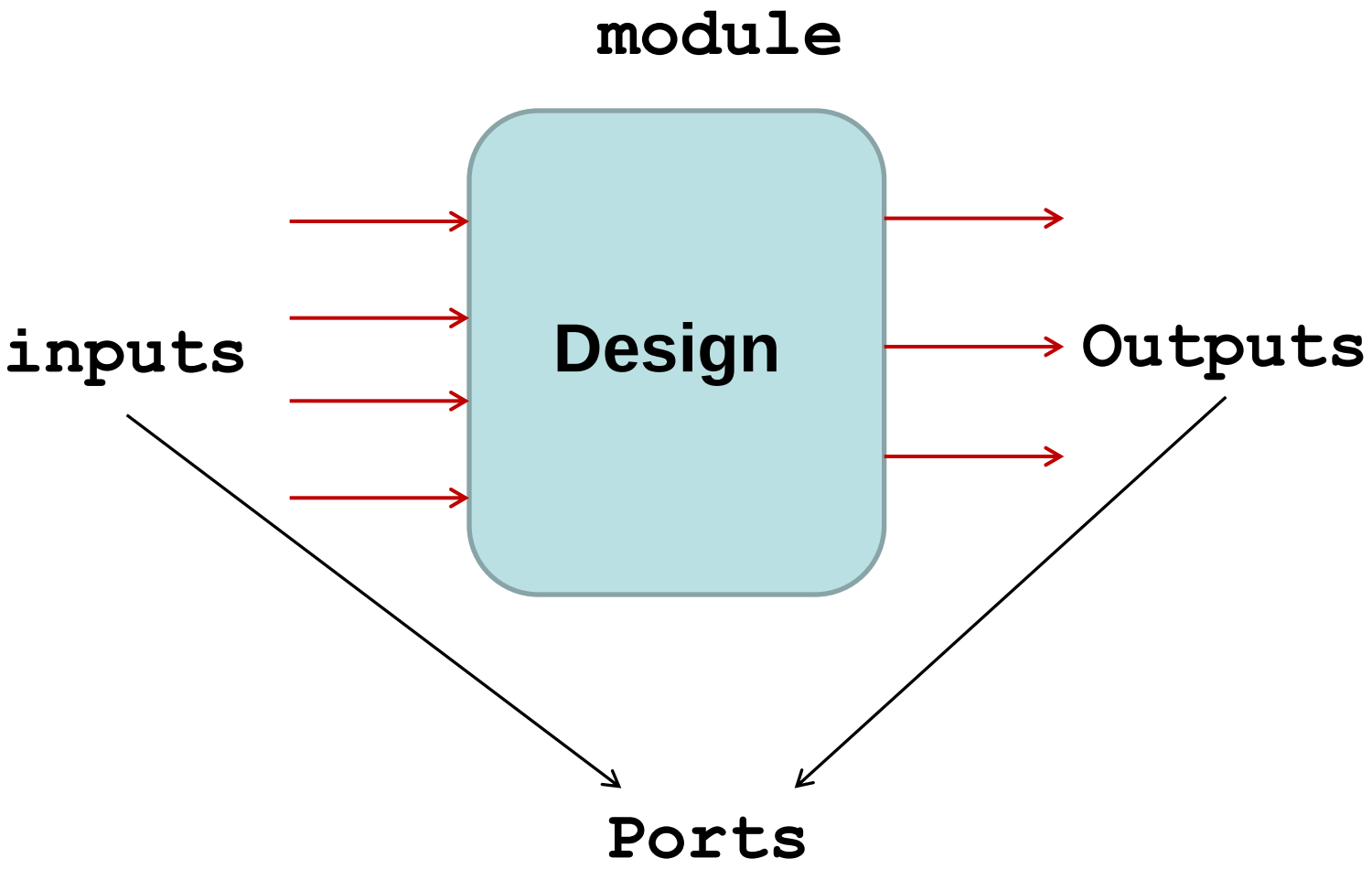
VerilogHDL

Module Structure



Basic Modeling structure

module *module_name(*port_list*);*

port declarations

data type declarations

circuit functionality

endmodule

- Begins with keyword **module** and ends with keyword **endmodule**
- Case –sensitive
- All keywords are lowercase
- Whitespace for readability
- Semicolon is the statement terminator
- // single line comment
- /*…*/ Multiline comment

Modules and Ports

```
module SR_latch(Q, Qbar, Sbar, Rbar);

output Q, Qbar;
input Sbar, Rbar;

...
...
...

endmodule
```

- Module name includes port list
- Port types
  - Input
  - Output
  - inout
- Port declarations

<port type> <port name>

First Exercise – Gate level

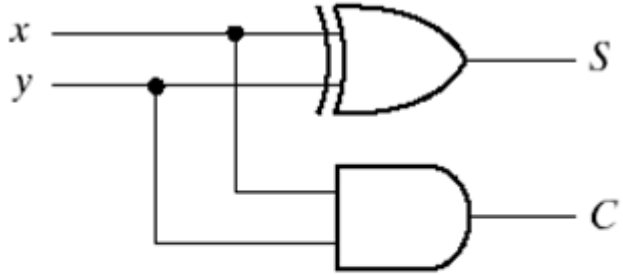
- Develop the Boolean function of output
- Draw the circuit with logic gates/primitives
- Connect gates/primitives with net (usually wire)
- write HDL description

Primitives

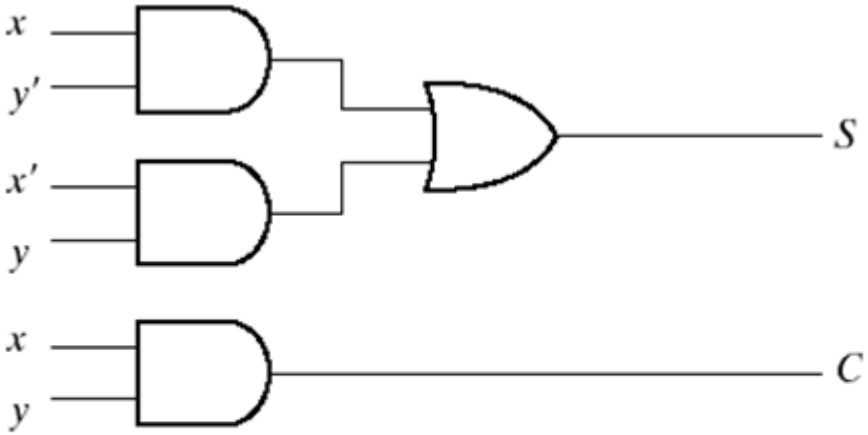
- Primitives are modules ready to be instanced
- Verilog build-in primitive gate
  - ***and, or, xor, nand, nor, xnor***
    - <prim_name>< inst_name>(out0, in0, in1,....);
  - ***not, buf***
    - <prim_name>< inst_name>(out0, out1, ..., in0);

Gate-level Modeling

- Examples



$S = x \oplus y$
$C = xy$



$S = xy' + x'y$
$C = xy$

```
module halfadder (S,C,x,y);
    input x,y;
    output S,C;
//Instantiate primitive gates
    xor (S,x,y);
    and (C,x,y);
endmodule
```

Data Types

- Value set
- Nets
- Registers
- Vectors
- Integer & Real
- Strings
- Arrays
- Memory

Value Set

- 0 – Logic zero, false condition
- 1 – Logic one, true condition
- x – Unknown logic value
- z – High Impedance, floating state

Strength Level	Type	Degree
supply	Driving	strongest
strong	Driving	
pull	riving	
large	Storage	
weak	Driving	
medium	Storage	
small	Storage	
highz	High Impedance	weakest

Vectors

- Multiple bit widths
- Nets or reg data types can be declared as vectors

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

input [3:0] a; // 4-bit signal
wire [7:0] bus; // 8-bit bus
```

- Vectors can be declared as

```
[high# : low#]

[low# : high#]
```

Gate Types

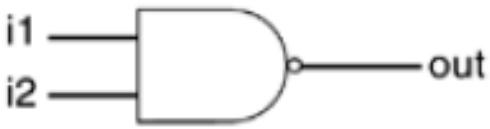
and

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

or

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

Gate Types



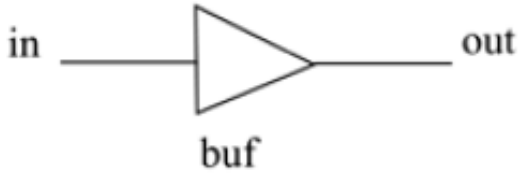
nand

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

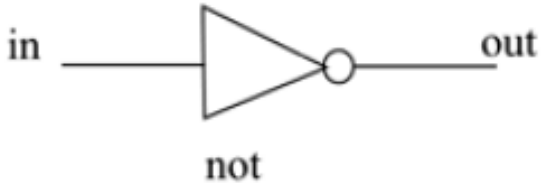
nor

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

Gate Types



buf	in	out
	0	0
	1	1
	x	x
	z	x



not	in	out
	0	1
	1	0
	x	x
	z	x

- Keywords: **`buf`** **`not`**

Gate Types

- **`bufif / notif`**

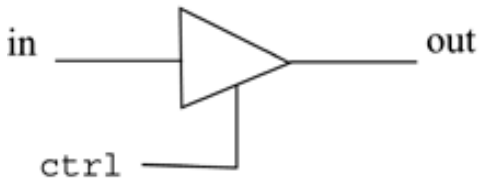
- additional control signals on **`buf`** and **`not`** gates

```
bufif1      notif1
bufif0      notif0
```

- gates propagate only if their control signal is asserted

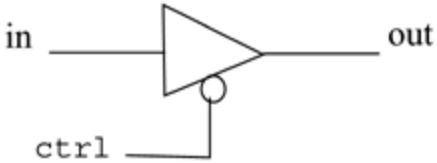
- propagate z if their control signal is deasserted

Gate Types



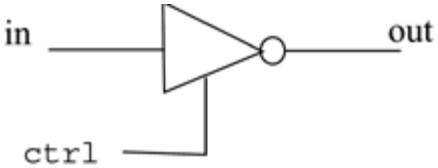
bufif1

bufif1 (in \ ctrl)	0	1	x	z
0	z	0	L	L
1	z	1	H	H
x	z	x	x	x
z	z	x	x	x



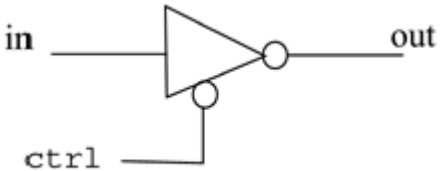
bufif0

bufif0 (in \ ctrl)	0	1	x	z
0	0	z	L	L
1	1	z	H	H
x	x	z	x	x
z	x	z	x	x



notif1

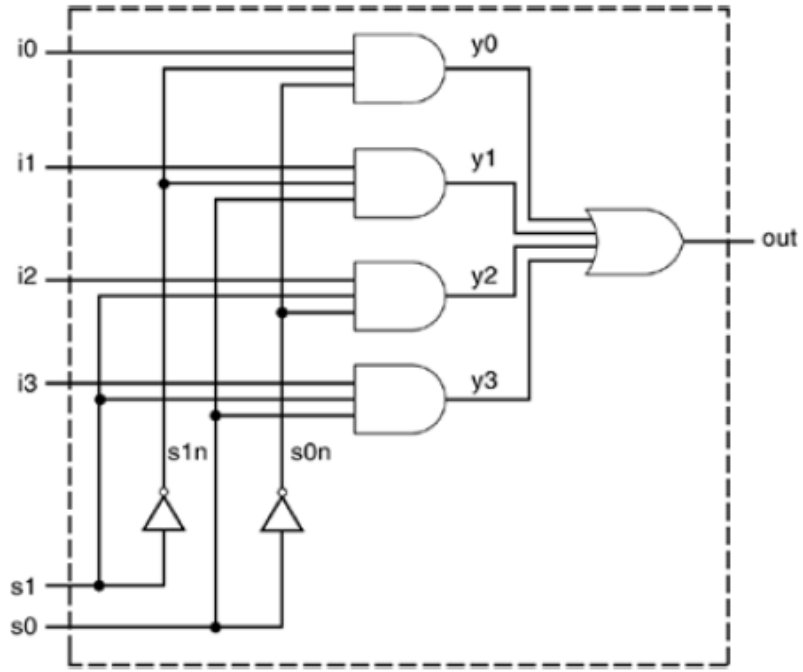
notif1 (in \ ctrl)	0	1	x	z
0	z	1	H	H
1	z	0	L	L
x	z	x	x	x
z	z	x	x	x

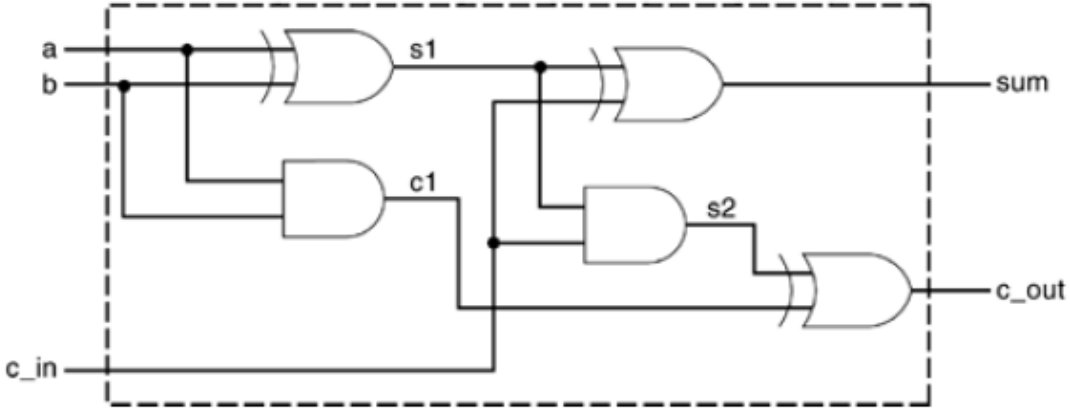


notif0

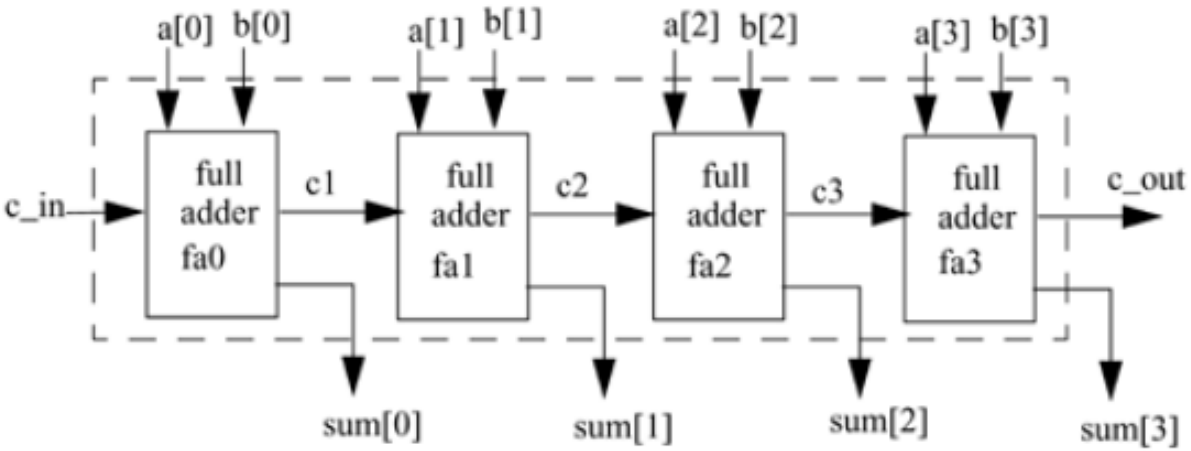
notif0 (in \ ctrl)	0	1	x	z
0	1	z	H	H
1	0	z	L	L
x	x	z	x	x
z	x	z	x	x

Examples





Examples


a[0]
b[0]
a[1]
b[1]
a[2]
b[2]
a[3]
b[3]
c_in
full adder fa0
c1
full adder fa1
c2
full adder fa2
c3
full adder fa3
c_out
sum[0]
sum[1]
sum[2]
sum[3]

Stimulus block

```
module stimulus;
// Declare variables to be connected to inputs
reg IN0, IN1, IN2, IN3;
reg S1, S0;
// Declare output wire
wire OUTPUT;
// Instantiate the multiplexer
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
// Stimulate the inputs. Define the stimulus module(no ports)
initial
begin
// set input lines
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
#100 S1 = 0; S0 = 0;
#100 S1 = 0; S0 = 1;
#100 S1 = 1; S0 = 0;
#100 S1 = 1; S0 = 1;
end
endmodule
```

Dataflow Modeling