Índice de contenido

Introducción

Estado actual de la lógica programable

El ..

Lenguaje de descripción

Introducción

El error más grave que cometemos es intentar programar en VHDL como si éste fuera un lenguaje de programación de computadoras de alto nivel. Los lenguajes de programación de alto nivel son ***secuenciales*** por naturaleza; VHDL no. VHDL se inventó para describir hardware y de hecho es un lenguaje ***concurrente*** (todo al mismo tiempo). Los dos propósitos principales de un lenguaje de descripción de hardware son: primero, que puede ser usado para modelar circuitos y sistemas digitales; segundo, teniendo el modelo, permite la subsecuente prueba y simulación del sistema.

El acto de conectar un montón de líneas y bloques, pronto se vuelve tedioso, con VHDL, tenemos herramientas para modelar circuitos digitales de una manera más inteligente. Esta metodología de diseño nos permitirá dedicar más tiempo en el diseño de nuestros circuitos y menos tiempo en construirlos.

A continuación listamos algunas *reglas que nos serán de utilidad, cuando trabajamos con VHDL*:

- *Cuando trabajamos con VHDL, no estamos programando, estamos diseñando hardware. Si el código en VHDL se parece al código en un lenguaje de alto nivel, entonces es un mal código en VHDL.*
- *Los circuitos digitales más complejos pueden ser expresados en términos de construcciones digitales básicas.*

Herramientas necesarias para el desarrollo con VHDL

La implementación exitosa de sistemas con VHDL comienza por la escritura del código, compilación, simulación y síntesis. Los fabricantes de CPLD y FPGA proporcionan un conjunto de herramientas hardware y software que podemos usar para llevar a cabo los mencionados pasos.

Consideraciones antes de continuar con VHDL

- *VHDL no es sensible a las mayúsculas* (Tienen el mismo significado: and y AnD).
- *VHDL no es sensible a los espacios en blanco* (El mismo significado: Q<=A; y Q <= A;).

- *Los comentarios comienzan con: "--"* (dos guiones), usar los comentarios apropiados facilita el entendimiento de la descripción.
- La mejor idea es *usar paréntesis libremente* para asegurar el entendimiento del código.
- Cada instrucción en *VHDL termina con ";"* (punto y coma).
- Tomar en cuenta lo siguiente para evitar errores en las instrucciones *if, case y loop*:
  - Cada "**if**" tiene su correspondiente "**then**".
  - Cada "**if**" debe terminar con "**end if**".
  - Si es necesario usar "**else if**", la versión en VHDL es "**elsif**".
  - Cada "**case**" debe terminar con "**end case**".
  - Cada "**loop**" debe terminar con "**end loop**".
- Para los *identificadores* (nombres de variables, constantes, señales, puertos, etcétera):
  - Los identificadores se deben explicar por sí mismos.
  - Pueden ser tan largos como se desee.
  - Pueden contener una combinación de letras, números y guión bajo.
  - Deben comenzar con un carácter alfabético.
  - No pueden finalizar con guión bajo, ni tener dos guiones consecutivos.
- No se pueden usar las palabras reservadas(process, label, after, etcétera) como identificadores.

Unidades básicas de diseño

Las descripciones de circuitos en VHDL están basadas en el concepto de "caja negra". En términos de VHDL, la caja negra está descrita por la declaración de una entidad, la cuál identifica con claridad las entradas y salidas generales del sistema.

```
entity nombre de la entidad is [generic (opcional, para declarar variables locales);]
   port (nombre de la señal: modo tipo;
         nombre de la señal:  modo tipo;
            :
         nombre de la señal: modo tipo);
end nombre de la entidad;
```

Modo (especifica la dirección de la señal)

in	Indica que la señal es de entrada.
out	Indica que la señal es de salida. Su valor puede sólo ser leído por otras entidades.
buffer	Permite hacer realimentaciones internas dentro de la entidad. Se comporta como salida.
inout	Señal bidireccional.

Tipo

bit	Puede tener el valor de 0 y 1.

bit_vector	Conjunto de bits para cada variable de entrada o salida.
integer	Puede tener un rango de valores enteros.
boolean	Puede tener TRUE y FALSE.
real	Puede tener un rango de valores reales.
character	Cualquier carácter.
time	Para indicar el tiempo.
std_logic_vector	*
std_logic	*

*Tipos contenidos en la librería **iee**, en el paquete **std_logic_1164**. Una vez que estas librerías han sido incluidas, tendremos acceso a una variedad de ventajas como: varios tipos de datos, funciones de conversión, funciones matemáticas, etcétera.

Las señales pueden ser expresadas como individuales o un conjunto de ellas. Por ejemplo, las señales en un "bus" pueden ser expresadas como un vector (std_ligic_vector), éstas pueden ser expresadas considerando dos tipos de orden por medio de las palabras reservadas "**TO**" y "**DOWNTO**". Si queremos que el bit más significativo del conjunto sea el primer bit de la izquierda usamos la palabra: "**DOWNTO**". Es decir:

DOWNTO = $B_7B_6B_5B_4B_3B_2B_1B_0$

TO = $B_0B_1B_2B_3B_4B_5B_6B_7$

La arquitectura describe lo que el circuito hace. En otras palabras, en VHDL describe la implementación interna de la entidad asociada a ella.

architecture *nombre de la arquitectura* **of** *nombre de la entidad* **is**
- declaraciones
  - componentes
  - señales
  - constantes
  - función
  - procedimiento
  - tipos

begin
- sentencias

end *nombre de la arquitectura;*

Una arquitectura puede ser modelada de diferentes maneras. *Entendiendo las diferentes técnicas de modelado y cómo usarlas, representa el paso más importante para aprender VHDL*. Una arquitectura puede ser escrita mediante tres técnicas de modelado, más la combinación de cualquiera de las tres:

- **Modelos**
  - Flujo de datos
  - Funcional

- Estructural
- Híbrido

Dentro de los tipos de objetos más usados tenemos las señales(**signal**), para representar un alambre; variables(**variable**), usadas para almacenar información local y las constantes (**constant**), que son como una variable, pero cuyo valor no puede ser cambiado.

- *Las señales se declaran al inicio de la arquitectura, antes de la palabra* "**begin**".
- *Las variables se declaran dentro de la estructura* "process", *antes de la palabra* "**begin**".

El paradigma de la descripción en VHDL se fundamenta en el paralelismo y la concurrencia con la descripción textual de los circuitos. El corazón de la descripción en VHDL es la sentencia concurrente. Existen cuatro ***tipos de sentencias concurrentes***:

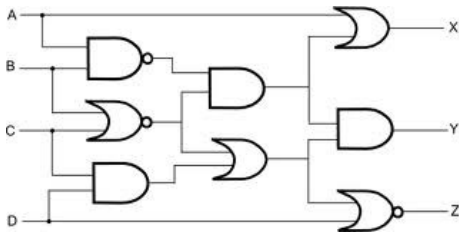
1. *Asignación concurrente de señal.*
2. *Asignación condicional de señal.*
3. *Asignación selectiva de señal.*
4. *Proceso.*

Asignación concurrente

La forma general de la asignación concurrente se muestra a continuación:

<destino> <= <expresión>;

En el siguiente ejemplo se muestran dos formas de implementar la función en VHDL:




```
--Ejemplo declaraciones concurrentes
--asignadas a una señal
--Autor:
--Fecha:

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity logico is

port(
        A,B,C,D: in std_logic ;
        x,y,z: out std_logic  );

end;

architecture booleana of logico is
begin
        x <= (A or ((A nand B) and (B nor C)));
        y <= ((A nand B) and (B nor C)) and ((B nor C) or (C and d));
        z <= ((B nor C) or (C and d)) nor D;

end booleana;
```

```
--Ejemplo declaraciones concurrentes
--asignación concurrente
--Autor: JLB
--Fecha:
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity logico is

port(
        A,B,C,D: in std_logic ;
        X,Y,Z: out std_logic  );

end;

architecture funciona of logico is
        signal X0,Y0,Z0,X1,Y1 : std_logic;  --señales intermedias
begin
        --primer nivel
        X0 <= (A nand B);
        Y0 <= (B nor C);
        Z0 <= (C and D);
        --segundo nivel
        X1 <= (X0 and Y0);
        Y1 <= (Y0 or Z0);
        --Salida
        X <= (A or X1);
        Y <= (X1 and Y1);
        Z <= (X1 nor D);
end funciona;
```

En los ejemplos notamos que la declaración intermedia de las señales es similar a las declaraciones del puerto en la entidad, excepto que no se especifica el modo (in, out, inout). Las señales intermedias deben ser declaradas en el cuerpo de la arquitectura porque no tienen enlace al mundo externo y por eso

no aparecen en la declaración de entidad. Aunque el uso de señales intermedias no es obligatorio, su uso trae consigo algunos beneficios. Primero, el uso de señales intermedias es la norma para muchos modelos VHDL. Conforme los circuitos se hacen más complejos, existen ocasiones en que el uso de señales intermedias se hace necesario. Segundo, el uso de éstas señales nos permite modelar circuitos digitales de una manera más fácil y, además, no implica que el hardware generado sea más complicado.

- *Las descripciones simples de circuitos tienen la ventaja de ser más fácilmente entendidos y sintetizados. Pero lo más importante, es que un modelo simple en VHDL no tiene relación con la longitud del código.*

Asignación condicional

La sintaxis de la asignación condicional se muestra a continuación:

```
<target> <=   <expression> when <condition> else
              <expression> when <condition> else
              <expression>;
```

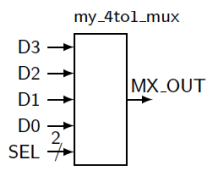
La mejor forma de entender ésta asignación es a través de un ejemplo. Consideremos la descripción en VHDL de la misma función del ejemplo anterior:

```
architecture f3_3 of my_ckt_f3 is
begin
F3 <= '1' when (L= '0' AND M = '0' AND N = '1') else
      '1' when (L= '1' AND M = '1') else
      '0';
end f3_3;
```

Esta descripción no implica una mejora sobre la anterior, en ella existe una función (target) y varias expresiones y condiciones. Si ninguna de las primeras condiciones se cumple, la última es la que se asigna a la función (target).

Es un ejemplo ilustrativo de que podemos usar la asignación concurrente o condicional para describir un circuito, sin embargo, existen usos más inteligentes de la asignación condicional; por ejemplo en la descripción de un multiplexor:

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_4t1_mux is
port(D3,D2,D1,D0 : in std_logic;
     SEL: in std_logic_vector(1 downto 0);
     MX_OUT : out std_logic);
end my_4t1_mux;
-- architecture
architecture mux4t1 of my_4t1_mux is
begin
    MX_OUT <= D3 when (SEL = "11") else
              D2 when (SEL = "10") else
              D1 when (SEL = "01") else
              D0 when (SEL = "00") else
              '0';
end mux4t1;
```




```
-------------------------------------------------------------------
-- entity and architecture of 4:1 Multiplexor implemented using
-- conditional signal assignment. The conditions access the
-- individual signals of the SEL bundle in this model.
-------------------------------------------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_4t1_mux is
port    (D3,D2,D1,D0   : in  std_logic;
         SEL           : in  std_logic_vector(1 downto 0);
         MX_OUT        : out std_logic);
end my_4t1_mux;
-- architecture
architecture mux4t1 of my_4t1_mux is
begin
    MX_OUT <= D3 when (SEL(1) = '1' and SEL(0) ='1') else
              D2 when (SEL(1) = '1' and SEL(0) ='0') else
              D1 when (SEL(1) = '0' and SEL(0) ='1') else
              D0 when (SEL(1) = '0' and SEL(0) ='0') else
              '0';
end mux4t1;
```

En el ejemplo, ambas descripciones utilizan la asignación condicional, sin embargo, la diferencia está en usar señales agrupadas (ilustración de la izquierda) o señales individuales (ilustración de la derecha).

Algunas notas respecto al ejemplo:

- En general, se usan comillas dobles para valores asociados con múltiples señales (ilustración izquierda) y comillas sencillas para valores asociados con señales individuales.
- Podemos acceder a señales individuales aunque hayan sido declaradas como grupo de señales (std_logic_vector), mediante una sintaxis especial (por ejemplo: **SEL(1)**).
- La solución utiliza el operador relacional "=", sin embargo, como se ilustra en la figura, existen seis tipos en VHDL.

Asignación selectiva

El tercer tipo de asignación de una señal es la asignación selectiva, cuya sintaxis se muestra a continuación:

```
with <choose_expression> select
    target <= <expression> when <choices>,
              <expression> when <choices>;
```

- Como en la asignación condicional, éste tipo de asignación también tiene sólo un operador de asignación.

Nuevamente describamos la función de los ejemplos anteriores, pero ahora usando este tipo de asignación:

$$F3 = \overline{L} \cdot \overline{M} \cdot N + L \cdot M$$

- Notemos el uso de la sentencia when others al final de la descripción.

```
architecture Behavioral of test_examples is
--Describe the function F3=L'M'N + LM using selecting model
begin
   with ((not(L) and not(M) and N)or(L and M)) select
   F3 <= '1' when '1',
         '0' when '0',
         '0' when others;
end Behavioral;
```

Consideremos nuevamente la descripción del multiplexor, para darnos cuenta del uso más inteligente de la asignación selectiva:

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_4t1_mux is
port (D3,D2,D1,D0 : in  std_logic;
      SEL         : in  std_logic_vector(1 downto 0);
      MX_OUT      : out std_logic);
end my_4t1_mux;
-- architecture
architecture mux4t1_2 of my_4t1_mux is
begin
with SEL select
    MX_OUT <= D3  when "11",
              D2  when "10",
              D1  when "01",
              D0  when "00",
              '0' when others;
end mux4t1_2;
```

- El único requisito para el uso de la sentencia when others es que se ubique al final de la sentencia select.

Otro ejemplo, muestra el uso del operador "|" como carácter de selección en la sección de opciones:

```vhdl
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_sz_ckt is
    port ( D_IN   : in  std_logic_vector(3 downto 0);
           SX_OUT : out std_logic_vector(2 downto 0));
end my_sz_ckt;
-- architecture
architecture spec_dec of my_sz_ckt is
begin
   with D_IN select
SX_OUT <= "100" when "0000"|"0001"|"0010"|"0011",
          "010" when "0100"|"0101"|"0110"|"0111"|"1000"|"1001",
          "001" when "1010"|"1011"|"1100"|"1101"|"1110"|"1111",
          "000" when others;
end spec_dec;
```

range of D_IN	SZ_OUT
0000 → 0011	100
0100 → 1001	010
1010 → 1111	001
unknown value	000

Ahora describamos la función de los ejemplos anteriores, pero usando este tipo de asignación:

```vhdl
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_ckt_f3 is
port ( L,M,N : in  std_logic;
       F3    : out std_logic);
end my_ckt_f3;
-- architecture
architecture f3_8 of my_ckt_f3 is
signal t_sig : std_logic_vector(2 downto 0); -- local bundle
begin
t_sig <= (L & M & N); -- concatenation operator

with (t_sig) select
    F3 <= '1' when "001" | "110" | "111",
          '0' when others;
end f3_8;
```

Proceso

Existen diferentes y variadas maneras de solucionar el mismo problema, entonces sólo queda al diseñador escribir su diseño de la manera más clara y dejar a la herramienta de síntesis ordenar los detalles.

La sentencia process es una herramienta que nos permite ejecutar un cierto número de instrucciones de manera secuencial, sin embargo, no debemos olvidar que la sentencia (process), en sí misma, es concurrente.

Estilos de programación

En la práctica, los circuitos más complejos, descritos en VHDL, usan una mezcla de los tres primeros estilos. El método estructural es esencialmente un método que combina un conjunto de modelos VHDL existentes, es por esto que, en lugar de ser un método de diseño, es más una facilidad para interconectar módulos existentes.

Flujo de datos

En las arquitecturas de flujo de datos se encuentran las sentencias concurrentes descritas anteriormente (asignación concurrente, condicional y selectiva). Por lo tanto, los ejemplos descritos anteriormente pertenecen a las arquitecturas diseñadas con estilo de flujo de datos. En estos ejemplos, podemos prácticamente "observar" el flujo de los datos por el circuito. Este tipo de modelado funciona bien para circuitos pequeños y relativamente simples. Pero, para circuitos más complejos, es mejor cambiar a los modelos funcionales.

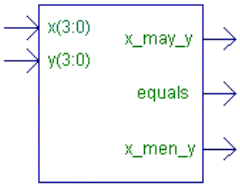
Funcional

El corazón de este estilo de programación es la sentencia process. El principal punto a recordar es que el cuerpo del proceso se compone de sentencias secuenciales. Es importante resaltar que la ejecución de una sentencia en el estilo funcional es controlada por las señales que aparecen en "sensivity_List". Por el contrario, las sentencias que aparecen en el modelo de flujo de datos, son re-evaluadas cada vez que las señales de entrada cambian. Esto es una diferencia funcional, no sólo una diferencia de apariencia.

```
-- this is my first process
my_label: process(sensitivity_list) is
   <item_declaration>
begin
   <sequential_statements>
end process my_label;
```

La sentencia process debería ser considerada como una manera que el diseñador tiene a su disposición para ejecutar una serie de sentencias secuenciales, sin olvidar que la sentencia process es en sí una sentencia concurrente.

Ejemplo: Describir el comparador mostrado en la figura, usando el modelo funcional.




```
-- Comparador de de magnitud de 4 bits
-- Declaración secuencial (if-then)
-- Autor:
-- Fecha:
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity comparador is

port(
        x: in std_logic_vector ( 3 downto 0 );
        y: in std_logic_vector ( 3 downto 0 );
        g: out std_logic ;
        e: out std_logic ;
        l: out std_logic  );

end;

architecture funcion of comparador is
begin
        process (x,y)
        begin
                if (x = y) then
                e <= '1';
                g <= '0';
                l <= '0';
                elsif (x > y) then
                g <= '1';
                e <= '0';
                l <= '0';
                else
                l <= '1';
                e <= '0';
                g <= '0';
                end if;
        end process;
end funcion;
```

- Sentencias secuenciales: La ejecución de una sentencia secuencial se inicia cuando ocurre un cambio en la lista sensitiva. Debemos recordar que la sentencia process es concurrente, debemos tomar ventaja de esta facilidad, para simplificar nuestras descripciones de circuitos. Existen tres tipos de sentencias secuenciales:
  - Sentencia de asignación de una señal " <= "
  - Sentencia "if", se utiliza para provocar una bifurcación en el flujo de ejecución de las sentencias secuenciales. Es el equivalente secuencial de la asignación condicional de una señal. Su forma general es:

```
if (condition) then
    <statements>
elsif (condition) then
    <statements>
else
    <statements>
end if;
```

    - Algunas notas acerca de la sintaxis:
      - Los paréntesis son opcionales, sin embargo, deberían incluirse para aumentar el entendimiento del código VHDL.
      - La sentencia "else" final no se asocia con un "then" final, si ninguna de las sentencias anteriores son ciertas, el else final se ejecuta. Ésto garantiza que al menos una de las secuencias es ejecutada.
      - La sentencia else es opcional, sin embargo, no incluirla representa la posibilidad de que ninguna de las secuencias sea ejecutada.
      - Ejemplo: escribir el código VHDL, usando la sentencia if, que describa la función: F(A,B,C) = AB'C'+BC.

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_ex is
port    (A,B,C : in  std_logic;
         F_OUT : out std_logic);
end my_ex;
-- architecture
architecture silly_example of my_ex is
begin
    proc1: process(A,B,C) is
    begin
        if (A = '1' and B = '0' and C = '0') then
            F_OUT <= '1';
        elsif (B = '1' and C = '1') then
            F_OUT <= '1';
        else
            F_OUT <= '0';
        end if;
    end process proc1;
end silly_example;
```

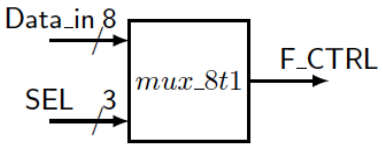
Una solución

```
-- architecture
architecture bad_example of my_ex_7 is
begin
    proc1: process(A,B,C)
    begin
        if (A='1' and B='0' and C='0') or (B='1' and C='1') then
            F_OUT <= '1';
        else
            F_OUT <= '0';
        end if;
    end process proc1;
end bad_example;
```

Una mejor solución

La sentencia process puede ir precedida con una etiqueta opcional. Una etiqueta debería siempre incluirse para identificar el proceso como una forma de auto descripción. Un uso más inteligente de la sentencia if se presenta en el siguiente ejemplo:

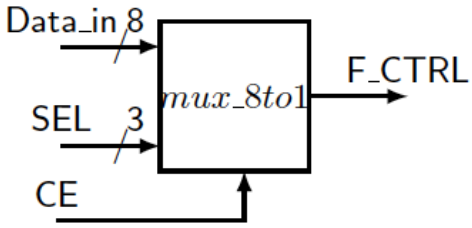
Ejemplo: Escribir el código VHDL que describe el MUX de la figura.




```vhdl
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity mux_8t1 is
    port ( Data_in : in  std_logic_vector (7 downto 0);
                SEL : in  std_logic_vector (2 downto 0);
             F_CTRL : out std_logic);
end mux_8t1;
-- architecture
architecture mux_8t1_arc of mux_8t1 is
begin
   my_mux: process (Data_in,SEL)
   begin
      if     (SEL = "111") then F_CTRL <= Data_in(7);
      elsif  (SEL = "110") then F_CTRL <= Data_in(6);
      elsif  (SEL = "101") then F_CTRL <= Data_in(5);
      elsif  (SEL = "100") then F_CTRL <= Data_in(4);
      elsif  (SEL = "011") then F_CTRL <= Data_in(3);
      elsif  (SEL = "010") then F_CTRL <= Data_in(2);
      elsif  (SEL = "001") then F_CTRL <= Data_in(1);
      elsif  (SEL = "000") then F_CTRL <= Data_in(0);
      else   F_CTRL <= '0';
      end if;
   end process my_mux;
end mux_8t1_arc;
```

En este ejemplo se puede borrar la sentencia elsif final y colocar la asignación final en la sentencia else. Sin embargo ésto no se considera una buena práctica en VHDL y debe ser evitada.

Un ejemplo más: Escribir el código VHDL para describir el funcionamiento del MUX de la figura.




```vhdl
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity mux_8to1_ce is
    port ( Data_in : in  std_logic_vector (7 downto 0);
                SEL : in  std_logic_vector (2 downto 0);
                 CE : in  std_logic;
             F_CTRL : out std_logic);
end mux_8to1_ce;
-- architecture
architecture mux_8to1_ce_arch of mux_8to1_ce is
begin
   my_mux: process (Data_in,SEL,CE)
   begin
      if (CE = '0') then
          F_CTRL <= '0';
      else
         if     (SEL = "111") then F_CTRL <= Data_in(7);
         elsif  (SEL = "110") then F_CTRL <= Data_in(6);
         elsif  (SEL = "101") then F_CTRL <= Data_in(5);
         elsif  (SEL = "100") then F_CTRL <= Data_in(4);
         elsif  (SEL = "011") then F_CTRL <= Data_in(3);
         elsif  (SEL = "010") then F_CTRL <= Data_in(2);
         elsif  (SEL = "001") then F_CTRL <= Data_in(1);
         elsif  (SEL = "000") then F_CTRL <= Data_in(0);
         else   F_CTRL <= '0';
         end if;
      end if;
   end process my_mux;
end mux_8to1_ce_arch;
```

- En la sentencia case, sólo se ejecuta un conjunto de sentencias secuenciales por cada ejecución de la sentencia case. Es el equivalente secuencial de la sentencia de with. Sin embargo, la sentencia case es una sentencia secuencial que se encuentra en el cuerpo de process mientras que with es una forma de asignación de una señal concurrente.

```
case (expression) is
    when choices =>
        <sequential statements>
    when choices =>
        <sequential statements>
    when others =>
        <sequential statements>
end case;
```

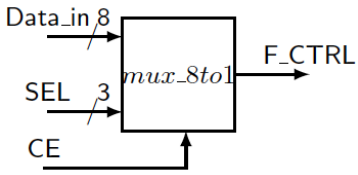
Ejemplo: Escribir el código para implementar: F(A,B,C) = AB'C'+BC. Primero, procedemos a completar la función con los términos faltantes:F(A,B,C)=AB'C'+BC(A+A')= AB'C'+ABC+A'BC.

En VHDL podemos usar la facilidad de "no importa", sin embargo, se debe evitar a toda costa, porque algunas herramientas de síntesis y algunos simuladores no pueden manejarla bien. Sólo mencionaremos cómo se haría en el ejemplo de abajo:

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_example is
port (A,B,C : in  std_logic;
      F_OUT : out std_logic);
end my_example;
-- architecture
architecture my_soln_exam of my_example is
    signal ABC: std_logic_vector(2 downto 0);
begin
    ABC <= A & B & C; -- group signals for case statement
    my_proc: process (ABC)
    begin
        case (ABC) is
        when "100"  => F_OUT <= '1';
        when "011"  => F_OUT <= '1';
        when "111"  => F_OUT <= '1';
        when others => F_OUT <= '0';
        end case;
    end process my_proc;
end my_soln_exam;
```

```
architecture my_soln_exam2 of my_example is
    signal ABC: std_logic_vector(2 downto 0);
begin
    ABC <= A & B & C; -- group signals for case statement
    my_proc: process (ABC)
    begin
        case (ABC) is
        when "100"  => F_OUT <= '1';
        when "-11"  => F_OUT <= '1';
        when others => F_OUT <= '0';
        end case;
    end process my_proc;
end my_soln_exam2;
```

Ejemplo: Volvamos a escribir el código que implemente el multiplexor de la figura:



Lo interesante de la solución mostrada en el código de la derecha, es que la sentencia case está embebida dentro de la sentencia if, dicho de otra manera, está "anidada". Esta facilidad es lo que hace al modelo funcional más poderoso que el de flujo de datos.

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity mux_8to1_ce is
    port ( Data_in : in  std_logic_vector (7 downto 0);
                SEL : in  std_logic_vector (2 downto 0);
                 CE : in  std_logic;
             F_CTRL : out std_logic);
end mux_8to1_ce;
-- architecture
architecture my_case_ex of mux_8to1_ce is
begin
   my_mux: process (SEL,Data_in,CE)
   begin
      if (CE = '1') then
         case (SEL) is
            when "000"  =>  F_CTRL <= Data_in(0);
            when "001"  =>  F_CTRL <= Data_in(1);
            when "010"  =>  F_CTRL <= Data_in(2);
            when "011"  =>  F_CTRL <= Data_in(3);
            when "100"  =>  F_CTRL <= Data_in(4);
            when "101"  =>  F_CTRL <= Data_in(5);
            when "110"  =>  F_CTRL <= Data_in(6);
            when "111"  =>  F_CTRL <= Data_in(7);
            when others => F_CTRL <= '0';
         end case;
      else
          F_CTRL <= '0';
      end if;
   end process my_mux;
end my_case_ex;
```

Algunas recomendaciones antes de concluir con el modelo funcional:

- Si no entendemos las entradas y salidas de VHDL al nivel bajo, el circuito no se sintetizará apropiadamente.
- Mantenga sus modelos VHDL lo más simple posibles, particularmente en la sentencia process.
- En VHDL, la mejor manera es mantener las sentencias process centradas alrededor de una función sencilla y tener varias sentencias process que se comuniquen con cada una.

Algunas notas acerca de los operadores que nos serán de utilidad. Los operadores en VHDL se agrupan en diferentes tipos, según muestra la tabla:

Operator type							
exponential, absolute value	**	abs					
logical	and	or	nand	nor	xor	xnor	not
multip., division, module, remainer	*	/	mod	rem			
addition, subtraction	+	−					
identity, negation	+	−					
concatenate, shift, rotate	&	sll	srl	sla	sra	rol	ror
relational	=	/=	<	<=	>	>=	

Algunos de ellos requieren de consideraciones especiales, por ejemplo, los operadores de desplazamiento requieren de las librerías *ieee.numeric_std* o *ieee.numeric_bit*.

Ejemplos de estos operadores, se muestran en la tabla:

Operator	Name		Example	Result
logical	sll	shift left logical	result <= "10010101" sll 2	"01010100"
	srl	shift right logical	result <= "10010101" srl 3	"00010010"
arithmetic	sla	shift left arithmetic	result <= "10010101" sla 3	"10101111"
	sra	shift right arithmetic	result <= "10010101" sra 2	"11100101"
rotate	rol	rotate left	result <= "101000" rol 2	"100010"
	ror	rotate right	result <= "101001" ror 2	"011010"

La diferencia entre los desplazamientos lógicos y aritméticos, es que en los aritméticos, el bit de signo nunca cambia y el bit que se alimenta al final puede ser diferente.

En la tabla de la derecha, se muestran otros operadores:

Operator		Name	Comment
addition	+	addition	
	−	subtraction	
unary	+	identity	
	−	negation	
multiplying	*	multiplication	
	/	division	often limited to powers of two
	mod	modulus	can operate only on specific types
	rem	remainder	can operate only on specific types
other	**	exponentiation	often limited to powers of two
	abs	absolute value	
	&	concatenation	can operate only on specific types

Otra recomendación es que no se debe emplear mucho esfuerzo en memorizar la sintaxis de VHDL, mejor se recomienda que siempre que estés describiendo un modelo, tengas a la mano un "acordeón".

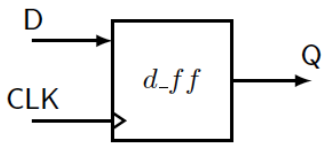
Desarrollar un verdadero entendimiento de VHDL, es lo que te convertirá en un buen diseñador de hardware. La habilidad para memorizar la sintaxis de VHDL nos proporciona "almost nothing".

Circuitos secuenciales

Hasta ahora todo el estudio y los ejemplos de VHDL han sido elaborados describiendo circuitos combinacionales, sin embargo, cuando usamos elementos de almacenamiento en el diseño digital, es necesario describir los circuitos secuenciales. En algunos casos es deseable usar el modelo de *flujo de datos* para describir elementos de memoria en VHDL, pero es más fácil usar el modelo *funcional*.

El estudio de los elementos de memoria comienza con los flip-flops tipo D, disparados por flancos.

```
-----------------------------------
-- Model of a simple D Flip-Flop --
-----------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity d_ff is
    port ( D, CLK : in  std_logic;
           Q :      out std_logic);
end d_ff;
-- architecture
architecture my_d_ff of d_ff is
begin
    dff:  process(CLK)
    begin
        if (rising_edge(CLK))      then
--or     if (CLK'event and CLK='1') then
            Q <= D;
        end if;
    end process dff;
end my_d_ff;
```

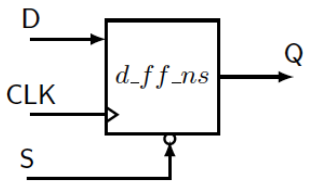


Podemos notar que la función rising_edge(CLK) también puede ser reemplazada por la popular sentencia: CLK' event and CLK='1';

El método usado para asegurar algún estado del flip-flop en caso de que la condición listada en la sentencia if no se cumpla, es usando else. Un "warning" generado por los sintetizadores de VHDL es la notificación de que se ha generado un *latch*. La causa del problema es que no se ha proporcionado explícitamente un estado de salida para todas las posibles condiciones de entrada. Esto se arregla colocando otra entrada que esté en sincronía con la señal de reloj.

La entrada "S" permite la operación del flip-flop en el pulso de subida del reloj del sistema. En el pulso de subida del reloj, la entrada "S" toma precedencia sobre la entrada "D" porque se revisa primero el estado de la entrada "S" en lugar de examinar la entrada "D". La entrada "D" se transfiere a la salida sólo en el pulso de subida del reloj y sólo si la entrada "S" está en nivel bajo.

Aquí el diagrama lógico y el código VHDL:




```vhdl
-------------------------------------------------------------
-- RET D Flip-flop model with active-low synchronous set input. --
-------------------------------------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity d_ff_ns is
    port (  D,S :  in  std_logic;
            CLK :  in  std_logic;
              Q :  out std_logic);
end d_ff_ns;
-- architecture
architecture my_d_ff_ns of d_ff_ns is
begin
   dff:  process (CLK)
   begin
      if (rising_edge(CLK)) then
         if (S = '0') then
            Q <= '1';
         else
            Q <= D;
         end if;
      end if;
   end process dff;
end my_d_ff_ns;
```

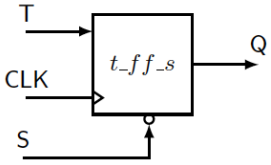
Con fines de comparación entre una entrada asíncrona y una síncrona, describiremos la misma estructura, pero con una entrada "R" (reset), que es independiente del reloj. La prioridad se determina poniendo "R" como la primera condición en la sentencia if. Asimismo, se utilizará la función falling_edge(), para hacer que el flip-flip se active con un pulso de bajada.

```vhdl
-------------------------------------------------------------
-- FET D Flip-flop model with active-high asynchronous reset input. --
-------------------------------------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity d_ff_r is
    port (   D,R :  in  std_logic;
             CLK :  in  std_logic;
               Q :  out std_logic);
end d_ff_r;
-- architecture
architecture my_d_ff_r of d_ff_r is
begin
   dff:  process (R,CLK)
   begin
      if (R = '1') then
         Q <= '0';
      elsif (falling_edge(CLK)) then
         Q <= D;
      end if;
   end process dff;
end my_d_ff_r;
```

Las formas generales a estas soluciones son consideradas como los patrones para las entradas síncronas y asíncronas en varias referencias de VHDL.

Otro ejemplo consiste en describir la función de un flip-flop tipo T, sin embargo ahora consideraremos la entrada "S" como una señal asíncrona que es activa en nivel bajo, para establecer la salida del flip-flop tipo T.

```vhdl
-------------------------------------------------------------
-- RET T Flip-flop model with active-low asynchronous set input. --
-------------------------------------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity t_ff_s is
    port ( T,S,CLK :  in  std_logic;
                 Q :  out std_logic);
end t_ff_s;
-- entity
architecture my_t_ff_s of t_ff_s is
   signal t_tmp : std_logic; -- intermediate signal declaration
begin
   tff:  process (S,CLK)
   begin
      if (S = '0') then
         t_tmp <= '1';
      elsif (rising_edge(CLK)) then
         t_tmp <= T XOR t_tmp; -- temp output assignment
      end if;
   end process tff;

   Q <= t_tmp; -- final output assignment
end my_t_ff_s;
```



Las señales que son declaradas como salidas no pueden aparecer en el lado derecho del operador de asignación. Para solventar este problema, hacemos uso de las señales intermedias, las cuales pueden ser usadas como entradas o salidas. Otra solución para evitar este problema es haciendo uso de la especificación de "buffer", pero nunca debemos usar ésto en VHDL. El buen uso de VHDL es cuando recurrimos a las señales intermedias.

Los flip-flops tipo "D" son los elementos de almacenamiento preferidos en VHDL. Si no hay alguna razón específica para utilizar otro tipo de flip-flop que no sea del tipo "D", *no deberíamos hacerlo*.

En VHDL, como en otros lenguajes, hay maneras específicas de hacer cosas y estas cosas deben ser siempre realizadas de estas maneras.

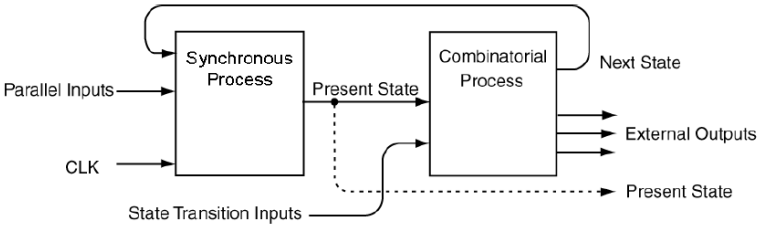
Estructural

...

Máquinas de estados finitos

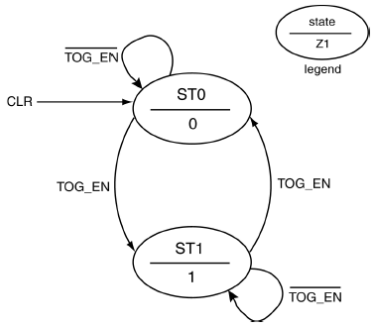
Las Máquinas de Estados Finitos(FSM) son abstracciones matemáticas utilizadas para resolver problemas de automatización electrónica, diseñar protocolos de comunicación, análisis y otras aplicaciones de ingeniería.

La versatilidad del comportamiento en VHDL evita la necesidad de hacer largos diseños en papel. En la figura podemos apreciar el modelo FSM usado en VHDL; "parallel Inputs" se utilizan para representar las entradas que actúan sobre cada elemento de almacenamiento (enables, presets, clears, etcétera), "State transition" incluyen entradas externas que controlan las transiciones de estado, "Present State" son utilizadas por el "Combinatorial Process" para decodificar el siguiente estado "Next State" y la salida "External Outputs".

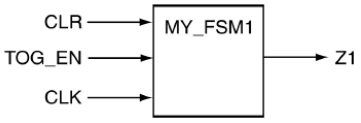


Existen diferentes métodos para describir una FSM, sin embargo los más usados son el dependiente e independiente PS/NS (Present State/Next State). El modelo representado en la figura es el dependiente, es más claro y nos permitirá entender fácilmente el modelo independiente.

Ejemplo: Usando un estilo de codificación dependiente, describir el diagrama de estados mostrado en la figura.



Es recomendable dibujar el diagrama de bloques.



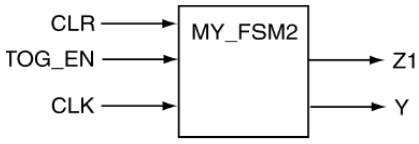

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_fsm1 is
    port (  TOG_EN  : in  std_logic;
            CLK,CLR : in  std_logic;
                 Z1 : out std_logic);
end my_fsm1;
-- architecture
architecture fsm1 of my_fsm1 is
   type state_type is (ST0,ST1);
   signal PS,NS : state_type;
begin
   sync_proc: process(CLK,NS,CLR)
   begin
     -- take care of the asynchronous input
     if (CLR = '1') then
        PS <= ST0;
     elsif (rising_edge(CLK)) then
        PS <= NS;
     end if;
   end process sync_proc;

   comb_proc: process(PS,TOG_EN)
   begin
      Z1 <= '0';        -- pre-assign output
      case PS is
         when ST0 =>    -- items regarding state ST0
            Z1 <= '0';  -- Moore output
            if (TOG_EN = '1') then NS <= ST1;
            else  NS <= ST0;
            end if;
         when ST1 =>    -- items regarding state ST1
            Z1 <= '1';  -- Moore output
            if (TOG_EN = '1') then NS <= ST0;
            else  NS <= ST1;
            end if;
         when others => -- the catch-all condition
            Z1 <= '0';  -- arbitrary; it should never
            NS <= ST0;  -- make it to these two statements
      end case;
   end process comb_proc;
end fsm1;
```

Algunas notas respecto al ejemplo anterior:

- Se ha declarado un tipo especial (type) llamado "state_type" para representar los estados de la FSM. Esta variable ha sido creada y no forma parte de los tipos nativos de VHDL.
- El elemento de almacenamiento está asociado solamente con la señal "PS" y no está especificado para cada combinación posible de las entradas.
- Existen dos procesos: El proceso síncrono que maneja las señales de "Reset" y la asignación de un nuevo estado al llegar la señal de reloj, el proceso combinacional maneja las salidas no utilizadas por el proceso síncrono.
- Debido a que los dos procesos operan de manera concurrente, puede considerarse que operan de manera concatenada, un cambio en el circuito combinacional, afecta al circuito secuencial y así sucesivamente.
- Este es el formato estándar para el estilo de código dependiente PS/NS.
- La salida de la máquina de Moore depende sólo del estado presente. La variable "Z1" está dentro de la sentencia when y fuera del if. Esto se debe a que las salidas de la máquina solamente son función de los estados, no de las entradas externas.
- La salida "Z1" es establecida a un valor, como primer paso del proceso combinacional. Esto previene señales de Z1 inesperadas.

Las variables de estados son representadas internamente y no necesitan ser precisas porque no son proporcionadas como una salida.

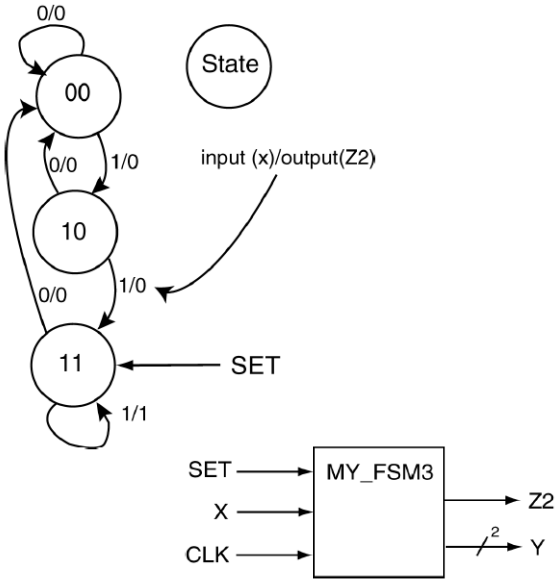
En algunos diseños, las variables de estado se proporcionan como salida como indica el diagrama y el código adjunto.



Este código sólo difiere del anterior en la declaración de la entidad donde se incluye la variable "Y" para la asignación del estado, además de la selección de señal "Y" basada en la condición de la variable de estado. Podemos notar que existen tres asignaciones concurrentes: los dos procesos y la asignación en la parte final del código.

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_fsm2 is
   port (   TOG_EN : in  std_logic;
          CLK,CLR : in  std_logic;
             Y,Z1 : out std_logic);
end my_fsm2;
-- architecture
architecture fsm2 of my_fsm2 is
   type state_type is (ST0,ST1);
   signal PS,NS : state_type;
begin
   sync_proc: process(CLK,NS,CLR)
   begin
      if (CLR = '1') then
         PS <= ST0;
      elsif (rising_edge(CLK)) then
         PS <= NS;
      end if;
   end process sync_proc;

   comb_proc: process(PS,TOG_EN)
   begin
      Z1 <= '0';
      case PS is
         when ST0 =>        -- items regarding state ST0
            Z1 <= '0';      -- Moore output
            if (TOG_EN = '1') then NS <= ST1;
            else  NS <= ST0;
            end if;
         when ST1 =>        -- items regarding state ST1
            Z1 <= '1';      -- Moore output
            if (TOG_EN = '1') then NS <= ST0;
            else  NS <= ST1;
            end if;
         when others =>     -- the catch-all condition
            Z1 <= '0';      -- arbitrary; it should never
            NS <= ST0;      -- make it to these two statements
      end case;
   end process comb_proc;

   -- assign values representing the state variables
   with PS select
      Y <= '0' when ST0,
           '1' when ST1,
           '0' when others;
end fsm2;
```

Otro ejemplo:

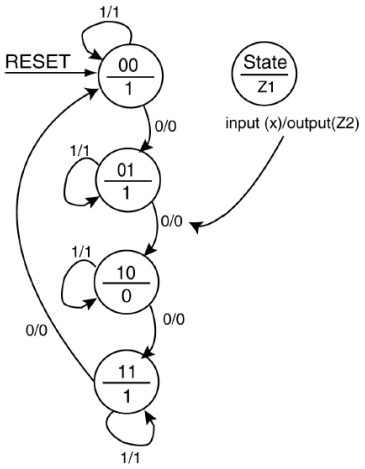


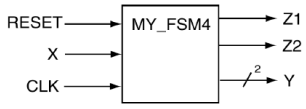
El diagrama muestra que es una máquina tipo Mealy con una salida externa y entrada externa. Como tiene tres estados, la solución requiere al menos dos variables de estado.

```vhdl
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_fsm3 is
    port ( X,CLK,SET : in  std_logic;
                   Y : out std_logic_vector(1 downto 0);
                  Z2 : out std_logic);
end my_fsm3;
-- architecture
architecture fsm3 of my_fsm3 is
   type state_type is (ST0,ST1,ST2);
   signal PS,NS : state_type;
begin
   sync_proc: process(CLK,NS,SET)
   begin
      if (SET = '1') then
         PS <= ST2;
      elsif (rising_edge(CLK)) then
         PS <= NS;
      end if;
   end process sync_proc;

   comb_proc: process(PS,X)
   begin
      Z2 <= '0';           -- pre-assign FSM outputs
      case PS is
         when ST0 =>       -- items regarding state ST0
            Z2 <= '0';     -- Mealy output always 0
            if (X = '0') then NS <= ST0;
            else  NS <= ST1;
            end if;
         when ST1 =>       -- items regarding state ST1
            Z2 <= '0';     -- Mealy output always 0
            if (X = '0') then NS <= ST0;
            else  NS <= ST2;
            end if;
         when ST2 =>       -- items regarding state ST2
            -- Mealy output handled in the if statement
            if (X = '0') then NS <= ST0; Z2 <= '0';
            else  NS <= ST2;  Z2 <= '1';
            end if;
         when others =>    -- the catch all condition
            Z2 <= '1'; NS <= ST0;
      end case;
   end process comb_proc;

   -- faking some state variable outputs
   with PS select
      Y <= "00" when ST0,
           "10" when ST1,
           "11" when ST2,
           "00" when others;
end fsm3;
```

Otro ejemplo:

```vhdl
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_fsm4 is
    port ( X,CLK,RESET : in  std_logic;
                     Y : out std_logic_vector(1 downto 0);
                 Z1,Z2 : out std_logic);
end my_fsm4;
-- architecture
architecture fsm4 of my_fsm4 is
   type state_type is (ST0,ST1,ST2,ST3);
   signal PS,NS : state_type;
begin
   sync_proc: process(CLK,NS,RESET)
   begin
     if (RESET = '1') then  PS <= ST0;
     elsif (rising_edge(CLK)) then  PS <= NS;
     end if;
   end process sync_proc;

   comb_proc: process(PS,X)
   begin
      -- Z1: the Moore output; Z2: the Mealy output
      Z1 <= '0';    Z2 <= '0'; -- pre-assign the outputs
      case PS is
         when ST0 =>    -- items regarding state ST0
            Z1 <= '1';  -- Moore output
            if (X = '0') then NS <= ST1; Z2 <= '0';
            else  NS <= ST0; Z2 <= '1';
            end if;
         when ST1 =>    -- items regarding state ST1
            Z1 <= '1';  -- Moore output
            if (X = '0') then NS <= ST2; Z2 <= '0';
            else  NS <= ST1; Z2 <= '1';
            end if;
         when ST2 =>    -- items regarding state ST2
            Z1 <= '0';  -- Moore output
            if (X = '0') then NS <= ST3; Z2 <= '0';
            else  NS <= ST2; Z2 <= '1';
            end if;
         when ST3 =>    -- items regarding state ST3
            Z1 <= '1';  -- Moore output
            if (X = '0') then NS <= ST0; Z2 <= '0';
            else  NS <= ST3;  Z2 <= '1';
            end if;
         when others => -- the catch all condition
            Z1 <= '1';  Z2 <= '0';  NS <= ST0;
      end case;
   end process comb_proc;

   with PS select
      Y <= "00" when ST0,
           "01" when ST1,
           "10" when ST2,
           "11" when ST3,
           "00" when others;
end fsm4;
```





Tengamos en mente que un ingeniero raramente se basa en recetas de cocina. Para las FSM, el ingeniero es el creador y probador del diagrama de estados.

Ambiente de desarrollo

Las Máquinas de Estados Finitos(FSM) son abstracciones matemáticas utilizadas para resolver problemas de automatización electrónica, diseñar protocolos de comunicación, análisis y otras aplicaciones de ingeniería.

En ISPLEVER

Crear un proyecto

- File->New Project
  - Project Name: "Nombre del proyecto"
  - Location: "Nombre de la carpeta"
  - Design Entry Type: VHDL
  - Next
    - Activar Show Obsolete Devices
    - Family: GAL Device
    - Device: GAL22V10D
    - Package type: 24PDIP
    - Next
      - Add Source: Podemos insertar un archivo existente o desarrollado con otra herramienta.
      - Next
        - Finish
- Ubicarse en Nombre del proyecto
- Botón derecho
- New
  - New: VHDL Module
  - OK
    - File Name:
    - Entity:
    - Architecture:
    - Port Name Direction MSB LSB
    - OK
  - Crea un archivo, editamos el código.

Compilar el proyecto

- Nos ubicamos en el archivo VHDL
- Damos doble "click" en Synplify Synthesize VHDL file

Generar el mapa de fusibles

- Nos ubicamos en el dispositivo
- Damos doble "click" en Create Fuse Map

Observar la asignación de pines en el dispositivo

- Nos ubicamos en el dispositivo
- Damos doble “click” en Chip Report

En ISE Xilinx (WebPack)

Crear un proyecto

- File->New Project
  - Name: “Nombre del proyecto” (Creará una nueva carpeta para el proyecto)
  - Location: “Nombre de la carpeta”
  - Working Directory: “Directorio de trabajo”
  - Top-level source type: HDL
  - Next
    - En esta ventana, tomar y llenar los datos descritos en la siguiente tabla, tomando en cuenta la tarjeta de desarrollo usada:

Development Board	**CoolRunner2**	**BASYS 2**	**NEXYS 2**	**PAPILIO ONE**
Family	CoolRunner2	Spartan3E	Spartan3E	Spartan3E
Device	XC2C256	XC3S100E	XC3S500E	XC3S500E
Package	TQ144	CP132	FG320	VQ100
Speed	-7	-4	-4	-4

    - Preferred Language: VHDL
    - Next
      - Finish
- Ubicarse en Nombre del proyecto
- Botón derecho
- New Source
  - VHDL Module
  - File name: “Nombre del archivo HDL”
  - Seleccionar “Add project”
  - Next
    - Entity name: “Nombre de la entidad”
    - Architecture: “Nombre de la arquitectura”
    - Port Name Direction BUS MSB LSB
    - Next
  - Crea un archivo, editamos el código.

*Cuando salvamos el proyecto nos indica si hay errores de sintaxis.

Compilar el proyecto (CPLD o FPGA)

- Nos ubicamos en el archivo VHDL.
- Damos doble “click” en “Implement Design”.

Cargar en el dispositivo (CPLD o FPGA)

- Entrar a “Design”.
- Nos ubicamos en el dispositivo.
- Botón derecho.
- New Source
  - Implementation Constraints File
  - File name: “Nombre del archivo”
  - Seleccionar “Add project”
  - Next
    - Finish
  - En ese archivo ligamos nuestro diseño a las señales de la tarjeta de desarrollo
    - NET “etiqueta” LOC = “Pin de la tarjeta”; (debe llevar las comillas y terminar con “;”)
      *Tomar esta información del manual de referencia o descargar el archivo UCF genérico.
- Volvemos a “Design”
- Seleccionamos el archivo fuente (esquemático o VHDL)
- En la ventana de Processes, damos doble “Click” en Implement Design
- Si todo está bien, debe ponerse en verde: “Synthesize”, “Implement” y “Generate”
- Para copiar el archivo del mapa de fusibles, hacemos lo siguiente:

Tarjeta de desarrollo	**CoolRunner2**	**BASYS 2 y NEXYS2**	**PAPILIO ONE**
Procedimiento	-Entrar a “Configure Target Device” -Conectamos la tarjeta -Luego, “Manage Configuration - - - -En la opción “Project”: *Damos “Click” en el icono “Launch Wizard” *Damos “OK” *Damos “Yes” *Abrimos el archivo “.jed” *Damos “OK” *Nos ubicamos en el icono de “Xilinx” y damos botón derecho *Seleccionamos “Program”	-Abrimos el software “Adept”. -Conectamos la tarjeta. -En la opción Connect aparece el nombre de la tarjeta. -La seleccionamos. -Después de conectarse, damos la opción “browse”. -Buscamos el archivo con el nombre que lo creamos y la extensión “.bit”. -Abrimos el archivo, aparece un mensaje “warning” y damos “yes”. -Damos “click” en “program”. -Aparece un mensaje “warning” damos “yes” -Verificamos en la tarjeta que el diseño funciona	Spartan3E

Generar el mapa de fusibles

- Nos ubicamos en el dispositivo
- Damos doble “click” en Create Fuse Map

Observar la asignación de pines en el dispositivo

- Nos ubicamos en el dispositivo
- Damos doble “click” en Chip Report

Bibliografía

Maxinez, D. G., & Alcalá Jara, J. (2007). *VHDL: el arte de programar sistemas digitales*. México: COMPAÑÍA EDITORIAL CONTINENTAL.

What is Programmable Logic? (s. f.). Recuperado 11 de marzo de 2013, a partir de http://www.xilinx.com/company/about/programmable.html