Lenguaje ABEL

- Introducción
- Estructura de un archivo fuente ABEL
  - Formato
  - Declaraciones
  - Operadores
  - Conjuntos
- Descripción lógica
  - Ecuaciones
  - Tablas
  - Diagramas de estados
- Vectores de test

Introducción

- ABEL significa
  - Advanced Boolean Equation Language
- Introducido por DATA I/O Corporation en 1984
  - Diversos fabricantes lo siguen manteniendo
- Es un **lenguaje de descripción de *hardware* ...**
  - Permite describir un circuito digital
  - Permite definir vectores de test para comprobar el funcionamiento
  - Permite generar el mapa de fusibles para programar el PLD

…de bajo nivel

- Muy cercano a la circuitería que representa
- Muy eficaz para describir diseños digitales pequeños (< algunos miles de puertas)

Estructura de un archivo fuente ABEL

Encabezamiento

Declaraciones

Descripción lógica

Vectores de test

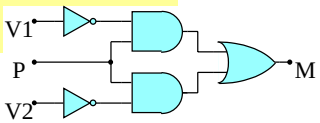
Final

```
module Bici

  U1 device 'P22V10';
  V1, V2, P  pin  2, 3, 4;
  M          pin  23;

  equations
    M = !V1 & P  #  !V2 & P;

  test_vectors
    ([ V1,  V2, P] -> [M])
     [.X., .X., 0] -> [0];
     [ 0 , .X., 1] -> [1];
     [ 1 ,  0 , 1] -> [1];

end
```

Declaración de la PAL

Declaración de pines

Los vectores de test se usan para simulación



Estructura de un archivo fuente ABEL

- Formato del texto:
  - Palabras clave: da igual mayúsculas que minúsculas
  - Nombres definidos por el usuario (identificadores): hay que mantener el estilo de su declaración
    - Por ejemplo, **V1** es distinto de **v1**
  - Las sentencias acaban en **;**
- Comentarios:
  - Empiezan por comillas
  - Se extienden hasta las siguientes comillas o hasta el final de línea

```
"pines de entrada
A, B, C  pin  2, 3, 4;
```

- Encabezamiento y final
  - Las palabras clave **`module`** y **`end`** marcan los límites de un diseño

Declaraciones

- Declaración del **dispositivo**

```
U1 device 'P22V10';
```

  - Es opcional e identifica el PLD a usar

```
identificador device cadena
```

Cadena alfanumérica que empieza por _ o una letra

Caracteres entre comillas simples

- Declaración de **pines**

```
V1, V2, P  pin  2, 3, 4;
```

```
id_pin1, id_pin2, ... pin nº_pin1, nº_pin2, ...
```

Lista de identif. para los pines

Lista de nº de pines (opcional)

- Declaración de **señales** internas

```
N1, N2, N3  node;
```

  - En las arquitecturas tipo SPLD se asignan a un pin de salida (si no se pueden simplificar en la síntesis)

Declaraciones

- Declaración de **constantes**

```
H, L = 1, 0;
```

```
id_cte1, id_cte2, ... = expr1, expr2, ...
```

Lista de identif. para constantes

Lista de expresiones

  - Los números en ABEL siempre representan **valores naturales**
  - Igual ocurre con los operadores aritméticos y relacionales
  - La base por defecto es decimal

Base	Símbolo	Ejemplo: 17 en decimal
Binaria	^b	^b010001
Octal	^o	^o21
Decimal	^d (defecto)	^d17 o 17
Hexadecimal	^h	^h11

Operadores

- Lógicos

!	NOT (complemento a 1)
&	AND
#	OR
$	XOR
!$	XNOR

- Relacionales
  - Comparan operandos en una expresión
  - El resultado puede ser 1 (verdadero) o 0 (falso)

==	Igual	**A==B**
!=	No igual	**A!=B**
>	Mayor que	**A>B**
>=	Mayor o igual que	**A>=B**
<	Menor que	**A<B**
<=	Menor o igual que	**A<=B**

- Aritméticos
  - Con conjuntos sólo **-** y **+**

-	Negación (compl. a 2)	**-A**
-	Resta	**A-B**
+	Suma	**A+B**
<<	Desplazamiento a izda.	**A<<B**
>>	Desplazamiento a dcha.	**A>>B**
*****	Multiplicación	**A*B**
/	División entera	**A/B**
%	Módulo (resto de A/B)	**A%B**

- Asignación

=	Asignación combinacional
:=	Asignación secuencial (en el siguiente flanco de reloj)

Conjuntos

- Lista de señales o constantes separadas por comas o por el operador de rango (**..**), encerradas entre corchetes
  - Simplifican la escritura de ecuaciones lógicas
  - Declaración de conjuntos

```
ASET = [A2,A1,A0];
BSET = [Y2..Y0];
```

  - Operaciones con conjuntos
    - Se realizan sobre cada elemento del conjunto
    - Los números se truncan o se rellenan con ceros

```
ASET & BSET
ASET & B0
2 & ASET
ASET = 2
```

⟺

```
[A2&B2, A1&B1, A0&B0]
[A2&B0, A1&B0, A0&B0]
[0&A2, 1&A1, 0&A0]
A2=0; A1=1; A0=0;
```

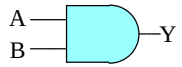
Descripción lógica

- Se puede realizar de varias formas:
  - Ecuaciones
    - Palabra clave: **`equations`**
    - Sentencia `when-then-else`
    - Extensiones de punto
  - Tablas
    - Palabra clave: **`truth_table`**
  - Diagrama de estados
    - Palabra clave: **`state_diagram`**
    - Sentencia `if-then-else`
    - Sentencia `goto`
    - Sentencia `with-endwith`

Ecuaciones

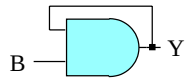
- Sentencias de asignación:
  - Operador de asignación combinacional (`=`)

```
equations
 Y = A & B;
```



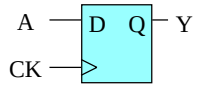
    - Evitar realimentación combinacional

```
equations
 Y = Y & B;
```



  - Operador de asignación secuencial (`:=`)

```
equations
 Y := A;
 Y.clk = CK;
```

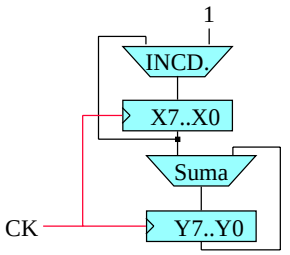


- El orden textual no importa

```
equations
 X := X+1;
 Y := Y+X;
```

```
equations
 Y := Y+X;
 X := X+1;
```

```
[X,Y].clk = CK;
```



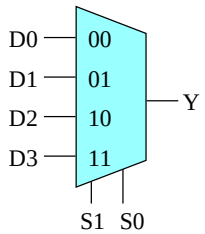
Ecuaciones

- Sentencia **`when-then-else`**
  - Equivale a una ecuación

```
equations
  when cond1 then
    Y = exp1
  else when cond2 then
    Y = exp2
  else
    Y = exp3;
```

→

```
equations
  Y = (cond1)&(exp1)
    # !(cond1)& (cond2) &(exp2)
    # !(cond1)& !(cond2)&(exp3);
```




```
SEL=[S1,S0];
equations
  when SEL==0 then Y=D0
  else when SEL==1 then Y=D1
  else when SEL==2 then Y=D2
  else Y=D3;
```

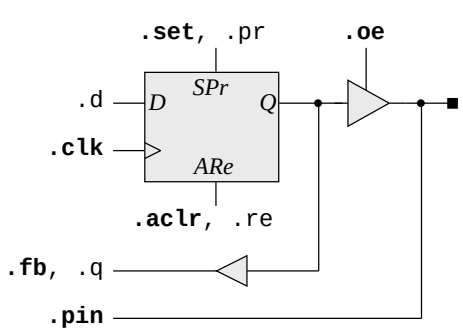

Extensiones de punto

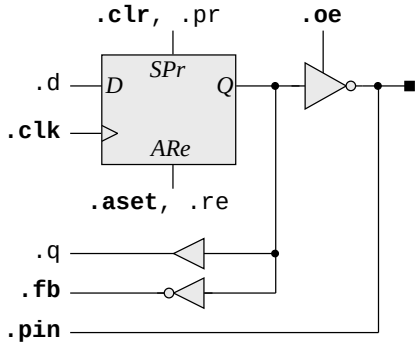
- Se aplican a las salidas y permiten:
  - Manejar señales especiales de la **estructura de salida** (OE, CLK; …)
  - Eliminar ambigüedad en la descripción
- Las hay independientes del dispositivo (*pin-to-pin*) y dependientes (*detailed*)

Independientes		Dependientes	
Extensión	**Descripción**	**Extensión**	**Descripción**
`.oe`	*Output enable*	`.re`	*Reset*
`.clk`	Reloj de biestable	`.ar`	*Reset* asíncrono
`.aclr`	*Reset* asíncrono	`.sr`	*Reset* síncrono
`.clr`	*Reset* síncrono	`.pr`	*Preset*
`.aset`	*Preset* asíncrono	`.ap`	*Preset* asíncrono
`.set`	*Preset* síncrono	`.sp`	*Preset* síncrono
`.fb`	Realim. Q de biestable	`.d`	Entrada de biestable D
`.pin`	Realim. de la salida	`.q`	Realim. Q de biestable

Extensiones de punto

- Recomendable usar las extensiones independientes del dispositivo
  - Se normalizan al pin de salida
- Ejemplos:





Ejemplos

- Contador **ascendente/descendente** módulo 8 con **RESET asíncrono**
  - Cuenta hacia arriba o hacia abajo, según el valor de la entrada **UP**.

```
module Cont8
  Cont8 device 'P22V10';
  CLK,RST,UP pin 1,2,3;
  Q2,Q1,Q0   pin 23,22,21;
  CN = [Q2,Q1,Q0];
equations
  when UP then CN:=CN+1
  else CN:=CN-1;
  CN.clk=CLK;
  CN.aclr=RST;
end
```

- Contador módulo 13 con **PRESET síncrono** y **RESET asíncrono.**
  - La entrada de PRESET coloca al contador a su valor máximo.

```
module Cont13
  Cont13 device 'P22V10';
  CLK,RST,PRT pin 1,2,3;
  Q3,Q2,Q1,Q0 pin 23,22,21,20;
  CN = [Q3,Q2,Q1,Q0];
equations
  when PRT then CN:=12
  else when (CN==12) then CN:=0
  else CN:=CN+1;
  CN.clk=CLK;
  CN.aclr=RST;
end
```

¡Ojo! Simulación de ecuaciones OK porque las salidas siempre están habilitadas
Para que la simulación sea OK indep. del PLD, **realimentar con la extensión `.fb`**

Realimentación en ecuaciones

- Posibilidades:
  - **Sin extensión**: Se realimenta del pin; si no existe, se realimenta del biestable con polaridad normalizada al pin.
  - **Con `.fb`**: Se realimenta del biestable con polaridad normalizada al pin; si no existe, se realimenta del pin.
  - **Con `.pin`**: Se realimenta del pin; si no existe resulta un error.
- Recomendable especificar de dónde se realimenta con **`.fb`** o **`.pin`**
  - Resulta un diseño independiente del PLD
  - La simulación de ecuaciones es igual que la simulación del JEDEC

  • Si no ponemos extensión:
    – Al compilar existen todos los caminos de realimentación <=> `.pin`
    – Pero al encajar el diseño en una 22V10, se tiene en cuenta que la realimentación es de biestable <=> `.fb`

  Aparecen diferencias entre una simulación y otra

Tablas

- Descripción tabular: salidas = *f* (combinaciones de entradas)
- Permite describir:
  - Circuitos combinacionales (tabla de verdad)

```
truth_table ([A,B] -> [Y])
             [0,0] -> [0];
             [0,1] -> [1];
             [1,0] -> [1];
```

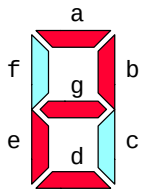
  - Circuitos secuenciales (tabla de estados)

```
truth_table ([QB,QA] :> [QB,QA] -> [Y])
              [0,0]  :>  [0,1]  -> [0];
              [0,1]  :>  [1,0]  -> [0];
              [1,0]  :>  [1,1]  -> [0];
              [1,1]  :>  [0,0]  -> [1];
```

- El valor de la salida para combinaciones de entradas no especificadas se toma como no importa

Ejemplo de tabla

- Decodificador BCD a 7 segmentos
  - Entradas: código BCD
  - Salidas: visualizador de cátodo común




```
module BCD7
  D3,D2,D1,D0         pin;
  a,b,c,d,e,f,g       pin;
  BCD = [D3..D0];
  ON,OFF = 1,0; "Invertir para ánodo común
  truth_table(BCD ->[  a,  b,  c,  d,  e,  f,  g])
              0  ->[ ON, ON, ON, ON, ON, ON,OFF];
              1  ->[OFF, ON, ON,OFF,OFF,OFF,OFF];
              ...
              9  ->[ ON, ON, ON, ON,OFF, ON, ON];
end BCD7
```


Diagramas de estados

- ABEL permite describir máquinas de estados (ME) introduciendo el diagrama de estados
  - Se realiza con la palabra clave **`state_diagram`**
- 4 Pasos en la descripción de una ME en ABEL:
  - Definir las variables de estado.
  - Definir/codificar los estados.
  - Definir la función de transición de estados
    - Sección **`state_diagram`**
  - Definir la función de salida
    - Sección **`state_diagram`** o **`equations`**

Diagramas de estados

- La estructura de la sección **`state_diagram`** es:

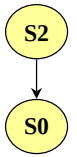
```
state_diagram reg_estado
    state id_estado:
          [ecuaciones;]
          cond_trans;
    ...
```

  - **reg_estado**, identificador o conjunto de identificadores que especifican los biestables de estado
  - **id_estado**, identificador del estado actual
  - **ecuaciones**, definen las salidas para el estado actual (opcional)
  - **cond_trans**, condición de transición de estado que define:
    - Estado siguiente: **`if-then-else`** y **`goto`**
    - Opcionalmente ecuaciones de salida asociadas a las transiciones entre estados: **`with-endwith`**

Diagramas de estados

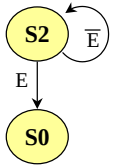
- Tipos de condiciones de transición de estado:
  - Transiciones incondicionales

```
state_diagram ESTADO
  state S2: goto S0;
  ...
```



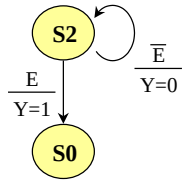
  - Transiciones condicionales

```
state_diagram ESTADO
  state S2: if E then S0
            else S2;
  ...
```



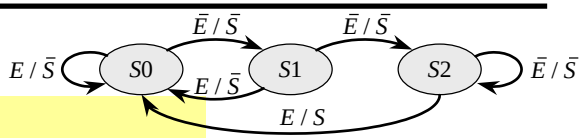
  - Transiciones con **`with-endwith`**

```
state_diagram ESTADO
  state S2:
    if E then S0 with Y=1;endwith;
    else S2 with Y=0; endwith;
```



Ejemplo: máquina de Mealy

- Detector …001...




```
module Mly_001
  CK,RST,E,S  pin;
  Q1,Q0       pin istype 'reg';
  EST = [Q1, Q0];
  S0 = [0, 0];
  S1 = [0, 1];
  S2 = [1, 1];
  state_diagram EST
    state S0:  if !E then S1
               else S0;
    state S1:  if !E then S2
               else S0;
    state S2:  if E then S0
               else S2;
  equations
    S = E & (EST == S2);
    EST.clk = CK; EST.aclr = RST;
end
```

1 Definir variables de estado

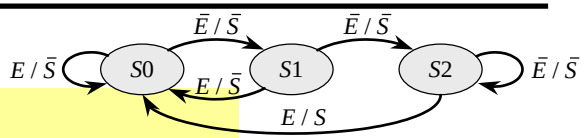
2 Definir/codificar los estados

3 Función de transición de estados

4 Función de salida

Ejemplo con with-endwith

- Detector …001...




```
module With_001
  CK,RST,E,S  pin;
  Q1,Q0       pin;
  ESTADO = [Q1, Q0];
  S0 = [0, 0];
  S1 = [0, 1];
  S2 = [1, 1];
  state_diagram EST
    state S0:
      S = 0;
      if !E then S1 else S0;
    state S1:
      S = 0;
      if !E then S2 else S0;
    state S2:
      if E then S0 with S = 1; endwith;
      else S2 with S = 0; endwith;
  equations
    EST.clk = CK;  EST.aclr = RST;
end
```

- Salida codificada en la sección *state_diagram*

Vectores de test

- Palabra clave: **`test_vectors`**
  - Define los vectores de prueba para verificar la funcionalidad del diseño
  - Especifican la salida esperada en función de las entradas (similar a las tablas)
- Se usan constantes especiales:
  - .X. No importa
  - .Z. Alta impedancia
  - .C. Pulso de reloj
  - .D. Flanco descendente
  - .U. Flanco ascendente
  - .X. en simulación:
    - Como entrada, el simulador escogerá un valor (0 ó 1)
    - Como salida, dará lo mismo el valor que aparezca a la salida

```
"Contador con reset asíncrono activo en H y OEN
test_vectors
   ([CLOCK, RESET, OE ] -> [ Q2,  Q1,  Q0])
    [ .X.,    1,    1 ] -> [ 0,   0,   0 ];
    [ .C.,    0,    1 ] -> [ 0,   0,   1 ];
    [ .C.,    0,    1 ] -> [ 0,   1,   0 ];
    [ .C.,    0 ,   0 ] -> [.Z., .Z., .Z.];
```