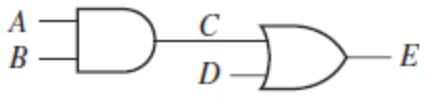
INTRODUCTION TO VERILOG

HDL – continuous assignments

- Modeling combinational circuits in Verilog – concurrent statements or continuous assignments – always ready to execute.

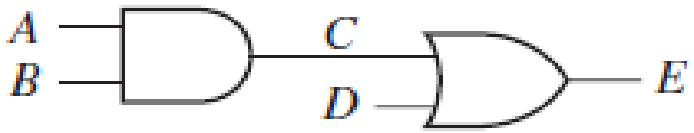
- Evaluated anytime and every time a signal on the right side of the statement changes.

- Signal in Verilog – corresponds to signal in a physical system.




```
assign #5 C = A && B;
assign #5 E = C || D;
```

HDL – continuous assignments




```
assign #5 C = A && B;
assign #5 E = C || D;
```

- && AND gate, || OR gate
- #5 – delay symbol of 5 ns, = signal assignment operator.
- Assign statement.
- First statement evaluated anytime A or B changes. Second statement evaluated anytime C or D changes.
- Continuous assignments.

HDL – continuous assignments

- Verilog simulator monitors the right side of each continuous statements
- Anytime a signal changes, the expression on the right side is immediately evaluated.
- New value assigned to the signal on the left side after an appropriate delay.
- The order of the continuous assignment unimportant.

```
assign #5 E = C || D;
assign #5 C = A && B;
```

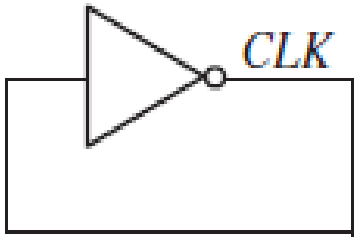
HDL – continuous assignments

```
assign [#delay] signal_name = expression;
```

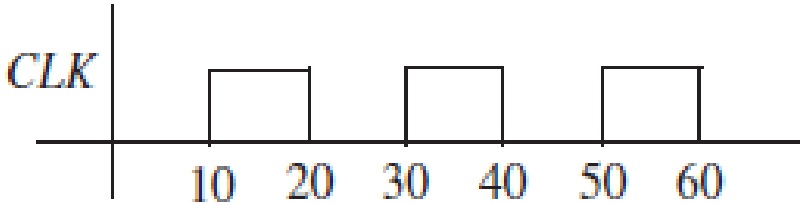
- Signal on the left side is scheduled to change after a delay.
- Delay is optional - Delay omitted – signal updated immediately.
- Time at which the statement executes and the time at which the signal is updated are not the same – specification of delay.

HDL – continuous assignments

- No explicit loops – continuous statements execute repeatedly as of they were in a loop.




```
assign #10 CLK = ~CLK;
```



assign #10 CLK = ~CLK;

Inverter with feedback

HDL – continuous assignments

- Verilog case sensitive - Compiler and simulator treats lower case and upper case differently.
- Every Verilog statement terminated with semicolon.
- Lexical conventions – similar to those in the C programming language.
- Anything follows with a // - treated as a comment to the end of the line.
- Comments: for readability and documentation.

HDL

```
a = b && c; // This is a one-line comment

/* This is a multiple line
    comment */

/* This is /* an illegal */ comment */

/* This is //a legal comment */
```

- Keywords – special identifiers reserved to define language constructs - in lower case.
- And, or, always – reserved keywords – special meaning to Verilog compiler.

List of Verilog keywords

always
and
assign
automatic
begin
buf
bufif0
bufif1
case
casex
casez
cell
cmos
config
deassign
default
defparam
design
disable
edge
else
end
endcase
endconfig
endfunction
endgenerate
endmodule
endprimitive
endspecify
endtable
endtask
event
for
force
forever
fork
function
generate
genvar
highz0
highz1
if
ifnone
incdir
include
initial
inout
input
instance
integer
join
large
liblist
library
localparam
macromodule
medium
module
nand
negedge
nmos
nor
noshowcancelled
not
notif0
notif1
or
output
parameter
pmos
posedge
primitive
pull0
pull1
pulldown
pullup
pulsestyle_onevent
pulsestyle_ondetect
rcmos
real
realtime
reg
release
repeat
rnmos
rpmos
rtran
rtranif0
rtranif1
scalared
showcancelled
signed
small
specify
specparam
strong0

List of verilog keywords

strong1	tri0	wait
supply0	tri1	wand
supply1	triand	weak0
table	trior	weak1
task	trireg	while
time	unsigned	wire
tran	use	wor
tranif0	uwire	xnor
tranif1	vectored	xor
tri		

HDL

- Identifiers – names given to objects – can be referenced in the design.

```
reg value;          // reg is a keyword; value is an identifier

input clk;          // input is a keyword, clk is an identifier
```

```
adder
Mux_input
_error_code
Index_bit
vector_sz
_$five
Count
XOR
```

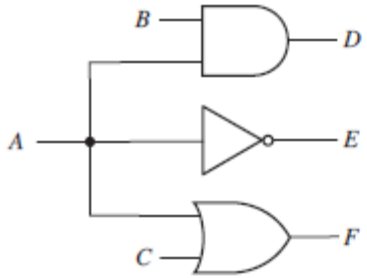
The following are invalid identifiers:

```
4bitadder
$error_code
```

HDL

- Assignment statements cannot start with a number or dollar sign.
- Dollar sign reserved as the first character for system tasks.

HDL




```
// when A changes, these concurrent
// statements all execute at the
// same time
  assign #2 D = A && B;
  assign #1 E = ~A;
  assign #3 F = A || C;
```

Three gates with a common input and different delays.

- Three concurrent statements execute simultaneously whenever A changes.
- Gates with different delays – gate outputs change at different times.

HDL

Operators:

```
a = ~ b; // ~ is a unary operator. b is the operand
a = b && c; // && is a binary operator. b and c are operands
a = b ? c : d; // ?: is a ternary operator. b, c and d are operands
```

Number specifications:

```
4'b1111 // This is a 4-bit   binary number
12'habc // This is a 12-bit  hexadecimal number
16'd255 // This is a 16-bit  decimal number.
```

- Numbers specified without base format – decimal numbers by default.
- Un sized numbers – default number of bits depends on simulator and machine specific (at least 32 bits).

HDL

```
23456 // This is a 32-bit   decimal number by default
'hc3 // This is a 32-bit   hexadecimal number
'o21 // This is a 32-bit   octal number
```

HDL – value sets

- Verilog supports four values – model the functionality of real hardware.

Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
z	High impedance, floating state

HDL – value sets

- Verilog's nets and registers hold four-valued data.
- **0, 1**: Obvious.
- **Z**: Output of an un-driven tri-state driver. Models case where nothing is setting a wire's value.
- **X**: Models when the simulator can't decide the value
- Initial state of registers
- When a wire is being driven to 0 and 1 simultaneously
- Output of a gate with Z inputs

Truth tables for logic gates

and	i1: 0	1	x	z
i2: 0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

nand	i1: 0	1	x	z
i2: 0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

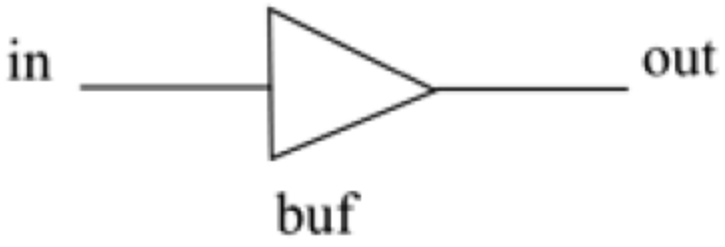
or	i1: 0	1	x	z
i2: 0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

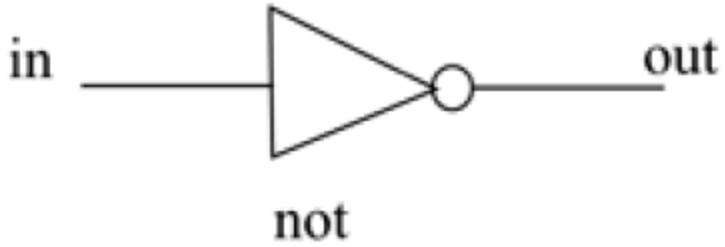
nor	i1: 0	1	x	z
i2: 0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

xor	i1: 0	1	x	z
i2: 0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

xnor	i1: 0	1	x	z
i2: 0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

Truth tables for logic gates





buf	in	out
	0	0
	1	1
	x	x
	z	x

not	in	out
	0	1
	1	0
	x	x
	z	x

Pre-defined data types in Verilog

- nets - connections between hardware elements declared with keywords such as wire.
- variables - data storage elements that can retain values - declared with the keywords such as reg.
- integer - an integer is a variable data type - declared with the keyword integer.
- real - real number constants and real variable data types for floating point number - declared with the keyword real.

Pre-defined data types in Verilog

- time - a special variable data type to store time information - declared with the keyword time.
- Vectors - wire or reg data types can be declared as vectors (multiple bits).
- Vectors can be declared with [range1 : range2])

HDL - Nets

- Nets- connections between hardware elements.

- a - net - Declare nets – keyword wire.
- One bit values by default – unless declared explicitly as vectors.
- Default value of net - z

HDL - Nets

wire a; // Declare net a for the above circuit

wire b,c; // Declare two wires b, c for the above circuit.

wire d = 1'b0; // Net d is fixed to logic value 0 at declaration.

Net is not a keyword – represents a class of data type such as wire.

HDL - registers

- Registers – data storage elements.
- Retain their value, until another value placed onto them.
- Verilog – register means a variable that can hold a value.
- Value of register can be changed anytime in simulation – assigning new value to the register.
- Commonly declared by a keyword reg.
- Default value of reg data type – x.

HDL - registers

```
reg reset;                  // declare a variable reset that can hold
                            its value

initial
begin
reset = 1'b1;               //initialize reset to 1 to reset the
                            digital circuit.
#100 reset = 1'b0;          // after 100 time units reset is
                            deasserted.
end
```

HDL - Registers

- Vectors – nets or reg data types – can be declared as vectors – multiple bit width - One dimensional array of bit signals

- Bit width not specified – default scalar 1 bit.

```
wire a;                          // scalar net variable, default

wire [7:0] bus;                  // 8-bit bus

wire [31:0] busA, busB, busC;          // 3 buses of 32-
                                          bit width.
reg clock;                       // scalar register, default
reg [0:40] virtual_addr;         // Vector register, virtual
```

HDL - Registers

- Left number in the squared bracket – MSB of the vector.
- Bit 0 – MSB of vector – virtual addr.
- Vector part select:

busA[7] // bit # 7 of vector busA

bus[2:0] // Three LSBs of vector bus,

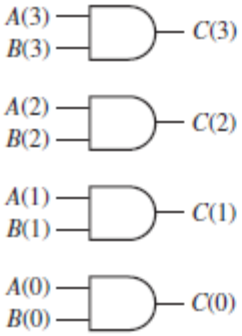
- Using bus[0:2] is illegal because the significant bit should always be on the left of a range specification.
- virtual_addr[0:1] // Two MSBsof vector virtual_addr

HDL - Vectors

Multiple bit wire can be declared using a statement:

```
wire B[3:0];
```

The statement B = 4'b1100 assigns 1 to *B[3]*, 1 to *B[2]*, 0 to *B[1]*, and 0 to *B[0]*.




```
// the hard way
assign C[3] = A[3] && B[3];
assign C[2] = A[2] && B[2];
assign C[1] = A[1] && B[1];
assign C[0] = A[0] && B[0];
```

```
// the easy way assuming C, A and
// B are 4-bit vectors

assign C = A & B;
```

Array of AND gates

Understand the symbols for normal AND and bitwise AND.

HDL – system tasks

- System tasks – provides for certain routine operations.
- System tasks appear in the form $<keyword>.
- Display on the screen, monitoring values of nets, stopping and finishing – done by system tasks.
- Displaying information -
- Usage: $display(p1, p2, p3,....., pn);

HDL – system tasks

- Monitoring information – mechanism to monitor a signal when its value changes.

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

```
begin
$monitor($time,
" Value of signals clock = %b reset = %b", clock,reset);
End
```

Partial output of the monitor statement:

```
-- 0 Value of signals clock = 0 reset = 1
-- 5 Value of signals clock = 1 reset = 1
-- 10 Value of signals clock = 0 reset = 0
```

Stop and finish tasks

```
// Stop at time 100 in the simulation and examine the
results
// Finish the simulation at time 1000.

initial                 // to be explained later. time = 0
begin
clock = 0;
reset = 1;
#100 $stop;             // This will suspend the simulation
                        at time = 100
#900 $finish;           // This will terminate the simulation
                        at time = 1000
end
```

Verilog modules

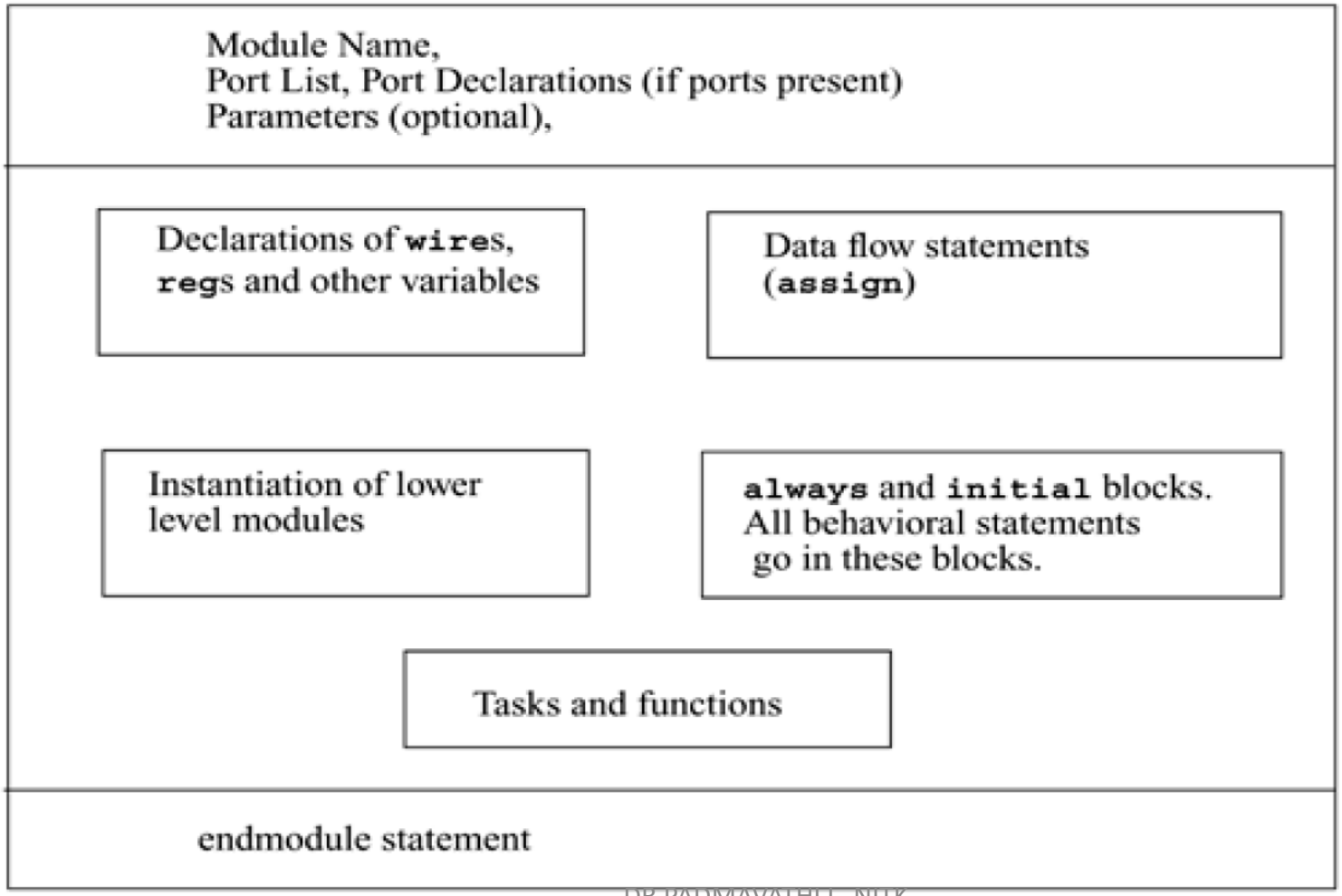
- When you describe a system in Verilog – we must specify input, output signals, specify functionalities of the module that are part of the system.

- Each module declaration – includes a list of interface signals- can be used to connect to other modules, or to the outside world.

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

Internals of the module

Components of a Verilog module



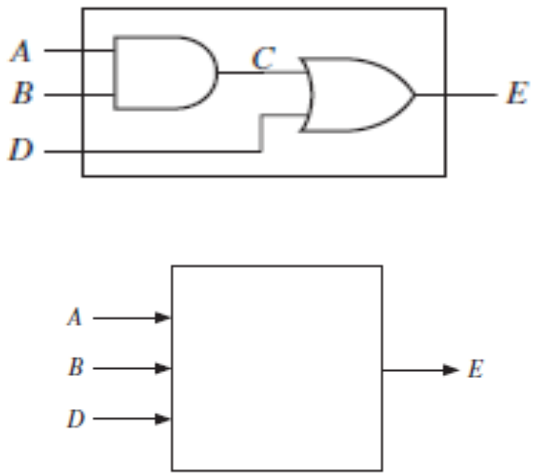
Internals of the module

- Five components within a module
- Components can be any order – any place in the module definitions.
- Multiple modules can be defined in a single file.
- Modules can be of any order in the file.

Concept of a module

- Module name – identifier for the module.
- Module terminal list – input and output terminals of the module.

  General structure of Verilog modules with two gates




```
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
```

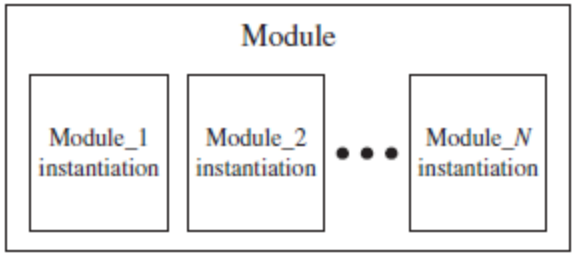
Concept of a module

```
module T_FF (q, clock, reset);
 .
 .
<functionality of T-flipflop>
 .
 .
endmodule
```

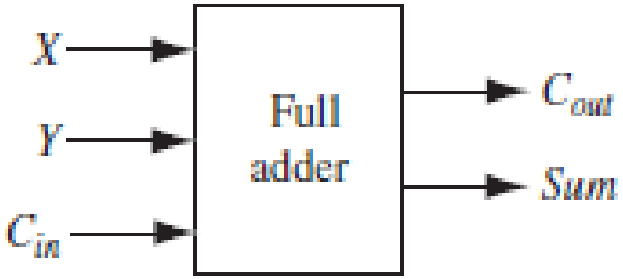
Concept of a module

- Illegal to nest modules - One module cannot contain another module within the module and end module statements.
- Module definitions – incorporate copies of other modules by instantiating them.
- Input port signals are of keyword input.
- Output port signals are of keyword output.
- Bidirectional signals are of keyword inout.

Verilog modules



Verilog program structure




```
Sum = X ⊕ Y ⊕ Cin
Cout = XY ⊕ YCin + XCin
```

Verilog module for full adder

- Verilog assignment statements for sum and carry – represent the logic equations for the full adder.

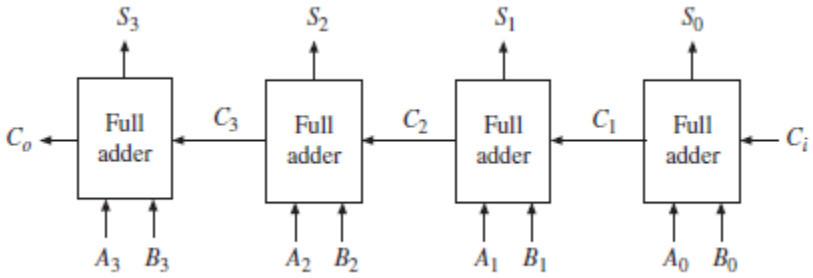
Verilog modules

```
module FullAdder(X, Y, Cin, Cout, Sum);
output Cout, Sum;

input X, Y, Cin;

  assign #10 Sum = X ^ Y ^ Cin;
  assign #10 Cout = (X && Y) || (X && Cin) || (X && Cin);
endmodule
```

Verilog module for full adder

Verilog modules

Other crcitectural descriptions – truth tables, interconnection of gates



4 bit binary adder

Four full adders connected to form a 4 bit binary adder.
3 bit internal carry signal C declared as data type wire.
Several instances of the full adder created.
Instantiate four copies of full adder.

Verilog modules

```
module Adder4 (S, Co, A, B, Ci);
output [3:0] S;
output Co;
input  [3:0] A, B;
input  Ci;

wire   [3:1] C;  // C is an internal signal

// instantiate four copies of the FullAdder

FullAdder FA0 (A[0], B[0], Ci,   C[1], S[0]);
FullAdder FA1 (A[1], B[1], C[1], C[2], S[1]);
FullAdder FA2 (A[2], B[2], C[2], C[3], S[2]);
FullAdder FA3 (A[3], B[3], C[3], Co,   S[3]);

endmodule
```

```
module FullAdder(X, Y, Cin, Cout, Sum);
output Cout, Sum;

input X, Y, Cin;

  assign #10 Sum = X ^ Y ^ Cin;
  assign #10 Cout = (X && Y) || (X && Cin) || (X && Cin
endmodule
```

Positional association

Structural description of 4 bit binary adder

Four full adders connected to form a 4 bit binary adder.
3 bit internal carry signal C declared as data type wire.
Several instances of the full adder created.
Instantiate four copies of full adder.

Hierarchical modeling concepts

Components of a simulation.

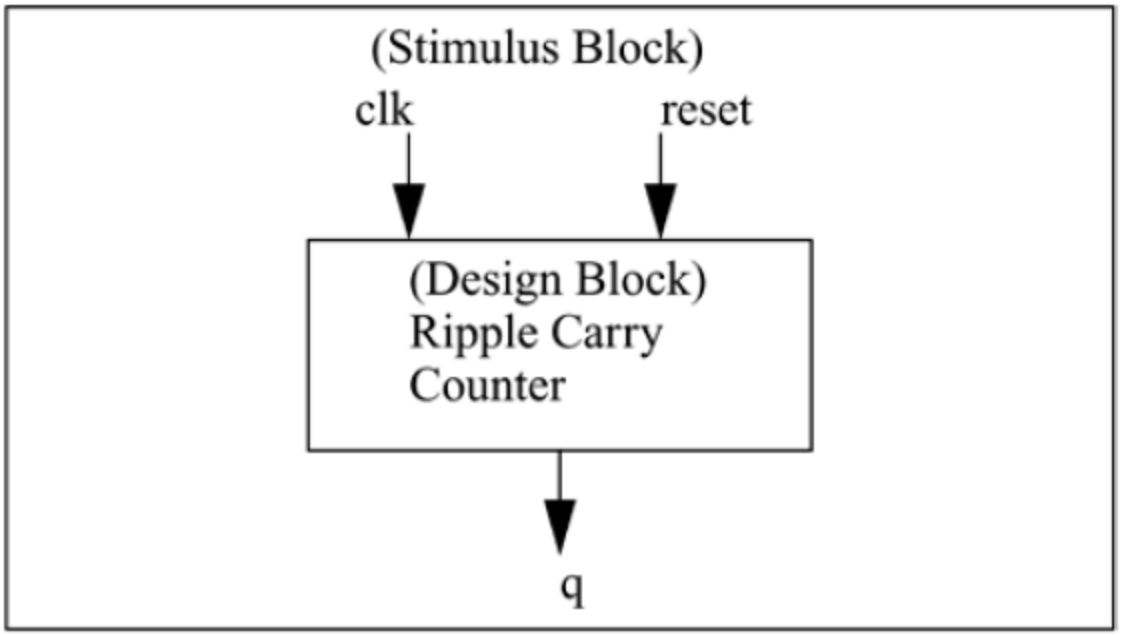
Test the functionality of the design block – applying stimulus and checking results.

Stimuls block – commonly called the test bench.

Stimulus and design block are kept seperate. – good practice in the design.

Hierarchical modeling concepts

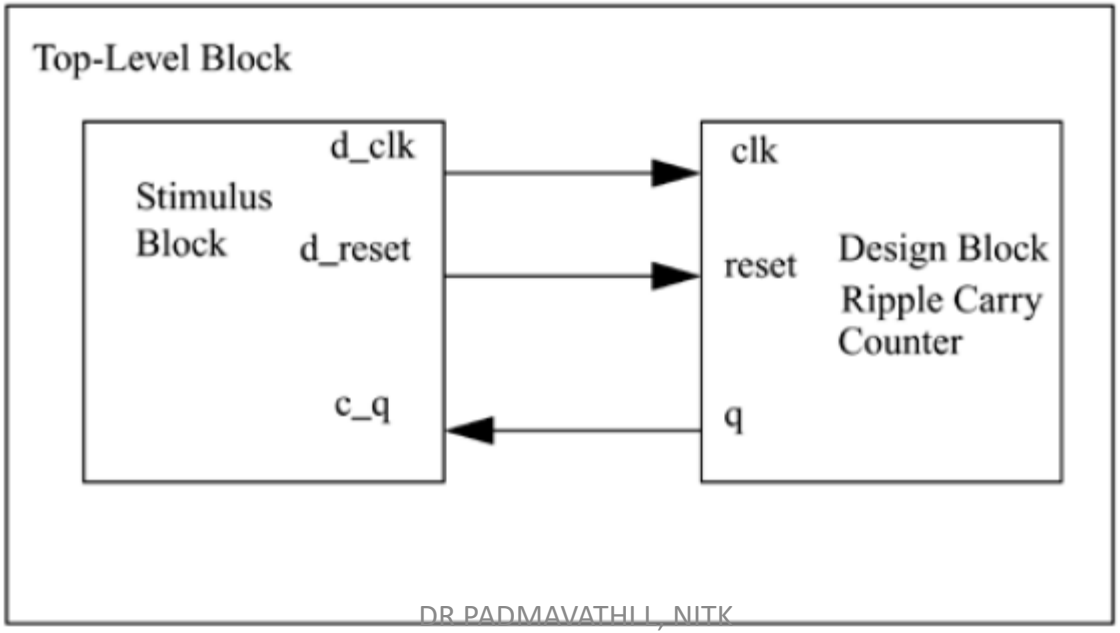
Stimulus block instantiating design block.



Hierarchical modeling concepts

Instantiating both the stimulus and design blocks in a top-level dummy module.

Stimulus block interacts with the design block through interface.
Top level block simply instantiates the design and stimulu block.



ports

Provide the interface by which a module can communicate with its environment.

Input/output pins of IC chip.

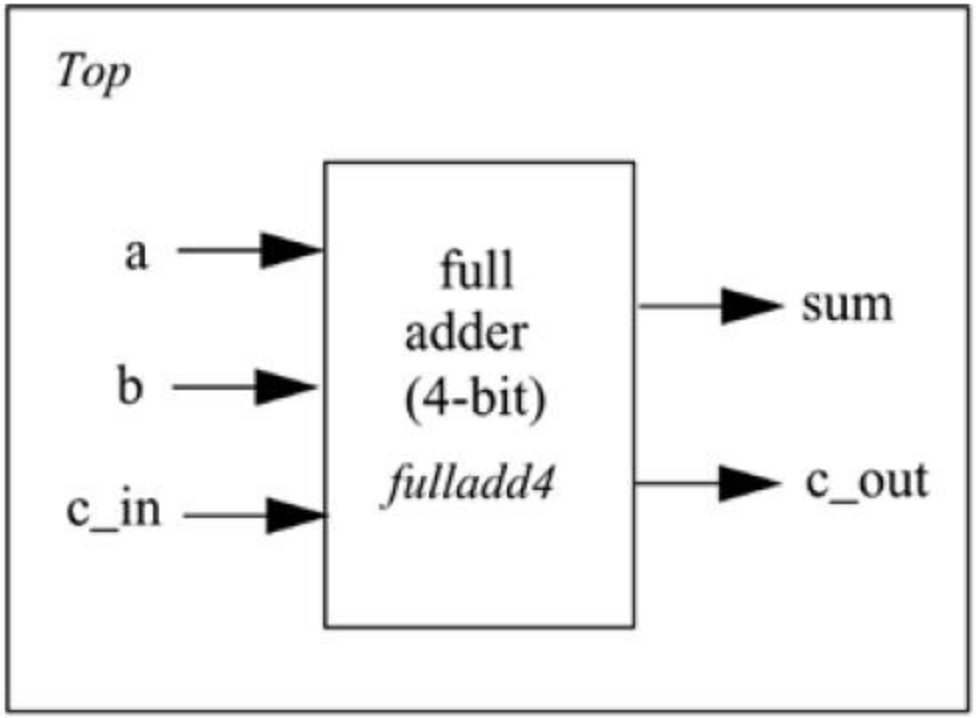
Interaction of environment with the module - only through ports.

Internals of the module – not visible to the environment – very powerful flexibility to the designer.

Ports – terminals.

ports

I/O ports of a full adder



ports

Port declaration

Port intended to be a wire – sufficient to decalre it as output, input , inout.

Output ports – hold their values – must be declared as reg.

Port declarations for D Flip-flop.

```
module DFF(q, d, clk, reset);
output q;
reg q; // Output port q holds value; therefore it is declared as reg.
input d, clk, reset;
...
...
endmodule
```