CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

ELECTRÓNICA DIGITAL

Unidad 1

Prácticas de Laboratorio

Agosto, 2014.

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 1.0, Agosto 2014.

ÍNDICE

Este curso incluye 6 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 dos últimas sobre periféricos de la tarjeta de desarrollo.

Herramientas de desarrollo y equipo empleado

Software

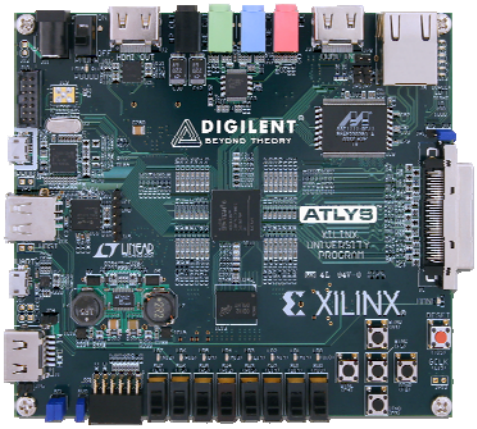
ISE WebPACK 14.2 de Xilinx

ISE Simulator (ISim)

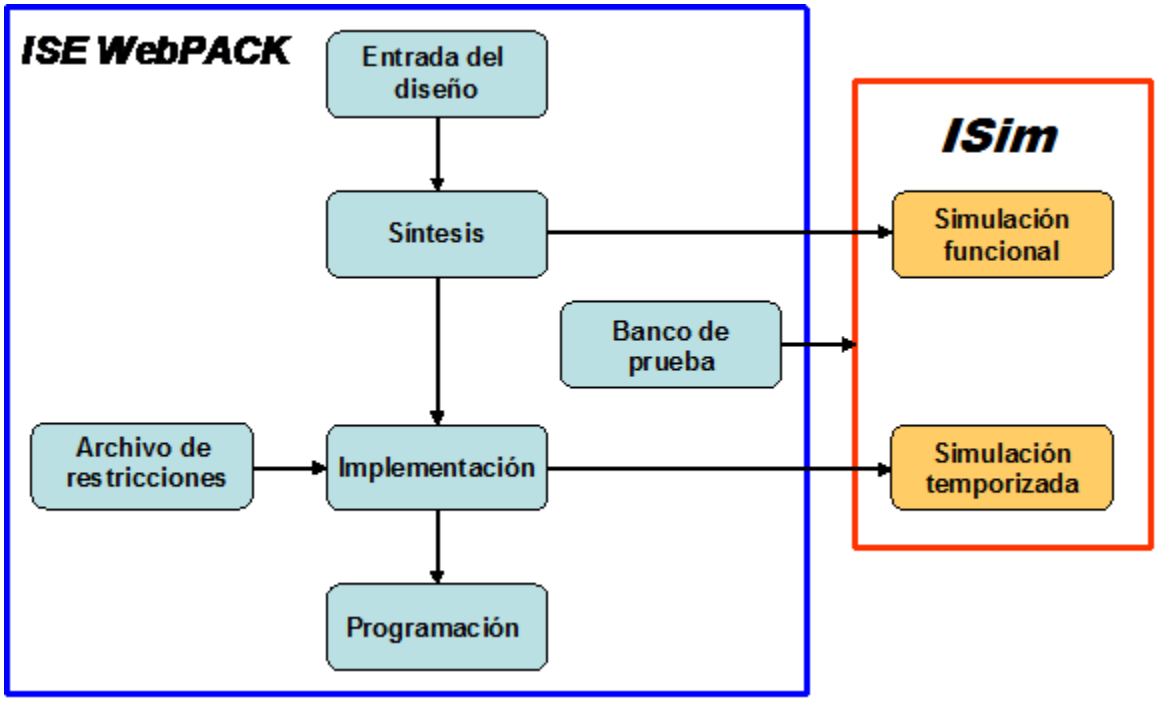
Hardware

Computadora personal con procesador Corei5

Tarjeta de Desarrollo Spartan-6



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.

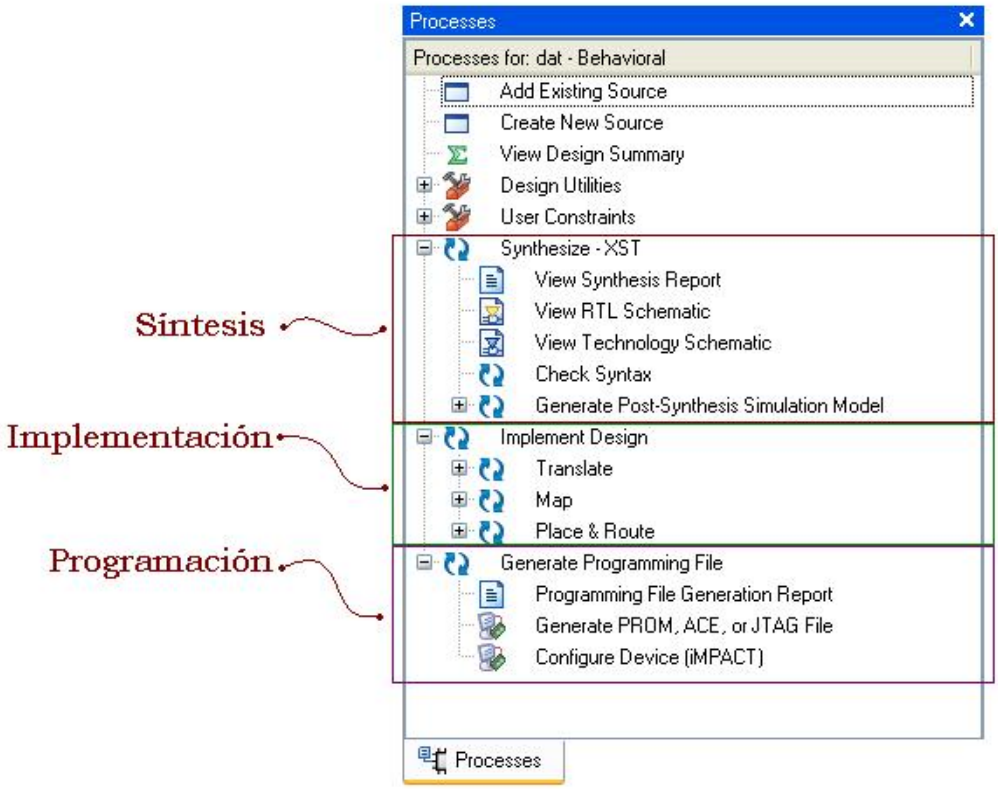
Este proceso, genera un archivo de programación *.bit* que permite descargar la información del diseño, generada en la etapa de implementación, en el dispositivo FPGA, para lo cual debe cumplirse con los procesos anteriores. Comprende los siguientes procesos:

- ***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
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.

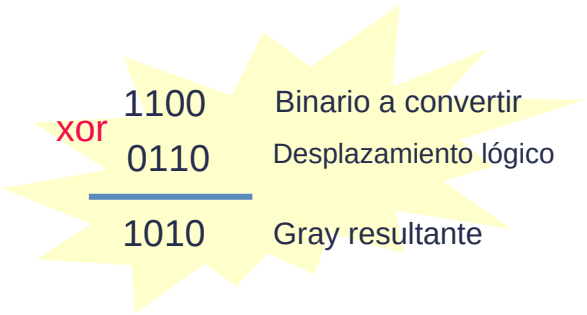
Ejemplo 1. Convertidor de Código Binario a Gray.

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.

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.

B3	B2	B1	B0	A3	A2	A1	A0	Z2	Z1	Z0	
0	0	0	0	0	0	0	0	0	0	1	A=B
0	0	0	0	0	0	0	1	0	1	0	A>B
0	0	0	1	0	0	0	0	1	0	0	A<B
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0	0	0	0	1	1	0	1	0	0	A<B
1	1	1	1	1	1	1	1	0	0	1	A=B

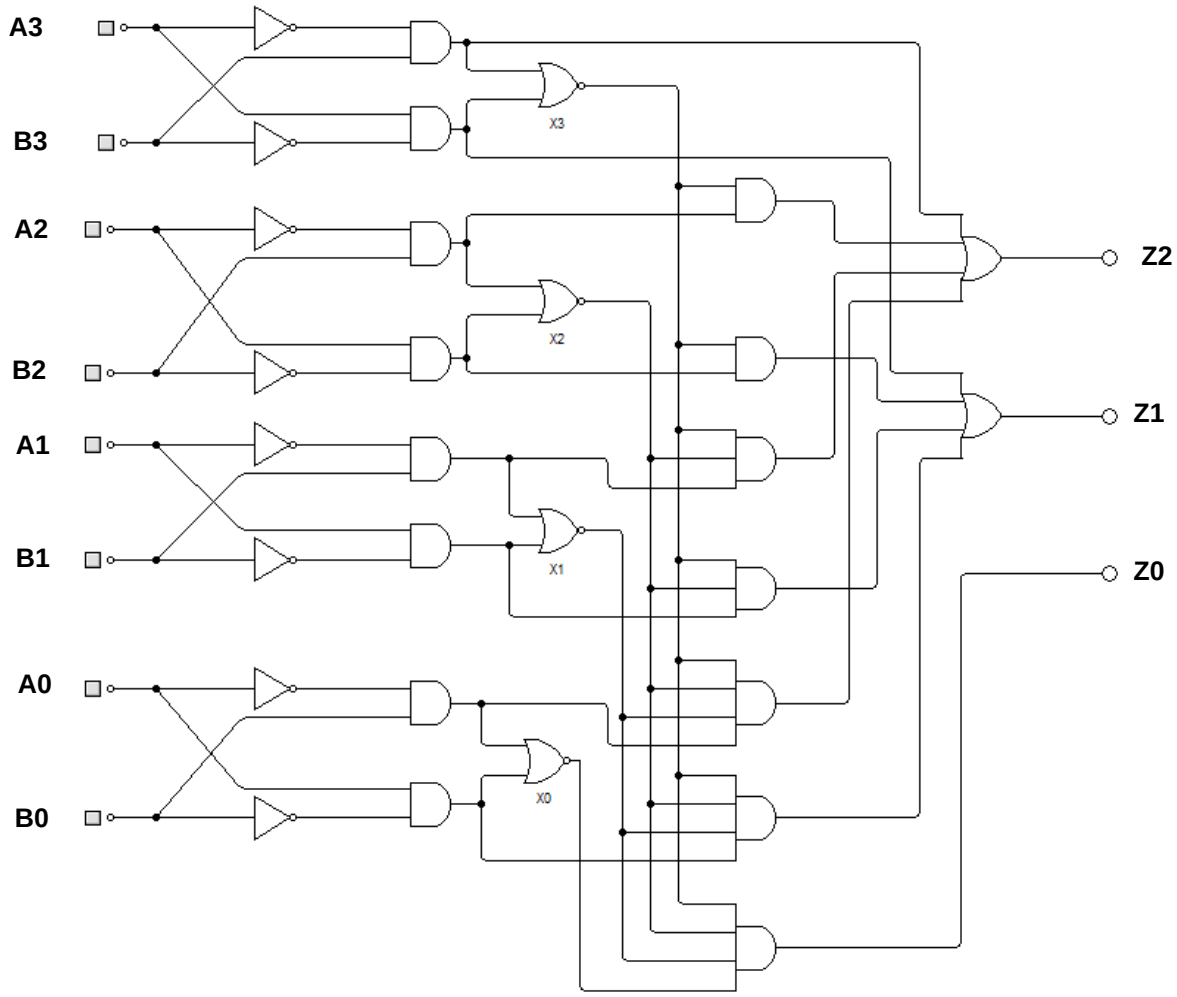
Desarrollo: Mediante la tabla de verdad, realizar el código VHDL correspondiente que cumpla con los requerimientos de la misma, emplear el enunciado "*if-then-elsif-then*" empleando vectores de entrada y de salida para los datos.

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 (3 downto 0);
             Z : out  STD_LOGIC_VECTOR (2 downto 0));
end comparador;

architecture Arq_comparador of comparador is

begin

process (A,B)
    begin
    if (A = B) then
        Z <= "001";
    elsif (A < B) then
        Z <= "100";
    else
        Z <= "010";
    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***.



	Z2	Z1	Z0
A(3:0)<B(3:0)	1	0	0
A(3:0)>B(3:0)	0	1	0
A(3:0)=B(3:0)	0	0	1

(Diseño obtenido de *"Digital Design" - Morris Mano*. Consultar material de apoyo)

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.

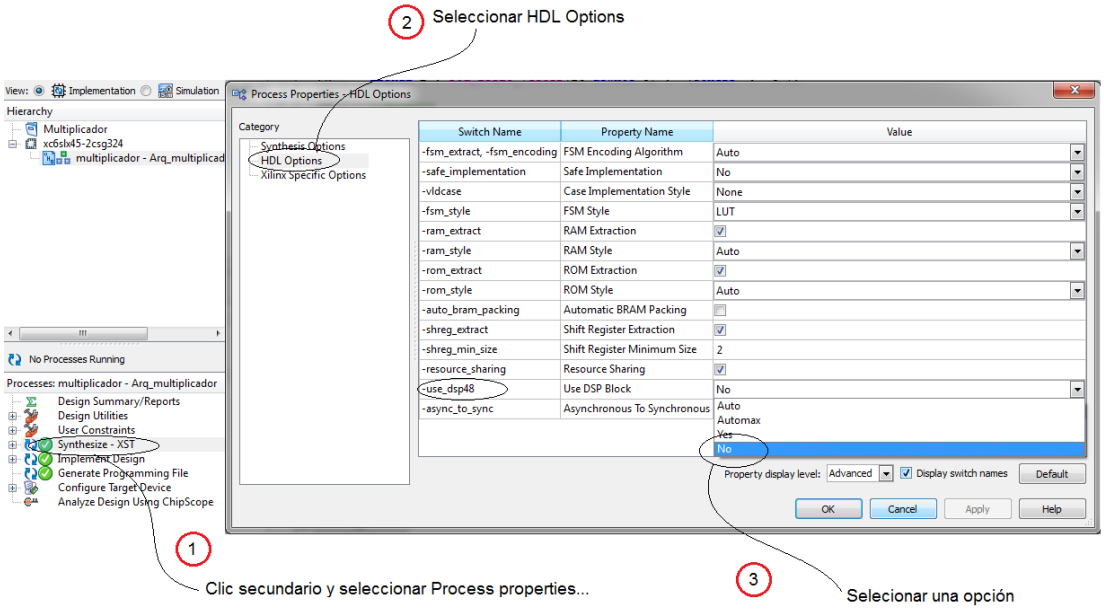
Desarrollo: Realizar un multiplicador de magnitud para dos señales de 4 bits mediante operadores aritméticos.

Descripción en VHDL:

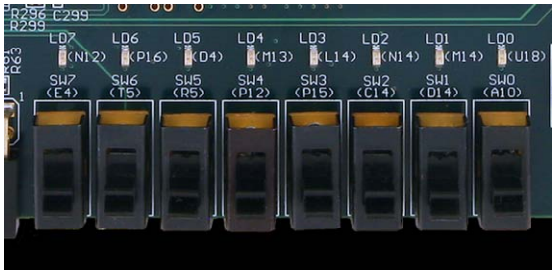
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity multiplicador is
    Port ( A,B : in   STD_LOGIC_VECTOR (3 downto 0);
             P : out  STD_LOGIC_VECTOR (7 downto 0));
end multiplicador;

architecture Arq_multiplicador of multiplicador is

begin

P <= A * B;

end Arq_multiplicador;
```

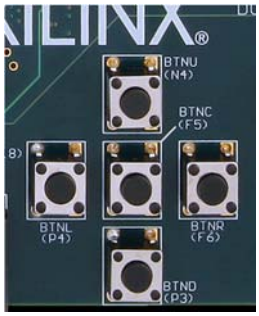
Una vez verificado el código, realizar el flujo de diseño para lógica programable que se presentó al inicio de este documento.

Ejercicio extra-clase: Realizar el ejercicio anterior modificando los vectores de entrada a 17 bits y cambiando las propiedades del proceso Synthesize-XST, en HDL Options - Use DSP Block. Con el objetivo de medir tiempos de retardo de propagación en la simulación temporizada para cada estilo (empleando bloque multiplicador o celdas lógicas).



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






```
# Terminales para el convertidor de código
NET "g<0>" LOC = "U18";
NET "g<1>" LOC = "M14";
NET "g<2>" LOC = "N14";
NET "g<3>" LOC = "L14";
NET "b<0>" LOC = "A10";
NET "b<1>" LOC = "D14";
NET "b<2>" LOC = "C14";
NET "b<3>" LOC = "P15";

# Terminales para el comparador de magnitud
NET "Z<0>" LOC = "U18";
NET "Z<1>" LOC = "M14";
NET "Z<2>" LOC = "N14";
NET "A<0>" LOC = "A10";
NET "A<1>" LOC = "D14";
NET "A<2>" LOC = "C14";
NET "A<3>" LOC = "P15";
NET "B<0>" LOC = "P12";
NET "B<1>" LOC = "R5";
NET "B<2>" LOC = "T5";
NET "B<3>" LOC = "E4";

# Terminales para el multiplicador de magnitud
NET "P<0>" LOC = "U18";
NET "P<1>" LOC = "M14";
NET "P<2>" LOC = "N14";
NET "P<3>" LOC = "L14";
NET "P<4>" LOC = "M13";
NET "P<5>" LOC = "D4";
NET "P<6>" LOC = "P16";
NET "P<7>" LOC = "N12";
NET "A<0>" LOC = "A10";
NET "A<1>" LOC = "D14";
NET "A<2>" LOC = "C14";
NET "A<3>" LOC = "P15";
NET "B<0>" LOC = "P12";
NET "B<1>" LOC = "R5";
NET "B<2>" LOC = "T5";
NET "B<3>" LOC = "E4”;
```

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

Electrónica Digital (Unidad I)

Práctica No. 2
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 (26 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(26),rst,q,ud)
begin
    if rst='1' then
       q <= "0000";
    elsif clkdiv(26)'event and clkdiv(26)='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: Modificar 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:

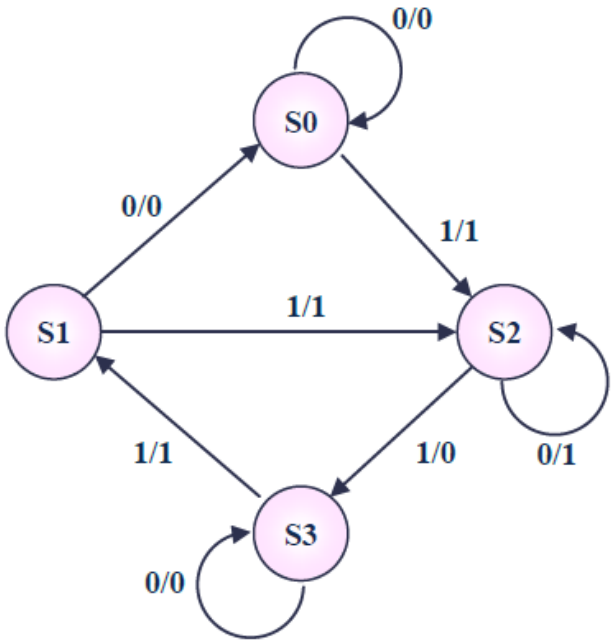
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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)

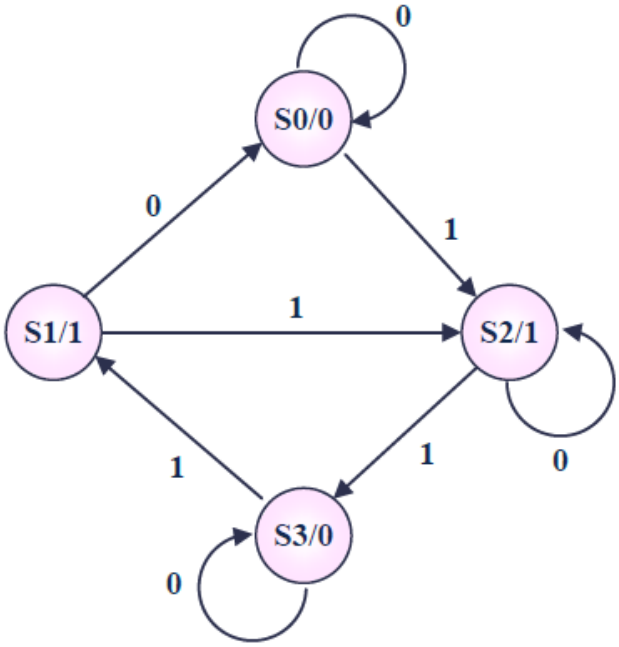
Antecedentes: Se denomina **máquina de estados** tipo Mealy aquella que sus salidas dependen del estado que guarde el sistema y de los valores de las señales de entrada.

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

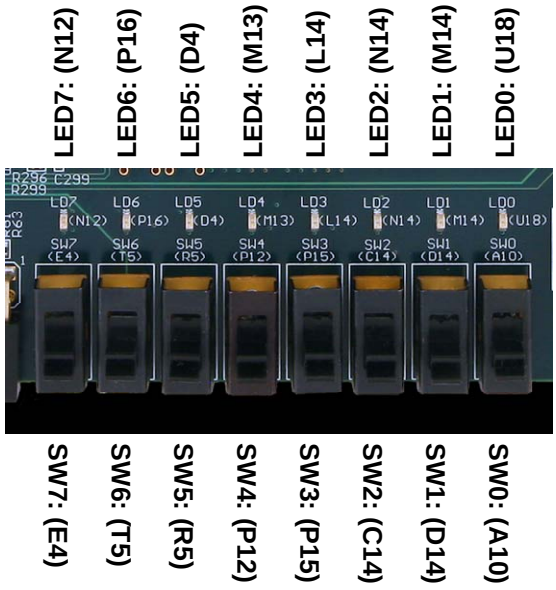


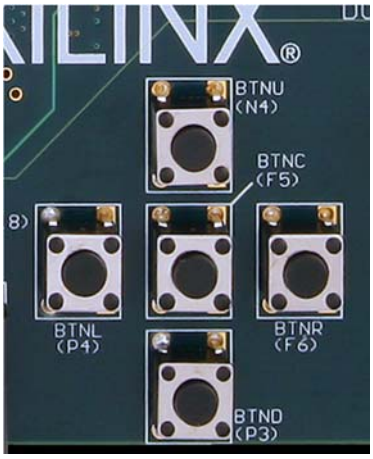
Una vez generado el código VHDL, realizar una simulación funcional para verificar el funcionamiento de la máquina de estados.

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 = "L15";
NET "rst" LOC = "F5";
NET "ud" LOC = "A10";
NET "q<0>" LOC = "U18";
NET "q<1>" LOC = "M14";
NET "q<2>" LOC = "N14";
NET "q<3>" LOC = "L14";

# Terminales para el dado digital
NET "clk" LOC = "L15";
NET "btn" LOC = "P3";
NET "rst" LOC = "F5";
NET "d<0>" LOC = "U18";
NET "d<1>" LOC = "M14";
NET "d<2>" LOC = "N14";
```

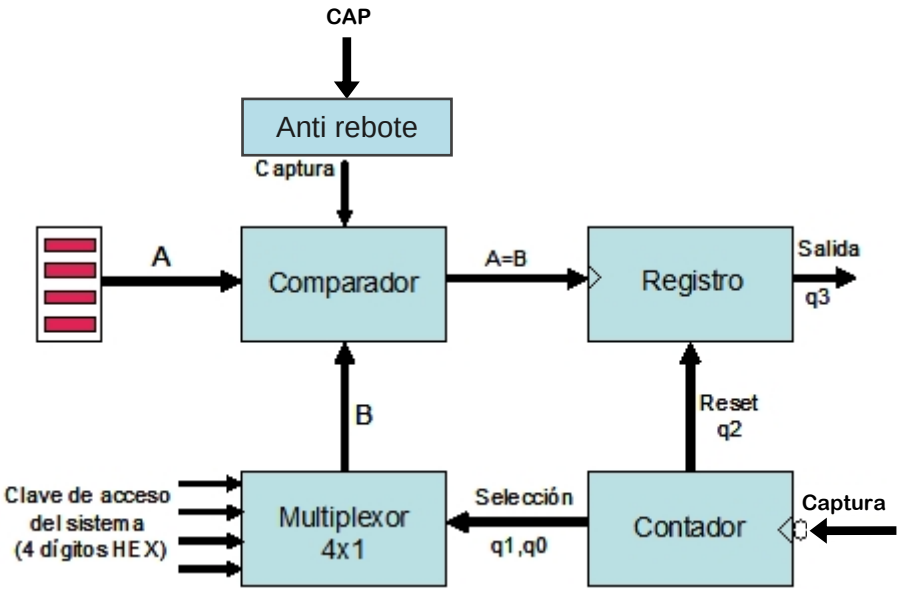
CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

Electrónica Digital (Unidad I)

Práctica No. 3
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.
Anti rebote: Debido a que la operación de los interruptores 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.

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;
```

Anti rebote

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

architecture Behavioral of ANTI_REBOTE 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;
```

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 ANTI_REBOTE 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;
SIGNAL clkdiv: STD_LOGIC_VECTOR (4 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;
```

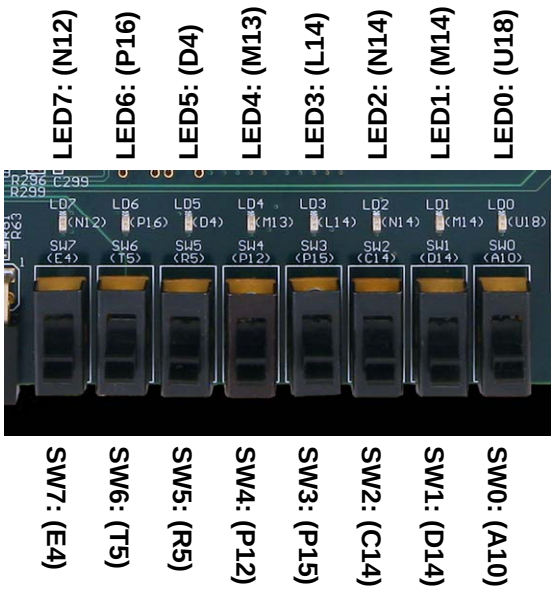
```
MODULO_1 : COMPARADOR   PORT MAP ( DATOA=>ENTRADA, DATOB=>MULTIPLEXOR1, CAPTURA=>ANTIR1,
COMP=>COMPARADOR1 );
```

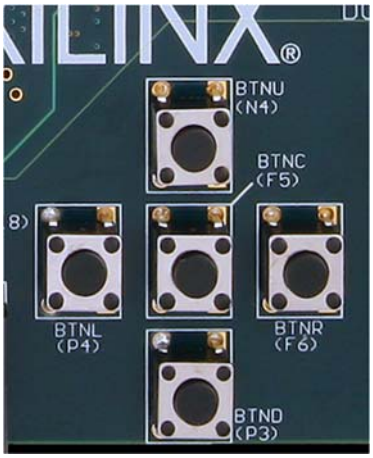
```
MODULO_2 : CONTADOR     PORT MAP ( CAPTURA=>ANTIR1, RESET=>RST, CONTEO=>CONTADOR1 );
MODULO_3 : MULTIPLEXOR  PORT MAP ( DATO0=>X"A", DATO1=>X"F", DATO2=>X"8", DATO3=>X"0", CONT=>CONTADOR1(1 DOWNTO
0), MUX=>MULTIPLEXOR1);
MODULO_4 : REGISTRO     PORT MAP ( PULSO=>COMPARADOR1, RESET=>RST, CON=>CONTADOR1, REGOUT=>REGISTRO1 );
MODULO_5 : ANTIRREBOTE  PORT MAP ( CLK=>clkdiv(4), 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 = "L15";
NET "RST" LOC = "F5";
NET "CAP" LOC = "P3";
NET "LEDS<0>" LOC = "U18";
NET "LEDS<1>" LOC = "M14";
NET "LEDS<2>" LOC = "N14";
NET "LEDS<3>" LOC = "L14";
NET "ENTRADA<0>" LOC = "A10";
NET "ENTRADA<1>" LOC = "D14";
NET "ENTRADA<2>" LOC = "C14";
NET "ENTRADA<3>" LOC = "P15";
```





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

Electrónica Digital (Unidad I)

Práctica No. 4
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.

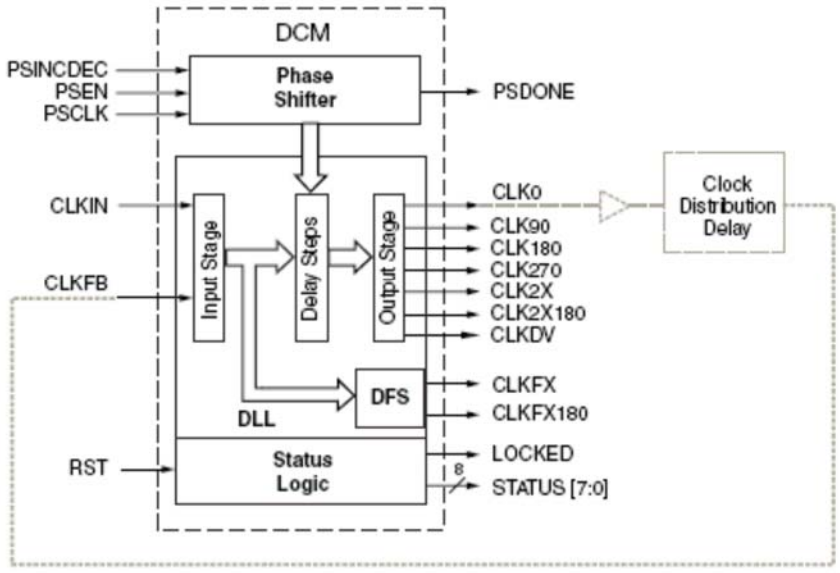
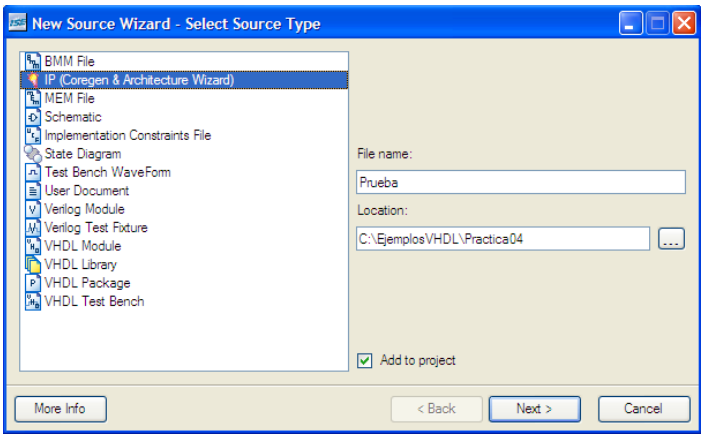
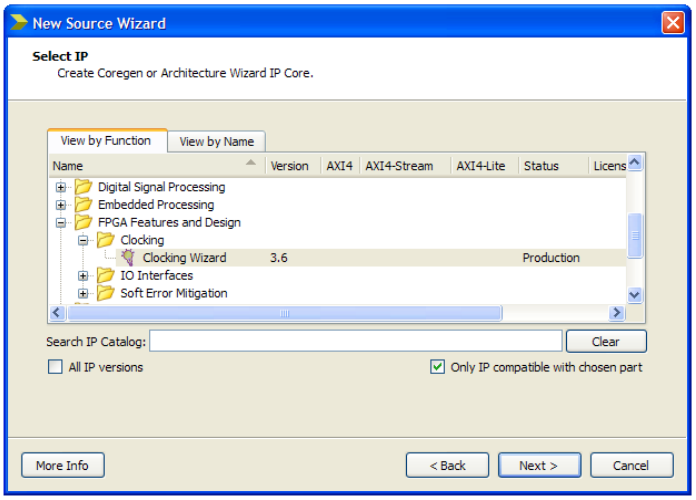
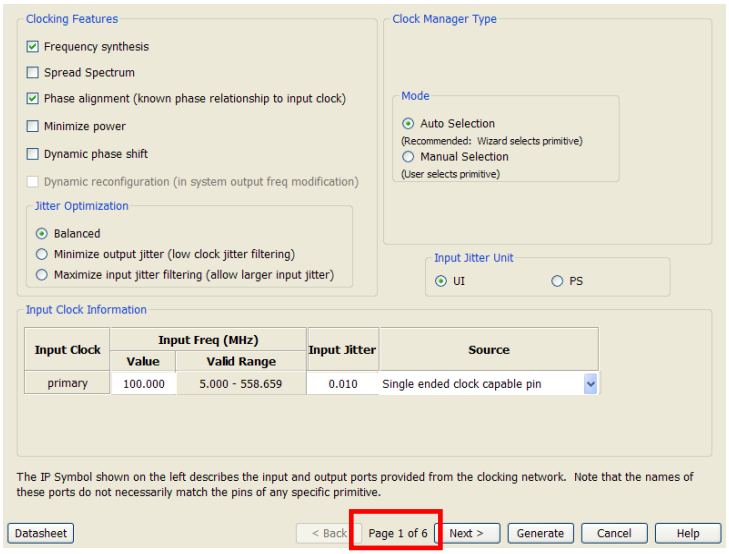


Figura 1. Bloques funcionales y señales asociadas del DCM.

Desarrollo: Mediante la descripción vista anteriormente, seleccione una nueva fuente (Project > New Source) del tipo IP (Coregen & Architecture Wizard), elija Clocking Wizard y configure sus parámetros.







Output Clock Settings

The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives	Use Fine Ps
	Requested	Actual	Requested	Actual	Requested	Actual		
CLK_OUT1	100.000	100.000	0.000	0.000	50.000	50.0	BUFG	
☑ CLK_OUT2	10.000	10.000	0.000	0.000	50.000	50.0	BUFG	
☑ CLK_OUT3	10.000	10.000	90.000	90.000	50.000	50.0	BUFG	
☐ CLK_OUT4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	
☐ CLK_OUT5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	
☐ CLK_OUT6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	

Datasheet | < Back | Page 2 of 6 | Next > | Generate | Cancel | Help

I/O and Feedback

Optional Inputs / Outputs

- ☑ RESET
- ☑ LOCKED
- ☐ INPUT_CLK_STOPPED
- ☐ STATUS
- ☐ CLK_VALID

Clock Feedback Source

- ◉ Automatic control on-chip
- ○ Automatic control off-chip
- ○ User-controlled on-chip
- ○ User-controlled off-chip

Clock Feedback Signaling

- ◉ Single-ended
- ○ Differential

Datasheet | < Back | Page 3 of 6 | Next > | Generate | Cancel | Help

DCM_SP Settings

These are the settings based on your input. If you change them, you may lose the optimal settings found by the wizard, and selections made on previous pages will not apply.

☐ Allow override mode (manual edits to the DCM_SP attributes)

Attribute	Value
CLKDV_DIVIDE	10.0
CLKFX_DIVIDE	20
CLKFX_MULTIPLY	2
CLKIN_DIVIDE_BY_2	☐
CLKIN_PERIOD	10.000
CLKOUT_PHASE_SHIFT	NONE
CLK_FEEDBACK	1X
DESKEW_ADJUST	SYSTEM SYNCHRONOUS
PHASE_SHIFT	0
STARTUP_WAIT	☐

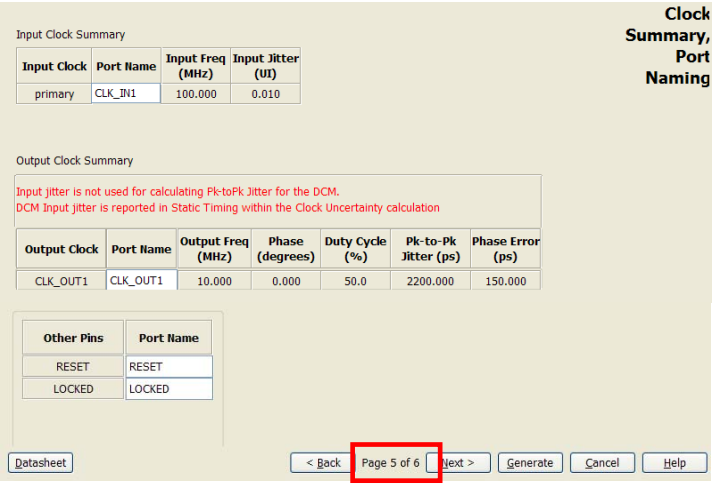
Output Clock Mapping

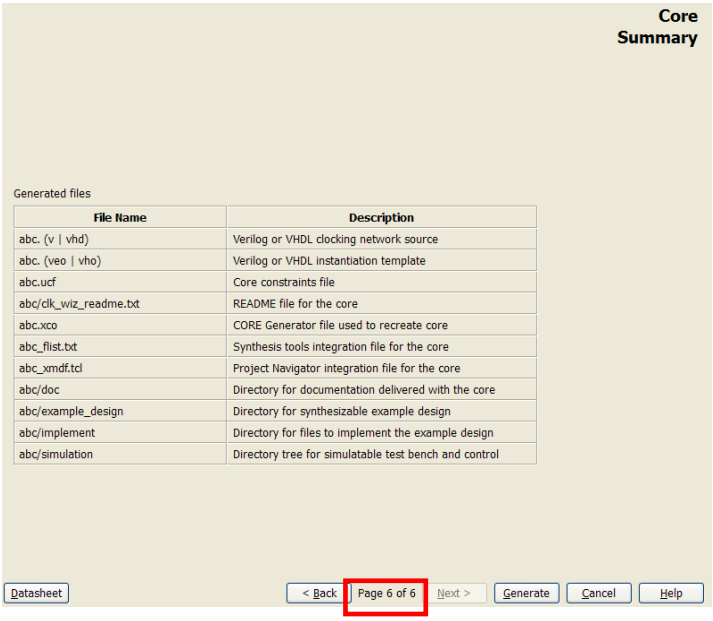
CLK_OUT1	CLKFX

Notes to be included in the comments of the generated component:

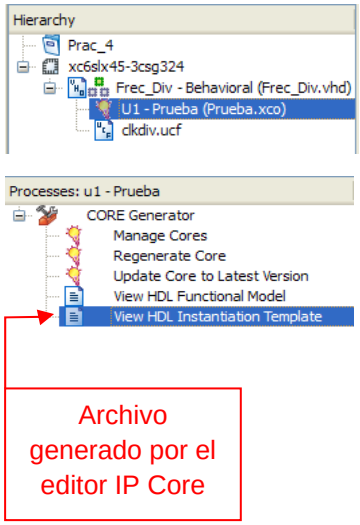
None

Datasheet | < Back | Page 4 of 6 | Next > | Generate | Cancel | Help





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 CLKDIV is
PORT (RESET,CLK_IN1 : IN STD_LOGIC;
      CLK_0UT1, CLK_0UT2,CLK_0UT3,LOCKED: OUT STD_LOGIC);
end CLKDIV;

Architecture Behavioral of CLKDIV is

Component Prueba port
(CLK_IN1: in std_logic;
 CLK_0UT1: out std_logic;
 CLK_0UT2: out std_logic;
 CLK_0UT3: out std_logic;
 RESET  : in   std_logic;
 LOCKED: out   std_logic);
End component;

Begin

U1:Prueba port map
(CLK_IN1 => CLK_IN1, CLK_0UT1 => CLK_0UT1, CLK_OUT2 => CLK_OUT2,
CLK_OUT3 => CLK_OUT3,, RESET => RESET, LOCKED => LOCKED);

End Behavioral;
```

Actividades complementarias:

Primeramente, cambie el nivel de corriente de la salida correspondiente a la señal CLK_OUT2 en el archivo de restricciones de usuario y observe las variaciones en el osciloscopio.

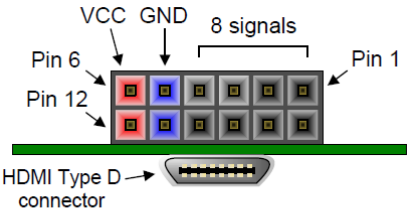
Spartan-6 FPGA Programmable Output Drive Current Supported by I/O Standard

I/O STANDARD	Output Drive Current (mA)						
	2	4	6	8	12	16	24
LVTTL	All	All	All	All	All	All	All
LVCMOS33	All	All	All	All	All	All	All
LVCMOS25	All	All	All	All	All	All	Banks 1, 3, 4, 5
LVCMOS18, LVCMOS18_JEDEC	All	All	All	All	All	All	Banks 1, 3, 4, 5
LVCMOS15, LVCMOS15_JEDEC	All	All	All	All	Banks 1, 3, 4, 5	Banks 1, 3, 4, 5	N/A
LVCMOS12, LVCMOS12_JEDEC	All	All	All	Banks 1, 3, 4, 5	Banks 1, 3, 4, 5	N/A	N/A

Ejercicio extra-clase: Tomar la señal CLK_OUT2, 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:

```
#clock pin for Atlys rev C board
NET "CLK_IN1" LOC = "L15";
#RESET
NET "RESET" LOC = "F5";
#PMOD Connector
NET "LOCKED" LOC = "R3";
NET "CLK_OUT1" LOC = "T3" | IOSTANDARD = LVCMOS25 | SLEW = SLOW | DRIVE = 2;
NET "CLK_OUT2" LOC = "R3" | IOSTANDARD = LVCMOS25 | SLEW = SLOW | DRIVE = 2;
NET "CLK_OUT3" LOC = "P6" | IOSTANDARD = LVCMOS25 | SLEW = SLOW | DRIVE = 2;

PIN "u1/clkout1_buf.O" CLOCK DEDICATED ROUTE = FALSE;
PIN "u1/clkout2_buf.O" CLOCK DEDICATED ROUTE = FALSE;
PIN "u1/clkout3_buf.O" CLOCK DEDICATED ROUTE = FALSE;
```



Pmod Connectors – front view as loaded on PCB

Pmod Pinout

Pin 1: T3
Pin 2: R3
Pin 3: P6
Pin 4: N5
Pin 5: GND
Pin 6: VCC
Pin 7: V9
Pin 8: T9
Pin 9: V4
Pin 10: T4
Pin 11: GND
Pin 12: VCC

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

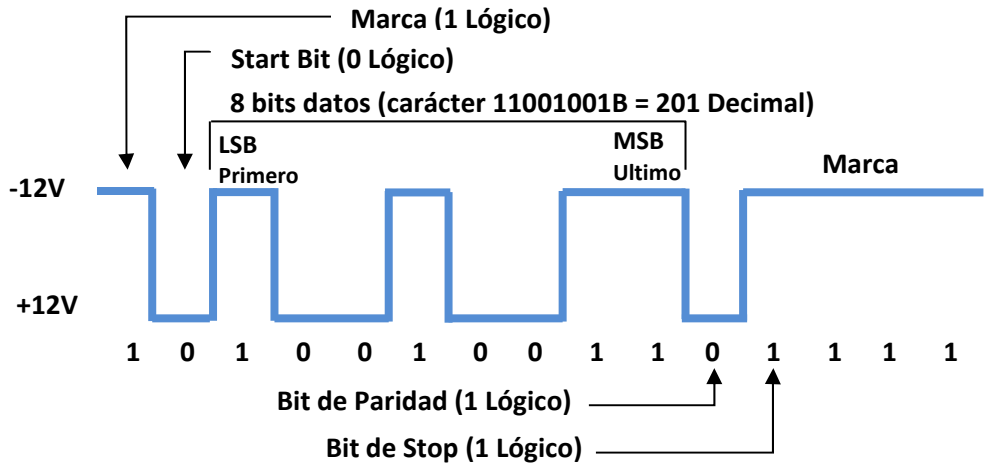
Electrónica Digital (Unidad I)

Práctica No. 5
Puerto serial 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 “Tera Term” 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 puerto RS-232 (UART) de la tarjeta SPARTAN-6, 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):="0010100010110000"; --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;
```

```
-- Asignación 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;
```

Ejercicio extra-clase: Realizar un programa que despliegue el mensaje **“CINVESTAV-IPN”** y el número de equipo **“EQUIPO ##”** en la terminal.

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, una velocidad de transmisión de **115200** baudios.

Descripción en VHDL:

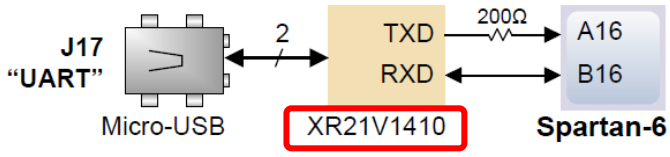
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

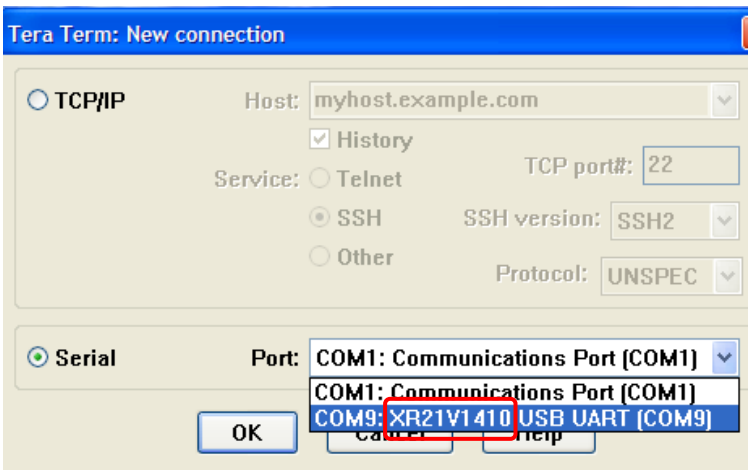
```
entity Recep is
    Port ( clk,rst,rx: in  std_logic;
           leds: out std_logic_vector(7 downto 0));
end Recep;

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(8 downto 0):= "100100001";;--(115200*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: Realizar un contador módulo *n* cuyo valor de módulo sea recibido a través de la comunicación serial asíncrona.

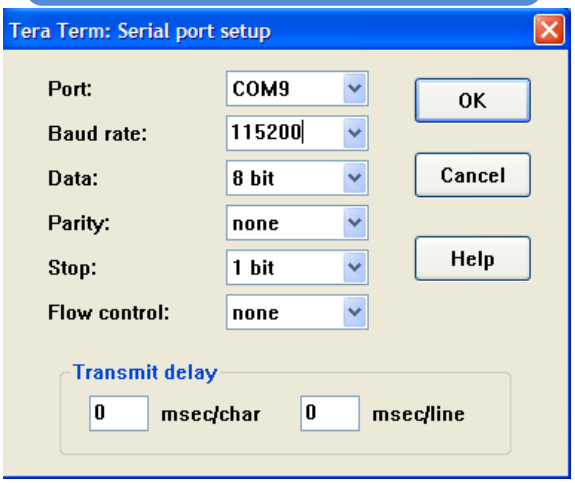
Archivos de restricciones del usuario:

```
#clock pin for Atlys rev C board
NET "clk" LOC = "L15";
NET "rst" LOC = "F5";

#USB UART Connector
NET "rx" LOC = "A16";
NET "tx" LOC = "B16";

#onBoard Leds
NET "leds<0>" LOC = "U18";
NET "leds<1>" LOC = "M14";
NET "leds<2>" LOC = "N14";
NET "leds<3>" LOC = "L14";
NET "leds<4>" LOC = "M13";
NET "leds<5>" LOC = "D4";
NET "leds<6>" LOC = "P16";
NET "leds<7>" LOC = "N12";
```





Configurar Baud Rate (115200)
Setup > Serial Port…



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

Electrónica Digital (Unidad I)

Práctica No. 6
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.

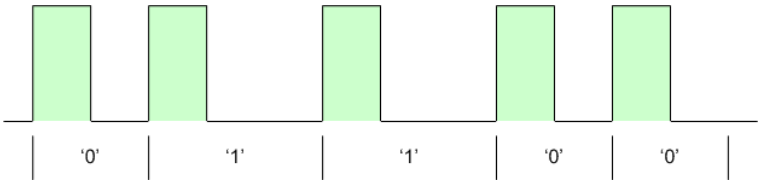
En términos de transmisión, la modulación se refiere a poner al led infrarrojo en corte o en conducción de manera rápida a razón de la frecuencia portadora. El receptor estará sincronizado solo con esta frecuencia para asegurar que se reciba solo la señal requerida, ya que utiliza esta frecuencia para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

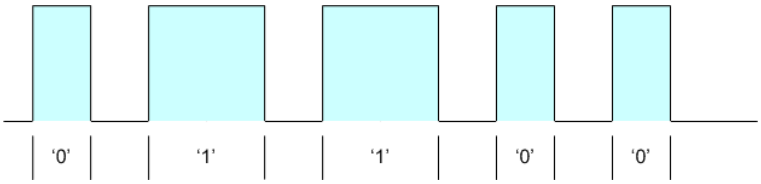
Codificación por separación de pulso

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

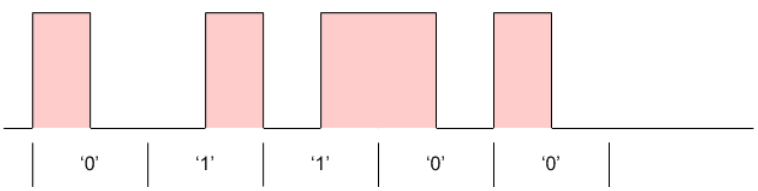
En este método de codificación, la duración en estado alto es diferente para el ‘0’ y el ‘1’ lógicos, siendo mayor el tiempo para el ‘1’ lógico.



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.

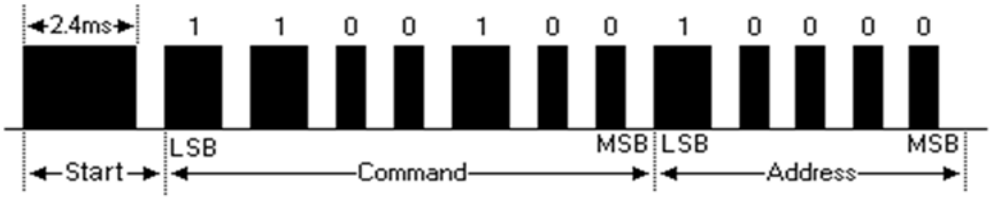


Protocolo SIRC de SONY

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

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.

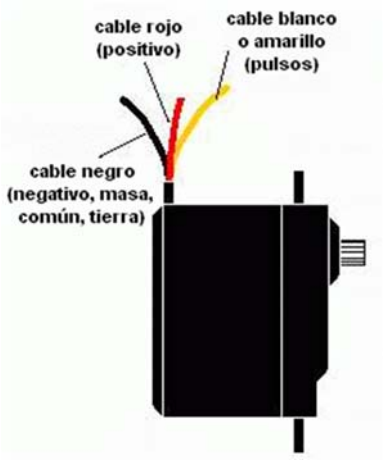
```
library IEEE;
use IEEE.STD LOGIC 1164.ALL;
use IEEE.STD LOGIC ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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(13 downto 0):="00000000000000";--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="10011100010000" 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(13),rst,nref,rstin) begin

if rst='1' or rstin='1'  then
              rstin <='0';
elsif clkdiv(13)'event and clkdiv(13)='0' then
       if nref = "01111" then
              rstin <= '1';
       end if;
end if;

end process;



-- Se cuenta el número de pulsos a partir del flanco de bajada del pulso de sincronía
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(13),rst,nref,sony,rstin) begin

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


elsif clkdiv(13)'event and clkdiv(13)='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(13),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(13)'event and clkdiv(13)='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(13),rst,npulsos,vector) begin
if rst = '1' then
             leds <= (others => '0');
elsif clkdiv(13)'event and clkdiv(13)='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.

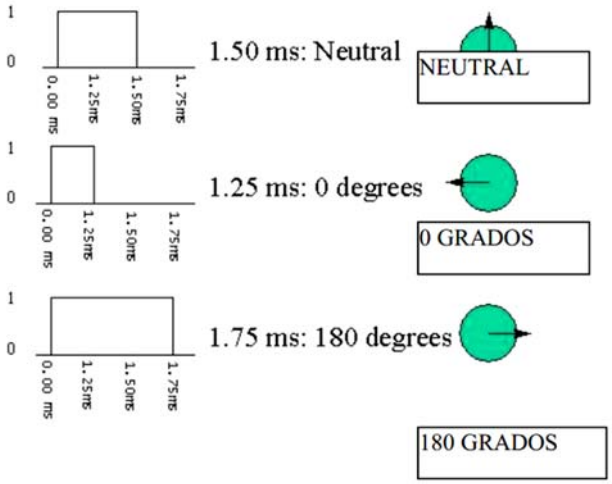


El servomotor tiene algunos circuitos de control y un potenciómetro que está conectada al eje central del servo. Este permite a la etapa de control, supervisar el ángulo actual del servo, si el eje está en el ángulo correcto, entonces el motor está apagado. Si el circuito checa que el ángulo no es el 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.

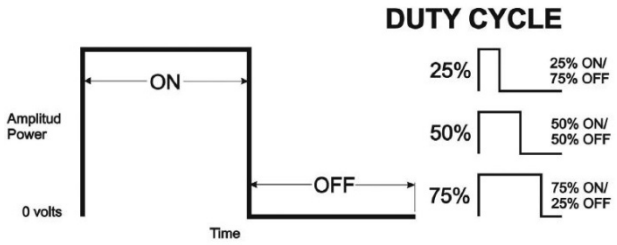
Para comunicarnos con el servomotor debemos hacerlo por la terminal de control, aquí le asignamos un ángulo en el que debe posicionarse dependiendo del ancho del pulso. En esta terminal es decir usamos:

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 anti rebote 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 límites para el servomotor en 0º y 180º.

Se asignan las señales de cada módulo en el componente anti rebote.

```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"3E8" then -- 20ns * 500 ciclos = referencia 10us
       cnt <= (others => '0');
        clk100K <= '1';
    elsif clk'event and clk='1' then
        cnt <= cnt + 1 ;
        if cnt= X"1F4" 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 (15ms)
       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;
```

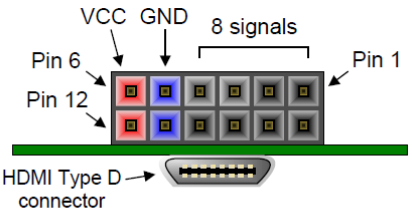
El componente antirrebote es el siguiente, tome en cuenta que la señal de reloj que recibe es la señal de referencia de 10us.

```
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.

Mando Infrarrojo



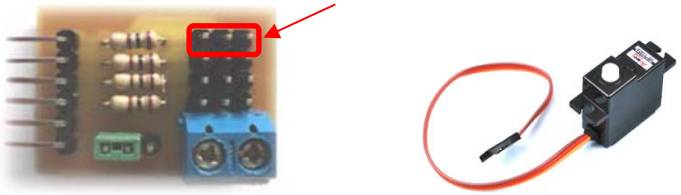
Pmod Connectors – front view as loaded on PCB

Pmod Pinout

Pin 1: T3
Pin 2: R3
Pin 3: P6
Pin 4: N5
Pin 5: GND
Pin 6: VCC
Pin 7: V9
Pin 8: T9
Pin 9: V4
Pin 10: T4
Pin 11: GND
Pin 12: VCC

Conectar el modulo en lo pines (1 a 6) del conector “Pmod”

Tres primeros pines del modulo



Conectar el modulo en lo pines (7 a 12) del conector “Pmod” y el servo motor en los tres primeros pines del modulo.

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

```
#---------------------------------
#---MANDO INFRAROJO
#---------------------------------

#clock pin for Atlys rev C board
NET "clk" LOC = "L15";
NET "rst" LOC = "F5";
# PMOD Connector
NET "ent" LOC = "T3";
NET "ent" CLOCK_DEDICATED_ROUTE =
FALSE;
# onBoard Leds
NET    "leds<0>"    LOC    ="U18";
NET    "leds<1>"    LOC    ="M14";
NET    "leds<2>"    LOC    ="N14";
NET    "leds<3>"    LOC    ="L14";
NET    "leds<4>"    LOC    ="M13";
NET    "leds<5>"    LOC    ="D4";
NET    "leds<6>"    LOC    ="P16";
```

```
#---------------------------------
#---SEÑAL PWM
#---------------------------------

#clock pin for Atlys rev C board
NET "clk" LOC = "L15";
NET "rst" LOC = "F5";

NET " btderecha" LOC = "P4";
NET " btizquierda" LOC = "F6";

# PMOD Connector
NET "ent" LOC = "T3";
```