# ENGN4213/6213
# Digital Systems and Microprocessors

Semester 1, 2023

*Week 1, Lecture B*:

## Inside FPGAs & Digital design flow

Australian
National
University

# What’s in this lecture

- How the FPGA hardware works
- Digital design flow steps
  - Combinational *full adder*

# Resources

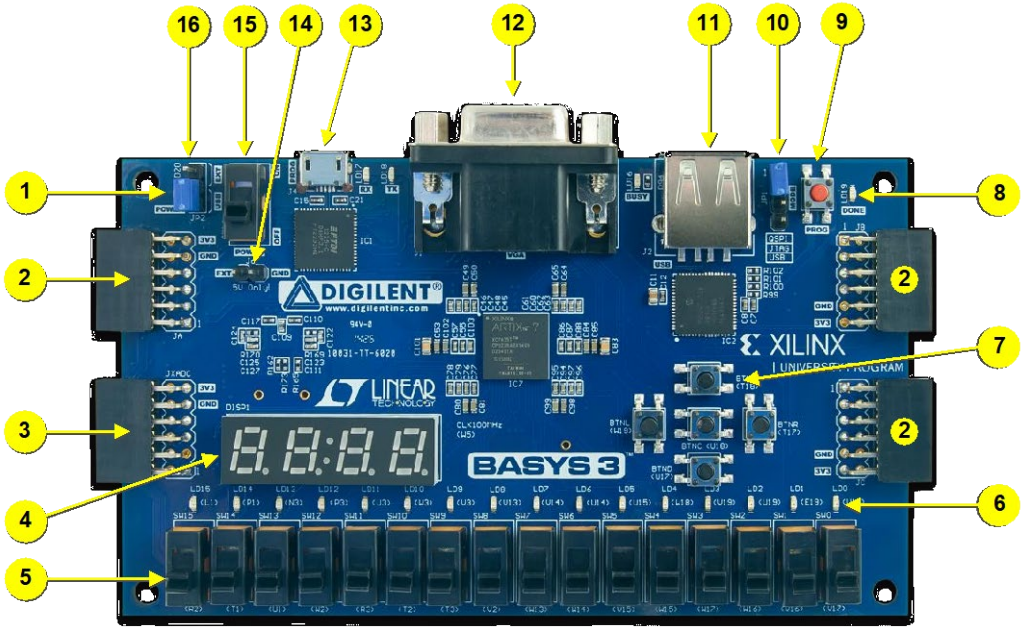
- Wakerly (5th edition) Ch 1.13 and Ch 4 – Digital Design Practices

  *Might give you some high-level ideas about digital design (don't expect to understand everything)*
- Wakerly (5th edition) 15.5 on FPGAs
- A good video on Artix 7 FPGAs: https://www.youtube.com/watch?v=rYTWykLVyak
- Xilinx Artix 7 technical documents (in Wattle/web)

  (Chapter "*Functional Details*" in the *7 Series FPGAs Configurable Logic Block: User Guide* has a large amount of detail on the FPGA chip we use: Don't worry if you don't understand some parts, it is a useful reference for future use)

# Our Development Platform

## Basys 3 Artix-7 FPGA Trainer Board



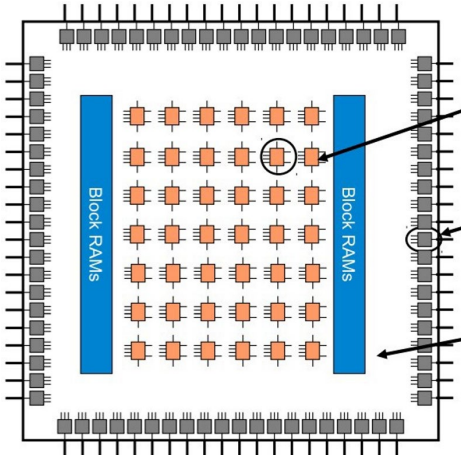

*Source*: Basys-3 FPGA Board Reference Manual

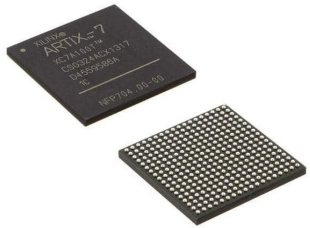
| Callout | Component Description | Callout | Component Description |
|---|---|---|---|
| 1 | Power good LED | 9 | FPGA configuration reset button |
| 2 | Pmod port(s) | 10 | Programming mode jumper |
| 3 | Analog signal Pmod port (XADC) | 11 | USB host connector |
| 4 | Four digit 7-segment display | 12 | VGA connector |
| 5 | Slide switches (16) | 13 | Shared UART/ JTAG USB port |
| 6 | LEDs (16) | 14 | External power connector |
| 7 | Pushbuttons (5) | 15 | Power Switch |
| 8 | FPGA programming done LED | 16 | Power Select Jumper |

# Field-Programmable Gate Arrays (FPGAs)

- FPGAs are integrated circuits that contain arrays of **configurable logic blocks** that can be wired together via **re-programmable interconnects**. They interface to the outside world via **I/O Blocks**.
    - This allows FPGAs to be reprogrammed in the field (hence the name field-programmable)
    - Individual logic blocks can be configured to perform basic combinatorial logic operations (e.g., AND, OR and XOR operations) and store information using registers.
    - Multiple logic blocks can be combined through series and/or parallel interconnections to perform complex digital signal processing algorithms.
- FPGAs have ***parallel and deterministic architectures***, making them well suited to low-latency, high-throughput applications that would otherwise not be implementable using a central processing unit (CPU).

# Main FPGA components




**Configurable logic blocks:** The blocks constitute the main logic resource for implementing synchronous as well as combinatorial circuits.

**Input/output blocks:** The blocks which deal with I/O operations, i.e., acquiring external signals or communicating internal signals to the outside world.

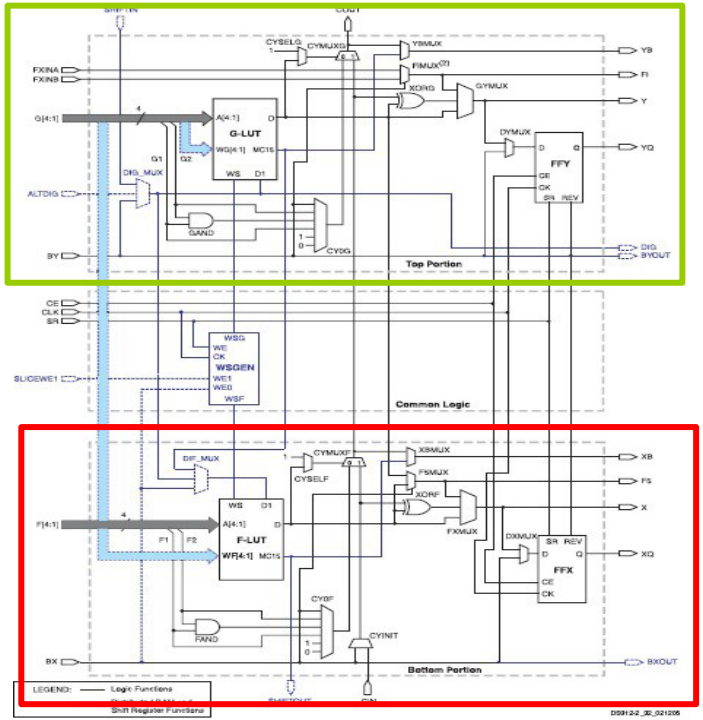
**Block RAM:** Extra memory that can be used for data-intensive applications. (This is optional, although our Xilinx chips do have it)

**Programmable interconnect:** A “fabric” of available signal paths which can be selectively activated (“routed”) in order to suitably connect the blocks

The next slides will provide a general overview of these components based on schematics taken from Spartan-3E FPGA Family Data Sheet

# Main FPGA components

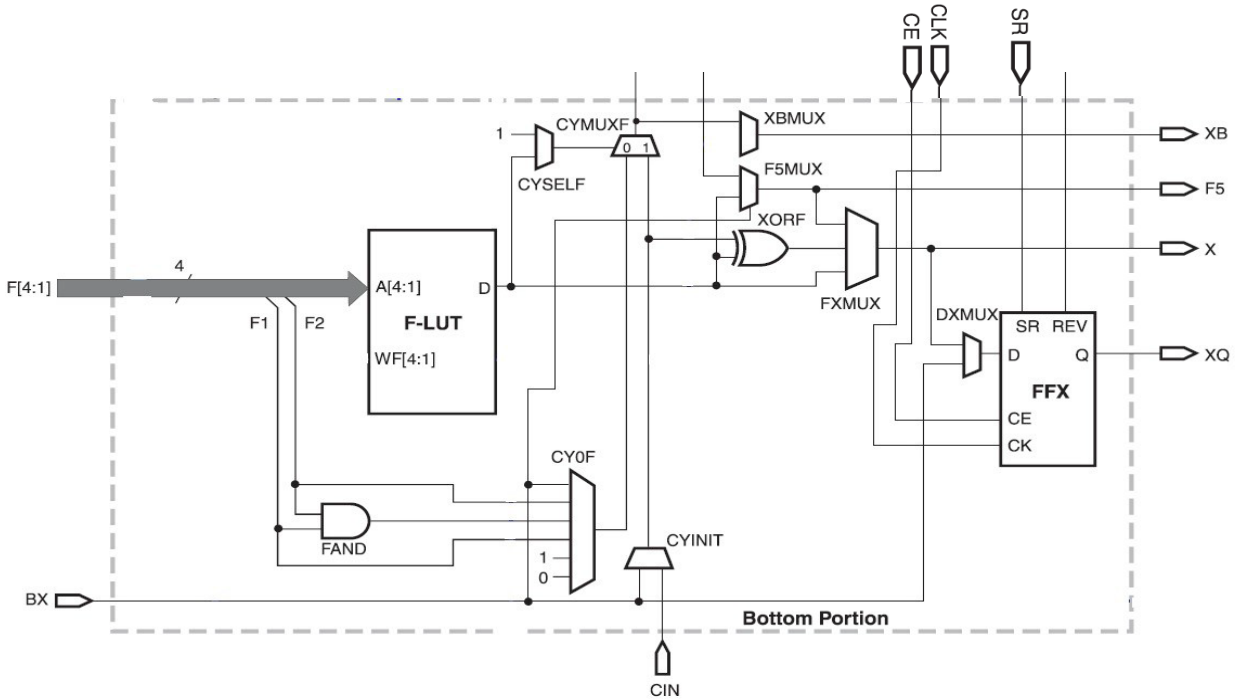
## The Configurable Logic Block (CLB)




- Schematic from Spartan-3E FPGA Family datasheet DS312
- This is a “slice” of a CLB (subunit)
- In the next slided, we focus on the bottom part (red)
- Note the top part (green) is just a duplicate of the bottom part
- In the middle there is a memory component (we won’t worry about it)
- It’s complex, but you do have most of the tools to understand this enough for our purposes

# Main FPGA components

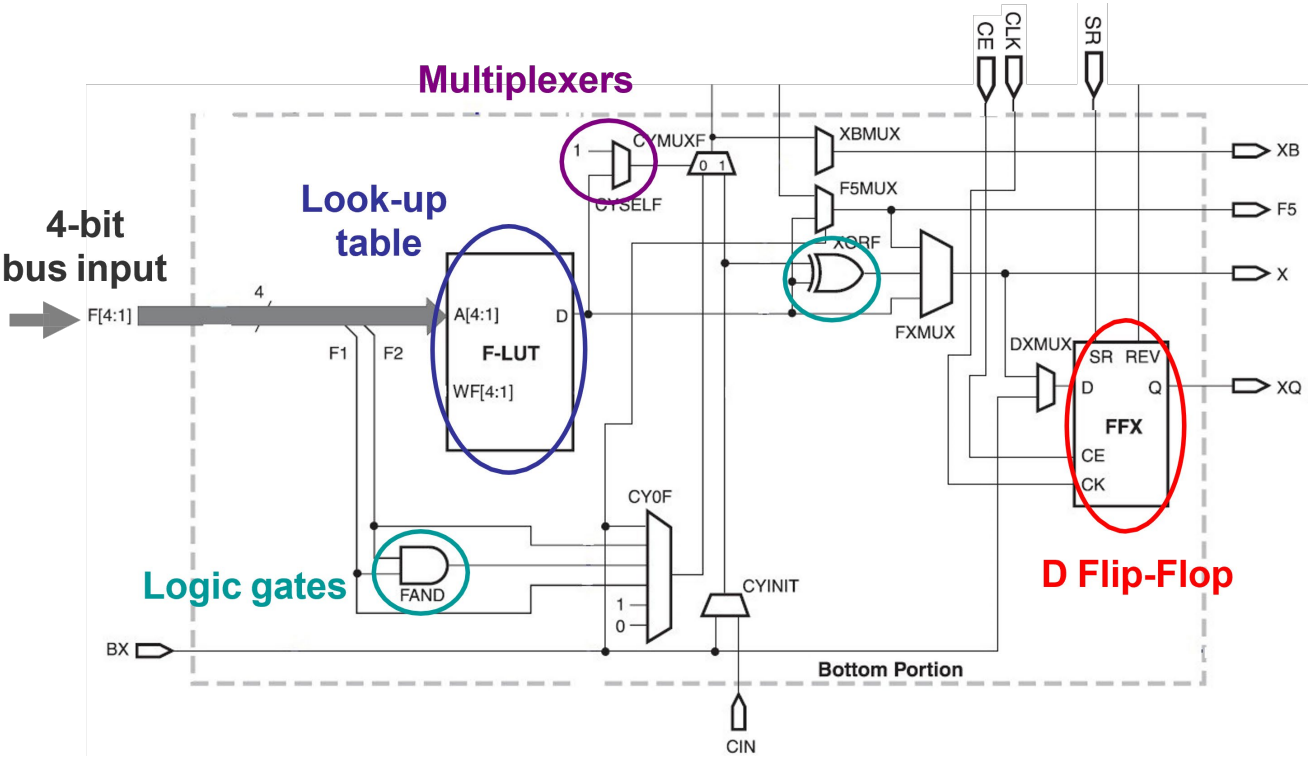
## The Configurable Logic Block (CLB)




**Bottom portion of the slice.**
There are two of these structures in each “slice” (half a CLB)

# Main FPGA components

## The Configurable Logic Block (CLB)



# Main FPGA components

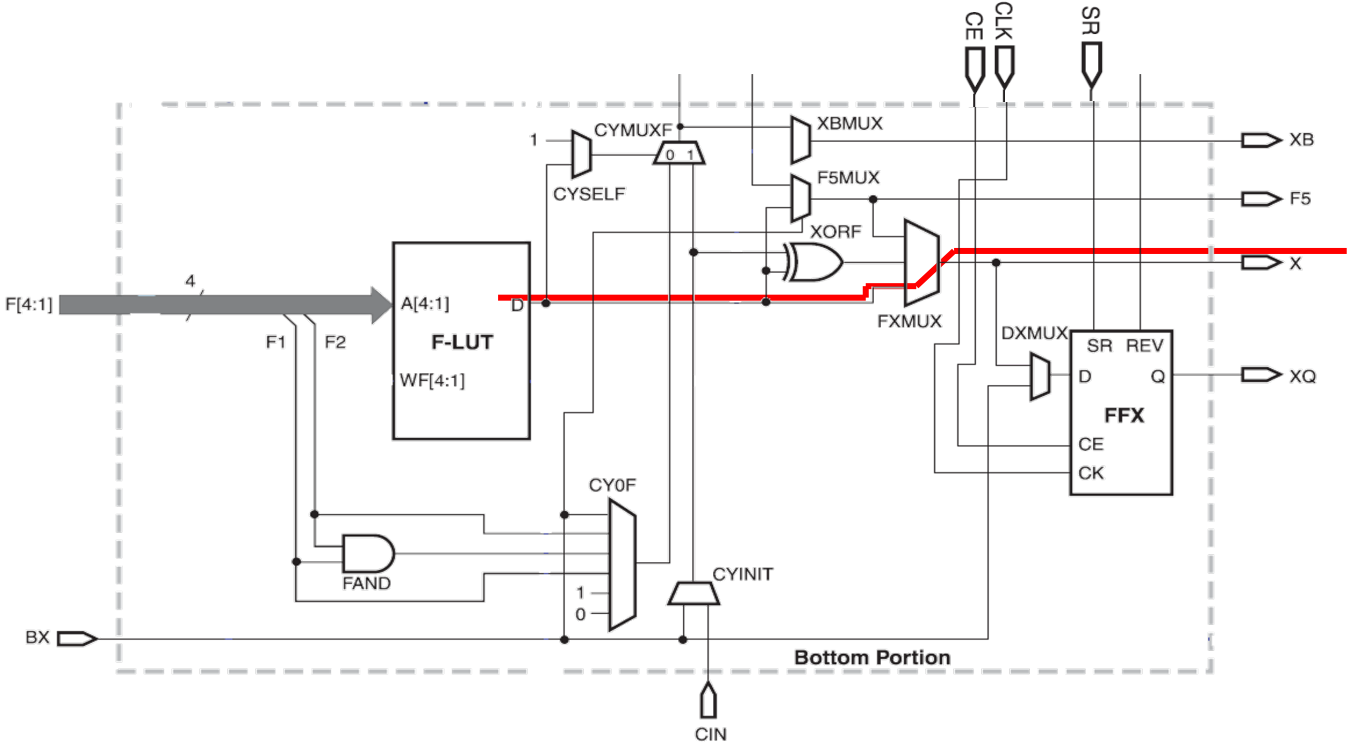
## The Configurable Logic Block (CLB)

- The **Look-up Table (LUT)** can implement *any* *4-input 1-output* combinational logic function *(up to 6 input possible)*
  - It is a memory (*SRAM-based*) block that can be programmed to contain a **table** with 16 rows and 1 column
  - The 4 input bits determine the address of a cell, and the output will be given by the value of a cell
- The output of the LUT is then directed on an appropriate path by various multiplexers. Its path may involve the D flip-flop if the value of the output is to be stored over time.

| address | LUT |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 1 |
| 0011 | 1 |
| 0100 | 0 |
| 0101 | 1 |
| … | … |

# Main FPGA components

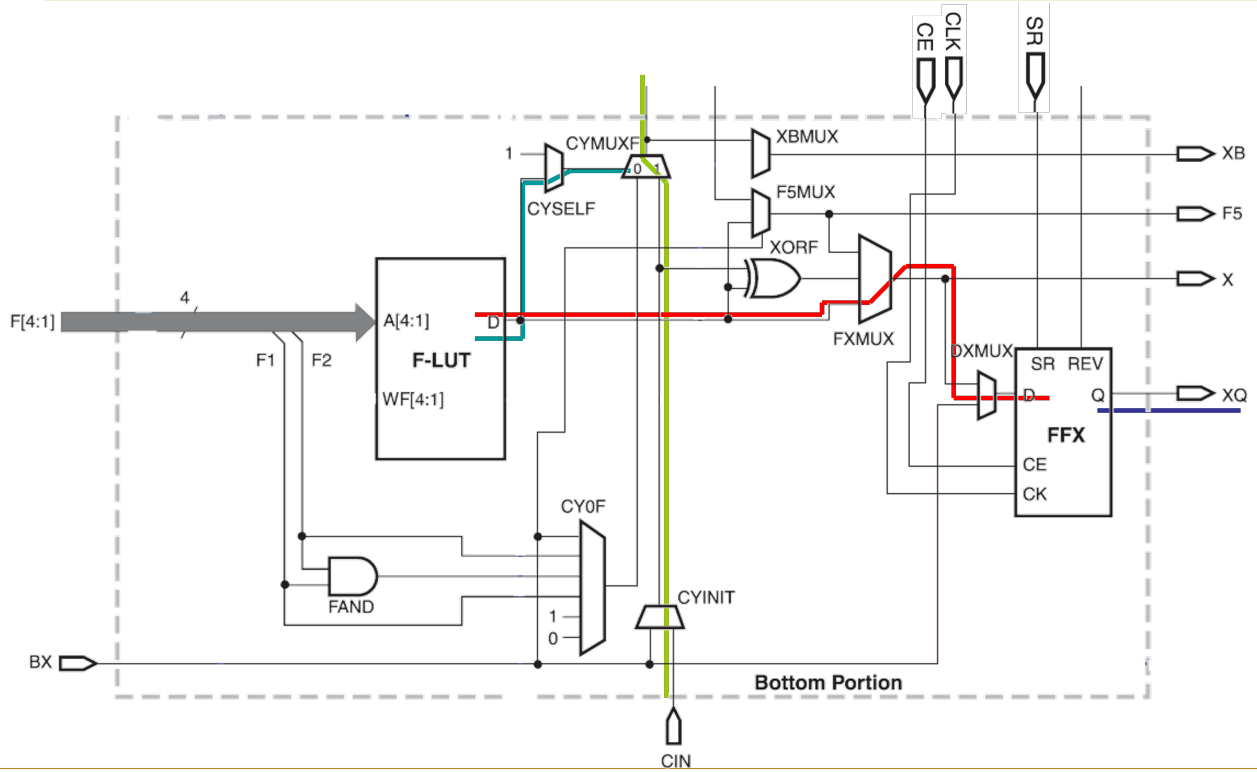
## The Configurable Logic Block (CLB)




Main CLB logic paths: result directly out to interconnect

# Main FPGA components

## The Configurable Logic Block (CLB)

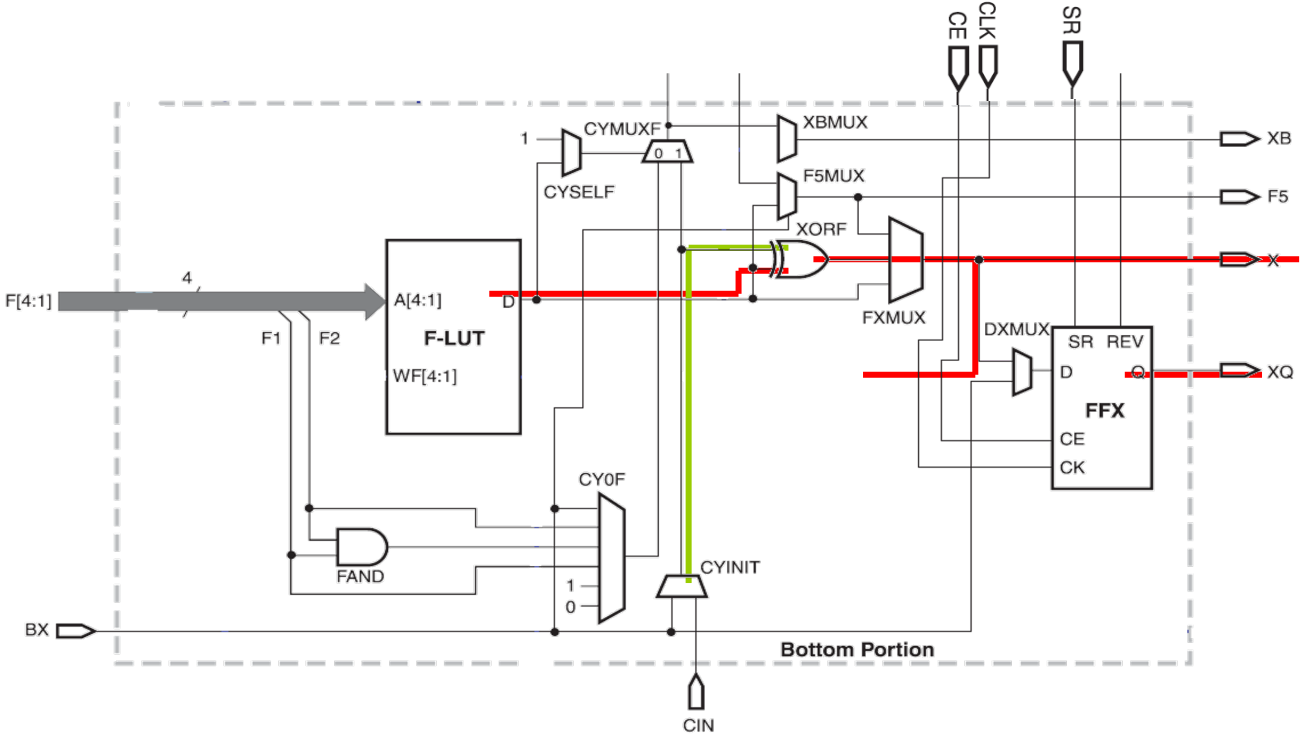



Result stored in FF, which in turn drives the output

Result used to control the carry chain

# Main FPGA components

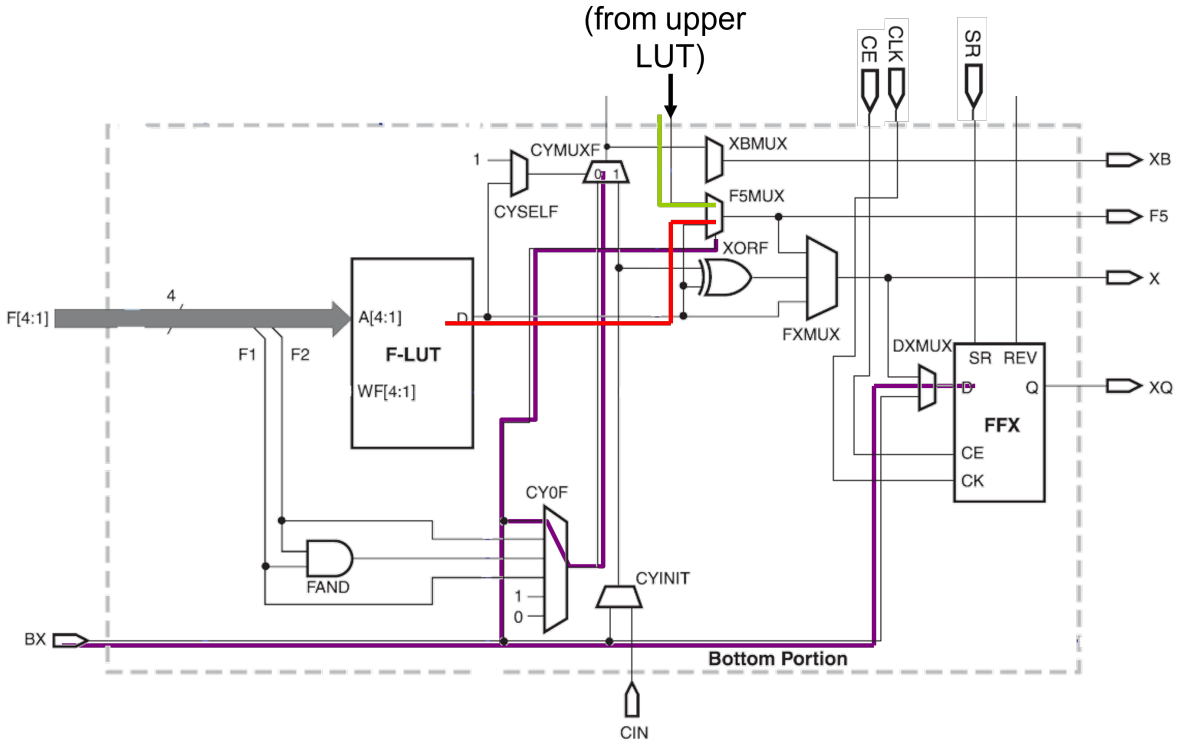
## The Configurable Logic Block (CLB)




Result used in a XOR operation (arithmetic).

# Main FPGA components

## The Configurable Logic Block (CLB)

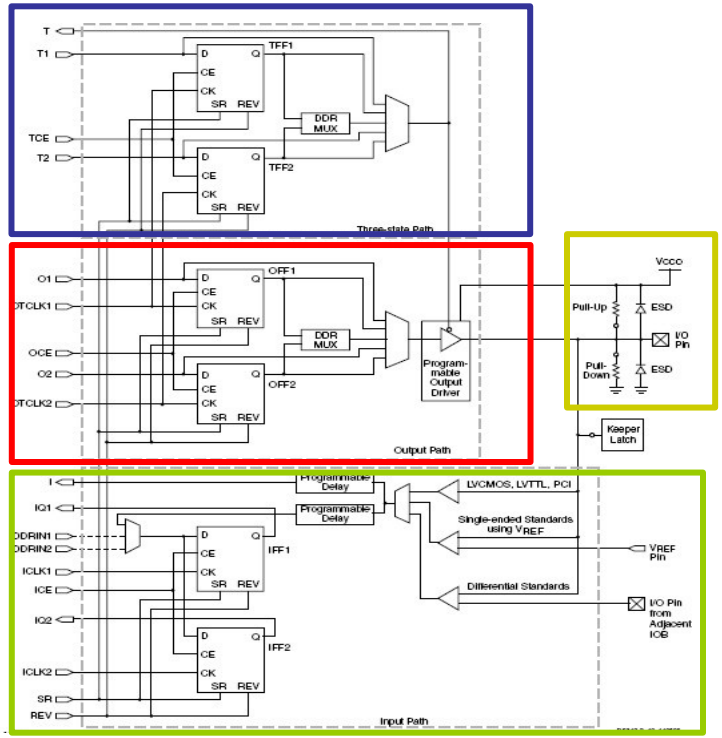



Result multiplexed with result from LUT from top portion.

An auxiliary input can also be used to increase complexity.

# Main FPGA components

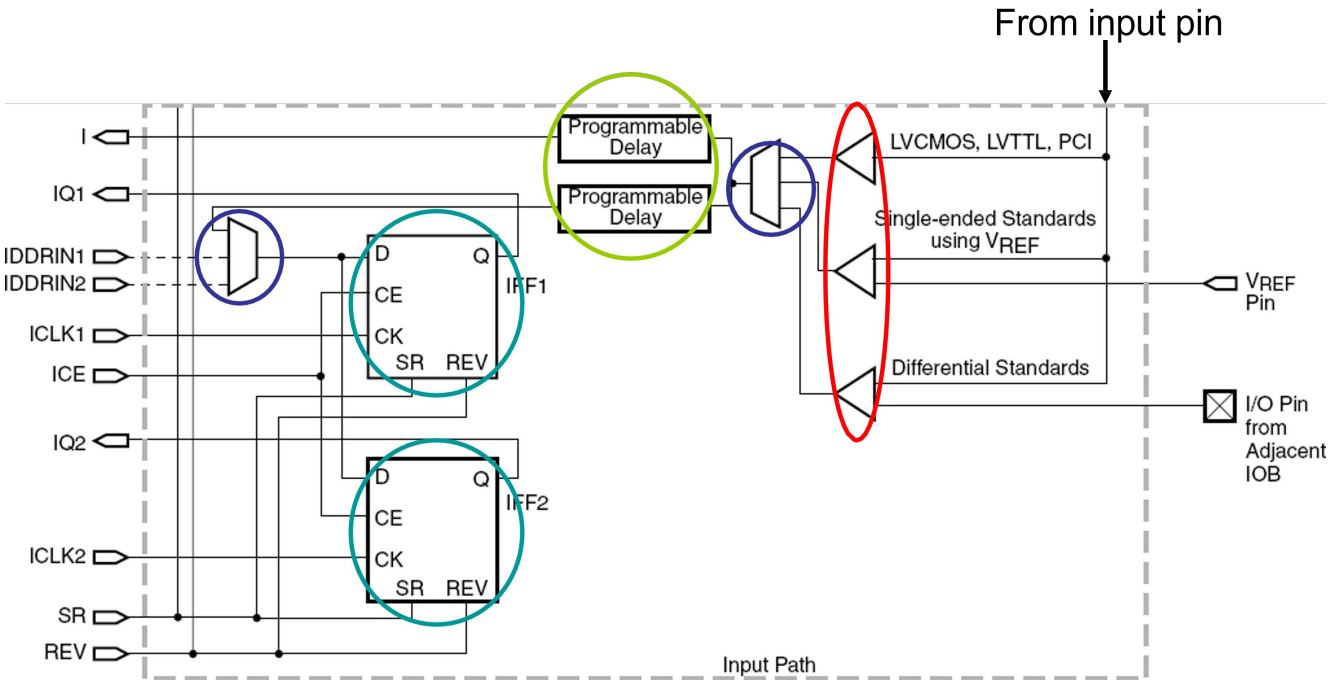
## The Input/Output Block (IOB)




- Schematic from Spartan-3E FPGA Family datasheet DS312
- Located in proximity of an **I/O pin** (yellow)
- The green part is the **input circuit**
- The red part is the **output circuit**
- The blue part is a ***three-state*** **output control path**, we won't worry about it for our purposes.

# Main FPGA components

## The Input/Output Block (IOB): *Input Circuit*




**Op-Amps**

**MUX**

**D flip-flops**

**Programmable Delays**

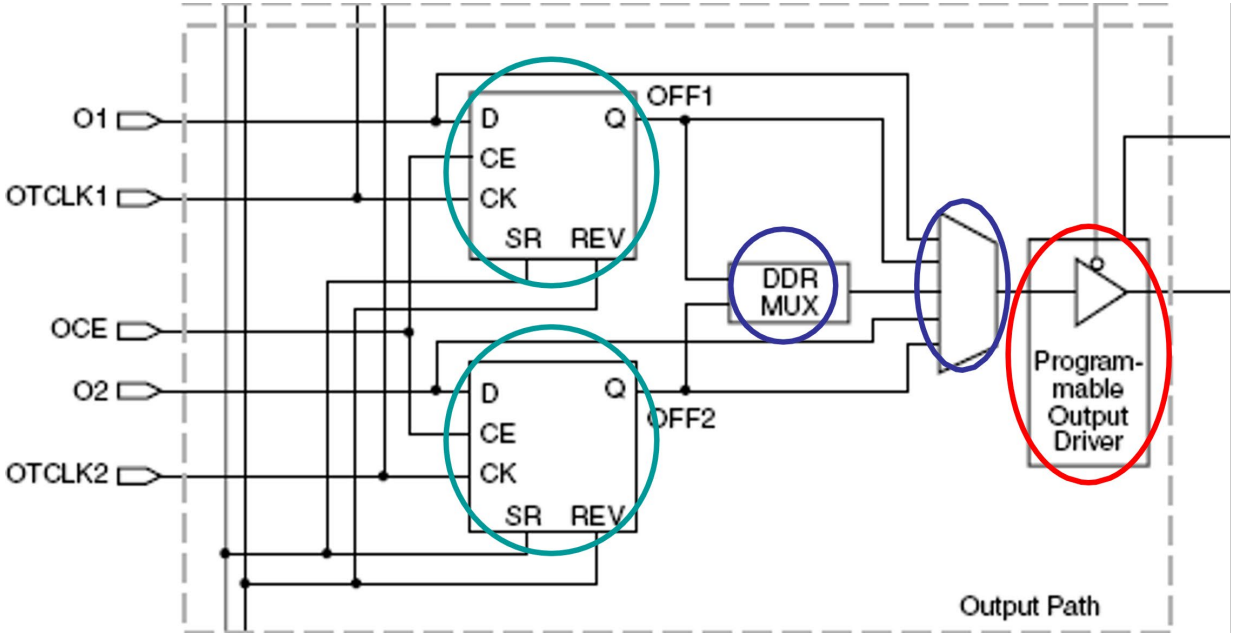
# Main FPGA components

## The Input/Output Block (IOB): *Input Circuit*

- **Op-Amps**
  - Can perform voltage level conversions (e.g. from different logic families)
  - Can act as a buffer or in differential mode
- **Mux**
  - Signal selection and routing around the block
- **Programmable delays**
  - Used to deal with set-up/hold time requirements for flip-flops
- **Flip-Flops**
  - Input bit storage

# Main FPGA components

## The Input/Output Block (IOB): *Output Circuit*




**Op-Amps**

**MUX**

**D flip-flops**

# Main FPGA components

## The Input/Output Block (IOB): *Output Circuit*

- **Op-Amps**
  - Can be programmed in the amount of current it delivers to the output and in its slew-rate
- **Mux**
  - Signal selection and routing around the block
- **Flip-Flops**
  - Output bit storage

# Main FPGA components

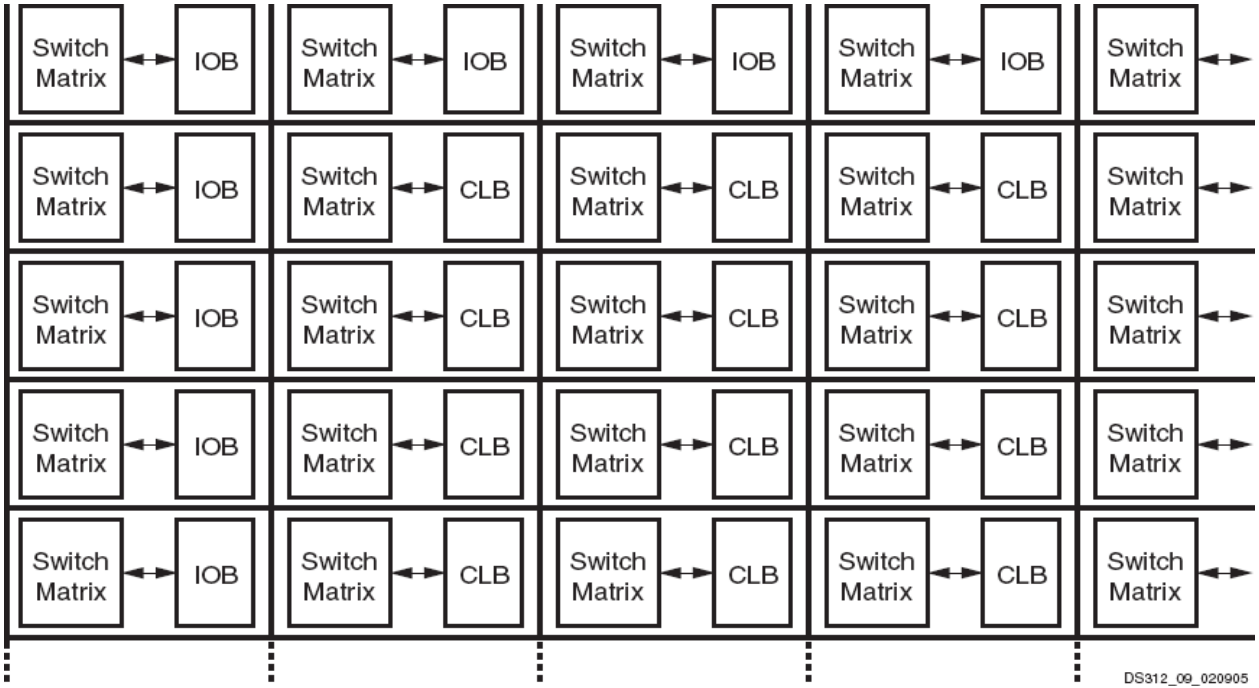
## Programmable interconnect

- True complexity in circuit behavior is achieved by suitably connecting the many CLB to each other and to IOBs/RAM.
- The many CLBs in the chip sit on top of a "*fabric*" *of* *interconnection pathways* called **lines**.
- Each CLB is connected with its immediate neighbors through direct connections and can communicate with distant CLBs by hooking on to the lines
    - This task is achieved by a programmable **switch matrix**
- Although the CLBs are all on the same chip, there are different delays along shorter or longer paths and this can affect *performance at high operating frequencies*

# Main FPGA components

## Programmable interconnect

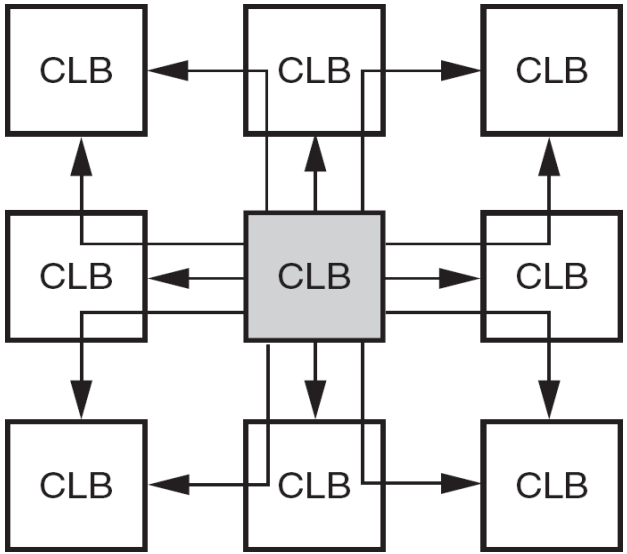
The **“fabric”**



# Main FPGA components

## Programmable interconnect

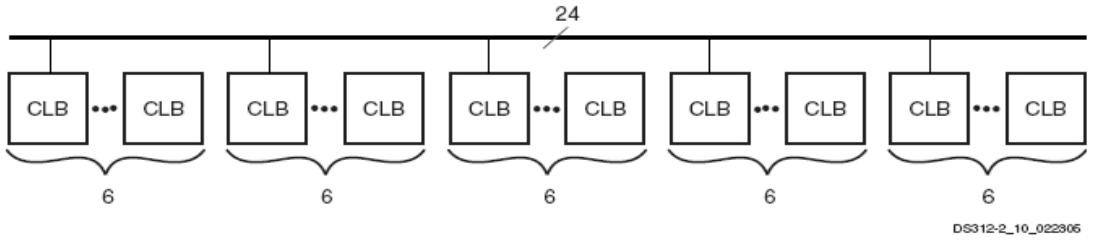
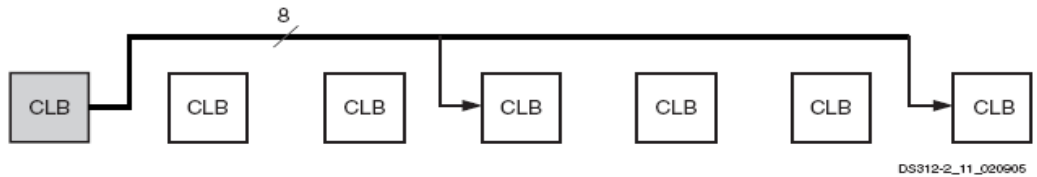
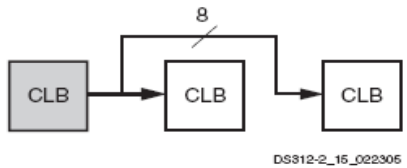
- Direct interconnections (fastest): do not require the use of a switch matrix



# Main FPGA components

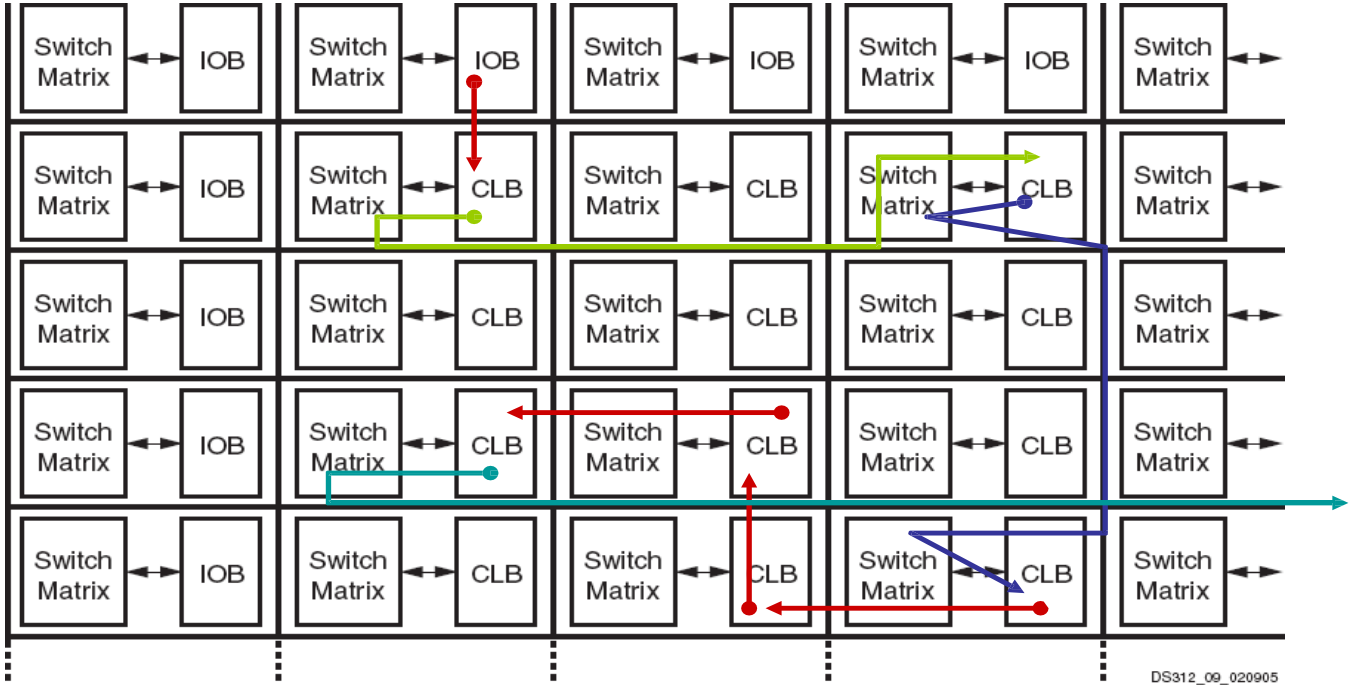
## Programmable interconnect

- Connection *lines*: slower, involve switching matrix

| | |
|---|---|
| **Horizontal and Vertical Long Lines** (horizontal channel shown as an example) |  |
| **Horizontal and Vertical Hex Lines** (horizontal channel shown as an example) |  |
| **Horizontal and Vertical Double Lines** (horizontal channel shown as an example) |  |

# Main FPGA components

## Programmable interconnect




Examples of interconnects:

**Direct**

**Double lines**

**Hex line**

**Long lines**

# Main FPGA components

## Programmable interconnect

- There are also a few ***global lines*** which go past every single block
- Can you think of a signal which all blocks may want to receive at once, with the minimum amount of delay involved?

# Main FPGA components

**How many Flip-flops in the FPGA used by Basys-3?**

| Device | Slices | SLICEL | SLICEM | 6-input LUTs | Distributed RAM (Kb) | Shift Register (Kb) | Flip-Flops |
|---|---|---|---|---|---|---|---|
| 7A12T | 2,000 | 1,316 | 684 | 8,000 | 171 | 86 | 16,000 |
| 7A15T | 2,600 | 1,800 | 800 | 10,400 | 200 | 100 | 20,800 |
| 7A25T | 3,650 | 2,400 | 1,250 | 14,600 | 313 | 156 | 29,200 |
| **7A35T** | 5,200 | 3,600 | 1,600 | 20,800 | 400 | 200 | 41,600 |
| 7A50T | 8,150 | 5,750 | 2,400 | 32,600 | 600 | 300 | 65,200 |
| 7A75T | 11,800 | 8,232 | 3,568 | 47,200 | 892 | 446 | 94,400 |
| 7A100T | 15,850 | 11,100 | 4,750 | 63,400 | 1,188 | 594 | 126,800 |
| 7A200T | 33,650 | 22,100 | 11,550 | 134,600 | 2,888 | 1,444 | 269,200 |

# Main FPGA components

**Summing up:**

- **FPGA:** A complex chip with configurable logic blocks (LUT-based), input/output blocks and programmable interconnects
  - CLBs are the basic building blocks of an FPGA, consisting of look-up tables (LUTs), flip-flops, and other circuit elements that can be programmed to implement digital logic functions.
  - Programmable interconnects provide routing paths between the CLBs, allowing them to be connected to form larger circuits.
  - The memory blocks and multipliers provide additional resources for implementing more complex designs
  - Input-output blocks manage the interactions with the outside environment
- Operation of an FPGA involves configuring the logic blocks and interconnects to create the desired digital logic circuit.

# Main FPGA components

**Summing up:**

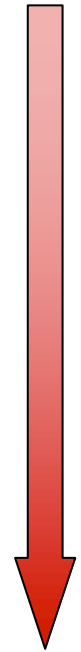
- It is not necessary for you to memorize the exact inner workings of the FPGA hardware
  - After all, it is complex hardware designed to make your life simpler
  - Every FPGA device is different. If you have a general idea, you can always refer to the data sheet
- The main point is that this general overview will
  - help you understand how what you do on a computer screen is translated into hardware
  - help you troubleshoot the second part of the design flow when issues arise

# Digital Design

# Digital Design

## Levels

- **Physical level**
  - materials, geometry, etc.
- **Transistor level**
  - arranging them to create logic gates
- **Logic (gate) level**
  - combining gates to create Boolean functions
- **Building blocks level**
  - creating units with a defined and useful function (e.g., a multiplexer, a counter, a register, etc.)
- **System level**
  - combining blocks to obtain a system which exhibits greater complexity (e.g., a device, such as a microprocessor)



# Digital Design

## Hardware implementation considerations

- Prototyping / testing alternative designs
- Circuit footprint considerations, performance considerations
- Cost considerations
- Production volume

| **Multi-component assembled circuits** | **Programmable logic devices CPLD / FPGA** | **Application-specific IC (ASIC)** |
|---|---|---|
| Need re-assembly for each design | Reprogrammable, prototyping is fast | Need re-engineering for each design |
| Maximum footprint | Fixed footprint | Minimum footprint |
| Worst performance | Mid-range performance | Best performance |
| Very low cost on low production volumes | Affordable on low production volumes | High cost for small volumes |
| Potentially high cost on high production volumes | High cost on high production volumes | The higher the volume the lower the unit cost |

# Digital Design

## FPGA Design Flow

1. Concept
2. Composition
3. Simulation
4. Synthesis
5. Implementation on FPGA hardware
6. Hardware operation and testing

In today's lecture

Learn more in Labs

# Digital Design

## FPGA Design Flow

*In this course, we use the Xilinx IDE called **Vivado** to take care of all steps of the design flow while implementing digital designs on the Basys-3 FPGA board.*

It is a development suite with multiple functionalities:

- Project organizer (file keeping, design hierarchy, etc.)
- RTL Visualization tools
- Text editor for HDL coding
- Simulator
- And other functions for Synthesis and Implementation steps of the design flow

# Digital Design

## FPGA Design Flow: *1) Concept*

- We will be working from a **logic design level** upward (no Physics, and we assume – unless we observe otherwise – that the transistors will deliver satisfactory performance)
- It seems obvious, but *before* you can create a design, it needs to be well thought out first in your head.

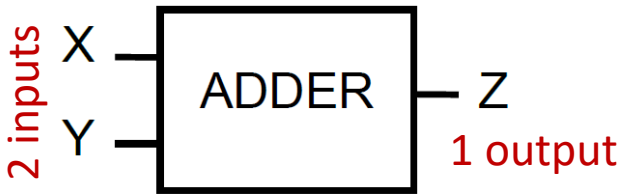
- Things that should be clear to you at this stage:
  - What gates / building blocks do you intend to use?
  - How do you intend to connect them to each other and to the outside world (inputs/outputs)?
- As you proceed with your design, you may discover some issues which may force you to rethink your concept.

# Digital Design

## FPGA Design Flow: *1) Concept*

An example of how to conceptualize a very simple problem:

*(1-bit simple adder)*




Truth Table

| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

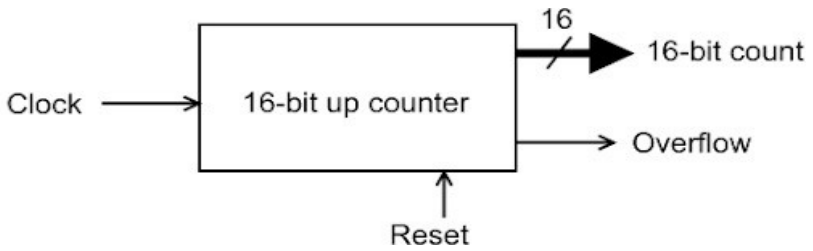
This looks like an XOR operation.

So, *I will realize it as a single XOR gate*

# Digital Design

## FPGA Design Flow: *1) Concept*

- Conceptualization of a more complex design – a *counter*:
  - It's a time-dependent system, will need to store current count in some form of memory / register
  - Will need other a trigger to count (up / down?)
  - When the trigger is asserted, the count should update
  - Does the counter need a reset trigger to return to zero count?
  - What will be the maximum count? This will affect the size of the count readout bus



# Digital Design

## FPGA Design Flow: *2) Composition*

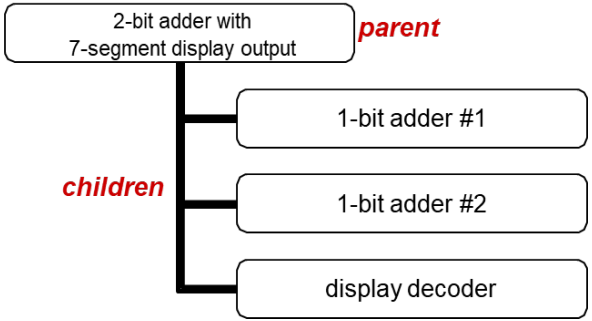
- It concerns the translation of the concept into a *workable format*:
  - A **schematic** description (on a PC this will be a ***.sch*** file such as you may have seen in PSPICE)
  - A language-based description using a **Hardware Description Language (HDL)**
    - This course will teach you **Verilog HDL.** Verilog files have extension ***.v***

- You can exploit hierarchical design, by identifying ***modules*** which can be repeated throughout a more complex design
  - e.g., if your design uses 10 1-bit multiplexers, it makes no sense to write out the logic gate structure of all 10 of them. You can create 10 *instances* of the same module
  - Example (in slide 45): multi-bit adder made up of many single-bit adders

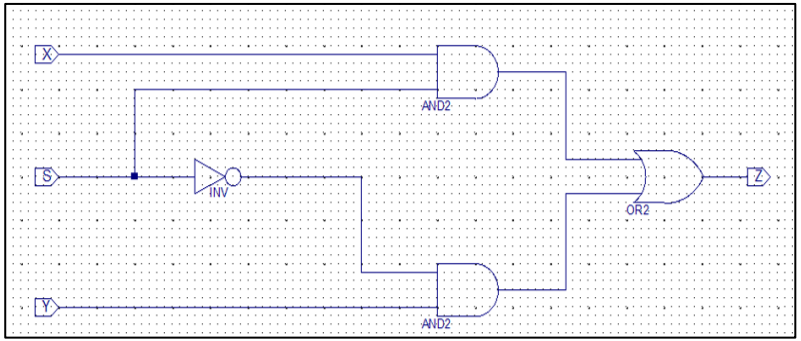
# Digital Design

## FPGA Design Flow: *2) Composition*

### Hierarchical diagram




### A circuit schematic (MUX.sch)




### Some Verilog code (fulladder.v) :

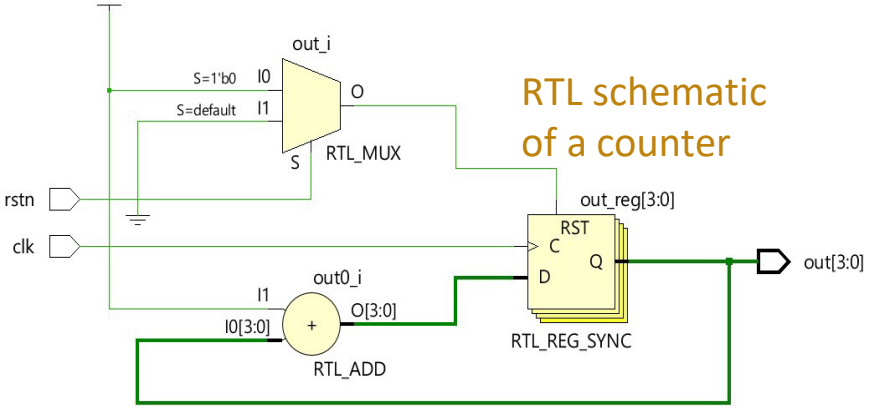
```
//fulladder.v
module fulladder( input wire a,
input wire b,
input wire cin,      //this is a NOTE about cin
output reg s,
output reg cout);     //declaration of module i/o variables

always @ ( * )       //procedural block
   begin
   s = a ^ b ^ cin;    //a blocking assignment
...
```

# Digital Design

## FPGA Design Flow: *2) Composition*

### RTL (Register-Transfer Level) design

- RTL design is about considering data storage elements in a digital system and how information flow is operated on as it moves from one storage element to another.
- It is a higher level of abstraction than *gate-level design,* but it uses constructs that have a viable logic implementation.
- RTL design can be synthesized and optimized on different platforms, keeping data flow the same.
- It is a lower level of abstraction than *behavioral* design, which may not be implementable in hardware (more on this when we discuss Verilog)



# Digital Design

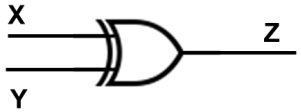
## FPGA Design Flow: *3) Simulation*

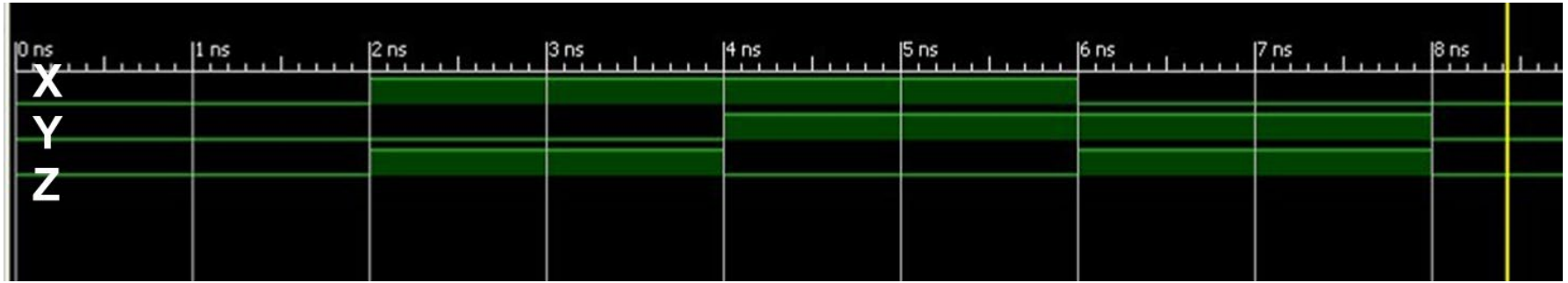
- Once a proposed design is ready, simulation is essential to *verify that it operates as intended*.
  - Useful, especially before money is committed to buying hardware which may malfunction or breakdown
- In order to run a simulation, a ***test bench file*** is required.
  - It specifies the *timing and magnitude of the input/output signals* which will be fed to the design.
  - This is written as *HDL code* (Verilog HDL in our course)
- The simulation software generates *a list of expected signal values and outputs* based on the test bench inputs.
  - Note that the Simulator *will treat the design as using an "ideal" hardware* (e.g. instantaneous logic transitions, no delays) unless otherwise specified

# Digital Design

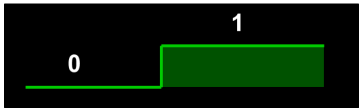
## FPGA Design Flow: *3) Simulation*

Example of simulation results for a XOR gate:




How to read the ISim traces



# Digital Design

## FPGA Design Flow: *4) Synthesis*

- Process carried out by a Synthesizer software: Vivado has an internal synthesizer
- The schematic / Verilog HDL is turned into a **netlist** file
  - A description of block hierarchy, components and their interconnections (netlist) down to gate level
- The *synthesis report* can contain useful information to identify some problems with the design. It also contains an initial maximum frequency estimate
  - There is such thing as a *design that can be simulated but not synthesized*. We will encounter *non-synthesizable constructs* later on in the course. The synthesizer will let you know if you have introduced non-synthesizable constructs in your design.
    - E.g. : Delay or wait statements, Loops with a variable trip count, etc.

# Digital Design

## FPGA Design Flow: *5) Implementation on FPGA hardware*

A 3 step process (on the Xilinx platform):

- **Translate**
  - The content of the synthesized netlist undergoes some reprocessing
- **Map**
  - A gate optimization routine is run and the primitives (LUT, Flip-Flops, RAM) are mapped to specific device resources.
  - This step often picks up on further issues arising from bad design practices, such as unconnected wires or implicit latches.
- **Place and Route**
  - The blocks are allocated to specific CLBs on the chip
  - The programmable interconnect is routed as required to meet any timing constraints if applicable

# Digital Design

## FPGA Design Flow: *5) Implementation on FPGA hardware*

- During the implementation stage, the software refers to the ***user constraint file (UCF)*** for information on timing constraints and/or usage of input/output resources and tries to meet these constraints
- Finally, the FPGA configuration is translated into a binary stream called a **fuse** file (extension *.bit*) which can be downloaded to the board via a USB cable.
- A dedicated circuit on the development board takes care of writing the USB data stream onto the actual FPGA chip.
- Once the transfer is complete, you are ready to interact with your digital design and *test its operation in hardware*!
- Note: the LUTs in the FPGA are made of volatile memory, so if the power is cycled, the chip will lose its configuration.

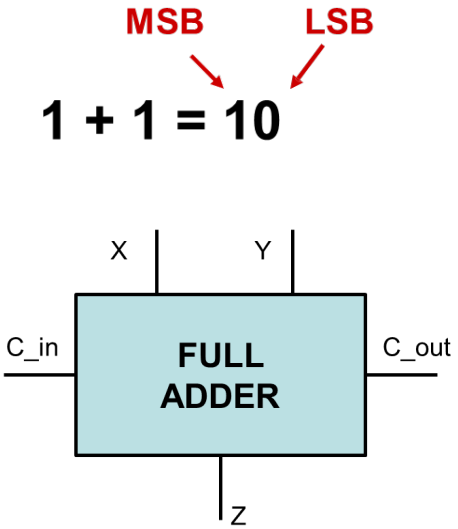
# Digital Design

## Example: 4-bit adder

### 1-bit full adder

How does it work?

- Binary addition of 1-bit signals $X$ and $Y$.
- The Least Significant Bit (LSB) of the result is output as $Z$
- The Most Significant Bit (MSB) of the result is output as $C\_out$ (Carry out)
- $C\_in$ is not useful here. It is an additional input which is used when connecting several full adders in series.
    - When C_in ≠ 0, the block carries out the binary addition of the three inputs $X + Y + C\_in$



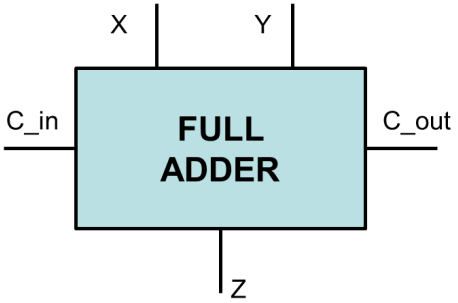
# Digital Design

## Example: 4-bit adder

### 1-bit full adder

Truth Table

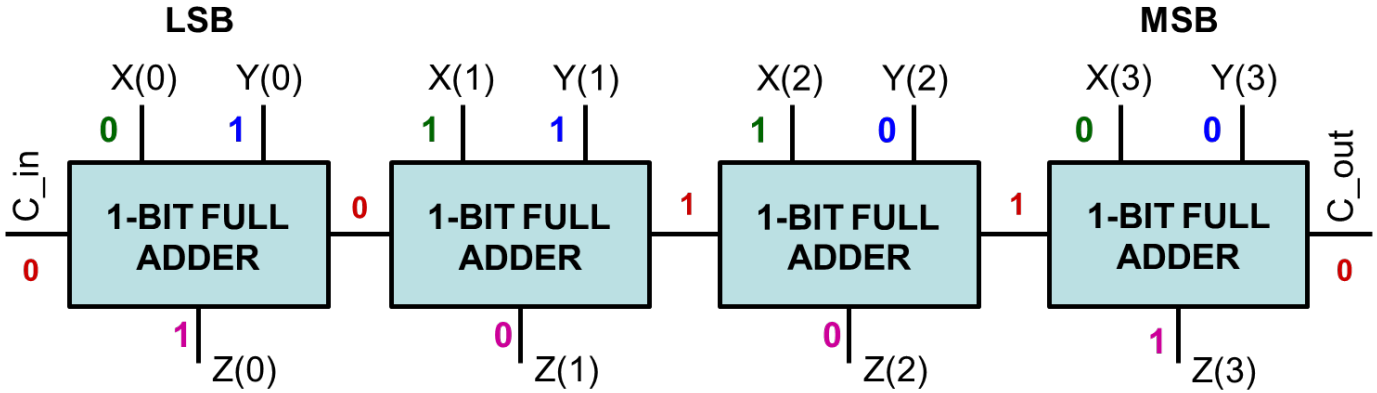
| X | Y | C_in | Z | C_out |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |




*Determine the Boolean function for the 1-bit adder (LAB 1).*

# Digital Design

## Example: 4-bit adder




$$\begin{array}{r} {}^{1}0{}^{1}1{}^{1}0 \\ 0011 \\ \hline 1001 \end{array} \begin{array}{l} + \\ = \end{array}$$

$$X = [X(3)\ X(2)\ X(1)\ X(0)]$$

$$Y = [Y(3)\ Y(2)\ Y(1)\ Y(0)]$$

$$Z = [Z(3)\ Z(2)\ Z(1)\ Z(0)]$$

# Digital Design

## Example: 4-bit adder

- Exploits **hierarchical design** with four 1-bit adders
  (how would you go about designing a 16-bit adder?)
- Performs an operation between signals fed by *parallel buses*. You can think of a bus as a "bundle of wires", a component that can carry words. It is called "parallel" because *all word bits are received by the circuit at once*.
  - A *word* is a collection of bits which are homogeneous in their meaning

    In our case the words X, Y, Z are all 4-bit words (represent binary numbers)
  - The opposite of *parallel* is *serial*

# Digital Design

**Summing up:**

- Digital design using FPGA has certain advantages in terms of versatility and cost
- There are standard steps to digital design using FPGA: Concept, Composition, Simulation, Synthesis, Implementation, and Hardware Testing
  - importantly, remember that *if the design is already unclear at the concept stage, things can only get worse*.
  - Complex logic functions are created by configuring the components: using LUTs inside individual CLBs and by combining multiple CLBs through appropriate routing / multiplexing.
  - Configuration is done using the hardware description language (HDL) which the FPGA synthesis tools convert and generate a bitstream file.

# Digital Design

**Summing up:**

- Last stages are carried out by the software almost automatically, but understanding the basic inner workings of a FPGA chip will help you successfully implement your designs. The key facts:
  - The number of available flip-flops per chip is large but not limitless
  - There is a relationship between routing and circuit performance
- Don't worry if this seems a bit abstract at this stage. The LABS will give you the opportunity to get plenty of hands-on experience.
- You should understand the operation of a full adder, which you will encounter in Lab-1.
- A multi-bit implementation of the adder has allowed us to raise some points about *hierarchical design*, buses, and *data words*.
  - these are simple yet essential concepts for the rest of the course

# Announcements

**Next week:**

- Lab-1
  - Install Vivado and make sure your PCs are ready to use in the labs. (In case of difficulty in installation, you can use the desktop computer in the lab). Follow the installation guide provided in Wattle.
  - Familiarize with Verilog syntax. You can refer to the *Appendix: VERILOG Syntax* of the ENGN3213 Reading Brick from 2008 (in Wattle -> Lectures and Other Teaching Resources).
  - Read the lab manual and try to attempt the tasks before attending your session.
  - Collect Basys-3 boards when you attend the lab next week
- Tutorial-1
  - Download Tutorial-1 problem document and try to attempt the problem questions.
  - Attend your tutorial session next week



# THANK YOU

## Acknowledgement

*These presentation slides are the modified version of slides prepared by Dr. Jihui (Aimee) Zhang based on the original version by Dr. Nicolo Malagutti*

## Contact:

**Amy Bastine**
School of Engineering
ANU College of Engineering, Computing and Cybernetics
The Australian National University

Office: B147, Brian Anderson Building (Bldg. 115)
Email: amy.bastine@anu.edu.au

Australian National University