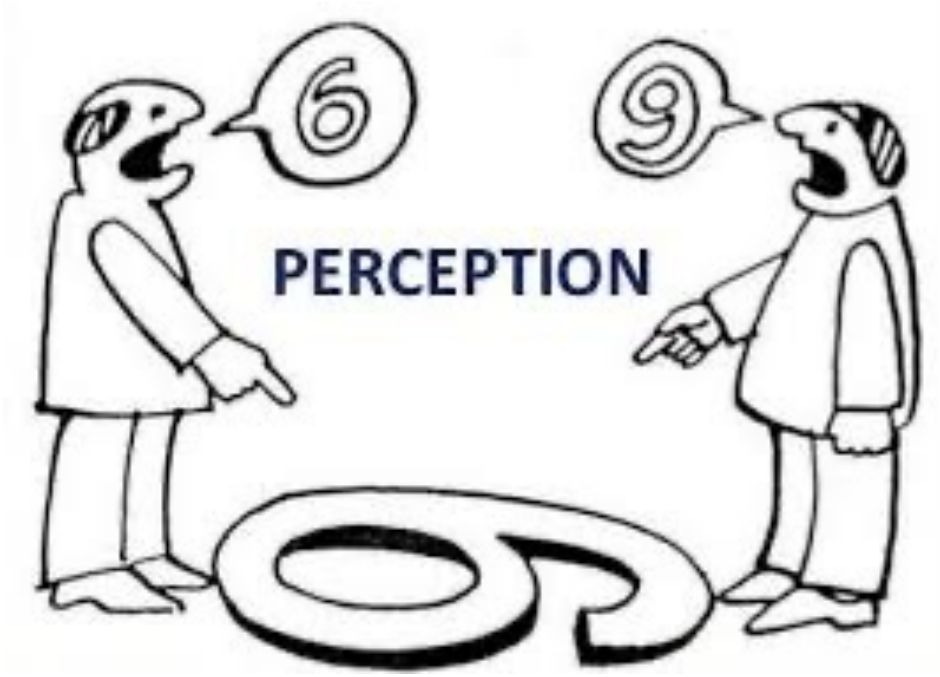
Verilog

Smita Khole


6
9
PERCEPTION


FACTS

- The concept of a hardware description language as a medium for design capture was first introduced in the1950s
- Hardware Description Language (HDL) is a programming language that is used to describe the structure, behaviour and timing of electronic circuits.
- HDLs serve the purpose of simulating the circuit and verifying its response. Many HDLs are available, but the most popular HDLs so far are Verilog and VHDL.

Purpose of HDL

- **Circuit Design:** It provides a way to design digital circuits that meet the required specifications.
- **Simulation:** It helps the designer to test and verify the digital circuit before it is built.
- **Verification:** It provides designers to verify the functionality of the Digital circuit by testing it against different inputs and ensuring that the circuit functionality is correct and meets the desired functionality.
- **Synthesis:** HDLs can be used to synthesize digital circuits. Synthesis is a process of automatically generating circuits from HDL code.
- **Timing Analysis:** It provides designers to analyse the timing behaviour of digital circuits and ensure that the circuits meet the timing requirements.
- **Design Reusability:** HDLs provide a way to design reusable components that can be used for multiple circuits design and reduce time and effort, which improves overall design quality.
- **Optimization:** It Provides a way to optimise the design of digital circuits for performance.

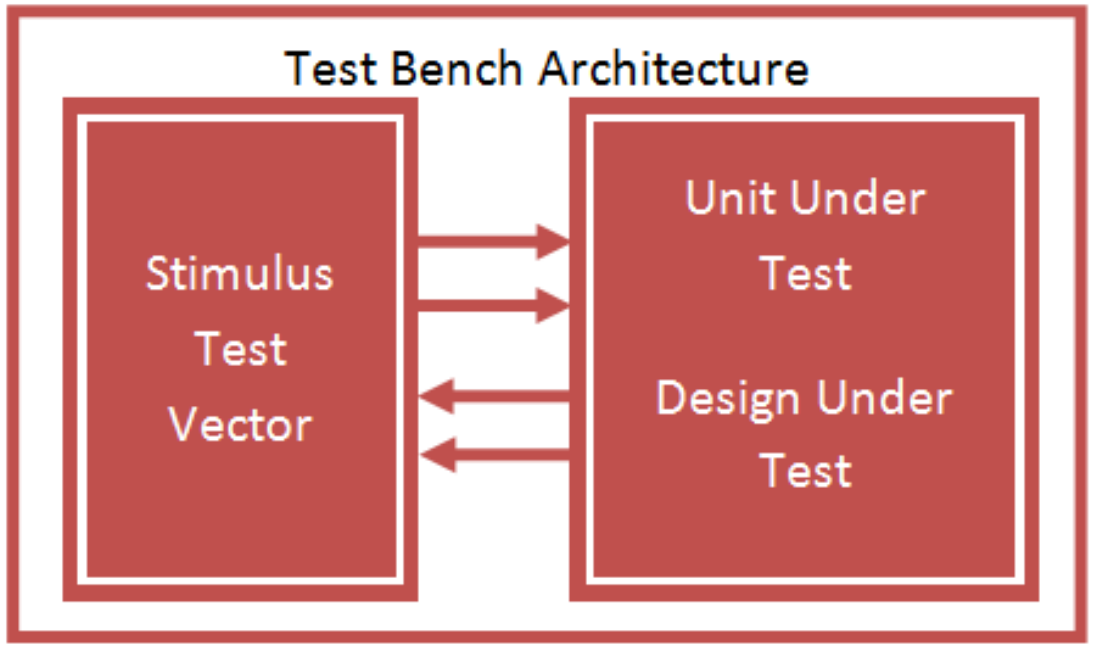
	Hardware based languages(HDLs)	Software based languages(HLS Tools)
processing	Inherently concurrent	Inherently sequential
Word width	Data word width is optimized	Fixed data word width (8,16,32 bits)
Time taken	Time taken for task is depend on timing constraints	Time taken is not considered
Memory management	Memory is divided into a large number of small blocks. Local variables are not implemented in memory.	Software treats memory as single large block, compiler may map local variables to registers
Processes communication	Parallel processes can communicate directly. Communication relies on hardware	It uses shared memory (including mailboxes) to communicate between processes. Processes cannot communicate directly.
Algorithm representation	The algorithm is demonstrated by hardware connection between blocks. Saving current state is difficult. Recursive algorithms are impractical.	An algorithm is demonstrated as a sequence of instructions stored in memory.
Other advantages	Available for high level languages and high capacity FPGA	Support floating point computation

Difference between Verilog and C

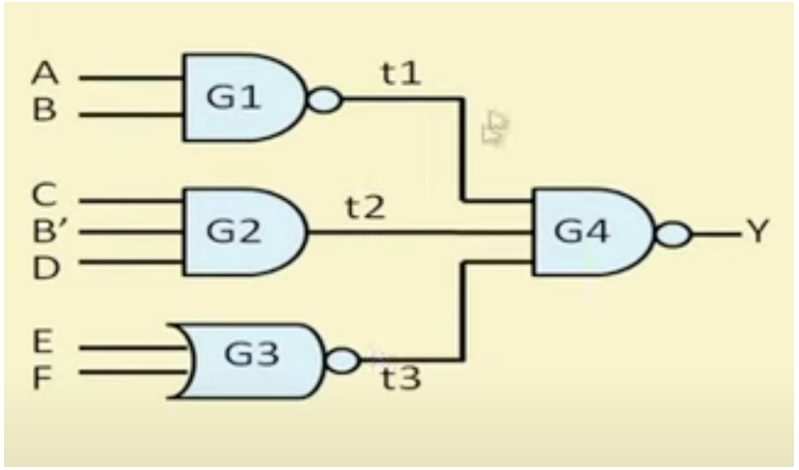
- Used to describe the architecture and behavior of digital hardware like ASICs and FPGAs.
- Allows simulating the hardware design before synthesis/implementation.
- Logic is designed using modules, ports, gates, registers, etc. Parallel execution.
- Used for modeling IC RTL and designing PCBs, processors, SoCs.
- C and C++ are software programming languages used for embedded firmware and applications. Key aspects:
- Targets the software execution on processors and microcontrollers.
- Sequential execution of instructions.
- Rich language features like functions, data structures, OOP support.
- Used for writing device drivers, real-time code, communication stacks, etc.
- Can manipulate I/O pins and registers by mapping to memory.
- No concept of simulating timing or logic circuits.
- So in essence, Verilog/VHDL models hardware while C/C++ executes software instructions on that hardware sequentially. Both are essential skills for embedded systems design.

Test Bench

- To verify functionality of the design coded in Verilog (DUT-Design Under Test)
  - Set stimulus for DUT
  - Monitor or analyse outputs of DUT
- Requirement:
  - Inputs of DUT connected to test Bench
  - Output of DUT connected to Testbench



Verilog Basics



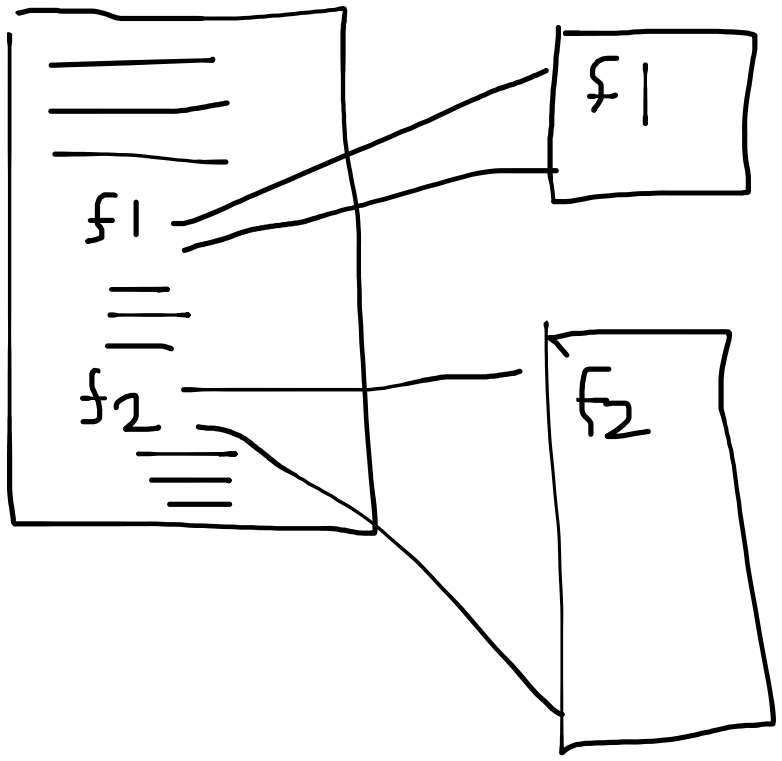

```
module  example (A,B,C,D,E,F,Y);
  input A,B,C,D,E,F;
  output Y;
  wire  t1, t2, t3, Y;
  nand  #1 G1 (t1,A,B);
  and   #2 G2 (t2,C,~B,D);
  nor   #1 G3 (t3,E,F);
  nand  #1 G4 (Y,t1,t2,t3);
endmodule
```

```
  nand  #1 G1 (t1,A,B),
           G4 (Y,t1,t2,t3);
```

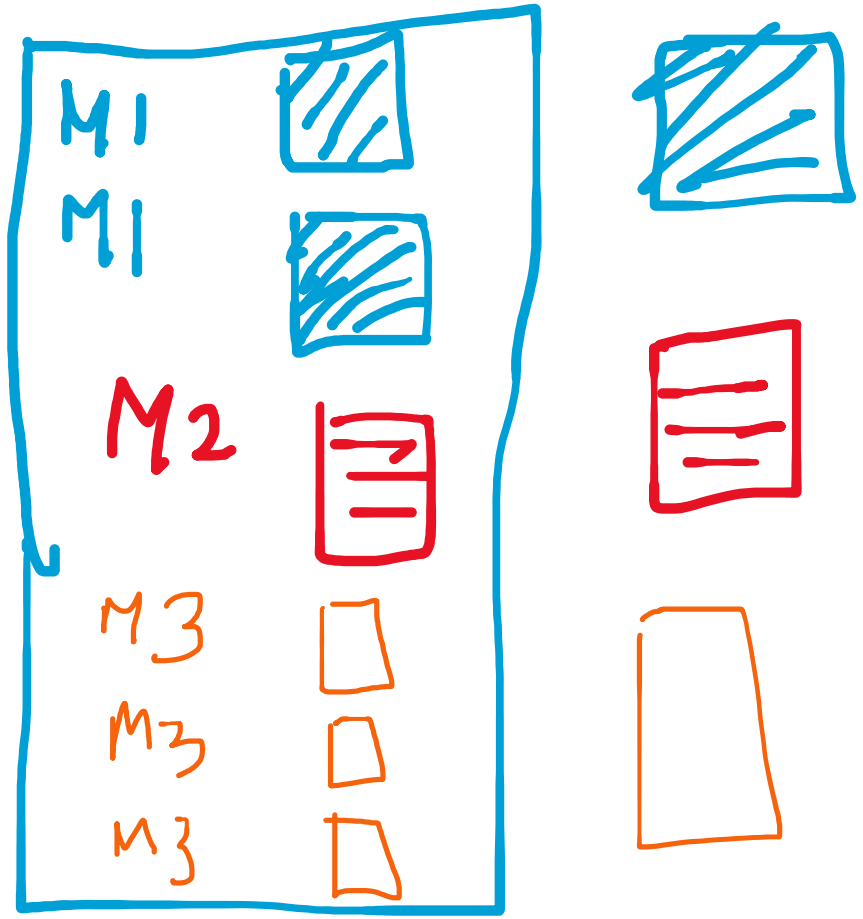
Module

- In verilog basic unit of hardware is called **module**
- Each program will start with keyword “**module**” and end with “**endmodule**”
- A module can’t contain definition of other module
- However, a module can be **instantiated** in another module

```
module module_name (list_of_ports);
    input/output declarations
    local net declarations
    Parallel statements
endmodule
```



C – Function call

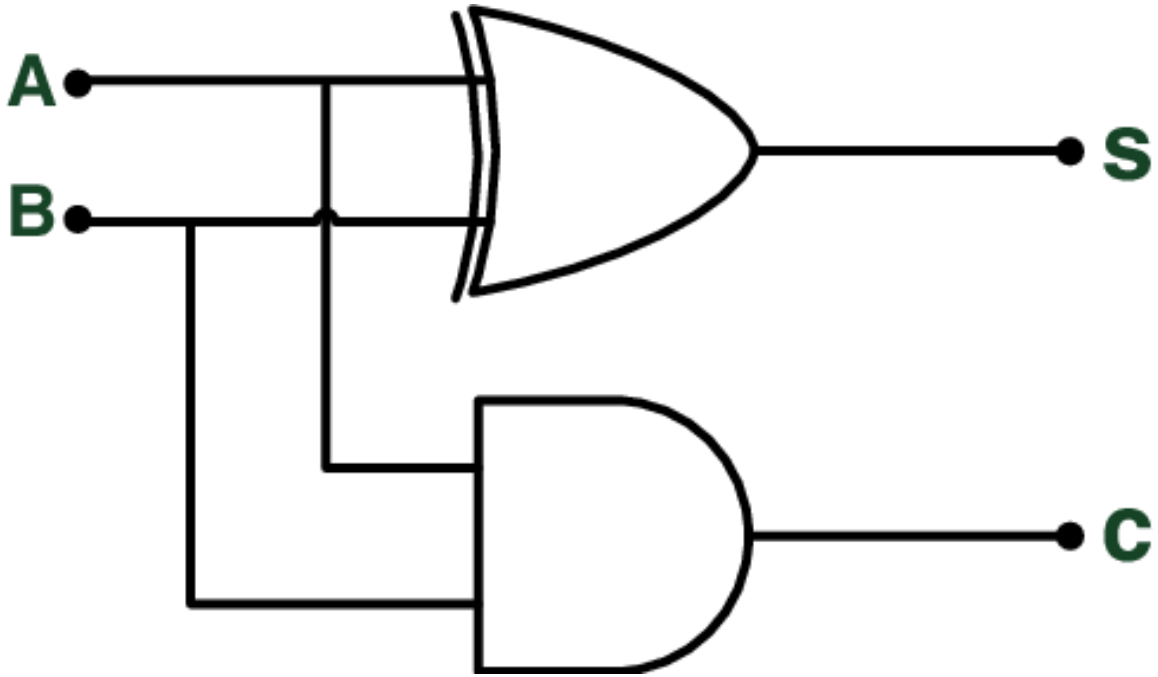


Problem 1. A behavioral model of a full-adder is given by the following Verilog code,

```
module full_adder(sum,cout,a,b,cin);

   output sum,cout;
   input a,b,cin;

   reg sum,cout;                    // lhs in always must be reg

   always @(a,b,cin)
   begin
      sum=a^b^cin;                 // blocking assignments
      cout=a&b|cin&a|b&cin;
   end

endmodule
```

How is this full-adder implemented and what is the worse case delay for this full-adder in a Artix-7 FPGA? Compile it with Xilinx Vivado for an Artix-7 FPGA and find out.

- **module** half_adder(S, C,A,B);
- input A,B ;
- output S,C;

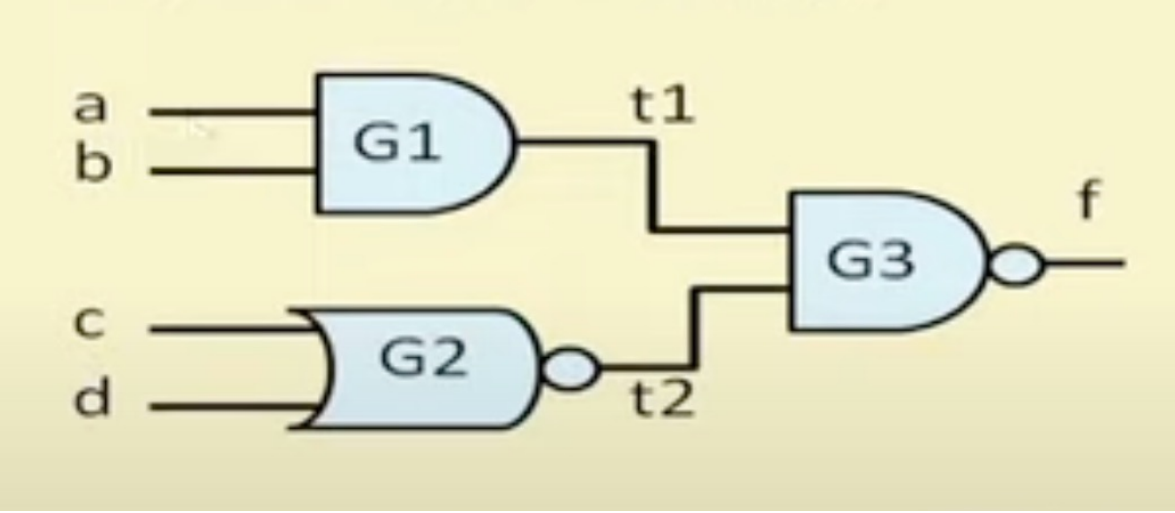


Behavioral description

- Synthesis tool will decide how to realize the function
- Using single AND
- NAND followed by NOT

```
// A simple AND function
module  simpleand (f, x, y);
      input  x, y;
      output  f;
      assign  f = x & y;
endmodule
```

Write behavioral modelling Verilog code for




```
/* A 2-level combinational circuit */
module  two_level (a, b, c, d, f);
   input  a, b, c, d;
   output  f;
   wire  t1, t2;  // Intermediate lines
   assign  t1 = a & b;
   assign  t2 = ~(c | d);
   assign  f = ~(t1 & t2);
endmodule
```

Point to note:

- The "**assign**" statement represents continuous assignment, whereby the variable on the LHS gets updated whenever the expression on the RHS changes.
  assign variable = expression;
- The LHS must be a "net" type variable, typically a "wire".
- The RHS can contain both "register" and "net" type variables.
- A Verilog module can contain any number of "assign" statements; they are typically placed in the beginning after the port declarations.
- The "assign" statement models behavioral design style, and is typically used to model combinational circuits.

Data Types

- A variable in Verilog belongs to one of two data types:

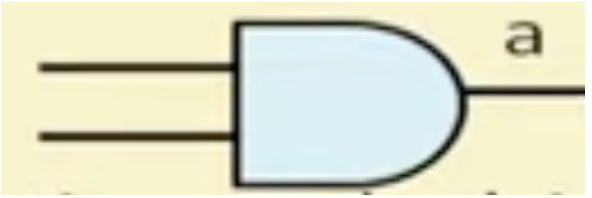
a) Net

- Must be continuously driven.
- Cannot be used to store a value.
- Used to model connections between continuous assignments and instantiations.

b) Register

- Retains the last value assigned to it.
- Often used to represent storage elements, but sometimes it can translate to combinational circuits also.

Nets

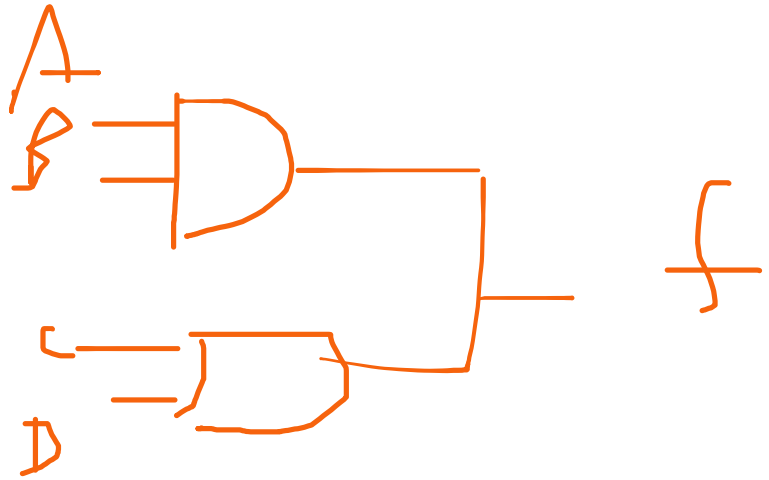


- Nets represents connection between hardware elements.
- Nets are continuously driven by the outputs of the devices they are connected to.
- Net "a" is continuously driven by the output of the AND gate.
- Nets are 1-bit values by default unless they are declared explicitly as vectors.
- **Default value of a net is ”z”**
- Various "Net" data types are supported for synthesis in Verilog: wire, wor, wand, tri, supply, supply1, etc.
- "wire" and "tri" are equivalent; when there are multiple drivers driving them, the driver outputs are shorted together.
- "wor" and "wand" inserts an OR and AND gate respectively at the connection
- "supply0" and "supply1" model power supply connections.

```
module use_wire (A, B, C, D, f);
   input  A, B, C, D;
   output f;
   wire   f;
    // net f declared as 'wire'

   assign f = A & B;
   assign f = C | D;
endmodule
```

```
module use_wand (A, B, C, D, f);
   input  A, B, C, D;
   output f;
   wand   f;
    // net f declared as 'wand'

   assign f = A & B;
   assign f = C | D;
endmodule
```

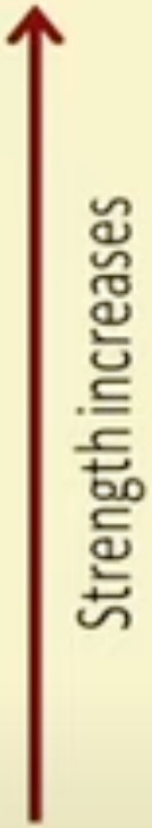
For A = B = 1, and C = D = 0,
will be indeterminate.



```
module using_supply_wire (A, B, C, f);
    input   A, B, C;
    output  f;
    supply0 gnd;
    supply1 vdd;
    nand  G1 (t1, vdd, A, B);
    xor   G2 (t2, C, gnd);
    and   G3 (f, t1, t2);
endmodule
```

Value Level	Represents
0	Logic 0 state
1	Logic 1 state
x	Unknown logic state
z	High impedance state

Strength	Type
supply	Driving
strong	Driving
pull	Driving
large	Storage
weak	Driving
medium	Storage
small	Storage
highz	High impedance



If two signals of unequal strengths get driven on a wire,
the stronger signal will prevail.
These are particularly useful for MOS level circuits,
e.g. dynamic MOS.

- All unconnected nets are set to “z”
- All register variables are set to “x”