Práctica:

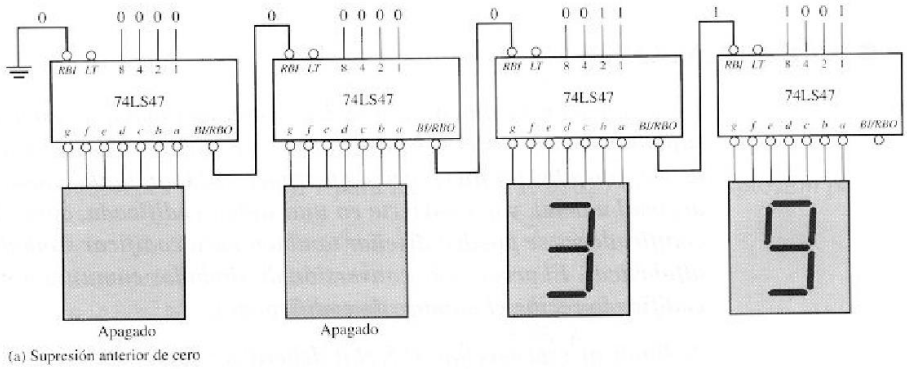
Proyectos.

1	

Diseñar[1] un circuito contador de 0 a 99. La cuenta se debe mostrar sobre dos displays de 7 segmentos avanzando una cuenta aproximadamente por segundo. Tendrá, además, una entrada de puesta a cero y una de reloj. La cuenta siguiente a 99 es 0. Los puntos decimales deben estar apagados.
Como contadores emplear la megafunción "lpm_counter" incluida en la biblioteca "library lpm". Para los decodificadores de BCD a 7 segmentos emplear las macrofunciones "a_7447" incluidas en la biblioteca "library altera". El segmento de código es el siguiente:

```
library lpm;
        use lpm.lpm_components.all;
library altera;
        use altera.maxplus2.all;
```

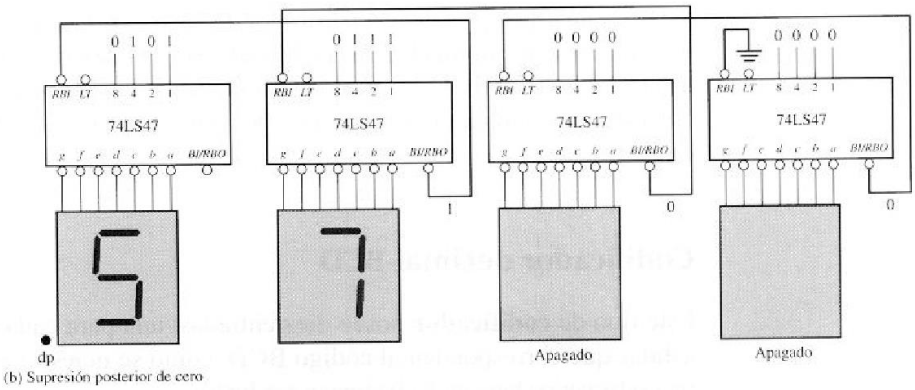
Del oscilador que tiene la placa de desarrollo generar una señal de aproximadamente 1 Hz. De ella alimentar de manera simultanea las entradas de reloj de todos los contadores (tratar de realizar siempre un diseño sincróno).
Los conversores de BCD a 7 segmentos tienen dos entradas especiales, empleadas para no mostrar los ceros a la izquierda del primer dígito no nulo, son RBI (ripple blanking input) y BI/RBO (blanking input/ripple blanking output) se emplean de la siguiente forma[2]:



(a) Supresión anterior de cero

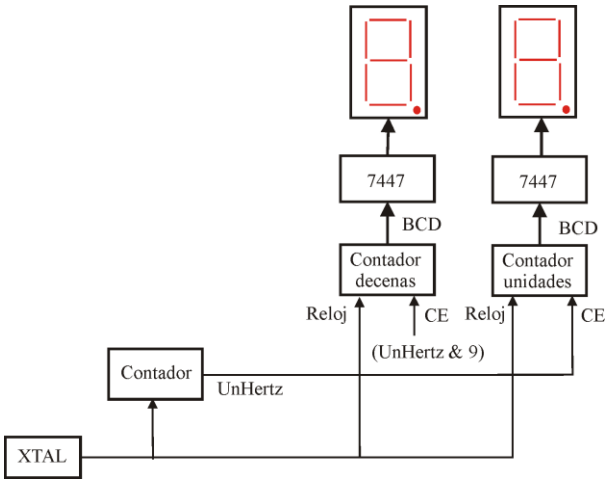
O para suprimir los ceros posteriores:

[1] Realizar todos los diseños sobre una Cyclone II EP2C35F672C6.
[2] http://www.dte.upct.es/personal/andres_iborra/docencia/elec_ind/pdfs/t3.pdf

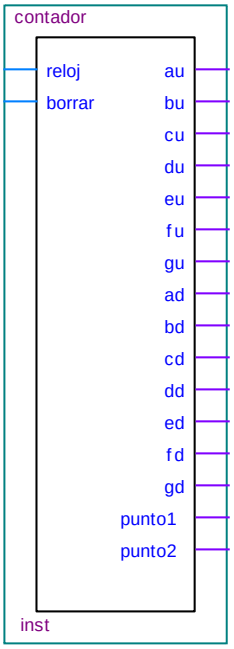


(b) Supresión posterior de cero

El diagrama en bloques propuesto es el siguiente:



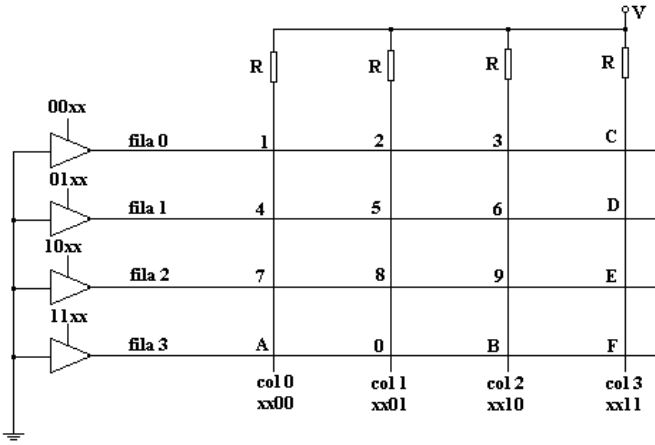
El circuito sintetizado total debe tener el siguiente aspecto:

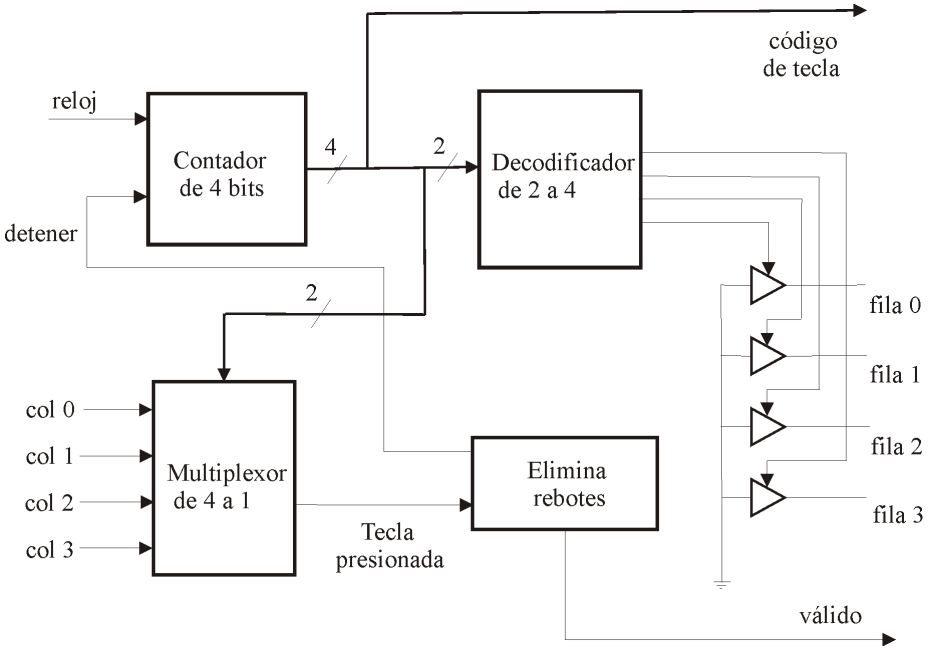


Opcionalmente realizar una segunda versión del mismo contador pero esta vez sin emplear ni las megafunciones ni las macrofunciones. Crear primero un circuito que realice la conversión de BCD a 7 segmentos. Este circuito se puede invocar como componente en el proyecto principal.

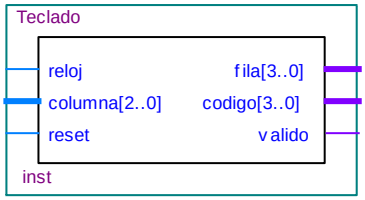
2	

Diseñar un sistema que controle un teclado hexadecimal, la tecla oprimida debe mostrarse en un display de 7 segmentos. Para asegurar los posibles "rebotes" de las teclas, considere como válida la opresión de ella si el tiempo en que se mantiene oprimida es mayor que 10 mseg. El teclado funcionará de la siguiente manera: cuando una tecla es oprimida se producirá la correspondiente unión eléctrica entre una fila y una columna de la matriz de conexiones vista en la figura; por ejemplo la opresión de la tecla "9" producirá la unión de la fila 2 con la columna 2. Para la detección de dicha conexión se realiza un barrido sistemático del teclado controlado por una palabra de 4 bits, con los dos bits más significativos de dicha palabra se produce la habilitación de los buffers que conectan cada fila a un "0", la presión de una tecla hará que su columna correspondiente pase a "0" (en cualquier otro caso las columnas se encuentran en "1"), leyendo secuencialmente (con los 2 bits menos significativcos) las columnas se detectará el "0" y por lo tanto la tecla presionada.

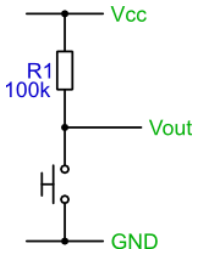


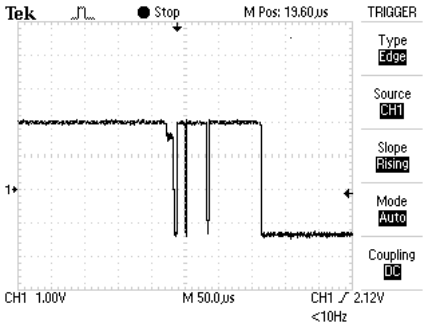


El circuito tendrá además una salida de 4 bits que indica el código BCD de la tecla presionada y una entrada de reset. La señal valido debe permanecer en '1' durante un ciclo de reloj.



Rebotes. En la parte izquierda de la figura siguiente se muestra un pulsador conectado a una resistencia de pullup a fin de introducir información binaria en un circuito. La imagen de la derecha muestra la captura con osciloscopio de la señal presentada, como se puede ver al presionar el pulsador no se produce una transición clara de '1' a '0'. La misma condición se produce al liberar el pulsador. La anulación de este rebote se puede lograr por hardware mediante el agregado de un schmitt trigger, un diodo, un capacitor y resistencias. Otra forma de validar un pulso es mediante un circuito de espera, se detecta la primera transición y se muestrea la señal un tiempo después, si el cambio se mantiene la transición se toma como válida.





Conectar el teclado al conector interior de la placa DE2. La línea roja del cable plano debe coincidir con el pin 1 del conector. La asignación de pines para le teclado debe ser la siguiente:[3]

D25	col2
J22	fila3
E26	col1
E25	fila1
F24	fila2
F23	col0
J21	fila0

3	

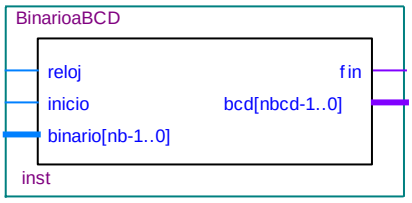
Diseñar un circuito conversor de binario, de 8 bits, a BCD. El problema se puede resolver utilizando el algorítmo de desplazamiento y suma 3, como se indica a continuación.
a) desplazar el número binario un bit a la izquierda.
b) si el valor binario de cualquier columna es 5 o mayor sumar 3 a esa columna.
d) si quedan bits en el registro ir al paso a, de lo contrario fin.

Operación	Decenas	Unidades	Binario
Inicio			1 1 1 0

[3] Por no ser un teclado de 16 caracteres la col3 debe conectarse a '1'.

← 1 bit		1	1 1 0
← 1 bit		1 1	1 0
← 1 bit		1 1 1	0
Suma 3		1 0 1 0	0
← 1 bit	1	0 1 0 0	
BCD	1	4	

El circuito debe tener una entrada para la señal de inicio, de duración igual a un período de reloj, que le indica al circuito que tiene en su entrada un nuevo número a convertir. También debe tener una salida que se pondrá en alto cuando finalice la conversión.

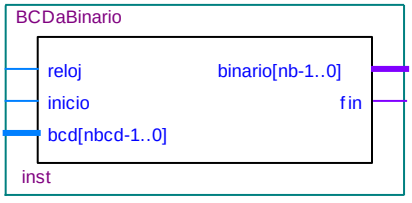


4	

Diseñar un circuito conversor de BCD a binario de 8 bits. El problema se puede resolver utilizando el algorítmo de desplazamiento y resta 3, como se indica a continuación.
a) desplazar el número BCD un bit a la derecha.
b) restar 3 de todas aquellas columnas BCD que contienen un número mayor o igual que 5.
d) si el registro binario no se ha llenado ir al paso a, de lo contrario fin.

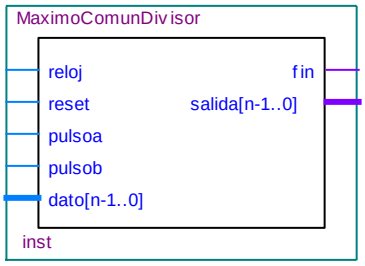
Operación	Centenas	Decenas	Unidades	Binario
BCD	1	1	7	
Inicio	0 0 0 1	0 0 0 1	0 1 1 1	
1 bit →	0 0 0	1 0 0 0	1 0 1 1	1
Resta 3	0 0 0	0 1 0 1	0 1 0 1	1
1 bit →	0 0	0 0 1 0	1 1 0 0	0 1
Resta 3	0 0	0 0 1 0	1 0 0 1	0 1
1 bit →	0	0 0 0 1	0 1 0 0	1 0 1
1 bit →		0 0 0 0	1 0 1 0	0 1 0 1
Resta 3		0 0 0 0	0 1 1 1	0 1 0 1
1 bit →		0 0 0	0 0 1 1	1 0 1 0 1
1 bit →		0 0	0 0 0 1	1 1 0 1 0 1
1 bit →		0	0 0 0 0	1 1 1 0 1 0 1
…				
Binario				0 0 0 1 1 1 0 1 0 1

El circuito debe tener una entrada para la señal de inicio, de duración igual a un período de reloj, que le indica al circuito que tiene en su entrada un nuevo número a convertir. También debe tener una salida que se pondrá en alto cuando finalice la conversión.

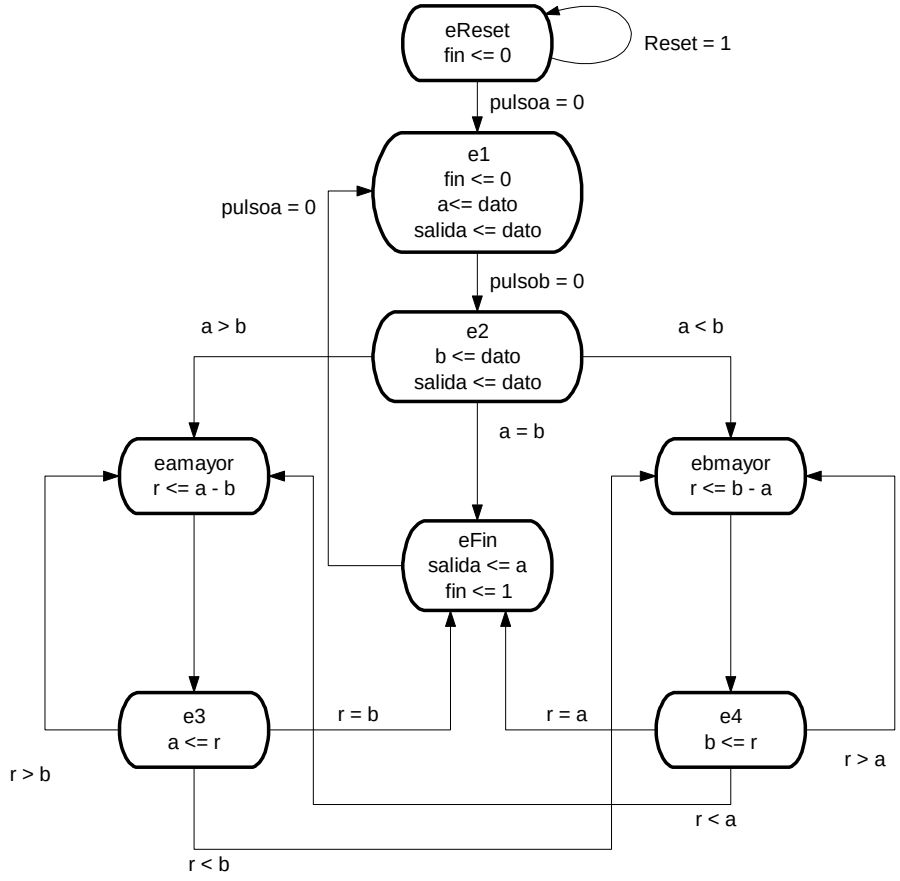


5	

Diseñar un sistema que calcule el Máximo Común Divisor entre dos números binarios positivos, A y B, de n bits. En la entrada dato(n-1..0) se presentan los dos enteros a los cuales se debe calcular el MCD. La señal pulsoa indica que en dato se presenta el número A. pulsob indica que se dispone el valor de B en dato y que se debe comenzar el cálculo. El resultado se debe presentar en salida(n-1..0). El sistema posee una entrada de inicialización, reset, y otra de temporización, reloj. El circuito debe indicar la finalización del cálculo colocando la salida fin en 1. Los puertos de entrada y salida del circuito deben tener el siguiente aspecto:



Un diagrama de flujo posible es el siguiente:



6	

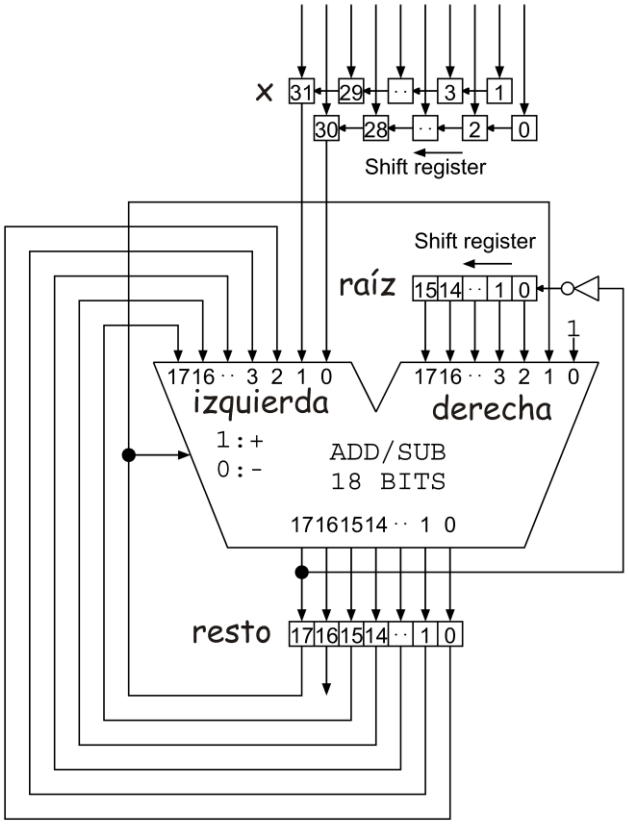
La operación $\sqrt{a^2 + b^2}$ se puede calcular de manera aproximada mediante:

$$\sqrt{a^2+b^2} \approx \quad máx(((x-0{,}125\,x)+0{,}5\,y),\,x)$$
$$x \;=\; máx(\,|a|,|b|\,)$$
$$y \;=\; mín(\,|a|,|b|\,)$$

En donde, en binario, las operaciones 0,125 . *x* y 0,5 . *y* se pueden realizar mediante desplazamientos a la derecha de 3 y 1 posiciones. Sintetizar en VHDL un circuito que realiza la operación señalada para operandos a y b de n bits.

7	

En [1][4] se muestra un algoritmo para determinar la raíz cuadrada de un número entero mayor o igual a cero. En la siguiente figura se ilustra este procedimiento para un número de 32 bits.



Entre las muchas maneras en que se puede codificar este algoritmo en VHDL a continuación se muestran dos posibles soluciones para enteros de n bits:

a) Mediante una función.

[4] A New Non-Restoring Square Root Algorithm and Its VLSI Implementations; Yamin Li and Wanming Chu; *International Conference on Computer Design (ICCD'96), October 7–9, 1996, Austin, Texas, USA.*

```
------------------------------------------------------------------------------------------------
--                                                                                            --
-- Algoritmo para calcular la raiz cuadrada de un número entero de n bits                     --
-- según: A New Non-Restoring Square Root Algorithm and Its VLSI                              --
-- Implementations; Yamin Li and Wanming Chu; International Conference on                     --
-- Computer Design (ICCD'96), October 7–9, 1996, Austin, Texas, USA.                          --
--                                                                                            --
------------------------------------------------------------------------------------------------

library work;
        use work.RaizCuadradaPaquete.all;
library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;



entity RaizCuadrada is
        generic
        (
        n       :       integer  :=     32              -- Debe ser un número par
        );
        port
        (
        x       : in std_logic_vector( n - 1 downto 0);         -- Operando
        raiz    : out std_logic_vector( n / 2 - 1 downto 0)     -- Resultado
        );
end RaizCuadrada;



architecture De32BitsMedianteFuncion of RaizCuadrada is

begin

        raiz <= std_logic_vector(RaizCuadrada2( unsigned(x)));

end De32BitsMedianteFuncion;


------------------------------------------------------
--
--  Biblioteca para el cálculo de la raiz cuadrada de un
--  entero sin signo de n bits bits
--  Se emplea el algoritmo según "A New Non-Restoring Square
--      Root Algorithm and Its VLSI Implementations" International
--      Conference on Computer Design (ICCD'96), October 7–9,
--      1996, Austin, Texas, USA
--
------------------------------------------------------

library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;



package RaizCuadradaPaquete is
```

```
        function  RaizCuadrada2  ( x : unsigned ) return unsigned; -- x es de cualquier número de n bits  con n par
end RaizCuadradaPaquete;


package body RaizCuadradaPaquete is


function  RaizCuadrada2  ( x : unsigned ) return unsigned is

        constant LEFT: integer := x'length;
        variable xaux : unsigned( LEFT - 1 downto 0 ) := x;                    -- Valor de entrada.
        variable raiz : unsigned( LEFT / 2 - 1 downto 0 ) := ( others => '0' );  -- Resultado
        variable izquierda, derecha, resto : unsigned(LEFT / 2 + 1 downto 0) := (others => '0' );  -- Resto de entrada al sumador/restador


        begin
                assert LEFT mod 2  = 0 report "x debe tener un número par de bits" severity ERROR;
                for i in 0 to LEFT / 2 - 1 loop
                        derecha(0) := '1';
                        derecha(1) := resto(LEFT / 2 + 1);
                        derecha(LEFT / 2 + 1 downto 2) := raiz;
                        izquierda(1 downto 0) := xaux(LEFT - 1 downto LEFT - 2);
                        izquierda(LEFT / 2 + 1 downto 2) := resto(LEFT / 2 - 1 downto 0);
                        xaux(LEFT - 1 downto 2) := xaux(LEFT - 3 downto 0);        -- Desplazamiento de 2 bits
                        if ( resto(LEFT / 2 + 1) = '1') then
                                resto := izquierda + derecha;
                        else
                                resto := izquierda - derecha;
                        end if;
                        raiz(LEFT / 2 - 1 downto 0) := raiz(LEFT / 2 - 2 downto 0) & ( not resto(LEFT / 2 + 1)
        );
                end loop;
                return raiz;
        end RaizCuadrada2;

end RaizCuadradaPaquete;
```

b) Mediante un proceso con aviso de inicio y de final.

```
--------------------------------------------------------------------------------------------------
--                                                                                              --
-- Algoritmo para calcular la raiz cuadrada de un número entero de n bits                       --
-- según: A New Non-Restoring Square Root Algorithm and Its VLSI                                --
-- Implementations; Yamin Li and Wanming Chu; International Conference on                       --
-- Computer Design (ICCD'96), October 7–9, 1996, Austin, Texas, USA.                            --
--                                                                                              --
--------------------------------------------------------------------------------------------------

library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;


entity RaizCuadrada is
        generic
        (
        n       : integer  := 32              -- Debe ser un número par
```

```
        );
        port
        (
        reloj     : in std_logic;
        inicio    : in std_logic;
        fin       : out std_logic;
        x         : in std_logic_vector( n - 1 downto 0);                    -- Operando
        raiz      : out std_logic_vector( n / 2 - 1 downto 0)                -- Resultado
        );
end RaizCuadrada;


architecture Mixta of RaizCuadrada is



        begin

        process( reloj, inicio)
                variable izquierda : unsigned( n / 2 + 1 downto 0 );
                variable derecha : unsigned( n / 2 + 1 downto 0 );
                variable resto : unsigned( n / 2 + 1 downto 0 );
                variable raiz2 : unsigned( n / 2 - 1 downto 0 );
                variable xaux : unsigned( n - 1 downto 0 );
                variable nl  : integer range 0 to n / 2;          -- Número de lazos de cálculo que se deben realizar
        begin
                if (  reloj'event and reloj = '1'  ) then
                        if inicio = '1' then
                                derecha := ( 0 => '1', others => '0');
                                izquierda( 1 downto 0 ) := unsigned( x( n - 1 downto n - 2) );
                                izquierda( n / 2 + 1 downto 2 ) := ( others => '0');
                                resto := ( others => '0');
                                raiz2 := ( others => '0');
                                xaux := unsigned( x );
                                nl := n / 2;
                                fin <= '0';
                        elsif       nl /= 0 then
                                if ( resto( n / 2 + 1 ) = '1') then
                                                resto := izquierda + derecha;
                                        else
                                                resto := izquierda - derecha;
                                end if;
                                raiz2( n / 2 - 1 downto 0 ) := raiz2( n / 2 - 2 downto 0 ) & ( not resto(n / 2 + 1)
);
                                if nl = 1 then
                                        fin <= '1';
                                end if;
                                nl := nl - 1;
                                raiz <= std_logic_vector( raiz2 );
                                derecha(0) := '1';
                                derecha(1) := resto( n / 2 + 1 );
                                derecha( n / 2 + 1 downto 2) := raiz2;
                                xaux( n - 1 downto 0 ) := xaux( n - 3 downto 0 )  & "00" ;
                                izquierda( 1 downto 0 ) := xaux( n - 1 downto n - 2);
                                izquierda( n / 2 + 1 downto 2 ) := resto( n / 2 - 1 downto 0);
                        end if;
                end if;
        end process;
end Mixta;
```

Sintetizar ambas opciones. Simular y sacar conclusiones.