DESD - Theory Lectures

Marco La Barbera

2023/2024

This document contains the notes for the *Digital Electronic System Design* course taught by Nicola Lusardi for the 2023/2024 Electronics Engineering class held at Politecnico di Milano.

A big thank you to those who helped me.

This work is licensed under CC BY-NC 4.0

Contents

1 FPGA Overview 5
1.1 CPLD 5
1.2 FPGA 6
1.3 SoC - System On Chip 6
1.4 Logic - LUTs and Registers 7
1.5 I/O Blocks 8
1.6 Connections - Interconnects Resources 8
1.7 Advanced Modules 9
1.8 Market Side 9

2 FPGA Power and Logic 10
2.1 FPGA Power 10
2.2 FPGA Logic 11

3 I/O Resources 17
3.1 Single-Ended vs Differential 18
3.2 Buffer configurations 19
3.3 Impedance Matching 20
3.4 I/O Logic 20

4 Timing 21
4.1 Timing of D Flip-Flop 21
4.2 Timing of D-Latch 22
4.3 Single-Clock D-FF System Timing 23
4.4 Jitter 26
4.5 Clock Domain Crossing (CDC) 26

5 Clock Resources 28
5.1 CMT - PLL vs MMCM 28
5.2 Clock Routing 29
5.3 BUFH 30
5.4 BUFR 30
5.5 Clock Input 30

6 Pipeline 32
6.1 Latency and Throughput improvement 33
6.2 Power Consumption 34
6.3 Optimizing pipeline and well design rules 34

7 Memories - RAM & FIFO 36
7.1 Random Access Memory (RAM) 36
7.2 FIFO 37

8 AXI-Stream Protocol 41
8.1 Protocol description 41
8.2 Common Mistakes 44

1 FPGA Overview

FPGA belongs to a greater family that is the so called "Programmable Logic Device" (PLD).

In this family we find the following kind of decives:

- Complex Proglammable Logic Device (CPLD)
- Field Programmable Gate array (FPGA)
- System-On-Chip (SoC)

Behavior of circuits are described with HDL code, devices assume configuration changing the connections to work as designed.

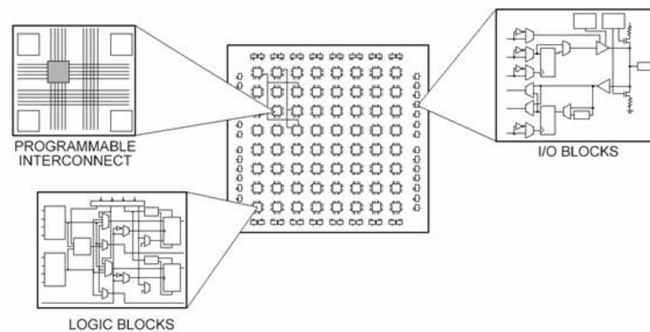


Figure 1: A general PLD configuration

1.1 CPLD

CPLD are the simplest devices of this family. A CPLD device is composed by few and simpler modules:

- **I/O blocks:** Simple Input, Output and Tri-State ports
- **Logic:** LUTs, Registers (D Latch, D Flip-Flop)
- **Connections:** Switching Matrix

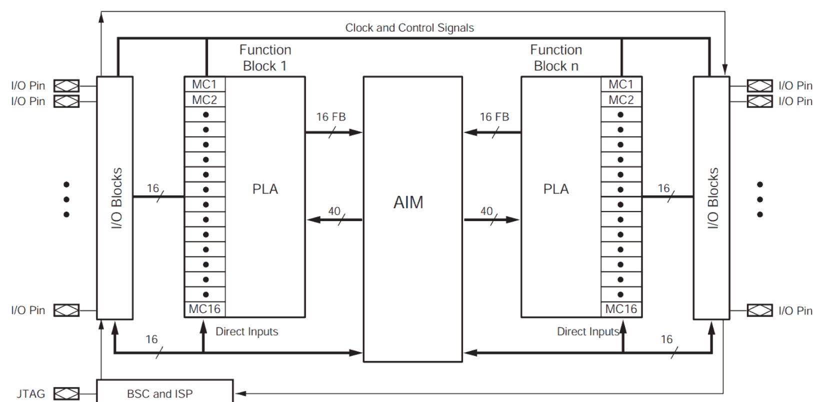


Figure 2: An example of a CPLD architecture

The bitstream is uploaded into a non-volatile FLASH/EEPROM.

1.2 FPGA

FPGA is a more complex family with respect to CPLD. It contains advanced blocks useful for high power computation.

- **I/O blocks:** Input, Output, Tri-State ports and registers for advanced configurations
- **Logic:** LUTs, registers (D Latch, D Flip-Flop)
- **Connections:** Switching Matrix
- **Advanced Modules:** Additional RAM, DSP, PCIe, and so on.

The bitstream is uploaded inside a volatile memory, a static RAM. At each power on, the FPGA has to be re-programmed.

Just before the SRAM, we can find some configuration pins that are used to put the information of the bitstream into the memory. Moreover, some "boot" choices are available to store permanently the bitstream into an SPI FLASH external memory and use it at each power on. For security reasons, the SPI FLASH memory is not inside the FPGA, in-fact, the bitstream is encrypted and it can be decrypted only by an engine just before entering the SRAM. The bitstream is kept encrypted into the SPI FLASH.

Static RAM: Composed by 6 MOS (2 MOS for two NANDS and one MOS for two switches). This Memory is used in FPGA.

Dynamic RAM: Composed by a capacitor and a switch. More compact, more dense, but it has some leakage and memory remains only for some milliseconds, we need to refresh it. So this Memory does not work properly for our purposes because it can't give us a fixed constant configuration.

So, the question now can be: why do we use a volatile memory?

Having the possibility to modify the bitstream at runtime, making "partial reconfiguration", is very useful. As an example, it can be useful for space problems.

1.3 SoC - System On Chip

SoC are temporal computer architectures connected to the FPGA. A microprocessor is used for accomplish some tasks while FPGA is used for other different tasks, an optimum solution can be if you want parallel computation, in which FPGA is very powerful.

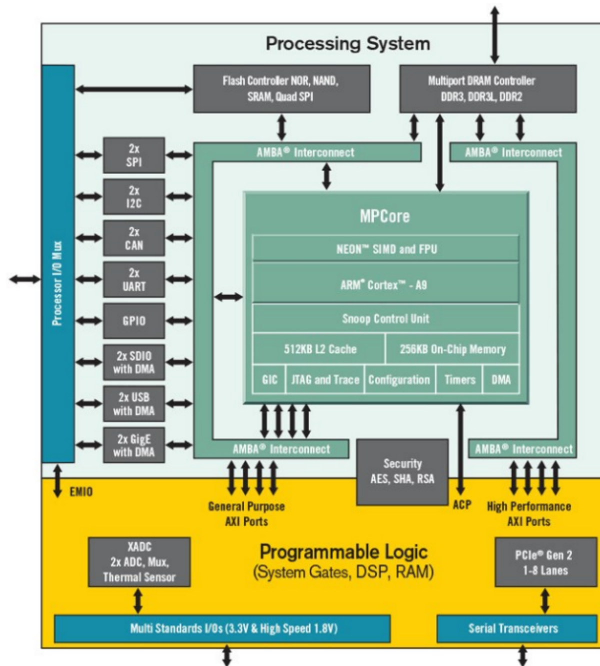


Figure 3: System-On-Chip design example

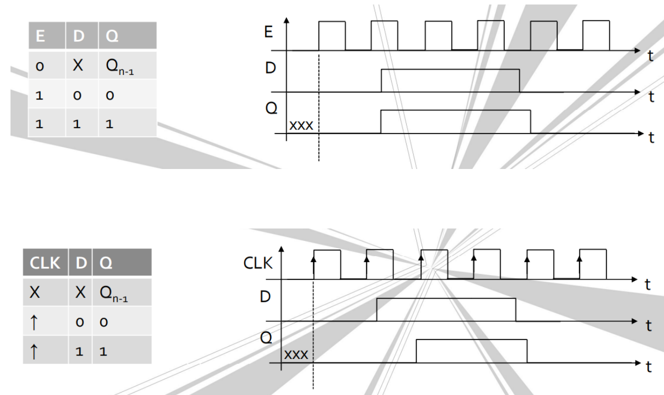
1.4 Logic - LUTs and Registers

LUT changes its configuration considering the bitstream. How is this possible?

The simpler way to build a Look Up Table (LUT) is to implement it using a mutliplexer.

If I want to change Look-Up table configuration from an OR to an AND, I can simply change the input of the multiplexer. In this way it's possible to implement all the logic gates.

The other actors inside the FPGA logic are the registers



E	D	Q
0	X	Q_{n-1}
1	0	0
1	1	1

CLK	D	Q
X	X	Q_{n-1}
↑	0	0
↑	1	1

Figure 4: Latch and Flip-Flop behaviour

Latch: is a register that is not sensitive to a clk but with an enable signal
Flip-Flop: a register where information is stored when **rising** or **falling** edge of the clock

A register could be a flip flop or a latch.

1.5 I/O Blocks

In micro-controllers we have only input and output peripheral. In FPGAs, a I/O block could be an input, a programmable output or a three state port.

Even more, input and output are not simple ports: we can find additional flip flop to performs additional stuffs through these ports.

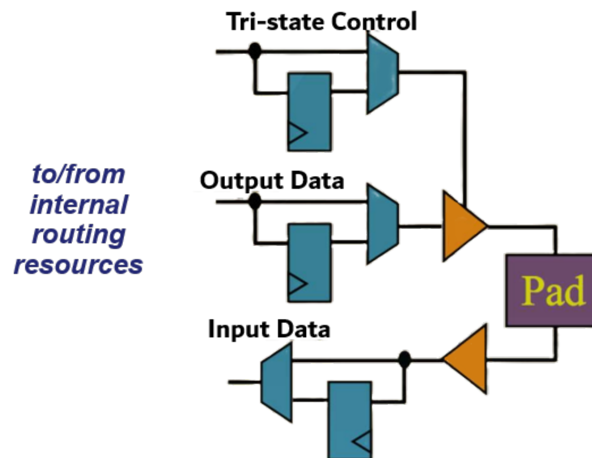


Figure 5: Input and Output pins configuration

Moreover, with **Programmable delay** and **matching impedance** features, we have the possibility to increase speed communication. (See Chapter 3)

1.6 Connections - Interconnects Resources

Interconnect is the programmable network of signal pathways between functional elements within the FPGA. It's the so called **Switching Matrix**, controlled by Programmable Interconnect Points (PIP).

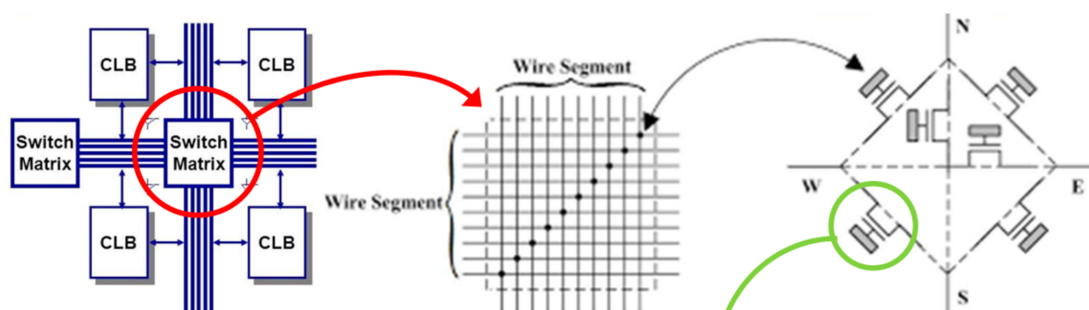


Figure 6: Switching Matrix Design

1.7 Advanced Modules

Differently from CPLD, FPGA has dedicated modules:

- Frequency synthesizer and jitter filter; e.g., Phase Locked Loop (PLL). For instance, if I enter in an FPGA with a 1MHz clk, thanks to a dedicated module it's possible to obtain a 100MHz clk. In CPLD, if I need a 100MHz clk, I must enter with a 100MHz clk.
- Digital Signal Processor (DSP), that are specialized ALUs to perform sum and multiplication.
- RAM grouped in Block (BRAM) or Distributed
- Communication primitive; e.g. PCIe, SERDES, Transceiver

1.8 Market Side

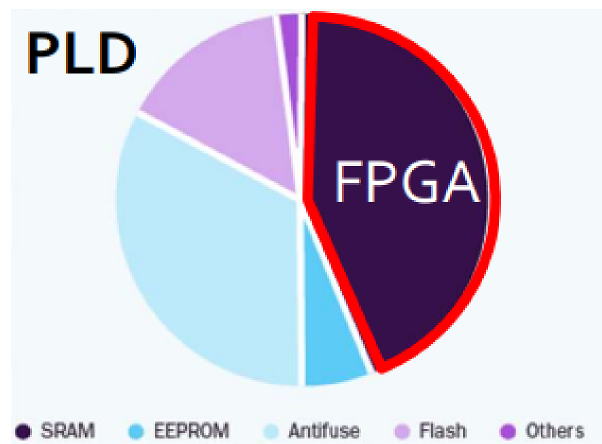


Figure 7: Market percentage on FPGA usage

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM(1)	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance(2)	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU(3)	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	–	16	32	96
Transceiver Speed	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	–	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	–	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

Figure 8: Xilinx 7-Series 28-nm

2 FPGA Power and Logic

2.1 FPGA Power

High level FPGA are very difficult to power-up, instead, low level FPGA are easier.

There are three main issues regarding the powering of an FPGA:

- It requires multi level voltages
- There are several Power-on sequencing
- Generally we take care of Power-Performance ratio

The price of reconfigurability is an higher power consumption. In fact, with respect to ASICs, power consumption is a struggle factor. For instance, we trade the speed of a simple FC-CMOS NOT gate with a slower and a power hungry implementation that use multiplexer in which we have to configure each input.

Let's see the FPGA main Power Lines:

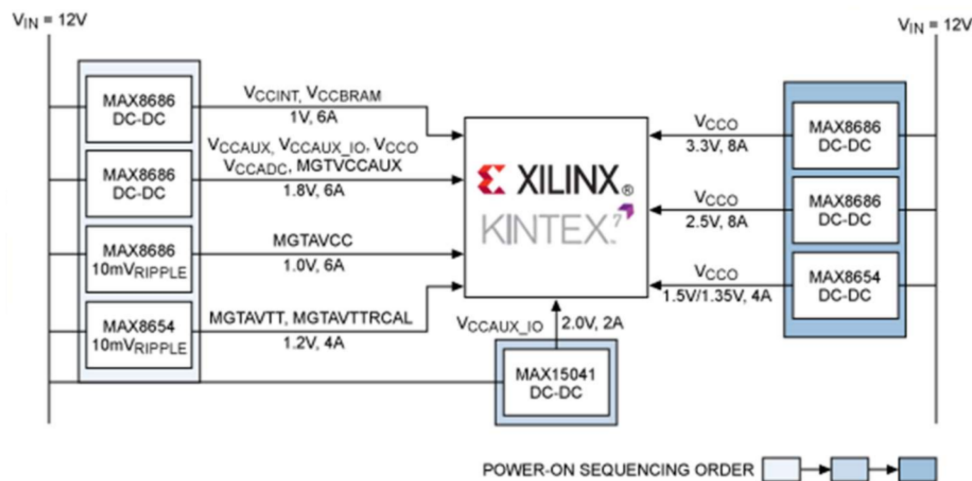


Figure 9: FPGA powering connection design

- **VCCINT:** Used to power-on the core of the internal logic (LUTs, registers), connection (Switching Matrix) and advanced blocks (DSP, RAM).
- **VCCO**: Used to power-on Input and Output Resources, expecially for output buffers.
- **VCCAUX**: Auxiliary power-lines used where there are not VCCINT and VCCO.
- Other power lines: In some FPGA they can be found to power-on very specific modules (external RAM, transceiver) and references (ADC, impedance matching, transceiver).

In a FPGA the static power consumption is 10-20% of the total and it's not 1o1 firmware dependent. Instead, the dynamic portion is more or less 80-90%, and is 1o1 firmware dependent.

With technologies scaling, power consumption decreases due to parasitic capacitances becoming smaller.

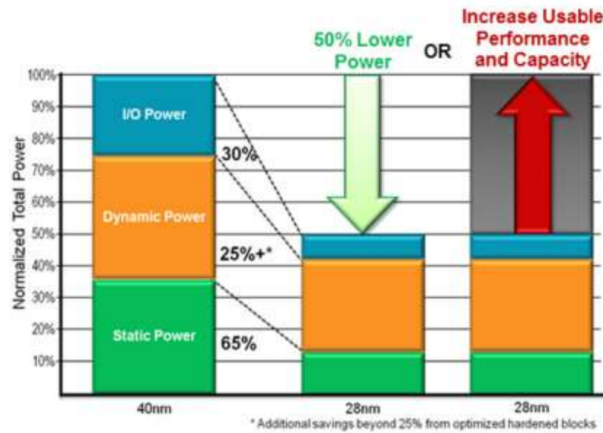


Figure 10: Performance and Power consumption on technology scaling

2.2 FPGA Logic

2.2.1 Configurable Logic Block (CLB)

The fabric of the 7-Series FPGA is arranged in a matrix way, each cell of the matrix is called Configurable Logic Block (CLB). Each CLB is divided into a pair of slices (Slice0, Slice1) that contain LUTs, registers, multiplexer and carry logic. These are only simple logics, without any advanced functionality.

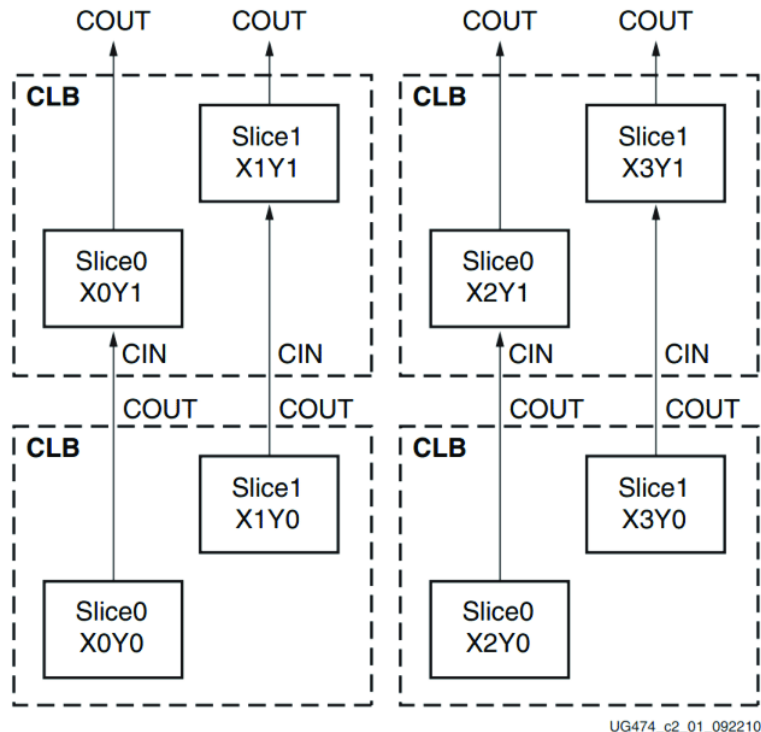


Figure 11: CLB design configuration and connections

The simple logic inside a CLB is composed by:

- LUTs
- Registers
- Distributed Memory and Shift Register
- Dedicated high-speed carry logic
- Multiplexers

CLBs are the main logic resources for implementing sequential circuits.

As it can be seen below, each CLB element is connected to a switch matrix for access to the general routing matrix.

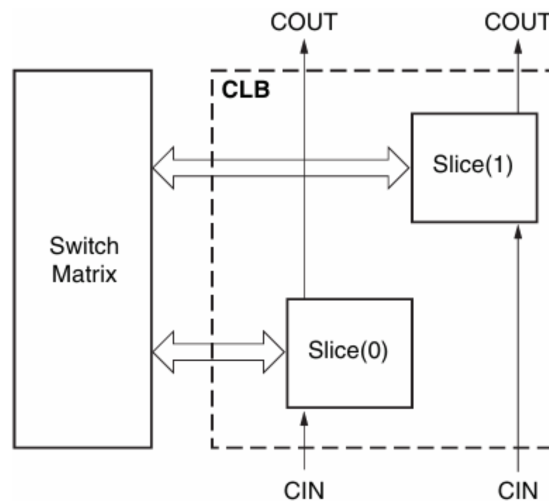


Figure 12: Inside a CLB design

Every slice contains the following logic:

- 4 look-up tables (LUTs)
- 8 storage elements (LATCHs/FFs)
- Multiplexers (MUXs), to move information from LUTs to registers
- Carry logic (CARRY4), dedicated high-speed carry logic

Slices can be **SLICEL** or **SLICEM**. The simplest kind of slice is the SLICEL, that contains only the elements we have seen before. SLICEM, thanks to some bits, supports two additional functions: storing data using distributed RAM, and shifting data with 32-bit registers integrated in LUTs.

Each CLB contains two SLICEL or a SLICEL + SLICEM, Slice0 can be L or M and Slice1 can be L or M too.

- SLICEM structure:

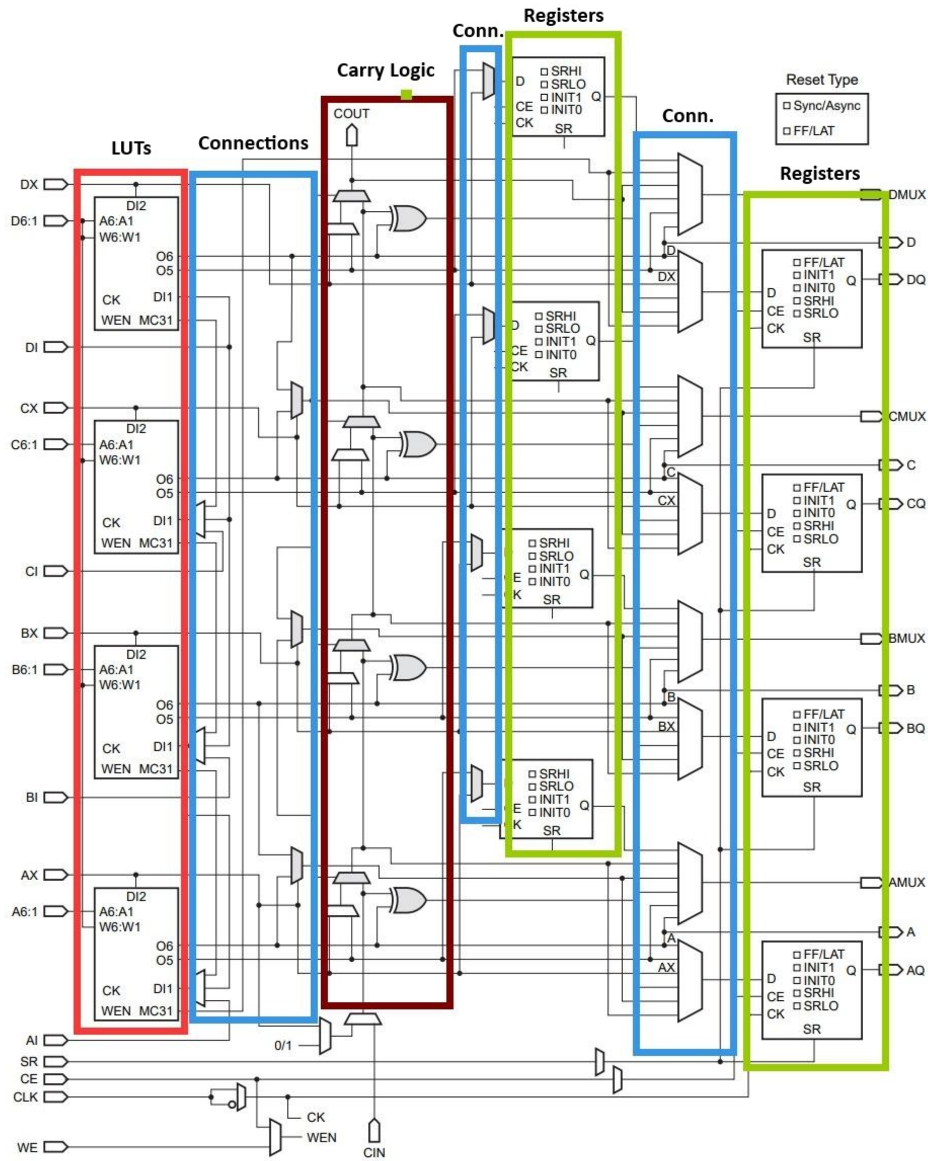


Figure 13: SLICEM design configuration

LUTs can be configured as: Look-Up table, distributed RAM or Shift Register. In fact, LUTs have the 32 bit information that can be configured as a distributed RAM (for this reason is also called LUT RAM), or as a 32-bit shift register.

The **multiplexer** just after the LUTs area are used to move the information from LUTs to the four Half-Adders (4-bit **Carry-Logic**, called "CARRY4" in Xilinx terminology).

The four **registers** of the third column can be used only as D-type Flip-Flop, instead the four of the last column can be used as D-type or D-type LATCH (FF/LAT in the image).

- SLICEL structure:

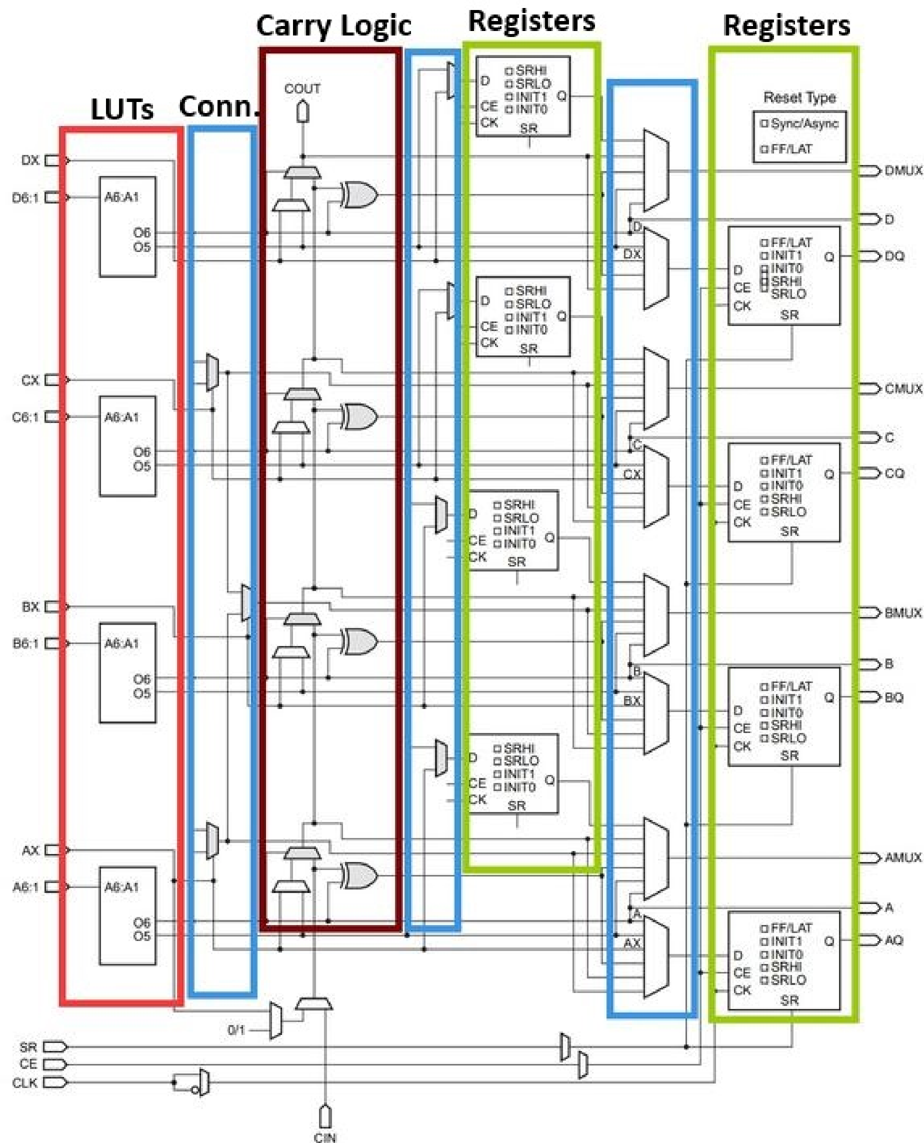


Figure 14: SLICEL design configuration

SLICEL is simpler than the previous slice, the only difference is that now the LUTs block becomes smaller because it has not the distributed RAM and Shift Register functions anymore.

What if we need to use more than one slice?

If a more complicated Logic is needed, (i.e. it requires more than 8 registers, 4 LUTs and 4 adders) the switching matrix comes to play, connecting slices between each other.

In fact, is not possible to connect "directly" any slice0 logic to slice1 logic of the same CLB. To connect them is only possible to use the switching matrix, but these connection are slower than direct connection. Direct connection and switching box connection have a difference more or less of one order of magnitude in terms of time: 100ps vs 1ns.

An ASIC has only direct connection, for this reason is better in timing performance, but it has not flexibility!

The cost in time and resources (Non recursive Engineering Cost, NRE) to design an ASIC is higher than FPGA. In fact, kilo-Units of those chips are needed to make visible this technology. Only in this way ASICs are better.

For FPGA, instead, only one unit is required, and time to develop a design is much less. But FPGA stops to be convenient when more than kilo-Units are sold.

Time To Market for FPGA is low, means "Fast prototyping".

(From professor opinion performance are not very relevant, only TTM and NRE should be considered.)

2.2.2 BRAM - Block RAM

Between the columns of CLBs we can have addition functionality that can be connected to them, the most important element into the Xilinx 7-Series devices is the 36kb **Block RAM (BRAM)**, each containing two independently controlled 18kb RAM blocks.

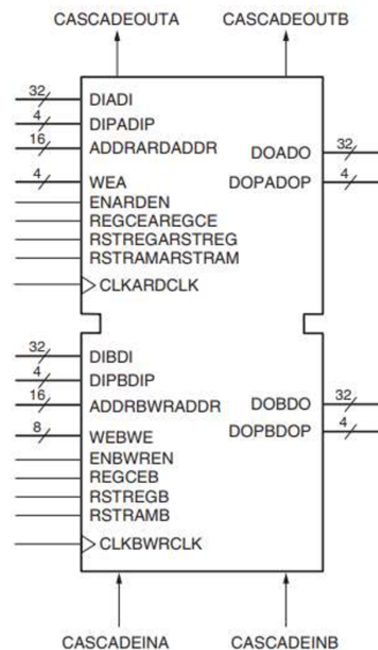


Figure 15: A BRAM memory

The number of BRAMs in the device depends on the specific model, ranging from 5 (180 kb) of the smallest Xilinx Spartan-7 to 1880 (67.68 Mb) of the biggest Xilinx Virtex-7.

Even if FPGA is very big, it has very few Mb of RAM inside. If we want increase the RAM capability, the FPGA has to be connected with an external DRAM (DDR), and we need a proper MIG controller to interface them (Xilinx provides IP-cores to do this).

2.2.3 DSP - Digital Signal Processor

FPGAs are very efficient for Digital Signal Processing (DSP) applications because they can implement custom, fully parallel algorithms.

DSP applications use many binary multipliers and accumulators that are best implemented in dedicated DSP slices.

We can have few or hundreds of DSP per FPGA with the following main feature:

- 25 × 18 two's-complement multiplier
- 48-bit accumulator
- Dual 24-bit or quad 12-bit add/subtract/accumulate

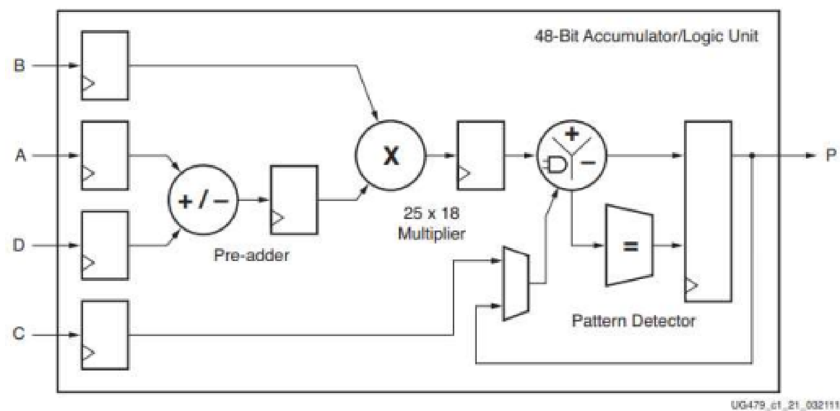


Figure 16: A DSP module

3 I/O Resources

7-Series FPGAs don't have only one VCCO that power-on all the I/O pins, in fact, I/O pins are distributed through different **banks**. Each bank has its own VCCO power supply. The first advantage is the possibility to turn off a specific bank. This structure has been added only for power saving reasons, killing static or leakage power.

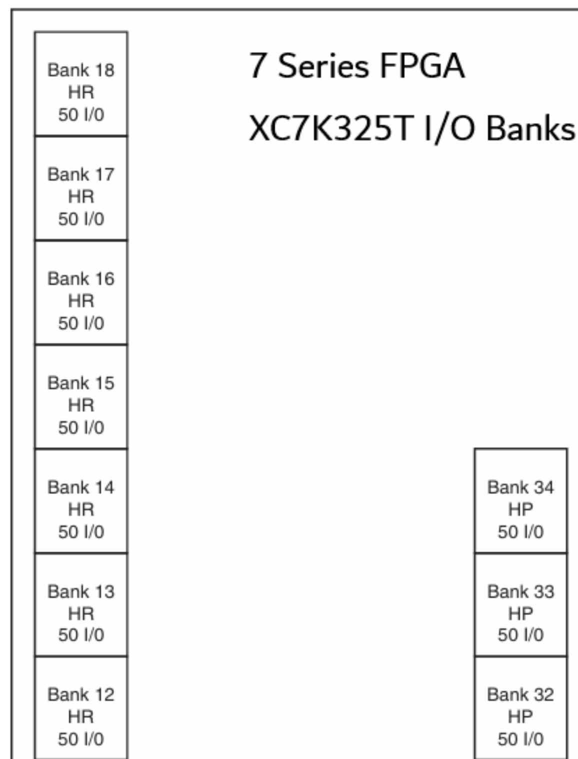


Figure 17: I/O banks

Each bank has 50 pins, and there are two types of them:

- High-Range (HR), has extended voltage level from 1.2V to 3.3V.
- High-Performance (HP), has limited voltage level (1.2V - 1.8V) with better electrical characteristics (impedance matching and jitter)

If we change VCCO, the output high level voltage changes (1.2V / 1.8V / 2.5V / 3.3V are the voltages that can be selected).

3.1 Single-Ended vs Differential

In Single-Ended implementation, the information is brought only by one wire, and it is measured with a common reference as the ground. So, if we use 50 single-ended pins, we can work with 50 different signals.

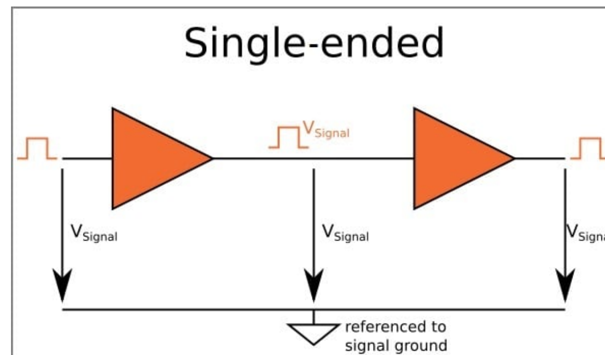


Figure 18: Single-ended configuration

However, this way can produces some problems: single ended signaling is not robust to disturbances. If the disturb signal is higher than Noise Margins, it's possible to confuse logical values.

Instead, with differential signaling, if we have 50 pins we turn out to have only 25 signals.

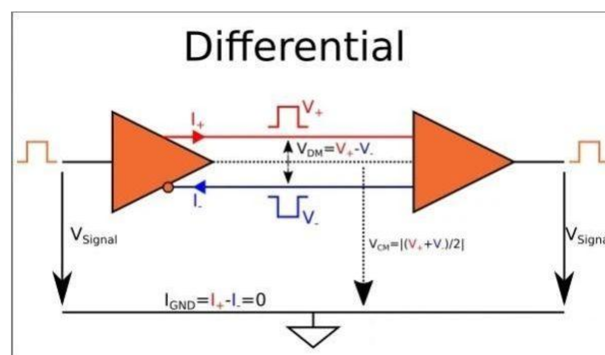


Figure 19: Differential configuration

Now, if a disturb acts, both lines vary and the disturb is rejected. Fully differential signaling helps also in terms of signal integrity, that comes from electromagnetic coupling issues (crosstalk). In fact, using fully differential signaling, it's possible to decrease the coupled area.

3.2 Buffer configurations

In our FPGAs we can find three different buffers:

- IBUF/OBUF
- IBUFDS/OBUFDS
- TRI STATE

The single-ended implementation requires only the simple **IBUF** for inputs, and **OBUF** for outputs. IBUFG is used when an input buffer is used as a clock input.

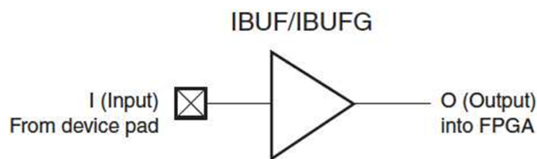


Figure 20: IBUF

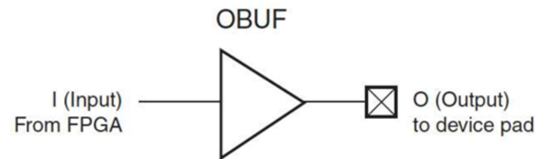


Figure 21: OBUF

Fully Differential implementation requires **IBUFDS** for inputs, and **OBUFDS** for outputs. IBUFGDS is used when an input buffer is used as a clock input.

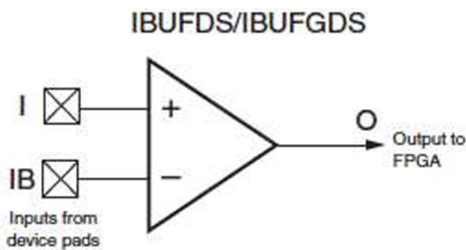


Figure 22: IBUFDS

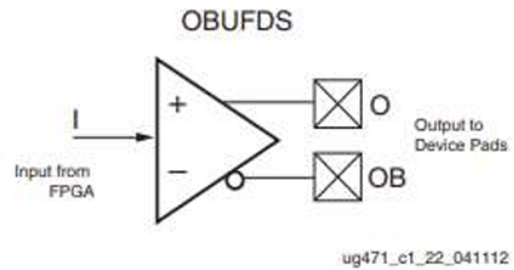


Figure 23: OBUFDS

Using Tri-State Buffer, if we turn on the input buffer, the buffer becomes an input buffer, and obviously if we turn on the output buffer the opposite happens. High impedance means big resistance and big parasitic capacitance, so the τ becomes high and the communication slow.

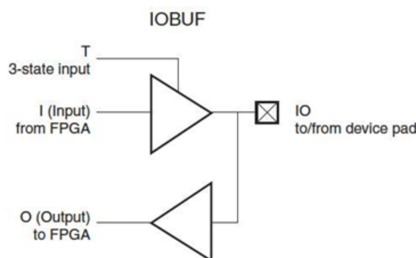


Figure 24: IOBUF

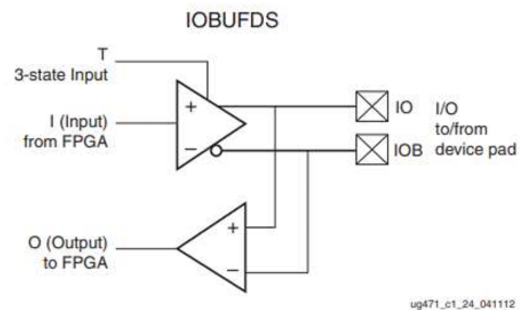


Figure 25: IOBUFDS

3.3 Impedance Matching

Impedance matching is the possibility to activate the output and input resistance in order to match with the transmission line impedance, to avoid reflections.

Moreover, a Digital Controlled Impedance (DCI) present in High-Performance pin, checks the fluctuation of the internal resistance of the FPGA in order to perfectly tune to the desired value.

3.4 I/O Logic

Let's see in details how input/output blocks give more functionality to FPGAs.

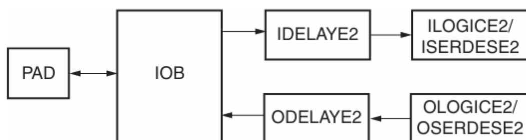


Figure 26: High-Performance I/O pin

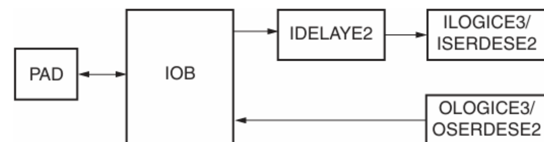


Figure 27: High-Range I/O pin

ILOGIC and **OLOGIC** are Double-Data-Rate or Edge-Trigger input/output D-type Flip-Flop. ILOGIC samples the external pin and put the Q value on the FPGA, vice-versa for OLOGIC.

IDELAY and **ODELAY** permit to program the delay of the signal. This functionality is very important to high speed communication and length matching, in fact, it's possible to compensate skew.

ISERDESE is a dedicated serial-to-parallel converter for data receive with specific clocking and logic features. **OSERDESE** is a dedicated parallel-to-serial converter for data transmission with specific clocking and logic resources.

In addition, we can find FIFOs at the input and output pins that are useful for timing performance.

4 Timing

A pure combinational logic is characterized by a propagation and contamination delay. Sequential logic are characterized by Setup, Hold and Propagation Delay.

4.1 Timing of D Flip-Flop

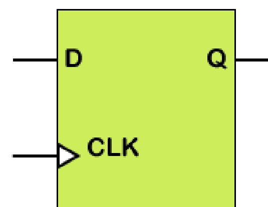


Figure 28: D Flip-Flop

- t_{CQ}: propagation delay D → Q at clock event, the delay we need to write the information to the output
- t_{SETUP}: minimum time D has to be stable before clock event
- t_{HOLD}: minimum time D has to be stable after clock edge

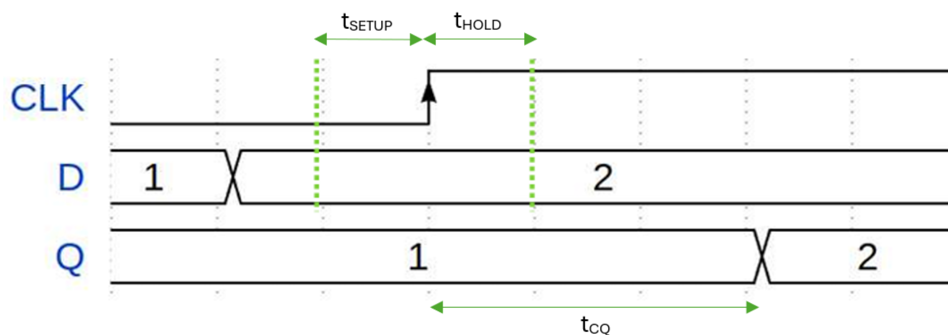


Figure 29: D Flip-Flop timing analysis parameters

If this values are not respected, it can happens that wrong information are showed on output. Moreover, if timing is not respected, an other huge problem called Meta-stability can happens.

If Meta-stability status occurs, it means that the electrical value measured on Q is not an HIGH or LOW value but a value inside the half dynamics. Moreover, the system starts to spend also in term of energy, increasing power consumption.

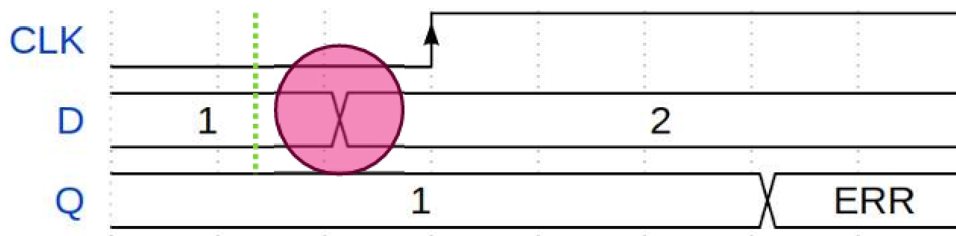


Figure 30: Setup violation

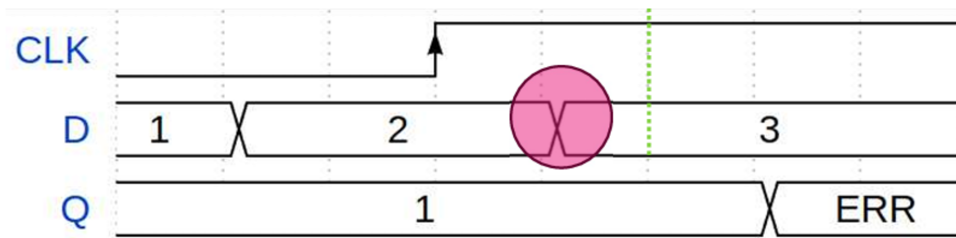


Figure 31: Hold violation

4.2 Timing of D-Latch

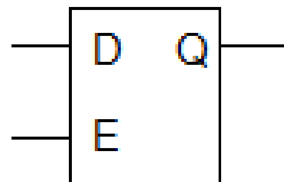


Figure 32: D-latch

- t_{DQ}: propagation delay D $\rightarrow$ Q when transparent (E = 1), the delay we need to write the information to the output
- t_{SETUP}: minimum time D has to be stable before latching (E from 1 to 0)
- t_{HOLD}: minimum time D has to be stable after latching (E from 0 to 1)

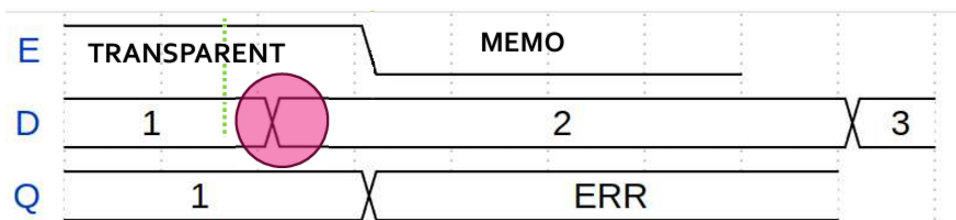


Figure 33: Setup violation

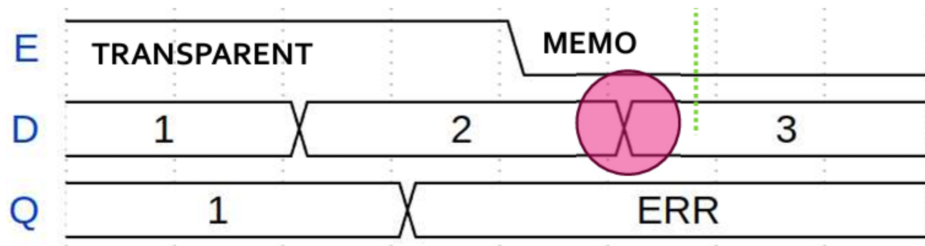


Figure 34: Hold violation

4.3 Single-Clock D-FF System Timing

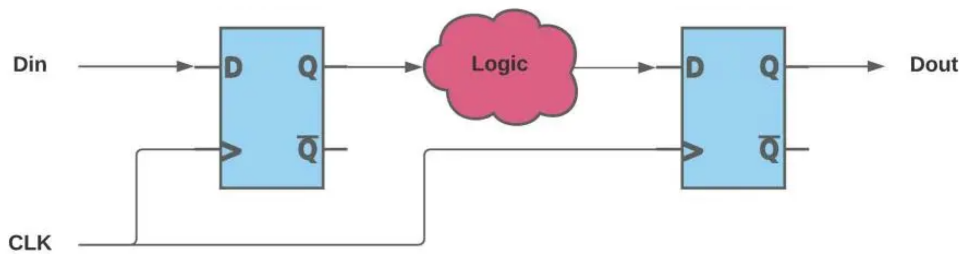


Figure 35: The sequential circuit taken for our upcoming analysis

Timing analysis does not check the source signal Din because it is non-deterministic and its timing behavior is entirely random. It's possible to respect only the timing violation of the dest register.

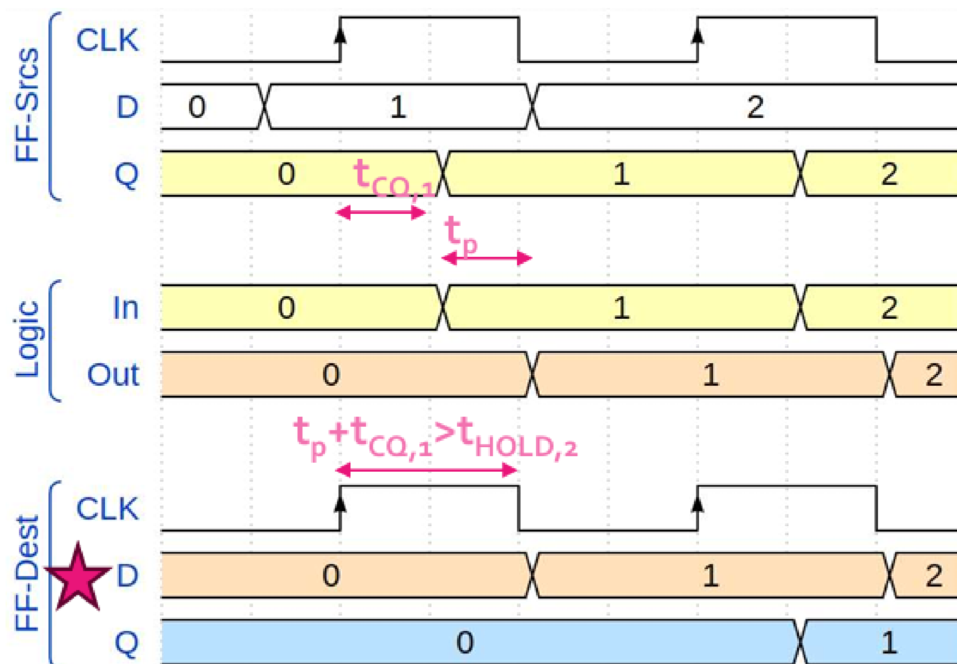


Figure 36: Hold computation

If a '0' is sampled in FF-Dest, FF-Dest needs that this value remains stable at least for a t_{HOLD} before the new information arrives ('1' sampled in the source FF in this case). So, the '1' information in FF-Srcs has to take $t_p + t_{CQ} > t_{HOLD}$ in order to respect hold timing.

We can say that the Hold is related to the "present" pulse of clock.

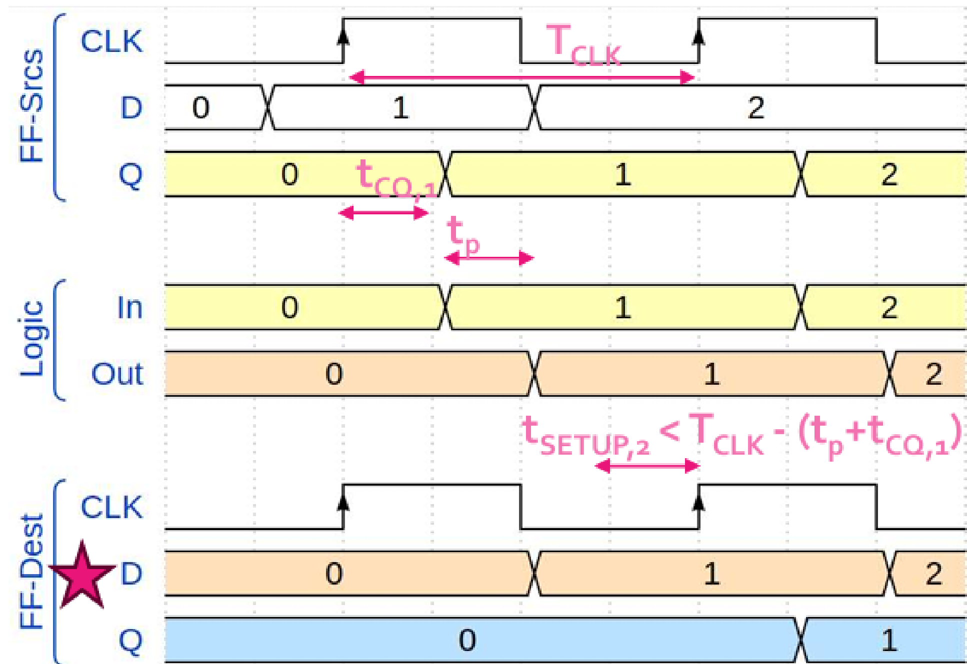


Figure 37: Setup computation

The new value D of FF-Dest has to be stable at least a t_{SETUP} before the sampling rising edge. So, the '1' information, has to arrive before the Setup window: $t_{SETUP} < T_{clk} - (t_p + t_{CQ})$.

If D is too low, we can choose to reduce the t_p of the logic and, if not possible, the only way is to reduce the clk frequency.

We can say that the Setup looks to the "future" regarding the clock pulse.

4.3.1 Forward and Backward Skew

Is not guaranteed that the clk inputs of source and destination are synchronized. A positive (Forward) Skew indicates that, from the source to the destination register, the clk is delayed.

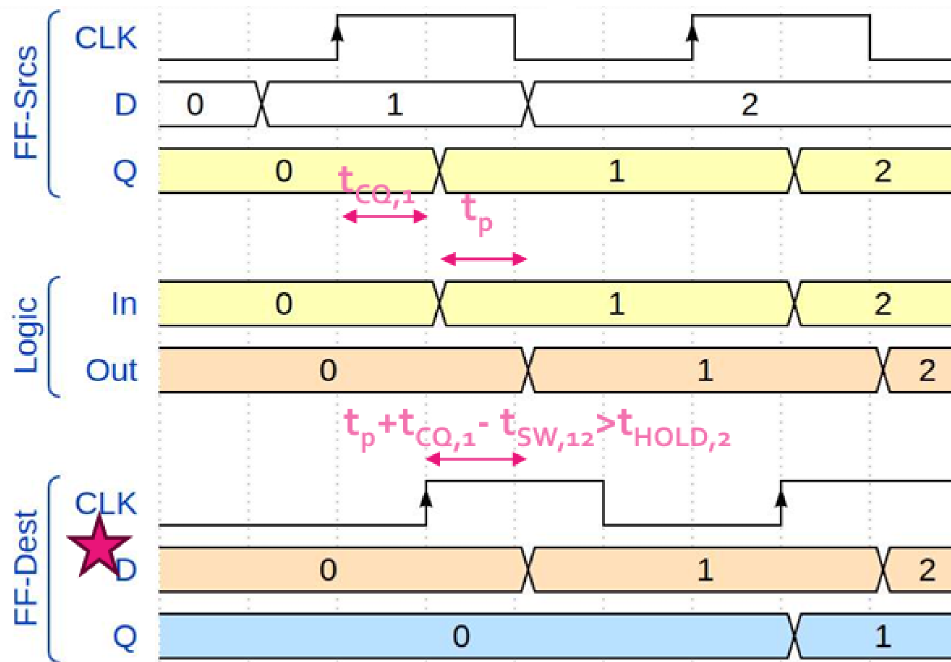


Figure 38: Hold with forward skew

Skew in Hold decrease the allowed window $t_{HOLD} < t_p + t_{CQ} - t_{skew}$ because the Dest clock sample is moved ahead in time.

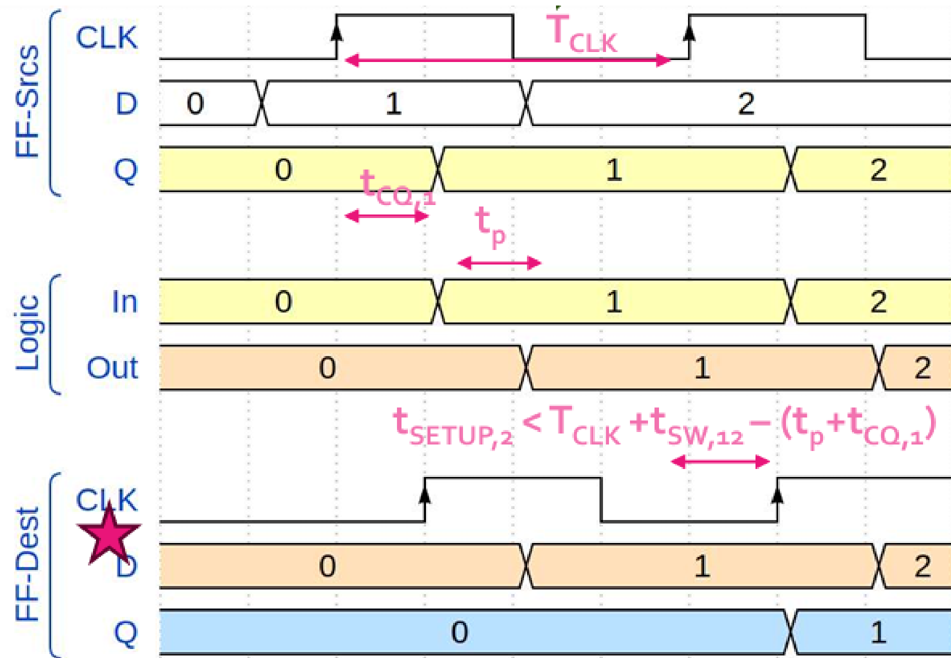


Figure 39: Setup with forward skew

Skew in Setup increases the allowed window $t_{SETUP} < T_{clk} + t_{skew} - (t_p + t_{CQ})$ because the Dest clock sample is moved ahead in time.

Overall, the timing analysis can be performed with these two formulas:

- $t_{SETUP} < T_{clk} + t_{skew} - t_{PATH}$
- $t_{HOLD} < t_{PATH} - t_{skew}$

Obviously, if a negative skew occurs, we have to invert the skew sign and we observe that Hold has a benefit and Setup is worsened.

4.4 Jitter

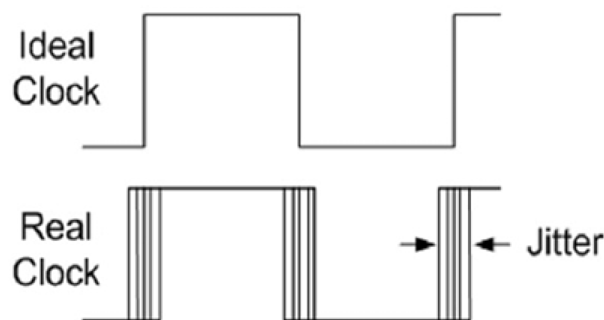


Figure 40: Jitter

The edge of real clocks fluctuate around the ones. This fluctuation is a random variable with null mean value. The standard deviation (tj) of the probability density function is called jitter.

If we consider the real source edge at $t = 0$ the destination has a positive/negative skew of t_j (worst case because it is random!).

Now, our timing equations have to be fixed with this new value. We need to handle the Jitter in a conservative way and, considering the worst situation, we put it into the equation with a negative sign.

- $t_{SETUP} < T_{clk} + t_{skew} - t_j - t_{PATH}$
- $t_{HOLD} < t_{PATH} - t_{skew} - t_j$

Both Setup and Hold are worsened with this choice.

4.5 Clock Domain Crossing (CDC)

If source and destination registers work with different clock domains, a δ between two clocks appears, and acts like a skew that changes for every pulse in a predictable way.

In order to avoid Setup, Hold violation and so meta-stability issues, we can add Flip-Flops in series (typically from 2 to 8) to stabilize the data. However, the most used CDC circuit is the Asynchronous FIFO.

Meta-stability is now avoided, but I can still achieve a wrong value at the output. In order to avoid this issue, we can build a circuit with a TVALID bit that ensure the right value of the data.

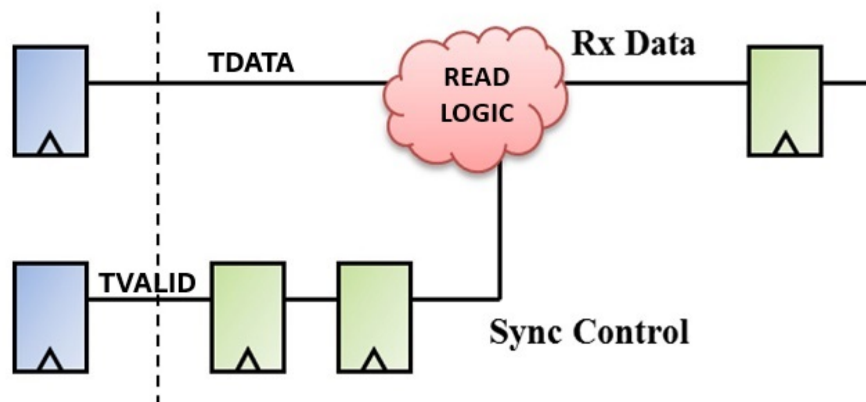


Figure 41: Clock Domain Crossing solution for TVALID in AXIS

A read logic has been built to check when the incoming data is valid or not.

How can we guarantee that no more than one TVALID pulse happens for each pulse of clock of the source? We have to perform a toggle for each time the data is valid.

5 Clock Resources

Work on clock implementations means find the best solution in order to minimize skew and jitter.

As an example, from the top to the bottom of the FPGA, with the direct connection of the CARRY4 modules, there is more or less 2 ns delay. Using the switching box, the time duration is more or less ten time slower, 20 ns. The maximum skew that we have today on our FPGA is more or less of 100 ps. How can we achieve this speed?

If we want to work with clk inside our FPGA, it is not possible to work with direct connection or switching box, but a faster mechanism has to be implemented.

Our goal is to mix circuits at a CLB level. The solution for our 28 nm technology is a "tree" connection that distributes the clock inside the FPGA.

Overall, inside our FPGA, there are up to 32 clks. A number of 50 different CLBs are grouped in a clk region point of view. Each clk region can manage up to 12 clock domains. So, each clock region has 12 different "trees". These 12 global clocks can be driven by any combination of the 32 global clock buffers.

The 7 series FPGAs clocking resources manage complex and simple clocking requirements with dedicated global and regional I/O and clocking resources.

The **Clock Management Tiles** (CMT) provide clock frequency synthesis, deskew and jitter filtering functionality. It is instantiated by the user to solve some issues and is divided in two:

- Phase Locked-Loop, PPL
- Mixed-Mode Clock Manager, MMCM

The **clock routing** and buffers, managed by vivado, allow to propagate clock without skew at low jitter and is divided as follow:

- Global Buffer, BUFG
- Horizontal Buffer, BUFH
- Regional Buffer, BUFR

5.1 CMT - PLL vs MMCM

Each 7 series FPGA has up to 24 CMTs, each consisting of one MMCM and one PLL.

A PLL is a feedback system that takes in input f_{in} and put in output $N \cdot f_{in}$. In the feedback path it has a frequency divider. Overall, we obtain a **frequency multiplier**. It has one input and N different output.

A MMCM can have a fractional number instead of N, so no more just an integer, it takes in input f_{in} and put in output $Q \cdot f_{in}$. It has also a **filter on the jitter**. Moreover, MMCM has **fine phase-shift capability** in either direction and can be used in dynamic phase-shift mode. So we can create a misalignment with a shift between the output and the input.

In both modules, there is a flag that permit us to dynamically change all the parameters (Q, phase-shift and so on). This functionality is very useful when we have to connect an external device for which we don't have information about the delay (it can have an other clk domain).

For instance, in order for handshakes to work properly on a DDR memory, it's needed to match the processor clk with the memory clk, and this can be dynamically adjusted changing parameters at run time.

5.2 Clock Routing

Each CLB is connected, by means of the Switch Matrix, to the global routing network, which is composed by wires running between two Switch Boxes.

While the global routing network offers reasonable propagation delays for a signal, these delays are not acceptable for clock signals, which require instead strict timings to avoid clock skew issues. To overcome them, dedicated clock lines with reduced propagation delays are available in the FPGA, each driven and accessed by a global clock buffer.

5.2.1 BUFG

Each 7 series device has 32 global clock lines that can clock all sequential resources in the whole device. Global clock buffers (BUFG/BUFGCE) are 32 buffers that, thanks to clock routing, receive the clks from all their input pins, and drive the global clock lines in order to distribute this clock in all the FPGA.

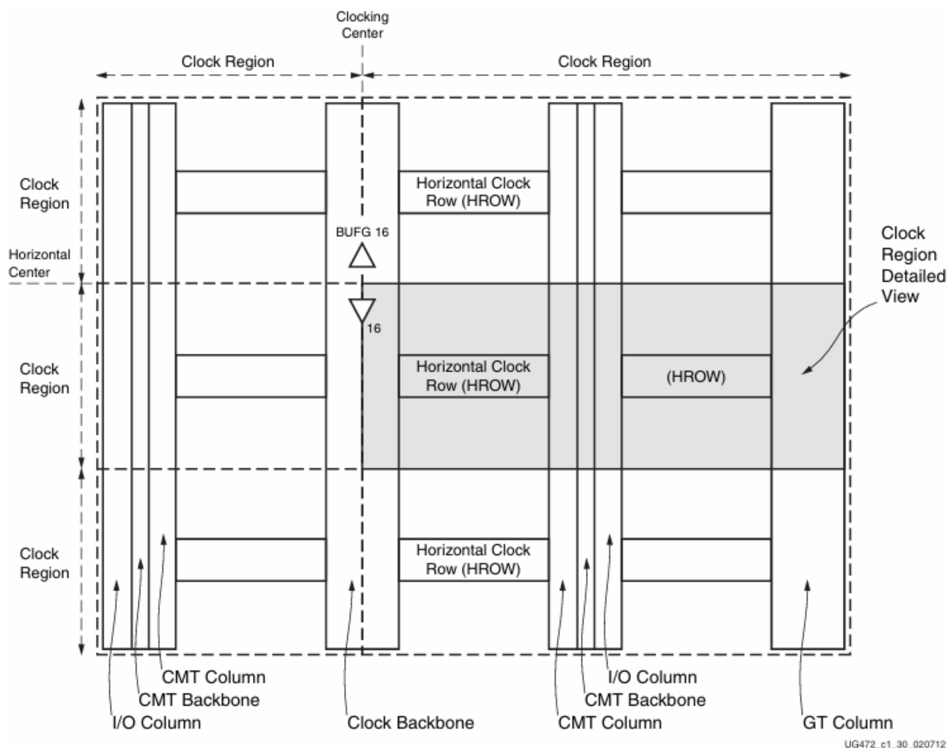


Figure 42: BUFG routing, they are not inside a clock region

Remember that, even if there are 32 BUFG, each clock region can only acquire 12 clock lines.

5.3 BUFH

The movement of the clk between clock regions are made by BUFH (Horizontal buffer). So we have our BUFG that moves clk potentially in one region, and from one region to an other the connection is performed by the BUFH.

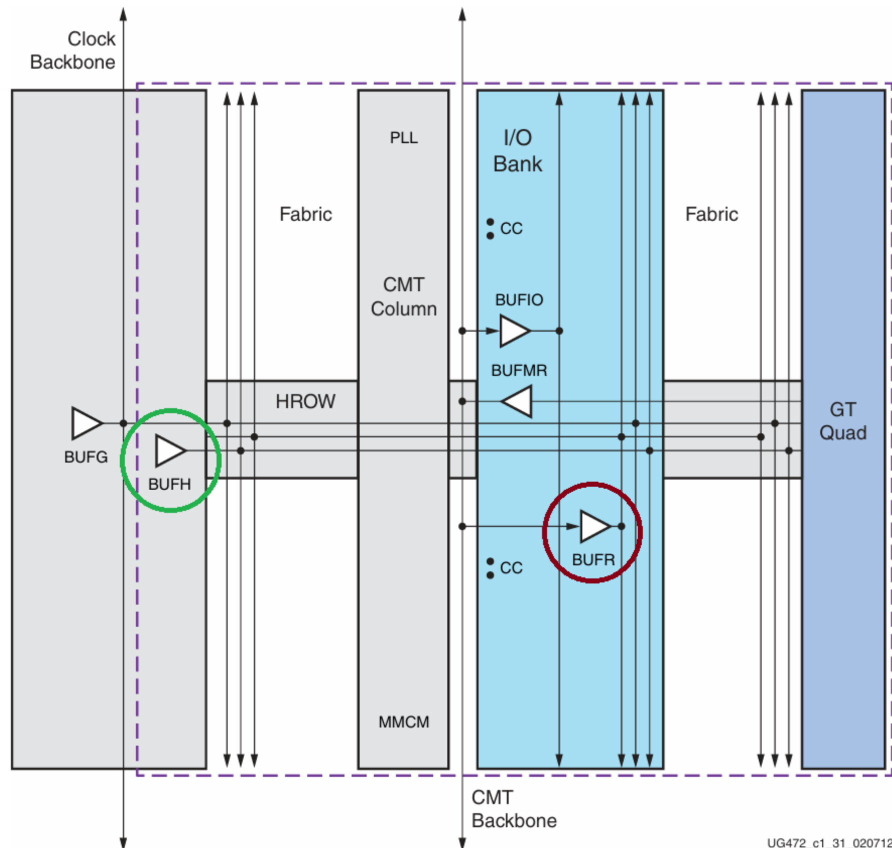


Figure 43: BUFH and BUFR routing

5.4 BUFR

BUFRs drive clock signals coming from clock-capable I/O pin to a dedicated clock net within a clock region, independent from the global clock tree. BUFRs can drive the I/O logic and logic resources (CLB, block RAM, etc.)

If we have a pin that is near to a clock region, and our implemented circuit clk is very small, we can consider to use a BUFR.

5.5 Clock Input

Each I/O bank contains few clock-capable input pins to bring user clocks onto the 7 series FPGA clock routing resources. In conjunction with dedicated clock buffers, the clock capable input bring user clocks on to:

- Global clock lines in the same top/bottom half of the device
- I/O clocks lines within the same I/O bank and vertically adjacent I/O banks
- Regional clock lines within the same clock region and vertically adjacent clock regions
- CMTs within the same clock region and, with limitations, vertically adjacent clock regions.

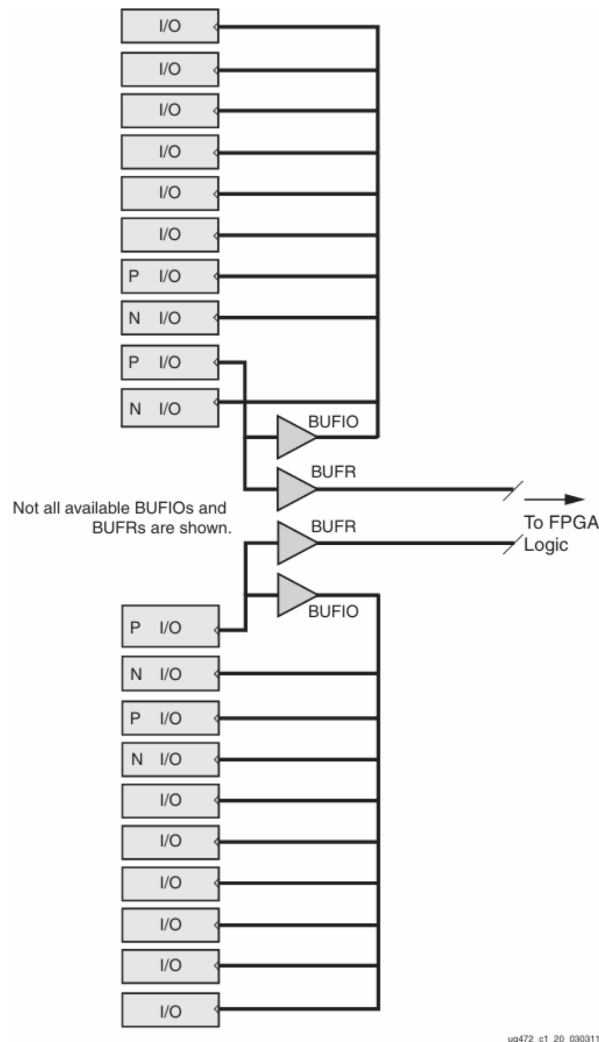


Figure 44: BUFIO Driving I/O Logic

If a standard net is used as a clock signal, there is a several increase on path delay, due to the fact that before arriving to a clock routing path, the signal has to overcome some other non optimized line.

In a circuit that is completely integrated into the FPGA, this is a simple delay from the input to the buffer.

Instead, the delay is converted in skew problem if we connect our FPGA to an external device. The best choice is to use a MMCM to compensate the skew between the two cock paths.

6 Pipeline

Adding in a proper way Flip-Flops and latches into our circuit, can allow us to speed-up the rate of the circuit.

The maximum propagation delay of a circuit is called **latency** and it is represented as τ, the **throughput** instead, is $\frac{1}{\tau}$.

The technique that allows these values to be decoupled is called pipelining, and it is the primary strategy for achieving parallelism.

There exist two kind of pipeline parallelism: Temporal and Spatial.

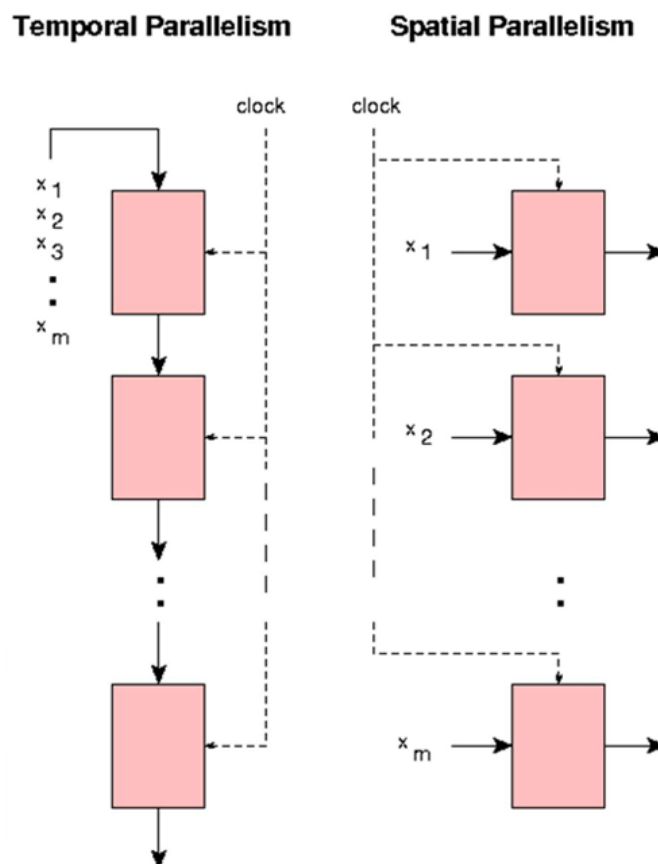


Figure 45: Temporal vs Spatial Parallelism

Pipeline in Temporal Computing Device means that the stages (i.e. Fetch, Decode and Execute) are overlapped.

In FPGA we speak about Spatial Parallelism.

6.1 Latency and Throughput improvement

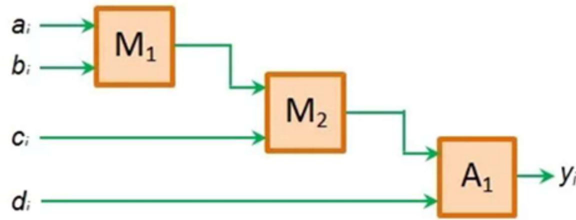


Figure 46: Non pipelined path

The **latency** is the sum of the propagation delay and the **throughput** is the inverse of this value.

Figure of Merit	Value
Slowest path (T_{SLOW})	$t_{p,1} + t_{p,2} + t_{p,3}$
Clock	Not Required
Total Area	$\alpha_1 + \alpha_2 + \alpha_2$

Figure of Merit	Value	
Throughput (or Rate)	$1/T_{SLOW}$	$1/(t_{p,1} + t_{p,2} + t_{p,3})$
Latency	T_{SLOW}	$(t_{p,1} + t_{p,2} + t_{p,3})$

Figure 47: Figure of merit of non-pipelined path

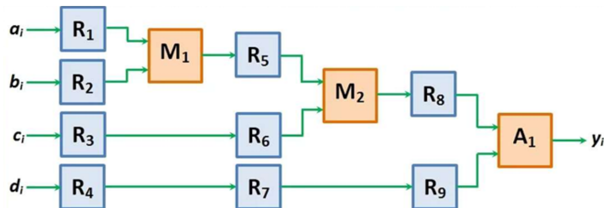


Figure 48: Pipelined path

With the insertion of registers, the logic is now temporized with clock. The clock must be bigger than the maximum propagation delay.

Latency $= N \cdot T_{clk}$, Throughput $= \frac{1}{T_{clk}}$

Figure of Merit	Value
Slowest path (T_{SLOW})	$\min\{t_{p,1}; t_{p,2}; t_{p,3}\}$
Clock	$T_{CLK} \geq T_{SLOW}$
Total Area	$\alpha_1 + \alpha_2 + \alpha_2 + r_1 + r_2 ... + r_9$

Figure of Merit	Value	
Throughput (or Rate)	$\sim 1/T_{SLOW}$	$1/T_{CLK}$
Latency	$> T_{SLOW}$	$N \times T_{CLK}$

Figure 49: Figure of merit of a pipelined circuit

6.2 Power Consumption

Today, area is not the only problem anymore. In fact, also power is a huge problem about pipeline. Static power or leakage power is proportional to the number of the off transistors. Also dynamical power consumption increases because node with pipeline is bigger than the node without pipeline due to the insertion of registers. Also the parasitic capacitance of clock input have to be taken into account.

$$P_D = P_{D_{no_pipe}} + \sum^{reg} f_{sw} \cdot C \cdot V_{DD}^2 + \sum^{reg} \frac{1}{T_{clk}} \cdot C \cdot V_{DD}^2$$

6.3 Optimizing pipeline and well design rules

Our goal now is to do pipeline where is strictly necessary in order to optimize everything.

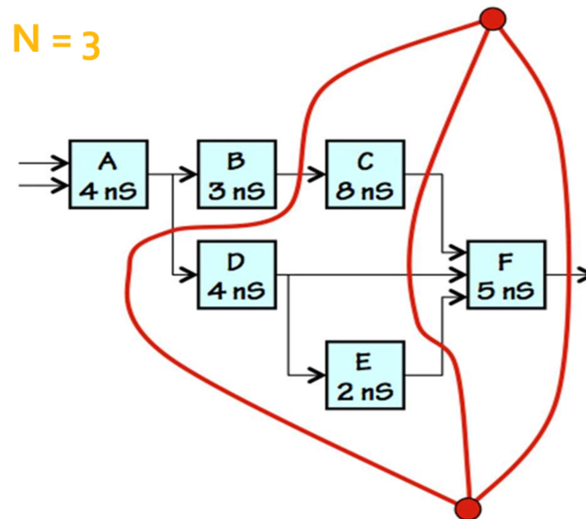


Figure 50: How to add registers to properly pipeline a circuit

1. Draw a first pipeline line that intercepts each output of the combinatorial circuit. This provides a model for implementing a 1 pipeline, usually an uninteresting implementation choice.

2. Draw additional pipeline lines that define the stages. Each contour must (a) divide the circuit by intersecting the signal lines and (b) intersect the lines so that each signal crosses the contour in the same direction. Each additional line increases the number of stages in the pipeline by one.

3. Implement the pipeline circuit by entering a register at each point of the combinatorial circuit where a signal line crosses a pipeline boundary

4. Choose a minimum clock period sufficient to cover the longest combinatorial path within the circuit. Unless ideal registers are assumed, the paths must include the upstream registers propagation delay and the downstream registers setting times.

Continue adding registers only if latency does not drastically increase and throughput remains unchanged.

7 Memories - RAM & FIFO

7.1 Random Access Memory (RAM)

The Random Access Memories is a volatile memory in which it is possible to read and write data, organized in a set of "words", each one identified by an "address". A flag will be used to specify if RAM has to be read or written.

If the value of each "word" is fixed, the memory is called Read-Only Memory (ROM).

RAMs have two main parameters: word width and memory depth.

A **Single-Port RAM** (SPRAM) module must have at least 4 ports:

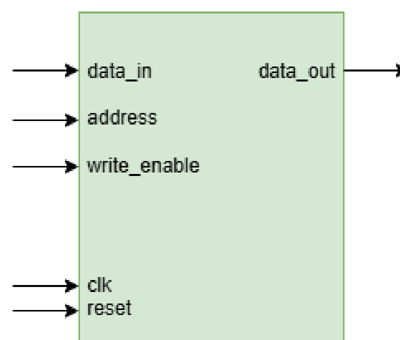


Figure 51: SPRAM design

- **address** port: (input), to select which word has to be read or written
- **write_enable** bit: (input), to specify whether the current operation is a read or a write be read or written
- **data_out** port: (output), to read the data coming from the memory (if reading)
- **data_in** port: (input), to provide the data to be written into the memory (if writing)

In a **Simple Dual-Port RAM** (SDPRAM) is possible to read and write data at the same time, in fact, SDPRAM has address_out and data_out port for reading operation and address_in and data_in port for writing operation. There exists also a **True Dual-Port RAM** (TDPRAM). Note that some conflicts can happen if the same address is read and written at the same time.

Read and Write operation are not instantaneously. When the write_enable is put at '1', data_in is moved inside the memory after a certain amount of clock pulses (latency), the same will be applied for the reading operation.

```
------ Sync Process --------
RAM_engine: process(reset, clk) is
begin
     if rising_edge(clk) then
          if reset = '1' then
                    ram_data  <= (Others => INIT_SLV);
                    dout                <= (Others => '0');
          else
                    -- Write to "addr", only if we = '1'
                    if we = '1' then
                              ram_data(to_integer(unsigned(addr))) <= din;
                    end if;

                    -- Read from "addr"
                    dout <= ram_data(to_integer(unsigned(addr)));
          end if;
     end if;
end process RAM_engine;
          ----------------------------
```

Figure 52: RAM process engine

Keep attention at the addr signal, it has to be an integer value.

In this VHDL description, the write and read operations have a latency of 1 pulse of clk.

Depending on the size of the implemented RAM, Vivado can choose to implement it using Distributed RAM (the 32-bits of LUTs when it has small dimension), or Block RAM (BRAM, when instead dimensions increase). The RAM dimension for which Vivado chooses to implement a Distributed or a Block RAM can be changed in settings.

7.2 FIFO

FIFOs are memories (without addresses) with a "write" and a "read" interface.

As the name suggests, the first "word" written into the FIFO in the WRITE interface is the first one that will be provided on the READ interface.

FIFOs are widely used in digital electronics, as they are the main memory element to handle data streams. In fact, if the data processor is temporary busy, data stream is stored inside a FIFO in order to avoid losing data.

The write interface must have at least three signals:

- **data_in** (input)
- **write_enable** (input), to notify the FIFO that the data on the data_in signal has to be written
- **full** (output), which is set to 1 by the FIFO when the internal memory is full (so any write request is ignored)

Likewise, the read interface must have at least three signals:

- **data_out** (output)
- **read_enable** (input), to notify the FIFO that the data on the data_out signal has been read and a new value can be put on data_out
- **empty** (output), which is set to 1 by the FIFO when the internal memory is empty (so any read request is ignored, and the value on the data_out signal has to be considered as invalid)

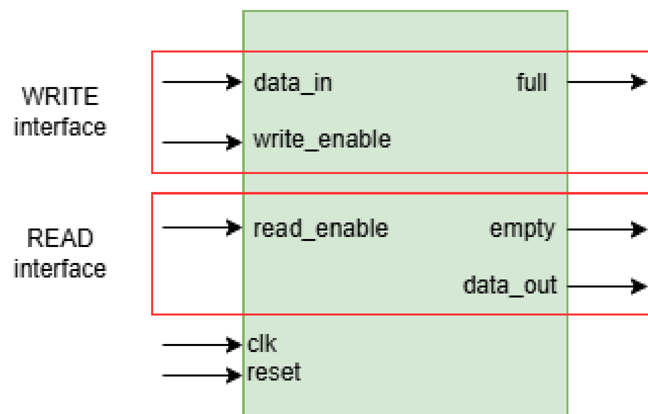


Figure 53: FIFO design

FIFO could have the read and write ports clocked with the same or different clocks. In that case we have Synchronous (SYNC) or Asynchronous (ASYNC) FIFOs, respectively.

FIFO could be standard (STD) or **First-Word Fall Through** (FWFT). The last one has the advantage of outputting immediately the valid data when ready is asserted.

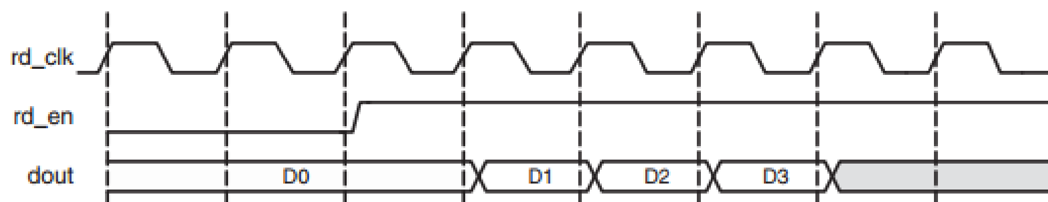


Figure 54: FWFT functioning waveform

One common way to implement FIFOs is by using circular buffers (also called ring buffers). Circular buffers have two “pointers”:

- **Write** (or head) pointer, which points to the location where the next word has to be written

- **Read** (or tail) pointer, which points to the location of the data that's going to be read

It is build using a RAM, in particular a Dual Port RAM type.

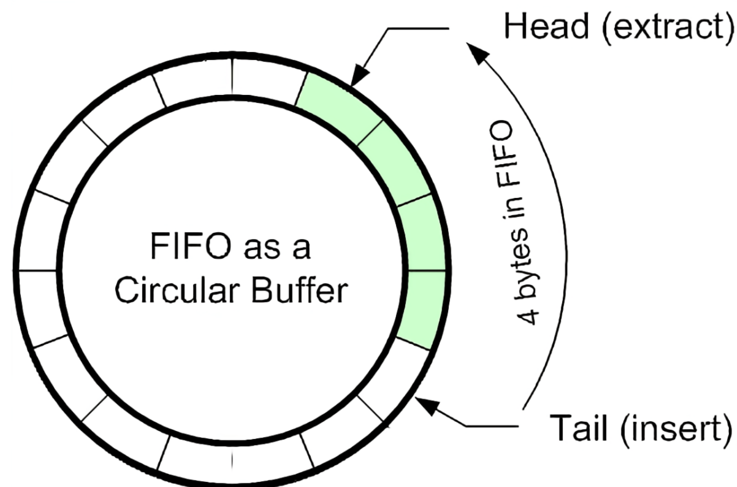


Figure 55: FIFO as a Circular Buffer

From a VHDL description perspective a circular FIFO can be written in the following way:

```
full_int     <=     '1' when fifo_count = FIFO_DEPTH     else
                    '0';

empty_int    <=     '1' when fifo_count = 0              else
                    '0';
```

Figure 56: Full and Empty Behavior (concurrent statement)

```
variable is_writing : std_logic;
variable is_reading: std_logic;


-- Set a couple of variables, to avoid writing
-- "wr_en and not full_int" and "rd_en and not empty_int"
-- every time
is_writing         := wr_en and not full_int;
is_reading         := rd_en and not empty_int;
```

Figure 57: Declaring variables for beauty-code (From this point forward, we are within the FIFO engine process.)

```
-- Keeps track of the total number of words in the FIFO
if is_writing = '1' and is_reading = '0' then
        fifo_count <= fifo_count + 1;
elsif is_writing = '0' and is_reading = '1' then
        fifo_count <= fifo_count - 1;
end if;
```

Figure 58: Word tracking

```
-- Keeps track of the write index (and controls roll-over)
if is_writing = '1' then
        if write_index = FIFO_DEPTH-1 then
                write_index <= 0;
        else
                write_index <= write_index + 1;
        end if;
end if;

-- Keeps track of the read index (and controls roll-over)
if is_reading = '1' then
        if read_index = FIFO_DEPTH-1 then
                read_index <= 0;
        else
                read_index <= read_index + 1;
        end if;
end if;
```

Figure 59: Write and read indexes tracking

```
-- Read the data if needed
dout <= fifo_data(read_index);


-- Write the data if needed
if is_writing = '1' then
        fifo_data(write_index) <= din;
end if;
```

Figure 60: Read and write operation, note that dout assignment is performed inside the process, so this becomes a standard FIFO and not a FWFT

8 AXI-Stream Protocol

8.1 Protocol description

AXI-Stream protocol is a not address mapped protocol, but it simple transfers streams of data. It's made up of a Master and a Slave that work synchronous with each other.

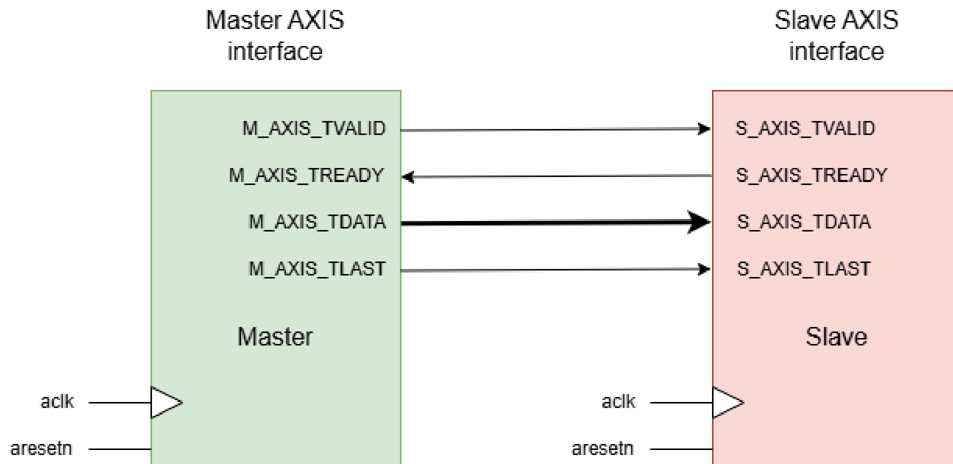


Figure 61: AXIS interface

Let's discuss all ports rules and specifics:

Signal	Dim.	Source	Function
ACLK	1	Clock source	The global clock signal. All signals are sampled on the rising edge of ACLK.
ARESETn	1	Reset source	The global reset signal. ARESETn is active-LOW

Figure 62: aclk and aresetn

Signal	Dim.	Source	Function
TVALID	1	Master	TVALID indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
TREADY	1	Slave	TREADY indicates that the slave can accept a transfer in the current cycle
TDATA	[(8*n-1):0]	Master	TDATA is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.

Figure 63: tvalid, tready and tdata

Signal	Dim.	Source	Function
TLAST	1	Master	TLAST indicates the boundary of a packet.

Figure 64: tlast

With these essential ports we can start doing simple transactions.

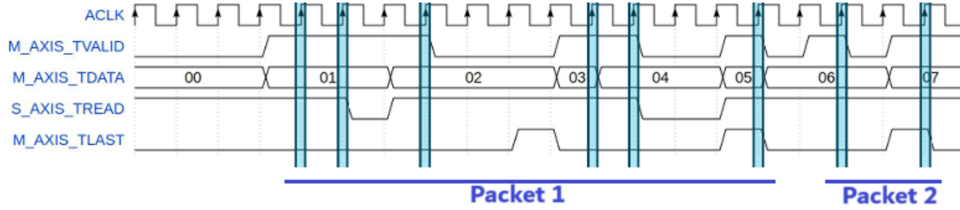


Figure 65: An example of packets transaction

The first high tlast signal does not represent a real last data of a packet. Moreover, each packet can have different number of data.

Additional signals can help to build more complex AXIS protocols.

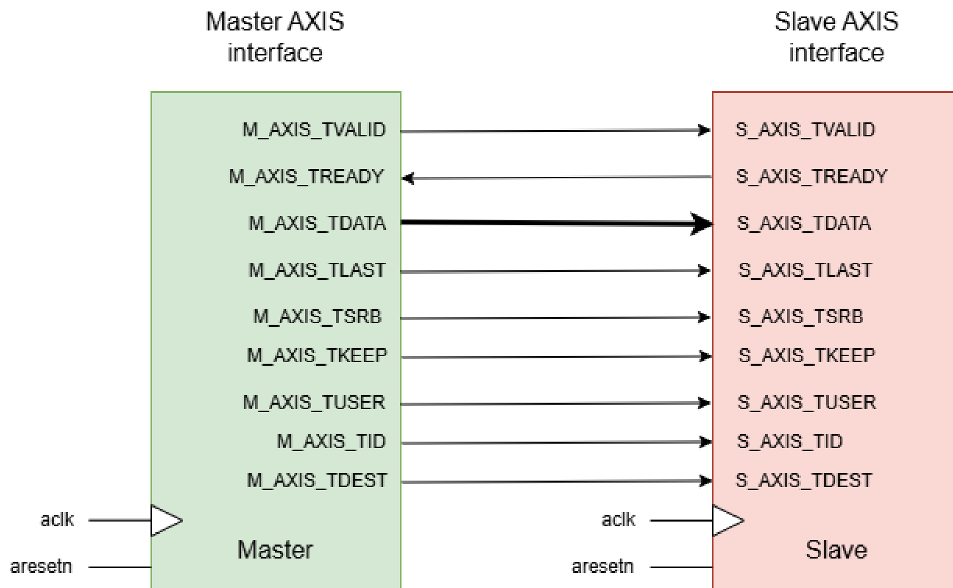


Figure 66: Complete AXIS interface

Let's discuss all the additional signal ports rules and specifics:

Signal	Dim.	Source	Function
TSRB	[(n-1):0]	Master	TSTRB is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte(*).
TKEEP	[(n-1):0]	Master	TKEEP is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream.

Figure 67: tsrb and tkeep

TKEEP	TSTRB	Data Type	Description
HIGH	HIGH	Data byte	The associated byte contains valid information that must be transmitted between source and destination.
HIGH	LOW	Position byte	The associated byte indicates the relative position of the data bytes in a stream, but does not contain any relevant data values.
LOW	LOW	Null byte	The associated byte does not contain information and can be removed from the stream.
LOW	HIGH	Reserved	Must not be used.

Figure 68: Behavior of tsrb and tkeep

Signal	Dim.	Source	Function
TUSER	[(u-1):0]	Master	TUSER is user defined sideband information that can be transmitted alongside the data stream.

Figure 69: tuser

Signal	Dim.	Source	Function
TID	[(i-1):0]	Master	TID is the data stream identifier that indicates different streams of data.
TDEST	[(d-1):0]	Master	TDEST provides routing information for the data stream.

Figure 70: tid and tdest

Master and Slave AXIS handshake is a one-to-one protocol. Anyway, thanks to TDEST signal, Xilinx is able to provide some special engines called broadcaster to route one Master to more Slaves.

8.2 Common Mistakes

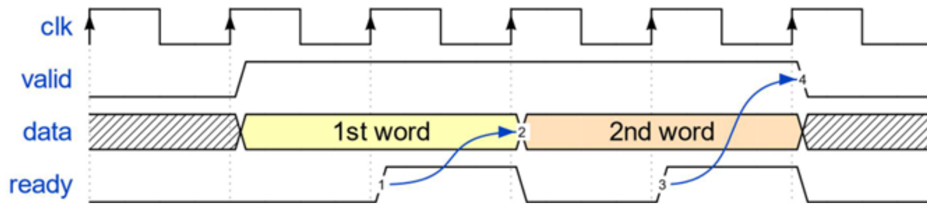


Figure 71: Valid and ready alignment

Data is transferred at each clock cycle when both VALID and READY are "1".
After each clock cycle when this condition is verified, the master can either:

- keep VALID high and provide a new word
- set VALID to 0 if it does not have a word to transfer right now

8.2.1 Valid must not wait for ready

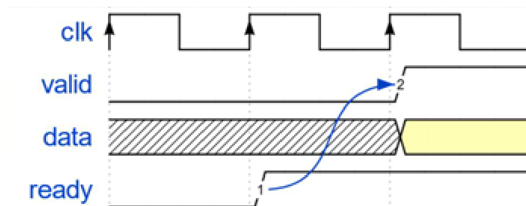


Figure 72: Wrong transaction

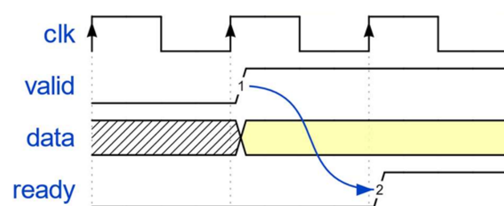


Figure 73: Ok

The VALID assertion ($0 \rightarrow 1$) must not wait for a high READY.

The Master, if it wants to transmit some data, must assert VALID, without waiting for a high READY.

8.2.2 Valid and Data are linked

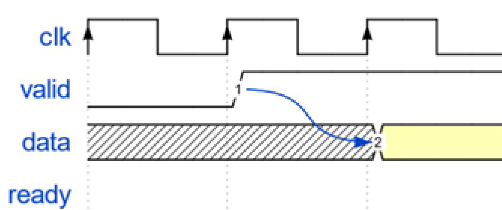


Figure 74: Wrong transaction

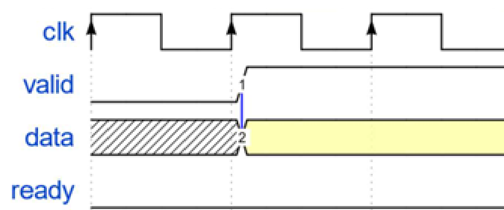


Figure 75: Ok

When VALID is 0, DATA must be considered invalid.

When VALID is 1, DATA is valid and can be used.

In other words, in the same clock cycle as the VALID assertion, the correct DATA value must be put on the bus, not later.

8.2.3 Once VALID is high, it must wait for READY

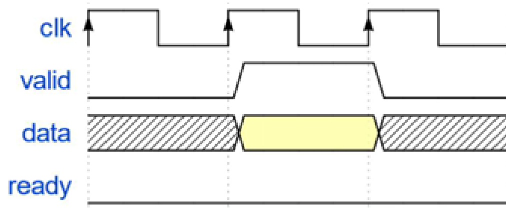


Figure 76: Wrong transaction

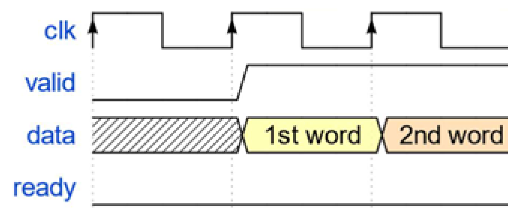


Figure 77: Wrong transaction

Once VALID goes to 1, it must stay to 1 until READY is asserted.

"Presented" data can not be "withdrawn" (left). Similarly, DATA can not be changed until a high READY (right)

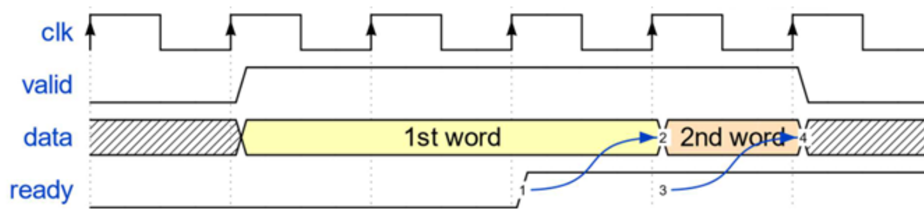


Figure 78: Ok

In the waveform, DATA is kept constant until a high READY. Also, VALID is kept to 1 until the master has no more data to send (in this case, after the second word).

8.2.4 Ready can be freely asserted or de-asserted

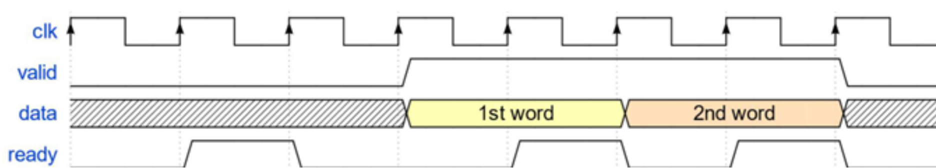


Figure 79: Ok

Unlike VALID, READY is free to be asserted or de-asserted. So, as MASTER, do not rely on a high READY to infer the value of READY in the following clock cycles.