Apéndice 2

Uso de Abel

A2.1 Introducción

La minimización de funciones booleanas basada en mapas de Karnaugh se torna impracticable a medida que el número de variables aumenta; no es recomendable usar esta metodología para más de cuatro variables. Es necesario disponer de una aplicación computacional para enfrentar diseños lógicos con varias variables.

A2.2 Lenguaje Abel.

Es un lenguaje para describir diseños de sistemas digitales en forma jerárquica. **A**dvanced **B**oolean **E**quation **L**anguage.

Las descripciones se realizan en un archivo de texto (con extensión abl). Su orientación es para compilar diseños en dispositivos lógicos programables y permitir la simulación de éstos. Como un resultado intermedio, se obtienen las ecuaciones minimizadas del diseño.

Introduciremos el lenguaje vía ejemplos.

Se requiere asignar un *nombre al módulo*, que describa resumidamente lo que realiza. El módulo debe estar terminado con la palabra reservada END.

Pueden agregarse *comentarios* de líneas completas, empezando el texto del comentario con comillas dobles.

Luego se *identifican* o declaran las variables de entrada y salida.

La palabra reservada PIN, indica que son entradas o salidas del módulo. En el caso de las señales de salida se le agrega un declarador de tipo de salida; en el ejemplo se indica que la salida es combinacional con el identificador reservado, entre comillas simples, 'com'.

Luego viene una sección que describe las *ecuaciones* lógicas de las salidas en función de las entradas. Para el operador lógico and se emplea el símbolo ampersand &. Para el or se emplea el #. Para el operador unario not se emplea el símbolo ! como prefijo. Para el xor se emplea el símbolo $.

A2.3. Sistemas Combinacionales.

Ejemplo A2.1.

Describiremos en Abel el chip (pastilla) de la familia TTL 7408.

En esta componente vienen 4 ands de dos entradas.

```
MODULE V7408
"  Quad two input AND gate.   Cuatro ands de dos entradas.

" Declaraciones de señales de entrada
A0, A1, B0, B1, C0, C1, D0, D1 PIN;

" Declaraciones de señales de salida
YA, YB, YC, YD          PIN istype 'com';

EQUATIONS
YA = A0 & A1;
YB = B0 & B1;
YC = C0 & C1;
YD = D0 & D1;

END
```

Las descripciones anteriores permiten obtener las ecuaciones minimizadas del diseño. El siguiente listado muestra dichas ecuaciones como suma de productos y productos de sumas.

```
Equations:
YA = (A0 & A1);
YB = (B0 & B1);
YC = (C0 & C1);
YD = (D0 & D1);
Reverse-Polarity Equations:
!YA = (!A0  # !A1);
!YB = (!B0  # !B1);
!YC = (!C0  # !C1);
!YD = (!D0  # !D1);
```

Además puede agregarse un segmento con *vectores de pruebas*, éstos son estímulos que aplicados en las entradas permiten revisar funcionalmente las salidas del diseño (simulación funcional) y también realizar una simulación temporal en la que se aprecian las formas de ondas de las entradas y salidas con los retardos promedios que tienen las diferentes compuertas.

Para el caso del 7408 un conjunto de vectores de prueba para un and podría describirse según:

```
Test_vectors
( [A1, A0 ] -> [YA])
[0, 0] ->[ 0 ];
[0, 1] ->[ 0 ];
[1, 0] ->[ 0 ];
[1, 1] ->[ 1 ];
```

Cada renglón define un vector, asignando valores a las entradas y salidas.

En caso de dispositivos programables, los vectores de prueba permiten verificar el diseño, una vez que se ha grabado el dispositivo.

La *simulación funcional* aplica en las entradas los estímulos y calcula mediante las ecuaciones los valores de las salidas; y revisa que éstos coincidan con las salidas asignadas a cada uno de los vectores. Al final da un resumen si las diferentes pruebas pasaron o no.

```
AA Y
0 1 A

V0001  0 0 L
V0002  1 0 L
V0003  0 1 L
V0004  1 1 H
4 out of 4 vectors passed.
```

Efectuada la *simulación temporal* puede obtenerse la forma de onda de la salida, para los estímulos descritos en los vectores, las cuales muestran el retardo de propagación del canto de subida de la señal de salida (YA):

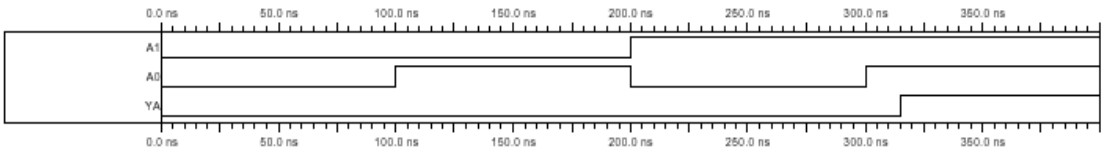


Figura A2.1 Simulación temporal.

Ejemplo A2.2.

El esquema siguiente ilustra, para una función de tres variables completamente especificada, la definición de la función mediante tablas de verdad.

```
"  Función de tres variables.
MODULE f3
" entradas
a, b, c  PIN;
" salidas
f     PIN ISTYPE 'COM';

EQUATIONS
truth_table ( [a, b, c]->f )
[0, 0, 0]-> 0;
[0, 0, 1]-> 1;
[0, 1, 0]-> 0;
[0, 1, 1]-> 1;
[1, 0, 0]-> 0;
```

```
[1, 0, 1]-> 0;
[1, 1, 0]-> 1;
[1, 1, 1]-> 1;
Test_vectors ( [a, b, c]->f )
[0, 0, 0]-> 0;
[0, 0, 1]-> 1;
[0, 1, 0]-> 0;
[0, 1, 1]-> 1;
[1, 0, 0]-> 0;
[1, 0, 1]-> 0;
[1, 1, 0]-> 1;
[1, 1, 1]-> 1;

END
```

Se obtiene la siguiente ecuación minimizada:

$$f = (a \,\&\, b \;\#\; !a \,\&\, c);$$

Con las siguientes formas de ondas:

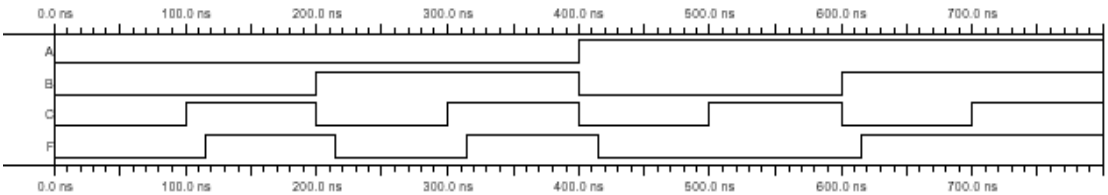


Figura A2.2 Formas de ondas Ejemplo A2.2.

Ejemplo A2.3.

El siguiente programa, permite obtener las ecuaciones minimizadas para una función de seis variables, descrita por sus mintérminos:

```
"  Ejemplo minimización .
"               g(a, b, c, d, e, f) = ∑(0, 2, 6, 7, 8, 10, 12, 14, 15, 41)
MODULE seisvar
" entradas
a, b, c, d , e, f  PIN;
" salida
g     PIN ISTYPE 'COM';

EQUATIONS
truth_table ( [a, b, c, d, e, f ]-> g)
0-> 1;          2-> 1;   6-> 1;
7-> 1;          8-> 1;   10-> 1;
12-> 1;         14-> 1; 15-> 1; 41-> 1;
```

END seisvar

Se obtienen las siguientes ecuaciones:
g = (a & !b & c & !d & !e & f) # (!a & !b & d & e) # (!a & !b & !d & !f) # (!a & !b & c & !f)
Con 22 entradas para suma de productos.

!g = (a & !c) # (d & !e & f) # (a & e) # (!a & !d & f) # (!c & d & !e) # (b) # (a & !f)
Con 23 entradas para producto de sumas.

Puede generarse una simulación temporal, que permite obtener las siguientes formas de ondas:

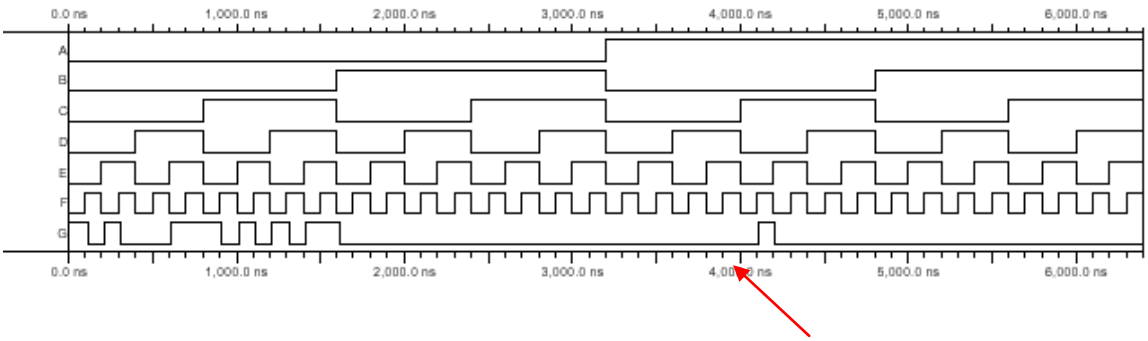


Figura A2.3 Simulación temporal Ejemplo A2.3.

Para lograr las formas de ondas, se generan los siguientes vectores de prueba:

Test_vectors ([a, b, c, d, e, f]-> g)
00-> 1;01-> 0;02-> 1;03-> 0;04-> 0;05-> 0;
06-> 1;07-> 1;08-> 1;09-> 0;10-> 1;11-> 0;
12-> 1;13-> 0;14-> 1;15-> 1;16-> 0;17-> 0;
18-> 0;19-> 0;20-> 0;21-> 0;22-> 0;23-> 0;
24-> 0;25-> 0;26-> 0;27-> 0;28-> 0;29-> 0;
30-> 0;31-> 0;32-> 0;33-> 0;34-> 0;35-> 0;
36-> 0;37-> 0;38-> 0;39-> 0;40-> 0;41-> 1;
42-> 0;43-> 0;44-> 0;45-> 0;46-> 0;47-> 0;
48-> 0;49-> 0;50-> 0;51-> 0;52-> 0;53-> 0;
54-> 0;55-> 0;56-> 0;57-> 0;58-> 0;59-> 0;
60-> 0;61-> 0;62-> 0;63-> 0;

Ejemplo A2.4.

En el siguiente ejemplo se emplea notación de buses (grupos de señales reconocidas por un nombre), también comentarios de fin de línea. Se agrega la opción de colocar un título al módulo.

MODULE V7442
TITLE 'Decodificador de BCD a Decimal. '

```
" entradas
A3..A0 pin;  " Definidas por rangos, desde A3 hasta A0
" salidas
Y9..Y0  pin;
" Declaración de Señal.  Se asigna nombre a un bus.
BCD = [A3..A0];

EQUATIONS
!Y0 = (BCD == ^h0);          "el prefijo ^h indica un número hexadecimal.
!Y1 = (BCD == ^h1);
!Y2 = (BCD == ^h2);
!Y3 = (BCD == ^h3);
!Y4 = (BCD == ^h4);
!Y5 = (BCD == ^h5);
!Y6 = (BCD == ^h6);
!Y7 = (BCD == ^h7);
!Y8 = (BCD == ^h8);
!Y9 = (BCD == ^h9);
END
```

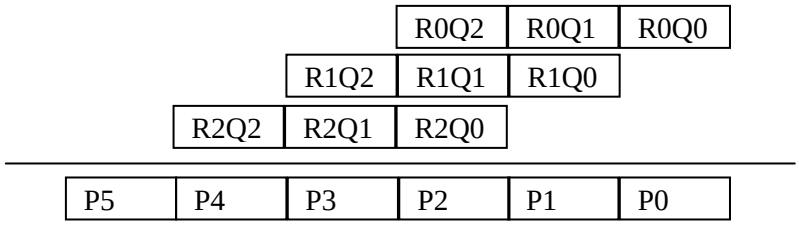
Ejemplo A2.5 Multiplicador combinacional.

Puede expresarse mediante una tabla de verdad el producto (con 2n bits) de un multiplicando por un multiplicador de n bits cada uno.

Para operandos de tres bits, la tabla tiene 64 renglones.

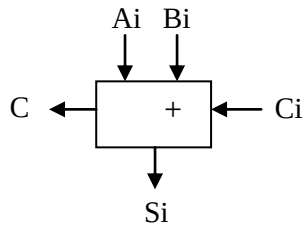
Multiplicando	Multiplicador	Producto
000	000	000000
000	001	000000
....	....	....
001	001	000001
....	....	
111	110	101010
111	111	110001

Consideremos el algoritmo de multiplicación para papel y lápiz que se emplea para los números decimales. Sea Q el multiplicando, formado por los bits: Q2, Q1, Q0. Y R el multiplicador, formado por los bits: R2, R1 y R0. Resulta un producto P, formado por los bits: P5, P4, P3, P2, P1 y P0.



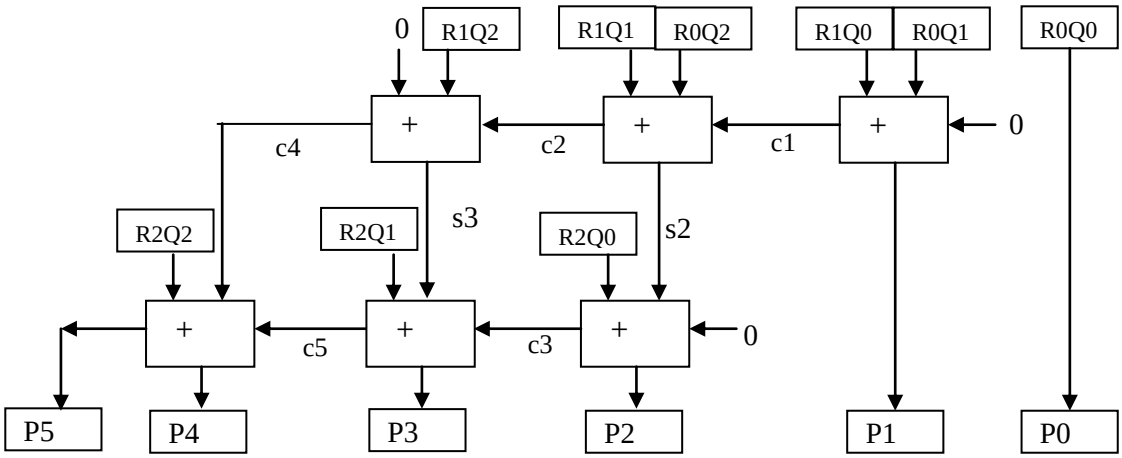
Cada producto individual RiQi puede expresarse como la operación and de sus términos. Es decir, RiQi = Ri & Qi, ya que sólo la multiplicación de 1 por 1 da 1.

Empleamos el siguiente símbolo para un sumador completo de dos bits(Ai, Bi) más una reserva de entrada(Ci). El sumador tiene dos salidas: la suma(Si) y la reserva de salida(Co).



Las ecuaciones para el sumador, pueden describirse según:
Si = (Ai $ Bi) $ Ci; Co = (Ai & Bi) # (Ai & Ci) # (Bi & Ci);

Puede entonces implementarse el siguiente diagrama para el multiplicador combinacional:



Del diagrama pueden escribirse las siguientes ecuaciones:
P0 = R0 & Q0;
P1= (R1 & Q0) $ (R0 & Q1) ;
c1 = (R1 & Q0) & (R0 & Q1);
s2= ((R1 & Q1) $ (R0 & Q2)) $ c1;
c2 = ((R1 & Q1) & (R0 & Q2)) # ((R1 & Q1) & c1) # ((R0 & Q2) & c1);

```
P2 = ( (R2 & Q0) $ s2);
c3 = ((R2 & Q0) & s2);
s3= ( (R1 & Q2) $ c2 ) ;
c4 =  (R1 & Q2) & c2 ;

P3 = ((R2 & Q1) $ s3) $ c3;
c5 = ((R2 & Q1) & s3) # ((R2 & Q1)  & c3) # (s3 & c3);
P4= ( (R2 & Q2) $ c4 ) $ c5;
P5 = ( (R2 & Q2) & c4 ) # ( (R2 & Q2) & c5) # (c4 & c5);
```

Si crece el número de bits de los operandos, no resulta adecuado plantear las ecuaciones a partir de un diagrama. Las siguientes ecuaciones realizan al multiplicador combinacional, de una manera más compacta, y pueden modificarse para tratar operandos mayores. Se deja al compilador Abel la tarea de minimizar las ecuaciones.

```
R = [R2..R0];
Q = [Q2..Q0];
P =  [P5..P0];
PP0 = R0 & [0,     0,   0,  Q2, Q1, Q0];
PP1 = R1 & [0,     0, Q2, Q1, Q0,    0];
PP2 = R2 & [0,  Q2, Q1, Q0,    0,    0];

equations
P = PP0 + PP1 + PP2;
```

A2.4. Diseño de Máquinas Secuenciales.

Ejemplo A2.5. Detector secuencia 0101.

A continuación se tiene el diagrama de estados de una máquina secuencial (Modelo de Mealy) que produce una salida alta cada vez que se detecta la secuencia 0101 en la entrada; y salida cero en el resto de los casos.

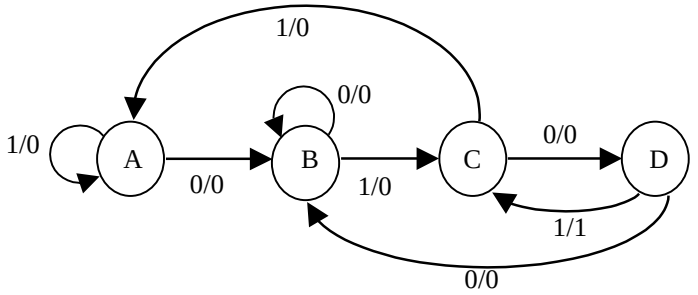


Figura A2.4 Diagrama de estados detector secuencia 0101.

La siguiente secuencia de entrada produce las siguientes secuencias de salida y de transiciones de estados.

Entrada										....
Salida										....
Pxo. Estado										….

Figura A2.5 Secuencias de entrada, salida y de estados.

El siguiente módulo describe el diagrama de estados.

En las declaraciones se emplea el tipo reg, para declarar una salida registrada; es decir, una salida de un flip-flop D. En el diseño se tienen dos salidas: z es una salida combinacional, que como veremos es asincrónica; mientras que la salida zs es de tipo sincrónica.

Luego se declara el registro de estado (sreg), en función de las salidas de los flip-flops.

También se efectúa un mapeo de nombres lógicos a físicos. Esto también se denomina asignación de estados. El estado inicial, será el estado A; cuyo nombre binario será [0, 0].

En las ecuaciones se indican las señales de entrada que ingresan al reloj (clock), a la habilitación de la salida (enab, no se usa enable ya que es palabra reservada) y al reseteo asincrónico de la macrocelda (reset).

Luego se describe el diagrama de estados de sreg, mediante la especificación de cada estado. En la cual se indica las transiciones de acuerdo a los valores de las entradas (esto se indica con la sentencia if then else); y los valores que deben tomar las salidas (esto se realiza con la sentencia with). Debe notarse que las salidas registradas se asignan con :=, y que a las salidas combinacionales se les asigna un valor con el signo igual. Pueden agruparse acciones, mediante el uso de paréntesis cursivos { }.

Finalmente se especifican vectores de prueba. Se emplea la constante .c. para modelar un pulso de reloj, equivale a generar tres vectores, uno con clock en cero, luego un valor de clock uno; y finalmente un valor de clock cero. Como el resto de las entradas no cambian, se cumplen automáticamente las especificaciones de tiempos de set-up y hold para la variable de entrada x.

La constante .C. es adecuada para flip-flops que operan con cantos de subida; la constante .K. genera la secuencia 101 en el reloj, que es adecuada para flip-flops que operan con cantos de bajada. Además se tienen los estímulos .D. que genera un canto de bajada, con la secuencia 10; la constante .U. genera un canto de subida, con la secuencia 01.

Se generan los vectores asociados a la tabla anterior con las secuencias de la entrada, salida y estados.

```
MODULE estado1

TITLE 'Diagrama de Estados. Modelo de Mealy'
"Declaraciones
q1, q0, zs pin istype 'reg';  "Estado y Salida Registrada
clock, enab, reset, x pin ;   "Entradas
z pin istype 'com';           "Salida Combinacional.

sreg  = [q1,q0];
"Valores de los estados
A= 0; B= 1; C= 2; D= 3;
```

```
equations
[q1,q0,zs].clk= clock;
[q1,q0,zs].oe = !enab;
[q1,q0,zs].ar = reset; "Con reset va a [0, 0](el estado A).

state_diagram sreg;
 State A:
   zs := 0;   if (!x) then B with z = 0;   else A with z = 0;
 State B:
   if (!x) then B with z = 0;   else C with z = 0;
 State C:
   if (!x) then D with z = 0;   else A with z = 0;
 State D:
   if (!x) then B with z = 0;   else C with {z = 1; zs := 1;}
test_vectors (       [clock,enab,reset, x]->[sreg,zs])
            [.c.   ,   0  ,   1  , 0 ]->[A   ,0];
            [.c.   ,   0  ,   0  , 1 ]->[A   ,0];
            [.c.   ,   0  ,   0  , 0 ]->[B   ,0];
            [.c.   ,   0  ,   0  , 1 ]->[C   ,0];
            [.c.   ,   0  ,   0  , 0 ]->[D   ,0];
            [.c.   ,   0  ,   0  , 1 ]->[C   ,1];
            [.c.   ,   0  ,   0  , 1 ]->[A   ,0];
            [.c.   ,   0  ,   0  , 0 ]->[B   ,0];
            [.c.   ,   0  ,   0  , 0 ]->[B   ,0];
            [.c.   ,   0  ,   0  , 1 ]->[C   ,0];
            [.c.   ,   0  ,   0  , 1 ]->[A   ,0];
    END
```

Después de la compilación, las ecuaciones reducidas resultan:

```
q1.OE = (!enab);
q1 := (q1.FB & !q0.FB & !x) # (q0.FB & x);
q1.AR = (reset);
q1.C = (clock);

q0.OE = (!enab);
q0 := (!x);
q0.AR = (reset);
q0.C = (clock);

zs.OE = (!enab);
zs := (q1.FB & q0.FB & x);
zs.AR = (reset);
zs.C = (clock);

z = (q1.FB & q0.FB & x);
```

En la Figura A2.6 se ilustra la notación empleada para las variables asociadas a cada salida registrada. Nótese que cuando q1 aparece al lado izquierdo de la asignación, está representando

a la entrada D del flip-flop, es el próximo estado que tomará dicha variable; es decir q1(k+1), con la notación empleada en clases.

Cuando aparece en el lado derecho el término q1.FB (por feed-back) se lo interpreta como la salida del flip-flop.

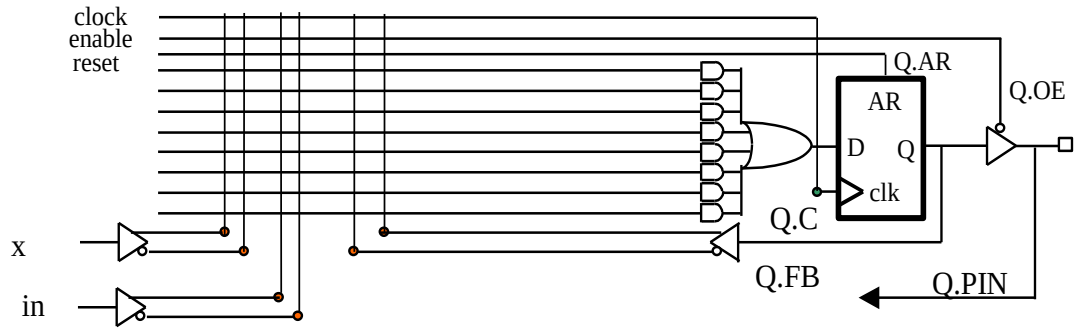


Figura A2.6 Variables internas. Visión del programador.

Nótese que la expresión booleana para la salida asincrónica z es idéntica a la de zs, que es salida de flip-flop.

La salida del reporte de simulación muestra los estímulos y respuestas debidos a los vectores elegidos.

```
                 c   r
                 l e e
                 o n s
                 c a e   q q z
                 k b t x 1 0 s

V0001 C 0 1 0  Z Z Z
      C 0 1 0  L L L
      C 0 1 0  L L L
      C 0 1 0  L L L
V0002 C 0 0 1  L L L
      C 0 0 1  L L L
      C 0 0 1  L L L
V0003 C 0 0 0  L L L          Clock = 0
      C 0 0 0  L H L          Clock = 1
      C 0 0 0  L H L          Clock = 0
V0004 C 0 0 1  L H L
      C 0 0 1  H L L
      C 0 0 1  H L L
V0005 C 0 0 0  H L L
      C 0 0 0  H H L
      C 0 0 0  H H L
V0006 C 0 0 1  H H L
      C 0 0 1  H L H
      C 0 0 1  H L H
```

```
        C 0 0 1   H L H
V0007   C 0 0 1   H L H
        C 0 0 1   L L L
        C 0 0 1   L L L
V0008   C 0 0 0   L L L
        C 0 0 0   L H L
        C 0 0 0   L H L
V0009   C 0 0 0   L H L
        C 0 0 0   L H L
        C 0 0 0   L H L
V0010   C 0 0 1   L H L
        C 0 0 1   H L L
        C 0 0 1   H L L
V0011   C 0 0 1   H L L
        C 0 0 1   L L L
        C 0 0 1   L L L
11 out of 11 vectors passed.
```

Debe notarse que cada vector genera tres estímulos (esto debido al pulso definido con .c.).
Los vectores generan las siguientes formas de ondas:

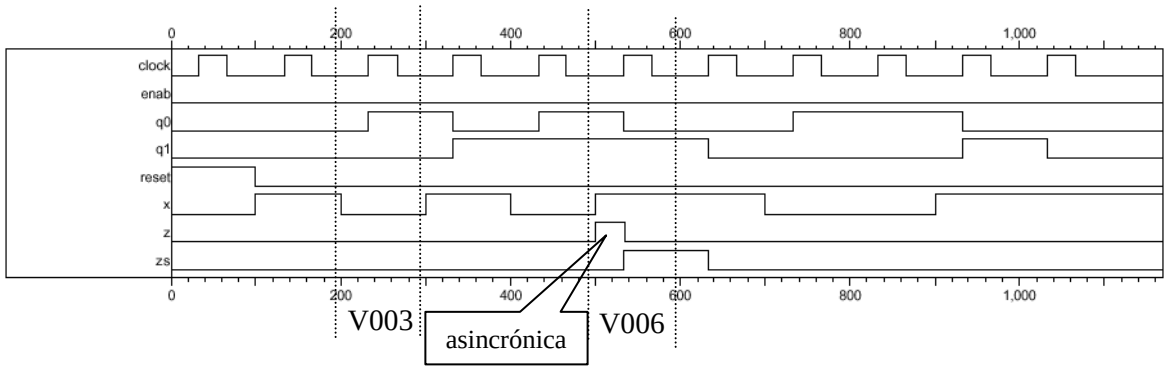


Figura A2.7 Simulación funcional. Formas de ondas generadas por estímulos.

La señal z cambia entre dos cantos de subida del reloj, por esto se dice que es asincrónica respecto de éste. Mientras que la señal zs, que se obtiene como salida de un flip-flop será sincrónica con éste.

La salida z cambia en la transición, apenas se produce la condición que la genera. La señal zs ocurre generalmente un poco después, ya que se activa cuando ocurre el siguiente canto de subida del reloj.

La señal z, es una salida de una máquina de Mealy, y depende de las entradas; lo cual puede verse ya que interviene la entrada x en la ecuación: z = (q1.FB & q0.FB & x);

Si esta salida está conectada como entrada a otra máquina secuencial, que ocupe el mismo reloj, podría no cumplir las condiciones para operación sincrónica. Por lo tanto es recomendable emplear la salida sincronizada zs.

Con dispositivos programables conviene emplear el modelo de Moore. En este caso las salidas están sincronizadas con el reloj.

Ejemplo A2.6. Reconocedor de un patrón finito. Modelo de Moore

Con entrada x y salida z; la salida se activa cada vez que se presenta la secuencia 010, y mientras que la secuencia 100 no se haya presentado, en cuyo caso la salida se desactiva permanentemente.

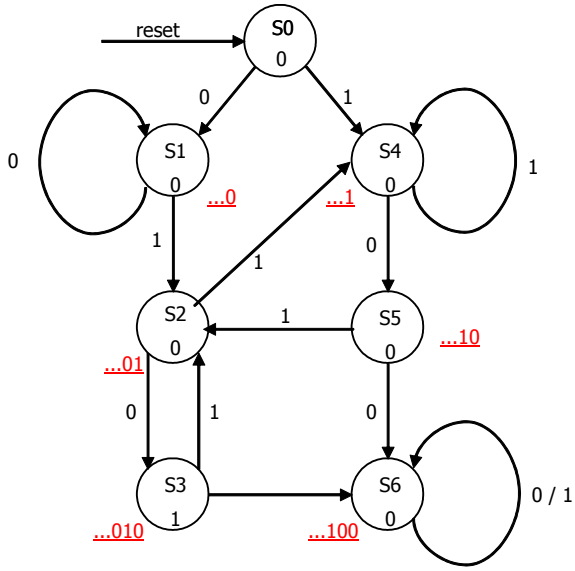


Figura A2.8 Diagrama de Moore Ejemplo A2.6.

El siguiente programa, implementa el diseño:

```
MODULE fsm010
TITLE 'fsm010'
"Activa Z cada vez que llega 010 y mientras no llegue 100.
"si llega 100 se activa Y, y permanece en ese estado.

"Declaraciones
"Pines de entrada
      clk, Xin, RESET   pin;
"Pines de Salida
      Q0, Q1, Q2 pin   istype 'reg';
      Z, Y pin istype 'com';
"Registro de Estado
SREG = [Q0, Q1, Q2];
"Definicion de Estados
S0   = [0,0,0];  " Estado Inicial. (Reset state)
S1   = [0,0,1];  " secuencias tipo ...0
S2   = [0,1,0];  " secuencias tipo ...01
S3   = [0,1,1];  " secuencias tipo ...010
```

```
S4   = [1,0,0];  " secuencias tipo ...1
S5   = [1,0,1];  " secuencias tipo ...10
S6   = [1,1,0];  " secuencias tipo ...100

equations
      [Q0,Q1,Q2].clk= clk;
      [Q0.ar, Q1.ar, Q2.ar] = RESET;              "Con Reset va a S0
      "          Z=!Q0&Q1&Q2; Y=Q0&Q1&!Q2;
      "Modelo de Moore

state_diagram SREG
state S0:  Z=0; Y=0;   if Xin then S4 else S1;
state S1:              if Xin then S2 else S1;
state S2:              if Xin then S4 else S3;
state S3:  Z=1;        if Xin then S2 else S6;
state S4:              if Xin then S4 else S5;
state S5:              if Xin then S2 else S6;
state S6:  Y=1;        goto S6;

test_vectors ([clk, RESET, Xin] -> [Z,Y])

      [.C.,1,.X.] ->[0,0];        "reset
      [.C.,0, 0] -> [0,0];        "secuencia 00101010010
      [.C.,0, 0] -> [0,0];
      [.C.,0, 1] -> [0,0];
      [.C.,0, 0] -> [1,0];
      [.C.,0, 1] -> [0,0];
      [.C.,0, 0] -> [1,0];
      [.C.,0, 1] -> [0,0];
      [.C.,0, 0] -> [1,0];
      [.C.,0, 0] -> [0,1];
      [.C.,0, 1] -> [0,1];
      [.C.,0, 0] -> [0,1];

END fsm010;
```

Nótese la definición de las salidas asociadas al estado. Y el salto incondicional, mediante el goto.

Las salidas también podrían haberse definido en el segmento de las ecuaciones, esto se insinúa como comentario.

Después de la compilación, resultan las ecuaciones reducidas:

```
Q0 := (!Q0.FB & Q1.FB & Q2.FB & !Xin) # (Q0.FB & !Q1.FB & !Xin) # (Q0.FB & !Q2.FB)
   # ( !Q2.FB & Xin);
Q0.AR = (RESET);
Q0.C = (clk);

Q1 := (!Q0.FB & Q1.FB & !Xin) # ( !Q0.FB & Q2.FB & Xin) # (Q0.FB & Q1.FB & !Q2.FB)
   # (Q0.FB & !Q1.FB & Q2.FB);
```

```
Q1.AR = (RESET);
Q1.C = (clk);

Q2 := (!Q1.FB & !Q2.FB & !Xin) # (!Q0.FB & !Q2.FB & !Xin) # (!Q0.FB & !Q1.FB & !Xin);
Q2.AR = (RESET);
Q2.C = (clk);

Z = (!Q0.FB & Q1.FB & Q2.FB);
Y = (Q0.FB & Q1.FB & !Q2.FB);
```

Con estas ecuaciones, podría haberse efectuado el diseño sin emplear diagrama de estados.

Con los vectores se generan las siguientes formas de ondas:

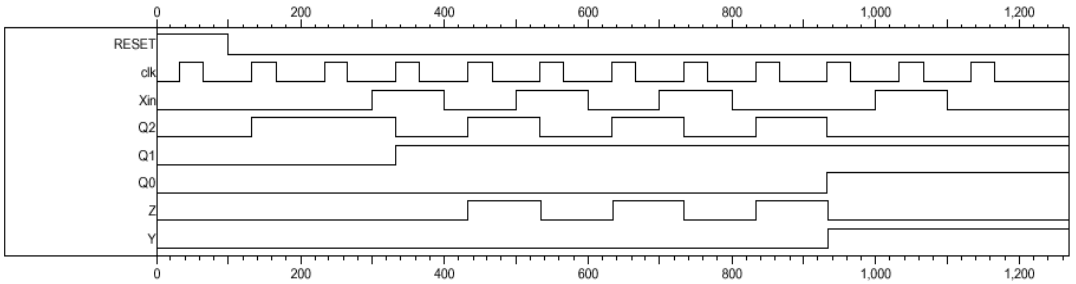


Figura A2.9 Formas de ondas Ejemplo A2.6.

Ejemplo A2.7. Contador sincrónico con control de modo M.

Si M = 0 el contador es binario ascendente; si M = 1 el contador avanza según código Gray. A continuación se indican las secuencias de cuentas:

binario: 000, 001, 010, 011, 100, 101, 110, 111
Gray: 000, 001, 011, 010, 110, 111, 101, 100

El diagrama de estados se construye para la secuencia binaria, con transiciones con entrada de control igual a cero. Luego, se marcan las transiciones para contar en Gray.

En este diseño las salidas corresponden al estado directamente. Como es usual en contadores. Ver ejemplo 9.6.9.

```
MODULE cntgray
TITLE 'Contador en Gray'

"Pines de Entrada
      clk, M, RESET   pin   1, 2, 3;
"Pines de Salida
      Z0, Z1, Z2      pin  istype 'pos, reg';
"Registro de Estado
SREG = [Z0, Z1, Z2];
S0   = [0,0,0];
S1   = [0,0,1];
S2   = [0,1,0];
```

```
S3   = [0,1,1];
S4   = [1,0,0];
S5   = [1,0,1];
S6   = [1,1,0];
S7   = [1,1,1];
equations
        [Z0,Z1,Z2].clk= clk;
        [Z0.ar, Z1.ar, Z2.ar] = RESET; "Reset asincrónico lleva al estado S0

state_diagram SREG
state S0: goto S1;
state S1: if M then S3 else S2;
state S2: if M then S6 else S3;
state S3: if M then S2 else S4;
state S4: if M then S0 else S5;
state S5: if M then S4 else S6;
state S6: goto S7;
state S7: if M then S5 else S0;
test_vectors ([clk, RESET, M] -> [Z0, Z1, Z2])
        [0,1,.X.] -> [0,0,0];
        [.C.,0,0] -> [0,0,1];
        [.C.,0,0] -> [0,1,0];
        [.C.,0,1] -> [1,1,0];
        [.C.,0,1] -> [1,1,1];
        [.C.,0,1] -> [1,0,1];
        [.C.,0,0] -> [1,1,0];
        [.C.,0,0] -> [1,1,1];
end cntgray
```

Las ecuaciones resultan, después de la compilación:
Equations:

```
Z0 := (Z0.FB & !Z1.FB & !M) # (Z0.FB & !Z2.FB & !M) # (Z0.FB & Z2.FB & M)
    # (!Z0.FB & Z1.FB & Z2.FB & !M) # (Z1.FB & !Z2.FB & M);
Z0.AR = (RESET);
Z0.C = (clk);

Z1 := (!Z0.FB & Z2.FB & M) # (!Z1.FB & Z2.FB & !M) # (Z1.FB & !Z2.FB);
Z1.AR = (RESET);
Z1.C = (clk);

Z2 := (!Z0.FB & !Z1.FB & M) # (Z0.FB & Z1.FB & M) # (!Z2.FB & !M);
Z2.AR = (RESET);
Z2.C = (clk);
```

Los estímulos generan las siguientes formas de ondas:

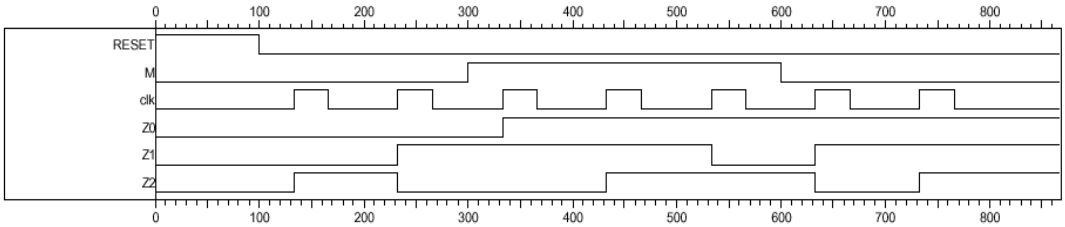


Figura A2.10 Formas de ondas contador Gray.

Ejemplo A2.8.

A través del lenguaje puede describirse un flip-flop JK, mediante:

Q:= (J& !Q) #(!K&Q);

La Figura A2.11 muestra la implementación de un JK, mediante un dispositivo lógico programable, los cuales tienen flip-flops de tipo D en su estructura interna:

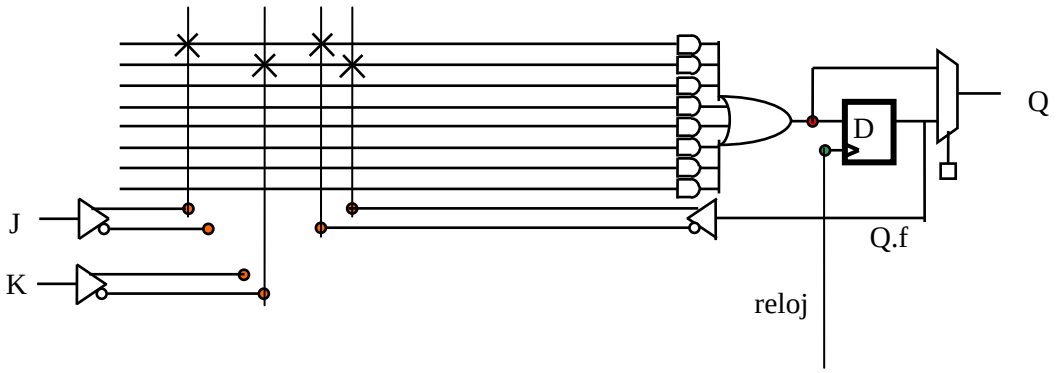


Figura A2.11. Implementación de JK en CPLD

Índice general

Índice de Figuras