

INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

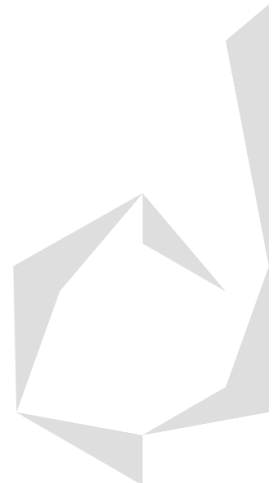
ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Verilog: Top Level Design (TLD)

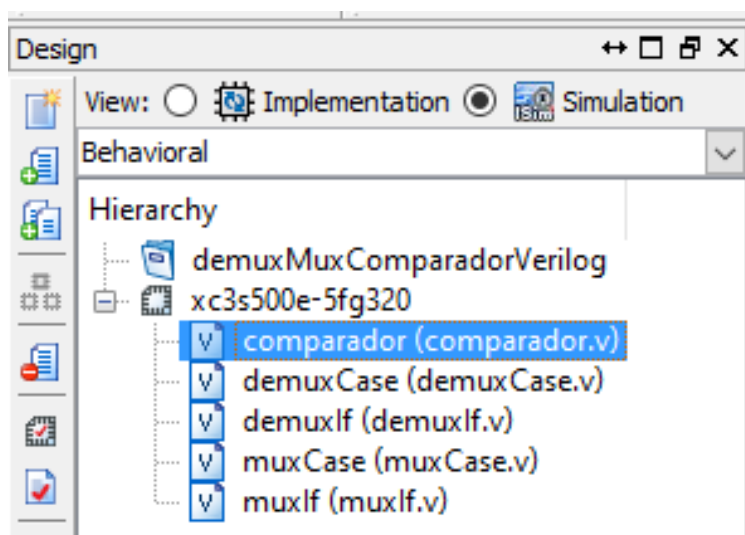
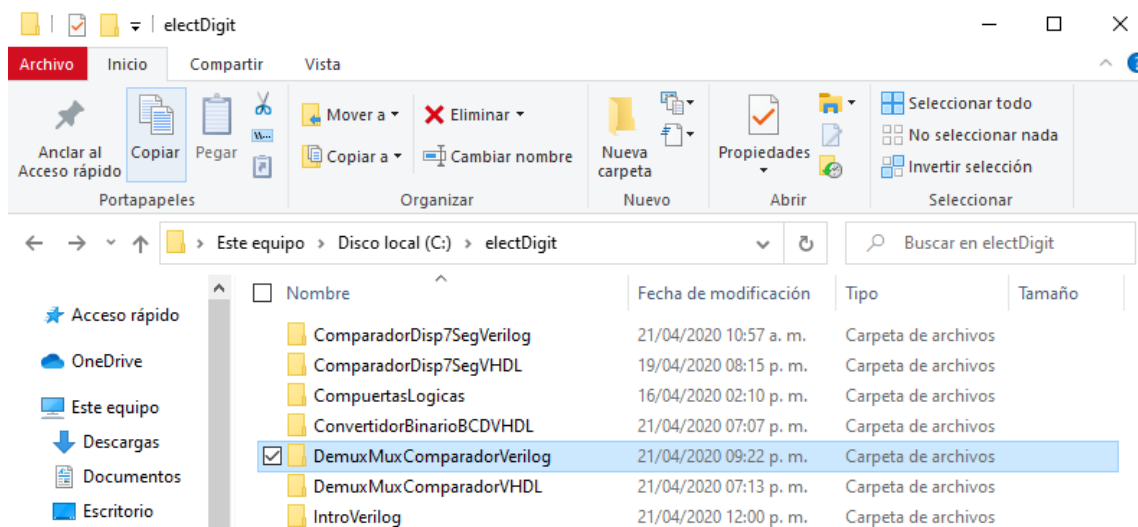
Contenido

Diseño de Alto Nivel (TLD) Verilog	2
Guardar Valores Temporalmente en Verilog.....	3
Sumador Completo	3
Aspecto de un TLD en ISE Xilinx	4
Código de Sumador Completo con TLD	6
Código Verilog:.....	6
Módulo Sumador HA1 y HA2 (Son Iguales):	6
Módulo Compuerta OR:.....	6
Módulo TLD:.....	6
Código VHDL:	7
Módulo Sumador HA1 y HA2 (Son Iguales):	7
Módulo Compuerta OR:.....	7
Módulo TLD:.....	7



Diseño de Alto Nivel (TLD) Verilog

El Top Level Design o TLD es una manera de unir todos los módulos que tengamos dentro de una carpeta de proyecto (programada en VHDL o Verilog).



Esto se debe hacer en proyectos complejos para simplificar y/ o modularizar el código, ya que es más difícil manejar un proyecto complejo en un solo módulo a dividirlo en varios módulos pequeños, de esta manera es más fácil encontrar donde está el error si es que ocurrió uno, además en VHDL o Verilog no se puede crear un archivo UCF (que asigna los puertos electrónicos físicos a las entradas y salidas de mi código) para cada módulo individualmente, solo puede existir uno por carpeta.

Lo que se hace en este caso es crear módulos individuales que tengan una función específica para después integrarlos todos por medio de un módulo principal que mande a llamar a los demás, a esto se le llama **instanciar**.

Usualmente para entender mejor este tipo de programas se suele hacer un diagrama de bloques explicando su funcionamiento paso a paso y la interacción entre sus submódulos para formar el módulo completo, también con esto me puedo dar cuenta cuáles van a ser mis entradas y salidas globales (declaradas como **in** o **out**) y cuáles van a ser las entradas o salidas de los submódulos que solo van a funcionar internamente (declaradas como variables tipo **wire**).

Guardar Valores Temporalmente en Verilog

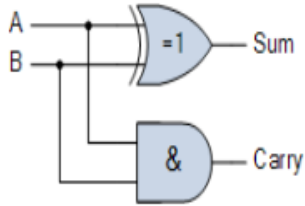
La palabra reservada **wire** existe solo en Verilog y es usada para poder guardar un valor dentro del código que solo pueda existir durante la ejecución del programa, sin que sea una entrada o salida, ya que estas dos deben estar vinculadas a algún elemento electrónico de la tarjeta de desarrollo. Las **wire** se declaran de la misma forma en la que se declara una entrada o salida, ya sea en forma de vector o de 1 bit:

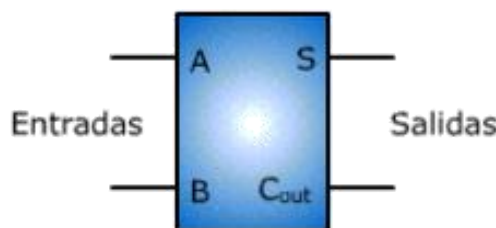
```
wire nombreWire1;      --wire de 1 bit
wire [n:0] nombreWire2; --wire tipo vector de n+1 bits
```

Sumador Completo

El ejemplo más simple de un TLD es el sumador completo, este se compone de dos medios sumadores y una compuerta lógica **or** para ser formado, lo que se hace es crear un módulo que represente al **medio sumador**, un módulo que represente la **compuerta or** y finalmente se une todo en un módulo principal llamado **sumador completo**.

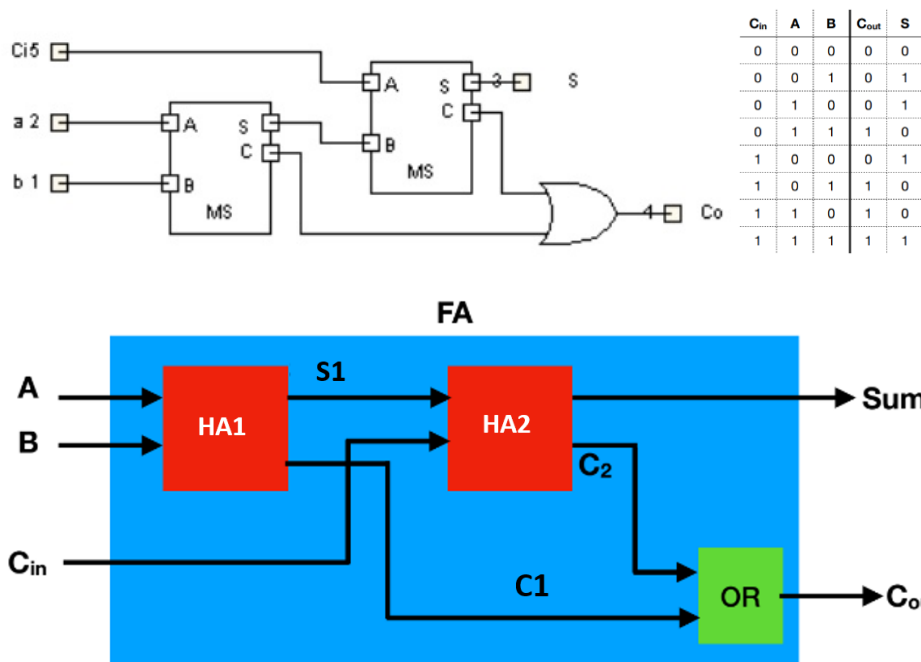
Medio Sumador

Symbol	Truth Table			
	B	A	SUM	CARRY
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1



Un medio sumador es capaz de sumar 2 dígitos binarios de un solo bit y producir un bit de acarreo, la salida de acarreo (carry) se puede obtener aplicando una compuerta **and** y la salida del resultado de la suma (sum) se puede obtener aplicando una compuerta lógica **xor**. Sin embargo, esta configuración del medio sumador puede ser combinada para que a través de él se pueda sumar más de 1 bit a la vez.

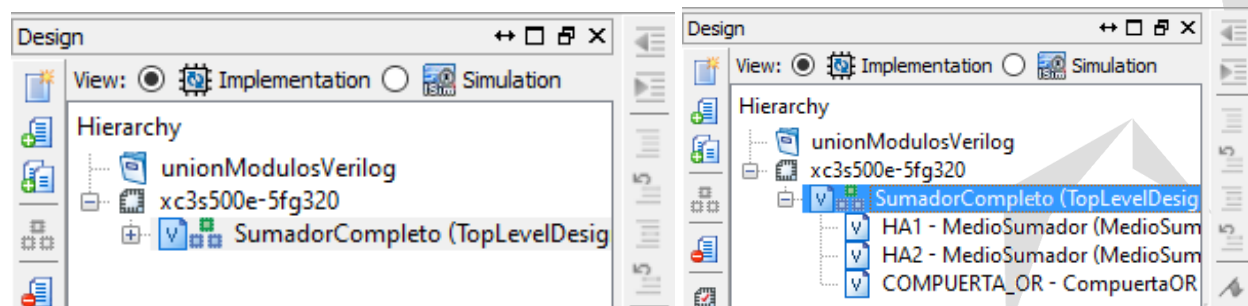
La combinación de dos **medios sumadores** se llama **sumador completo** y permite sumar 3 dígitos binarios de 1 solo bit (2 bits y 1 acarreo), obteniendo como resultado un bit de acarreo y el resultado de la suma, este se obtiene uniendo dos medios sumadores a través de una compuerta **OR** de la siguiente manera:



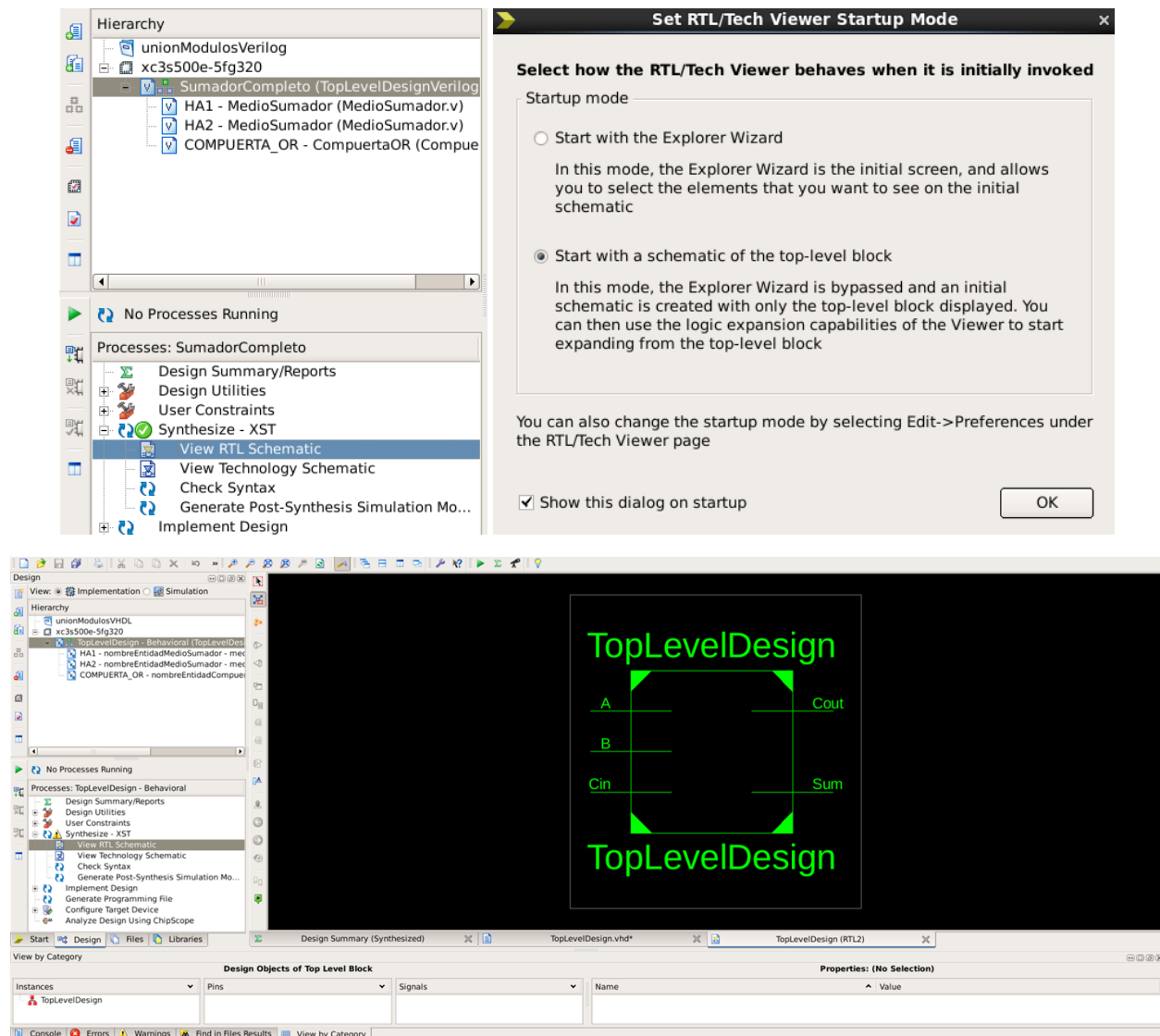
En el diagrama de bloques podemos ver que el módulo principal del TLD (Top Level Design) es el **sumador completo (Full Adder o FA)**, para conformarlo se deben declarar las entradas **A, B y Cin** como **input** y las salidas **Sum y Cout** como **output** porque salen del bloque global. Además, podemos ver que la salida **S1** del primer **medio sumador (Half Adder o HA)** y los acarreos de salida **C1 y C2** del primer y segundo medio sumador son de tipo **wire** porque nunca salen del bloque global.

Aspecto de un TLD en ISE Xilinx

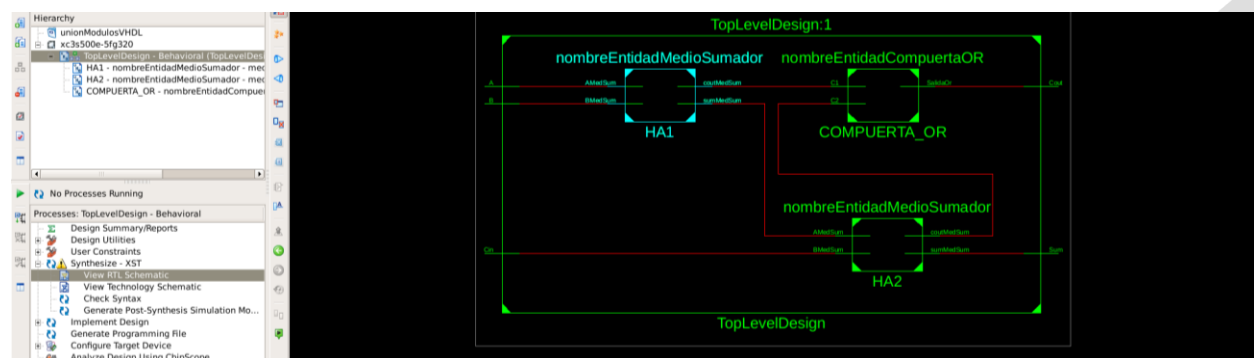
Al instanciar los submódulos desde el módulo principal, el nombre mostrado de cada submódulo en la pestaña de Design cambia al que le asigné cuando le hice su instancia dentro del módulo principal y los submódulos se ponen como subcarpetas de la siguiente manera:



Además después de haber Compilado el programa, desplegar el menú de Synthesize, dar clic sobre la opción de View RTL Schematic y seleccionar el radio button de Start with a schematic of the top-level block, podremos observar un diagrama que muestra las conexiones del diagrama TLD que hayamos codificado en el programa.



Además, si se da doble clic sobre el diagrama TLD, se podrá ver como interactúan sus diagramas internos.



Código de Sumador Completo con TLD

A continuación, se llevarán a cabo los mismos ejercicios resueltos con los lenguajes VHDL y Verilog para que se noten las diferencias al usar ambos en un diseño TLD.

Código Verilog:

Módulo Sumador HA1 y HA2 (Son Iguales):

En el caso del medio sumador HA, este se muestra dos veces en el diseño TLD no porque se repita el módulo, sino porque este se está usando dos veces.

```
//MEDIO SUMADOR: Sumador de 2 bits
module MedioSumador(
    input aMedSum,
    input bMedSum,
    output sumMedSum,
    output coutMedSum
);

    //Las compuertas lógicas en Verilog se aplican con la palabra reservada que las describe, osea and, or, not, nand,
    //nor, xor y xnor. Después debemos poner dentro de su paréntesis primero la salida que almacenara el resultado de
    //la operación y luego las entradas a las que se les vaya a aplicar la compuerta lógica separadas por comas.

    and(coutMedSum, aMedSum, bMedSum); //Acarreo del medio sumador.
    xor(sumMedSum, aMedSum, bMedSum); //Resultado de la suma del medio sumador.
endmodule
```

Módulo Compuerta OR:

```
//Compuerta OR, esta se usa para unir el acarreo de los dos medios sumadores para obtener el carry de salida del
//sumador completo llamado Cout
module CompuertaOR(
    input C1,
    input C2,
    output SalidaOR
);

    //Las compuertas lógicas en Verilog se aplican con la palabra reservada que las describe, osea and, or, not, nand,
    //nor, xor y xnor. Después debemos poner dentro de su paréntesis primero la salida que almacenara el resultado de
    //la operación y luego las entradas a las que se les vaya a aplicar la compuerta lógica separadas por comas.

    or(SalidaOR, C1, C2); //Compuerta OR que recibe los dos acarreos de salida del medio sumador 1 y 2.
endmodule
```

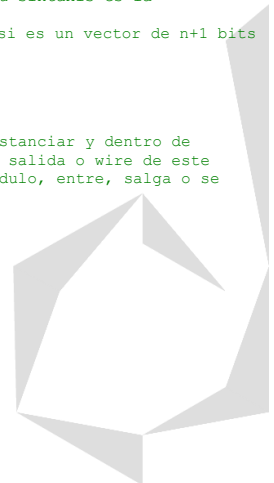
Módulo TLD:

```
//Top Level Design (TLD): Sumador completo usando dos medios sumadores y una compuerta OR
module SumadorCompleto(
    input A,
    input B,
    input Cin,
    output Sum,
    output Cout
);
    //Las entradas y salidas de este módulo son solo las que van a entrar y salir del sumador completo

    //WIRE: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
    //existe durante la ejecución del código y sirve para poder usar algún valor internamente, su sintaxis es la
    //siguiente:
    //wire nombreWire1; si es de 1 bit o wire [n:0]nombreWire1; si es un vector de n+1 bits
    wire s1; //Esta suma sale de un medio sumador hacia el otro sumador.
    wire c1; //Este acarreo sale de un medio sumador hacia la compuerta OR.
    wire c2; //Este es el otro acarreo que sale de un medio sumador hacia la compuerta OR.

    //INSTANCIAS:
    //Debo darle un nombre a cada instancia que cree, indicar el nombre del módulo que quiero instanciar y dentro de
    //un paréntesis asignarle a todas las entradas y salidas del módulo instanciado una entrada, salida o wire de este
    //módulo, separadas por comas una de la otra, esto hará que lo que entre o salga del otro módulo, entre, salga o se
    //guarde en este. La sintaxis que debemos usar es la siguiente:

    //Nombre_Del_Módulo_Que_Queremos_Instanciar NombreInstancia (
    //    .Nombre_De_La_Entrada_Del_Módulo_Instanciado(Entrada_En_Este_Módulo),
    //    .Nombre_De_La_Salida_Del_Módulo_Instanciado(Salida_En_Este_Módulo),
    //
    //    .Nombre_De_La_Entrada_Del_Módulo_Instanciado(Salida_En_Este_Módulo),
    //    .Nombre_De_La_Salida_Del_Módulo_Instanciado(Entrada_En_Este_Módulo),
    //
    //    .Nombre_De_La_Entrada_Del_Módulo_Instanciado(Signal_En_Este_Módulo),
    //    .Nombre_De_La_Salida_Del_Módulo_Instanciado(Signal_En_Este_Módulo)
    //);
```



```

//);

//INSTANCIAS DEL MODULO MedioSumador
//1er Medio Sumador o Half Adder 1 (HA1)
MedioSumador HA1(
    .aMedSum(A),      //La entrada A del 1er medio sumador se asigna a la entrada A del sumador completo.
    .bMedSum(B),      //La entrada B del 1er medio sumador se asigna a la entrada B del sumador completo.
    //Se asigna un wire cuando la salida o entrada de la instancia se usa solo internamente y no sale del
    //diagrama de bloques que describe el TLD (Top Level Desing).
    .sumMedSum(s1),    //La salida Suma del 1er medio sumador se guarda en la signal s1 de este módulo.
    .coutMedSum(c1)    //El acarreo de salida del 1er medio sumador se guarda en la signal c1 de este módulo.
);
//2do Medio Sumador o Half Adder 1 (HA1)
MedioSumador HA2(
    .aMedSum(s1),      //La entrada A del 2do medio sumador se asigna a la salida s1 de HA1 guardada en un wire.
    .bMedSum(Cin),     //La entrada B del 2do medio sumador se asigna a la entrada Cin del sumador completo.
    .sumMedSum(Sum),    //La salida Suma del 2do medio sumador se asigna a la salida Sum del sumador completo.
    .coutMedSum(c2)    //El acarreo de salida del 2do medio sumador se guarda en la wire c2 de este módulo.
);
//INSTANCIA DEL MODULO CompuertaOR.
//Compuerta OR que une los dos medios sumadores HA1 y HA2.
CompuertaOR COMPUERTA_OR(
    .C1(c1),           //La entrada C1 de la compuerta OR se asigna a la salida c1 de HA1 guardada en una signal.
    .C2(c2),           //La entrada C2 de la compuerta OR se asigna a la salida c2 de HA2 guardada en una signal.
    .SalidaOR(Cout)    //La salida de la compuerta OR se asigna a la salida Cout del sumador completo.
);
Endmodule

```

Código VHDL:

Módulo Sumador HA1 y HA2 (Son Iguales):

```

--MEDIO SUMADOR: Sumador de 2 bits
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--El medio sumador tiene dos entradas A y B de 1 bit y 2 salidas, el resultado de la suma y su acarreo (carry).
entity nombreEntidadMedioSumador is
    Port ( AMedSum : in  STD_LOGIC;
          BMedSum : in  STD_LOGIC;
          sumMedSum : out STD_LOGIC;
          coutMedSum : out STD_LOGIC);
end nombreEntidadMedioSumador;

architecture medioSumador of nombreEntidadMedioSumador is
begin
    sumMedSum <= AMedSum xor BMedSum;--Resultado de la suma.
    coutMedSum <= AMedSum and BMedSum;--Acarreo o Carry.
end medioSumador;

```

Módulo Compuerta OR:

```

--Compuerta OR, esta se usa para unir el acarreo de los dos medios sumadores para obtener el carry de salida del
--sumador completo llamado Cout.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Esta es una simple compuerta OR en el diagrama de bloques.
entity nombreEntidadCompuertaOR is
    Port ( C1 : in  STD_LOGIC;
          C2 : in  STD_LOGIC;
          SalidaOr : out STD_LOGIC);
end nombreEntidadCompuertaOR;

architecture compuertaOR of nombreEntidadCompuertaOR is
begin
    SalidaOr <= C1 or C2;--Compuerta OR que recibe los dos acarreos de salida del medio sumador 1 y 2.
end compuertaOR;

```

Módulo TLD:

```

--Top Level Desing: Sumador Completo usando dos medios sumadores y una compuerta OR.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías que me permiten usar el lenguaje VHDL.

--Entidad: Aquí se declaran las entradas y salidas.
entity TopLevelDesign is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Cin : in  STD_LOGIC;
          Sum : out STD_LOGIC;
          Cout : out STD_LOGIC);
    --Las entradas y salidas en este módulo son solamente las que van a entrar y salir del sumador completo.

```




```

    );
end TopLevelDesign;
--Estas entradas y salidas deben ser asignadas a las entradas o salidas de los demás módulos para que se pueda crear el
--diagrama de bloques TLD completo.

--Arquitectura: Aquí se declaran las instancias de los demás módulos para poder unir entradas, salidas y signals.
architecture Behavioral of TopLevelDesign is
--SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
--existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
--arquitectura y antes de su begin, su sintaxis es la siguiente:
--signal nombreSignal: tipo_de_dato_vector_o_bit;
signal s1:STD_LOGIC;--Esta suma sale de un medio sumador hacia el otro sumador.
signal c1:STD_LOGIC;--Este acarreo sale de un medio sumador hacia la compuerta OR.
signal c2:STD_LOGIC;--Este es el otro acarreo que sale de un medio sumador hacia la compuerta OR.
--Las signal declaradas son las salidas de los medios sumadores que solo hacen conexiones internas y no salen
--del bloque global.

--Dentro del begin de la arquitectura es donde debo hacer las instancias de los demás módulos para mandarlos a
--llamar y poder usarlos para crear el TLD (Top Level Desgin).
begin
    --INSTANCIAS:
    --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del módulo que quiero instanciar,
    --usar la palabra reservada port map(); y dentro de su paréntesis asignarle a todas las entradas y salidas del
    --módulo instanciado una entrada, salida o signal de este módulo separadas por comas una de la otra, esto hará que
    --lo que entre o salga del otro módulo, entre, salga o se guarde en este.
    --La sintaxis que debemos usar es la siguiente:

    --NombreInstancia      :                      entity work.Entidad_Del_Modulo_Que_Queremos_Instanciar      port map(
    --                      Entrada_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,
    --                      Salida_Del_Modulo_Instanciado  => Salida_En_Este_Modulo,

    --                      Entrada_Del_Modulo_Instanciado => Salida_En_Este_Modulo,
    --                      Salida_Del_Modulo_Instanciado  => Entrada_En_Este_Modulo,

    --                      Entrada_Del_Modulo_Instanciado => Signal_En_Este_Modulo,
    --                      Salida_Del_Modulo_Instanciado  => Signal_En_Este_Modulo
    --);

--INSTANCIAS DEL MDULO medioSumador:
--1er Medio Sumador o Half Adder 1 (HA1).
HA1: entity work.nombreEntidadMedioSumador port map(
    AMedSum => A,    --La entrada A del 1er medio sumador se asigna a la entrada A del sumador completo.
    BMedSum => B,    --La entrada B del 1er medio sumador se asigna a la entrada B del sumador completo.
    --Se asigna una signal cuando la salida o entrada de la instancia se usa solo internamente y no sale del
    --diagrama de bloques que describe el TLD (Top Level Desing).
    sumMedSum => s1, --La salida Suma del 1er medio sumador se guarda en la signal s1 de este módulo.
    coutMedSum => c1 --El acarreo de salida del 1er medio sumador se guarda en la signal c1 de este módulo.
);
--2do Medio Sumador o Half Adder 1 (HA2):
HA2: entity work.nombreEntidadMedioSumador port map(
    AMedSum => s1,    --La entrada A del 2do medio sumador se asigna a la salida s1 de HA1 guardada en una signal.
    BMedSum => Cin,   --La entrada B del 2do medio sumador se asigna a la entrada Cin del sumador completo.
    sumMedSum => Sum, --La salida Suma del 2do medio sumador se asigna a la salida Sum del sumador completo.
    coutMedSum => C2  --El acarreo de salida del 2do medio sumador se guarda en la signal c2 de este módulo.
);
--INSTANCIA DEL MODULO compuertaOR:
--Compuerta OR que une los dos medios sumadores HA1 y HA2.
COMPUERTA_OR: entity work.nombreEntidadCompuertaOR port map(
    C1 => c1,        --La entrada C1 de la compuerta OR se asigna a la salida c1 de HA1 guardada en una signal.
    C2 => c2,        --La entrada C2 de la compuerta OR se asigna a la salida c2 de HA2 guardada en una signal.
    SalidaOr => Cout --La salida de la compuerta OR se asigna a la salida Cout del sumador completo.
);
end Behavioral;

```

