# Hardware Description Language (HDL)

# Digital Design Flow

Behavior

RTL (Register Transfer Level)

Logic Circuit (Gates & FFs)

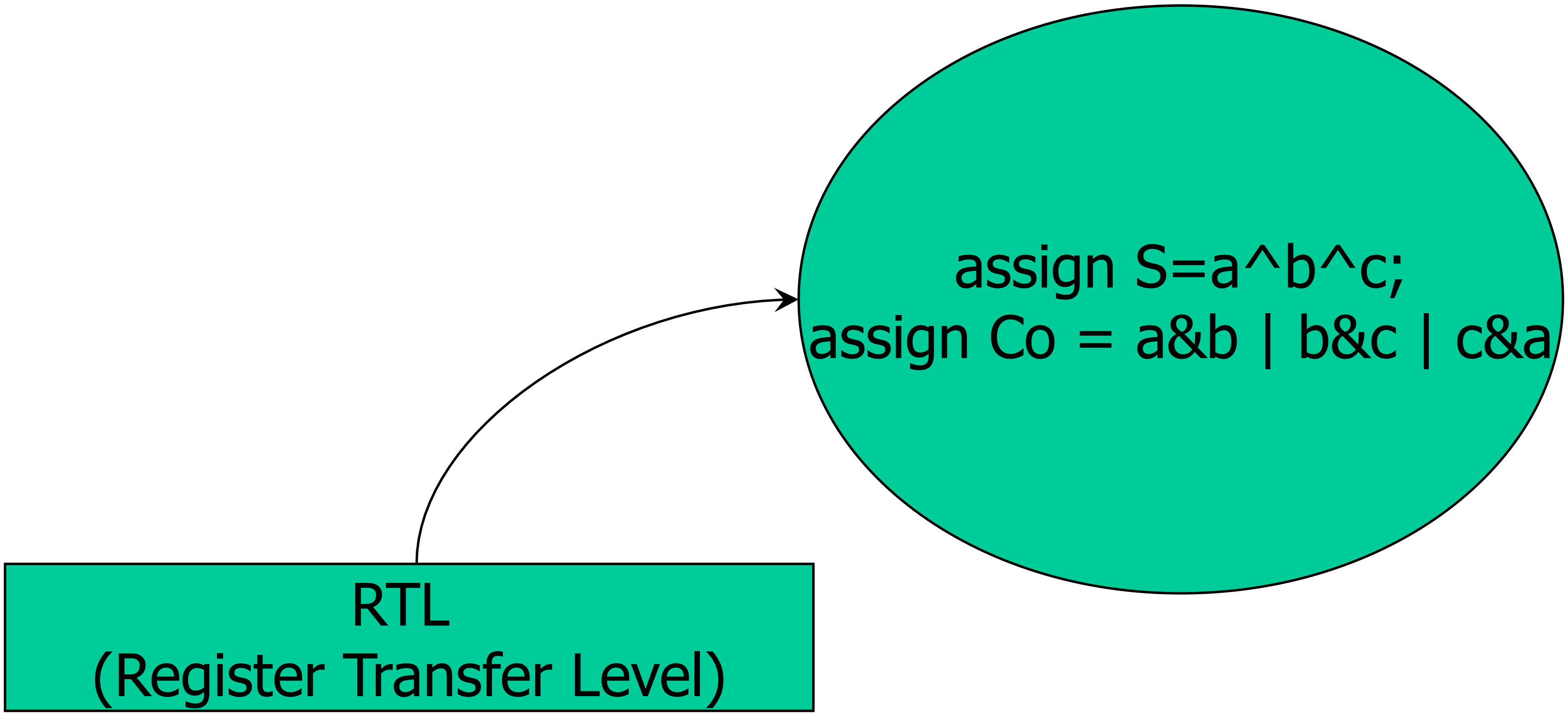
Layout

# Abstractions in Digital IC Design - Example [1]

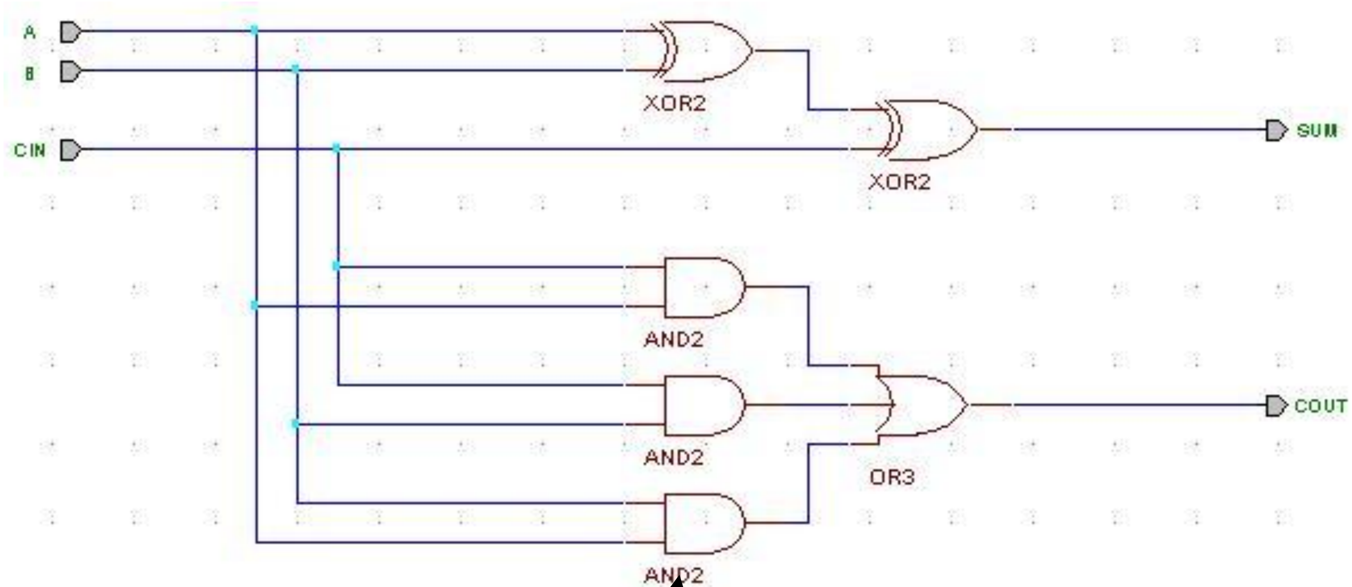
S = a + b + c
Co = Carry generated in the above operation

Behavior

# Abstractions in Digital IC Design - Example [2]



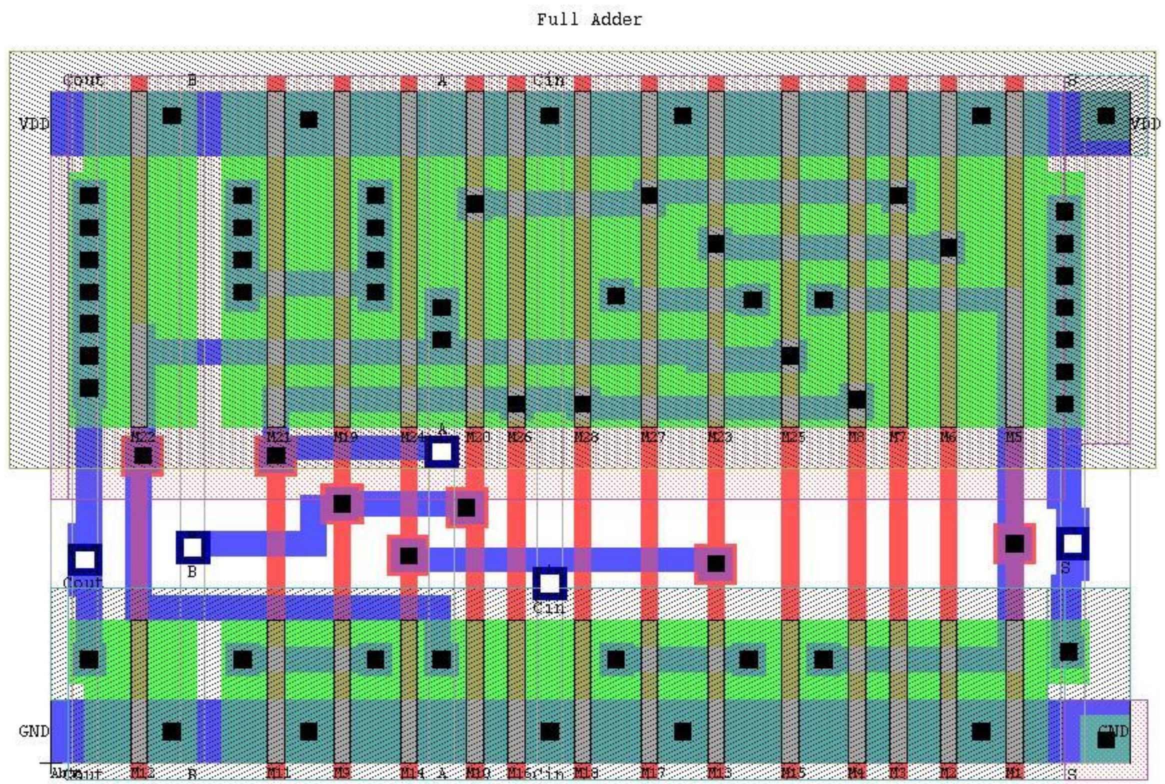
# Abstractions in Digital IC Design – Example [3]




Logic Circuit
(Gates & FFs)

# Abstractions in Digital IC Design

Layout



# Hardware Description Language (HDL)

- **A Programming language used to describe hardware**

- **It resembles a programming language, but is specifically oriented to describing hardware structures and behaviors.**

- **The main difference with the traditional programming languages is HDLs 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.**

# Hardware Description Language (HDL)

- **When a language is used for the above purpose (i.e. to provide an alternative to schematics), it is referred to as a *structural description* in which the language describes an interconnection of components.**

- **Such a structural description can be used as input to logic simulation just as a schematic is used.**

# Commercial HDLs…

- VHDL – VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

- Verilog HDL – **Veri**fying **Log**ic HDL

- System C, System Verilog – Gaining Popularity

## HDL Processing consists of two applications.

- Logic Simulation

A simulator interprets the HDL description, produces readable outputs ex. Timing diagram.

Simulation allows detection of functional errors in design

Stimulus that tests the functionality of design is test bench

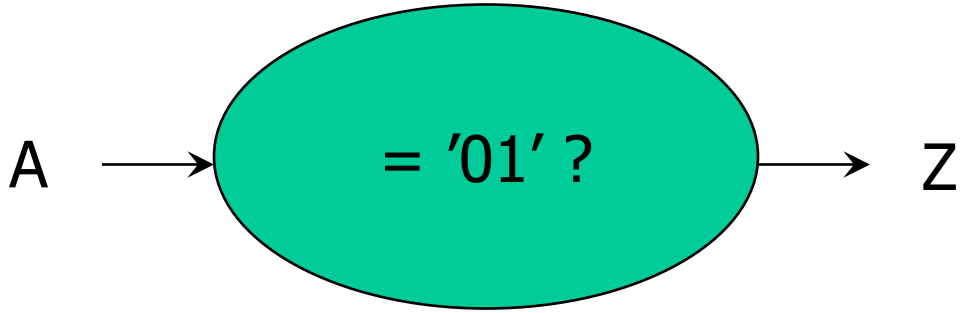
- Logic synthesis

Deriving a list of components and their interconnections from the HDL description of the digital system.

The gate level netlist is used to fabricate the IC.

# Using HDLs

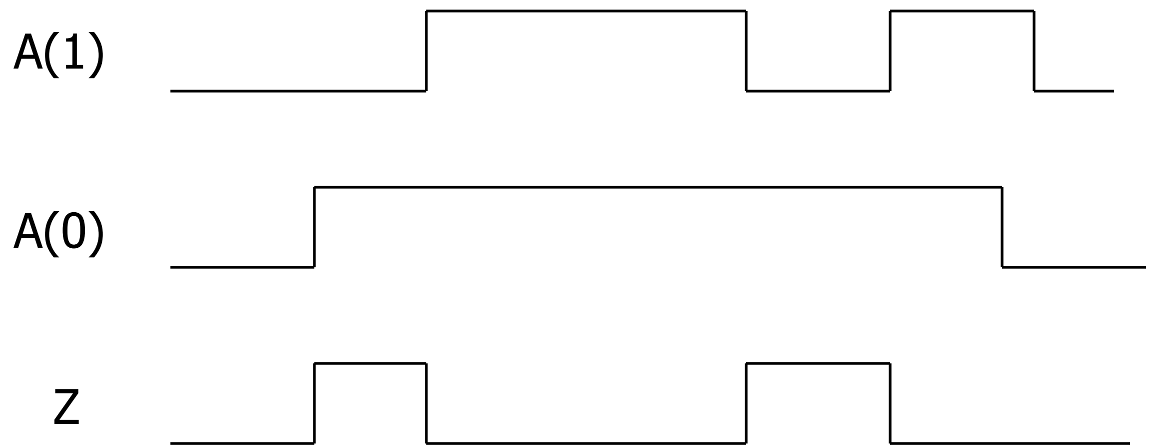
## Modeling a specification




Z <= '1' when A = '01' else '0'

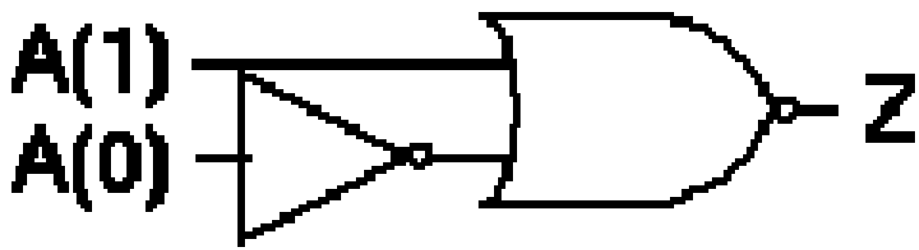
# Using HDLs... (2)

Simulating a model



# Using HDLs... (3)

Synthesizing a simulated model



# Verilog HDL

- Verilog HDL is
  - C - like syntax
  - Initially developed as a simulation tool
  - With synthesis tools added later used for hardware implementation of the code.
- History
  - Developed as proprietary language in 1985
  - Opened as public domain spec in 1990
  - Became IEEE standard in 1995

# Verilog HDL

- Verilog constructs are *keywords (lowercase)*
  - Examples: and, or, wire, input, output
- Verilog includes gate level primitives

  A two input AND gate with inputs x1, x2 and output y is denoted as

  **and (y,x1,x2)**
- Verilog has all the standard gates
    - **`and, nand`**
    - **`or,  nor`**
    - **`xor, xnor`**
    - **`not, buf`**

# Verilog HDL

- One important construct is the *module*
  - Modules have inputs and outputs
  - Modules can be built up of Verilog primitives or of user defined sub modules.
- A logic circuit is specified in the form of a module. (building block)
- The module is a text description of the circuit
- Starts with *module* and ends with *endmodule*
- Module may have inputs, outputs referred to as ports

# Simple Circuit Notes

- The module starts with ***`module`*** keyword and finishes with ***`endmodule`***.
- Internal signals are named with ***`wire`***.
- Comments follow ***`//`***
- ***`input`*** and ***`output`*** are ports. These are placed at the start of the module definition.
- Each statement ends with a semicolon, except ***`endmodule`***.

# GATE LEVEL MODELLING

# Example: Simple Circuit Diagram

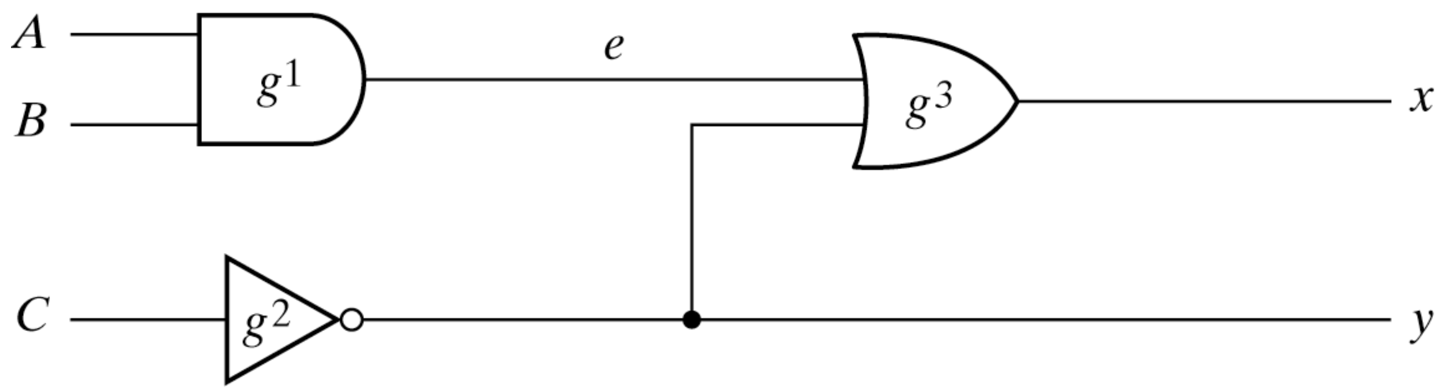



Fig. 3-37 Circuit to Demonstrate HDL

# Example: Simple Circuit HDL

```
module
smpl_circuit(A,B,C,x,y);
    input A,B,C;
    output x,y;
    wire e;
    and g1(e,A,B);
    not g2(y, C);
    or  g3(x,e,y);
endmodule
```

# Adding Delays

- To simulate a circuits real world behavior it is important that propagation delays are included.
- The units of time for the simulation can be specified with **`timescale`**.
  - Default is 1ns with precision of 100ps
- Component delays are specified as #(delay)

# Simple Circuit with Delay

```
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
```

# Input signals

- In order to simulate a circuit the input signals need to be known so as to generate an output signal.
- The input signals are often called the circuit *stimulus*.
- An HDL module is written to provide the circuit stimulus. This is known as a *testbench*.

# Signal Notation

- In Verilog signals are generalised to support multi-bit values (e.g. for buses)
  - The notation

```
A =  1'b0;
```

  - means signal A is one bit with value zero.
- The end of the simulation is specified with `$finish`.

# Testbench

- The *testbench* module includes the module to be tested.
- There are no input or output ports for the testbench.
- The inputs to the test circuit are defined with **`reg`** and the outputs with **`wire`**.
- The input values are specified with the keyword **`initial`**
- A sequence of values can be specified between **`begin`** and **`end`**.

# Stimulus module 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
```

# Effect of delay

| Time (ns) | Input A B C | Output y e x |
|---|---|---|
| - | 0 0 0 | 1 0 1 |
| - | 1 1 1 | 1 0 1 |
| 10 | 1 1 1 | 0 0 1 |
| 20 | 1 1 1 | 0 0 1 |
| 30 | 1 1 1 | 0 1 0 |
| 40 | 1 1 1 | 0 1 0 |
| 50 | 1 1 1 | 0 1 1 |

# Timing Diagram

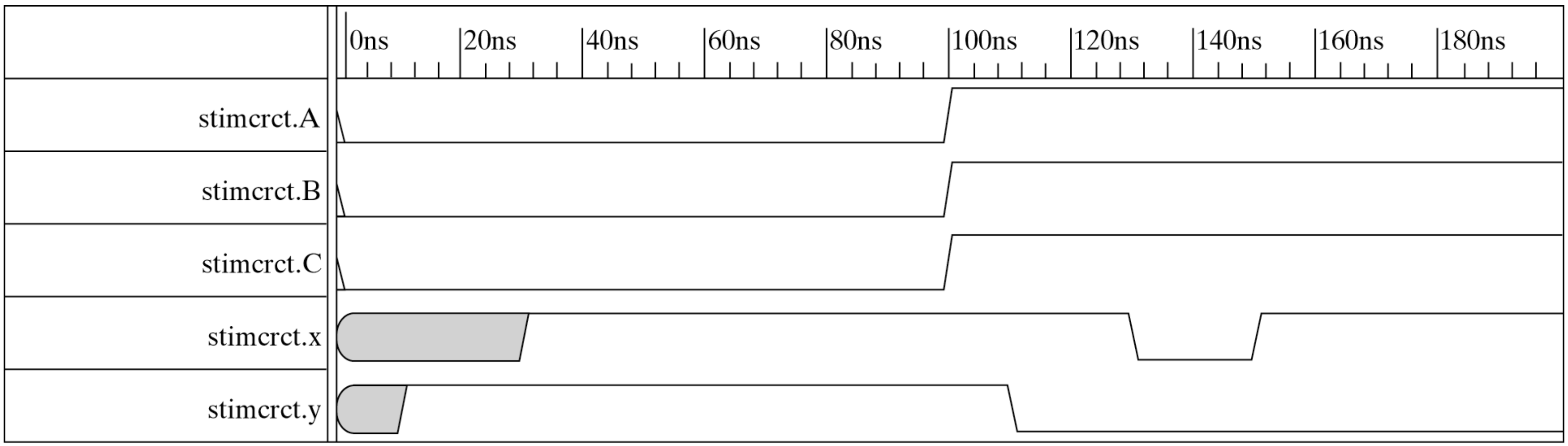
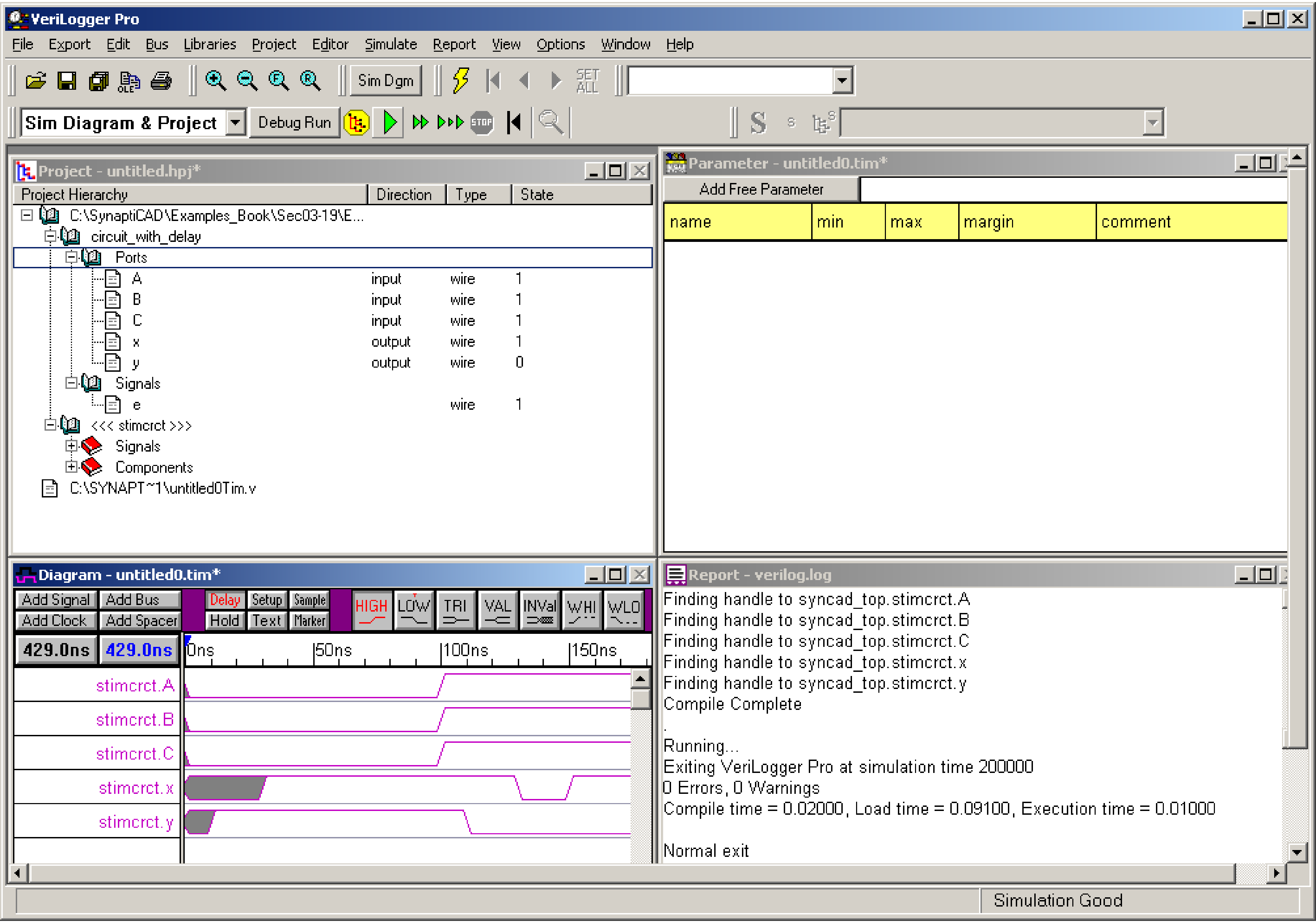
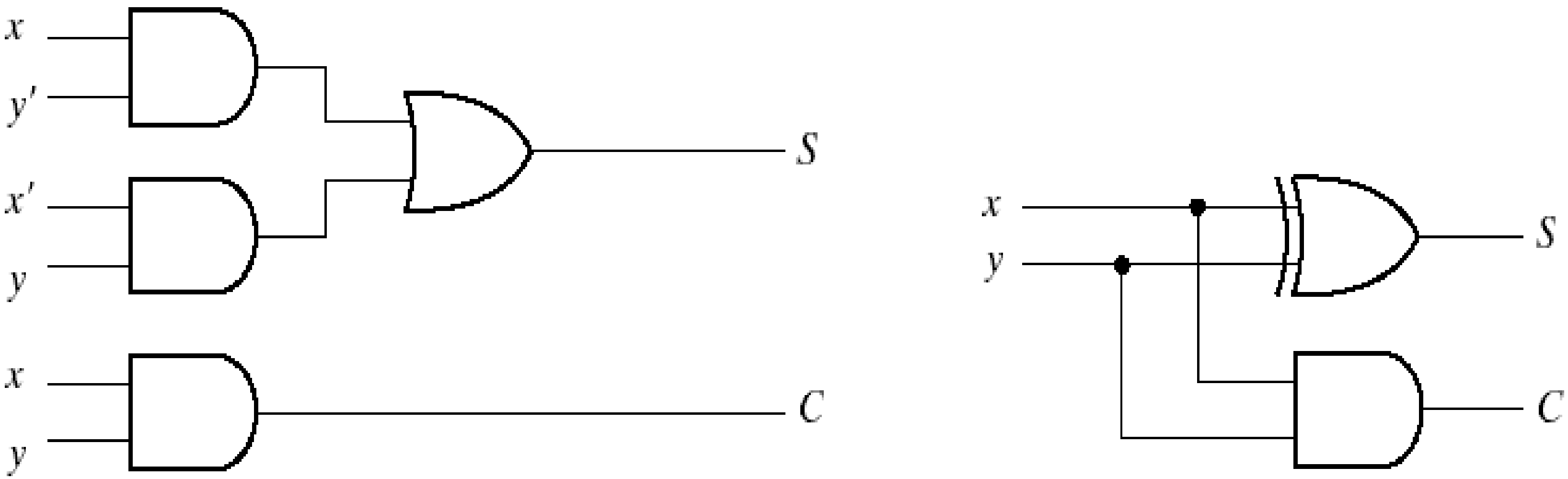



Fig. 3-38 Simulation Output of HDL Example 3-3


VeriLogger Pro
File Export Edit Bus Libraries Project Editor Simulate Report View Options Window Help
Sim Dgm
Sim Diagram & Project
Debug Run
Project - untitled.hpj*
Project Hierarchy
Direction
Type
State
C:\SynaptiCAD\Examples_Book\Sec03-19\E...
circuit_with_delay
Ports
A input wire 1
B input wire 1
C input wire 1
x output wire 1
y output wire 0
Signals
e wire 1
<<< stimcrct >>>
Signals
Components
C:\SYNAPT~1\untitled0Tim.v
Parameter - untitled0.tim*
Add Free Parameter
name
min
max
margin
comment
Diagram - untitled0.tim*
Add Signal
Add Bus
Add Clock
Add Spacer
Delay
Setup
Sample
Hold
Text
Marker
HIGH
LOW
TRI
VAL
INVal
WHI
WLO
429.0ns
429.0ns
0ns
50ns
100ns
150ns
stimcrct.A
stimcrct.B
stimcrct.C
stimcrct.x
stimcrct.y
Report - verilog.log
Finding handle to syncad_top.stimcrct.A
Finding handle to syncad_top.stimcrct.B
Finding handle to syncad_top.stimcrct.C
Finding handle to syncad_top.stimcrct.x
Finding handle to syncad_top.stimcrct.y
Compile Complete
Running...
Exiting VeriLogger Pro at simulation time 200000
0 Errors, 0 Warnings
Compile time = 0.02000, Load time = 0.09100, Execution time = 0.01000
Normal exit
Simulation Good

# 4 bit Full Adder




(a) $S = xy' + x'y$
$C = xy$

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

Fig. 4-5 Implementation of Half-Adder

# 4 bit Full Adder

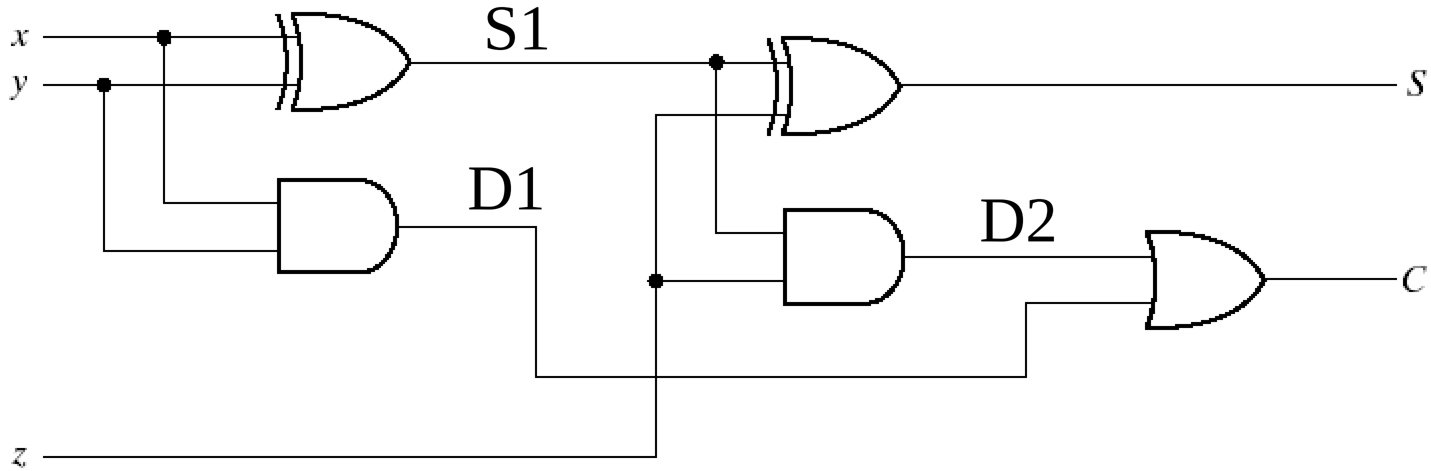



Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

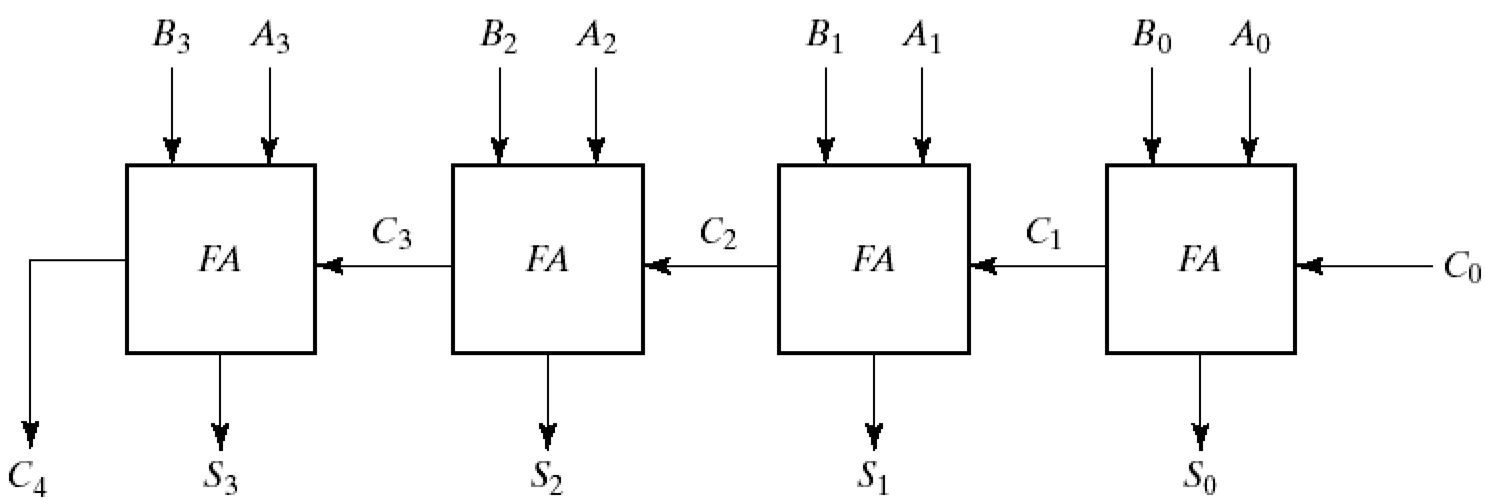



Fig. 4-9 4-Bit Adder

**A vector is specified within square brackets and two numbers separated with colon.**

output [0:3] D;

wire [7:0] SUM;

- output vector D with four bits 0 through 3

- wire vector SUM with eight bits numbered 7 through 0

# 4 bit Full Adder

```
//Gate-level hierarchical description of 4-bit adder
// Description of half adder
module halfadder (S,C,x,y);
   input x,y;
   output S,C;
//Instantiate primitive gates
   xor (S,x,y);
   and (C,x,y);
endmodule

//Description of full adder
module fulladder (S,C,x,y,z);
   input x,y,z;
   output S,C;
   wire S1,D1,D2; //Outputs of first XOR and two AND gates
//Instantiate the halfadder
    halfadder HA1 (S1,D1,x,y),
              HA2 (S,D2,S1,z);
    or g1(C,D2,D1);
endmodule
```

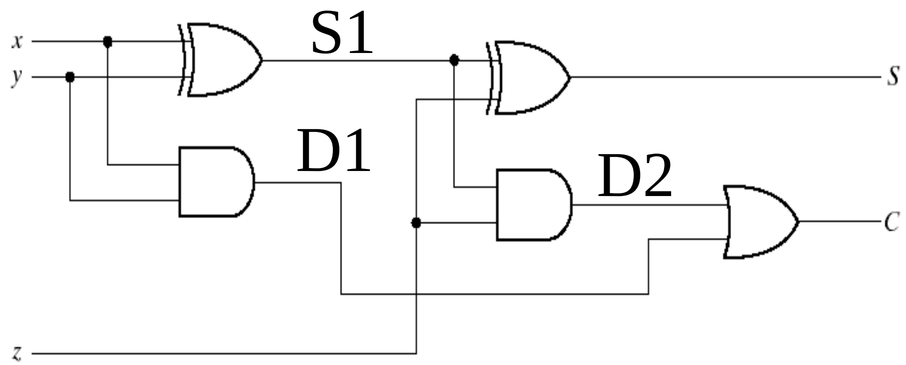



Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

```
//Description of 4-bit adder
module _4bit_adder (S,C4,A,B,C0);
   input [3:0] A,B;
   input C0;
   output [3:0] S;
   output C4;
   wire C1,C2,C3;  //Intermediate carries
//Instantiate the fulladder
   fulladder  FA0 (S[0],C1,A[0],B[0],C0),
              FA1 (S[1],C2,A[1],B[1],C1),
              FA2 (S[2],C3,A[2],B[2],C2),
              FA3 (S[3],C4,A[3],B[3],C3);
endmodule
```

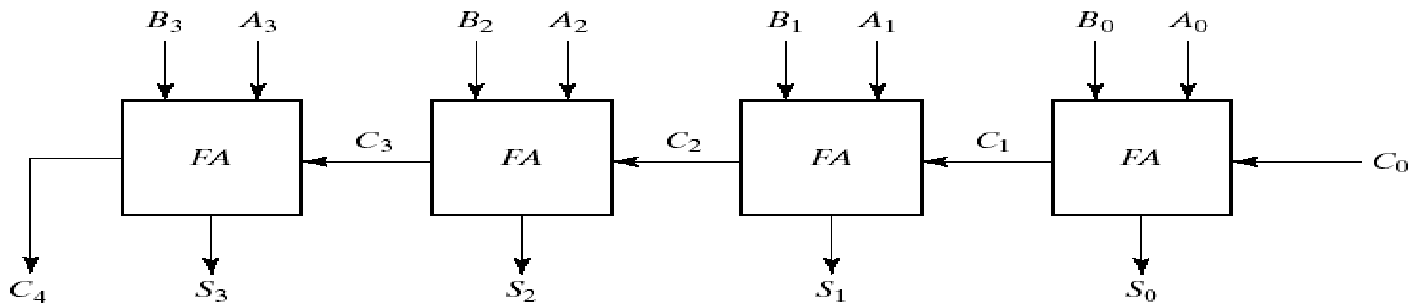



Fig. 4-9 4-Bit Adder

# DATA FLOW MODELLING

# Dataflow modelling

- Another level of abstraction is to model dataflow.
- Dataflow modeling uses a number of operators that act on operands to produce desired results.
- Verilog HDL provides about 30 operator types.
- In dataflow models, signals are continuously assigned values using the `assign` keyword.

# Dataflow Modeling

- For ex: assign Y = (A &B) | (C &D)
- **`assign`** can be used with Boolean expressions.
  - **Verilog uses & (and), | (or), ^ (xor) and ~ (not)**
- Logic expressions and binary arithmetic are also possible.

# Simple Circuit Boolean Expression

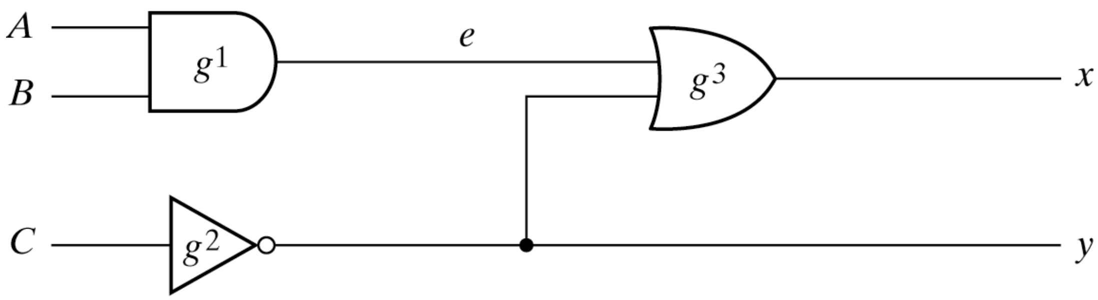



Fig. 3-37 Circuit to Demonstrate HDL

$$x = A.B + \overline{C}$$

$$y = \overline{C}$$

# Boolean Expressions

```
//Circuit specified with Boolean
equations
module circuit_bln (A,B,C,x,y);
    input A,B,C;
    output x,y;
    assign x = (A & B)|(~C);
    assign y = ~C ;
endmodule
```