Verilog 2

Hsi-Pin Ma 馬席彬

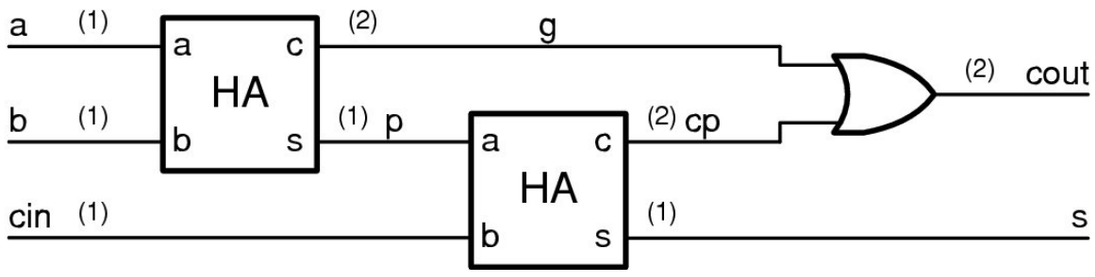
http://lms.nthu.edu.tw/course/40757

Department of Electrical Engineering

National Tsing Hua University

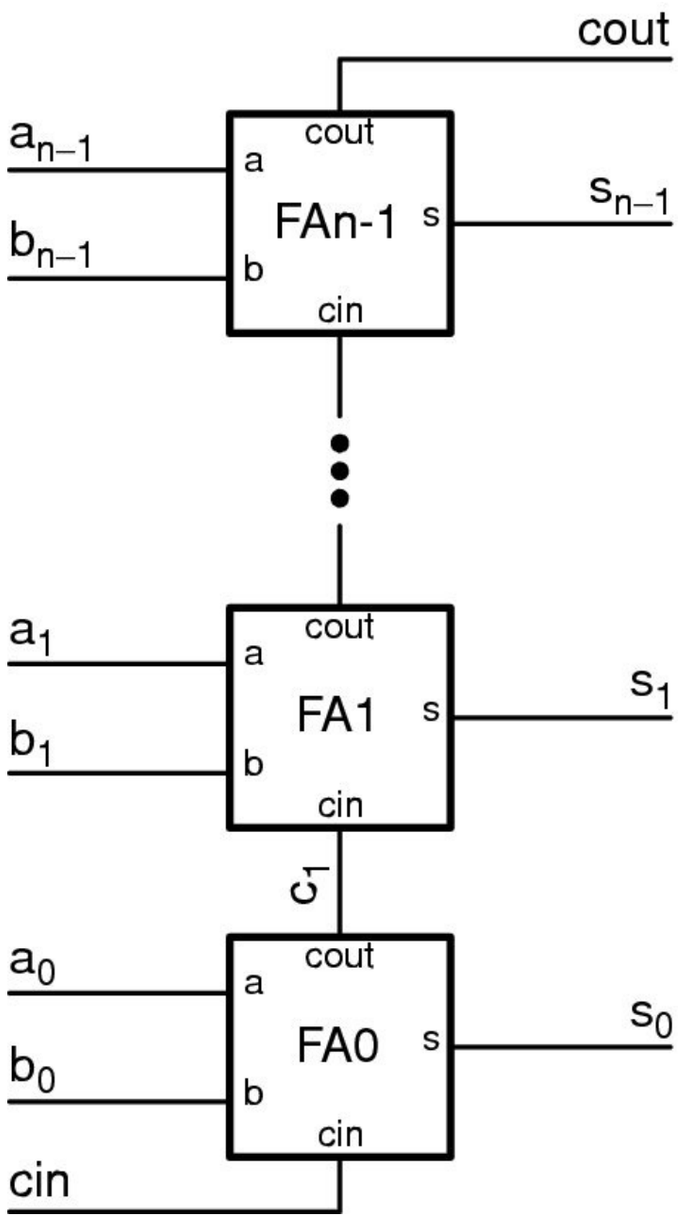
Verilog Description

```
module HalfAdder(a, b, c, s);
input a, b;
output c, s; // carry and sum

assign s = a ^ b;
assign c = a & b;

endmodule
```




```
module FullAdder(a, b, cin, cout, s);
input a, b, cin;
output cout, s; // carry and sum
wire g, p; // generate and propagate
wire cp;

HalfAdder ha1(.a(a), .b(b), .c(g), .s(p));
HalfAdder ha2(.a(cin), .b(p), .c(cp), .s(s));
assign cout = g | cp;

endmodule
```

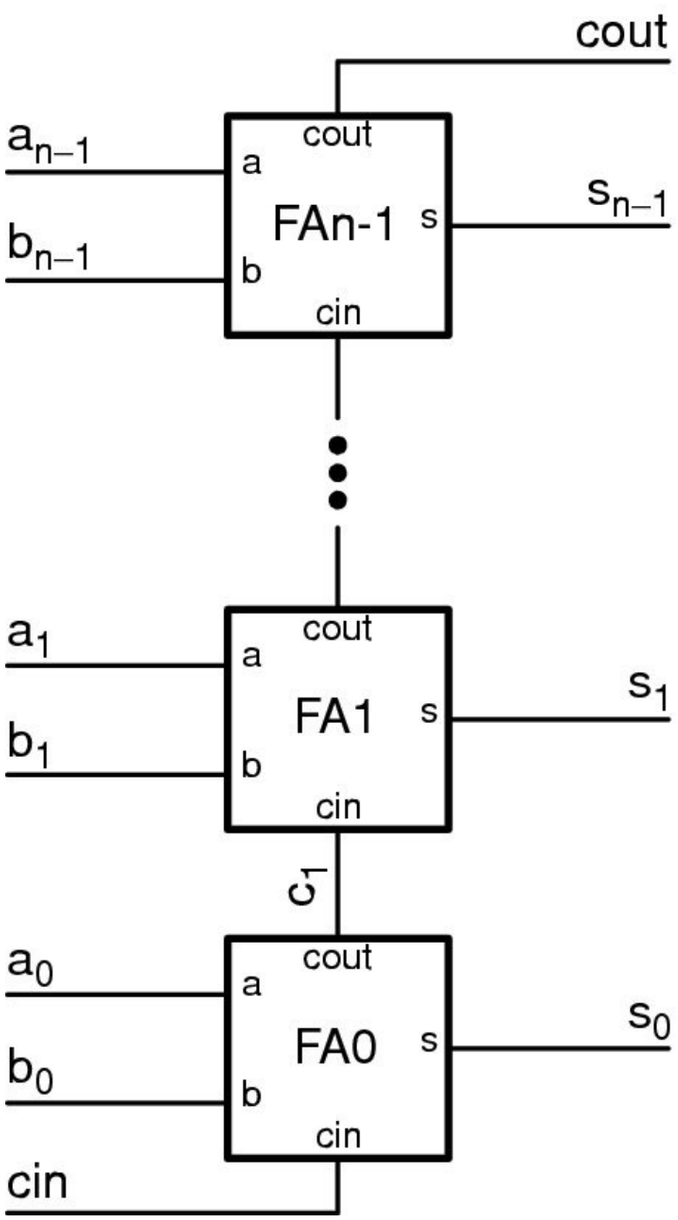
Ripple-Carry Adder (4/4)




```
module adder(a, b, cin, cout, s);
parameter n=4;
input [n-1:0] a, b;
input cin;
output reg [n-1:0] s;
output cout;
reg [n-1:0] c;
integer i;

assign cout = c[4];
always @*
  c[0] = cin;
always @*
  for (i=0;i<4;i=i+1)
    {c[i+1],s[i]} = a[i] + b[i] + c[i];

endmodule
```

```
assign {c[1],s[0]} = a[0] + b[0] + c[0];
assign {c[2],s[1]} = a[1] + b[1] + c[1];
assign {c[3],s[2]} = a[2] + b[2] + c[2];
assign {c[4],s[3]} = a[3] + b[3] + c[3];
```

Multi-bit Binary Adder

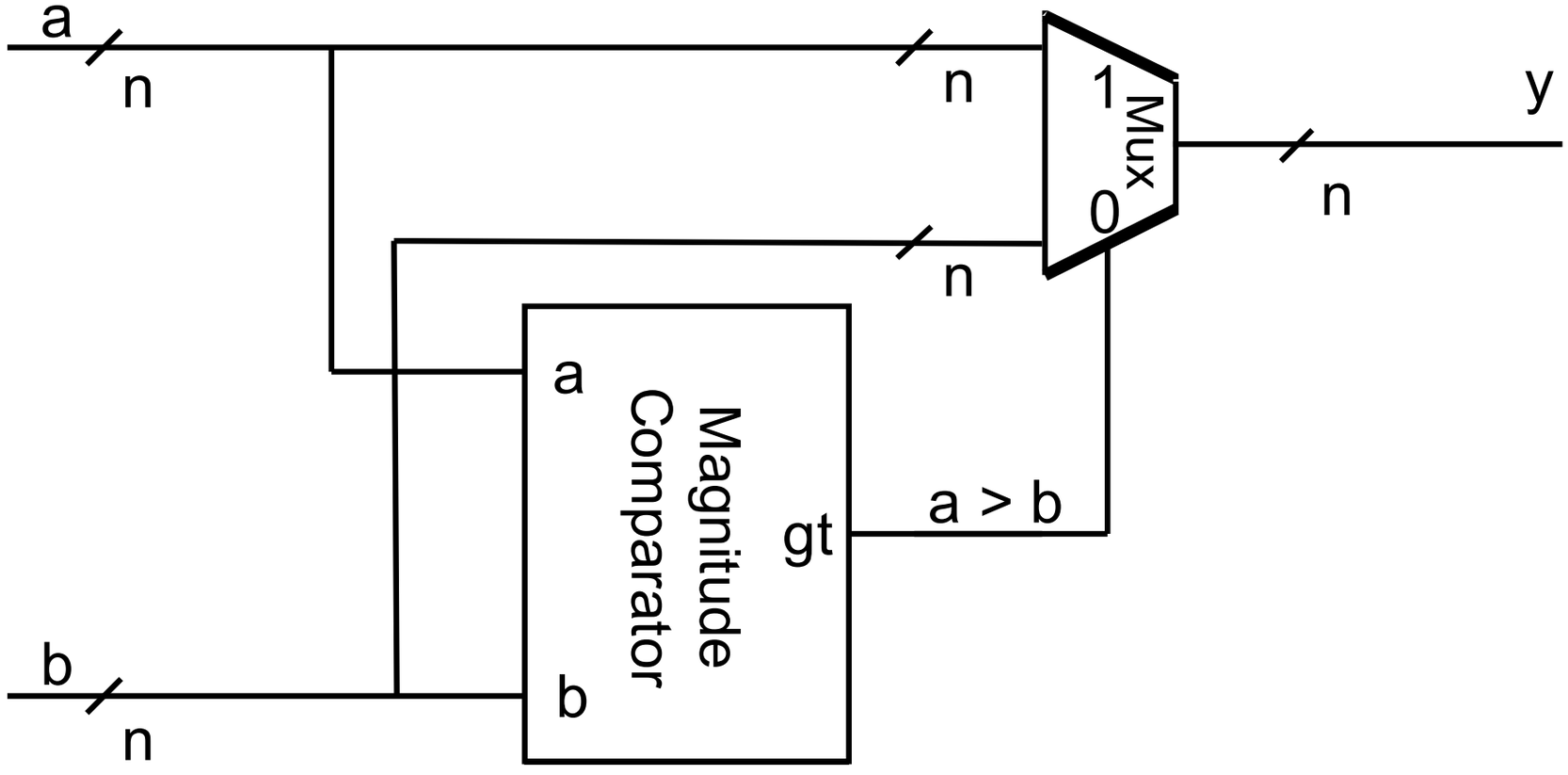


behavioral model

```
module Adder1(a, b, cin, cout, s);
parameter n=8;
input [n-1:0] a, b;
input cin;
output [n-1:0] s;
output cout;

assign {cout, s} = a + b + cin;

endmodule
```

Maximun Unit

$$y = \max\{a, b\}$$



Verilog Module Construction

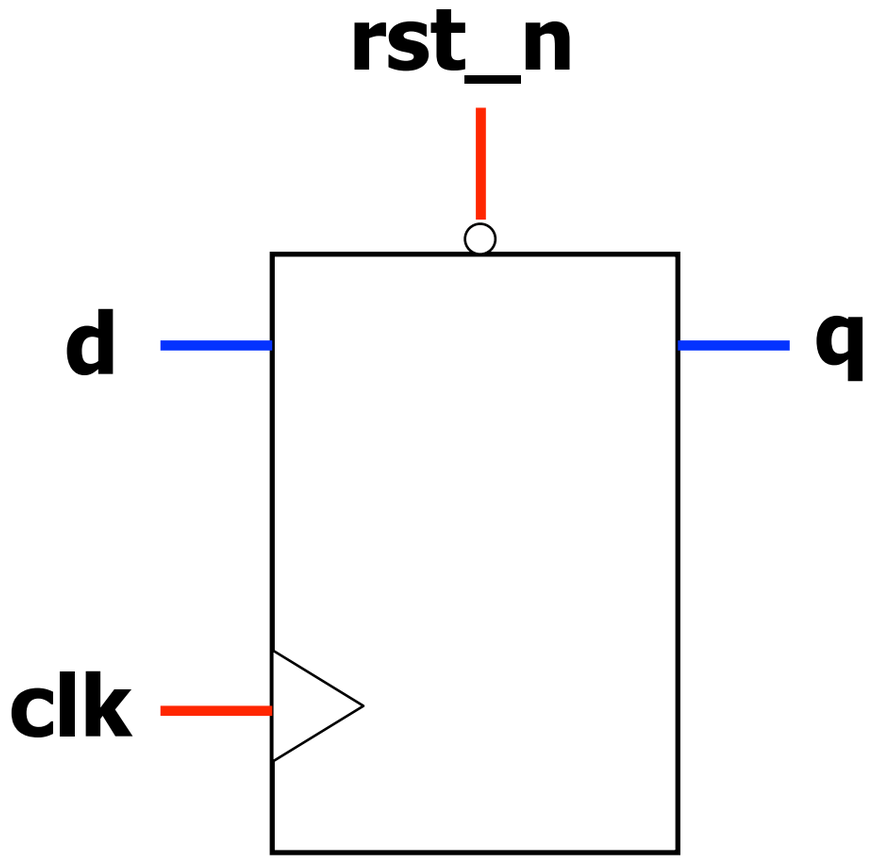
- Separate flip-flops with other logics (two types)
  - flip-flops (edge-triggered with clock, reset)
  - combinational logics (level sensitive)
- Combinational logics
  - simple logics (AND, OR, NOT)
  - coder/decoder (mapping, addressing)
  - comparison (conditional/equality test)
  - selection (select correct results, MUX)
  - arithmetic functions and superposition (+,-,*,binary shift)
- Finite state machine (FSM)

Magnitude Comparator

```
module MagComp_b(a, b, gt);
parameter k=8;
input [k-1:0] a, b;
output gt;

assign gt = (a > b) ? 1 : 0;

endmodule
```

D-type Flip Flop

```
module dff(
  q,  // output
  d,  // input
  clk,  // global clock
  rst_n // active low reset
);

output q;  // output
input d;  // input
input clk;  // global clock
input rst_n;  // active low reset

reg q; // output (in always block)

always @(posedge clk or negedge rst_n)
  if (~rst_n)
    q<=0;
  else
    q<=d;

endmodule
```



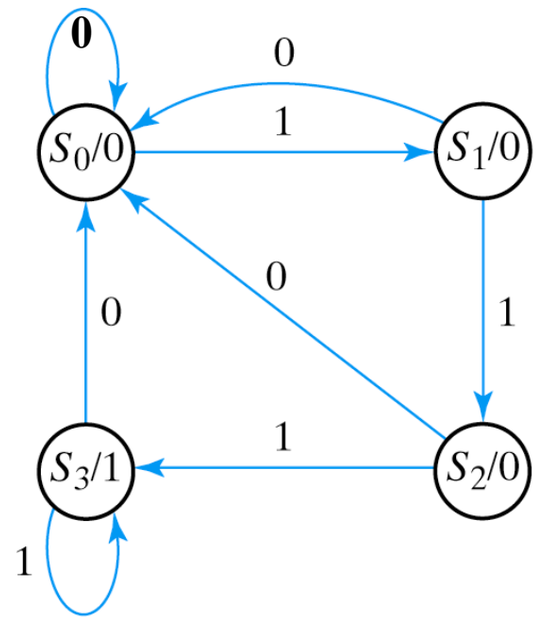
Style Guide: FSMs in Verilog

- All feedback through explicitly instantiated DFF module
- Next state and output functions are combinational - case (if/else) or assign
  - No "inferred latches"
- Make sure you can reset your FSM
- Use defines for state encoding (and width)

Verilog Design of FSM Ex1

```
// Gray code state assignment
`define SWIDTH 2
`define S0 2'b00
`define S1 2'b01
`define S2 2'b11
`define S3 2'b10

// define output codes
`define DETECTED 1'b1
`define NOT_DETECTED 1'b0

module Ex2(clk, rst, in, out);
input clk;
input rst;
input in;
output reg out;

reg [`SWIDTH-1:0] state;
reg [`SWIDTH-1:0] next_state;
```

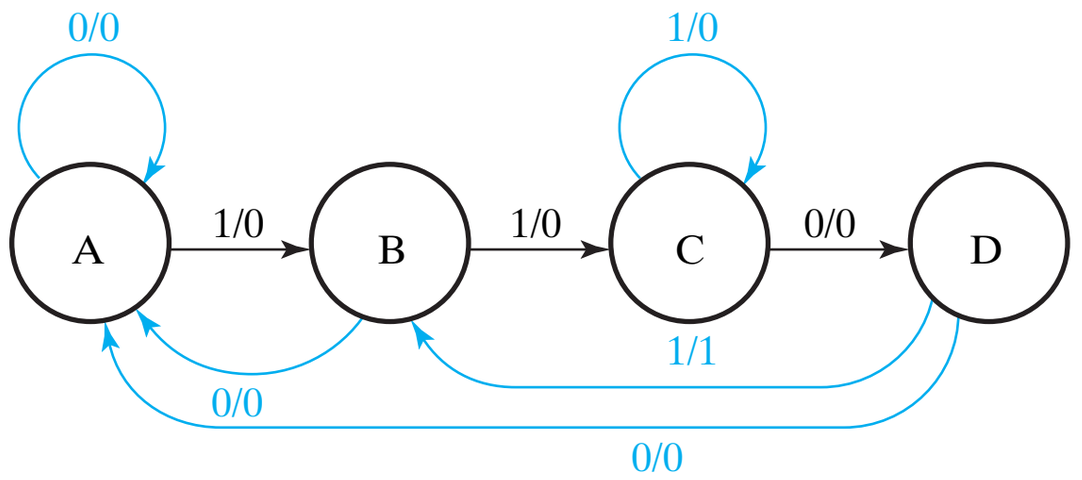
```
// low active asynchronous reset
always @(posedge clk or negedge rst_n)
 if (~rst_n)
   state <= `S0;
 else
   state <= next_state;

// state and output eqs - combinational logic
always @*
 case (state)
   `S0:  {next_state,out} = {(in?`S1:`S0), `NOT_DETECTED};
   `S1:  {next_state,out} = {(in?`S2:`S0), `NOT_DETECTED};
   `S2:  {next_state,out} = {(in?`S3:`S0), `NOT_DETECTED};
   `S3:  {next_state,out} = {(in?`S3:`S0), `DETECTED};
   default: {next_state,out} = {`SWIDTH+1{1'bx}};
 endcase

endmodule
```



```
module test_ex2;
reg clk;
reg rst_n;
reg in;
wire out;

Ex2   U0(.clk(clk), .rst(rst_n), .in(in), .out(out));

always
 #5 clk = ~clk;        // generate periodic clock

initial
begin
 clk=0;rst_n=0;in=0; //set initial input, let FF reset
 #10 rst_n=1; // disable reset
 #10 in=1;
 #10 in=0;
 #10 in=1;
 #10 in=1;
 …
end

endmodule
```

Verilog Design of FSM Ex2

```
// Gray code state assignment
`define SWIDTH 2
`define A 2'b00
`define B 2'b01
`define C 2'b11
`define D 2'b10

// define output codes
`define DETECTED 1'b1
`define NOT_DETECTED 1'b0

module Ex3(clk, rst_n, in, out);
input clk;
input rst_n;
input in;
output reg out;

reg [`SWIDTH-1:0] state;
reg [`SWIDTH-1:0] next_state;
```



Verilog Design of FSM Ex2

```
// low active asynchronous reset
always @(posedge clk or negedge rst_n)
 if (~rst_n)
   state <= `S0;
 else
   state <= next_state;

// state and output eqs - combinational logic
always @*
 case (state)
   `S0:  {next_state,out} = {(in?`B:`A), (in?`NOT_DETECTED:`NOT_DETECTED)};
   `S1:  {next_state,out} = {(in?`C:`A), (in?`NOT_DETECTED:`NOT_DETECTED)};
   `S2:  {next_state,out} = {(in?`C:`D), (in?`NOT_DETECTED:`NOT_DETECTED)};
   `S3:  {next_state,out} = {(in?`B:`A), (in?`DETECTED:`NOT_DETECTED)};
   default: {next_state,out} = {`SWIDTH+1{1'bx}};
 endcase

endmodule
```