Digital Devices

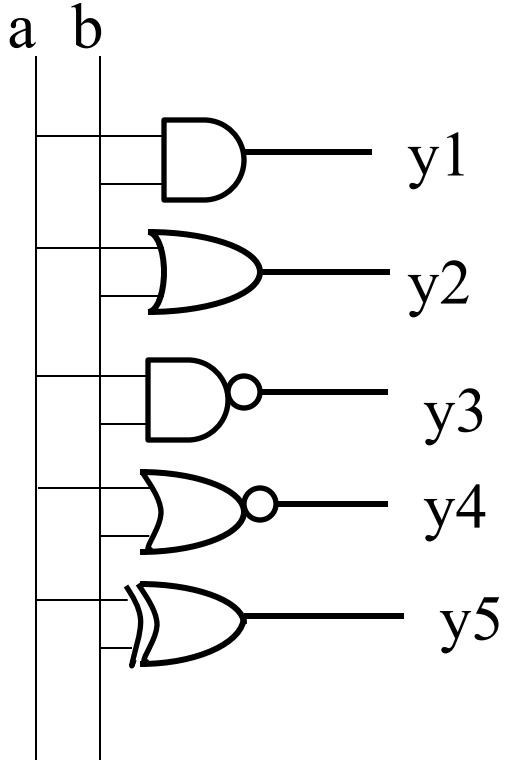
E155
9/16/2013

Outline

- Combinational Logic
  - 1-bit full adder
  - 32-bit adder
  - Priority decoder
  - Days of month
- Sequential Logic
  - Counter
  - Shift adder
  - RAM

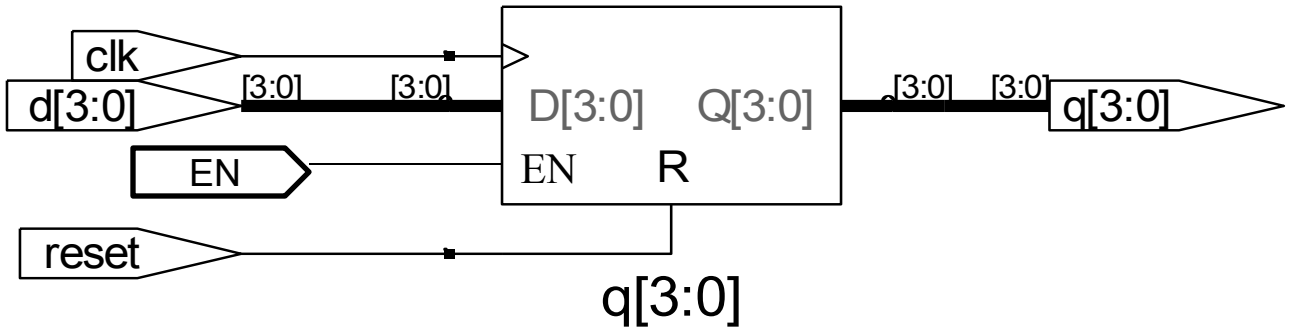
Combinational logic

```
module gates ( input logic [3:0] a, b,
               output logic [3:0] y1, y2, y3, y4, y5);

  assign y1 = a &b;
  assign y2 = a | b;
  assign y3 = ~y1;
  assign y4 = ~y2;
  assign y5 = a ^ b;

endmodule
```



Flip Flop

```
module flopr(input  logic      clk,
             input  logic      reset, en,
             input  logic [3:0] d,
             output logic [3:0] q);

  // asynchronous reset
  always_ff @ (posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else if (en)  q <= d;

endmodule
```



Rules

- Use always_ff and nonblocking <= to model synchronous sequential logic.

```
always_ff @ (posedge clk)
q <= d; // nonblocking
```

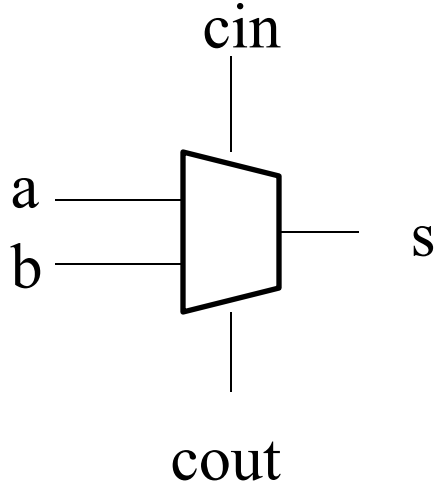
- Use assign to model simple combinational logic.

```
assign y = a & b;
```

- Use always_comb and blocking = to model complicated combinational logic.

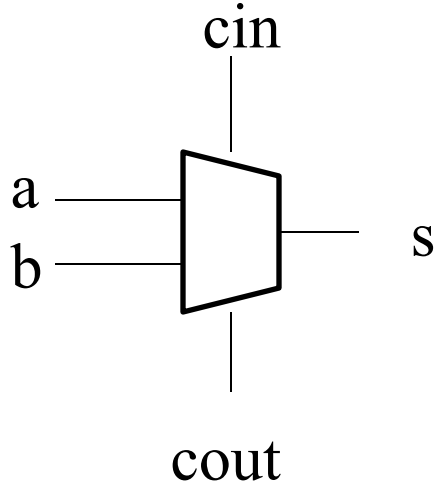
1-bit Full Adder

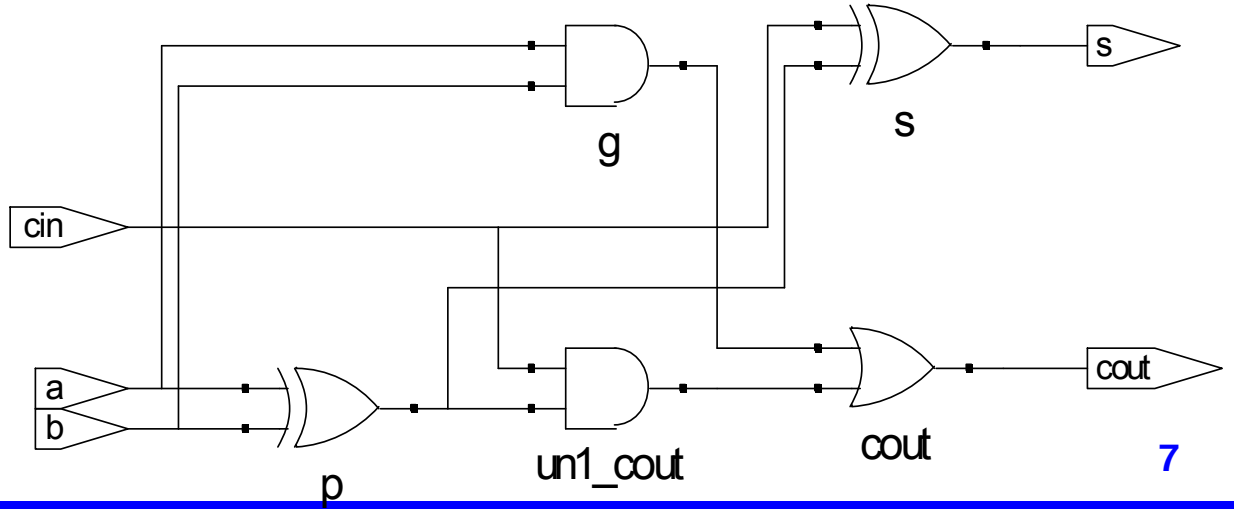
```
module fulladder(input logic  a, b, cin,
                 output logic s, cout);



endmodule
```



1-bit Full Adder

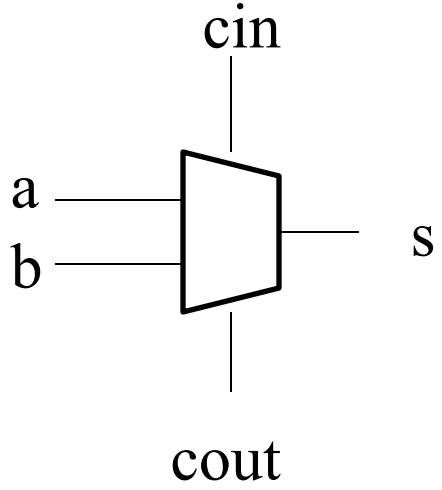
```
module fulladder(input logic  a, b, cin,
                 output logic s, cout);
  logic p, g;         // internal nodes
  assign p = a ^ b;
  assign g = a & b;
  assign s = p ^ cin;
  assign cout = g | (p & cin);

endmodule
```

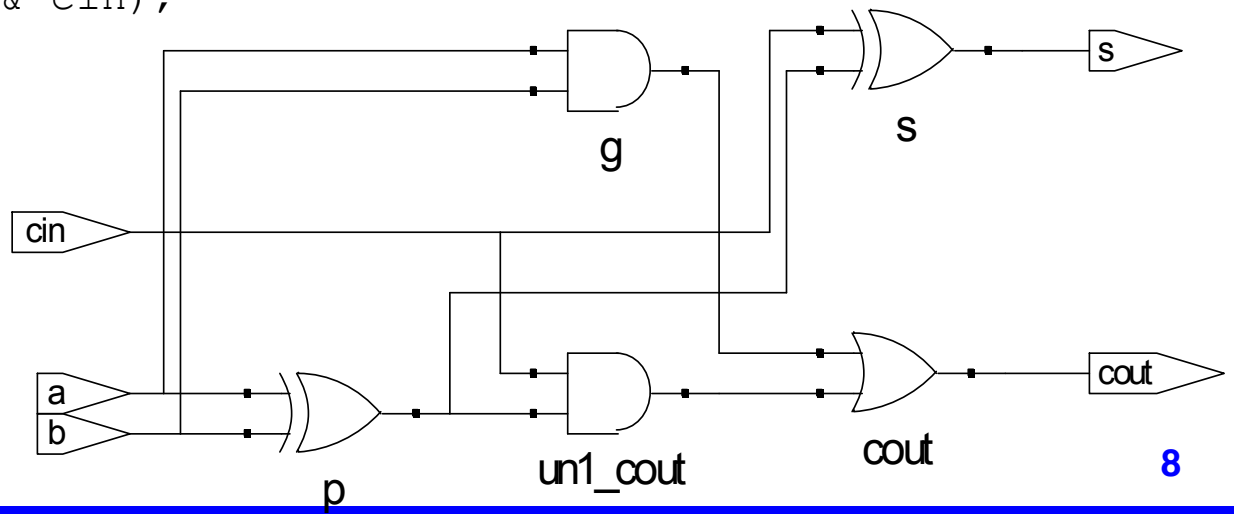




1-bit Full Adder

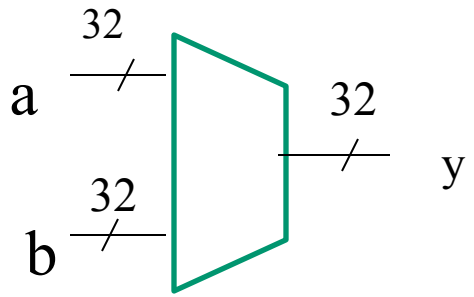
```
module fulladder(input logic  a, b, cin,
                 output logic s, cout);
  logic p, g;         // internal nodes
always_comb // equivalent to always@(a,b,cin)
 begin
  p = a ^ b;
  g = a & b;
  s = p ^ cin;
  cout = g | (p & cin);
 end
endmodule
```





32-bit Adder

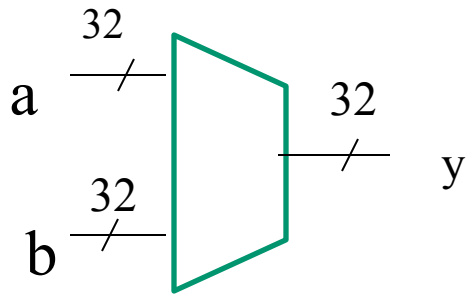
```
Module addr   ( input logic [31:0] a, b,
                output logic [31:0] y);
```




```
endmodule
```

32-bit Adder

```
Module addr   ( input logic [31:0] a, b,
                output logic [31:0] y);
        assign y= a+b;
endmodule
```



Priority Decoder

```
module priority (input  logic   [3:0] a,
                 output logic   [3:0] y);

  always_comb

endmodule
```

```
4'b1???: y = 4'b1000;
4'b01??: y = 4'b0100;
4'b001?: y = 4'b0010;
4'b0001: y = 4'b0001;
```

Priority Decoder

```
module priority (input  logic   [3:0] a,
                 output logic   [3:0] y);
  always_comb
    if (a[3]) y = 4'b1000;
    else if (a[2] y = 4'b0100;
    else if (a[1]) y = 4'b0010;
    else if (a[0]) y = 4'b0001;
    else y = 4'b0000;
endmodule
```

```
4'b1???: y = 4'b1000;
4'b01??: y = 4'b0100;
4'b001?: y = 4'b0010;
4'b0001: y = 4'b0001;
```

Priority Decoder

```
module priority (input  logic   [3:0] a,
                 output logic   [3:0] y);
  always_comb
    casez (a)
                                        4'b1???: y = 4'b1000;
                                        4'b01??: y = 4'b0100;
                                        4'b001?: y = 4'b0010;
                                        4'b0001: y = 4'b0001;

endcase
endmodule
```

Priority Decoder

```
module priority (input  logic   [3:0] a,
                 output logic   [3:0] y);

always_comb
    casez (a)
       4'b1???: y = 4'b1000;
       4'b01??: y = 4'b0100;
       4'b001?: y = 4'b0010;
       4'b0001: y = 4'b0001;
       default: y = 4'b0000;
    endcase
endmodule
```

```
4'b1???: y = 4'b1000;
4'b01??: y = 4'b0100;
4'b001?: y = 4'b0010;
4'b0001: y = 4'b0001;
```

Months with 31 Days

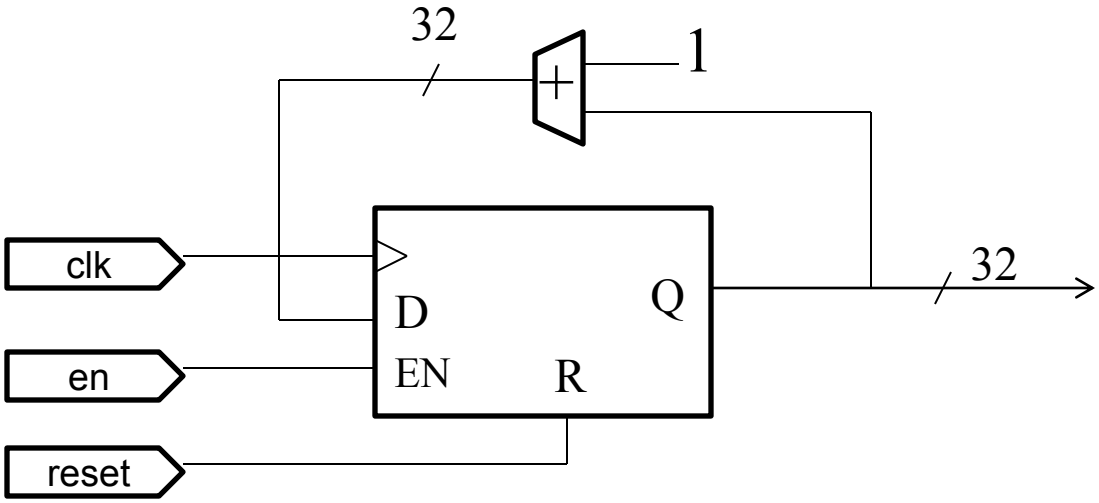
```
module month31days (input  logic   [3:0] a,
                    output logic   y);
always_comb
    case (a)









   endcase
endmodule
```

Months with 31 Days

```
module month31days (input  logic   [3:0] a,
                    output logic   y);
always_comb
    case (a)
       4'b0001: y = 1;
       4'b0011: y = 1;
       4'b0101: y = 1;
       4'b0111: y = 1;
       4'b1000: y = 1;
       4'b1010: y = 1;
       4'b1100: y = 1;
      default:  y = 0;
    endcase
endmodule
```

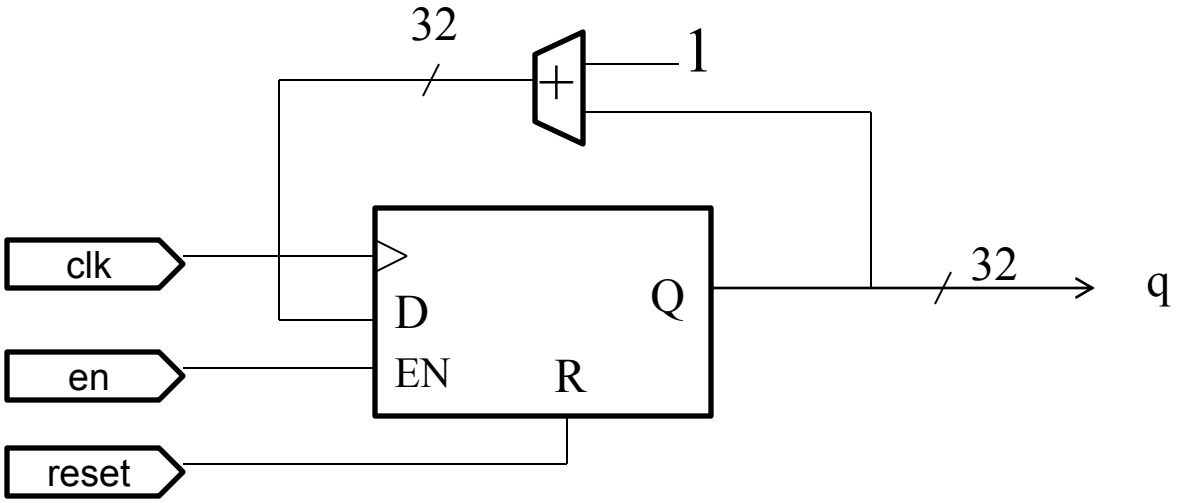
Counter

```
module flopr(input  logic      clk,
             input  logic      reset, en,
             output logic [3:0] q);

  // asynchronous reset

endmodule
```



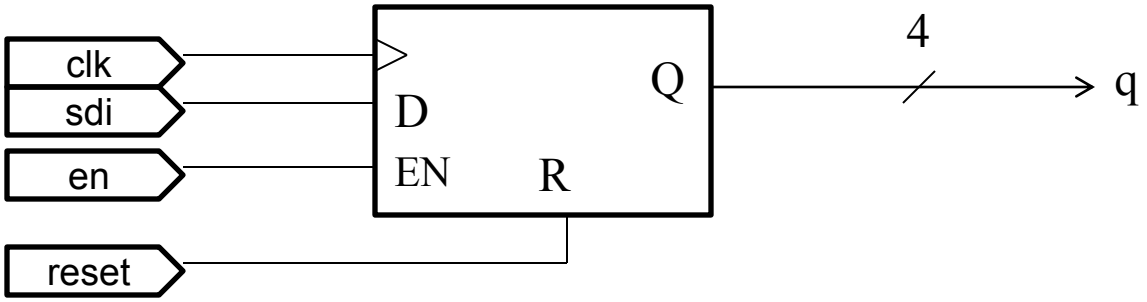
Counter

```
module flopr(input  logic      clk,
             input  logic      reset, en,
             output logic [3:0] q);

  // asynchronous reset
  always_ff @ (posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else if (en)  q <= q+1;

endmodule
```



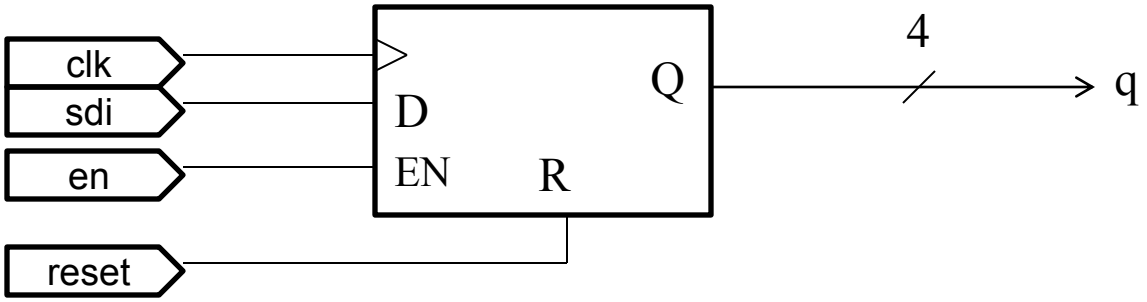
Shift Register

```
module flopr(input  logic      clk,
             input  logic      reset, en, sdi
             output logic [3:0] q);

  // asynchronous reset




endmodule
```



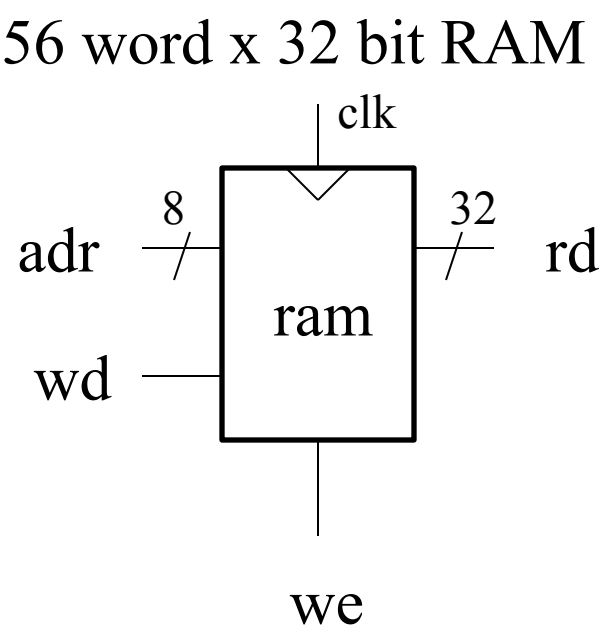
Shift Register

```
module flopr(input  logic      clk,
             input  logic      reset, en, sdi
             output logic [3:0] q);

  // asynchronous reset
  always_ff @ (posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else if (en)  q = {sdi, q[3:1]};

endmodule
```



RAM

```
module ram (input logic clk,
            input logic we,
            input logic [7:0] adr,
            input logic [31:0] wd,
            output logic [31:0] rd;

endmodule
```



RAM

```
module ram (input logic clk,
            input logic we,
            input logic [7:0] adr,
            input logic [31:0] wd,
            output logic [31:0] rd;
logic [31:0] mem[255:0];
always_ff @ @ posedge (clk)
   if (we) men[adr] <= wd;
   assign rd = mem[adr];//note block assignment used
                        //so that rd is the updated memory
                       // data including after a write
endmodule
```