

DIGITAL SYSTEM DESIGN USING VERILOG

**COURSE CODE: 19EC5DCDSV
(3 CREDITS)**

1

Faculty In-Charge: Dr. Dinesha P

drdinesh-ece@dayanandasagar.edu

SYLLABUS

- Module 1: Computer-Aided Design , Hardware Description Languages, Module modelling styles, Verilog Description of Combinational Circuits, Data flow modelling-dataflow modelling, operands, operators, Verilog Modules, Verilog Assignments, Procedural Assignments, Modeling Flip-Flops Using Always Block, Always Blocks Using Event Control Statements.[Text Book 1 and 3]
- Module 2: Delays in Verilog, Compilation, Simulation, and Synthesis of Verilog Code , Simple Synthesis Examples, Verilog Models for Multiplexers, Modeling Registers and Counters Using Verilog Always Statements, Constants, Arrays, Loops in Verilog, Testing a Verilog Model
- Additional topics in Verilog: Verilog Functions, Verilog Tasks, Multivalued Logic and Signal Resolution, Built-in Primitives User-Defined Primitives, Named Association, Generate Statements₂, System Functions, File I/O Functions.[Text Book 1]

- **Module 3: Sequential Basics:** Storage elements and Counters, Sequential circuit timing: Propagation Delays, Setup, and Hold Times, Timing Conditions for Proper Operations, Glitches In Sequential Circuits, Synchronous Design. Tristate Logic and Busses.

Design flow of ASIC and FPGA based systems. Design

translation(synthesis) Coding guidelines[Text Book 1,2,3]

- **Module 4: Numeric Basics:** Unsigned and Signed Integers, Fixed and Floating-point Numbers. Design examples: BCD to 7 segment decoder, 32 bit adders, Shift and add Multiplier, Array Multiplier, signed integer / fraction multiplier. [Text Book 1 and 2]

- **Module 5 :Synchronous sequential circuits:** Moore and Mealy machines, definition of state machines, FSM Design – overlapping and non-overlapping sequence detector. Design example- BCD to excess-3, NRZ to Manchester, Serial adder.[Text Book 1 and 4]

Implementation Fabrics: ICs, PLDs, Packaging and Circuit Boards, Interconnection and Signal Integrity. [Text Book 2]

Text Books:

- Charles H Roth Jr., Lizy Kurian John, Byeong –kill-lee “Digital System Design using Verilog”, *publisher Cengage learning*
- Peter J. Ashenden, “Digital Design: An Embedded Systems Approach Using VERILOG”, *Elesvier*, 2010.
- Ming-Bo Lin, “Digital System designs and Practices using Verilog HDL and FPGAs”, *John Wiley & Sons*, 2008
- Stephan Brown, Zvonko Vranesic “Fundamentals of Digital Logic with Verilog Design”, Reprint 2016, *McGrawHill*, 2nd Edition, 2007

Reference Books:

- Micheal .D. Ciletti “Advanced Digital Design with the Verilog HDL” Prentice hall PTR, 2nd editions. ISBN:0136019285
- Samir Palnitkar, “Verilog HDL-A guide to digital design and synthesis”, *Sunsoft Press*, 1996
- Cyril Prasanna Raj, “Fundamentals of HDL”, *Pearson / Sanguine*, 2010.
- Stephen Brown and Zvonko Vranesic, “Fundamentals of Digital Logic Design with VHDL”, Second Edition, *The McGraw-Hill*, 2009.
- Nazeih M. Botros, “HDL Programming (VHDL and Verilog)”, *John Wiley India Pvt. Ltd.* 2008
- J. Bhaskar, “A Verilog HDL Primer”, *BS Publications*, India.
- Volnei A. Pedroni, “Circuit Design with VHDL”, *PHI*, New Delhi, India
- Wayne Wolf, “FPGA based system design”, Reprint 2005, *Pearson Education* “Electronic Communication Systems”, *McGrawHill*, 4th Edition, 1992

COURSE OUTCOMES

CO1	Describe & model digital blocks of computing systems using Verilog hardware description language..
CO2	Demonstrate the ability to apply Verilog in modeling combinational and sequential circuits and to write a test bench for the same
CO3	Design clocked synchronous circuits and perform timing analysis
CO4	Analyze the Digital design flow with fabrics
CO5	Apply fixed and floating point arithmetic for digital System applications
CO6	Apply design Knowledge to FSM based digital Modules.

Introduction

❑ Evolution of IC technologies

- Moore's Law
- SSI (Small Scale Integration) – 1960 - 1 to 20 gates,
- MSI (Medium Scale Integration) -1966- 20 to 200 gates
- LSI (Large Scale Integration) – 1971- 200 to a 20 thousand gates
- VLSI (Very Large Scale Integration) – 1980 above 20,000 gates

Year	1947	1950	1961	1966	1971	1980	1990	2000
Technology	<i>Invention of the transistor</i>	<i>Discrete components</i>	<i>SSI</i>	<i>MSI</i>	<i>LSI</i>	<i>VLSI</i>	<i>ULSI*</i>	<i>GSI†</i>
Approximate numbers of transistors per chip in commercial products	1	1	10	100–1000	1000–20,000	20,000–1,000,000	1,000,000–10,000,000	>10,000,000
Typical products	—	Junction Transistor and diode	Planar devices Logic gates Flip-flops	Counters Multiplexers Adders	8 bit micro-processors ROM RAM	16 and 32 bit micro-processors Sophisticated peripherals GHM Dram	Special processors, Virtual reality machines, smart sensors	

* Ultra large-scale integration

† Giant-scale integration

Note: The boundary lines between technologies in the table are not artificially created. Crossing each boundary requires new design methodology, simulation approaches, and new methods for determining and routing communications and for handling complexity.

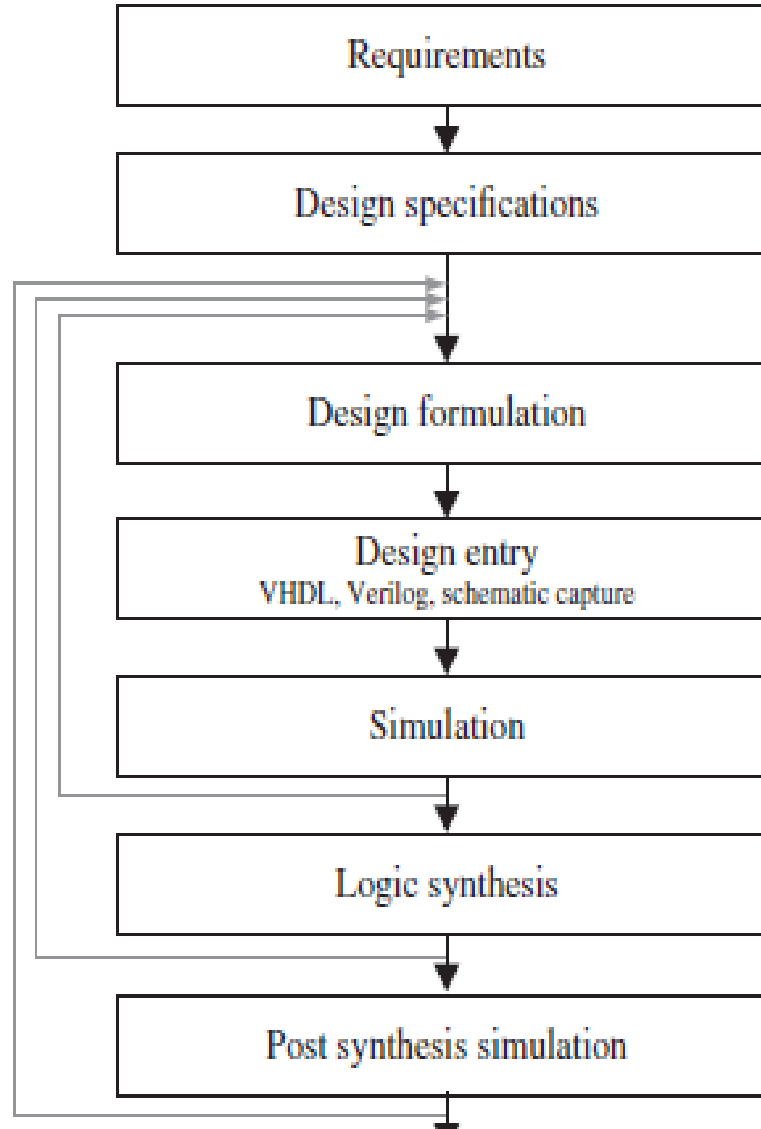
Digital Basics:

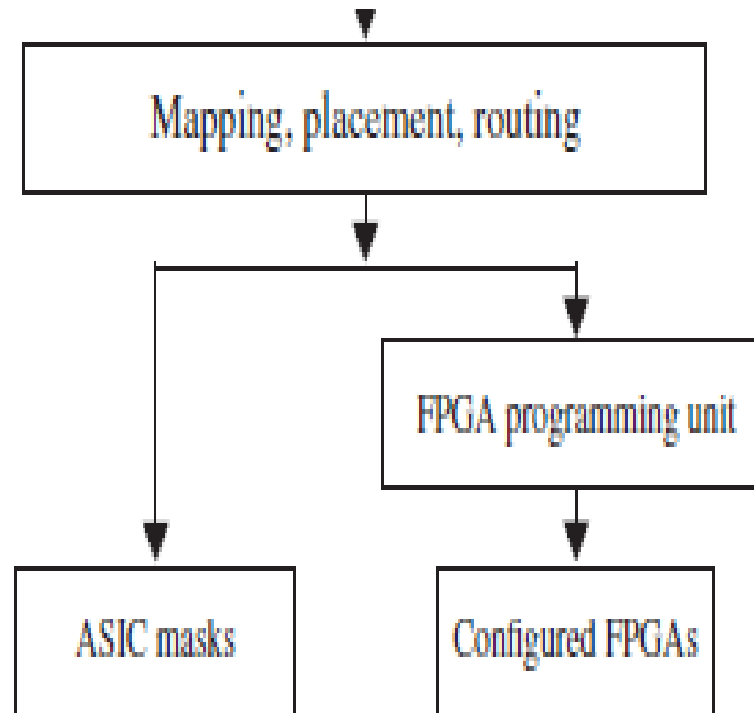
- ❑ Gates characteristics
 - Fan-in,
 - Fan-out,
 - Power Dissipation,
 - Speed (Propagation delay),
 - Noise Margin

- ❑ Logic families
 - Transistor-Transistor logic (TTL)
 - Emitter Coupled logic (ECL)
 - CMOS logic
 - BiCMOS logic

Computer-Aided Design

Design Flow in Modern Digital System Design





- Design requirements and Design specification

This include function requirements (what the product is to do), performance requirements (how fast it is to do it), and constraints on power consumption, cost and packaging, etc.,

- Formulate the design:

- Conceptual level, either at a block diagram level or at an algorithmic level.

- Design entry:

- Previously, this would have been a hand-drawn schematic or blueprint.
- Now with CAD tools, needs to be entered into the CAD system in an appropriate manner.
- Designs can be entered in multiple forms.

- **Schematic capture:** CAD tools used to provide a graphical method to enter designs.

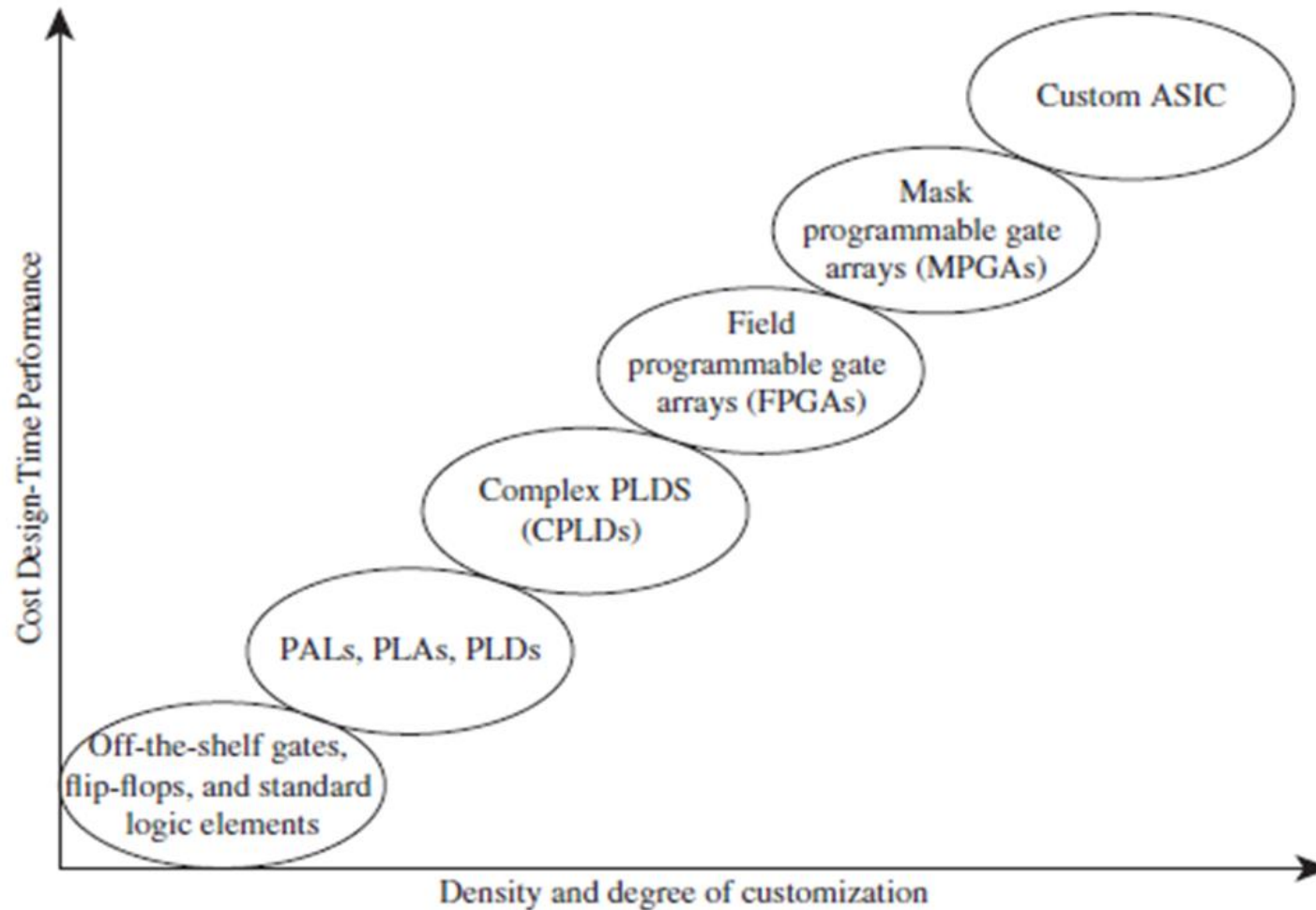
- The schematic editors were supplemented with a library of standard digital building blocks such as gates, flip-flops, multiplexers, decoders, counters, registers, and so forth.

- Hardware description languages (HDLs) are used to enter designs in textual form.
- Two popular HDLs are VHDL and Verilog.
- A hardware description language (HDL) allows a digital system to be designed and debugged at a higher level of abstraction than schematic capture.
- In schematic capture, a designer inputs a schematic with gates, flip-flops, and standard MSI building blocks. However, with HDLs, the details of the gates and flip-flops do not need to be handled during early phases of design.
- A design can be entered in what is called a **behavioral description of the design**.
- Another method to enter a design in VHDL and Verilog is the **structural description entry**.

- After design entry, simulate the design to confirm that the design does function correctly. Initially, simulation at the high-level behavioral model. This early simulation unveils problems in the initial design. If problems are discovered, then the designer goes back and alters the design to meet the requirements.
- Once the functionality of the design has been verified through simulation, the next step is **synthesis**.
- Synthesis is a process of converting the higher-level abstract description of the design in to actual components at the gate and flip-flop levels.
- The output of the synthesis tool, consisting of a list of gates and a list of interconnections, specifying how to interconnect them, is often referred to as a **netlist**.

- A synthesis tool is nothing but a compiler to convert design descriptions to hardware. Synthesis is analogous to writing software programs in a high-level language such as C and then using a compiler to convert the programs to machine language.
- The next step in the design flow is **post-synthesis simulation**. The earlier simulation at a higher level of abstraction does not take into account specific implementations of the hardware components that the design is using.
- If post-synthesis simulation unveils problems, then go back and modify the design to meet timing requirements.
- Arriving at a proper design implementation is an iterative process.
- Next, a designer moves into specific realizations of the design. A design can be implemented in several different target technologies.

- The target technologies that are commonly available now are illustrated in Figure



- The lowest level of sophistication and density is an old-fashioned printed circuit board with off-the-shelf gates, flip-flops, and other standard logic-building blocks.
- Slightly higher in density are programmable logic arrays (PLAs), programmable array logic (PALs), and simple programmable logic devices (SPLDs).
- PLDs with higher density and gate count are called complex programmable logic devices (CPLDs).
- In addition, there are the popular field programmable gate arrays (FPGAs) and mask programmable gate arrays (MPGAs), or simply gate arrays.
- The highest level of density and performance is a fully custom application-specific integrated circuit (ASIC).
- The most common target technologies currently are FPGAs and ASICs.

- The initial steps in the design flow are same for either realization. In the final stages of the design flow, different operations are performed depending on the target technology. This is shown in Figure.
- The design is **mapped** into specific target technology and **placed** into specific parts in the target ASIC or FPGA.
- The paths taken by the connections between components are decided during the **routing**.
- If an ASIC is being designed, the routed design is used to generate a photo mask that will be used in the integrated circuit (IC) manufacturing process.
- If a design is to be implemented in an FPGA, the design is translated to a format specifying what is to be done to various programmable points in the FPGA.
- In modern FPGAs, programming simply involves writing a sequence of 0s and 1s into the programmable cells in the FPGA

HARDWARE DESCRIPTION LANGUAGES (HDL)

- HDL is an acronym of Hardware Description Language
- Hardware Description Languages is a Computer aided Design tool for the modern design and synthesis of digital systems.
- Hardware description languages are a popular mode of design entry for digital circuits and systems. There are two popular HDLs.
 1. VHDL: Very high speed integrated circuits (VHSIC) Hardware Description Language .
 2. Verilog HDL or Simply Verilog
- Before the advent of HDLs designers used graphical schematics and schematic capture tools to document and simulate digital circuits.
- Using HDL, to document and simulate digital circuits using textual form.

- The VHDL language was originally developed under funding from the Department of Defense (DoD), USA in the year 1980. In 1985, IBM, TI and Intermetrics was introduced a standard and publicly available VHDL version 7.2. In 1986, it was standardized by IEEE and later recognized by American National Standard Institute (ANSI) in 1987. In the year 1993, VHDL was updated with more features, one of the important feature is `std_logic_1164` package.
- Verilog is a general-purpose hardware description language that can be used to describe and simulate the operation of a wide variety of digital systems, ranging in complexity from a few gates to an interconnection of many complex integrated circuits.
- Verilog was developed by the industry. It was initially developed as a proprietary language by a company called Gateway Design Automation around 1984.

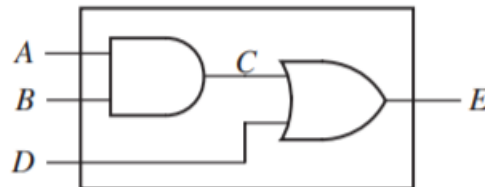
- In 1990, Cadence acquired Gateway Design Automation and became the owner of Verilog. Cadence marketed it as a language and as a simulator. Later it became standard and nonproprietary.
- In 1995, Verilog HDL became IEEE standard 1364-1995. Again, it was revised in 2001 and 2005. The language is presently maintained by the Open Verilog International (OVI) organization.
- HDLs can describe a digital system at several different levels—behavioral, data flow, and structural.
- For example, a binary adder could be described at the behavioral level in terms of its function of adding two binary numbers without giving any implementation details.
- The same adder could be described at the data flow level by giving the logic equations for the adder.
- Finally, the adder could be described at the structural level by specifying the gates and the interconnections between the gates that comprise the adder.

- HDLs lead naturally to a top-down design methodology.
- HDLs are designed to be technology independent.
- Verilog code structure is based on C software language.
- More recently, there also have been efforts in system design languages such as System C, Handel-C, and System Verilog. System C is created as an extension to C.
- Verilog is a hardware description language, it differs from an ordinary programming language in several ways.
- Most importantly, Verilog has statements that execute concurrently since they must model real hardware in which the components are all in operation at the same time.
- It is popularly used for the purposes of describing, documenting, simulating, and automatically generating hardware. Hence, its constructs are tailored for these purposes.

MODULE MODELING STYLES (MODULE DESCRIPTION):

Modules:

- A module is a basic building block that declares the input and output signals and specifies the internal operation of the module.
- Verilog HDL module consists of two parts
 - Internal or body
 - Interface or Port
- Body performs the required function of the module while the interface carries out the required communication between the core circuit and the outside world.
- Example:



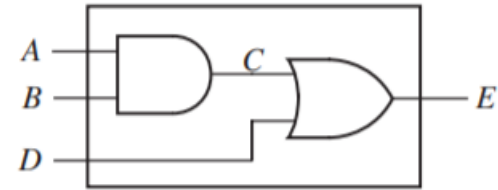
```

module two_gates (A, B, D, E);
output E;

input A, B, D;
wire C;

    assign C = A && B; // concurrent
    assign E = C || D; // statements
endmodule

```



- The module declaration has the name `two_gates` and specifies the inputs and outputs. `A`, `B`, and `D` are input signals, and `E` is an output signal.
- The signal `C` is declared within the module as a wire since it is an internal signal.
- The two concurrent statements that describe the gates are placed and the module ends with `endmodule`.
- All the input and output signals are listed in the module statement without specifying whether they are input or output.

```
module module-name (module interface list);  
[list-of-interface-ports]  
...  
[port-declarations]  
...  
[functional-specification-of-module]  
...  
endmodule
```

```
module half_adder (I1, I2, O1, O2);  
    input I1;  
    input I2;  
    output O1;  
    output O2;  
    //Blank lines are allowed  
  
    assign O1 = I1 ^ I2; //statement 1  
    assign O2 = I1 & I2; //statement 2  
endmodule
```

Modeling the internal of a module: In Verilog, the body can be modeled as one of the following modelling styles

- **Structural style:** Design is described as a set of interconnected components. The components may be primitive gates or primitive switches or modules.

Gate level: a design is said to be modelled at gate level when it only comprises a set of interconnected gate primitives.

Switch level: a design is said to be modelled at switch level when it only comprises a set of interconnected switch primitives.

- **Dataflow style:** The design is described by specifying the dataflow between registers and how the data is processed.

A module is specified as a set of continuous assignment statements.

- **Behavioral or Algorithmic style:** The design is described in terms of the desired design algorithm without concerning the hardware implementation details.

Design can be described in any high level programming languages.

- **Mixed style:** The design is described in terms of the mixing use of above three modelling styles.
 - Mixed style are used in modeling the larger style.

- **Port declaration:** Three types of port declaration in Verilog HDL.

input: Declare a group of signals as input ports

output: Declare a group of signals as output ports

inout: Declare a group of signals as bidirectional ports, i.e port can be used as input and output ports, but not at the same time

- The complete interface of a module is to divide it into three parts
 - Port list
 - Port declaration
 - Data type declaration
- Example: this style of port declaration is known as port style declaration.

```
// port list style
module adder (x, y, c_in, sum, c_out);
input [3:0] x , y;
input c_in;
output [3:0] sum ;
output c_out;
reg [3:0] sum;
reg c_out;
```

In the above example, the declaration of port and it's associated data type can be combined into a single line, as shown in the following.

// port list style

```
module adder (x, y, c_in, sum, c_out);  
input [3:0] x , y;  
input c_in;  
output reg [3:0] sum ;  
output reg c_out;
```

Port connection rules:

- Verilog allows ports to remain unconnected and with different sizes. In addition, unconnected inputs are driven to the “z” state, unconnected outputs are not used.
- Connecting ports to external signals can be done by one of the following two methods:

- Named association: The ports to be connected to external signals specified by listing their names. The order is not important.
- Positional association: The ports are connected to the external signals by an ordered list. The signals to be connected must have the same order as the ports in the port list, leaving the unconnected port blank.

However, these two methods can not be mixed in the same module.

Verilog HDL primitives can only be connected by positional association.

The operation to call, a built-in primitive, a user defined primitive or the other module is called a instantiation and each copy of the called primitive or module is called instance.

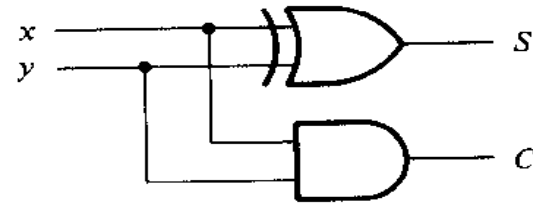
Coding styles:

- ❑ Module can not be declared with in another module
- ❑ A module can instantiate other module
- ❑ A module instantiation must have a module identifier (instance name) except for built in primitives, gate and switch primitives and user defined- primitives.
- ❑ It should use named association at the top level modules to avoid confusion that may arise from synthesis tools.

The following example shows how to instantiate gate primitives and user defined modules, as well as how to connect their ports through nets and input/output ports.

Port connection rules

```
module half_adder (x, y, s, c);  
input  x, y;  
output s, c;  
// -- half adder body-- //  
// instantiate primitive gates  
    xor xor1 (s, x, y);  
    and and1 (c, x, y);  
endmodule
```



(e) $S = x \oplus y$
 $C = xy$

Can only be connected by using positional association

Instance name is optional.

```
module full_adder (x, y, cin, s, cout);  
input  x, y, cin;  
output s, cout;  
wire  s1, c1, c2; // outputs of both half adders  
// -- full adder body-- //  
// instantiate the half adder  
    half_adder ha_1 (x, y, s1, c1);  
    half_adder ha_2 (.x(cin), .y(s1), .s(s), .c(c2));  
    or (cout, c1, c2);  
endmodule
```

Connecting by using positional association


Connecting by using named association

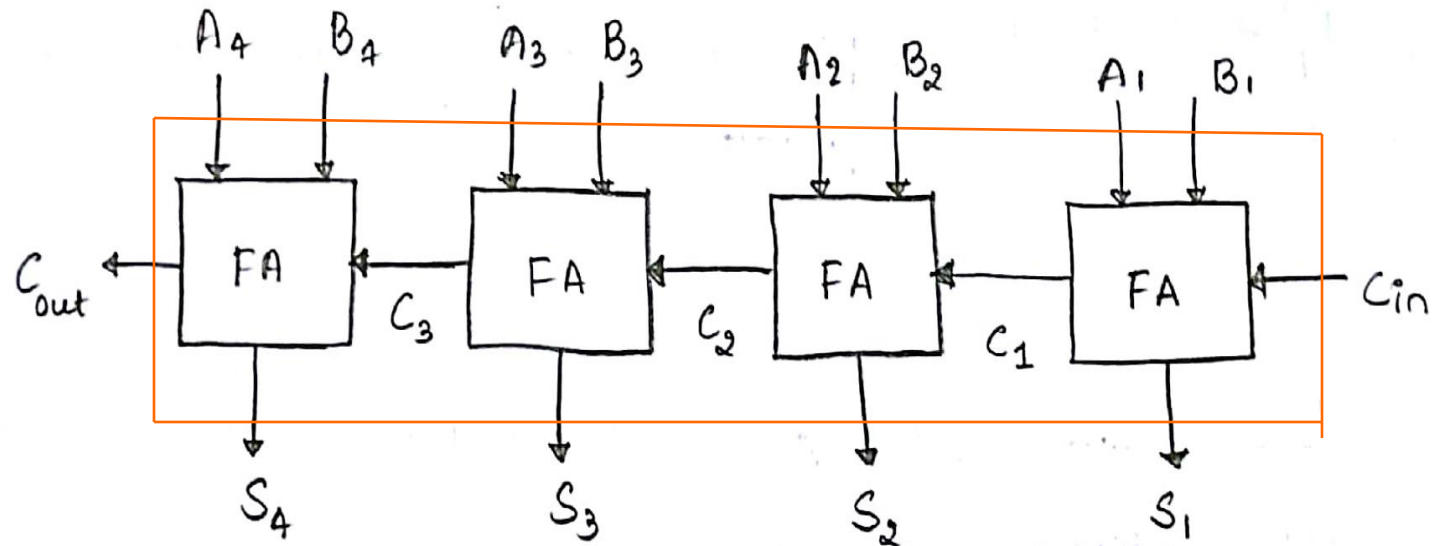
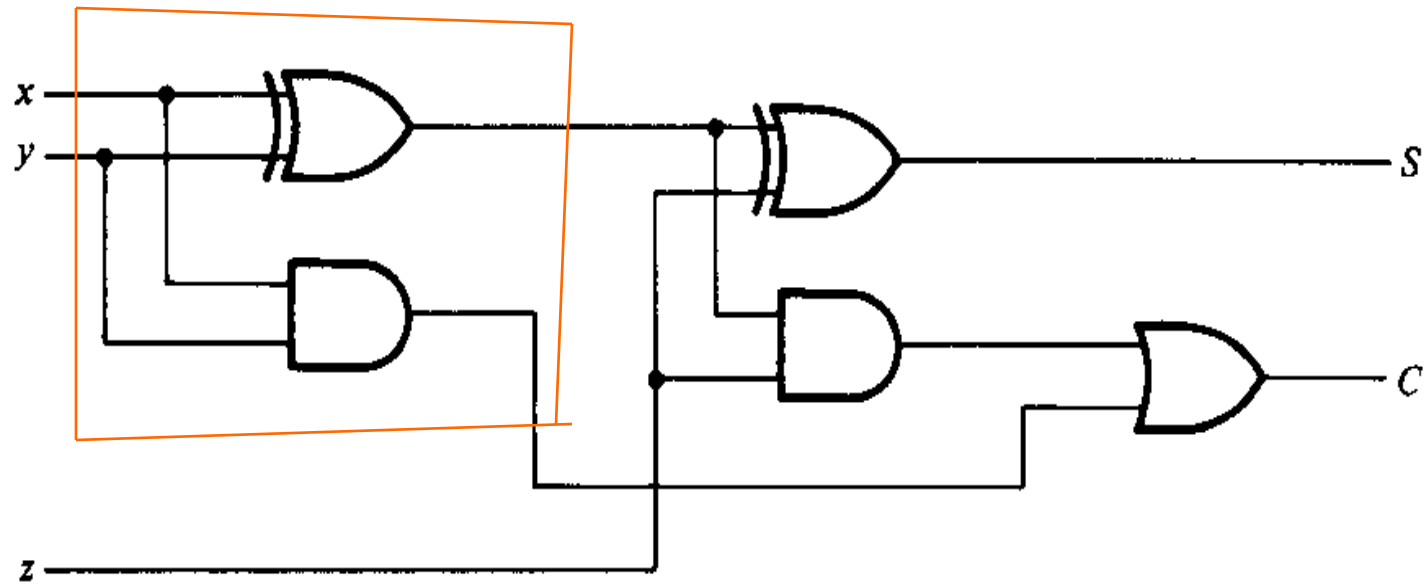
Instance name is necessary.

Structural Modelling:

Example: Structural Modelling at gate level

```
// gate-level hierarchical description of 4-bit adder
// gate-level description of half adder
module half_adder (x, y, s, c);
input  x, y;
output s, c;
// half adder body
// instantiate primitive gates
    xor (s,x,y);
    and (c,x,y);
endmodule
```






```

// gate-level description of full adder
module full_adder (x, y, cin, s, cout);
input  x, y, cin;
output s, cout;
wire   s1, c1, c2; // outputs of both half adders
// full adder body
// instantiate the half adder
    half_adder ha_1 (x, y, s1, c1);
    half_adder ha_2 (cin, s1, s, c2);
    or (cout, c1, c2);
endmodule

// gate-level description of 4-bit adder
module four_bit_adder (x, y, c_in, sum, c_out);
input  [3:0] x, y;
input  c_in;
output [3:0] sum;
output c_out;
wire   c1, c2, c3; // intermediate carries
// four_bit adder body
// instantiate the full adder
    full_adder fa_1 (x[0], y[0], c_in, sum[0], c1);
    full_adder fa_2 (x[1], y[1], c1, sum[1], c2);
    full_adder fa_3 (x[2], y[2], c2, sum[2], c3);
    full_adder fa_4 (x[3], y[3], c3, sum[3], c_out);
endmodule

```

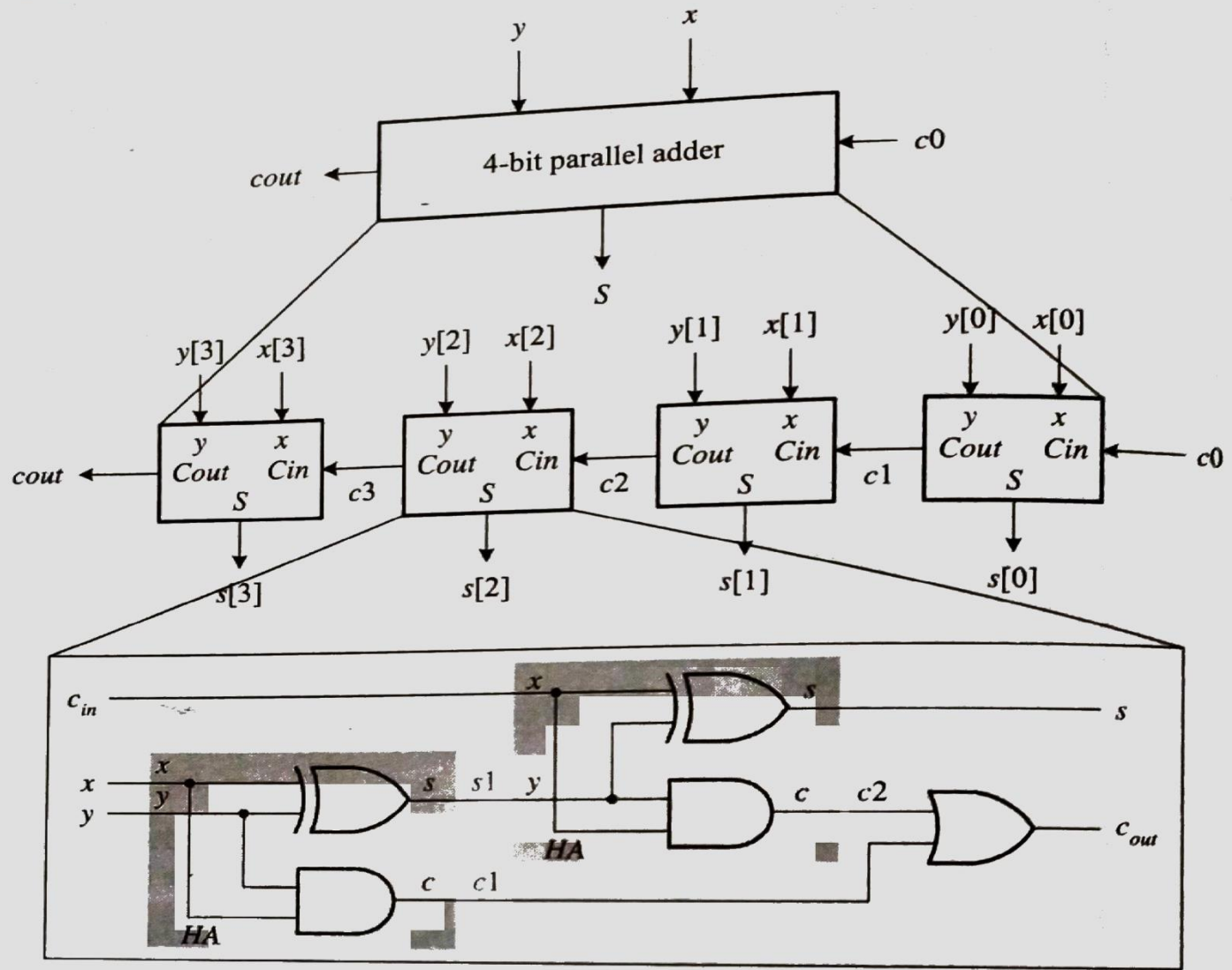


FIGURE 1.5 A hierarchical 4-bit adder

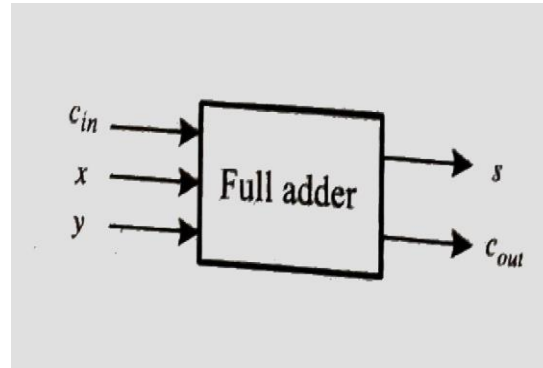
DATAFLOW DESCRIPTION:

In this description, the continuous assignment statement is used to describe the model of any design. The continuous assignment starts with the key word `assign` and has the syntax

```
assign [delay] l_value = expression :
```

The delay specifies the amount of time between a change of operand used in the expression and the assignment value.

If no delay is specified, the default is '0' delay.



```
module full_adder_dataflow(x, y, c_in, sum, c_out);  
  // I/O port declarations  
  input  x, y, c_in;  
  output sum, c_out;  
  
  // specify the function of a full adder  
  assign #5 {c_out, sum} = x + y + c_in;  
endmodule
```


BEHAVIORAL MODELING:

- Uses two constructs: Initial and Always
- Initial statement can be executed only once and therefore is usually used to set up the initial values of variable data types.
- Always statement is executed repeatedly. This statement is used to model combinational or sequential logic.
- All initial and always statements begin their execution at 0 simulation time concurrently.

A Full Adder Modeling in Behavioral Style

In this modeling style, description is similar to dataflow, except that it needs to be put in always statement

```
module full_adder_behavioral(x, y, c_in, sum, c_out);  
// I/O port declarations  
input  x, y, c_in;  
output sum, c_out;  
reg    sum, c_out; // sum and c_out need to be declared  
                // as reg types.  
  
// specify the function of a full adder  
always @(x, y, c_in) // can also use always @(*) or  
                // always@(x or y or c_in)  
    #5 {c_out, sum} = x + y + c_in;  
endmodule
```



Mixed modeling style:

- This description is used to construct a hierarchical design in large systems.
- In this example, a full adder is constructed with two half adders and an OR gate.
- The first HA is modeled in structural style, the second HA in dataflow and the OR gate in behavior style.


```
module full_adder_mixed_style(x, y, c_in, s, c_out);  
  // I/O port declarations  
  input  x, y, c_in;  
  output s, c_out;  
  reg    c_out;  
  wire   s1, c1, c2;  
  // structural modeling of HA 1.  
  xor xor_ha1 (s1, x, y);  
  and and_ha1(c1, x, y);  
  // dataflow modeling of HA 2.  
  assign s = c_in ^ s1;  
  assign c2 = c_in & s1;  
  // behavioral modeling of output OR gate.  
  always @(c1, c2) // can also use always @(*)  
    c_out = c1 | c2;  
endmodule
```