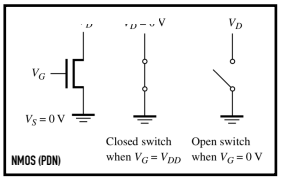
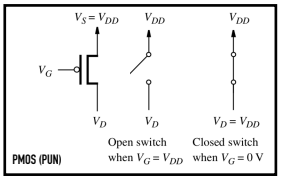
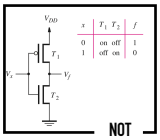
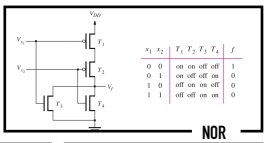
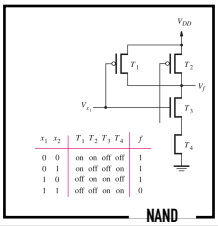
TRANSISTROS

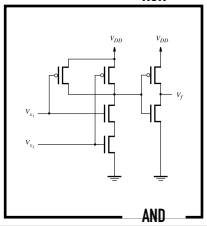




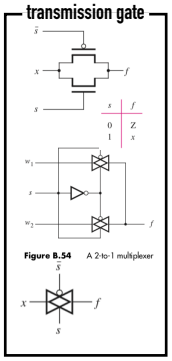








transmission gate



ENCODER

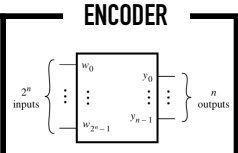
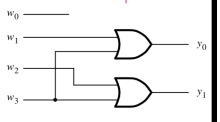
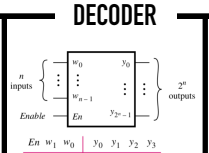


Figure 4.18 A 2^n-to-n binary encoder.

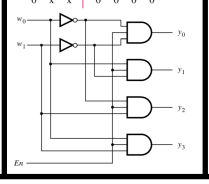
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



DECODER



En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0



SRAM

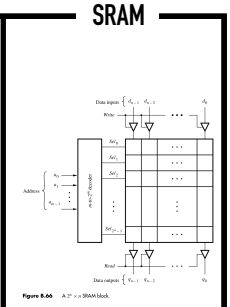


Figure 8.66 A $2^n \times s$ SRAM block

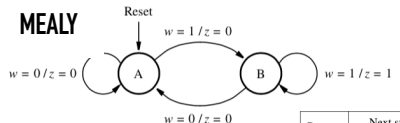
ONE HOT

	Present state	Next state $w=0$	Next state $w=1$	Output z
	$y_3y_2y_1$	$Y_3Y_2Y_1$	$Y_3Y_2Y_1$	
A	001	001	010	0
B	010	001	100	0
C	100	001	100	1

$Y_1 = \overline{w}$ $\quad Y_3 = w\overline{y}_1$

$Y_2 = wy_1$ $\quad z = y_3$

MEALY



Present state	Next state $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
A	A	B	0	0
B	A	B	0	1

Figure 6.24 State table for the FSM in Figure 6.23.

Present state	Next state $w=0$	Next state $w=1$	Output $w=0$	Output $w=1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

Figure 6.25 State-assigned table for the FSM in Figure 6.24.

TIMING ANALYSIS

$T_{min} = 1/F_{max}$

$T_{min} = t_{cQ} + 3(t_{AND}) + t_{XOR} + t_{su} = 6.4ns$

$F_{max} = 1/6.4$ ns $= 156.25$ MHz

$t_{cQ} + t_{XOR} = 0.8 + 1.2 = 2.0$ ns. Since 2.0 ns $> t_h = 0.4$ ns

there are no hold time violations.

WITH CLOCK SKEW:

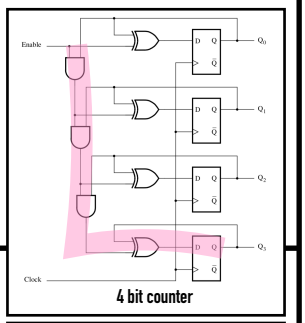
$t_{cQ} + 3(t_{AND}) + t_{XOR} + t_{su} - t_{skew} = 6.4 - 1.5$ ns $= 4.9$ ns

There is now a different critical path through the circuit

$T_{min} = t_{cQ} + 2(t_{AND}) + t_{XOR} + t_{su} = 5.2$ ns

$T_{min} = t_{cQ} + t_L + t_{su} - t_{skew}$

$t_{cQ} + t_L < t_h + t_{skew}$



COUNTERS

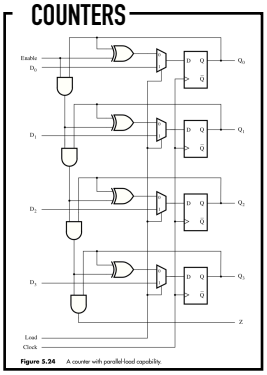
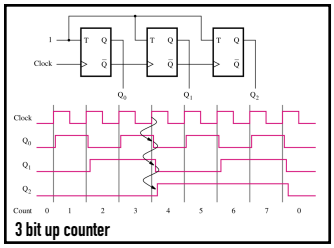


Figure 5.24 A counter with parallel-load capability.



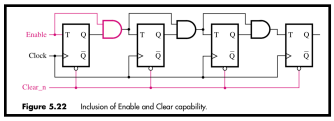
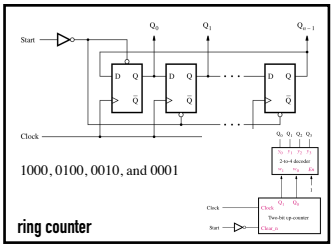
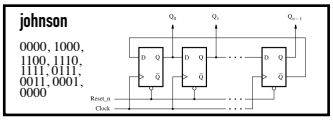
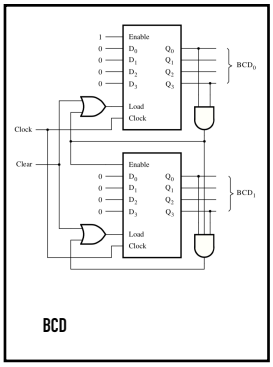


Figure 5.22 Inclusion of Enable and Clear capability.








```
module shiftrne (R, L, E, w, Clock, Q);
   parameter n = 4;
   input [n – 1:0] R;
   input L, E, w, Clock;
   output reg [n – 1:0] Q;
   integer k;

   always @(posedge Clock)
   begin
      if (L)
         Q <= R;
      else if (E)
      begin
         Q[n – 1] <= w;
         for (k = n – 2; k >= 0; k = k – 1)
            Q[k] <= Q[k + 1];
      end
   end

endmodule
```

Figure 5.60 A left-to-right shift register

mux2to1

```
module mux2to1(x, y, s, m);
    input x; //select 0
    input y; //select 1
    input s; //select signal
    output m; //output

    //assign m = s & y | ~s & x;
    // OR
    assign m = s ? y : x;

endmodule
```

shift register

```
module ShiftReg(
     input [3:0] D,
     input Clock,
     Resetn,
     Loadn,
     output SerialOut);
     reg [3:0] Q;

     always @(posedge Clock)
          if  (!Resetn)
               Q <= 0;
           else if (!Loadn)
               Q <= D;
            else begin
                 Q[0] <= 1'b1;
                 Q[1] <= Q[0];
                 Q[2] <= Q[1];
                 Q[3] <= Q[2];
           end
     assign SerialOut = Q[3];
endmodule
```

4 bit register

```
module reg4bit (D, Clock, Resetb,
Enable, Q);
     input [3:0] D;
     input Clock, Resetb, Enable;
     output reg [3:0] Q;

          always @(posedge Clock)
               if (!Resetb)
                    Q <= 0;
               else if (Enable)
                    Q <= D;
endmodule
```

rate divider

```
module rateDivider(input clock, input [25:0]
upperBound, output reg enable, output reg [25:0]
counter);
  always @(posedge clock) begin
    if (counter === 26'bx)
      counter <= 26'b0;
    else if (counter == upperBound) begin
      enable= 1'b1;
      counter <= 26'b0;
    end
    else begin
      enable = 1'b0;
      counter <= counter + 1 ;
    end
  end
endmodule
```

function select

```
module FunctionSelect(input [3:0] X, Y, input
[2:0] Sel,
                      output reg [3:0] Fout);
   wire [3:0] w1, w2;
   ModA U1(.X(X), .Y(Y), .ModAout(w1));
   ModB U2(.X(X), .Y(Y), .ModBout(w2));
   always @(*)
     case (Sel)
       3'b000:  Fout = w1;
       3'b001:  Fout = ~X;
       3'b010:  Fout = {X[3:2],Y[1:0]};
       3'b011:  Fout = w2;
       3'b100:  Fout = ~(X & Y);
       default:  Fout = 3'b000;
     endcase // case (Sel)
endmodule // FunctionSelect
```

3 bit add

```
module adder(A, B, S, cin, cout);
     input [2:0] A, B;
     input cin;
     output [2:0] Sum;
     output cout;

     wire [1:0] f_cout;

     full_adder F0(cin, A[0], B[0], S[0], f_cout[0]);
     full_adder
F1(f_cout[0]), A[1]), B[1]), S[1]), f_cout[1]);
     full_adder
F2((f_cout[1]), (A[2]), (B[2]), (S[2]), (cout) );
 endmodule
```

```
module serial_adder(A, B, Reset, Clock, Sum);
   input [7:0] A, B;
   input Reset, Clock;
   output wire [7:0] Sum;
   reg [3:0] Count;
   reg s, y, Y;
   wire [7:0] QA, QB;
   wire Run;
   parameter G = 1'b0, H = 1'b1;

   shiftrne shift_A (A, Reset, 1'b1, 1'b0, Clock, QA);
   shiftrne shift_B (B, Reset, 1'b1, 1'b0, Clock, QB);
   shiftrne shift_Sum (8'b0, Reset, Run, s, Clock, Sum);

   // Adder FSM
   // Output and next state combinational circuit
   always @(QA, QB, y)
      case (y)
         G: begin
               s = QA[0] ^ QB[0];
               if (QA[0] & QB[0])  Y = H;
               else  Y = G;
            end
         H: begin
               s = QA[0] ~^ QB[0];
               if (~QA[0] & ~QB[0])  Y = G;
               else  Y = H;
            end
         default:  Y = G;
      endcase

   // Sequential block
   always @(posedge Clock)
      if (Reset)  y <= G;
      else  y <= Y;

   // Control the shifting process
   always @(posedge Clock)
      if (Reset)  Count = 8;
      else if (Run)  Count = Count – 1;
   assign Run = |Count;

endmodule
```

Figure 6.49 Verilog code for the serial ad

```
module simple (Clock, Resetn, w, z);
   input Clock, Resetn, w;
   output z;
   reg [2:1] y, Y;
   parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

   // Define the next state combinational circuit
   always @(w, y)
      case (y)
         A: if (w)   Y = B;
            else     Y = A;
         B: if (w)   Y = C;
            else     Y = A;
         C: if (w)   Y = C;
            else     Y = A;
         default:    Y = 2'bxx;
      endcase

   // Define the sequential block
   always @(negedge Resetn, posedge Clock)
      if (Resetn == 0)  y <= A;
      else  y <= Y;

   // Define output
   assign z = (y == C);

endmodule
```

Figure 6.29 Verilog code for the FSM in Figure 6.3.

.do

```
vlib work
vlog mux.v
vsim mux
log {/*}
add wave {/*}
#signal names need to be in {} brackets
force {SW[0]} 0
force {SW[1]} 0
force {SW[9]} 0
run 10ns
```

ECE241 FINAL

BOOLEAN ALGEBRA

10a. $x \cdot y = y \cdot x$		10b. $x + y = y + x$	*commutative*
11a. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$		11b. $x + (y + z) = (x + y) + z$	
12a. $x \cdot (y + z) = x \cdot y + x \cdot z$		12b. $x + y \cdot z = (x + y) \cdot (x + z)$	*associative*
13a. $x + x \cdot y = x$		13b. $x \cdot (x + y) = x$	*absorption*
14a. $x \cdot y + x \cdot \bar{y} = x$		14b. $(x + y) \cdot (x + \bar{y}) = x$	*combining*
15a. $\overline{x \cdot y} = \bar{x} + \bar{y}$		15b. $\overline{x + y} = \bar{x} \cdot \bar{y}$	*DeMorgan's*
16a. $x + \bar{x} \cdot y = x + y$		16b. $x \cdot (\bar{x} + y) = x \cdot y$	*covering*
17a. $xy + yz + \bar{x}z = xy + \bar{x}z$		17b. $(x + y)(y + z)(\bar{x} + z) = (x + y)(\bar{x} + z)$	

5a. $x \cdot 0 = 0$ 5b. $x + 1 = 1$

6a. $x \cdot 1 = x$ 6b. $x + 0 = x$

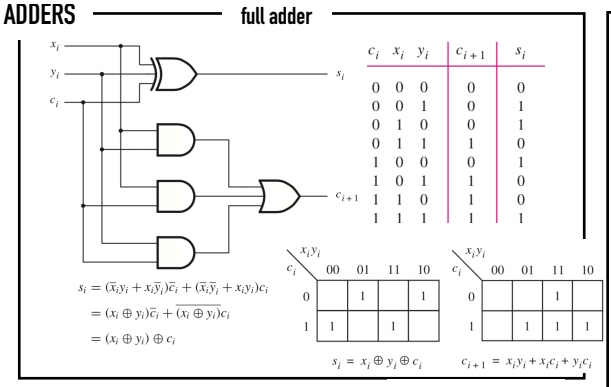
7a. $x \cdot x = x$ 7b. $x + x = x$

8a. $x \cdot \bar{x} = 0$ 8b. $x + \bar{x} = 1$

9. $!\bar{x} = x$

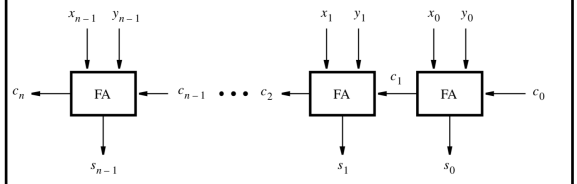
ADDERS

full adder



c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

ripple carry adder



Time: 2n+1

fast adder

Time: 3 gate delays

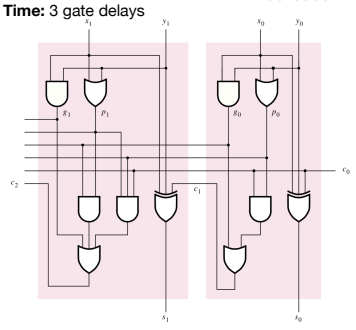


Figure 3.15 The first two stages of a carry-lookahead adder.

$c_{i+1} = x_i y_i + x_i c_i + y_i c_i \quad g_i = x_i y_i$

$c_{i+1} = x_i y_i + (x_i + y_i)c_i \quad p_i = x_i + y_i$

$c_{i+1} = g_i + p_i c_i \quad c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$

$c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_2 p_1 g_0 + p_i p_{i-1} \cdots p_1 p_0 c_0$

multiplication

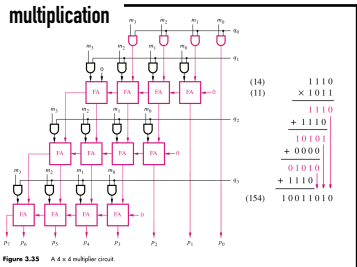
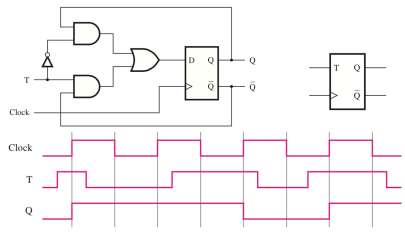


Figure 3.35 A 4 x 4 multiplier circuit.

				m_3	m_2	m_1	m_0	
				× q_3	q_2	q_1	q_0	
Partial product 0				m_3q_0	m_2q_0	m_1q_0	m_0q_0	
Partial product 1			+ m_3q_1	m_2q_1	m_1q_1	m_0q_1		
Partial product 2		+ m_3q_2	m_2q_2	m_1q_2	m_0q_2			
	+ m_3q_3	m_2q_3	m_1q_3	m_0q_3				
Product P	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

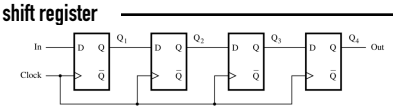
OTHER FLIP FLOPS AND REGISTERS

t flip flop

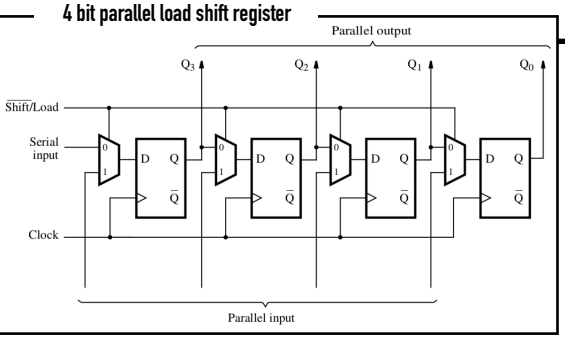


T	Q(t + 1)
0	Q(t)
1	$\bar{Q}(t)$

4 bit right shift register

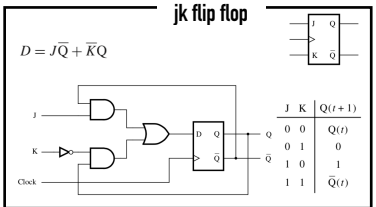


4 bit parallel load shift register



jk flip flop

$D = J\bar{Q} + \bar{K}Q$



J	K	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

~		1's complement
&		Bitwise AND
|		Bitwise OR
^		Bitwise XOR
~^	or	^~ Bitwise XNOR

Shift	>>	Right shift
	<<	Left shift
Concatenation	{,}	Concatenation
Replication	{{}}	Replication
Conditional	?:	Conditional

mod sim

vlib: set the working directory, where all the compiled Verilog goes, use vlib work

vlog: compiles Verilog modules to working directory, use vlog <filename>.v

vsim: starts vsim simulator, use vsim <filename>

log 2 signals: log {SigA, SigB}

add a wave to show two signals: add wave {SigA, SigB}

DE1_SoC.qsf file: maps port names in top-level module to pin numbers in the FPGA chip

FPGA: Field Programmable Gate Array, a prog. device for implementing digital circuits

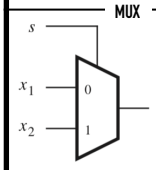
latch is level sensitive, **flip flop** is edge-sensitive

Why is simulation important?

- shows design works before implementation
- makes debugging easier and faster
- in simulation you can see any logic signal, not just input and output
- you can see responses that cannot be observed in hardware

GATES AND LATCHES

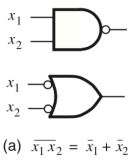
MUX



$f = \bar{s}x_1 + sx_2$

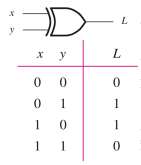
s	f(s, x_1, x_2)
0	x_1
1	x_2

NAND



(a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$

XOR



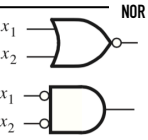
x	y	L
0	0	0
0	1	1
1	0	1
1	1	0

High on odd values of 1

XOR = $x\bar{y} + y\bar{x}$

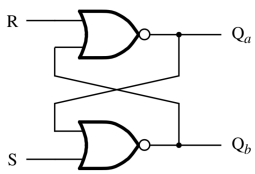
XNOR = $xy + \bar{x}\bar{y}$

NOR



(b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

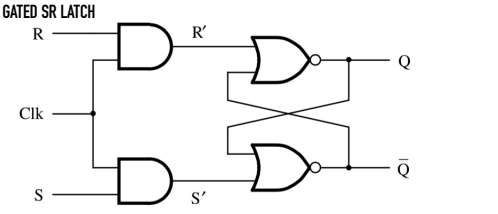
BASIC (SR) LATCH



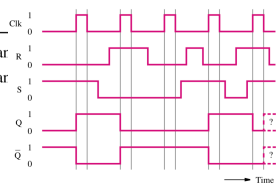
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no chan
0	1	0	1	
1	0	1	0	
1	1	0	0	

CAUTION: When S = R = 1 then S = R = 0, oscillation occurs

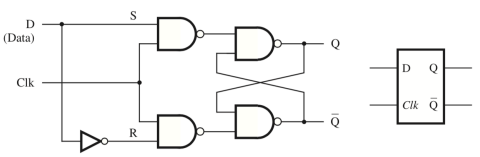
GATED SR LATCH



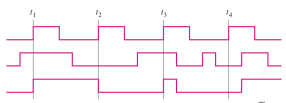
Clk	S	R	Q(t + 1)
0	x	x	Q(t) (no char
1	0	0	Q(t) (no char
1	0	1	0
1	1	0	1
1	1	1	x



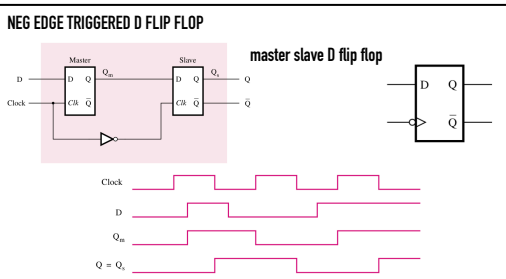
GATED D LATCH / D FLIP FLOP



Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

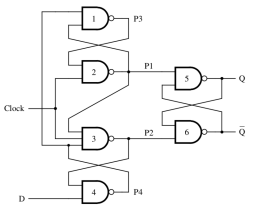


NEG EDGE TRIGGERED D FLIP FLOP



master slave D flip flop

POS EDGE TRIGGERED D FLIP FLOP



Use NOR gates to construct neg edge triggered gate

anais poirier