INTRODUCTION TO FPGA PROGRAMMING

Lesson 03: Boolean Algebra, Look-up Tables and IOBs

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

September 2024

MAX-PLANCK-INSTITUT
FÜR PHYSIK

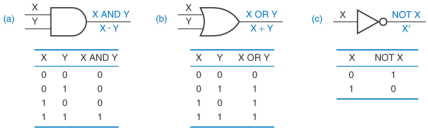
BOOLEAN ALGEBRA

- Digital logic hides the pitfalls of the analogue world by mapping the infinite set of real values into two subsets corresponding to just 2 possible logic values: 1 and 0, or high and low voltages.
- With this assumption, we can use Boolean algebra to describe the operation of well behaved 0s and 1s in a circuit
- Instead of multiplication or division, Boolean algebra employs operations like AND, OR or NOT.

LOGIC GATES

- Logic Functions are represented in digital systems with *Logic gates*.
- Logic gates can be described with *truth tables*
  - A table listing all possible input and output combinations for a boolean algebra equation
- Logic Gates are implemented in VHDL using logic operators AND, OR, NOT, NAND, NOR and XOR.

```
A <= X AND Y;
B <= X XOR Y;
```



X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

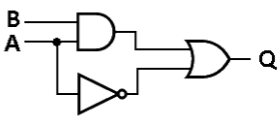
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

COMBINING GATES

- Gates can be combined to create different boolean equations
- VHDL follows the following logical operator order
  - AND > OR > NAND > NOR > XOR > XNOR
- Round parenthesis can be used to force a different order

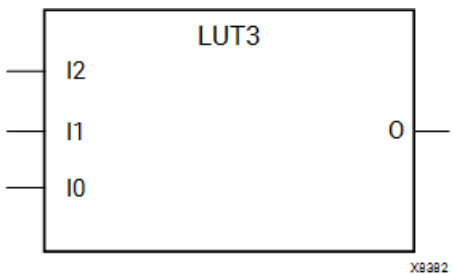
```
Q <= A AND B XOR A
```

Truth Table – A*B + A'		
Input A	Input B	Output Q
0	0	1
0	1	1
1	0	0
1	1	1



THE LOOK-UP TABLE

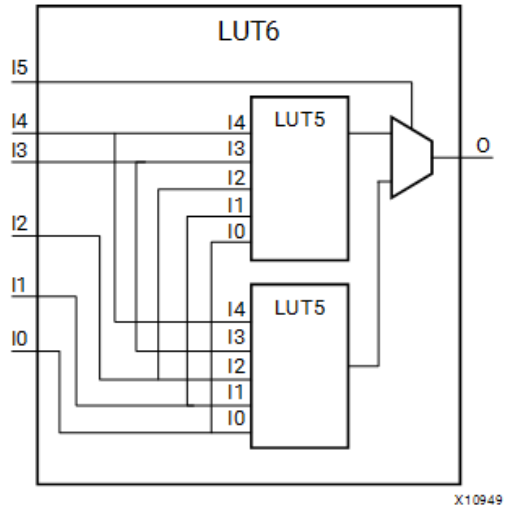
- On the FPGA, the are no physical logic gates that you can plug together to form a boolean algebra equation
- **Look-Up Tables** (LUTs) replace all functionalities of logic gates
- LUTs are devices that can be configured to implement any truth table
- They are defined by their number of inputs. E.g. LUT3



Inputs			Outputs
I2	I1	I0	O
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]
INIT = Binary equivalent of the hexadecimal number assigned to the INIT attribute			

XILINX 7-SERIES LUTS

- On 7-series Xilinx/AMD FPGAs, LUT6 are available
- More info, here

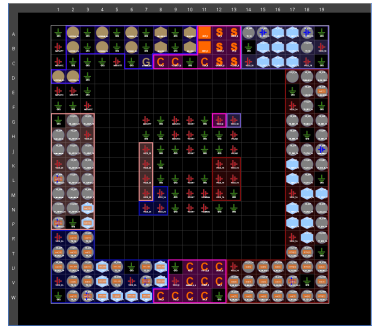


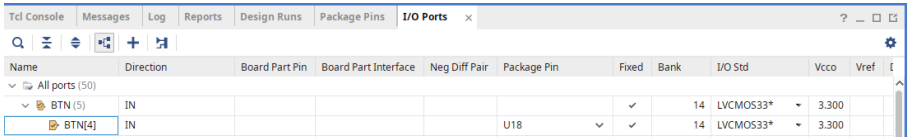
PIN CONSTRAINTS

- Logic signals in your design top module must be mapped to physical pins on the FPGA
- This ensures correct signal routing and interface with the external world
- If not specified, Vivado will place them randomly
  - It will fail, when creating a bitstream at Design Rule Check (DRC) because voltage standards are not defined
- Two options to constraint your design:
  - Using the I/O Planning graphic interface
  - With constraint files (`.xdc` for Vivado)

I/O PLANNING

- To access the I/O Planning, open the RTL analysis elaborated design
  - Then open Layout->I/O Planning
- The diagram shows the all the available pins on the device
- In the bottom panel, you can see the ports of your design, and can assign to them to a particular pin
  - Don't forget to define the IOSTANDARD
  - For our board, the standard is always LVCMOS33 (Low-Voltage CMOS 3.3V)





XDC FILES

- What the GUI actually does is writing the constraint in `.xdc` file
- XDC files follow the Tcl semantic and contain list of commands to be executed by Vivado
- Tcl (Tool Command Language) is a powerful interpreted language with a simple syntax
  - Most IDE including Vivado have a Tcl console. All Vivado commands are written in Tcl.
  - Tcl is case-sensitive, on the contrary of VHDL
- Both timing and physical constraint can be written in an XDC[1]
- AMD Xilinx user guide on Using Constraint here

[1] You can use also `.tcl` file to specify your physical constraints. They follow the same syntax.

SPECIFYING PHYSICAL CONSTRAINTS

- Physical constraints are properties of any object in your design
- Properties are set using the `set_property` command
- Design interface are selected using the `get_ports <port_name>` command
  - Port name can also be a wildcard to set properties of multiple ports at the same time
- You can also define variables inside the `xdc` with the `set <variable_name> <declaration> command`

```
# set_property syntax
set_property <property> <value> <object_list>
# Example
set rst_port [get_ports {rst}]
# Variable content is accessed with the $ symbol like in bash
set_property PACKAGE_PIN A1 $rst_port
```

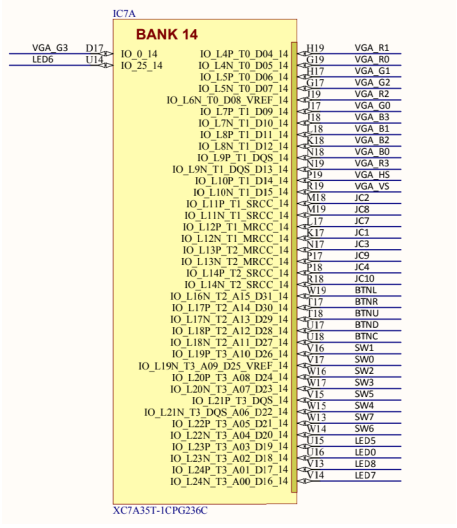
USEFUL XDC COMMANDS

- Some of these commands need to have the design open (RTL, Synthesis or Implementation)
- Each command has an help message. `<command_name> -h`

```
# get_ports: Get a list of ports in current design. If no argument, returns all ports
get_ports [<options>] [<port_name(s)>]
# list_property: List all the properties for a particular object
list_property [options] <object>
# list_property_value: Get a list of valid value for a specific property and object
list_property_value [options] <property> <object>
```

WHERE DO I GET INFORMATION ABOUT THE PIN MAPPING?

- How the FPGA pins are connected into your board is a choice of the board designer
- The FPGA developer should refer to the board schematic to check the pin assignment
- Basys3 board schematics: Page 6



SETTING PIN CONSTRAINTS FOR BASYS3

- We know have all the information to map the ports of our design to the physical pins on the Basys3 board
- For each port, we must define the `PACKAGE_PIN` and `IOSTANDARD` properties
  - Get the `PACKAGE_PIN` from the schematics
  - The `IOSTANDARD` for all Basys3 ports is `LVCMOS33`

```
# Example
set_property PACKAGE_PIN U16 [get_ports {led}]
set_property IOSTANDARD LVCMOS33 [get_ports {led}]
```

LAB 04: WIRING SWITCHES TO LEDS

LAB 05: IMPLEMENT A FULL ADDER

LAB 06: HIERARCHICAL DESIGN (A RIPPLE CARRY ADDER IMPLEMENTATION)

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

- nandland.com
- docs.amd.com