High Level Synthesis

Ranim Tom

September 2, 2025

Contents

Todo list

Chapter 1

Introduction

1.1 HLS Overview

This report will talk about high level synthesis (HLS) desing using FPGA.
Unlike traditional hardware descirption language (VHDL, Verilog), HLS uses high level languages such as `C` and `C++` to design product using FPGA. The advanatage is the speed of learning compared to traditional language.

introduction

Introduction:

- *To adapt more the introduction*
- *Insert references and books*

1.1.1 FPGA Benefits and introducing HLS

FPGA offers some extra benefits, compared to some other computing platform such as multicore CPU and GPU. These are:

- Reconfiguribilty: the architecture of an FPGA can be adapted depending on the demand of the application
- Parallelism
- Less power compared to other platform for the same task

However, FPGA was suffering from serious drawback such as:

- The complexity of hardware language: learning them, and debugging their application
- Lack of standard libraries to accelerate development

This why HLS has comes and introduce the idea of using ***high level languages to design hardware modules***.

1.2 Development Environment and Hardware

The development environment used is from `Vivado` company, and the baord will be the `Basys3` from `Digilent`.

1.3 FGPA Concepts

As stated in section 1.1, HLS is about bringing programming concept to the hardware using FPGA.
In this section, we present a high level overview about the FPGA concept that we should now in order to do some HSL programming. The overview map is presented in Figure 1.1

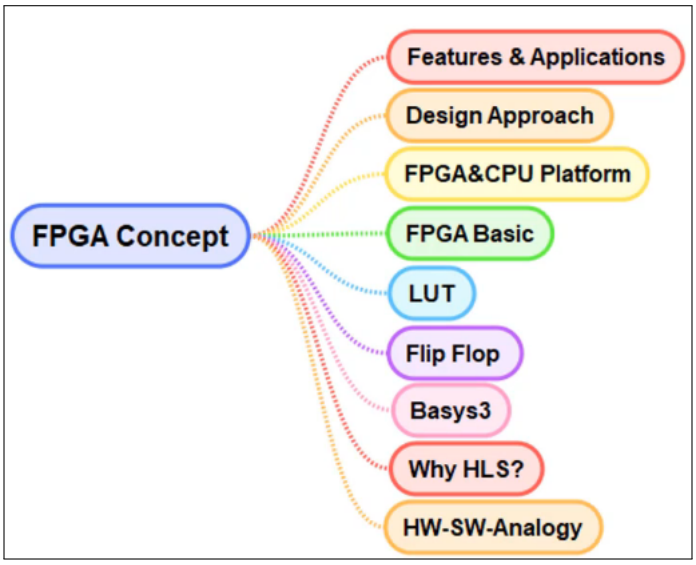

Figure 1.1: FPGA Concepts Overview

1.3.1 Design Level

FPGA desing is categorized in 3 different levels as shown in Figure 1.2.

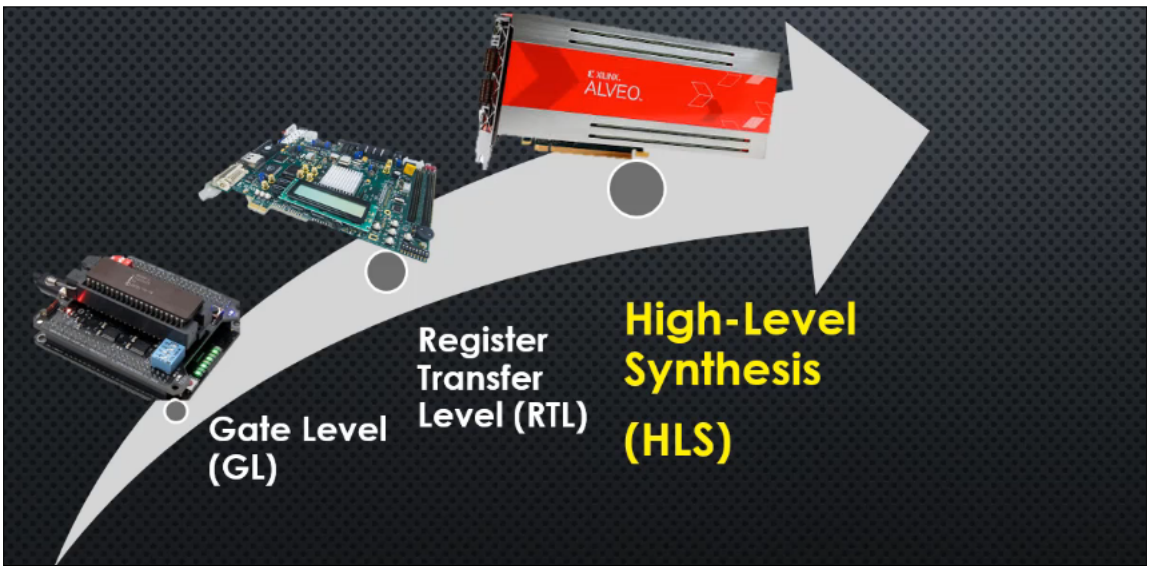

Figure 1.2: FPGA Design Levels

- Gate Level: using simple gates (AND,OR,$\cdots$). This allow hardware designers to implement small circuits on small FPGPA.

- RTL: with more advanced in resources and capacity on FPGA, RTL became the fundamental concept in desinging ciruit. RTL main languages are `VHDL` and `Verilog`.

  However, these languages are complex because they require deep hardware knowledge, and its very close to the gate level. It is the equivalent of `assembly` in the software world.

- HLS: it is using `C` and `C++` to map complex algorithm to the hardware, with low knowdlege on the hardware.

  The HLS is not simply a lanuage, but also an optimzation technique and some toolset, which can automatically generate the RTL code for us

Design Leve

Design Level:

- *To review and Adapt the design level points later*
- *see section 2, video 5, part 1*

1.3.2 FGPA and CPU

We hilight the differences between CPU and FPGA

- CPU: a computing platform with fixed architecture, with fixed hardware architecture, controlled by a set of commands or instructions.
- FPGA: also a computing platform, but with no dedicated architecture.

  Using HLS, each `C` or `C++` code has its own optmized hardware architecture, which leads high performance execution of some algorithm using FGPA.

Also, when CPU tends to translate the programming code into assembly code and executed *sequentialy*, whereas FGPA tools translate the code into a set of hardware block and execut the code in *parallel*.

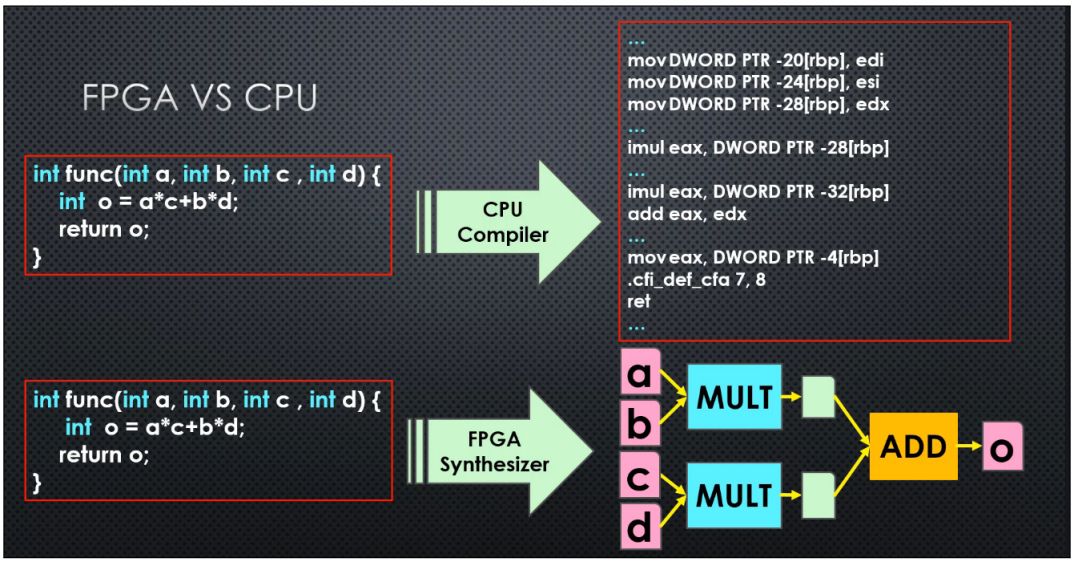

Figure 1.3: FPGA Vs CPU: Sequential vs Parallelism

PA vs CPU
allelism
Sequential

FGPA vs CPU Parallelism and Sequential:

- *Source: section 2, video 6, course 1*
- *To reinsert the picture where we illusrtate adding operation using CPU model (a register and an ALU)*
- *To read this model from the microcontroller book, how a program is executed on a CPU*

1.3.3 FPGA Structure

Internal FGPA structure is invisible to the user in HLS programming, but we explore the main components so we can undesrstand later the tools reports.

From a concept (or sofware point of view), an FPGA consist of 2 main layer (as shown in Figure 1.4):

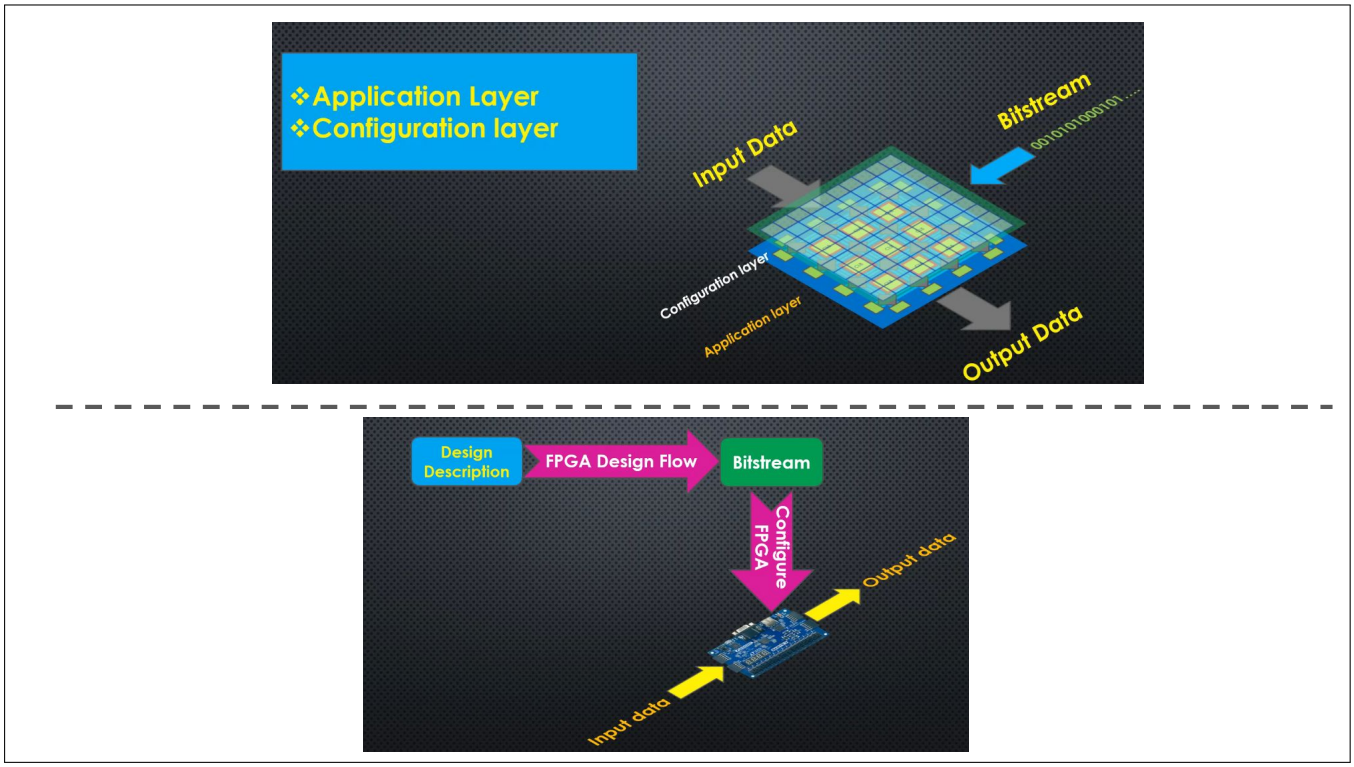

Figure 1.4: FPGA Structure: layer view

1. Application layer: contains all the hardware module to implement some algorithm

   Once this layer is configured we can apply the input data to obtain some output

2. Configuration layer: consist of memory cells to configure the application layer.

   The configuration data saved in this module is called *bit stream*, and it is generated by systhesis tools.

The role of sotware design suit is to transform the description language into bitstread, then use this bistream to impelemment some circuit using the FPGA, as shown in 2nd picture of Figure 1.4

synthesis tool and bitstream: *To review later this idea after doing some applications.*

synthesis to and bitstrea

Now from a physical side, an FGPA composition is shown in Figure 1.5.

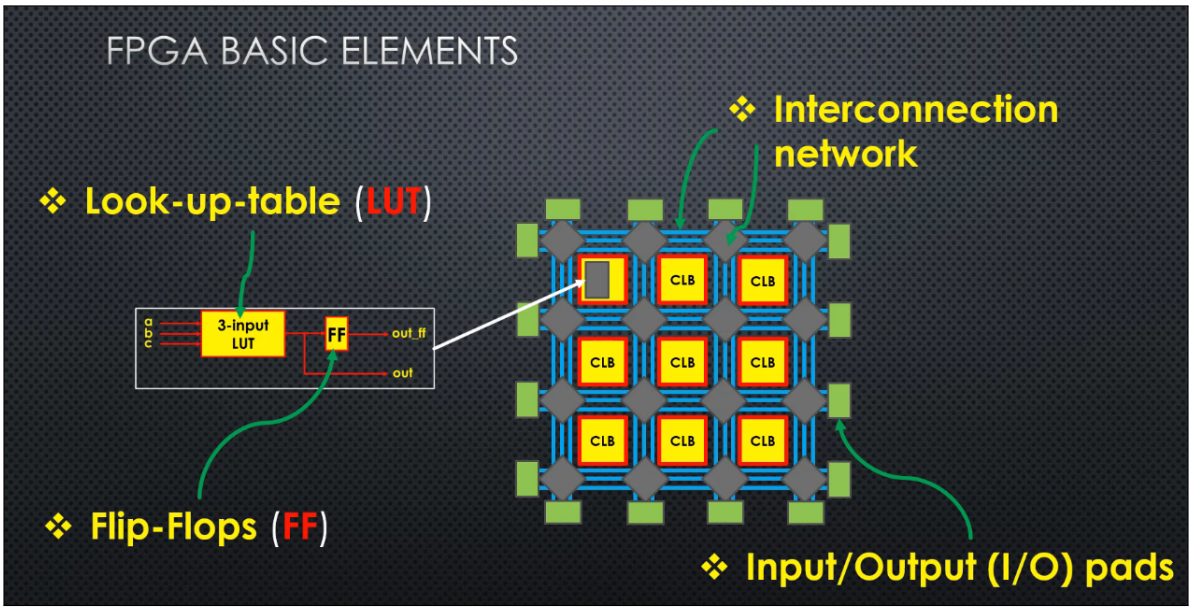

Figure 1.5: FPGA Structure: pyhsical composition

- A look up table (LUT): to implement some logic function
- A flip flop: a register which saves the output of LUT
- IO ports: for data manipulation
- Interconnection network: to connect different resources of FGPA

1.3.4 Look Up Table

A LUT keeps the output of some logic function for all the input combinations. Examples are shown in Figure 1.6

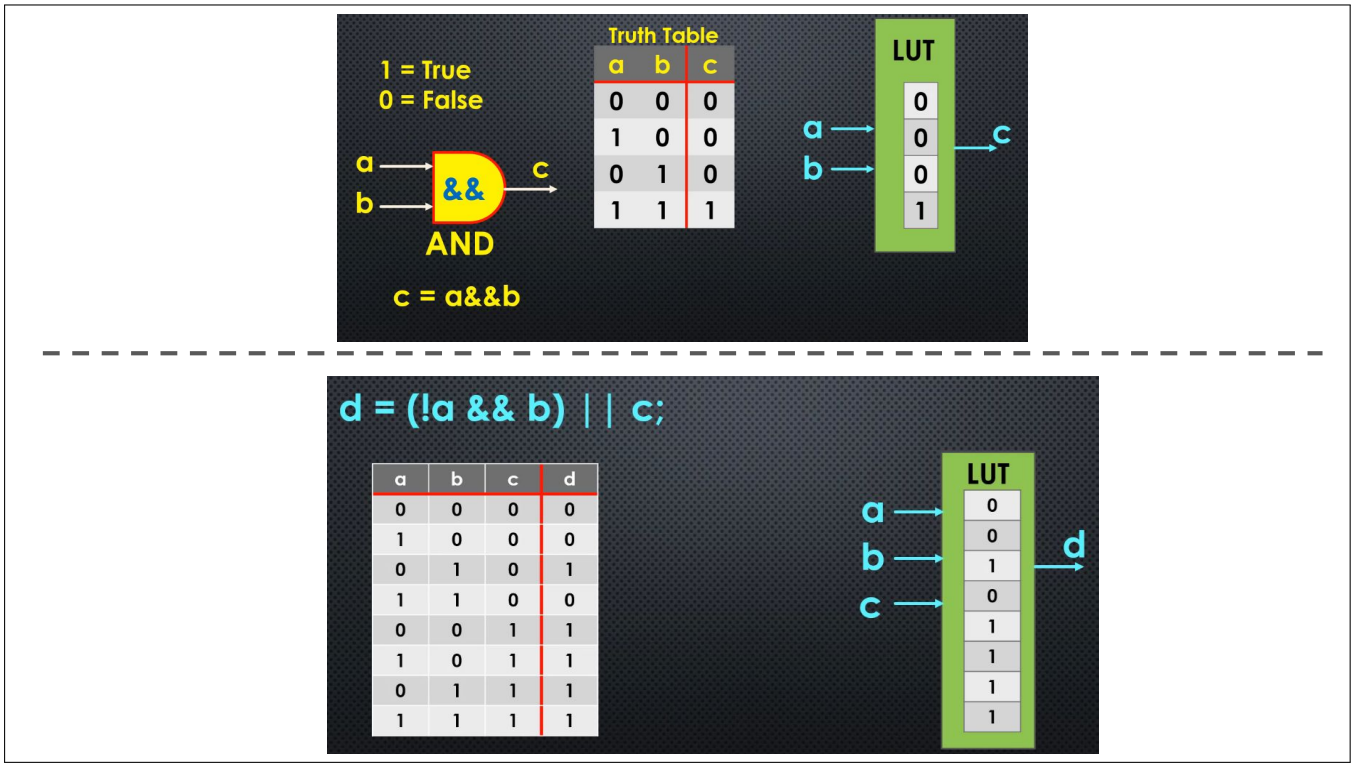

Figure 1.6: LUT Examples

- For a basice `AND` gate, a LUT needs only 2 input to track down different output
- For 2nd example, we have 3 inputs leading to 8 outputs. The LUT need 3 inputs to track down the output.

1.3.5 LUT in FPGA

Now in FPGA, a LUT has a configuration memory that keeps all the possible outputs of a logic function.
It also has a set of inputs to select the proper cell in the memory corresponding to the inputs logic.

LUT in FG

LUT in FGPA:

- *Source: course 1, section 2, video 8*
- *To redo the writing later*
- *Main idea*
  - *A LUT in FGPA uses multiplexer circuite to select the propoer output nedded, depending on the selections line of the MUX circuit*
  - *Write down the idea of MUX later*

1.3.6 Flip Flop

In addition to LUT, an FGPA consist also of flip flop, which is a register which can save output into memory. By having flip flop, we can implement design which depends also on the *past history*, and not only of the present states of the input as in pure combinational logic.

1.4 FPGA Example

Main idea:

- An FPGA is consisting of ***many slices***
- Each slice consist of LUT, flip flop and some basic logic gates

Figure 1.7 present a slice in Xilinx 7-series, which is one of the most common Xilinx FPGA families.

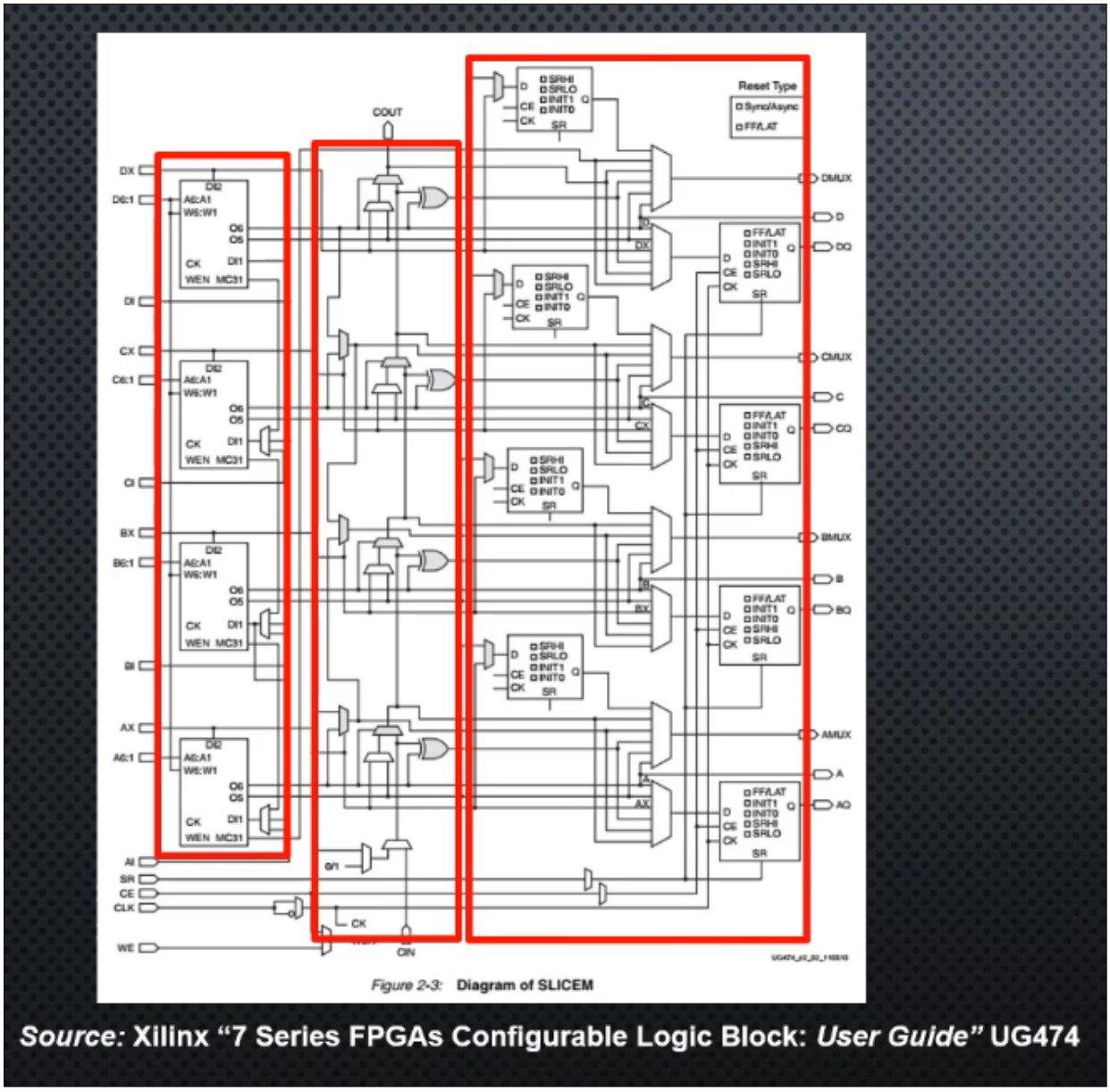

Figure 1.7: FPGA Slice Example

It has three layers of elements:

1. 6 LUT
2. An arithmetic logic cicruit
3. Flilp flop
4. A MUX

Now a ***set of 2 slices and a switch matrix*** to connect the slices to the routing on the FPGA is called a ***configurable logic block*** or ***CLB*** for short. ***An FPGA*** can be considered ***as a 2D array of CLBs*** as shown in Figure 1.8.

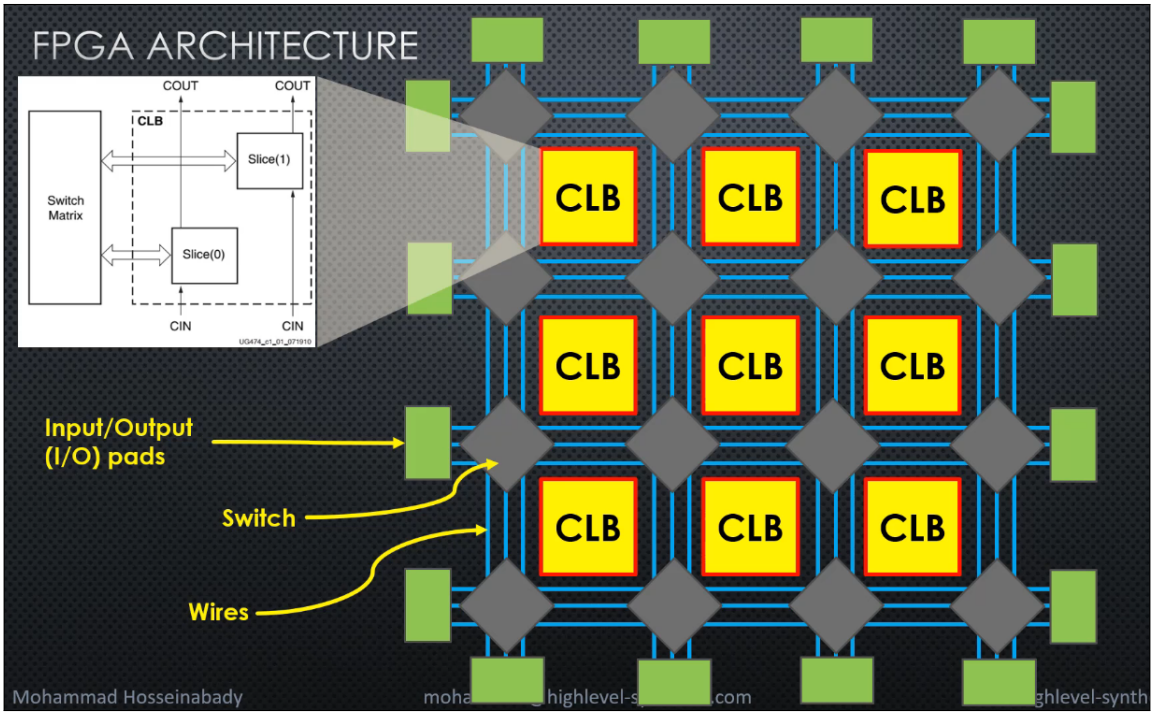

Figure 1.8: FPGA as a 2D array of CLB

Different CLB are connected through switches and wires to form different complex logic circuit.

Now an to a better efficient circuit to impement complex algorithm, an modern FGPA structure ***comes with different additional elements*** such as:

- Block RAM (BRAM) to save large amount of data
- PLL for driving fgpa at different clock rate
- High speed serial transeivers
- Multiply and Accumulate DSP block

The FGPA ship along with these extra elements form an development board. The Basys 3 development board will be explored next in 1.5.

1.5 Basys 3 Board Anatomy

Figure 1.9 presents Basys 3 development board, which we will use throughout our development HLS software.

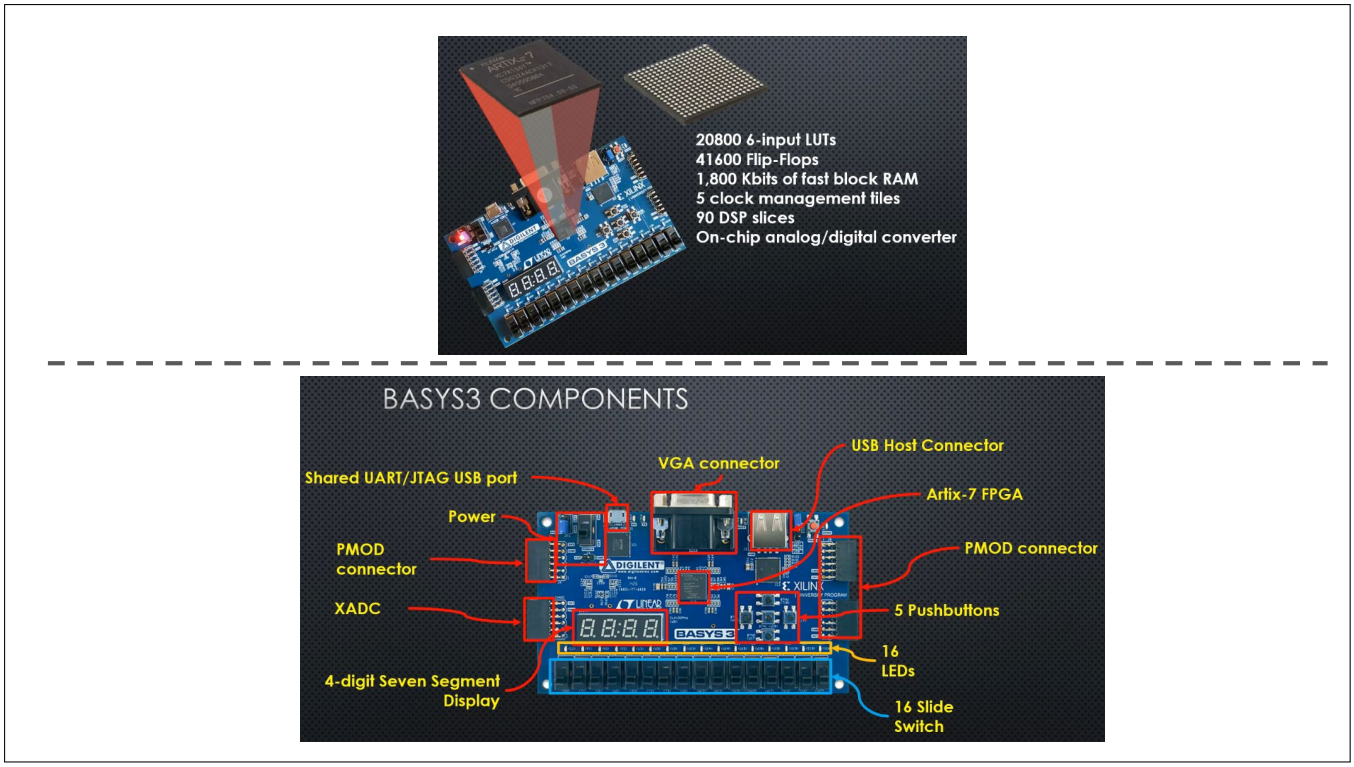

Figure 1.9: Basys 3 board

- The LED are arranged in a common cathode structure
- The LED of the board are connected to a 330 ohm resistors to prevent damage in case of short circuits

cathod and anode: cathod and anode

- *To review these concepts later, and insert them into an appendix*

1.5.1 Power Source

Figure 1.10 present the power circuit for the fpga.

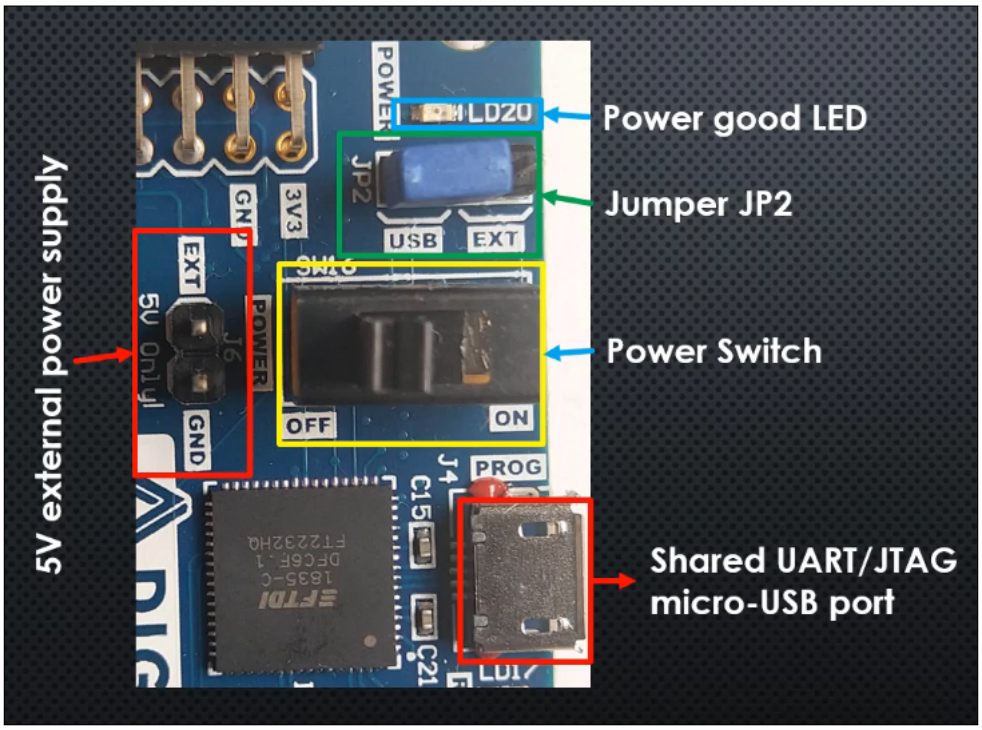

Figure 1.10: Basys 3 power circuit

The board can take power either from shared UART/JTAG port, or from 5v external board. A jumper j2 is used to select which source we choose, and a power switch can be used to turn on or off the board. The `LD20` Led is used to indicate if power is at normal state throughout the board.

1.5.2 FPGA Configuration

After powering up the FPGA, it must be configured. We have 3 modes for configuration as shown in Figure 1.11.

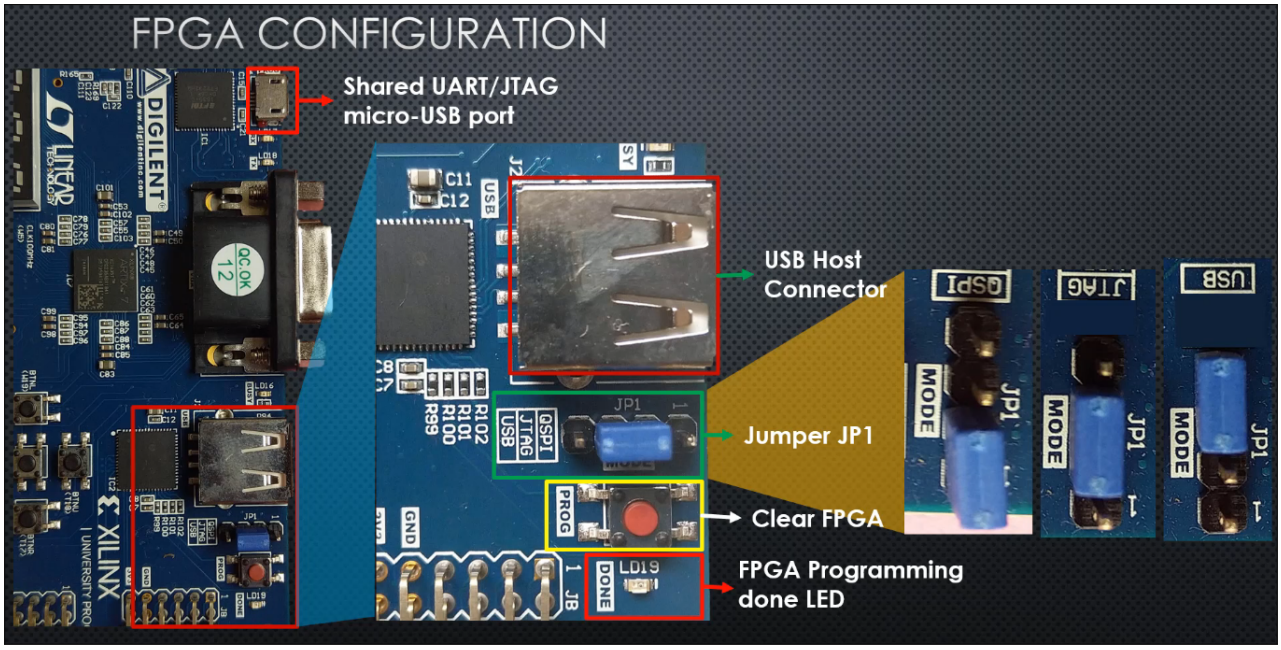

Figure 1.11: Configuration Modes

1. Through the shared JTAG usb port
2. Using a file from a usb connector
3. Using the SPI board

FPGA configuration: FPGA configuration

- *Source: course 1, seciton 2, video 10*
- *To review this section later after doing some examples*

Once the programming done, the `LD19` will be on to indicate firmware flashing has been done.

1.6 Hardware Software Analogy

Since we will desing hardware modules using software, it is a good idea to establish some analaogy between software and hardware. This help us to have hardware description in the eyes of software.
The analogy between hardware modules and high level langugae contians some similiarities and also some differences.

?? present the similarities between standard `C` function and some hardware module to impelement.

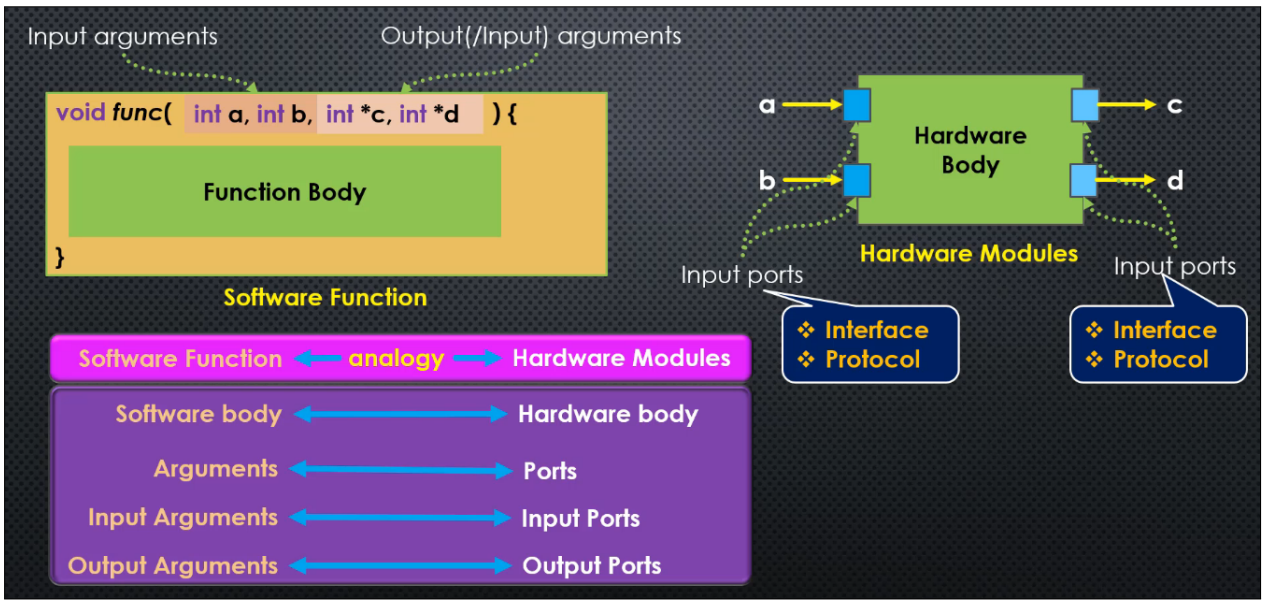

Figure 1.12: Similarities between software and hardware

Figure 1.13 present the differnces

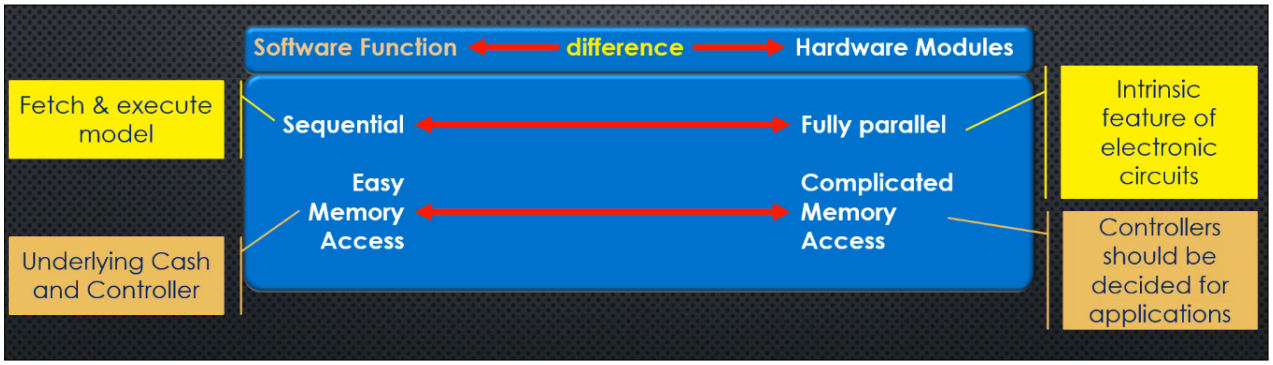

Figure 1.13: Differences between software and hardware

Now the take away is that HLS hide these details (these differences) through compiler directives using pragma, which guide synthesis tool to add these differences.

Soft Hard differences: Soft Hard differences

- *Source: course 2, section 2, video 12*
- *This is a very important topic*
- *To review the memory model inside a traditional cpu, the fetch exectue model, once I finishi this course, and the course of microcontroller drivers*
  - *Also once finished with this, to see how the memory model interact in FPGA, the protocol idea and implemeting an interface, I didn't quiet understood them yet*
  - *I quote: However, ports in a hardware module should implement a specific interface with the associated protocols to access a memory cell or communicate with other modules.*

1.7 Environment Setup

ado Steps *Vivado Steps*

- *Source: section 3, video 13 and 14*
- *Talk about different stages of design, from C programming, simulation for debuggign, types of simulation,· · ·*
- *To much details for now, I will get back to it once I understand more and do some concrete examples*