CAPÍTULO 4
DISEÑO DE SISTEMAS DIGITALES

DISEÑO DE UN SISTEMA DIGITAL DIVISOR

Diseñar un circuito **Divisor** de dos números binarios sin signo de ***n*** bits.

Dados dos números sin signo **DividendoA** y **DivisorB**, el circuito debe producir dos salidas de ***n*** bits **CocienteQ** y **RestoR**.

Los datos se cargarán en los registros internos, cuando la señal **LdA** sea verdadera. Al finalizar la división se debe generar la salida **Done**.

Utilizaremos en el diseño el método que se utiliza en el proceso de división manual, es decir, el método de desplazamientos y restas sucesivas.

DISEÑO DE UN SISTEMA DIGITAL DIVISOR

A continuación se muestra un ejemplo de una división manual en decimal y en binario:

```
    15
 9 ) 140
     9
    ---
     50
     45
    ---
      5
```

```
                00001111   ← Q
              ----------
B → 1001  )   10001100   ← A
              1001
              ----
               10001
               1001
               ----
                10000
                1001
                ----
                 1110
                 1001
                 ----
                  101   ← R
```

El circuito debe desplazar a la izquierda los dígitos del registro **Dividiendo A** para que vayan ingresando en el registro de desplazamiento **Resto R**.

Inicialmente el Resto **R** debe ser cero. En **R** se van ingresando uno a uno (desplazamiento a la izquierda) los bits de **A** desde el más significativo Para cada desplazamiento, mientras **R** sea menor que el Divisor **B,** cada bit del Cociente **Q** se va llenando con ceros (desplazamiento a la izquierda).

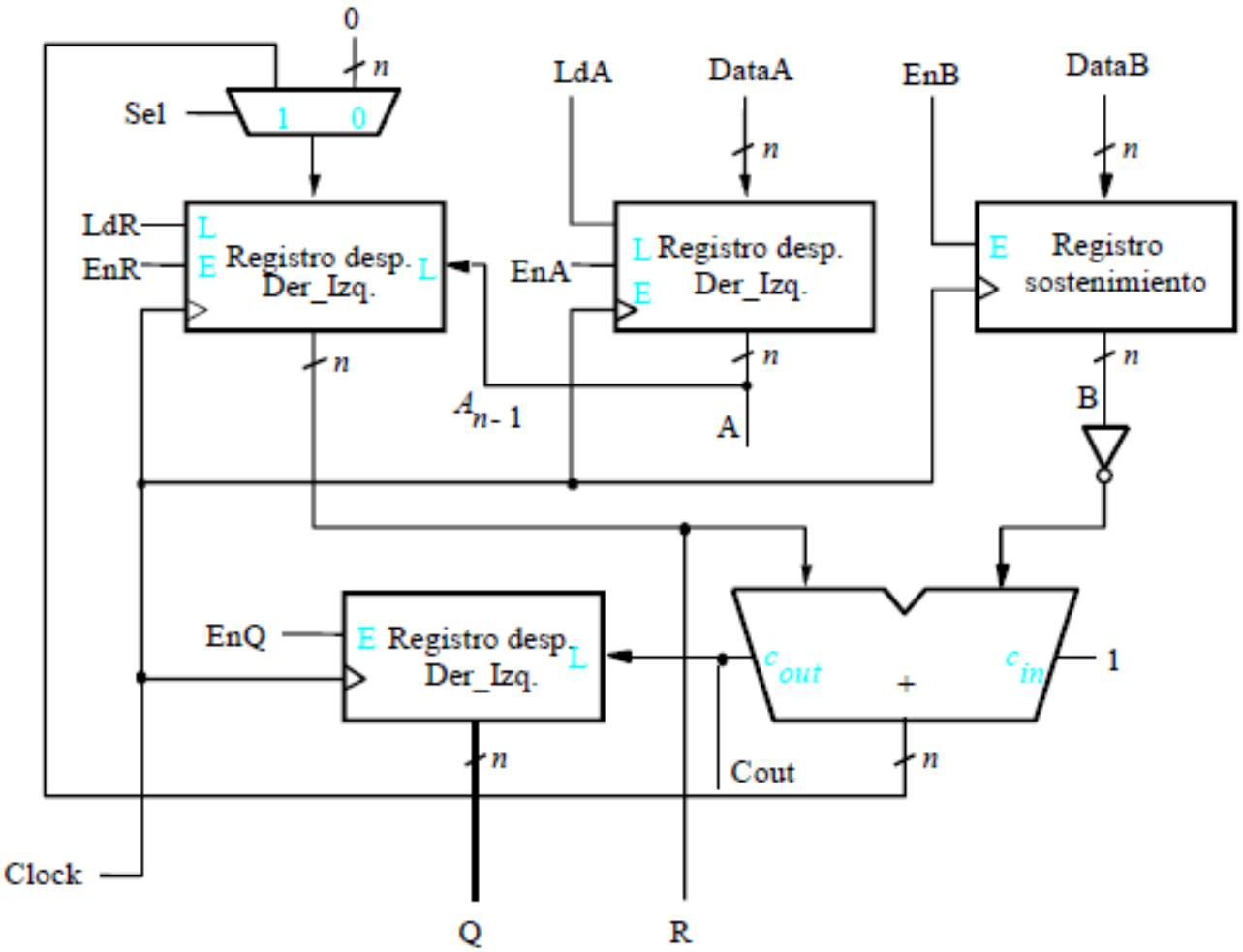
Si **R** es mayor o igual que **B**, en el cociente **Q** debe ingresar un 1 y el resultado de la resta entre el valor actual de R con B (**R – B**), debe convertirse en el nuevo valor del Resto **R.**

El proceso debe repetirse hasta que en **R** hayan ingresado todos los bits de **A**.

La implementación de la resta **R – B** es mediante la suma de **R** con el complemento a **2** de **B**. Si **R>=B** se producirá la señal de acarreo de salida **Cout = 1** que mediante un desplazamiento, se coloca en el registro Q.

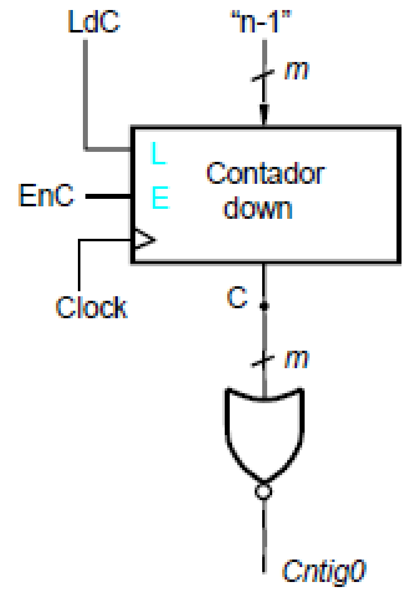
Además, cuando **Cout = 1** el resultado de la resta se cargará en **R** activando la entrada **Sel** de un **mux2a1**.

El circuito **Procesador de Datos** entonces debe tener tres registros de desplazamiento a la izquierda **(A**, **R** y **Q**) y un registro de sostenimiento **B**.



01101010010010100110000101101110

Para verificar que todos los dígitos de **A** han ingresado a **R**, se puede agregar un **Contador Down de modulo n** que se inicia con ***n-1.*** Su propósito es contar el número de desplazamientos realizados luego que se haya activado la entrada **Start.** También se requiere una puerta **NOR** para detectar que el **Contador Down** ha llegado a **0** y el proceso de la división ha terminado.



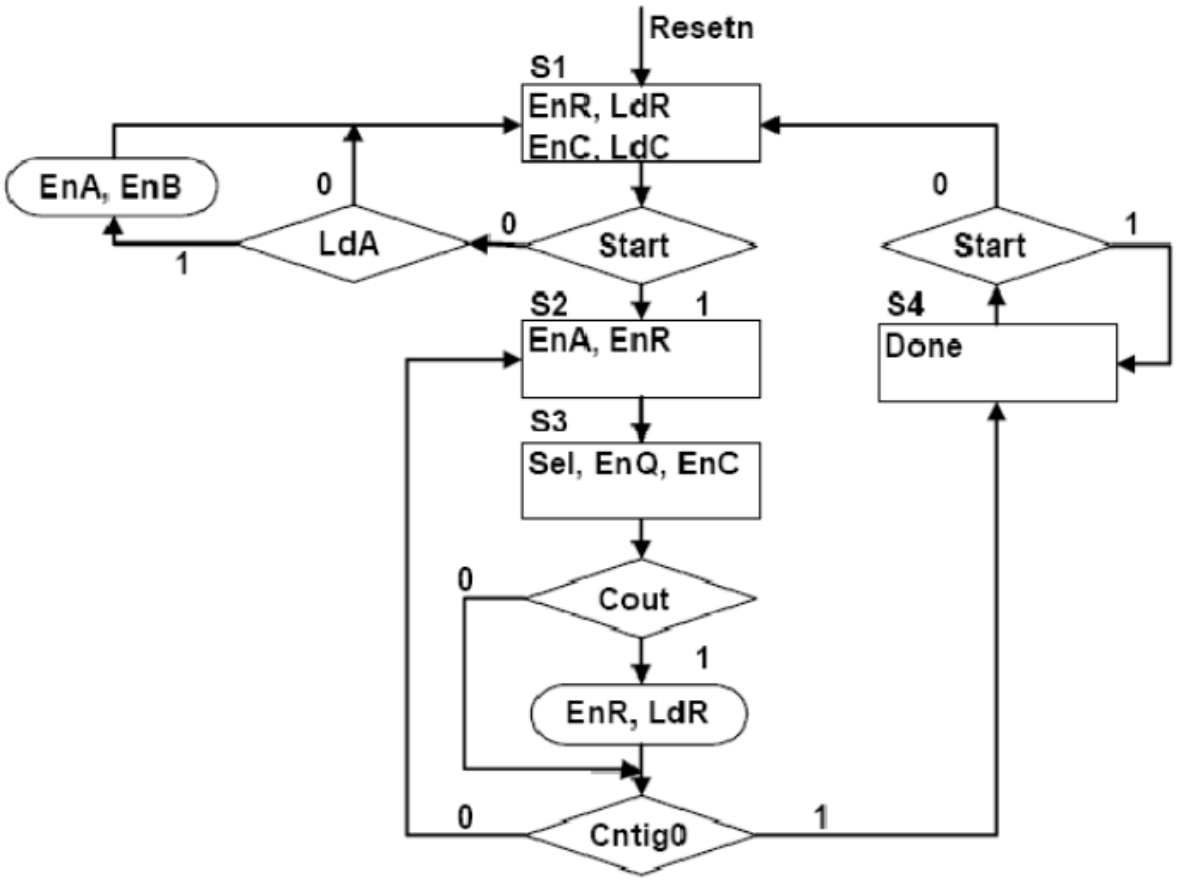
Para realizar la operación de resta mediante la suma con complemento a **2** se necesita un grupo de inversores

R – B = R + not B + (Ci =1).

R inicialmente se carga con **0** y cuando **R** sea mayor o igual que **B** se debe cargar con el resultado de la resta **R – B.** Para ésta operación por lo tanto se requiere un grupo de **mux2a1** para controlar la entrada de **R**.

0110101001100100101010110000101101110

DIAGRAMA ASM DEL CONTROLADOR



En el estado inicial **S1,** se carga el **ContadorDown (LdC** y **EnC**) con el número ***n-1*** y se coloca la señal **Sel** del **mux2a1,** que está en la entrada de **RestoR,** en **0** para cargar **R** con **0 (LdR y EnR**).

Si **Start = 0**, y si externamente se habilita la señal **LdA,** se carga el valor del **Dividiendo A** con **EnA** y del **Divisor B** con **EnB**.

Si **Start = 1**, el circuito **Controlador** va al estado **S2**. En el estado **S2** se desplazan a la izquierda los registros **R** y **A (EnR** y **EnA**). Luego de eso el **Controlador** va al estado **S3** donde se decrementa el **ContadorDown (EnC**), se habilita el registro **Q (EnQ**) para que el valor de **Cout** entre en **Q** en serie y se coloca la señal **Sel** del **mux2a1** en **1**.

Se pregunta si se generó **Cout =1** (R^3B), lo que significaría que hay que cargar **R** con la resta de **R – B**. También se pregunta si el **ContadorDown** llego a **0 (Cntig0** es la salida de la puerta **NOR**).

Si **Cntig0 = 0**, el **Controlador** regresa al estado **S2** y la operación de división continua. Si **Cntig0 = 1**, el circuito va al estado **S4**, genera la salida **Done** y espera que la entrada **Start** se desactive para regresar al estado inicial.

Si el divisor es de ***n*** bits el circuito **Controlador** va a pasar por los estados **S2** y **S3** en un tiempo igual a ***2n*** periodos de **Clock**.

En el estado **S2** se desplazan a la izquierda **R** y **A**.

En el estado **S3**, si **Cout = 1**, se carga el resultado de la resta desde el sumador a **R**.

REDUCCION:

Si las operaciones que se realizan en los estados S2 y S3 se ejecutarán en un solo estado, necesitaríamos solamente ***n*** periodos de **Clock**.
Podemos darnos cuenta que a medida que registro del **Dividiendo A** se vacía, el registro del **Cociente Q** se llena. Entonces, podemos usar un solo registro para **A** y **Q (A/Q)**.
Para combinar estos dos estados en uno solo es necesario cargar el resultado de la suma al registro **R** y al mismo tiempo desplazar el **MSB** de **A** al **LSB** de **R**.

Para realizar las dos operaciones en el mismo ciclo de **Clock**, necesitaremos un flip-flop **D** y un **MUX 2 a 1** adicionales.
El Flip Flop se usara para alimentar los desplazamientos del Resto **R** que vienen del Dividendo **A.** La salida del **FF D** esta indicada como **r0** .

La idea es tener en **R** el valor del resto actual (del último desplazamiento) y al mismo tiempo al concatenar **R** con el bit **ro**, poder tener el próximo valor de R y saber con anticipación si debemos cargar la resta o seguir desplazando.

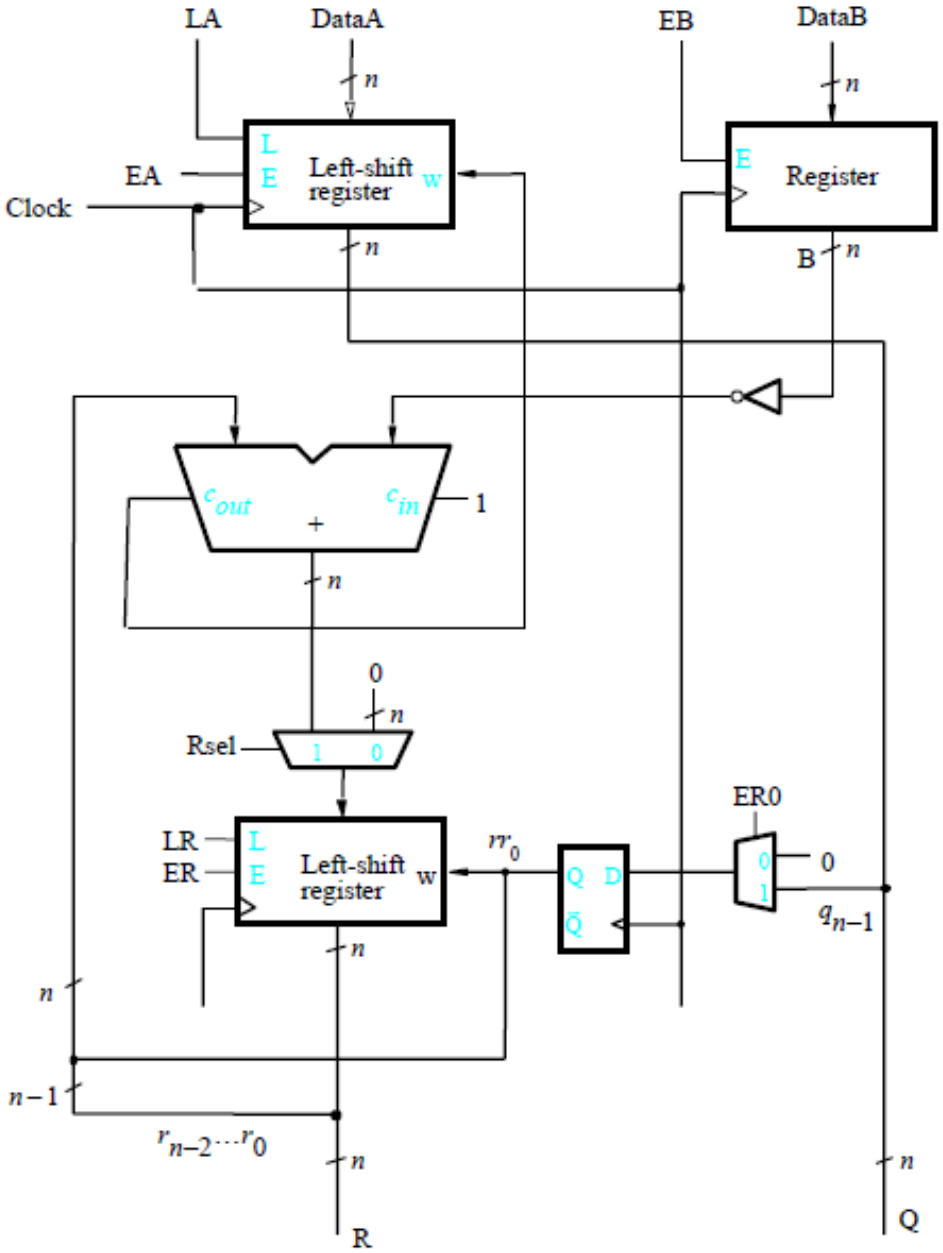
Ejemplo:

Si se quiere dividir 13 para 5:

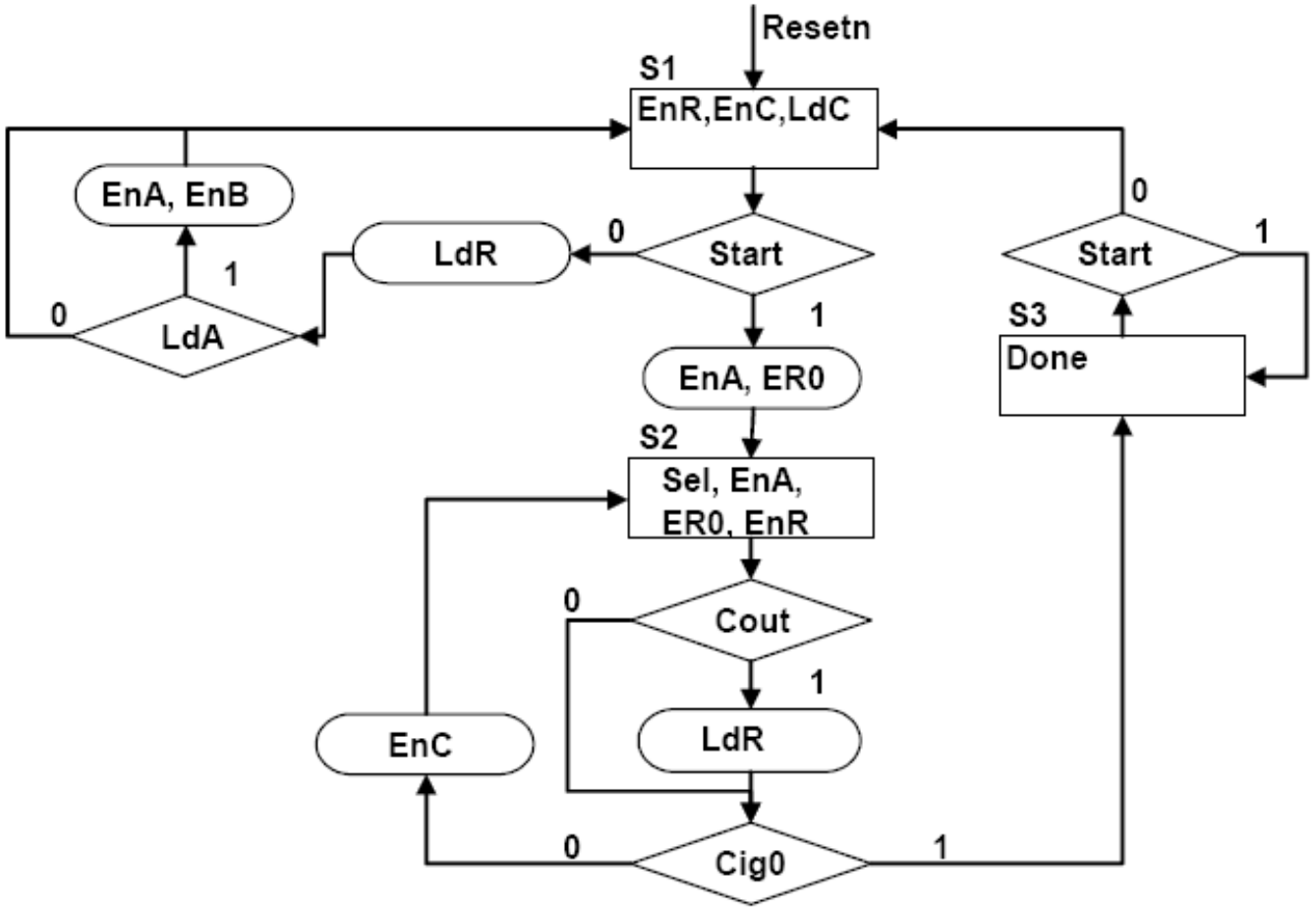
Divisor B	Cociente/dividendo Q/A	Resto R	Salida FFD ro	R ≥ B Cout	Contador
0101	1101	0000	0	0	3
	1010	0000	1	0	3
	0100	0001	1	0	2
	1000	0011	0	1	1
	0001	0001	1	0	0
	0010	0011			

Como podemos notar, para efectos de desplazamiento y de estimar el valor del residuo, usamos únicamente el registro R.

Pero para comparar **R** con **B** y por ende determinar si cargamos su resta como nuevo valor de R, usamos la concatenación de R con ro.


LA
DataA
EB
DataB
n
L
E
Left-shift register
w
EA
Clock
E
Register
B
n
c_{out}
c_{in}
1
+
0
Rsel
1
0
ER0
LR
ER
r'_0
Q
D
$\bar{Q}$
0
q_{n-1}
n-1
$r_{n-2} \cdots r_0$
R
Q

El **Diagrama ASM** del circuito **Controlador** puede ser estructurado de la siguiente manera :



En el estado **S1,** mientras **Start = 0 y se activa** la entrada externa **LdA,** se carga el **Dividiendo A** y el **Divisor B** generando las salidas condicionales **EnA** y **EnB**.

0110000101110011011000010110111001111010011000011

En el mismo estado, el registro **Resto R** y el flip-flop **D** se cargan con **0** respectivamente ya que las entradas de selección de los **mux2a1 Sel** y **ER0** no están activadas. También se carga el **ContadorDown** con el valor **n-1**. Note que ahora el registro R solo se carga con cero si start=0, caso contrario el ya comienza a desplazar para evitar el estado adicional.

Cuando se activa la entrada externa **Start = 1**, entonces, estando todavía en el estado **S1,** se generan las salidas condicionales **EnA** y **ER0**.
Con **EnA** se desplaza el registro **A/Q**. La señal **ER0** habilita el **mux2a1** que selecciona el **MSB** del registro **A/Q** y lo coloca en la entrada del flipflop **D**.

Cuando llegue el flanco de subida del **Clock** se produce el desplazamiento de **A/Q** y la carga de **D**. Note que en este estado **S2,** al sumador le llega la concatenación de **n-1** bits de **R** (desde Rn-2 hasta R0) con el bit **ro** que es la salida del **D**. Entonces **ro** queda como el LSB del **R** modificado.

El sumador realiza la resta **R - B**, y genera la salida **Cout** que entra como **LSB** del **A/Q** en el momento del siguiente desplazamiento. Esto es porque se usa un registro unificado A/Q para el **Dividiendo A** y el **Cociente Q.**

Como antes, dentro del estado **S2,** el **Resto R** se puede desplazar a la izquierda entrando la salida del flip-flop **D**, si **Cout =0**, o puede caragarse con el resultado de la resta **R - B**, si **Cout =1.** Para esto la señale **Rsel** está activada.

La señal **Cig0** es generada por una puerta **NOR** conectada con las salidas del **ContadorDown**.

Cuando el contador que estaba cargado con **n -1** llegue a **0**, **Cig0 = 1** y el **Controlador** pasa al estado **S3**.

En el estado **S3,** se genera la salida **Done**, se espera que **Start** se desactive y se regresa al estado inicial.

IMPLEMENTACION CON VHDL Vamos a detallar únicamente componentes que no se han usado antes.

Código VHDL del circuito ContadorDown de módulo m.

```
library ieee;
use ieee.std_logic_1164.all;

entity contador_down is
    generic (m : integer :=16);
    port (  Resetn, Clock   : in std_logic;
            En, Ld          : in std_logic;
            Q               : buffer integer range 0 to m-1);
end contador_down;

architecture comportamiento of contador_down is

begin
    process(Resetn, Clock)
    begin
        If resetn='0' then Q <= 0;
        elsif (Clock'event and clock ='1') then
            if En ='1' then
                if Ld ='1' then Q <= m-1;
                else Q <= Q-1; end if;
            end if;
        end if;
    end process;

end comportamiento;
```


01101010011001010110000101101110

Código VHDL del MUX 2 a 1 con Flip Flop tipo D

```
library ieee;
use ieee.std_logic_1164.all;

entity mux_dff is
    port (  I1, I0      : in std_logic;
            sel, Clock  : in std_logic;
            Q           : out std_logic);
end mux_dff;

architecture comportamiento of mux_dff is
    signal D : std_logic;

begin
    with sel select
    D <=    I1 when '1',
            I0 when others;
    process
    begin
        wait until (Clock'event and clock ='1');
        Q <= D;
    end process;

end comportamiento;
```

Código VHDL del circuito Divisor ORIGINAL

En la librería **work** ya tenemos todos los componentes necesarios para el **Procesador de Datos** compilados y simulados.
Todos ellos deben formar parte del ***package*.componentes**.
De esta manera podemos usar estos componentes y reducir significativamente el trabajo de elaboración del código **VHDL** para el **Procesador de Datos**.
Este paquete también debe estar en la carpeta del circuito **Divisor** compilado antes de que se compile el código **VHDL** del programa completo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.componentes.all;

entity divisor1 is
    generic (n: integer := 8);
    port (  Clock           : in std_logic;
            Resetn          : in std_logic;
            Start, LdA      : in std_logic;
            DataA, DataB    : in std_logic_vector (n-1 downto 0);
            R, Q            : buffer std_logic_vector (n-1 downto 0);
            Done            : out std_logic);
end divisor1;
```

Procedemos a declarar la arquitectura del programa final detallando como SIGNAL a todas las variables intermedias utilizadas.
Luego se declaran las transiciones del diagrama ASM.

```
architecture comportamiento of divisor1 is
    type estado is (S1, S2, S3, S4);
    signal y : estado;
    signal Zero, Cout, Cig0  : std_logic;
    signal EnA, EnB, Sel, LdR, EnR, LdQ, EnQ, LdC, EnC : std_logic;
    signal A, B, DataR, nZero : std_logic_vector (n-1 downto 0);
    signal Sum : std_logic_vector (n downto 0);
    signal Count: integer range 0 to n-1;

begin
    MSS_transiciones: process (Resetn, Clock)
    begin
        if Resetn ='0' then y <= S1;
        elsif (Clock'event and Clock ='1') then
            case y is
                when S1 => if Start ='0' then y <= S1; else y <= S2; end if;
                when S2 => y <= S3;
                when S3 => if Cig0  ='0' then y <= S2; else y <= S4; end if;
                when S4 => if Start ='1' then y <= S4; else y <= S1; end if;
            end case;
        end if;
    end process;
```


01101010010011000010110111010

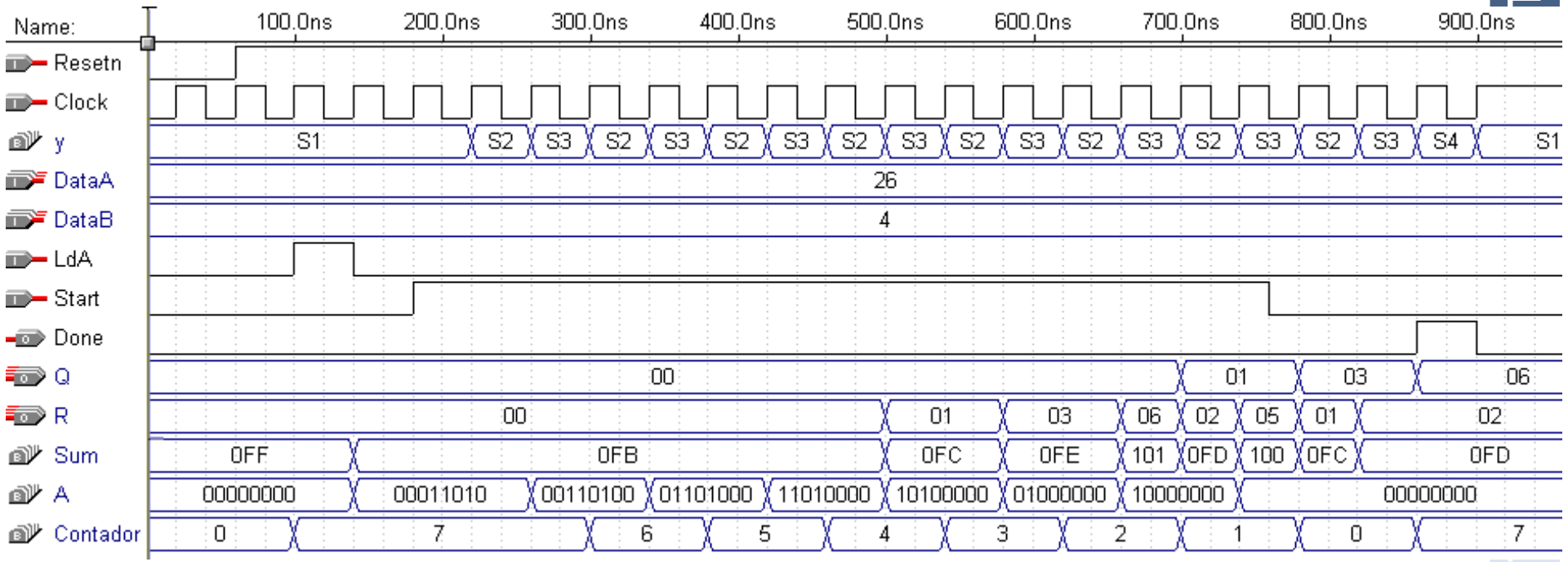
Aquí se detallan las salidas de la MSS controladora

```
MSS_salidas: process(Start, y, Cout, Cig0)
begin
    LdR <='0'; EnR <='0'; EnB <='0'; EnQ<='0';
    LdC <='0'; EnC <='0'; EnA <='0'; LdQ<='0';
    EnQ <='0'; Done <='0'; Sel <='0';
    case y is
        when S1 => LdC <='1'; EnC <='1'; EnR <='1'; LdR <='1';
            EnQ <='1'; LdQ<='1';
            if Start ='0' and LdA ='1' then EnA <='1'; EnB <='1';
            end if;
        when S2 => EnA <='1'; EnR <='1';
        when S3=> Sel <='1'; EnQ <='1'; EnC <='1';
            if Cout ='1' then EnR <='1'; LdR <='1'; end if;
        when S4 => Done <='1';
    end case;
end process;
```

```
-- Procesador de Datos
   Zero <='0';
   nZero <= (OTHERS =>'0');

   RestoR: registro_d_i generic map (n => n)
       port map (Resetn, Clock, EnR, LdR, A(n-1), DataR, R);

   DivisorB: registro_sost generic map (n => n)
       port map (Resetn, Clock, EnB, DataB, B);

   DividiendoA: registro_d_i generic map (n => n)
       port map (Resetn, Clock, EnA, LdA, Zero, DataA, A);

   CocienteQ: registro_d_i generic map (n => n)
       port map (Resetn, Clock, EnQ, LdQ, Cout, nZero, Q);

   Sum <= R + not B + '1' ;

   DataR <= (others =>'0') when Sel ='0' else Sum;

   ContadorDown: contador_down generic map (m => n)
   port map (Resetn, Clock, EnC, LdC, Count);

   Puerta_NOR: Cig0 <='1' when Count = 0  else '0';

   Cout <= Sum (n);

end comportamiento;
```

Finalmente se detalla el procesador de datos usando la descripción estructural e instanciando uno a unc los componentes detallados.

DIAGRAMA DE TIEMPO DEL CIRCUITO DIVISOR ORIGINAL



Código VHDL del circuito Divisor de dos palabras de *n* bits mejorado.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.componentes.all;

entity divisor is
    generic (n: integer := 8);
    port (  Clock           : in std_logic;
            Resetn          : in std_logic;
            Start, LdA      : in std_logic;
            DataA, DataB    : in std_logic_vector (n-1 downto 0);
            R, Q            : buffer std_logic_vector (n-1 downto 0);
            Done            : out std_logic);
end divisor;

architecture comportamiento of divisor is
    type estado is (S1, S2, S3);
    signal y : estado;
    signal Zero, Cout, Cig0 : std_logic;
    signal EnA, EnB, Rsel, LdR, EnR, ER0, LdC, EnC, r0 : std_logic;
    signal A, B, DataR : std_logic_vector (n-1 downto 0);
    signal Sum : std_logic_vector (n downto 0);
    signal Count : integer range 0 to n-1;
```


```
begin
    MSS_transiciones: process (Resetn, Clock)
    begin
        if Resetn ='0' then y <= S1;
        elsif (Clock'event and Clock ='1') then
            case y is
                when S1 => if Start ='0' then y <= S1; else y <= S2; end if;
                when S2 => if Cig0  ='0' then y <= S2; else y <= S3; end if;
                when S3 => if Start ='1' then y <= S3; else y <= S1; end if;
            end case;
        end if;
    end process;

    MSS_salidas: process(Start, y, Cout, Cig0)
    begin
        LdR <='0'; EnR <='0'; ER0 <='0';
        LdC <='0'; EnC <='0'; EnA <='0';
        Done <='0'; Rsel <='0';
        EnB <= EnA;
        case y is
            when S1 => LdC <='1'; EnC <='1'; EnR <='1';
                if Start ='0' then LdR <='1';
                    if LdA ='1' then EnA <='1'; else EnA <='0'; end if;
                else EnA <='1'; ER0 <='1';
                end if;
            when S2 => Rsel <='1'; EnR <='1'; ER0 <='1'; EnA <='1';
                if Cout ='1' then LdR <='1'; else LdR <='0'; end if;
                if Cig0 ='0' then EnC <='1'; else EnC <='0'; end if;
            when S3 => Done <='1';
        end case;
    end process;
```


01101010010010100110000101101110

```
-- Procesador de Datos
    Zero <='0';
    DivisorB: registro_sost generic map (n => n)
        port map (Resetn, Clock, DataB, EnB, B);

    RestoR: registro_d_i generic map (n => n)
        port map (Resetn, Clock, DataR, LdR, EnR, r0, R);

    Mux_FFD: mux_dff port map (A(n-1), Zero, ER0, Clock, r0);

    DividiendoA: registro_d_i generic map (n => n)
        port map (Resetn, Clock, DataA, LdA, EnA, Cout, A);
    Q <= A ;

    ContadorDown: contador_down generic map (modulo => n)
        port map (Clock, LdC, EnC, Count);

    Puerta_NOR: Cig0 <='1' when Count = 0 else '0';

    Sum <= R & r0 + not B + 1 ;
    Cout <= Sum (n);
    DataR <= (others =>'0') when Rsel ='0' else Sum;

end comportamiento;
```


01101010010011001010110000101101110

La salida del **mux2a1** que necesitamos para la entrada de **R** esta representado por la señal **DataR**.