ECE241 MIDTERM

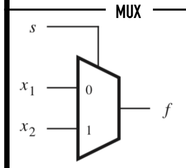
BOOLEAN ALGEBRA

$5a.\ x \cdot 0 = 0$	$5b.\ x + 1 = 1$
$6a.\ x \cdot 1 = x$	$6b.\ x + 0 = x$
$7a.\ x \cdot x = x$	$7b.\ x + x = x$
$8a.\ x \cdot \bar{x} = 0$	$8b.\ x + \bar{x} = 1$
$9.\ !\bar{x} = x$	

$10a.\ x \cdot y = y \cdot x$	$10b.\ x + y = y + x$	*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*
$17a.\ xy + yz + \bar{x}z = xy + \bar{x}z$	$17b.\ (x + y)(y + z)(\bar{x} + z) = (x + y)(\bar{x} + z)$	

GATES AND LATCHES

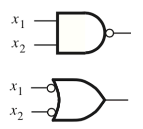
MUX



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

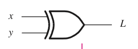
s	$f(s, x_1, x_2)$
0	x_1
1	x_2

NAND



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

XOR

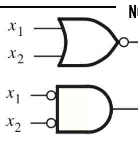


x	y	L
0	0	0
0	1	1
1	0	1
1	1	0

High on odd values of 1

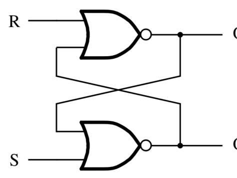
XOR = x$\bar{y}$ + y$\bar{x}$
XNOR = xy + $\bar{x}\bar{y}$

NOR



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

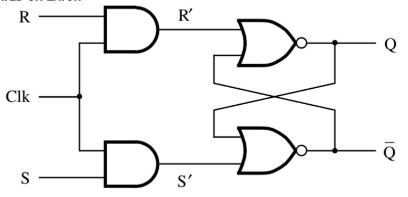
BASIC (SR) LATCH



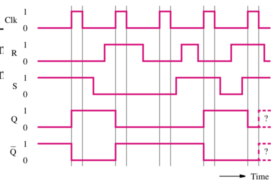
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no chan
0	1	0	1	
1	0	1	0	
1	1	0	0	

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

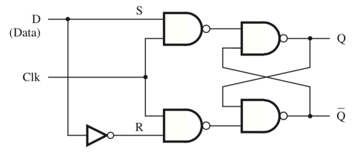
GATED SR LATCH

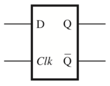


Clk	S	R	$Q(t+1)$
0	x	x	$Q(t)$ (no char
1	0	0	$Q(t)$ (no char
1	0	1	0
1	1	0	1
1	1	1	x

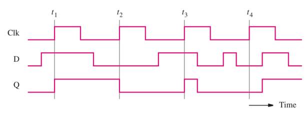


GATED D LATCH / D FLIP FLOP

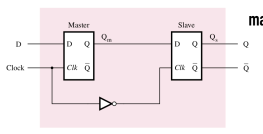




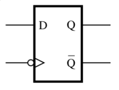
Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

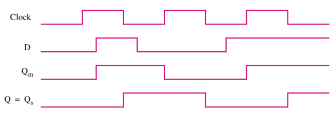


NEG EDGE TRIGGERED D FLIP FLOP

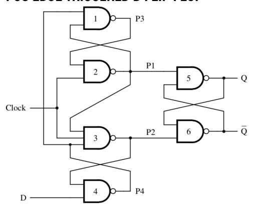


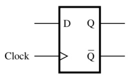
master slave D flip flop





POS EDGE TRIGGERED D FLIP FLOP

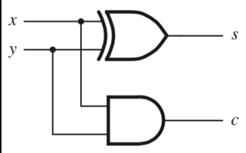




Use NOR gates to construct neg edge triggered gate

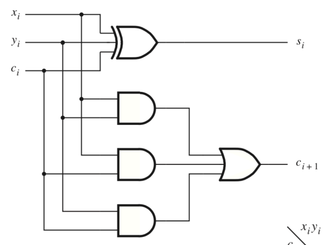
ADDERS

half adder



		Carry	Sum
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

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

$$s_i = (\bar{x}_i y_i + x_i \bar{y}_i)\bar{c}_i + (\bar{x}_i \bar{y}_i + x_i y_i)c_i$$
$$= (x_i \oplus y_i)\bar{c}_i + (\overline{x_i \oplus y_i})c_i$$
$$= (x_i \oplus y_i) \oplus c_i$$

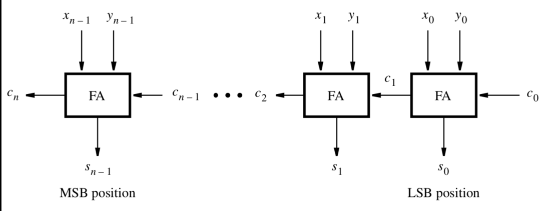
$c_i \backslash x_iy_i$	00	01	11	10
0		1		1
1	1		1	

$s_i = x_i \oplus y_i \oplus c_i$

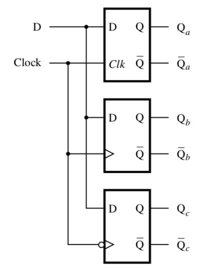
$c_i \backslash x_iy_i$	00	01	11	10
0			1	
1		1	1	1

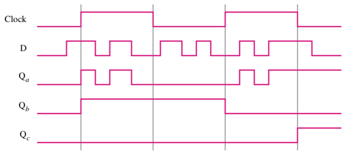
$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$

ripple carry adder



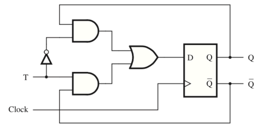
level / pos / neg triggered

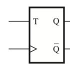




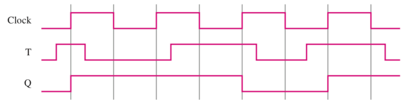
OTHER FLIP FLOPS AND REGISTERS

t flip flop

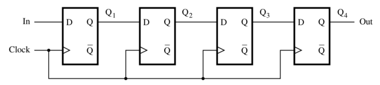




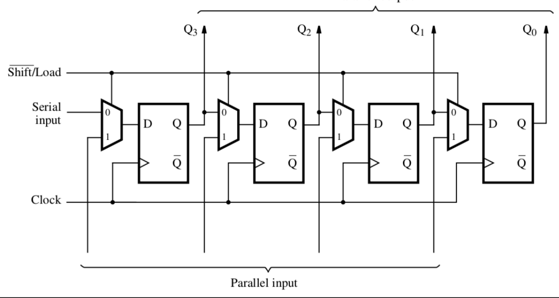
T	$Q(t+1)$
0	$Q(t)$
1	$\bar{Q}(t)$



4 bit right shift register

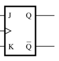


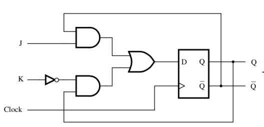
4 bit parallel load shift register



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	$\bar{Q}(t)$

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 results that cannot be observed in hardware

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

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

anais poirier

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
```

add 8

```
module add8(
     input [7:0] A,
     input [7:0] B,
     output [7:0] Sum,
     output Cout);
     assign {Cout, Sum} = A+B;
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
```

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(
          .ci(cin),
          .a(A[0]),
          .b(B[0]),
          .s(S[0]),
          .co(f_cout[0])
          );
     full_adder F0(
          .ci(f_cout[0]),
          .a(A[1]),
          .b(B[1]),
          .s(S[1]),
          .co(f_cout[0])
          );

     full_adder F0(
          .ci(f_cout[1]),
          .a(A[2]),
          .b(B[2]),
          .s(S[2]),
          .co(f_cout[0])
          );

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
```

.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
```