



VHDL Y FPGA

LENGUAJE VHDL

VHDL es un lenguaje de alto nivel que describe todas las características de circuitos electrónicos digitales de variada complejidad. El significado de las siglas VHDL es V de VHSIC (Circuitos Integrados de muy alta Velocidad – *Very High Speed Integrated Circuits*) y HDL de (Lenguaje de Descripción del Hardware – *Hardware Description Language*).

- El VHDL nace por iniciativa del Departamento de Defensa de EE.UU. a comienzos de los 80.
- En julio de 1983 tres compañías (Texas Instruments, IBM e Internetics) reciben el encargo de desarrollarlo.
- La primera versión fue publicada en agosto de 1985.
- En 1986 se convierte en un estándar del IEEE (IEEE 1076-1987).
- Posteriormente, con el estándar IEEE 1164 se le añaden nuevas características.

FPGA

FPGA (*Field Programmable Gate Array*), es un circuito de alta escala de integración.

MODELACIÓN DEL HARDWARE

Los comentarios que se hagan a las instrucciones en el programa deben comenzar con 2 guiones así: -- COMENTARIOS

Está compuesto por las LIBRERÍAS, la ENTIDAD y de la ARQUITECTURA.

LIBRERÍAS (*LIBRARY*): la librería estándar es siempre visible, si se requieren otras librerías se deben importar.

ENTIDAD (*Entity*): primero se define un bloque dándole un nombre y luego se especifica la interfaz del circuito declarando las líneas de entrada y de salida (llamadas puertos).

ARQUITECTURA (*Architecture*): es la descripción interna del circuito, describe lo que el módulo hace, o sea, la forma en que las salidas se relacionan con las entradas. Puede describirse de diferentes maneras (deferentes formas de modelar).



OPERADORES LÓGICOS

Operadores Lógicos	not	Not A
	and	(A and B)
	or	(A or B)
	xor	(A xor B)
	nand	(A nand B) o not (A and B)
	nor	(A nor B) o not (A or B)
	xnor	(A xnor B) o not (A xor B)

Ejemplos:

X <= not A and B;	-- $\overline{A} \times B$
Y <= not (A and B);	-- $\overline{(A \times B)}$
Z <= A nand B;	-- $\overline{(A \times B)}$

OPERADORES DE ASIGNACIÓN

Operador	Descripción
<=	Asignar valores a una señal X <= '1'; -- Asignar "1" a la señal X
:=	Asignar valores a una variable y/o constante. Establece valores iniciales. Y := "0010"; -- Asignar "0010" a la variable Y
=>	Asignar valores a los elementos de un vector individual. Z <= "1000"; -- Asignar "1" al LSB y a los demás "0" si la salida Z fue declarada como to. Asignar "1" al MSB y a los demás "0" si la salida Z fue declarada como downto. W <= (3 => '1', others => '0'); -- Asignar "1" al LSB y a los otros "0" si la salida Z fue declarada como to. Asignar "1" al MSB y a los otros "0" si la salida Z fue declarada como downto.

LIBRERÍAS – LIBRARY

Al declarar una librería, son necesarias dos líneas de código: una que contenga el nombre de la librería y otra con la sentencia USE, así:

library *Nombre de la librería*;
use *Nombre de la librería. Nombre del paquete. Parte del paquete*;

```

library IEEE;           ①
use IEEE.STD_LOGIC_1164.ALL;  ②
use IEEE.STD_LOGIC_ARITH.ALL; ③
use IEEE.STD_LOGIC_UNSIGNED.ALL; ④

```

- ① Nombre de la librería
- ② Librería IEEE. Paquete STD_LOGIC_1164. Todo el paquete;
- ③ Librería IEEE. Paquete STD_LOGIC_ARITH.ALL. Todo el paquete;
- ④ Librería IEEE. Paquete STD_LOGIC_UNSIGNED. Todo el paquete;

ENTIDAD – ENTITY

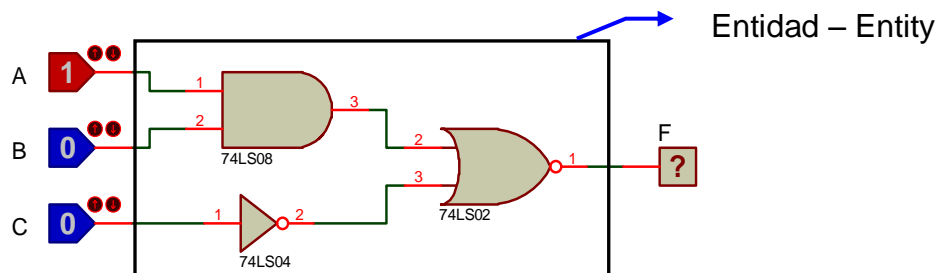
Se identifica al módulo como un bloque donde se definen los puertos de entrada y salida.

La entidad define el NOMBRE de los puertos, el MODO (entradas o salidas) y el TIPO (integer, bit, vector,...).

entity *Nombre_entidad* **is**

Port (NOMBRE de la señal: MODO y TIPO de dato;
NOMBRE de la señal: MODO y TIPO de dato);

end *Nombre_entidad*;



entity *compuertas* **is**

Port (A, B, C: **in** STD_LOGIC;
F: **out** STD_LOGIC);

end *compuertas*;

Diseño de Entidades mediante especificaciones individuales:

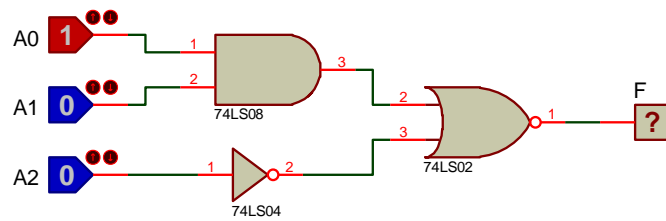
entity *compuertas* **is**

Port (A: **in** STD_LOGIC;
B: **in** STD_LOGIC;
C: **in** STD_LOGIC;
F: **out** STD_LOGIC);

end *compuertas*;

Diseño de Entidades mediante vectores:

Conjuntos de palabras de varios bits, $A = [A2, A1, A0]$ y se definen con la sentencia BIT_VECTOR.



entity compuertas_1 is

Port (A : in STD_LOGIC_VECTOR (2 downto 0);

F : out STD_LOGIC);

end compuertas_1;

Si el vector tiene el orden: $A = [A0, A1, A2]$ se define:

Port (A : in STD_LOGIC_VECTOR (0 to 2);

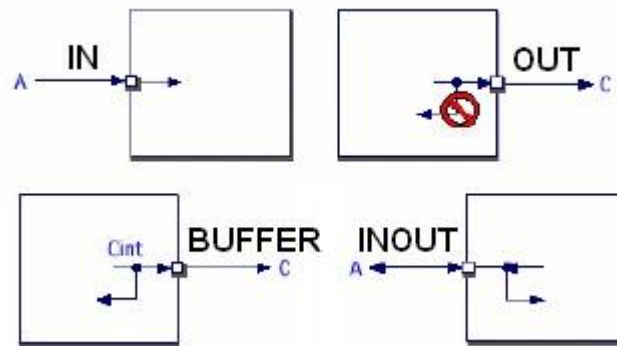
PUERTOS

Son los puntos de conexión entre una entidad y el medio exterior, se representan por entradas y salidas y requiere de un sentido ya sea como entrada, salida o bidireccional.

- Los identificadores son los nombres válidos para referir variables, constantes, señales, procesos, etc.
- No tienen longitud máxima.
- Puede contener caracteres de la 'A' a la 'Z', de la 'a' a la 'z', caracteres numéricos de '0' al '9' y el carácter subrayado '_'.
- No se diferencia entre mayúsculas y minúsculas (CONTADOR, contador y ConTadoR son el mismo identificador.)
- Debe empezar por un carácter alfabético, no puede terminar con un subrayado, ni puede tener dos subrayados seguidos.
- No puede usarse como identificador una palabra reservada.

MODOS

Modo	Descripción
IN	Las señales solo entran en la entidad y no salen. La señal puede ser leída pero no escrita
OUT	Las señales salen de la entidad y no pueden ser leídas dentro de ella
BUFFER	Este modo se utiliza para las señales que además de salir de la entidad pueden usarse como entradas realimentadas
INOUT	Este modo se utiliza para señales bidireccionales. Se emplea en salida con tres estados (tri-state)



TIPOS DE DATOS BIT VECTOR

Bit_vector: vector compuesto de varios bits

Port (A, B : **in** bit_vector (0 to 2);

Port (A, B : **in** bit_vector (2 downto 0);

A(2) <= '0'; -- Se le asigna 0 al bit 2 del vector A, se usa comillas simples

A(2 downto 0) <= "011"; -- Se le asignan "1" a los bits 0 y 1 y "0" al bit 2 del vector A, se usa comillas dobles

ARQUITECTURA – ARCHITECTURE

Es la estructura que define el funcionamiento de una entidad. Indica el tipo de procesamiento que se realiza con las señales de entradas declaradas en la entidad.

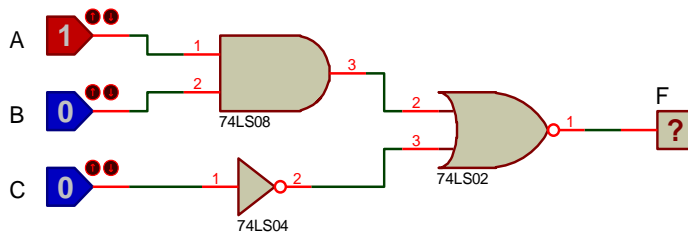
Estilos de modelación:

- Estilo de Flujo de Datos = con ecuaciones booleanas y con las instrucciones WHEN (cuando) – ELSE (el resto).
- Estilo Estructural = en el modelado estructural se declaran previamente los dispositivos que componen el circuito y se realizan las interconexiones.
- Estilo Comportamental o Funcional = expone la forma en que trabaja el sistema, relación que hay entre las entradas y salidas del circuito, sin importar como esté organizado en su interior (caja negra), este modelado se basa en el uso de procesos y declaraciones secuenciales.

MODELADO POR FLUJO DE DATOS

Se utilizan con Ecuaciones booleanas y con las instrucciones WHEN (cuando) – ELSE (el resto).

Ecuaciones Booleanas



$$F = \overline{(A \times B) + C}$$

-- EJEMPLO DOCUMENTO - COMPUERTAS

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Compuertas is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          C : in  STD_LOGIC;
          F : out STD_LOGIC);
end Compuertas;

-- El nombre de la entidad es compuertas
-- Entrada A es de tipo lógica
-- Entrada B es de tipo lógica
-- Entrada C es de tipo lógica
-- Salida F es de tipo lógica
-- Fin de la entidad

architecture Comportamiento of Compuertas is
begin
    F <= not ((A and B) or (not C)); -- Función booleana
end Comportamiento;
-- Fin de la arquitectura

```

-- EJEMPLO DOCUMENTO - COMPUERTAS_1

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Compuertas_1 is
    Port ( A : in  STD_LOGIC_VECTOR (2 downto 0);
          F : out STD_LOGIC);
end Compuertas_1;

-- El nombre de la entidad es compuertas_1
-- Vector lógico A de 2 a 0
-- Salida F es de tipo lógica
-- Fin de la entidad

architecture Comportamiento of Compuertas_1 is
begin
    F <= not ((A(0) and A(1)) or not A(2)); -- Función booleana con vector
end Comportamiento;
-- Fin de la arquitectura

```



Instrucciones When – Else

C	B	A	$A \times B$	\overline{C}	$F = \overline{(A \times B) + \overline{C}}$	
0	0	0	0	1	0	
0	0	1	0	1	0	
0	1	0	0	1	0	
0	1	1	1	1	0	
1	0	0	0	0	1	$\overline{A} \times \overline{B} \times C$
1	0	1	0	0	1	$A \times \overline{B} \times C$
1	1	0	0	0	1	$\overline{A} \times B \times C$
1	1	1	1	0	0	

-- EJEMPLO DOCUMENTO - COMPUERTAS_2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Compuertas_2 is
    Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : in STD_LOGIC;
          F : out STD_LOGIC);
end Compuertas_2;

architecture Comportamiento of Compuertas_2 is
begin
    F <= '1' when (A='0' and B='0' and C='1') else -- F=1 cuando CBA es "100"
          '1' when (A='1' and B='0' and C='1') else -- F=1 cuando CBA es "101"
          '1' when (A='0' and B='1' and C='1') else -- F=1 cuando CBA es "110"
          '0'; -- En el resto F=0
end Comportamiento;
```

MODELADO POR ESTILO ESTRUCTURAL

En el modelado estructural se declaran previamente los dispositivos que componen el circuito y se realizan las interconexiones. El propósito es permitir que dispositivos comunes de código sean reutilizados y compartidos.

Un “COMPONENT” es un dispositivo con código: declaración de librería + entidad + arquitectura. Por declarar el código como un componente, puede ser utilizado dentro de otro circuito, lo que permite la construcción de diseños jerárquicos.

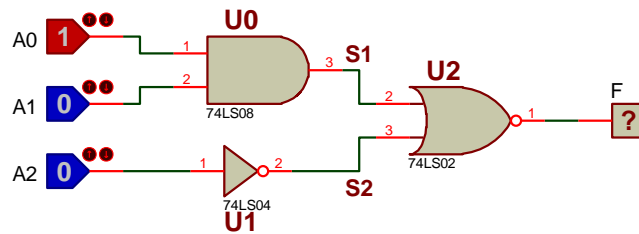


Para hacer uso de un COMPONENT, debe ser declarado. La sintaxis de declaración se muestra a continuación:

Component *nombre_del_componente* **is**

Port (NOMBRE del puerto: MODO y TIPO de dato;
 NOMBRE del puerto: MODO y TIPO de dato);

End Component;



-- EJEMPLO DOCUMENTO - COMPUERTAS_3

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

entity *Compuertas_3* **is**

```
    Port ( A : in STD_LOGIC;  
          B : in STD_LOGIC;  
          C : in STD_LOGIC;  
          F : out STD_LOGIC);
```

end *Compuertas_3*;

architecture *Estructural of Compuertas_3* **is**

```
    signal S1,S2: STD_LOGIC;                    -- Señales internas de la entidad
```

```
    component and2                              -- Componente AND de 2 entradas
```

```
        port(I0,I1: IN STD_LOGIC; O: OUT STD_LOGIC);    -- Entradas y Salidas
```

```
    end component;
```

```
    component inv                              -- Componente INV
```

```
        port(I: IN STD_LOGIC; O: OUT STD_LOGIC);        -- Entrada y Salida
```

```
    end component;
```

```
    component nor2                              -- Componente NOR de 2 entradas
```

```
        port(I0,I1: IN STD_LOGIC; O: OUT STD_LOGIC);    -- Entradas y Salidas
```

```
    end component;
```

begin



```
U0:and2 port map (I0=>A, I1=>B, O=>S1); -- Conexiones de la AND
U1:inv port map (I=>C, O=>S2); -- Conexiones de la INV
U2:nor2 port map (I0=>S1, I1=>S2, O=>F); -- Conexiones de la NOR
end Estructural;
```

Modelado Estructural basado en la declaración de las entradas como vectores:

-- EJEMPLO DOCUMENTO - COMPUERTAS_4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

entity *Compuertas_4* **is**

```
Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
       F : out STD_LOGIC);
```

end *Compuertas_4*;

architecture *Estructural of Compuertas_4* **is**

```
signal S1,S2: STD_LOGIC; -- Señales internas de la entidad
```

```
component and2 -- Componente AND de 2 entradas
port(I0,I1: IN STD_LOGIC; O: OUT STD_LOGIC); -- Entradas y Salidas
end component;
```

```
component inv -- Componente INV
port(I: IN STD_LOGIC; O: OUT STD_LOGIC); -- Entrada y Salida
end component;
```

```
component nor2 -- Componente NOR de 2 entradas
port(I0,I1: IN STD_LOGIC; O: OUT STD_LOGIC); -- Entradas y Salidas
end component;
```

begin

```
U0:and2 port map (I0=>A(0),I1=>A(1),O=>S1); -- Conexiones de la AND
U1:inv port map (I=>A(2),O=>S2); -- Conexiones de la INV
U2:nor2 port map (I0=>S1,I1=>S2,O=>F); -- Conexiones de la NOR
end Estructural;
```

MODELADO COMPORTAMENTAL O FUNCIONAL

Expone la forma en que trabaja el sistema, relación que hay entre las entradas y salidas del circuito, sin importar como esté organizado en su interior (caja negra), este modelado se basa en el uso de procesos y declaraciones secuenciales.

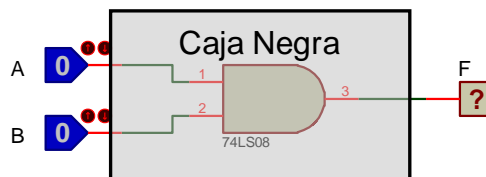
Los procesos se componen de una parte declarativa y de otra procedimental. En la primera se incluyen las señales y variables locales. En la segunda las operaciones secuenciales.

Los procesos se definen con la etiqueta *PROCESS*, la cual incluye el Nombre del Proceso. Entre *PROCESS* y la palabra *BEGIN* tenemos la parte declarativa del proceso llamada la lista de sensibilidad de señales (*sensitivity list*) que hacen que el proceso se dispare, de tal forma que cada vez que cambia el valor de alguna de las señales incluida en esta lista, se ejecutan las instrucciones del proceso.

Después de *BEGIN* se coloca la serie de sentencias secuenciales. La lista de sensibilización se hace entre paréntesis después de la palabra *PROCESS*. Si no hay lista de señales el proceso no se ejecuta.

- Un proceso puede contener órdenes secuenciales, pero todos los procesos se ejecutan concurrentemente.
- Las acciones descritas en el cuerpo del proceso se ejecutan en orden, de la primera a la última.
- Cuando se ejecuta la última acción se dice que el proceso queda suspendido y cuando ocurre un cambio en una señal de la lista de sensibilidades el proceso se inicia de nuevo, volviéndose a ejecutar las acciones de la primera a la última.

Ejemplo:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity caja_negra is
```

```
    Port ( A, B : in STD_LOGIC;
          F: out STD_LOGIC );
```

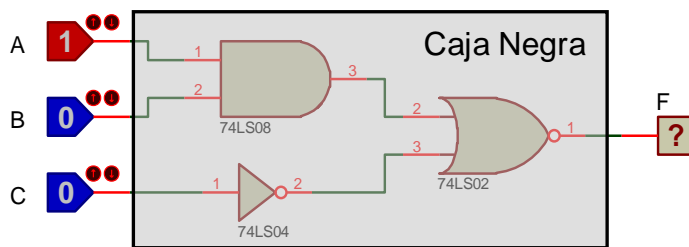
```
end caja_negra;
```

```
-- El nombre de la entidad
-- Entradas A y B de tipo lógica
-- Salida F de tipo lógica
-- Fin de la entidad
```

```

architecture Comportamental of caja_negra is -- Inicio de la arquitectura
begin
  process (A, B)                                -- Lista de sensibilidad
  begin
    if ((A and B) = '1') then                  -- Si AxB es igual "1" entonces
      F <= '1';                                -- Lleve "1" a F
    else                                        -- Si no
      F <= '0';                                -- Lleve "0" a F
    end if;                                    -- Fin de Si
  end process;                                -- Fin de proceso
end Comportamental;                          -- Fin de la arquitectura

```



$$F = \overline{(A \times B) + C}$$

-- EJEMPLO DOCUMENTO - COMPUERTAS_5

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

entity Compuertas_5 is                        -- El nombre de la entidad
  Port ( A : in  STD_LOGIC;                  -- Entradas A de tipo lógica
         B : in  STD_LOGIC;                  -- Entradas B de tipo lógica
         C : in  STD_LOGIC;                  -- Entradas C de tipo lógica
         F : out STD_LOGIC);                 -- Salida F de tipo lógica
end Compuertas_5;                            -- Fin de la entidad

```

```

architecture Comportamental of Compuertas_5 is -- Inicio de la arquitectura
begin
  process (A, B, C)                          -- Lista de sensibilidad
  begin
    if (((A and B) nor (not C))='1') then    -- Si la función = "1" entonces
      F <= '1';                                -- Lleve "1" a F
    else                                        -- Si no
      F <= '0';                                -- Lleve "0" a F
    end if;                                    -- Fin de Si
  end process;                                -- Fin de proceso

```



end Comportamental;

-- Fin de la arquitectura

Otra forma de modelarlo:

-- EJEMPLO DOCUMENTO - COMPUERTAS_6

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Compuertas_6 **is**

-- El nombre de la entidad

Port (A : **in** STD_LOGIC;

-- Entradas A de tipo lógica

B : **in** STD_LOGIC;

-- Entradas B de tipo lógica

C : **in** STD_LOGIC;

-- Entradas C de tipo lógica

F : **out** STD_LOGIC;

-- Salida F de tipo lógica

end Compuertas_6;

-- Fin de la entidad

architecture Comportamental **of** Compuertas_6 **is**
begin

-- Inicio de la arquitectura

process (A, B, C)

-- Lista de sensibilidad

begin

if ((**not** A **and** **not** B **and** C)='1') **then**

-- Si la función = 1 entonces

F <= '1';

-- Lleve "1" a F

elsif ((A **and** **not** B **and** C)='1') **then**

-- Si no Si función=1 entonces

F <= '1';

-- Lleve "1" a F

elsif ((**not** A **and** B **and** C)='1') **then**

-- Si no Si función=1 entonces

F <= '1';

-- Lleve "1" a F

else

-- Si no

F <= '0';

-- Lleve "0" a F

end if;

-- Fin de Si

end process;

-- Fin de proceso

end Comportamental;

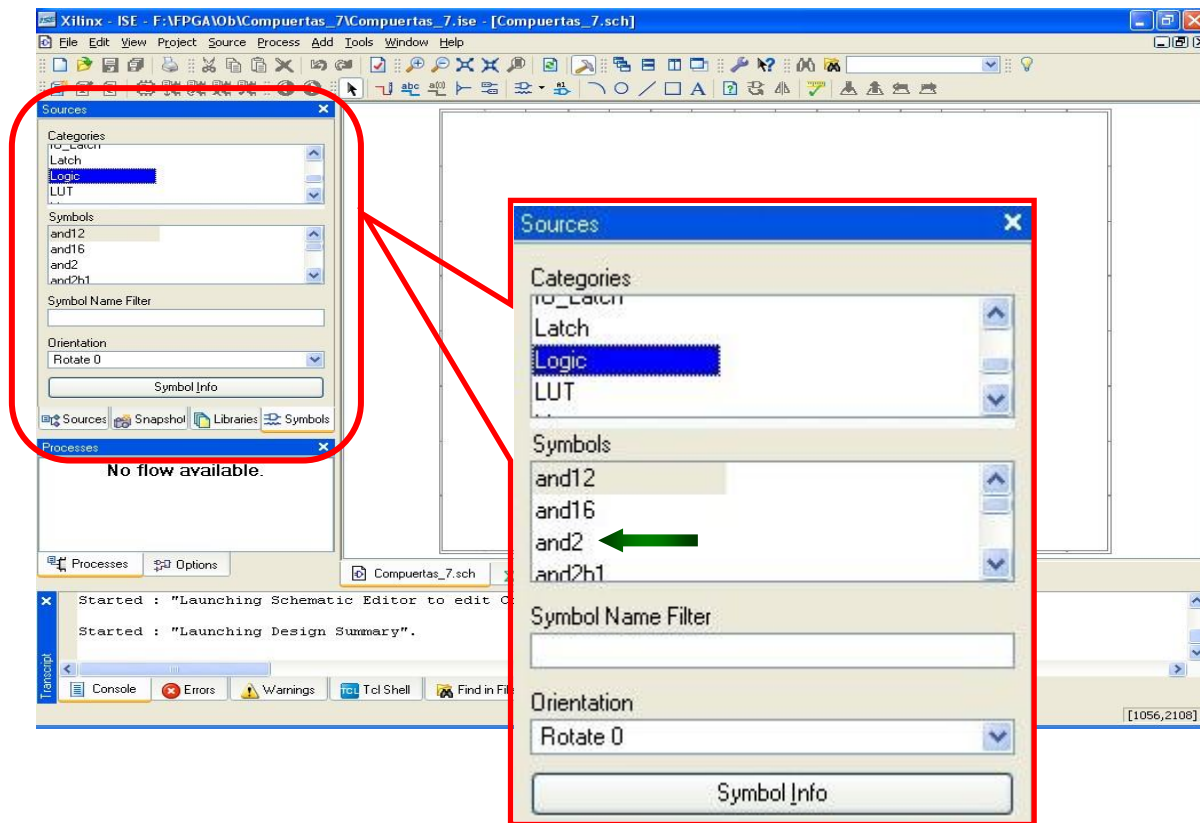
-- Fin de la arquitectura

MODELADO ESQUEMÁTICO

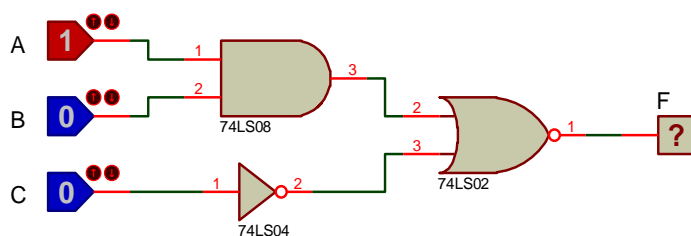
Con clic en la pestaña *Symbols* se activa la ventana Fuentes (Sources) y selecciona la categoría Lógica (*lógica*) y de los Símbolos (*Symbols*) se toman las compuertas lógicas necesarias para la implementación del diseño del circuito esquemático.

Símbolos está dividido en varios cuadros de dialogo. En el primero de ellos, Categorías (*Categories*) se puede escoger el tipo de elementos que se quiere usar, como puertas lógicas, comparadores, multiplexores, memorias, etc. Si se

selecciona (*All Symbols*) podrá tenerse acceso a un listado de todos los componentes pertenecientes a la librería de componentes. Este actúa como filtro para el cuadro de dialogo Símbolos (*Symbols*), en este es donde finalmente se selecciona el componente específico (por ejemplo and2, inv y nor2). También si se conoce el nombre exacto se puede utilizar el cuadro de filtraje *Symbol Name Filter*.



Se inicia colocando las compuertas lógicas necesarias para el diseño del circuito.




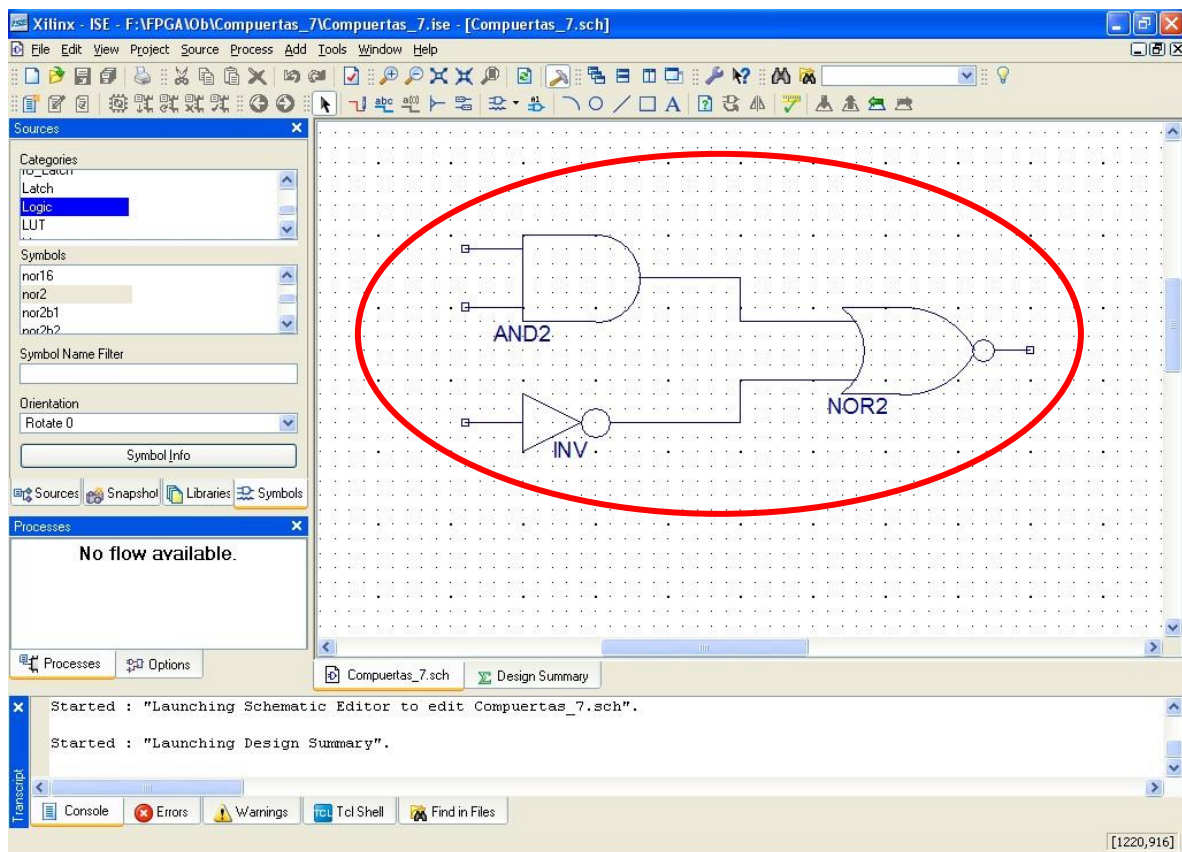
$$F = \overline{(A \times B) + C}$$


Las compuertas que se necesitan para el diseño son:

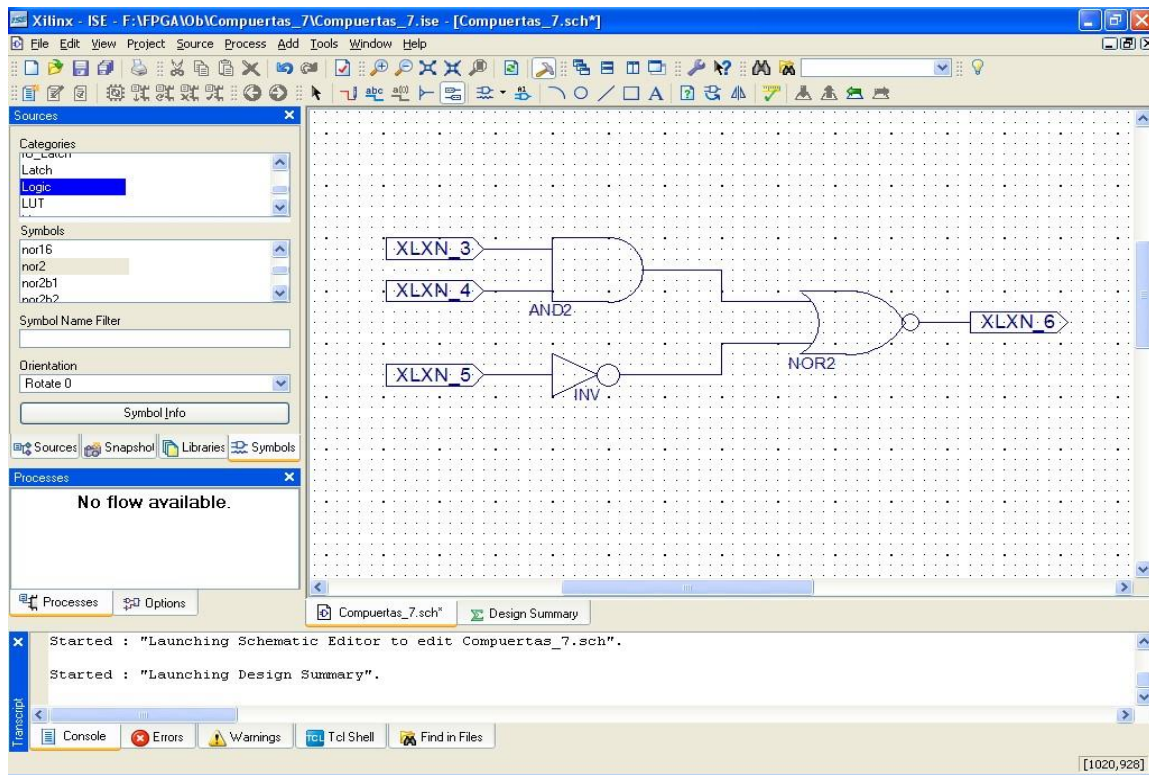
- and2: compuerta AND de 2 entradas
- inv: compuerta Inversora
- nor2: compuerta NOR de 2 entradas

Si se desea rotar una componente puede hacerse empleando la ventana inferior izquierda de la ventana Fuentes (*Sources*), la opción Orientación (*Orientation*). Para eliminar una componente simplemente hay que seleccionarla y pulsar la tecla Suprimir. Así mismo, para mover una componente bastará con seleccionarlo y arrastrarlo hasta el lugar deseado mientras se mantiene pulsado el botón izquierdo del *mouse*.

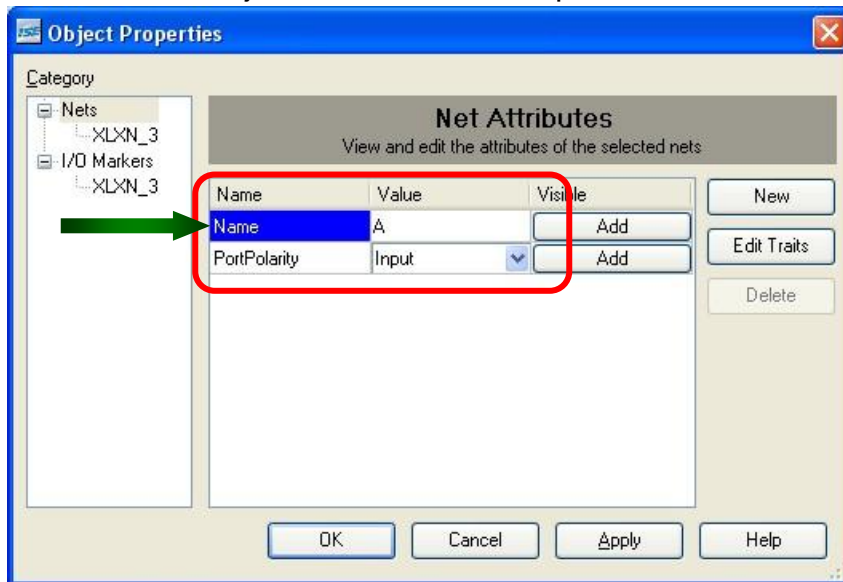
Ahora, se realizan las conexiones entre las compuertas lógicas que componen el diseño y para ello se utiliza el icono  de la barra de herramientas, donde el cursor se transformará en una cruz y se da clic en los dos extremos que se quieren unir por medio del alambre. Si queremos unir una entrada o salida de una componente y dejar el otro lado del cable “al aire” tendremos que dar doble clic para indicar que el extremo final queda suelto.

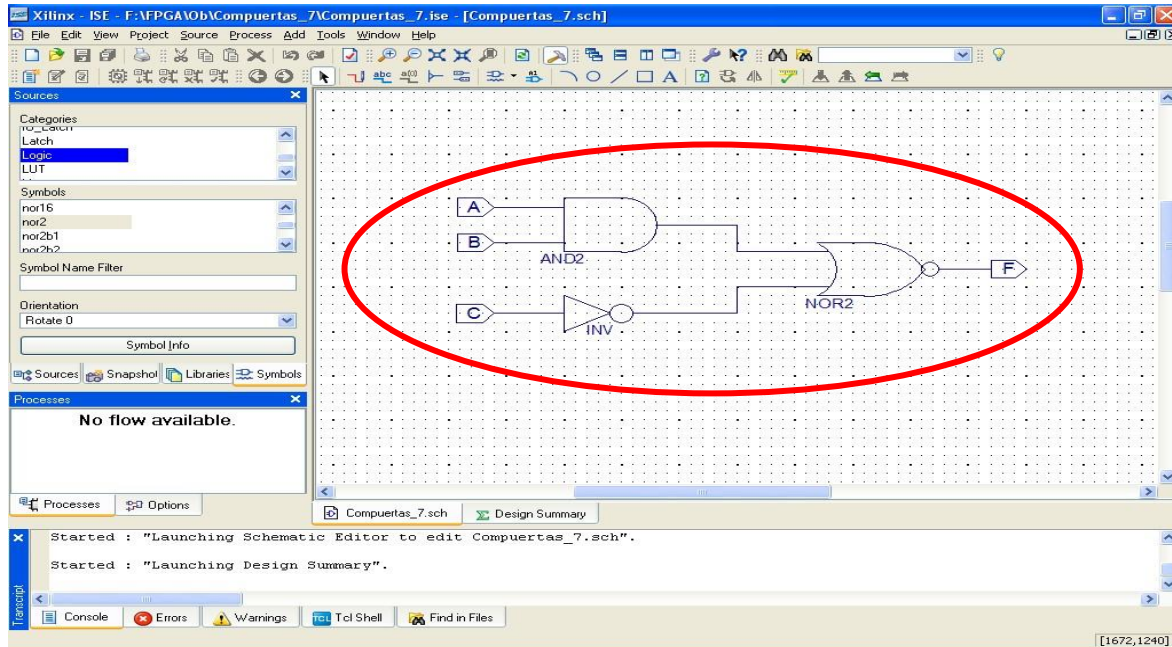



Para colocarle los dispositivos de entrada y salida al circuito se usa el icono  de la barra de herramientas.



Basado en la función booleana del ejercicio se observa que las variables de entrada tienen los literales A, B y C, y el nombre de la función o sea la salida tiene la letra F. Entonces, procedemos a editar las terminales colocadas en el paso anterior para asignarle las letras correspondientes y se realiza con doble clic sobre la terminal a editar y en la ventana que se activa se cambia el dato que tiene en el campo Nombre (*Name*) por la letra correspondiente; se procede igualmente con todas las demás terminales de entrada y la de salida.





Una vez terminado el diseño del circuito debe hacer una verificación para asegurar que todas las conexiones están bien hechas y que no hay cables sueltos o compuertas sin conectar; este se realiza dando clic en el icono Chequear Esquemático  o elija en la barra de herramientas la opción Herramientas (Tools) y luego Check Schematic.

