Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000

Oscar Gabriel Fuentes Lanfur
Asesorado por el Ing. Eddy Medinilla

Guatemala, mayo de 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

OSCAR GABRIEL FUENTES LANFUR
ASESORADO POR EL ING. EDDY MEDINILLA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICA

GUATEMALA, MAYO DE 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Luis Diego Aguilar Ralón
VOCAL V	Br. Christian Daniel Estrada Santizo
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. José Aníbal Silva de los Angeles
EXAMINADOR	Ing. Walter Giovanni Álvarez Marroquín
EXAMINADOR	Ing. Byron Odilio Arrivillaga Méndez
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica con fecha 10 de enero de 2019.

Oscar Gabriel Fuentes Lanfur

Guatemala 20 de febrero, 2019

Ingeniero Otto Fernando Andrino González
Director de Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Por medio de la presente me permito informarle que he procedido a revisar el trabajo de graduación titulado **"DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000"** elaborado por el estudiante Oscar Gabriel Fuentes Lanfur quien se identifica con el número de DPI 2621 62296 0101 y carnet 201403662. A su vez, quiero mencionar que el mismo cumple los objetivos trazados de acuerdo con el protocolo presentado, por lo que le doy por APROBADA. De tal manera, se solicita darle trámite correspondiente.

Atentamente,
Ing. Eddy Augusto Medinilla Rodríguez
Colegiado No.10576

Ing. Eddy Medinilla
Colegiado 10576

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 11 de marzo de 2019

Señor Director
Ing. Otto Fernando Andrino González
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado **DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000,** desarrollado por el estudiante **Oscar Gabriel Fuentes Lanfur**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Julio César Solares Peñate
Coordinador de Electrónica

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 17. 2019.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación de la estudiante: OSCAR GABRIEL FUENTES LANFUR titulado: DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000, procede a la autorización del mismo.

Ing. Otto Fernando Andrino González

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 27 DE MARZO 2019.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 250.2019

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO DEL CURSO INTRODUCTORIO AL DESARROLLO DE SISTEMAS BASADOS EN FPGA SYSTEM-ON-CHIP UTILIZANDO VIVADO, XILLINUX Y XILLYBUS SOBRE LA TARJETA DE DESARROLLO ZYBO ZYNQ-7000,** presentado por el estudiante universitario: **Oscar Gabriel Fuentes Lanfur,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, mayo de 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

/gdech

ACTO QUE DEDICO A:

Dios	Por darme la inspiración y la fuerza, por cuidarme y guiarme a lo largo de mi carrera.
Mis padres	Oscar Fuentes y Patricia Lanfur de Fuentes, su amor será siempre mi inspiración.
Mi hermano	Rodrigo Fuentes, por ser una importante influencia en mi carrera y por todo el apoyo brindado.
Mis tíos y primos	Por su gran apoyo y por el voto de confianza hacia mi persona.
Mis amigos	Por estar en las buenas y en las malas apoyándonos mutuamente.
Mi novia	Marielena Ramazzini, por su apoyo, su compresión y todo el tiempo compartido.

AGRADECIMIENTOS A:

Dios	Por cada día más de vida y su apoyo incondicional.
Mis padres	Por su sacrifico; y por haberme forjado como la persona que soy en la actualidad.
Universidad de San Carlos de Guatemala	Por todos los conocimientos, las experiencias y las habilidades adquiridas en el transcurso de la carrera.
Facultad de Ingeniería	Por ser una importante influencia en mi carrera y por permitir mi desarrollo profesional.
Asesor de tesis	Ingeniero Eddy Medinilla, por compartir su experiencia y sus conocimientos para el desarrollo de este trabajo de graduación.
Coordinador del Laboratorio de Electrónica	Ingeniero Byron Arrivillaga, por permitirme impartir el Laboratorio de Electrónica 6 y realizar este trabajo de graduación enfocado en la innovación del Laboratorio de Electrónica 6.
Ing. Iván Morales	Por compartir sus conocimientos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

LISTA DE SÍMBOLOS

Símbolo	Significado
FPGA	Arreglo de compuertas lógicas programables
VGA	Arreglo de gráficas de video
PLL	Bucles de enganche de fase
USB	Bus de datos serial universal
ASIC	Circuito integrado de aplicación específica
ASCII	Código estándar americano para el intercambio de información
LED	Diodo emisor de luz
GPIO	Entradas y salidas de propósito general
LUT	Estructura o vector que relaciona dos tipos de datos
GB	Gigabytes, unidad de información base
HDMI	Interfaz multimedia de alta definición
VHDL	Lenguaje de descripción de hardware.
RAM	Memoria de acceso aleatorio
SRAM	Memoria de tipo S de acceso aleatorio
DSP	Procesamiento de señales digitales
UART	Protocolo de recepción y transmisión asíncrona universal
EEPROM	ROM programable y borrable eléctricamente
SoC	Sistema integrado completo
CPU	Unidad central de procesamiento
MHz	Unidad de medida para la frecuencia

GLOSARIO

Alt	Tecla modificadora que se encuentra cerca de la esquina inferior izquierda.
Bits	Dígito binario que representa una unidad de información.
Consola	Programa que se utiliza para interactuar con una computadora por medio de la línea de comandos.
Ctrl	Tecla modificadora que se encuentra en la esquina inferior izquierda.
Enter	Tecla central, permite ejecutar un comando previamente escrito.
Hardware	Describe todos los circuitos integrados físicos dentro de un dispositivo.
Instanciación	Proceso por el cual se conectan módulos entre sí.
Ip Core	Segmento de código previamente programado dentro de un módulo.
Linux	Sistema operativo libre de multiplataforma.

Memoria micro SD	Dispositivo de almacenamiento de datos digitales, formato de tarjeta de memoria flash.
Micro Zed	Tarjeta de desarrollo avanzada de bajo costo.
Problema	Se dará si el concepto es demasiado largo por lo que deberá corregir los tabuladores.
Vivado	Entorno de desarrollo enfocado a la programación de los FPGA.
Xillinx	Empresa privada creadora y distribuidora de FPGA.
Xillinux	Sistema operativo basado en Linux para los FPGA.
Xillybus	Ip Core encargado de comunicar el CPU con el FPGA.
Zed Board	Tarjeta de desarrollo avanzada de bajo costo.
Zybo	Tarjeta de desarrollo avanzada de bajo costo.

RESUMEN

El siguiente trabajo de graduación presenta el diseño de un curso introductorio al desarrollo de sistemas de alto nivel basados en FPGA utilizando la tarjeta de desarrollo Zybo-Zynq-7000; se enseñarán los conceptos básicos de programación implementando un lenguaje de descripción de hardware sobre el entorno de desarrollo Vivado Design Suite. Todo el trabajo de graduación se desarrolla utilizando de Vivado enfocado al desarrollo de aplicaciones para la gama media de FPGA. Se enseñará a crear nuevos proyectos, realización de síntesis, implementación y generación de *bitstream* a nuevos módulos.

Se enseñará a como instalar el sistema operativo Xillinux sobre un procesador Cortex A9 propio de la Zybo-Zynq-7000 y la comunicación entre el sistema operativo y la FPGA. La Zybo-Zynq-7000 es una tarjeta de desarrollo capaz de combinar procesos programados dentro de una unidad central de procesamiento y la tecnología FPGA. Se explicará el funcionamiento de la transmisión de datos entre el CPU y la FPGA y se aprenderá a configurar el Ip Core Xillybus (Ip Core encargado de la comunicación entre las dos tecnologías).

Se agregaron prácticas educativas para que el estudiante puede realizar programas experimentales y aprender a utilizar Vivado con el objetivo de reforzar la teoría. El desarrollo del trabajo de graduación se encuentra orientado al Laboratorio de Electrónica 6 de la Universidad San Carlos de Guatemala.

OBJETIVOS

General

Diseñar un curso avanzado dedicado a la enseñanza de tecnologías para el desarrollo de aplicaciones utilizando los FPGA de gama alta.

Específicos

1. Desarrollar la sintaxis y lógica para el diseño de aplicaciones en FPGA utilizando el lenguaje de programación VHDL.

2. Desarrollar sentencias concurrentes y secuenciales en FPGA utilizando VHDL dentro del entorno de desarrollo Vivado.

3. Desarrollo de máquinas de estado (FSM por sus siglas en ingles) en FPGA utilizando VHDL dentro del entorno de desarrollo Vivado.

4. Desarrollar memorias y vectores en FPGA utilizando VHDL dentro del entorno de desarrollo Vivado.

5. Presentar la metodología de instalación y funcionamiento del sistema operativo Xillinux.

6. Presentar la comunicación bidireccional entre el sistema operativo Xillinux y el FPGA utilizando el interface Xillybus.

7. Aplicar los conocimientos adquiridos de FPGA, VHDL, VIVADO, Xillinux y XIllybus para el desarrollo de aplicaciones que resuelven problemas del mundo real.

INTRODUCCIÓN

Desde 1984, la tecnología de arreglos de compuertas programables, nombrada por sus siglas en inglés FPGA, se ha utilizado enormemente en varias aplicaciones a lo largo de las décadas y que actualmente continúa siendo impulsada. Desde que Xillinx los invento los FPGA han pasado de ser circuitos lógicos sencillos a remplazar a los circuitos integrados de aplicación específica y hoy en día remplazan a los procesadores de señales y aplicaciones de control. Dicho tema desempeña un papel muy importante en diversas áreas de aplicación debido a que ya están siendo aplicadas en el ámbito laboral.

El desarrollo de este trabajo de investigación consiste en diseñar un curso introductorio orientado al Laboratorio de Electrónica 6 de la Facultad de Ingeniería enfocado en el desarrollo de sistemas basados en FPGA utilizando como programa principal Vivado y su respectivo leguaje de programación VHDL.

Se basa en verificar todos los aspectos necesarios para la implementación del curso e introducir al estudiante a la tecnología de punta; se comenzará con el contenido base orientado a la programación avanzada en FPGA y sus derivados como Xillinux y Xillybus. Se enseñará a utilizar el FPGA en paralelo con el CPU, ejecutando un sistema operativo Xillinux, especial para las FPGA incluyendo el bus de datos Xillybus. Todos estos aspectos propondrán una alternativa al Laboratorio de Electrónica 6 de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, más adecuada a la era moderna utilizando la tarjeta de desarrollo Zybo ZYNQ-7000.

1. ARREGLO DE COMPUERTAS PROGRAMABLES (FPGA)

1.1. Concepto

El nombre FPGA, o arreglo de compuertas lógicas reprogramables, es una tarjeta de desarrollo digital capaz de configurarse para cualquier aplicación. Contiene bloques de lógica configurables, conexiones internas programables que puede ser configurado en el instante por medio de un lenguaje de descripción de hardware. La lógica programable es capaz de implementar funciones simples como las de una compuerta lógica simple o un sistema de circuitos secuenciales hasta complejos sistemas en un integrado.

Los FPGA son circuitos hechos de silicio que permiten utilizar bloques de lógica configurables, compuertas lógicas y recursos de conexiones internas programables para realizar la conexión interna de la FPGA con el objetivo de implementar aplicaciones personalizas de alto nivel en hardware sin tener que utilizar ASIC. En el mercado actual, el costo las FPGA es mucho más alto comparado con un solo ASIC; sin embargo, si se comparan todas las implementaciones posibles de los FPGA contra las aplicaciones de los ASIC se tendrá que el precio de un FPGA es extremadamente bajo contra todos los ASIC juntos.

Los FPGA exceden la potencia de cómputo y velocidad de procesamiento que rompe el paradigma de ejecución secuencial y logran más en cada ciclo de reloj aprovechando del paralelismo del hardware. Los FPGA en la actualidad tienen la misma función que a los ASIC; sin embargo, el procesamiento de los FPGA es más lento, desde el punto de vista físico no pueden abarcar sistemas

muy personalizados como los ASIC y tienen un mayor consumo de energía. A pesar de esto, los FPGA son reprogramables otorgando flexibilidad al diseño del programa; comprende un costo de diseño y tiempo de implementación mucho menor con entornos de desarrollo amigables al usuario.

1.1.1. Xillinx

Xillinx es una empresa privada que se dedica a la distribución de tecnología de punta en el mundo. A Xillinx se le atribuye la invención de los FPGA en el mundo; fue la primera empresa en distribuir los primeros dispositivos lógicos programables. Empresa creadora de la tarjeta de desarrollo Zybo-Zynq-7000, el sistema operativo Xillinux, el entorno de desarrollo Vivado, entre otras tecnologías.

1.1.2. ASIC

En la antigüedad el su uso de ASIC era indispensable para la realización de circuitos que ejecutaran ciertas funciones específicas según la aplicación. Un circuito integrado para aplicaciones específicas (o ASIC, por sus siglas en inglés) es un circuito integrado elaborado para realizar una tarea única de la forma más perfecta posible, se pueden pedir a precio de mayoreo y comprende costos muy bajos.

Los ASIC están organizados y seccionados en función de su aplicación, por ejemplo, los circuitos integrados de la serie 74XX comprenden toda la gama de compuertas lógicas, circuitos combinacionales y circuitos secuenciales que se pueden utilizar para variedad inmensa de aplicaciones. A diferencia de los FPGA, cada función de los ASIC se vende por separado y si la función es muy

específica es muy difícil de encontrar en el mercado el respectivo ASIC dicha función.

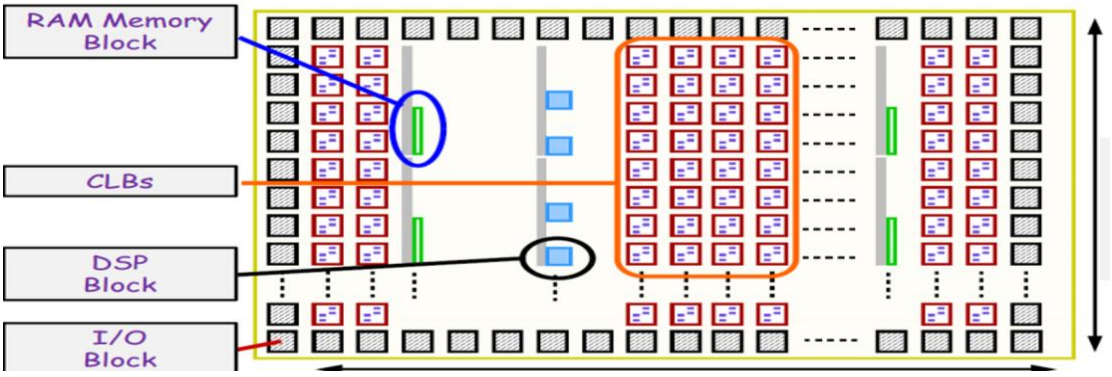
1.2. Funcionamiento interno de un FPGA

El funcionamiento interno de un FPGA consiste en interconectar varios bloques lógicos entre sí para implementar una función en específico. Un FPGA contiene bloques dedicados como memorias, controladores Ethernet, conexión inalámbrica, procesadores embebidos, bloques DSP, PLL, control de relojes, entre otros, dependiendo del modelo de FPGA que se posea.

1.2.1. Arquitectura de un FPGA

La arquitectura de un FPGA comprende los bloques lógicos configurables, bloques de entradas o salidas, bloques de memoria RAM y bloques de procesamiento de señales digitales.

Figura 1. **Arquitectura de un FPGA**



Fuente: *Presentación FPGA.* https://www.google.com/search?q=presentacion+FPGA&source=lnms&tbm=isch&sa=X&vedwjQg-WsyM3hAhWGZlAKHfaiDdAQ_AUIDigB&biw=663&bih=594. Consulta: 3 de enero de 2019.

1.2.1.1. Bloques de memoria RAM

Los FPGA cuentan con memoria de acceso aleatorio donde se cargan todas las instrucciones que ejecuta el FPGA. Los bloques de memoria RAM pueden tener múltiples tamaños de 2Kb, 4kb, 8Kb y 16Kb; pueden poseer múltiples configuraciones y distintos modos de operación de escritura y lectura o solo lectura.

1.2.1.2. Bloques lógicos configurables (CLB)

Los bloques lógicos configurables son los encargados de realizar las operaciones lógicas entre las señales de entrada retornando una señal de salida o múltiples señales de salida según su configuración. Se utilizan para implementar marcos y toras funciones diseñadas proporcionando el soporte físico para un diseño lógico implementado. Un bloque lógico configurable está compuesto por celdas lógicas llamadas ALM, LE, Slice, entre otras. Una celda típica consiste en una LUT de 4 entradas, un sumador completo, y un *flip flop* tipo D dependiendo del modelo del FPGA.

1.2.1.3. Bloques de procesamiento de señales digitales (DSP)

Los bloques de procesamiento de señales digitales son los encargados de realizar las operaciones numéricas a muy alta velocidad; existen bloques de memoria físicamente para el procesamiento de señales dentro del FPGA. Los elementos básicos de un bloque DSP son: conversor de señales analógicas a señales digitales en las entradas, conversor de señales digitales a señales analógicas en las entradas y salidas respectivamente, memoria de datos, memoria de programa y acceso directo de memoria.

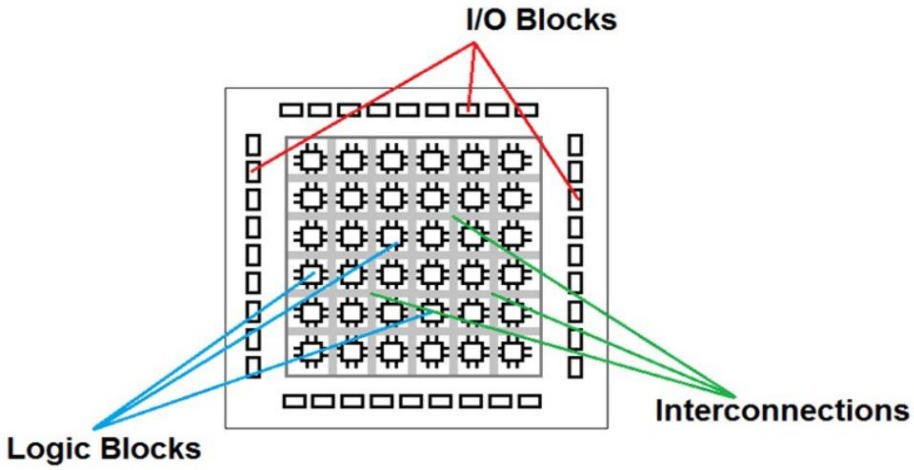
1.2.1.4. Bloques de entrada y salida (I/O block)

Estos bloques definen las señales de entrada digitales las cuales pueden ser provenientes de sensores, módulos, conversores analógicos digitales, entre otros, para su análisis y procesamiento según lo requiera la aplicación. Los bloques de entradas y salidas también definen las señales digitales de salida provenientes de la FPGA después de todo el procesamiento lógico digital. Estos bloques también representan los pines físicos del FPGA ordenados por puertos y segmentos.

1.2.2. Enrutamiento

El enrutamiento consiste en realizar las conexiones internas entre los bloques lógicos configurables y los bloques de entradas y salidas en un FPGA.

Figura 2. **Enrutamiento de un FPGA**



Fuente: *Introduction to FPGA acceleration.* https://www.google.com/search?q=presentacion+FPGA&source=lnms&tbm=isch&sa=X&vedwjQg-WsyM3hAhWGZlAKHfaiDdAQ_AUIDigB&biw=663&bih=594. Consulta: 4 de enero de 2019.

1.2.3. FPGA vs microcontrolador

Existen ventajas y desventajas entre un microprocesador y un FPGA dependiendo de la aplicación o función que posean.

1.2.3.1. Ventajas

El paralelismo y la flexibilidad de diseño representan una de las ventajas más importantes que posee un FPGA sobre el microprocesador gracias a la implementación física de procesos y módulos que se ejecutan en paralelo. Un FPGA permite que la velocidad de procesamiento de datos se ejecute mucho más rápido y eficiente. También, permite que el diseñador pueda modificar el programa la cantidad de veces que sean necesarias y a su vez modificar los circuitos físicos que la FPGA sintetiza.

La implementación de módulos permite que la programación de un FPGA sea más ordenando, la organización modular facilita el análisis de errores en el código y permite que el código sea escalable o fácil de editar.

Implementación de IP-Cores previamente diseñados por ingenieros de alto nivel y por la misma compañía de distribuidora de los FPGAs Xillinx. Estos IP-Cores son segmentos de código previamente programados que se encargan de realizar una función en específico; únicamente se agregan de forma modular en el proyecto nuevo. Xillinx posee un cátalo de IP-Cores disponible a sus usuarios, existe una versión de estudiante que pone al alcance el uso de ciertos IP-Cores de forma gratuita.

Implementación de sistemas que integran todos o gran parte de los módulos que componen un computador en un solo integrado mejor conocido

como Soc. Los modelos de FPGA más sofisticados cuentan con SoCs capaces de soportar sistemas operativos como Xillinux, un sistema operativo basado en Linux especial para el desarrollo de aplicaciones en FPGA.

1.2.3.2. Desventajas

El precio y la complejidad de un FPGA son mucho mayor en comparación con un micro-controlador.

Para la realización de la implementación y síntesis de las compuertas lógicas físicas es necesario utilizar las herramientas del fabricante, utilizar un entorno de desarrollo propietario con licencias.

El cambio de mentalidad que representa la descripción de hardware comparado con la programación de microcontroladores suele ser un obstáculo para el desarrollo de aplicaciones en FPGA.

1.3. Tarjeta de desarrollo Zybo Zynq 7000

Zybo (Zynq Board) es una plataforma de desarrollo de software integrado y software de nivel básico lista para utilizarse, construida alrededor del miembro más pequeño de la familia Xillinx Zynq-7000, el Z-7010. El Z-7010 se basa en arquitectura Xillinx SOC. Se integra perfectamente un procesador de doble núcleo ARM Cortex-A9 con Xillinx, matriz de puertas programable lógica (FPGA).

Cuando se combina con el amplio conjunto de periféricos multimedia y de conectividad disponibles en la Zybo, puede albergar un diseño de sistema completo. Cuenta con memorias integradas, entradas y salidas de video y

audio, ranura USB de doble función, conector Ethernet y memoria SD tendrán su diseño listo, sin necesidad de módulos o circuitos adicional. Además, existen disponibles seis conectores Pmod para colocar cualquier diseño en una ruta de fácil crecimiento.

La Zybo ofrece una alternativa de costo ultra bajo al ZedBoard para los diseñadores que no requieren entradas y salidas de alta densidad de conectores especiales, pero que aún desean aprovechar la capacidad de procesamiento masivo y la capacidad de ampliación de la arquitectura Zynq SoC.

1.3.1. Características

- 28 000 celdas lógicas
- RAM secciones de 240 KB
- 80 secciones DSP
- Procesador Cortex -A9 de doble núcleo a 650 MHz
- Convertidor de UART a USB
- Ranura para memoria microSD (compatible con el sistema de archivos Linux)
- Controladores de periféricos de alto ancho de banda: 1G Ethernet y USB 2,0
- Controlador de periféricos de bajo ancho de banda: SPI y UART
- Puerto HDMI de doble función (fuente / receptor)
- Puerto de salida VGA de 16 *bits* por píxel
- Puerto USB 2 0 (compatible con *host* y dispositivo)
- Memoria EEPROM externa
- Controlador de audio con salida de auriculares, micrófono y entrada de línea

- GPIO: 6 pulsadores, 4 interruptores deslizantes, 5 led
- 6 puertos (1 procesador dedicado)

1.4. Vivado Design Suite

Vivado Design Suite es un entorno de desarrollo producido por Xillinx para la síntesis y análisis de diseños basados en lenguajes de descripción de circuitos físicos, que reemplaza al entorno Xillinx ISE con características adicionales para el sistema en un desarrollo de circuitos y síntesis de alto nivel. Vivado representa un replanteamiento de todo el flujo de diseño (en comparación con ISE), y ha sido calificado como un entorno de desarrollo completo, intuitivo, escalable y rápido.

A diferencia de ISE, que se basó en simulaciones de lenguajes de descripción de hardware, la edición del sistema Vivado incluye un simulador lógico incorporado. Vivado también presenta síntesis de alto nivel, con una cadena de herramientas que convierte el código C en lógica programable.

Vivado permite a los desarrolladores sintetizar sus diseños, realizar análisis de tiempo real, examinar diagramas, simular la reacción de un diseño a diferentes estímulos y configurar el dispositivo de destino con el programador. Vivado es un entorno de diseño para productos FPGA de Xillinx, y está estrechamente relacionado con la arquitectura de dichos circuitos, y no se puede utilizar con productos FPGA de otros proveedores.

Vivado se presentó en abril de 2012, y es un entorno de diseño integrado (IDE) con herramientas de nivel de sistema a circuitos integrados basadas en un modelo de datos escalable compartido y un entorno de depuración común. Vivado incluye herramientas de diseño de nivel de sistema electrónico

(ESL) para sintetizar y verificar la IP algorítmica basada en C; empaquetado basado en estándares de algorítmico y RTL IP para reutilización; costura IP basada en estándares e integración de sistemas de todos los tipos de bloques de construcción de sistemas; y la verificación de bloques y sistemas. Una versión gratuita de la edición WebPack de Vivado proporciona a los diseñadores una versión limitada del entorno de diseño.

Vivado admite dispositivos de alta capacidad más nuevos y acelera el diseño de lógica programable. Vivado proporciona una integración y una implementación más rápidas para los sistemas programables en dispositivos con tecnología de interconexión de silicio apilado 3D, sistemas de procesamiento ARM, señal mixta analógica y muchos núcleos de propiedad intelectual de semiconductores.

Vivado está dirigido a los FPGA más grandes de Xillinx, y está reemplazando lentamente al entorno de desarrollo Xillinx ISE como su cadena de herramientas principal. A partir de 2014, Vivado cubre los FPGA medianos y grandes de Xillinx, Xillinx ISE cubrió los FPGA medianos y pequeños.

1.4.1. Instalación de VIVADO WebPack

El paquete de Vivado WebPack está orientado a los estudiantes y a todas aquellas personas que deseen aprender en este entorno de desarrollo y es una versión gratuita. Para realizar la instalación se debe de contar con una conexión a internet y 33 GB de memoria libre en el disco duro.

Se dirige a la página web oficial de Xillinx www.Xillinx.com, para instalar Vivado WebPack; primero se debe crear una cuenta personal de estudiante; en los datos personales Xillinx pide una dirección de correo electrónico, se

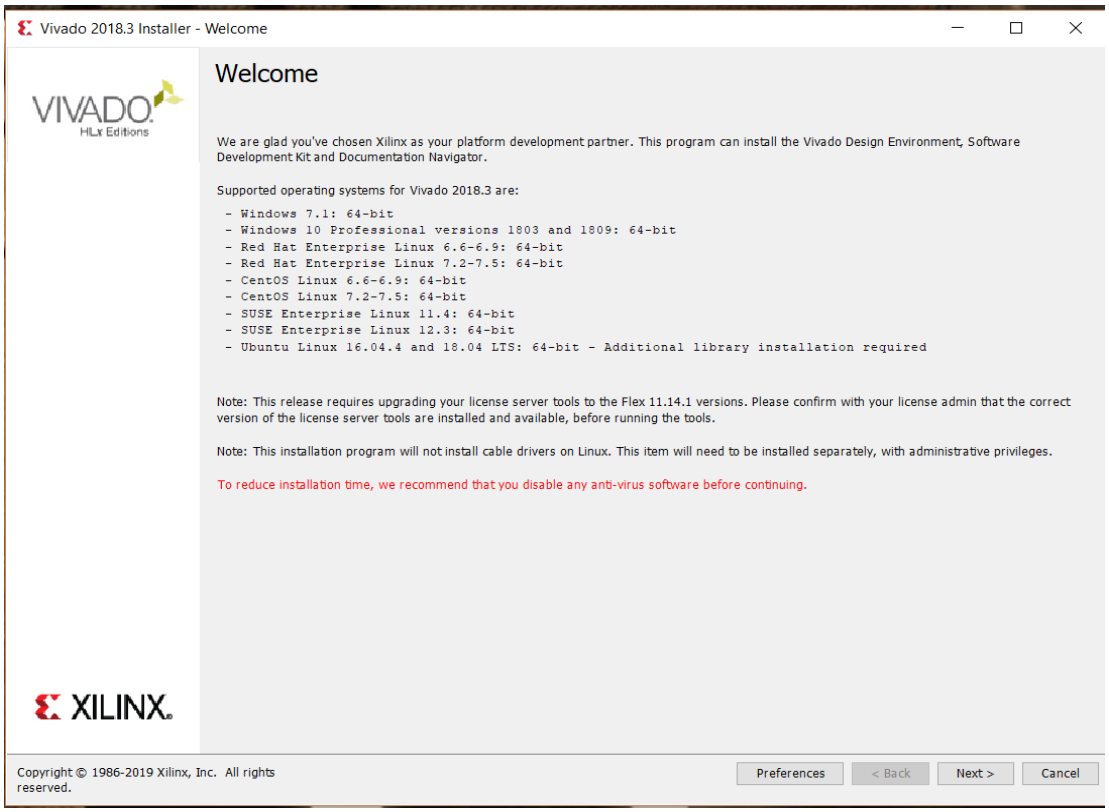
recomienda proporcionar el correo que otorga la universidad para que Xillinx pueda verificar que efectivamente es un estudiante el que solicita la cuenta. Luego de crear una cuenta se inicia sesión y se dirige a la pestaña de productos, *Developer Zone* y se ingresa en la opción Hardware *Developer Zone*; se selecciona la opción Vivado HLx and ISE Design Suites.

Se selecciona la opción *Download Vivado Design Suite – HLx Editions* redirigirá a una nueva vista de la página web donde se podrá seleccionar la versión del programa y el sistema operativo donde se instalará; seleccionar la opción y redirigirá a un formulario que solicita Xillinx.

Se deberá llenar el formulario con los datos personales (se recomienda utilizar dirección de correo de estudiante con extensión .edu); y se iniciara la descarga del instalador seleccionando el botón de descarga. Se puede descargar Vivado en su versión más reciente; para el desarrollo de este trabajo se estará trabajando con la versión de Vivado 2018.3 en Windows 10; sin embargo, se puede realizar la instalación para la versión más actualizada de la misma forma.

Luego de descargar el instalador, se debe ejecutarlo como administrador para iniciar la instalación de Vivado.

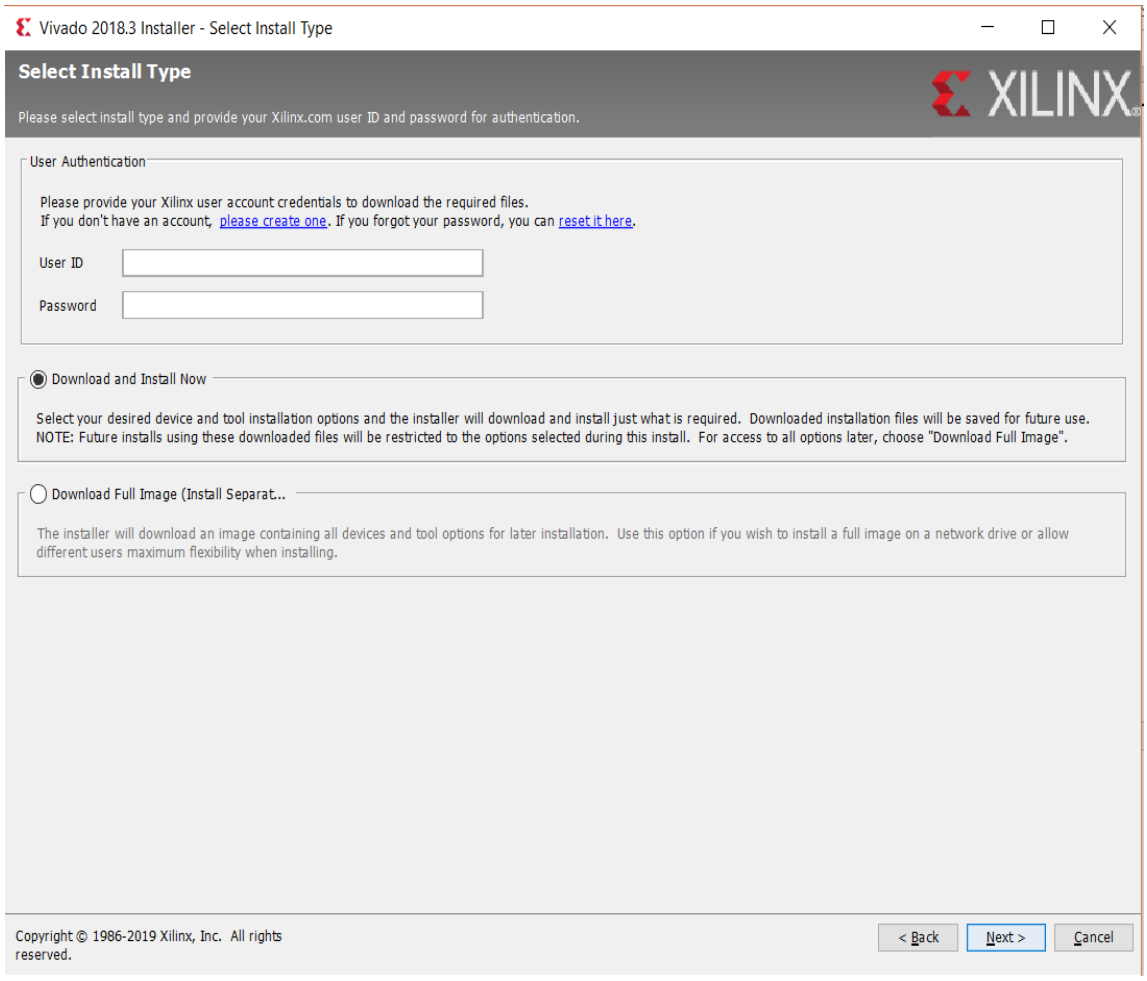
Figura 3. **Instalador Vivado 2018.3 especificaciones**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se abrirá una ventana con las especificaciones de los sistemas operativos soportados por el instalador, como se observa en la figura 3. Se recomienda desactivar el antivirus para que Vivado se pueda instalar de forma correcta y sin retrasos. Se selecciona el botón de siguiente (Next) se prosigue a la siguiente ventana.

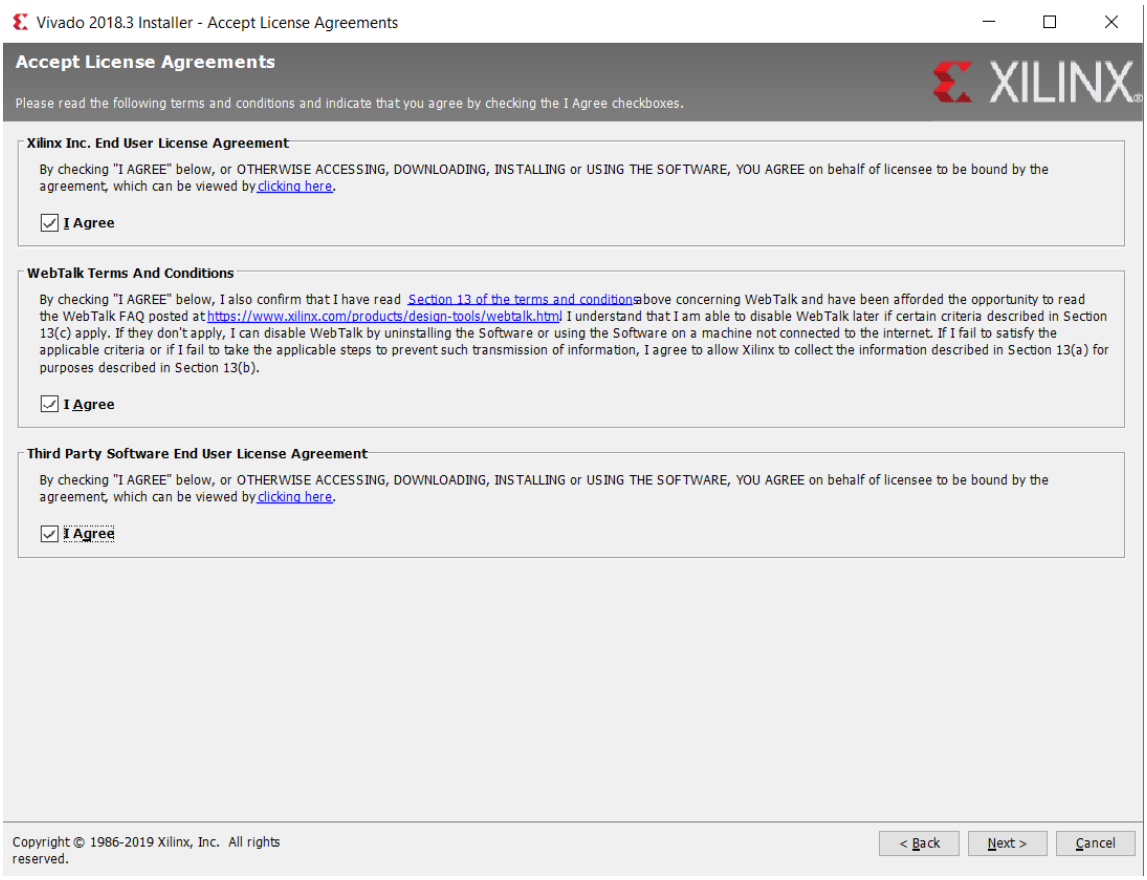
Figura 4. **Ingreso de cuenta en Xillinx**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Proporcionar el usuario y la contraseña de la cuenta previamente creada en la página web de Xillinx y seleccionar la opción descargar e instalar ahora *(Download and Install Now)* como se observa en la figura 4. Se selecciona el botón de siguiente *(Next)* y se prosigue a la siguiente ventana.

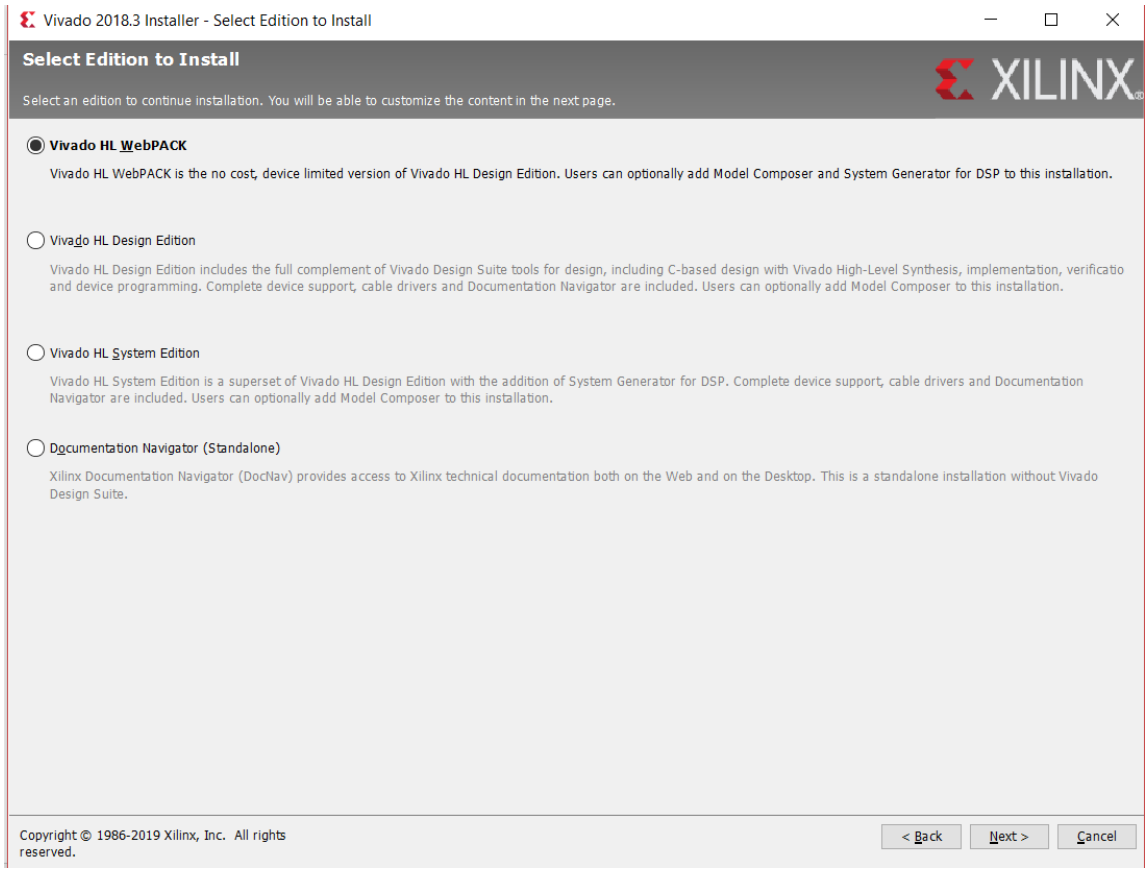
Figura 5. **Términos y condiciones de la licencia**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se aceptan los términos y las condiciones de Xillinx acerca de las licencias de usuario para utilizar Vivado. Se seleccionan las tres opciones (*I Agree*) afirmando que se está de acuerdo con los términos y las condiciones de Xillinx para el uso de Vivado como se observa en la figura 5. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

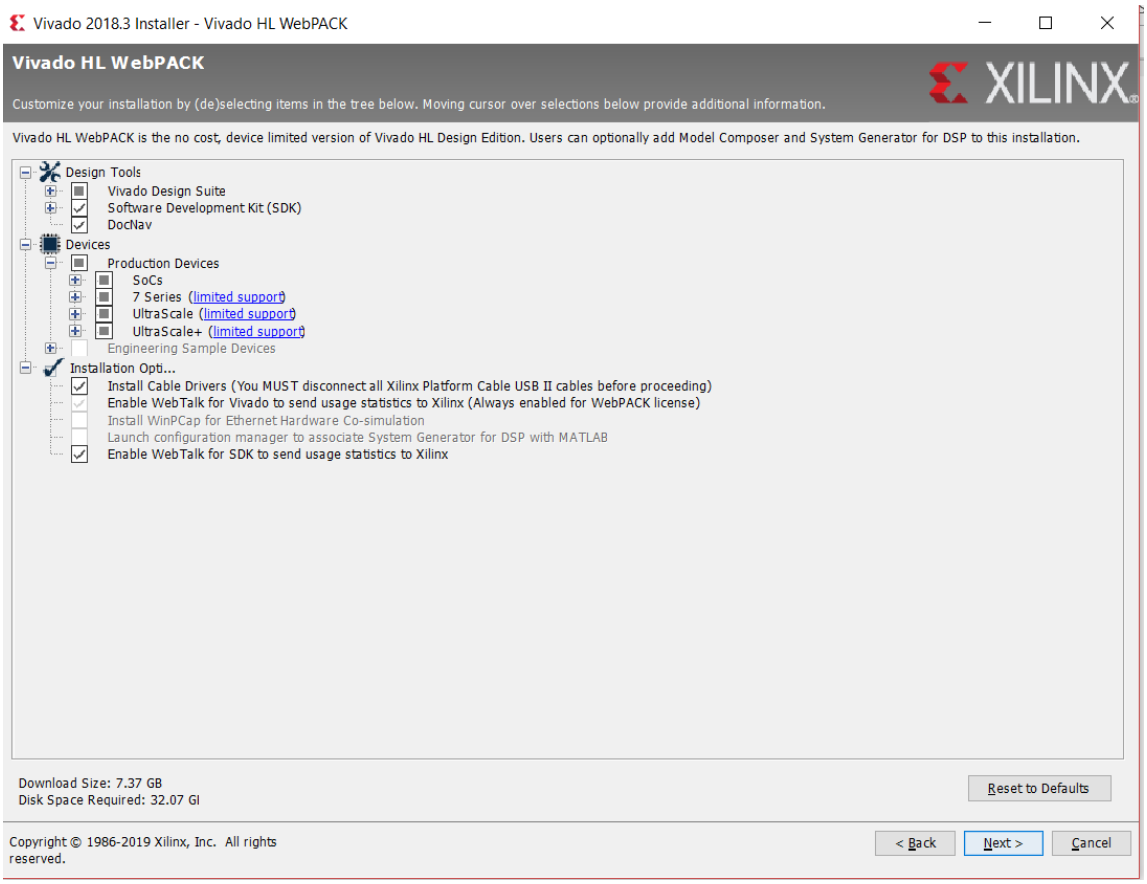
Figura 6. **Selección de edición Vivado**



Fuente: elaboración propia, utilizando Vivado 2018.3.

En la siguiente ventana se presentan cuatro ediciones diferentes de Vivado; se selecciona la opción Vivado HL WebPack como se observa en la figura 6. Vivado HL WebPack es la edición limitada y gratuita que Vivado otorga a los estudiantes con propósitos de aprendizaje. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

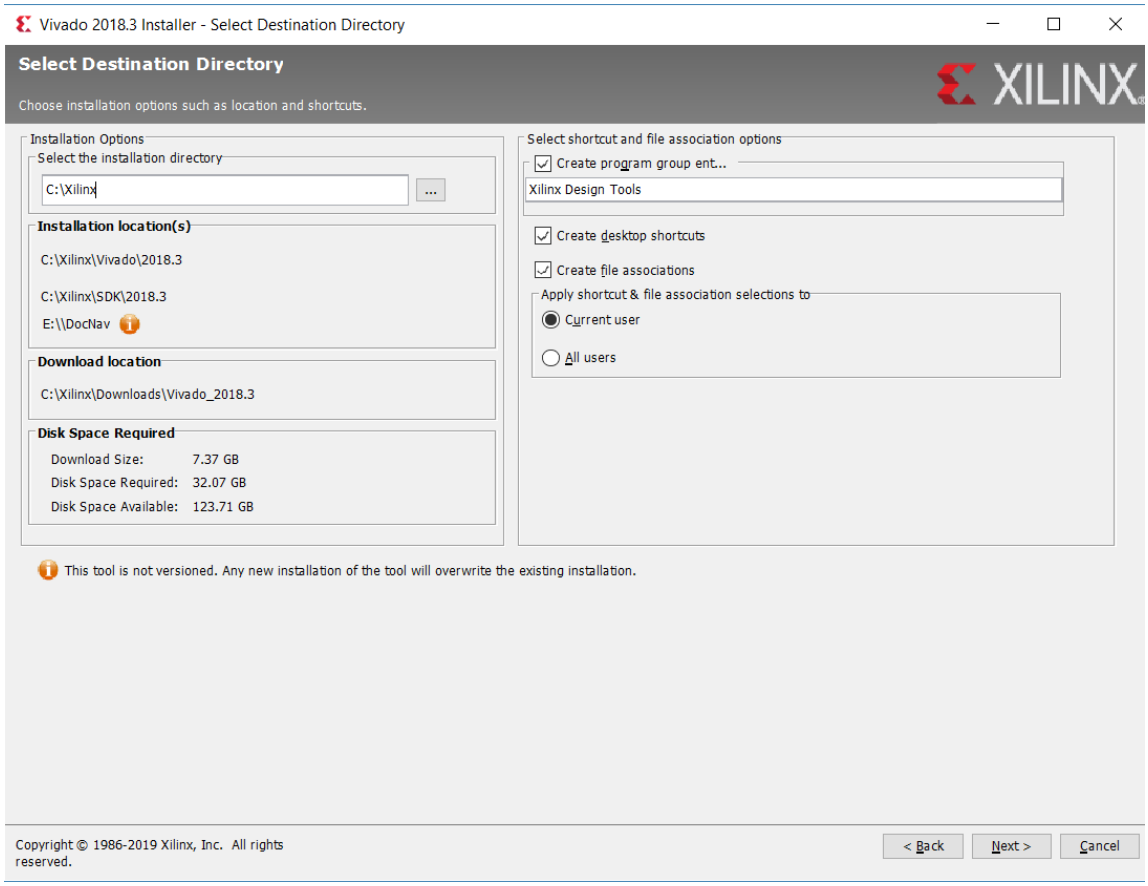
Figura 7. **Instalador de Vivado HL WebPack especificaciones**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Dejar las especificaciones de Vivado WebPack seleccionadas por defecto para evitar un error de instalación como se observa en la figura 7. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

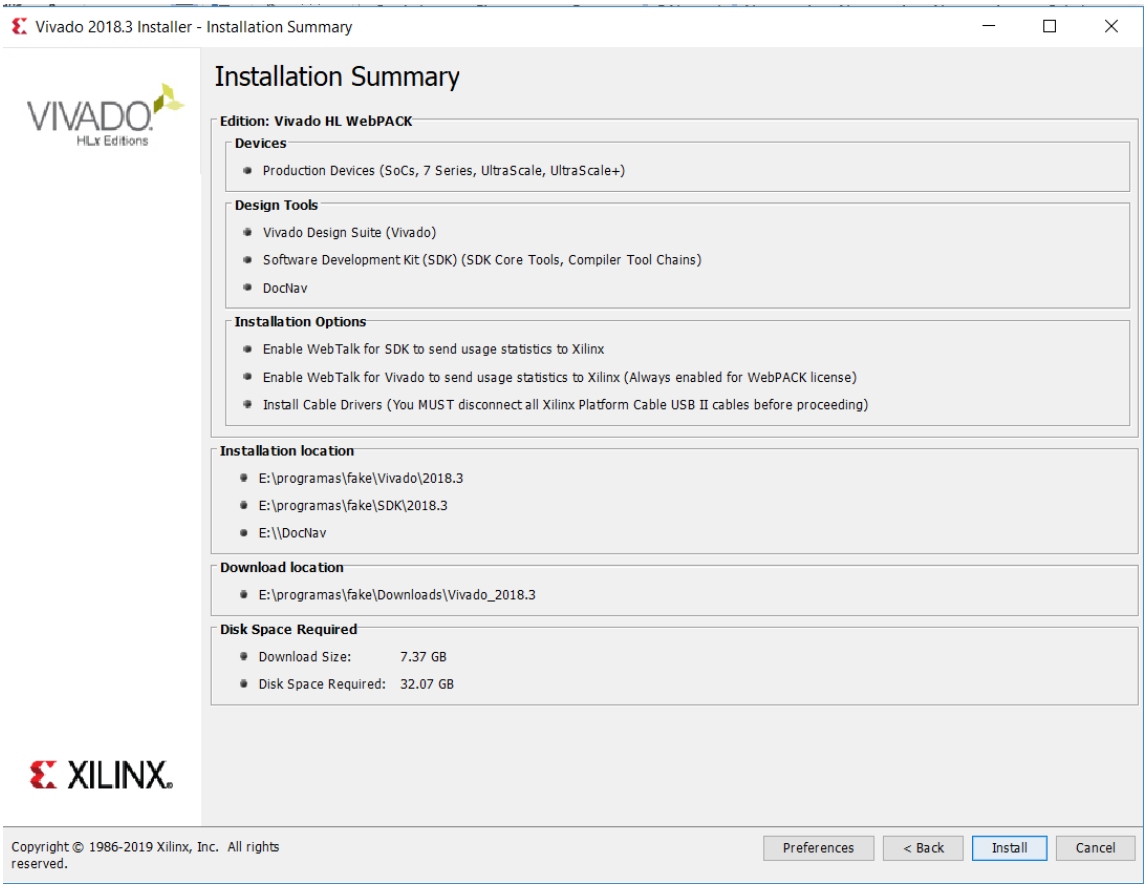
Figura 8. **Directorio destino de la instalación**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se selecciona el directorio destinatario donde se instalará el programa indicando la carpeta de instalación deseada por el usuario como se observa en la figura 8; se debe cumplir con 33 GB de memoria libre en el disco duro y con una conexión a internet debido a que descargará 7,37 GB de Vivado. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

Figura 9. **Resumen de la instalación**



Fuente: elaboración propia, utilizando Vivado 2018.3.

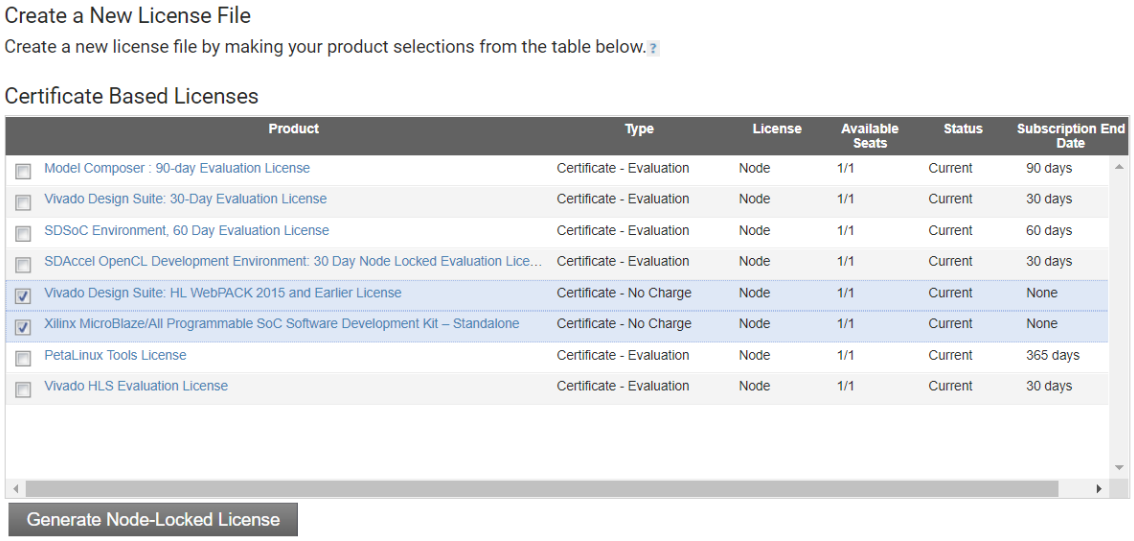
La siguiente ventana muestra el resumen de la instalación con todos los parámetros y las especificaciones previamente seleccionadas como se observa en la figura 9; se verifica que todo esté de forma correcta y se prosigue con la instalación seleccionando el botón de *Install*.

Iniciará con la descarga del contenido del programa y posteriormente la instalación completa de Vivado WebPack.

1.4.1.1. Licencia de estudiante

Para adquirir la licencia gratuitita de estudiante se debe ingresar a la página web de Xillinx; se inicia sesión con la cuenta de Xillinx; se dirige a la pestaña de Support; se selecciona la opción *Download and Licensing* donde se enviará a otra sección de la página Web. Se selecciona la opción *Licensing Help*; se busca la opción *Obtain a license for Free or Evaluation product* y se presiona el hipervínculo *Xillinx Product Licensing Site.* Ingresar los datos personales correctamente y en la dirección de correo electrónico colocar un correo electrónico educativo referente a la universidad con extensión .edu. Presionar el botón de siguiente.

Figura 10. **Licencia de estudiante**

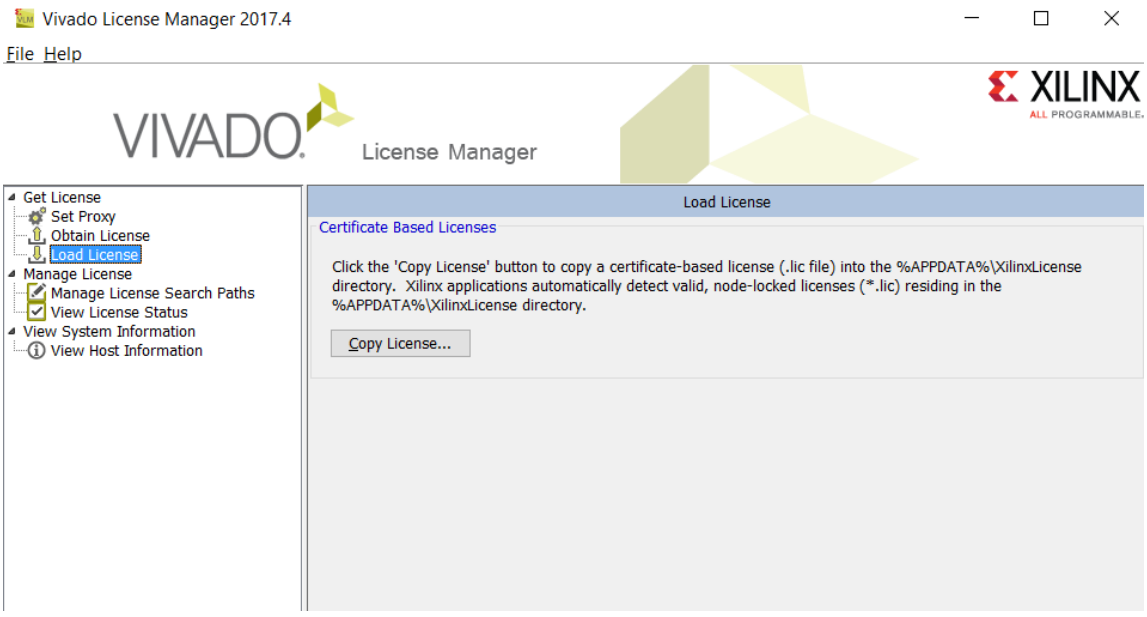


Fuente: Xillinx. *Licenciamiento del producto*. https://www.google.com/search?q=presentacion+FPGA&source=lnms&tbm=isch&sa=X&vedwjQg-WsyM3hAhWGZlAKHfaiDdAQ_AUIDigB&biw=663&bih=594. Consulta: 9 de enero de 2019.

Se selecciona las opciones Vivado Design Suite HL WebPack y Xillinx MicroBlaze como se observa en la figura 10. Se presiona el botón de *Generate Node-Locked License* y se prosigue a la siguiente ventana. Se enviará un correo de parte de Xillinx que contendrá la licencia de estudiante, un archivo con extensión .lic o descargar el archivo presionando el botón de *Download* desde la misma página de Xillinx.

Luego de descargar el archivo con extensión .lic y con Vivado previamente instalado, se prosigue con la activación de Vivado. Se ejecuta el programa, se busca la pestaña de Help y se selecciona la opción *Manage License.*

Figura 11. **Activación de Vivado**



Fuente: elaboración propia, utilizando Vivado 2018.3.

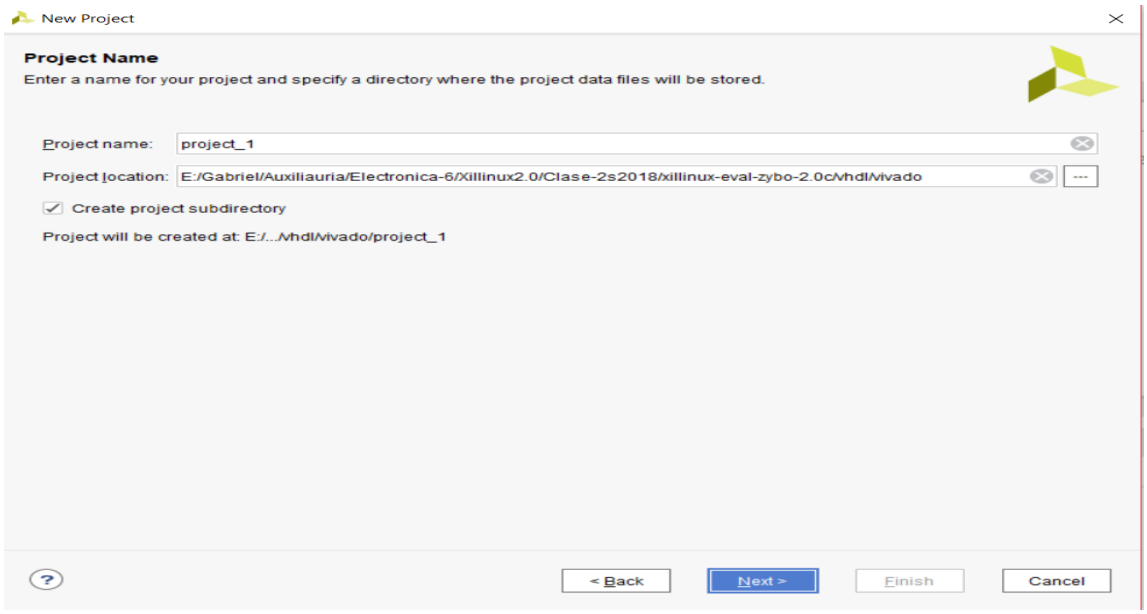
Se selecciona la opción *Load License* como se muestra en la figura 11 y se presiona el botón de *Copy license*. Se abrirá un directorio donde se tendrá

que ubicar el archivo con extensión .lic que se descarga desde la página o del correo electrónico; una vez ubicado el archivo se selecciona y el programa estará activado listo para utilizarse. Si el programa no se encuentra activado no dejará sintetizar las líneas de código ni realizar la implementación posterior a la sintonización. En la opción de *View license Status* se puede observar el estado y la vigencia de la licencia proporcionada.

1.4.2. Proyecto nuevo en Vivado

Para el siguiente ejemplo enfocado a la creación de un proyecto nuevo se utilizará la versión de Vivado Design Suite 2017.4. Luego de instalarla y activar el programa, se ejecuta. En la página principal se selecciona la pestaña de archivo y se selecciona la opción de nuevo proyecto.

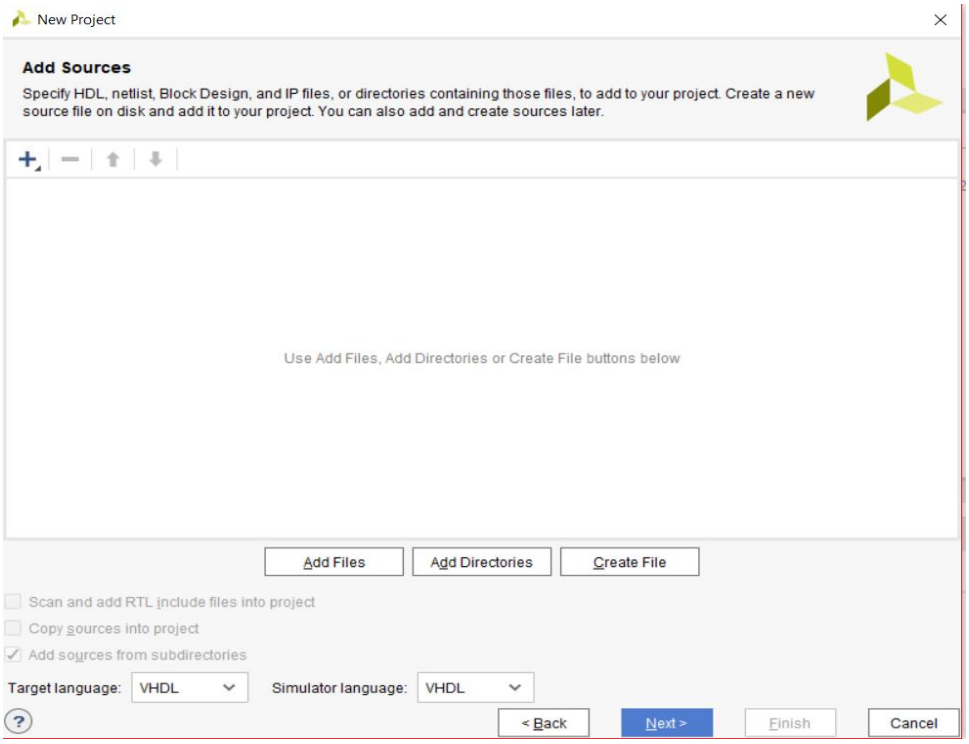
Figura 12. **Nuevo proyecto**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se coloca el nombre del proyecto y el directorio donde este se guardará posteriormente como se observa en la figura 12. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

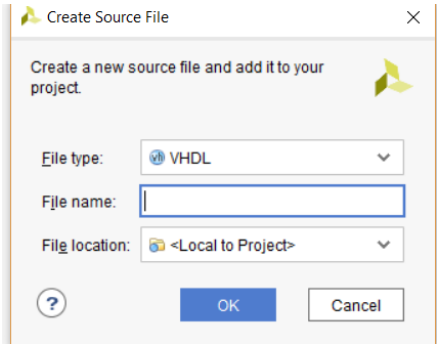
Figura 13. **Especificación de archivos y lenguaje de programación**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Especificación del tipo de lenguaje de programación que se utilizará, para el desarrollo de los demás capítulos y para este ejemplo se estará trabajando con VHDL que más adelante se conceptualizará. Se selecciona la opción VHDL en *Target Simulation y Simulator Language*. Se crea el archivo donde se estará trabajando presionando el botón de *Create File* como se muestra en la figura 13. Se desplegará una segunda ventana donde se especificará el archivo fuente.

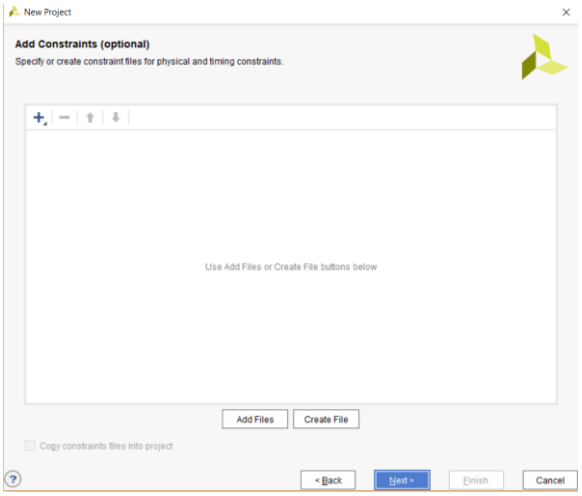
Figura 14. ***Create Source File***



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se coloca el tipo de lenguaje de programación, el nombre del archivo fuente, se indica el directorio destino donde se guardará el archivo como se muestra en la figura 14. Se presiona el botón de Ok y luego *Next* para continuar con la creación del proyecto.

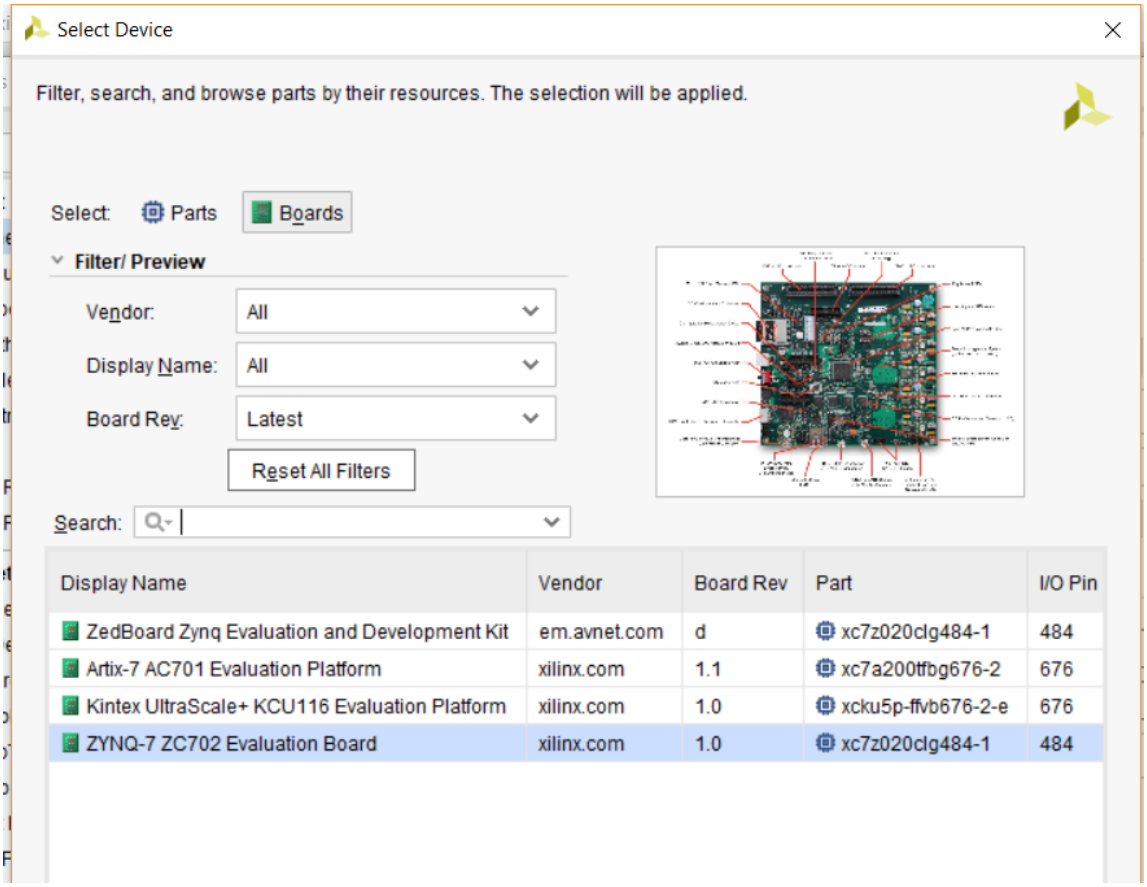
Figura 15. **Crear archivo con extensión .''xdc''**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se crea el archivo con extensión .’’xdc’’ y se prosigue con la creación del proyecto como se muestra en la figura 15. Más adelante se conceptualizará el archivo con extensión .’’xdc’’ y se explicarán sus funciones. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

Figura 16. **Selección de tarjeta de desarrollo**

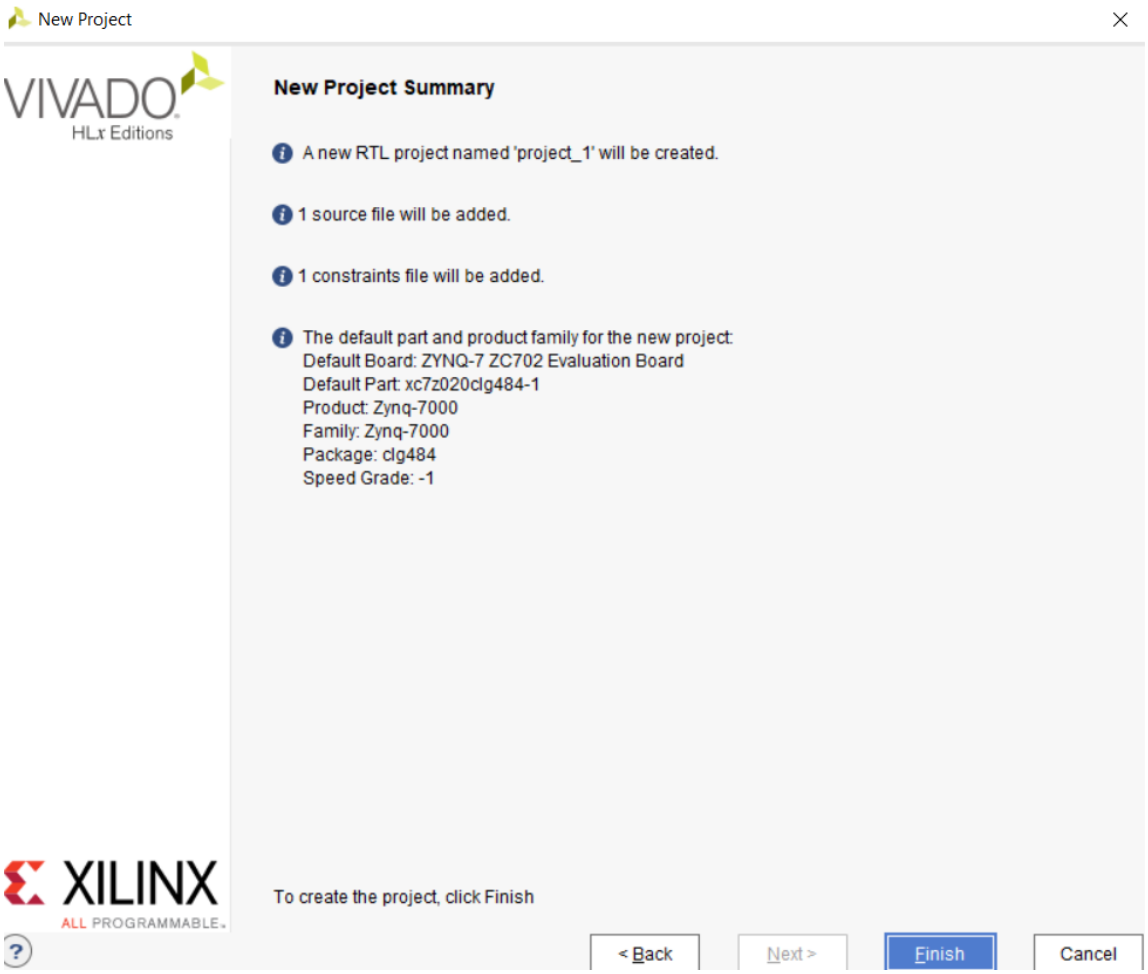


Fuente: elaboración propia, utilizando Vivado 2018.3.

Se selecciona la tarjeta de desarrollo que se estará utilizando, para este ejemplo se estará utilizando la tarjeta de desarrollo Zynq-7000 la cual se buscará en la pestaña de *boards* y se selecciona *Zynq-7 ZC702 Evaluation*

Board como se muestra en la figura 16. Se selecciona el botón de siguiente (*Next*) y se prosigue a la siguiente ventana.

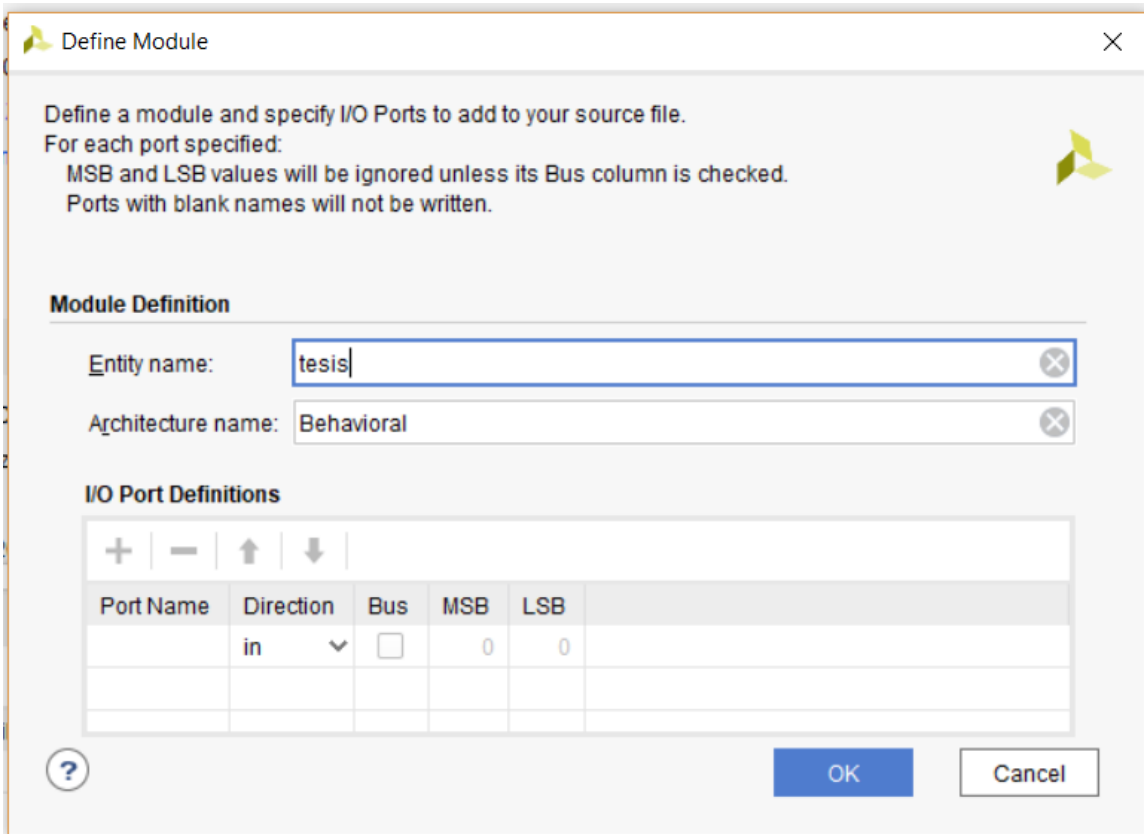
Figura 17. **Parámetros del proyecto**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Se verifica que todo esté correctamente y se presiona el botón *Finish* para terminar la creación del proyecto como se muestra en la figura 17.

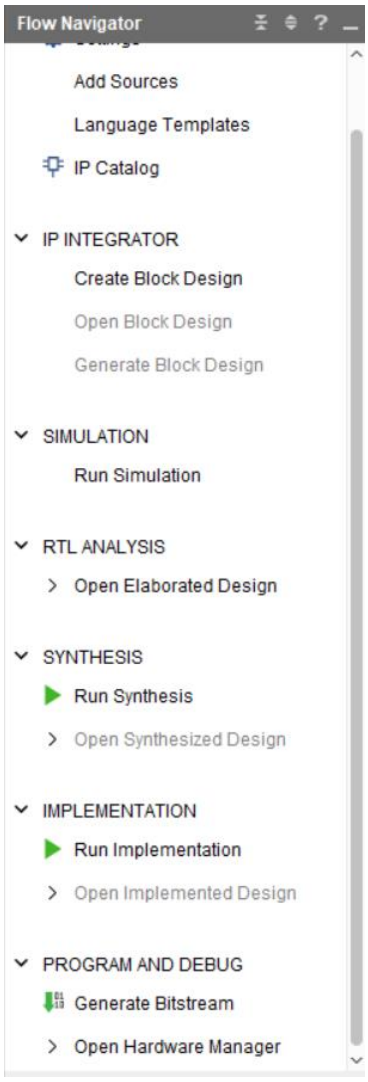
Figura 18. **Declaración de señales**



Fuente: elaboración propia, utilizando Vivado 2018.3.

En la siguiente ventana se pueden colocar las señales de entrada y salida que llevarán el módulo principal como se observa en la figura 18. Se recomienda no definir señales ya que más adelante se conceptualizarán en el capítulo de programación en VHDL. Se selecciona el botón de Ok para continuar con el proyecto.

Figura 19. **Navegador y herramientas de Vivado**



Fuente: elaboración propia, utilizando Vivado 2018.3.

Navegador de Vivado Design Suite 2017.4 donde se pueden observar las herramientas que se utilizará para el diseño de programas en VHDL. Se explicará brevemente cual es la función de la síntesis, implementación y generación de *bit stream* del proyecto.

1.4.2.1. Síntesis

La primera herramienta que se estará utilizando en Vivado es la síntesis de código de descripción de circuitos y compuertas lógicas; la síntesis es el proceso donde se genera una descripción del código descrito por el usuario representado en compuertas lógicas; es una traducción de código a circuitos lógicos y puede presentar distintos resultados para un mismo código. La síntesis también indica los posibles errores de sintaxis cometidos por el programador dependiendo del lenguaje de programación; para este trabajo se utilizará VHDL.

1.4.2.2. Implementación

La implementación es el proceso donde se realiza la conexión interna de los módulos y procesos previamente sintetizados; inicia el ruteo interno de la FPGA de los bloques de análisis de señales digítales, los bloques lógicos configurables y los bloques de entradas y salidas. La implementación de un detecta errores de lógica programable del usuario, por ejemplo, tomar en cuenta todas las posibles combinaciones de una sentencia de selectiva y los errores de instanciación de módulos.

1.4.2.3. Generación del *Bit Stream*

La generación del Bit Stream es el proceso final donde Vivado realiza la creación del archivo con extensión .bit, el cual contiene la implementación y síntesis del código descrito para cargarlo a la FPGA física. El archivo con extensión .bit es el encargado de modificar la memoria SRAM la cual tiene como función la conexión interna de los circuitos lógicos e implementar el código descrito en Vivado.

1.4.2.4. Simulaciones

Las simulaciones de módulos en VHDL únicamente requieren la síntesis previa del código descrito para la detección de errores provocados por la mala sintaxis. Las simulaciones sirven para predecir el comportamiento de un módulo en específico; se comporta como una caja negra donde se ingresan señales de entrada y la simulación nos retorna señales de salida. Sin embargo, hay casos donde la simulación de un módulo pueda llevarse a cabo y la implementación del módulo presente un error; esto debido a que existen sentencias que únicamente pueden ser sintetizables y simuladas, pero no pueden ser implementadas físicamente en circuitos.

1.4.2.5. Ip Core

Un Ip Core es un módulo previamente creado por los desarrolladores de Xillinx con el objetivo de facilitar la programación en Vivado, cada Ip Core segmentos de código diseñados con una aplicación en específico, por ejemplo, la resolución de una operación aritmética o trigonométrica, un contador binario de 8 *bits*, interfaces de comunicaciones, entre otros Vivado posee un catálogo de Ip Core listos para ser agregados como módulos secundarios y listos para instanciarse.

1.4.3. Práctica 1: instalación de Vivado

La práctica consiste en realizar la instalación del entorno de desarrollo Vivado Design Suite de preferencia la versión 2017.4 utilizando los pasos previamente descritos en la sección 1.4.2. Se debe activar Vivado con la licencia de estudiante.

2. PROGRAMACIÓN UTILIZANDO UN LENGUAJE DESCRIPTIVO DE ALTA VELOCIDAD (VHDL)

2.1. VHDL

VHDL es un lenguaje descriptivo de circuitos lógicos físicos utilizado especialmente para la implementación de aplicaciones sobre la FPGA. El nombre VHDL proviene del acrónimo en inglés *very high speed integrated circuit* hardware *descripción lenguage.* VHDL no es un lenguaje de programación como tal pero si cuenta con su propia sintaxis; conocer su sintaxis no implica necesariamente saber diseñar módulos con él. VHDL es un lenguaje de descripción de hardware que permite describir procesos, módulos, sentencias concurrentes, sentencias secuenciales, circuitos asíncronos y circuitos asíncronos. Para utilizar VHDL de la forma correcta y entender su funcionamiento se debe tomar en cuenta lo siguiente:

- Pensar en compuertas lógicas y procesos, no en variables ni funciones.
- Evitar bucles en las sentencias secuenciales y relojes condicionados.
- Saber qué parte del circuito es concurrente y cuál secuencial.
- Todo el código descrito será ejecutado de forma paralela o concurrente, a pesar de que se escriba línea por línea.
- El código se ensambla en el instante en el que se enciende la FPGA.

2.1.1. ¿Por qué usar un lenguaje de descripción hardware?

- Poder descubrir problemas en el diseño antes de su implementación física.

- La complejidad de los sistemas electrónicos crece exponencialmente, es necesaria una herramienta que trabaje con el ordenador.

2.2. Elementos básicos de lenguaje

Se utilizan los mismos protocolos y reglas que en la mayoría de los lenguajes de programación. VHDL no es sensitivo a letras mayúsculas o minúsculas permitiendo programar de las dos formas sin generar errores de sintaxis.

2.2.1. Tipos de datos

Los tipos de datos son los que se encargan de almacenar la información de una variable, señal o simplemente una constante.

2.2.1.1. *Integer*

Es un tipo de dato que comprende números enteros, se pueden utilizar números de 32 *bits* con signo en otras palabras valores de -2,147,483,648 a +2,147,485,647.

2.2.1.2. *Natural*

Comprende todos los números positivos del tipo de dato *integer,* valores posibles de 0 a +2,147,485,647.

2.2.1.3. *Character*

Comprende todos los caracteres ASCII.

2.2.1.4. *String*

Es un arreglo de caracteres ASCII.

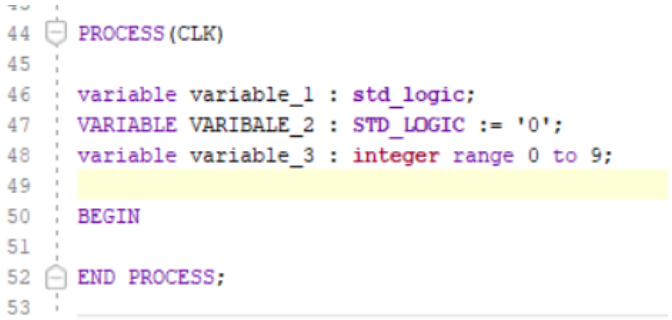
2.2.2. Objetos de datos

A continuación, se describen los objetos de datos.

2.2.2.1. Variables

Son las encargadas de almacenar información al instante del dato asignado a la variable; solamente pueden ser utilizadas dentro de procesos, procedimientos o funciones; puede ser modificada o leída depende del caso y no necesitan se inicializadas.

Figura 20. **Declaración de variables**



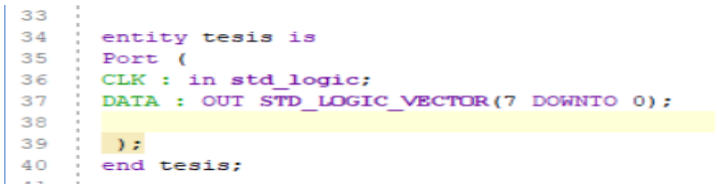
Fuente: elaboración propia, utilizando Vivado 2018.3.

Las variables se declaran dentro de un proceso antes del *begin*. Para la asignación de variables se utiliza el símbolo de dos puntos seguido de un signo igual; se pueden asignar valores binarios y números enteros, como se observa en la figura 20.

2.2.2.2. Señales

Las señales, a diferencia de las variables, contienen información sobre el valor o dato almacenado; los valores almacenados en una señal pueden ser modificados o leídos. Las señales sirven para contactar elementos concurrentes y secuenciales entre sí como si fueran cables; todos los valores de entradas y salidas de los módulos, puertos de una entidad y pines de la FPGA física son señales. Muy importante tener en cuenta que las señales no se actualizan al instante dentro de procesos, actualizan su valor al final de cada proceso. Dentro de una instanciación las señales transportan información entre módulos secundarios. Un ejemplo de declaración de señales son las entradas y salidas de la entidad del módulo como se observa en la figura 21.

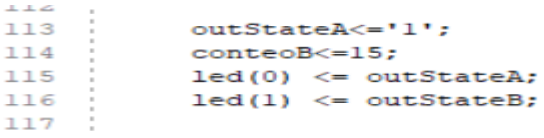
Figura 21. **Declaración de señales**



Fuente: elaboración propia, utilizando Vivado 2017.4.

La asignación de datos a las señales se realiza colocando el símbolo de menor que seguido de un signo igual y el valor del dato como se observa en la figura 22. Tomando en cuenta que outStateA es una señal de tipo 'std_logic', conteo B es una señal de tipo integer y led es una señal de tipo 'std_logic_vector'.

Figura 22. **Asignación de señales**

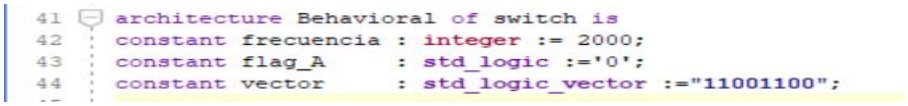


Fuente: elaboración propia, utilizando Vivado 2017.4.

2.2.2.3. Constantes

Las constantes son objetos que almacenan un único valor de un tipo de dato en específico y obtienen su valor al ser declaradas. Las constantes no pueden cambiar su valor dentro de la ejecución del programa y únicamente pueden ser leídas. Las constantes se declaran luego de la arquitectura del módulo como se muestra en la figura 23.

Figura 23. **Declaración de constantes**



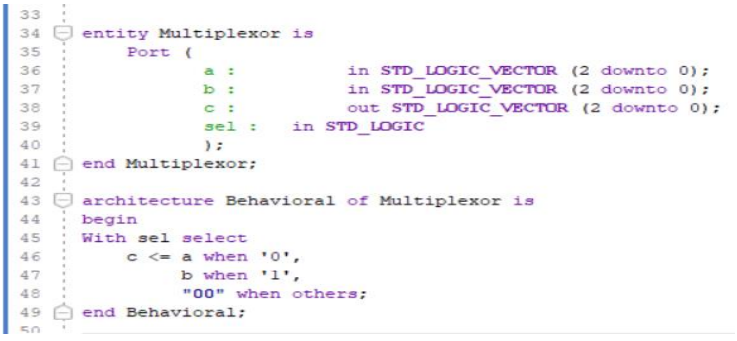
Fuente: elaboración propia, utilizando Vivado 2017.4.

2.2.3. Módulos VHDL

Un módulo en VHDL está compuesto por una entidad y una arquitectura. La entidad es el espacio donde se declaran todas las señales de entradas y salidas del módulo y la arquitectura es el espacio donde se manipulan nuestras señales de entrada retornando señales de salida. En la arquitectura del módulo se coloca toda la programación referente al análisis de las señales de entrada.

En la figura 24 se muestra un ejemplo de un multiplexor, en la entidad se declaran las señales de entrada a, b, sel y la señal de salida c. En la arquitectura del módulo se programa la función del multiplexor intercambiando a la salida c las señales a y b según la señal selectora.

Figura 24. **Multiplexor**

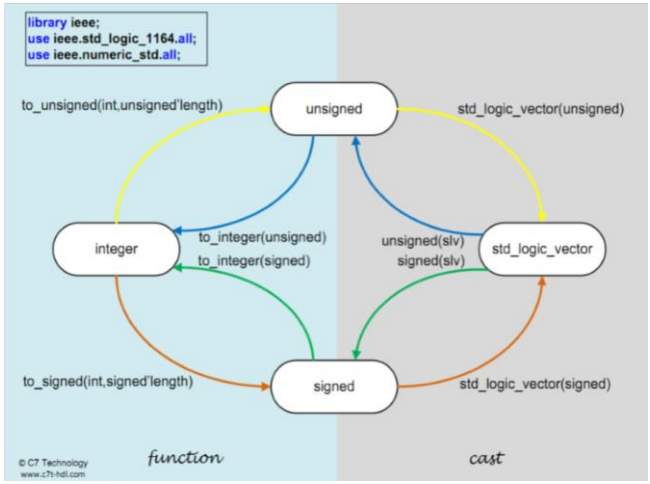


Fuente: elaboración propia, utilizando Vivado 2017.4.

2.2.4. Conversión de tipos

La conversión de tipos es necesaria para convertir valores de tipo 'std_logic_vector' a enteros con signo o sin signo; para realizar operaciones matemáticas es necesario tener valores de tipo integer y para colocar valores numéricos en los pines físicos de la FPGA es necesario tener valores de tipo 'std_logic_vecor'. Para realizar la conversión de tipos de dato se utiliza la siguiente sintaxis como se puede observar en la figura 25.

Figura 25. **Conversión de tipos de datos**



Fuente: Ing. MORALES, Iván. *Presentación FPGA.* https://www.google.com/search?biw=663&bih=594&tbm=isch&sa=1&ei=fxiyXOD1AcTFwQLko4aACg&q=Ing.+MORALES%2C+Iv%C3%A1n.+Presentaci%C3%n+FPGA.+&oq=Ing.+MORAL. Consulta: 3 de enero de 2019.

Para realizar la conversión de std_logic_vetor a integer primero se decide si se desea un valor con signo o sin signo y luego se prosigue con la conversión al número entero. De la misma forma se realiza la conversión de *integer* a std_logic_vetor.

2.3. Asignación concurrente

La asignación concurrente es la asignación de datos o valores a señales, son todos aquellos procesos que se ejecutan en paralelo al mismo tiempo, no llevan un proceso secuencial y no necesitan una señal de activación. Asignaciones concurrentes:

- Asignación simple
- Asignación condicional
- Asignación selectiva

2.3.1. Asignación simple

La asignación simple estable un solo valor a una señal, como se observa en la figura 26. En este ejemplo hay dos asignaciones simples a las señales x y d; tomar en cuenta que estas dos asignaciones ocurren en paralelo.

Figura 26. **Asignación simple**

```
44  architecture Behavioral of Multiplexor is
45  signal x : std_logic_vector (2 downto 0);
46  begin
47
48      x <= (a and b);
49      d <= (c and b)or (not x);
50
51  end Behavioral;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

2.3.2. Asignación condicional

La asignación condicional lleva la misma lógica de los If anidados. Este tipo de asignación establece un valor a una señal en función de una condición, aceptando cualquier tipo de condición. A diferencia de la asignación simple, la sintaxis incluye un *When* y un *Else* como se observa en el ejemplo de un multiplexor en la figura 27.

Figura 27. **Asignación condicional**

```
34  entity Multiplexor is
35      Port (
36             a :          in STD_LOGIC_VECTOR (2 downto 0);
37             b :          in STD_LOGIC_VECTOR (2 downto 0);
38             c :          in STD_LOGIC_VECTOR (2 downto 0);
39             d :          out STD_LOGIC_VECTOR (2 downto 0);
40             sel :        in STD_LOGIC_VECTOR (1 DOWNTO 0)
41             );
42  end Multiplexor;
43
44  architecture Behavioral of Multiplexor is
45
46  begin
47      d <= a when sel = "00" else
48           b when sel = "01" else
49           c when sel = "10" else
50           "000";
51  end Behavioral;
52
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

2.3.3. Asignación selectiva

La asignación selectiva se utiliza cuando existen varias condiciones de una misma señal selectora, a diferencia de la asignación condicional, la sintaxis incluye *with* y *select.* Es recomendable agregar la selección *when others* para evitar errores de implementación en un futuro. La selección *when others* comprende todos los valores que no se tomaron en cuenta en selecciones anteriores como se observa en la figura 28.

Figura 28. **Asignación selectiva**

```
34  entity Multiplexor is
35      Port (
36              a :          in STD_LOGIC_VECTOR (2 downto 0);
37              b :          in STD_LOGIC_VECTOR (2 downto 0);
38              c :          in STD_LOGIC_VECTOR (2 downto 0);
39              d :          out STD_LOGIC_VECTOR (2 downto 0);
40              sel :        in STD_LOGIC_VECTOR (1 DOWNTO 0)
41              );
42  end Multiplexor;
43
44  architecture Behavioral of Multiplexor is
45
46  begin
47      d <= a when sel = "00" else
48           b when sel = "01" else
49           c when sel = "10" else
50           "000";
51  end Behavioral;
52
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

2.4. Asignación secuencial

Comprende todas las sentencias y asignaciones que siguen un orden de ejecución, permite describir el comportamiento del módulo como una secuencia de eventos ejecutados ordenadamente línea por línea. La asignación secuencial permite llevar un orden de ejecución, por lo tanto, el orden de los enunciados es muy importante, a diferencia de la asignación concurrente.

2.4.1. Procesos

Todas las sentencias secuenciales se encuentran ejecutadas dentro de procesos. Un proceso se considera una sentencia concurrente ejecutándose en paralelo con otros procesos; sin embargo, todo el código que se encuentre dentro del proceso se ejecutara de forma secuencial. Un proceso es similar al estilo de programación de los microcontroladores y se ejecutan como interrupciones.

Varios procesos pueden coexistir en una misma arquitectura ejecutándose en paralelo, tomando en cuenta que la única forma de comunicarlos es utilizando señales. Los procesos se ejecutan si se cumplen ciertas condiciones que el usuario define. Existen dos modos de operación para los procesos: ejecución y espera. Cuando el proceso está en modo espera, el proceso se mantiene a la espera de una condición de activación; cuando se cumpla esta condición el proceso pasará al modo de ejecución. El modo de ejecución es cuando el código dentro del proceso se está ejecutando luego de activarse la condición.

Todas las señales globales que se encuentren dentro de un proceso serán actualizadas al final de ejecución del mismo proceso, durante la ejecución del proceso las señales no cambiarán su valor. Las señales son las encargadas de comunicar varios procesos, son la interfaz lógica entre la asignación concurrente y secuencial.

2.4.1.1. Elementos de un proceso

Los procesos están compuestos de tres elementos: la lista sensitiva, declaraciones y sentencias secuenciales.

2.4.1.1.1. Lista sensitiva

La lista sensitiva comprende el listado de señales que activarán el proceso, cualquier cambio en estas señales activará el proceso automáticamente.

2.4.1.1.2. Declaraciones

Las declaraciones comprenden todas aquellas variables, tipos, y funciones que se declaren dentro del proceso. Las variables que se declaren dentro de un proceso serán locales; esto significa que no se podrán utilizar en procesos paralelos únicamente en el proceso local.

2.4.1.1.3. Sentencia secuencial

Las sentencias secuenciales comprenden toda la programación dentro del proceso que será ejecutado en orden de línea; pueden ser sentencias *if, when case,* máquinas de estado, entre otros

Figura 29. **Proceso**

```
44  Process(CLK) --LISTA SENSITIVA
45  variable contador : integer range 0 to 255;     --DECLARACIONES
46  begin
47      if(Rising_Edge(CLK))then                    --SENTENCIAS SECUENCIALES
48          contador := contador + 1;               --SENTENCIAS SECUENCIALES
49      end if;                                     --SENTENCIAS SECUENCIALES
50  end Process;
```

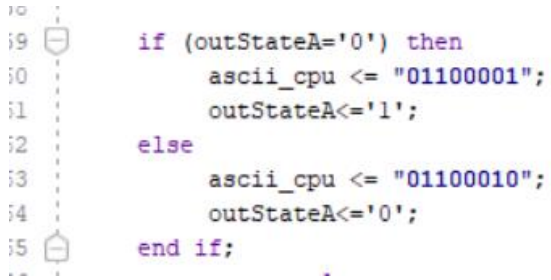
Fuente: elaboración propia, utilizando Vivado 2017.4.

En la figura 29 se observa un ejemplo de un proceso y sus elementos. La lista sensitiva está compuesta por solamente una señal CLK; en la sección de declaraciones se encuentra una variable local declarada con el nombre contador, de tipo entero en rango de 0 a 255. En la sección de sentencias secuenciales se encuentra toda la lógica del proceso, una sentencia *If* la cual únicamente se activará cuando la señal de CLK ejecute un flanco positivo (que pase de estado bajo a estado alto). Se activará un conteo de todos los flancos de subida del reloj hasta 255.

2.4.2. Sentencia *If*

Las sentencias *If* únicamente se pueden utilizar dentro de procesos debido a que pertenecen a las asignaciones secuenciales; las sentencias *If* únicamente ejecutarán su código sí se cumple una condición inicial. Pueden contener varias condiciones y pueden ser anidados. Las sentencias *If* pueden ser de varias ramas y siempre tomar en cuenta la sentencias *else*, como se observa en la figura 30.

Figura 30. **Sentencia *If***

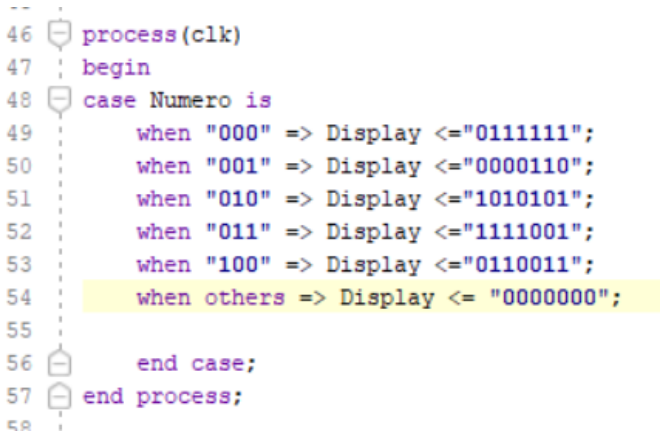


Fuente: elaboración propia, utilizando Vivado 2017.4.

2.4.3. Sentencia *when case*

La sentencia *when case* es una estructura de control enfocada a las decisiones múltiples en función de una sola señal. Cumple la misma función de los if anidados pero únicamente depende de una expresión haciendo que su programación sea más ordenada; de la misma forma que la sentencia if, la sentencia case debe ir programada dentro de un proceso. Tomando en cuenta que VHDL es un programa de descripción de hardware, es muy importante tomar en cuenta la opción *when others* con el propósito de abarcar todos los resultados posibles como se observa en la figura 31.

Figura 31. **Sentencia *when case***



Fuente: elaboración propia, utilizando Vivado 2017.4.

2.4.4. Máquina de estados finitos

Una máquina de estados finitos es un proceso secuencial que tiene un número definido de estados cuya función es facilitar la toma de decisiones en un determinado orden. Está diseñada para realizar ciertas acciones, dependiendo del estado presente en la que se encuentre la máquina de estados. Las máquinas de estados son una gran herramienta para realizar el control y la toma de decisiones en sistemas de lógica digital. Para la implementación de una máquina de estados en VDHL se utiliza la sentencia secuencial *when case* combinada con vectores o tipos definidos por el usuario.

2.4.4.1. Máquina de estados de tipo Moore

Este tipo de máquina de estados depende únicamente de sus mismos estados, el cambio de estado se realiza en función del estado actual y en base al estado actual cambia sus salidas. Los cambios de señales de salida se

encuentran sincronizados con las entradas de reloj de la FPGA. Por ejemplo, el control de un semáforo que realiza el cambio luego de un intervalo de tiempo según el estado actual (si se encuentra en rojo pasará a verde).

2.4.4.1.1. Máquina de estados de tipo Mealy

La máquina de estados tipo Mealy cambia sus estados en función de sus entradas presentes; los cambios de señales pueden ser síncronas a la señal de reloj o asíncronas en función de las señales de entrada. Por ejemplo, el control de una línea de producción industrial con contactares y sensores ópticos.

2.4.5. Tipos definidos

Los tipos definidos permiten crear un tipo de dato personalizado por el usuario con el objetivo de tener un código más ordenado y la simplificación del mismo. Las máquinas de estado se declaran por medio de los tipos definidos, como se observa en la figura 32.

Figura 32. **Declaración de una máquina de estados**

```
48  type FSM is (bit0,bit1,bit2,bit3,bit4,bit5,bit6,bit7,guardar);
49  signal Maquina_2: FSM;
50  signal Maquina_1: FSM;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

2.4.5.1. Creación de una máquina de estados

Primero se debe identificar el problema y definir todos los posibles estados que requiera la máquina. Se diseña un diagrama de estados incluyendo sus entradas y salidas posibles. Se declara el tipo definido con base en la información recopilada y se identifica a qué tipo de máquina de estados pertenece el problema para la declaración de señales de entrada y salida. Se declaran las señales de control y se programa la máquina de estados, como se muestra en la figura 33.

Figura 33. **Máquina de estados**

```
57  process(clk)
58  begin
59
60  If Rising_Edge(clk) Then          --SE EJECUTA
61                  Case Estado is
62                       when bit0 =>
63                            contador <= contador+1;
64                            if contador = 2 then
65                                 sclk<='0';
66                                 Estado <= bit1;
67                                 contador <= 0;
68                            end if;
69                       when bit1 =>
70                            sclk<='1';
71                            contador <= contador+1;
72                            if contador = 2 then
73                                 sclk<='0';
74                                 Estado <= bit2;
75                                 contador <= 0;
76                            end if;
77                       when bit2 =>
78                            sclk<='1';
79                            contador <= contador+1;
80                            if contador = 2 then
81                                 sclk<='0';
82                                 Estado <= bit3;
83                                 contador <= 0;
84                            end if;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

2.4.6. Memorias

Una memoria es un espacio con el objetivo de almacenar información para luego procesarla, una memoria es sinónimo de un vector o matriz con varias localidades. En VHDL una memoria es un tipo definido por el usuario, se pueden programar memorias vectoriales o matriciales de n datos según la aplicación que se desee.

2.4.6.1. Arreglos

Un arreglo es un vector o una combinación de vectores para formar una matriz con el objetivo de almacenar información de un tipo definido. Los arreglos usualmente se utilizan para crear memorias multidimensionales como se observa en la figura 34. Se tiene un tipo definido de 10 posiciones, cada posición puede almacenar un número de tipo 'std_logic_vector' de tamaño 8 *bits*. Se declaran dos señales: memoria 1 y memoria 2 con los parámetros del tipo definido anteriormente; se tienen dos memorias de 10 posiciones con capacidad de almacenar números de 8 *bits*.

Figura 34. **Declaración de memorias**

```
45  type tipo_mem is array( 0 to 9) of std_logic_vector(7 downto 0);
46  signal memoria_1 : tipo_mem;
47  signal memoria_2 : tipo_mem;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

2.4.6.2. Práctica 2: máquinas de estado

La práctica consiste en programar una máquina de estados capaz de controlar dos semáforos para dos vías distintas; la luz verde debe durar 45 segundos, la luz roja debe durar 50 segundos y la luz amarilla debe oscilar por 5 segundos para ambos semáforos. Realizar el cálculo del tiempo en función de la velocidad de reloj de la Zybo y utilizar VHDL como único lenguaje de programación.

2.4.6.3. Práctica 3: decodificador BCD a decimal

La práctica consiste en realizar un decodificador BCD a 7 segmentos utilizando únicamente procesos concurrentes; se ingresará un número binario de 4 *bits* (0-15) obteniendo a la salida dos números de 8 *bits* decodificados para dos *displays* de 7 segmentos. Realizar el decodificador utilizando los dos tipos de asignaciones (concurrente y secuencial). Ejemplo: se ingresa un numero binario “1100”, el programa tiene que ser capaz de devolver “0000110” y “1011011”.

2.4.6.4. Práctica 4: *motor stepper*

La práctica consiste crear un módulo en VHDL encargado de generar las secuencias de movimiento de un motor paso a paso y desarrollar la simulación de dichas secuencias utilizando Vivado. Dicho módulo debe contar con las siguientes entradas y salidas:

2.4.6.4.1. Entradas

- Una señal de tamaño 8 *bits* que simulará la entrada de un carácter ASCII y dependiendo de qué carácter ASCII se simule en la entrada así será la respuesta del módulo.

- Pulso de reloj.

2.4.6.4.2. Salidas

- Una única señal de tamaño 4 *bits* que simularan la activación de las bobinas A, B, C y D del motor paso a paso.

- La siguiente tabla indica las secuencias que se deben simular a la salida según el carácter ASCII que se ingrese en la entrada.

2.4.6.4.3. Caracteres ASCII

- V (01010110): activa la secuencia de velocidad
- T (01010100): activa la secuencia de torque
- D (01000100): movimiento hacia la derecha
- I (01001001): movimiento hacia la izquierda
- O (01001111): apagado del motor

2.4.6.5. Práctica 5: contador de 0 a 20

La práctica consiste en programar un contador de 0 a 20 en *display* de 7 segmentos en VHDL; se deben utilizar sentencias secuenciales y una señal de reloj de entrada. Por cada 5 flancos positivos de la señal de reloj el contador

debe aumentar su valor por uno y desarrollar la simulación de la señal de reloj utilizando Vivado. Luego de llegar al número 20 el contador debe reiniciar su valor a 0. Debe existir un *display* para las decenas y otro para las unidades.

2.4.6.6. Práctica 6: comunicación SPI

La práctica consiste crear un módulo en VHDL encargado de realizar una comunicación SPI utilizando la Zybo como maestro y un módulo esclavo; en este caso el MCP3008 (ADC de 10 bit de resolución con 8 canales ADC), el cual cuenta con interfaz SPI. Para efectos de simplicidad se simulará el módulo esclavo con las herramientas de simulación que posee Vivado. Los datos que serán simulados son: la señal de reloj CLK (reloj de la Zybo) y la línea de datos de entrada. El módulo programado en VHDL debe ser capaz de enviar el número de canal, recibir los datos del ADC y almacenar los datos en una memoria FIFO.

2.4.7. Diseño con jerarquía

Una de las ventajas más importantes de programar en FPGA es la organización modular y el diseño con jerarquías, que facilitan la detección de errores y orden en la programación. El diseño de jerarquías facilita la organización modular permitiendo crear varios módulos con distintas funciones que es encentren ejecutándose en paralelo. Para la unificación de todos los módulos agregados se requiere un módulo principal capaz de conectar todos los módulos dentro del FPGA de forma física.

Con el diseño de jerarquías es posible integrar módulos de terceros, reciclar líneas de código y simulación individual de cada módulo. En Vivado es

necesario declarar el módulo principal que conectará todos los módulos secundarios; este módulo principal se le conoce como módulo *top*.

Dentro del módulo *top* se encuentran todos los módulos declarados e instanciados, con sus respectivas señales de entrada y salida. El módulo *top* no debe llevar lógica de programación, únicamente la conexión de los módulos secundarios. También, comprende la declaración de señales de entrada y salida que corresponden a los pines físicos de la FPGA. Para realizar el diseño de jerarquías se requiere de dos pasos: la declaración e instanciación de todos los módulos secundarios dentro del módulo top.

2.4.7.1. Declaración de módulos

La declaración de módulos comprende la creación del componente de cada módulo secundario dentro del módulo top, en la sección de la arquitectura, antes del *begin*. Para el siguiente ejemplo se observa la entidad de un módulo secundario en la figura 35 que posteriormente se declarará al módulo *top*. Se crea un módulo secundario con el nombre de *switch* el cual realizará una función específica que en este momento no interesa.

Figura 35. **Puerto del módulo secundario *switch***

```
33  entity switch is
34      Port (
35              clk :          in STD_LOGIC;
36              cpu_ascii :    IN std_logic_vector(7 DOWNTO 0);
37              led :         out STD_LOGIC_vector(1 downto 0)
38            );
39  end switch;
40
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Se procede a crear el componte del módulo *switch* dentro del módulo *top* en función del puerto del módulo *switch* en la sección de la arquitectura, como se observa en la siguiente figura 36. Se observa que tiene las señales de entrada clk y 'cpu_ascii' y una señal de salida led, estas señales se estarán conectando posteriormente a otros módulos o a pines físicos del FPGA por medio de la instanciación.

Figura 36. **Creación del componente *switch***

```
209   component switch
210       Port (
211       clk : IN STD_LOGIC;
212       cpu_ascii : IN std_logic_vector(7 DOWNTO 0);
213       led : out STD_LOGIC_vector(1 downto 0)
214       );
215    end component;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

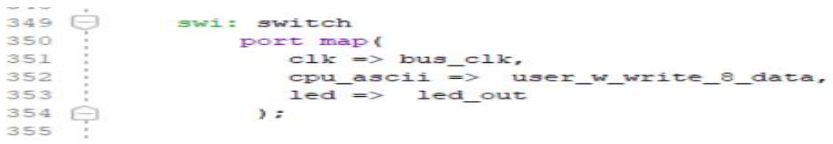
Las palabras *component* y *end component* son parte de la sintaxis de VHDL. El siguiente paso es la instanciación o conexión de las señales del módulo.

2.4.7.2. Instanciación

La instanciación del módulo comprende las conexiones internas entre las señales de todos los módulos secundarios y los pines físicos de la FPGA. Cada instanciación realizada debe llevar un nombre distinto y se debe realizar una instanciación por componente creado.

Las instanciaciones se declaran en la sección de la arquitectura luego del *begin,* es importante tomar en cuenta que solo se pueden instanciar señales del mismo tipo para evitar errores de implementación en el futuro. Cada instanciación relaciona las señales del componente proveniente a otras señales de módulos destinos o puertos, como se observa en la figura 37.

Figura 37. **Instanciación del módulo *switch***



Fuente: elaboración propia, utilizando Vivado 2017.4.

La etiqueta de instanciación es swi, el *Port map* indica el inicio de la conexión entre las señales provenientes del módulo y las señales destino. Las señales del lado izquierdo son las señales del módulo *switch* y las señales del lado derecho son las señales destino. La señal Clk se conectara con la señal Bus_clk, la señal de entrada 'cpu_ascii' se conectará con una señal 'destion User_w_write_8_data' y las señal led se conectara al puerto 'Led_out'.

2.4.8. Archivo ‘‘XDC’’

El archivo con extensión .’’xdc’’ comprende todos los pines físicos del FPGA y es el encargado de relacionar las señales de salida o de entrada del módulo principal (módulo top) con los pines físicos de la FPGA. Los archivos ‘‘XDC’’ se pueden agregar en el momento que se crea un nuevo proyecto o desde la pestaña Add source, seleccionando la opción de agregar un archivo Constraints y se agrega un archivo con extensión ‘‘XDC’’, como se observa en la figura 38.

Figura 38. **Agregar constraints**

Fuente: elaboración propia, utilizando Vivado 2017.4.

Cada modelo de FPGA posee un *Constraints* o un archivo ‘‘XDC’’ específico para ese modelo en especial debido a que cada FPGA tiene distintas cantidades de pines y numeración de pines diferente. En la figura se observa una sección del código descrito en el archivo ‘‘XDC’’ para la tarjeta de desarrollo Zybo Zynq 7000.

Figura 39. **Archivo ‘‘XDC’’**

```
91  ## Pmod Header JC
92  set_property -dict "PACKAGE_PIN V15 IOSTANDARD LVCMOS33" [get_ports "led_out[0]"]
93  set_property -dict "PACKAGE_PIN W15 IOSTANDARD LVCMOS33" [get_ports "led_out[1]"]
94  set_property -dict "PACKAGE_PIN T11 IOSTANDARD LVCMOS33" [get_ports "cmp1"]
95  set_property -dict "PACKAGE_PIN T10 IOSTANDARD LVCMOS33" [get_ports "cmp2"]
96  #set_property -dict "PACKAGE_PIN W14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[44]"]
97  #set_property -dict "PACKAGE_PIN Y14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[45]"]
98  #set_property -dict "PACKAGE_PIN T12 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[46]"]
99  #set_property -dict "PACKAGE_PIN U12 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[47]"]
100
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

En la figura 39 se observan los 8 pines del puerto JC de la Zybo Zynq 7000 y se relaciona el pin físico V15 con la señal 'Led_out[0] ' declarada en el puerto del módulo principal. También, se relacionan los pines W15 con la señal 'Led_out[0]', el pin T11 con la señal Cmp1 y el pin T10 con la señal Cmp2.

2.4.8.1. Práctica 7: Instanciación de módulos

La práctica consiste en instanciar 4 módulos diferentes, cada módulo tendrá dos entradas y una salida, cada módulo representará una compuerta básica a excepción del módulo principal. Dos de los módulos deben simular una operación And cada uno y el tercer módulo simulará una operación Or. El módulo principal debe usar para conectar las salidas y entradas a discreción propia; sin embargo, se debe cumplir con la regla de que cada módulo posea 2 entradas y una salida, todas conectadas entre sí exceptuando el módulo principal.

3. XILLINUX

Xillinux es un sistema operativo basado en Linux creado en conjunto con Xillinx para las tarjetas de desarrollo Zed board, Zybo y Micro Zed. Posee una interfaz gráfica amigable al usuario con un entorno similar a Ubuntu de Linux; sin embargo, es posible utilizar Xillinux desde la consola sin iniciar el escritorio gráfico. Cuando se instala Xillinux también se implementa de forma paralela el Ip Core Xillybus encargado de realizar interfaz de comunicación entre el procesador y la FPGA.

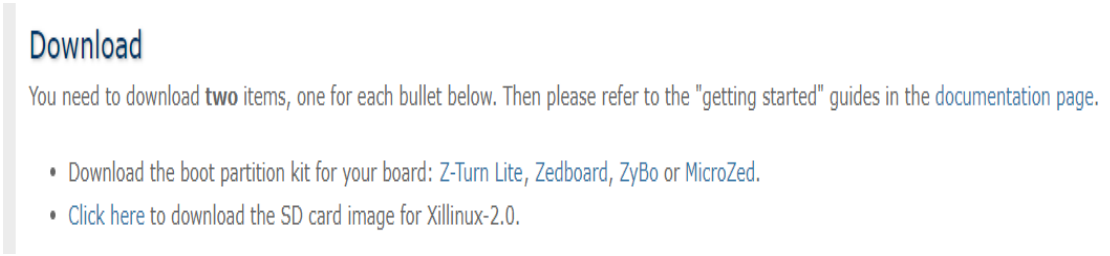
3.1. Instalación del sistema operativo Xillinux

Se realizará la instalación de Xillinux sobre la tarjeta de desarrollo Zybo-Zynq-7000 en su procesador, para realizar la instalación se requiere de una memoria micro SD de preferencia de la marca SandDisk de 4 GB de espacio como mínimo.

Para la instalación del sistema operativo primero se debe descargar el kit de partición y la imagen del sistema operativo Xillinux que proporciona Xillinx en su página oficial como se observa en la figura 40, utilizando el siguiente vínculo:

- http://xillybus.com/xillinux

Figura 40. **Xillinux, sección de descargas**



Fuente: Xillinux. https://www.google.com/search?biw=663&bih=594&tbm=isch&sa=1&ei=OBqyXMOPN8LIQLO. Consulta 21 de enero de 2018.

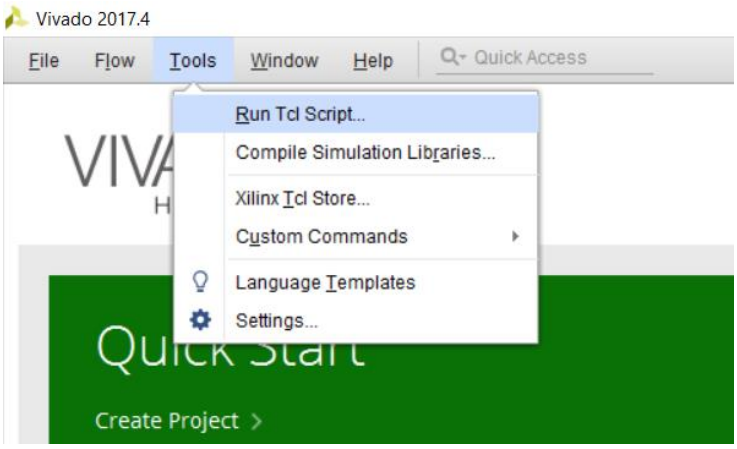
Se descargarán dos archivos:

- La imagen de Xillinux, el archivo con extensión .img
- Descomprimir el kit de instalación, el archivo con extensión .zip
- Verificar que el directorio donde se encuentra descomprimido el kit de partición no tenga espacios para evitar errores de consola; el nombre de todas las carpetas no tiene que contener espacios.

3.1.1. Generación del *bitstream*

Se ejecuta Vivado, desde la página de inicio se abre la pestaña *Tools* y se selecciona la opción *Run Tcl Script,* como se observa en la figura 41.

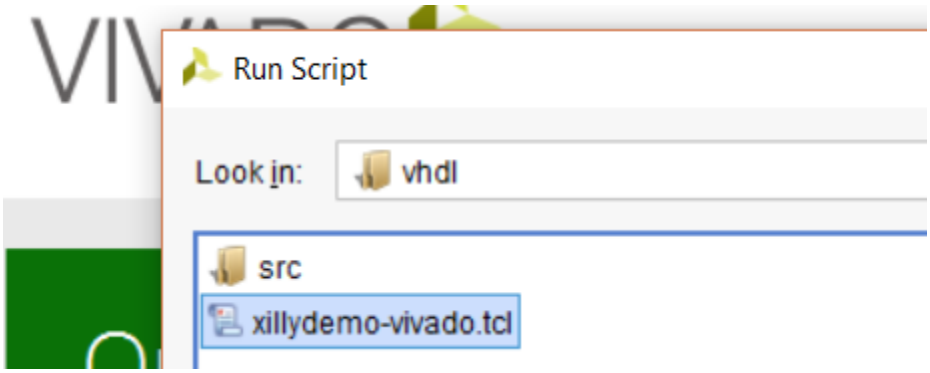
Figura 41. ***Run Tcl Script***



Fuente: elaboración propia, utilizando Vivado 2017.4.

Se busca la carpeta donde se encuentre descomprimido el kit de partición, se abre la carpeta Xillinux-eval-Zybo-2.0c; luego se abre la carpeta vhdl; se selecciona el archivo 'xillydemo-Vivado.tcl' y se presiona Ok para continuar, como se observa en la figura 42.

Figura 42. **Xillydemo-Vivado.tcl**



Fuente: elaboración propia, utilizando Vivado 2017.4.

Vivado creará un nuevo proyecto que Xilinix proporciona para la instalación de su sistema operativo. Si el proceso se realizó correctamente, en consola mostrará un mensaje que el proyecto fue creado con el nombre de xillydemo. Dentro del panel de control sources, se debe abrir el módulo principal Xillydemo.vhd, dentro del *Port* del módulo se comentaría o se elimina las señales 'PS_CLK', 'PS_PORB' y 'PS_SRSTB', como se observa en la figura 43.

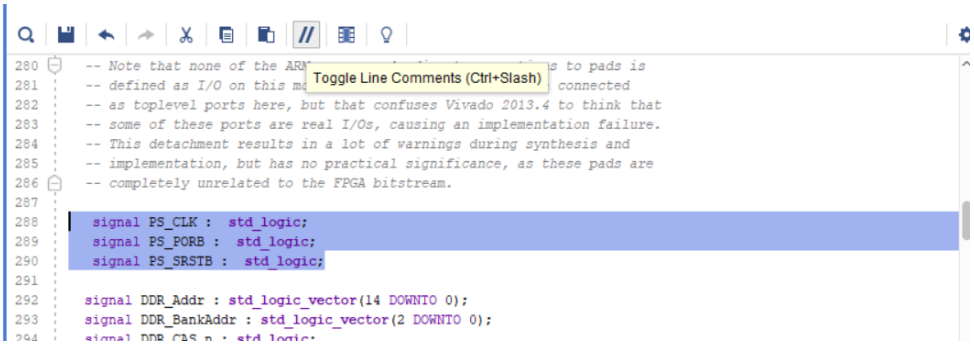
Figura 43. **Port Xillydemo**

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  entity xillydemo is
7    port (
8      -- For Vivado, delete the port declarations for PS_CLK, PS_PORB and
9      -- PS_SRSTB, and uncomment their declarations as signals further below.
10
11 --     PS_CLK : IN std_logic;
12 --     PS_PORB : IN std_logic;
13 --     PS_SRSTB : IN std_logic;
14     clk_100 : IN std_logic;
15     otg_oc : IN std_logic;
16     PS_GPIO : INOUT std_logic_vector(55 DOWNTO 0);
17     GPIO_LED : OUT std_logic_vector(3 DOWNTO 0);
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

En el mismo módulo 'Xillydemo.vhd' se busca la sección declaraciones de señales y se Desconectarían las señales 'S_CLK', 'PS_PORB' y 'PS_SRSTB', como se observa en la figura 44.

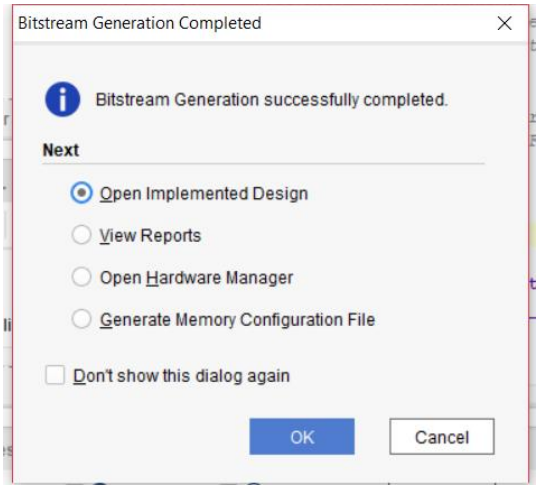
Figura 44. **Declaración de señales dentro del Xillydemo**



Fuente: elaboración propia, utilizando Vivado 2017.4.

Luego se procede a sintetizar, implementar y generar el *bitstream* con las herramientas de Vivado que se encuentran en el *flow navigator;* si el proceso se realizó de manera correcta, Vivado desplegará un mensaje de confirmación, como se muestra en la figura 45.

Figura 45. ***Bits*tream Generation completed**



Fuente: elaboración propia, utilizando Vivado 2017.4.

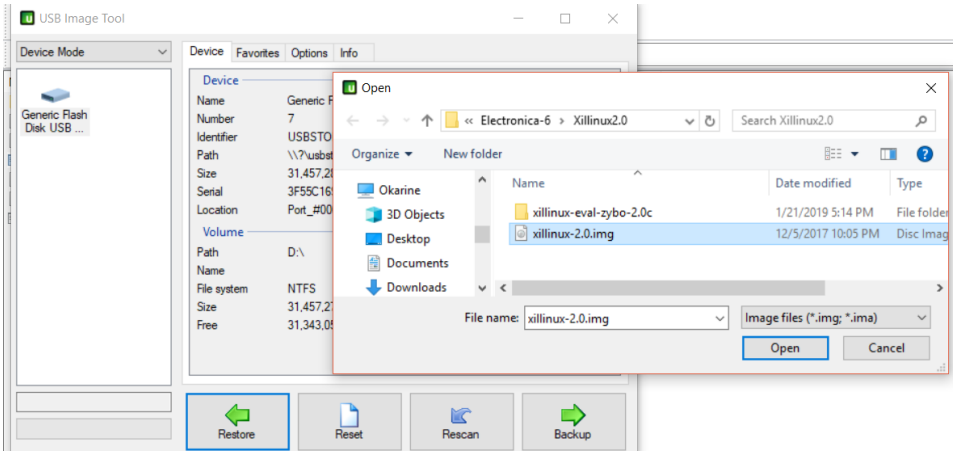
El archivo con extensión .bit se encuentra ubicado en la dirección de carpetas xillinux-eval-Zybo-2.0c, vhdl, Vivado, 'xillydemo.runs, 'impl_1' y el nombre del archivo con extensión .bit es ‘xillydemo.bit’.

3.1.2. Preparación de la memoria micro SD

Se recomienda utilizar una memoria de marca SandDisk de clase 4 o mayor con capacidad de 4 Gb como mínimo para que se pueda ejecutar la interfaz gráfica sin ningún problema.

Se procede a descargar la herramienta ‘‘USB Image Tool.exe’’ del siguiente link: https://download.cnet.com/USB-Image-Tool/3000-2242_4-75449768.html. Esta herramienta no necesita estar instalada en la computadora, es suficiente con iniciar el ejecutable con método gráfico. Luego de descomprimir el archivo, se procede a ejecutar el archivo ‘USB Image Tool.exe’, como se observa en la figura 46.

Figura 46. ***USB Image Tool***

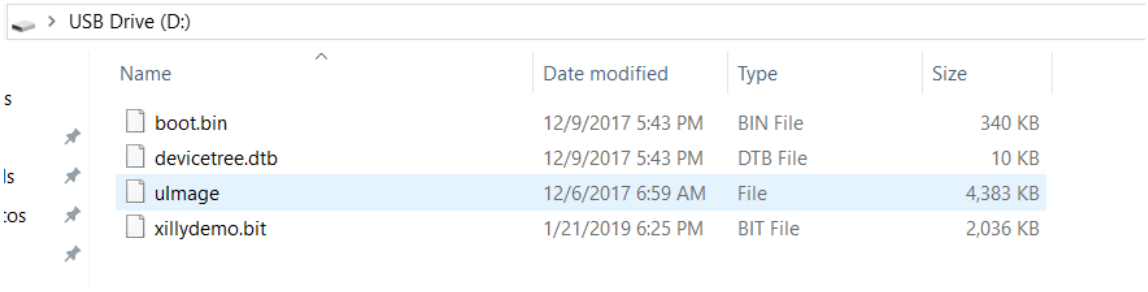


Fuente: elaboración propia, utilizando *USB Image Tool.*

Se selecciona la memoria micro SD; luego, se presiona el botón de 'Restore' y se busca el archivo Xillinux-2.0.img previamente descargado el cual posee la imagen del sistema operativo. Se presiona el botón de open para que inicie con la restauración de la imagen dentro de la memoria SD. Luego de la restauración es posible que Windows no sea capaz de reconocer la memoria micro SD y despliegue errores; esto es normal debido a que el sistema Xillinux se encuentra basado en Linux.

En el kit de partición Xillinux-eval-Zybo-2.0c en la carpeta 'bootfiles' se encuentran los archivos los archivos 'boot.bin' y 'devicetree.dtb', estos archivos se deben de copiar y pegar dentro de la memoria micro SD. Por último, se debe de copiar el archivo con extensión .bit (generado por Vivado en la sección 3.1.1) dentro de la memoria micro SD, como se observa en la figura 47.

Figura 47. **Memoria micro SD terminada**



Fuente: elaboración propia, utilizando explorador de Windows.

La memoria micro SD se encuentra lista para insertarla en la tarjeta de desarrollo Zybo-Zynq-7000 e iniciarla con un monitor y un teclado.

3.2. Prueba de retroalimentación

La prueba de retroalimentación o 'loopback' consiste en verificar la conexión que tiene el sistema operativo con la FPGA de la Zybo-Zynq-7000, internamente se encuentran conectadas por defecto dos memorias FIFO en modo de retroalimentación permitiendo verificar que el sistema operativo se encuentre correctamente instalado y evalúa la existencia de una comunicación correcta entre la FPGA y el mismo sistema operativo.

Se inicia la Zybo-Zynq-7000 conectándola, una vez cargada la consola se inicia la interfaz gráfica colocando 'startx' y luego se presiona 'enter'. Se iniciará el protocolo de ejecución de la interfaz gráfica del sistema operativo Xillinux.

Se abren dos consolas, una se configurará para recibir y mostrar toda la información recibida por la FPGA y la otra se configurará para escribir o mandar información a la FPGA.

La FPGA cuenta con dos memorias FIFO implementas:

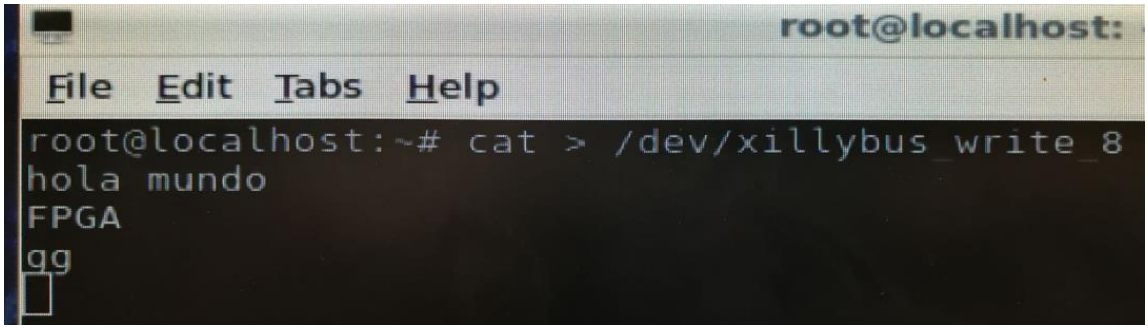
- Memoria FIFO de bus de datos 8 *bits* y 2048 celdas
- Memoria FIFO de bus de datos 32 *bits* y 512 celdas

La prueba de retroalimentación se puede realizar utilizando cualquiera de las dos memorias; únicamente se debe cambiar el número 8 por un número 32. Es importante tener en cuenta que las dos consolas deben estar enfocadas en una solo memoria FIFO para que resulte exitosa.

3.2.1. Consola en modo escritura

Se abre una consola con el comando 'ctrl + alt +t' y se coloca el comando: 'cat > /dev/xillybus_write_8', como se puede observar en la figura

Figura 48. **Consola en modo escritura**



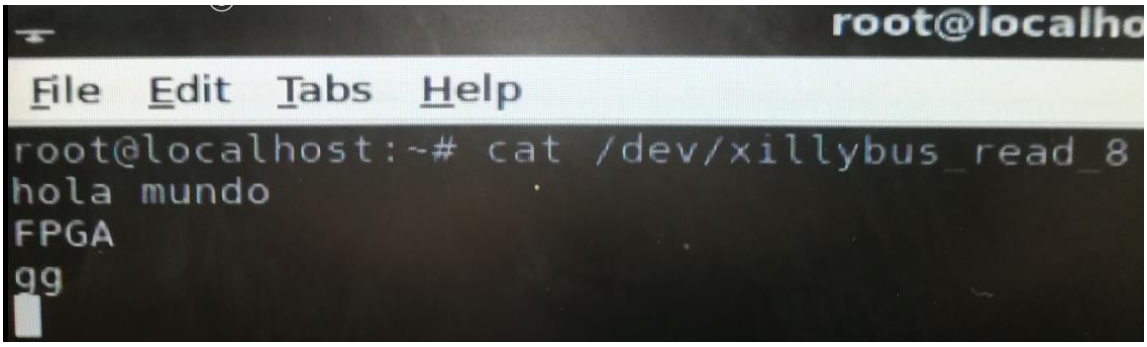
Fuente: elaboración propia, utilizando Xillinux.

Esto configura la consola en modo escritura sobre la memoria FIFO de bus de datos 8*bits*, todo lo que escribamos dentro de esta consola se enviará a la FPGA presionando la tecla enter por medio de la memoria FIFO.

3.2.2. Consola en modo lectura

Se abre otra consola paralela con el comando ctrl + alt +t y se coloca el comando: 'cat /dev/xillybus_read_8' con el objetivo de recibir la información, como se observa en la figura 49.

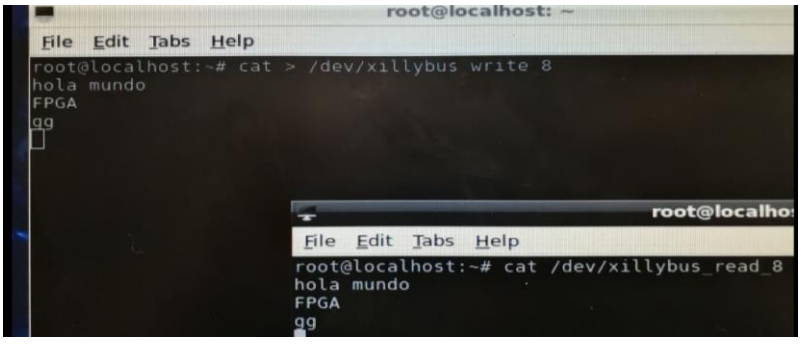
Figura 49. **Consola en modo lectura**



Fuente: elaboración propia, utilizando Xillinux.

Esto configura la consola en modo lectura sobre la memoria FIFO de bus de datos 8*bits*; toda la información que reciba el sistema operativo de la FPGA se podrá capturar en la consola para su análisis posterior. La prueba resulta exitosa cuando la misma data que se envía por la consola en modo escritura aparece en la consola configurada en modo lectura, como se observa en la figura.

Figura 50. **Prueba de Loopback**



Fuente: elaboración propia, utilizando Xillinux.

Toda la información viaja por medio del Ip Core Xillybus de ida y vuelta. Todos los caracteres que sean escritos en la consola en modo escritura serán enviados y reenviados por el bus de datos Xillybus incluyendo el salto de línea realizado con la tecla enter, esto debido a que se toman en cuenta toda la tabla de caracteres ASCII.

3.3. Práctica 8: instalación de Xillinux

La práctica consiste en instalar el sistema operativo Xillinux sobre la tarjeta de desarrollo Zybo-Zynq-7000. Se debe generar el *bitstream* por medio de Vivado y cargarlo en la tarjeta SD. Se debe realizar la prueba de 'loopback' desde la consola de Xillinux para verificar que el sistema operativo y el *bits*tream estén correctamente instalados utilizando los pasos previamente descritos en el capítulo 3.

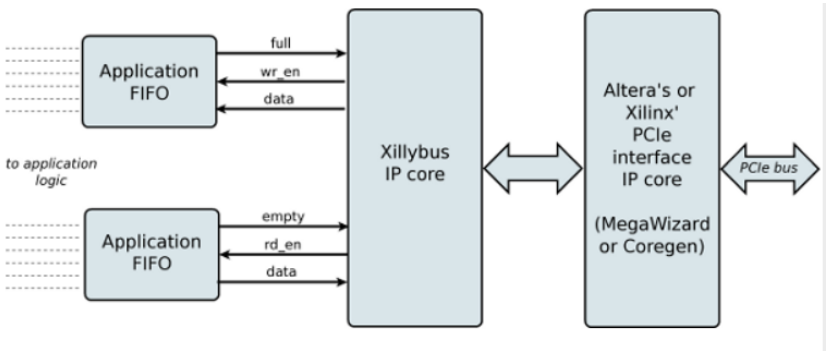
4. COMUNICAIÓN ENTRE EL PROCESADOR CORTEX A9 Y LA FPGA

El procesador donde se encuentra instalado el sistema operativo y la FPGA son dos entornos distintos, el Ip Core que se encarga de comunicar la información entre ellos es el Xillybus.

4.1. Xillybus

El Xillybus es un Ip Core proporcionado por Xillinx que trabaja como un bus de datos comunicando las dos memorias FIFO de 8 *bits* y 32 *bits* con el sistema operativo, como se observa en la figura 51.

Figura 51. **Xillybus**



Fuente: Xillinx. http://xillybus.com/doc/Xillinx-pcie-principle-of-operation, Ip Core Xillybux, Consulta 22 de enero de 2019.

El Xillybus es proporciona flujo de datos en ambas direcciones con una interfaz de usuario directa. Los datos que se transportar por el bus de direcciones pertenecen a la tabla de caracteres ASCII.

4.1.1. Caracteres ASCII

Los caracteres ASCII pertenecen un estándar de código americano para el intercambio de información proveniente de los estados unidos. Los caracteres ASCII son la base de cualquier tipo de comunicación en texto, se utilizan en mensajería digital, móvil, vía internet, entre otros. Un carácter ASCII es un código binario de 8 *bits*, existen 255 caracteres ASCII que comprende todos los símbolos, números, letras mayúsculas y minúsculas; absolutamente todos los códigos posibles que un teclado puede escribir representado en unos y ceros.

4.2. Comunicación CPU – FPGA

La comunicación CPU – FPGA comprende todos los datos que se transmiten desde la consola de Xillinux a la FPGA. Primero, se analizará la información en la dirección CPU a FPGA, para el análisis de esta comunicación se utilizará un ejemplo de elaboración propia. El ejemplo consiste en transmitir dos caracteres ASCII, la letra a y b, cada letra deberá activar y desactivar un led, cada led se encenderá y se pagará después de un intervalo de tiempo continuamente.

Los led estarán alternando su estado en dos pines diferentes de la Zybo. Se agregará un módulo secundario al proyecto Xillydemo encargado de controlar la activación de los led en función de los caracteres ASCII. Se estará trabjando dentro del proyecto Xillydemo previamente creado en el capítulo 3.

4.2.1. Creación del nuevo módulo *switch*

Primero se crea un módulo llamado *switch* en cargado de interpretar los caracteres ASCII recibidos desde el Xillybus y en función del carácter ASCII recibido se activará uno de los dos led osciladores. Este módulo se debe crear dentro del proyecto Xillydemo debido a que la instalación del sistema operativo se encuentra dentro del mismo. Para la creación del módulo *switch* se debe recurrir a la herramienta 'add sources' de Vivado como se explicó en la sección 1.4.3 del documento.

Después de crearlo, se agregará las librerías respectivas y se declaran las señales con las que se estarán trabajando como se observa en la figura 52. Las librerías estandarizadas por el Instituto de Ingenieros Eléctricos y Electrónicos permiten la utilización de valoras de tipo 'std_logic y std_logic_vector', entre otras sentencias y valores.

Figura 52. **Señales del módulo *switch***

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.NUMERIC_STD.ALL;
25
26  entity switch is
27      Port (
28            clk :         in STD_LOGIC;
29            cpu_ascii :   IN std_logic_vector(7 DOWNTO 0);
30            led :        out STD_LOGIC_vector(1 downto 0)
31          );
32  end switch;
33
34  architecture Behavioral of switch is
35  signal outStateA : std_logic;                       --LED A
36  signal outStateB : std_logic;                       --LED B
37  signal flagA : std_logic;                           --indica
38  signal flagB : std_logic;                           --indica
39  signal ConteoA: integer range 0 to 119999999;       --variab
40  signal ConteoB: integer range 0 to 119999999;       --variab
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

En la figura 52 se observan las señales declaradas dentro del puerto y las señales locales para el funcionamiento interno; las señales son:

- Clk: señal de reloj proveniente de la FPGA, su frecuencia de oscilación es de125Mhz.

- Cpu_ascii: señal de tamaño 8 *bits* encargada de recibir el código ASCII enviado por el Xillybus.

- Led: señal de tamaño 2 *bits* encargada de modificar el estado (encendido o apagado) de los dos pines físicos de la FPGA.

- FlagA: señal de tamaño un bit encargada de indicar si el carácter ASCII ‘a’ fue enviado desde el bus de direcciones Xillybus.

- FlagB: señal de tamaño un bit encargada de indicar si el carácter ASCII ‘b’ fue enviado desde el bus de direcciones Xillybus.

- OutstateA: determina el estado de salida del led, cambia su valor cuando se activa la señal FlagA.

- OutstateB: determina el estado de salida del led, cambia su valor cuando se activa la señal FlagB.

- ConteoA: señal de tipo entero con rango de 1 a 119999999 que se utilizará más adelante para un contador, con el objetivo de generar la frecuencia de oscilación de la señal outstateA.

- ConteoB: señal de tipo entero con rango de 1 a 119999999 que se utilizará más adelante para un contador, con el objetivo de generar la frecuencia de oscilación de la señal outstateB.

4.2.2. Funcionamiento del módulo *switch*

Para la activación y desactivación de los led oscilantes, primero se realizó un registro de tamaño de un bit utilizando las señales FlagA y FlagB. Este registro cambiará su estado entre 1 lógico y 0 lógico cada vez que se reciba el código ASCII correspondiente a la señal 'cpu_ascii' como se observa en la figura 53.

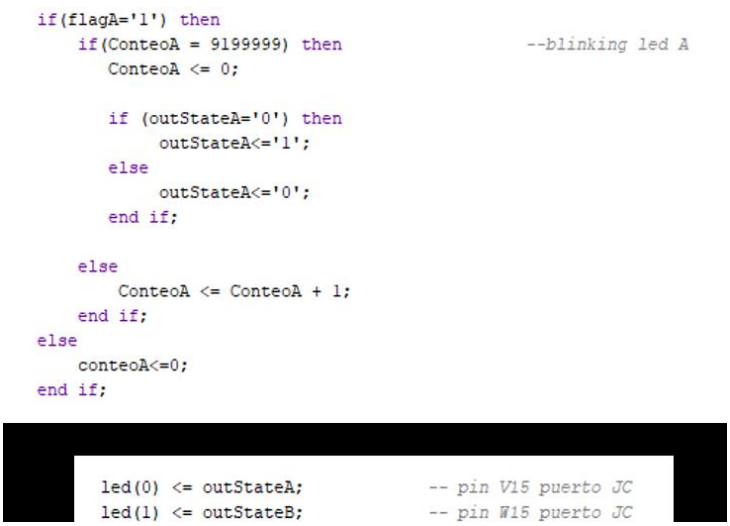
Figura 53. **Activación de los leds oscilantes**

```
process (clk)
    begin
        if (rising_edge(clk))then
            case cpu_ascii is
                when "01100001" =>                  -- caracter ascci 'a'
                    if (flagA='0')then              --flipflop
                        flagA<='1';                 --flipflop
                    else                            --flipflop
                        flagA<='0';                 --flipflop
                    end if;                         --flipflop
                when "01100010" =>                 -- caracter ascci 'b'
                   if (flagB='0')then               --flipflop
                        flagB<='1';                 --flipflop
                    else                            --flipflop
                        flagB<='0';                 --flipflop
                    end if;                         --flipflop
                when others =>
                    flagA <=flagA;
                    flagB <=flagB;
            end case;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Para generar de la oscilación de los led, ubicados en los pines físicos de la Zybo, se implementó la sentencia secuencial IF la cual permite llevar un conteo de flancos positivos de la señal de reloj dentro de las señales ConteoA y ConteoB respectivamente de cada led, como se observa en la figura 54.

Figura 54. **Contador para la frecuencia de oscilación**



```
if(flagA='1') then
    if(ConteoA = 9199999) then                --blinking led A
       ConteoA <= 0;

       if (outStateA='0') then
            outStateA<='1';
       else
            outStateA<='0';
       end if;

    else
        ConteoA <= ConteoA + 1;
    end if;
else
    conteoA<=0;
end if;
```

```
led(0) <= outStateA;            -- pin V15 puerto JC
led(1) <= outStateB;            -- pin W15 puerto JC
```

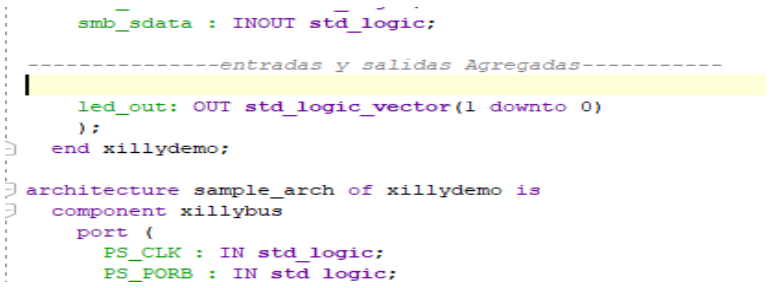
Fuente: elaboración propia, utilizando Vivado 2017.4.

Los contadores no empezarán a contar hasta que las señales FlagA y FlagB se activen, respectivamente, cada una para su propio contador, como se observa en la figura 54. Cada contador comprende un límite distinto (diferentes frecuencias de oscilación), cuando se llega a este límite el valor del contador se reinicia a 0. Se observa en la figura 54 que la señal led en la posición 0 y led en la posición 1 recibirán el valor de las señales outstateA y outstateB, respectivamente.

4.2.3. Instanciación dentro del Xillydemo

El segundo paso es instanciar el módulo *switch* dentro del proyecto Xillydemo. Dentro del puerto del Xillydemo se declaran todas las señales físicas a utilizarse en la FPGA; en este caso se utilizará el vector 'led_out' tipo 'std_logic_vector' y tamaño 2 *bits*, como se observa en la figura 55.

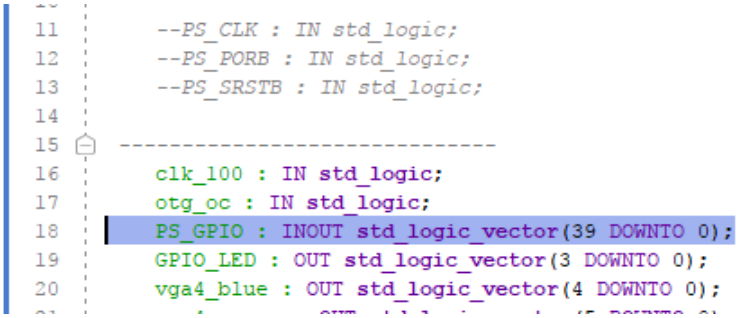
Figura 55. **Entradas y salidas agregadas**



Fuente: elaboración propia, utilizando Vivado 2017.4.

Dentro del puerto también se modifica en tamaño del vector 'PS_GPIO a 39 *downto* 0' con el objetivo de desbloquear ciertos pines físicos de la FPGA, como se observa en la figura 56.

Figura 56. **Desbloqueo de pines**



Fuente: elaboración propia, utilizando Vivado 2017.4.

Como se observa en la figura 57, se liberaron los puertos JC y JD (pines del 40 al 55), se modifica el nombre de los pines V15 y W15 a 'led_out[0] ' y 'led_out[1]', respectivamente. El resto de pines que no se estén utilizando se comentaría. Para futuras aplicaciones se pueden disponer de cualquier pin que

se encuentre en los puertos JC y JD; si se desea utilizar más pines aparte de estos dos puertos se necesita modificar los valores del vector PS_GPIO para desbloquear más puertos.

Figura 57. ***Constraints* de la Zybo**

```
## Pmod Header JC
set_property -dict "PACKAGE_PIN V15 IOSTANDARD LVCMOS33" [get_ports "led_out[0]"]
set_property -dict "PACKAGE_PIN W15 IOSTANDARD LVCMOS33" [get_ports "led_out[1]"]
#set_property -dict "PACKAGE_PIN V15 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[40]"]
#set_property -dict "PACKAGE_PIN W15 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[41]"]
#set_property -dict "PACKAGE_PIN T11 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[42]"]
#set_property -dict "PACKAGE_PIN T10 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[43]"]
#set_property -dict "PACKAGE_PIN W14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[44]"]
#set_property -dict "PACKAGE_PIN Y14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[45]"]
#set_property -dict "PACKAGE_PIN T12 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[46]"]
#set_property -dict "PACKAGE_PIN U12 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[47]"]

### Pmod Header JD
#set_property -dict "PACKAGE_PIN T14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[48]"]
#set_property -dict "PACKAGE_PIN T15 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[49]"]
#set_property -dict "PACKAGE_PIN P14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[50]"]
#set_property -dict "PACKAGE_PIN R14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[51]"]
#set_property -dict "PACKAGE_PIN U14 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[52]"]
#set_property -dict "PACKAGE_PIN U15 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[53]"]
#set_property -dict "PACKAGE_PIN V17 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[54]"]
#set_property -dict "PACKAGE_PIN V18 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[55]"]
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Se crea el componente del módulo *Switch,* como se observa en la figura 58.

Figura 58. **Componente del módulo *switch***

```
component switch
    Port (
    clk : IN STD_LOGIC;
    cpu_ascii : IN std_logic_vector(7 DOWNTO 0);
    led : out STD_LOGIC_vector(1 downto 0)
    );
 end component;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Se procede a instanciarlo, como se observa en la figura 59.

Figura 59. **Instanciación del módulo switch**

```
    swi: switch
        port map(
           clk => bus_clk,
           cpu_ascii =>  user_w_write_8_data,
           led =>  led_out
         );
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

La señal de clk se conecta con la señal 'bus_clk' la cual comprende la señal de reloj de la Zybo. Cpu_ascii se conecta a la señal 'user_w_write_8_data' (comprende toda la información enviada del Xillybus a la memoria FIFO de 2048 y de tamaño 8 *bits*). La señal led se conecta a la señal de los pines físicos con el nombre led_out como se observa en la figura 59.

Se crea una señal auxiliar con el nombre de ‘complemento’ para modificar la instanciación del Xillybus, esto con el objetivo de liberar los puertos físicos JC y JD de la FPGA, como se observa en la figura 60.

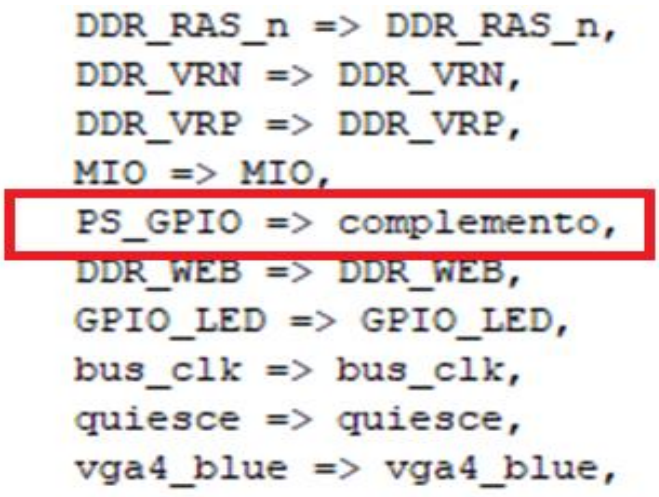
Figura 60. **Señal auxiliar ‘complemento’**

```
signal complemento :std_logic_vector(55 downto 0);

begin

complemento(55 downto 40) <= (others => 'Z');
complemento(39 downto 0) <= PS_GPIO(39 downto 0);
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Todas las posiciones de la 0 a la 39 se conectan a la señal PS_GPIO y las posiciones de la 40 a la 55 se colocan en alta impedancia, como se observa en la figura 60. Por último, dentro de la instanciación del Xillybus es necesario conectar la señal PS_GPIO a la señal complemento como se observa en la figura 61.

Figura 61. **Instanciación del Xillybus**



Fuente: elaboración propia, utilizando Vivado 2017.4.

Desde la consola de la figura 48 desde el Xillinux, cuando se envíe un carácter ASCII ‘a’, se activará un la oscilación de un led en el pin V15 y cuando se vuelva a mandar el mismo carácter ASCII se desactivará la oscilación del mismo led. El funcionamiento para el carácter ‘b’ es el mismo, lo único distinto es el pin W15 que activaría un led distinto. En la figura 62 se observa los nombres que corresponden a los pines de la Zybo.

Figura 62. **Configuración de pines físicos de la Zybo**

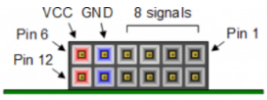


Figure 16. Pmod diagram.

Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Hi-Speed)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15

Fuente: Laboratorio de Electrónica Universidad de San Carlos. *Electrónica 6.* www.labelectronica.weebly.com/electronica6.html. Consulta: 3 de enero 2019.

4.2.4. Práctica 9: comunicación CPU – FPGA

La práctica consiste en controlar una modulación por ancho de pulso desde la consola del sistema operativo Xillybus. Se deben seleccionar 10 caracteres de la tabla ASCII para enviarlos desde la consola; cada carácter ASCII representará un ancho de pulso distinto. La frecuencia de la señala debe ser constante, se debe modificar el tiempo en bajo y el tiempo en alto para cada codigo ASCII sin alterar el periodo de la señal. Se debe utilizar el Ip Core Xillybus para la transmisión de datos.

4.3. Comunicación FPGA – CPU

La comunicación FPGA – CPU es la transmisión de códigos ASCII enviados desde la FPGA hasta el sistema operativo, para el análisis de esta

comunicación se utilizará un ejemplo de elaboración propia. A diferencia de la comulación CPU a FPGA, se utiliza un registro de tipo 'std_logic' que funciona como habilitador en la escritura dentro del Xillybus, el nombre de la señal que comprende el registro es user_r_read_8_Empty; como se observa en la figura 63. El I'p Core Xillybus posee un registro por cada memoria FIFO, uno para la memoria de bus de datos 8 *bits* y otra para la memoria de bus de datos 32 *bits*.

Figura 63. ***User_r_read_8_Empty***

```
101     user_r_read_32_eof : IN std_logic;
102     user_r_read_32_open : OUT std_logic;
103     user_r_read_8_rden : OUT std_logic;
104     user_r_read_8_empty : IN std_logic;
105     user_r_read_8_data : IN std_logic_vector(7 DOWNTO 0);
106     user_r_read_8_eof : IN std_logic;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

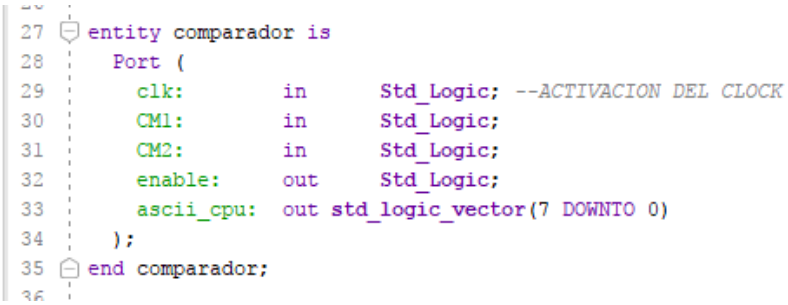
4.3.1. Creación de nuevo módulo comparador

El siguiente ejemplo consta de enviar dos caracteres: ‘a’ y ‘b’ desde, la FPGA a la consola del sistema operativo. Se contará con dos botones que simularán la activación del envió de los caracteres ASCII: un botón activará la transmisión de datos y otro botón la detendrá. Los botones se encontrarán ubicados en los pines de la FPGA de la Zybo. Los caracteres ASCII se mostrarán en la consola de la figura 49.

Se creará un nuevo módulo llamado 'comparador' el cual se encargará del control de los botones y generar la transmisión de datos desde la FPGA a la consola con el comando ejecutado, como se observa en la figura 49. El módulo

comprende la lectura del estado en dos pines de la FPGA, la transmisión de los códigos ASCII y el control del registro *user_r_read_8_Empty.* En la figura 64 se observa la declaración de las señales de entrada y salida del módulo comparador; este módulo se agregará en paralelo con el módulo *switch* dentro del proyecto Xillydemo.

Figura 64. **Señales del módulo comparador**



Fuente: elaboración propia, utilizando Vivado 2017.4.

A continuación se enlistan las señales del módulo comparador.

- Clk: señal de reloj proveniente de la FPGA, su frecuencia de oscilación es de125Mhz.

- Cpu_ascii: señal de tamaño 8 *bits* encargada de recibir el código ASCII enviado por el Xillybus.

- CM1: señal de entrada donde se encuentra conectado el primer botón.

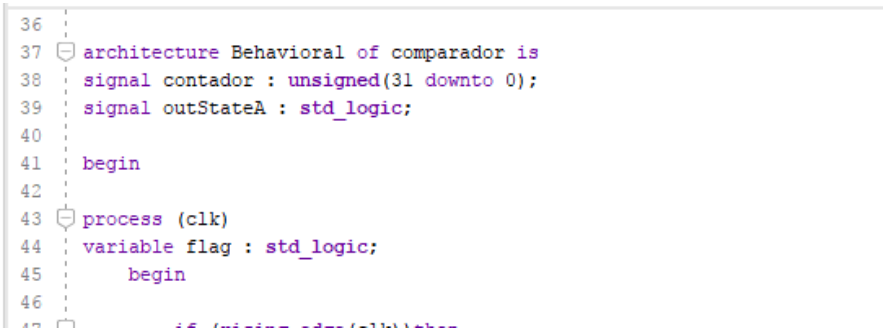
- CM2: señal de entrada donde se encuentra conectado el segundo botón.

- *Enable*: señal de salida encargada de indicar una escritura dentro del Xillybus, se conectara posteriormente a la señal user_r_read_8_Empty.

- ascii_cpu: señal de salida de tipo 'std_logic_vector' *de tamaño* 8 *bits*, encargada de transmitir los códigos ASCII uno a uno al módulo Xillybus.

Se declaran señales internas para el funcionamiento del módulo, como se observa en la figura 65.

Figura 65. **Señales internas**



Fuente: elaboración propia, utilizando Vivado 2017.4.

- Contador: señal de tipo entero sin signo encargada de contar ciclos de reloj con el objetivo de generar un tiempo de espera dentro de la ejecución del proyecto.

- outStateA: señal encargada de guardar el estado de un flip flop utilizado más adelante.

- flag: señal de tipo std_logic encargada de controlar la activación de los botones colocados en CM1 y CM2.

4.3.2. Funcionamiento del módulo comparador

La activación de los botones se realiza por medio de la lectura de las señales CM1 y CM2; el estado de la señal flag cambiará en función del botón que se presione. Si se activa el botón de la señal CM1, flag guardará un 1 lógico y si se activa el botón ubicado en la señal CM2 flag guardará un 0 lógico, como se observa en la figura 66.

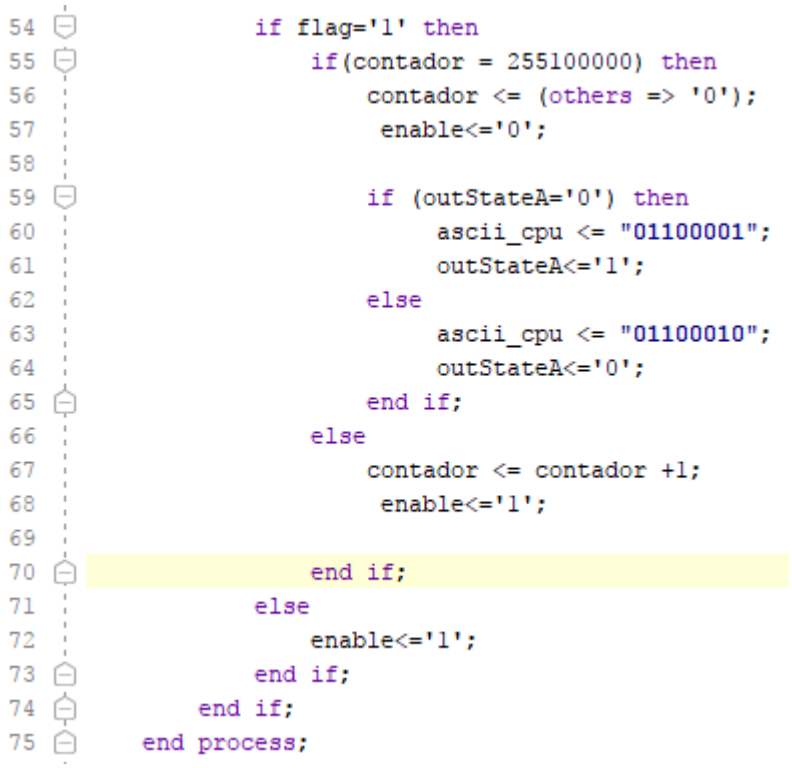
Figura 66. **Lectura de señales**

```
3  process (clk)
4  variable flag : std_logic;
5      begin
6          if (rising_edge(clk))then
7              if CM1='1' then
8                  flag := '1';
9              end if;
0              if CM2='1' then
1                  flag := '0';
2              end if;
3
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

El funcionamiento del módulo consiste en la transmisión de los códigos ASCII “01100001” y “01100010” equivalentes a los caracteres a y b. Cuando la señal flag tiene un estado de 1 lógico, la señal contador inicia su conteo de flancos positivos de la señal de reloj. Después de que la señal contador se llega al límite de 255100000, la señal enable cambia su estado lógico de 0 a 1 habilitando la transmisión en el 'user_r_read_8_Empty' y se guarda un código ASCII en la señal ascii_cpu. Para alternar el carácter se utiliza la señal outStateA combinada con una sentencia secuencial para intercambiar los caracteres ‘a’ y ‘b’ de forma oscilante como se observa en la figura 67.

Figura 67. **Funcionamiento del módulo comparador**



```
54          if flag='1' then
55              if(contador = 255100000) then
56                  contador <= (others => '0');
57                   enable<='0';
58
59                  if (outStateA='0') then
60                       ascii_cpu <= "01100001";
61                       outStateA<='1';
62                  else
63                       ascii_cpu <= "01100010";
64                       outStateA<='0';
65                  end if;
66              else
67                  contador <= contador +1;
68                   enable<='1';
69
70              end if;
71          else
72              enable<='1';
73          end if;
74      end if;
75  end process;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

El Xillybus transmite la información al CPU cuando la señal 'user_r_read_8_Empty' recibe un flanco de subida, la señal enable se encarga de generar ese flanco de subida cambiando su estado en función de la señal contador. El siguiente paso es agregar las señales físicas CM1 y CM2 que se utilizarán en el módulo, desde el puerto del Xillydemo, como se observa en la figura 68.

Figura 68. **Puerto del Xillydemo**

```
38      smb_sdata : INOUT std_logic;
39
40  ---------------entradas y salidas Agregadas-----------
41      cmp1:   in std_logic;
42      cmp2:   in std_logic;
43      led_out: OUT std_logic_vector(1 downto 0)
44      );
45  end xillydemo;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Se agregan las señales cmp1 y cmp2 de tipo 'std_logic' para conectarlas posteriormente con las señales CM1 y CM2.

4.3.3. Instanciación dentro del Xillydemo

Se procede a crear a la instanciación del módulo dentro del proyecto general Xillydemo; primero se crea el componente del módulo en la sección de componentes, como se observa en la figura 69.

Figura 69. **Componente del módulo comparador**

```
217   component comparador
218     Port (
219     clk:        in      Std_Logic;
220     CM1:        in      Std_Logic;
221     CM2:        in      Std_Logic;
222     enable:     out     Std_Logic;
223     ascii_cpu:  out     std_logic_vector(7 DOWNTO 0)
224   );
225   end component;
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

A diferencia de la comunicación CPU – FPGA es necesario Desconectar las señales de la memoria FIFO debido a que la memoria también escribe en el Xillybus y el objetivo es que el módulo comparador escriba en el Xillybus, por ende, es imposible hacer que dos señales de escritura modifiquen una de lectura recordando que VDHL es un lenguaje de descripción de su hardware. Para descontar las señales de la memoria FIFO se procede a declarar dos señales extra dentro del Xillydemo, como se observar en la figura 70.

Figura 70. **Señales agregadas**

```
338  --------------Señales Agregadas ------------------------
339  signal complemento :std_logic_vector(55 downto 0);
340  signal desconectar1 :  std_logic_vector(7 DOWNTO 0);
341  signal desconectar2 :  std_logic;
342
343
344  begin
345
346  complemento(55 downto 40) <= (others => 'Z');
347  complemento(39 downto 0) <= PS_GPIO(39 downto 0);
348
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Se agrega una señal con el nombre de Desconectar1 de tipo 'std_logic' y una señal Desconectar2 de tipo 'std_logic_vector' de tamaño 8 *bits*. Luego se procede a modificar la instanciación de la memoria FIFO que se estará utilizando, para este ejemplo la memoria FIFO de 8 *bits*. Se conectará la señal *dout* de la memoria FIFO con la señal Desconectar1 y la señal *Empty* con Desconcetar2, como se observa en la figura 71.

Figura 71. **Instanciación de la memoria FIFO de 8*bits***

```
fifo_8 : fifo_8x2048
  port map(
    clk        => bus_clk,
    srst       => reset_8,
    din        => user_w_write_8_data,
    wr_en      => user_w_write_8_wren,
    rd_en      => user_r_read_8_rden,
    dout       => desconectar1,
    full       => user_w_write_8_full,
    empty      => desconectar2
    );
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

La modificación de la instanciación de la memoria FIFO permite Desconectar las señales del Xillybus para transmitir en el CPU la información que se decidan con el módulo comparador. El siguiente paso es realizar la instanciación del módulo comparador, como se observa en la figura 72.

Figura 72. **Instanciación del módulo comparador**

```
crn: comparador
     port map(
         clk => bus_clk,
         CM1 => cmp1,
         CM2 => cmp2,
         enable => user_r_read_8_empty,
         ascii_cpu =>  user_r_read_8_data
      );
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Se conecta la señal de clk con la señal de reloj de la Zybo bus_clk, se conecta las señales CM1 y CM2 con las señales cmp1 y cmp2, respectivamente, que representarán a los pines de la Zybo; se conecta la señal enable con la señal 'user_r_read_8_Empty' del Xillybus y se conecta la señal ascii_cpu con la señal 'user_r_read_8_data' del Xillybus.

El siguiente paso es relacionar las señales cmp1 y cmp2 con los pines físicos de la Zybo T11 y T10, respectivamente, utilizando el *Constraints* de la Zybo como se observa en la figura 73.

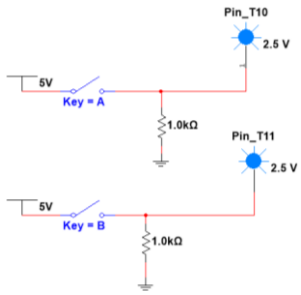
Figura 73. ***Constraints* de la Zybo**

```
## Pmod Header JC
set_property -dict "PACKAGE_PIN V15 IOSTANDARD LVCMOS33" [get_ports "led_out[0]"]
set_property -dict "PACKAGE_PIN W15 IOSTANDARD LVCMOS33" [get_ports "led_out[1]"]
set_property -dict "PACKAGE_PIN T11 IOSTANDARD LVCMOS33" [get_ports "cmp1"]
set_property -dict "PACKAGE_PIN T10 IOSTANDARD LVCMOS33" [get_ports "cmp2"]
```

Fuente: elaboración propia, utilizando Vivado 2017.4.

Por último, se colocan dos teclas o botones en los pines físicos en la parte externa de la FPGA, se polariza el botón a 5 voltios y se coloca una resistencia a 0 voltios conectando a la resistencia la entrada de los pines de la Zybo, como se observa en la figura 74.

Figura 74. **Conexiones externas de la Zybo**



Fuente: elaboración propia, utilizando Vivado 2017.4.

Cuando se presiona el botón ubicado en el pin T10 se activa la transmisión de los caracteres ASCII los cuales aparecerán en la consola de la figura 49. Se mostrará una secuencia de caracteres 'abababababa' en consola y no se detendrá hasta que se presione el botón conectado al pin T11.

4.3.4. Práctica 10: comunicación FPGA – CPU

La práctica consiste en realizar un cronometro digital mostrado desde la consola desde el sistema operativo. Se requieren dos botones ubicados en dos pines de la Zybo que simularán el inicio y el final del cronometro. Debe iniciar su conteo con un flanco de subida, detectado en un determinado pin de la Zybo y se deberá detener cuando se detecte otro flanco de subida en otro pin distinto. El cronometro debe ir programado en la FPGA y debe transmitir por medio del Xillybus los códigos ASCII que representan los números del 1 al 10.

CONCLUSIONES

1. Se utilizó el entorno de desarrollo Vivado Design Suite para el desarrollo de la sintaxis y la lógica de aplicaciones orientado a la gama alta de los FPGA utilizando la tarjeta de desarrollo Zybo Zynq 7000.

2. Se comprendieron las diferencias entre el paralelismo de la asignación concurrente y la ejecución ordenada de la asignación secuencial, utilizando VHDL dentro del entorno de desarrollo VIVADO.

3. Se comprendió el funcionamiento y declaración de una máquina de estados finitos utilizando VHDL dentro del entorno de desarrollo VIVADO.

4. Se comprendió el funcionamiento y la declaración de vectores, matrices y memorias, utilizando VHDL dentro del entorno de desarrollo VIVADO.

5. Los sistemas que combinan el paralelismo de los FPGA con un sistema operativo son mucho más complejos y costosos; sin embargo, son sistemas altamente eficientes capaces de resolver cualquier necesidad.

6. Se desarrollaron dos ejemplos dedicados a la comunicación bidireccional entre el sistema operativo Xillinux y el FPGA, los cuales conectan la consola del sistema operativo con los pines físicos de la FPGA.

7. Se realizaron enunciados de diez prácticas enfocadas al Laboratorio de Electrónica 6 de la Universidad de San Carlos de Guatemala y al estudiante en general.

RECOMENDACIONES

1. Para más información acerca del tema se recomienda visitar la documentación proporcionada por Xillinx con base en el desarrollo de sistemas sobre FPGA.

2. Visitar la página del Laboratorio de Electrónica de la Universidad de San Carlos de Guatemala, Sección Electrónica 6 donde se encuentra publicado todo el material didáctico; además, se encuentra publicado el código fuente de los ejemplos utilizados para el desarrollo de este trabajo de graduación.

BIBLIOGRAFÍA

1. Digilent. *Zybo Zynq 7000*. [en línea]. <https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/>. [Consulta: 24 de enero de 2019.]

2. Laboratorio de Electronica 6. *Presentacion FPGA.* [en línea]. <http://labelectronica.weebly.com/electronica6.html>. [Consulta: 20 de enero de 2019.]

3. MORALES, Iván. *ICTP Xillinux tutorial. Giving your first steps on Xillinux with Zedboard.* [en línea]. <https://docs.google.com/document/d/1rXdNUEsIm9EYX3NmkObekfFvgKon2pgHPawZhVkG-EI/edit>. [Consulta: 18 de enero de 2019.]

4. _____. *Universidad San Carlos de Guatemala.* [en línea]. <http://labelectronica.weebly.com/electronica6.html>. [Consulta: 24 de enero de 2019.]

5. Sistema Operativo Xillinux y Xillybus. *Xillinx*. [en línea]. <http://xillybus.com/xillinux>. [Consulta: 24 de enero de 2019.]

6. Stemmer Imaging. *FPGA. Stemmer Imaging.* [en línea]. <https://www.stemmer-imaging.com/en-gb/technical-tips/introduction-to-fpga-acceleration/>.[Consulta: 20 de enero de 2019.]

7. Vivado Design Suite - HLx Editions. [en línea]. <https://www.Xillinx.com/products/design-tools/Vivado.html>. [Consulta: 24 de enero de 2019.]

8. Xillinx. *Getting started with Xillinux for Zynq-7000.* [en línea]. <http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf>. [Consulta: 25 de enero de 2019.]