***Laboratorio 1 – Compuertas Básicas***

Objetivos del laboratorio son conocer:

* El entorno de desarrollo Vivado (Recomendable tener la última versión si existiesen laboratorios en Vitis)
* Construir un archivo en VHDL
* Simular su entidad
* Navegar por el entorno de desarrollo
* Validar el diseño y descargar al FPGA

*¿Qué es VHDL?*

Es un lenguaje de descripción de hardware para ayudar a las fases de diseño digital. VHDL se deriva de Very High Speed Integrated Circuits Hardware Description Language. Desarrollado en 1983 por el Departamendo de Defensa de los Estados Unidos se convirtio en un estándar más tarde en 1987.

Desde entonces se ha ido actualizando el lenguaje en varias versiones 1993, 2000, 2002, 2008, etc. El lenguaje ha sido revisado y mejorado en cada versión. Las características de VHDL son:

* Puede ser utilizado enteramente en el ciclo de diseño
* Soporta varios niveles de abstracción como de comportamiento, RTL, flujo de datos y estructurado
* Soporta inherentemente el paralelismo, es decir practica que todo se ejecuta independientemente del orden
* Soporta diferentes estilos de modelado
* Soporta buenos principios de diseño
* Es extendible, es decir, se pueden reutilizar paquetes de librerías
* Continuamente está evolucionando y es ampliamente soportado.

Sin embargo no todo en VHDL es fácil, algunas desventajas son:

* Difícil de dominar completamente
* El ecosistema (las herramientas de trabajo) evolucionan lentamente
* Carece de alto nivel de programación, por ejemplo no existen enteros de más de 32 bits
* VHDL no fue diseñado para operaciones de alta síntesis

*Primer paso, definir la entidad y su comportamiento*

Normalmente en VHDL tenemos algo llamado la entidad, la entidad es el bloque que define entradas y salidas solamente, no define el comportamiento de el ‘bloque’

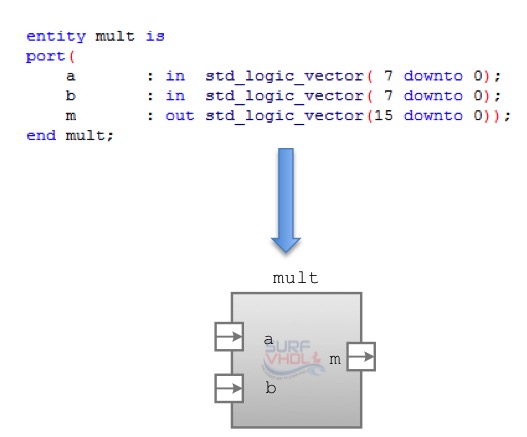


Figura 1. Descripción en VHDL y gráfica de una entidad de un multiplicador.

La arquitectura es el cuerpo donde se define el comportamiento del módulo (el comportamiento de la entidad). Cada valor lógico es asignado desde su entrada a su salida o si se utilizan al conjunto de señales.

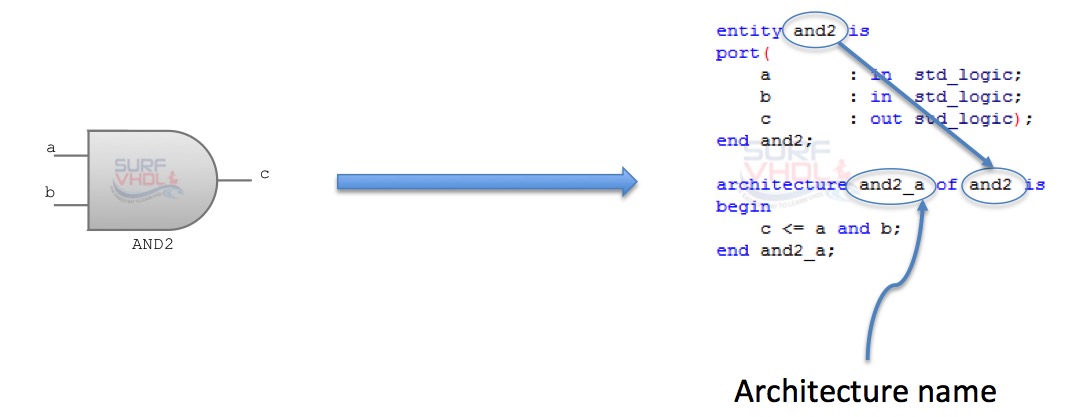


Figura 2. Definición de la Entidad por medio de su comportamiento

*Segundo paso, Análisis, Elaboración, HDL Síntesis, Mapeo, Place & Route*

De todos estos pasos solamente especificaremos la sección de síntesis. En VHDL no existe la compilación como tal, la síntesis es uno de los pasos más importantes que define el diseño de HDL, en este paso el código es pasado a bloques de diseños digitales por medio de elementos de hardware.

*Tercer Paso, Simulación en VHDL:*

La simulación es necesaria para corroborar el diseño antes de la implementación.

En microcontroladores generalmente se tiene un software (IDE) en el cual podemos integrar breakpoints que son puntos donde el programa es detenido y observar las variables, si bien es cierto esto es muy fácil de realizar en MCUs, en FPGAs y VHDL es prácticamente y virtualmente poco probable de hacer esto.

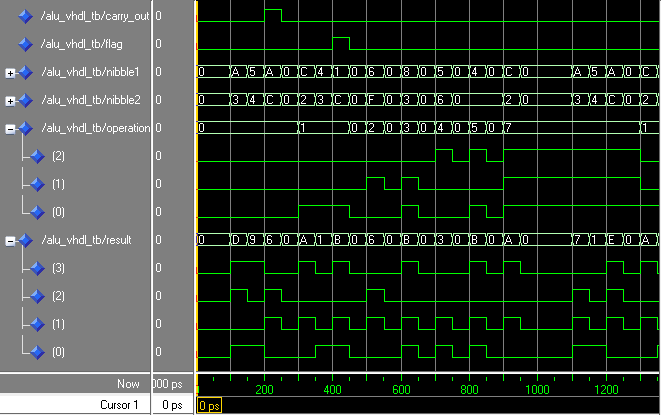


Figura 3 – Depuración en FPGAs/CPLDs/SPLDs

El diseño prácticamente se descarga en la plataforma de desarrollo y se verifica si el funcionamiento final es el correcto. Usualmente el diseño sintetiza, el código está bien escrito pero la manera de depurar el código es por medio de la visualización de las formas de onda y la observación de las entradas vs salida todo esto observando la línea de tiempo.

Esto no es del todo malo, esta simulación nos sirve para documentar y por solamente documentando la forma de onda un ingeniero puede entender mejor su diseño.

Para diseños sencillos como este no es necesaria la simulación, pero para diseños más complejos puede determinar cuanto tiempo puede encontrar un error.

Lo necesario para poder simular:

* Haber completado el archivo o archivos VHDL para poder simular
* Completar el “test bench”.
  + El test bench es una prueba de validación de nuestro circuito específicamente para probar lo sintetizado.
  + Un “test bench” no es sintetizable.
  + Se debe integrar el diseño en VHDL con el test bench, usualmente llamada UUT o Unit Under Test.
* Si se trabaja con ModelSim, se debe crear un TCL scrpit.
  + Modelsim se utiliza con Altera Quartus II tanto como Xilix Vivado si se requiere.
  + Acelera el diseño, simulación y se utiliza para automatizar el diseño pues posee herramientas de línea de comandos

*Cuarto Paso, Modificación del Archivo de pines XDC y descarga en la tarjeta:*

Cada tarjeta o más bien cada número de parte utilizado, en su entorno de desarrollo ya tiene definido los pines que asocian las “salidas” (pues pueden ser también entradas) físicas al mundo real, el archivo en Xilinx tiene extensión XDC y se ve como sigue:

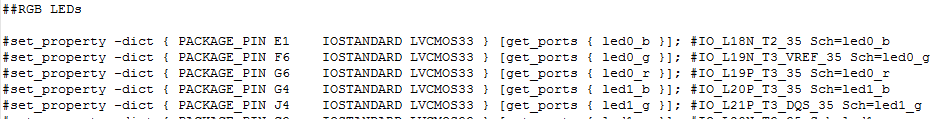


Figura 5. Archivo XDC de descripción de pines.

¡Dato importante!. Para el registro de pines, los nombres declarados en la entidad deben coincidir en mayúsculas y minúsculas para cuando se asocian los elementos. Por ejemplo, si poseemos en la entidad un pin de entrada que se llama A, en el archivo XDC debe llamarse el pin de esta misma manera A.

Ejemplo:

Tabla 1. Ejemplos de asociación en el archivo XDC y la entidad (archivo principal VHD)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Tipo | Archivo XDC | Pin en XDC | Descripción |
| A | STD\_LOGIC | B | E1 | Asocia solo un pin |
| Led | STD\_LOGIC\_VECTOR(2 DOWN TO 0) | Led(0) | F6 | Asocia solo un pin |
| - | - | Led(1) | G6 | Asocia el siguiente bit |

El siguiente y ultimo paso es descargar la aplicación en la tarjeta. Asociados los pines de entrada y de salida podemos validar que lo diseñado (y simulado) se comporta igual que en la vida real, para esto simplemente conectamos la tarjeta y vamos modificando las entrada para ver su comportamiento real.

**Laboratorio**

*Ejemplo de diseño – Compuerta XOR:*

* Para realizar el siguiente ejemplo realizaremos una compuerta XOR básica de 1 bit de entrada
* Simularemos la compuerta

*Compuerta XOR*

Para empezar a entender la ideología de diseño empezaremos a realizar una compuerta XOR en VHDL, si bien es un circuito básico y está embebido en las librerías de IEEE, este ejemplo nos servirá de base para tener referencia pues es un diseño sencillo que cubre lo esencial de diseño digital.

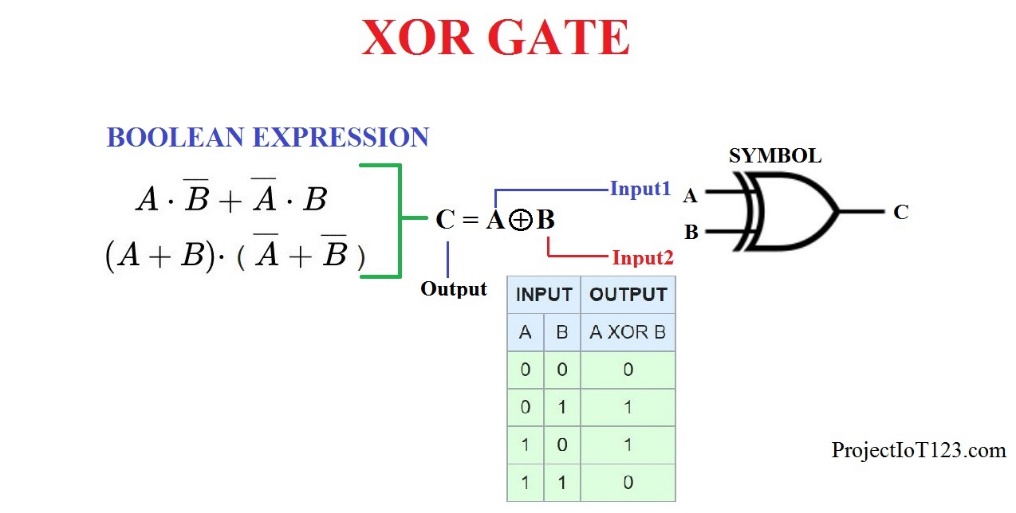


Figura 6. Compuerta XOR. Esquemático de la compuerta XOR acompañado de compuertas básicas, NOT, ANDO y OR. Adicionalmente se muestra la tabla de la verdad.

*Creación del Proyecto*

1 - Abrir Vivado en la pantalla principal y presionar en crear proyecto

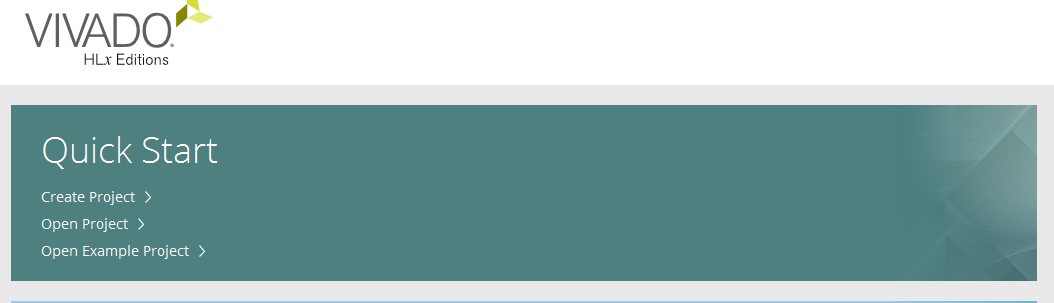


Figura 7 – Creación del proyecto en Vivado.

2 – Hacer click en siguiente

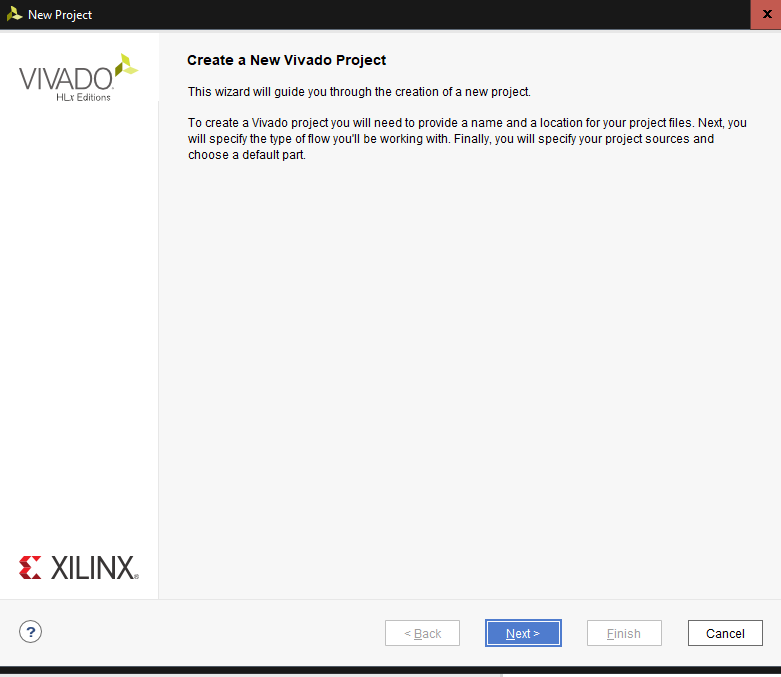
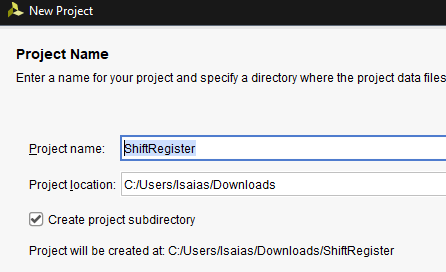


Figura 8 – Ventana de nuevo proyecto.

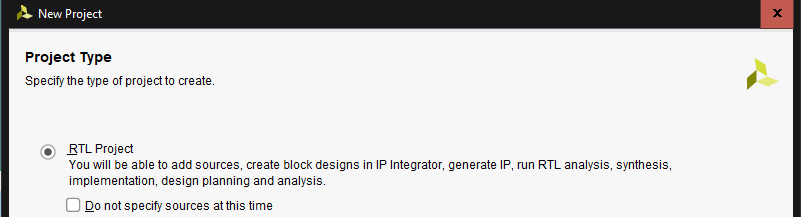
3 – Crear el nuevo proyecto en la carpeta de su conveniencia, asegúrese de tener la marca (checkbox) para que cree el subdirectorio.



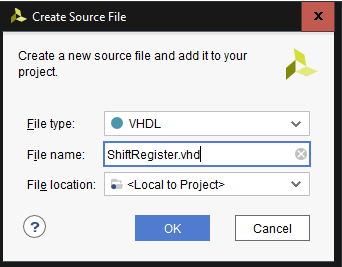
XorGate

Figura 9 – Ventana de Nombre de Proyecto.

4 – Todos los proyectos que crearemos serán de RTL (Register Transfer Logic)

  
Figura 10 – Ventana de Tipo de Proyecto.

5 – Seleccione crear un archivo y llamelo XORGate.vhd



XorGate.vhd

Figura 11 – Ventana de Archivo fuente (VHDL)

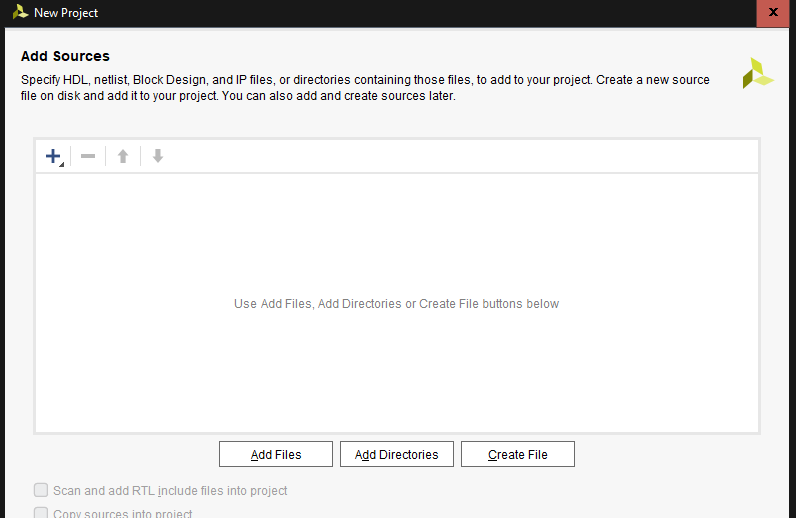
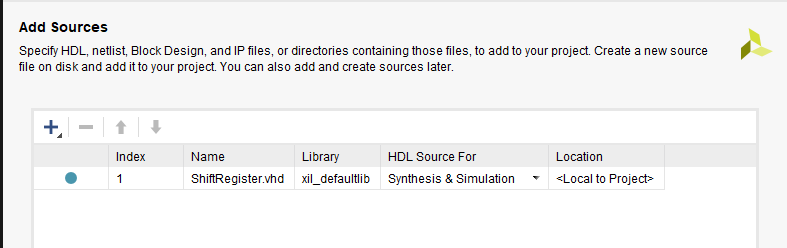


Figura 12 – Ventana de adición de fuentes. Aquí se muestra el archiv a añadir.

6 – Creado el archivo navegue al siguiente paso (7)



XorGate.vhd

Figura 13 – Ventana de adición de fuentes. Con archivo añadido.

7 – No añadiremos ninguna archivo de limitantes por el momento, por lo que podemos cambiar a la siguiente pantalla.

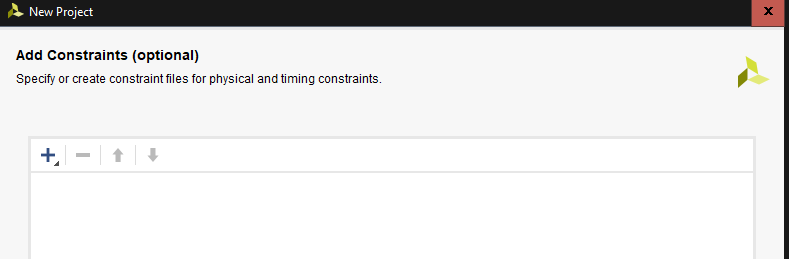


Figura 14 – Ventana de limitantes. Se explicará más adelante para que sirve.

8 – Busque su tarjeta relacionada dependiendo del FPGA. En mi caso es para la Arty-A7

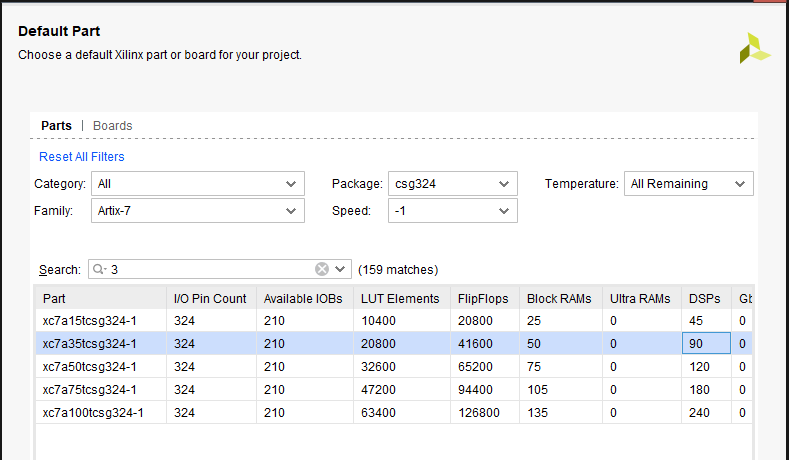


Figura 15 – Ventana para seleccionador de IC. Cada IC posee características diferentes en capacidad de LUTs, DSP, RAM, etc.

9 – Haga click en finalizar

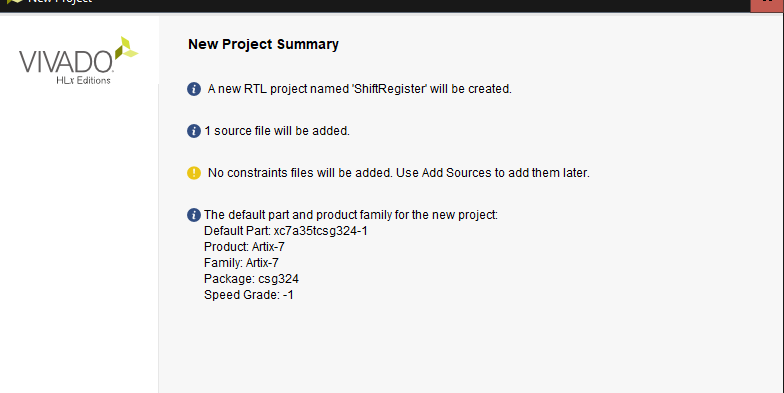
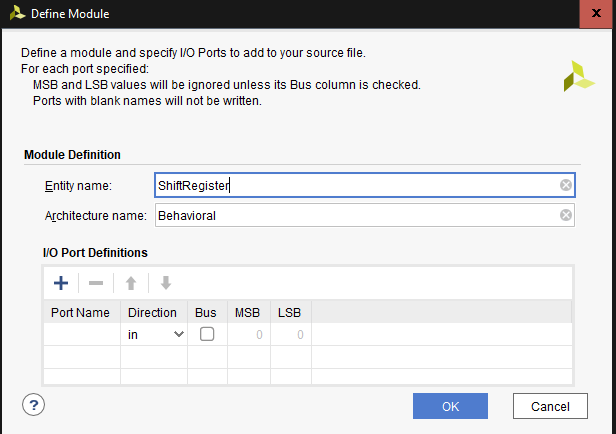


Figura 16 – Completado el proyecto luego podemos iniciar a codificar.

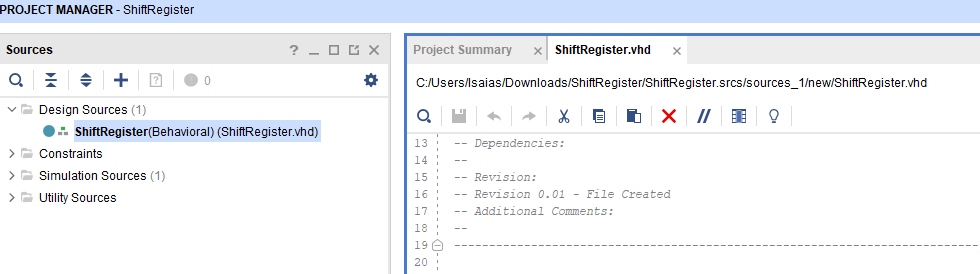
10 – Nos pedirá definir las entradas y salidas pero nosotros pasaremos este punto por alto para completar.



XorGate

Figura 17 – Definición de Módulo. Este paso ayuda al generador automático a realizar la configuración del archivo VHD.

11 – Al ir al menú principal podemos abrir el archivo VHD y poder codificar



XorGate

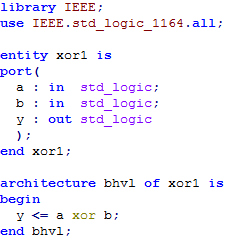
XorGate

XorGate

Figura 18 – Ventana de Project Manager. Desde aquí se pueden acceder al proyecto o definir generalidades.

12 – Cree el siguiente código en VHDL

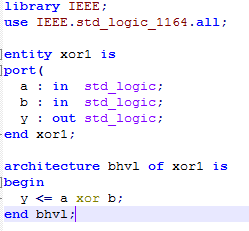
Listado 1 – Código en VHDL para XOR.



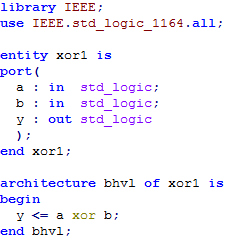
Ahora explicaremos cada parte:

La librería más común que incluye todos los paquetes o bloques básicos es la librería IEEE, esta está subdividida en otros paquetes.

Normalmente la descripción o llamado es use.<libreria>.<paquete>.<bloque\_o\_funcion>. Puede observar que dentro de este paquete se encuentran varias funciones básicas. Ir al sitio <https://bit.ly/2vqtyAv> para verificar



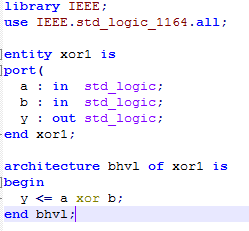
La entidad es el bloque, solamente define las entradas o salidas del sistema y es lo único que se puede detallar aquí. Note que hemos llamado XOR1 a nuestra entidad debido a que entraría en conflicto con nuestra compuerta XOR primitiva en el paquete STD\_LOGIC\_116



Observe también que dentro de la entidad existe un token (variable reservada, p.e. entity, end, if, son variables reservadas) llamado Port, dentro de este definimos cuales son entradas o salidas. También observe que la última línea de port no lleva ‘;’ y que luego del paréntesis sí.

Finalmente la entida es cerrada con el token *end*.

Luego de definir la entidad tenemos que darle vida a este bloque, es aquí donde entra la parte de funcional de comportamiento. Architecture provee que podamos definir el cuerpo de nuestro archivo VHDL declaranto el nombre de tipo *bhvl* el cual puede ser cambiado por cualquier nombre, seguido del token *of* y luego si es mandatorio el nombre de la entidad, que tiene que coincidir con la utilizada a nivel superior, en este caso *xor1*.



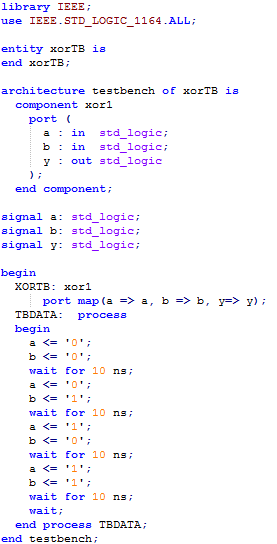
Abrimos la declaración de la entidad con el token *begin* y finalmente realizamos la lógica de la función de nuestra compuerta. El último paso es cerrar la entidad con end y el nombre de la arquitectura, que tiene que conicidir con el nombre de la arquitectura inicial, en este caso *bhvl*.

Otro detalle a destacar son los errores, estos se van marcando en rojo cada vez que tenemos un error, esto nos ayuda a depurar nuestro archivo.

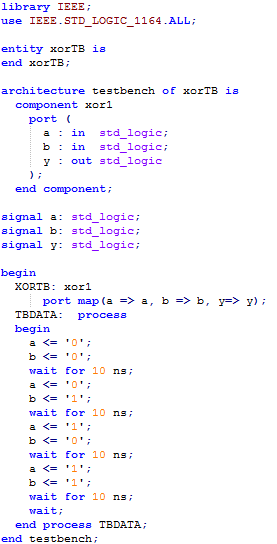
Terminado este paso lo siguiente es realiar nuestro componente archivo de simulación de VHDL.

13 – Cree el siguiente archivo VHD que representa el test bench dentro del proyecto (siga los pasos anteriores para creación de documentos .VHD)

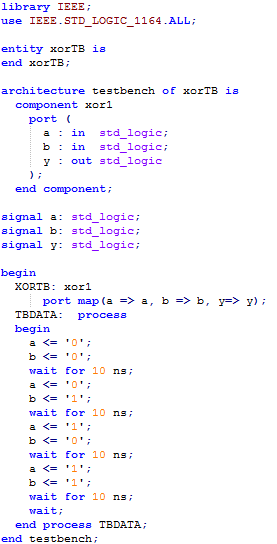
Listado 2 – Listado para Test Bench. El test bench es un archivo para probar la arquitectura a base de la entidad.



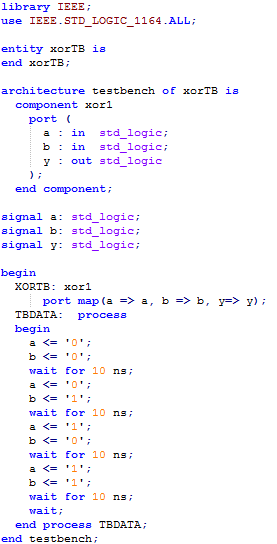
Ahora explicaremos cada línea o segmento de código VHDL para que se entienda su objetivo. Las primeras dos línas son conocidas, librerías y paquetes necesarios para construir nuestro testbench en este caso.



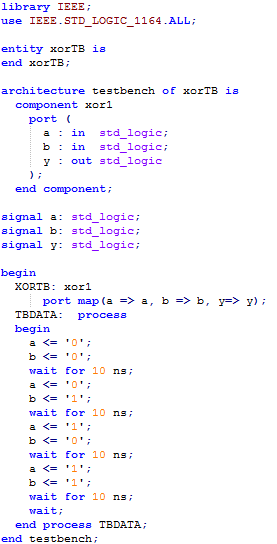
Como este es un archivo de simulación necesitamos crear una entidad vacía. Esta entidad solamente es para sociar los diferentes componentes VHD a probar.



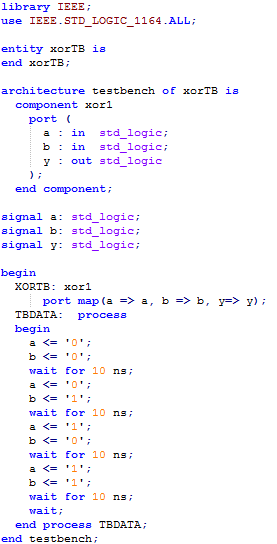
Ahora empezamos a definir los componentes internos dentro de la arquitectura, note que ahora le ubicamos un nombre a la arquitectura llamado *testbench*, como aclaramos en secciones anteriores este nombre es solamente una etiqueta. Como vamos ahora a definir el comportamiento de nuestra simulación tenemos que asociarlo a la entidad vacía *xorTB*. Creamos el componente, del mismo nombre de la xor que tenemos en el otro archivo VHDL, con misma cantidad de entradas y salidas, lo último es finalizar el componente.



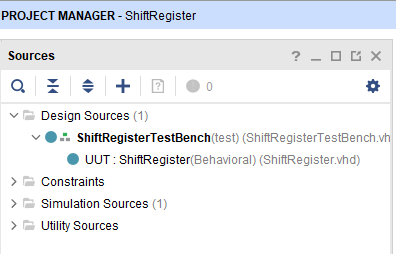
Seguido tenemos estas señales, las señales son conexiones internas en un archivo VHDL, recuerde de sus clases pasadas de digitales que un componente combinacional depende de entradas y salidas, pero que internamente otras salidas viajan a otras entradas y llevan la señal de un punto a otro. Este sería el caso, sin embargo estas señales solamente van asociadas a la entidad en sí.



Lo último es declarar la entidad, note que hacemos algo llamado port mapping, aquí lo definimos de la siguiente manera <Etiqueta>:<Entidad> port map (<entradas y salidas>);. Al final declararemos un proceso llamado TBDATA, este simplemente son los estímulos que recibe en el tiempo, hemos declarado entonces las combinaciones para las entradas que serán 00, 01, 10, 11 y esperaremos 10ns entre cada cambio. Finalmente esperamos o anclamos la simulación con la sentencia wait, de lo contrario nunca terminaría.



14 – El Project manager debe lucir como sigue:



Xor(Behavioral) (Xor.vhd)

**XorTB**(test) (XorTB.vhd)

Figura 19 – Archivos VHD de Shift Register y Test Bench. Note que el archivo de registro de corrimiento (test bench) contiene la entidad.

15 - Seguidamente corra la simulación

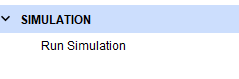


Figura 20 – Ejecución de Simulación

16 – Se debe observar la ventana del lado derecho al finalizar como sigue

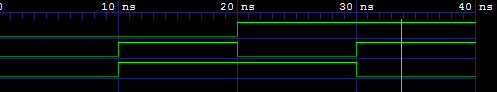


Figura 21 – Simulación. Actualmente se muestra solo una parte de la simulación.

17 – Presione Botón derecho sobre la simulación y deje la ventana en Full View o Ctrl+0

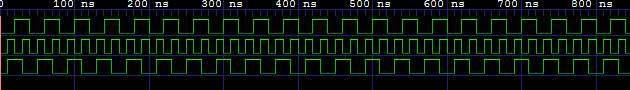


Figura 22 – Se muestra luego de hacer “fit” en la ventana toda la corrida de simulación en VHDL.

Valide que lo diseñado responde apropiadamente es decir, la XOR se cumple si las entradas son diferentes.

19 – Utilice los botones de simulación y navegue por toda la pantalla para observar como navegar en Vivado con la herramienta.

Finalizados con la simulación lo siguiente es generar el archivo necesario para el FPGA y que pueda ser descargado en hardware.

20 – A continuación detallamos las partes más importantes de la tarjeta Arty como la fuente de poder, puertos de I/O e interfaces, esto nos servirá más adelante para declarar las salidas en el archivo XDC.

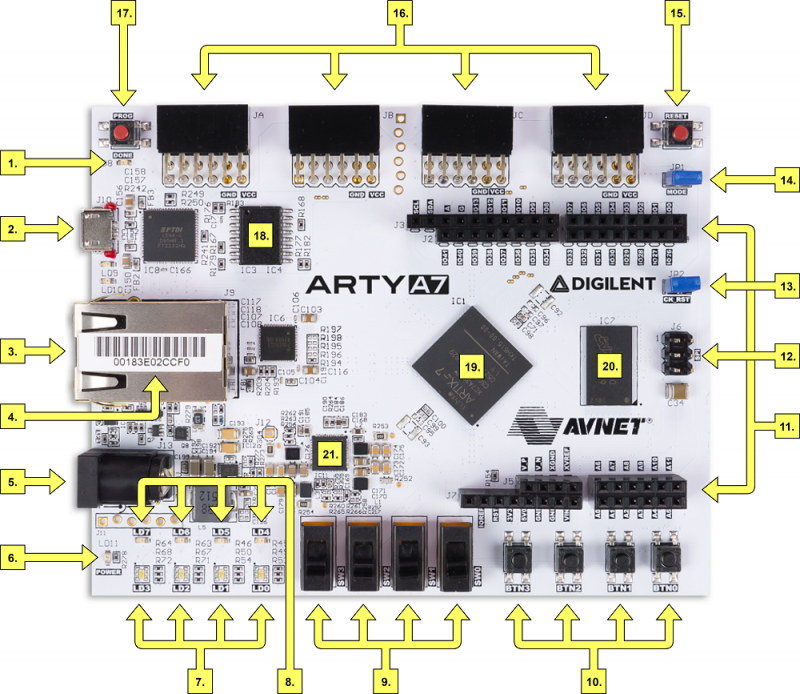
Primeramente mostramos los componentes que posee nuestra tarjeta y donde están ubicados.

Figura 23 – Componentes de Tarjeta Arty A7

Tabla 2 – Documentación de Componentes de Arty-A7

| **Callout** | **Description** | **Callout** | **Description** | **Callout** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | FPGA programming DONE LED | 8 | User RGB LEDs | 15 | chipKIT processor reset |
| 2 | Shared USB JTAG / UART port | 9 | User slide switches | 16 | Pmod connectors |
| 3 | Ethernet connector | 10 | User push buttons | 17 | FPGA programming reset button |
| 4 | MAC address sticker | 11 | Arduino/chipKIT shield connectors | 18 | SPI flash memory |
| 5 | Power jack for optional external supply | 12 | Arduino/chipKIT shield SPI connector | 19 | Artix FPGA |
| 6 | Power good LED | 13 | chipKIT processor reset jumper | 20 | Micron DDR3 memory |
| 7 | User LEDs | 14 | FPGA programming mode | 21 | Dialog Semiconductor DA9062 power supply |

21 - Dados los componentes de la tarjeta ahora notaremos en referencia cuales son las entradas y salidas.

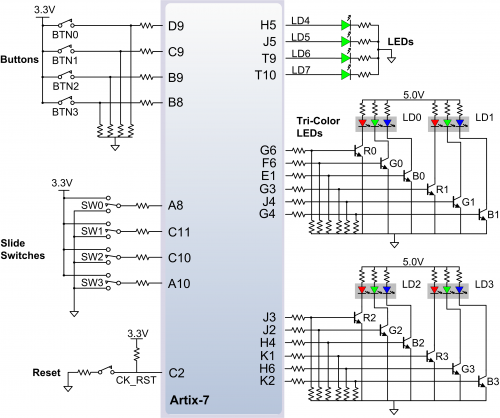
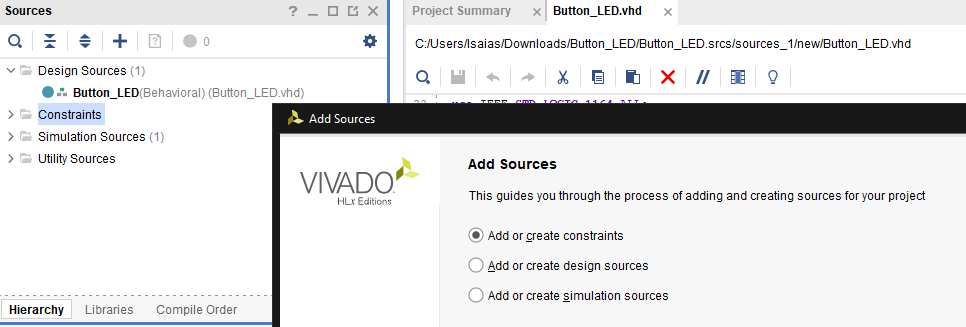


Figura 24 – Muestra de ubicación de los botones y LEDs en la Arty A7

22 – Crear el archivo de “constraints” o limitantes, que son necesarios para síntesis.

Situarse sobre Constraints y presionar add sources.

Agregar el archivo de constraints



XorGate

XorGate

Figura 25 – Creación del archivo de “constraints”.

23 - Agregar el archivo de Arty para constraints

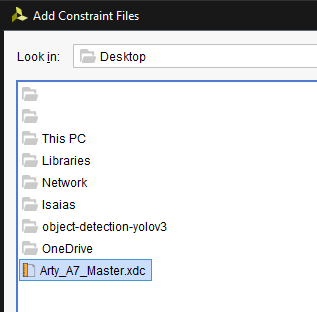


Figura 26 – Copia del archivo de definiciones.

24 - Asegurese de tener el click seleccionado para la copia del archivo maestro de constraints

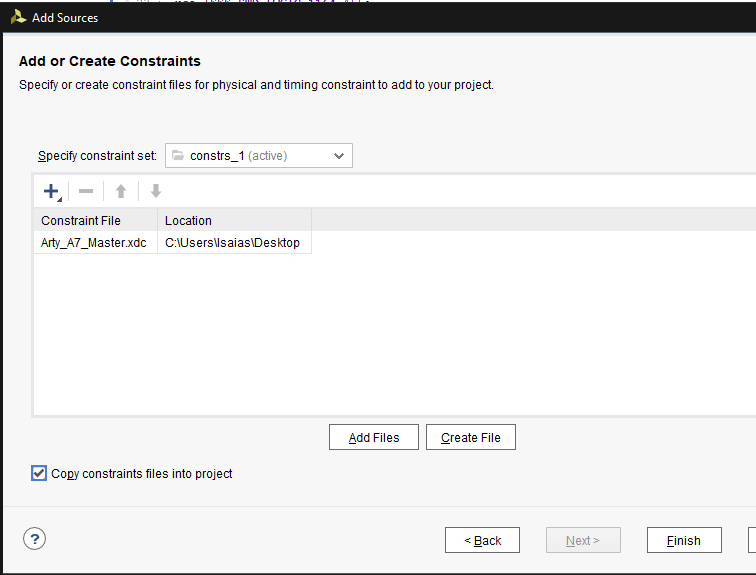


Figura 27 – Archivo de definiciones a copiarse a la carpeta de proyecto.

25 - Verifique la ventana de Constraints y observe que se añadió al proyecto

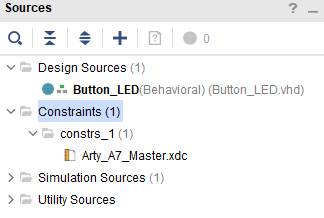


Figura 28 – Verificación del archivo de definiciones/limitantes (constraints).

26 - Abrir el archivo XDC y habilitar la salida ‘y’ y las entradas ‘a’ y ‘b’

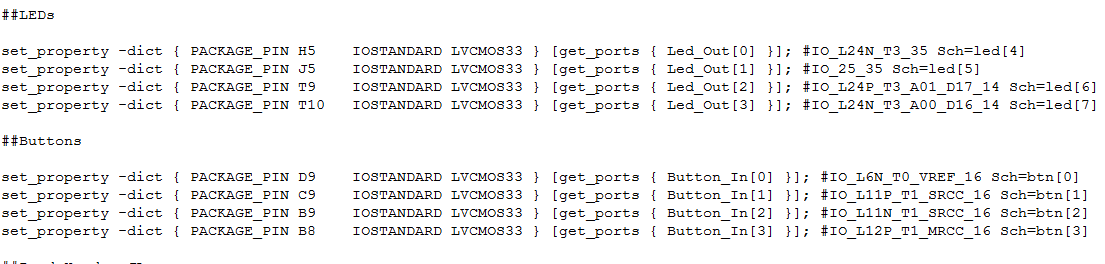
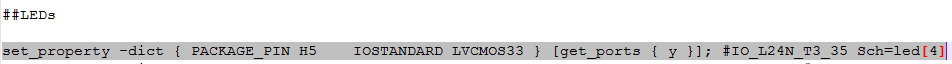


Figura 29. Archivo original Master de la Arty, Archivo XDC.



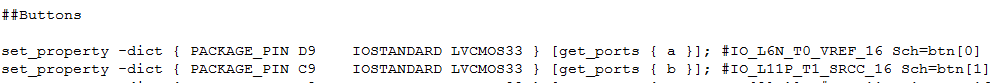


Figura 30. Archivo modificado Master de la Arty, Archivo XDC. Note ‘a’, ‘b’, ‘y’.

27 – Genere el archivo de salida para programación del FPGA

En este punto estamos listos para genera el archivo que en Xilinx es un bit file, necesario para que nuestro FPGA opere.

Para esto necesitamos un archivo en el nivel superior. En este caso el archivo de Xor.

Adicionalmente un archivo de constraints que relaciona los pines con su proyecto.

Seguidamente realizar la corrida de síntesis. Lo que hace la síntesis es verificar que la sintaxis y asignación es la correcta, sería el equivalente a compilación en microcontroladores.

Cuando la síntesis completa enteramente significa que el sintetizador genero el diseño sin inconvenientes.

Si su diseño no cabe (por cantidad lógica de compuertas o LUT) entonces la síntesis fallará. Para este diseño sencillo quedará emplazado en el componente pues es de pocos recursos.

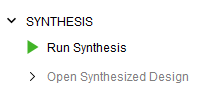


Figura 31 – Ejecución de Síntesis. Este es uno de los procesos más importantes en el diseño digital con FPGAs.

28 – La síntesis en ejecución puede verse en la pantalla Project Summary, y tenemos entonces que nos puede dar advertencias las cuales debemos atender si son muy importantes, de lo contrario, la ventana se muestra como sigue:

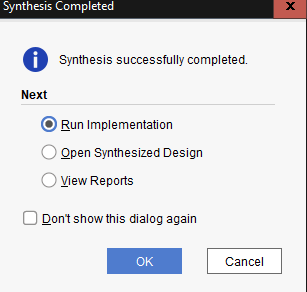


Figura 32 – Finalización de síntesis satisfactoria. Puede dar advertencias.

29 – Como se mencionó anteriormente la síntesis puede pasar pero puede ser que al correr la implementación no quepa en el FPGA. Seguidamente se muestra la pantalla de información de la corrida de implementación.

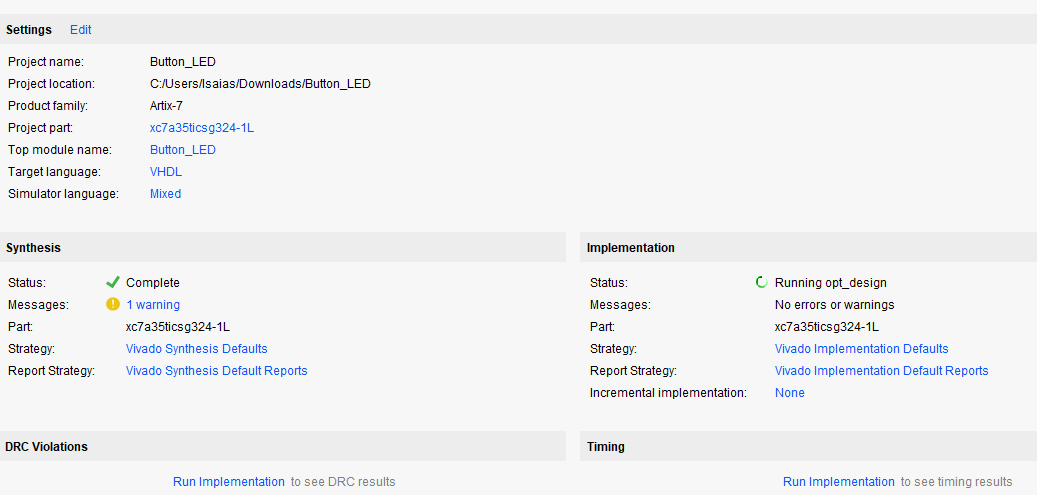


Figura 33 – Pantalla de sumarización de proyecto. Actualmente con ejecución de implementación.

30 – Debería mostrarse la siguiente pantalla si completa exitosamente

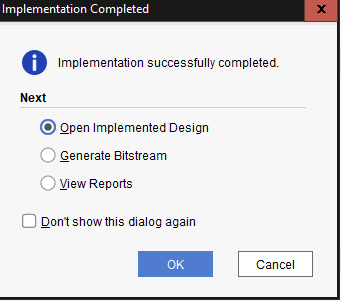


Figura 34 – Implementación Exitosa. Luego de este paso se puede ir a generar el bitstream.

31 – Para generar el archivo de salida en el FPGA tenemos que generar el archivo bitstream seleccionando su implementación. Nuevamente, en la ventana de Project summary se puede ver el resultado de esto.

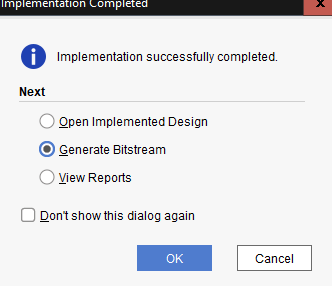


Figura 35 – Generación del Bitstream. El bitstream es el último paso del flujo de diseño continuo para programación de la tarjeta.

32 – Seguidamente se abrirá la siguiente ventana cuando complete y nos preguntará la siguiente situación, en la cual eligiremos abrir el hardware manager.

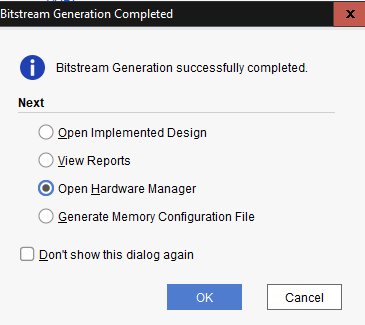


Figura 36 – Completada la fase de Bitstream. Ahora procedemos a realizar otra acción, p.e., la acción final sería implementar en la tarjeta.

33 – El hardware manager es el proceso final de la implementación del diseño en la tarjeta real y se observa en la ventana principal si se posee una tarjeta de desarrollo de Xilinx.

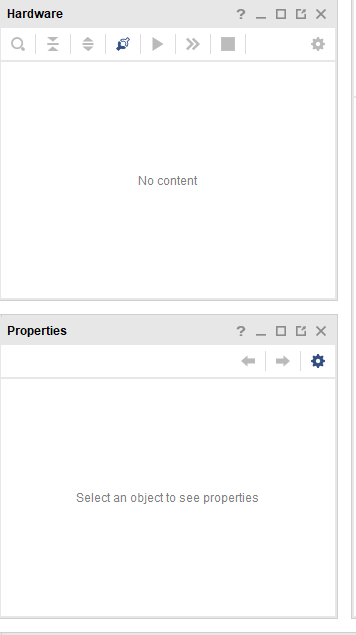


Figura 37 – Ventana de Hardware Manager. Actualemente no se tiene conectada la tarjeta, se verificará más adelante su implementación.

Por el momento cerraremos esta ventana y observaremos que advertencias nos dio el proyecto para ver que tan importantes son.

34 – Una de las advertencias es como sigue:

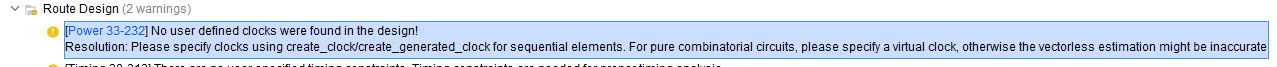


Figura 38 – Ventana de Mensajes. Esta ventana está en la parte inferior de su proyecto. Muestra advertencias, errores, logs y demás situaciones relevantes al diseño.

Esto no significa que no funcionará el diseño sino es una señal que nos avisa (del logger de Vivado) que no hemos implementado una señal de reloj, lo cual es muy común para VHDL

35 – Para programar la Arty debe asegurarse de energizarla por el cable USB. Verificar que al conectar se encienda el LED DONE (LD11) que es el de alimentación de la tarjeta.

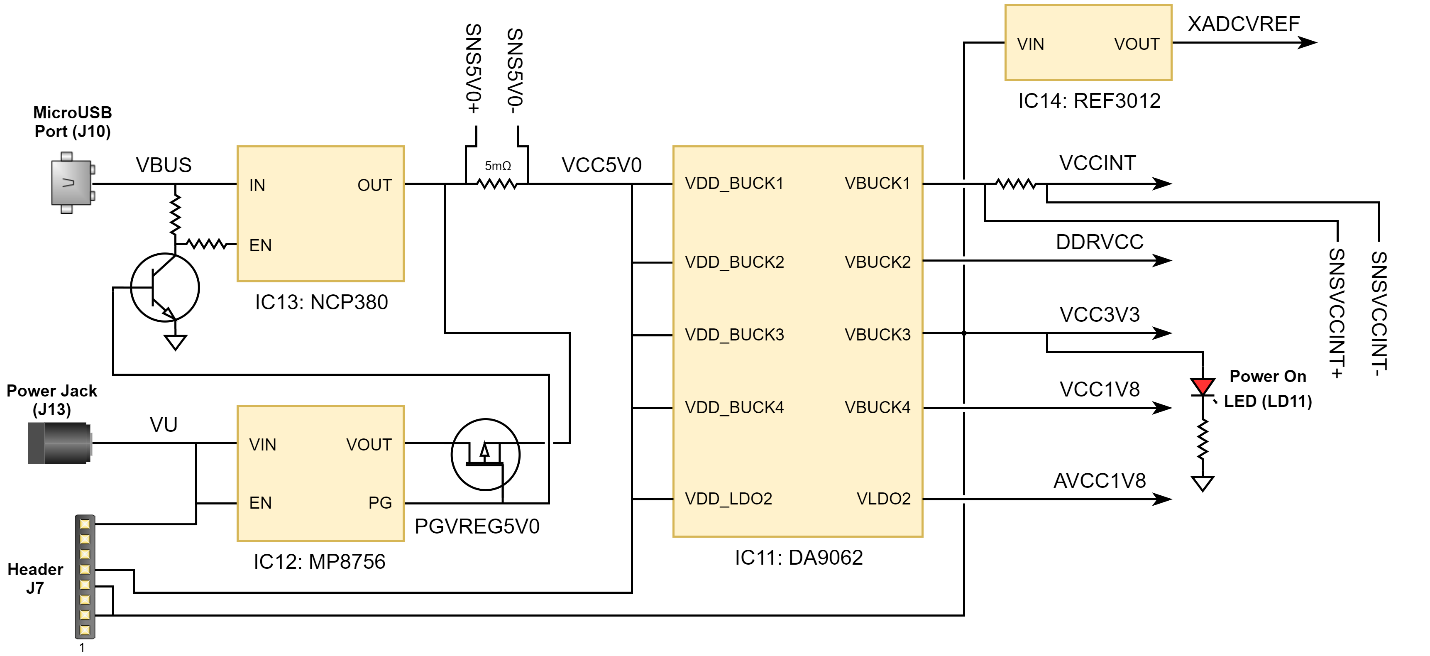


Figura 39 – Esquemático de circuito de fuente de poder de Arty A7.

Programación del BitStream en el FPGA:

36 – Conecte primeramente su tarjeta por medio del cable micro USB a la PC

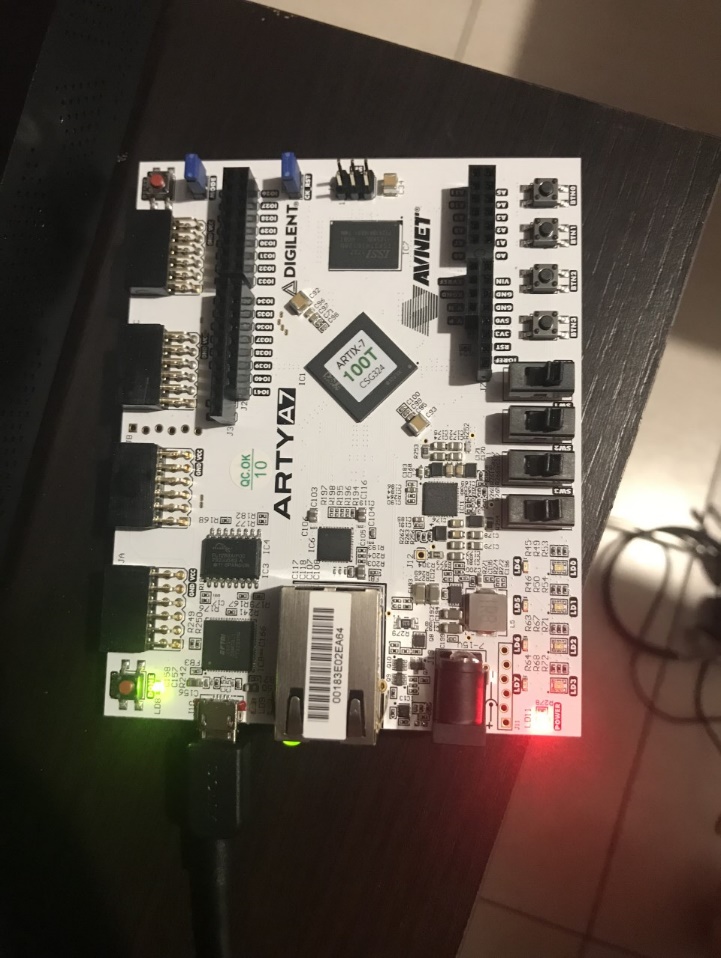


Figura 40 – Muestra de nuestra tarjeta Arty A7. Cable de USB conectado.

Debe observar los LEDs rojo (PWR) verde (no programado) y el de Ethernet (no importante aún).

37 – Abrir la ventana de Hardware Manager

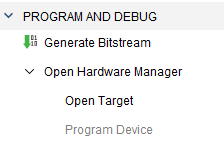


Figura 1.39 – Fase final, apertura del Programador.

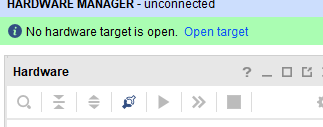


Figura 41 – Ventana de detección del dispositivo.

38 – Hacer click en Open Target y luego en Auto-Connect. Esta función solo funcionará en Vivado con tarjetas de Xilinx. Cualquier otro fabricante, aunque el proceso es el mismo, los archivos generados y las herramientas son diferentes sin embargo, el flujo de diseño en FPGAs es coincidente.

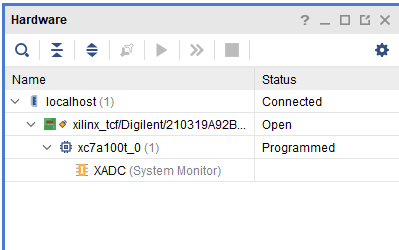


Figura 42 – Ventana de Auto Conexión. Observamos que ha detectado el componente.

39 – Encontrada la tarjeta observamos de la figura anterior que el FPGA viene programado de fábrica con una configuración. Nosotros la sobreescribiremos. Hacer click en Program Device.

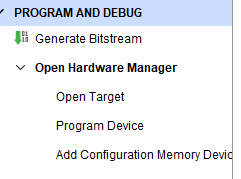
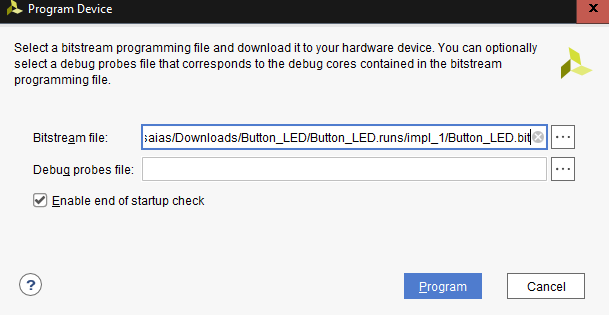


Figura 44. Ventana de Programación. Al realizar Program device realizaremos la copia del archivo en la tarjeta.

40 – Seleccione program Device. Busque el archivo de configuración en la carpeta de proyecto en runs/impl\_1. El archivo \*.bit estará dentro de esta carpeta.



/XorGate/XorGate.runs/impl\_1/XorGate.bit

Figura 45 – Ventana de Selección del BitStream. Como observa las impleemtaciones están en la carpeta \*.runs/impl\_\*/\*.bit.

41 – En caso personal, se tuvo que realizar nuevamente la sintetización, implementación y síntesis debido a que la pieza escogida no era la adecuada referente a la que se debía programar

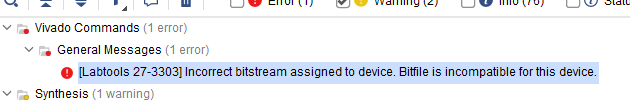
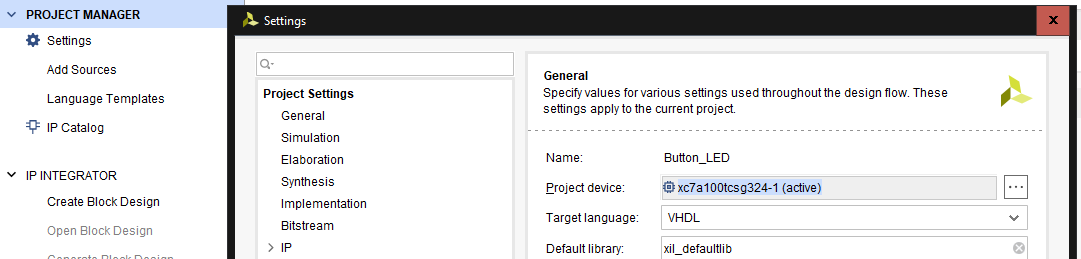


Figura 46 – Error en escritura del bitstream. Como el componente es diferente al programar, por consiguiente el bitstream generado no encaja en el FPGA, se debe de volver a realizar los procesos.



XorGate

Figura 47 – Cambio de pieza. Esto se puede realizar fácilmente en los ajustes del proyecto, opción del dispositivo de proyecto.

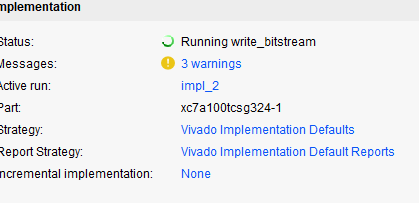


Figura 48 – Resultado de la ejecución de implementación.

42 – El resultado de la implementación ha sido exitoso y ha sido programado el FPGA

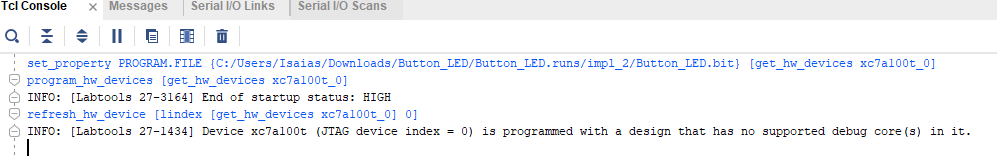


Figura 49 – Resultado final de la programación del dispositivo.

43 – Pruebe la tarjeta:

* Cambie el switch o botón (depende de que haya puesto en el XDC) a un estado logico
* Valide la tabla de verdad
* Observe la Observe el LED que se apague o encienda.



Figura 50 – Test del programa de botones y LEDs.

Evaluación Laboratorio 1:

25% - Completar todos los pasos anteriores:

* Simulación de registro de corrimiento.
* Ejecución del programa XOR en un FPGA.

50% - Realizar los siguientes cambios

* Generar una XOR de 1 bit en VHDL, simular y descargarlo al FPGA utilizando la función lógica
  + A!B + B!A
* Generar una XOR de 2 bits en VHDL, simular y descargarla al FPGA
* Generar una XOR utilizando la sentencia IF (puede también ayudarse con ELSE)

25% - Realizar los siguientes cambios y utilizar las siguientes tabla para entradas y salidas (cada persona tiene asignada entradas y salidas diferentes a completar,

* Generar las siguientes compuertas de 1 bit y simular, utilizar la tabla inferior
  + AND, OR, NOT
* Generar la siguiente combinación, simular solamente, no utilizar la tabla inferior.
  + Y = AB + BC + !C!D

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | A (SW) | B (SW) | Enable (BTN) |
| 1 | Y | 1 | 2 | 0 |
| 2 | LED | 3 | 2 | 1 |
| 3 | LED | 0 | 1 | 0 |
| 4 | LED | 0 | 2 | 1 |
| 5 | LED | 0 | 3 | 2 |
| 6 | LED | 3 | 1 | 2 |
| 7 | LED | 0 | 1 | 3 |
| 8 | LED | 1 | 0 | 2 |
| 9 | LED | 0 | 2 | 0 |
| 10 | LED | 1 | 2 | 2 |
| 11 | LED | 0 | 2 | 3 |
| 12 | LED | 2 | 1 | 1 |
| 13 | LED | 3 | 1 | 1 |
| 14 | LED | 2 | 3 | 2 |
| 15 | LED | 3 | 2 | 2 |
| 16 | LED | 3 | 2 | 1 |
| 17 | LED | 3 | 2 | 0 |