Training Course
on

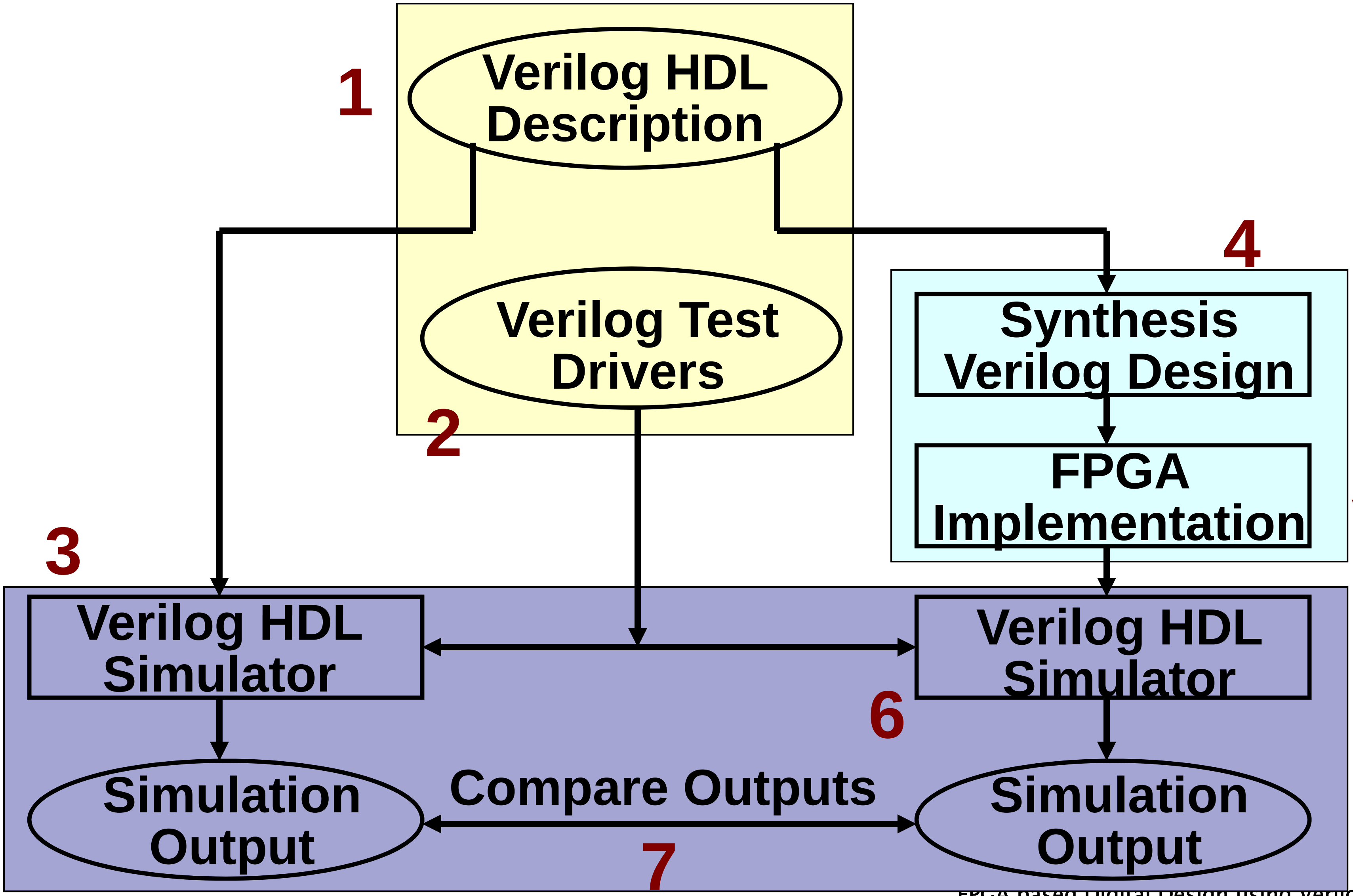
FPGA based Digital Design using Verilog HDL

By

NAUMAN MIR
(HDL Designer)

**** Organized by*** **Skill Development Council,**
(*Ministry of Labour, Manpower and overseas Pakistani)*
Govt. of Pakistan.

Design Methodology



Digital Design Objectives

Digital Design process have following objectives:

- Area on the chip required by the design
- The critical path delay of the design
- The testability of the design
- Power dissipation of the circuit

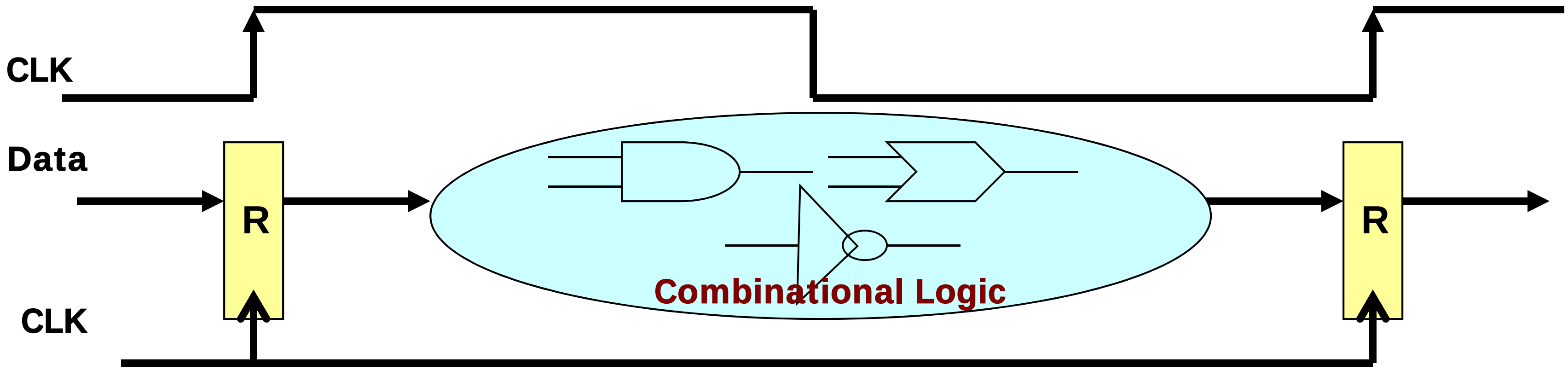
Combinational & Sequential Circuits

- ***Combinational Circuits:*** **Arrangement of logic gates with a set of inputs and outputs**
- ***Sequential Circuits:*** **Interconnection of Flip-Flops and Gates**

Synchronous Digital Circuits

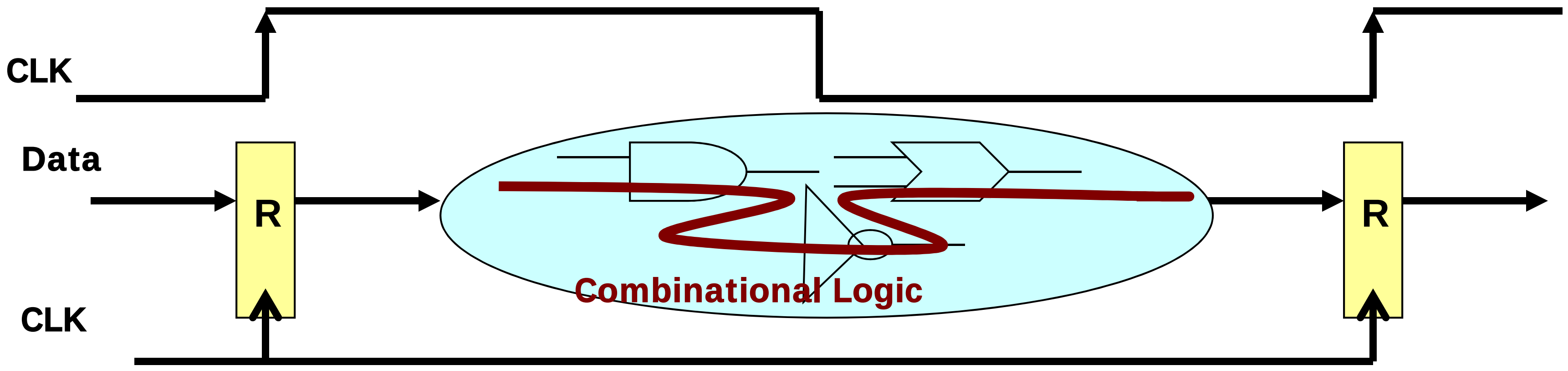
- ***Synchronous:*** **Its basically a Clocked… all changes are controlled by system clock**
- ***Asynchronous:*** **Its independent of clock… these are level sensitive**
- ***Digital & Analog:*** **All signals have discrete values in digital system… Analog has continuous values**

Clocking Methodology



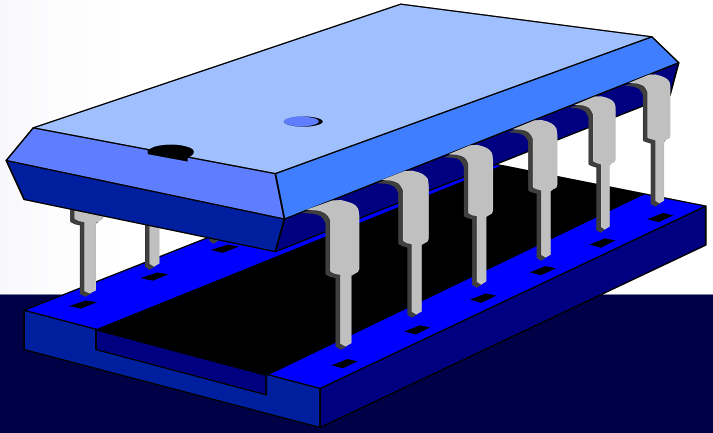
- All storage elements are clocked by the same clock edge
- The combination logic block's:
  - Inputs are updated at each clock tick
  - All outputs MUST be stable before the next clock tick

Critical Time & Cycle Time

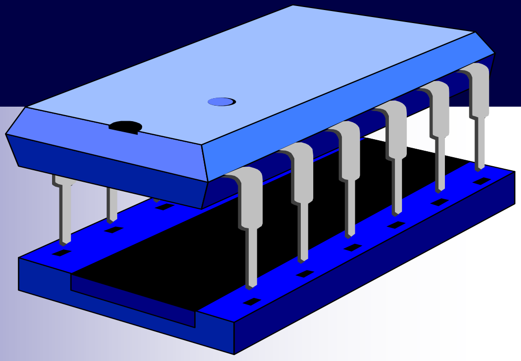


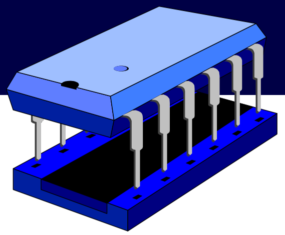
- ➢ **Critical Path: The Slowest Path b/w two storage devices**

- ➢ **Cycle Time is a function of the critical path and it must be greater than:**
  → Longest path through the combination logic + setup + …



Verilog HDL Basics



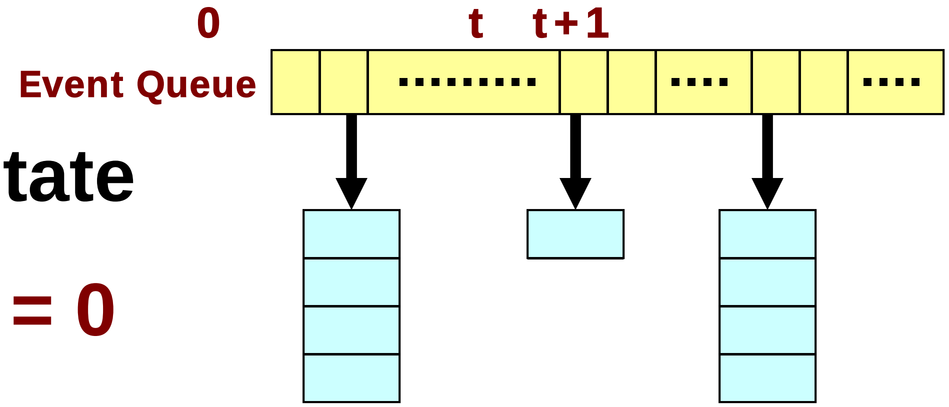


What is Verilog

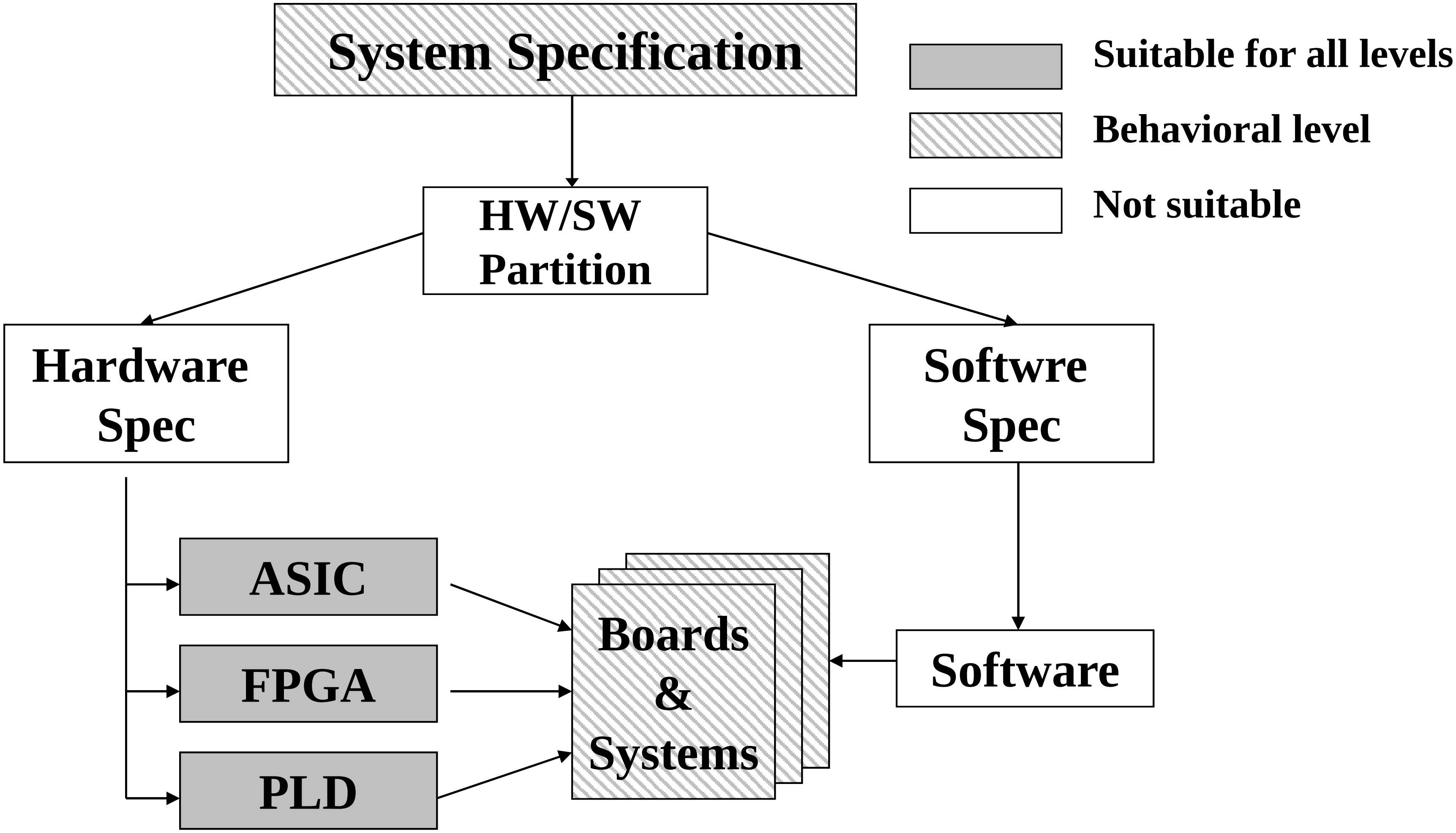
- Hardware Description Language (HDL)
- Verilog was introduced in 1984 by Gateway Design System Corporation, now a part of Cadence Design Systems
- Verilog-based synthesis tool introduced by Synopsys in 1987
- Standard: IEEE 1364 in 1995

Event Driven Simulation

- Verilog is really a language for modeling event driven systems
  - Event : change in state
- Simulation starts at t = 0
  - Processing events generates new events
  - When all events at time t have been processed simulation time advances to t+1
  - Simulation stops when there are no more events in the queue



Application Areas of Verilog



System Specification

HW/SW Partition

Hardware Spec

Softwre Spec

ASIC

FPGA

PLD

Boards & Systems

Software

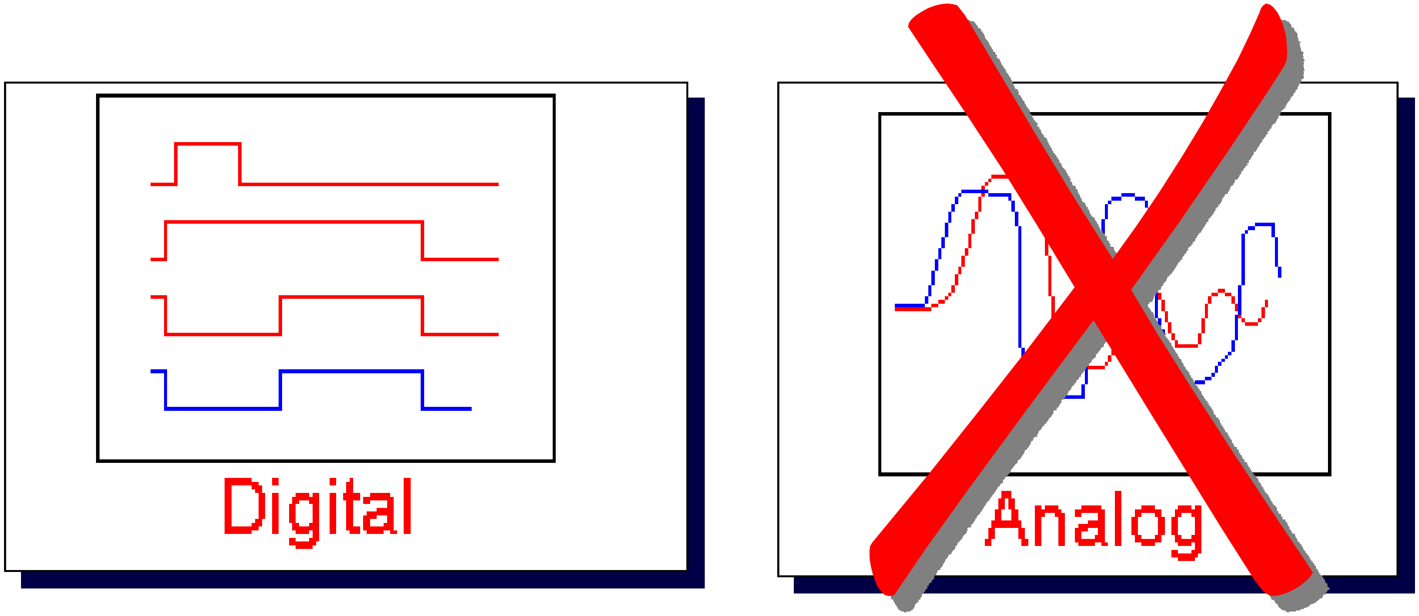
■ Suitable for all levels

▨ Behavioral level

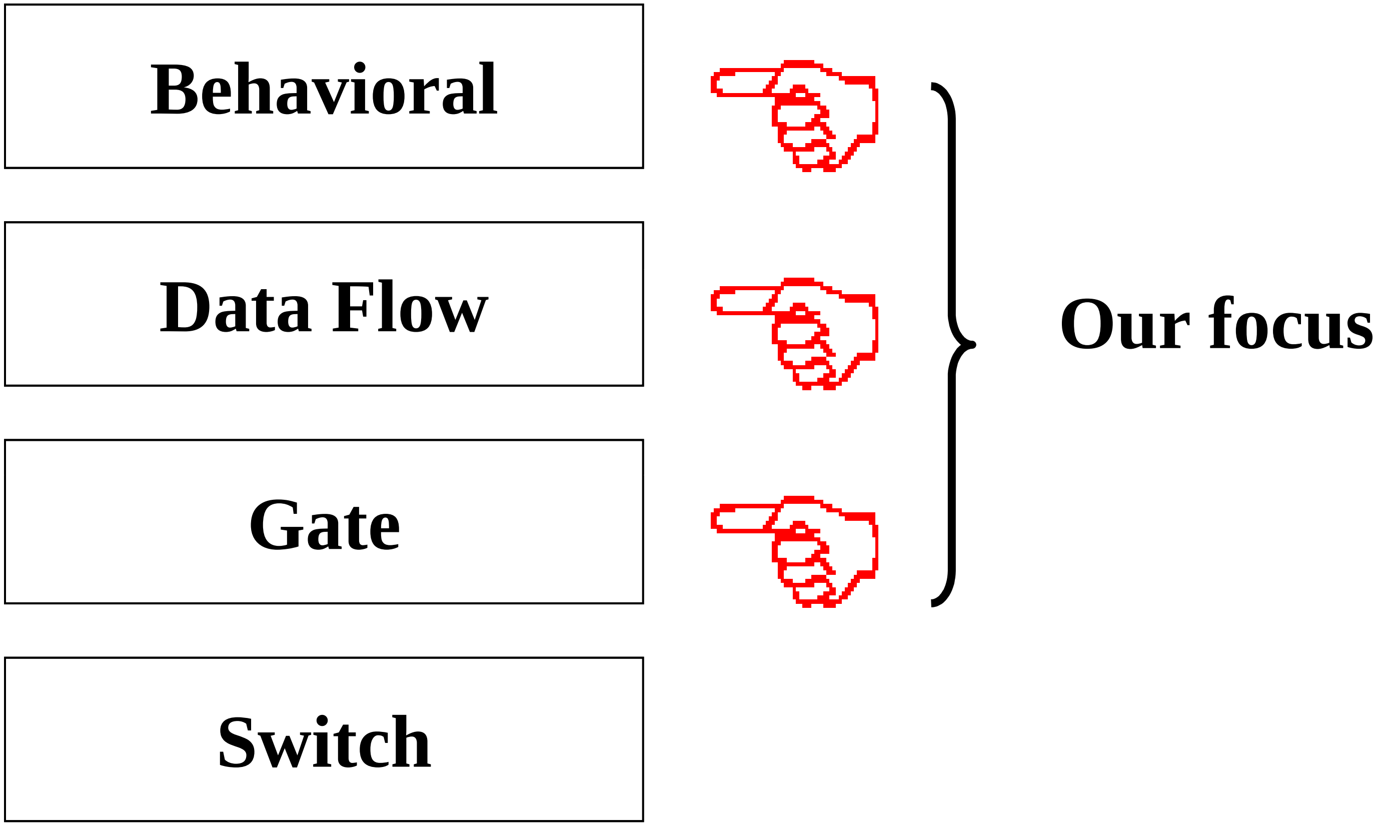
□ Not suitable

Basic Limitation of Verilog

- **Description of digital systems only**



Abstraction Levels in Verilog



Behavioral

Data Flow

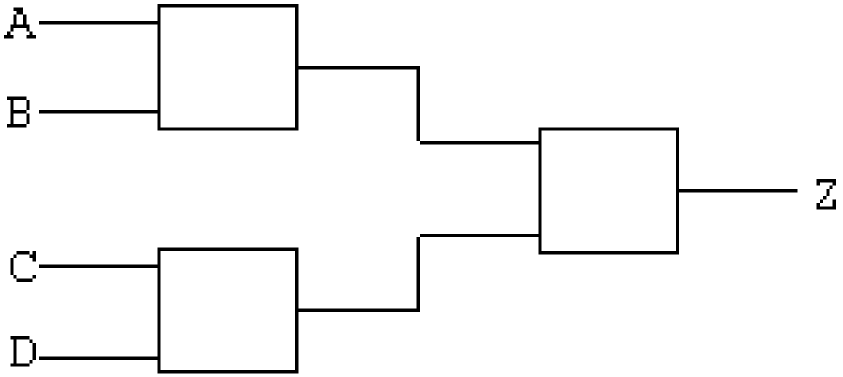
Gate

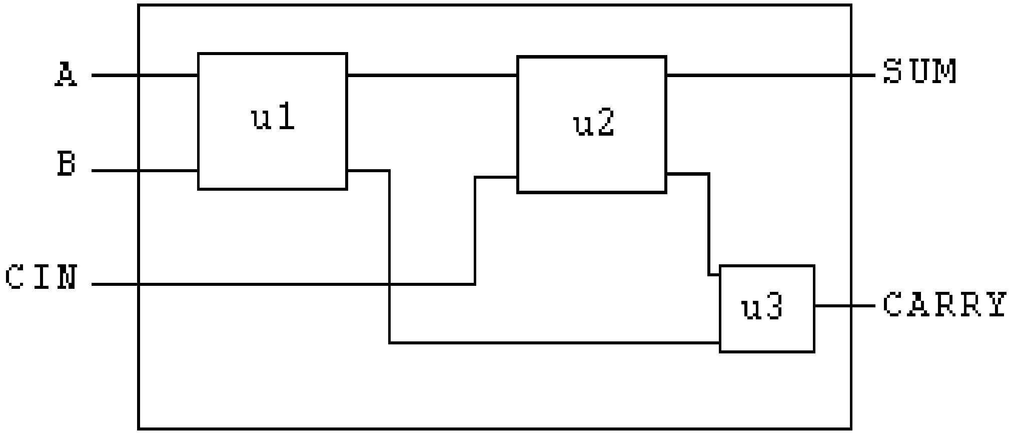
Switch

Our focus

Main Language Concepts (i)

- Concurrency

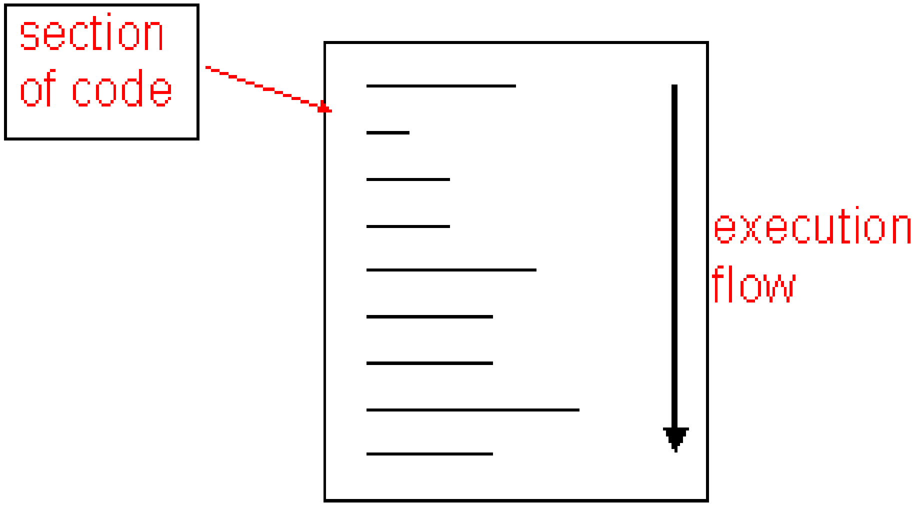


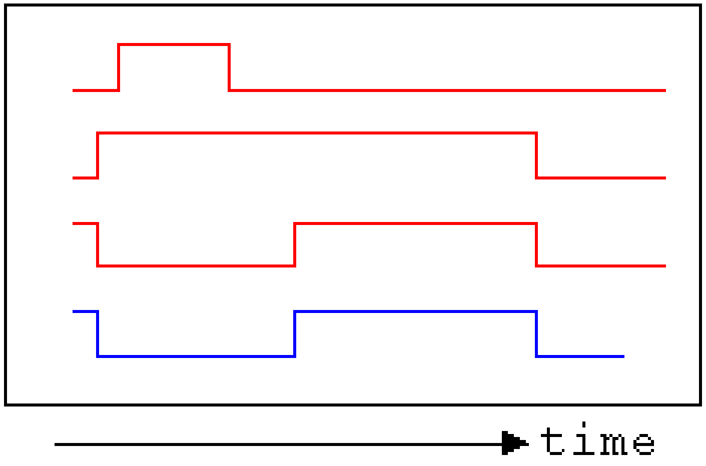


- Structure

Main Language Concepts (ii)

- **Procedural Statements**





- **Time**

User Identifiers (i)

- Formed from {[A-Z], [a-z], [0-9], _, $}, but ..
- .. can’t begin with $ or [0-9]
  - □ myidentifier
  - □ m_y_identifier
  - □ 3my_identifier (X)
  - □ $my_identifier (X)
  - □ _myidentifier$
- Case sensitivity
  - □ myid ≠ Myid

Comments

- // The rest of the line is a comment

- /* Multiple line
  comment*/

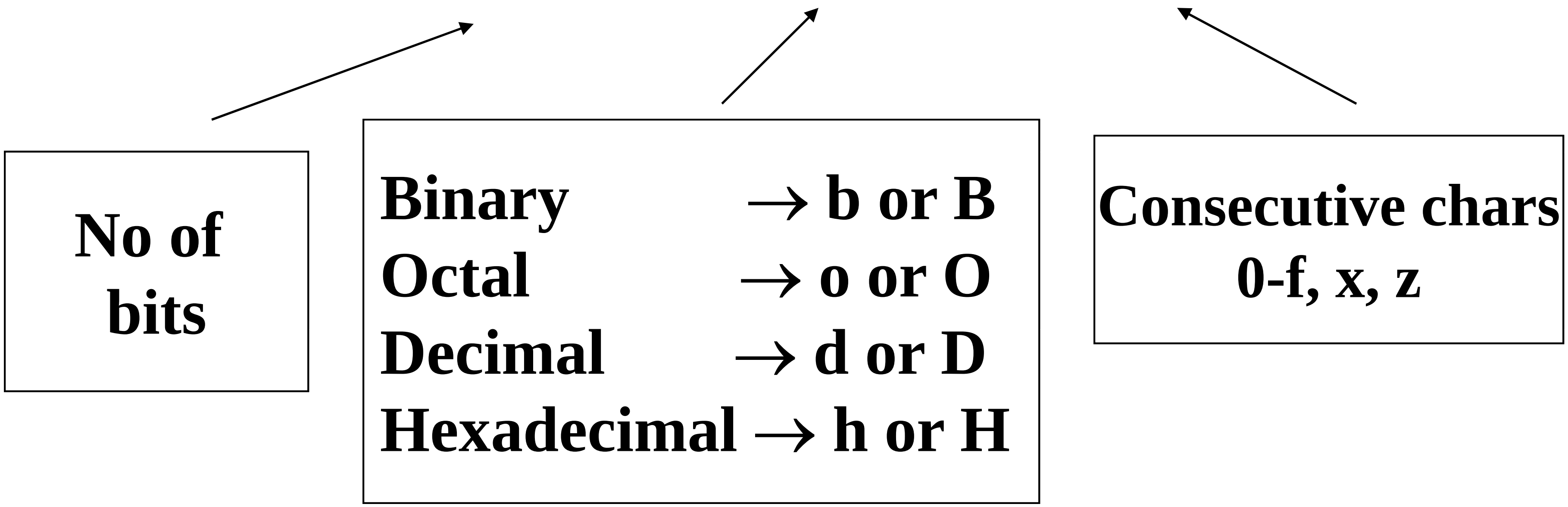
- /* Nesting /* comments */ do NOT work */

Verilog Value Set

- ***0*** represents low logic level or false condition
- ***1*** represents high logic level or true condition
- ***x*** represents unknown logic level
- ***z*** represents high impedance logic level

Numbers in Verilog (i)

<size>'<radix> <value>



No of bits

Binary → b or B
Octal → o or O
Decimal → d or D
Hexadecimal → h or H

Consecutive chars 0-f, x, z

- □ 8'h ax = 1010xxxx
- □ 12'o 3zx7 = 011zzzxxx111

Numbers in Verilog (ii)

- You can insert "_" for readability
  - 12'b 000_111_010_100
  - 12'b 000111010100
  - 12'o 07_24

Represent the same number

- Bit extension
  - MS bit = 0, x or z ⇒ extend this
    - 4'b x1 = 4'b xx_x1
  - MS bit = 1 ⇒ zero extension
    - 4'b 1x = 4'b 00_1x

Nets (i)

- Nets represent connections between hardware elements.

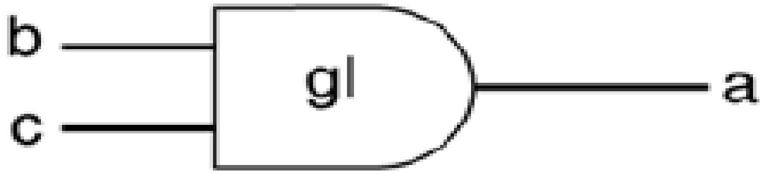
  → Just as in real circuits, nets have values continuously driven on them by the outputs of devices that they are connected to.

- Can be thought as hardware wires driven by logic
- Equal *z* when unconnected

Nets (ii)

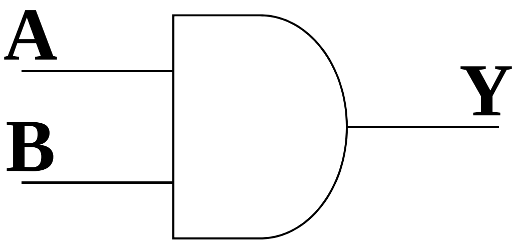
- Various types of nets
  - wire
  - wand (wired-AND)
  - wor (wired-OR)
  - tri (tri-state)

In Figure net *a* is connected to the output of *and* gate g1. Net a will continuously assume the value computed at the output of gate g1.

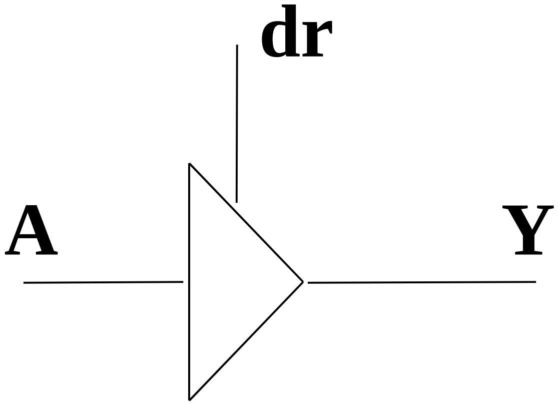


Nets (iii)

- In following examples: Y is evaluated, *automatically*, every time A or B changes

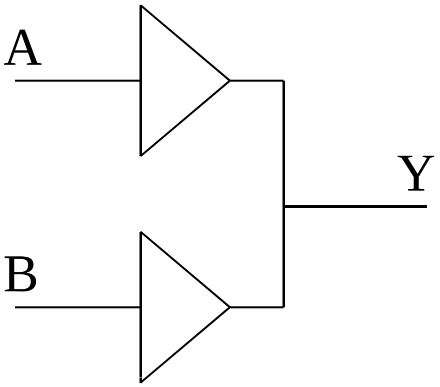


```
wire Y;  // declaration
assign Y = A & B;
```



```
tri Y;  // declaration
assign Y = (dr) ? A : z;
```

Nets (iv)



```
wand Y;   // declaration
assign Y = A;
assign Y = B;
```

Y		A	
		0	1
B	0	0	0
	1	0	1

```
wor Y;   // declaration
assign Y = A;
assign Y = B;
```

Y		A	
		0	1
B	0	0	1
	1	1	1

Registers (i)

- **Registers represent data storage elements.**

  Registers retain value until another value is placed onto them.

  → **Do not confuse the term registers in Verilog with hardware registers built from edge-triggered flipflops in real circuits.**

- **Variables that store values**
- ***Do not represent real hardware but ..***

  .. real hardware can be implemented with registers

Registers (ii)

- **Only one type: reg**

```
reg A, C ;     // declaration
// assignments are always done inside a procedure
     A = 1 ;
     C = A ;    // C gets the logical value 1
     A = 0 ;    // C is still 1
     C = 0 ;    // C is now 0
```

- **Register values are updated explicitly !!**

Vectors (i)

- Nets or reg data types can be declared as vectors (multiple bit widths).
  → If bit width is not specified, the default is scalar (1-bit).
- Represent buses

```
wire [3:0] busA;
reg [1:4] busB;
reg [1:0] busC;
```

- Left number is MS bit

Vectors (ii)

- **Slice management**

$$\texttt{busC = busA[2:1];} \Leftrightarrow \begin{cases} \texttt{busC[1] = busA[2];} \\ \texttt{busC[0] = busA[1];} \end{cases}$$

- **Vector assignment (*by position!!*)**

$$\texttt{busB = busA;} \quad \Leftrightarrow \begin{cases} \texttt{busB[1] = busA[3];} \\ \texttt{busB[2] = busA[2];} \\ \texttt{busB[3] = busA[1];} \\ \texttt{busB[4] = busA[0];} \end{cases}$$

Arrays (i)

- Arrays are allowed in Verilog for reg, integer, time, real and vector register data types.

  → Multi-dimensional arrays can also be declared with any number of dimensions.

- Arrays of nets can also be used to connect ports of generated instances. Each element of the array can be used in the same fashion as a scalar or vector net. Arrays are accessed by

  <array_name>[<subscript>].

Arrays (ii)

For multi-dimensional arrays, indexes need to be provided for each dimension.

```
integer count[0:7]; // An array of 8 count
                       variables
time chk_point[1:100]; // Array of 100 time
                          checkpoint variables
reg [4:0] port_id[0:7];// Array of 8 port_ids
                    each port_id is 5 bits wide
integer matrix[4:0][0:255];
              // Two dimensional array of integers
wire [7:0] w_array2[5:0];
           // Declare an array of 8 bit vector wire
```

Arrays (iii)

- It is important…not to confuse arrays with net or register vectors. A vector is a single element that is n-bits wide. On the other hand, arrays are multiple elements that are 1-bit or n-bits wide.

Examples:

```
count[5] = 0; // Reset 5th element of array of count var.
chk_point[100] = 0;// Reset 100th time chk_point value
port_id[3] = 0; // Reset 3rd element (a 5-bit value)
                  of port_id array.
matrix[1][0] = 33559; // Set value of
                        element indexed by [1][0] to 33559
```

Memories (i)

- In digital simulation, one often needs to model register files, RAMs, and ROMs.
- Memories are modeled in Verilog simply as a one-dimensional array of registers.
- Each element of the array is known as an element or word and is addressed by a single array index. Each word can be one or more bits.

Memories (ii)

- It is important to differentiate between n 1-bit registers and one n-bit register. A particular word in memory is obtained by using the address as a memory array subscript.

```
reg mem1bit[0:1023]; // Memory mem1bit with
                       1K 1-bit words
reg [7:0] membyte[0:1023];
     // Memory membyte with 1K 8-bit words(bytes)
membyte[511] // Fetches 1 byte word whose
               address is 511.
```

Parameters (i)

- Verilog allows constants to be defined in a module by the keyword parameter. Parameters cannot be used as variables.
- Parameter sizes can also be defined.

```
parameter port_id = 5; // Defines a constant
                          port_id
parameter cache_line_width = 256;
          // Constant defines width of cache line
parameter [15:0] WIDTH; // Fixed range for
                           parameter WIDTH
```

Parameters (ii)

- Module definitions may be written in terms of parameters. Hard-coded numbers should be avoided.
- Parameters values can be changed at module instantiation or by using the defparam statement, which is discussed later…

Strings

- Strings can be stored in reg. The width of the register variables must be large enough to hold the string. Each character in the string takes up 8 bits (1 byte).

```
reg [8*18:1] string_value;
  // Declare a variable that is 18 bytes wide

initial
   string_value = "Hello Verilog World";
                     // String can be stored in variable
```

System Tasks (i)

- **Verilog provides standard system tasks for certain routine operations.**
- **All system tasks appear in the form $<keyword>.**
- **Operations such as displaying on the screen, monitoring values of nets, stopping, and finishing are done by system tasks.**

System Tasks (ii)

→ Displaying information

$display is the main system task for displaying values of variables or strings or expressions. This is one of the most useful tasks in Verilog.

Usage: $display(p1, p2, p3,....., pn);

p1, p2, p3,..., pn can be quoted strings or variables or expressions.

→ Monitoring information

Verilog provides a mechanism to monitor a signal when its value changes. This facility is provided by the $monitor task.

Usage: $monitor(p1,p2,p3,....,pn);

The parameters p1, p2, ... , pn can be variables, signal names, or quoted strings.

System Tasks (iii)

→ Stopping and Finishing in a Simulation

The task $stop is provided to stop during a simulation.

Usage: $stop;

The $stop task puts the simulation in an interactive mode. The designer can then debug the design from the interactive mode. The $stop task is used whenever the designer wants to suspend the simulation and examine the values of signals in the design.

The $finish task terminates the simulation.

Usage: $finish;

Compiler Directives (i)

- **Compiler directives are provided in Verilog. All compiler directives are defined by using the `<keyword> construct.**

`define

The `define directive is used to define text macros in Verilog. The Verilog compiler substitutes the text of the macro wherever it encounters a `<macro_name>. This is similar to the *#define construct in C*.

Compiler Directives (ii)

`include

The `include directive allows you to include entire contents of a Verilog source file in another Verilog file during compilation. This works similarly to the *#include in the C*. This directive is typically used to include header files, which typically contain global or commonly used definitions.

NOTE: Two other directives, `ifdef and `timescale, are used frequently. They are discussed later…