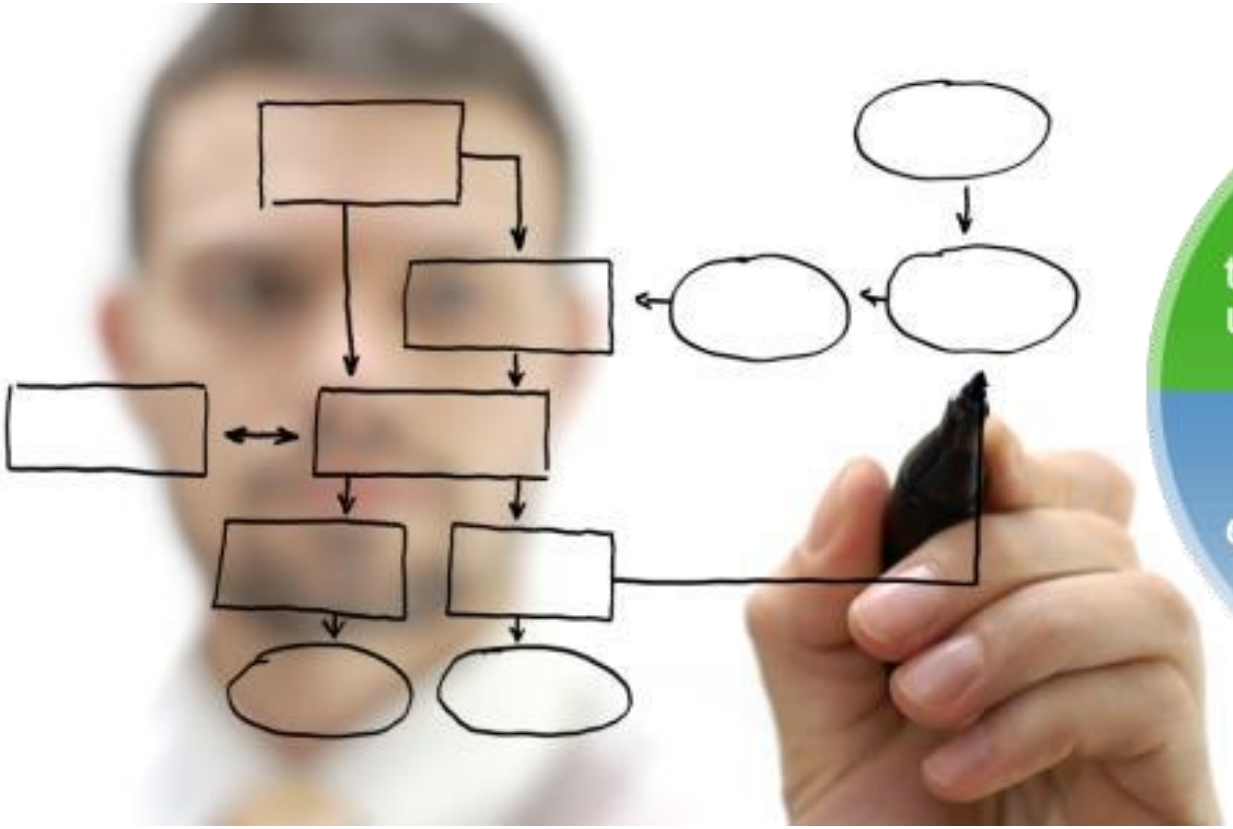
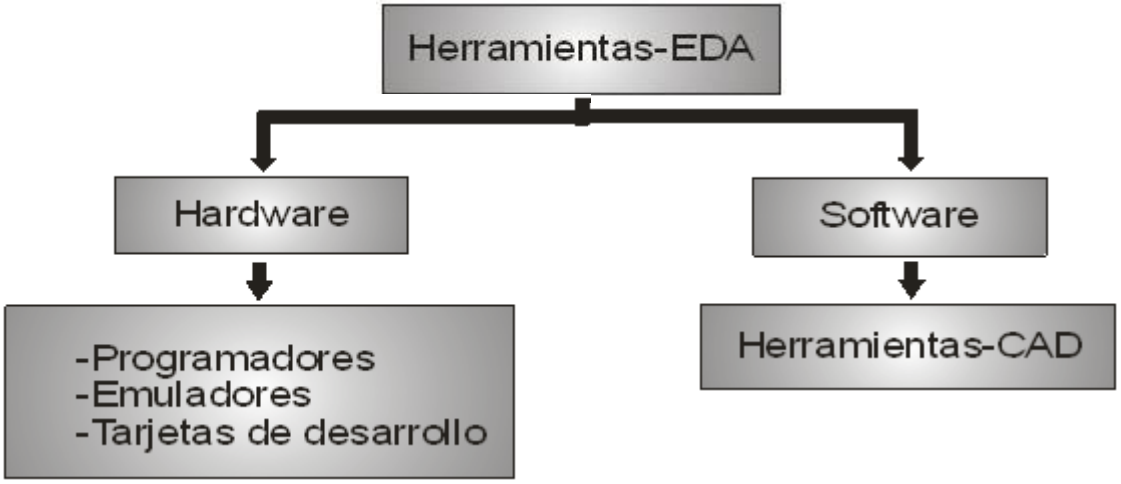
Metodología de diseño de un sistema digital



Dr. Enrique Guzmán Ramírez
Universidad Tecnológica de la Mixteca

Herramientas EDA

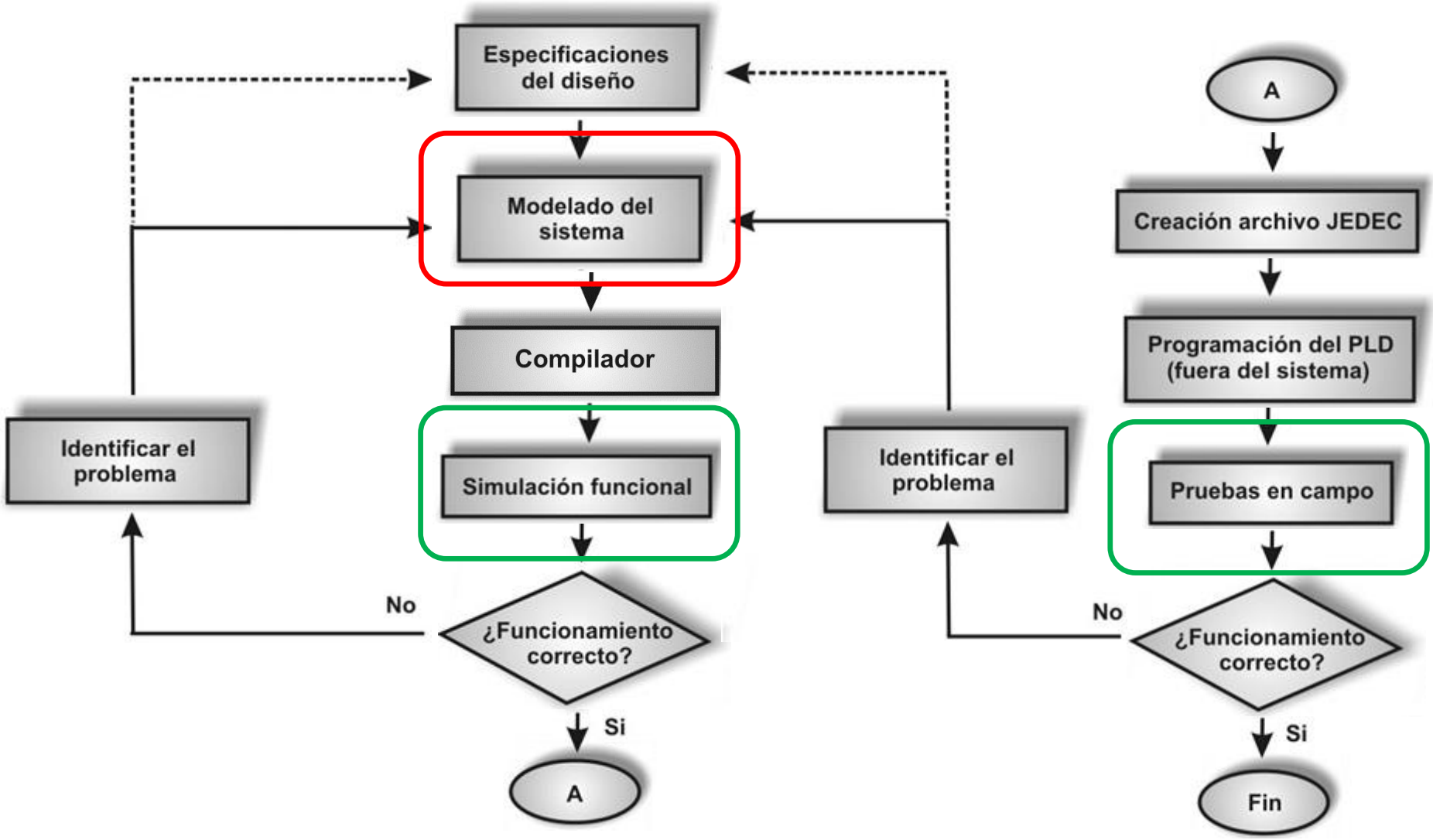
- Todas las herramientas, hardware o software, empleadas en el diseño de sistemas electrónicos reciben el nombre de Herramientas para la **“Automatización del Diseño Electrónico”**, *EDA (Electronic Design Automation).*



Herramientas CAD para el diseño de hardware.

- **Diseño asistido por computadora**, *CAD (Computer Aided Design)*.
- Herramientas EDA-CAD para CDC:
  - Modelado o descripción.
  - Simuladores.
    - Simulación funcional.
    - Simulación temporal.
  - Implementación.
    - Síntesis.
    - Mapeo.
    - Ruteo.
  - Programación.

Ciclo de desarrollo - SPLD



Ciclo de desarrollo – SPLD

Etapa de Modelado

- En el modelado de un sistema digital basado en SPLD, se debe definir.
  - La metodología de diseño que será empleada en el desarrollo del sistema.
  - El tipo de modelado que se utilizara.
  - El nivel de abstracción empleado en dicho modelado.

Metodología de diseño

- En la "metodología de diseño" se analiza el sistema para determinar cuáles son los elementos que lo integran y definir cómo se abordara su modelado.
- El paradigma "Jerárquico" es el más comúnmente usado
  - Se describe el circuito como un esquema jerárquico de bloques interconectados.
  - Adecuada para diseños grandes.
  - Tiene 2 variantes:

Metodología *top-down*

Se divide el diseño en bloques
Se diseña cada bloque

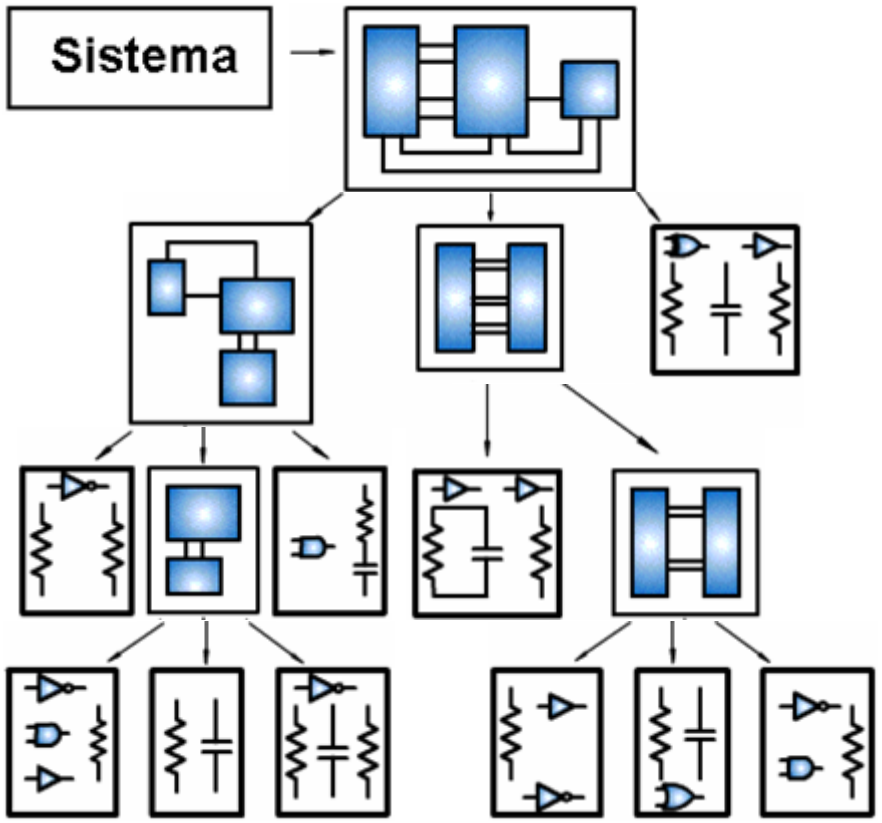
Metodología *bottom-up*

Se diseñan bloques
Se construye el diseño con ellos

Metodología de diseño

- Metodología “top-down” (descendente)
  - El modelado de un sistema comienza con un nivel de abstracción alto.
  - Después, el sistema es dividido en sub-sistemas.

- Los sub-sistemas son nuevamente divididos.
- La finalidad es que en cada división el nivel de abstracción disminuya.
- Se considera que un sub-sistema llego a su nivel más bajo cuando
  - Puede ser representado por las primitivas que la herramienta de diseño ofrece.
  - Puede ser descrito por elementos estrechamente relacionados con hardware



Tipo de modelado

- Definida la metodología de diseño, usada para determinar los elementos que formaran parte del sistema, se debe decidir cuál "tipo de modelado" será empleado en la descripción de cada elemento.
- Existen diversas formas de modelar un diseño digital sobre un CDC:
  - Esquemático.
  - Diagramas de máquinas de estado.
  - Lenguajes Descriptores de Hardware (HDL, *Hardware Description Languages*).
- Para un SPLD solo existe uno, los HDLs.

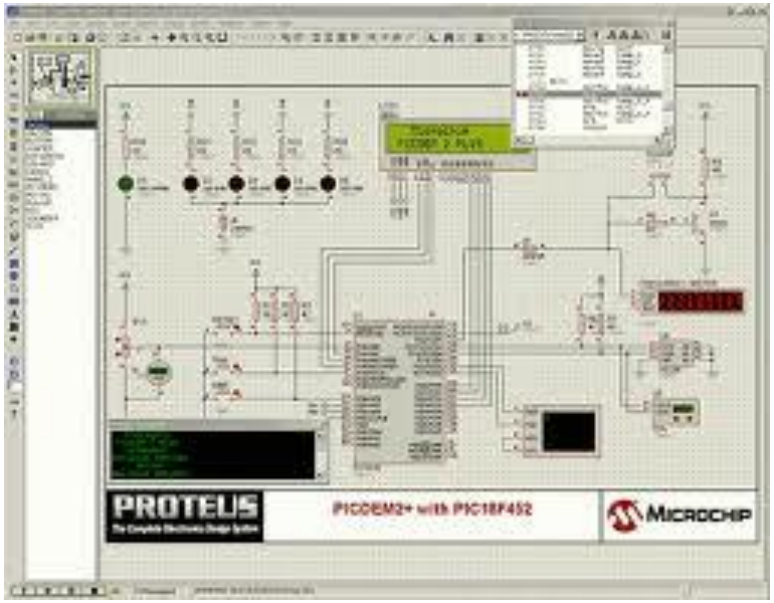
Modelado mediante HDL

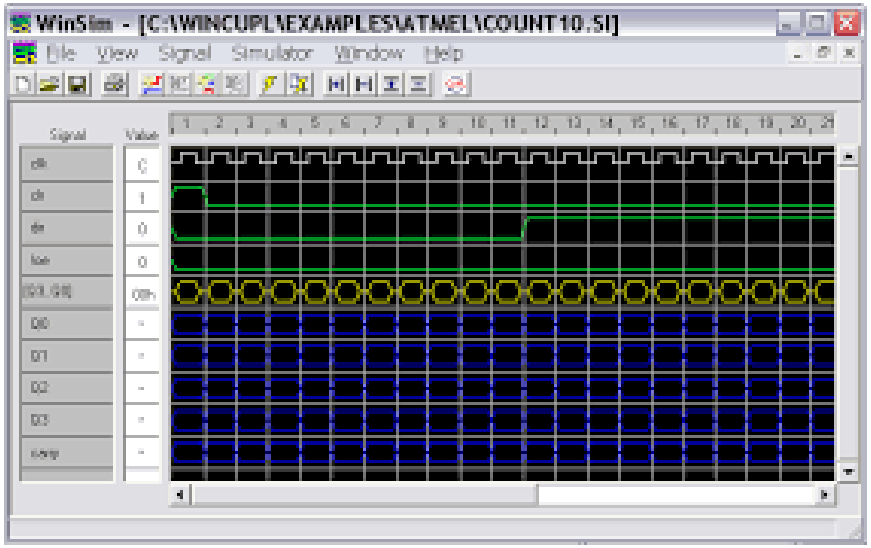
- ¿Qué es un HDL?

  “Un HDL es un lenguaje especializado para modelar o describir y/o simular la estructura o el comportamiento de un sistema digital utilizando un enfoque de software”.

Modelado mediante HDL

- La descripción con un HDL de un circuito puede ser la entrada de una *herramienta de simulación*.
- La herramienta de simulación permite verificar el comportamiento de un sistema y/o experimentar con diferentes descripciones antes de su implementación física.

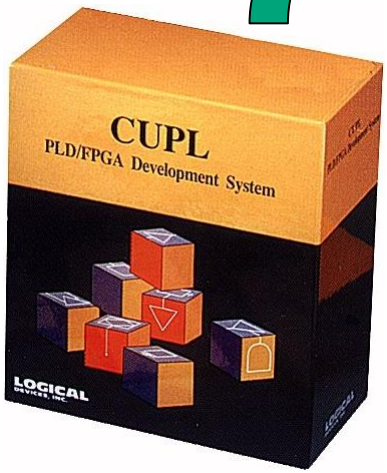


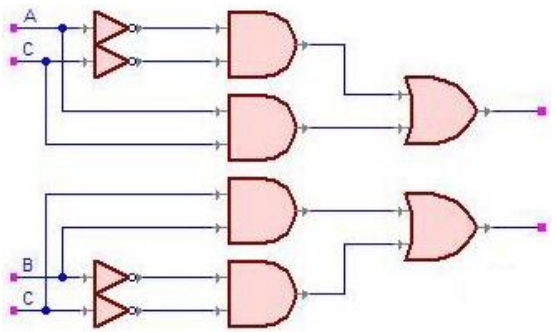


Modelado mediante HDL

- La descripción con un HDL de un circuito puede ser la entrada de una *herramienta de síntesis*.
- La herramienta de síntesis transforma la descripción HDL a una representación que puede ser implementada físicamente en un dispositivo (*a nivel de compuerta*).

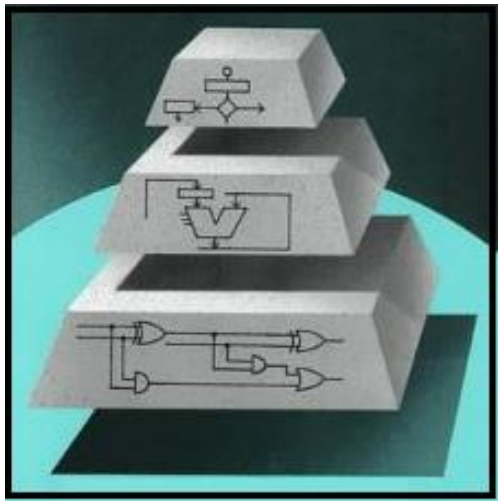
```
/* implement FSM                                    */
SEQUENCED state_n {
PRESENT S0
              IF !H & LS & !RS NEXT S1;
              IF !H & !LS & RS NEXT S2;
              IF H # !LS & !RS NEXT S3;
              DEFAULT NEXT S3;
PRESENT S1
              IF H NEXT S0;
              IF !H & !LS & RS NEXT S2;
              IF !H & !RS NEXT S3;
              DEFAULT NEXT S3;
PRESENT S2
              IF H NEXT S0;
              IF !H & LS & !RS NEXT S1;
              IF !H & !LS NEXT S3;
              DEFAULT NEXT S3;
PRESENT S3
              IF H NEXT S0;
              IF !H & LS & !RS NEXT S1;
              IF !H & !LS & RS NEXT S2;
              IF !H & !LS & !RS NEXT S3;
              DEFAULT NEXT S3;
}
```





Nivel de abstracción

- Cuando se modela un sistema digital mediante un HDL, se debe definir el "nivel de abstracción" que será utilizado.
- Los HDL usados para modelar sobre SPLDs, ofrecen los siguientes niveles de abstracción:
  - Nivel de compuertas.
  - Algorítmico, funcional o de comportamiento.



Lenguaje Descriptor de Hardware

CUPL

(Universal Compiler for Programmable Logic)

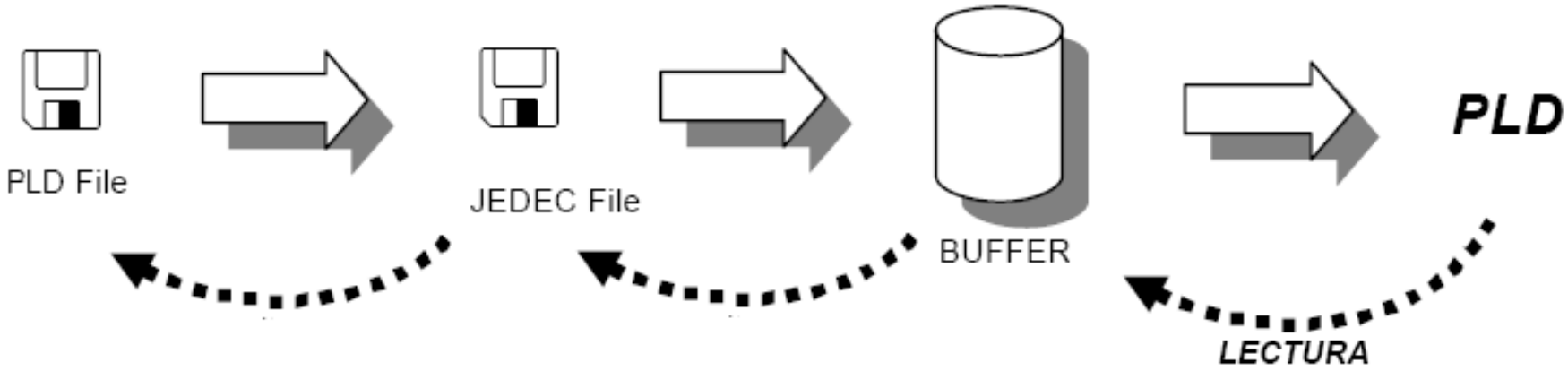
CUPL

- Información:
  - Documentos
  - CUPL ver 5.30.4

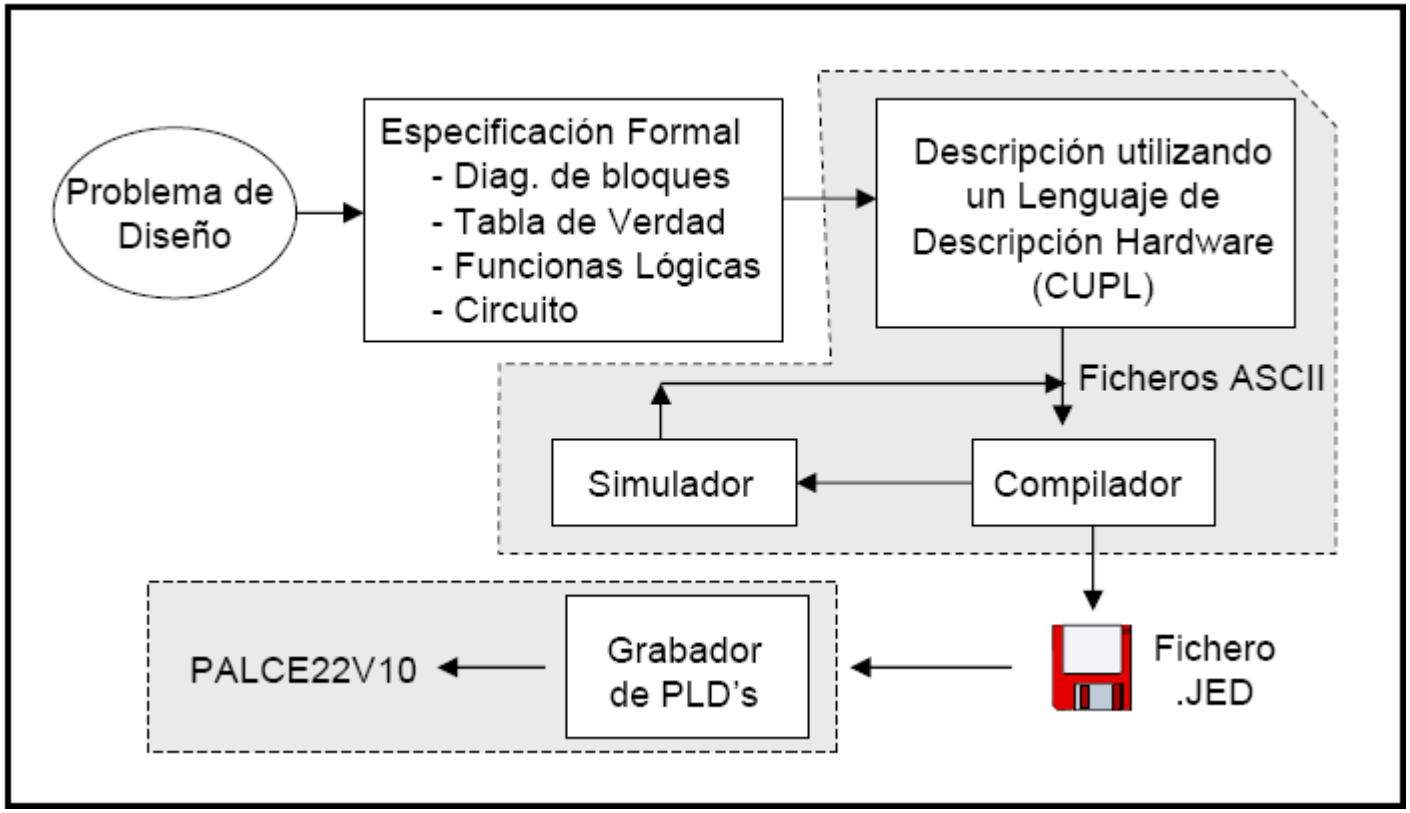
www.atmel.com

CUPL

- CUPL es un HDL (herramienta EDA-CAD) creado por la compañía **Logical Devices. Inc**.
- Se trata de un **compilador lógico** que es utilizado para el diseños lógico sobre SPLD´s.
- Tiene por función generar un archivo programable (JEDEC, *Joint Electronic Device Engineering Council Standard*) a partir del modelado de un sistema digital: *archivo.jed*



Flujo de modelado para un SPLD mediante CUPL



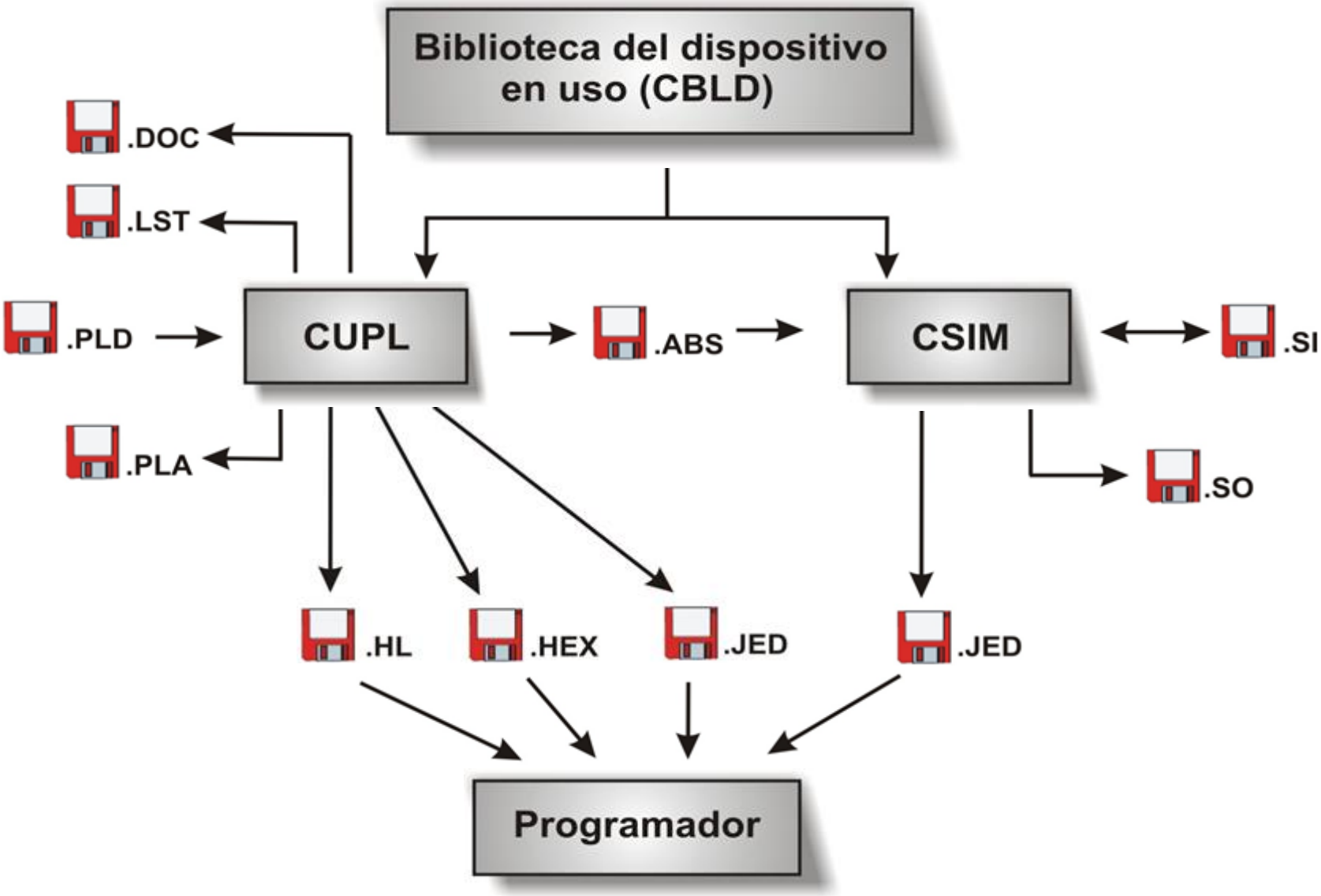
CUPL

- La herramienta EDA-CAD CUPL está constituida por:
  - **Compilador (CUPL)**. Este módulo permite sintetizar una descripción lógica escrita con la sintaxis del CUPL; además permite asignarla a un dispositivo lógico específico.
  - **Simulador (CSIM)**. Permite visualizar el comportamiento del sistema modelado.
  - **Administrador de librerías (CBLD)**. Permite manejar las librerías del programa.

CUPL, características

- Las características básicas que ofrece el CUPL son:
  - Aplicabilidad Universal.
  - Modelar usando un enfoque de software.
  - Documentación Flexible.
  - Simplificador.
  - Simulador.

CUPL, estructura de archivos



CUPL

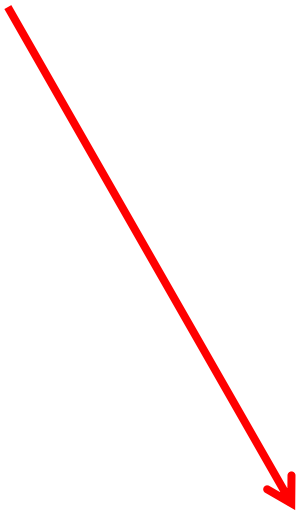
- CUPL permite especificar formalmente un sistema digital mediante:
  - Ecuaciones.
  - Tabla de verdad.
  - Maquinas de estado.
  - Elementos de diseño abstracto de hardware.
  - Cualquier combinación de éstas.

CUPL

```
o0 = sel:'h'0 & ctr:'h'4;
o1 = sel:'h'1 & ctr:'h'4;
o2 = sel:'h'2 & ctr:'h'4;
o3 = sel:'h'3 & ctr:'h'4;
```

```
table x,cont => [q2,q1,q0].d{
'b'1000 => 1;
'b'1001 => 2;
'b'1010 => 3;
'b'1011 => 4;
'b'1100 => 5;
'b'1101 => 6;
'b'1110 => 7;
'b'1111 => 0;
}
```

```
SEQUENCE qm{
present 0
   If ema&asc next 1;
   If emd&des next 7;
   If hol next 0;
   default next 0;
present 1
   If ema&asc next 2;
   If emd&des next 0;
   If hol next 1;
   default next 1;
```



```
q0.d = !q0;
q1.d = q1&!q0#!q1&q0;
q2.d = q2&!q0#!q1&q2#!q2&q1&q0;
```

Sintaxis del CUPL

```
Name Funciones lógicas;
Partno GAL16V8;
Revision 01;
Date 03/01/03;
Designer J.Beltran;
Company Universidad Nacional;
Location X;
Assembly X;
Device G16V8;

/*************************************************************/
/* Archivo Fuente de ejemplo en CUPL para implementar funciones lógicas */
/*************************************************************/

/* Definición de las entradas */

Pin 1 = a;
Pin 2 = b;

/* Definiciónde las salidas */

Pin 12 = inva;
Pin 13 = invb;
Pin 14 = and;
Pin 15 = nand;
Pin 16 = or;
Pin 17 = nor;
Pin 18 = xor;
Pin 19 = xnor;

/* Definición de Ecuaciones Lógicas*/

inva = !a;                  /* Inversión de las entradas a y b*/
invb = !b;
and = a & b;               /* Función AND */
nand = !(a & b);          /* Función NAND */
or = a # b;                 /* Función OR*/
nor = !(a # b);            /* Función NOR */
xor = a $ b;               /* Función XOR */
xnor = !(a $ b);          /* Función XOR Negada*/
```

CUPL, notación

- Comentarios. Permiten documentar el modelado del diseño.

```
/* Forma de añadir
    cometarios, varias líneas  */
// Comentarios, una línea o segmento de línea
```

- Las líneas de comando, como en 'C', deben estar separadas por **punto y coma ';'**

CUPL, notación

- Nombres de variables (identificadores).
- Reglas:
  - Se pueden usar caracteres alfanuméricos y el símbolo “_”. Ejemplos:

    Entrada_1 sal2_1 var_int
  - Los nombres de las variables no pueden tener espacios.
  - El lenguaje es sensible al uso de mayúsculas y minúsculas.

    NOM1 ≠ nom1 ≠ Nom1
  - Se puede definir un conjunto de variables indexadas en forma simplificada:

    [I0..3] /* similar a definir I0,I1,I2,I3 */

CUPL, notación

- Definición de variables no indexadas:

  [up,down,hold,rst]

- No puede usarse palabras reservadas.

APPEND	ASSEMBLY	ASSY	COMPANY	CONDITION
DATE	DEFAULT	DESIGNER	DEVICE	ELSE
FIELD	FLD	FORMAT	FUNCTION	FUSE
GROUP	IF	JUMP	LOC	LOCATION
MACRO	MIN	NAME	NODE	OUT
PARTNO	PIN	PINNNODE	PRESENT	REV
REVISION	SEQUENCE	SEQUENCED	SEQUENCEJK	SEQUENCERS
SEQUENCET	TABLE			

CUPL, notación

- Números.
  - Los números pueden ser representados en:

Número	Base	Dígitos
'b'1101; 'B'1101	Binario	0,1
'o'663; 'O'663	Octal	0,…,7
'd' 92; 'D' 92	Decimal	0,…,9
'h' BA; 'H' BA	Hexadecimal	0,…,9,A,…,F

  - La base por default es la Hex.

CUPL, notación

- Definición de rangos, únicamente en números oct, bin y hex.

  'h'[3F00..3FFF] 'o' [12..15] 'b' [000..011]

- Números expresados en bin, oct o hex pueden tener valores no importa:

  'O'0X6 'B'11X1 'H'1XX0

  'H'[1,3,7,E];

CUPL, comandos de preprocesado

- Comandos de preprocesado.
- Son comandos especiales cuya función es facilitar la escritura del modelado del diseño.
  - Ejemplos:

    $define

    $repeat

    $macro

CUPL, operadores

- El CUPL maneja 4 tipos de operadores:
  - Operadores de asignación.
  - Operadores lógicos.
  - Operador relacional
  - Operadores aritméticos.

CUPL, operadores

- Operadores de asignación

  “=”

  Usado en casi cualquier asignación:

  - Pines, Fields.
  - Ecuaciones, (asignar una expresión a una señal).

  “=>”

  Usado únicamente en tablas de verdad.

  Estos operadores tienen la prioridad más baja.

CUPL, operadores

- Operadores lógicos.

Operador	Descripción	Ejemplo	Prioridad
!	NOT	!A	1
&	AND	A & B	2
#	OR	A # B	3
$	XOR	A $ B	4

CUPL, operadores

- Operador relacional (operador de igualdad).

  “ : ” Este operador:

  - Permite determinar si existe igualdad entre un conjunto de variables (**FIELD)** y una constante.
  - Permite definir expresiones intermedias.
  - Permite definir operadores de reducción.
  - No permite comparar conjuntos de variables entre si

```
s = a:b;          /* s = !b0 & b1 & !b2 */
s = a:b:c;        /* ERROR */
s = (a:b):c;      /* ERROR */
```

CUPL, operadores

- Operadores aritméticos.

Operador	Descripción	Ejemplos	Prioridad
**	Exponente	2**3	1
*	Multiplicación	2*1	2
/	División	4/2	2
%	Modulo	9%8	2
+	Suma	2+4	3
-	Resta	4- 1	3

CUPL, operadores

- Operadores aritméticos.
  - Su finalidad es facilitar el modelado de un sistema.
  - Únicamente pueden ser usados como parte de los comandos de preprocesado **$REPEAT** y **$MACRO**.
  - Deben aparecer entre { }, sintaxis:

    { x **operador** y }

    {i+i}

    {2*i}

    {(i+1)%8}

Pin [2,3]=[a,b]

{a+b}

{a*i}

{(b+1)%8}

CUPL, operadores

- Prioridad de los operadores.
  - La prioridad determina en que orden serán evaluados dentro de una expresión.
  - Para la prioridad sólo se consideran los operadores de asignación y lógicos (no se consideran los operadores relacional ni los aritméticos).
  - Los operadores de igual prioridad serán evaluados según su orden de aparición en la ecuación de izquierda a derecha.
  - Los “( )” pueden emplearse para cambiar el orden de evaluación de la expresión, las sub-expresiones dentro de los paréntesis son evaluadas primero.

CUPL, estructura del lenguaje

- La estructura del lenguaje CUPL está compuesta por las siguientes secciones:
  - Encabezado
  - Declaraciones o definiciones
  - Cuerpo de programa (Cuerpo de Modelado)
    - Ecuaciones
    - Tabla de verdad
    - Diagramas de estado

CUPL, “Encabezado”

- Sección que permite definir:
  - El vínculo con un proyecto.
  - La documentación general del diseño.
  - Definir el dispositivo que se utilizará en el diseño.

CUPL, “Encabezado”

- Ejemplo:

```
Name     cont0059 ;
PartNo   00 ;
Date     02/07/2008 ;
Revision 01 ;
Designer Engineer ;
Company  Atmel ;
Assembly None ;
Location  ;
Device   g22v10 ;       /* Device   virtual ; */
```

CUPL, “Declaraciones”

- En la sección de DECLARACIONES se puede:
  - Asignar etiquetas a pines de entrada y salida.
  - Declarar señales tipo “Buried”.
  - Declarar “constantes”.
  - Declarar “conjuntos de señales” (**Field**).
  - Definir subexpresiones.

CUPL, “Declaraciones”

- Definición de pines. Permite definir:
  - Un identificador para una señal.
  - A cuál pin del dispositivo se le asignara cada señal del diseño.

    PIN #_pin = [!]identificador;

    PIN 2 = rst;

    PIN 3 = !oe;

    Pin [2,3,4,5] = [b, c, d, e];

    Pin [2,6,10] = [f,g,!h];

CUPL, “Declaraciones”

PIN 16 = q0;
PIN 17 = q1;
PIN 18 = q2;
PIN 19 = q3;
PIN [19..16] = [q3..q0];
PIN [16..19] = [q0..q3];
PIN [19..16] = [q3..0];
PIN [16..19] = [q0..3];

Pin [16..19] = ![q3..q0];
Pin [16..19] = [q3,!q2,q1,!q0];

CUPL, “Declaraciones”

- Definición de Nodos.
  - Usados para definir señales tipo buried.
  - Usados para dispositivos que contienen macroceldas sin pin asociado, macroceldas E-S-B o macroceldas S-B.

  - Existen dos comandos para definir nodos:

    PINNODE [nodo o pin] = [!]identificador;

    NODE = [!]identificador;

CUPL, “Declaraciones”

PINNODE 34 = bur_1;

PINNODE = !bur_2;

PINNODE [34,44,31] = ![bur_2,bur_1,bur_0];

PINNODE = [bur_2,bur_1,bur_0];

NODE = reg_bur1;

NODE = ![bur_2,bur_1,bur_0];

CUPL, “Declaraciones”

- Declaración de “constantes”.
  - La declaración de constantes es realizada mediante un comando de preprocesado.
  - **$define**, sintaxis:

    $define identificador constante

    $define cte1 'b'01011111
    $define s0 2E
    $define cte1 'D'255
    $define C 'b'01

CUPL, “Declaraciones”

- Declaración de “conjuntos de señales”.
- Lista de elementos que pueden ser tratados como una sola entidad. Usadas para:
  - Definir variables de más de un bit.
  - Permite referenciar a un grupo de variables por un mismo nombre.
  - Se usa el comando **FIELD**, sintaxis:

    Field nom = [nom_var1, nom_var2, …,nom_varn];

Field datos = [d0,d1,d2,d3];
Field datos = [d0..3];
Field control = [!cs,rd,wr];
Field control = ![cs,rd,wr];
Field direc1 = [ad6..3];

CUPL, “Declaraciones”

- Declaración de “conjuntos de señales” **FIELD**
  - Variables indexadas. DATOS.

    FIELD p=[q3..0]; **FIELD** p=[q3,q2,q1,q0];

    FIELD p=[q0..3]; **FIELD** p=[q0,q1,q2,q3];
  - Variables no indexadas. CONTROL.

    field ctr=[cs,rd,wr];
  - Tamaño máximo de una variable **FIELD** : 32 bits.
  - Se puede definir un **FIELD** dentro de otro **FIELD**

    Field y=[a,b,c];

    Field j=[m,y];

CUPL, “Declaraciones”

- Es posible mezclar variables indexadas con no indexadas en un **FIELD**.

  field ctr=[x,up1,up0,up2]; **field** ctr=[up0,x,up2,up1];

- No se pueden usar variables diferentes con el mismo índice dentro de un **FIELD**.

  field ctr=[up0,x0,up2,up1,x1]

  field ctr=[up0,up2,x1]

- Un **FIELD** puede contener identificadores y/o símbolos lógicos.

  field x = [a,b,'b'1,'b'0]

  field y = [m, 'd'5]; **field** y = [m, 'd'6];

CUPL, “Declaraciones”

- Operaciones entre **FIELDS**.
  - Operador de igualdad “ : ”.
  - El operador de igualdad permite comparar, bit a bit, un conjunto de variables y una constante.
  - Sintaxis:

    [var1, var2, ...varn]:constante;

    field_var:constante;

```
OK = [Q3,Q2,Q1,Q0]:'d'9;        OK = Q:'d'9;  //field Q =[Q3..0];
OK = [Q3,Q2,Q1,Q0]:[D3,D2,D1,D0];
CS = D:'h'3F;         //field D=[D3..0];
```

CUPL, “Declaraciones”

- Rangos.
- La operación de rango es similar a la operación de igualdad, excepto que la constante es ahora un rango de valores.
- Sintaxis:

  [var1, var2, ...varn]:[cons1..cons2];

  field_var:[cons1..cons2];

  OK = Q:'h' [9..B]; // **field** Q =[Q3..0];

  CS = [D0..3]:’h’[C..F];

  Wr = [a,b,c]:’h’[1,3,5,7];

  Var1 = ctr:'b'[010..101]; //**field** ctr=[a,b,c];

  Var2 = ctr:'b'1XX;

  Var3 = ctr:'b'1X0;

CUPL, “Declaraciones”

– Reglas para operaciones que involucran a un **FIELD**.

*Reglas para operaciones entre **FIELD´s**:*

1. En una operación lógica entre 2 **FIELD´s**, ambos deben tener el mismo número de elementos.

   [a3..0] # [b3..0]; [a3..0]$[x,y,z];

2. El resultado de una operación lógica que tiene como operandos a 2 **FIELD´s**, es un **FIELD** con el mismo número de elementos que los operandos.

   [a2..0] # [b2..0] = [a2#b2, a1#b1, a0#b0];

   [a2..0] & [x,y,z] = [a2&x, a1&y, a0&z];

3. ¿Qué pasa con una operación relacional entre 2 **FIELD´s**?

   Genera un “comportamiento erroneo”.

   [a2..0]:[b2..0]; /* Error lógico: !a0 & a1 & !a2; */

CUPL, “Declaraciones”

*Reglas para operaciones entre un **FIELD** y una variable de 1 bit:*

1. Operaciones relacionales de este tipo generan un “funcionamiento erroneo”.
2. El resultado de una operación lógica entre un **FIELD** y una variable de 1 bit genera un **FIELD** con el mismo número de elementos del **FIELD** involucrado en la operación.

   [a2..0] & m = [a2&m, a1&m, a0&m];

   [a0..2] & m = [a0&m, a1&m, a2&m];

CUPL, “Declaraciones”

*Reglas para operaciones entre **FIELD ´s** y números:*

1. 5 = [1,0,1], 8 = [1,0,0,0]
2. Si el número de elementos del **FIELD** es mayor que el número de bits que representan al número, éste es extendido con ceros a su izquierda.

   [b2..0]:'b'01; /* 'b'01 se extiende a 'b'001 */

   [b2..0]#'b'01 = [b2#0, b1#0, b0#1]
3. Si el número de elementos del **FIELD** es menor que el número de bits que representan al número, éste es cortado a su izquierda.

   [b2..0]:'b'1001; /* 'b'1001 se corta a 'b'001 */

   [b2..0]#'d'9 = [b2#0, b1#0, b0#1]

CUPL, “Declaraciones”

*Reglas de asignación de **FIELD´s**:*

1. El resultado de una operación lógica entre **FIELD´s** debe ser asignada a un **FIELD**.

   [c2..0] =[a2..0]&[b2..b0];

   [c2..0] =[a2..0]&[x,y,z];

2. Si una variable binaria es asignada a un **FIELD**, el valor de la variable será asignada a cada elemento del **FIELD**.

   [c2..0] = x

   [c1..0] = [a2..0]:'b'010;

CUPL, “Declaraciones”

- Ejemplos:

  a=[a2..0]; b=[b0..2]; z=[z2..0]

Asignación por posición

1. z=a; :

   z=b; :

2. z = a&b;

   a=[a2..0]; b=[b2..0]; z=[z2..0]:

   a=[a2..0]; b=[b2..0]; z=[z0..2]:

   a=[a0..2]; b=[b2..0]; z=[z2..0]:

   a=[a2..0]; b=[b0..2]; z=[z2..0]:

CUPL, “Declaraciones”

a=[a2..0]; b=[b2..0]; c=[c2..0]; d=[d2..0]; z=[z2..0], sal0 y sal1 son variables de 1 bit:

3. sal0 = a:b; : sal0 => !a0 & a1 & !a2;

4. sal1 = a:b:c; :
   sal1 = (a:b):c; :

5. z = a&b:c&d; :
   z = (a&b):(c&d); :
   sal1 = a&b:c&d; :

CUPL, “Declaraciones”

6. z = ni;

   z=[z2..0]; ni=[m,n,p]:

   z=[z0..2]; ni=[m,n,p]:

Asignación por posición

7. z = ni & a;

   a=[a2..0]; z=[z2..0]; ni=[m,n,p]:

   a=[a2..0]; z=[z0..2]; ni=[m,n,p]:

   a=[a0..2]; z=[z0..2]; ni=[m,n,p]:

   a=[a0..2]; z=[z2..0]; ni=[m,n,p]:

8. z = a & m;

   a=[a2..0]; z=[z2..0]:

   a=[a0..2]; z=[z2..0]:

   a=[a0..2]; z=[z0..2]:

   a=[a2..0]; z=[z0..2]:

CUPL, “Declaraciones”

9. z = 'b'100;

   z=[z2..0]:
   z=[z0..2]:

Asignación por peso (índice)

z=[z2..0]: z0 = 0; z1 = 0; z2 = 1;
z=[z0..2]: z0 = 0; z1 = 0; z2 = 1;

10. z = a & 'b'100;

    a=[a2..0]; z=[z2..0]:
    a=[a0..2]; z=[z2..0]:
    a=[a0..2]; z=[z0..2]:
    a=[a2..0]; z=[z0..2]:

Operación lógica y asignación por peso, respecto a variable de salida “z”;

a=[a2..0]; z=[z2..0]: z0 = 0; z1 = 0; z2 = a2;
a=[a0..2]; z=[z2..0]: z0 = 0; z1 = 0; z2 = a0;
a=[a0..2]; z=[z0..2]: z0 = 0; z1 = 0; z2 = a2;
a=[a2..0]; z=[z0..2]: z0 = 0; z1 = 0; z2 = a0;

11. z = a:'b'100;

    a=[a2..0]; z=[z0..2]:
    a=[a0..2]; z=[z2..0]:
    a=[a0..2]; z=[z0..2]:
    a=[a2..0]; z=[z2..0]:

Evaluación por peso

En todos los casos:
z0 = z1 = z2 => a2 & !a1 & !a0;

CUPL, “Declaraciones”

- Ejemplos:

1. [a,b] & [c,d];
2. [a,b]:[c,d];
3. [a,b]:[c,d]:[e,f];
4. [a,b,c] & 'd'9;
5. [a,b,c]:'h'5;
6. [a,b,c] # 3;
7. [a,b] & [c,d,e] ;
8. [a,b] & c;
9. [a,b]:[c,d]:e;
10. [a,b] & 'o'5;
11. [a,b,c] = [c,d]:'b'01;
12. [a,b] = [c,d] & [e,f];
13. [a,b] = e;
14. [a,b] = 1;
15. Eql = [a,b,c]:11;
16. [a,b] ='h'5;

CUPL, “Declaraciones”

- Sub-expresiones.
  - Una sub-expresión es la asignación de una expresión lógica o relacional a una “variable intermedia”.
  - Una “variables intermedia” es aquella que no tienen asociado un pin del dispositivo.
  - Tienen por objetivos:
    - Mejorar la legibilidad del modelado.
    - Facilitar la escritura de las ecuaciones lógicas.
  - Se puede añadir variables intermedias en cualquier parte del programa.

CUPL, “Declaraciones”

- Ejemplos de declaración de variables intermedias.

  PIN 2 = x;

  PIN 3 = y;

CUPL, “Cuerpo del programa”

- Esta sección contiene el modelado o descripción del diseño y está formada por 1 o más bloques, los cuales pueden ser:
  - Bloque de Ecuaciones.
  - Bloque de Tabla de Verdad.
  - Bloque de Maquina de Estado.

  - Elementos de diseño abstracto de hardware.

CUPL, “Cuerpo del programa”

- ***Bloque de ecuaciones.***
- Formado por expresiones lógicas, sintaxis:

  [!]var[.ext] = exp;

  Z1 = s1 $ s2;
  Z2 = !s1 & s2 & s0 # s1 # s2 # Z3;
  !Z3 = (!s1 # s2) & (s0 # s1 # s2) & (a # b);
  q0.d = !q0;

- El bloque de ecuaciones define el tipo de señal.

CUPL, “Cuerpo del programa”

```
q0 = !q0;
q1 = (q1&!q0)#(!q1&q0);
q2 = (q2&!q0)#(!q1&q2)#(!q2&q1&q0);
```

```
q2 = (q2&!q0)#(!q1&q2)#(!q2&q1&q0);
q0 = !q0;
q1 = (q1&!q0)#(!q1&q0);
```

CUPL, “Cuerpo del programa”

- Polaridad de la señal.
- Una señal puede ser definida para tener:
  - Polaridad negativa (Activa en bajo).
  - Polaridad positiva (Activa en alto).
- La polaridad de la señal depende de dos factores:
  - La definición de la señal en el bloque de declaraciones.
  - La definición de la señal en el bloque de ecuaciones.

		Bloque de ecuaciones	
		Señal	/Señal
Bloque de declaraciones	Señal	Polaridad (+)	Polaridad (-)
	/Señal	Polaridad (-)	Polaridad (+)

CUPL, “Cuerpo del programa”

PIN [2..3] = [e0..1];
PIN [17,18,19,20] = [s0,!s1,s2,!s3];

s0 = e0&e1; s1 = e0&e1;
!s2 = e0&e1; !s3 = e0&e1;

PIN [2,3,4,5] = [e0,!e1,e2,!e3];
PIN [17,18] = [s0,s1];

s0 = e0#e1;
s1 = !e2&!e3;

CUPL, “Cuerpo del programa”

- Uso de subexpresiones. Facilitan la escritura de las ecuaciones.

PIN [2..5] = [e0..3];
PIN [17,18,19,20] = [s0,s1,s2,s3];

varint1 = e1#e2;

1. s0 = e0 & varint1 & e3; -> ¿?
2. s1 = e0 # !varint1 # e3; -> ¿?

1. Evalúa la sub-expresión; 2. Substituye;
3. Evalúa la expresión

CUPL, “Cuerpo del programa”

- Sentencia **APPEND.**
- Normalmente, sólo una expresión puede ser asignada a una variable.
- La sentencia **APPEND** permite asignar múltiples expresiones a una misma variable; sintaxis:

  APPEND[!]var[.ext] = exp;

```
APPEND   Y = A0 & A1;
APPEND   Y = B0 & B1;
APPEND   Y = C0 & C1;

Y = A0 & A1 # B0 & B1 # C0 & C1;
```

```
APPEND Y = A0 & A1;
APPEND !Y = B0 & B1;
APPEND Y = C0 & C1;

!Y = A0 & A1 # B0 & B1 # C0 & C1;
```

CUPL, “Cuerpo del programa”

- Ejemplos con **APPEND**: 1. Substituye - une; 2 Evalúa.

1. **append** Y=a0&b0;
   append Y=a1&b1;
   append Y=c0#c1;

2. Y=a0&b0;
   append Y=a1&b1;
   append Y=c0#c1;

3. **append** Y=a0&b0;
   append Y=a1&b1;
   append Y=c0#c1&c2;

4. Y=a0&b0;
   append Y=c0$c1;

CUPL, “Cuerpo del programa”

- El **operador de igualdad “ : ”** también puede ser usado para facilitar la escritura de algunas expresiones

```
s = [a2,a1,a0]:&    ->    s = a2 & a1 & a0;
s = [a2..a0]:#      ->    s = a2 # a1 # a0;
s = [a2,a1,a0]:$    ->    s = a2 $ a1 $ a0;
```

```
Field a=[a2..0];
s = a:#;      ->    s = a2 # a1 # a0;
s = a:&;      ->    s = a2 & a1 & a0;
s = a:$;      ->    s = a2 $ a1 $ a0;
```

CUPL, “Cuerpo del programa”

- La operación de rango, también es de gran utilidad en la simplificación de expresiones.

Field x=[a,b,c];

Sel = x:'o'[5..7];

Sel = x:'b'0x01;

Sel = x:'o'[2,4,6];

CUPL, “Cuerpo del programa”

- También se puede usar el comando de preprocesado **$REPEAT**.

```
$repeat índice=[rango]
        Expresiones
$repend
```

```
field a=[a1..0];
field b=[b1..0];
field s=[s1..0];

$repeat i=[0..1]
        s{i} = a{i} & b{i};
$repend
```

CUPL, “Cuerpo del programa”

- El comando de preprocesado **$MACRO** puede ser empleado en la escritura de ecuaciones; sintaxis:

```
$macro id_macro var_i1,var_i2,…,var_in
  expresiones                          /* declaración de la macro */
$mend

id_macro (var1,var2,…,varn)            /* llamado de la macro */
```

```
PIN [2..4]=[a0..2];
PIN 14 = s0;

$macro and e0 e1 e2 sal;
  sal = e0 & e1 & e2;
$mend

and(a0,a1,a2,s0);
```

CUPL, “Cuerpo del programa”

PIN [2..4]=[a0..2];
PIN [5..7]=[b0..2];
PIN [14..16] = [s0..2];

$macro and e0 e1 e2 sal;
  sal = e0 & e1 & e2;
$mend

1. and(a0,a1,a2,!s0);
   and(!b0,b1,b2,s1);
   and(!b0,a1,!a2,s2);

1. Evalúa; 2. Substituye.

2. c=a1&b1;
   and(a0,b0,c,s2);

3. c=a1#b1;
   and(a0,b0,c,s2);

4. c=a1$b1;
   and(a0,b0,c,s2);

5. and(a0,a1,a2,s0);
   and(b0,b1,s0,s1);

6. and(a0,a1,a2,c);
   and(b0,b1,c,s1);

CUPL, “Cuerpo del programa”

1.
```
PIN 2=a0;
PIN 5=b0;
PIN [8,9]=[f,g];
PIN 22=c;
PIN 14 = s0;

$macro and e0 e1 e2 sal;
   sal = e0 & e1 & e2;
$mend

and(a0,b0,c,s2);

TABLA DE VERDAD {
      implementa
      c=f#g;
}
```

2.
```
PIN 2=a0;
PIN 5=b0;
PIN [8,9]=[f,g];
PIN 14 = s0;

$macro and e0 e1 e2 sal;
   sal = e0 & e1 & e2;
$mend

and(a0,b0,c,s2);

TABLA DE VERDAD{
      implementa
      c=f#g;
}
```

CUPL, “Declaraciones”

- Extensiones.
  - Permiten acceder a funciones específicas asociadas a una macrocelda.
  - Extensiones disponibles en el CUPL:

Extension	Description
.AP	Asynchronous preset of flip-flop
.AR	Asynchronous reset of flip-flop
.CE	CE input of enabled D-CE type flip-flop
.CK	Programmable clock of flip-flop
.CKMUX	Clock multiplexer selection
.D	D input of D-type flip-flop
.DFB	D registered feedback path selection
.DQ	Q output of D-type flip-flop
.INT	Internal feedback path for registered macrocell
.IO	Pin feedback path selection
.J	J input of JK-type output flip-flop

Extension	Description
.K	K input of JK-type output flip-flop
.L	D input of transparent latch
.LE	Programmable latch enable
.LQ	Q output of transparent input latch
.OE	Programmable output enable
.R	R input of SR-type output flip-flop
.S	S input of SR-type output flip-flop
.SP	Synchronous preset of flip-flop
.SR	Synchronous reset of flip-flop
.T	T input of toggle output flip-flop
.TFB	T registered feedback path selection

CUPL, “Declaraciones”

– En un bloque de ecuaciones si se omiten extensiones en las variables de salida, se trata de un diseño combinacional.

```
q0 = !q0;
q1 = q1&!q0#!q1&q0;
q2 = q2&!q0#!q1&q2#!q2&q1&q0;
```

CUPL, “Declaraciones”

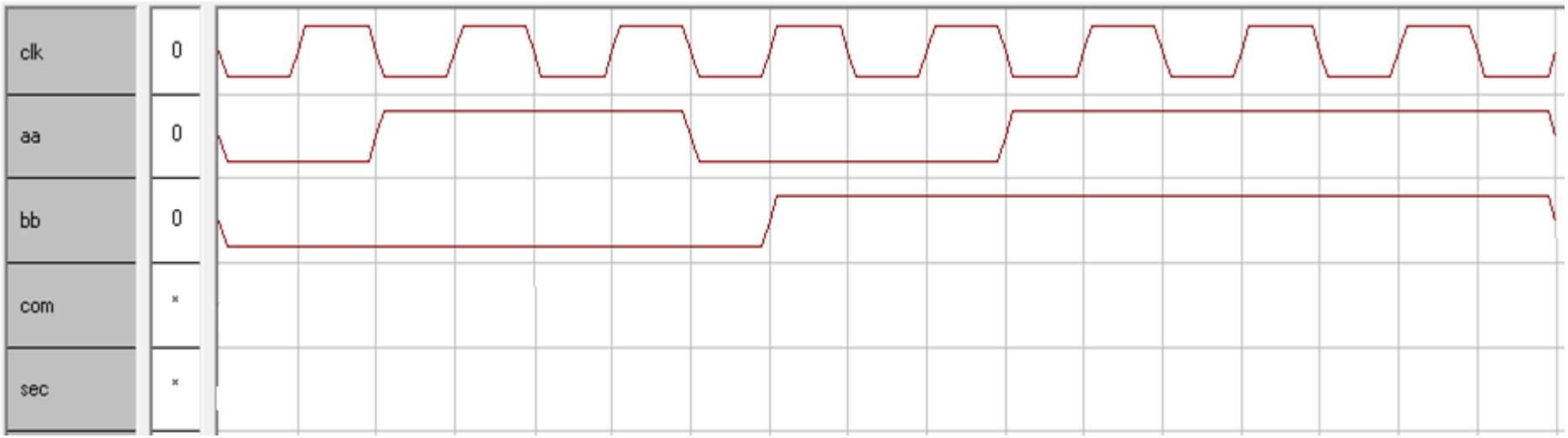
– Para modelar un sistema secuencial se debe hacer uso de algunas extensiones sobre las variables de salida.

```
q0.d = !q0;
q1.d = q1&!q0#!q1&q0;
q2.d = q2&!q0#!q1&q2#!q2&q1&q0;
```

CUPL, “Declaraciones”

com = aa & bb;

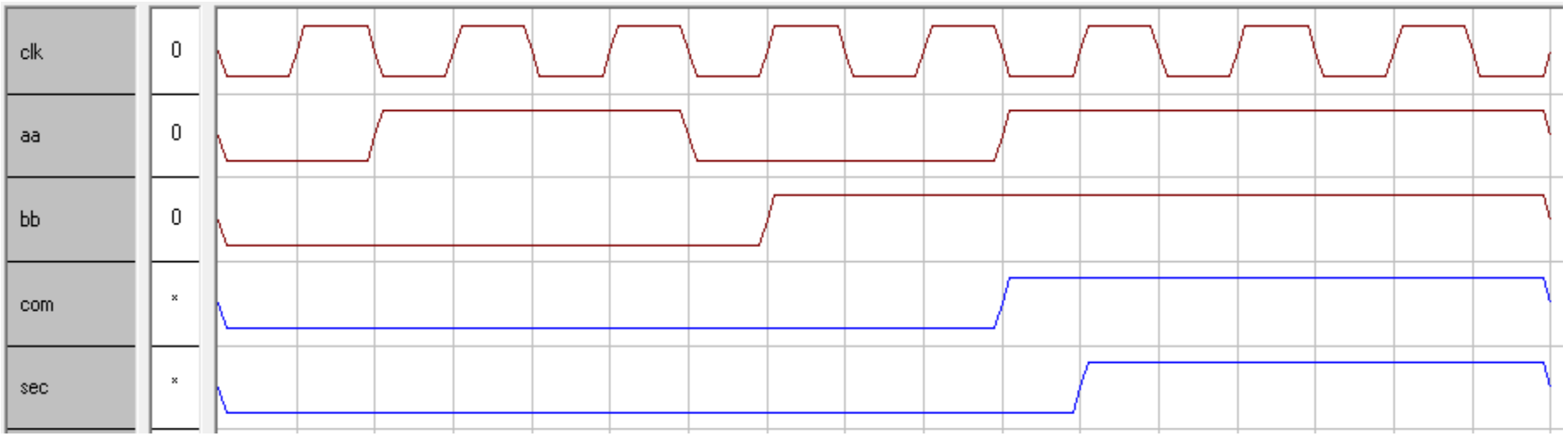
sec.d = aa & bb;



CUPL, “Declaraciones”

com = aa & bb;

sec.d = aa & bb;



CUPL, “Cuerpo del programa”

q0.J = a + b * c + d * q1

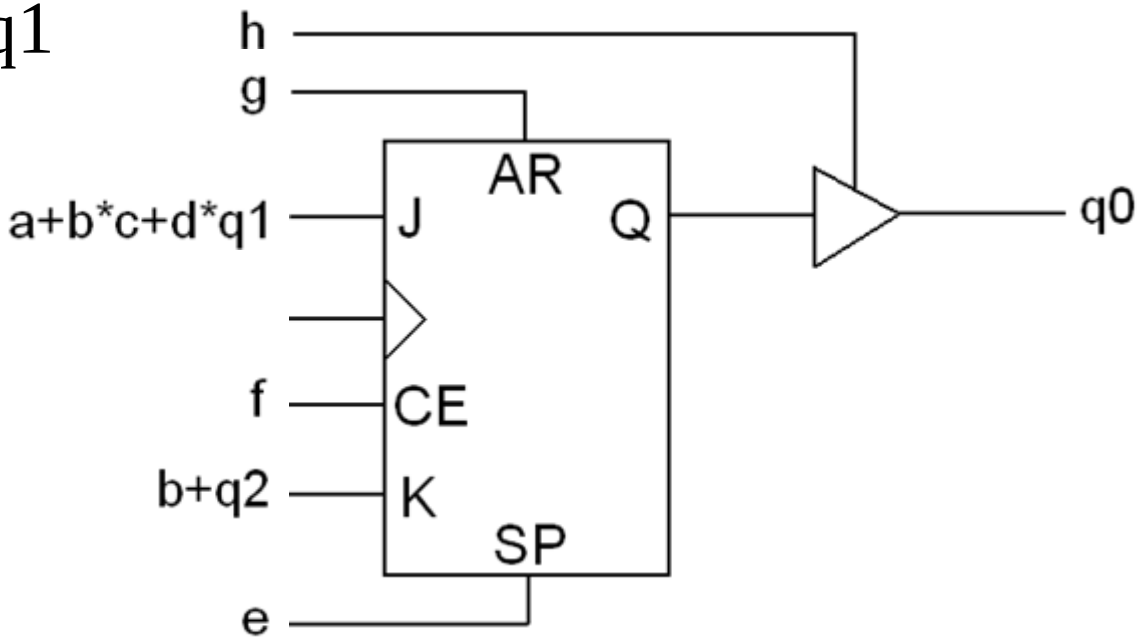
q0.K = b + q2

q0.CE = f

q0. SP = e

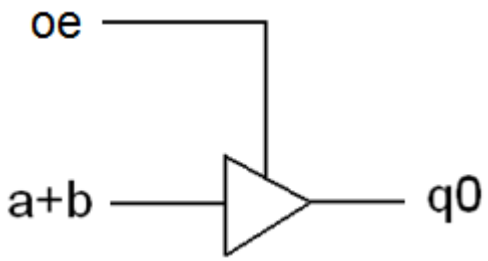
q0.AR = g

q0.OE = h



q0 = a + b

q0.oe = oe



CUPL, “Cuerpo del programa”

- ***Bloque de tablas de verdad.***
- CUPL provee el comando **TABLE**, para crear tablas de verdad, la cuales relacionan variables de entrada con variables de salida.
- Sintaxis:

```
TABLE var_in => var_out {
        input_1 => output_1;
        input_2 => output_2;
            ...
        input_n => output_n;
}
```

CUPL, “Cuerpo del programa”

- Ejemplos

```
FIELD in = [a,b];
FIELD comp = [x,y,z,w];

table in => comp {
    'b'00 =>  'b'0010;
    'b'01 =>  'b'1001;
    'b'10 =>  'b'1001;
    'b'11 =>  'b'1100;
}
```

CUPL, “Cuerpo del programa”

1. **table** f,g => h {
   0 => 0;
   1 => 1;
   2 => 1;
   3 => 1;
   }
   h => f # g;

2. **table** a0,a2 => h {
   0 => 0;
   1 => 1;
   2 => 1;
   3 => 1;
   }
   h => !a2;

3. **table** a0,a1 => h {
   0 => 0;
   1 => 1;
   2 => 1;
   3 => 1;
   }
   h => a0 # a1;

4. **table** a0,b2 => h {
   0 => 0;
   1 => 1;
   2 => 1;
   3 => 1;
   }
   h => !b2;

5. **table** a0,b1 => h {
   0 => 0;
   1 => 1;
   2 => 1;
   3 => 1;
   }
   h => a0 # b1;

6. **table** a0,b0=> h {
   0 => 0;
   1 => 1;
   2 => 1;
   3 => 1;
   }
   h => a0 & b0 # !a0 & !b0;

CUPL, “Cuerpo del programa”

1. **Field** d=[d1,d0]

```
table d => h {
        0 => 0;
        1 => 1;
        2 => 0;
        3 => 0;
 }

h = !d1 & d0;
```

2. **Field** d=[a,b]

```
table d => h {
        0 => 0;
        1 => 1;
        2 => 0;
        3 => 0;
 }

h =  !a & b;
```

3. **Field** d=[q0,a]

```
table d => h {
        0 => 0;
        1 => 1;
        2 => 0;
        3 => 0;
  }

h =  !q0 & a;
```

CUPL, “Cuerpo del programa”

- Se pueden agregar variables intermedias o resultados de una ecuación a la tabla de verdad. Uso de las condiciones no importa.

```
PIN [2,3] = [a,b];
PIN 4 = x;
PIN 5 = y;
PIN [14..17] = [s0..3];

FIELD in = [a,b];
FIELD s = [s0..3];
var=x&y;
FIELD in_f = [in,var];
```

```
table in_f => s {
        'b'001 => 'b'0001;
        'b'011 => 'b'0010;
        'b'101 => 'b'0100;
        'b'111 => 'b'1000;
        'b'XX0 => 'b'0000;
}
```

No permite uso de alta impedancia: 'b'z

CONDICIONES NO IMPORTA. Solo son validas para valores de entrada no para vectores de salida.

Cuando se usan variables intermedias, igual que con ecuaciones, primero evalúa la sub-expresión, luego la substituye.

CUPL, “Cuerpo del programa”

- Uso de rangos.

```
PIN [2..5] = [ad0..3];
PIN [16..19] = [ADC,DAC,DISP,TEC];

FIELD ad = [ad3..0];
FIELD peri = [DAC,ADC,DISP,TEC];

table ad => peri {
        [0..2] => 8;
        [3..8] => 4;
        [9..d] => 2;
        E => 1;
}
```

F(A,B,C) = Σ (0,2,4,6)

```
Field d = [A,B,C];

table d => q {
            [0,2,4,6]=>1;
        //  [1,3,5,7]=>0;
}
```

CUPL, “Cuerpo del programa”

- Diseños secuenciales con tablas de verdad.

```
PIN 1 = clk;
PIN [14..15] = [q1..0]

Field q = [q1..0];

table q => q.d {
        0 => 1;
        1 => 2;
        2 => 3;
        3 => 0;
}
```

CUPL, “Cuerpo del programa”

- ***Bloque de maquinas de estado.***
- Una maquina de estado representa a un sistema secuencial.
- El lenguaje de maquinas de estado, utilizado en CUPL, permite al usuario abstraerse del hardware;
- Es decir, este tipo de modelado permite al usuario describir el comportamiento del circuito en lugar de su estructura.

CUPL, “Cuerpo del programa”

- La sintaxis, usada en el CUPL para modelar una maquina de estados, es la siguiente:

```
SEQUENCE field_var {
        PRESENT state_n1
                declaración_transición;
        PRESENT state_n3
                declaración_transición;
                ...
        PRESENT state_n4
                declaración_transición;
}
```

- Se pueden declarar múltiples maquinas de estado:
  - Independientes o dependientes entre sí.
  - Interactuar con otro tipo de diseño (ecuaciones, tabla de verdad).

CUPL, “Cuerpo del programa”

- Formato de los bits de estado:
  - Para variables indexadas.

    [q2,q1,q0] [q0,q1,q2] [q1,q0,q2]

  - Para variables no indexadas

    [a,b,c]
  - Para variables mezcladas

    [a,q1,q0]
  - Comportamiento erroneo

    [q0,q2,a] [a,q0,p0]

CUPL, “Cuerpo del programa”

- Estructura de un estado.

  present codigo_edo

  declaracion_transicion; [sentencia **out**];

  [sentencia **default**]

- La declaración de transición indica cual es el estado siguiente
- La sentencia **OUT** permite activar a una señal dentro de un estado.
- La sentencia **DEFAULT** permite especificar el estado siguiente en condiciones indeterminadas.
- Dentro de un estado no se pueden declarar expresiones, macros o funciones.

CUPL, “Cuerpo del programa”

- Declaraciones de transición.
  - Función: determinar el siguiente estado.
  - Existen 2 tipos de declaraciones de transición:
    - Incondicional

      next estado_sig**;**
    - Condicional

      If condición **next** estado_sig**;**

      – Condición:

      » una variable o un pin: x

      » con el operador “:” ad:9

      » con rango a:[0..9]

      » Una expresión lógica x&y&z

CUPL, “Cuerpo del programa”

- **next**

next edo_x;

next edo_y;

Donde

$define edo_x 'b'00

$define edo_y 'b'01

next 0;

next 1;

// Ejemplo

pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
  present 0
    next 1;
  present 1
    next 2;
  present 2
    next 3;
  present 3
    next 0;
}

CUPL, “Cuerpo del programa”

- **<u>If</u>**

i) **if** x **next** estado_x;

ii) **if** x **next** estado_x;
default next estado_y;

iii) **if** x **next** estado_x;
if z **next** estado_y;

iv) **if** x **next** estado_x;
if z **next** estado_y;
default next estado_y;

v) **if** x **next** estado_x;
default if z **next** estado_y;
default next estado_y;

CUPL, “Cuerpo del programa”

```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
    present 1
        if x next 2;
    present 2
        if x next 3;
    present 3
        if x next 0;
}
```

```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
        default next 0;
    present 1
        if x next 2;
        default next 0;
    present 2
        if x next 3;
        default next 0;
    present 3
        if x next 0;
        default next 0;
}
```

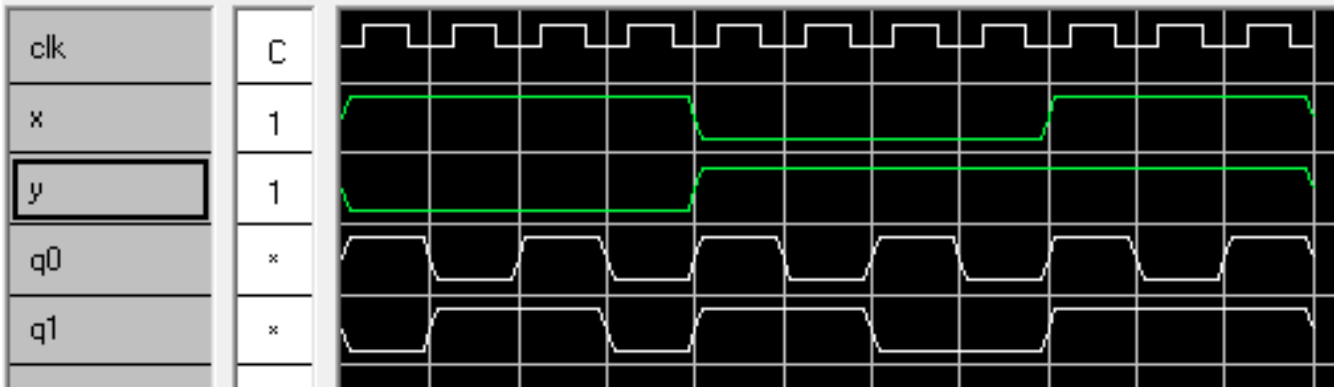
```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
        default next 0;
    present 1
        if x next 2;
        default next 1;
    present 2
        if x next 3;
        default next 2;
    present 3
        if x next 0;
        default next 3;
}
```

CUPL, “Cuerpo del programa”

```
pin 1 = clk;
pin [2,3] = [x,y];
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
        if y next 3;
    present 1
        if x next 2;
        if y next 0;
    present 2
        if x next 3;
        if y next 1;
present 3
        if x next 0;
        if y next 2;
}
```



CUPL, “Cuerpo del programa”

POSIBLES SOLUCIONES:

```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
        if !x next 3;
    present 1
        if x next 2;
        if !x next 0;
    present 2
        if x next 3;
        if !x next 1;
present 3
        if x next 0;
        if !x next 2;
}
```

```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
        default next 3;
    present 1
        if x next 2;
        default next 0;
    present 2
        if x next 3;
        default next 1;
    present 3
        if x next 0;
        default next 2;
}
```

```
pin 1 = clk;
pin [2,3] = [x,y];
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
        if (!x&y) next 3;
    present 1
        if x next 2;
        if (!x&y) next 0;
    present 2
        if x next 3;
        if (!x&y) next 1;
    present 3
        if x next 0;
        if (!x&y) next 2;
}
```

CUPL, “Cuerpo del programa”

```
pin 1 = clk;
pin [2,3] = [x,y];
pin [14..15] = [q0..1];

sequence q1,q0{
        present 0
                if x next 1;
                if y next 0;
            //  default next 0;
        present 1
                if x next 2;
                if y next 1;
            //  default next 0;
        present 2
                if x next 3;
                if y next 2;
            //  default next 0;
        present 3
                if x next 0;
                if y next 3;
            //  default next 0;
}
```

¿Qué funciones implementa?

¿Cuáles SOP´s genera?

CUPL, “Cuerpo del programa”

```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
        present 0
                if x next 1;
                next 1;
        present 1
                 if x next 2;
                 next 1;
        present 2
                if x next 3;
                next 1;
        present 3
                if x next 0;
                next 1;
}
```

¿Cuáles SOP´s genera?

CUPL, “Cuerpo del programa”

- Sentencia **OUT**. Permite “activar” una señal dentro de una maquina de estados.

OUT A equivale a **OUT** !A;

- La polaridad de la señal de salida puede ser modificada solo en su declaración:

PIN 20 = !A;

No admite expresión: **OUT** A=B&C;

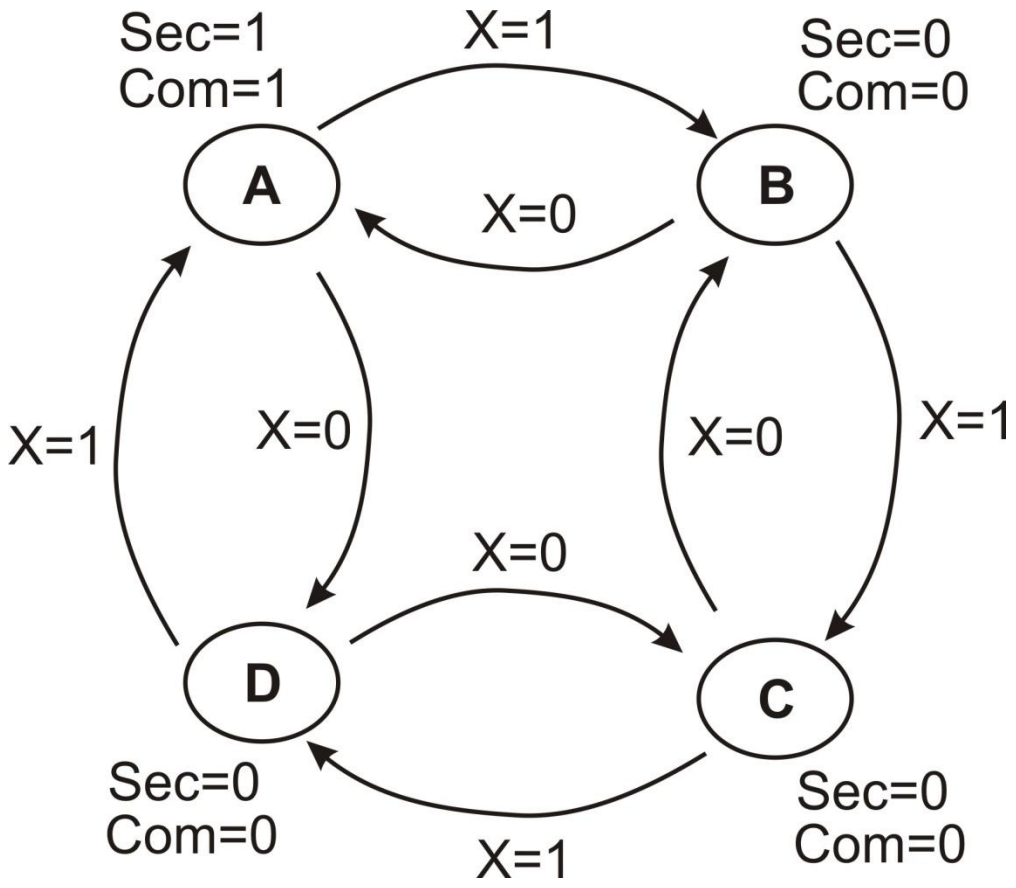
CUPL, “Cuerpo del programa”

- Estructura de un estado:

present edo_x
If cond1 **next** edo_y; **out** var1;
If cond2 **next** edo_z;
default next edo_w; **out** var2;
present edo_y
If cond1 **next** edo_y; **out** var1;
next edo_w;
present edo_z
next edo_x;

CUPL, “Cuerpo del programa”

- Ejemplo.

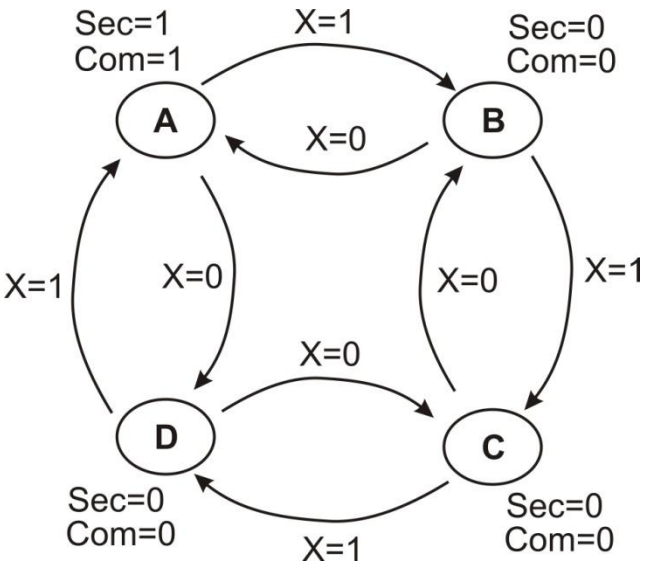


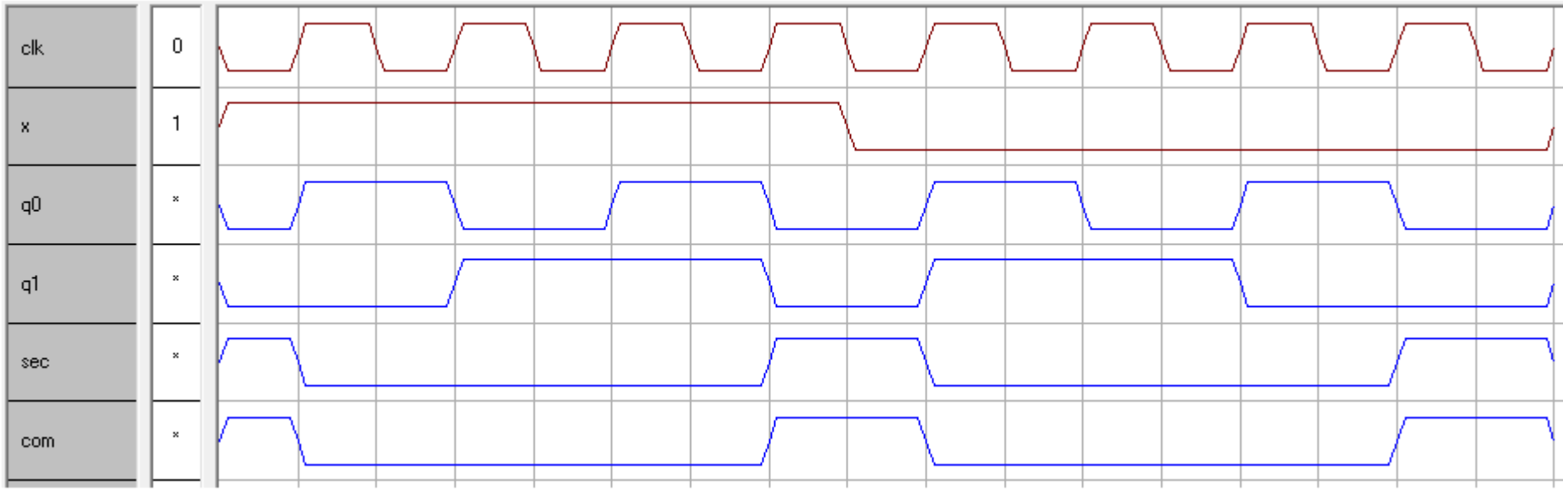
- 1er caso.

```
PIN 1 = clk;
PIN 2 = x;
PIN [14..15] = [q0..q1];
PIN 18 = com;
PIN 19 = sec;

field mq=[q0..q1];
// $define eA 'b'00
// $define eB 'b'01
// $define eC 'b'10
// $define eD 'b'11
```

```
SEQUENCE mq{
        present 0
                If x next 1; out sec out com;
                If!x next 3;
        present 1
                If x next 2;
                If!x next 0;
        present 2
                If x next 3;
                If!x next 1;
        present 3
                If x next 0;
                If!x next 2;
}
```



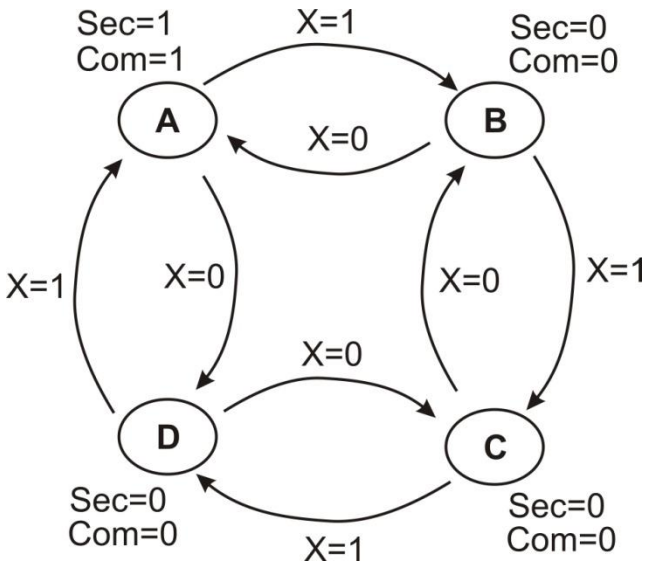


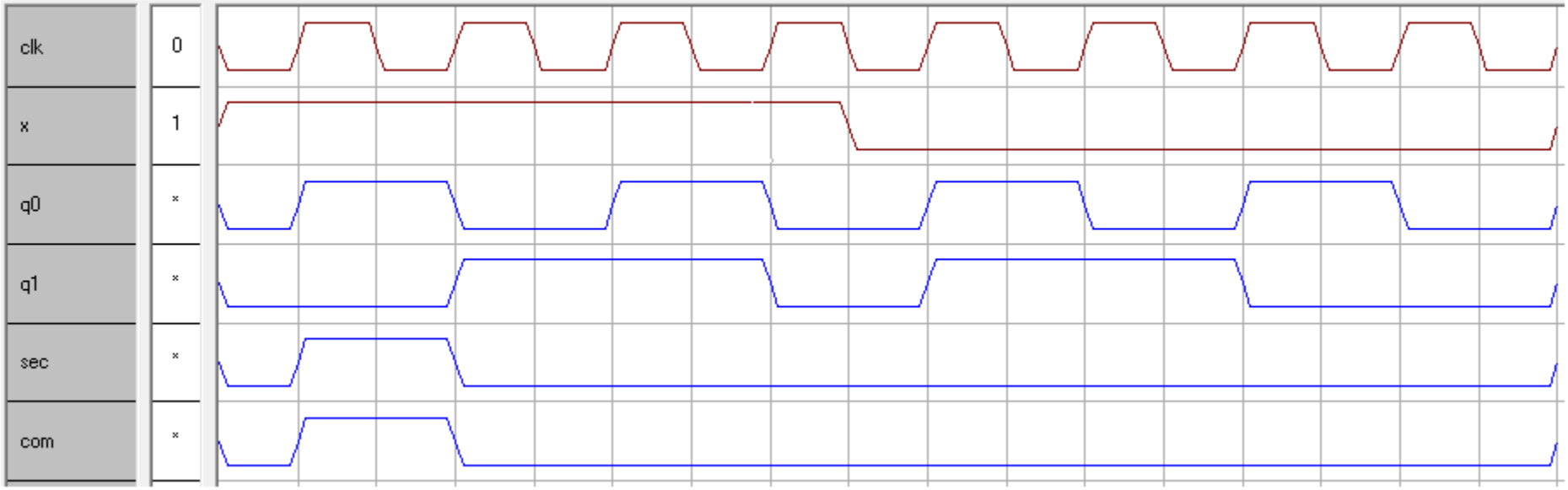
- 2o caso.

```
PIN 1 = clk;
PIN 2 = x;
PIN [14..15] = [q0..q1];
PIN 18 = com;
PIN 19 = sec;

field mq=[q0..q1];
// $define eA 'b'00
// $define eB 'b'01
// $define eC 'b'10
// $define eD 'b'11
```

```
SEQUENCE mq{
        present 0
                If x next 1  out sec out com;
                If!x next 3;
        present 1
                If x next 2;
                If!x next 0;
        present 2
                If x next 3;
                If!x next 1;
        present 3
                If x next 0;
                If!x next 2;
}
```



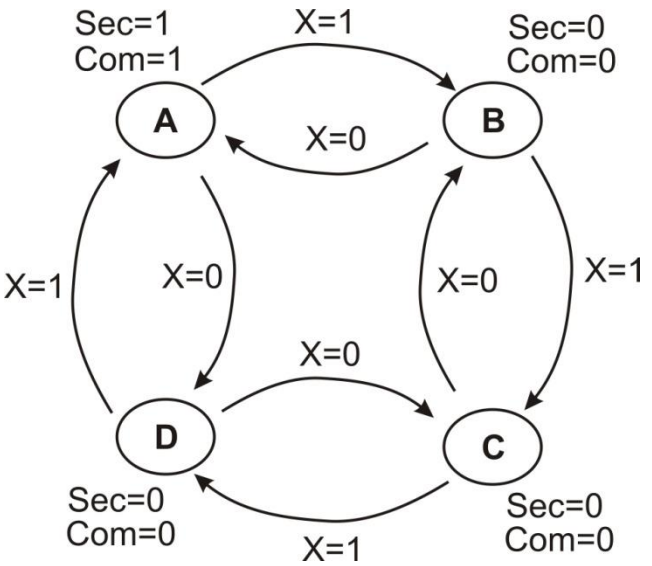


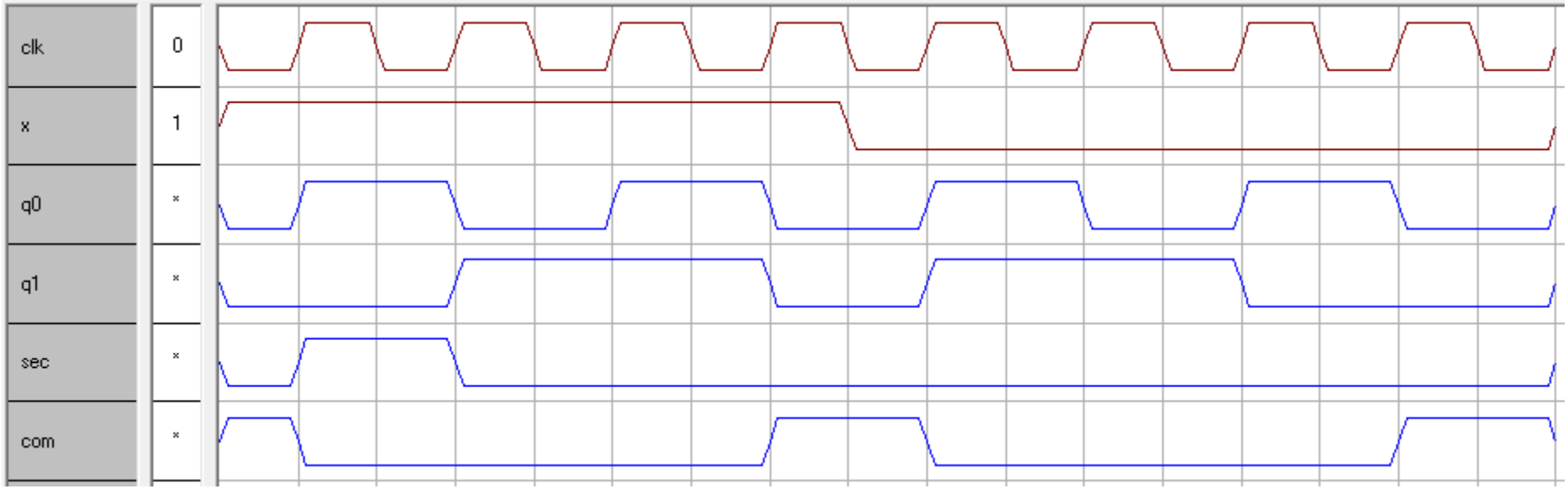
- 3er caso.

```
PIN 1 = clk;
PIN 2 = x;
PIN [14..15] = [q0..q1];
PIN 18 = com;
PIN 19 = sec;

field mq=[q0..q1];
// $define eA 'b'00
// $define eB 'b'01
// $define eC 'b'10
// $define eD 'b'11
```

```
SEQUENCE mq{
        present 0
                If x next 1  out sec; out com;
                If!x next 3;
        present 1
                If x next 2;
                If!x next 0;
        present 2
                If x next 3;
                If!x next 1;
        present 3
                If x next 0;
                If!x next 2;
}
```



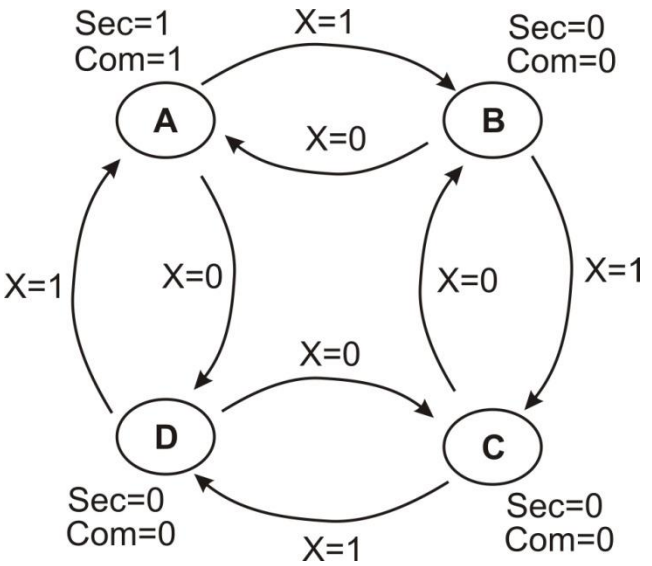


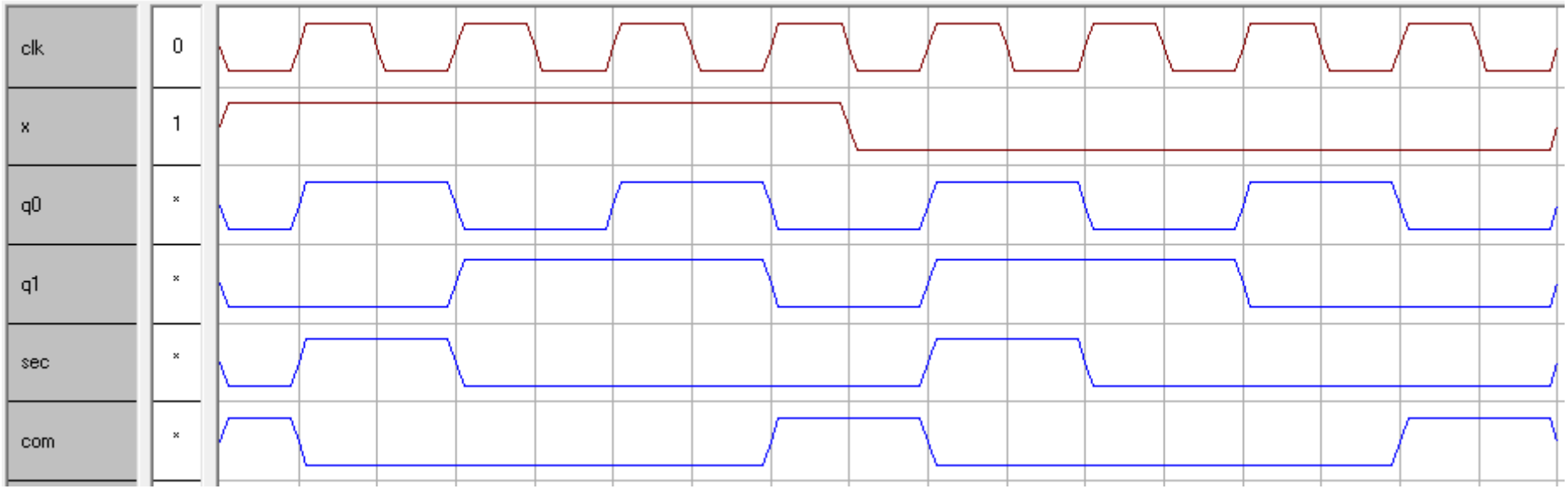
- 4o caso.

```
PIN 1 = clk;
PIN 2 = x;
PIN [14..15] = [q0..q1];
PIN 18 = com;
PIN 19 = sec;

field mq=[q0..q1];
// $define eA 'b'00
// $define eB 'b'01
// $define eC 'b'10
// $define eD 'b'11
```

```
SEQUENCE mq{
    present 0
        If x next 1; out sec.d out com;
        If!x next 3;
    present 1
        If x next 2;
        If!x next 0;
    present 2
        If x next 3;
        If!x next 1;
    present 3
        If x next 0;
        If!x next 2;
}
```





CUPL, “Cuerpo del programa”

- Resumen de la sentencia **OUT**:

Present 0

If x **next** 1; **out** sec **out** com;
If x **next** 1 **out** sec **out** com;
If x **next** 1 **out** sec; **out** com;
If x **next** 1; **out** sec.d **out** com;
If x **next** 1 **out** sec.d; **out** com;

CUPL, “Cuerpo del programa”

- Lo mismo aplica a **next** y **default**

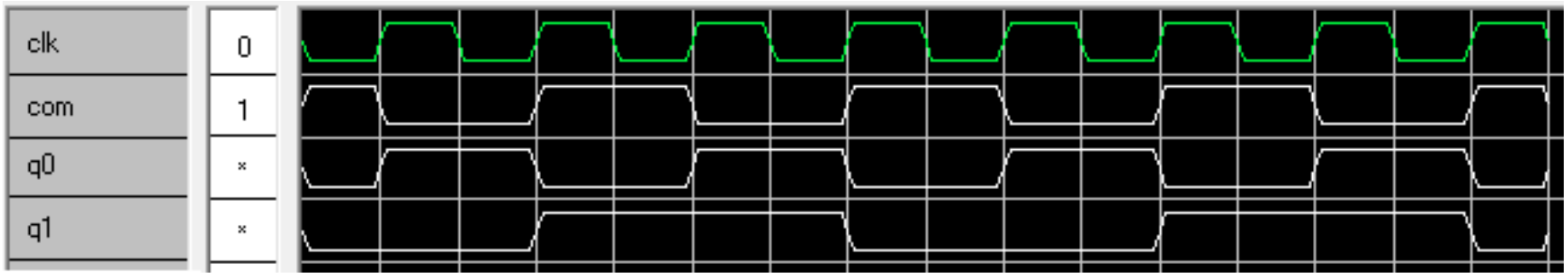
```
SEQUENCE q1,q0{
        present 0
                next 1;
        present 1
                if x next 2;
                default next 0 out com;
        present 2
                if x next 3;
                next 0; out sec;
        present 3
                next 0;
}
```

¿Cuáles son las expresiones de “com” y “sec”’?

CUPL, “Cuerpo del programa”

```
SEQUENCE q1,q0{
        present 0
                next 1; out com;
        present 1
                next 2;
        present 2
                next 3; out com;
        present 3
                next 0;
}
```

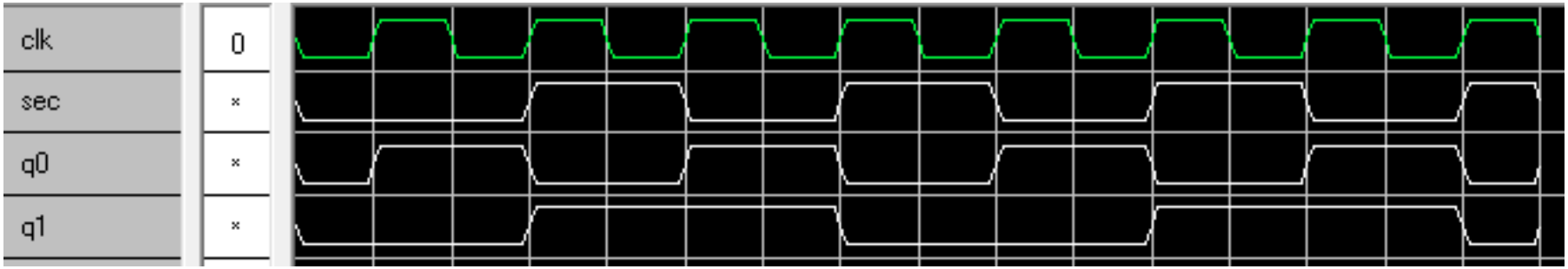
com => !q1 & !q0
 # q1 & !q0



CUPL, “Cuerpo del programa”

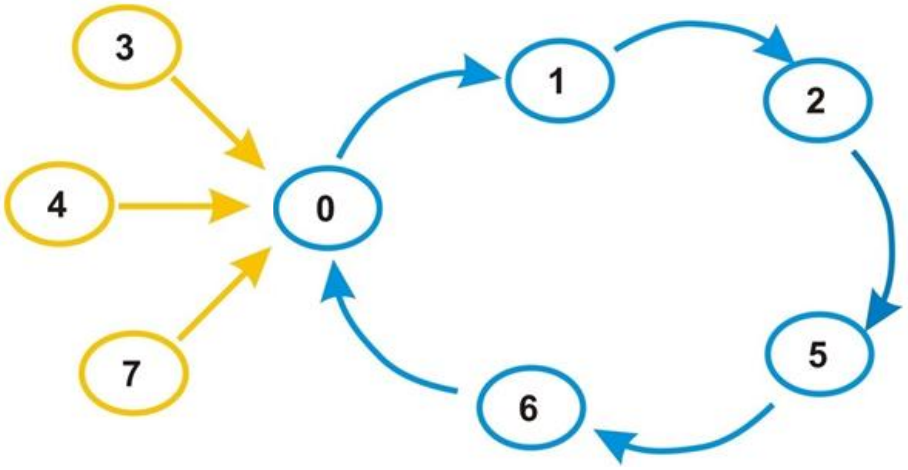
```
SEQUENCE q1,q0{
        present 0
                next 1;
         present 1
                next 2   out sec;
         present 2
                next 3
         present 3
                next 0  out sec;
}
```

sec.d => !q1 & q0
 # q1 & q0



CUPL, “Cuerpo del programa”

- Estados indeterminados o no contemplados en una maquina de estados:
  - No existe una forma de indicar cuál será el estado siguiente a un estado de este tipo.
  - CUPL modela el diseño considerando que el estado siguiente de un estado no definido será aquel con valor ‘0’ en todos los bits de estado.



q2(t)	q1(t)	q0(t)	q2(t+1)	q1(t+1)	q0(t+1)
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	0	0

CUPL, “Cuerpo del programa”

- Ejemplo.

```
pin 1 = clk;
pin 2 = x;
pin [14..15] = [q0..1];

sequence q1,q0{
    present 0
        if x next 1;
    present 1
        if x next 3;
present 3
        if x next 0;
}
```

q1(t)	q0(t)	q1(t+1)	q0(t+1)
0	0	0	1
0	1	1	1
1	0	0	0
1	1	0	0

q0.d => !q1 & x

q1.d => !q1 & q0 & x

CUPL, “Cuerpo del programa”

- Comando **CONDITION**. Una forma alternativa de modelar sistemas combinacionales.
- Sintaxis:

CONDITION {

IF expr0 **OUT** var;

…

IF exprn **OUT** var;

DEFAULT OUT var;

}

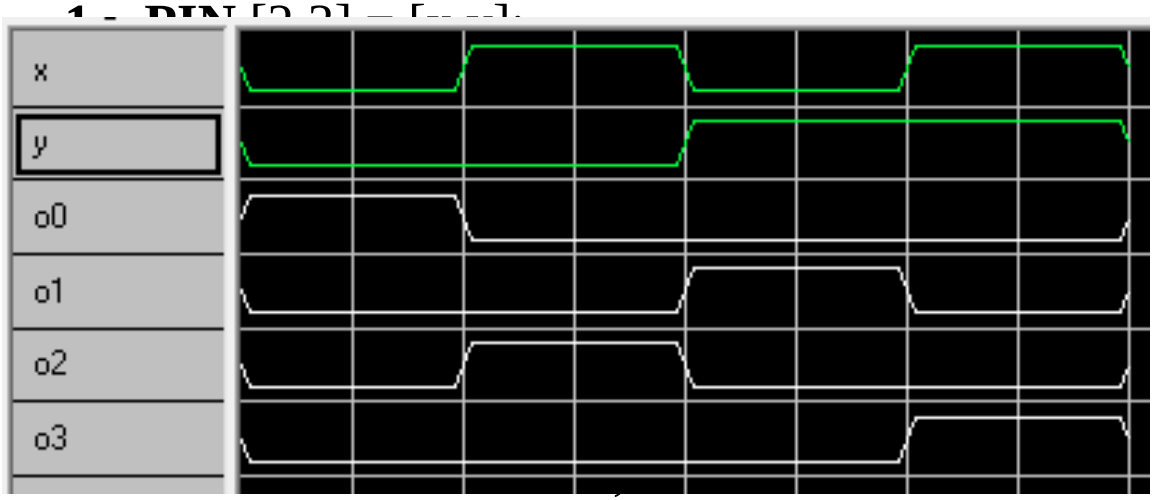
CUPL, “Cuerpo del programa”

```
PIN [2..5] = [ad0..3];
PIN [14..17] = [DAC,ADC,LCD,TEC];
PIN [18..20] = x,y,z;

field ad = [ad0..3];

condition {
        IF ad:[0..2] OUT DAC;
        IF ad:[3..8] OUT ADC;
        IF ad:[9..D] OUT LCD;
        IF ad:E OUT TEC; OUT x;  OUT y;
       default OUT z;
}
```

CUPL, “Cuerpo del programa”



}

2. **PIN** [2,3] = [x,y];
   PIN[14..17] = [o0..3];

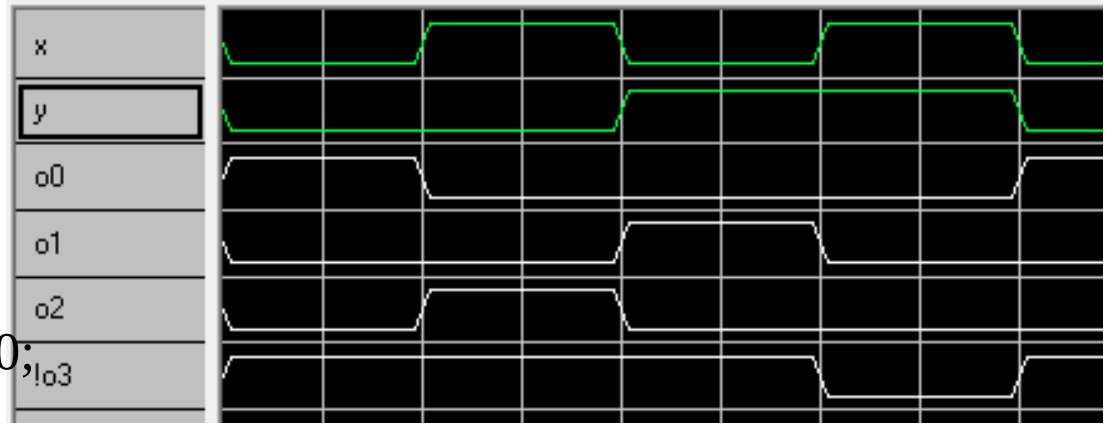
condition {
    IF !x & !y **OUT** !o0;
    IF !x & y **OUT** o1;
    IF x & !y **OUT** o2;
    default OUT o3;

}

3. **PIN** [2,3] = [x,y];
   PIN 14=o0;
   PIN 15=o1;
   PIN 16=o2;
   PIN 17=!o3;

condition {
    IF !x & !y **OUT** !o0;
    IF !x & y **OUT** o1;
    IF x & !y **OUT** o2;
    default OUT o3;

}



CUPL, “Cuerpo del programa”

- **Funciones**.
  - Permite encapsular expresiones lógicas.
  - Permite volver código reutilizable.
  - Sintaxis:

    FUNCTION name ([Parametro_0 ,...,Parametro_n])

    { body }

  - La función puede operar sola y/o mediante su “llamado” puede realizar vinculo con otros elementos del diseño.

CUPL, “Cuerpo del programa”

- Cuando no se usan parámetros, cumple la tarea de encapsular código:

```
PIN [2..4]=[a0..2];

PIN 21 = s0;
PIN 22 = s1;

function and() {
   s0 = a0 & a1 & a2;
}

and();
```

```
PIN [2..4]=[a0..2];

PIN 21 = s0;
PIN 22 = s1;

function and() {
   s0 = a0 & a1 & a2;
   s1 = a0 # a1 # a2;
}

and();
```

CUPL, “Cuerpo del programa”

- Cuando se usan parámetros, cumple la tarea de volver código reutilizable:

PIN [2..4]=[a0..2];

PIN [5..7]=[b0..2];

PIN [21,22] = s0,s1;

```
function and (e0,e1,e2,sal){
   sal = e0 & e1 & e2;
}

and(a0,a1,a2,s0);
and(b0,a0,b1,s1);
```

```
function and (e0,e1,e2){
    and = e0 & e1 & e2;
}

s0 = and(a0,a1,a2);
s1 = and(b0,b1,b2);
```

CUPL, “Cuerpo del programa”

```
PIN 1 = clk;
PIN [2..4]=[a0..2];
PIN [5..7]=[b0..2];
PIN [21,22] = [s0..1];
PIN [19,20] = [z0..1];

function and (e0,e1,e2,or){
    and = e0 & e1 & e2;
    or = e0 # e1 # e2;
}

s0 = and(a0,a1,a2,s1);
z1 = and(a0,b1,b2,z0);
```

CUPL, “Cuerpo del programa”

1. **function** and() {
   s0.d = a0 & a1 & a2;
   s1 = a0 # a1 # a2;
   }

2. **function** and (e0,e1,e2,or){
   and = e0 & e1 & e2;
   or.d = e0 # e1 # e2;
   }

   s0.d= and(a0,a1,a2,s1);
   z0 = and(b0,b1,b2,z1);

3. **function** and (e0,e1,e2,or){
   and.d = e0 & e1 & e2;
   or = e0 # e1 # e2;
   }

4. **function** and (e0,e1,e2,or){
   and = e0 & e1 & e2;
   or = e0 # e1 # e2;
   }

   s0.d = and(a0,a1,a2,s1.d);
   z0 = and(a0,a1,a2,z1);

CUPL, “Cuerpo del programa”

- Declaración **MIN**. Los niveles de minimización disponibles son:

Number	Corresponding Minimization
0	None
1	Quick
2	Quine-McCluskey
3	Presto
4	Expresso

- Sintaxis: **MIN** id_sal=Nivel_MIN;

  MIN q0 = 0;

  MIN a = 3;

CUPL, “Cuerpo del programa”

- Comparativo de prestaciones de los diferentes métodos de minimización de expresiones: