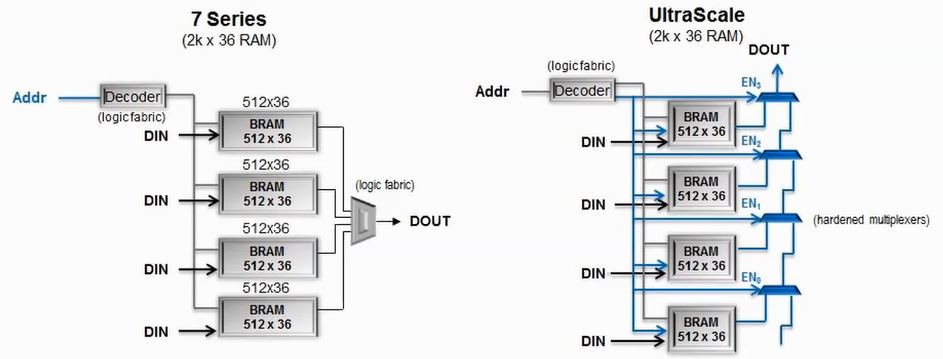
Lowers dynamic power due to global clock net switching.

Balanced clock skew across clock distributions.

100s of **global** **clock** **buffers** with 1000s of placement options.

Flexible global clock placement.

**UltraScale** **Block** **RAM**:



Built-in, high-speed memory cascading:

Eliminates CLB usage.

Reduces routing congestion.

Reduces dynamic power consumption.

Enhanced FIFO:

Lower power and greater performance than soft FIFO.

Asymetric read and write port widths for clock domain crossings.

User-accessible power gating of active BRAM:

Reduces dynamic power when access to block RAM contents is temporarily not needed.

**UltraScale** **Transceivers** (Faster performance with lower power consumption):

GTH:

Up to 16 Gbps.

Enables:

**PCIe** **Gen4** (16G)

JESD204B (12.5G)

CPRI (16.3G)

Serial Memory (HMC & MoSys)

GTY:

Up to 32 Gbps.

Enables:

28 Gbps backplane support for Nx100G to 400G Systems.

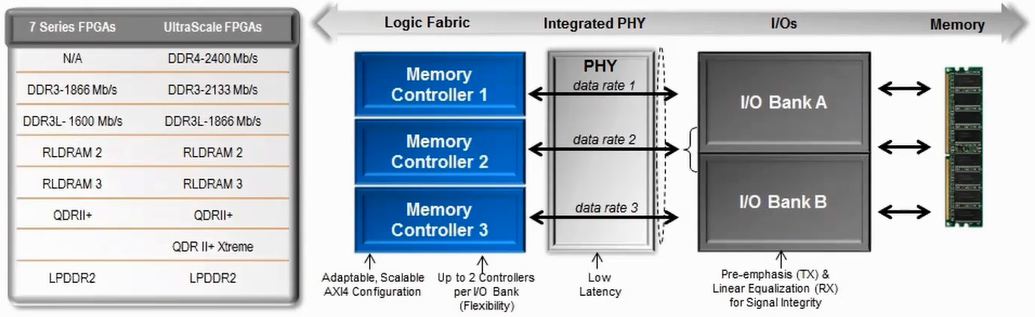
Interlaken.

OTU4 over CFP4.

802.3bj (28G Ethernet Packplane).

Major Power Reduction – 40% lower power for 10G backplanes.

**UltraScale** **built-in Memory Controllers**:



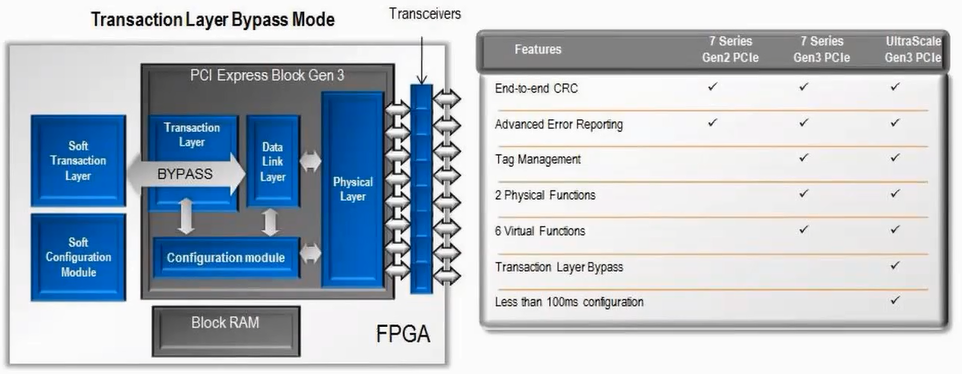
**DDR4** support (up to **2400** Mbps)

Up to **2** controllers per I/O bank

**Integrated** **PHY** – high performance and lower power.

TX Pre-emphasis and RX linera equalization (CTLE)

**UltraScale** **Enhanced PCIe Gen3 Integrated Core**:



Built on existing Virtex-7 XT/HT core.

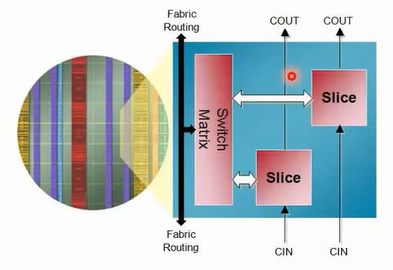
**Gen3** support in all devices and speed grades.

Transaction Layer Bypass mode allows additional physical and virtual functions for diverse topologies.

Integrated circuitry for 100 ms configuration for PCIe specifications compliance.

**CLBs:**

**CLB** **Resources** (7 Series Architecture) Coved:



**Fabric** Routing and **CLB** Arrangement.

Available **CLB** and **Slice** resources in 7 Series FPGAs.

**Distributed** **RAM** and **SRL** (Shift Register LTU) Capability.

CLBs are utilized as primary resources for

Combinatorial functions.

Flip-Flops.

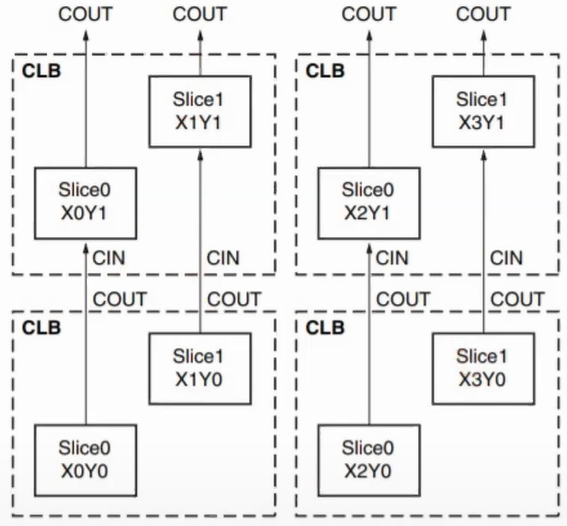
CLBs contain 2 types of slices:  
 **SLICE\_L**

Logic and Carry Slice.

Carry chain runs vertically thru the **Logic Column** from one slice to the next slice.

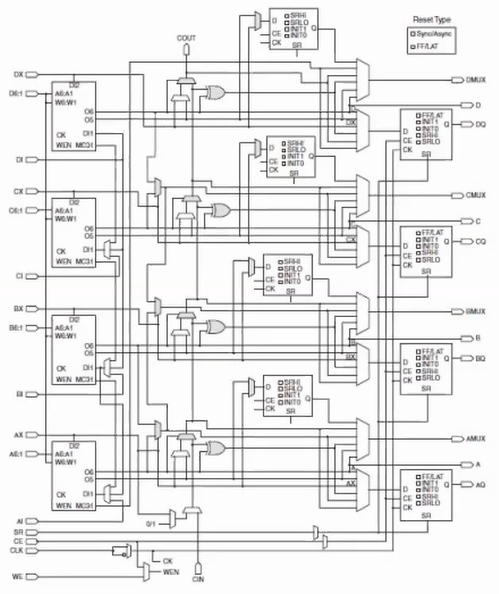
**SCLICE\_M** – Memory capable slice (Distributed Memory)

Floor Planning layout:



Notice from the figure above how each CLB can be referenced by the Y coordinate of the Slice and each Slice within the CLB can be referred to by both the CLB’s Y coordinate and the Slice’s X coordinate.

**Slice details**:



4 6-input LUTs

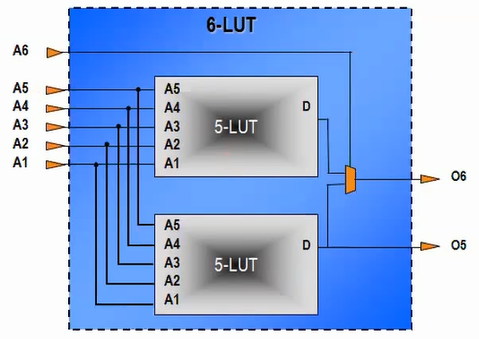
Wide multiplexers

Carry chain

4 flops/lateches per slice

4 additional output flops per slice

**The 6-input LUTS details:**



Implemented as 2 mux’d 5-input LUTs with shared inputs

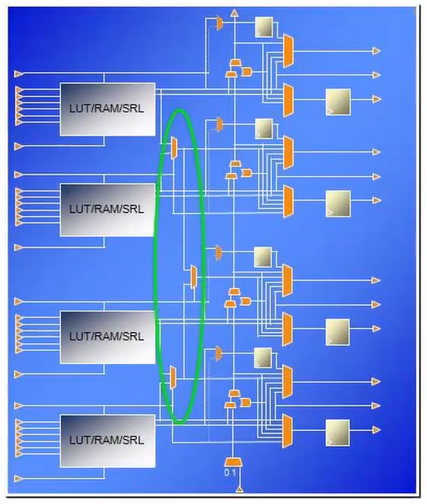
One or two outputs

Can be used as either:

a single function with up to 6 variables

two independent functions with up to 5 variables.

**Wide Multiplexers:**



**LUT (Qt:4) -> F7MUX (Qt:2) -> F8MUX (Qt:1)**.

**F7MUX** (combines the outputs of the LUTs):

Can be used to implement a function with up to 7 inputs.

Can be used to implement an 8-1 multiplexer.

**8MUX** (combines the outputs of the F7MUXs):

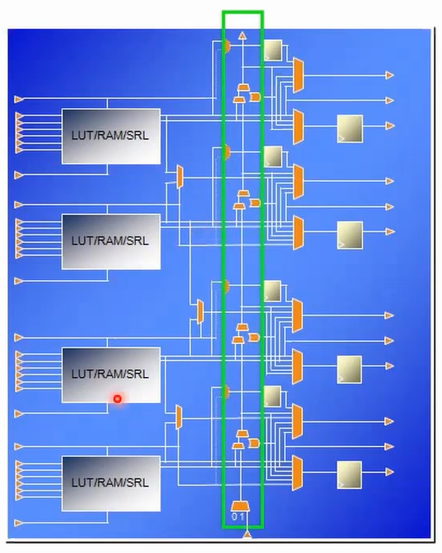
Can be used to implement a function with up to 8 inputs.

Can be used to implement an 16-1 multiplexer.

MUX is controller by the BX/CX/DX slice input.

MUX output can be driven out combinatorially or to the flop/latch.

Carry Chain:

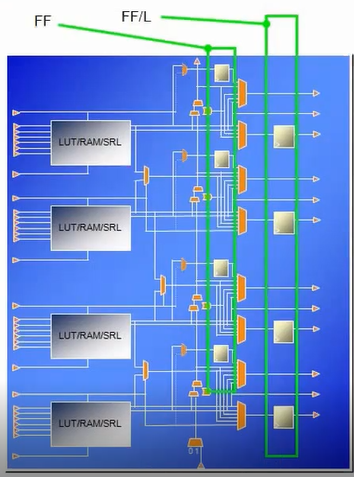


Dedicated logic used for **addition**/**subtraction**.

Propagated vertically thru the 4 LUTs **in a slice**, and from **slice to slice** in the same column in the **CLB** above.

Implements combinatorial carry look-ahead over the 4 **LUTs in a slice**, and implements faster carry cascading from **slice** **to** **slice.**

Flip-Flops and Latches:



Each Slice has two sets where:

FF/L (Flip-Flop/Latch) Register:

The 4 flops at the output of the slice can be configured as either Flop or Latch.

The FF/L Flop’s D input comes from either:

O6 output of the LUT (LUTs have 2 outputs; O5 and O6 as previously explained)

Carry Chain

Wide Multiplexer

AX,BX,CX,DX Slice input.

FF (Flip-Flop) Register:

The 4 flops to the left can only be used as flops and available only when all of the 4 flops above are configured as flops and not latches.

The FF Flop’s D input comes from either:

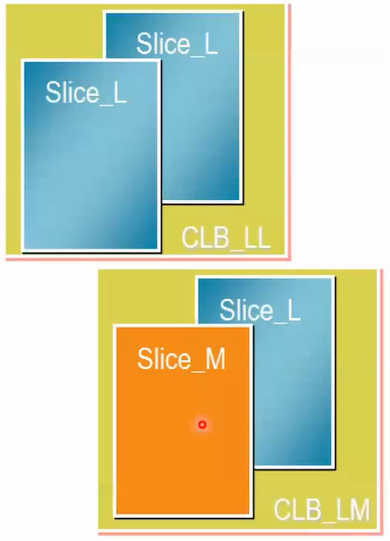
O5 output of the LUT

AX,BX,CX,DX Slice input.

NO Carry Chaine, Wide MUX, or Slice Inputs???

Note: inferring latches will make the FF Flops unavailable!

Slices:



Two types:  
 **SLICEM** (25% of all Slices): Full Slice with wide multiplexers and Carry Chain and an LUT that can be used for:

Logic.

Arithmetic.

Memory (Distributed SRAM).

SRL.  
 **SLICEL** (75% of all Slices): Slice with wide Multiplexers and Carry Chain and LUT that can be used for:

Logic.

Arithmetic.

**SLICEM as Distributed** **SRAM**:



Utilizes the LUTs.

Synchronous Write, Asynchronous Read (can be made synchronous by utilizing the output flop).

Configurations supported:

Single port

1 read/write port???

One LUT6=64x1 or 32x2 RAM

Cascadable up to 256x1 RAM

Dual Port (D):

1 read/write port + 1 read-only port.

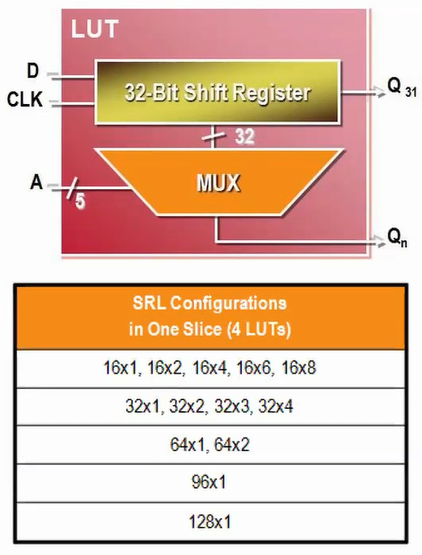
Simple Dual Port (SDP):

1 write only port + 1 read-only port.

Quad-port (Q):

1 read/write port + 3 read-only ports.

**SLICEM as 32-bit Shift Register**:



Can be utilized as:

Variable-length shift register.

Synchronous FIFOs

Content-Addressable Memory (CAM)

Pattern generator.

Compensate for delay/latency.

Shift Register depth is determined by the address width.

Constant value for fixed delay line.

Dynamic Addressing for elastic buffer.

Cascadable up to 128x1 shift register in a single slice.

Shift Register **LookUpTable** does not use a **RESET**!!! Do not use reset in code!!! Reset can be added at the output flop (not recommended).

Example of LUT FIFO efficiency over Flopped FIFO:



**Slice Resources Utilization Methods:**

**Inference**: the synthesis tool does all the work for you based on your code.

**Instantiation**: instantiate the Slice resource by using the name of the primitive and manually connect the ports and set the attributes.

**IP Core Wizard**: IP Core Instantiation using IP Catalog Wizard.

**Slices Summary:**

All slices contain **4 6-input LUTs** and **8 registers**.

Slices contain **carry logic**, and **MUXF7**/**MUXF8** **multiplexers**.

**LUTs** in **SLUCEM** can be used as **32-bit shift registers** or **64-bit memories**.

Slice resources are **inferred** by synthesis tools, but also can be instantiated/accessed via **IP** **Catalog**.

**HDL Coding Techniques:**

Objective:  
 How using Control Signals (set,reset, clock\_enable) in your code impacts FPGA implementation.

Xilinx recommendations on resets, and other recommended coding techniques.

Inference vs instatiation.

**Control** **Signals**:

* Flip-flops have 3 control signals (a.k.a. a **Control** **Set**; look up hdl attributes for **SR**s, and **CE**s!!!)

**CK**: Clock.

**CE**: Clock Enable

**SR**: Asynchronous/synchronous Set/Reset (active high); XILINX does NOT allow use of both **sets** and **resets** – you must choose one!

* Minimize the **Control** **Signal** usage - the fewer of the control signals are utilized in the design the easier it is on placement!
* **CK** (Clock) and **Asynchronous** **SR** (Set/Reset) are always connected to FF control ports, and cannot be moved to the **datapath**.
* **CE** (Clock Enable) and **Synchronous** **SR** (Set/Reset).
* If most **FF**s in the Slice utilize the same type (**Synchronous**) of Control Signals (Sets/Resets), the **CE** and **Synchronous** **SR** are connected to the **FF** **Control** **Ports**.
* **Asynchronous** **SR** > **Synchronous** **SR**:

When tool is deciding which Control Signals connect to FF Control Ports and which to the datapath (**LUT** inputs) and **FF**s in the **Slice** utilize different types (some **Synchronous** others **Asynchronous**) of **Control** **Signals**, the **Asynchronous** **SR** have priority over **Synchronous** **SR** therefore when mixed, the **Asynchronous** **SR** will be connected to dedicated **FF** **Control** **Ports** while **Synchronous** **SR** will be routed via **datapath** (**LUT** inputs).

* **Resets**: two types of resets:
* **Global** **Reset**:
* If every synchronous block of your design is driven by the same reset, a second **global** **reset** is inferred by your HDL!
* If every signal in the design is initialized to a default value, your design may not need a second reset right after configuration is performed; XILINX recommends to consider **not** using a second **global reset**. Instead, utilize **Local** **Resets** as needed. Especially true if design is not expected to be **reset** on the run and having a **reset** occur means the system has been power cycled.
* Automatically occurs upon completion of FPGA initialization during programming process.

1. By default, initializes FFs to 0, but can be set to 1 per FF when default values are provided in HDL.

* The Global Reset net is user accessible via a global set/reset (GSR) port from the Startup component.

1. Seldomly needed, and only used when you want to perform a **Global** **Reset** a second time.
2. When the Global Reset is coded into HDL, it is a second reset is what actually being implemented on the device.

* **Local** **Reset**:
* Internally generated (??).
* Targeted reset.
* Often a standard part of components behavior (FSM, counters, etc)
* **Synchronous** VS **Asynchronous** **Set**/**Reset**:
* Asynchronous Resets:
* Synchronous de-assertion is required to prevent metastability condition!
* Use a **reset** **bridge** (back to back FFs) to provide an **Asynchronous** **Reset** with a **synchronous** **de-assertion**.
* Synchronous Set/Reset:
* Makes FPGA design **more** reliable, **more** predictable, **more** stable, and **less** prone to race conditions, **less** susceptible to missing timing, **less** susceptible to runt pulses.
* Simplifies timing constrains.
* DSP48s’ FFs only support synchronous resets; otherwise the synthesis tool cannot infer DSP48s
* Output FFs only have synchronous resets!
* Mixing of **Set**s and **Reset**s on a FF:
* Mixing **Set**s and **Reset**s in your design requires additional logic since
* a FF must utilize LUTs to have both **Set**s and **Reset**s.
* May create extra level of logic on the datapath.
* Do **NOT** **EVER** use asynchronous set and reset: will affect timing and resource utilization.
* Example that would require extra logic to generate single asynch set and asynch reset from 2 different sources:

always\_ff @(posedge reset, posedge set, posedge clk)

if (reset) a <= ‘0;

else if (set) a <= ’1;

else a <= b;

* Coding Recommendations:
* Utilize hard blocks to use **SRL**s, **DSP Slice**s, **Block RAM**s, **FIFO**s, **ISERDES** and to map large register arrays.
* Dedicated resources are Faster than **LUT**s/**FF** (up to 3x faster).
* Dedicated resources consume less power.
* Timing of the Dedicated blocks is already taken care of.
* **DSP48E, FIFO, BlockRAM, ISERDES**
* Turn off the logic Replication synthesis option to reduce your design size, but may complicate **timing** due to resultant increased **FAN** out.
* Control the use of **Clock Enable**s with HDL Code, and utilize **MMCM** to derive multiple, phase aligned clock sources.
* Provide initial values for **FF**, **SRL**, and **RAM** to control how a register powers up:
* The initial value is used during **Global** **Reset**.
* You can still provide a different value to be set during local reset un
* Example:

signal reg : std\_logic := ‘1’;

process (clk) begin

if rising\_edge(clk) then

if (rst = ‘1’) then

Reg <= ‘0’;

else

Reg <= val;

end if;

end if;

end process;

* Inferring Resources:
* Components that can be inferred:
* **Shift Register LUT**s (**SRLC32E**).
* **F7, F8, F9 MUX**s.
* **Carry** **Logic.**
* **DSP** **Multipliers** and **DSP** **Counters**.
* **Global** **Clock** **Buffers** (**BUFG**).
* **SelectIO** (**single-ended**) standard.
* **I/O registers** (**single data rate**).
* **Input DDR Registers** (clocking input on both edges of the same clock using 2 different FFs).
* **Memories.**
* **Global Clock Buffers (BUFGCE, BUGFCTRL).**
* **Complex DSP Functions.**
* Components that can **NOT** be inferred (must be instantiated via **Wizard**, **Template**, or **attribute**):
* **SelectIO** (**differential**) standard.
* **Output DDR register**s.
* **MMCM**/**PLL**.
* **Local** **Clock** **Buffers** (**BUFIO**, **BUFR**, **BUFG\_LEAF**).
* Xilinx Recommends instantiating these resources instead of inferring:
* **BlockRAM**s (recommended to instantiate via IP Catalog)
* **SelectIO** Technology.
* **MMCM** (must be instantiated via IP **Catalog**; non-inferable).
* **PLL** (must be instantiated via **IP Catalog**; non-inferable).
* **IBUFG**, **BUFGMUX**, **BUFGCE**, **BUFIO**, **BUFR**.