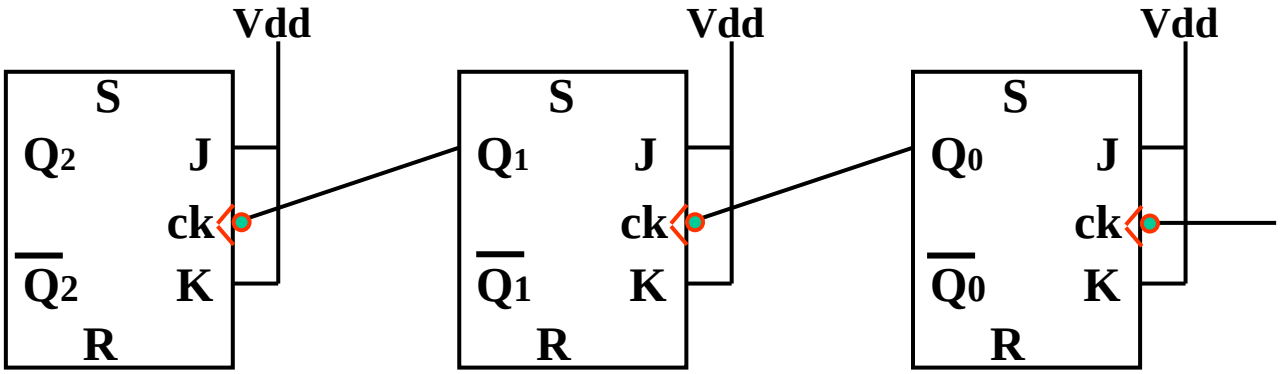
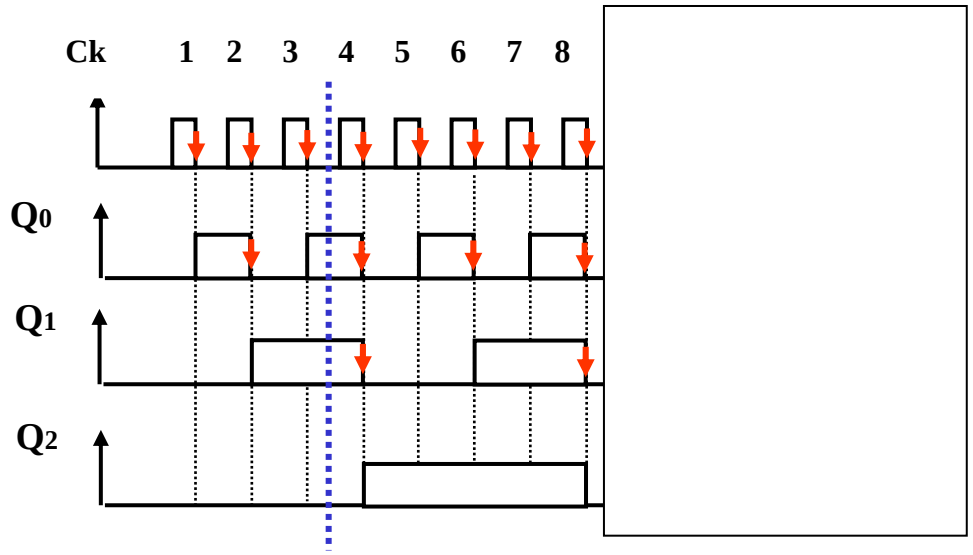
CONTADORES

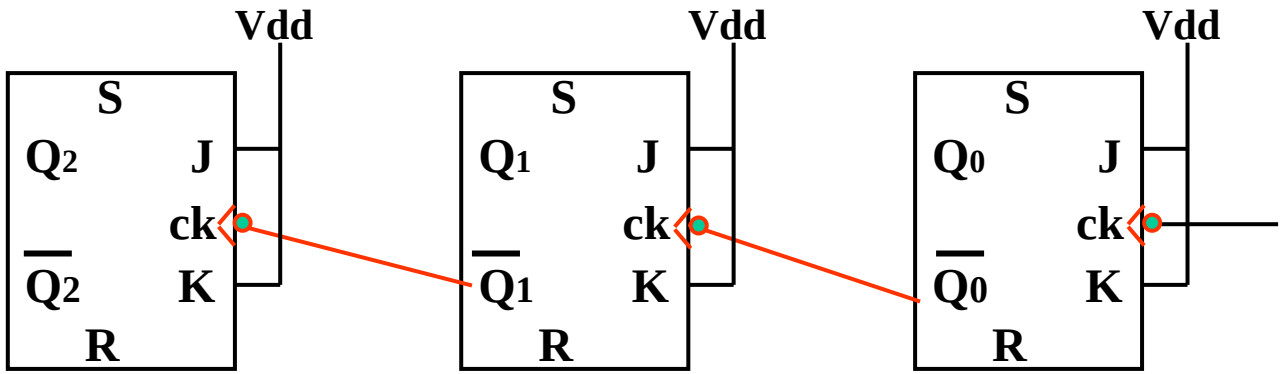
CONTADORES ASINCRONOS ASCENDENTE



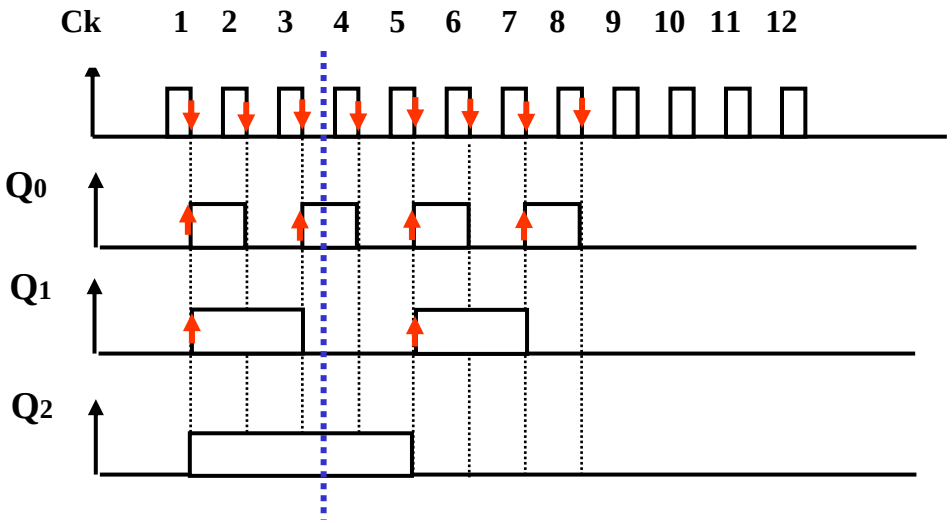
Las entradas asincronas S y R estan desactivadas -- CI : 0 0 0



CONTADOR ASINCRONO DESCENDENTE

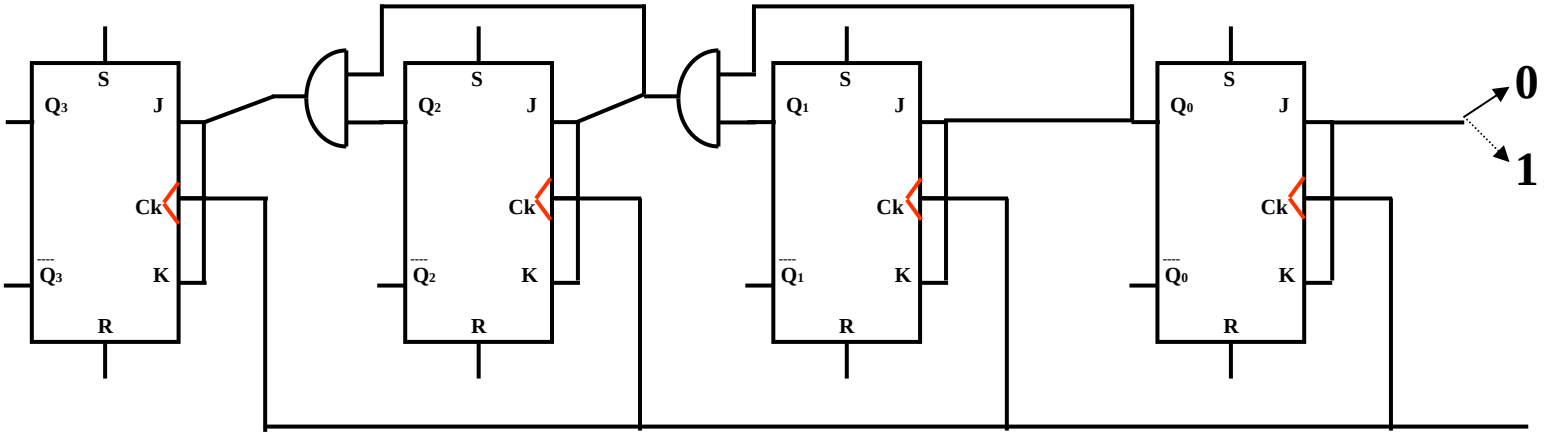


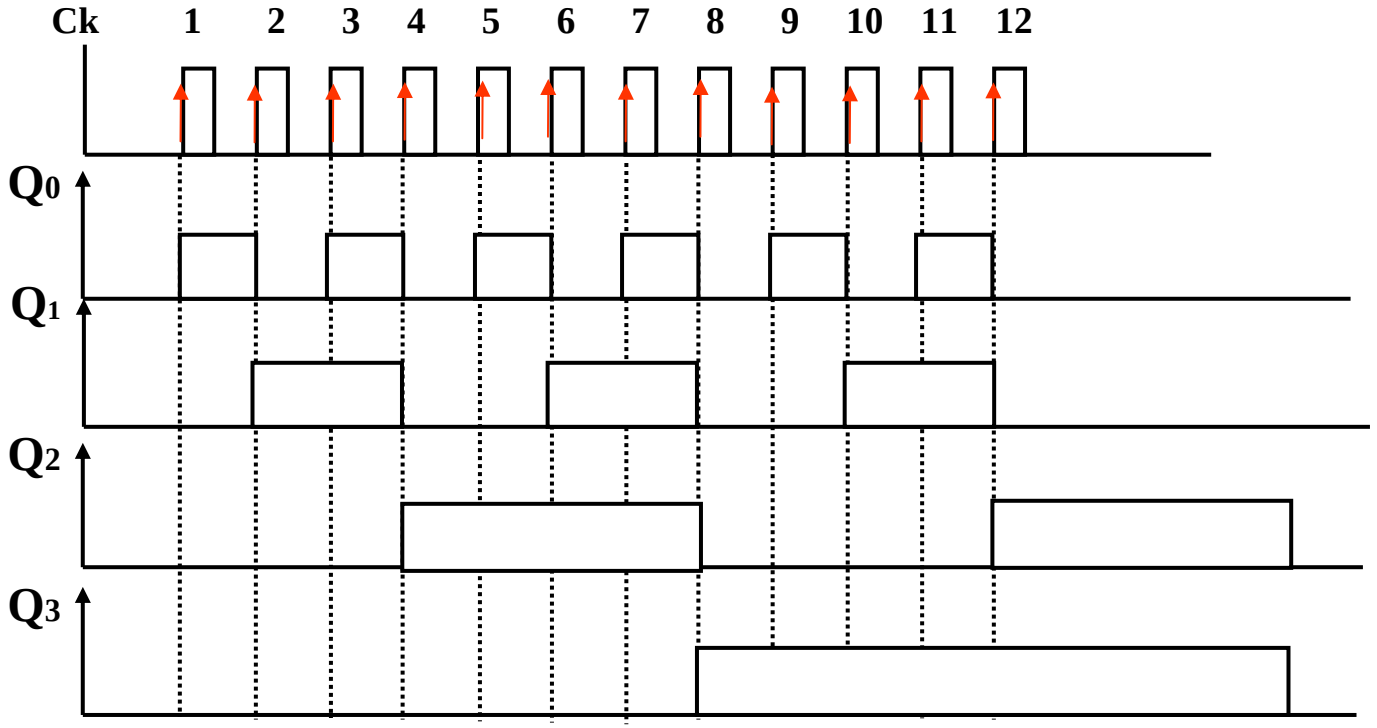
Las entradas asincronas S y R estan desactivadas -- CI : 0 0 0



CONTADORES SINCRONOS 1

(CONECTAR LAS ENTRADAS ASINCRONAS)

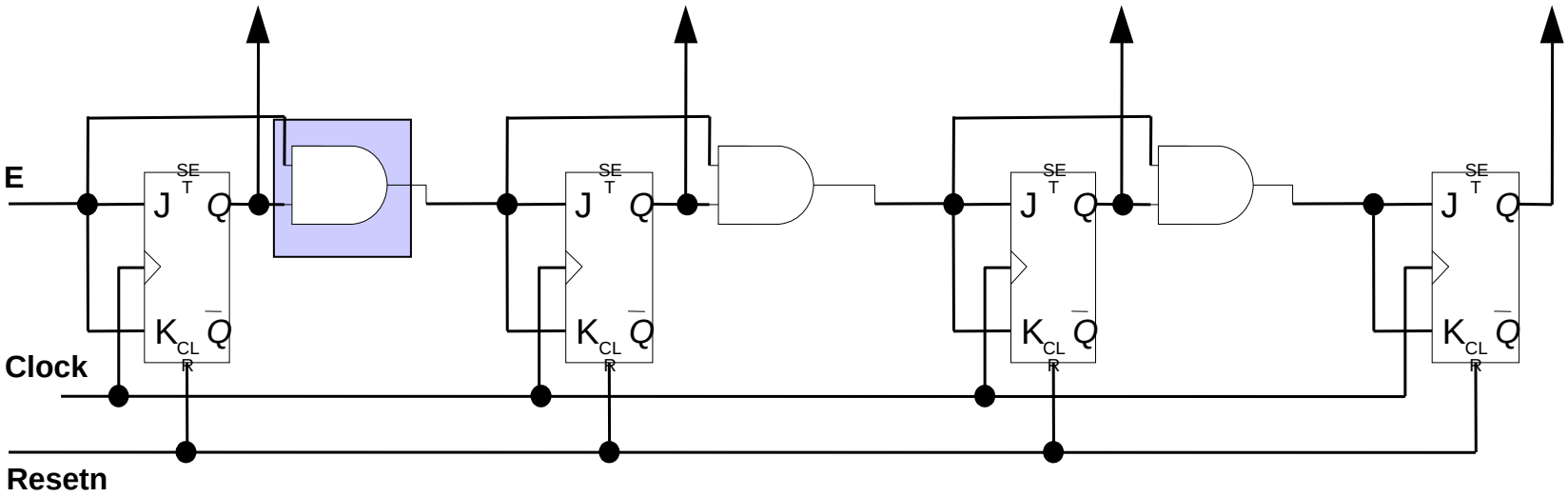




MODELO VHDL – CONTADOR – ARQUITECTURA

```
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 ;
END Behavoir ;
```

MODELO VHDL - CONTADOR



Este contador es identico al anteriormente visto, solo que hemos agregado una compuerta adicional con la funcion de establecer una entrada de habilitacion E. Se muestra el circuito de un contador ascendente de 4 bit, (de la diapositiva anterior) con una entrada reset *Resetn* , y una entrada de habilitación, *E* . En el cuerpo de la arquitectura los FFs en el contador estan representados por una señal denominada *Count* (recordemos que una señal puede almacenar información y representan conexiones o terminales físicos en el circuito) La sentencia process especifica un reset asíncrono de *Count* si *Resten = 0* . La clausula ELSIF especifica que en el flanco positivo del Clock, si *E = 1*, la cuenta es incrementada .. Si *E = 0* , el código explícitamente asigna *Count <= Count*. Las salidas Q estan asignadas al valor de *Count* al final del código

MODULO PROGRAMABLE

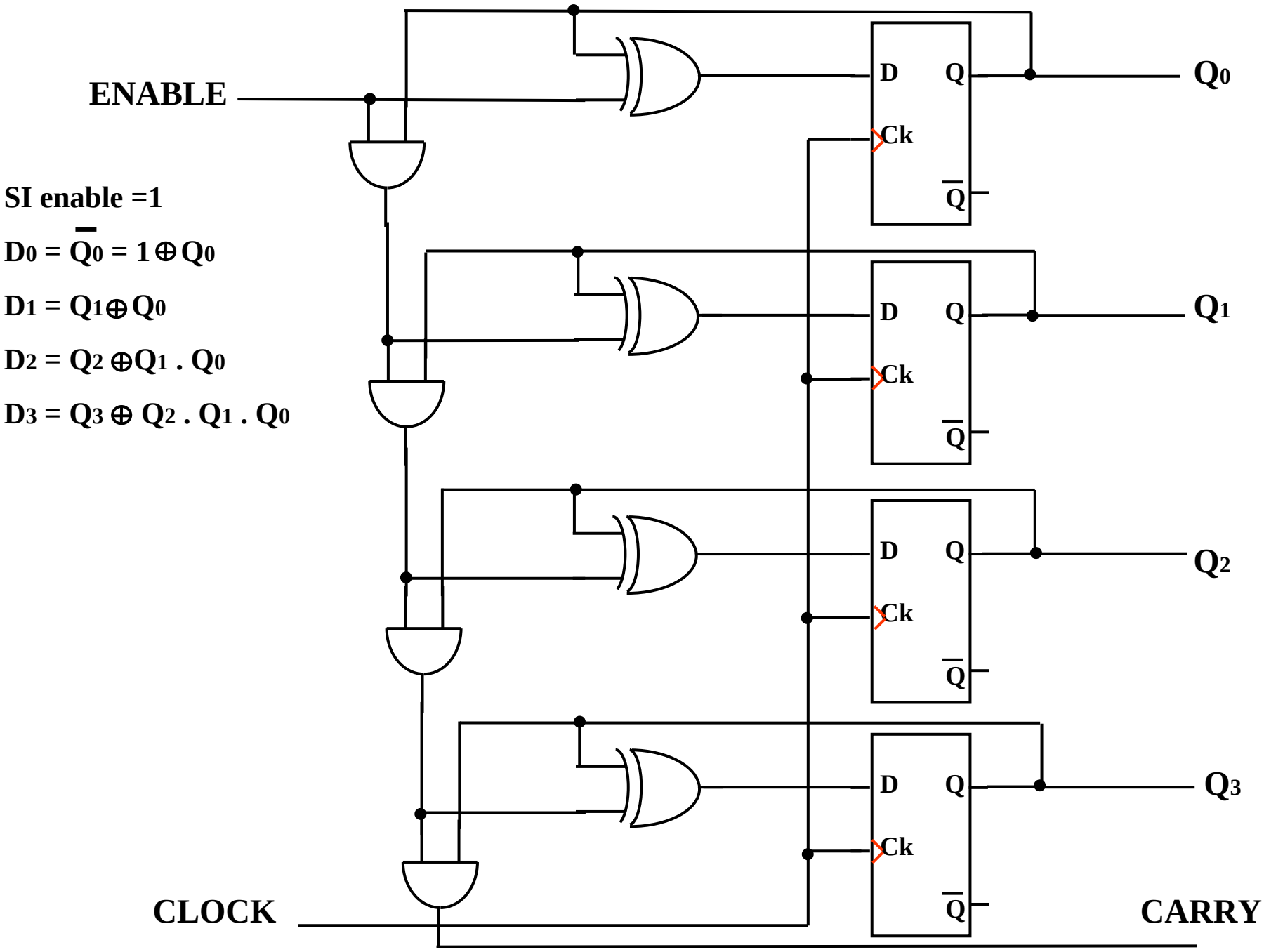
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY downcnt IS
        GENERIC ( modulus : INTEGER := 8 ) ;
        PORT (    Clock, L, E    : IN     STD_LOGIC ;
                  Q              : OUT    INTEGER RANGE 0 TO modulus-1 ) ;
END downcnt ;

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

Figure 7.55 Code for a down-counter

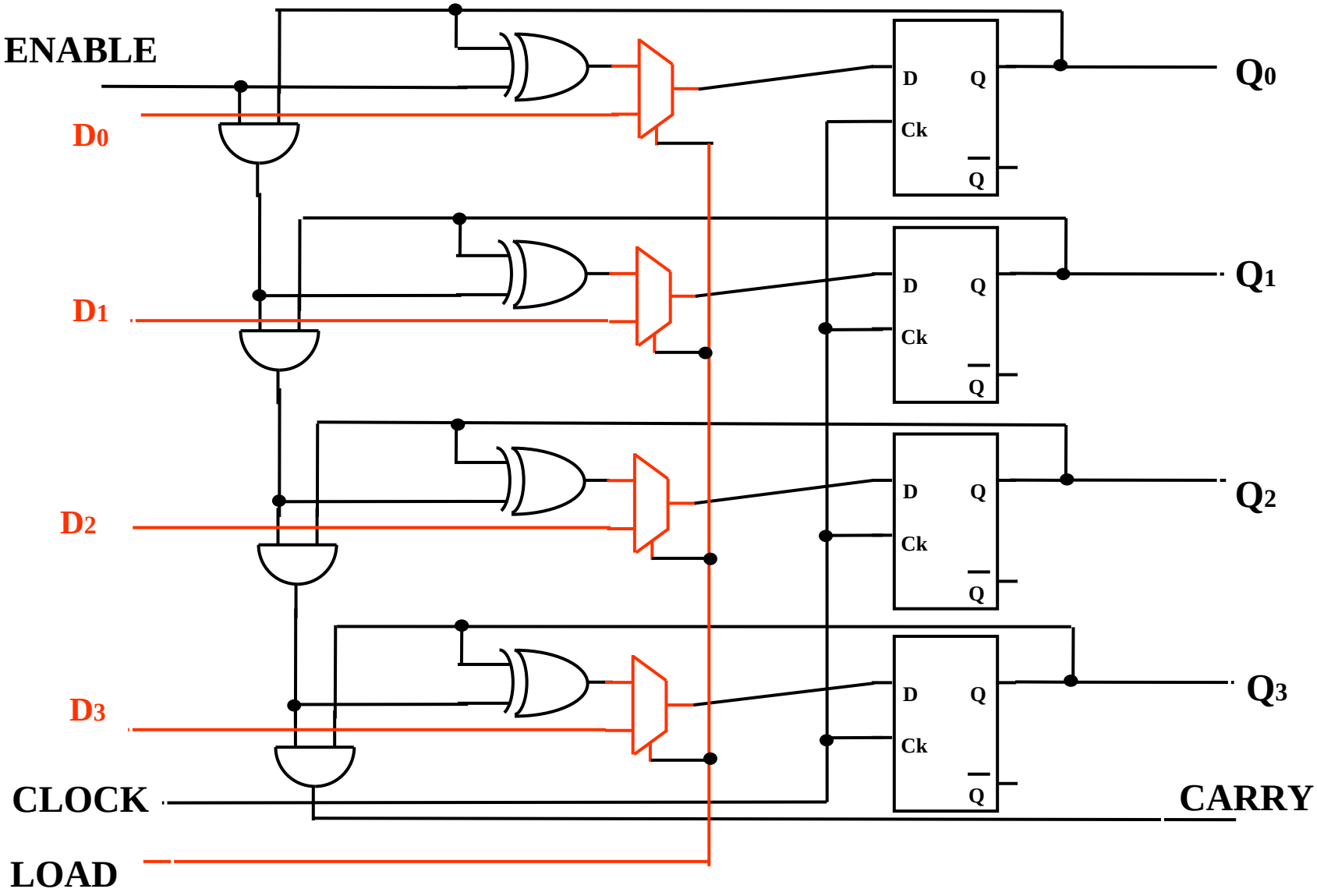
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 = '1') THEN
                        IF L = '1' THEN
                                Q <= R ;
                        ELSE
                                Q <= Q + 1 ;
                        END IF;
                END IF;
        END PROCESS;
END Behavior;
```

Figure 7.54 A four-bit counter with parallel load, using INTEGER signals

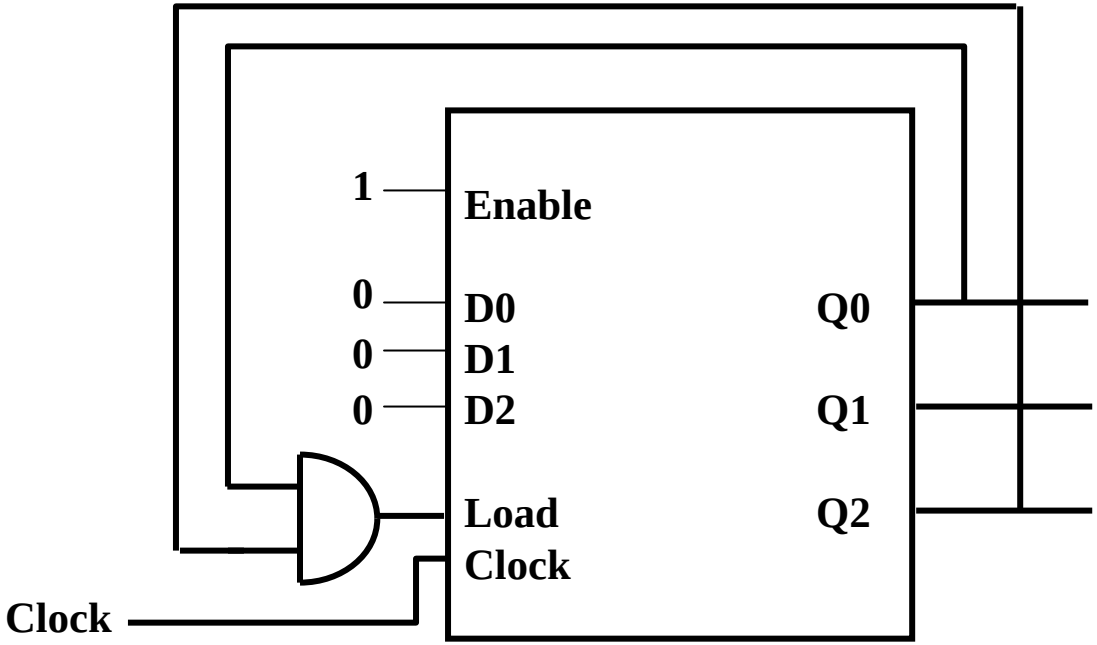
CONTADORES SINCRONOS 2 - UP

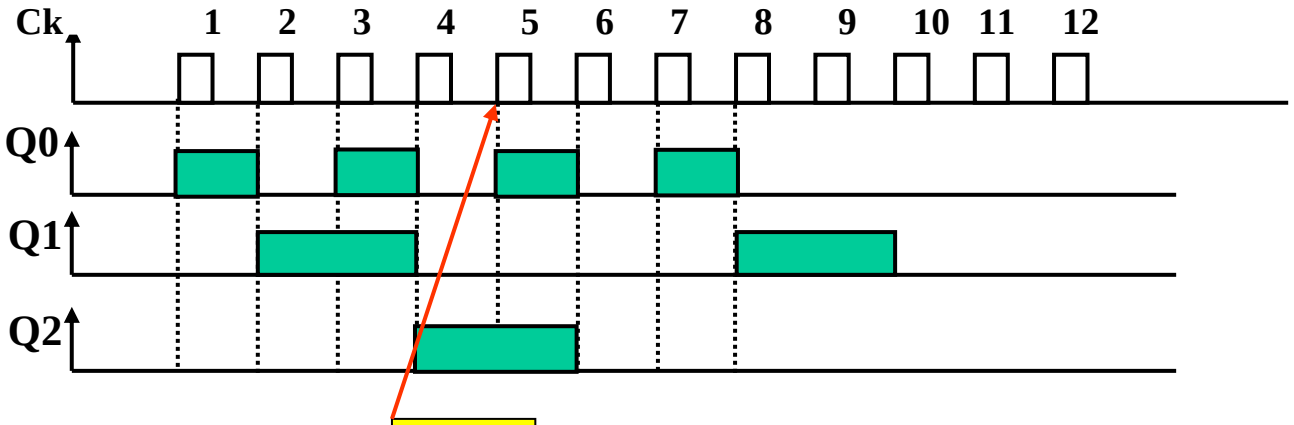


CONTADOR DE CARGA PARALELA



CONTADOR MODULO 6 – M : 6





Cuando comienza el Ck – 5 si bien es cierto que en Load hay un 1, dicha carga no se hace efectiva hasta que ingresa el Ck – 6. Alli los Qi = 0

DISEÑO CONTADOR CRECIENTE M=5

MODULO: N DE ESTADOS POR LOS QUE ATRAVIESA EL CONTADOR

n = CANTIDAD DE FLIP-FLOP NECESARIOS PARA EL DISEÑO

1) CANTIDAD DE FLIP-FLOP n = 3

2) DIAGRAMA DE ESTADOS ⇨ 2exp (n) > M

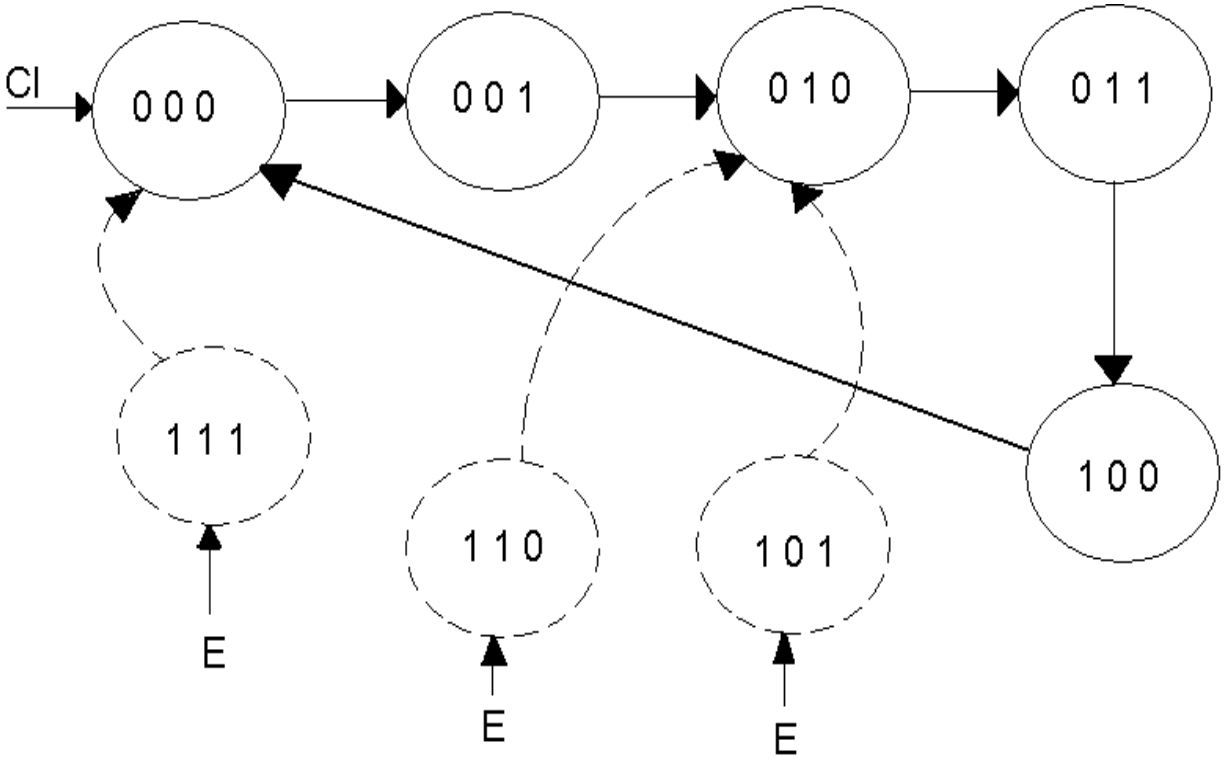


TABLA DE ESTADOS

EST PRES Qt	EST. FUT Qt + 1	ENT	
		J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

TABLA DE ESTADOS

ESTADO PRESENTE			ESTADO FUTURO			ENTRADAS DE LOS FLIP-FLOP						FUNCION DE RESET
Q2	Q1	Q0	Q2	Q1	Q0	J2	K2	J1	K1	J0	K0	fu
0	0	0	0	0	1	0	X	0	X	1	X	0
0	0	1	0	1	0	0	X	1	X	X	1	0
0	1	0	0	1	1	0	X	X	0	1	X	0
0	1	1	1	0	0	1	X	X	1	X	1	0
1	0	0	0	0	0	X	1	0	X	0	X	0
1	0	1	X	X	X	X	X	X	X	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X	1
1	1	1	X	X	X	X	X	X	X	X	X	1

TABLA DE ESTADOS

EST PRES Qt	EST. FUT Qt + 1	ENT	
		J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

IMPLEMENTACION

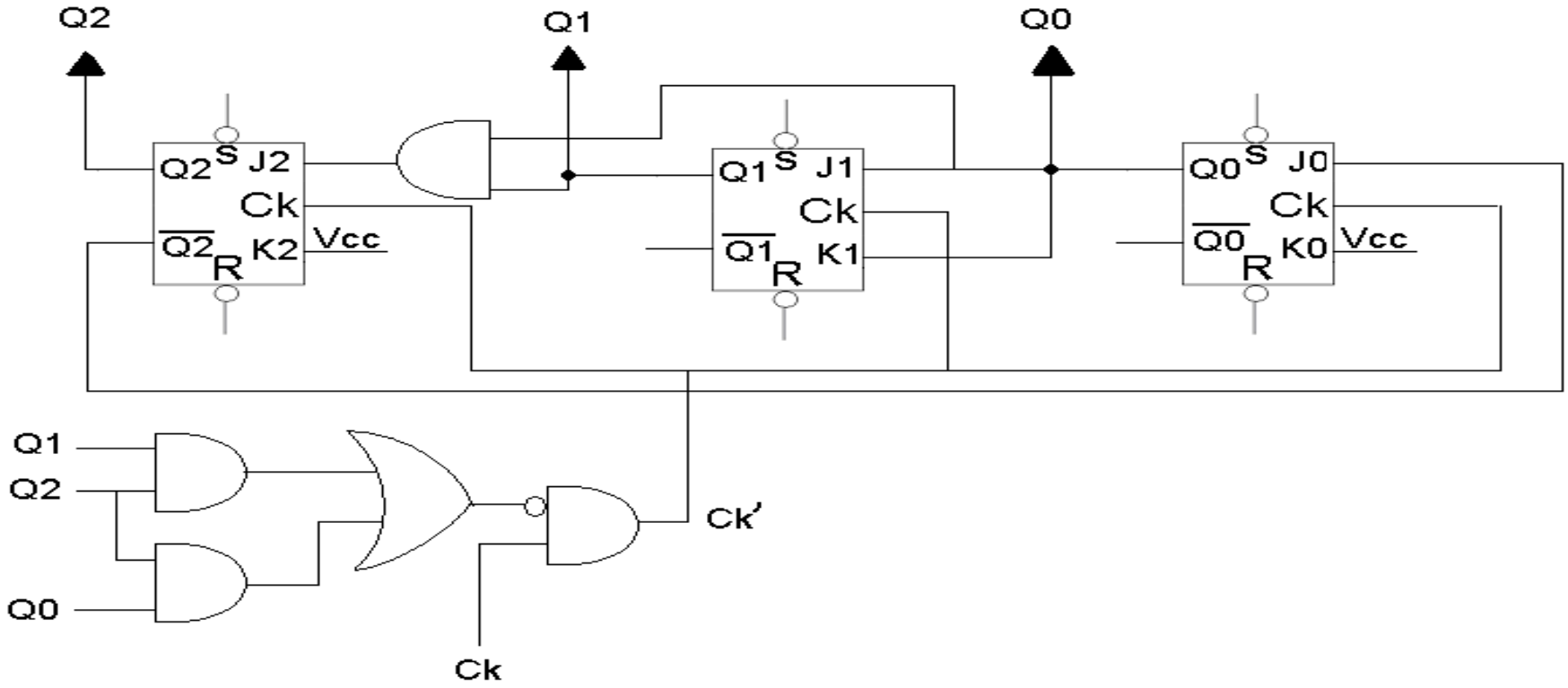
4) SIMPLIFICACION: UN DIAGRAMA PARA CADA ENTRADA JK DE TODOS LOS F/F, MAS UNA FUNCION RESET (fu).

RESULTADO DE LA SIMPLIFICACIÓN

$J2 = Q1\ Q0$ $\quad J1 = Q0$ $\quad J0 = \overline{Q2}$

$K2 = 1$ $\quad K1 = Q0$ $\quad K0 = 1$

$fu = Q1\ Q2 + Q1\ Q2$



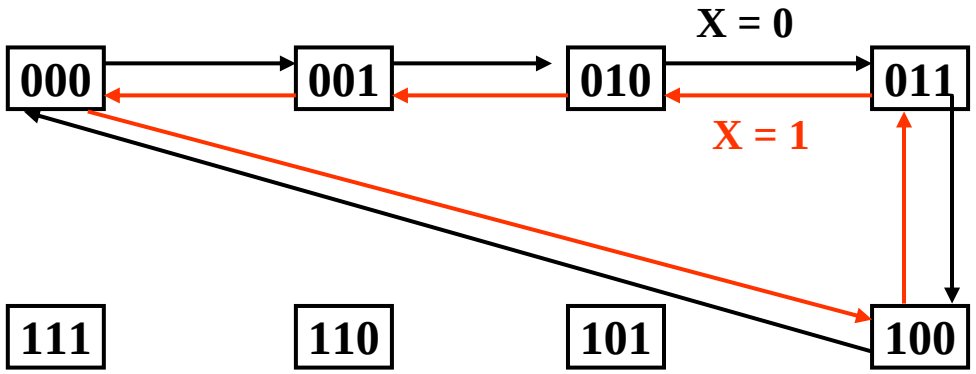
ALTERNATIVA PARA ESTADOS NO USADOS

LA ALTERNATIVA CONSISTE EN FORZAR AL CONTADOR CUANDO ENTRE EN UN ESTADO NO USADO A IR POR EJEMPLO AL ESTADO 000 U OTRO

ESTADO PRESENTE			ESTADO FUTURO			ENTRADAS DE LOS FLIP-FLOP						
Q2	Q1	Q0	Q2	Q1	Q0	J2	K2	J1	K1	J0	K0	
0	0	0	0	0	1	0	X	0	X	1	X	
0	0	1	0	1	0	0	X	1	X	X	1	
0	1	0	0	1	1	0	X	X	0	1	X	
0	1	1	1	0	0	1	X	X	1	X	1	
1	0	0	0	0	0	X	1	0	X	0	X	
1	**0**	**1**	**0**	**0**	**0**	**X**	**1**	**0**	**X**	**X**	**1**	
1	**1**	**0**	**0**	**0**	**0**	**X**	**1**	**X**	**1**	**0**	**X**	
1	**1**	**1**	**0**	**0**	**0**	**X**	**1**	**X**	**1**	**X**	**1**	

CONTADOR UP/DW - M = 5

DIAGRAMA DE ESTADOS



X = 0, UP

X = 1, DW

CONTROL	PRESENTE	FUTURO	ENTRADAS
X	Q_2 Q_1 Q_0	Q_2 Q_1 Q_0	J2 K2 J1 K1 J0 K0
0	0 0 0	0 0 1	0 X 0 X 1 X
0	0 0 1	0 1 0	0 X 1 X X 1
0	0 1 0	0 1 1	0 X X 0 1 X
0	0 1 1	1 0 0	1 X X 1 X 1
0	1 0 0	0 0 0	X 1 0 X 0 X
1	1 0 0	0 1 1	X 1 1 X 1 X
1	0 1 1	0 1 0	0 X X 0 X 1
1	0 1 0	0 0 1	0 X X 1 1 X
1	0 0 1	0 0 0	0 X 0 X 1 X
1	0 0 0	1 0 0	1 X 0 X 0 X
X	X	X	X

CONTADOR UP/DW – MAPA K

XQ2 \ Q1Q0	00	01	11	10
00	0	0	1	0
01	X	X	X	X
11	X	X	X	X
10	1	0	0	0

$$J2 = X\overline{Q1}\,\overline{Q0} + \overline{X}Q1Q0$$

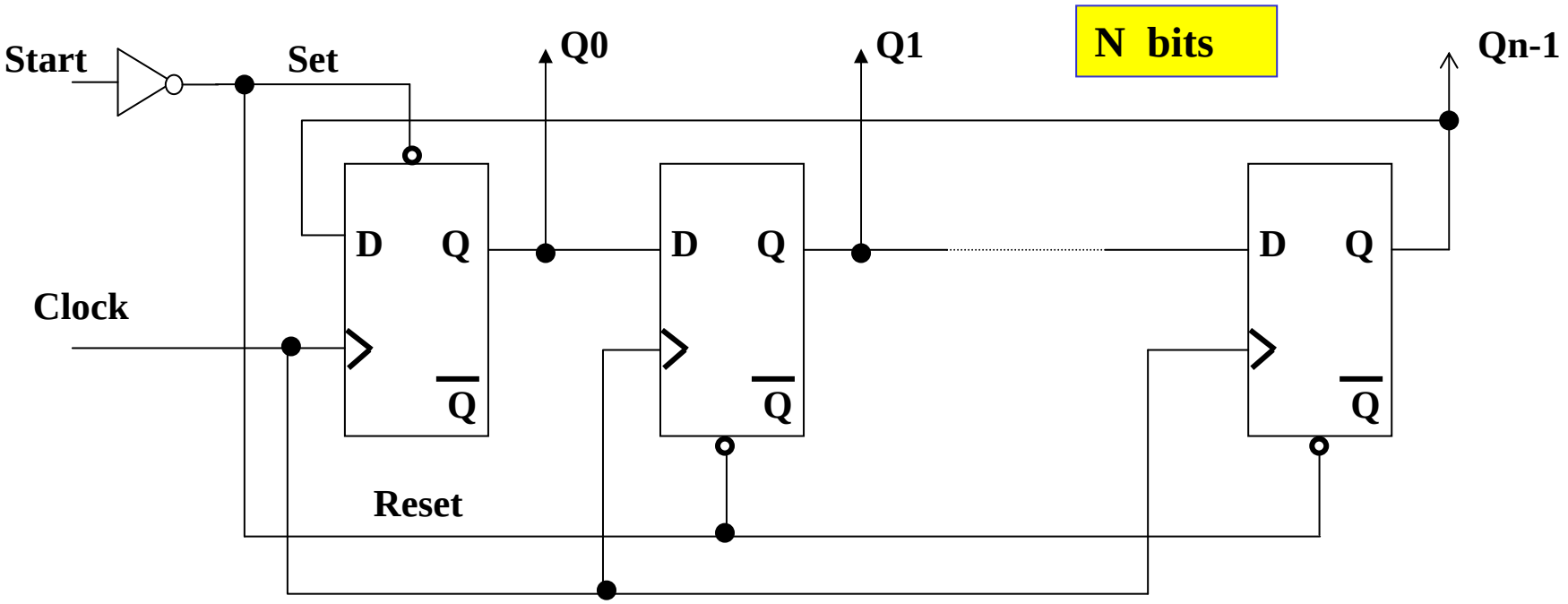
CONSTRUYENDO EL RESTO DE LOS MAPAS, ENCONTRAMOS:

$J1 = XQ2 + \overline{X}Q0$; $J0 = \overline{X}\,\overline{Q2} + XQ2 + Q1$

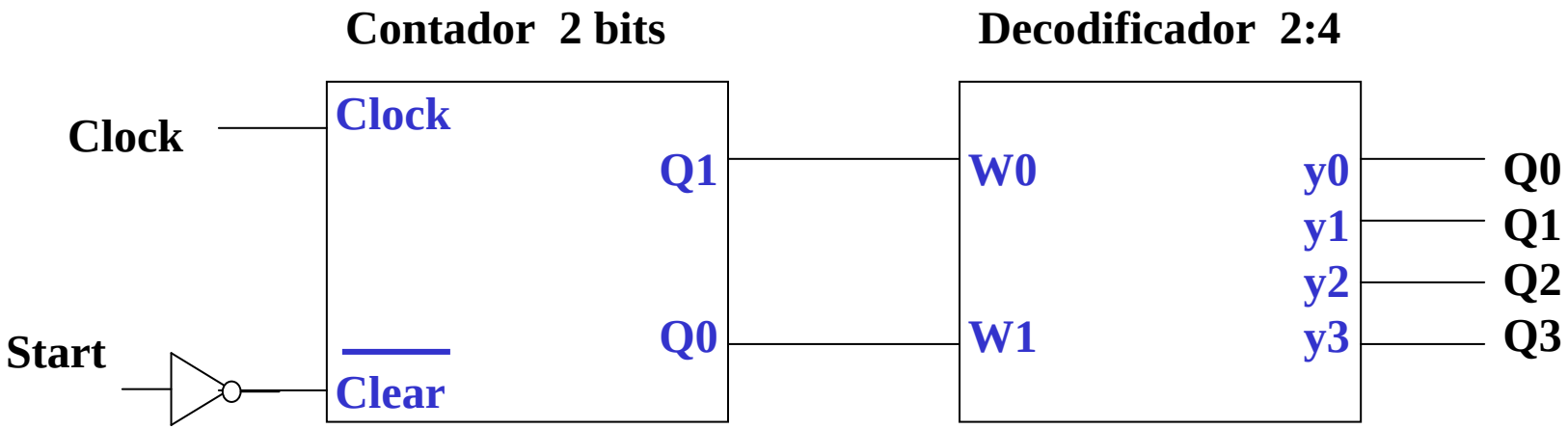
$K2 = 1$; $K1 = X\overline{Q0} + \overline{X}Q0$; $K0 = 1$

SIGUIENTE PASO LA IMPLEMENTACION Y PRUEBA DE FUNCIONAMIENTO

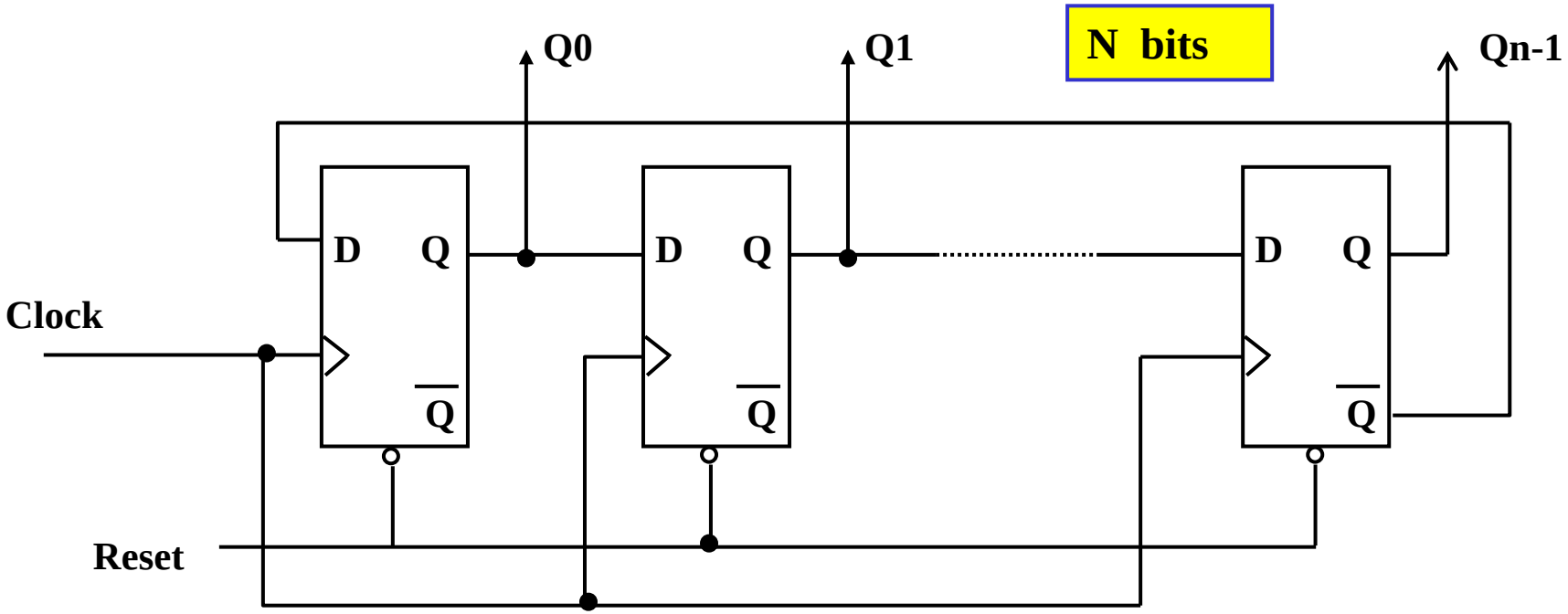
CONTADOR ANILLO



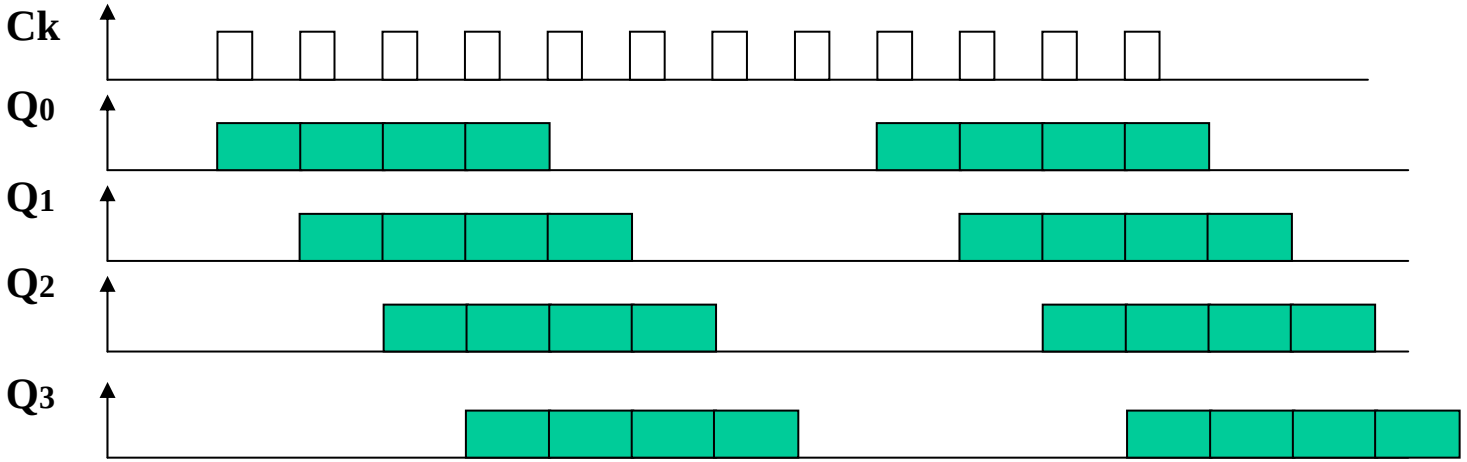
4 bits



CONTADOR JOHNSON

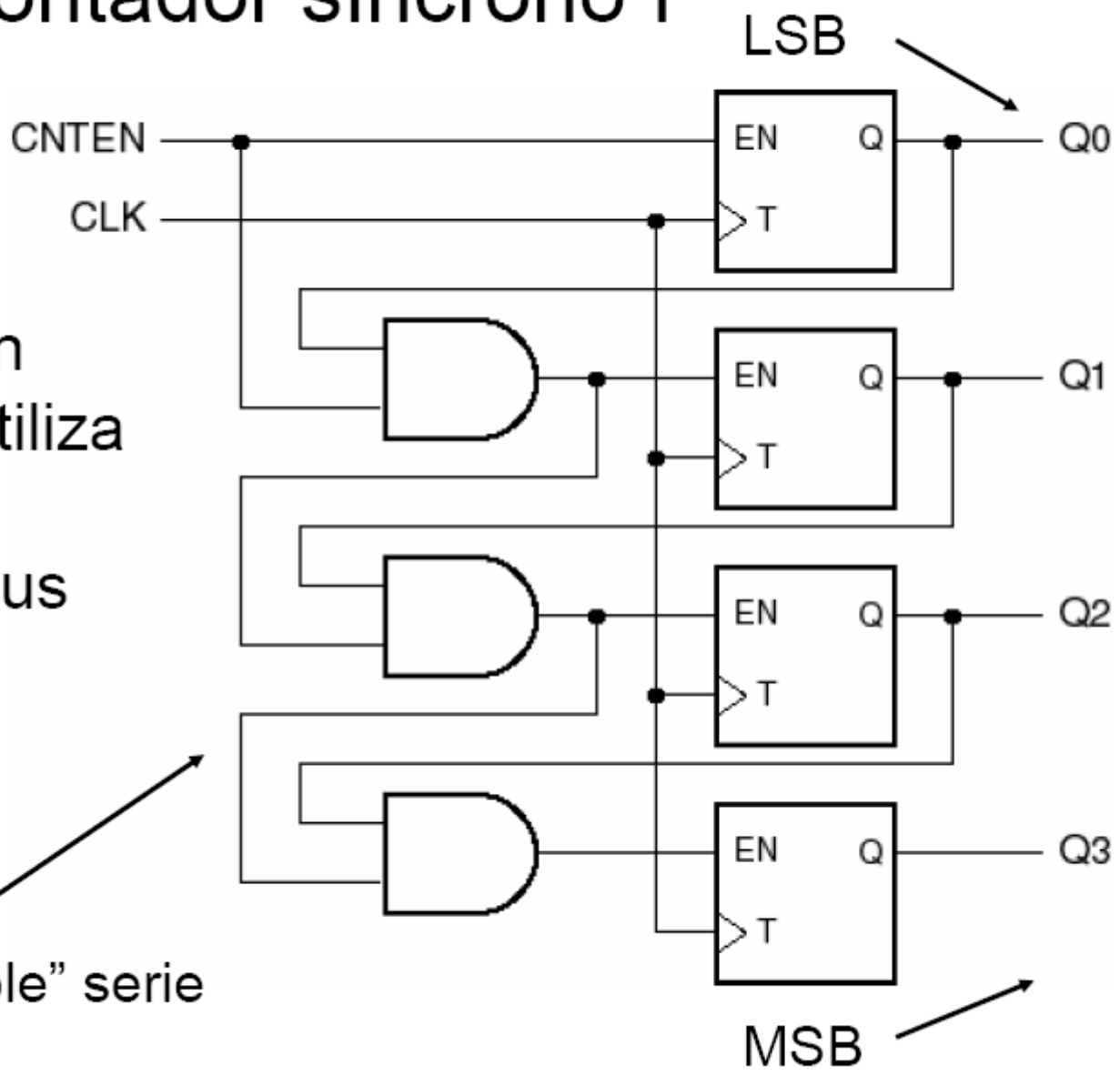


CONTADOR JOHNSON - 4 BITS



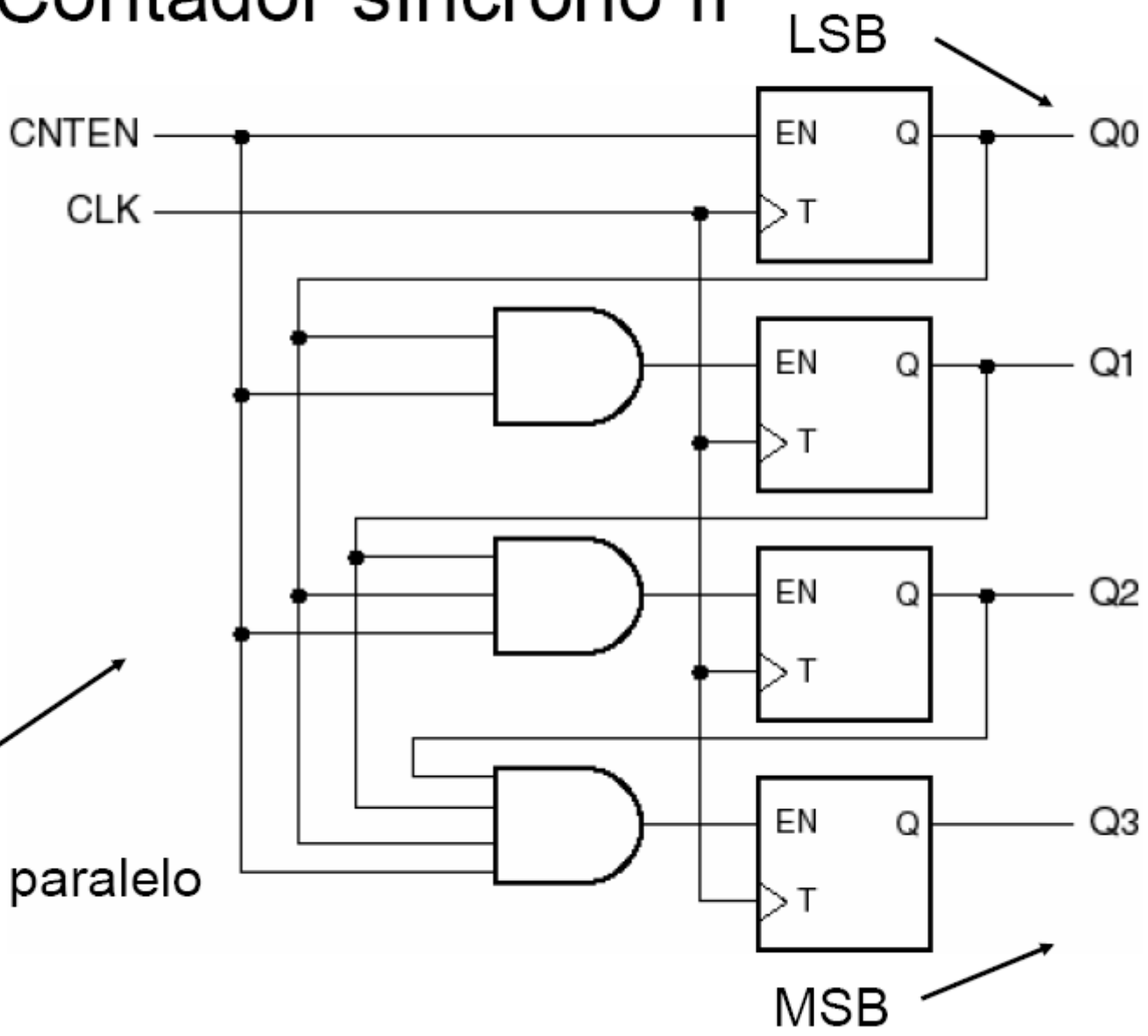
Contador síncrono I

- Se colocan Flip-Flops T en cascada y se utiliza una lógica de "enable" para sus relojes



Contador síncrono II

- Versión con lógica de "enable" en paralelo



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

Figure 7.55 Code for a down-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 ;
END Behavior ;
```

Figure 7.53 Code for a four-bit up-counter

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

ENTITY BCDcount IS
      PORT (  Clock          : IN      STD_LOGIC ;
              Clear, E       : IN      STD_LOGIC ;
              BCD1, BCD0 : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END BCDcount ;

ARCHITECTURE Behavior OF BCDcount IS
BEGIN
      PROCESS ( Clock )
      BEGIN
            IF Clock'EVENT AND Clock = '1' THEN
                  IF Clear = '1' THEN
                        BCD1 <= "0000" ; BCD0 <= "0000" ;
```

… con’t

Figure 7.78a Code for a two-digit BCD counter

```
            ELSIF E = '1' THEN
                IF BCD0 = "1001" THEN
                    BCD0 <= "0000" ;
                    IF BCD1 = "1001" THEN
                        BCD1 <= "0000";
                    ELSE
                        BCD1 <= BCD1 + '1' ;
                    END IF ;
                ELSE
                    BCD0 <= BCD0 + '1' ;
                END IF ;
            END IF ;
        END IF;
    END PROCESS;
END Behavior ;
```

Figure 7.78b Code for a two-digit BCD counter (con't)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
    PORT (  Clear, Clock  : IN      STD_LOGIC ;
            Q             : BUFFER  STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    upcount: PROCESS ( Clock )
    BEGIN
        IF (Clock'EVENT AND Clock = '1') THEN
            IF Clear = '1' THEN
                Q <= "00" ;
            ELSE
                Q <= Q + '1' ;
            END IF ;
        END IF;
    END PROCESS;
END Behavior ;
```

Figure 7.73 Code for a two-bit up-counter with asynchronous reset