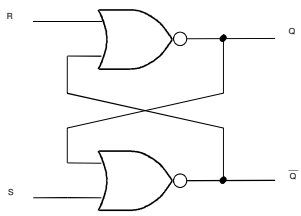
Memory Elements

Engineering 304

- Latches and Flip-Flops
- Registers
  - VHDL versions
- Shift Registers and Counters
  - VHDL versions

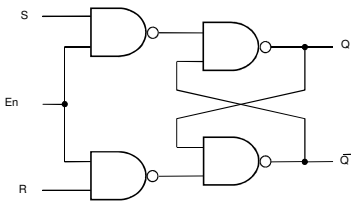
Cross-Coupled NOR Latch

- Simplest form of memory
- Avoid invalid state
  (Q-bar=Q)



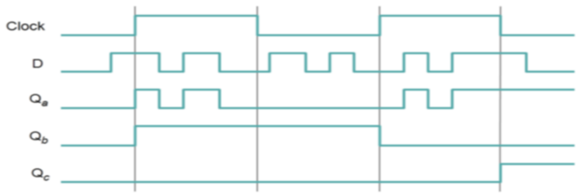
Gated Latch

- Cross-Coupled NAND need S and R inverted
- Often SR latches include a control line (Enable)



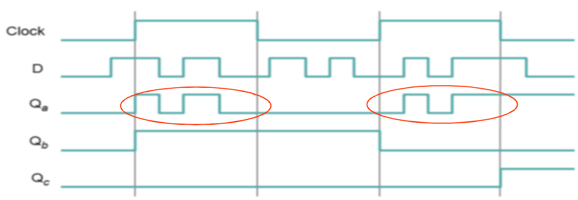
Level vs. Edge

- Latches are level-sensitive using Enable input (Q_a)
  - book calls the Enable the "CLK"
- Flip-flops are edge-triggered using Clock input
  - Rising (Q_b) or Falling (Q_c)



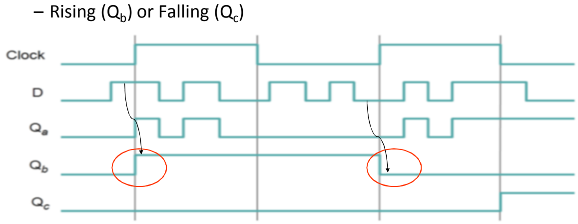
Level vs. Edge

- Latches are level-sensitive using Enable input (Q_a)
  - book calls the Enable the "CLK"
- Flip-flops are edge-triggered using Clock input
  - Rising (Q_b) or Falling (Q_c)



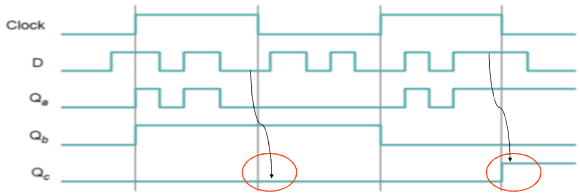
Level vs. Edge

- Latches are level-sensitive using Enable input (Q_a)
  - book calls the Enable the "CLK"
- Flip-flops are edge-triggered using Clock input
  - Rising (Q_b) or Falling (Q_c)



Level vs. Edge

- Latches are level-sensitive using Enable input (Q_a)
  - book calls the Enable the "CLK"
- Flip-flops are edge-triggered using Clock input
  - Rising (Q_b) or Falling (Q_c)



Characteristic Table

- Truth Table that includes time aspect (Q(t) -> Q(t+1))
  - Given certain inputs, what output is expected from the memory device?
  - SR
  - D
  - T
  - JK
- An "Excitation Table" lists the required inputs needed to cause a particular output change

Characteristic Tables

SR Latch

S	R	Q(t)	Q(t+1)	
0	0	0	0	No Change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Invalid
1	1	1	X	

JK Flip-Flop

J	K	Q(t)	Q(t+1)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

D Flip-Flop

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

T Flip-Flop

T	Q(t)	Q(t+1)
0	0	
0	1	
1	0	
1	1	

Characteristic Tables

SR Latch

S	R	Q(t)	Q(t+1)	
0	0	0	0	No Change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Invalid
1	1	1	X	

JK Flip-Flop

J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

D Flip-Flop

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

T Flip-Flop

T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Excitation Tables

SR Latch

Q(t)	Q(t+1)	R	S
0	0	d	0
0	1	0	1
1	0	1	0
1	1	0	d

JK Flip-Flop

Q(t)	Q(t+1)	J	K
0	0		
0	1		
1	0		
1	1		

D Flip-Flop

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

T Flip-Flop

Q(t)	Q(t+1)	T
0	0	
0	1	
1	0	
1	1	

Excitation Tables

SR Latch

Q(t)	Q(t+1)	R	S
0	0	d	0
0	1	0	1
1	0	1	0
1	1	0	d

JK Flip-Flop

Q(t)	Q(t+1)	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

D Flip-Flop

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

T Flip-Flop

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

VHDL D Flip-Flop

```
ENTITY flipflop IS
  PORT (  D, Resetn, Clock     : IN      STD_LOGIC ;
          Q                    : OUT     STD_LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS ( _______________)
  BEGIN
     IF Resetn = '0' THEN
          Q <= '0' ;
     ELSIF Clock'EVENT AND Clock = '1' THEN
          Q <= D ; -- notice there is no "Else" clause
     END IF ;
  END PROCESS ;
END Behavior ;
```

VHDL D Flip-Flop

```
ENTITY flipflop IS
  PORT (  D, Resetn, Clock     : IN      STD_LOGIC ;
          Q                    : OUT     STD_LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
     IF Resetn = '0' THEN
          Q <= '0' ;
     ELSIF Clock'EVENT AND Clock = '1' THEN
          Q <= D ; -- notice there is no "Else" clause
     END IF ;
  END PROCESS ;
END Behavior ;
```

Sens. List must be exactly this!

If-elsif must be exactly this!

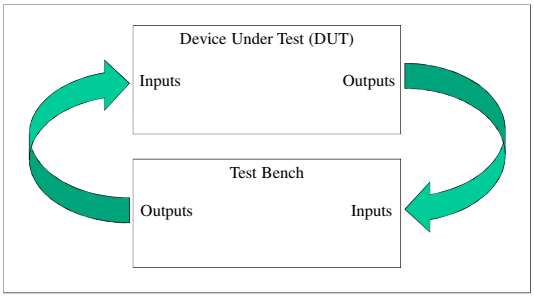
VHDL n-Bit Register

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY regn IS
      GENERIC ( N : INTEGER := 16 ) ;
      PORT ( D                 : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
             Resetn, Clock     : IN  STD_LOGIC ;
             Q                 : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;
ARCHITECTURE Behavior OF regn IS
BEGIN
      PROCESS ( ____________________)
      BEGIN
            IF Resetn = '0' THEN
                  Q <= (OTHERS => '0') ;
            ELSIF ____________________________ THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

VHDL n-Bit Register

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY regn IS
      GENERIC ( N : INTEGER := 16 ) ;
      PORT ( D                : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
             Resetn, Clock    : IN  STD_LOGIC ;
             Q                : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;
ARCHITECTURE Behavior OF regn IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  Q <= (OTHERS => '0') ;
            ELSIF Clock'EVENT AND Clock = '1' THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

Flip-Flop Test Bench



Test Bench Code

```
ENTITY flipflop_TestBench IS
END flipflop_TestBench;
ARCHITECTURE test OF flipflop_TestBench IS
  component flipflop IS port ( -- declare the component
     D, Resetn, Clock : IN STD_LOGIC ;
     Q : OUT STD_LOGIC) ;
  end component flipflop ;
  signal Dsig : std_logic; -- declare the internal signals
  signal Resetnsig : std_logic;
  signal Clocksig : std_logic;
  signal Qsig : std_logic;
BEGIN
DUT:  flipflop  port map (    -- instantiate the flipflop
  D => Dsig, Resetn => Resetnsig, Clock => Clocksig, Q => Qsig
     );
```

Test Bench Code (cont.)

```
process is   -- no sensitivity list => use "wait" instead
begin
  Resetnsig <= '0';
  Clocksig <= '0';
  Dsig <= '1';
  wait for 10 ns;  -- reset everything
  assert Qsig = '0' report "Q not reset to zero"
    severity WARNING;
  Resetn <= '1';
  wait for 10 ns;  -- reset off, but still Q=0
  assert Qsig = '0' report "Q not staying at zero after reset"
    severity WARNING;
  Clocksig <= '1'; -- rising edge should clock in D=1
  wait for 10 ns;  -- let clock edge do something
  assert Qsig = '1' report "Q should now be 1"
    severity WARNING;
```

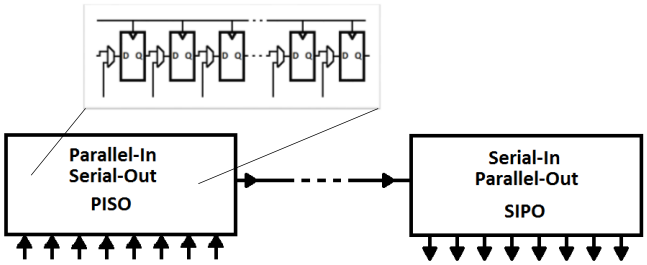
Test Bench Code (cont.)

```
    Dsig <= '0';
    wait for 10 ns;  -- should be no change when changing D
    assert Qsig = '1' report "Q not staying as 1"
      severity WARNING;
    Clocksig <= '0';
    wait for 10 ns;  -- should be no change with falling clk
    assert Qsig = '1' report "Q changed on falling clock"
      severity WARNING;
    Clocksig <= '1'; -- rising edge
    wait for 10 ns;  -- D=0 should be clocked in
    assert Qsig = '0' report "D=0 not clocked in"
      severity WARNING;
    wait; -- never go past this spot - infinite loop-like
  end process;
END test;
```

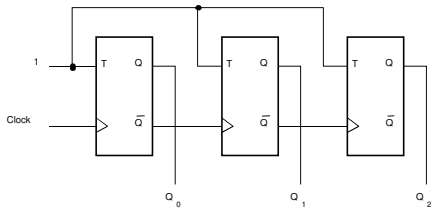
Sequential Devices

- Registers
  - Shift
  - Parallel In, Serial Out
- Counters
  - Up, down, up/down, modulo-n
  - Ring, Johnson (also shift registers)

Shift Register Example

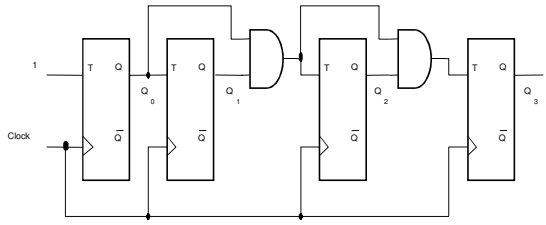


Asynchronous Counter



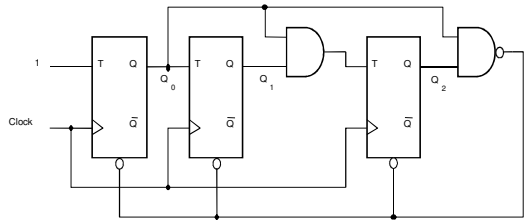
Flip-flops are always set to toggle – must control the clock

Synchronous Counter



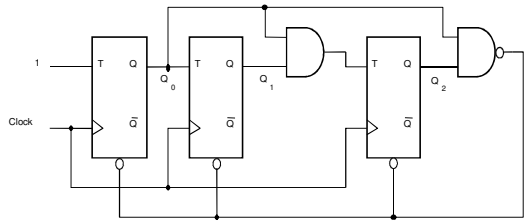
Always clocked – must control toggling with proper flip-flop inputs -> preferred design!

Modulo-n Counters



- As soon as count(Q2 Q1 Q0) = _____then asynchronously clear flip-flops.
  - (5 is never really seen by the user)

Modulo-n Counters



- As soon as count(Q2 Q1 Q0) = "1d1" = 5 (or 7), then asynchronously clear flip-flops. (5 is never really seen by the user)
- Also, consider the 001 to 010 transition

Modulo-n Counter

- Asynchronous load or clear
  - Check for trouble with bits changing slightly out of order
- Best practice is to only use asynchronous clearing through the global chip reset

VHDL Enabled Counter

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
  PORT ( Clock, Resetn, E  : IN STD_LOGIC ;
         Q                 : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS
       SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
       PROCESS ( Clock, Resetn )
       BEGIN
              IF Resetn = '0' THEN  Count <= "0000" ;
              ELSIF (Clock'EVENT AND Clock = '1') THEN
                     IF E = '1' THEN  Count <= Count + 1 ;
                     ELSE   Count <= Count ;
                     END IF ;
              END IF ;
       END PROCESS ;
       Q <= Count ;  -- finally, assign the output Q from the signal Count
END Behavior ;
```

VHDL Enabled Counter

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
  PORT ( Clock, Resetn, E  : IN STD_LOGIC ;
         Q                 : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS
       SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
       PROCESS ( Clock, Resetn )
       BEGIN
              IF Resetn = '0' THEN  Count <= "0000" ;
              ELSIF (Clock'EVENT AND Clock = '1') THEN
                     IF E = '1' THEN  Count <= Count + 1 ;
                     -- ELSE   Count <= Count ;  -- not needed as assumed
                     END IF ;
              END IF ;
       END PROCESS ;
       Q <= Count ;  -- finally, assign the output Q from the signal Count
END Behavior ;
```

VHDL Modulo-n Counter

```
 1 library ieee;
 2 use ieee.std_logic_1164.all;
 3 use ieee.std_logic_arith.all;
 4 use ieee.std_logic_unsigned.all;
 5 entity counter_ex is
 6   generic ( NUM_BITS : integer := 4; MIN_CNT : integer := 4;
 7             MAX_CNT : integer := 13   );
 8   port ( CLK, RSTn : IN std_logic;
 9          Count : OUT std_logic_vector(NUM_BITS - 1 downto 0) );
10 end entity counter_ex;
11 architecture behave of counter_ex is
12   signal cnt : std_logic_vector(NUM_BITS - 1 downto 0);
13 begin
14   cntr: process (RSTn, CLK) is        begin
15     if (RSTn = '0') then
16       cnt <= conv_std_logic_vector(MIN_CNT, NUM_BITS);
17     elsif (CLK = '1' and CLK'event) then -- rising edge
18       if (cnt = MAX_CNT) then
19         cnt <= conv_std_logic_vector(MIN_CNT, NUM_BITS);
20       else
21         cnt <= cnt + 1;  -- increment the count
22       end if;
23     end if;  -- no else, so inferring a register here
24   end process;
25   Count <= cnt;
26 end architecture behave;
```

VHDL Enabled Counter

```
ARCHITECTURE Behavior OF counter IS
  component counter_ex is
    generic(NUM_BITS : integer := 4; MIN_CNT : integer := 4;
            MAX_CNT : integer := 13);
    port(CLK, RSTn : IN std_logic;
         Count : OUT std_logic_vector((NUM_BITS - 1 downto 0));
  end component counter_ex;
BEGIN
C0 : component counter_ex
     generic map (NUM_BITS => 4, MIN_CNT => 2, MAX_CNT => 11)
     port map (CLK => my_clock, RSTn => my_reset, Count => Count0);
C1 : component counter_ex
     generic map (NUM_BITS => 6, MIN_CNT => 3, MAX_CNT => 49)
     port map (CLK => my_clock, RSTn => my_reset, Count => Count1);
C2 : component counter_ex
     generic map (NUM_BITS => 3, MIN_CNT => 1, MAX_CNT => 7)
     port map (CLK => my_clock, RSTn => my_reset, Count => Count2);
END Behavior ;
```

VHDL Modulo-n Counter - Quartus