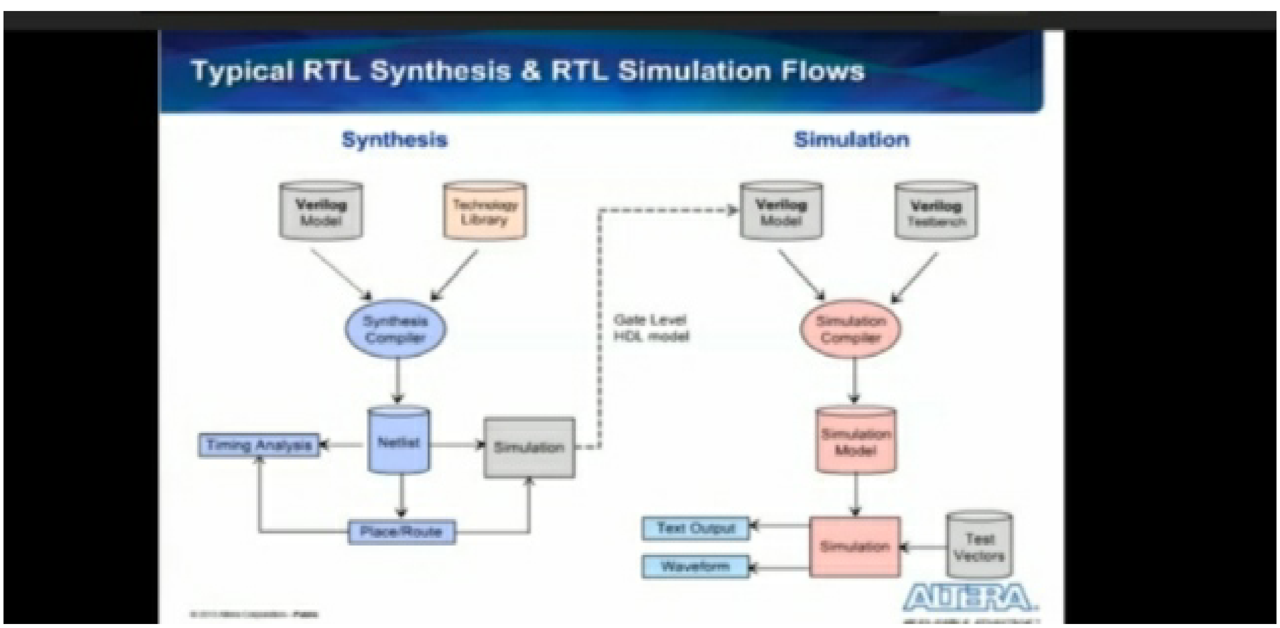
DAILY ASSESSMENT REPORT

Date:	02/06/2020	Name:	Abhishek
Subject:	Digital Design Using HDL	USN:	4AL17EC001
Topic:	1] FPGA Basics: Architecture, Applications and Uses 2] Verilog HDL Basics by Intel 3] Verilog Testbench code to verify the design under test (DUT)	Semester & Section:	6th 'A'
Github Repository:	Abhishek-online-courses		

FORENOON SESSION DETAILS

Image of session



Report

FPGA Architecture

- A basic FPGA architecture (Figure 1) consists of thousands of fundamental elements called **configurable logic blocks** (CLBs) surrounded by a system of programmable interconnects, called a fabric, that routes signals between CLBs. Input/output (I/O) blocks interface between the FPGA and external devices.
- Depending on the manufacturer, the CLB may also be referred to as a logic block (LB), a logic element (LE) or a logic cell (LC).

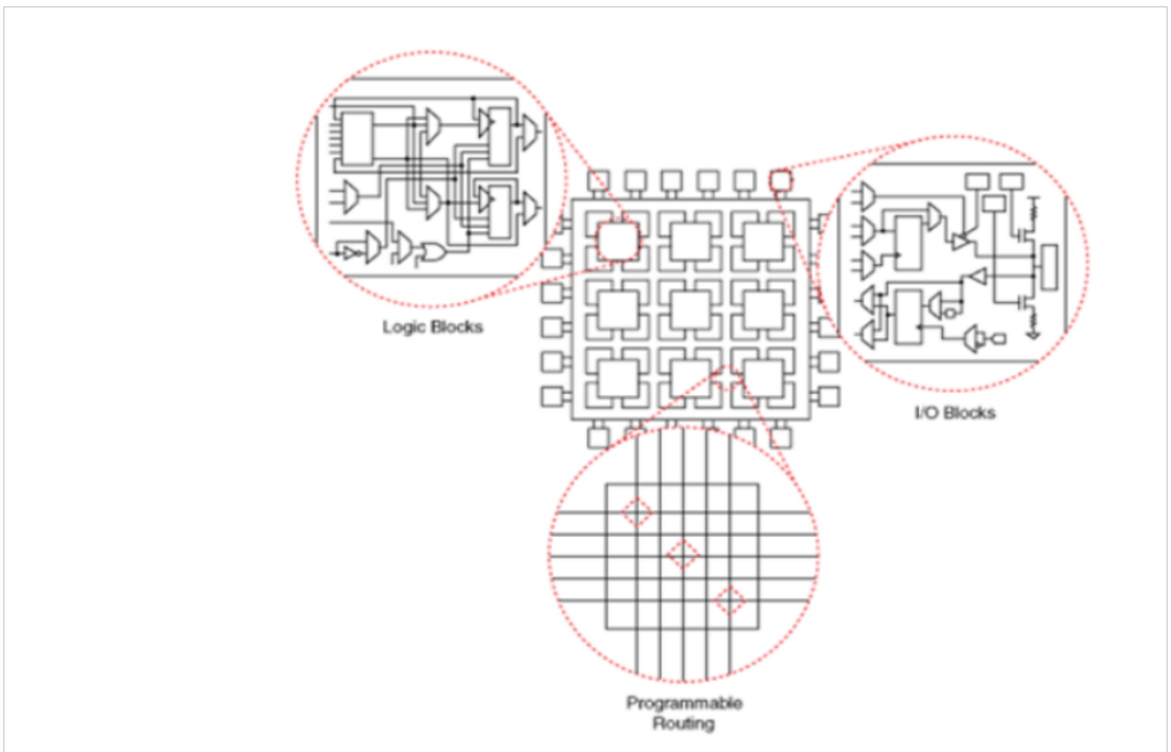


Figure 1: The fundamental FPGA architecture.

- An individual CLB (Figure 2) is made up of several logic blocks. A **lookup table** (LUT) is a characteristic feature of an FPGA. An LUT stores a predefined list of logic outputs for any combination of inputs: LUTs with four to six input bits are widely used. Standard logic functions such as multiplexers (mux), full adders (FAs) and flip-flops are also common.

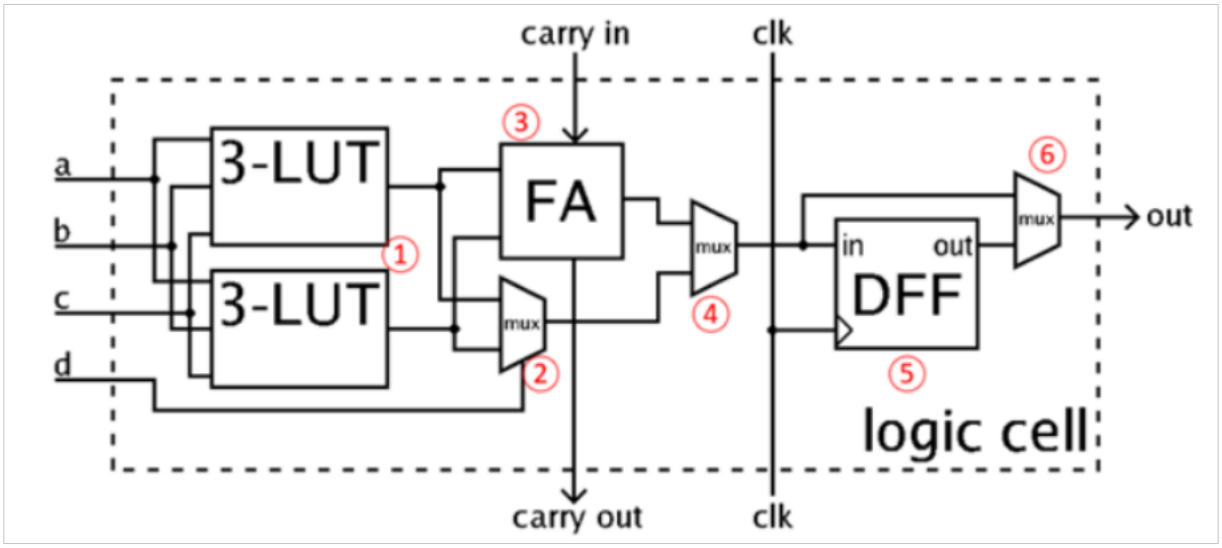


Figure 2: A simplified CLB: The four-input LUT is formed from two three-input units.

- The number and arrangement of components in the CLB varies by device; the simplified example in Figure 2 contains two three-input LUTs (1), an FA (3) and a D-type flip-flop (5), plus a standard mux (2) and two mux's, (4) and (6), that are configured during FPGA programming.

- This simplified CLB has two modes of operation. In normal mode, the LUTs are combined with Mux 2 to form a four-input LUT; in arithmetic mode, the LUT outputs are fed as inputs to the FA together with a carry input from another CLB. Mux 4 selects between the FA output or the LUT output. Mux 6 determines whether the operation is asynchronous or synchronized to the FPGA clock via the D flip-flop.

- Current-generation FPGAs include more complex CLBs capable of multiple operations with a single block; CLBs can combine for more complex operations such as multipliers, registers, counters and even digital signal processing (DSP) functions.

Verilog HDL Basics

- **Verilog** is a **HARDWARE DESCRIPTION LANGUAGE** (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using HDL we can describe any digital hardware at any level.
- Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

- Verilog supports a design at many levels of abstraction. The major three are:
- **Behavioral level**
  - ✓ This level describes a system by concurrent algorithms (Behavioral). Every algorithm is sequential, which means it consists of a set of instructions that are executed one by one. Functions, tasks and blocks are the main elements. There is no regard to the structural realization of the design.
- **Register−Transfer Level**
  - ✓ Designs using the Register−Transfer Level specify the characteristics of a circuit using operations and the transfer of data between the registers. Modern definition of an RTL code is "Any code that is synthesizable is called RTL code".
- **Gate Level**
  - ✓ Within the logical level, the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (basic gates). Gate level modelling may not be a

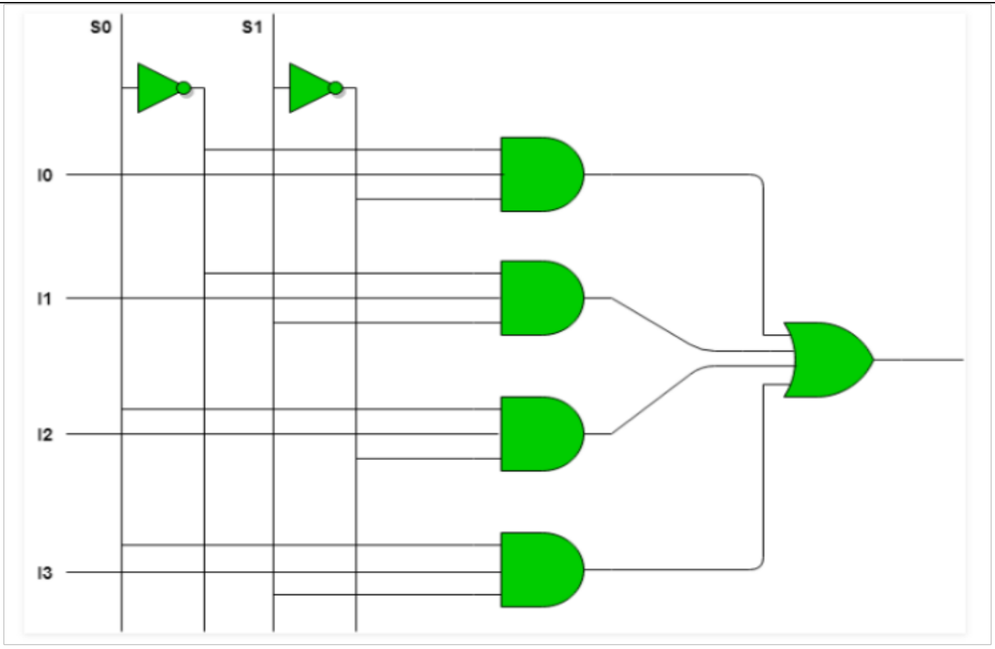
right idea for logic design. Gate level code is generated using tools like synthesis tools and his netlist is used for gate level simulation and for backend.

- Some of the operators used in Verilog HDL
  - ✓ **Arithmetic Operators** - These operators perform arithmetic operations (+, -, /, *, %).
  - ✓ **Relational Operators** - These operators compare two operands and return the result in a single bit, 1 or 0 (==, !=, >, <, >=, <=).
  - ✓ **Bit-wise Operators** - Bit-wise operators which are doing a bit-by-bit comparison between two operands (&, |, ^, ~, ^~).
  - ✓ **Logical Operators** - Logical operators are bit-wise operators and are used only for single-bit operands. They return a single bit value, 0 or 1 (!, &&, ||).
  - ✓ **Reduction Operators** - Reduction operators are the unary form of the bitwise operators and operate on all the bits of an operand vector (&, |, ~&, ~|, ^, ~^).
  - ✓ **Shift Operators** - Shift operators, which are shifting the first operand by the number of bits specified by second operand in the syntax (>>, <<).

Task (DAY - 2)

Implement a 4:1 MUX and write the test bench code to verify the module

Logic circuit for 4:1 MUX

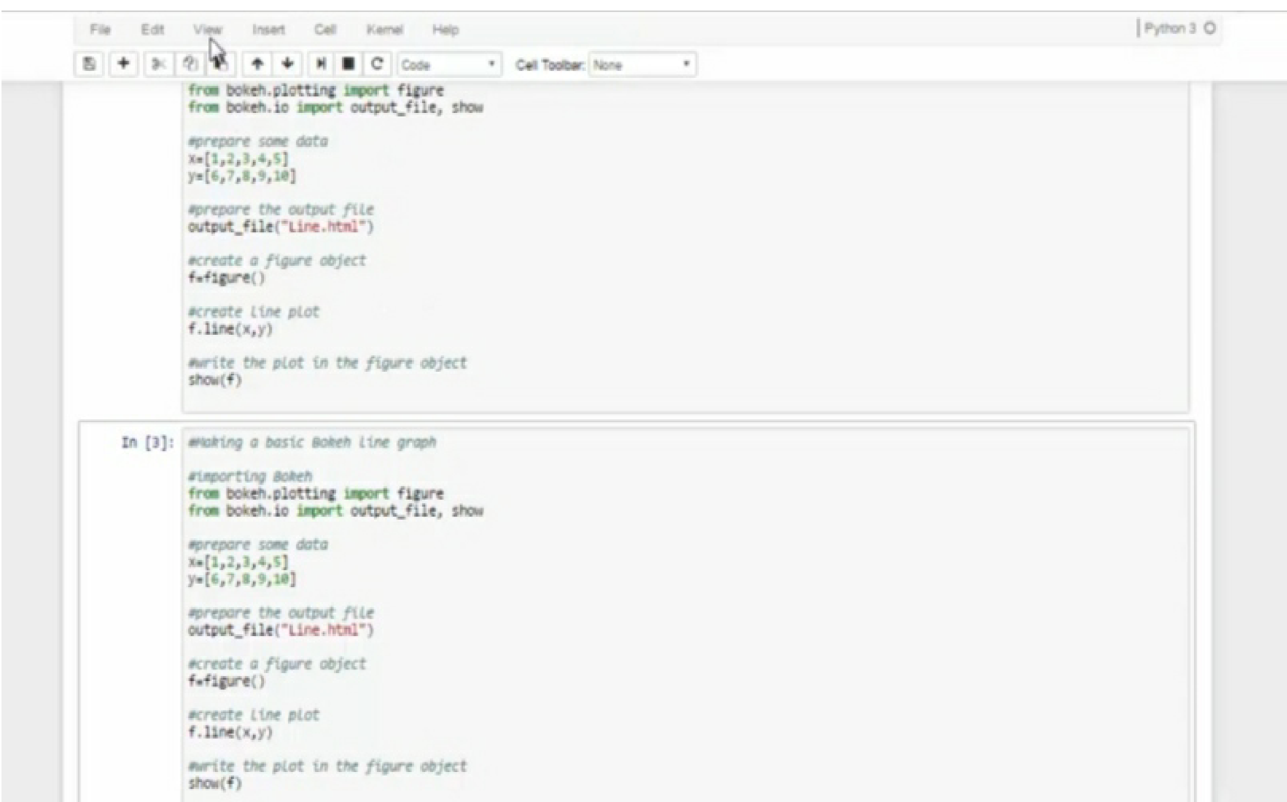


Verilog code:

```
module m41 (input a,

input b,

input c,

input d,

input s0, s1,

output out);



 assign out = s1 ? (s0 ? d : c) : (s0 ? b : a);



endmodule
```

Date:	02/06/2020	Name:	Abhishek
Course:	The Python Mega Course: Build 10 Real World Applications	USN:	4AL17EC001
Topic:	1] Interactive Data Visualization with Bokeh	Semester & Section:	6th 'A'
Github Repository:	Abhishek-online-courses		

AFTERNOON SESSION DETAILS
Image of session



Report

Interactive Data Visualization with Bokeh

- **Bokeh** is a data visualization library in Python that provides high-performance interactive charts and plots. Bokeh output can be obtained in various mediums like notebook, html and server. It is possible to embed bokeh plots in **Django** and **flask apps**.

- Bokeh provides two visualization interfaces to users:

    - ✓ **bokeh.models** : A low level interface that provides high flexibility to application developers.

    - ✓ **bokeh.plotting** : A high level interface for creating visual glyphs.

- Building a visualization with Bokeh involves the following steps:

    - ✓ Prepare the data.

    - ✓ Determine where the visualization will be rendered.

    - ✓ Set up the figure(s).

    - ✓ Connect to and draw your data.

    - ✓ Organize the layout.

    - ✓ Preview and save your beautiful data creation.

- Some of the functions used under **bokeh** library:

    - ✓ The **figure ()** function under bokeh.plotting is used to create a new Figure for plotting. A subclass of Plot that simplifies plot creation with default axes, grids, tools, etc.

- ✓ The **output_file ()** function configures the default output state to generate output saved to a file when **show ()** function is called.
- ✓ The **Show ()** function will immediately display a Bokeh object or application. **Show ()** may be called multiple times in a single Jupyter notebook cell to display multiple objects. The objects are displayed in order.
- ✓ The **line ()** method generates a single line glyph from one dimensional sequence of x and y points.
- ✓ To scatter circle, triangle and square markers on the plot **circle (), triangle ()** and **square ()** methods are used respectively.