CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

ELECTRÓNICA DIGITAL

Unidad 1

Prácticas de Laboratorio

Agosto, 2013.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital

PRACTICAS DE LABORATORIO

Unidad 1: Descripción y Simulación de Circuitos Digitales Utilizando VHDL

Profesor Titular: Dr. Mario Alfredo Reyes Barranca
Profesores Auxiliares: Dr. Oliverio Arellano Cárdenas
M. en C. Luis Martín Flores Nava

Revisión 2.0, Febrero 2013.

INDICE

Este curso incluye 9 prácticas, de las cuales las tres primeras son sobre ejercicios vistos en la clase, la siguiente revisa un módulo que tiene internamente el FPGA y las cinco últimas tratan las diferentes interfaces y periféricos que tiene la tarjeta de desarrollo.

Herramientas de desarrollo y equipo empleado

Software

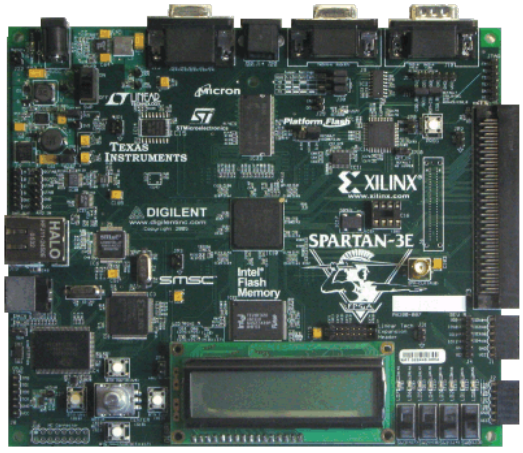
ISE WebPACK 14.2 de Xilinx

ISE Simulator (ISim)

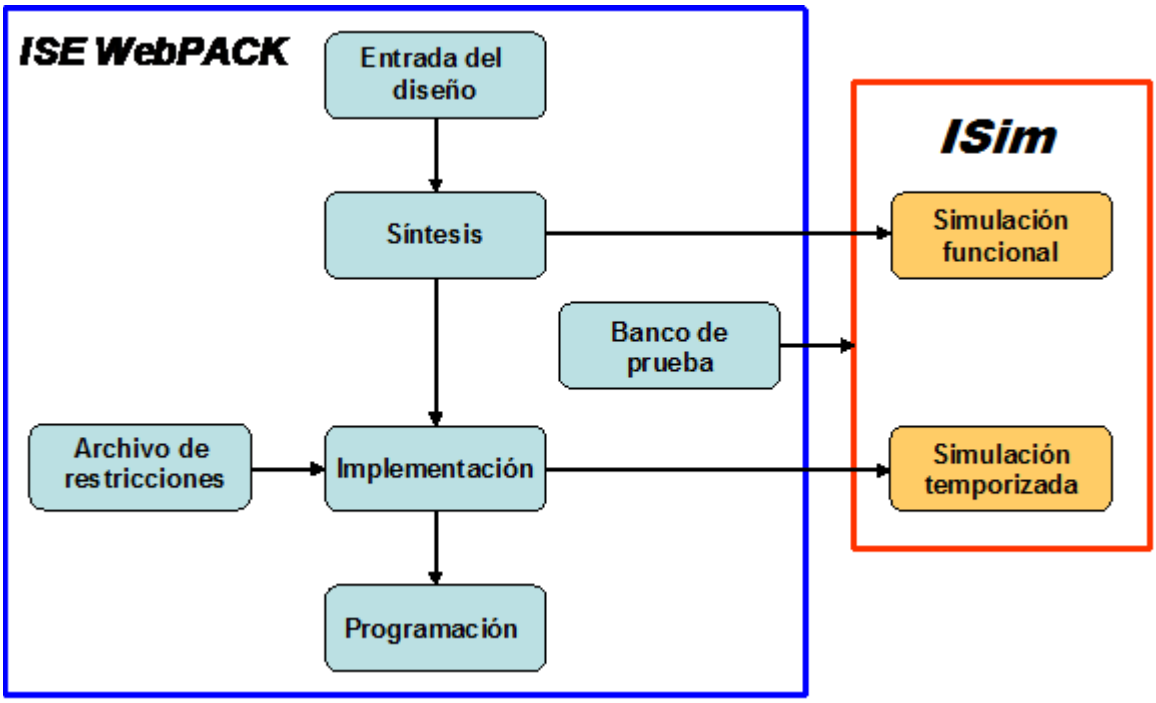
Hardware

Computadora personal con procesador Corei5

Tarjeta de Desarrollo Spartan-3E



Flujo de Diseño para Lógica Programable



Entrada del diseño.

Corresponde a la descripción del sistema a implementar ya sea con lenguaje de alto nivel (VHDL), en forma esquemática, o ambos. Esto se almacena en un archivo con extensión *.vhd* que sirve de entrada a la siguiente etapa.

Síntesis.

Esta herramienta permite realizar una revisión de sintaxis del código fuente *.vhd* de nuestro proyecto y genera un reporte de advertencias y errores en caso de existir.

Banco de prueba.

Para poder simular el circuito primero se crea un banco de pruebas (testbench waveform) donde se definen los estímulos de entrada, en función de los cuales se obtendrá la salida que deberá coincidir con el valor esperado de acuerdo a la funcionalidad del circuito. Estos estímulos incluyen señales de reloj, contadores (ascendentes y descendentes), etc.

Simulación funcional.

Consiste en realizar una simulación puramente lógica del sistema que se pretende realizar, para lo cual se emplea el banco de pruebas definido previamente. Esta simulación no toma en cuenta retardos propios de los componentes con los que se implementará físicamente el sistema, por lo que es una simulación ideal.

Archivo de restricciones.

Este archivo consta de las especificaciones de las terminales del FPGA, el tipo de señal que se va a emplear, voltajes y la señalización que se realizará al momento de descargar el archivo a la tarjeta de desarrollo.

Implementación.

Después de haber creado la fuente del diseño lógico en el primer paso del diagrama de flujo, el proceso de implementación convierte este diseño (con todas las fuentes existentes en forma jerárquica) en un archivo de formato físico que puede ser programado en el dispositivo. Cabe mencionar que los detalles de implementación de un diseño CPLD difieren de aquellos pertenecientes a un FPGA.

En la implementación se llevan a cabo las siguientes tareas:

- ***Traducción***

Combina todas las listas de entrada y restricciones de diseño en una base de datos genérica y nativa de Xilinx (archivo *.NGD*), la cual describe el diseño lógico ya reducido mediante primitivas lógicas.

- ***Mapeo***

Convierte la definición lógica realizada por un archivo *.NGD* a elementos FPGA, tales como CLBs e IOBs, etc. La salida es un archivo de descripción nativa del circuito (*.NCD*), que representa físicamente el diseño de los componentes como elementos FPGA.

- ***Colocación y enrutado***

A partir del archivo *.NCD*, se determina cuáles serán y cómo se interconectarán los componentes del dispositivo FPGA que se utilizarán para implementar el diseño, y se produce un nuevo archivo *.NCD*.

Simulación temporizada.

Esta simulación se realiza tomando en cuenta los retardos propios de los componentes y las rutas de interconexión generados en la etapa previa, con la finalidad de determinar si la operación de la entidad es correcta en términos de tiempo y desempeño.

Programación.

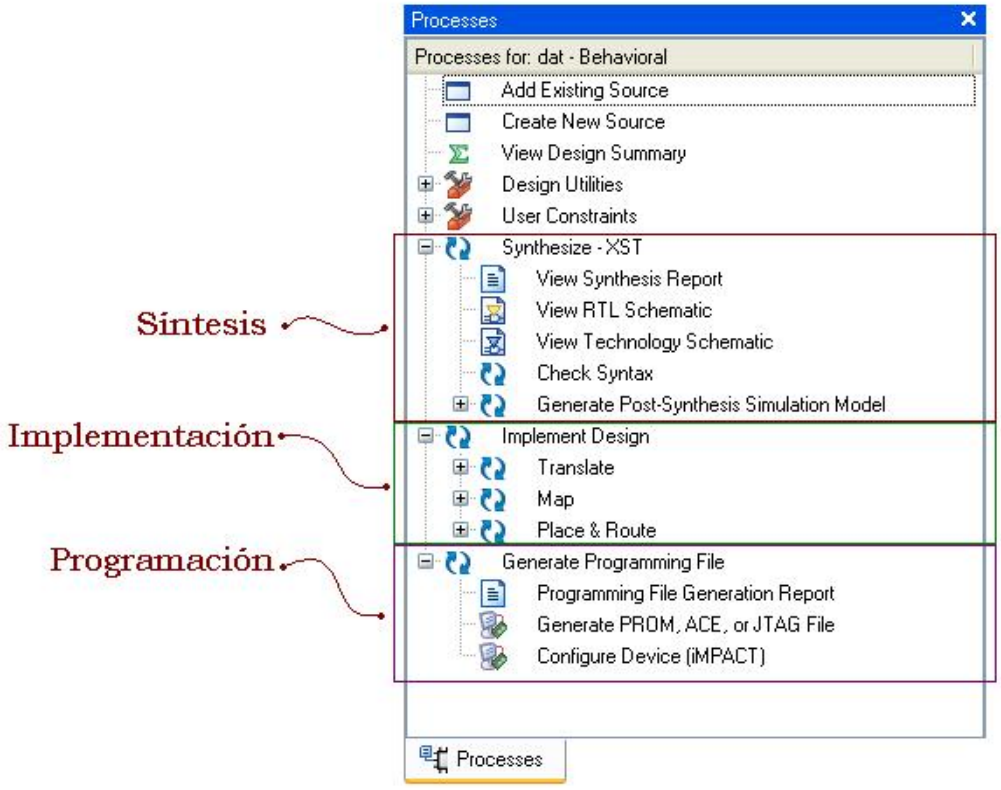
Este proceso, genera un archivo de programación *.bit* que permite descargar la información del diseño, generada en la etapa de implementación, en el dispositivo FPGA, para lo cual debe cumplirse con los procesos anteriores. Comprende los siguientes procesos:

- ***Generar archivo PROM, ACE o JTAG***

Genera el archivo ejecutable ya sea de tipo PROM, ACE o JTAG, el cual se desea implementar, una vez creado este archivo generalmente de tipo BIT o ISC, se puede generar un reporte mostrando las características del mismo.

- ***Comunicación y descarga a la tarjeta.***

Se realiza la descarga del programa ejecutable hacia la tarjeta de práctica, o cualquier dispositivo reconocido por la herramienta ISE mediante la opción "iMPACT", la cual enlazará y determinará las conexiones lógicas de la tarjeta previamente determinadas en el archivo de restricción.



Localización de las herramientas dentro del cuadro Processes.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 1
Titulo: Circuitos Lógicos Combinatorios

Objetivo: El alumno pondrá en práctica las distintas técnicas de programación en VHDL mediante tres ejemplos, en los cuales se implementarán algunas de las estructuras o enunciados más comunes dentro del lenguaje con la finalidad de que se familiarice con el programa, así como con la tarjeta de desarrollo.

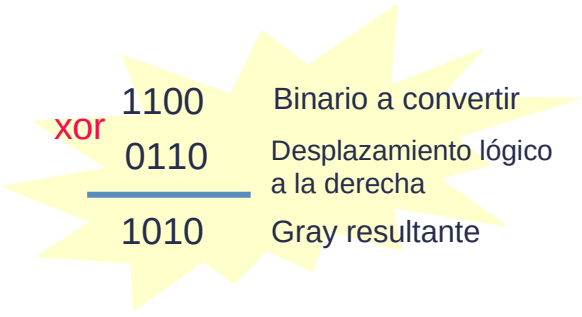
Ejemplo 1. Convertidor de Código Binario a Gray.

Antecedentes: El Código Gray es un caso particular del sistema binario. Consiste en una ordenación de 2^n números binarios de tal forma que cada número sólo tenga un dígito binario distinto a su predecesor. Esta técnica de codificación se originó cuando los circuitos lógicos digitales se realizaban con válvulas de vacío y dispositivos electromecánicos. Los contadores necesitaban potencias muy elevadas a la entrada y generaban picos de ruido cuando varios bits cambiaban simultáneamente. El uso de código Gray garantizó que en cualquier transición variaría tan sólo un bit. En la actualidad, el código Gray se sigue empleando para el diseño de cualquier circuito electrónico combinatorio, ya que el principio de diseño de buscar transiciones más simples y rápidas entre estados sigue vigente, a pesar de que los problemas de ruido y potencia se hayan reducido.

b3	b2	b1	b0	g3	g2	g1	g0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Es posible realizar esta conversión mediante una operación lógica XOR entre el número binario a convertir y el mismo número con un desplazamiento lógico a la derecha.

Ejemplo:



Desarrollo: Mediante la tabla de verdad anterior, realizar el código VHDL correspondiente que cumpla con los requerimientos de la misma, emplear el enunciado "*with-select-when*" empleando vectores de entrada y de salida para los datos.

Descripción en VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity convertidor is
Port ( b : in   STD_LOGIC_VECTOR (3 downto 0);
       g : out  STD_LOGIC_VECTOR (3 downto 0));
end convertidor;

architecture Arq_convertidor of convertidor is

begin

with b select
g <=    "0000"  when "0000",
"0001"  when "0001",
"0011"  when "0010",
"0010"  when "0011",
"0110"  when "0100",
"0111"  when "0101",
"0101"  when "0110",
"0100"  when "0111",
"1100"  when "1000",
"1101"  when "1001",
"1111"  when "1010",
"1110"  when "1011",
"1010"  when "1100",
"1011"  when "1101",
"1001"  when "1110",
"1000"  when "1111",
"0000"  when others;

end Arq_convertidor;
```

Una vez verificado el código, realizar el flujo de diseño para lógica programable que se presentó al inicio de este documento.

Ejercicio extra-clase: Realizar la descripción en VHDL del convertidor de código de Binario a Gray mediante el algoritmo de conversión mencionado.

Ejemplo 2. Comparador de magnitud.

A1	A0	B1	B0	Z1	Z0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

Antecedentes: Los circuitos comparadores son sistemas combinatorios que comparan la magnitud de dos números binarios de *n* bits e indican cuál de ellos es mayor, menor o si existe igualdad entre ellos. Dependiendo del número de bits a comparar, será la relación del comparador. Existen comparadores de 4 bits y de 8 bits. Además de las correspondientes entradas de datos, disponen de tres entradas más que pueden informar sobre una situación anterior, y que se usan para conectar en cascada distintos comparadores, de manera que pueda construirse uno de mayor capacidad.

Desarrollo: Mediante la tabla de verdad, realizar el código VHDL correspondiente que cumpla con los requerimientos de la misma, emplear el enunciado "*if-then-elsif-then*" empleando vectores de entrada y de salida para los datos.

Descripción en VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comparador is
    Port ( A,B : in   STD_LOGIC_VECTOR (1 downto 0);
              Z : out  STD_LOGIC_VECTOR (1 downto 0));
end comparador;

architecture Arq_comparador of comparador is

begin

process (A,B)
    begin
    if (A = B) then
        Z <= "11";
    elsif (A < B) then
        Z <= "01";
    else
        Z <= "10";
    end if;
end process;

end Arq_comparador;
```

Una vez verificado el código, realizar el flujo de diseño para lógica programable que se presentó al inicio de este documento.

Ejercicio extra-clase: Realizar un comparador de magnitud para dos vectores de 4 bits mediante operadores lógicos.

Ejemplo 3. Multiplicador de magnitud.

Antecedentes: Es un circuito digital capaz de multiplicar dos palabras de *m* y *n* bits para obtener un resultado de *n* + *m* bits. A veces el tamaño del resultado está limitado al mismo tamaño que las entradas.

Multiplicador paralelo. Es el más rápido y está formado por una matriz de lógica combinatoria que, a partir de todas las combinaciones posibles de las entradas, genera sus productos a la salida.
Suma y desplazamiento. Está formado por un sumador y un registro de desplazamiento. El sumador comienza con el valor del multiplicando y lo va desplazando y le vuelve a sumar el multiplicando cada vez que el bit correspondiente del multiplicador vale 1. El tiempo en realizar esta operación es igual al número de bits por el periodo de la señal de reloj.

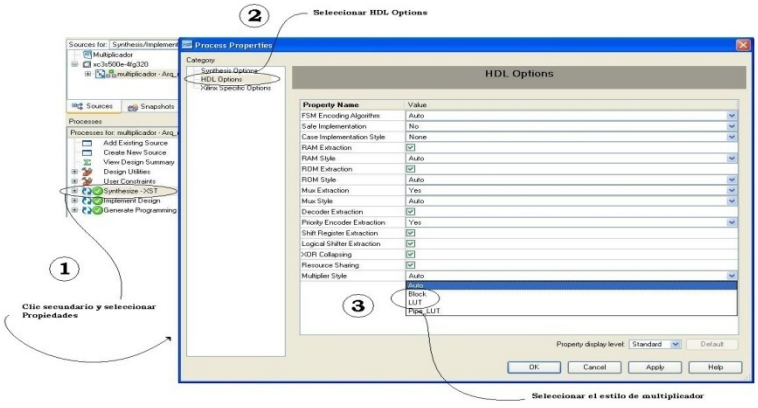
Desarrollo: Realizar un multiplicador de magnitud para dos señales de 2 bits mediante operadores aritméticos.

Descripción en VHDL:

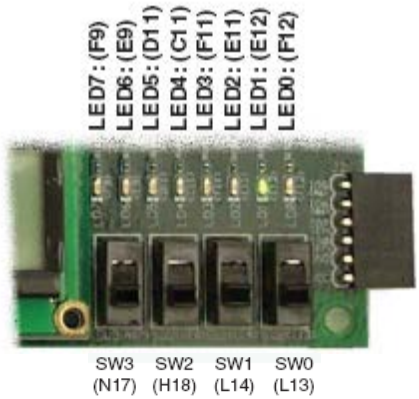
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity multiplicador is
    Port ( A,B : in   STD_LOGIC_VECTOR (1 downto 0);
              P : out  STD_LOGIC_VECTOR (3 downto 0));
end multiplicador;

architecture Arq_multiplicador of multiplicador is

begin

P <= A * B;

end Arq_multiplicador;
```

Una vez verificado el código, realizar el flujo de diseño para lógica programable que se presentó al inicio de este documento.

Ejercicio extra-clase: Realizar el ejercicio anterior modificando los vectores de entrada a 17 bits y cambiando las propiedades del módulo Synthesize-XST, en las modalidades Block y LUT.



Archivos de restricciones del usuario para los ejemplos 1, 2 y 3, respectivamente:




```
# Terminales para el convertidor de código
NET "b<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "b<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "b<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
NET "b<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;
NET "g<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "g<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "g<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "g<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# Terminales para el comparador de magnitud
NET "B<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "B<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "A<0>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
NET "A<1>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;
NET "Z<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "Z<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# Terminales para el multiplicador de magnitud
NET "B<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "B<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "A<0>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
NET "A<1>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;
NET "P<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "P<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "P<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "P<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
```

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 2
Titulo: Circuitos Lógicos Secuenciales

Objetivo: El alumno realizará diseños de máquinas de estado, utilizando las sentencias *case* e *if-then-else*, además aprenderá el uso de una herramienta gráfica especializada para la construcción de este tipo diseños *(StateCAD)*.

Ejemplo 1. Contador ascendente/descendente con inicialización

Un contador es un circuito secuencial construido a partir de biestables y compuertas lógicas capaz de realizar el cálculo de los pulsos que recibe en la entrada destinada a tal efecto, almacenar datos o actuar como divisor de frecuencia.

Desarrollo: Basándose en el principio de funcionamiento de un contador binario antes mencionado, implementar un programa el cual realice un conteo ascendente para un vector de 4 bits, además de un conteo descendiente para el mismo, dependiendo la posición que conserve un selector. Y respetando una señal de reset con la cual se reinicia la secuencia del programa.

Una vez verificado el código, realizar el flujo de diseño para lógica programable que se presentó al inicio de este documento.

Descripción en VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador is
    Port ( clk,rst,ud : in  STD_LOGIC;
                      q : inout  STD_LOGIC_VECTOR
(3 downto 0));
end contador;

architecture Arq_contador of contador is

signal clkdiv: std_logic_vector (25 downto 0);

begin

process (clk,rst,clkdiv)
begin
    if rst='1' then
       clkdiv <= (others => '0');
    elsif clk'event and clk='1' then
        clkdiv <= clkdiv + 1 ;
    end if;
end process;

process (clkdiv(25),rst,q,ud)
begin
    if rst='1' then
       q <= "0000";
    elsif clkdiv(25)'event and clkdiv(25)='1'
then
        if ud ='1' then
           q <= q + 1;
        else
            q <= q - 1;
        end if;
    end if;
end process;

end Arq_contador;
```

Ejercicio extra-clase: Cambiar la descripción del contador para que la secuencia contenga números impares solamente.

Ejemplo 2. Máquina de Estados (Dado digital)

Antecedentes: Se denomina **máquina de estados** a un sistema cuyas señales de salida dependen no sólo del estado de las señales de entrada actuales sino también de las señales de salida anteriores que han configurado un cierto "estado".
El estado del sistema depende del estado anterior y del estado de las entradas en ese instante. En ocasiones los estados son sucesivos monótonamente. En otros casos, existen desvíos a estados anteriores o posteriores. Los cambios de estado son gobernados por una señal de pulsos sincrónicos o asincrónicos.

Desarrollo: Crear una máquina de estados que entregue pseudo aleatoriamente los valores de un dado *(del 1 al 6)* al oprimir un botón.

Descripción en VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dado is
    port (clk, rst, btn: in std_logic;
           d: out std_logic_vector(2 downto 0));
end;

architecture Arq_dado of dado is
    type estados is (S1,S2,S3,S4,S5,S6);
    signal presente, futuro : estados;
begin

process (clk,rst,btn)
    begin
       if rst = '1' then
           presente <= S1;
        elsif clk'event and clk = '1' and btn = '1' then
            presente <= futuro;
        end if;
end process;

process (presente)
    begin
    case presente is
        when S1 =>  d <= "001"; futuro<=S2;
        when S2 =>  d <= "010"; futuro<=S3;
        when S3 =>  d <= "011"; futuro<=S4;
        when S4 =>  d <= "100"; futuro<=S5;
        when S5 =>  d <= "101"; futuro<=S6;
        when S6 =>  d <= "110"; futuro<=S1;
        when others => null;
    end case;
end process;

end Arq_dado;
```

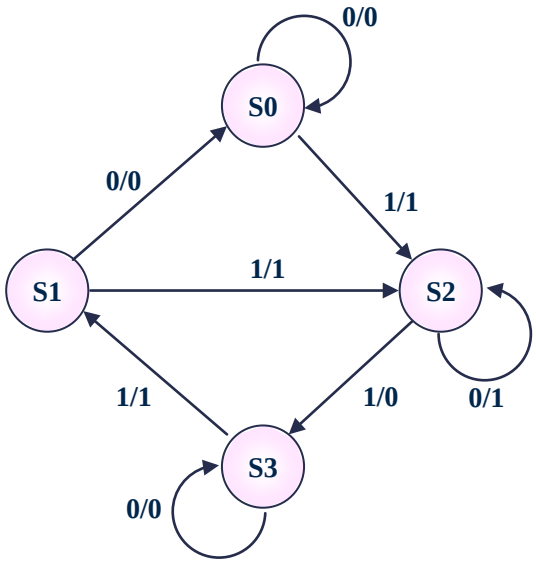
Una vez verificado el código, realizar el flujo de diseño para lógica programable que se presentó al inicio de este documento.

Ejercicio extra-clase: Cambiar el código de tal forma que uno de los números del dado tenga una probabilidad del 0.5 con respecto a los demás de 0.1, inicialmente todos tienen una probabilidad de 1/6.

Ejemplo 3. Máquina de Estados (Tipo Mealy)

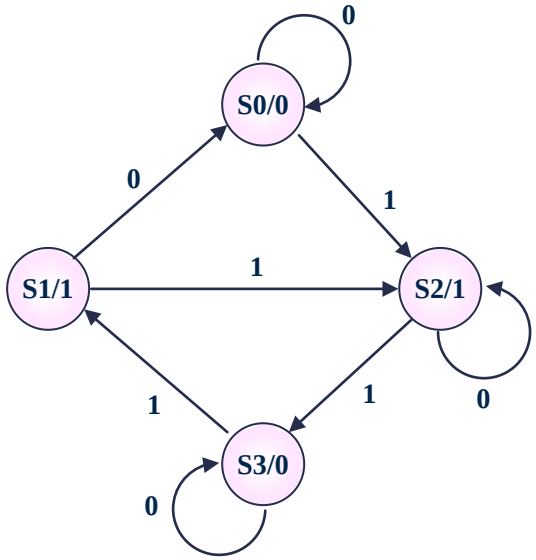
Antecedentes: Se denomina **máquina de estados** tipo Mealy aquella que sus salidas dependen del estado que guarde el sistema y de los valores de las señales de entrada.

Desarrollo: Capturar una máquina de estados tipo Mealy con la herramienta StateCAD.

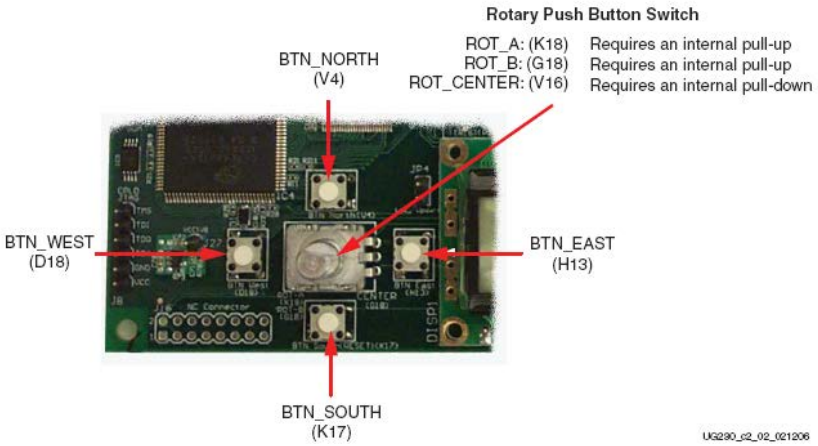


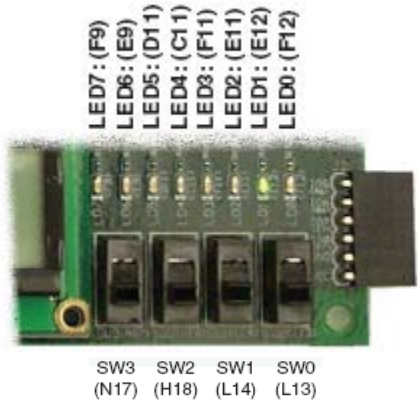
Una vez generado el código VHDL, realizar una simulación funcional para verificar el funcionamiento de la máquina de estados.

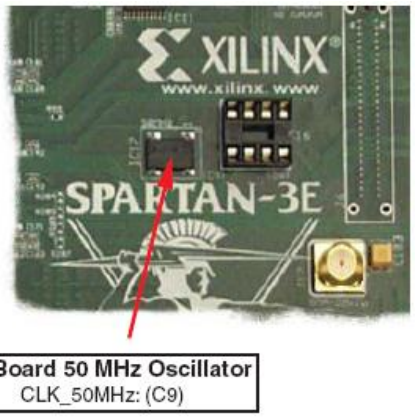
Ejercicio extra-clase: Ahora capturar una **máquina de estados** tipo Moore, vista en clase, generar el código VHDL y realizar una simulación funcional.



Archivos de restricciones del usuario:








```
# Terminales para el contador de 4 bits
NET "clk"  LOC = "C9"  | IOSTANDARD = LVCMOS33 ;
NET "ud"   LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "rst"  LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
```

```
# Terminales para el dado digital
NET "clk"  LOC = "C9"  | IOSTANDARD = LVCMOS33 ;
NET "btn"  LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "rst"  LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "d<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "d<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "d<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

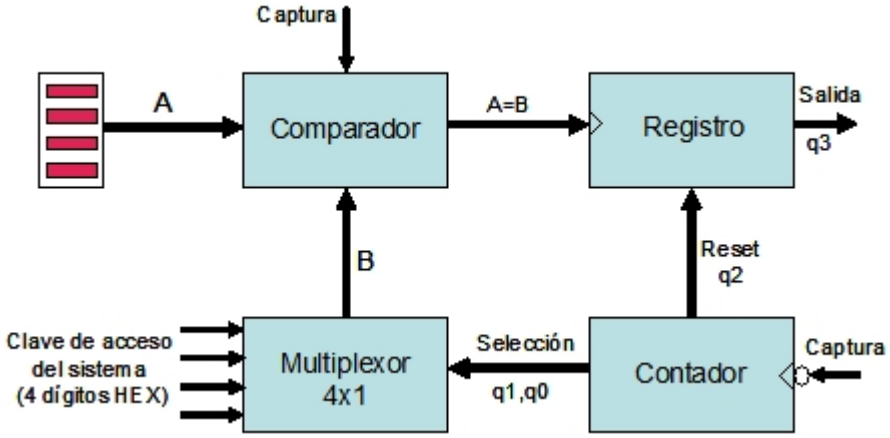
CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 3
Titulo: Diseño Jerárquico.

Objetivo: El alumno realizará un sistema digital (cerradura electrónica) empleando la metodología jerárquica.

Antecedentes: Una cerradura electrónica es un dispositivo que permite el acceso o la apertura de un sistema, siempre y cuando la clave o combinación que se ingrese coincida con la que se encuentra predefinida en dicha cerradura, para lo cual se propone el siguiente diagrama de componentes y conexiones**.**



Donde:

Multiplexor (4x1): Cuenta con cuatro entradas de información de 4bits respectivamente, las cuales se encuentran preestablecidas en el sistema, la entrada de selección es tomada de la salida del contador (los 2 bits menos significativos), con lo cual se realiza un ciclo entre dichos datos conforme el contador se incrementa.
Comparador: Compara dos entradas (A y B) de 4bits respectivamente cada vez que se cuenta con un nivel alto en su entrada *Captura*, de tal modo que si la comparación A = B es verdadera se obtiene un 1 en su salida.
Registro (4bits): Realiza un corrimiento desde el LSB hasta el MSB cada vez que se muestra un flanco de subida en su entrada, la cual se toma de la salida del comparador. Además de contar con una entrada de reset para reiniciar el sistema a su estado inicial.
Contador (3bits): Incrementa su magnitud en 1 cada vez que la entrada *Captura* envía un flanco de bajada en su señal.
Cerradura: Es la entidad principal y su función es mostrar un “1” en el MSB de su salida cuando el registro contenga “1111” en su señal de salida, de tal modo, que si el MSB del contador es igual a “1” y no se ha llenado el registro, el sistema se reinicia.

Ejemplo 1. Cerradura electrónica

Desarrollo: Crear cada uno de los componentes antes mencionados y verificar su funcionamiento de manera independiente.

Descripción en VHDL de los componentes:

Comparador de magnitud

```
entity COMPARADOR is
    Port(DATOA,DATOB: in STD_LOGIC_VECTOR(3 downto 0);
         CAPTURA : in  STD_LOGIC;
         COMP : out  STD_LOGIC);
end COMPARADOR;

architecture Behavioral of COMPARADOR is

begin

process (CAPTURA, DATOA, DATOB)
begin
  if CAPTURA ='1' and (DATOA = DATOB) then
      COMP <= '1';
  else
      COMP <= '0';
  end if;
end process;

end Behavioral;
```

Multiplexor

```
entity MULTIPLEXOR is
    Port ( DATO0 : in STD_LOGIC_VECTOR (3 downto 0);
           DATO1 : in STD_LOGIC_VECTOR (3 downto 0);
           DATO2 : in STD_LOGIC_VECTOR (3 downto 0);
           DATO3 : in STD_LOGIC_VECTOR (3 downto 0);
           CONT  : in STD_LOGIC_VECTOR (1 downto 0);
           MUX: out STD_LOGIC_VECTOR (3 downto 0));
end MULTIPLEXOR;

architecture Behavioral of MULTIPLEXOR is
begin

MUX <=      DATO0 when  CONT    = "00"  else
            DATO1 when  CONT    = "01"  else
            DATO2 when  CONT    = "10"  else
            DATO3;

end Behavioral;
```

Contador

```
entity CONTADOR is
    Port (CAPTURA : in  STD_LOGIC;
          RESET  : in  STD_LOGIC;
          CONTEO : inout STD_LOGIC_VECTOR (2 downto 0));
end CONTADOR;

architecture Behavioral of CONTADOR is

begin

process (CAPTURA,RESET, CONTEO)
begin
 if RESET = '1' or CONTEO(2) = '1' then
    CONTEO <= (others =>'0');
 elsif (CAPTURA'event and CAPTURA = '0') then
    CONTEO <= CONTEO + 1;
 end if;

end process;

end Behavioral;
```

Registro

```
entity REGISTRO is
    Port(PULSO: in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        CON   : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
        REGOUT: inout STD_LOGIC_VECTOR(3 downto 0));
end REGISTRO;

architecture Behavioral of REGISTRO is

begin

process (PULSO, RESET, CON, REGOUT)
begin
    if RESET = '1' then
        REGOUT <= (others=>'0');
    elsif CON(2)='1' then
         REGOUT(2 downto 0) <= (others=>'0');
    elsif PULSO'event and PULSO = '1' then
        REGOUT <= REGOUT (2 downto 0) & '1';
    end if;
end process;

end Behavioral;
```

Una vez realizados y verificados los componentes se procede a crear el Paquete de componentes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package MODULOS is

COMPONENT COMPARADOR is
    Port ( DATOA, DATOB : in  STD_LOGIC_VECTOR (3 downto 0);
           CAPTURA : in  STD_LOGIC;
           COMP : out  STD_LOGIC);
end COMPONENT;

COMPONENT CONTADOR is
    Port ( CAPTURA  : in  STD_LOGIC;
           RESET    : in  STD_LOGIC;
           CONTEO   : inout  STD_LOGIC_VECTOR (2 downto 0));
end COMPONENT;

COMPONENT MULTIPLEXOR is
    Port (  DATO0   : in    STD_LOGIC_VECTOR (3 downto 0);
            DATO1   : in    STD_LOGIC_VECTOR (3 downto 0);
            DATO2   : in    STD_LOGIC_VECTOR (3 downto 0);
            DATO3   : in    STD_LOGIC_VECTOR (3 downto 0);
            CONT    : in    STD_LOGIC_VECTOR (1 downto 0);
            MUX     : out   STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;

COMPONENT REGISTRO is
    Port ( PULSO    : in  STD_LOGIC;
           RESET    : in  STD_LOGIC;
           CON      : in  STD_LOGIC_VECTOR (2 DOWNTO 0);
           REGOUT   : inout STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;

COMPONENT ANTIRREBOTE is
    Port ( CLK      : in  STD_LOGIC;
           CAPTURA  : in  STD_LOGIC;
           CAP     : out STD_LOGIC);
end COMPONENT;


end MODULOS;
```

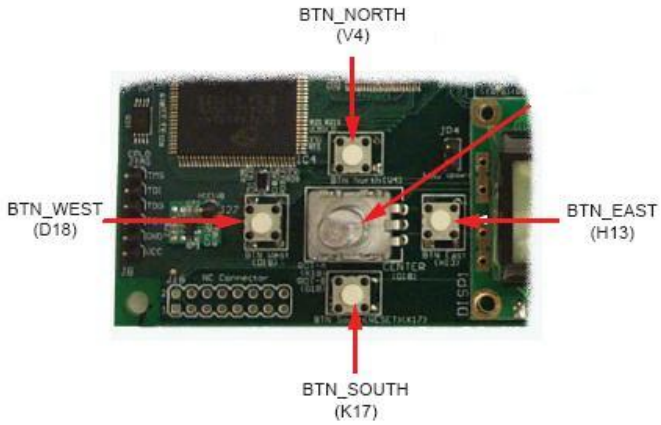
Por último se describe la entidad principal (nivel jerárquico superior) en donde se hace el llamado de los componentes y se especifican sus interconexiones.

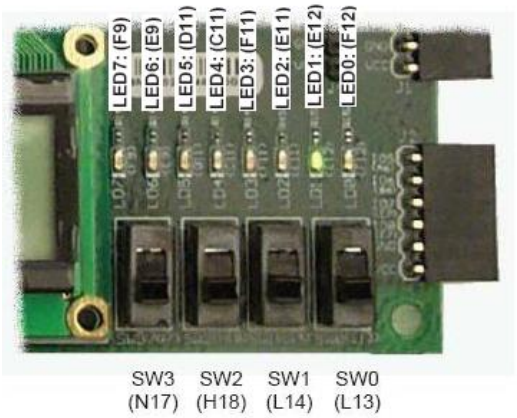
```
entity CERRADURA is
    Port ( ENTRADA : in  STD_LOGIC_VECTOR (3 downto 0);
           RST : in  STD_LOGIC;
           CLK : in STD_LOGIC;
           CAP : in  STD_LOGIC;
           LEDS: out  STD_LOGIC_VECTOR (3 downto 0));
end CERRADURA;

architecture Behavioral of CERRADURA is

SIGNAL COMPARADOR1  : STD_LOGIC;
SIGNAL CONTADOR1    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL MULTIPLEXOR1 : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL REGISTRO1    : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL ANTIR1       : STD_LOGIC;

begin

MODULO_1 : COMPARADOR   PORT MAP ( DATOA=>ENTRADA, DATOB=>MULTIPLEXOR1, CAPTURA=>ANTIR1, COMP=>COMPARADOR1 );
MODULO_2 : CONTADOR     PORT MAP ( CAPTURA=>ANTIR1, RESET=>RST, CONTEO=>CONTADOR1 );
MODULO_3 : MULTIPLEXOR  PORT MAP ( DATO0=>X"A", DATO1=>X"F", DATO2=>X"8", DATO3=>X"0", CONT=>CONTADOR1(1 DOWNTO 0),
MUX=>MULTIPLEXOR1);
MODULO_4 : REGISTRO     PORT MAP ( PULSO=>COMPARADOR1, RESET=>RST, CON=>CONTADOR1, REGOUT=>REGISTRO1 );
MODULO_5 : ANTIRREBOTE  PORT MAP ( CLK=>CLK, CAPTURA=>CAP, CAP=>ANTIR1 );

LEDS <= REGISTRO1;

end Behavioral;
```

Ejercicio extra-clase: Realizar el diseño anterior mediante captura esquemático (creando los símbolos de cada componente).

Archivos de restricciones del usuario:

```
NET "CLK"         LOC = "C9"  | IOSTANDARD = LVCMOS33 ;
NET "CAP"         LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "RST"         LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "ENTRADA(0)" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "ENTRADA(1)" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "ENTRADA(2)" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "ENTRADA(3)" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
NET "LEDS(3)"     LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS(2)"     LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS(1)"     LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS(0)"     LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```





Nota: Debido a que la operación de los push-button generan una frecuencia de ruido cada vez que son activados, y tomando en cuenta que en la implementación de esta práctica es necesaria una señal discreta en el pulso de captura, resulta necesaria la implementación de una función que elimine dichos rebotes, la cual se muestra a continuación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ANTIRREBOTE is
    Port ( CLK      : in  STD_LOGIC;
           CAPTURA  : in  STD_LOGIC;
           CAP      : out STD_LOGIC);
end ANTIRREBOTE;

architecture Behavioral of ANTIRREBOTE is

SIGNAL CNT: STD_LOGIC_VECTOR (1 DOWNTO 0) := (OTHERS => '0');

begin

process (CLK,CNT,CAPTURA)
  begin
    if CAPTURA = '0' then
      CNT <= "00";
    elsif (CLK'event and CLK = '1') then
      if (CNT /= "11") then
        CNT <= CNT + 1;
        end if;
    end if;
    if (CNT = "10") and (CAPTURA = '1') then
        CAP <= '1';
     else
        CAP <= '0';
     end if;
end process;

end Behavioral;
```

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 4
Titulo: Administrador de Reloj Digital (DCM)

Objetivo: El alumno empleará los IP cores para describir en VHDL algunas funciones del módulo administrador de reloj digital (**D**igital **C**lock **M**anager) que incluye el Spartan 3E.

Antecedentes: Un DCM consiste en cuatro unidades funcionales interrelacionadas, como se muestra en la figura 1: un DLL (**D**elay-**L**ocked **L**oop), un DFS (**D**igital **F**recuency **S**ynthesizer), un PS (**P**hase **S**hifter) y un SL (**S**tatus **L**ogic). Las tres funciones principales del DCM son:

1. **Eliminación del corrimiento de reloj**: este corrimiento es típicamente causado por la red de distribución de reloj; es indeseable en aplicaciones de alta frecuencia y es eliminado alineando la fase de la señal de reloj de salida que es generada con respecto a la señal de reloj de entrada.
2. **Sintetizador de frecuencia**: puede generar un amplio rango de frecuencia en la señal de reloj de salida a partir de una señal de reloj de entrada, que se multiplica por un factor como se observa en la ecuación de la tabla 3.
3. **Corrimiento de fase**: es capaz de generar desfasamiento en todas las señales de reloj de salida con respecto a la señal de reloj de entrada.

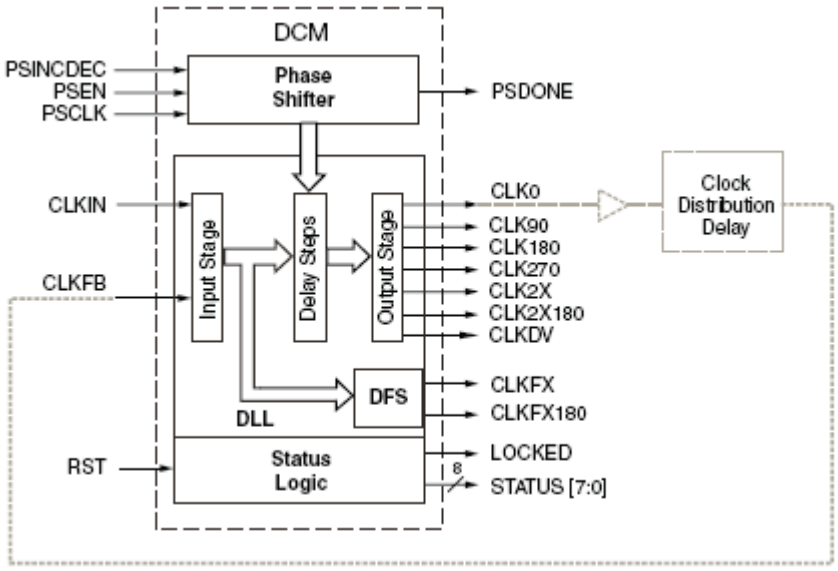


Figura 1. Bloques funcionales y señales asociadas del DCM.

Signal	Direction	Description
CLKIN	Input	Receives the incoming clock signal. See Table 30, Table 31, and Table 32 for optimal external inputs to a DCM.
CLKFB	Input	Accepts either CLK0 or CLK2X as the feedback signal. (Set the CLK_FEEDBACK attribute accordingly).
CLK0	Output	Generates a clock signal with the same frequency and phase as CLKIN.
CLK90	Output	Generates a clock signal with the same frequency as CLKIN, phase-shifted by 90°.
CLK180	Output	Generates a clock signal with the same frequency as CLKIN, phase-shifted by 180°.
CLK270	Output	Generates a clock signal with the same frequency as CLKIN, phase-shifted by 270°.
CLK2X	Output	Generates a clock signal with the same phase as CLKIN, and twice the frequency.
CLK2X180	Output	Generates a clock signal with twice the frequency of CLKIN, and phase-shifted 180° with respect to CLK2X.
CLKDV	Output	Divides the CLKIN frequency by CLKDV_DIVIDE value to generate lower frequency clock signal that is phase-aligned to CLKIN.

Tabla 1. Señales del DLL (**D**elay **L**ocked **L**oop)

Attribute	Description	Values
CLK_FEEDBACK	Chooses either the CLK0 or CLK2X output to drive the CLKFB input	NONE, ***1X***, 2X
CLKIN_DIVIDE_BY_2	Halves the frequency of the CLKIN signal just as it enters the DCM	***FALSE***, TRUE
CLKDV_DIVIDE	Selects the constant used to divide the CLKIN input frequency to generate the CLKDV output frequency	1.5, ***2***, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6.0, 6.5, 7.0, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, and 16
CLKIN_PERIOD	Additional information that allows the DLL to operate with the most efficient lock time and the best jitter tolerance	Floating-point value representing the CLKIN period in nanoseconds

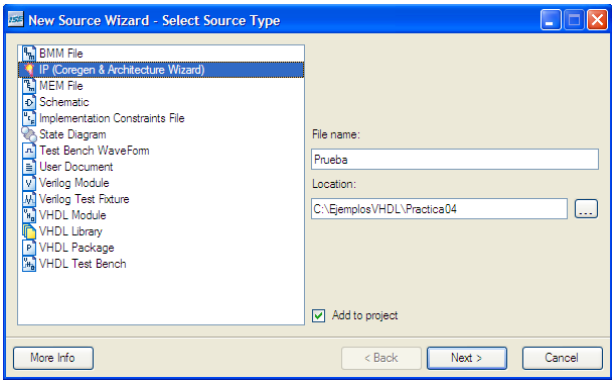
Tabla 2. Atributos del DLL

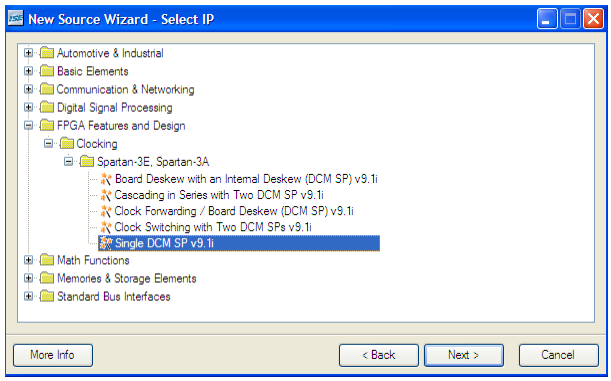
Signal	Direction	Description
CLKFX	Output	Multiplies the CLKIN frequency by the attribute-value ratio (CLKFX_MULTIPLY/ CLKFX_DIVIDE) to generate a clock signal with a new target frequency.
CLKFX180	Output	Generates a clock signal with the same frequency as CLKFX, but shifted 180° out-of-phase.

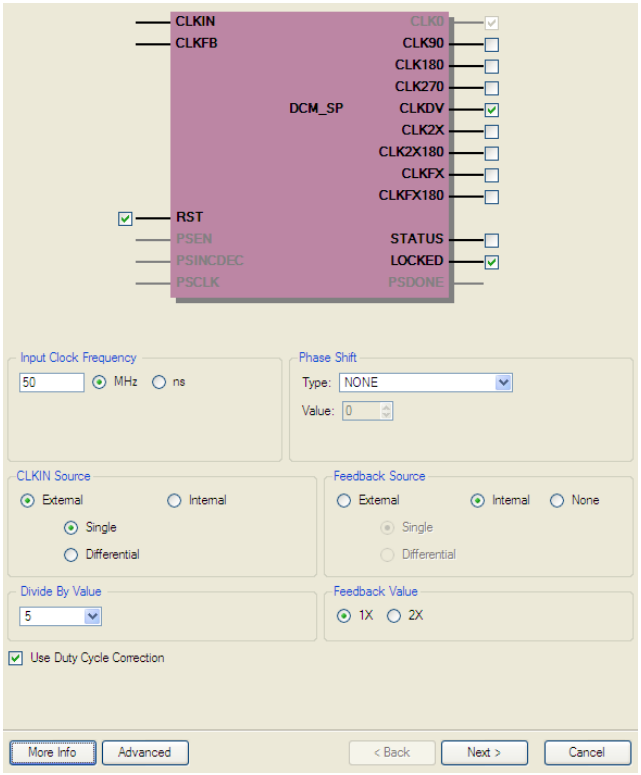
$$f_{CLKFX} = f_{CLKIN} \cdot \left(\frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE} \right)$$

Tabla 3. Señales del DFS (**D**igital **F**requency **S**ynthesizer)

Desarrollo: Mediante la descripción vista anteriormente, seleccione una nueva fuente del tipo IP (Coregen & Architecture Wizard), elija Single DCM y configure sus parámetros.







Descripción en VHDL: Copie y pegue las secciones de declaración de componente y llamado del mismo del archivo generado por el editor de IP Cores.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FrecDiv is
    Port ( clk,rst : in  STD_LOGIC;
           clkdv,clkin,clk0,lock : out  STD_LOGIC);
end FrecDiv;

architecture Behavioral of FrecDiv is

    COMPONENT Prueba
    PORT(
        CLKIN_IN        : IN std_logic;
        RST_IN          : IN std_logic;
        CLKDV_OUT       : OUT std_logic;
        CLKIN_IBUFG_OUT : OUT std_logic;
        CLK0_OUT        : OUT std_logic;
        LOCKED_OUT      : OUT std_logic
        );
    END COMPONENT;

begin

    Inst_Prueba: Prueba PORT MAP(
        CLKIN_IN        => clk,
        RST_IN          => rst,
        CLKDV_OUT       => clkdv,
        CLKIN_IBUFG_OUT => clkin,
        CLK0_OUT        => clk0,
        LOCKED_OUT      => lock
    );

end Behavioral;
```

Actividades complementarias:

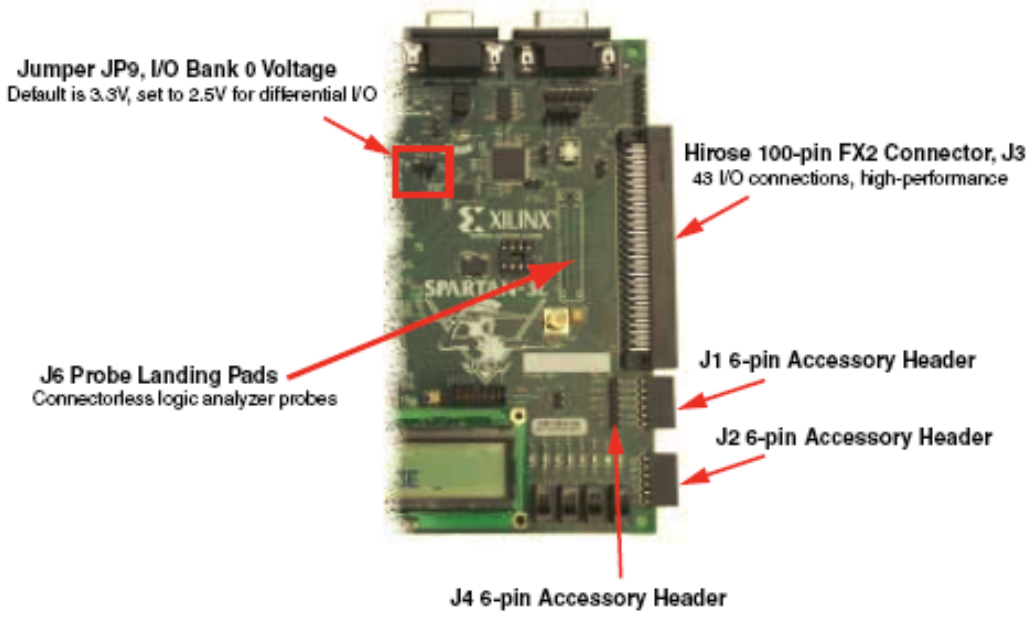
Primeramente, cambie el nivel de corriente de la salida correspondiente a la señal clkdv en el archivo de restricciones de usuario y observe las variaciones en el osciloscopio. Posteriormente, cambie de posición el conector en JP9 y observe la amplitud de la señal clkdv.

	Output Drive Current (mA)					
IOSTANDARD	2	4	6	8	12	16
LVTTL	✓	✓	✓	✓	✓	✓
LVCMOS33	✓	✓	✓	✓	✓	✓
LVCMOS25	✓	✓	✓	✓	✓	-
LVCMOS18	✓	✓	✓	✓	-	-
LVCMOS15	✓	✓	✓	-	-	-
LVCMOS12	✓	-	-	-	-	-

Ejercicio extra-clase: Tomar la señal clkdv, generada anteriormente por el módulo DCM, como señal de reloj para diseñar un contador binario ascendente/descendente de 3 bits, observar las salidas del contador en el osciloscopio.

Archivos de restricciones del usuario:

```
# Terminal del reloj de 50 MHz
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33;

# Terminal de inicialización
NET "rst" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;

# ==== 6-pin header J4 ====
# Estas terminales son compartidas con el conector FX2
NET "clkdv"  LOC = "D7" |IOSTANDARD = LVTTL | SLEW = SLOW  | DRIVE = 6;
NET "clkin" LOC = "C7" |IOSTANDARD = LVTTL | SLEW = SLOW   | DRIVE = 6;
NET "clk0"  LOC = "F8" |IOSTANDARD = LVTTL | SLEW = SLOW   | DRIVE = 6;
NET "lock"  LOC = "E8" |IOSTANDARD = LVTTL | SLEW = SLOW   | DRIVE = 6;
```



CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

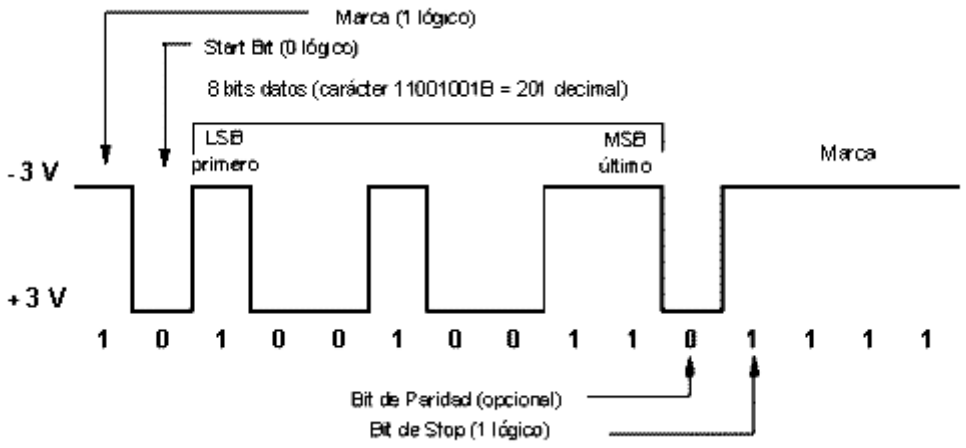
Electrónica Digital (Unidad I)

Práctica No. 5
Titulo: Puerto RS-232.

Objetivo: El alumno realizará un algoritmo para generar un controlador sencillo que permita la transmisión y recepción mediante el puerto RS-232.

Antecedentes: El RS-232 fue originalmente adoptado en 1960 por EIA *(Electronic Industries Association)*. El estándar evolucionó a través de los años y en 1969 la tercera revisión, el RS-232C, fue el estándar elegido por los fabricantes de computadoras personales compatibles con IBM. En 1987 se adoptó la cuarta revisión, el RS-232D, también conocida como EIA-232D.

El Puerto Serie es más difícil de hacer interactuar con dispositivos externos que el Puerto Paralelo. En la mayoría de las ocasiones, incluso, cualquier comunicación realizada a través del Puerto Serie será convertida en una comunicación paralela antes de ser empleada. Esto se logra con un chip denominado "**UART"** (Universal Asynchronous Receiver/Transmitter), el cual se encarga de permitir la transferencia entre dispositivos. Normalmente la velocidad de conexión va desde 300 baudios hasta 115200 baudios aunque la velocidad proporcionada por el UART 8250 (Primer chip Standard) es de 9600 baudios.
Este puerto permite que los cables que se emplean para la comunicación sean más largos, ya que toma como ´1´ cualquier voltaje que se encuentre entre –3 y –25 V y como ´0´, entre +3 y +25 V, a diferencia del puerto paralelo, cuyo rango de voltajes está entre 0 y 5 V. Por esta razón la pérdida de señal introducida por la resistencia intrínseca de los conductores no es un problema para los cables empleados en este tipo de comunicación.
La siguiente figura muestra la estructura de un carácter que se trasmite en forma serial asíncrona.



Nota: Emplear la herramienta HyperTerminal para verificar la correcta ejecución del código.

Ejemplo 1. Transmisión.

Desarrollo: Con base a la descripción presentada anteriormente, realizar el algoritmo que permita el envío de datos a través de puertoRS-232 de la tarjeta SPARTAN3E, empleando una velocidad de transmisión de 9600 baudios, tomando como datos de envío únicamente los caracteres *numéricos y vocales mayúsculas.*

Descripción en VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Trans is
    Port ( clk,rst: in  STD_LOGIC;
               tx : out  STD_LOGIC);
end Trans;


architecture Arq_Trans of Trans is

signal regtx:       STD_LOGIC_VECTOR( 7 downto 0);
signal cnttx:       STD_LOGIC_VECTOR(15 downto 0):="0000000000000000";
signal ttx:         STD_LOGIC_VECTOR( 3 downto 0):="0000";
signal caracter:    STD_LOGIC_VECTOR( 7 downto 0);
constant baudtx:    STD_LOGIC_VECTOR(15 downto 0):="0001010001011000"; --9600 bauds

begin


-- Reloj de transmisión
process (clk,rst,cnttx)begin
      if (rst='1')then
            cnttx<= baudtx - 1;
            ttx <= "0000";
            caracter <= "00000000";
      elsif(caracter = "00001111") then -- n caracter + 1
            cnttx<= baudtx - 1;
            ttx <= "0000";
      elsif (clk'event and clk='1')then
                  cnttx<=cnttx+1;
                  if (cnttx=baudtx)then
                        ttx<=ttx+1;
                        cnttx<=(others=>'0');
                        if(ttx="1010")then
                              caracter<=caracter+1;
                        end if;
                  end if;
      end if;
end process;
```

```
-- Asignacion de Caracter
with caracter select
             regtx <=     X"30"  when "00000000", --0
                          X"31"  when "00000001", --1
                          X"32"  when "00000010", --2
                          X"33"  when "00000011", --3
                          X"34"  when "00000100", --4
                          X"35"  when "00000101", --5
                          X"36"  when "00000110", --6
                          X"37"  when "00000111", --7
                          X"38"  when "00001000", --8
                          X"39"  when "00001001", --9
                          X"41"  when "00001010", --A
                          X"45"  when "00001011", --E
                          X"49"  when "00001100", --I
                          X"4F"  when "00001101", --O
                          X"55"  when "00001110", --U
                          X"00"  when others;


-- Protocolo de transmisión
with ttx select
             tx     <=     '1'    when "0000",
                          '0'    when "0001",   --bit de start
                          regtx(0) when "0010",--inicia transmision de dato
                          regtx(1) when "0011",
                          regtx(2) when "0100",
                          regtx(3) when "0101",
                          regtx(4) when "0110",
                          regtx(5) when "0111",
                          regtx(6) when "1000",
                          regtx(7) when "1001",--fin de transmision de dato
                          '1'    when "1010",  --bit de stop
                          '1'    when others;

end Arq_Trans;
```

Ejemplo 2. Recepción.

Desarrollo: Con base a la descripción presentada anteriormente, realizar el algoritmo que permita la recepción de datos a través del puerto RS-232.

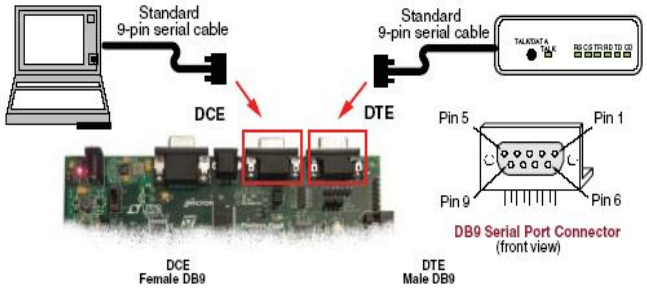
Descripción en VHDL:

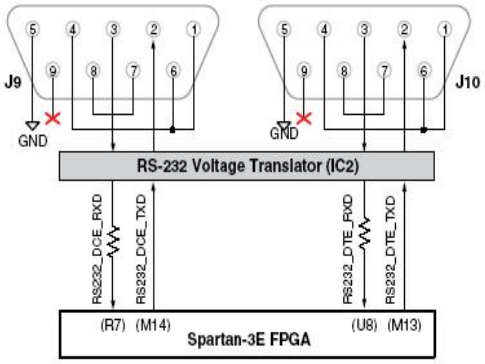
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Recep is
    Port ( clk,rst,rx: in  std_logic;
           leds: out std_logic_vector(7 downto 0));
end Recep;
```

```
architecture Arq_Recep of Recep is

signal    erx:      std_logic:='0';
signal    trx:      std_logic_vector(4 downto 0);
signal    regrx:    std_logic_vector(7 downto 0):="00000000";
signal    cntrx:    std_logic_vector(10 downto 0);
constant baudrx:    std_logic_vector(10 downto 0):="11011001000";--(9600*3)bauds
begin

process (rst,rx,trx)
begi
       if rst='1' then
              erx <= '0';
       elsif rx='0' then           -- Detecta el bit de inicio
              erx <= '1';          -- Habilita la recepción
       elsif trx="11011" then
              erx <= '0';          -- Deshabilita la recepción
       end if;
end process;


process (clk,rst,cntrx,trx,erx)begin
       if (rst='1' or erx='0')then
              cntrx <= (others=>'0');
              trx   <= (others=>'0');
       elsif (clk'event and clk='1' and erx='1')then
                  cntrx <= cntrx + 1;
                      if (cntrx=baudrx)then
                             trx <= trx + 1;
                             cntrx<=(others=>'0');
                      end if;
       end if;
end process;


process (clk,rst,trx,rx,regrx)
begin
   if rst='1' then
              regrx <= (others=>'0');
              leds  <= (others=>'0');
       elsif (clk'event and clk='1') then
              case (trx) is
                     when "00100" =>  regrx(0) <=      rx;
                     when "00111" =>  regrx(1) <=      rx;
                     when "01010" =>  regrx(2) <=      rx;
                     when "01101" =>  regrx(3) <=      rx;
                     when "10000" =>  regrx(4) <=      rx;
                     when "10011" =>  regrx(5) <=      rx;
                     when "10110" =>  regrx(6) <=      rx;
                     when "11001" =>  regrx(7) <=      rx;
                     when "11010"  => leds    <= regrx; -- Carga código ASCII a los leds
                     when others => null;
              end case;
        end if;
end process;

end Arq_Recep;
```

Ejercicio extra-clase: Diseñar e implementar el código que permita efectuar tanto transmisión como recepción de manera simultánea empleando diseño jerárquico.

Componentes requeridos.





Convertidor USB – Serial DB9

On-Board 50 MHz Oscillator
CLK_50MHz: (C9)

Archivos de restricciones del usuario:

```
#CLK INTERNO DE 50 MHz
NET "clk" LOC = "C9"  | IOSTANDARD = LVCMOS33;
#RESET DE USUARIO
NET "rst" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;

#--------------------------------------#
# Terminales para la Recepción de datos #
#--------------------------------------#
#PUERTOS DE COMUNICACION SERIAL RS232
NET "rx" LOC = "R7" | IOSTANDARD = LVTTL;

#LEDS PARA MOSTRAR CODIGO ASCII RECIBIDO
NET "leds<7>" LOC = "F9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<6>" LOC = "E9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "leds<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

#---------------------------------------#
# Terminales para la Transmisión de datos #
#---------------------------------------#
#PUERTOS DE COMUNICACION SERIAL RS232
NET "tx"   LOC = "M14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW;
```

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 6
Titulo: Sensores y Actuadores.

Objetivo: El alumno realizara un algoritmo para la recepción de un código infrarrojo (SONY de 12 bits), además aprenderá a controlar servomotores en VHDL a partir de la técnica de Modulación por Ancho de Pulso (PWM)

Ejemplo 1. Lector de mando Infrarrojo

Antecedentes: La manera más económica de controlar un dispositivo de manera remota dentro de un rango visible es utilizando luz infrarroja. Casi todos los equipos de audio y video pueden ser controlados de esta manera, debido a su amplio campo de aplicación, los componentes requeridos para la implementación de proyectos son de bajo costo.

La luz infrarroja es luz normal pero con un color particular. El ojo humano no puede percibir este color debido a que su longitud de onda de 950 nm está por debajo del espectro visible. Esta es otra razón por la que es utilizada para aplicaciones de control remoto, que se desea utilizar mas no es importante verla. Otra razón es que los leds infrarrojos son de fácil fabricación y por lo tanto son económicos.

Desafortunadamente existen muchas fuentes de luz infrarroja. El sol es la más grande de todas, pero hay otras como: focos, velas, sistemas térmicos e incluso el cuerpo humano. De hecho cualquier objeto que radie calor, también radiara luz infrarroja. Es por esto que es necesario tomar precauciones para garantizar que la información infrarroja llegue al receptor sin distorsión.

Modulación y métodos de Codificación

Para asegurar que una señal infrarroja llegue correctamente a su destino, o que el dispositivo destino reciba solo la señal correcta, es necesario modular. Los controles remotos infrarrojos utilizan modulación por código de pulso donde la frecuencia portadora se encuentra en el rango de 30 KHz a 58 KHz.

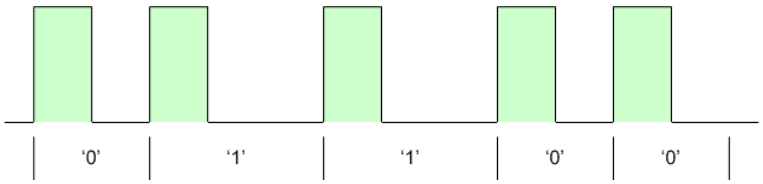
En términos de transmisión, la modulación se refiere a poner al led infrarrojo en corte o en conducción de manera rápida a razón de la frecuencia portadora. El receptor estará sincronizado solo con esta frecuencia para asegurar que se reciba solo la señal requerida, ya que utiliza esta frecuencia para demodular la señal.

Cuando el led infrarrojo no emite luz, el transmisor está en estado bajo y la salida del receptor será un estado alto; durante la emisión de luz el transmisor se encuentra en estado alto y la salida del receptor en estado bajo.

Los métodos de codificación determinan la manera de representar el ‘0’ y el ‘1’ lógicos en función de los estados alto y estados bajo. Los métodos más utilizados en sistemas de control remoto se presentan a continuación.

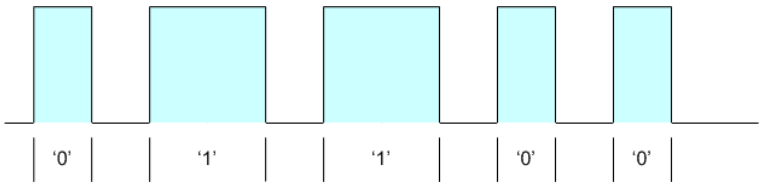
Codificación por separación de pulso

En este método, el estado alto es constante para todos los pulsos, sin embargo el tiempo en estado bajo es distinto dependiendo del ‘0’ o del ‘1’ lógicos transmitidos. El estado bajo para un ‘1’ lógico es mayor que para el ‘0’ lógico.



Codificación por ancho de pulso

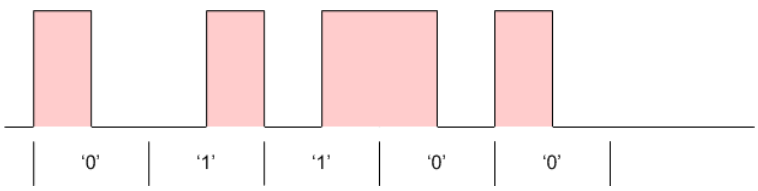
En este método de codificación, la duración en estado alto es diferente para el ‘0’ y el ‘1’ lógicos, siendo mayor el tiempo para el ‘1’ lógico.



Código Manchester

En este método de codificación, todos los bits tienen la misma duración, con la mitad de su periodo en estado alto y la mitad en estado bajo. Un ‘0’ lógico se representa con el estado alto durante la primera mitad de su tiempo de bit y el estado bajo en la segunda mitad, presentando a mitad de periodo una transición de alto a bajo.

Un ‘1’ lógico se representa con el estado bajo durante la primera mitad de su tiempo de bit y el estado alto en la segunda mitad, presentando a mitad de periodo una transición de bajo a alto.

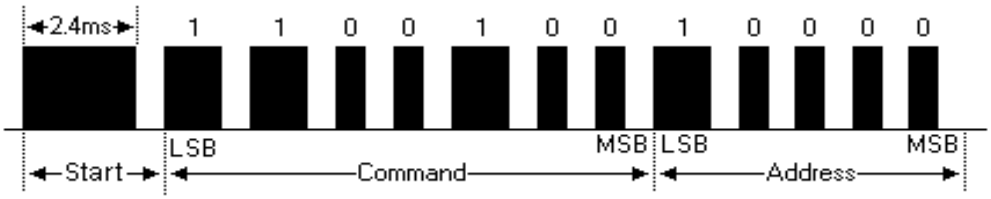


Protocolo SIRC de SONY

Las características principales de este protocolo son las siguientes.

- La portadora modulada deriva de una frecuencia de 480 kHz, siendo 1/12 de esta Frecuencia con 1/3 de ciclo útil.
- Cuando la información se transmite repetidamente, el ciclo de cada trama es de 45 ms.
- Cada trama se compone de un pulso de sincronía, un código de datos de 7 bits y un código de identificación de dispositivo de 5 bits.

Los tiempos definidos para la forma de onda del código de salida se muestran a continuación.



Ejemplo de una trama

Dato	Tiempo (s)	Tiempo (no. De periodos)
Pulso de sincronía	2.4 ms	8T
Dato en estado bajo	0.61 ms	2T
Dato en estado alto (0)	0.59 ms	2T
Dato en estado alto (1)	1.19 ms	4T
Periodo de dato (0)	1.2 ms	4T
Periodo de dato (1)	1.8 ms	6T
Ciclo de trama	45 ms	150T

Donde T= 0.3 ms

En la siguiente tabla se muestran algunos de los comandos correspondientes a cada tecla y el código binario asignado a los leds.

Comando	**Botón** Pulsado
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	0
16	Chan+
17	Chan -
18	Vol +
19	Vol -

Desarrollo: Realizar un algoritmo que lea el comando de un mando infrarrojo con protocolo SIRC y lo muestre en 7 leds, apoyándose en la teoría antes expuesta

Descripción en VHDL: Invertimos la señal de entrada para una mejor interpretación

La función de los procesos es la siguiente:

1. Genera una señal de referencia de 100us.
2. Detecta el pulso de sincronía y genera un reset interno.
3. Cuenta el número de flanco de bajada de nuestra señal invertida sony.
4. Escanea el tiempo en alto para determinar si es un dato ‘0’ o ‘1’.
5. Genera el vector de datos del comando recibido.
6. Despliega el comando en los leds, solo si ha terminado la trama.

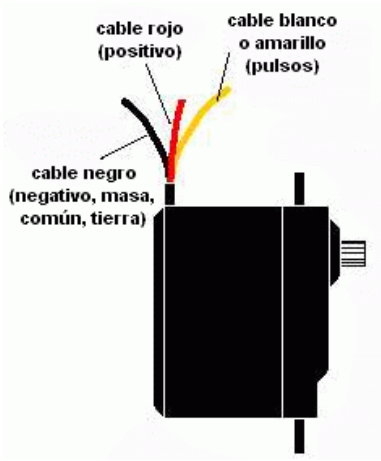
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Receptor is
    port ( ent,clk,rst: in   STD_LOGIC;
           leds: out STD_LOGIC_VECTOR (6 downto 0));
end Receptor;

architecture Arq_Receptor of Receptor is

signal vector: STD_LOGIC_VECTOR (12 downto 0):="0000000000000";--Vector que almacena la trama
signal npulsos: STD_LOGIC_VECTOR (3 downto 0):="0000";          --Bandera de cuenta de pulsos
signal clkdiv: STD_LOGIC_VECTOR (12 downto 0):="0000000000000";--Señal de referencia de 100us
signal nref:   STD_LOGIC_VECTOR (4 downto 0):="00000";          --Evalua la duración en alto
signal pivote: STD_LOGIC_VECTOR (4 downto 0):="00000";--Recibe la cuenta de nref, asigna 0 o 1
signal rstin:  STD_LOGIC:='0';--Señal de reinicio de trama
signal sony:  STD_LOGIC:='0';--Invierte la salida del receptor para leer los 12 bits correctamente

begin

-- Señal de entrada invertida (protocolo Sony standard)
sony <= not ent ;

-- Señal de referencia de 100us
process(clk,rst,clkdiv,rstin) begin

if rst='1' or clkdiv="1001110001000" or rstin ='1' then
       clkdiv <=(others => '0');
elsif clk 'event and clk = '1' then
              clkdiv <= clkdiv + 1;
end if;

end process;

-- Se detecta el pulso de sincronia
process(clkdiv(12),rst,nref,rstin) begin

if rst='1' or rstin='1'  then
              rstin <='0';
elsif clkdiv(12)'event and clkdiv(12)='0' then
       if nref = "01111" then
              rstin <= '1';
       end if;
end if;

end process;
```

```
-- Se cuenta el numero de pulsos a partir del flanco de bajada del pulso de sincronia
process (sony,rst,npulsos,rstin) begin

if rst='1' or rstin ='1' then
          npulsos <= "0000";
elsif sony'event and sony='0' then
               npulsos <= npulsos + 1;
end if;

end process;

-- Contador de tiempo en estado alto
process(clkdiv(12),rst,nref,sony,rstin) begin

if rst='1' or rstin ='1'  then
               nref <=(others => '0');
               pivote  <=(others => '0');


elsif clkdiv(12)'event and clkdiv(12)='0' then
       if sony = '1' then
                      nref <= nref+1;
       elsif sony = '0' then
                      pivote <= nref;
                      nref<=(others => '0');
       end if;
end if;

end process;

-- Generación del vector de datos
process (clkdiv(12),rst,pivote,rstin)

variable i: integer range 0 to 13 := 0 ;
begin
if rst = '1' or rstin ='1' then
       vector <=(others => '0');
       i:= 0;
elsif clkdiv(12)'event and clkdiv(12)='1' and pivote > "0000" then
       if pivote  < "1000" then
                      vector(i)<= '0';
                      i:= i+1;
       else
                      vector(i)<= '1';
                      i:= i+1;
       end if;
end if;

end process;

-- Proceso de despliegue de datos en leds
process (clkdiv(12),rst,npulsos,vector) begin
if rst = '1' then
               leds <= (others => '0');
elsif clkdiv(12)'event and clkdiv(12)='0' then
       if npulsos="1101" then
               leds <= vector(7 downto 1);
       end if;
end if;

end process;

end Arq_Receptor;
```

Ejemplo 2. Control de un servomotor por PWM

Antecedentes: Hoy en día se usan servomotores para posicionar superficies de control, pequeños ascensores, en radio control, títeres y por supuesto en robots. Debido a su diminuto tamaño y su gran capacidad de torque.

El servomotor es un dispositivo que tiene un eje de rodamiento controlado y que puede ser llevado a posiciones angulares específicas al enviar una señal codificada. Si el ciclo de utilidad es constante el servomotor mantendrá la posición.



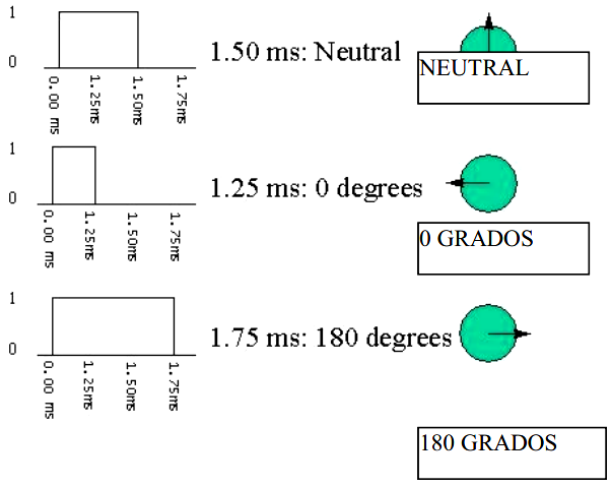
El servomotor tiene algunos circuitos de control y un potenciómetro que está conectada al eje central del servo. Este permite a la etapa de control, supervisar el ángulo actual del servo, si el eje está en el ángulo correcto, entonces el motor está apagado. Si el circuito checa que el ángulo no es el correcto, el motor girará en la dirección adecuada hasta llegar al ángulo correcto.
Normalmente este tipo de motores trabajan en un rango de 0-180º ±10º. La cantidad de voltaje aplicado al motor es proporcional a la distancia que necesita viajar.

Para comunicarnos con el servomotor debemos hacerlo por la terminal de control, aquí le asignamos un ángulo en el que debe posicionarse dependiendo del ancho del pulso. En esta terminal es decir usamos:

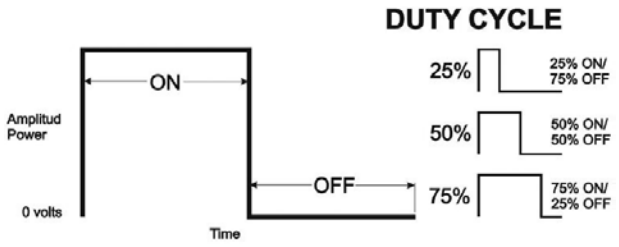
La Modulación por ancho de pulso (PWM) que es una técnica en la que se modifica el ciclo de trabajo de una señal periódica comúnmente utilizada para controlar la potencia en dispositivos eléctricos. También es utilizado en algunos sistemas de comunicación donde debido a su ciclo de trabajo son usados para transmitir información a través de un canal de comunicaciones.

Si queremos modificar dicha posición en un servomotor debemos variar el ancho del pulso que por lo regular debe estar entre 1ms y 2ms haciendo excepciones según el fabricante, es decir que tenemos un rango de 0-180º y 1-2ms siendo la posición neutral 1.5ms es decir 90º. El servo espera ver un pulso cada 13-20ms es decir que el tiempo off es muy grande comparado con el pulso.

Esto se ve más claro en las siguientes imágenes.



Relación ancho de pulso- posición



Tiempo ON/OFF

Desarrollo: Realizar un algoritmo que controle un servomotor con 2 botones pulsadores en un rango de 0-180º y una resolución de 5º.

Descripción en VHDL: Se requiere de un componente antirrebote para esta práctica similar al de la practica 3.

La función de los procesos es la siguiente.

1. Genera una señal de referencia de 10us con 50% ciclo utilidad.

2. Genera una señal de 12ms (tiempo ON y OFF), pregunta por la coincidencia de a y aux entonces viene el flanco de bajada de la señal PWM.

3. Incrementa o decrementa en 5 cada vez que un pulsador es presionado según sea el caso, además acota los limites para el servomotor en 0º y 180º.

Se asignan las señales de cada modulo en el componente antirrebote.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PWM is
    Port ( clk,rst,btderecha,btizquierda : in  STD_LOGIC;
              pwm_servo : out std_logic);
end PWM;

architecture arq_PWM of PWM is

component  ANTIRREBOTE is
    Port ( CLK      : in  STD_LOGIC;
           CAPTURA  : in  STD_LOGIC;
           CAP       : out STD_LOGIC);
end component;

signal clk100K: std_logic;
signal cnt: std_logic_vector (12 downto 0);
signal aux: std_logic_vector (11 downto 0);
signal a: std_logic_vector (11 downto 0);
signal DER: std_logic;
signal IZQ: std_logic;
begin

-- Periodo PWM ( 50% duty cicle)
process (clk,rst,cnt)
begin
    if rst='1' or cnt=X"1F3" then -- 20ns * 500 ciclos = referencia 10us
       cnt <= (others => '0');
        clk100K <= '1';
    elsif clk'event and clk='1' then
        cnt <= cnt + 1 ;
        if cnt= X"0F9" then --20ns * 249 ciclos = 1/2 Periodo
            clk100K <= '0';
        end if;
    end if;
end process;

process (clk100K,rst,aux)
begin
    if rst='1' or aux=X"5DC" then  -- Tiempo off (12ms)
       aux <= (others => '0');
        pwm_servo <= '1';
    elsif clk100K'event and clk100K='1' then
        aux <= aux + 1 ;
        if aux = a then
            pwm_servo <= '0';
        end if;
    end if;
end process;

process (clk100k,DER,IZQ,rst,a)
    begin
        if rst ='1' then
          a <= x"091";      --1.45ms
        elsif clk100k'event and clk100k ='1' then
            if DER = '1' then
                    if a = x"0EB" then  --2.35ms
                    a <= x"0EB";
                else
                    a <= a+5;
                end if;
            end if;

            if IZQ ='1' then
                if a = x"037" then  --0.55ms
                    a <= x"037";
                else
                    a <= a-5;
                end if;
            end if;
        end if;
end process;
ant1: ANTIRREBOTE port map (clk100k,btderecha,DER);
ant2: ANTIRREBOTE port map (clk100k,btizquierda,IZQ);
end arq_PWM;
```

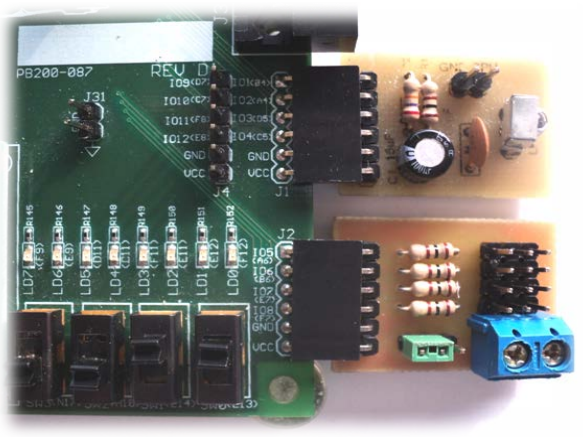
El componente antirrebote es el siguiente, tome en cuenta que la señal de reloj que recibe es la señal de referencia de 10us.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ANTIRREBOTE is
    Port ( CLK      : in  STD_LOGIC;
           CAPTURA  : in  STD_LOGIC;
              CAP       : out STD_LOGIC);
end ANTIRREBOTE;

architecture Behavioral of ANTIRREBOTE is

SIGNAL CNT: STD_LOGIC_VECTOR (10 DOWNTO 0) := (OTHERS => '0');

begin

process (CLK,CNT,CAPTURA)
  begin
    if CAPTURA = '0' then
      CNT <= "00000000000";
    elsif (CLK'event and CLK = '1') then
      if (CNT /= "11111111111") then
        CNT <= CNT + 1;
        end if;
    end if;
    if (CNT = "10011011000") and (CAPTURA = '1') then
        CAP <= '1';
     else
        CAP <= '0';
     end if;
end process;

end Behavioral;
```

Ejercicio extra-clase: Diseñar un algoritmo que permita posicionar el servomotor en un rango de 0-180º con una resolución de 10º, los comandos serán enviados por un mando infrarrojo utilizar las teclas numéricas y las de navegación. Se debe emplear diseño jerárquico.

Componentes requeridos.

Módulos para sensor y actuador



Header J1 y J2

Mando Infrarrojo

Servomotor

Consideraciones: Los Header J1 y J2 proporcionan una alimentación de 3.3V insuficientes para la mayoría de los servomotores convencionales. Sin embargo la tarjeta SPARTAN 3E cuenta en su conector Hirose 100pin con una terminal de 5V (Fig 1).
Se sugiere seleccionar mediante el jumper en el modulo de servomotores la alimentación externa es decir a través del borne (Fig 2).

Fig 1. Conector Hirose 100pin, terminal 50(5V).

Fig 2. Jumper a la derecha Mod. Servomotores.

Table 15-1: **Hirose 100-pin FX2 Connector Pinout and FPGA Connections (J3)**

Signal Name	FPGA Pin	Shared Header Connections					FX2 Connector		FPGA Pin	Signal Name
		LED	J1	J2	JP4	J6	A (top)	B (bottom)		
5.0V							49	49		5.0V
5.0V							50	50		SHIELD

Archivos de restricciones de usuario para ejemplo 1 y 2 respectivamente:

```
# Terminal de entrada de comando
NET "ent" LOC = "B4" | IOSTANDARD = LVTTL;
NET "ent" CLOCK_DEDICATED_ROUTE = FALSE;
# Terminal del Reloj de 50 Mhz
NET "clk"  LOC = "C9"  ;
NET "rst"  LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN;
# Despliegue de datos en leds
NET "leds<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<6>" LOC = "E9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;


#Terminal del Reloj de 50 Mhz
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
#Terminal de Reset
NET "rst" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;
#Terminal de los Pulsadores
NET "btderecha"   LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN;
NET "btizquierda" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN;
#Terminal de salida PWM
NET "pwm_servo"LOC = "A6"| IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 12;
```

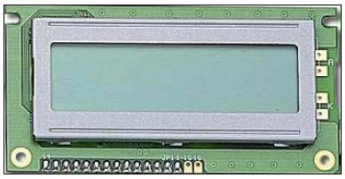
CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 7
Titulo: Pantalla LCD

Objetivo: El alumno diseñará e implementará un algoritmo que permita el control y despliegue de texto mediante la pantalla de cristal liquido (LCD).

Antecedentes: LCD (***L**iquid **C**rystal **D**isplay*) son las siglas en inglés de Pantalla de Cristal Líquido, dispositivo inventado por Jack Janning, quien fue empleado de NCR. Se trata de un sistema eléctrico de presentación de datos formado por 2 capas conductoras transparentes y en medio un material especial cristalino (cristal líquido) que tienen la capacidad de orientar la luz a su paso. Cuando la corriente circula entre los electrodos transparentes con la forma a representar (por ejemplo, un segmento de un número) el material cristalino se reorienta alterando su transparencia.



La tarjeta del Spartan-3E tiene una pantalla de cristal líquido de 2 líneas por 16 caracteres. Este FPGA controla el LCD por una interfaz de datos de 4 bits que se muestra en la Figura 1. A pesar de que este LCD soporta una interfaz de datos de 8 bits, en la tarjeta emplea la interfaz de 4 bits para hacerla compatible con otras tarjetas de Xilinx y para minimizar el número de terminales empleadas.

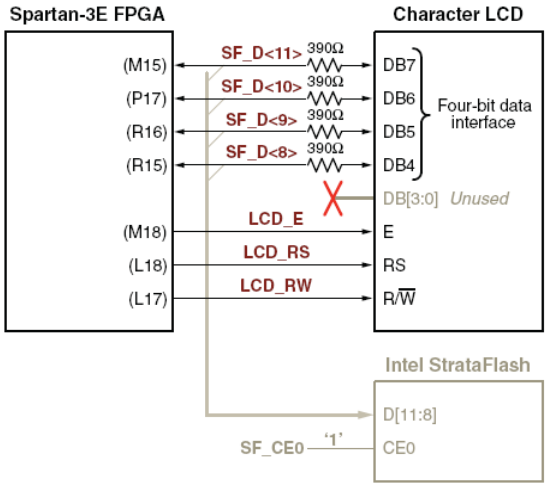


Figura 1. Interfaz del LCD

Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	

Tabla 1. Señales de la interfaz del LCD

Compatibilidad de Voltaje

El LCD se polariza con 5V. Las señales de entrada/salida del FPGA se activan con 3.3V. Sin embargo, los niveles de salida del FPGA son reconocidos como niveles lógicos altos o bajos por el LCD. El controlador del LCD acepta niveles TTL de 5V y las salidas de 3.3V LVCMOS proporcionadas por el FPGA satisfacen los requisitos del nivel voltaico de 5 V TTL. Los resistores en serie de 390Ω en las líneas de datos previenen la sobre-saturación en el FPGA y en los pines de entrada/salida de la StrataFlash cuando el LCD tiene un valor lógico alto. El LCD controla las líneas de datos cuando LCD_RW está en alto. La mayor parte de las aplicaciones de los LCD, toman al dispositivo como un periférico de salida y no leen la información del display.

Interacción con la memoria StrataFlash de Intel

Como se muestra en la Figura 1, las cuatro señales de datos del LCD también se comparten con las líneas de datos de la memoria StrataFlash SF_D<11:8>. Según la tabla 2, la interacción LCD/StrataFlash depende de la aplicación en el diseño. Cuando la memoria StrataFlash es desactivada (SF_CE0 = alto), el FPGA tiene completo acceso lectura/escritura al LCD. Inversamente, cuando la lectura del LCD es desactivada (LCD_RW = bajo), el FPGA tiene completo acceso lectura/escritura a la memoria StrataFlash.

SF_CE0	SF_BYTE	LCD_RW	Operation
1	X	X	StrataFlash disabled. Full read/write access to LCD.
X	X	0	LCD write access only. Full access to StrataFlash.
X	0	X	StrataFlash in byte-wide (x8) mode. Upper data lines are not used. Full access to both LCD and StrataFlash.

Notes:

1. 'X' indicates a don't care, can be either 0 or 1.

Tabla 2. Control de la interacción LCD/StrataFlash

Si la memoria StrataFlash está en modo byte-wide (x8) (SF_BYTE = *bajo*) el FPGA tiene acceso completo y simultáneo de lectura/escritura al LCD y a la memoria StrataFlash. En el modo byte-wide, la memoria StrataFlash no usa las líneas de datos SF_D<15:8>.

Controlador del LCD

El LCD de 2x16 tiene un controlador interno de gráficos (Sitronix ST7066U) que es funcionalmente equivalente con los siguientes dispositivos:

- Samsung S6A0069X o KS0066U
- Hitachi HD44780
- SMOS SED1278

Mapa de Memoria

El controlador tiene tres regiones de memoria internas, cada una con un propósito específico. El LCD debe ser inicializado antes de acceder a cualquiera de las regiones de memoria.

DD RAM

El Display de Datos RAM (DD RAM) almacena el código del carácter que se exhibirá en la pantalla. La mayor parte de las aplicaciones interactúan primero con el DD RAM.

La figura 2 muestra la dirección que tienen por defecto las 32 localizaciones del carácter de exhibición. La línea superior de caracteres se almacena entre las direcciones 0x00 y 0x0F. La segunda línea de caracteres es almacenada entre las direcciones 0x40 y 0x4F.

	Character Display Addresses																Undisplayed Addresses		
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

Figura 2. Direcciones en hexadecimal de la DD RAM

Físicamente hay en total 80 localizaciones en el DD RAM con 40 caracteres disponibles para cada línea. Las localidades (0x10 a la 0x27) y (0x50 a la 0x67) se pueden utilizar para almacenar datos no visibles.

CG ROM

El generador de carácter ROM (CG ROM) contiene el mapa de bits fuente para cada uno de los caracteres predefinidos, que el LCD puede mostrar en la pantalla, figura

3. El código del carácter almacenado en la DD RAM para cada localización del carácter se refiere posteriormente a una posición con el CG ROM.

Figura 3. Conjunto de caracteres del LCD

CG RAM

Generador de carácter de la RAM. Es la memoria volátil de 64 bytes que permite almacenar hasta 8 caracteres personalizados para ser mostrados en la pantalla. Cada localización de carácter personalizado consiste de un mapa de bits de 5 puntos x 8 líneas, como se muestra en la figura 4.

						Upper Nibble				Lower Nibble			
						Write Data to CG RAM or DD RAM							
A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Character Address			Row Address			Don't Care			Character Bitmap				
0	1	1	0	0	0	-	-	-	0	■	0	■	0
0	1	1	0	0	1	-	-	-	■	0	■	0	■
0	1	1	0	1	0	-	-	-	0	■	0	■	0
0	1	1	0	1	1	-	-	-	■	0	■	0	■
0	1	1	1	0	0	-	-	-	0	■	0	■	0
0	1	1	1	0	1	-	-	-	■	0	■	0	■
0	1	1	1	1	0	-	-	-	0	■	0	■	0
0	1	1	1	1	1	-	-	-	0	0	0	0	0

Figura 4. Ejemplo del tablero de ajedrez con el código de carácter 0x03

Conjunto de comandos

La tabla 3 resume los comandos disponibles del controlador del LCD y de los bits. Puesto que el LCD requiere de 4 bits por operación, cada comando de 8 bits se envía como dos palabras de 4 bits. La palabra superior (con los 4 bits más significativos) se transfiere primero, seguida por la palabra inferior (con los 4 bits menos significativos).

INSTRUCCIONES CON EL LCD

Instrucción	Código										Descripción	Tiempo max de ejecución
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Borra el display y coloca el cursor en la primera posición 0 DDRAM	82μs~1.64ms
Return home	0	0	0	0	0	0	0	0	1	*	Coloca el cursor en la posición de inicio y hace que el display comience a desplazarse desde la posición original. El contenido de la DDRAM no varía	40μs~1.64ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Establece el sentido de desplazamiento de la información en el display. Esta operación se realiza durante la lectura o escritura de la DDRAM	40μs
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Activa o desactiva poniendo en ON/OFF tanto el display D=0 (off) o D=1(on),como el cursor C=0(off) o C=1(on) y establece si este debe parpadear o no B=0(off) o B=1(on)	40μs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	*	*	Mueve el cursor y desplaza el display sin cambiar el contenido de la DDRAM	40μs
Funtion set	0	0	0	0	1	DL	N	F	*	*	Establece el tamaño de interfase con el bus de datos(DL),el número de líneas del display(N) y la font de los caracteres	40μs
CG RAM address set	0	0	0	1	Dirección CGRAM						Establece la dirección de CGRAM a partir de la cual se almacenan los caracteres de usuario	40μs
DD RAM address set	0	0	1	Dirección de la DDRAM							Estable la dirección DDRAM a partir de la cual se almacenan los datos a visualizar	40μs
Read Busy Flag and Address	0	1	BF	Dirección de DDRAM o CGRAM							Lectura del flag de Busy e indica de la dirección de la CGRAM o DDRAM última empleada.	1μs
Write data into the CG RAM or the DDRAM	1	0	Dato a escribir								Escribe en DDRAM o CGRAM los datos que se quieren presentar en el LCD	40μs
Read data from the CG RAM or the DDRAM	1	1	Dato a leer								Lee de la DDRAM o CGRAM los datos que se direccionen	40μs

Abreviaturas	Valor	Descripción
S	1	*Desplaza la información cada vez que se escribe un dato*
	0	*Modo normal*
I/D	1	*Incremento del cursor*
	0	*Decremento del cursor*
S/C	1	*Desplaza la información de pantalla*
	0	*Mueve el cursor*
R/L	1	*Desplazamiento a la derecha*
	0	*Desplazamiento a la izquierda*
BF	1	*Módulo ocupado*
	0	*Módulo disponible*
DL	1	*Ducto de datos de 8 bits*
	0	*Ducto de datos de 4 bits*
N	1	*LCD de dos líneas*
	0	*LCD de una línea*
F	1	*Caracter de 5 x 10 puntos*
	0	*Caracter de 5 x 7 puntos*
B	1	*Parpadeo de cursor encendido*
C	1	*Cursor encendido*
D	1	*Pantalla encendido*
X	1	*Interlineado*

Tabla 3. Conjunto de comandos del LCD

Operación *(Interfaz de datos de cuatro bits)*

Este tipo de interfaz es la que emplea la tarjeta. La Figura 5 ilustra una operación de escritura al LCD, mostrando el tiempo mínimo permitido para la precolocación, mantenimiento y habilitación, de acuerdo a la duración relativa del pulso de reloj de 50MHz (20ns de periodo) provisto en la tarjeta.

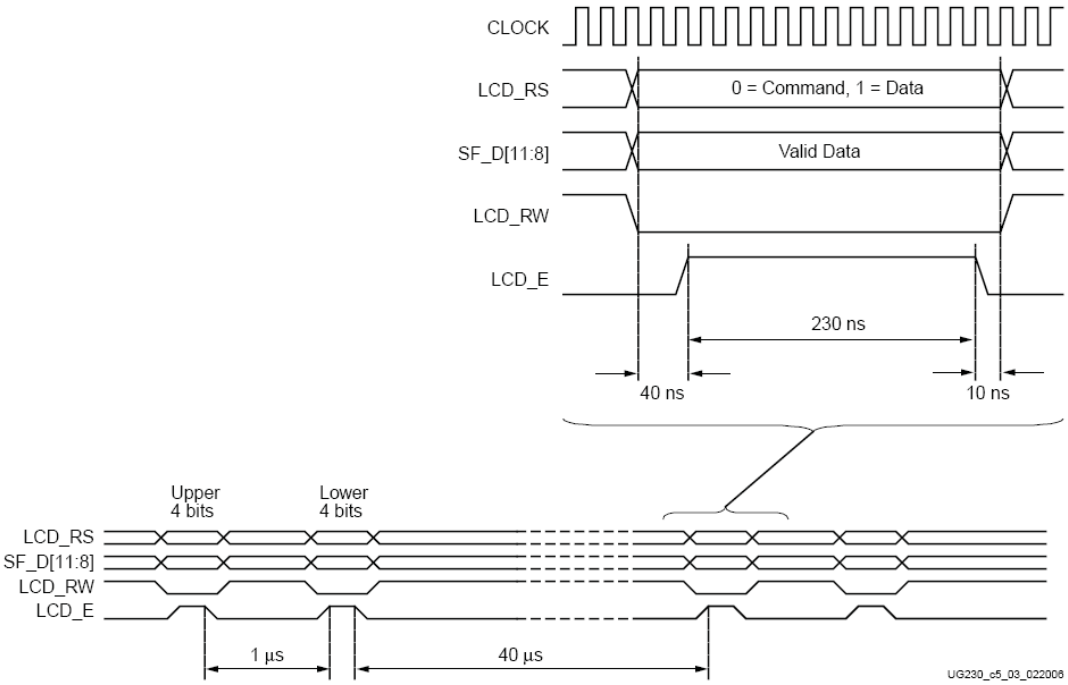


Figura 5. Sincronización de la interfaz del LCD de caracteres

Los valores de los datos sobre SF_D<11:8>, el selector del registro (LCD_RS) y las señales de control de lectura/escritura (LCD_RW) deben ser instalados y estabilizados al menos 40ns antes de que el habilitador LCD_E se ponga en *alto*. La señal de habilitación debe mantenerse en *alto* al menos 230ns o más, el equivalente a 12 o más ciclos de reloj de 50MHz.

En varias aplicaciones, la señal LCD_RW puede conectarse en *bajo*, porque el FPGA generalmente no tiene razón para leer información del LCD.

Transfiriendo datos de 8 bits sobre la interfaz de 4 bits

Después de inicializar el LCD y establecer comunicación, todos los comandos y datos de 8 bits se transfieren al LCD empleando dos operaciones secuenciales de 4 bits, espaciados al menos por 1µs, como se muestra en la Figura 5. Los cuatro bits más significativos se transfieren primero, seguidos de los cuatro menos significativos. Una operación de escritura de 8 bits debe ser espaciada al menos por 40µs antes de la siguiente comunicación. Después de un comando de *Limpiar Pantalla* es necesario dejar un retardo de al menos 1.64ms.

Inicializando del LCD

La secuencia de inicialización establece primero que la aplicación del FPGA desea emplear la interfaz de datos de cuatro bits a la LCD como sigue:

- ✓ Esperar 15ms o más, a pesar de que el display está generalmente listo cuando la configuración del FPGA termina. El intervalo de 15ms es equivalente a 750,000 ciclos de reloj a 50MHz.
- ✓ Escribir SF_D<11:8>=0x3, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- ✓ Esperar 4.1ms o más que es equivalente a 205,000 ciclos de reloj a 50MHz.
- ✓ Escribir SF_D<11:8>=0x3, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- ✓ Esperar 100µs o más que es equivalente a 5,000 ciclos de reloj a 50MHz.
- ✓ Escribir SF_D<11:8>=0x3, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- ✓ Esperar 40µs o más que es equivalente a 2,000 ciclos de reloj a 50MHz.
- ✓ Escribir SF_D<11:8>=0x2, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- ✓ Esperar 40µs o más que es equivalente a 2,000 ciclos de reloj a 50MHz.

Configuración del LCD

Después de que la inicialización se completa, la interfaz de cuatro bits se establece. La siguiente parte de la secuencia es configurar el LCD:

- ✶ Configurar el display para operar sobre la tarjeta Spartan-3E con el comando Function_Set, 0x28.
- ✶ Preparar el display para incrementar automáticamente el indicador de la dirección con el comando Entry_Mode_Set, 0x06.
- ✶ Encender el display y deshabilitar el cursor y el parpadeo con el comando Display_On/Off, 0x0C.
- ✶ Finalmente, permitir al menos 1.64ms (82,000 ciclos de reloj) después de emitir el comando Clear_Display.

Escribiendo datos al LCD

Para escribir datos en el display se especifica la dirección de inicio seguida de uno o más valores de datos. Antes de escribir cualquier dato, se introduce el comando Set_DD_RAM_Address para especificar la dirección inicial de 7 bits en la DD_RAM.

Escribir datos en el display empleando el comando Write_Data_to_CG_RAM o DD_RAM. El valor de datos de 8 bits representa la tabla de búsqueda de la dirección dentro de la CG_ROM o la CG_RAM. El bitmap almacenado en la CG_ROM o la CG_RAM maneja la matriz de 5x8 puntos para representar el carácter asociado.

Si el contador de la dirección está configurado para auto-incrementarse, la aplicación puede escribir secuencialmente múltiples códigos de caracteres y cada carácter es almacenado automáticamente y mostrado en la siguiente localización disponible.

Sin embargo, continuando con la escritura de caracteres, eventualmente decae de la última a la primera línea del display. Los caracteres adicionales no aparecen automáticamente en la segunda línea porque el mapa DD_RAM no es consecutivo de la primera a la segunda línea.

Nota: Si la aplicación no utiliza al LCD de caracteres, el pin LCD_E se pone en *bajo* para deshabilitarlo. También el pin LCD_RW se pone en *bajo* para prevenir a la pantalla LCD de datos presentes.

Ejemplo 1. Despliegue de caracteres en el LCD.

Desarrollo: Con base en la descripción presentada anteriormente, realizar el algoritmo que permita desplegar una cadena de caracteres a través de la pantalla LCD.

Descripción en VHDL:

Se define una ROM en la cual va implícito el bus de datos, las líneas de control y el tiempo entre cada nibble. Entonces se hace un barrido completo del arreglo y posteriormente se retorna solo a la localidad en la que se escribe datos de forma que el algoritmo entre en un bucle infinito.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LCD is

    Port(
        CLK:        in      STD_LOGIC;
        RST:        in      STD_LOGIC;
        DATA:       inout STD_LOGIC_VECTOR( 0 to 3 ):=X"0";
        LCD_E:  inout   STD_LOGIC:='0';
        LCD_RS: inout   STD_LOGIC:='0';
        LCD_RW: inout STD_LOGIC:='0';
        SF_CE0: out STD_LOGIC
        );

end LCD;

architecture Behavioral of LCD is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 )     := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM!
    -- D Y C0 C1 C2 C3 C4 C5
    type ROM_TYPE is array( 0 to 32 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (  -- Power On Inicializacion
                          X"380B71B0", -- Esperar 15ms y escribir 0x3
                          X"3000000C", -- 12 clks despues limpiar LCD_E = '0'
                          X"380320C8", -- 4.1ms
                          X"3000000C", -- 12 clks
                          X"38001388", -- 100us
                          X"3000000C", -- 12 clks
                          X"280007D0", -- 40us
                          X"2000000C", -- 12 clks
                          -- P.O.I. termina
```

```
                        -- Inicia configuracion de Display
                        Control  de nibble Y es LDC_E, LCD_RS, LCD_RW, 0 (X)
                        -- Function Set 0x28
                        X"280007D0", -- 40us entonces  D = 0x2, Y = 1000
                        X"2000000C", -- 230ns entonces D = 0x2, Y = 0000
                        X"88000032", -- 1us entonces   D = 0x8, Y = 1000
                        X"8000000C", -- 230ns entonces D = 0x8, Y = 0000
                        -- Entry Mode 0x06
                        X"080007D0", -- 40us
                        X"0000000C", -- 230ns
                        X"68000032", -- 1us
                        X"6000000C", -- 230ns
                        -- Display On 0x0C
                        X"080007D0", -- 40us
                        X"0000000C", -- 230ns
                        X"C8000032", -- 1us
                        X"C000000C", -- 230ns
                        -- Finalmente Clear Display 0x01
                        X"080007D0", -- 40us
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C", -- 230ns
                        X"10014050", -- 1.64ms
                        -- Termina configuracion de Display
                        -- Seleccionar DDRAM
                        X"880007D0", -- 40us
                        X"8000000C", -- 230ns
                        X"58000032", -- 1us
                        X"5000000C", -- 230ns
                        -- Escribiendo datos
                        X"4CFFFFFF", -- 335ms
                        X"4400000C", -- 230ns
                        X"1C000032", -- 1us
                        X"1400000C"  -- 230ns
                        );
begin
    SF_CE0 <= '1';  -- Inhabilitar acceso a StrataFlash
process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=29;
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

Ejercicio extra-clase: Despliegue el mensaje “CINVESTAV-IPN” en el renglón 1 del LCD y que se desplace a la izquierda.

Archivos de restricciones del usuario:

```
# Terminal del reloj de 50 MHz
NET "clk" LOC = "c9" | IOSTANDARD = LVCMOS33;

# Terminales de control del LCD
NET "LCD_E"  LOC = "M18" |IOSTANDARD= LVCMOS33 |DRIVE= 4 |SLEW = SLOW;
NET "LCD_RS" LOC = "L18" |IOSTANDARD= LVCMOS33 |DRIVE= 4 |SLEW = SLOW;
NET "LCD_RW" LOC = "L17" |IOSTANDARD= LVCMOS33 |DRIVE= 4 |SLEW = SLOW;

# Terminales de los datos del LCD
#(son compartidas con la StrataFlash)
NET "SALIDA<0>" LOC= "R15" |IOSTANDARD= LVCMOS33|DRIVE= 4 |SLEW = SLOW;
NET "SALIDA<1>" LOC= "R16" |IOSTANDARD= LVCMOS33|DRIVE= 4 |SLEW = SLOW;
NET "SALIDA<2>" LOC= "P17" |IOSTANDARD= LVCMOS33|DRIVE= 4 |SLEW = SLOW;
NET "SALIDA<3>" LOC= "M15" |IOSTANDARD= LVCMOS33|DRIVE= 4 |SLEW = SLOW;

# Terminal de deshabilitación de la StrataFlash
NET "SF_CE0"   LOC = "D16" |IOSTANDARD= LVCMOS33|DRIVE= 4 |SLEW = SLOW;
```

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

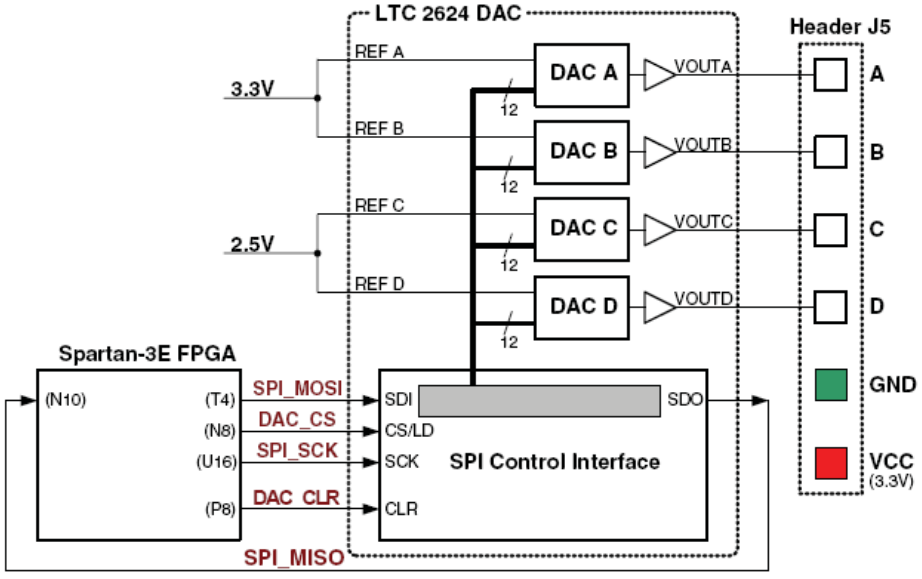
Práctica No. 8
Titulo: DAC (*Convertidor Digital-Analógico*)

Objetivo: El alumno será capaz de diseñar un circuito controlador para el DAC (*LTC2624***)** compatible con SPI (**S**erial **P**eripheral **I**nterface).

Antecedentes: Un convertidor Digital/Analógico (DAC), es un elemento que recibe información de entrada digital, en forma de una palabra de *n* bits y la transforma a señal analógica. Cada una de las combinaciones binarias de entrada es convertida en niveles de tensión de salida.

La tarjeta Spartan 3E cuenta con una interfaz periférica serial (SPI), con 4 canales de conversión digital/analógica (DAC); el dispositivo DAC cuenta con una tecnología de tipo lineal con 12 bits de resolución sin signo.

Comunicación SPI. La SPI permite comunicar los valores digitales a cada uno de los cuatro conectores o canales mencionados, el bus SPI es un canal tipo full-duplex, síncrono, orientado a caracteres, que emplea una interfaz simple de 4 líneas. Un bus maestro conduce la señal de reloj (SPI_SCK) y transmite el dato serial (SPI_MOSI) hacia el bus esclavo seleccionado, al mismo tiempo el bus esclavo proporciona un dato serial (SPI_MISO) de regreso al bus maestro.



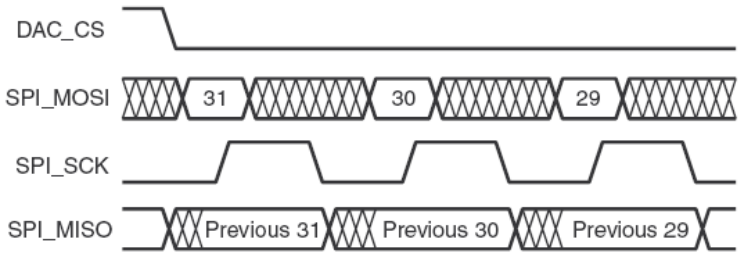
Esquema de conecciones Digital - Analógico

Interfaz de señales. Las señales SPI_MOSI, SPI_MISO y SPI_SCK, son compartidas con otros dispositivos en el bus SPI, la señal DAC_CS (*Activo Bajo*) es la selección de entrada del DAC, el DAC_CLR (*Activo Bajo*) es la entrada de reset asíncrona del DAC.

Puesto que el bus SPI comparte información con otros dispositivos es necesario deshabilitar aquellos que no se requieran para la función de conversión, para lo cual deben establecerse los siguientes valores.

Signal	Disable Value
SPI_SS_B	1
AMP_CS	1
AD_CONV	0
SF_CE0	1
FPGA_INIT_B	1

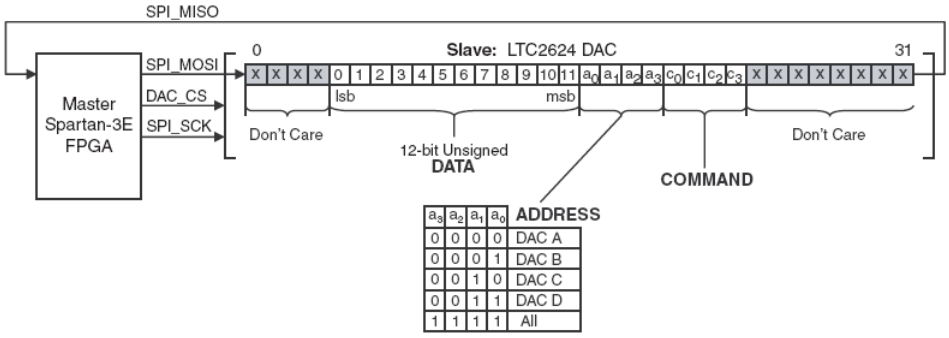
Detalles de la comunicación SPI. A continuación se muestra un esquema que describe los tiempos de comunicación del bus SPI, cada bit es enviado o recibido en relación a la señal de reloj SPI_SCK, el bus es completamente estático y soporta una velocidad de reloj máxima de 50MHz.



Formas de onda de la comunicacion SPI

Después de que la señal DAC_CS pasa al estado bajo, el FPGA transmite los datos en la señal SPI_MOSI, siendo el MSB el primero. El LTC2624 captura el dato de entrada en el flanco de subida del SPI_SCK y el dato debe ser validado por lo menos durante 4ns en relación al flanco de subida, el LTC2624 transmite este dato a través de la señal SPI_MISO en el flanco de bajada del SPI_SCK. Posteriormente el FPGA captura este dato en el próximo flanco de subida del SPI_SCK, el FPGA debe leer el primer valor contenido en el SPI_MISO en el primer flanco de subida de SPI_SCK, después de que la señal DAC_CS sea baja. Una vez transmitidos los 32 bits en su totalidad, el proceso de comunicación del bus SPI termina y el DAC_CS regresa a su estado alto.

Protocolo de comunicación. El protocolo de comunicación requiere una interfaz con el LTC2624 DAC. El DAC soporta protocolos tanto de 24 bits como 32 bits, la interfaz SPI está formada por un registro de 32 bits el cual se describe en la siguiente figura.



a_3	a_2	a_1	a_0	ADDRESS
0	0	0	0	DAC A
0	0	0	1	DAC B
0	0	1	0	DAC C
0	0	1	1	DAC D
1	1	1	1	All

Protocolo de comunicación

Voltajes de salida (DAC). Cada nivel de salida analógica proporcionada por el DAC equivale a un valor digital de 12 bits sin signo (D[11:0]), escrito por el FPGA a la salida correspondiente mediante el bus SPI, el voltaje de salida está descrito por la siguiente ecuación.

$$V_{OUT} = \frac{D[11:0]}{4096} \times V_{REFERENCE}$$

Sin embargo se cuenta con dos voltajes de referencia para las terminales de salida, por lo cual el rango de voltaje de salida estará determinado de la siguiente manera:

Salidas A y B:
$$V_{OUTA} = \frac{D[11:0]}{4096} \times (3.3V \pm 5\%)$$

Salidas C y D:
$$V_{OUTC} = \frac{D[11:0]}{4096} \times (2.5V \pm 5\%)$$

Ejemplo 1. Generar una señal Rampa de 0 a 3.3 Volts, de frecuencia variable.

Desarrollo: **Co**n base en la descripción presentada anteriormente, realizar el código VHDL que entregue una señal rampa utilizando el DAC con el protocolo de 24 bits.

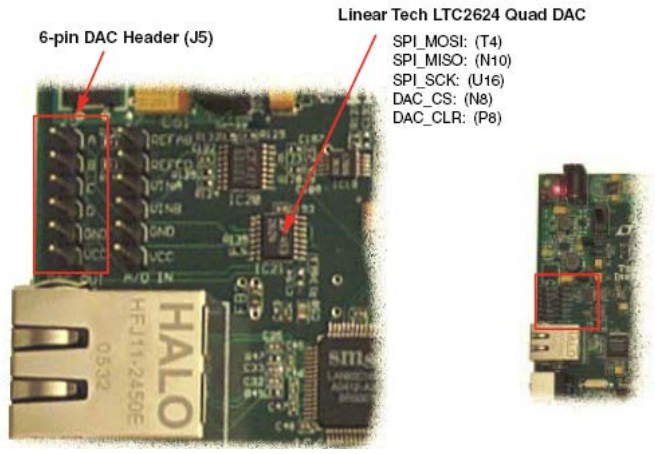
Descripción en VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DAC is
    Port ( CLK,RST : in  STD_LOGIC;
           SPI_MOSI,DAC_CLR : out  STD_LOGIC;
                SPI_SCK,DAC_CS   : inout STD_LOGIC;
           SPI_SS_B,AMP_CS,AD_CONV,SF_CE0,FPGA_INIT_B : out  STD_LOGIC);
end DAC;

architecture Arq of DAC is

signal clk25:   std_logic := '0';
signal clki:   std_logic_vector (1 downto 0)   := "00";
signal cnt:    std_logic_vector (11 downto 0)  := "000000000000";
signal aux:    std_logic_vector (5 downto 0)   := "000000";
signal reg1:   std_logic_vector (23 downto 0)  := "000000000000000000000000";
signal reg2:   std_logic_vector (23 downto 0)  := "000000000000000000000000";

begin
-- Desabilitar otros dispositivos que comparten el ducto SPI
       SPI_SS_B    <= '1';
       AMP_CS          <= '1';
       AD_CONV         <= '0';
       SF_CE0    <= '1';
       FPGA_INIT_B <= '1';

process (clk,rst,clki) begin
       if rst='1'  then
              clki <= (others => '0');
       elsif clk'event and clk='1' then
              clki <= clki + 1;
       end if;
end process;

process (clki(1),rst,clk25) begin
       if rst='1' then
              clk25 <= '0';
   elsif clki(1)'event and clki(1)='1' then
              clk25 <= not clk25;
       end if;
end process;

process (clki(1),rst,aux) begin
       if rst='1' or aux = "110011" then
              aux <= (others => '0');
       elsif clki(1)'event and clki(1)='1' then
              aux <= aux + 1;
       end if;
end process;
```

```
-- Generación de la señal DAC_CLR
DAC_CLR  <= '0' when rst='1' else '1';

-- Generación de la señal DAC_CS
process (clki(1),rst,aux) begin
       if rst='1' or aux = "110011" then
       DAC_CS  <= '1';
       elsif clki(1)'event and clki(1)='1' then
          if aux > "000001" then
                    DAC_CS  <= '0';
             end if;
       end if;
end process;

-- Generación de la señal SPI_SCK
process (clk25,rst,aux,DAC_CS) begin
       if rst='1' then
       SPI_SCK  <= '0';
       elsif DAC_CS = '0' and aux > "000010" then
              SPI_SCK <= clk25;
       else
              SPI_SCK  <= '0';
       end if;
end process;

-- Generación de la señal RAMPA
process (clki(1),aux,rst,cnt) begin
       if rst='1' then
              cnt <= (others => '0');
       elsif clki(1)'event and clki(1)='1' then
              if aux = "000010" then
              cnt <= cnt + 1;
              end if;
       end if;
end process;

-- Protocolo de comunicación
process (DAC_CS,rst,cnt) begin
       if rst='1' then
              reg1 <= "00110000000000000000000";
       elsif DAC_CS'event and DAC_CS='0' then
              reg1 <= "0011" & "0000" & cnt(11 downto 0) & "0000" ;
       end if;
end process;

process (SPI_SCK,rst,reg1,reg2,aux) begin
       if rst='1' or aux = "000010" then
              reg2 <= reg1;
       elsif SPI_SCK'event and SPI_SCK='0' then
              reg2 <= reg2(22 downto 0) & '0';
       end if;
end process;

SPI_MOSI <= reg2(23);

end Arq;
```

Ejercicio extra-clase: Cambiar el código VHDL para generar una señal triangular utilizando el DAC.

Componentes requeridos.



Archivos de restricciones del usuario:

```
# SEÑAL DE RELOJ DE 50 MHZ
NET "CLK"  LOC = "C9" | IOSTANDARD = LVCMOS33 ;

# SEÑAL DE RESET
NET "RST" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;

# SEÑALES DEL DAC
NET "SPI_MISO" LOC= "N10"| IOSTANDARD= LVCMOS33 ;
NET "SPI_SCK"  LOC= "U16"| IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 8;
NET "SPI_MOSI" LOC= "T4" | IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 8;
NET "DAC_CLR"  LOC= "P8" | IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 8;
NET "DAC_CS"   LOC= "N8" | IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 8;

# SEÑALES DE OTROS DISPOSITIVOS
NET "SPI_SS_B" LOC= "U3" | IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 6;
NET "AMP_CS"   LOC= "N7" | IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 6;
NET "AD_CONV"  LOC= "P11"| IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 6;
NET "SF_CE0"   LOC= "D16"| IOSTANDARD= LVCMOS33| SLEW= SLOW | DRIVE= 4;
NET "FPGA_INIT_B" LOC= "T3"|IOSTANDARD= LVCMOS33|SLEW= SLOW | DRIVE= 4;
```

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

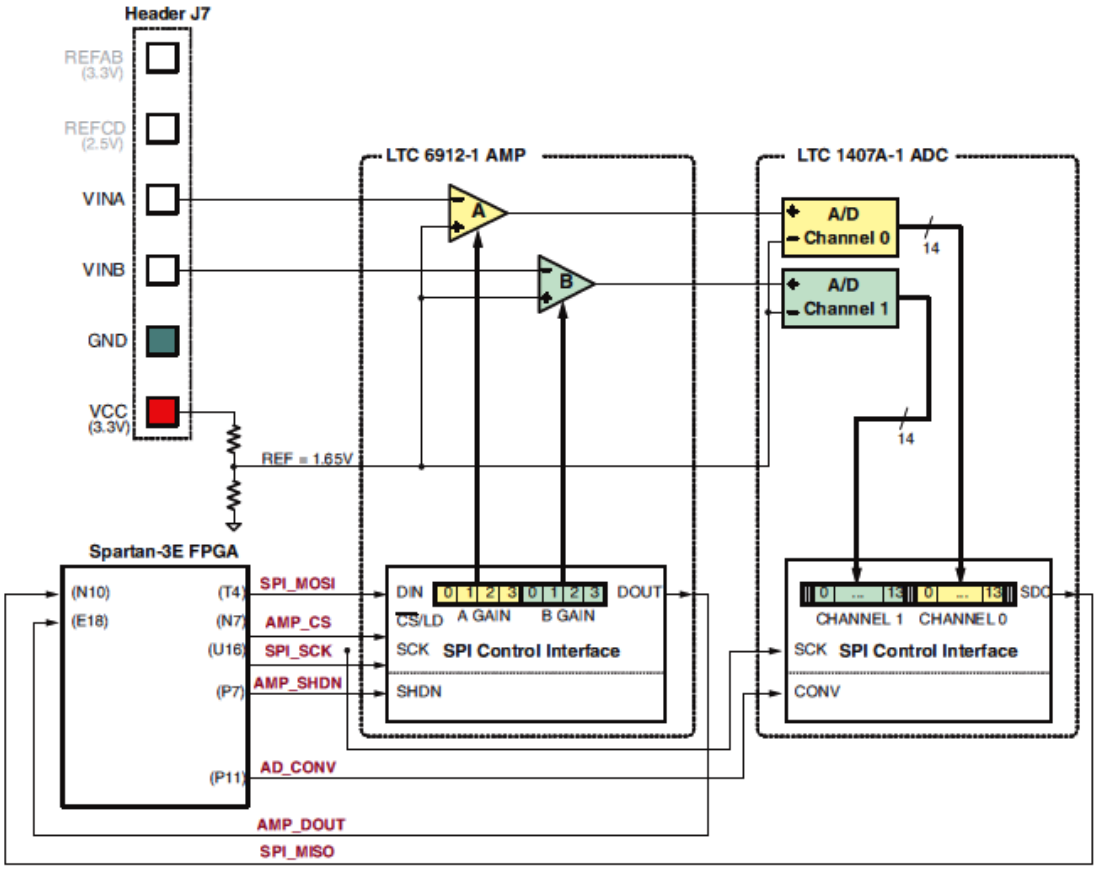
Electrónica Digital (Unidad I)

Práctica No. 9
Titulo: ADC (*Convertidor Analógico - Digital*)

Objetivo: El alumno será capaz de diseñar un circuito controlador para el ADC (*LTC1407A*) así como para el amplificador (*LTC6912A*), conectados a través del bus SPI (**S**erial **P**eripheral **I**nterface).

Antecedentes: Un convertidor Analógico/Digital (ADC), es un elemento que recibe información de entrada analógica, y la transforma a una palabra de *n* bits.

La tarjeta Spartan 3E cuenta con una interfaz periférica serial (SPI), con 2 canales de captura analógica; que consiste de un circuito preamplificador programable y un convertidor ADC.



Vista detallada del circuito de captura analógica

Salidas digitales de entradas analógicas

El circuito de captura analógica convierte los voltajes sobre VINA o VINB en una representación de 14 bits, D[13:0], como expresa la siguiente ecuación.

$$D[13:0] = GAIN \times \frac{(V_{IN} - 1.65V)}{1.25V} \times 8192$$

La ganancia es cargada en el preamplificador programable. Los valores permitidos en Ganancia y en voltaje aplicados a las entradas VINA y VINB aparecen en la siguiente tabla.

La referencia de voltaje para el amplificador y el ADC es 1.65V, generado por un divisor de voltaje.

El máximo rango de el ADC es 1.25, centrado alrededor de la referencia de voltaje, 1.65V. por lo tanto, 1.25 aparece en el denominador de la escala de la entrada analógica.

Finalmente, el ADC presenta 14 bits, con salida complemento a 2. Es decir se pueden representar valores entre -2^{13} y $2^{13}-1$. Por lo tanto, la cantidad es escalada por 8192, o 2^{13}.

El preamplificador programable

El LTC6912 provee 2 amplificadores inversores con ganancia programable. El propósito de el amplificador es escalar los voltajes entrantes sobre VINA o VINB así que esto maximiza el rango de conversión de el ADC, normalmente 1.65 ±1.25.

Interfaz

La siguiente tabla lista la interfaz de señales entre el FPGA y el amplificador. La SPI_MOSI, SPI_MISO, y SPI_SCK son señales compartidas con otros dispositivos sobre el bus SPI. La señal AMP_CS es Activa en bajo.

AMP Interface Signals

Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA→AD	Serial data: Master Output, Slave Input. Presents 8-bit programmable gain settings, as defined in Table 10-2.
AMP_CS	N7	FPGA→AMP	Active-Low chip-select. The amplifier gain is set when signal returns High.
SPI_SCK	U16	FPGA→AMP	Clock
AMP_SHDN	P7	FPGA→AMP	Active-High shutdown, reset
AMP_DOUT	E18	FPGA←AMP	Serial data. Echoes previous amplifier gain settings. Can be ignored in most applications.

Ganancia programable

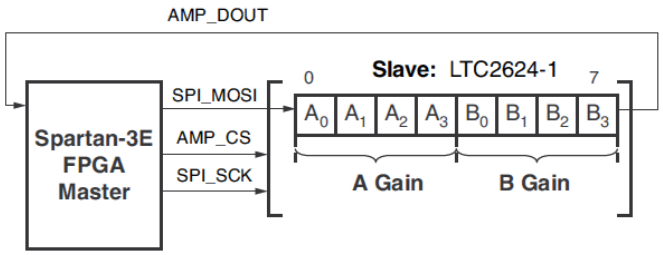
Cada canal analógico tiene una ganancia programable en el amplificador asociada. Señales analógicas presentada sobre las entradas VINA o VINDB en el header J7 son amplificadas relativas a 1.65.

Programmable Gain Settings for Pre-Amplifier

Gain	A3	A2	A1	A0	Input Voltage Range	
	B3	B2	B1	B0	Minimum	Maximum
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	1.7125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

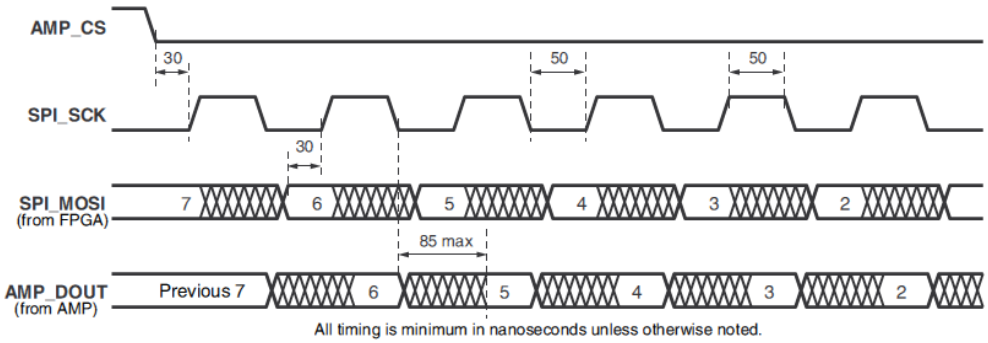
Control de interfaz SPI

La siguiente figura destaca la comunicación con el amplificador, la ganancia por cada amplificador es enviada como una palabra de 8 bits. Formada por 2 nibles de 4 bits. El bit mas significativo es enviado primero.



SPI Serial Interface to Amplifier

La transacción empieza cuando el FPGA afirma AMP_CE bajo. La captura serial del amplificador se coloca sobre SPI_MOSI en cada flanco de subida de la señal SPI_SCK.



SPI Timing When Communicating with Amplifier

ADC

El *LTC1407A* provee 2 ADCs. Ambas entradas analógicas son muestreadas simultáneamente cuando la señal AD_CONV es aplicada.

Interfaz

La siguiente tabla lista las señales entre el FPGA y el ADC. Las señales SPI_MOSI, SPI_MISO and SPI_SCK son compartidas con otros dispositivos sobre el bus SPI.

ADC Interface Signals

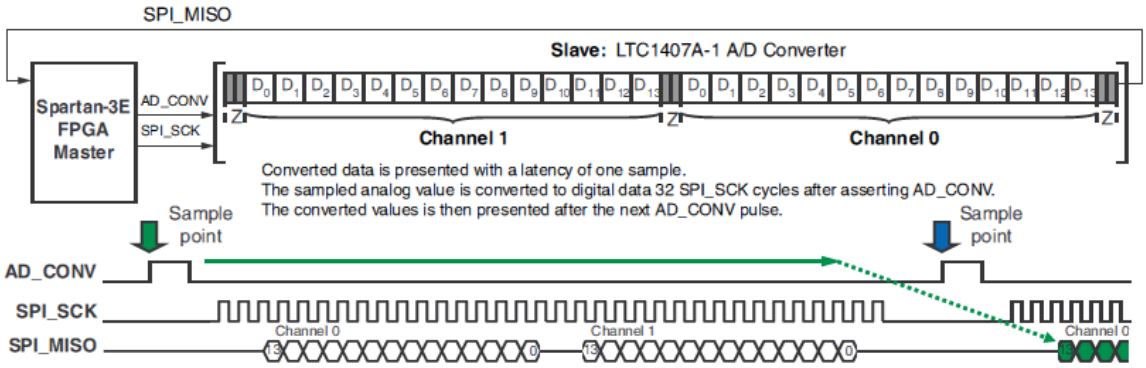
Signal	FPGA Pin	Direction	Description
SPI_SCK	U16	FPGA→ADC	Clock
AD_CONV	P11	FPGA→ADC	Active-High shutdown and reset.
SPI_MISO	N10	FPGA←ADC	Serial data: Master Input, Serial Output. Presents the digital representation of the sample analog values as two 14-bit two's complement binary values.

Control de interfaz SPI

La figura provee un ejemplo de una transacción del bus SPI a la ADC.

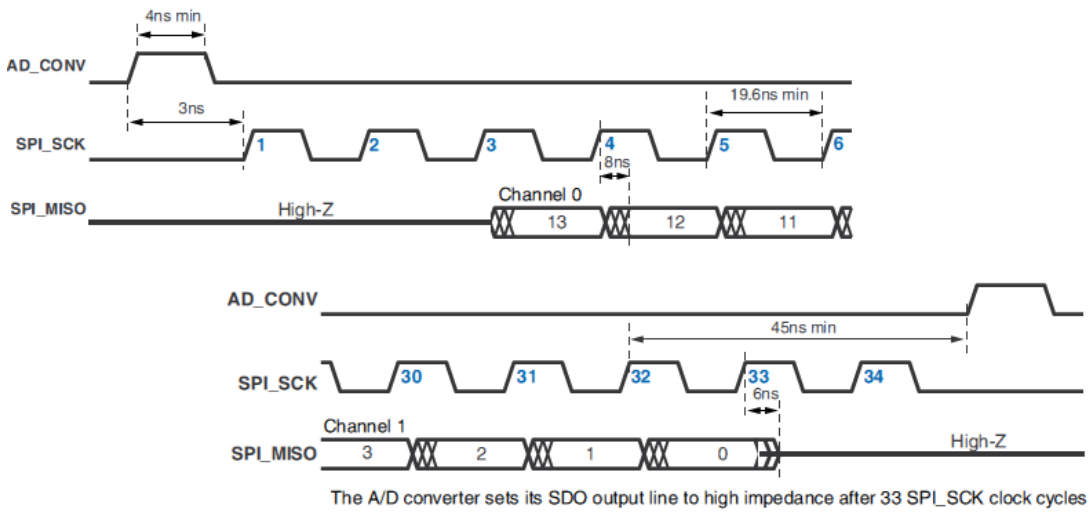
Cuando la señal AD_CONV pasa a alta, el ADC simultáneamente muestras ambos canales analógicos. Los resultados de esta conversión no se presentan hasta la próxima vez que AD_CONV es afirmado.

La frecuencia de máxima de muestreo es de aproximadamente 1,5 MHz.
El ADC presenta la representación digital de los valores de las muestras analógicas como un 14-bit, en valor complemento a dos.



Analog-to-Digital Conversion Interface

La figura muestra a detalle los tiempos de transacción. Como se muestra el ADC usa una secuencia de 34 ciclos. El ADC saca valores de alta impedancia 2 ciclos antes y después de cada 14 bits de datos transferidos.



Detailed SPI Timing to ADC

Deshabilitar otros dispositivos sobre el bus SPI

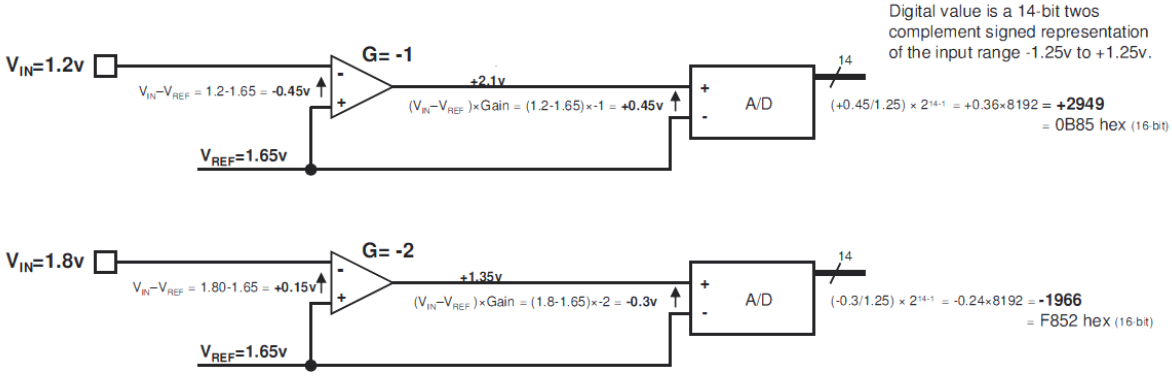
EL bus SPI comparte señales con otros dispositivos sobre la tarjeta. Es vital que sean deshabilitados cuando el FPGA se comunica con el AMP o el ADC.

Disable Other Devices on SPI Bus

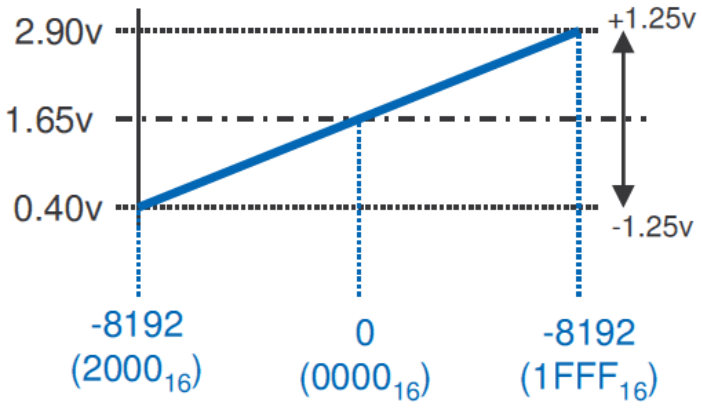
Signal	Disabled Device	Disable Value
SPI_SS_B	SPI Serial Flash	1
AMP_CS	Programmable Pre-Amplifier	1
DAC_CS	DAC	1
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	1

Conversión de Entradas Analógicas

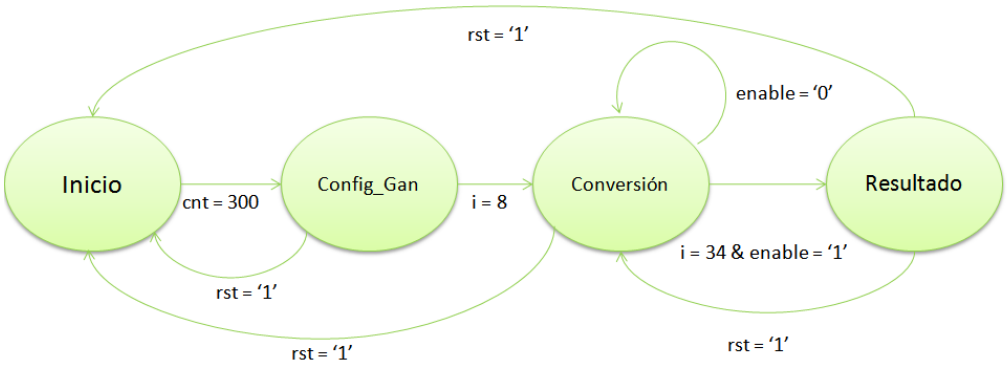
Estos 2 ejemplos indican la respuesta ideal de las entradas analógicas y deriva la fórmula para esta tarjeta. En la práctica habrá ligeras variaciones en los voltajes de referencia debido al ruido lo que hará que los resultados precisos sean imposibles de obtener.



Cada trama de 14 bits mostrada en complemento a 2, esta representa a la entrada analógica.



Desarrollo: **Co**n base en la descripción presentada anteriormente, realizar el código VHDL que entregue la lectura del ADC en 2 tramas de 7 bits multiplexadas con los interruptores SW0 y SW1 para ambos canales, usar maquina de estados.



Descripción en VHDL:
Deshabilitamos señales sobre el bus SPI y configuramos la ganancia.

La función de los estados es la siguiente:

1. Inicializa las señales del preamplificador y el convertidor.
2. Entrega bit a bit sobre SPI_MOSI, el bit MSB es enviado primero.
3. Lee bit a bit de SPI_MISO, y lo almacena en la señal *conv*.
4. Valida la conversión a partir de la segunda lectura, y separa las tramas de 14 bits.

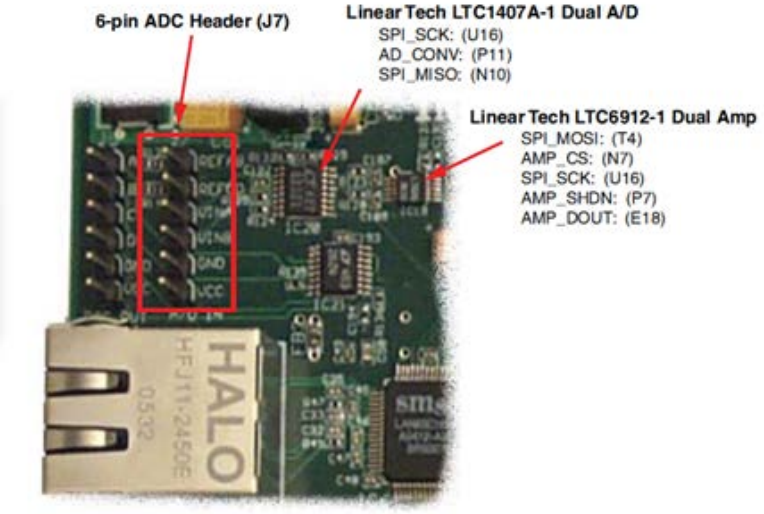
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ADC is
	port(
	clk, rst:	in std_logic;
	SEL:	in std_logic_vector(1 downto 0);
	LEDS:	out std_logic_vector(6 downto 0):="0000000";
	SPI_SCK:	out std_logic;
	SPI_MOSI, AMP_CS, AMP_SHDN:	out std_logic; --Salidas de control del Preamp.
	AD_CONV:	out std_logic; --Entradas y Salidas de control del ADC
	SPI_MISO:	in  std_logic;
	DIS_SIG:	out std_logic_vector(3 downto 0) --Señales sobre el bus SPI
	);
	end ADC;
architecture ADC_Arq of ADC is

	type estados is (Inicio, Config_Gan, Conversion, Resultado);
	signal sig_estado:	estados;
	signal enable: std_logic;
	signal gan: std_logic_vector(0 to 7);
	signal conv: std_logic_vector(33 downto 0);

begin
	DIS_SIG <= X"F";
	AMP_SHDN <= rst;
	gan <= X"11"; --Ganancia B - A
	process(clk, gan, conv, sig_estado, SEL, rst)
		variable cnt,i: integer:= 0;
	begin
		if (clk'event and clk = '1') then
			case sig_estado is
				when Inicio =>
					if cnt < 300 then
						AMP_CS  <= '1';
						SPI_SCK <=	'0';
						SPI_MOSI <= '0';
						AD_CONV <= '0';
						cnt := cnt + 1;
						sig_estado <= Inicio;
					else
						cnt := 0;
						sig_estado <= Config_Gan;
					end if;

				when Config_Gan =>
					AMP_CS <= '0';
					cnt := cnt + 1;
					if i<8 then
						if cnt < 6 then
							SPI_MOSI <= gan(i);
						elsif cnt < 12 then
							SPI_SCK <= '1';
						elsif cnt < 15 then
							SPI_SCK <= '0';
						else
							cnt :=0;
							i := i+1;
						end if;
						sig_estado <= Config_Gan;
					else
						i := 0;
						AMP_CS <= '1';
						SPI_MOSI <= '0';
						cnt := 0;
						sig_estado <= Conversion;
					end if;
```

```
                        when Conversion =>
                                cnt := cnt + 1;
                                if i<34 then
                                        if cnt <= 2  then
                                                if i = 0 then
                                                        AD_CONV <= '1';
                                                end if;
                                        elsif cnt < 4 then
                                                if i = 0 then
                                                        AD_CONV <= '0';
                                                end if;
                                        elsif cnt < 6 then
                                                SPI_SCK <= '1';
                                        elsif cnt < 9 then
                                                conv (i) <= SPI_MISO;
                                        elsif cnt < 10 then
                                                SPI_SCK <= '0';
                                        else
                                                cnt := 0;
                                                i := i+1;
                                        end if;
                                        sig_estado <= Conversion;
                                else
                                        i := 0;
                                        cnt := 0;
                                        sig_estado <= Resultado;
                                end if;

                        when Resultado =>
                                cnt := cnt + 1;
                                if cnt < 50  and enable = '1' then
                                        case SEL is

                                        when "00" => LEDS <= conv(31 downto 25);
                                        when "01" => LEDS <= conv(24 downto 18);
                                        when "10" => LEDS <= conv(15 downto 9);
                                        when "11" => LEDS <= conv(8 downto 2);
                                        when others => LEDS <= (others => '0');

                                        end case;
                                        sig_estado <= Resultado;
                                else
                                        i := 0;
                                        enable <= '1';
                                        cnt := 0;
                                        sig_estado <= Conversion;
                                end if;

                        end case;

                if rst = '1' then
                        cnt := 0;
                        sig_estado <= Inicio;
                        LEDS <= (others => '0');
                end if;

        end if;
    end process;
end ADC_Arq;
```

Ejercicio extra-clase:

Medir el voltaje de la referencia por medio de un canal del ADC y mostrarlo en el LCD en formato hexadecimal (4 dígitos) colocando ceros en los bits mas significativos, con una razón de muestreo de aproximadamente un segundo.

Componentes requeridos.



El modulo se conecta sobre el header J7 con los componentes hacia el exterior de la tarjeta.

Archivos de restricciones del usuario:

```
#Terminal de Reloj 50 Mhz
        NET "CLK"  LOC = "C9" | IOSTANDARD = LVCMOS33 ;
#Terminal de Reset
        NET "RST" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
#Terminales de Seleccion de Vi y parte de la trama;
        NET "SEL<0>"    LOC = "L13" | IOSTANDARD = LVTTL ;
        NET "SEL<1>"    LOC = "L14" | IOSTANDARD = LVTTL ;
#Terminales I/O para la interfaz con el preamplificador
        NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
        NET "AMP_CS"   LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
        NET "AMP_SHDN" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
#Terminales I/O para la interfaz con el ADC
        NET "AD_CONV"  LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
        NET "SPI_SCK"  LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
        NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
# Señales de visualización de la conversión
        NET "LEDS<6>"    LOC = "E12" | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
        NET "LEDS<5>"    LOC = "E11" | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
        NET "LEDS<4>"    LOC = "F11" | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
        NET "LEDS<3>"    LOC = "C11" | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
        NET "LEDS<2>"    LOC = "D11" | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
        NET "LEDS<1>"    LOC = "E9"  | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
        NET "LEDS<0>"    LOC = "F9"  | IOSTANDARD = LVTTL  | DRIVE = 8  | SLEW = SLOW ;
# Terminales para deshabilitar  sobre el bus  SPI
        NET "DIS_SIG<3>" LOC = "U3" | IOSTANDARD = LVCMOS33  | DRIVE = 6  | SLEW = SLOW ;
        NET "DIS_SIG<2>" LOC = "N8" | IOSTANDARD = LVCMOS33  | DRIVE = 8  | SLEW = SLOW ;
        NET "DIS_SIG<1>" LOC = "D16"| IOSTANDARD = LVCMOS33  | DRIVE = 4  | SLEW = SLOW ;
        NET "DIS_SIG<0>" LOC = "T3" | IOSTANDARD = LVCMOS33  | DRIVE = 8  | SLEW = SLOW ;
```