FPGA Prototyping Using Verilog

Session 1 – Introduction to FPGA and Verilog HDL

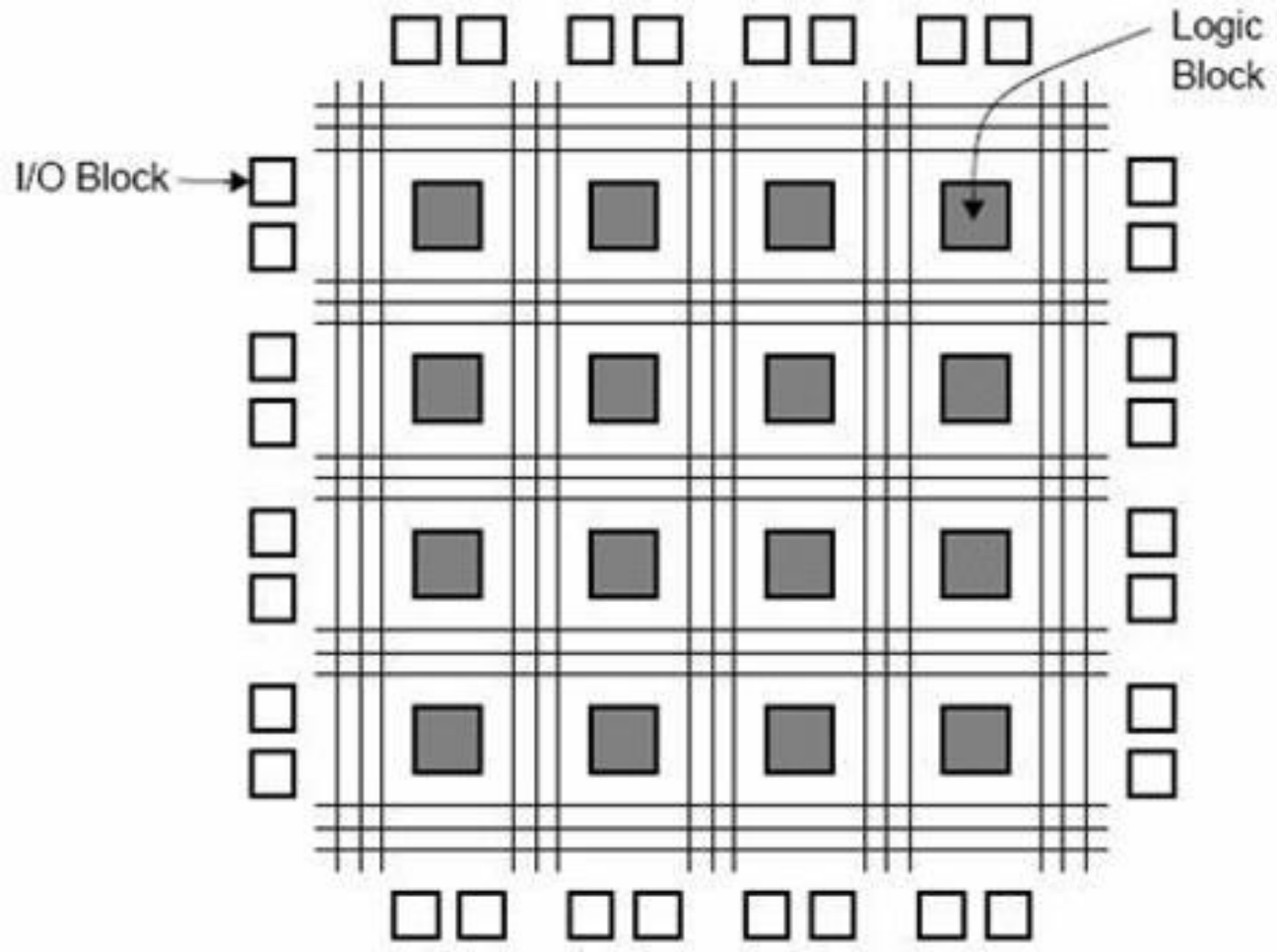
Dated: January 3rd, 2021

Instructor: Muhammad Saad Bin Riaz

What is FPGA?

- FPGA – Field Programmable Gate Array
- Field Programmable means that device can be programmed in the *Field* after Fabrication
- Gate Array means that device has huge numbers of *Logic Gates*
- Typically, an FPGA consists of a large number of Configurable Logic Blocks (CLBs) connected via programmable interconnects
- CLBs are based on Look-Up Table (LUT) and Flip Flop pairs
- LUT is the basic reconfigurable part – can be used to implement any logic expression
- Programmable Interconnect used to connect Logic Blocks with each other and with I/O Blocks
- I/O Blocks are used to provide off-chip connections

Example FPGA



FPGA vs ASIC

- ASIC – Application Specific Integrated Circuits, cannot change their functionality once fabricated
- FPGAs are reconfigurable but ASICs are not
- FPGAs have lower development cost and faster Time-to-Market, but ASICs usually have higher development cost and longer TTM
- FPGAs can achieve reasonable performance, ASICs have the highest performance
- In terms of Unit Cost and Power Consumption, ASICs are clear winner as they out-perform FPGAs
- Typically, FPGAs provide a rapid prototyping platform for ASIC development

What are HDLs?

- HDL stands for Hardware Description Language
- High level programming language which actually synthesizes hardware
- Constructs are inherently parallel
- Supports Modular programming approach to define hardware
- Different programming Paradigm than C-like software programming
  - C programs are sequential whereas HDLs are parallel
- Most commonly used HDLs are Verilog and VHDL
  - We will be focusing on Verilog HDL in this course

Introduction to Verilog HDL

- Developed in mid 1980s and then revised in 2001 as Verilog 2001
- IEEE Standard 1364
- Intended for describing/modelling digital systems
- Somewhat C-like syntax but semantics is concurrent
- Modular programming language arranged hierachically to design complex systems
- Subset of keywords are synthesizable, known as Register Transfer Level (RTL) code
- Will focus more on the hardware prototyping using Verilog than just learning the language keywords

Abstraction Levels

- Verilog supports 4 abstraction levels
  - Behavioural – Highest level of Abstraction and defines a cicuit on the basis of its Behaviour
  - Data Flow – Defines the circuit on the basis of data flow
  - Gate Level – Defines the circuit with the help of primitive gates like AND, OR, NOT etc
  - Switch Level – Defines the behaviour of the circuit on the basis of ON and OFF states of switch
- Behavioural and Data flow abstractions are commonly used
- Theoretical concept only, seldom used in practice

Programming Strategies

- Typically, two strategies/approaches exist
  - Top Down – Starting from the Top breaking down the tasks into smaller tasks until a reasonable granularity is achieved
  - Bottom Up – Starting from leaf modules, combining them together to implement a complex system
- In practice, a combination of both are used in Digital Design
- Verilog supports both of these approaches
- Again, a theoretical concept only but useful in understanding the design flow

Basic Data Types

- Four Value System – 0 (Low), 1 (High), x (Uninitialized), z (high impedance)
- Data Type Groups – Net groups (wire) and Variable groups (reg, integer, real, time and realtime)
- Commonly used synthesizable data types are wire and reg

```
wire [7:0] data; // net of 8-bits named as data
reg out; // register of 1-bit named as out
wire enable, reset; // single bit enable and reset wire
reg [15:0] array [7:0]; // 2 dimensional array of size 16x8
```

Number Representation

- General representation – [sign] [size] '[base] [value]
- Base Representation – binary (b or B), octal (o or O), decimal (d or D) and hexadecimal (h or H)
- Value term specifies the actual value in corresponding base
- Underscore (_) can be used in between value for better readability
- Size specifies the size of the number in bits and is optional
- Examples: 10'd100, 32'h0000_ABCD, 1'b1, 'b0001, 'd50, -25, -32'd5

Operators

- There are about two dozen operators in Verilog
- Logical – Returns only single bit result e.g. &&, ||, !
- Bitwise – Returns same number of bits as operands e.g. &, |, ^, ~
- Reduction – Reduces multibit signals into single bit e.g. &, |, ^
- Relational – Returns relation between two operands e.g. <, <=, >, >=
- Equaltiy – Checks for equality or non-equality e.g. ==, !=
- Arithmetic - +, -, *, /, %, ** (exponentiation)
- Conditional – Ternary operator (? :)

Ports Declaration

- There are three types of ports in Verilog
  - input – Declares a port as input
  - output – Declares a port as output
  - inout – Declares a port as bidirectional
- General format – <direction> <data_type> <size> <name>

  input wire [7:0] data, addr, // declares inputs data and addr of 8-bits

  output reg [31:0] count, // declares 32-bit reg count as output
- If data_type is not mentioned then it is wire by default
- If size is not mentioned, it is 1-bit by default
- Inputs are always wire, outputs can be either reg or wire

Program Body

- Verilog program consists of module, which includes:
  - Parameters declaration (will be covered later)
  - Ports declaration
  - Signals declaration
  - Continuous Assignments using assign statement
  - Procedural blocks using always statement
  - Module instantiation
  - Comments (Under-rated but extremely important)

Simple Example

```
// Module Declaration
module and_2x (
  input op1, op2, // single bit inputs op1 and op2
  output out      // single bit output out
);

  // assign keyword is used for continous assignment
  assign out = op1 && op2; // two operands are ANDed together and result is stored in out

endmodule
```

References

- https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html
- https://en.wikipedia.org/wiki/Verilog#Verilog_2001
- https://medium.com/verilog-novice-to-wizard/abstraction-levels-in-verilog-2f663786b03f
- http://www.asic-world.com/verilog/intro1.html
- FPGA Prototyping by Verilog Examples – Xilinx Spartan 3 version, by Pong P. Chu

THANK YOU
Any Questions???