DISEÑO DE SISTEMAS DIGITALES

Tópico Práctico. No. 3
"ISIS de Proteus para simulación"
"Diseño de máquinas de estado con ABEL-HDL"

Dr. Juan Carlos Herrera Lozada.
jlozada@ipn.mx

Centro de Innovación y Desarrollo Tecnológico en Cómputo
Lab. de Diseño de Sistemas Digitales
Instituto Politécnico Nacional

Abril 2013

Campo 1: **Datos Personales.**

Campo 2: **Objetivos**.

- Diseño de circuitos de lógica secuencial.
- Descripción de contadores con ABEL – HDL, utilizando máquinas de estado.
- Simulación de circuitos secuenciales utilizando un simulador funcional (ISIS de Proteus) e implementación en PLDs de arquitectura simple.

Campo 3: **Desarrollo de la Práctica**.

Nota: Para los diseños, anexar los respectivos códigos y simulaciones según el caso.

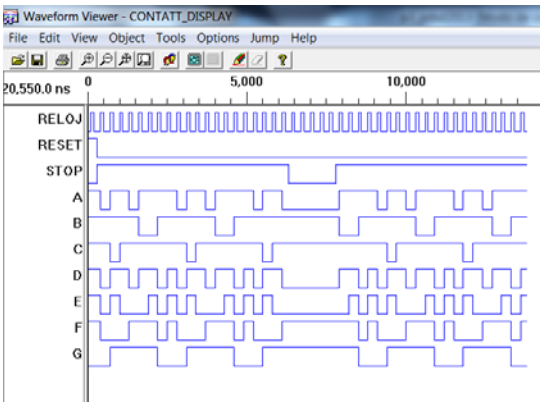
Preparación: Para el diseño secuencial será necesario que utilices una señal de reloj. Para generar este pulso de reloj, refiérete a la práctica 2.

1. (3 puntos) Introducción a la simulación con ISIS de Proteus.

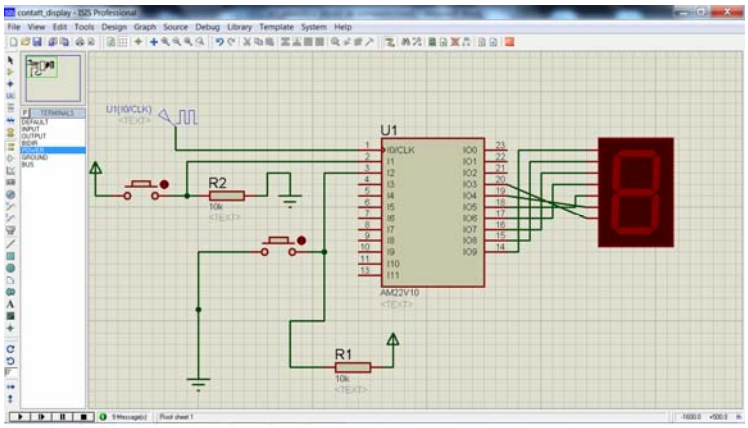
Comenzaremos refiriéndonos al último diseño de la práctica 1, el cual infiere un contador ascendente con tabla de transiciones (tabla de verdad) de 3 bits y un decodificador para un display a siete segmentos, ambos elementos dentro del mismo dispositivo GAL22V10. El contador debe soportar un *reset* y un *stop*. La solución es la siguiente:

```
MODULE contt_display
"entradas
reloj, reset, stop pin 1, 2, 3;
"Salidas
Q2,Q1,Q0 node istype 'reg';
a, b, c, d, e, f, g pin 14,15,16,17,18,19,20 istype 'com';
Y =[Q2,Q1,Q0];
Display=[a, b, c, d, e, f, g];
equations
Y.clk = reloj;
Y.ar = reset;
Y.ce = stop;

truth_table
([Y]:>[Y]->Display)
[0]:>[1]->126;
[1]:>[2]->48;
[2]:>[3]->109;
[3]:>[4]->121;
[4]:>[5]->51;
[5]:>[6]->91;
[6]:>[7]->95;
[7]:>[0]->112;

test_vectors
([reloj, reset, stop]->[Display])
[.C., 1, 0]->[.X.];
@repeat 20 {[.C., 0, 1] -> [.X.];};
@repeat 5 {[.C., 0, 0] -> [.X.];};
@repeat 20 {[.C., 0, 1] -> [.X.];};
END
```

La directiva punto, *Y.ce = stop*, nos indica que la variable *Y* de 3 bits, se detendrá cuando *stop* sea cero lógico. Si se realiza la simulación funcional, se obtiene el siguiente diagrama de formas de onda.



1.a No obstante que *ISP Lever* permite realizar una simulación denominada como funcional (*Functional Simulation*), en realidad lo que hace es entregarnos las formas de onda (estados lógicos en alto y bajo) como se aprecia en la figura anterior. Un simulador funcional nos permite observar el comportamiento de nuestro diseño incorporando los elementos físicos que lo conforman, aunque a razón cierta, sigue siendo un ambiente virtual. Descarga los archivos necesarios para la práctica 3 y verifica en *ISIS de Proteus* el funcionamiento del diseño. Requieres el archivo JEDEC del contador y el archivo de Proteus. Es altamente recomendable que revises alguna bibliografía que muestre el uso del simulador. Cabe mencionar que los archivos proporcionados para cumplir este numeral de la práctica, están realizados en una versión "portable" de *ISIS de Preoteus*, por lo que no pudiera ser compatible con otras versiones; si este fuera el caso, tendrías que construir tu propio diagrama esquemático para verificar la simulación. A partir del *Chip Report* de *ISP Lever*, puedes interpretar el pinout utilizado para este diseño.



1.b Tomando como referencia el contador 00-99 diseñado en la práctica 2, modifica tus códigos para simular en ISIS de Proteus un contador 00-59, que obedece al módulo básico para un reloj digital.

2. (4 puntos) Diseño de contadores utilizando máquinas de estado.

En ABEL-HDL es muy sencillo modelar máquinas de estados. A continuación podemos observar la descripción de un contador ascendente-descendente de 4 bits. La variable *dir* permite cambiar la dirección del conteo. La salida *P* es combinatoria y sólo se pone a cero lógico en el estado quince.

```
Module count_edos
Title'Contador de 4 bits por diagrama de estados'

"Pines de entrada
```

```
       clk,rst,dir pin;
"Pines de salida
       q3,q2,q1,q0 pin istype 'reg,dc';
"Set
sreg=[q3,q2,q1,q0];

"Estados
E0=[0,0,0,0];
E1=[0,0,0,1];
E2=[0,0,1,0];
E3=[0,0,1,1];
E4=[0,1,0,0];
E5=[0,1,0,1];
E6=[0,1,1,0];
E7=[0,1,1,1];
E8=[1,0,0,0];
E9=[1,0,0,1];
E10=[1,0,1,0];
E11=[1,0,1,1];
E12=[1,1,0,0];
E13=[1,1,0,1];
E14=[1,1,1,0];
E15=[1,1,1,1];

c,x=.c.,.x.; "Sólo para el test_vectors se hace esta declaración

Equations

sreg.clk=clk;
sreg.re=rst;

State_diagram sreg

state E0:
if dir then E1 else E15;

state E1:
if dir then E2 else E0;

state E2:

if dir then E3 else E1;

state E3:
if dir then E4 else E2;

state E4:
if dir then E5 else E3;

state E5:
if dir then E6 else E4;

state E6:
if dir then E7 else E5;

state E7:
if dir then E8 else E6;

state E8:
if dir then E9 else E7;

state E9:
if dir then E10 else E8;

state E10:
if dir then E11 else E9;

state E11:
if dir then E12 else E10;

state E12:
if dir then E13 else E11;

state E13:
```

```
if dir then E14 else E12;

state E14:
if dir then E15 else E13;

state E15:
if dir then E0 else E14;


Test_vectors
([clk,rst,dir]->[q3,q2,q1,q0]);
 [c,1,1]->[x,x,x,x];
@repeat 20 {[ c,0,1] -> [x,x,x,x];};
@repeat 1 {[ c,1,1] -> [x,x,x,x];};
@repeat 20 {[ c,0,0] -> [x,x,x,x];};
 [c,1,0]->[x,x,x,x];

End
```

2.a Incorpora una variable *stop* para detener el conteo y una salida combinatoria (*P*) que detecte cuando un estado sea un número primo (1,2,3,5,7,11,13) y simula tu diseño en las dos vertientes tratadas al momento: con el simulador de forma de ondas de ISP Lever y con ISIS de Proteus. Posteriormente impleméntalo en un GAL22V10 y muestra su funcionamiento, conectándole un decodificador hexadecimal (0-F) implementado en otro GAL para visualizar el conteo en un display a siete segmentos.

2.b Siguiendo la misma lógica del diseño anterior, modifica tu código para describir un contador que de manera ascendente cuente sólo números pares y que de manera descendente cuente números primos, de 4 bits. Dibuja el diagrama de estados incluyendo todas las variables. No es necesaria una salida combinatoria, refiriéndonos al caso de *P* en el diseño realizado en *2.a*. Si tuvieras algún problema con el uso de recursos internos del GAL (no debería suceder), reduce este diseño a números de 3 bits. Simúlalo e impleméntalo físicamente.

3. **(3 puntos)** Existen casos prácticos que en el análisis estructurado se definen claramente a través de máquinas de estado, por ejemplo, los semáforos convencionales. A continuación podemos apreciar un código en ABEL-HDL que implementa un semáforo en un GAL22V10.

```
"Semáforo simple, codificado en ABEL e implementado en un GAL22V10
"Dr. Juan Carlos Herrera Lozada.
"Del STOP (rojo) debe pasar a GO (verde) de manera directa.
"Al término del tiempo en GO (verde), habilitará al mismo tiempo CAUTION (ambar).
"El trabajo del semáforo puede variar a criterio del diseñador.
"Lo limitado de las esperas (4 bits) se debe a que ISP Lever realiza asignaciones
"no concebidas y que afectan el mapeo final.

module sem_tra
title 'Controlador de Semáforo'
reloj, reset pin;
verde, ambar, rojo pin istype 'reg';

"Las salidas del contador se declaran como nodos
Count3..Count0 node istype 'reg';
Counter = [Count3..Count0];

"Se definen las condiciones de los estados
Luz = [verde, ambar, rojo];
GO = [ 1, 0, 0 ];
CAUTION = [ 1, 1, 0 ];
STOP = [ 0, 0, 1 ];
INI= [0, 0, 0];

"Monitoreo de señales para el reloj y el reset
Equations
Counter.ar= reset;
Counter.clk = reloj;
Luz.clk = reloj;
```

```
"Comienza diagrama de estados, nótese que el diagrama se nombra igual que el set de salida
state_Diagram Luz

"Condición inicial, necesaria para el arranque
state INI:
IF reset then INI
ELSE GO;

"Estados del semáforo
state GO:
IF (Counter < 15) then GO with
Counter := Counter + 1;
ELSE goto CAUTION with
Counter := Counter + 1;

state CAUTION:
IF (Counter != 3) then CAUTION with
Counter := Counter + 1;
ELSE goto STOP with
Counter := Counter + 1;

state STOP:
IF (Counter < 15) then STOP with
Counter := Counter + 1;
ELSE goto GO with
Counter := 1;

"Vectores de prueba, con la directiva @repeat
TEST_VECTORS
([reloj, reset]->[Luz])
@repeat 2 {[.c., 1]->[.x.];};   "Se propone un reset
@repeat 70 {[.c., 0]->[.x.];};  "Trabajo normal
end
```

3.a Simula en ISP Lever y en ISIS de Proteus, este semáforo. Posteriormente impleméntalo en el GAL22V10 y observa su funcionamiento.

DISEÑO EXTRA Y DISCUSIÓN
Valor: 2 puntos extras en la evaluación escrita.

La sincronización de semáforos no es problema trivial. Considerando un crucero, la idea es tener dos semáforos (pudiera ser que los dos estén en un mismo GAL o que estén en dispositivos independientes), y que ambos semáforos estén sincronizados, es decir, mientras un semáforo está en verde, el otro debe estar en rojo. Es muy importante recordar que cuando el que está en verde va a hacer su cambio a rojo, éste debe pasar a un estado intermedio que es el ámbar mientras que el segundo semáforo se mantiene en rojo y no cambiará a verde, sino hasta que el primero se ponga en rojo. Si quieres los dos puntos, ¿cómo harías esta sincronización? ¿Será muy complicado incluir una rutina de parpadeo (en el verde, antes de pasar al ámbar) para que sea mas real el comportamiento de nuestros semáforos? Muestra funcionando tu diseño.

Campo 4: **Conclusiones individuales.**