Lattice ispLEVER

- Características
  - Conjunto de herramientas para el diseño con CPLD y FPGA
  - Versión ispLEVER starter gratuita (licencia 6 meses)
  - Entorno de desarrollo integrado -> incluye
    - Gestión de proyectos
    - Diseño
    - Validación del diseño
    - Simulación
    - Programación de dispositivos
- Gestión de proyectos
  - El “Project Navigator” permite gestionar proyectos y acceder a todas las herramientas de ispLEVER
  - Creación del proyecto:
    - Creación del proyecto en un directorio
    - Especificación del tipo de proyecto
    - Selección del dispositivo

- Diseño
  - Permite usar esquemáticos o HDL (ABEL, verilog o VHDL)
  - 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
- Validación/compilación del diseño
  - A partir del diseño se compila, generando las ecuaciones y preparando las conexiones en el dispositivo
  - Genera el fichero JEDEC para la programación
- Simulación
  - Uso de vectores de test
  - Simulación previa a la programación

XILINX ISE Webpack

- Características
  - Versión gratuita del ISE Foundation
  - Soporte de la mayoría de los dispositivos de xilinx
  - Versiones Windows/Linux.
  - Muy similar a Lattice ispLEVER
  - Posibilidad de entrada como
    - Esquemáticos
    - HDL (ABEL, Verilog, VHDL)
    - Diagramas de estados
  - Entorno de desarrollo integrado -> incluye
    - Gestión de proyectos
    - Diseño
    - Validación del diseño
    - Simulación
    - Programación de dispositivos

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 (obligatorio)
  - ***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

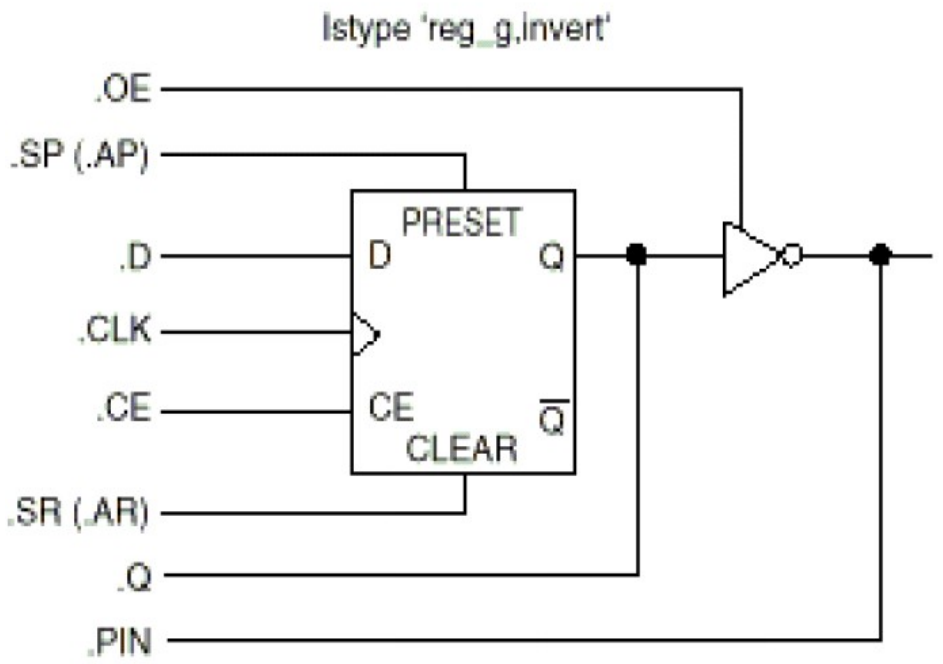
- Declaraciones
  - pines
    - Define los pines conectados al exterior, su función, polaridad y les asigna nombre
    - Sintaxis: [!] nombre_pin1 [, [!] nombre_pin2 ...] ***PIN*** [num1 [, num2]..] [***ISTYPE*** atributos]
    - ! -> activo a nivel bajo
    - nombre_pin = nombre que se asigna al pin. 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) .....
-

- 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)

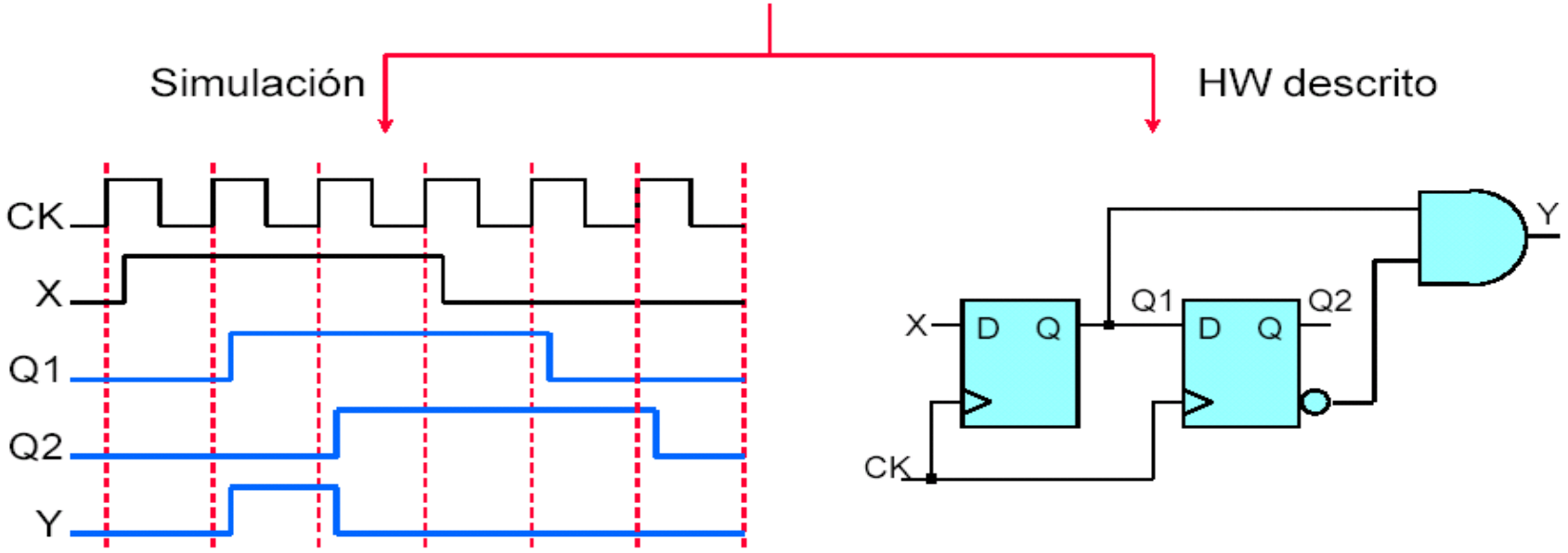
  Q1.D = !Q1 -> vale también para registros pero con expresiones detalladas

  := -> 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 (control de luz con interruptor y sensor):

```
MODULE luz;
Interruptor, bombilla, sensor_luz PIN;
EQUATIONS
        WHEN (interruptor == 0) THEN
                bombilla = 0;
        ELSE
                WHEN ( sensor_luz=0)  THEN
                        bombilla = 0;
                ELSE
                        bombilla = 1;
```

- ejemplo de contador módulo 10 realizado con ecuaciones registradas

```
MODULE mapa
TITLE 'Prueba de mapas registrados'

[S3..S0] pin istype 'reg';     “las 4 salidas del contador
reloj, reset pin;

salida=[S3..S0];

Equations
[S3..S0].CLK=reloj;
[S3..S0].AR=reset;    “reset asíncrono del contador

when salida==9 then salida=0    “da la vuelta al llegar a 9
else salida = salida+1;              “o si no, sigue contando

END
```

- Tablas de verdad
  - Permiten introducir tablas de verdad directamente
  - Posibilidad 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
  - Dos tipos de máquinas de estados
    - Con codificación binaria -> mínimo número de biestables, más lógica (normalmente mejor para CPLDs)
    - Con codificación “one-hot” -> un biestable por estado, menos lógica (puede ser mejor en FPGAs)
  - Autómatas con codificación “one-hot”
    - Declaración de la máquina de estado (sección de declaraciones, al principio)
      - Declara una máquina de estados (sólo para codificación one-hot)
      - Sintaxis: nombre1 [, nombre2 ...] ***STATE_REGISTER*** [***ISTYPE*** atributos]
      - ejemplo:
        - automata STATE_REGISTER; -> declara una maquina de estados llamada “automata”
    - Estados (sección de declaraciones, al principio)
      - Definen los distintos estados y los biestables que se va a a usar para cada estado (un biestable por estado)
      - Sintaxis: nombre1 [, nombre2 ...] ***STATE*** [valor1 [, valor2]..] [***ISTYPE*** atributos]
        - nombre -> señales asociadas al estado
        - valor -> valor asociado al estado (opcional, si no se especifica, el programa intenta optimizar)
      - ejemplos:
        - S0..S3 state; -> define 4 pines (y cuatro estados) S0..S3Definición del diagrama de estados:
    - Asignación de reloj (en la sección de ecuaciones)
      - Asignación del reloj a los relojes de los biestables.
      - ejemplo:
        - [S3..S0].CLK=reloj; -> conecta a los CLK de los biestables la señal “ reloj”, que será la señal de reloj en el circuito.

- Definición del diagrama de estados
  - Incluye
    - Cabecera
    - Descripción de cada estado -> una entrada por 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)
    - ejemplo:
      - STATE_DIAGRAM MiAutomata; -> donde MiaAutomata estará declarado en la seccion STATE_REGISTER
  - 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

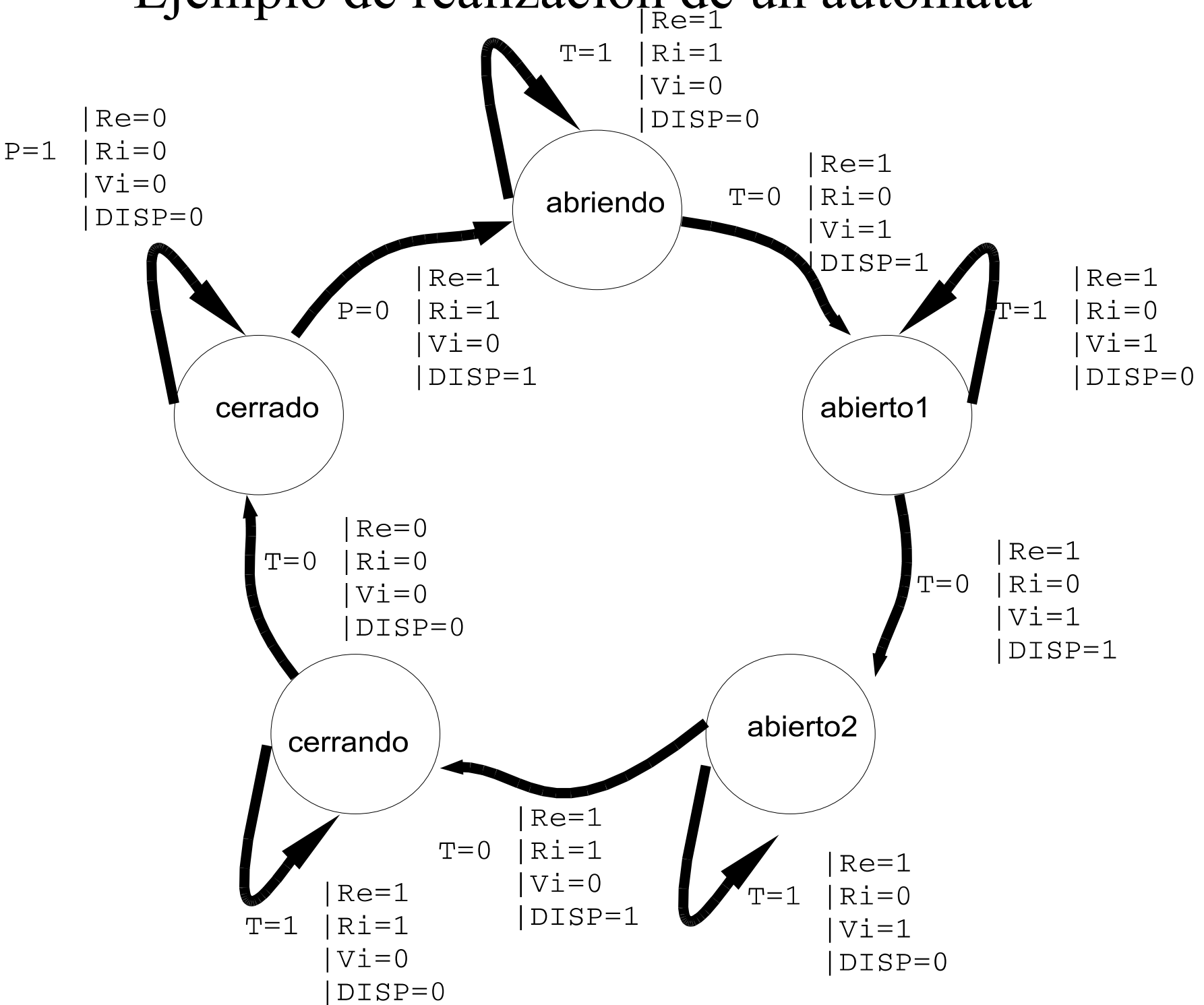
      condición : estado_próximo [***WITH*** salidas];

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

      .....

      ENDCASE ;

Ejemplo de realización de un autómata



Explicación del circuito

- Problema 1 del bloque de problemas de combinacionales (semáforo de la rampa del garaje)
- Solución hecha con un único temporizador de 22s.
  - El temporizador se dispara en flanco de subida de su señal de disparo
  - Su salida se pone a nivel alto durante 22s.
- El temporizador se va disparando en los distintos estados:
  - Se dispara una vez para temporizar la subida de la puerta del garaje (22s)
  - Se dispara dos veces consecutivas para temporizar el tiempo que la puerta está abierta (44s)
  - Se dispara una vez para temporizar el cierre de la puerta del garaje (22s)

- Ejemplo de codificación one-hot

```
MODULE semaforo
clock, P, T, Re, Ri, Vi, DISP pin;          “declaracion de pines
automata state_register;                    “declaramos el automata
cerrado,abriendo,abierto1, abierto2,cerrando state;    “declaramos los estados

equations
automata.clk = clock;

state_diagram automata                      “definimos el automata
state cerrado:
    if (P==1) then cerrado with {Re=0; Ri=0; Vi=0; DISP=0;}
   else abriendo with {Re=1; Ri=1; Vi=0; DISP=1;}
state abriendo:
    if (T==1) then abriendo with {Re=1; Ri=1; Vi=0; DISP=0;}
    else abierto1 with {Re=1; Ri=0; Vi=1; DISP=1;}
state abierto1:
    if (T==1) then abierto1 with {Re=1; Ri=1; Vi=0; DISP=0;}
    else abierto2 with {Re=1; Ri=0; Vi=1; DISP=1;}
state abierto2:
    if (T==1) then abierto2 with {Re=1; Ri=1; Vi=0; DISP=0;}
    else cerrando with {Re=1; Ri=1; Vi=0; DISP=1;}
state cerrando:
    if (T==1) then cerrando with {Re=1; Ri=1; Vi=0; DISP=0;}
    else cerrado with {Re=0; Ri=0; Vi=0; DISP=0;}
END
```

- Autómatas con codificación binaria
  - No hay declaración de la máquina de estado
  - Declaración de los estados (sección de declaraciones)
    - se da nombre a los estados y se le asigna un valor a cada uno.
    - Ejemplos
      - cero = [0,0]; uno =[0,1], dos=[1,0]
      - A=0; B=1; C=2;
  - Asignación de reloj -> igual que en los one-hot
  - Definición del diagrama de estados
    - Igual que en los one-hot, excepto la cabecera, que en vez del nombre de la máquina de estados lleva una lista de las señales de estado usadas
    - ejemplo:
      - STATE_DIAGRAM [Q2,Q1,Q0]
  - Ejemplo codificación binaria

- Ejemplo codificación binaria

```
MODULE semaforo
clock, P, T, Re, Ri, Vi, DISP pin;          “declaración de pines
[Q2..Q0] pin istype 'reg';     “declaramos los biestables de estado
“definimos la codificación de los estados
cerrado=[0,0,0]; abriendo=[0,0,1]; abierto1=[0,1,0];
abierto2=[0,1,1]; cerrando=[1,0,0];
equations
[Q2..Q0].clk = clock;
state_diagram [Q2..Q0]                      “definimos el automata
state cerrado:
    if (P==1) then cerrado with {Re=0; Ri=0; Vi=0; DISP=0;}
   else abriendo with {Re=1; Ri=1; Vi=0; DISP=1;}
state abriendo:
    if (T==1) then abriendo with {Re=1; Ri=1; Vi=0; DISP=0;}
    else abierto1 with {Re=1; Ri=0; Vi=1; DISP=1;}
state abierto1:
    if (T==1) then abierto1 with {Re=1; Ri=1; Vi=0; DISP=0;}
    else abierto2 with {Re=1; Ri=0; Vi=1; DISP=1;}
state abierto2:
    if (T==1) then abierto2 with {Re=1; Ri=1; Vi=0; DISP=0;}
    else cerrando with {Re=1; Ri=1; Vi=0; DISP=1;}
state cerrando:
    if (T==1) then cerrando with {Re=1; Ri=1; Vi=0; DISP=0;}
    else cerrado with {Re=0; Ri=0; Vi=0; DISP=0;}
END
```