Actividad 3.1

1) La elección del ciclo a “robra” se puede seleccionar con un LFSR de módulo máximo M-1 para no limitar la máxima velocidad de funcionamientto del divisor. De esta manera, y dado que el contador no sigue el orden binario, los pulsos que se “roban” no serán extraídos de manera consecuitiva, no rompiendo tanto la fase del Clock de salida.

2)

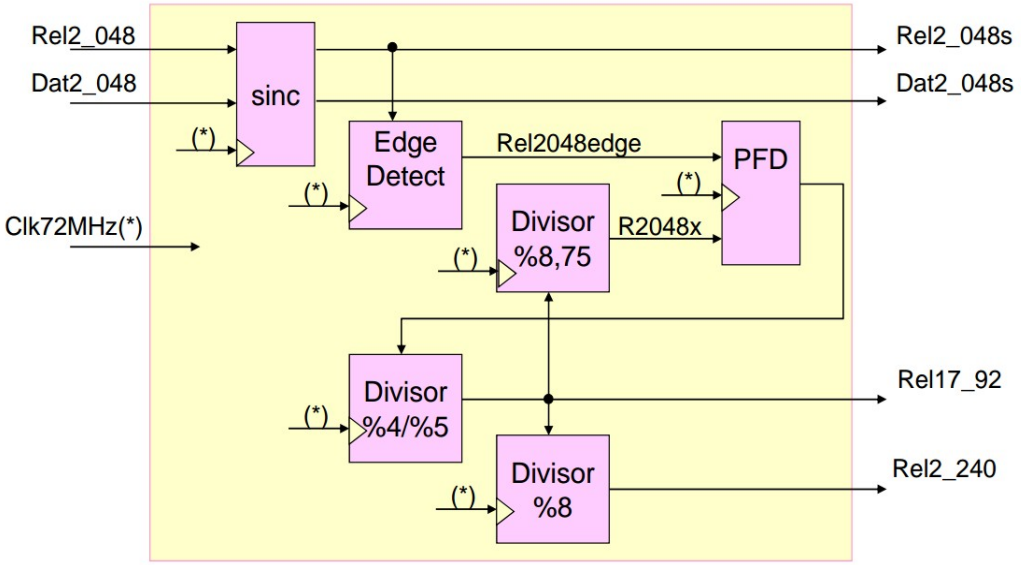
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity En_Rate_Multiplier is
  Generic (M : NATURAL := 5; N : NATURAL := 5) ;
   Port ( CLK_IN  : in  STD_LOGIC;
          Reset   : in  STD_LOGIC;
          CLK_OUT : out STD_LOGIC;
          Q_LFSR  : out STD_LOGIC_VECTOR(M downto 1));
end En_Rate_Multiplier;

architecture Arq_Rate_Multiplier of En_Rate_Multiplier is
  subtype salida is STD_LOGIC_VECTOR(M downto 1);

  function LFSR ( FB, n_stages, n_counts : integer;
                  init_value : STD_LOGIC_VECTOR(M downto 1);
                  type_of_FB : std_ulogic) return salida is -- type_of_FB = '0' es XOR, '1' es XNOR
      variable Q    : salida;
  begin
    Q   := init_value;

    loop1 : for i in 1 to n_counts loop
        Q := Q(n_stages-1 downto 1) & (type_of_FB XOR Q(FB) XOR Q(n_stages));
      end loop loop1;
    return Q;
  end LFSR;

  signal sQ_LFSR    : salida;
  signal Reset_AUX, Reset_LFSR, Valid_Pulse : STD_LOGIC;
begin

LFSR_5: entity work.En_LFSR(Arq_LFSR)
  generic map( N => M) -- LFSR módulo 31 = 2**M-1, siendo M el módulo del Rate-Multiplier
  port map(
    CLK    => CLK_IN,
    Reset   => Reset_LFSR,
    Type_FB => '1', -- XNOR
    FB      => 2,
    Q      => sQ_LFSR
  );

  Q_LFSR <= sQ_LFSR;

  -- Este process trabaja registrando la salida del LFSR esperando el último valor del contador,
  --de esta manera se busca "estirar el Reset" de manera de simular un contador de módulo M
  process(CLK_IN)
  begin
    if rising_edge(CLK_IN) then
      if sQ_LFSR = LFSR(2,M,2**M-2,STD_LOGIC_VECTOR(to_unsigned(0,M)),'1') then
        Reset_AUX <= '1';
      else
        Reset_AUX <= '0';
      end if;
    end if;
  end process;

  Reset_LFSR  <= Reset_AUX or Reset;

  Valid_Pulse <= '1' when ( ( ( (2**M)-1 - to_integer(unsigned(sQ_LFSR)) )  <= N ) and ( N < (2**M)/2) )
              or   ( ( to_integer(unsigned(sQ_LFSR)) < N-1 ) and ( N  >= (2**M)/2) ) -- Tiene en cuenta le doble cero
              else '0';

  CLK_OUT <= CLK_IN when Valid_Pulse = '1' and Reset = '0' else '0';

end Arq_Rate_Multiplier;
```

Esta implementación está hecha para un M = 32 (LFSR de 5 etapas con realimentación XNOR con la sgunda etapa) y N variable de 0 a M-1 (31).

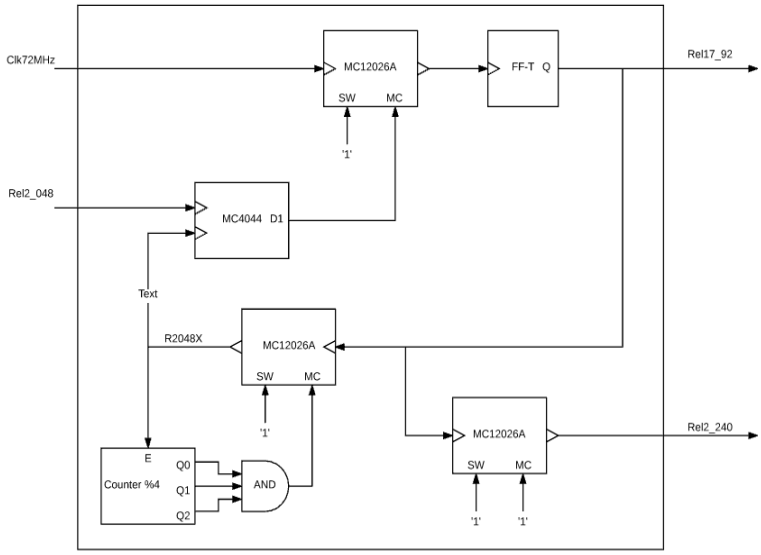
Para llevarlo a cabo se “estiró” el rango del LFSR repitiendo el cero con un reset virtual. De esta manera se puede utilizar comparando directamente.

El seguimiento no conscutivo de la numeración del LFSR permite una distribución más homogénea a lo largo del rango total del módulo del Rate Multiplier.

3) Como circuito de uso real se propone el mostrado en la teoría (resuelto en VHDL en la Actividad 3.2), donde se propone lo siguiente

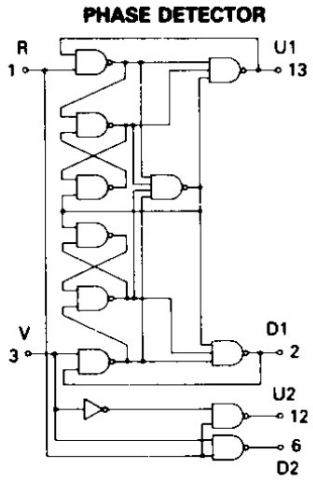


En este caso el MC12026A se utilizó como divisor de módulo 8/9 para generar lo 2.048MHz (divisor 8.75) que se compararán en fase con la referencia de entrada. La entrada a este módulo son 17.92MHz que se generan dividiendo un CLK interno de 72MHz con la misma metología (si bien el módulo necesario sería un 4/5, se propone hacer un 4/4.5 dividiendo a la mitad la salida de este MC12026A con un Flip-Flop tipo T). La selección de 4/4.5 se realiza con la salida D1 de un MC4044, el cual compara tanto en fase y frecuencia la referencia de 2.048MHz de entrada y la generada a través de los 17.92MHz que se desean recuperar. Dicha diferencia logrará compenzar los corrimientos de fase y frecuencia inherentes del sistema. Como último se usa otro MC12026A como divisor por 8 para conseguir los 2.240 MHz.



4) El comparador de fase y frecuencia MC4044 provee dos configuraciones posibles para manejar dos salidas (U1 y D1) las cuales indican si la entrada V (de feedback) está atrasada o adelantada en fase respecto a la señal de entrada de referencia (R).

En el siguiente esquema se visualizan ambos sistemas



La diferencia de complejidad entre uno y el otro es notable en el esquemático, pero también lo es funcionalmente. El primero de ambos sigue un diagrama de estados que converge a posibles estados estables sea cual sea el corrimiento de fase de la señal V respecto a la señal R. En la siguiente tabla se resume el diagrama de estados

R-V	R-V	R-V	R-V	U1	D1
0-0	0-1	1-1	1-0		
(1)	2	3	(4)	0	1
5	(2)	(3)	8	0	1
(5)	6	7	8	1	1
9	(6)	7	12	1	1
5	2	(7)	12	1	1
5	2	7	(8)	1	1
(9)	(10)	11	12	1	0
5	6	(11)	(12)	1	0

Donde los estados entre paréntesis indican estados estables, mientras que los que no tienen paréntesis indican estados de transición. A medida que cambian las señales de entrada el movimiento por la tabla es por filas. Una vez realizado el cambio de columna se va a asignar un nuevo estado (estable o no). En el caso de serlo nada cambia, pero en el caso de no serlo, se genera un cambio hacia la fila donde se encuentra este nuevo estado pero estable, de esta manera lo que se ven afectadas son las salidad U1 y D1.

En VHDL se pensó por un lado la resoución de la asignación de los estados dependiendo del cambio de entrada y el estado actual, y por otro la asignación de las salidas dependiendo del estado estable en el que se encuentre.

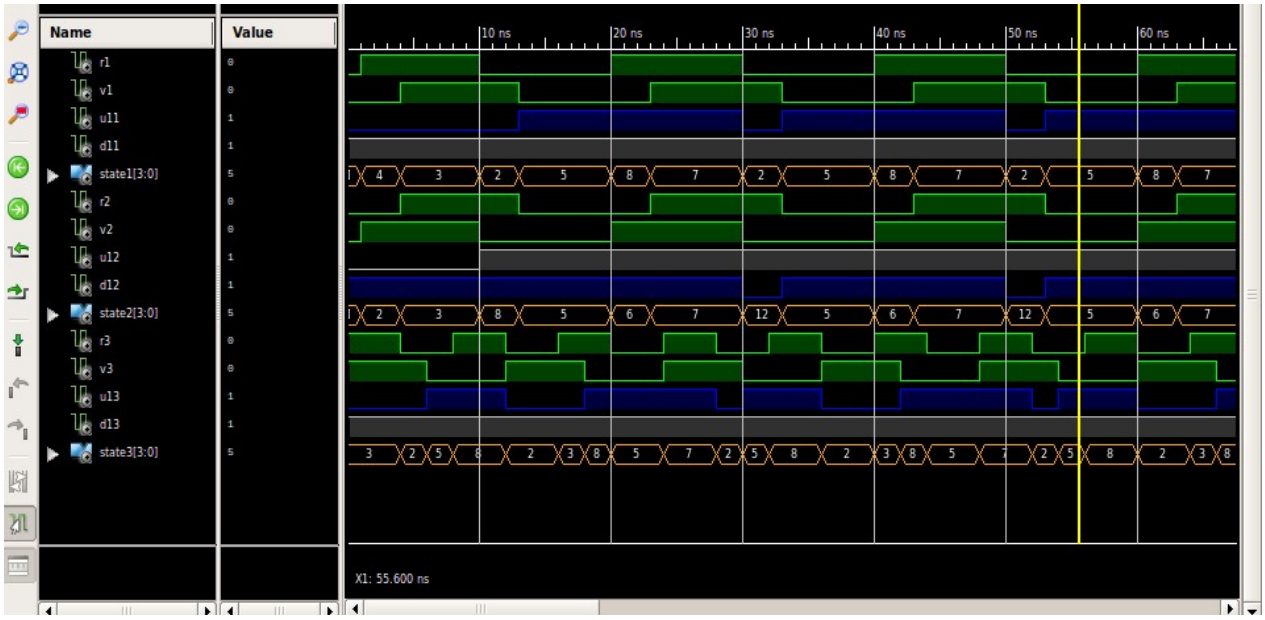
Asignación de estados

```
Actual_State <= Next_State & R & V;

 with Actual_State select
    Next_State <=   x"1" when x"1" & "00",
                    x"2" when x"1" & "01",
                    x"3" when x"1" & "11",
                    x"4" when x"1" & "10",

                    x"5" when x"2" & "00",
                    x"2" when x"2" & "01",
                    x"3" when x"2" & "11",
                    x"8" when x"2" & "10",

                    x"5" when x"3" & "00",
                    x"2" when x"3" & "01",
                    x"3" when x"3" & "11",
                    x"8" when x"3" & "10",

                    x"1" when x"4" & "00",
                    x"2" when x"4" & "01",
                    x"3" when x"4" & "11",
                    x"4" when x"4" & "10",

                    x"5" when x"5" & "00",
                    x"6" when x"5" & "01",
                    x"7" when x"5" & "11",
                    x"8" when x"5" & "10",

                    x"9" when x"6" & "00",
                    x"6" when x"6" & "01",
                    x"7" when x"6" & "11",
                    x"C" when x"6" & "10",

                    x"5" when x"7" & "00",
                    x"2" when x"7" & "01",
                    x"7" when x"7" & "11",
                    x"C" when x"7" & "10",

                    x"5" when x"8" & "00",
                    x"2" when x"8" & "01",
                    x"7" when x"8" & "11",
                    x"8" when x"8" & "10",

                    x"9" when x"9" & "00",
                    x"A" when x"9" & "01",
                    x"B" when x"9" & "11",
                    x"C" when x"9" & "10",

                    x"9" when x"A" & "00",
                    x"A" when x"A" & "01",
                    x"B" when x"A" & "11",
                    x"C" when x"A" & "10",

                    x"5" when x"B" & "00",
                    x"6" when x"B" & "01",
                    x"B" when x"B" & "11",
                    x"C" when x"B" & "10",

                    x"5" when x"C" & "00",
                    x"6" when x"C" & "01",
                    x"B" when x"C" & "11",
                    x"C" when x"C" & "10",

                    x"1" when others;
```

Asignación de la salida

```
with Actual_State(5 downto 2) select
    Salida <= "01" when x"1",
              "01" when x"2",
              "01" when x"3",
              "01" when x"4",
              "11" when x"5",
              "11" when x"6",
              "11" when x"7",
              "11" when x"8",
              "10" when x"9",
              "10" when x"A",
              "10" when x"B",
              "10" when x"C",

              "00" when others;

U1 <= Salida(1);
D1 <= Salida(0);
```

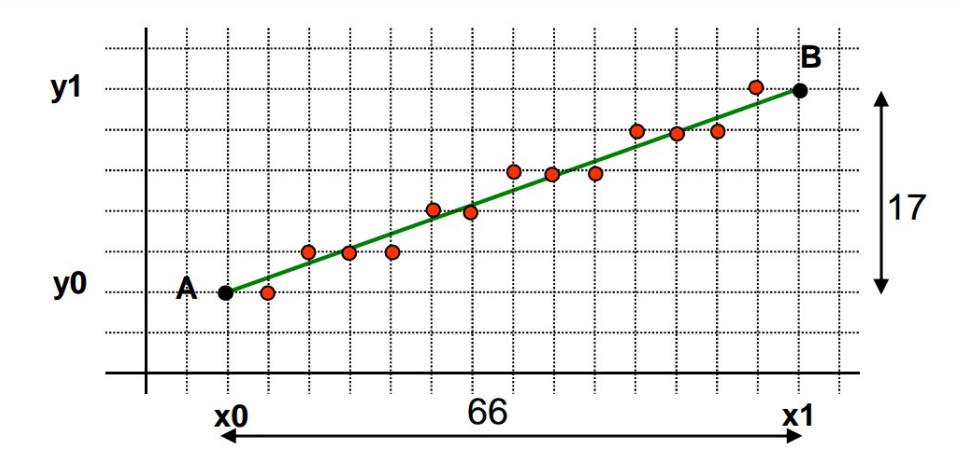
De esta manera se simularon los tres casos que se presentan en la hoja de datos obteniendo lo siguiente



Donde la única diferencia que se observa en relación al Datasheet es en la última cadena de estados, ya que aquí se presencia el patrón […]-5-8-2-3-8-5-7-2-[...] y en el Datasheet se observa que debería ser […]-5-8-2-3-8-1-3-2-[...]. Esto se debe a que en la tabla propuesta por el Datasheet no existe posible paso del estado 8 al estado 1, por lo que no se entiende la secuencia de estados propuesta por la misma ya que la correcta es la obtenida por simulación.

Respecto a la limitación de la síntesis es que no se puede asegurar que los T_{COM} de cada uno de las configuraciones sean iguales, no pudiendo simular las posibles metaestabilidades inherentes al sistema.

5) Siguiendo el ejemplo del Algoritmo de Bresenham presentado en el módulo



se puede observar que la distancia de la recta a los puntos no solo que siempre es menor a medio “cradradito” o medio pulso del CLK de referencia, sino que está balanceado en atraso y adelanto, somo así también pasando por zonas sin error. Esto se logra porque dentro de la excursión total se cuentan con muchos cambios en Y cada pocos X. Lo que significaría que si la pendiente es muy chica (cercana a 0) o muy grande (tendiendo a “infinito”), el balance está muy perjudicado ya que se estaría realizando un cambio en X o en Y en todo el recorrido de la recta, nunca pasando por zonas de error nulo.