University of British Columbia
Electrical and Computer Engineering
Introduction to Microcomputers EECE259

Lecture 10: Synchronous Counters.

Dr. Jesús Calviño-Fraga. P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

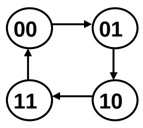
February 7/9, 2017

Objectives

- Design synchronous counters.
- Learn about shift registers.
- Write code for VHDL counters.

Synchronous Counters

- ALL the flip-flops are clocked at the same time by a common clock signal.
- We can easily design these counters using the sequential logic design process.
- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).



Current state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

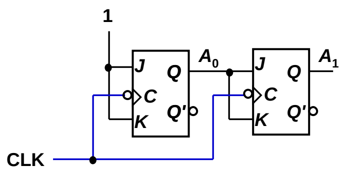
Synchronous Counters

- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).

Current state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

$TA_1 = A_0$

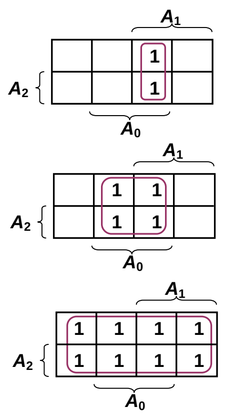
$TA_0 = 1$



Synchronous Counters: 3-bit counter

- 3-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J, K inputs).

Current state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



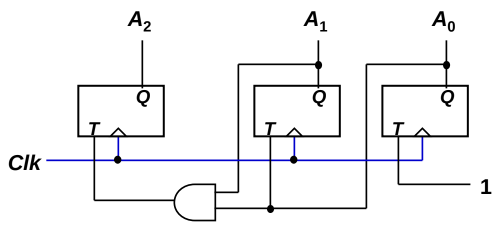
$TA_2 = A_1.A_0$

$TA_1 = A_0$

$TA_0 = 1$

Synchronous Counters: 3-bit counter

$TA_2 = A_1.A_0 \quad TA_1 = A_0 \quad TA_0 = 1$



4-bit Synchronous Counter (with T FF)

Current state				Next state				Flip-flop inputs			
A_3	A_2	A_1	A_0	A_3^+	A_2^+	A_1^+	A_0^+	TA_3	TA_2	TA_1	TA_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	1	0	1	1	0	0	0	1
1	0	1	1	1	1	0	0	0	1	1	1
1	1	0	0	1	1	0	1	0	0	0	1
1	1	0	1	1	1	1	0	0	0	1	1
1	1	1	0	1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0	1	1	1	1

$TA_0=1$

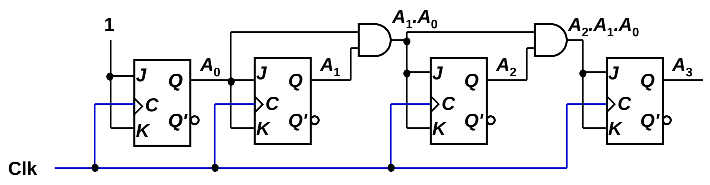
$TA_1=A_0$

$TA_2 = A'_3.A'_2.A_1.A_0+A'_3.A_2.A_1.A_0+A_3.A'_2.A_1.A_0+A_3.A_2.A_1.A_0=A_1.A_0$

$TA_3 = A'_3.A_2.A_1.A_0+A_3.A_2.A_1.A_0=A_2A_1.A_0$

Synchronous Counters: 4-bit counter

$TA_3 = A_2 \cdot A_1 \cdot A_0$
$TA_2 = A_1 \cdot A_0$
$TA_1 = A_0$
$TA_0 = 1$



Can you see the pattern? What about a 5-bit counter?

BCD Synchronous Counter

Current state				Next state				Flip-flop inputs			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+	TA_3	TA_2	TA_1	TA_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1

$TA_0 = 1$

$TA_1 = Q_3'.Q_0$

$TA_2 = Q_1.Q_0$

$TA_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$

Assuming no “Invalid” current state (for example 1010).

BCD Synchronous Counter

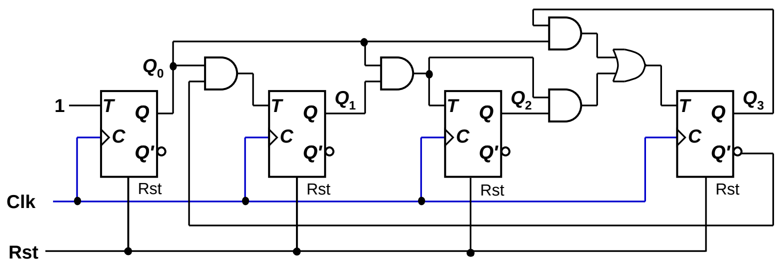
$T_0 = 1$

$T_1 = Q_3'.Q_0$

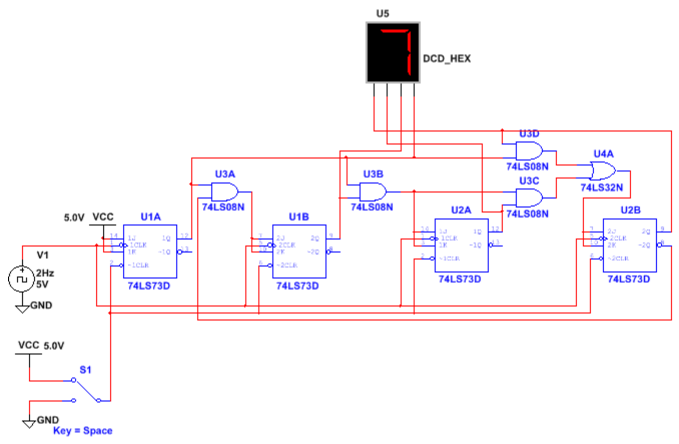
$T_2 = Q_1.Q_0$

$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$

Rst must be asserted at some time to prevent illegal states.



BCD Synchronous Counter



BCD Synchronous Counter

Current state				Next state				Flip-flop inputs			
A_3	A_2	A_1	A_0	A_3^+	A_2^+	A_1^+	A_0^+	TA_3	TA_2	TA_1	TA_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	0	0	0	1	0	1	0
1	0	1	1	0	0	0	0	1	0	1	1
1	1	0	0	0	0	0	0	1	1	0	0
1	1	0	1	0	0	0	0	1	1	0	1
1	1	1	0	0	0	0	0	1	1	1	0
1	1	1	1	0	0	0	0	1	1	1	1

The solution is:

Exercise at the end!

Designing With D Flip-Flops

- For a T Flip-Flop if the next state changes a bit, then the T input for that bit is set to 1.
- For a D Flip-Flop we just need to make the input D equal to the next state!

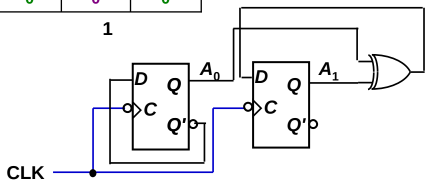
Synchronous Counters

- Example: 2-bit synchronous binary counter (using D flip-flops).

Current state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	DA_1	DA_0
0	0	0	1	0	1
0	1	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0

$DA_1 = A_0 \oplus A_1$

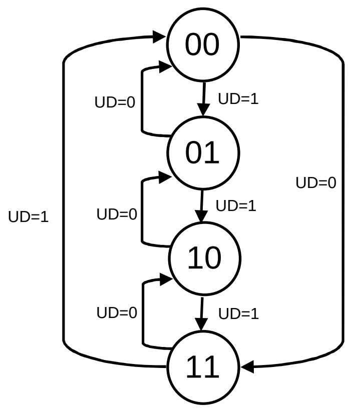
$DA_0 = A'_0$



Arbitrary Order Synchronous Counter

- Just as in the previous slide, use a D-type FF (or any FF for that matter!), obtain the table, get the FF input functions and that is it!
- For example, design a 2-bit up/down counter using D-type FFs:

2-bit Up/Down Counter



2-bit Up/Down Counter

Current state			Next state	
UD	A_1	A_0	D_1	D_0
0	**0**	**0**	**1**	**1**
0	**0**	**1**	**0**	**0**
0	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**0**

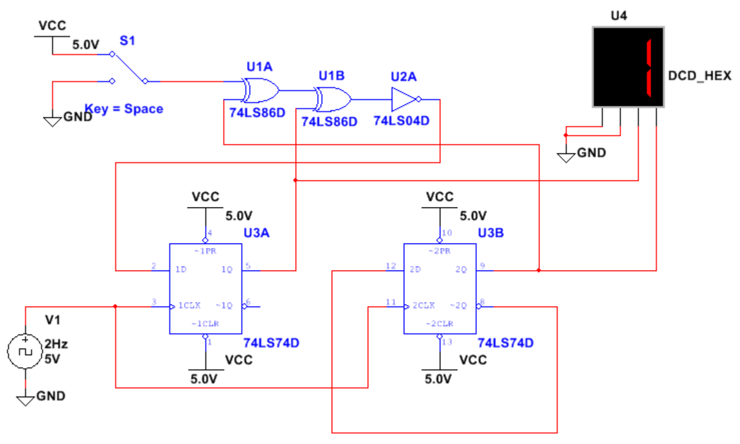
$D_0=A'_0$

$D_1=UD'.A'_1.A'_0+UD'.A_1.A_0+UD.A'_1.A_0+UD.A_1.A'_0$

$D_1=UD.(A_1\oplus A_0)+UD'.(A_1\oplus A_0)'$

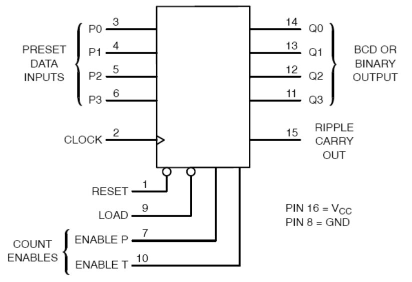
$D_1=(UD\oplus A_1\oplus A_0)'$

2-bit Up/Down Counter



Parallel Load

- Many commercially available counters can be set (synchronously or asynchronously) to any state.
- Parallel load is very easy to implement in VHDL.

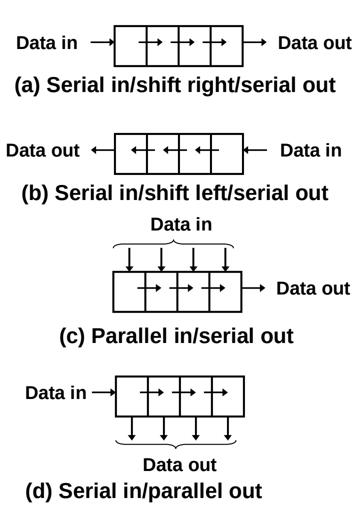


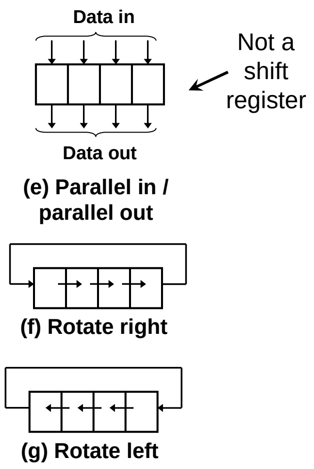
74LS161/74LS163

Shift Registers

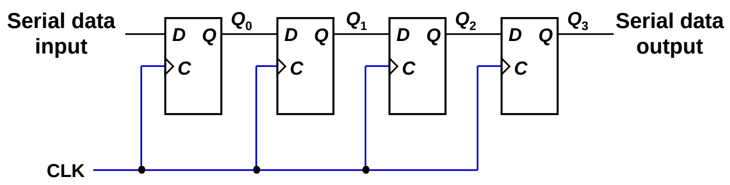
- A shift register is an arrangement of (usually) synchronous flip-flops that are used to move bits.
- One flip-flop is needed for each bit in the shift register. Each bits moves from flip-flop to flip-flop on an edge of the clock.

Shift Register Configurations

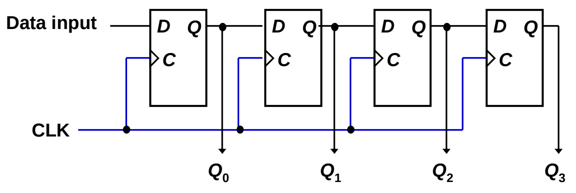

Data in
Data out
(a) Serial in/shift right/serial out
Data out
Data in
(b) Serial in/shift left/serial out
Data in
Data out
(c) Parallel in/serial out
Data in
Data out
(d) Serial in/parallel out



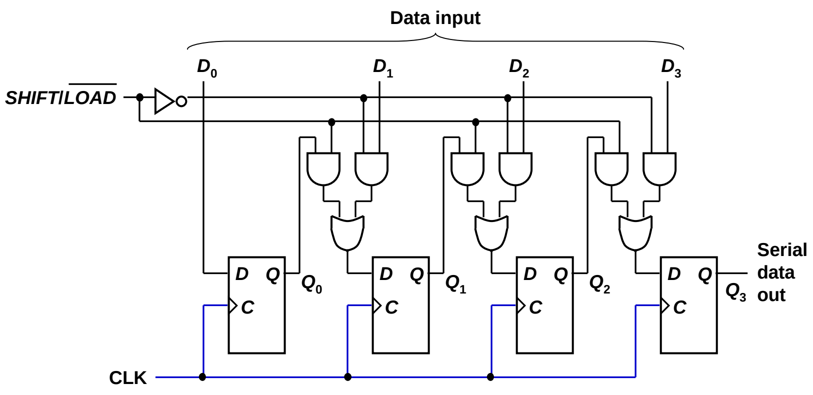
Serial in/Serial out Shift Register



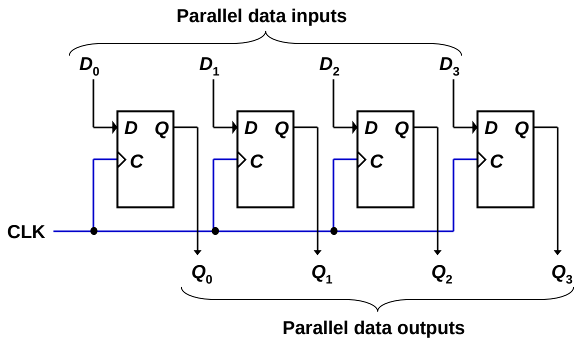
Serial In / Parallel Out



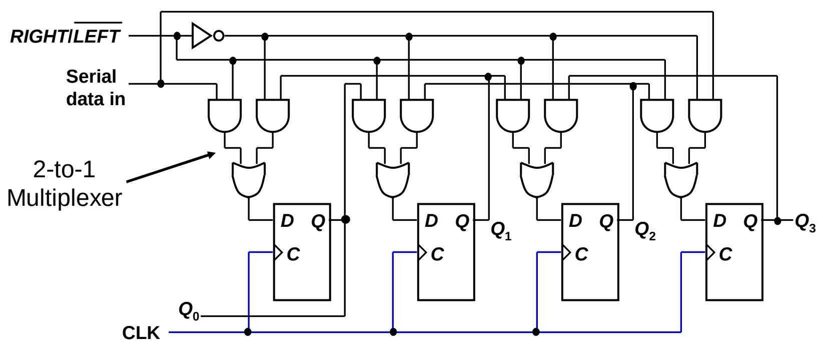
Parallel In / Serial Out



Parallel In / Parallel Out

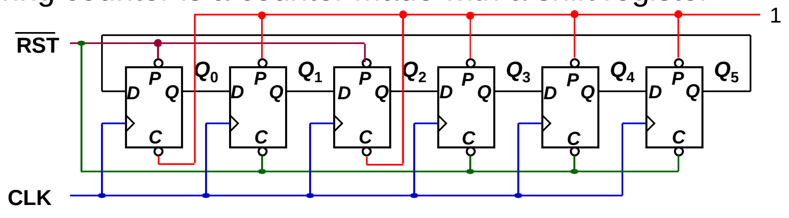


Shift Right/Left Register

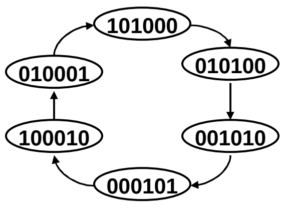


Ring Counters

- A ring counter is a counter made with a shift register.

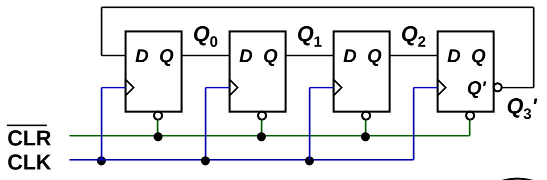


Clock	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5
0	1	0	1	0	0	0
1	0	1	0	1	0	0
2	0	0	1	0	1	0
3	0	0	0	1	0	1
4	1	0	0	0	1	0
5	0	1	0	0	0	1

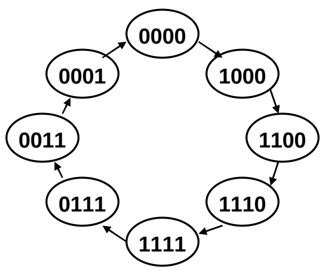


Johnson Counter

- Similar to the ring counter but Q' of the last flip-flop is connected to D of the first flip-flop.



Clock	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1



Registers and Counters in VHDL

- These examples are from Digital Logic with VHDL Design, 3rd Edition, by Brown and Vranesic.
  - 8-bit, n-bit register with clear
  - D flip-flop With a 2-to-1 Multiplexer on the D Input.
  - 4-bit Shift Register Using D FFs.
  - Alternative Code for a Shift Register.
  - Generic n-bit Shift Register.
  - 4-bit Counter With Synchronous Load.
  - 4-bit Counter With Using INTEGER Signals.
  - 4-bit Down Counter.

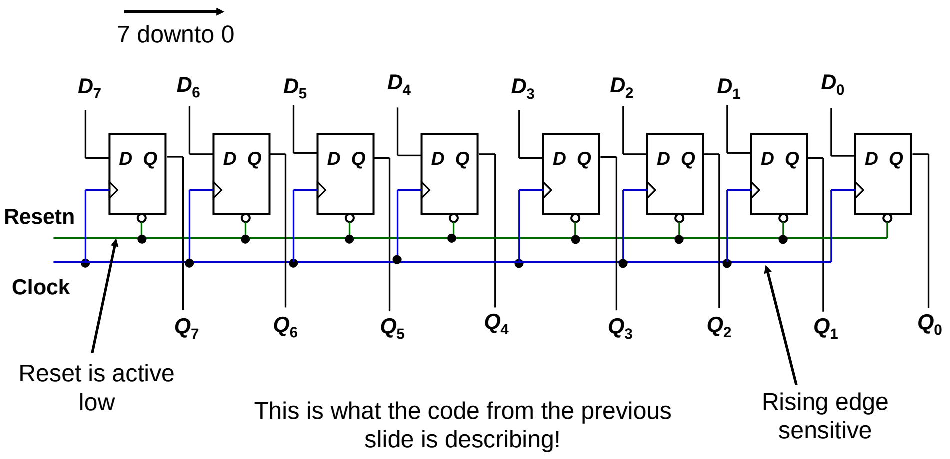
8-bit Register With Clear

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY reg8 IS
    PORT (  D               : IN    STD_LOGIC_VECTOR(7 DOWNTO 0) ;
            Resetn, Clock   : IN    STD_LOGIC ;
            Q               : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END reg8 ;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Reset is active low

Rising edge sensitive

8-bit Register With Clear



This is what the code from the previous slide is describing!

n-bit Register With Clear

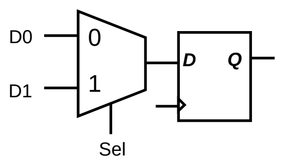
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```

D flip-flop With a 2-to-1 Multiplexer on the D Input.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY muxdff IS
    PORT (  D0, D1, Sel, Clock   : IN    STD_LOGIC ;
            Q                    : OUT   STD_LOGIC ) ;
END muxdff ;

ARCHITECTURE Behavior OF muxdff IS
BEGIN
    PROCESS (Clock, Sel)
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Sel = '0' THEN
            Q <= D0 ;
        ELSE
            Q <= D1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Forces synthesis of FF

D flip-flop With a 2-to-1 Multiplexer on the D Input.

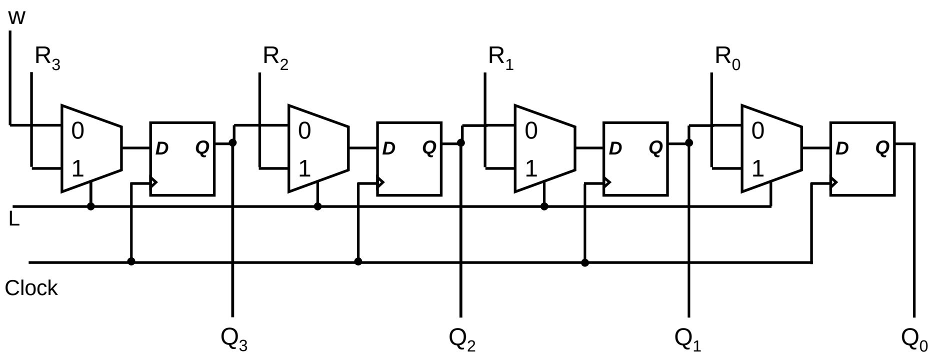


4-bit Shift Register Using D FFs

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shift4 IS
    PORT (  R                : IN       STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            L, w, Clock      : IN       STD_LOGIC ;
            Q                : BUFFER   STD_LOGIC_VECTOR(3 DOWNTO 0) )
;
END shift4 ;

ARCHITECTURE Structure OF shift4 IS
    COMPONENT muxdff
        PORT (  D0, D1, Sel, Clock  : IN    STD_LOGIC ;
                Q                   : OUT   STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) ) ;
    Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) ) ;
    Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) ) ;
    Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) ) ;
END Structure ;
```

Or… use the graphic editor!

4-bit Shift Register Using D FFs



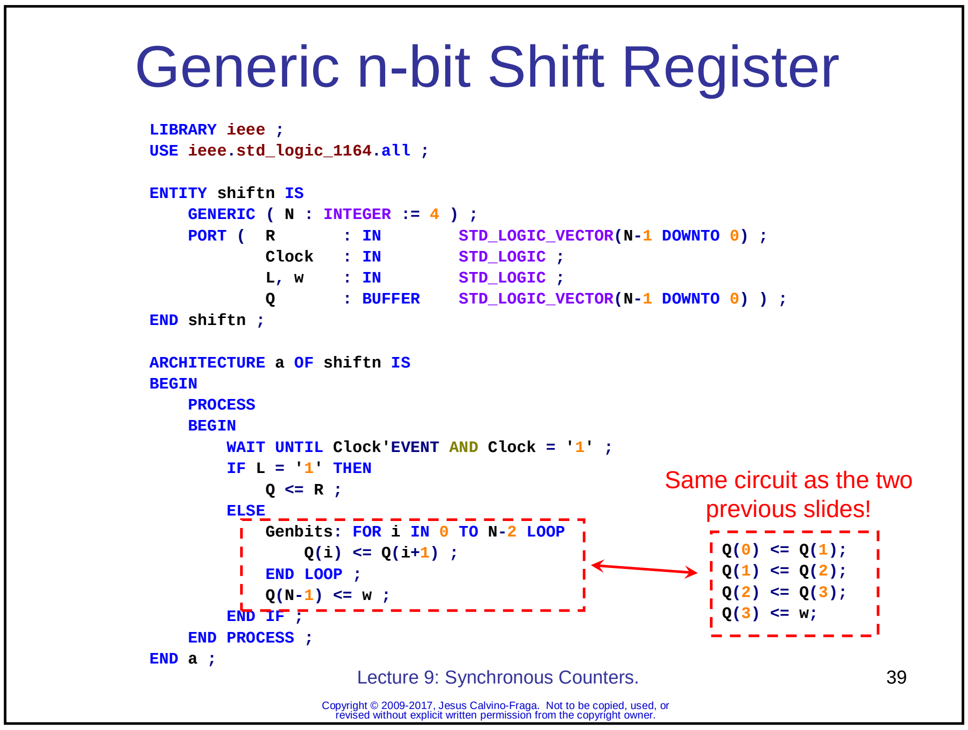
Alternative Code for a Shift Register

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shift4 IS
    PORT (  R       : IN        STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            Clock   : IN        STD_LOGIC ;
            L, w    : IN        STD_LOGIC ;
            Q       : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '1' THEN
            Q <= R ;
        ELSE
            Q(0) <= Q(1);
            Q(1) <= Q(2);
            Q(2) <= Q(3);
            Q(3) <= w;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Alternative Code for a Shift Register

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shift4 IS
    PORT (  R       : IN        STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            Clock   : IN        STD_LOGIC ;
            L, w    : IN        STD_LOGIC ;
            Q       : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '1' THEN
            Q <= R ;
        ELSE
            Q(3) <= w;
            Q(2) <= Q(3);
            Q(1) <= Q(2);
            Q(0) <= Q(1);
        END IF ;
    END PROCESS ;
END Behavior ;
```

Change the order of this statements and the result is… exactly the same as in the previous slide. These statements represent WIRES. They are not 'executed' sequentially. They happen at the same time.

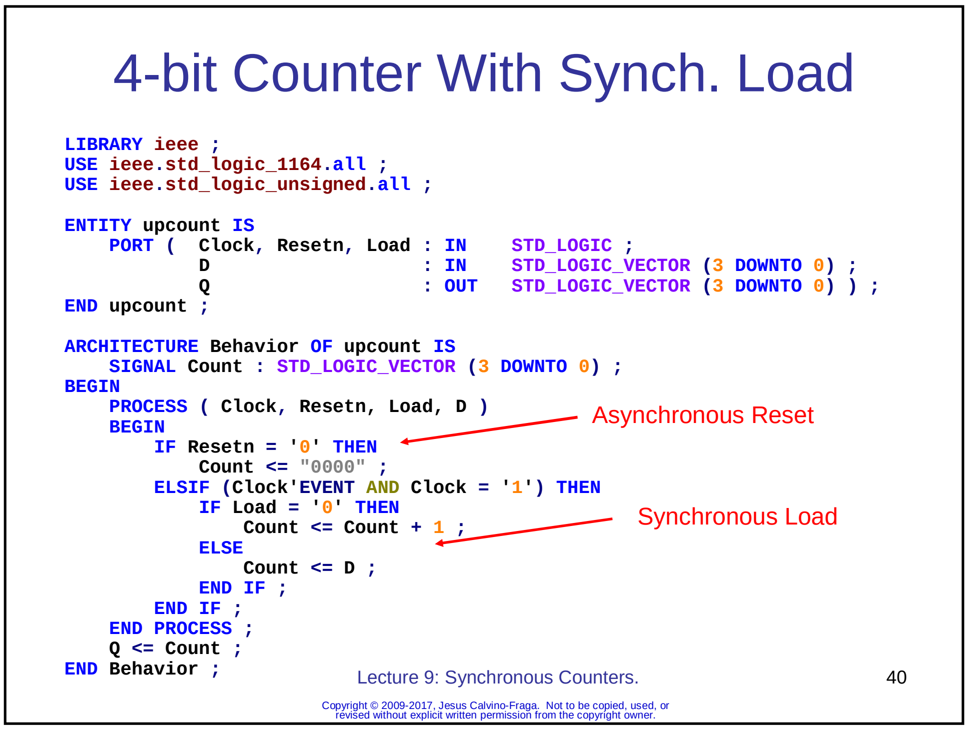


Generic n-bit Shift Register

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shiftn IS
    GENERIC ( N : INTEGER := 4 ) ;
    PORT (  R       : IN        STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
            Clock   : IN        STD_LOGIC ;
            L, w    : IN        STD_LOGIC ;
            Q       : BUFFER    STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shiftn ;

ARCHITECTURE a OF shiftn IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '1' THEN
            Q <= R ;
        ELSE
            Genbits: FOR i IN 0 TO N-2 LOOP
                Q(i) <= Q(i+1) ;
            END LOOP ;
            Q(N-1) <= w ;
        END IF ;
    END PROCESS ;
END a ;
```

Same circuit as the two previous slides!

```
Q(0) <= Q(1);
Q(1) <= Q(2);
Q(2) <= Q(3);
Q(3) <= w;
```



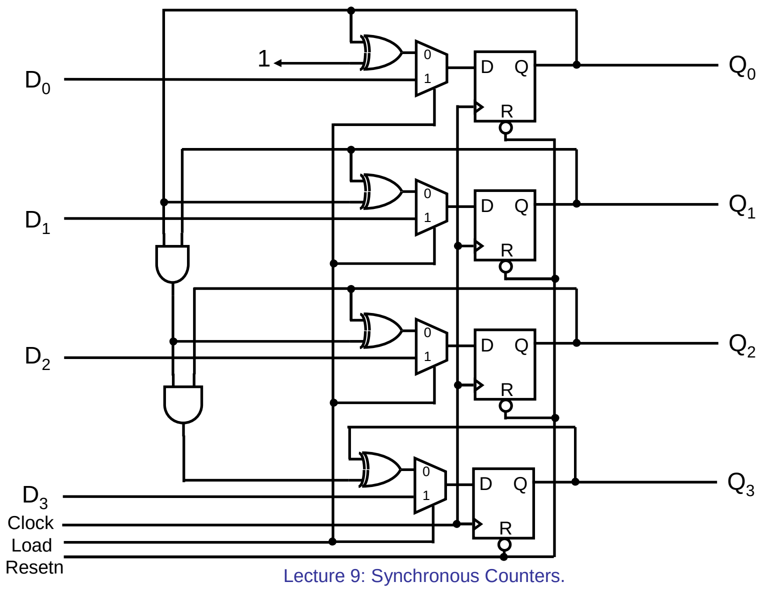
4-bit Counter With Synch. Load

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY upcount IS
    PORT (  Clock, Resetn, Load : IN    STD_LOGIC ;
            D                   : IN    STD_LOGIC_VECTOR (3 DOWNTO 0) ;
            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, Load, D )
    BEGIN
        IF Resetn = '0' THEN
            Count <= "0000" ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF Load = '0' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= D ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

Asynchronous Reset

Synchronous Load

4-bit Counter With Synch. Load



4-bit Counter With Using INTEGER Signals

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY upcount IS
    PORT (  R                    : IN      INTEGER RANGE 0 TO 15 ;
            Clock, Resetn, L     : IN      STD_LOGIC ;
            Q                    : BUFFER  INTEGER RANGE 0 TO 15 ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= 0 ;
        ELSIF (Clock'EVENT AND Clock = '0') THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                Q <= Q + 1 ;
            END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Changes are on falling edge of Clock.

4-bit Down Counter

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY downcnt IS
    GENERIC ( modulus : INTEGER := 8 ) ;
    PORT ( Clock, L, E : IN  STD_LOGIC ;
           Q           : OUT INTEGER RANGE 0 TO modulus-1 ) ;
END downcnt ;

ARCHITECTURE abc OF downcnt IS
    SIGNAL Count : INTEGER RANGE 0 TO modulus-1 ;
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL (Clock'EVENT AND Clock = '1') ;
        IF L = '1' THEN
            Count <= modulus-1 ;
        ELSE
            IF E = '1' THEN
                Count <= Count-1 ;
            END IF ;
        END IF ;
    END PROCESS;
    Q <= Count ;
END abc ;
```

Exercises

- Design a 5-bit synchronous counter using T flip-flops.
- Design a BCD synchronous counter were all the invalid states reset back to zero.
- Design a 3-bit Gray code counter using D flip-flops.
- Write the VHDL code for a 4-bit binary up/down counter, rising edge sensitive clock, active low reset, and active low parallel load.
- Write the VHDL code for one digit BCD down counter with active low reset and falling edge sensitive clock.
- Design a synchronous digital circuit that produces the count 0, 1, 2, 3, 5, 7, 11, 13, 0, 1, 2, 3, …