UNIVERSIDAD NEBRIJA

Escuela Politécnica Superior,
Grado en Informática

Diseño Automático de Sistemas

Prof: Ignacio Azcárate Ramos

Práctica 2:

Generador de reloj sincrono. Registro de desplazamiento con carga en paralelo.

Alvaro Terron
Gonzalo Vázquez
José Maria Fernández

UNIVERSIDAD NEBRIJA

1. Introducción

1.1 Presentación

En esta práctica se van a completar dos módulos independientes: un generador de reloj síncrono y un registro de desplazamiento parametrizable. En el contexto práctico se puede observar un caso en el que ambos módulos son utilizados en el diseño de una FPGA.

1.2 Funcionamiento

A partir del reloj de sistema (10 MHz), realizar un divisor de frecuencia que genere tres frecuencias síncronas inferiores. Para esto, usa un contador de ciclos de una estructura igual a la usada en la práctica anterior. El registro de desplazamiento parametrizable va a actuar como un registro de desplazamiento de entrada en serie o en paralelo a elegir, y salida simultánea en serie y paralelo. Ambos funcionamientos se explican en detalle más adelante.

Esto quiere decir, el registro tomará una entrada de un bit en serie, y una entrada de un vector de tamaño *g_N*. De acuerdo a los bits de control (*s0, s1*) se cargará el bit de entrada rotando

1.3 Material provisto

En la carpeta correspondiente de la práctica 2 del campus virtual se encuentra un fichero que contiene el código a completar para el registro de desplazamiento *shift_register.vhd* y del divisor de frecuencia *divisor_3.vhd*, así como su correspondiente testbench *tb_shift_register.vhd* y *tb_divisor_3.vhd*.

1.4 Evaluación y entrega

Estas prácticas van a ser realizadas en grupos. Estos se mantendrán durante el transcurso de las cinco prácticas. La entrega se debe realizar **dos semanas después de acabar el laboratorio**.

En esta entrega se deben incluir:

- Los ficheros .vhd modificados o creados de acuerdo con lo pedido. Incluyendo los testbench que se hayan realizado.

- Un breve informe en el que se describa brevemente la solución implementada. Se pueden incluir los diagramas que hayáis podido dibujar para comprender el diseño. También imágenes de la visualización mediante simulación de las señales de interés, descripción del test realizado y el porqué de las pruebas propuestas en él, indicando brevemente porque reflejan el funcionamiento correcto del sistema.

Este hito se corresponde con un **30%** del valor total de la nota de prácticas.

2. Contexto útil: Caso de uso

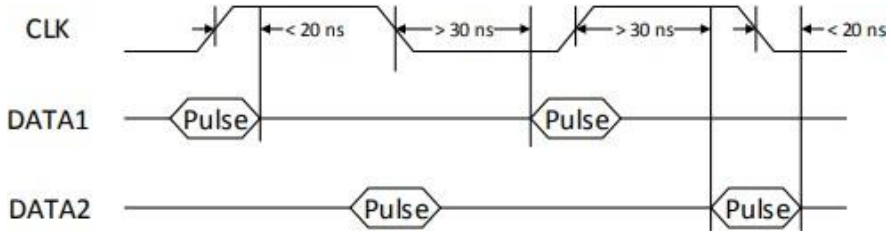
2.1 Introducción

Importante: Todo este Capítulo 2 solo incluye el contexto de un caso de uso real de los dos módulos que se van a implementar. La información aquí incluida no es relevante para el diseño que vais a implementar. De hecho podéis saltaros su lectura, si así lo preferís.

En algunas FPGAs ofrecidas por Xilinx se puede encontrar periféricos de audio, y memoria integrada. Por ejemplo en la, ya descatalogada, placa Nexys 4 DDR se incorpora un micrófono ADMP421, de tipo MEMS (Micro-Electro-Mechanical Systems). La interfaz de este módulo es la siguiente:

Esta interfaz consta de CLK (en el puerto J5 de la FPGA), como la que debemos generar en nuestro diseño. El puerto DATA (en el puerto H5 de la FPGA) de los datos que vienen del micrófono que es un stream de datos de un bit de ancho con modulación PDM (Pulse Density Modulation). Por último, tenemos el puerto de L/R SEL(en el puerto F5 de la FPGA) que nos permite elegir el canal (izquierdo o derecho).

La señal L/R decide si muestreamos en el flanco de subida o en el flanco de bajada.

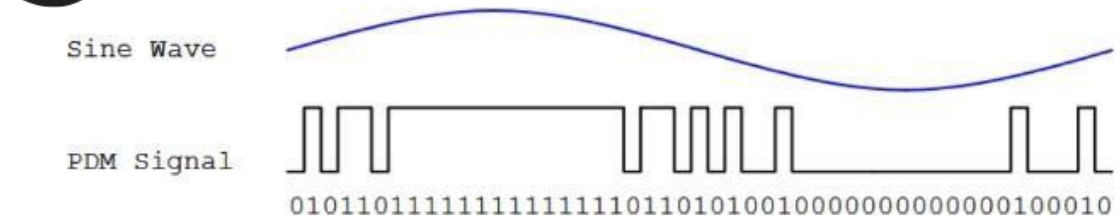


Este micrófono solo acepta como entrada en la señal de reloj frecuencias desde 1 MHz a 3.3 MHz. Además, a mayor frecuencia, mayor calidad de audio. Por lo tanto, si quiere integrar la lectura de audio con este periférico con otros sistemas a una frecuencia superior, se necesita implementar un generador de reloj síncrono. Uno de los módulos a implementar en esta práctica.

2.1 Pulse Density Modulation (PDM)

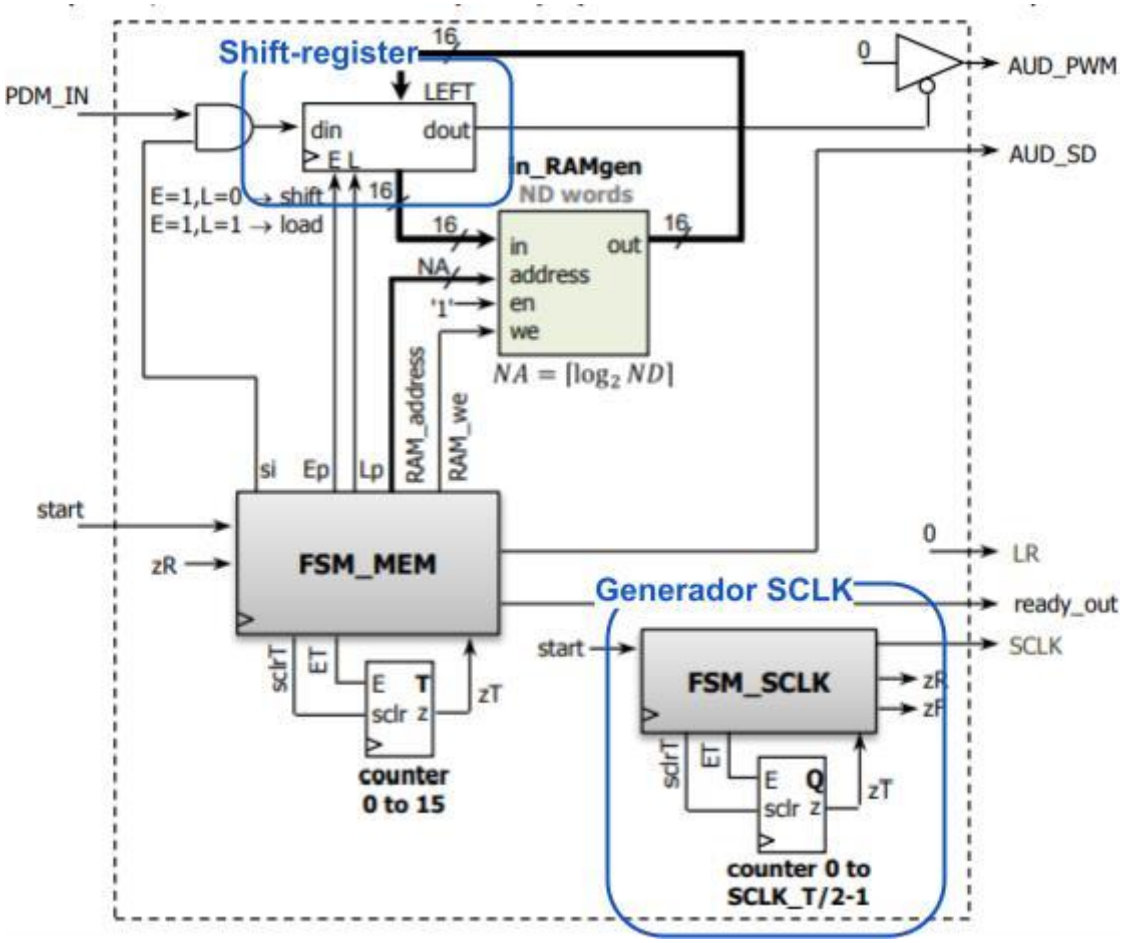
Como se ha indicado se trata de un micrófono MEMS. Estos suelen utilizar una modulación PDM para representar el audio grabado. Este método es popular en sistemas portables pues un canal de audio necesita un único cable para ser transmitido.

En un bitstream PDM, se transmite la amplitud de una onda mediante la frecuencia de pulsos positivos (o pulsos negativos) enviados por el canal. De forma que un envío continuo de señal “1” se corresponde con el valor máximo de amplitud, mientras que un envío constante de ‘0’ se corresponde con el mínimo. En la siguiente figura se observa la representación PDM de una función senoidal.



Para el registro de esta señal en la propia FPGA, así como para su futura conversión en otros formatos como PCM (Pulse Code Modulation), es mucho más cómodo procesar y almacenar los valores recibidos por el micrófono como palabras de un tamaño dado de bits. Aquí entra en juego el segundo módulo a implementar: un registro de desplazamiento. Encargado de la conversión de la señal en serie recibida en una señal en paralelo.

Estos dos módulos, junto a varios más como un controlador de memoria, por ejemplo. Pueden ser usados para programar una FPGA Nexys 4 DDR para el registro integrado de lectura de audio. En el siguiente esquema se muestran estos dos componentes en un entorno conectado a la memoria de la FPGA y a un módulo controlador de memoria.



Podéis encontrar más información de la placa Nexys 4 DDR, y de sus periféricos, incluido el micrófono, en el siguiente enlace: https://docs.rs-online.com/2bb6/0900766b81592f18.pdf

3. Generador de reloj síncrono

3.1 Introducción

Como se ha visto en el caso de uso, hay diseños en los que debemos generar un reloj a una frecuencia inferior al del sistema. Un reloj síncrono derivado tiene sus flancos de subida y de bajada a la vez que las subidas y bajadas del reloj principal, pero al tener menor frecuencia sus ciclos son de una duración superior. Para resolver estos casos se usan los divisores de frecuencia, un divisor de frecuencia es un circuito secuencial que, a partir de una señal de entrada, de frecuencia F, obtiene otra de frecuencia F/N, siendo N el factor de división. Un divisor de frecuencia se realiza mediante contadores con salida de fin de cuenta (FDC o EOC).

Empleando contadores con salida de fin de cuenta (señal que se activa cuando ha llegado al último estado de cuenta durante un ciclo de reloj) pueden realizarse divisores de frecuencia que obtengan señales con una frecuencia que sea un factor de la del reloj del circuito: el factor de división N es el módulo del contador, así pues, un contador de módulo 3 generaría una señal que divida la frecuencia de entrada entre 3. Por lo general con los contadores se puede generar el reloj completo o simplemente una habilitación del reloj derivado, señal que tiene la frecuencia del reloj derivado, pero se conforma por un único pulso del reloj principal.

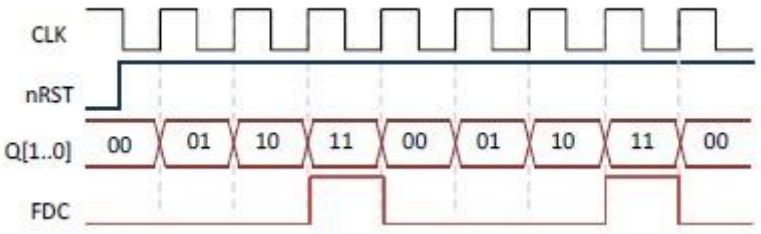
3.2 Clock enable

Al usar en un diseño varios dominios de reloj distintos se pueden dar una serie de errores de metaestabilidad. Por esto, es fundamental que en los circuitos que usan un reloj derivado **nunca** se use esta señal para sincronizar, **siempre utilizar el reloj del sistema**, salvo en casos muy específicos. Se usa en cambio la señal del reloj principal y una señal auxiliar que marca que en este ciclo se ha levantado el reloj derivado (*SCLK_rise* en la figura). Por ello muchas veces es más optimo generar una señal de habilitación de reloj en lugar del reloj completo, la mayoría de registro de las FPGAs tienen una entrada de habilitación de reloj, lo cual no supone un gasto de lógica adicional.

NO
```
process (clk, rst_n)
begin
    if rst_n = '0' then
        reg <= '0';
    elsif rising_edge(SCLK) then
        reg <= state;
    end if;
end process;
```

SI
```
process (clk, rst_n)
begin
    if rst_n = '0' then
        reg <= '0';
    elsif rising_edge(clk) and SCLK_rise = '1' then
        reg <= state;
    end if;
end process;
```

La siguiente figura muestra la salida de fin de cuenta de un divisor de frecuencia de N = 4 (contador de módulo 4), es decir estamos generando una señal derivada del reloj del sistema con una frecuencia dividida entre 4.



En esta parte se **os pide implementar**, a partir de un reloj de sistema de 10 MHz, tres señales con una frecuencia de 2.5MHz, 1.25MHz y 500KHz. Realizar el diseño para que las tres salidas generen niveles altos con una duración de un periodo del reloj del sistema.

3.3 Interfaz del módulo

Para este apartado, se entrega un fichero divisor_3.vhd y tb_divisor_3.vhd con las declaraciones de entidad definidas para la realización de la práctica. En el caso del fichero divisor_3.vhd se deberá completar el cuerpo de arquitectura con el código necesario para satisfacer el diseño descrito en la actividad anterior, la interfaz consta de 2 entradas: *clk* (reloj del sistema) y *nRst* (reset asíncrono) y 3 salidas *f_div_2_5*, *f_div_1_25* y *f_div_500*, correspondientes a las 3 frecuencias pedidas. Para implementar los contadores usad de ejemplo el utilizado en la práctica 1 por el debouncer.

```
architecture rtl of divisor_3 is
   signal cnt_div_2_5: std_logic_vector(1 downto 0);
   signal cnt_div_1_25 : std_logic;
   signal cnt_div_500 : std_logic_vector(2 downto 0);
 begin

   -- Generador de reloj s ncrono para f_div_2_5
     process(clk, nRst)
     begin
       if nRst = '0' then
         cnt_div_2_5 <= (others=> '0');
       elsif rising_edge(clk) then
         if (cnt_div_2_5 < 4) then
           cnt_div_2_5 <= (others => '0');
         else
           cnt_div_2_5 <= cnt_div_2_5 + 1;
         end if;
       end if;
     end process;
```

En este proceso, se implementa un contador de dos bits (**cnt_div_2_5**) para generar la frecuencia **f_div_2_5**. El contador se incrementa en cada flanco de subida del reloj, y cuando alcanza el valor máximo (3 en este caso), se genera un pulso de salida en **f_div_2_5**. Este proceso asegura que la frecuencia de salida sea un cuarto de la frecuencia del reloj de sistema.

```
   f_div_2_5 <= '1' when cnt_div_2_5 = 3 else '0'; -- Frecuencia dividida por 2.5

   -- Registro de desplazamiento para f_div_1_25
     process(clk, nRst)
     begin
       if nRst = '0' then
         cnt_div_1_25 <= '0';
       elsif rising_edge(clk) then
       if (f_div_2_5 = '1') then
         if (cnt_div_1_25 = '1') then
           cnt_div_1_25 <= '0';
         else
           cnt_div_1_25 <= '1';
         end if;
         end if;
       end if;
     end process;

   f_div_1_25 <= '1' when cnt_div_1_25 = '1' and f_div_2_5 = '1' else '0'; -- Frecuencia dividida por 1.25
```

Aquí se implementa un registro de desplazamiento para generar la frecuencia **f_div_1_25**. Este proceso está condicionado por la señal **f_div_2_5**, lo que significa que la frecuencia **f_div_1_25** se genera solo cuando la frecuencia **f_div_2_5** está activa. El registro alterna entre los valores '0' y '1' para generar la frecuencia deseada. Cuando **f_div_2_5** es activa, el registro se desplaza cada vez que hay un flanco de subida del reloj, alternando entre '0' y '1'.

```
   -- Contador para f_div_500
process(clk, nRst)
    begin
      if nRst = '0' then
        cnt_div_500 <= (others=> '0');
      elsif rising_edge(clk) then
      if (f_div_2_5 = '1') then
        if (cnt_div_500 < 5) then
          cnt_div_500 <= (others => '0');
        else
          cnt_div_500 <= cnt_div_500 + 1;
        end if;
        end if;
      end if;
    end process;

  f_div_500 <= '1' when cnt_div_500 = 5 and f_div_2_5 = '1' else '0'; -- Frecuencia dividida por 500
```

Este proceso implementa otro contador para generar la frecuencia **f_div_500**. Al igual que el proceso anterior, está condicionado por la señal **f_div_2_5**, generando la frecuencia **f_div_500** solo cuando la frecuencia **f_div_2_5** está activa. El contador cuenta hasta un valor específico (5 en este caso) y luego genera un pulso de salida en **f_div_500**. Esto asegura que la frecuencia de salida sea una vigésima parte de la frecuencia del reloj de sistema, cuando **f_div_2_5** está activa.

Este módulo divide el reloj de sistema en tres frecuencias síncronas inferiores (**f_div_2_5**, **f_div_1_25** y **f_div_500**) utilizando diferentes técnicas: un contador para **f_div_2_5** y **f_div_500**, y un registro de desplazamiento para **f_div_1_25**. Cada frecuencia se genera condicionalmente según el estado de la frecuencia **f_div_2_5**.

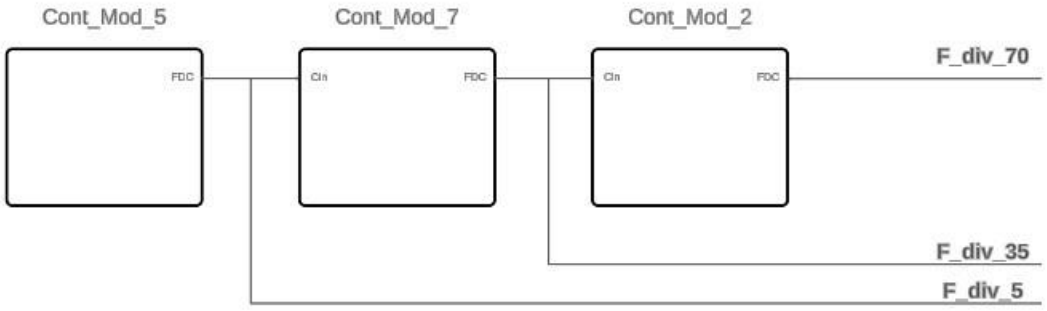
En el caso del fichero tb_divisor_3.vhd deberá completar con la generación del reloj de sistema (10MHz) y las pruebas necesarias para comprobar el correcto funcionamiento del sistema.

FOTOS TB

3.4 Economización de recursos hardware

Para la realización de este apartado es interesante tener en cuenta un concepto usado por los diseñadores hardware, como es la economización de recursos de la FPGA. A diferencia de un procesador, toda funcionalidad añadida a la FPGA tiene un "coste" en recursos hardware, para diseños amplios hay que modelar con este concepto en mente ya que se puede dar el caso de quedarse sin espacio en la FPGA. Este apartado de la práctica supone un ejemplo muy sencillo de como economizar recursos dentro de la FPGA.

Observe el diagrama de bloques de la siguiente figura, este circuito genera las frecuencias deseadas mediante el encadenamiento de contadores. El ahorro viene dado ya que su realización requiere menos registros que en el caso de generar las señales con contadores independientes. Esta solución puede adoptarse cuando las distintas frecuencias que se desean generar tienen factores comunes, como es el caso de esta práctica. Si por ejemplo se desea obtener tres frecuencias cuya N son 3, 11 y 13, no es posible realizar esta metodología. Si por el contrario los factores de división son 5, 35 y 70, como es el caso de la figura se pueden encadenar contadores para que con un contador de módulo 5, un contador de módulo 7 y un contador de módulo 2 encadenados baste para generar las 3 frecuencias necesarias. **Haga uso de esta metodología para la realización de este apartado de la práctica.**



3.5 Simulación

Se debe implementar un testbench que compruebe que el funcionamiento de los divisores de frecuencia genera indefinidamente tres señales periódicas con una frecuencia que es una cuarta parte de la señal de reloj (2.5MHz) en la salida *f_div_2_5*, octava parte de la señal de reloj (1.25MHz) en la salida *f_div_1_25* y vigésima parte de la señal de reloj (500KHz) en la salida *f_div_500*.

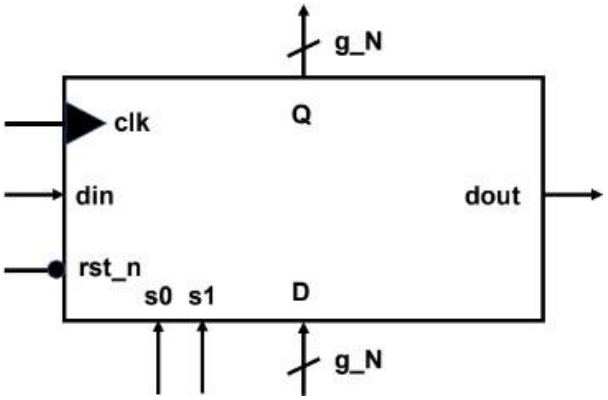
Con observar **5 periodos** de la señal más lenta será suficiente para corroborar el correcto funcionamiento. Se recomienda hacer uso del comando *assert report*. Este comando permite comprobar valores internos de las señales o genéricos en el funcionamiento interno del módulo por lo que puede ser útil para debugging. También añadir markers y utilizar el cursor principal para comprobar que la duración de la señal es la deseada.

4. Registro de desplazamiento en paralelo

4.1 Introducción

En el caso de uso mencionado, el micrófono utiliza una interfaz de un solo hilo o bit para enviar los datos registrados a la FPGA. Sin embargo, para procesar y almacenar estos valores es mucho más cómodo hacer uso de palabras de un tamaño fijo de bits.

Para poder pasar una señal recibida de serie a paralelo es necesario hacer uso de un registro de desplazamiento (shift register). En esta práctica vamos a implementar un shift register parametrizable con carga en paralelo. Este registro nos va a permitir tanto pasar de serie a paralelo como viceversa, con tamaño del registro variable. También se podrá hacer rotar los bits en ambos sentidos.



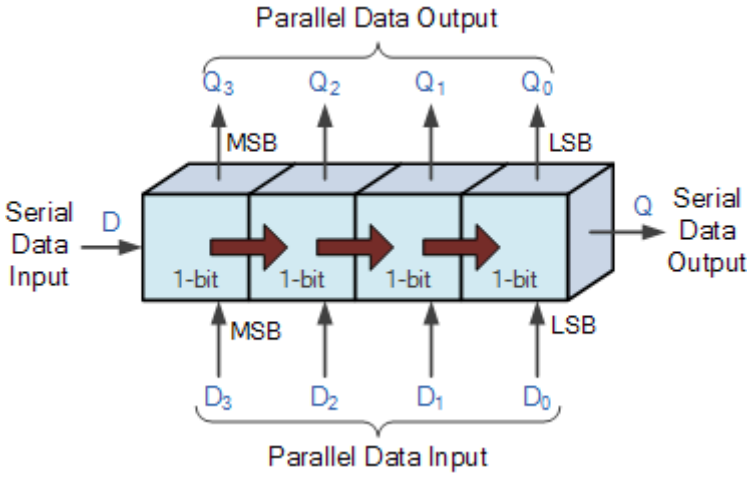
4.2 Diseño y funcionamiento

El registro va a permitir cargar datos en paralelo por ello necesitamos dos señales de control que nos permitan bien desplazar el dato o bien cargar en paralelo el registro. El registro va a tener dos entradas de datos (*din, D)* y dos salidas (*dout*, *Q*), dos en paralelo (*D, Q*) y dos en serie (*din, dout*).

Las señales s0 y s1 funcionan de la siguiente tabla:

s0	**s1**	**Acción**
0	0	Sin cambios
0	1	Rotar hacia la izquierda. (din => LSB)
1	0	Rotar hacia la derecha. (din =>MSB)
1	1	Carga paralela. (D => register)

En la salida Q se corresponde siempre con los bits que hay actualmente en el registro. Por *din* los datos entran con la frecuencia correspondiente y salen por *dout* a la misma frecuencia. El reset asíncrono debe cargar el registro con 0s. Si el sistema no está “enabled”, la salida por *dout* es ‘0’ y el registro interno se mantiene igual. *D* y *Q* dependen su tamaño del genérico *g_N* que determina el tamaño del registro de desplazamiento.



El funcionamiento de un desplazamiento a la derecha es sencillo, imaginando un registro de tamaño 4 y que los valores que llegan a din son “10110000”, en la siguiente tabla se muestra el valor de tanto Q, como dout.

TIME	0	1	2	3	4	5	6	7	8
din	1	0	1	1	0	0	0	0	
Q3	0	1	0	1	1	0	0	0	0
Q2	0	0	1	0	1	1	0	0	0
Q1	0	0	0	1	0	1	1	0	0
Q0	0	0	0	0	1	0	1	1	0
dout	0	0	0	0	1	0	1	1	0

Para el caso de rotar hacia la izquierda (S1= 0 S0 = 1), *dout* mantendrá el mismo valor que **el bit más significativo**, Q3 para el caso del registro de 4 bits, los valores de *din* irán entrando por el bit menos significativo (Q0), desplazando el resto de bits hacia la izquierda.

TIME	0	1	2	3	4	5	6	7	8
din	1	1	0	0	1	1	1	0	
Q3	0	0	0	0	1	1	0	0	1
Q2	0	0	0	1	1	0	0	1	1
Q1	0	0	1	1	0	0	1	1	1
Q0	0	1	1	0	0	1	1	1	0
dout	0	0	0	0	1	1	0	0	1

Para la salida en serie *dout* siempre tendrá el valor del bit menos significativo de Q, salvo para cuando registro esté configurado en modo rotar hacia la izquierda, en ese caso la salida serie *dout* tendrá el valor del bit más significativo de Q. Utilizar una asignación concurrente para esta sentencia.

4.3 Implementación y simulación

Para la implementación de este módulo van a ser muy útiles los atributos predefinidos *low* y *high*. Dado un vector de la forma *res := std_logic_vector(4 downto 0);* El atributo *res'high* se refiere a la posición del MSB del vector, de la misma forma *res'low* es la posición del LSB. Es decir, podemos tomar el bit menos significativo como *res(res'low).*

También van a hacer falta hacer uso de bucles for, o en su defecto de la concatenación de vectores:

```
for i in 4 downto 0 loop
    --- proceso
end loop;
```

```
vec <= "1111" & "0";
--- vec es "11110"
```

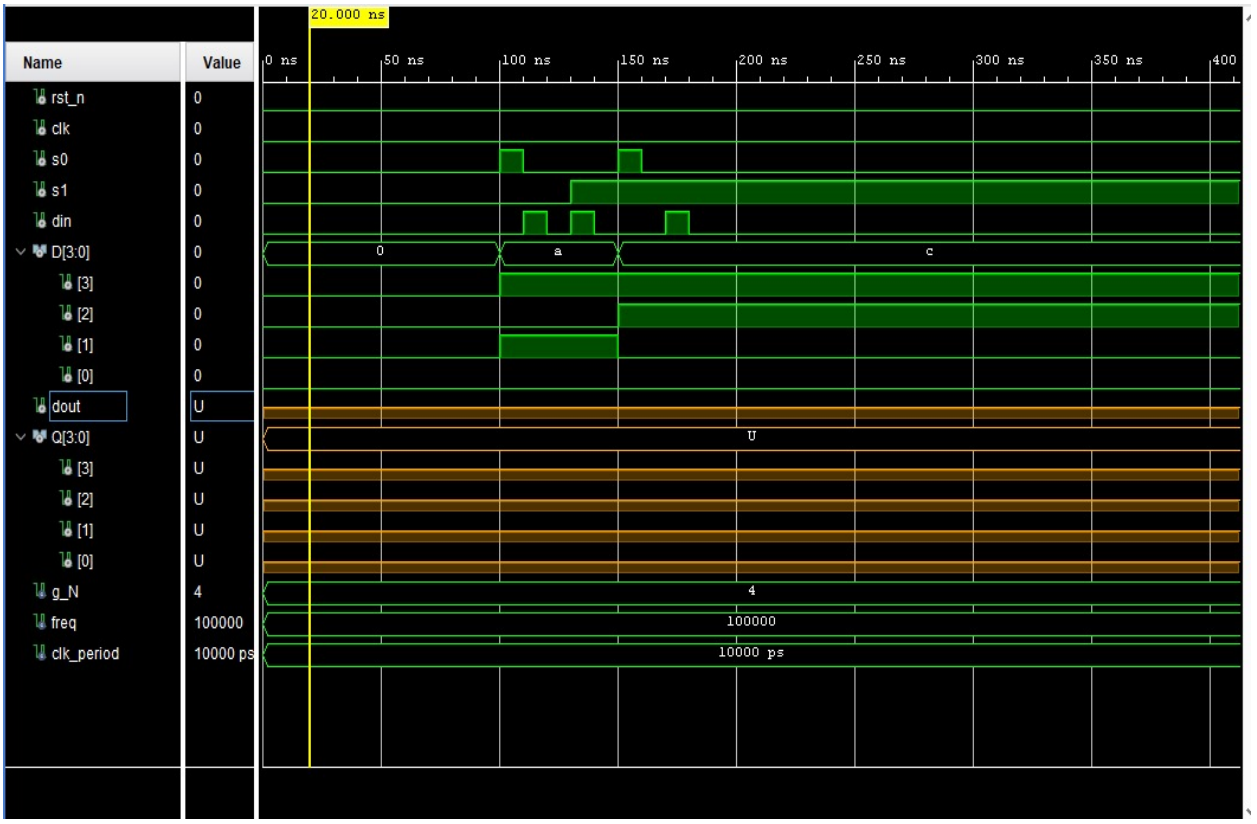
```
ARCHITECTURE behavioural OF shift_register IS

    SIGNAL registro : std_logic_vector(g_N - 1 DOWNTO 0);

BEGIN
    PROCESS (rst_n, clk)
    BEGIN
        IF rst_n = '0' THEN
            --- Reset asincrono del registro
            registro <= (OTHERS => '0');
        ELSIF rising_edge(clk) THEN
            ------------------------
            -- Completar la l gica combinacional de acuerdo a los valores de entrada s0 y s1
            ------------------------
            IF (s0 = '1') AND (s1 = '0') THEN
                registro <= din & registro(g_N - 2 DOWNTO 0);
            ELSIF (s0 = '1') AND (s1 = '1') THEN
                registro <= D;
            ELSIF (s0 = '0') AND (s1 = '1') THEN
                registro <= '0' & registro(g_N - 2 DOWNTO 0);
            END IF;
        END IF;
    END PROCESS;
    ------------------------
    -- Completar con la asignaci n final a Q y dout con los valores correspondientes del registro
    ------------------------
     process(clk) begin
        if (rising_edge (clk))then
            dout <=registro(g_N -1);
            Q<=registro;
        end if;
    end process;
END behavioural;
```

Se implementa un proceso sensibilizado al cambio en el reset asincrónico y al flanco de subida del reloj. Cuando el reset está activo, se restablece el registro a ceros. En cada flanco de subida del reloj, se evalúan las señales de control s0 y s1. Dependiendo de sus valores, se realiza una de tres operaciones:

- Si s0 es alto y s1 es bajo, se realiza un desplazamiento hacia la izquierda, donde el valor de din se coloca en el bit menos significativo del registro.
- Si s0 y s1 son altos, se carga el registro con los datos de entrada en paralelo D.
- Si s0 es bajo y s1 es alto, se realiza un desplazamiento hacia la derecha, donde el bit más significativo del registro se descarta y se introduce un cero en el bit más significativo.

También se implementa un proceso sensibilizado al flanco de subida del reloj para asignar los valores de salida **Q** y **dout** en el contenido del registro. En cada flanco de subida del reloj, la salida en serie **dout** refleja el bit más significativo del registro, mientras que la salida en paralelo **Q** refleja todo el contenido del registro.

Para este módulo se debe completar el código otorgado en el fichero shift_register.vhd. En este se incluye la interfaz y genéricos necesarios. Se deberá realizar una simulación que verifique que las rotaciones de los datos son correctas. Para esto, se ofrece un fichero testbench a completar tb_shift_register.vhd. Se recomienda elegir un *g_N* pequeño (4 como el usado en la explicación) para reducir el número de ciclos que tarda en salir el dato por la salida dout. En la implementación puede que sea necesario usar bucles for.



UNIVERSIDAD
NEBRIJA

[
10