EE 618
Microelectronics Lab

TA: Kailash Prasad
PhD in Electrical Engineering

Instructor: Prof. Joycee Mekie

nanoDC Lab

Outline

- **Introductions**
- Verilog Syntax
- Structural Modelling
- Behavioural Modelling
- State Machine
- Random but Important Stuff

Introduction

The Verilog Language

- HDL is a language that describes the hardware of digital systems in a textual form.
- It resembles a programming language, but is specifically oriented to describing hardware structures and behaviors.
- The main difference with the traditional programming languages is HDL’s representation of extensive parallel operations whereas traditional ones represents mostly serial operations.
- The most common use of a HDL is to provide an alternative to schematics.
- Most commonly-used languages in digital hardware design (other - VHDL)

How Verilog Is Used

- Virtually every chip (FPGA, ASIC, etc.) is designed in part using one of these two languages
- Behavioral modeling with some structural elements
- “Synthesis subset”
    - Can be translated using Synopsys’ Design Compiler or others into a netlist
- Design written in Verilog
- Simulated to death to check functionality
- Synthesized (netlist generated)
- Static timing analysis to check timing

```
module top_four_bit_comparator(A,B,A_greater,B_greater,Both_equal);
          input [3:0] A,B;
          output A_greater,B_greater,Both_equal;

         wire a,b,c,d,e,f,g,h,i;
        //A_greater cases
         assign a = A[3]&(~B[3]);
         assign b = (~(A[3]^B[3]))&(A[2]&(~B[2]));
         assign c = (~(A[3]^B[3]))&(~(A[2]^B[2]))&(A[1]&(~B[1]));
         assign d = (~(A[3]^B[3]))&(~(A[2]^B[2]))&(~(A[1]^B[1]))&(A[0]&(~B[0]));

        //B_greater Cases

         assign e = B[3]&(~A[3]);
         assign f = (~(B[3]^A[3]))&(B[2]&(~A[2]));
         assign g = (~(B[3]^A[3]))&(~(B[2]^A[2]))&(B[1]&(~A[1]));
         assign h = (~(B[3]^A[3]))&(~(B[2]^A[2]))&(~(B[1]^A[1]))&(B[0]&(~A[0]));

         //equal case
         assign i = (~(B[3]^A[3]))&(~(B[2]^A[2]))&(~(B[1]^A[1]))&(~(B[0]^A[0]));

         assign A_greater = a|b|c|d;
         assign B_greater = e|f|g|h;
         assign Both_equal= i;


endmodule
```

```
module onebitcomp(
    input a1,
    input b1,
    output gt1,
    output eq1,
    output lt1
    );
    wire a1bar,b1bar;
    not g1(b1bar,b1);
    not g2(a1bar,a1);
    assign gt1=a1&b1bar;
    assign eq1=~(a1^b1);
    assign lt1= a1bar&b1;
endmodule
```

```
module twobitcomp(
    input [1:0] A,
    input [1:0] B,
    output [2:0] f
    );

    wire gt0,eq0,lt0,gt1,eq1,lt1;
    onebitcomp roshni(A[1],B[1],gt1,eq1,lt1);
    onebitcomp req(A[0],B[0],gt0,eq0,lt0);
    assign f=gt1?3'b100:{gt0,eq0,lt0};

endmodule
```

```
module top_FOURBITCOMPA(
    input [3:0] A,
    input [3:0] B,
    output [2:0] F
    );
   wire [2:0]f1;
    wire [2:0]f2;
    reg [2:0]F;
    twobitcomp ros(A[3:2],B[3:2],f1);
    twobitcomp roshn(A[1:0],B[1:0],f2);
    always@(f1 or f2)
    begin
    if(f1==3'b100)
    F=3'b100;
    else
    F=f2;
    end
    endmodule

    //**F is concatanation of greater ,equal,less than
quantities in the same order **//
```

```
module top_decoder1(i0,i1,i2,en,o);
    input i0,i1,i2;
    input en;
    output [0:7]o;
    reg [0:7]o;
    always@(i0,i1,i2,en)
    begin
        if((en==1)&&(i0==0)&&(i1==0)&&(i2==0))
        o=8'b10000000;
        else if ((en==1)&&(i0==1)&&(i1==0)&&(i2==0))
        o=8'b01000000;
        else if ((en==1)&&(i0==0)&&(i1==1)&&(i2==0))
        o=8'b00100000;
        else if ((en==1)&&(i0==1)&&(i1==1)&&(i2==0))
        o=8'b00010000;
        else if ((en==1)&&(i0==0)&&(i1==0)&&(i2==1))
        o=8'b00001000;
        else if ((en==1)&&(i0==1)&&(i1==0)&&(i2==1))
        o=8'b00000100;
        else if ((en==1)&&(i0==0)&&(i1==1)&&(i2==1))
        o=8'b00000010;
        else if ((en==1)&&(i0==1)&&(i1==1)&&(i2==1))
        o=8'b00000001;
        else
        o=8'b00000000;
    end
endmodule
```

```
module top_decoder(i_bit,Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7);
        input [2:0] i_bit;
        output Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;

        assign Y0 = (~i_bit[2])&(~i_bit[1])&(~i_bit[0]);

        assign Y1 = (~i_bit[2])&(~i_bit[1])&(i_bit[0]);

        assign Y2 = (~i_bit[2])&(i_bit[1])&(~i_bit[0]);

        assign Y3 = (~i_bit[2])&(i_bit[1])&(i_bit[0]);

        assign Y4 = (i_bit[2])&(~i_bit[1])&(~i_bit[0]);

        assign Y5 = (i_bit[2])&(~i_bit[1])&(i_bit[0]);

        assign Y6 = (i_bit[2])&(i_bit[1])&(~i_bit[0]);

        assign Y7 = (i_bit[2])&(i_bit[1])&(i_bit[0]);

endmodule
```

Outline

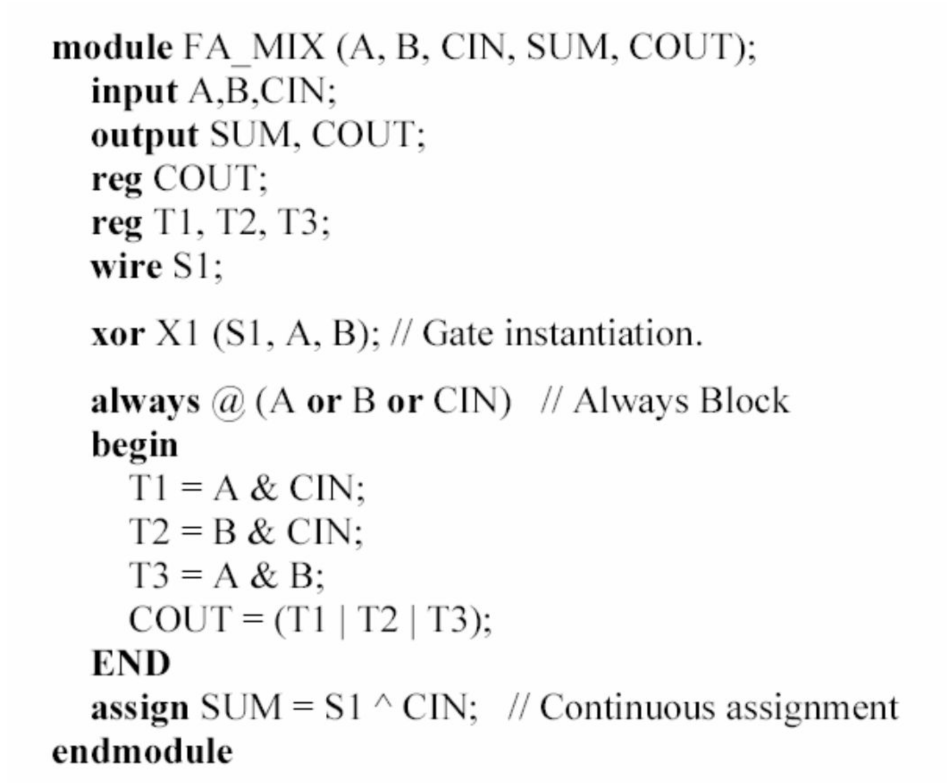
Verilog Syntax

Module

module module_name(port_list);
Declarations:
  Port declaration (input, output, inout,...)
  Data type declaration (reg, wire , parameter,...)
  Task and function declaration
Statements:

Initial block	**Behavioral**
Always block	
Module instantiation	**Structural**
Gate Instantiation	
UDP Instantiation	**Data-Flow**
Continuous assignment	

endmodule



```
module FA_MIX (A, B, CIN, SUM, COUT);
  input A,B,CIN;
  output SUM, COUT;
  reg COUT;
  reg T1, T2, T3;
  wire S1;

  xor X1 (S1, A, B); // Gate instantiation.

  always @ (A or B or CIN)   // Always Block
  begin
    T1 = A & CIN;
    T2 = B & CIN;
    T3 = A & B;
    COUT = (T1 | T2 | T3);
  END
  assign SUM = S1 ^ CIN;   // Continuous assignment
endmodule
```

Source: http://www.cas.mcmaster.ca/~lawford/3TB4/ref/VerilogPresent-v1.3.pdf

Lexical Conventions

- Case Sensitivity
    - Verilog HDL is case sensitive
- White Spaces
    - Blank Spaces, Tabs, New Line etc
- Comments
    - Single Line   //
    - Multiline   /* Text   */
- Identifiers
    - Must begin with (**a-z, A-Z, _**)
    - May Contain (**a-z, A-Z, _, 0-9,$)**
    - Upto **1024** character long

Lexical Conventions

- Numbers
  - Integers
    - <Size> ‘ <Radix> <Value>
    - 3   00000000000000000000000000000011
    - 8’h3C   00111101
    - 5’b1_0111   10111
    - ‘hF   00000000000000000000000000001111
  - Real Numbers
    - 1.6, 0.9, 7.4E6
  - Signed and Unsigned Numbers

Lexical Conventions

- Ports
  - Input
  - Output
  - Inout
- Data Types
  - Nets - represent structural connections between components.
  - Registers - represent variables used to store data.

Wire

Net Data Type	Functionality
wire, tri	Interconnecting wire - no special resolution function
wor, trior	Wired outputs OR together (models ECL)
wand, triand	Wired outputs AND together (models open-collector)
tri0, tri1	Net pulls-down or pulls-up when not driven
supply0, supply1	Net has a constant logic 0 or logic 1 (supply strength)
trireg	Retains last value, when driven by z (tristate).

Reg

Data Type	Functionality
reg	Unsigned variable
integer	Signed variable - 32 bits
time	Unsigned integer - 64 bits
real	Double precision floating point variable

Logic Values

Logic Value	**Description**
0	Zero, low, false
1	One, high, value
Z	High impedance, floating
X	Unknown, uinitialised

Operators

Arithmetic Operators	+, -, *, /
Relational Operators	>,<,<=,>=
Equality Operators	===, !==, ==, !=
Logical Operators	!, &&, ||
Bit-wise Operators	~, &, |, ^, ^~ or ~^
Reduction Operators	&, ~&, |, ~|, ^, ^~ or ~^
Shift Operators	<<, >>
Concatenation Operators	{a, b[3:0], c, 4’b1001}
Replication Operations	{n{m}}
Conditional Operations	cond_expr? true_expr: false_expr

Outline

- Introductions
- Verilog Syntax
- **Structural Modelling**
- Behavioural Modelling
- State Machine
- Random but Important Stuff

Structural Modelling

Structural Modelling

- When Verilog was first developed (1984) most logic simulators operated on netlists
- Netlist: list of gates and how they’re connected
- A natural representation of a digital logic circuit
- Not the most convenient way to express test benches

Module Instantiation

- Instances of
  - module mymod(y, a, b);
- look like
  - mymod mm1(y1, a1, b1);   // Connect-by-position
  - mymod (y2, a1, b1), (y3, a2, b2);   // Instance names omitted
  - mymod mm2(.a(a2), .b(b2), .y(c2)); // Connect-by-name

Gate Level Primitives

Verilog provides the following:

and	nand	logical AND/NAND
or	nor	logical OR/NOR
xor	xnor	logical XOR/XNOR
buf	not	buffer/inverter
bufif0	notif0	Tristate with low enable
bifif1	notif1	Tristate with high enable

Delays on Primitive Instances

- Instances of primitives may include delays
  - buf b1(a, b); // Zero delay
  - buf #3 b2(c, d); // Delay of 3
  - buf #(4,5) b3(e, f); // Rise=4, fall=5

Outline

- Introductions
- Verilog Syntax
- Structural Modelling
- **Behavioural Modelling**
- State Machine
- Random but Important Stuff

Behavioural Modelling

Behavioral Modeling

- A much easier way to write testbenches
- Also good for more abstract models of circuits
  - Easier to write
  - Simulates faster
- More flexible
- Provides sequencing
- Verilog succeeded in part because it allowed both the model and the testbench to be described together

Procedural Blocks

- initial : initial blocks execute only once at time zero (start execution at time zero).
- always : always blocks loop to execute over and over again; in other words, as the name suggests, it executes always.

Blocking and Nonblocking assignment

- Blocking
    - a=b
    - Sequential
- Non-Blocking
    - a<=b
    - Parallel

The Conditional Statement

- If
- If-else
- If, elseif, … , else

The Case statement

```
case ()
< case1 > : < statement >
< case2 > : < statement >
.....
default : < statement >
endcase
```

```
always @ (a or b or c or d or sel)
 case (sel)
  0 : y = a;
  1 : y = b;
  2 : y = c;
  3 : y = d;
 endcase
```

- casez : Treats z as don't care.
- casex : Treats x and z as don't care.

Looping Statements

- Forever
  - The forever loop executes continually, the loop never ends
- Repeat
  - The repeat loop executes < statement > a fixed < number > of times.
- While
  - while (< expression >) < statement >
- For
  - for (< initial assignment >; < expression >, < step assignment >) < statement >

Clock Generation

-
  ```
  initial begin
          clk = 0;
          forever begin
                  #10 clk = ~clk;
          end
  ```

-
  ```
  Initial clk = 0;
  always #10 clk = ~clk;
  ```

Testbenches

```
module circuit_with_delay (A,B,C,x,y);
input A,B,C;
output x,y;
wire e;
and #(30) g1(e,A,B);
or #(20) g3(x,e,y);
not #(10) g2(y,C);
endmodule
```

- **Module circuit_with_delay is instantiated**
- **reg keyword indicates that values are stored (driven)**
- **Stimulus signals are applied sequentially**
- **$finish indicates simulation should end**

```
//Stimulus for simple circuit
module stimcrct;
reg A,B,C;
wire x,y;
circuit_with_delay cwd(A,B,C,x,y);
initial
begin
A = 1'b0; B = 1'b0; C = 1'b0;
#100
A = 1'b1; B = 1'b1; C = 1'b1;
#100 $finish;
end
endmodule
```

Testbenches

```
module test;
reg a, b, sel;

mux m(y, a, b, sel);

initial begin
  $monitor($time,, “a = %b b=%b sel=%b y=%b”,
                  a, b, sel, y);
  a = 0; b= 0; sel = 0;
  #10 a = 1;
  #10 sel = 1;
  #10 b = 1;
end
```

Outline

- Introductions
- Verilog Syntax
- Structural Modelling
- Behavioural Modelling
- **State Machine**
- Random but Important Stuff

State Machine

Modeling FSMs Behaviorally

- There are many ways to do it:
- Define the next-state logic combinationally and define the state-holding latches explicitly
- Define the behavior in a single always @(posedge clk) block

FSM with Combinational Logic

```
module FSM(o, a, b, reset);
output o;
reg o;
input a, b, reset;
reg [1:0] state, nextState;

always @(a or b or state)
 case (state)
    2’b00: begin
       nextState = a ? 2’b00 : 2’b01;
       o = a & b;
    end
    2’b01: begin nextState = 2’b10; o = 0; end
 endcase
```

FSM with Combinational Logic

```
always @(a or b or state)
 case (state)
   2’b00: begin
      nextState = a ? 2’b00 : 2’b01;
      o = a & b;
   end
   2’b01: begin nextState = 2’b10; o = 0; end
 endcase

always @(posedge clk or reset)
  if (reset)
    state <= 2’b00;
  else
    state <= nextState;
```

This is a Mealy machine because the output is directly affected by any change on the input

FSM from a Single Always Block

```
module FSM(o, a, b);
output o; reg o;
input a, b;
reg [1:0] state;

always @(posedge clk or reset)
  if (reset) state <= 2’b00;
  else case (state)
    2’b00: begin
       state <= a ? 2’b00 : 2’b01;
       o <= a & b;
    end
    2’b01: begin state <= 2’b10; o <= 0; end
 endcase
```

Expresses Moore machine behavior:
Outputs are latched
Inputs only sampled at clock edges

Nonblocking assignments used throughout to ensure coherency. RHS refers to values calculated in previous clock cycle

Outline

- Introductions
- Verilog Syntax
- Structural Modelling
- Behavioural Modelling
- State Machine
- **Random but Important Stuff**

Random but Important Stuff

Tools

- Icarus
  - Icarus Verilog is a free compiler implementation for the IEEE-1364 Verilog hardware description language.
- VCS
  - The Synopsys VCS® functional verification solution is the primary verification tool.
- Questasim
  - The Questa Advanced Simulator is the core simulation and debug engine of the Questa Verification Solution
- ModelSim
  - ModelSim packs an unprecedented level of verification capabilities in a cost-effective HDL simulation solution

Tools

- ISE
  - Xilinx ISE (Integrated Synthesis Environment)[3] is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.
- Vivado
  - Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis

Source: Wikipedia

Tools

- Design Compiler
  - Synthesize a block-level RTL design to generate a gate-level netlist with acceptable post-placement timing and congestion
- Innovus
  - Embedded Processor: placement, optimization, routing, and clocking
- Spectre
  - Circuit Simulator provides fast, accurate SPICE-level simulation
- Virtuoso
  - Virtuoso Schematic Editor XL
  - Virtuoso Layout Suite XL
- Liberate
  - Cadence® Virtuoso® Liberate™ characterization solution is a cloud-ready, ultra-fast standard cell, I/O, and complex multi-bit-cell library characterization solution.

Verilog Code - Where we can see

- DC netlist
- Cadence

What we should not do?

- Forgetting else statement after if block
    - Generate Latches
- Updating the same variable in multiple always block
    - Critical Warning - Multiconnected net
- For loop depending on N
    - Hardware can’t be dynamic

What we can't do?

- Instantiate structural block inside behavioural code
  - Generate can be used to do that
- Always block with both sensitivity
  - Should be either edge sensitive or level sensitive

Good Habits

- Always Comment your code
- Provide proper indentation
- Break your code in multiple modules or atleast multiple always block
- **Follow Honour code - Compulsory**

FPGA

Outline

- **Introductions**
- Logic Blocks
- Routing
- Demo on Basys3

Introduction

- FPGA - **Field Programmable Gate Array**
- FPGA - 1984
  - Alternative to Programmable Logic Devices
- Field Programmable Gate Array
  - Two dimensional array of logic blocks and flip-flops
  - Electrically Programmable Switches for interconnections between logic blocks.
- ASIC
  - Custom IC : Fabrication Technology

Introduction

- ASIC
  - High Performance
  - Low Power
  - Low Cost in high volumes
- FPGA
  - Low Development Cost
  - Short Time to Market
  - Reprogrammable

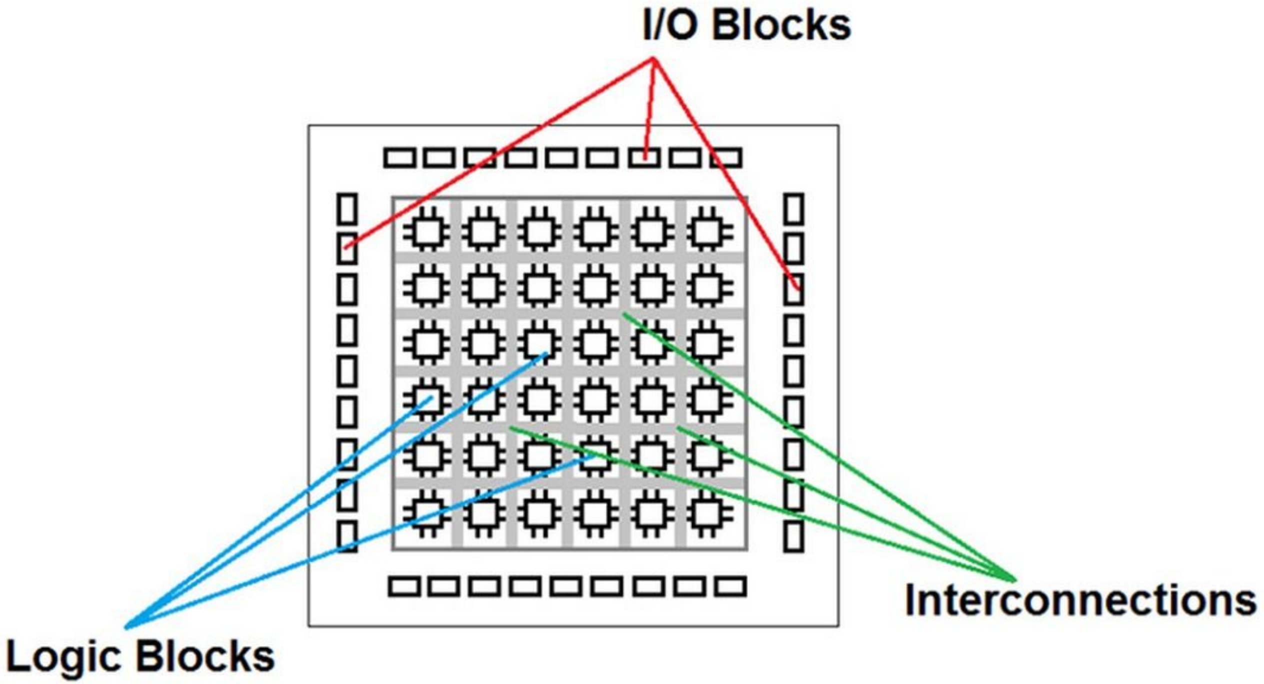
Introduction

- Manufacturing cycle for ASIC is very costly, lengthy and engages lots of manpower
- Mistakes not detected at design time have large impact on development time and cost
- FPGAs are perfect for rapid prototyping of digital circuits
- Easy upgrades like in case of software
- Unique applications reconfigurable computing

Logic Blocks

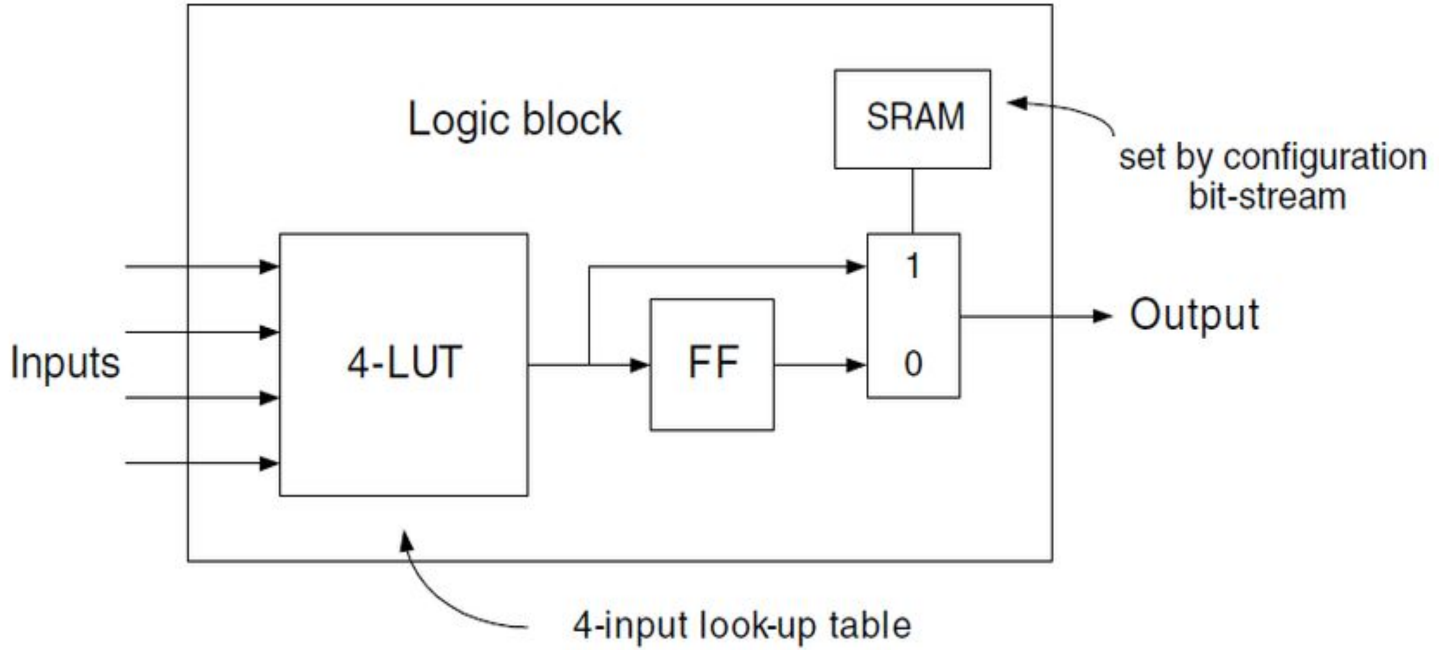
- Logic block
  - A configurable Hardware
  - Transistor to Microprocessors
- Logic blocks of an FPGA can be implemented by any of the following:
  - Transistor pairs
  - combinational gates like basic NAND gates or XOR gates
  - **n-input Lookup tables**
  - Multiplexers

Logic Blocks



Source:http://www.ee.columbia.edu/~kinget/EE6350_S16/04_FPGA_Tom_
Robert_Harrison_Guanshun/architecture.html

Logic Blocks



Source:https://www.researchgate.net/figure/Figure-2-2-Basic-FPGA-config
urable-logic-block-CLB_fig2_265553148

Routing

- Wire segments of varying lengths can be interconnected via electrically programmable switches.
- Density of logic block used in an FPGA depends on length and number of wire segments used for routing.
- Number of segments used for interconnection typically is a tradeoff between density of logic blocks used and amount of area used up for routing.

Demo On Basys3

- https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- https://reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-getting-started/start

Reference

- http://www.asic-world.com/verilog/verilog_one_day2.html#Control_Statements
- https://www.tutorialspoint.com/vlsi_design/vlsi_design_verilog_introduction.htm
- http://www.referencedesigner.com/tutorials/verilog/verilog_01.php
- https://www.nandland.com/verilog/tutorials/tutorial-introduction-to-verilog-for-beginners.html
- http://vol.verilog.com/VOL/main.htm
- https://www.chipverify.com/verilog/verilog-design-abstraction-layers
- https://ece.umd.edu/courses/enee359a/verilog_tutorial.pdf
- web.cecs.pdx.edu/~mperkows/CLASS_VHDL_99/verilog.ppt
- https://numato.com/kb/learning-fpga-verilog-beginners-guide-part-1-introduction/

Thank You