ABEL Para Síntesis

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

Caso de Estudio 1: Diseño Combinacional

ABEL permite la descripción de circuitos mediante tablas de verdad, ecuaciones y diagramas de estado. Para iniciar el caso de estudio, utilizaremos el siguiente código escrito en ABEL para ispLEVER. En él, se modela un comparador de números de 4 bits.

```
" Comparador de 2 números de cuatro bits cada uno, implementado en un GAL22V10
" CIDETEC - IPN

MODULE compara;
TITLE 'Comparador de 4 bits';
"Definiciones
     a3..a0 pin 2..5;
     b3..b0 pin 6..9;
     may, men, eq pin 21..23 istype 'com';
"Declaración de Sets
     a = [a3..a0];
     b = [b3..b0];
"Descripción a través de when then else
EQUATIONS
         when (a==b) then {may=0; eq= 1; men=0}
     when (a<=b) then {may=0; eq= 0; men=1}
         when (a>b)  then {may=1; eq= 0; men=0}

TEST_VECTORS
([a, b] -> [may, eq, men]);
[2, 2] -> [.x., .x., .x.];
[5, 2] -> [.x., .x., .x.];
[0, 2] -> [.x., .x., .x.];
[9, 9] -> [.x., .x., .x.];
END
.
```

Analiza el **Código 1**. Se trata de la descripción de un sumador de 2 + 2 bits. Es posible advertir que el código comienza con la declaración del módulo y de los pines del mismo. En esta primera parte, sólo consideraremos diseños combinacionales por lo que los pines de salida serán tipo 'com'.

La palabra reservada TRUTH_TABLE, indica la descripción mediante una tabla de verdad. Las variables de entrada y salida que definen los pines físicos, se integran a la descripción mediante ([entradas] -> [salidas]). Nótese que los valores de las variables están por default en decimal. La tabla de vectores de prueba muestra las salidas a Don't Care (.X.), con la intención de que el sintetizador de ABEL no realice la comprobación contra el mapeo dentro del dispositivo, sino sólo la traducción a formas de onda.

```
"Código 1
"Código ABEL que describe el funcionamiento de un sumador de dos más dos bits, sin considerar
"el acarreo inicial. Se utilizó una tabla de verdad como elemento de descripción.
"ISP Expert hacia un GAL22V10.

MODULE sum_tt
TITLE 'Mi sumador 2 + 2 en tabla de verdad'
"Pines de entrada
        A1,A0,B1,B0 pin 1,2,3,4;

"Pines de Salida
        S2, S1, S0 pin 17,18,19 istype 'com';

"Descripción del funcionamiento a través de una tabla de verdad
TRUTH_TABLE
([A1,A0,B1,B0]->[S2,S1,S0])
[0,0,0,0]->[0,0,0];
```

```
[0,0,0,1]->[0,0,1];
[0,0,1,0]->[0,1,0];
[0,0,1,1]->[0,1,1];
[0,1,0,0]->[0,0,1];
[0,1,0,1]->[0,1,0];
[0,1,1,0]->[0,1,1];
[0,1,1,1]->[1,0,0];
[1,0,0,0]->[0,1,0];
[1,0,0,1]->[0,1,1];
[1,0,1,0]->[1,0,0];
[1,0,1,1]->[1,0,1];
[1,1,0,0]->[0,1,1];
[1,1,0,1]->[1,0,0];
[1,1,1,0]->[1,0,1];
[1,1,1,1]->[1,1,0];

"Vectores de prueba para la simulación
TEST_VECTORS
([A1,A0,B1,B0]->[S2,S1,S0])
[0,0,0,0]->[.x.,.x.,.x.];
[0,0,0,1]->[.x.,.x.,.x.];
[0,1,0,1]->[.x.,.x.,.x.];
[1,0,0,1]->[.x.,.x.,.x.];
[1,0,1,0]->[.x.,.x.,.x.];
[1,0,1,1]->[.x.,.x.,.x.];
[1,1,1,0]->[.x.,.x.,.x.];
[1,1,1,1]->[.x.,.x.,.x.];
END
```

El **Código 2** muestra una descripción mediante ecuaciones. Se trata del mismo sumador de dos bits anterior, sólo que sí incluye el acarreo de entrada. El diseño parte del planteamiento de dos sumadores completos conectados en cascada. El acarreo de salida del primer sumador (bit menos significativo) se conecta como el acarreo de entrada del segundo sumador; por lo mismo, este acarreo no genera salida física ya que se mantiene interno al proceso y en el código se declara como una ecuación intermedia, antes de la palabra reservada EQUATIONS. También es posible definir desde el principio un nodo (NODE); sin embargo, ISP muestra algunos problemas con esta asignación.

```
"Código 2
"Código ABEL que describe el funcionamiento de un sumador de dos más dos bits, considerando
"el acarreo inicial. Se utilizó una captura de ecuaciones como elemento de descripción.
"ISP Expert hacia un GAL22V10.


MODULE sum_eq
TITLE 'Mi sumador 2+2 descrito con ecuaciones'
"Declaración de una constante
        Cin0=0;
"Pines de entrada
        A0, A1, B0, B1 pin 1..4;
"Pines de salida
        SUM0, SUM1, SUM2 pin istype 'com';
"Ecuación Intermedia (no causa salida, se trata como un nodo).
NOD0=A0&B0#A0&Cin0#B0&Cin0;

EQUATIONS
SUM0=A0$B0$Cin0;
SUM1=A1$B1$NOD0;
SUM2=A1&B1#A1&NOD0#B1&NOD0;

TEST_VECTORS
([A1, A0, B1, B0] ->[SUM2, SUM1, SUM0]);
 [0, 1, 0, 1] -> [.X., .X., .X.];          "1+1
 [1, 0, 1, 0] -> [.X., .X., .X.];          "2+2
 [1, 1, 1, 1] -> [.X., .X., .X.];          "3+3
 [0, 0, 0, 1] -> [.X., .X., .X.];          "0+1
END
```

ABEL también permite la descripción de circuitos aritméticos combinacionales a través de operaciones aritméticas directas. En el caso de las sumas, ABEL genera de forma automática sumadores en configuración Carry Look-ahead con predicción de acarreo (arquitectura paralela) por lo que resultan en grandes cantidades de hardware. Para mantener la configuración en cascada, es necesario incluir la directiva @Carry 1, que indica que se limitarán los acarreos a un solo bit. Obsérvese el **Código 3**.

```
“Código 3
"Sunador de cuatro bits, con directiva @carry
"ISP Expert, para un GAL22V10

MODULE add4
TITLE 'Mi sumador de 4 + 4 con operador directo y @carry'
"Líneas de datos
        A3, A2, A1, A0, B3, B2, B1, B0 pin 1..8;
        R4, R3, R2, R1, R0 pin 15, 16, 17, 18, 19 istype 'com';

"constantes, para mantener números de 5 bits. ABEL no permite trabajar con tamaños diferentes
A4=0;
B4=0;

"Declaración de Sets
a=[A4, A3, A2, A1, A0];
b=[B4, B3, B2, B1, B0];
r=[R4, R3, R2, R1, R0];
"Declaración de Sets, sólo para simulación
a_in=[A3, A2, A1, A0];
b_in=[B3, B2, B1, B0];

EQUATIONS
@carry 1; "Directiva que limita el Carry look-ahead
r=a+b;    "Adición directa

TEST_VECTORS
([a_in, b_in] ->[r]);
[3, 3] -> [6];              "3+3
[5, 5] -> [10];             "5+5
[7, 7] -> [14];             "7+7
[8, 10] -> [18];            "8+10
END
```

El **Código 4** muestra el diseñó de un multiplicador de 3 x 3 a través de ecuaciones. Se sigue el diseño de sumadores conectados matricialmente, por lo que se tienen ecuaciones intermedias.

```
“Código 4
"Multiplicador 3 x 3 en un GAL22V10

MODULE mul3X3
TITLE 'Mi multiplicador de 3 x3 bits'
        a2, a1, a0, b2, b1, b0   pin 2..7;
        p5,p4,p3,p2,p1,p0    pin 18..23 istype 'com';

"Declaración de Sets, sólo para simulación
a = [a2,a1,a0];
b = [b2,b1,b0];
p = [p5,p4,p3,p2,p1,p0];

"Ecuaciones Intermedias (internas, para la conexión en cascada de los sumadores)
      sum00 = a0 & b0;

      sum01 = a1 & b0;

      sum02 = a2 & b0;

      sum10 = ((a0 & b1) $ sum01);
      carry10 = (a0 & b1 & sum01);

      sum11 = ((a1 & b1) $ sum02);
      carry11 = (a1 & b1 & sum02);

      sum12 = (a2 & b1) $ carry11;
      carry12 = (a2 & b1) & carry11;


      sum20 = ((a0 & b2) $ sum11) $ carry10;
      carry20 = (carry10 & a0 & b2)
              # (a0 & b2 & sum11)
              # (sum11 & carry10);

      sum21 = ((a1 & b2) $ sum12) $ carry20;
      carry21 = (carry20 & a1 & b2)
              # (a1 & b2 & sum12)
              # (sum12 & carry20);

      sum22 = ((a2 & b2) $ carry12) $ carry21;
      carry22 = (carry21 & a2 & b2)
              # (a2 & b2 & carry12)
              # (carry12 & carry21);

"Comienza declaración de ecuaciones de salida
EQUATIONS
      p0 = sum00;
      p1 = sum10;
```

```
      p2 = sum20;
      p3 = sum21;
      p4 = sum22;
      p5 = carry22;

TEST_VECTORS
([a,b] -> [p])
[3, 3]->[.X.];
[7, 7]->[.X.];
[5, 4]->[.X.];
[0, 7]->[.X.];
[1, 6]->[.X.];
END
```

Para circuitos combinacionales que infieren alguna condición de comparación, se premedita la descripción When Then Else, tal y como se aprecia en el multiplexor 8 a 1 del **Código 5** y el comparador del **Código 6**.

```
"Código 5
"Multiplexor 8 a 1
MODULE mi_mux
TITLE 'Mi multiplexor 8 a 1, diseñado por jcrls'
"Líneas de datos, nótese cómo se definen los pines
        L0..L7 pin 1..8;
"Líneas de control
        A2, A1, A0 pin 19,18,17;
"Salida única
        Y pin 16 istype 'com';
"Declaración de un Set
       A=[A2, A1, A0];
"Declaración de un Set. Observa que este Set es sólo para simulación, no tiene efecto
"directo sobre el funcionamiento del multiplexor, por lo que se puede omitir si tu código
"no incluye vectores de simulación
       L=[L7, L6, L5, L4, L3, L2, L1, L0];
"Descripción del funcionamiento
EQUATIONS
        WHEN (A==0) THEN Y=L0;
        WHEN (A==1) THEN Y=L1;
        WHEN (A==2) THEN Y=L2;
        WHEN (A==3) THEN Y=L3;
        WHEN (A==4) THEN Y=L4;
        WHEN (A==5) THEN Y=L5;
        WHEN (A==6) THEN Y=L6;
        WHEN (A==7) THEN Y=L7;
TEST_VECTORS
([L, A] ->[Y]);
[15, 0] -> [.X.];          "L3, L2, L1 y L0 están en 1, Y = 1 por la referencia a L0
[15, 4] -> [.X.];          "Y debe mostrar un 0, ya que está referido a L4
[64, 0] -> [.X.];          "Y debe mostrar un 0, ya que está referido a L0
[64, 6] -> [.X.];          "Y debe mostrar un 1, ya que está referido a L6
END


"Código 6
"Comparador de 2 números de cuatro bits cada uno, implementado en un GAL22V10

MODULE compara;
TITLE 'Mi comparador de 4 bits';
"Definiciones
        a3..a0  pin;
        b3..b0  pin;
        may, men, eq pin istype  'com';
"Declaración de Sets
        a = [a3..a0];
        b = [b3..b0];
"Descripción a través de when then else
"Observa que con {} agrupas una serie de respuestas a la misma condición.
"cada respuesta debe terminar con ";" no así el {}.
EQUATIONS
        when (a==b) then {may=0; eq= 1; men=0;}
        else when (a<=b) then {may=0; eq= 0; men=1;}
        else  {may=1; eq= 0; men=0;}

TEST_VECTORS
([a, b] -> [may, eq, men]);
[2, 2] -> [.x., .x., .x.];
[5, 2] -> [.x., .x., .x.];
[0, 2] -> [.x., .x., .x.];
[9, 9] -> [.x., .x., .x.];
END
```

Los **Códigos 7** y **8**, muestran la aproximación al diseño de Unidades Resolutivas, similares a las que incluye una ALU simple. Se evidencia la facilidad de la estructuración en ABEL comparándolo con OPAL en el diseño combinacional.

“Código 7

```
"Unidad Lógica simple de 4 bits: con fn=00, se realiza la AND entre a y b.
"ISP Expert, para un GAL22V10

MODULE uni_log
TITLE 'Mi unidad lógica simple en un GAL22V10'
"Líneas de datos
        A3, A2, A1, A0, B3, B2, B1, B0, FN1, FN0 pin 1..10;
        R3, R2, R1, R0 pin istype 'com';
"Declaración de Sets
a=[A3, A2, A1, A0];
b=[B3, B2, B1, B0];
fn=[FN1, FN0];
r=[R3, R2, R1, R0];

EQUATIONS
WHEN fn==0 THEN r=a&b;              "a and b
ELSE WHEN fn==1 THEN r=a#b;         "a or b
ELSE WHEN fn==2 THEN r=a$b;         "a xor b
ELSE r=!a;                          "not a

TEST_VECTORS
([a, b, fn] ->[fn]);
[3, 3, 0] -> [.X.];           "and
[5, 1, 1] -> [.X.];           "or
[7, 0, 2] -> [.X.];           "xor
[4, 2, 3] -> [.X.];           "not
END
```

“Código 8

```
"Unidad Aritmética básica de 3 bits con directiva @carry, para suma y resta
"ISP Expert, para un GAL22V10

MODULE un_arit
"Pines de entrada
        A2, A1, A0, B2, B1, B0, op pin 1..7;
"Pines de salida
        R3, R2, R1, R0 pin 15, 16, 17, 18  istype 'com';

"constantes, para mantener números de 5 bits. ABEL no permite trabajar con tamaños diferentes
A3=0;
B3=0;

"Declaración de Sets
a=[A3, A2, A1, A0];
b=[B3, B2, B1, B0];
r=[R3, R2, R1, R0];
"Declaración de Sets, sólo para simulación
a_in=[A2, A1, A0];
b_in=[B2, B1, B0];

EQUATIONS
@carry 1;                "Directiva que limita el Carry look-ahead
when op==0 then r=a+b;   "Adición directa
else r=a-b;

TEST_VECTORS
([a_in, b_in, op] ->[r]);
[3, 3, 0] -> [.x.];
[5, 2, 1] -> [.x.];
[7, 7, 0] -> [.x.];
[3, 2, 1] -> [.x.];
END
```

Para cambiar de manera simple un diseño combinacional a un diseño secuencial, sólo es necesario incluir una señal de sincronización (reloj) y asignar los pines de salida como ‘reg’, tal y como lo muestra el **Código 9** (referirse al **Código 2**, para la contraparte combinacional). Observa que el reloj se asigna obligatoriamente en el pin 1. Las salidas registradas deben monitorearse por medio del reloj, por lo tanto a cada salida en las ecuaciones se le indica que espere por el reloj a través de `SUM0.c = reloj`. En las ecuaciones de salida resultantes, la asignación se hace con `:=` para indicar que la salida cambiará con el reloj. Finalmente en la simulación, se incorpora la señal de reloj y su estímulo por default `.C.` para simular los pulsos.

“Código 9

```
"Código ABEL que describe el funcionamiento de un sumador de dos más dos bits. Diseño secuencial.

MODULE sum_reg
```

```
TITLE 'Mi sumador con salidas registradas de 2+2, descrito con ecuaciones'
"Declaración de una constante
        Cin0=0;
"Pines de entrada, incluyendo el reloj (obligatoriamente asignado al pin 1)
        reloj, A0, A1, B0, B1 pin 1..5;
"Pines de salida tipo registrados
        SUM0, SUM1, SUM2 pin istype 'reg';
"Ecuación Intermedia (no causa salida, se trata como un nodo).
NOD0=A0&B0#A0&Cin0#B0&Cin0;

EQUATIONS
SUM0.C=reloj;
SUM1.C=reloj;
SUM2.C=reloj;


SUM0 :=A0$B0$Cin0;
SUM1 :=A1$B1$NOD0;
SUM2 :=A1&B1#A1&NOD0#B1&NOD0;

TEST_VECTORS
([reloj, A1, A0, B1, B0] ->[SUM2, SUM1, SUM0]);
 [.c., 0, 1, 0, 1] -> [.X., .X., .X.];          "1+1
 [.c., 1, 0, 1, 0] -> [.X., .X., .X.];          "2+2
 [.c., 1, 1, 1, 1] -> [.X., .X., .X.];          "3+3
 [.c., 0, 0, 0, 1] -> [.X., .X., .X.];          "0+1
END
```

Ejercicios Propuestos, a implementarse en un GAL22V10:

1. Modifica el **Código 1** para que diseñes un decodificador hexadecimal a siete segmentos de cátodo común.
2. Considerando que en ocasiones no se dispone de un display de cátodo común o viceversa, diseña un decodificador que con una línea de selección te permita trabajar con un display de cátodo o con uno de ánodo.
3. Realiza un restador de dos números de 3 bits cada uno, partiendo de la conexión en cascada de sumadores configurados en complemento a dos. Realiza la captura mediante ecuaciones de manera similar al diseño del **Código 2**.
4. Analiza el **Código 3**, cambiando la directiva @Carry a diferentes valores y observando las ecuaciones generadas por el sintetizador de ABEL.
5. Realiza una ALU simple que te permita realizar una suma y operaciones lógicas con dos números de dos bits cada uno. Considera algunas señales de bandera, si así lo estimas necesario.