Utilización de Lattice ispLEVER

- Creación del proyecto:
  - Creación del proyecto en un directorio
  - Especificación del tipo de proyecto
  - Selección del dispositivo
- Esquemáticos
  - Uso de celdas prediseñados (librerías)
  - Celdas se corresponden con ecuaciones lógicas y/o biestables
  - Diseños jerárquicos -> un bloque diseñado se puede usar como una celda prediseñada
  - Posicionamiento
    - celdas
    - conexiones
    - etiquetas + puertos (pines) de E/S

ABEL

- HDL = Hardware Description Language
  - Verilog
  - VHDL
  - ABEL
- Estructura de un programa:
  - Cabecera
  - Declaraciones
  - Descripciones lógicas
  - Vectores de test
- Cabecera
  - ***MODULE*** nombre -> nombre del módulo (oblogatorio)
  - ***INTERFACE*** -> declaración del interfaz con el exterior: E y S (opcional)
  - ***TITLE*** 'título' -> título del diseño. aparece en la cabecera de los ficheros de salida

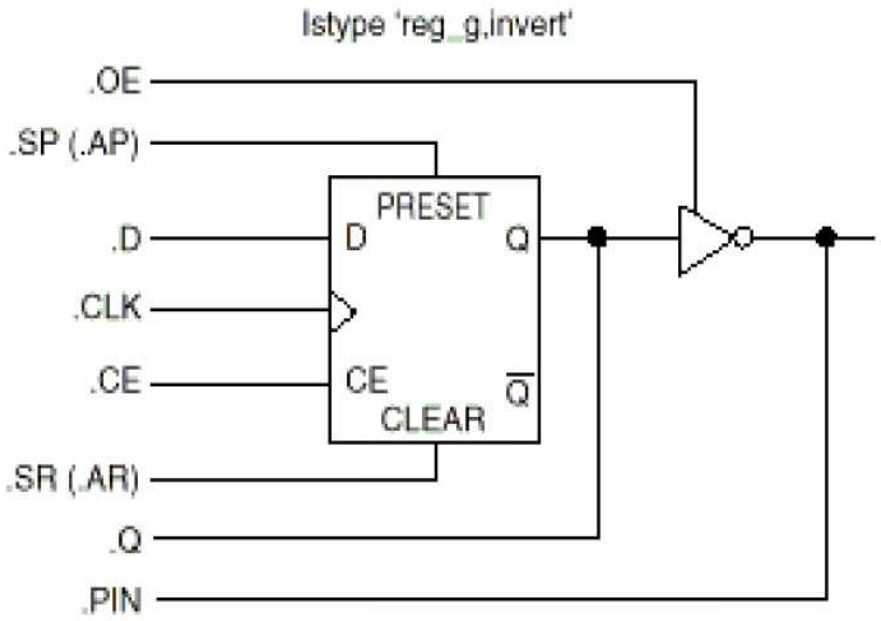
- Declaraciones
  - pines
    - Define los pines conectados al exterior, su función, polaridad y les asigna nombre
    - Sintaxis: [!] nombre_pin1 [, [!] nombre_pin2 ...] ***PIN*** [num1 [, num2]..] [***ISTYPE*** atributos]
    - ! -> activo a nivel bajo
    - nombre_pin = nombre que se asigna al pin. Definición de varios pines a la vez:
      - Pines "sueltos" -> pin1, !pin2 PIN 1,2
      - Buses -> pin3..0 PIN 1..4
    - num1 -> número de pin asignado a cada etiqueta (opcional)
  - atributos
    - definen características de señales (pines o nodos internos)
    - sintaxis: señal [, señal2 ...] [***PIN*** | ***NODE*** [num]] ***ISTYPE*** atributos
    - Algunos atributos: reg (registro), invert (salida mediante inversor), reg_jk, reg_t (tipos de registro), com (combinacional)
  - nodos
    - Define nodos internosr, su función, polaridad y les asigna nombre
    - Sintaxis: [!] nombre_nodo1 [, [!] nombre_nodo2 ...] ***NODE*** [num1 [, num2]..] [***ISTYPE*** atributos]

- constantes
  - Sintaxis: nombre1 [, nombre2 ...] = expresión1 [, expresión2 ..]
  - ejemplos:
    - valor = 16*3
    - dato = [1, 0, 0, 1]
    - constantes predefinidas: .X. (no importa) .Z. (alta Z), .C. (flanco de subida) .....
- maquinas de estado
  - Declara una máquina de estados
  - Sintaxis: nombre1 [, nombre2 ...] ***STATE_REGISTER*** [***ISTYPE*** atributos]
  - ejemplo:
    - automata STATE_REGISTER; -> declara una maquina de estados llamada "automata"
- estados
  - Definen los distintos estados y los pines que se va a ausar para cada estato. Utiliza la codificación one-hot, es decir, un bit para cada estado
  - Sintaxis: nombre1 [, nombre2 ...] ***STATE*** [valor1 [, valor2]..] [***ISTYPE*** atributos]
    - nombre -> señales asociadas al estado
    - valor -> valor asociado al estado
  - ejemplos:
    - S0..S3 state; -> define 4 pines (y cuatro estados) S0..S3

- Descripción lógica
  - Distintas posibilidades:
    - Ecuaciones
    - Tablas de verdad
    - Diagramas de estados
    - Fusibles
    - Factores XOR
  - Extensiones punto
    - Permiten referirse de forma precisa a señales internas del circuito asociadas a puertas, biestables, etc.
    - Ejemplo:

```
Q1 PIN ISTYPE 'reg_g.invert'


Q1.clk = Clock;
Q1.D = !Q1.Q # Preset;
```



- Expresiones pin-a-pin y expresiones detalladas
  - Pin a pin -> se refieren a las señales en los pines => independientes de la arquitectura del dispositivo

    Q1 := !Q1
  - Detalladas -> utilizan señales internas para describir en detalle => dependientes de la arquitectura del dispositivo

    Q1.D = !Q1
- Operadores
  - Lógicos ->bit a bit. ! (negación=complemento a 1), & (and), # (or), $(XOR)
  - Relacionales -> resultado = 1/0. igual que en C (==, !=, >= .....)
  - Aritméticos -> operan sobre símbolos o conjuntos de bits:

    -A (cambio de signo -complemento a dos) A – B (resta)

    A + B (suma)

    A * B (multiplicación)

    A / B (división entera)

    A % B (resto de la división entera)

    A >> b (rotación a derecha b bits) A << b (rotación a izquierda)

- de asignación

  = -> asignación combinacional

  y = (!a & b)

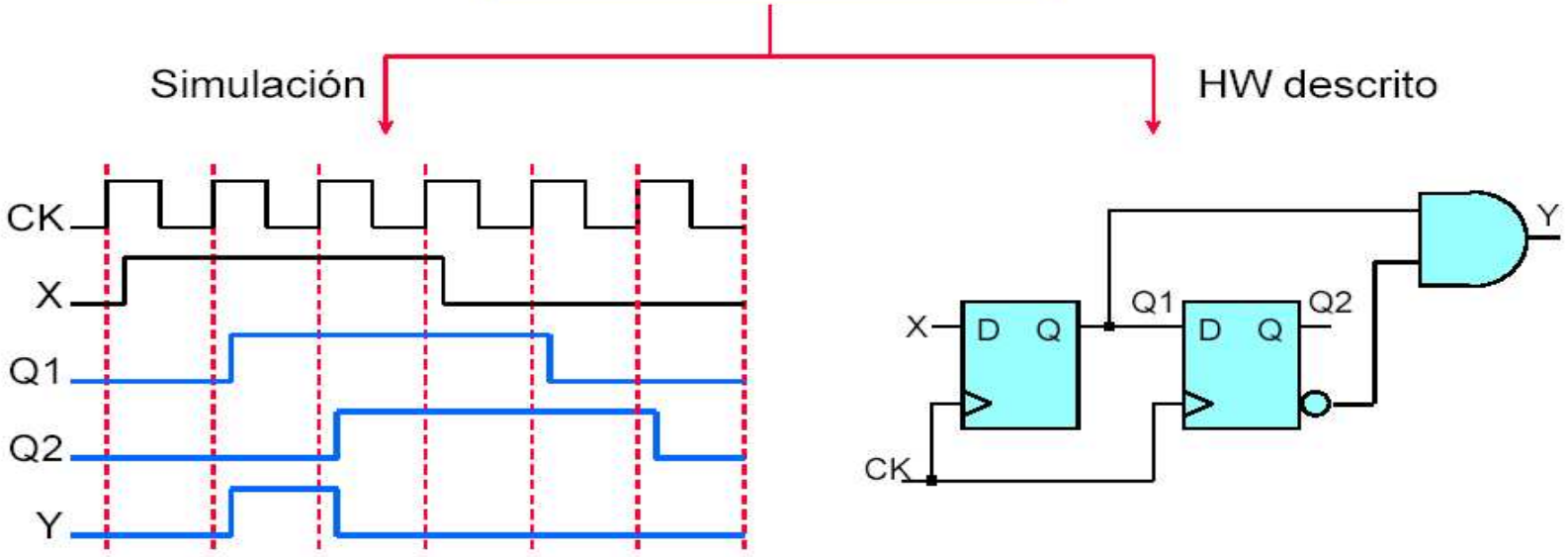
  Q1.D = !Q1 -> vale también para registros pero con expresiones detalladas

  := -> asignación registrada (efectiva en el siguiente flanco de reloj)

  Q1 = !Q1 -> sólo válido para registros en expresión pin a pin

- Ecuaciones -> asignación
  - p. clave ***EQUATIONS***
  - No importa el orden

```
equations
 Q1  := X;
 Q2  := Q1;
 Y   = Q1&!Q2;
 [Q1,Q2].clk = CK;
```



- Ecuaciones -> sentencias when-then-else
  - Sintaxis:

    WHEN condición ***THEN*** ecuación1 [***ELSE*** ecuación2] ;
  - Se pueden anidar
  - Pueden afectar a bloques de expresiones (entre {})
  - ejemplo:

```
WHEN (Mode == S_Data)
        THEN { Out_data := S_in; S_Valid := 1; }
ELSE
        WHEN (Mode == T_Data)
        THEN { Out_data := T_in; T_Valid := 1; }
```

- Tablas de verdad
  - Permiten introducir tablas de verdad directamente
  - Posiblidad de combinaciones “no importa”
  - Dos posibilidades
    - Tablas combinacionales -> tablas de verdad “normales”
      - Sintaxis:

        TRUTH_TABLE (entradas -> salidas)

        entradas -> salidas ;
      - Ejemplo:

        TRUTH_TABLE ([en, A , B] -> C)
        [0,.X.,.X.] -> .X.;
        [1, 0 , 0] -> 0 ;
        [1, 0 , 1] -> 1 ;
        [1, 1 , 0] -> 1 ;
        [1, 1 , 1] -> 0 ;
    - Tablas registradas -> tablas de estados
      - Sintaxis:

        TRUTH_TABLE (entradas :> salidas_registradas [-> salidas_combinacionales])

        entradas -> salidas_registradas [-> salidas_combinacionales] ;
      - Salidas_registradas -> valor de la salida registrada correspondiente que se activará en el siguiente flanco de reloj (síncrona)
      - Salidas_combinacionales -> valor de la salida combinacional que cambia al cambiar la entrada (asíncrona)
      - Se pueden usar para especificar tablas de estados si entradas = estado actual + entradas y salidas_registradas=estado próximo.
      - Si se usan las salidas_combinacionales => autómata de MEALY, si no, de MOORE.

- Ejemplo (contador módulo 4, con fin de cuenta -MEALY-)

```
TRUTH_TABLE ( [Q1,Q0] :> [D1,D0] -> Fin_cuenta )
    0 :> 1 -> 0;
    1 :> 2 -> 0 ;
    2 :> 3 -> 0 ;
    3 :> 0 -> 1 ;
```

- Ejemplo (contador módulo 4, con fin de cuenta -MOORE-)

```
TRUTH_TABLE ( [Q1,Q0] :> [D1,D0, Fin_cuenta] )
    [0,0] :> [0,1,0];
    [0,1] :> [1,0,0];
    [1,0] :> [1,1,1];
    [1,1] :> [0,0,0];
```

Diagramas de estado

- Definición del diagrama de estados:
  - Cabecera
  - Descripción de cada estado -> una entrada para cada estado
  - Descripción de las transiciones -> dentro de cada estado, una entrada para cada transición
- Cabecera
  - Define las señales implicadas en el autómata: bits de estado y salidas
  - Sintaxis: ***STATE_DIAGRAM*** registros_estado [-> salidas]
  - Donde
    - registros_estado -> lista de las señales de estado (o bien maquina de estado declarada con STATE_REGISTER)
    - salidas -> listas de salidas para cada estado (MOORE)

- Dos formas de definir los estados
  - Codificación one-hot -> un bit para cada estado – minimiza la lógica

```
sreg1 state_register;
S0..S3 state;
equations
sreg1.clk = clock;
state_diagram sreg1
state S0:
goto S1 with {x = a & b;
y = 0; }
state S1: if (a & b)
then S2 with {x = 0;
y = 1; }
```

  - Codificación n biestables 2^n estados -> menos biestables – más lógica

```
sreg = [q1,q0];
"State Values...
A = 0; B = 1; C = 2;
state_diagram sreg;
State A: " Hold in state A until start is active.
in_B = 0;
in_C = 0;
IF (start & !reset) THEN B WITH halt := 0;
ELSE A WITH halt := halt.fb;
State B: " Advance to state C unless reset is active
in_B = 1;
```

- Descripción del estado
  - Después de la cabecera. Una entrada para cada estado
  - sintaxis:

    STATE estado_actual :
    [salidas_síncronas]
    transiciónes;
  - donde
    - salidas_síncronas (MOORE) -> ecuación (o ecuaciones) salida = expresión
    - transiciones -> especifica transiciones y salidas (MEALY) con expresiones IF-THEN-ELSE, CASE, GOTO o WITH.
- Transiciones
  - Sentencia IF-THEN_ELSE
    - Sintaxis:

      IF condición
      THEN
        estado_próximo [***WITH*** salidas].;
      [***ELSE***
        estado_próximo [***WITH*** salidas]] ;
    - Donde:
      - condición = expresión lógica
      - salidas = ecuación (o ecuaciones) salida = expresión

- Sentencia GOTO
  - Define una transición incondicional
  - Sintaxis: ***GOTO*** estado_próximo [***WITH*** salidas];
- Sentencia CASE
  - sintaxis:

CASE

condición : estado_próximo [***WITH*** salidas];

[condición : estado_próximo [***WITH*** salidas];]

.....

ENDCASE ;