COMUNICACIÓN SPI ENTRE FPGA ZEDBOARD Y SENSORES INERCIALES PMODACL

Oscar Alonso Quintero Grisales

Ingeniería Electrónica

Director Luis Fernando Castaño

INSTITUTO TECNOLÓGICO METROPOLITANO

Agosto 27 de 2018

RESUMEN

Actualmente, con el constante uso de la tecnología y la proliferación de la información, surge la necesidad de implementar nuevas alternativas a los procesadores convencionales, que permitan mayor velocidad en el procesamiento de la información. Esta solución puede ser encontrada en los FPGAs. El presente proyecto tiene como objetivo realizar la adquisición simultánea de dos sensores inerciales PmodACL utilizando un sistema basado en FPGA.

En este trabajo, se presenta información acerca de los dispositivos empleados y se describen los pasos a seguir para implementar la comunicación SPI entre los sensores y el FPGA de una ZedBoard. Una vez realizado este producto de laboratorio, se pudo concluir que es posible establecer una comunicación SPI entre una FPGA Zedboard y dos sensores inerciales PmodACL, para ser leídos en simultáneo. Para lograr esta lectura, se tuvieron que realizar modificaciones tanto en el diseño del hardware en el software Vivado, como en el diseño del código en el software SDK del fabricante.

Palabras clave**:** Xilinx, Digilent, FPGA, Zedboard, VHDL, comunicación SPI, sensores inerciales, PmodACL, Vivado, SDK.

RECONOCIMIENTOS

Agradezco a Dios por permitirme superar a mí mismo, por ayudarme a alcanzar mis metas y emprender un nuevo camino como profesional. Agradezco enteramente a mi novia Erika Varela, por su ayuda y dedicación durante todo el proyecto, su respaldo y disciplina me sirvieron de aliento para vencer mis propias barreras. También, agradezco a mi familia por su apoyo y voz de aliento para lograr mis metas.

Finalmente, agradezco al asesor de este trabajo de grado por su ayuda y gestión frente a los equipos electrónicos e información que necesité durante el proyecto. Mis agradecimientos al personal docente y compañeros del ITM, que me acompañaron durante mi formación profesional y personal.

ACRÓNIMOS

Zedboard Zynq Evaluation Board

FPGA Field Programmable Gate Array

SDK Software Development Kit

Pmod Peripheral Module

PmodACL Peripheral Module Accelerometer

SPI Serial Peripheral Interface

MOSI Master Output Slave Input

MISO Master Input Slave Output

SCLK Serial Clock

SS Slave Select

CPOL Clock Polarity

CPHA Clock Phase

USB Universal Serial Bus

GPIO General Purpose Input/Output

PS Processing System

PL Programmable Logic

MCU Microcontroller Unit

EPP Enhanced Parallel Port

AXI Advanced extensible Interface

CPU Central Processing Unit

CPLD Complex Programmable Logic Device

VHDL Combinación de los acrónimos VHSIC y H0DL

VHSIC Very High Speed Integrated Circuit

HDL Hardware Description Language

ASIC Application Specific Integrated Circuit

DSP Digital Signal Processor

I2C Inter-Integrated Circuit

SoC System on Chip

UART Universal Asynchronous Receiver-Transmitter

JTAG Joint Test Action Group

CLB Configurable Logic Block

IOB Input/Output Block

VADJ Voltage Adjustable

HW Hardware

IDE Integrated Development Environment

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Actualmente, con el constante uso de la tecnología y la proliferación de la información, surge la necesidad de implementar nuevas alternativas a los procesadores convencionales, que permitan mayor velocidad en el procesamiento de la información. Esta solución puede ser encontrada en los FPGAs, dispositivos que además de mayor velocidad, ofrecen una importante versatilidad para el diseño del hardware y posibilitan el ahorro de tiempo en la ejecución de un código, gracias a su ejecución concurrente.

Aprovechando esta característica, el presente trabajo tiene como finalidad establecer una comunicación SPI entre una FPGA Zedboard y dos sensores inerciales PmodACL. Para lograr esto, se realizó la recolección de la información proporcionada por los fabricantes de la FPGA y los sensores inerciales. Además, se realizó el acondicionamiento del hardware, el software y la modificación del diseño para la lectura simultánea de dos sensores inerciales.

Este proyecto está compuesto por marco teórico, metodología, resultados, discusión, conclusiones, recomendaciones, referencias bibliográficas y apéndices. En el primer apartado, se brindará el conocimiento teórico necesario para el desarrollo de este proyecto, en el cual se incluyen las características del software y hardware a utilizar. En la metodología, se describirán los pasos a seguir para la implementación de este producto de laboratorio. En la tercera sección, se presentan los resultados obtenidos en este trabajo. Por último, se presentan los apéndices, el cual contiene toda la información complementaria como los códigos utilizados para llevar a cabo este proyecto, además, de

los resultados obtenidos al implementar la comunicación SPI con dos sensores en simultáneo.

2. MARCO TEÓRICO

2.1. FPGA

“Ross Freman, co-fundador de Xilinx, inventó el arreglo matricial de compuertas. La raíz histórica de las FPGA son los dispositivos de lógica programable compleja (CPLD) de mediados de los 1980. CPLD y FPGA incluyen un relativo gran número de elementos lógicos programables” (Marín, s.f., p. 23).

Un FPGA, del inglés “Field programmable gate array” conocido como matriz de puertas reprogramable, según Maiocchi (2013), es un dispositivo semiconductor que contiene bloques lógicos programables que se interconectan al configurarse por medio de un lenguaje de descripción especializado, tal como el VHDL y Verilog. Esta lógica programable permite llevar a cabo funciones sencillas como:

> Duplicar la funcionalidad de puertas lógicas básicas como AND, OR, XOR, NOT o funciones combinacionales más complejas tales como decodificadores o simples funciones matemáticas. En muchos FPGA, estos componentes lógicos programables… incluyen elementos de memoria, los cuales pueden ser simples flip-flops o bloques de memoria más complejos (Marín, s.f., p. 21).

Este mismo autor enumera una serie de ventajas y desventajas de este dispositivo programable. Entre las primeras, menciona un menor tiempo de producción, reprogramación del dispositivo para corrección de errores y reducción de costos, utilizando la FPGA como un prototipo de prueba para un producto final. Respecto a las desventajas, señala que esta es más lenta en comparación a los circuitos integrados de aplicaciones específicas (ASIC), además de su dificultad de soporte a los diseños más complejos y un mayor consumo de energía.

Adicionalmente, Maiocchi (2013), señala que al estar ligado el lenguaje utilizado en la FPGA, a la programación del hardware, se puede encontrar con una realización en más alto nivel en comparación con sistemas mucho más simples como un procesador, un microprocesador, un microcontrolador, un DSP, entre otros; por lo cual, es de gran utilidad para prestaciones críticas que normalmente no son desarrolladas por otros dispositivos.

Una FPGA estructuralmente se compone por compuertas lógicas, biestables, puertos de entrada y salida, entre otros; que en conjunto conforman una plantilla en blanco sin conexiones establecidas, para lo cual se requiere cargar un archivo de programación llamado bitstream, el cual es generado en un programa ofrecido por el fabricante del hardware, que permite establecer la ruta interna de los datos y recursos a utilizar.

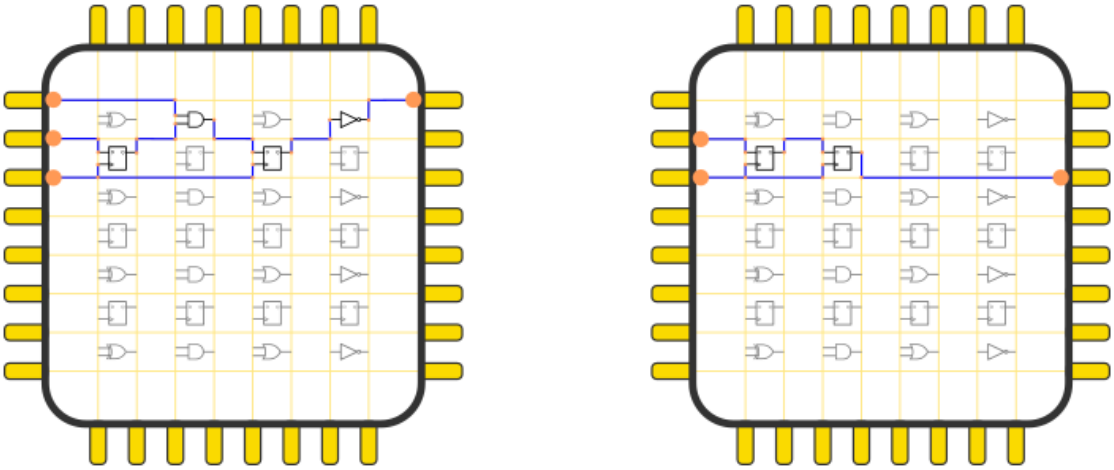


Figura 1. Estructura interna de una FPGA (tomado de Crespo, 2017).

A diferencia de otros dispositivos programables que ejecutan su programación de forma secuencial, la FPGA tiene la capacidad de procesar múltiples señales en un mismo ciclo de reloj, conocido como concurrencia; lo que permite ahorrar tiempo en ejecución de múltiples tareas en simultáneo, pero que a su vez estará limitada por el espacio en memoria.

Por otro lado, es importante resaltar que para la programación de la FPGA es necesario tener claro el circuito electrónico a implementar en el proyecto, estableciendo una

arquitectura que será descrita mediante el lenguaje de programación Verilog o VHDL en el mayor de los casos. Luego se da paso a una de las etapas fundamentales del proceso llamada síntesis, en la cual se escogen los elementos a utilizar y la interconexión entre ellos, para crear así el archivo bitstream que se encargará de emplazar y enrutar la FPGA.

A continuación, es ilustrado cómo es el proceso para la programación de una FPGA:

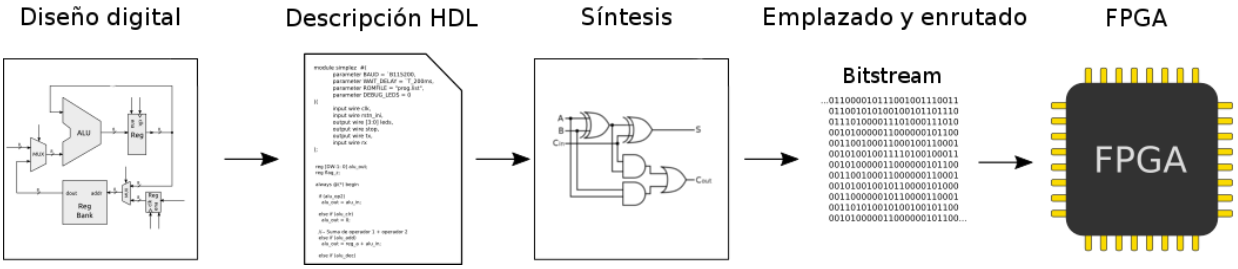


Figura 2. Configuración de una FPGA (tomado de Crespo, 2017).

2.1.1. Aplicaciones

Las aplicaciones de las FPGA abarcan todo lo relacionado con procesamiento de señales externas e internas que permite conexiones con otros dispositivos a través de sus múltiples protocolos de comunicación como el SPI, I2C, entre otros. Según Marín, (s.f.) entre las aplicaciones más comunes para las FPGA se encuentran:

> Los DSP (Digital Signal Processor), sistemas aeroespaciales y de defensa, prototipos de los ASIC, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, y tienen un crecimiento de aplicaciones en otras áreas. Las FPGA encuentran aplicaciones en muchas áreas donde se requiera del paralelismo ofrecido por su arquitectura (p. 24).

Por otro lado, el fabricante Digilent (2018)destaca como principales aplicaciones para la FPGA Zedboard lo siguiente: procesamiento de video, control de motor, sistemas

embebidos como Linux, Android y Windows, software de aceleración, creación de prototipos generales basados en el Zynq-7000 AP SoC.

2.1.2. Fabricantes de algunas FPGAs

En la actualidad existen múltiples productores de FPGA como Lattice, Actel, Quicklogic, Atmel, Achronix y Mathstar. Sin embargo, Xilinx y Altera son los principales líderes en el mercado. Para desarrollar este producto de laboratorio se utilizó la FPGA Zedboard de Digilent Inc., la cual es distribuida por Avnet, distribuidor mundial de componentes electrónicos; y cuyo procesador es Xilinx Zynq-7000 AP SoC XC7Z020-CLG484.

Digilent Inc. es una compañía de National Instruments, la cual es un gran líder en el campo de diseño, fabricación y distribución de herramientas de diseño electrónico alrededor del mundo. Además de sus productos, Digilent ofrece servicios de fabricación y diseño de OEM para las empresas de tecnología líderes, entre ellas Xilinx, Analog Devices, Cypress Semiconductor y National Instruments (Digilent, 2018).

Xilinx Inc. es una compañía de tecnología que distribuye dispositivos lógicos programables, inventor del FPGA y proveedor del software computacional utilizado para la programación de la Zedboard, los cuales son Xilinx ISE y Vivado Design Suite. Asimismo, proporciona el procesador de la familia Zynq - 7000 para la FPGA Zedboard.

2.2. ZEDBOARD

La Zedboard es un dispositivo programable basado en un system on chip (SoC) el cual se caracteriza por tener “un único circuito integrado de silicio que implementa la funcionalidad de un sistema completo, en lugar de necesitarse varios chips físicos diferentes interconectados entre sí” (Álvarez, 2016, p.7).

Los dispositivos Zynq-7000 All Programmable SoC de Xilinx comprenden dos partes principales integradas en un único chip: un Sistema de Procesado (PS) basado en un microprocesador Dual-Core ARM Cortex A9 y Lógica Programable (PL) basada en la arquitectura de las FPGA 7-series de Xilinx. La comunicación entre el PS y la PL se realiza mediante interfaces AXI (Advanced eXtensible Interface) estándar, que proporcionan conexiones de alta velocidad de transmisión y baja latencia (Crocket, Elliot, Enderwitz & Stewart, citados en Álvarez, 2016).

2.2.1. Sistema de procesado (PS)

En la parte de PS se encuentran los sistemas operativos completos y rutinas de software. La base del sistema de procesado de todos los dispositivos Zynq es un microprocesador Dual-Core ARM Cortex A9 que puede operar a frecuencia de reloj de hasta 1 GHz, dependiendo del modelo Zynq concreto. Es un procesador “hard", es decir, existe dentro del SoC como un elemento de silicio dedicado y optimizado (en contraposición a los procesadores “soft" que se implementan utilizando la lógica programable de la FPGA, como el Xilinx MicroBlaze). El sistema de procesado incluye interfaces para periféricos de entrada/salida (SPI, I2C, USB, UART, GigE, entre otros), memoria cache, interfaces de memoria, hardware de interconexión y circuitería de generación de reloj (Álvarez, 2016).

2.2.2. Lógica programable (PL)

En la parte de PL está toda la carga lógica de alta velocidad desarrollada en la programación, la cual permite la aceleración de tareas con una alta carga computacional, como lo son sistemas aritméticos, procesado de video en tiempo real, entre otros. La lógica programable de Zynq está basada en la arquitectura de las FPGA Artix-7 y Kintex-7 de Xilinx. Está predominantemente compuesta de CLBs (Configurable Logic Blocks). Para proporcionar una interfaz entre la lógica y los pines físicos del dispositivo, existen IOBs (Input/Output Blocks) alrededor del perímetro de la PL. La PL recibe cuatro entradas separadas de reloj desde el

PS y, adicionalmente, tiene la posibilidad de generar y distribuir sus propias señales de reloj independientemente del PS (Álvarez, 2016).

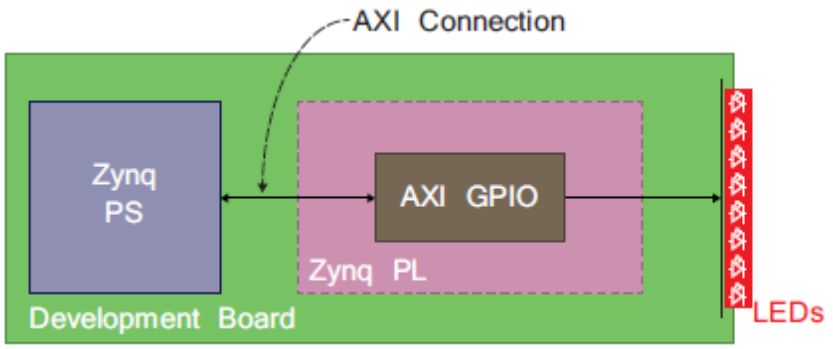


Figura 3. Diseño de un sistema Zynq (Crockett, Elliot, Enderwitz y Northcote, 2015).

2.2.3. Características

Según el manual de usuario de la Zedboard proporcionado por Avnet, en el año 2014, estas son sus principales características:

- Xilinx® XC7Z020-1CSG484CES EPP
  - o Primary configuration = QSPI Flash
  - o Auxiliary configuration options
    - Cascaded JTAG
    - SD Card
- Memory
  - o 512 MB DDR3 (128M x 32)
  - o 256 Mb QSPI Flash
- Interfaces
  - o USB-JTAG Programming using Digilent SMT1-equivalent circuit
    - Accesses PL JTAG
    - PS JTAG pins connected through PS Pmod
  - o 10/100/1G Ethernet
  - o USB OTG 2.0
  - o SD Card
  - o USB 2.0 FS USB-UART bridge
  - o Five Digilent Pmod™ compatible headers (2x6) (1 PS, 4 PL)
  - o One LPC FMC
  - o One AMS Header
  - o Two Reset Buttons (1 PS, 1 PL)

- Seven Push Buttons (2 PS, 5 PL)
- Eight dip/slide switches (PL)
- Nine User LEDs (1 PS, 8 PL)
- DONE LED (PL)

- On-board Oscillators
  - 33.333 MHz (PS)
  - 100 MHz (PL)
- Display/Audio
  - HDMI Output
  - VGA (12-bit Color)
  - 128x32 OLED Display
  - Audio Line-in, Line-out, headphone, microphone
- Power
  - On/Off Switch
  - 12V @ 5A AC/DC regulator
- Software
  - ISE® WebPacK Design Software
  - License voucher for ChipScope™ Pro locked to XC7Z020

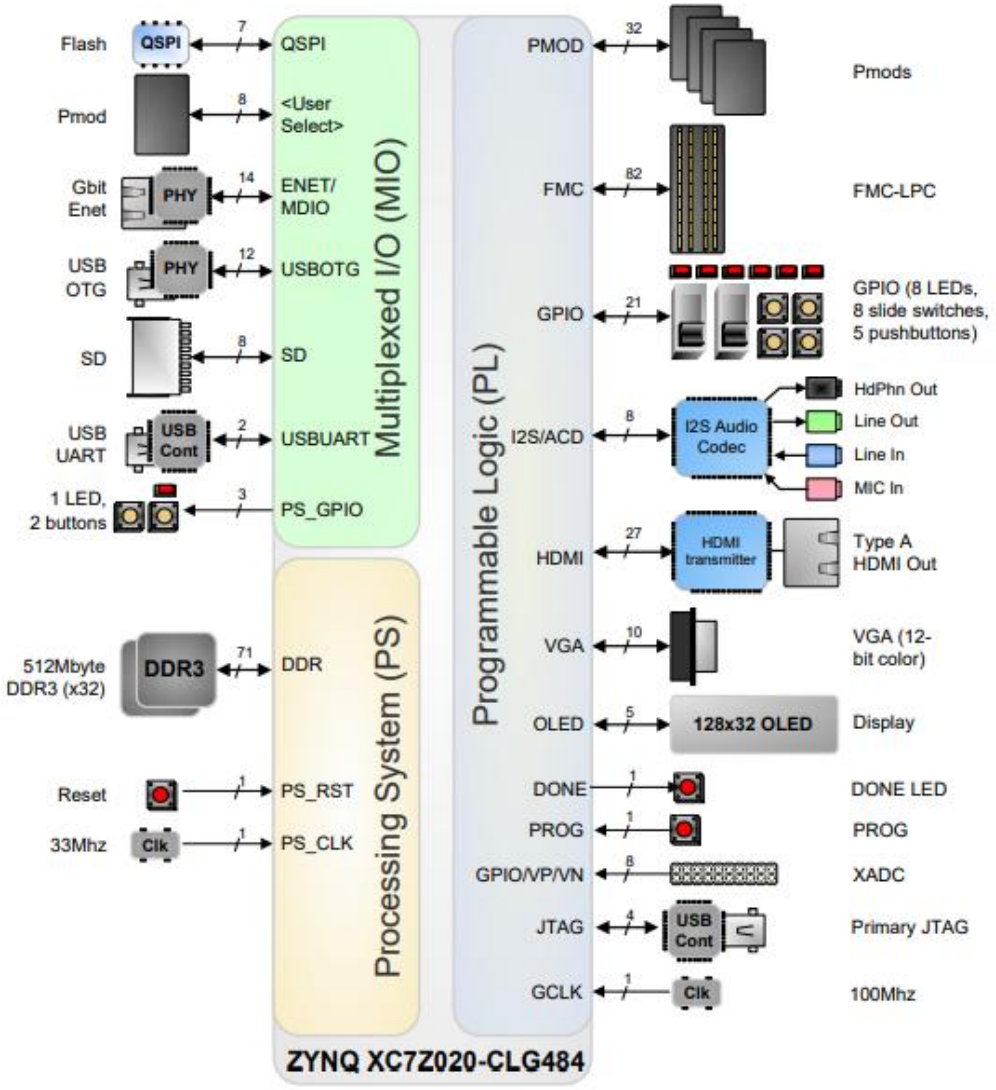


Figura 4. Diagrama de bloques (Tomado de Avnet, 2014).

2.2.4. Asignación banco de pines de Zynq

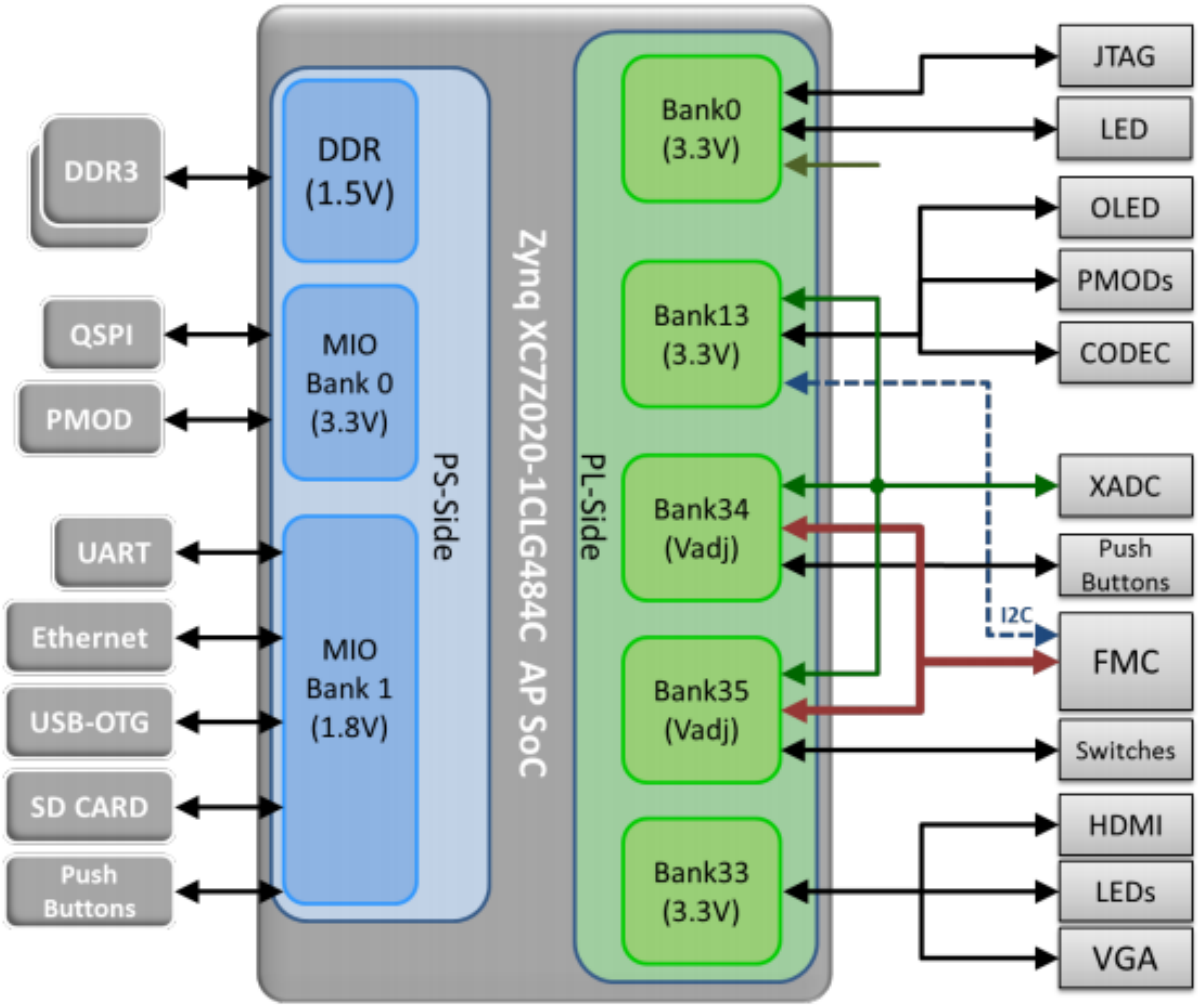


Figura 5. Asignación banco de pines de Zynq (Tomado de Avnet, 2014).

2.2.4.1. USB-JTAG

La Zedboard provee una funcionalidad JTAG, basada en el módulo JTAG USB de alta velocidad de Digilent. Esta circuitería USB JTAG está totalmente soportada e integrada dentro de las herramientas Xilins ISE, incluyendo iMPACT, ChipScope y SDK Debugger. Este último, es el utilizado para programar la Zedboard en este producto de laboratorio.

2.2.4.2. Conectores compatibles con digilent Pmod

La Zedboard tiene 5 conectores hembra (2x6) compatibles con Digilent Pmod. Cada conector Pmod tiene 12 puertos de conexión, los cuales son distribuidos de la siguiente manera: 8 entrada/salida de señales, 2 de voltaje 3.3V y 2 señales de tierra.

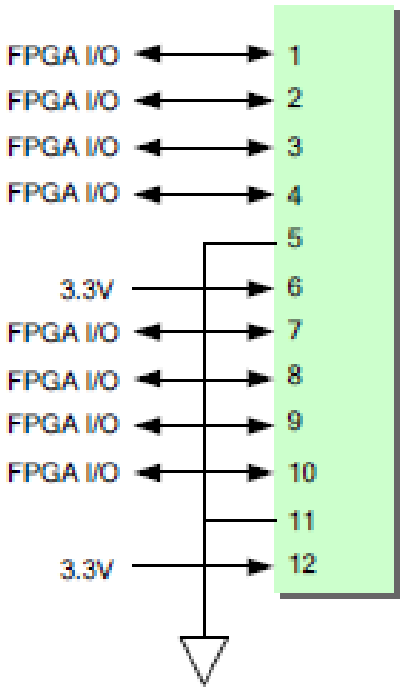


Figura 6. Conector Pmod (Tomado de Avnet, 2014).

4 conectores Pmod están ubicados en la interfaz del PL de la EPP, los cuales irán conectados al banco 13 (3.3V) de la EPP. El Pmod restante (JE1), se conecta en el lado PS de la EPP, en los pines MIO (7, 9-15) del banco MIO de 0/500 (3.3V) de la EPP. Entre los usos principales para los conectores Pmod, se encuentran 9 tipos de periféricos MIO (SPI, GPIO, CAN, I2C, UART, SD, QSPI, Trace, Watchdog). A continuación, se muestra cada uno de los cinco Pmod con su respectiva distribución y nombramiento de pines:

Pmod	Signal Name	Zynq EPP pin	Pmod	Signal Name	Zynq EPP pin
JA1	JA1	Y11	JB1	JB1	W12
	JA2	AA11		JB2	W11
	JA3	Y19		JB3	V10
	JA4	AA9		JB4	W8
	JA7	AB11		JB7	V12
	JA8	AB10		JB8	W10
	JA9	AB9		JB9	V9
	JA10	AA8		JB10	V8

Pmod	Signal Name	Zynq EPP pin	Pmod	Signal Name	Zynq EPP pin
JC1 Differential	JC1_N	AB6	JD1 Differential	JD1_N	W7
	JC1_P	AB7		JD1_P	V7
	JC2_N	AA4		JD2_N	V4
	JC2_P	Y4		JD2_P	V5
	JC3_N	T6		JD3_N	W5
	JC3_P	R6		JD3_P	W6
	JC4_N	U4		JD4_N	U5
	JC4_P	T4		JD4_P	U6

Pmod	Signal Name	Zynq EPP pin
JE1 MIO Pmod	JE1	A6
	JE2	G7
	JE3	B4
	JE4	C5
	JE7	G6
	JE8	C4
	JE9	B6
	JE10	E6

Figura 7. Conexiones Pmod (Tomado de Avnet, 2014).

El voltaje para GPIO es ajustado mediante jumpers en Vadj, que es un voltaje ajustable mediante el jumper J18, el cual permite escoger entre 1.8V, 2.5V y 3.3V según sea necesario. En la figura a continuación se muestran los jumpers utilizados en este producto:

Ref Designator	Description	Default Setting	Function
JP7	Boot_Mode[3]/MIO[2]	GND – Cascaded JTAG	JTAG Mode. GND cascades PS and PL JTAG chains. VCC makes JTAG chains independent.
JP8 JP9 JP10	Boot_Mode[0]/MIO[3] Boot_Mode[1]/MIO[4] Boot_Mode[2]/MIO[5]	110 – SD Card	Boot Device Select See Zynq Configuration Modes
JP11	Boot_Mode[4]/MIO[6]	GND – PLL Used	PLL Select. GND uses PS PLLs. VCC bypasses internal PS PLLs
JP12	XADC Ferrite Bead Disable	Open	Short bypasses XADC-GND ferrite bead connection to board GND.
JP13	JTAG PS-RST	Open	Short connects JTAG PROG-RST to PS Reset.
J18	Vadj Select	1.8V	Selects Vadj (1.8V, 2.5V, or 3.3V)

Figura 8. Configuración de jumpers (Tomado de Avnet, 2014).

En la siguiente figura se muestran los diferentes jumpers que posee la Zedboard:

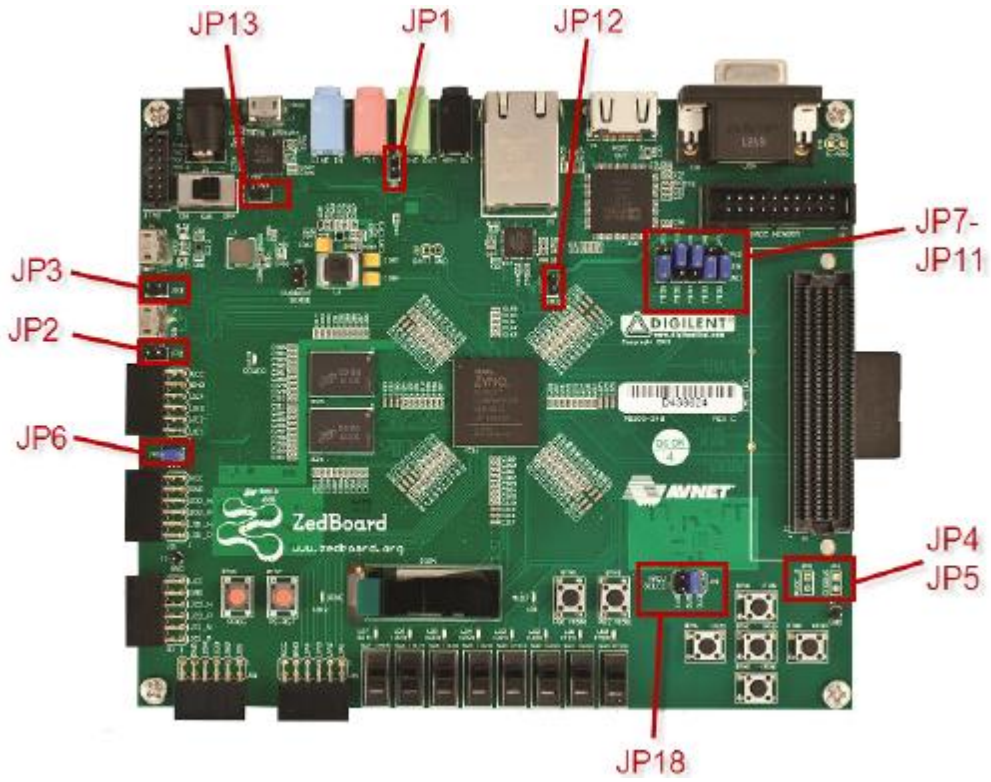


Figura 9. Jumpers Zedboard (Tomado de Avnet, 2014).

2.3. LENGUAJE VHDL

VHDL es un lenguaje de descripción de sistemas electrónicos digitales, más no es un lenguaje de programación en sí. Este lenguaje surge del programa VHSIC (Very High Speed Integrated Circuits) impulsado por el Departamento de Defensa del gobierno de los Estados Unidos de América, ante la necesidad de un lenguaje que describiera circuitos electrónicos, de ahí el significado de sus siglas VHDL (VHSIC Hardware Design Language).

Mediante el lenguaje VHDL se cubren varias necesidades que surgen durante el proceso de diseño, permitiendo realizar una descripción funcional o comportamental del circuito, así como la declaración de sus entidades y subentidades para establecer la jerarquía e interconexiones entre ellas. También, permite simular y sintetizar el diseño para su posterior manufacturado y encapsulado como un microcircuito, eliminando así la fase de prototipado y ahorrando costos en diseño y producción de los componentes.

Al utilizar un lenguaje de descripción de hardware debemos pensar en compuertas y biestables, no como un lenguaje de programación centrado en variables, funciones, ciclos combinacionales y relojes condicionados. En particular, VHDL permite descubrir errores en el diseño antes de su implementación física, además de describir la estructura o el comportamiento lógico del circuito a partir de subcircuitos más sencillos (Sánchez- Élez, 2014).

Es importante resaltar que en HW todos los circuitos trabajan a la vez para obtener un resultado, ejecutando todo en paralelo; esto es conocido como concurrencia y es una de las principales ventajas a la hora de trabajar con FPGA. Por otro lado, en software el orden de las instrucciones delimita la actualización de las variables, basándose en instrucciones secuenciales para llegar al resultado (Sánchez- Élez, 2014).

Un sistema digital está compuesto por entradas y salidas (aspecto exterior) y la interconexión existente entre ellas (estructura). El aspecto exterior hace referencia a el

número de puertos de entrada y salida, lo cual es denominado entity. La estructura o architecture hace referencia a la descripción comportamental del circuito que debe estar asociada a una entity. No se puede olvidar incluir al comienzo, las librerías y paquetes a utilizar en el proyecto (Sánchez- Élez, 2014).

Tras describir el circuito mediante VHDL, se puede realizar una simulación del mismo donde se comprueba su funcionalidad antes de continuar con su impresión. Posteriormente se recomienda realizar la síntesis del diseño donde se crea las uniones entre componentes y se crea como tal el circuito.

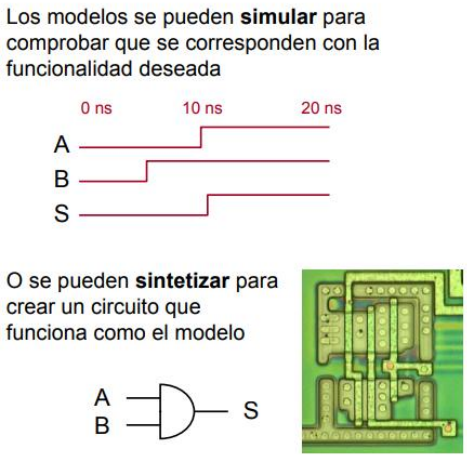


Figura 10. Simulación y síntesis (Tomado de López-Buedo, 2007).

2.3.1. Ejemplo VHDL

Para emprender la descripción de un circuito en VHDL se comienza con la definición de entidad (*Entity*), donde se definen sus entradas y salidas.

```
ENTITY Mux IS

      PORT( a:     IN bit;
            b:     IN bit;
            sel:   IN bit;

            c:     OUT bit
      );

END Mux;
```

Figura 11. Definición entidad (Tomado de Schweers, 2002).

Como se puede observar en la imagen anterior, la entidad definida con el nombre *Mux* tiene tres entradas y una salida de tipo bit, el cual está predefinido en el lenguaje y abarca los valores '0' y '1'. La entidad tiene asociada 3 arquitecturas, cada una con un estilo de descripción y un nivel de abstracción diferente: descripción algorítmica (mayor nivel de abstracción), descripción de flujo de datos (nivel medio de abstracción) y descripción estructural (menor nivel de abstracción). Todas las anteriores, se definen en VHDL mediante la palabra *Architecture*.

Una descripción VHDL puede ser mixta, es decir, puede comprender varios estilos. Por ejemplo, en una descripción estructural pueden definirse en forma de flujo de datos los elementos más bajos de la jerarquía de diseño, empleando operadores lógicos en la descripción.

2.3.1.1. Descripción algorítmica

En la descripción algorítmica se hace referencia al funcionamiento del multiplexor, sin tener en cuenta los operadores lógicos presentes en su esquema. Esta descripción, a diferencia de la descripción concurrente, se ejecuta de manera secuencial a través de la palabra clave *Process.* También, las funciones y los procedimientos son construcciones secuenciales que permiten realizar descripciones funcionales.

```
ARCHITECTURE Algoritmica OF Mux IS
BEGIN
      PROCESS(a, b, sel)
      BEGIN
            IF (sel='1') THEN
                  c<=b;
            ELSE
                  c<=a;
            END IF;
      END PROCESS;

END Algoritmica;
```

Figura 12. Descripción algorítmica (Tomado de Schweers, 2002).

Una descripción de mayor abstracción se puede realizar por medio de la palabra clave *Procedure*, de la siguiente forma:

```
PROCEDURE Mux(a,b,sel:IN bit;c:OUT bit) IS
BEGIN
      IF (sel='1') THEN
            c:=b;
      ELSE
            c:=a;
      END IF;

END Mux;
```

Figura 13. Descripción procedure (Tomado de Schweers, 2002).

2.3.1.2. Descripción flujo de datos

Esta descripción permite la paralelización de instrucciones y es a su vez la más parecida a una descripción estructural, donde las sentencias que se encuentran entre el Begin y el End del cuerpo de la arquitectura, se ejecutan de forma paralela en el tiempo que dure la simulación. Este tipo de descripción se presenta en la siguiente imagen:

```
ARCHITECTURE FlujoDeDatos OF Mux IS
BEGIN
      c<=(a and (not sel)) or (b and sel);

END FlujoDeDatos;
```

Figura 14. Descripción flujo de datos (Tomado de Schweers, 2002).

2.3.1.3. Descripción estructural

Este tipo de descripción es la más cercana a la implementación física de un circuito. En ella, se hace referencia, de una forma simple y fácil de interpretar, a los componentes que intervienen y a sus interconexiones. En la arquitectura son declarados los componentes que se utilizan e instanciados en el cuerpo de la misma las veces que se requiera.

```
ARCHITECTURE Estructural OF Mux IS

      COMPONENT not IS

            PORT( a:    IN bit;
                  b:    OUT bit
                  );

      END COMPONENT;

      COMPONENT and IS

            PORT( a:    IN bit;
                  b:    IN bit;
                  c:    OUT bit
                  );

      END COMPONENT;

      COMPONENT or IS

            PORT( a:    IN bit;
                  b:    IN bit;
                  c:    OUT bit
                  );

      END COMPONENT;

SIGNAL x,y,z:bit;

BEGIN
      Not1: not PORT MAP(sel,x);
      And1: and PORT MAP(b,sel,y);
      And2: and PORT MAP(a,x,z);
      Or1: or PORT MAP(y,z,c);

END Estructural;
```

Figura 15. Descripción estructural (Tomado de Schweers, 2002).

2.4. SOFTWARE

Xilinx Inc, es una compañía de tecnología americana reconocida por inventar los FPGA y por ser, principalmente, un distribuidor de dispositivos lógicos programables. Xilinx posee las mayores familias de productos de FPGAs, incluyendo las series Virtex (alto rendimiento),

Kintex (rango medio), y Artix (bajo costo), además del software computacional Xilinx ISE y Vivado Design Suite.

2.4.1. Vivado

Vivado design suite es un entorno de desarrollo de software creado por Xilinx para el diseño, descripción e implementación de circuitos electrónicos en diversas plataformas. Este software se caracteriza por ser un entorno de desarrollo integrado (IDE), con el cual se puede diseñar y sintetizar el hardware que será implementado en la lógica programable de las FPGAs. Además de trabajar con diseños electrónicos que emplean los lenguajes VHDL y Verilog, permite trabajar a más alto nivel formando e integrando bloques IP hardware, aumentando la capacidad de diseño basado en reutilización (Álvarez, 2016).

Tras la descripción de los circuitos electrónicos mediante este software, se puede proceder a la generación del bitstream, el cual se encargará de implementar la configuración de hardware en la FPGA. Una vez configurada la FPGA, se pueden habilitar las conexiones externas mediante los bloques periféricos de entrada y salida, tales como los módulos PMOD que permiten la comunicación con dispositivos externos mediante protocolos de comunicación como SPI e I2C previamente configurados.

Xilinx cuenta con múltiples familias de tarjetas, entre las cuales se encuentra la familia Zynq-7000 SoC, la cual integra la programabilidad del software (PS) de un procesador basado en ARM con la programabilidad del hardware (PL) de un FPGA, el objetivo es aprovechar la FPGA para integrar bloques IP hardware que cumplen con la finalidad esperada durante el diseño de los bloques IP en el software.

Dentro del portafolio de software de Xilinx, se encuentra la edición Vivado HL Webpack, la cual es una licencia limitada gratuita a por un año, ideal para estudiantes y principiantes en este entorno de descripción de hardware. Esta versión Vivado HL Webpack permite el acceso a algunas características y funcionalidades básicas de Vivado, además de admitir los

siguientes dispositivos: Zynq UltraScale + MPSoC (XCZU2CG / EG, XCZU3CG / EG), Zynq-7000 todos los SoC programables (XC7Z007S - XC7Z7030), Spartan-7 (XC7S50), Artix-7 (XC7A15T, XC7A35T , XC7A50T, XC7A75T, XC7A100T, XC7A200T), Kintex-7 (XC7K70T, XC7K160T), Kintex UltraScale (XCKU025 - XCKU035) y Kintex UltraScale + (XCKU3P, XCKU5P).

A continuación, se muestran las características principales de las versiones diferentes disponibles de Vivado:

Vivado Design Suite - HLx Edition Features	Vivado HL Design Edition	Vivado HL System Edition	Vivado Lab Edition	Vivado HL WebPACK (Device Limited)	Free 30-day Evaluation
Accelerating Implementation					
Synthesis and Place and Route	✓	✓		✓	✓
Partial Reconfiguration	✓	✓		Can be purchased as an option	✓
Accelerating Verification					
Vivado Simulator	✓	✓		✓	✓
Vivado Device Programmer	✓	✓	✓	✓	✓
Vivado Logic Analyzer	✓	✓	✓	✓	✓
Vivado Serial I/O Analyzer	✓	✓	✓	✓	✓
Debug IP (ILA/VIO/IBERT)	✓	✓		✓	✓
Accelerating High Level Design					
Vivado High-Level Synthesis	✓	✓		✓	✓
Vivado IP Integrator	✓	✓		✓	✓
System Generator for DSP		✓			✓

Figura 16. Versiones software Vivado (Tomado de Xilinx, 2018).

La familia Zynq-7000 se caracteriza por integrar un sistema completo basado en el procesador Cortex – A9 de ARM MPCore en una FPGA, muy similar a un procesador estándar desde el cual se cuenta con un arrancado de inmediato en el encendido y es capaz

de ejecutar una gran variedad de sistemas operativos con independencia de la lógica programable; gracias a ello permite crear aplicaciones de alto rendimiento tales como vídeo-vigilancia, sistemas inalámbricos, entre otras. La arquitectura que implementa Zynq es diferente a la usada en anteriores matrimonios de lógica programable y procesadores integrados, basándose en un modelo de procesador céntrico.

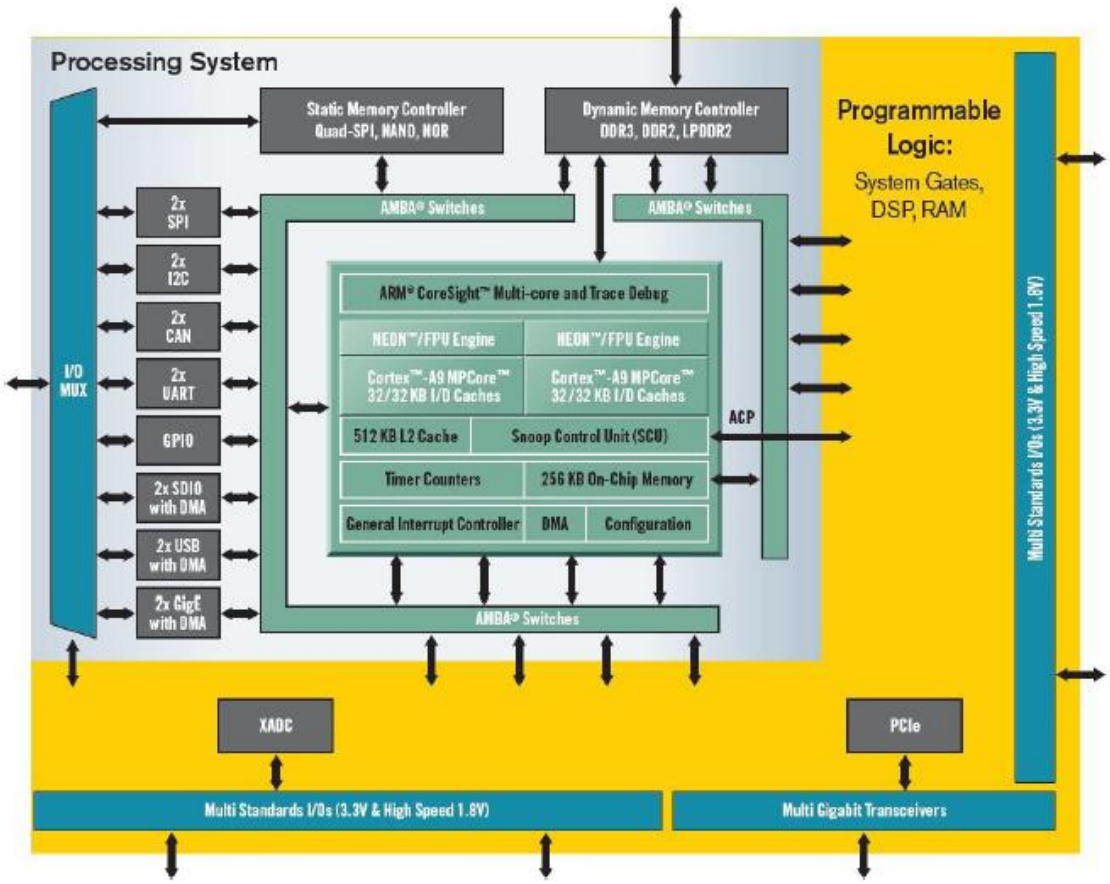


Figura 17. Zynq-7000, PS y PL (Tomado de Bustos, 2013).

2.4.2. SDK Xilinx

El software Xilinx SDK (Software Development Kit) es un ambiente de diseño integrado (IDE por sus siglas en inglés) para el diseño de software basado en el estándar de código abierto Eclipse, el cual proporciona un entorno para crear plataformas de software y aplicaciones destinadas a procesadores integrados Xilinx. Asimismo, permite desarrollar en C y C++ para

ARM, herramientas de debug y una serie de utilidades adicionales. Este software funciona con diseños de hardware creados en Vivado y según Xilinx, entre sus principales características se incluyen:

- Editor de código C / C ++, abundante en funciones y entorno de compilación
- Gestión de proyectos
- Soporte completo de diseño de software y flujos de depuración, incluidas capacidades de co-depuración multiprocesador y hardware / software
- Editor, compiladores, herramientas de compilación, administración de memoria flash e integración de depuración JTAG
- Conjunto completo de bibliotecas y controladores de dispositivos
- Control de versión del código fuente
- Programador de FPGA: se usa para programar FPGA de Xilinx FPGA con el bitstream

Un software SDK le permite al programador o desarrollador de software crear una aplicación informática para un sistema concreto, como lo son plataformas de hardware, sistemas operativos, entre otros. Frecuentemente se incluye documentación de soporte para profundizar en el material de referencia primario, tales como códigos de ejemplo y notas técnicas.

2.5 COMUNICACIÓN SPI

SPI (Serial Peripheral Interface) es un protocolo de comunicación serial de hardware / firmware desarrollado por Motorola, el cual permite la comunicación entre circuitos electrónicos inmersos en una misma tarjeta o entre tarjetas continuas. Esta interfaz de comunicación síncrona se realiza por medio de 4 hilos, en una configuración maestro/esclavo, donde se cuenta con dos tipos de protocolos: maestro único y maestro múltiple.

2.5.1. Estructura del SPI

El protocolo SPI permite una comunicación full dúplex entre el MCU y los dispositivos periféricos, donde sus partes principales son: estado, control y registro de datos, lógica de cambio, generador de tasa de baudios, la lógica de control maestro/esclavo y la lógica de control del puerto. El software puede sondear las banderas de estado SPI o la operación también puede ser impulsada por interrupciones. Un módulo SPI, según Naqvi (2015), tiene un total de 4 pines externos:

• Master Output Slave Input (MOSI): Datos de salida desde el maestro a las entradas de los esclavos.

• Master Input Slave Output (MISO): Datos de salida de un esclavo a la entrada del maestro.

• Serial Clock (SCLK): Reloj dirigido por el maestro a los esclavos, es utilizado para sincronizar los bits de datos.

• Slave Select (SS): Selecciona la señal dirigida por el maestro a los esclavos individuales, utilizados para elegir el esclavo de destino

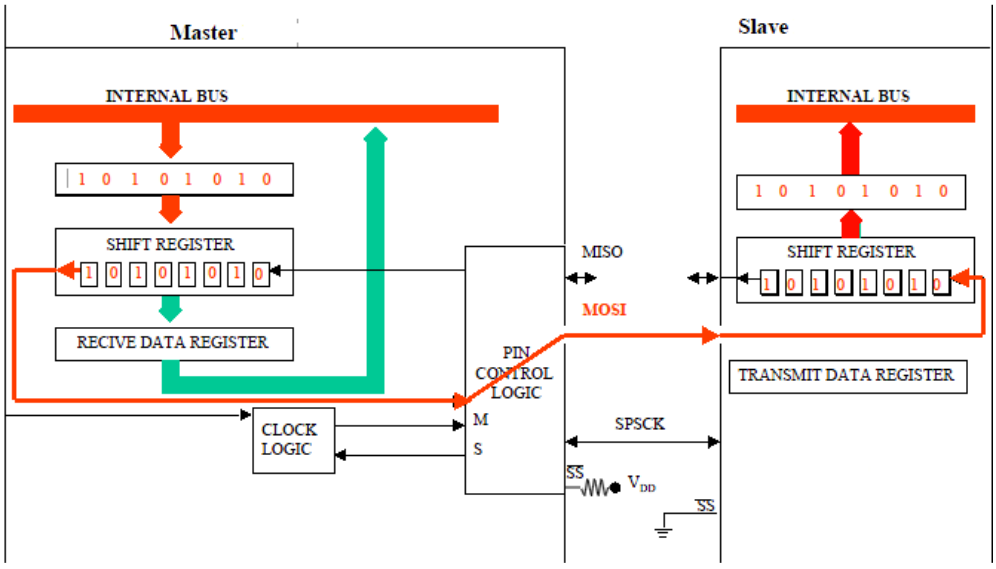


Figura 18. SPI maestro único (Tomado de Naqvi, 2015).

A continuación se muestra la configuración en modo maestro múltiple:

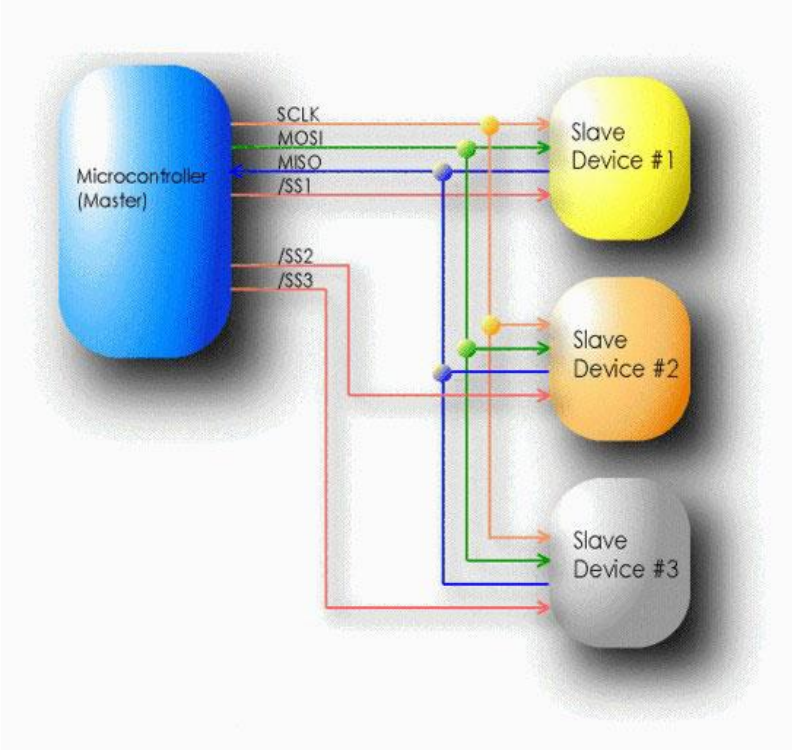


Figura 19. SPI maestro múltiple (Tomado de López, s.f.).

2.5.2. Transmisión de datos SPI

La interfaz SPI permite transmitir y recibir datos simultáneamente en dos líneas (MOSI y MISO). La polaridad del reloj (CPOL) y la fase del reloj (CPHA) son los parámetros principales que definen un formato de reloj para ser utilizado por el bus SPI. Dependiendo del parámetro CPOL, el reloj SPI puede invertirse o no invertirse. Cuando la polaridad del reloj se establece en 1, el estado inactivo para SCLK es alto. El parámetro CPHA se usa para cambiar la fase de muestreo. Si CPHA = 0, los datos se muestrean en el borde delantero (primero) del reloj. Si CPHA = 1, los datos se muestrean en el borde del reloj posterior (segundo), independientemente de si el borde del reloj está subiendo o bajando (Naqvi, 2015, p.2).

2.5.2.1. Polaridad del reloj=0, fase del reloj=0

Los datos deben estar disponibles antes de la primera señal de reloj subiendo. El estado inactivo del reloj es cero. Los datos en las líneas MISO y MOSI deben ser estables mientras el reloj está en alto y pueden cambiarse cuando el reloj está en bajo. Los datos son capturados en la transición de bajo a alto del reloj y propagados en la transición de alto a bajo del reloj (Naqvi, 2015).

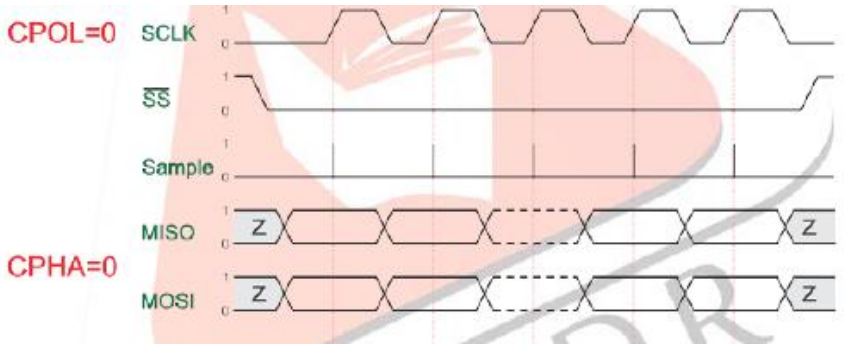


Figura 20. Polaridad del reloj=0, Fase del reloj=0 (Tomado de Naqvi, 2015).

2.5.2.2. Polaridad del reloj = 0, fase del reloj = 1

La primera señal de reloj subiendo puede ser usada para preparar los datos. El estado inactivo del reloj es cero. Los datos en las líneas MISO y MOSI deben ser estables mientras el reloj está en bajo y puede ser cambiado cuando el reloj está en alto. Los datos son capturados en la transición de alto a bajo del reloj y propagados en la transición de bajo a alto del reloj (Naqvi, 2015).

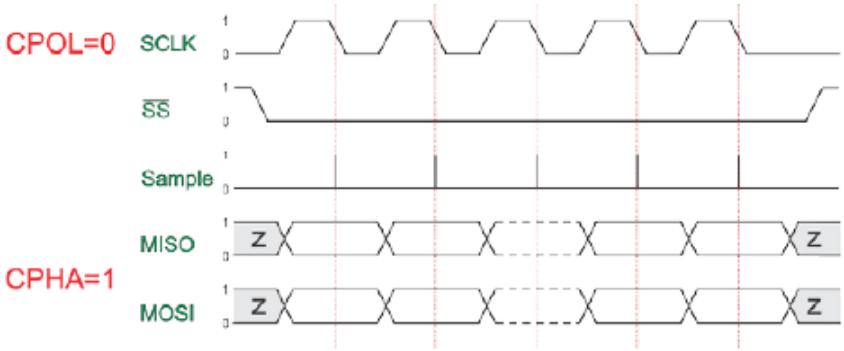


Figura 21. Polaridad del reloj=0, Fase del reloj=1 (Tomado de Naqvi, 2015).

2.5.2.3. Polaridad del reloj = 1, fase del reloj = 0

Los datos deben estar disponibles antes la primera señal de reloj bajando. El estado inactivo del reloj es uno. Los datos en las líneas MISO y MOSI deben ser estables mientras el reloj está en bajo y puede ser cambiado cuando el reloj está en alto. Los datos son capturados en la transición de alto a bajo del reloj y propagados en la transición de bajo a alto del reloj (Naqvi, 2015).

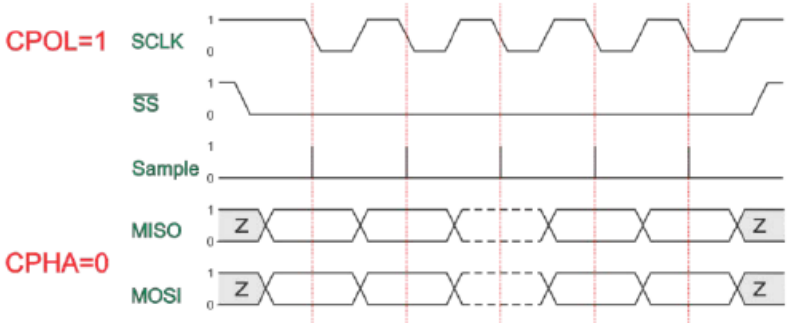


Figura 22. Polaridad del reloj=1, Fase del reloj=0 (Tomado de Naqvi, 2015).

2.5.2.4. Polaridad del reloj = 1, fase del reloj = 1

La primera señal de reloj bajando puede ser usada para preparar los datos. El estado inactivo del reloj es uno. Los datos en las líneas MISO y MOSI deben ser estables mientras el reloj está en alto y puede ser cambiado cuando el reloj está en bajo. Los datos son

capturados en la transición de bajo a alto del reloj y propagados en la transición de alto a bajo del reloj (Naqvi, 2015).

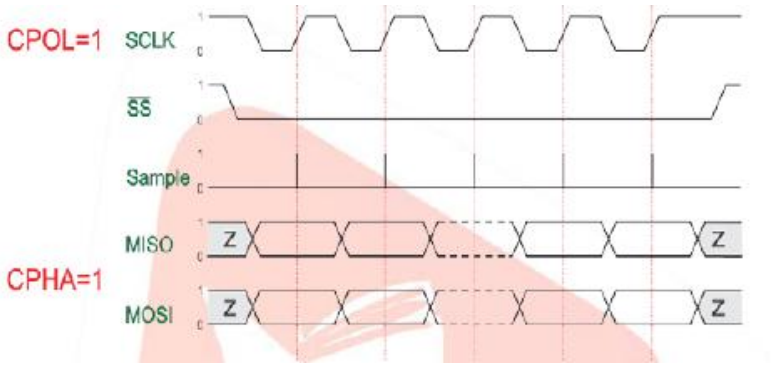


Figura 23. Polaridad del reloj=1, Fase del reloj=1 (Tomado de Naqvi, 2015).

2.6. SENSORES INERCIALES

Los sensores inerciales son dispositivos que integran acelerómetros, magnetómetros y giroscopios, capaces de medir movimientos tridimensionales (ejes X, Y, Z) tales como la aceleración lineal o la velocidad angular, es decir, cambios de velocidad con respecto al tiempo, independientemente de la influencia de la gravedad. Estos sensores no requieren de referencias externas ni de instalaciones especiales para su funcionamiento. A través del tiempo, han ido evolucionado para brindar mejores características como consumo de potencia, tamaño, reducción de costos, rangos de medición y sensibilidad, lo que permite el libre movimiento de los cuerpos bajo medición; motivo por el cual, son utilizados en air-bags, celulares, robótica, instrumentación médica, dispositivos de juego y de navegación personal, actividad sísmica en edificios, inclinación en objetos, sistemas de control de procesos, instalaciones de seguridad, medición de vibración en los coches, entre otros (Martínez y Romero, 2013).

2.6.1 PmodACL

Digilent PmodACL es un módulo acelerómetro digital de 3 ejes, que utiliza Analog Devices ADXL345, el cual proporciona cambios de aceleración de alta resolución, tales como los cambios de inclinación de menos de 1.0 °.

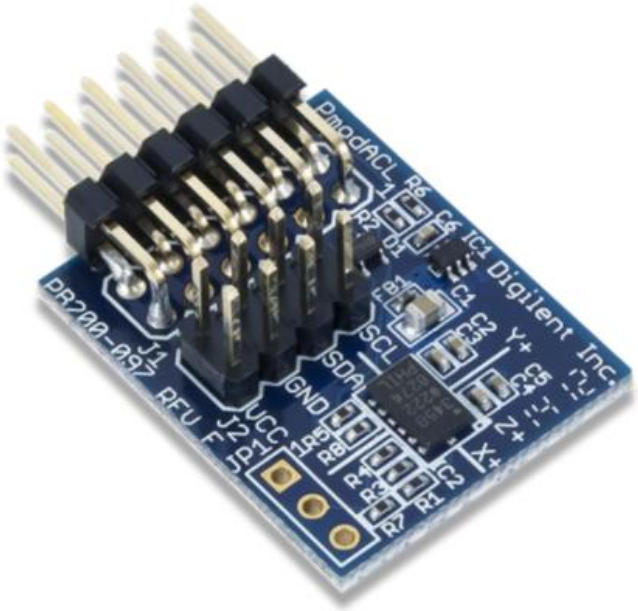


Figura 24. Módulo PmodACL (Tomado de Digilent, 2018).

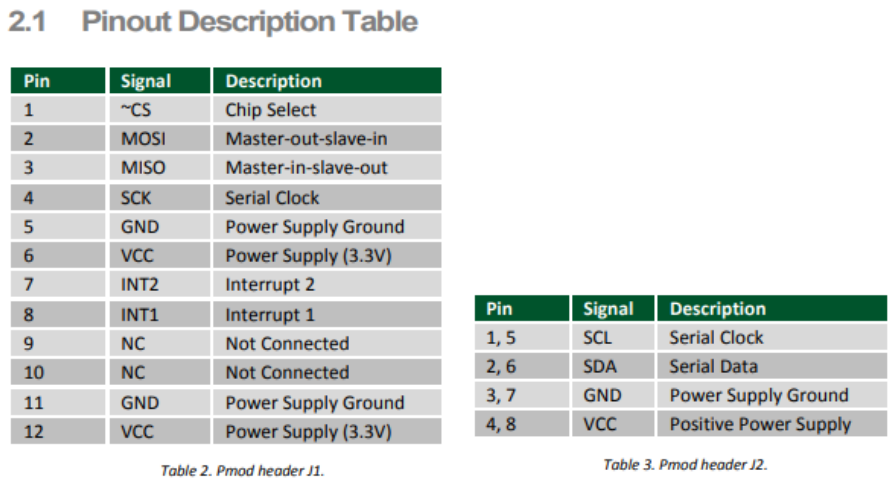
Algunas de sus principales características según Digilent (2018) son:

- 3-ejes, ±2/4/8/16g acelerómetro
- Resolución seleccionable por el usuario
- Monitoreo de actividad / inactividad
- Detección de toque simple / doble y de caída libre
- PCB de tamaño pequeño para diseños flexibles de 1.0 in × 0.8 in (2.5 cm × 2.0 cm)
- Puerto Pmod de 12 pines con interfaz SPI e interfaz I²C de 2 × 4 pines

• Biblioteca y código de ejemplo disponible

2.6.1.1. Interfaz Pmod

El PmodACL se comunica con la placa principal, en este caso con la Zedboard, a través del protocolo SPI o el protocolo I²C. En ambos casos, el ADXL345 funciona como un esclavo. La comunicación SPI se habilita cuando la línea Chip Select es llevada a un nivel lógico bajo; por el contrario, el protocolo I²C se habilita cuando la línea Chip Select es llevada a un nivel lógico alto. Además, este módulo puede ser configurado por el usuario para activar una interrupción cuando los datos de aceleración del eje medidos superen un límite de umbral definido. A continuación se muestra la descripción de pines para la comunicación por SPI (J1) e I2C (J2):



2.1 Pinout Description Table

Pin	Signal	Description
1	~CS	Chip Select
2	MOSI	Master-out-slave-in
3	MISO	Master-in-slave-out
4	SCK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V)
7	INT2	Interrupt 2
8	INT1	Interrupt 1
9	NC	Not Connected
10	NC	Not Connected
11	GND	Power Supply Ground
12	VCC	Power Supply (3.3V)

Table 2. Pmod header J1.

Pin	Signal	Description
1, 5	SCL	Serial Clock
2, 6	SDA	Serial Data
3, 7	GND	Power Supply Ground
4, 8	VCC	Positive Power Supply

Table 3. Pmod header J2.

Figura 25. Descripción de pines PmodACL (Tomado de digilent, 2018).

Para alimentar el PmodACL se necesita un voltaje entre 2.0V y 3.6V; sin embargo, se recomienda enfáticamente que el Pmod funcione a 3.3V.

2.6.1.2. ADXL345

El chip ADXL345 inmerso en el PmodACL, es un acelerómetro de 3 ejes, pequeño, delgado y de potencia ultrabaja con medición de alta resolución (13 bits). El ADXL345 es ideal para aplicaciones de dispositivos móviles. Mide la aceleración estática de la gravedad en las aplicaciones de detección de inclinación, así como la aceleración dinámica resultante del movimiento o el impacto. Se proporcionan varias funciones de detección especiales. La actividad y la detección de inactividad detectan la presencia o la falta de movimiento comparando la aceleración en cualquier eje con los umbrales establecidos por el usuario. Tap Sensing detecta golpes simples y dobles en cualquier dirección. La detección de caída libre detecta si el dispositivo se está cayendo. Estas funciones se pueden asignar individualmente a cualquiera de los dos pines de salida de interrupción
(Analog devices, 2015).

3. METODOLOGÍA

El presente trabajo se llevó a cabo en el laboratorio de Microelectrónica y Nanotecnología del Bloque O del ITM (sede Fraternidad), donde fue facilitada una FPGA Zedboard. Adicionalmente, fueron proporcionados dos sensores PmodACL por parte del Laboratorio de Sistemas de Control y Robótica, ubicado en el "Parque i" del ITM.

En este apartado se desarrolla la metodología implementada en este trabajo, la cual está dividida en 5 etapas. En la primera etapa se abarca la instalación, licenciamiento e instalación de librerías del software de Xilinx. En la segunda etapa se crea el proyecto IP en Vivado. En la tercera etapa se exporta el archivo bitstream al programa SDK y se crea un proyecto con el código ejemplo para los módulos Pmod. En la cuarta etapa se realizan los cambios necesarios al proyecto en Vivado y SDK para leer dos PmodACL simultáneamente. Finalmente, se descarga los archivos de configuración a la Zedboard y se realiza el montaje final para la lectura de dos PmodACL en simultáneo.

3.1. ETAPA 1

La implementación de este producto de laboratorio se llevó a cabo con los siguientes equipos y software:

- Tarjeta FPGA Zedboard de Xilinx
- Cables micro USB
- Dos sensores PmodACL
- Vivado 2015.4 Xilinx
- SDK 2015.4 Xilinx

- Computador portátil Asus Core i5 RAM 6Gb, 750Gb de disco duro equipado con Windows 7 home 64 bit.

3.1.1. Instalación del software

Para comenzar, es necesario descargar el software desde la página oficial del fabricante desde el siguiente enlace: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

Nota: En este producto de laboratorio se utilizó la versión del software **Vivado 2015.4**

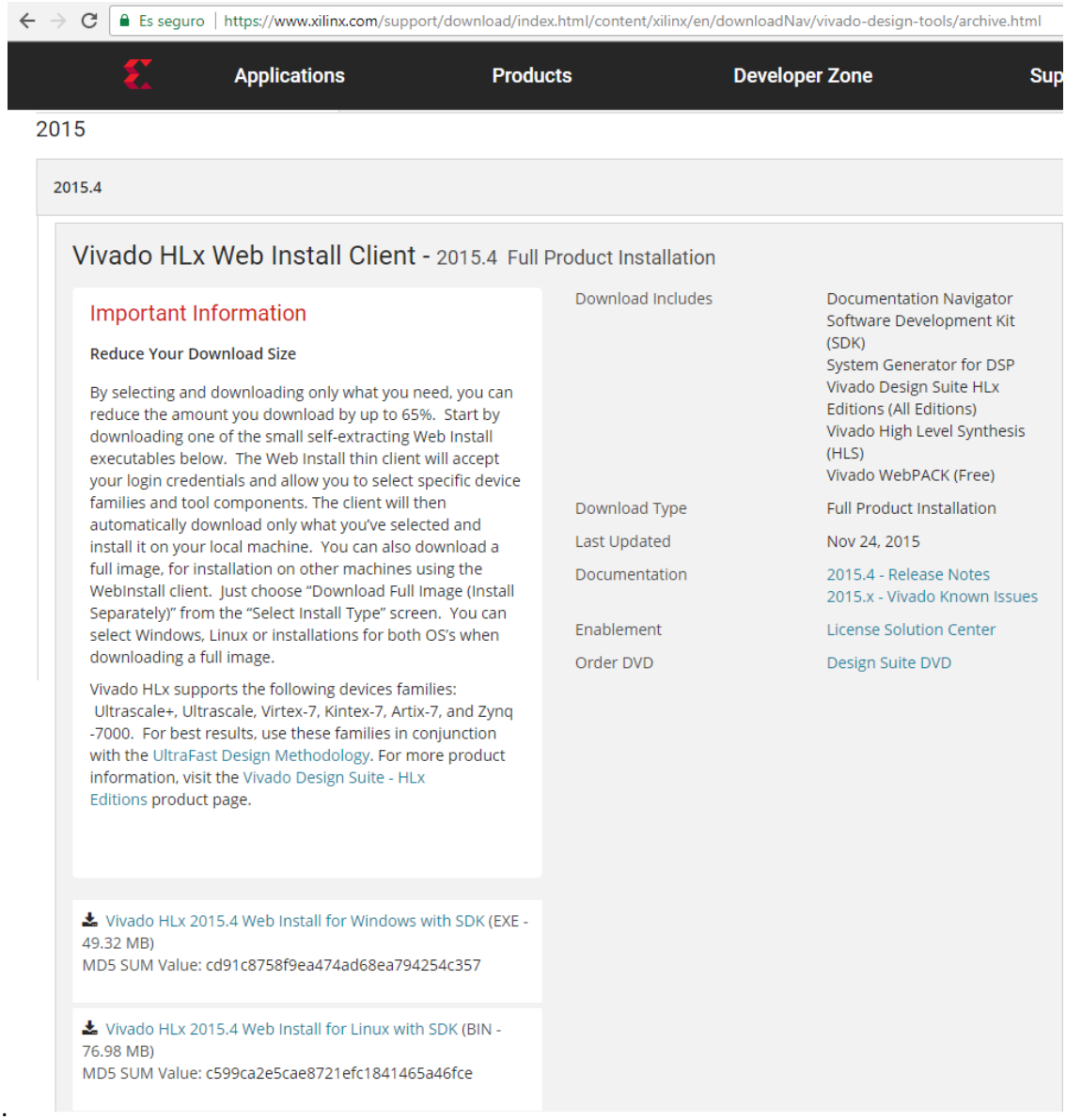


Figura 26. Descarga Vivado 2015.4 (Tomado de Xilinx, 2018).

Para realizar la descarga e instalación del software es necesario registrarse en la página del fabricante. Existe una versión gratuita limitada del software llamada "Vivado HL WebPacK", utilizada en este proyecto. Es importante tener en cuenta que el fabricante recomienda instalar el software en un sistema operativo de 64 bits. Durante la instalación se deben

aceptar los términos y condiciones de uso del software, tal como se muestra en la figura 27. Además, se recomienda desactivar el antivirus para reducir el tiempo de instalación y cualquier percance.

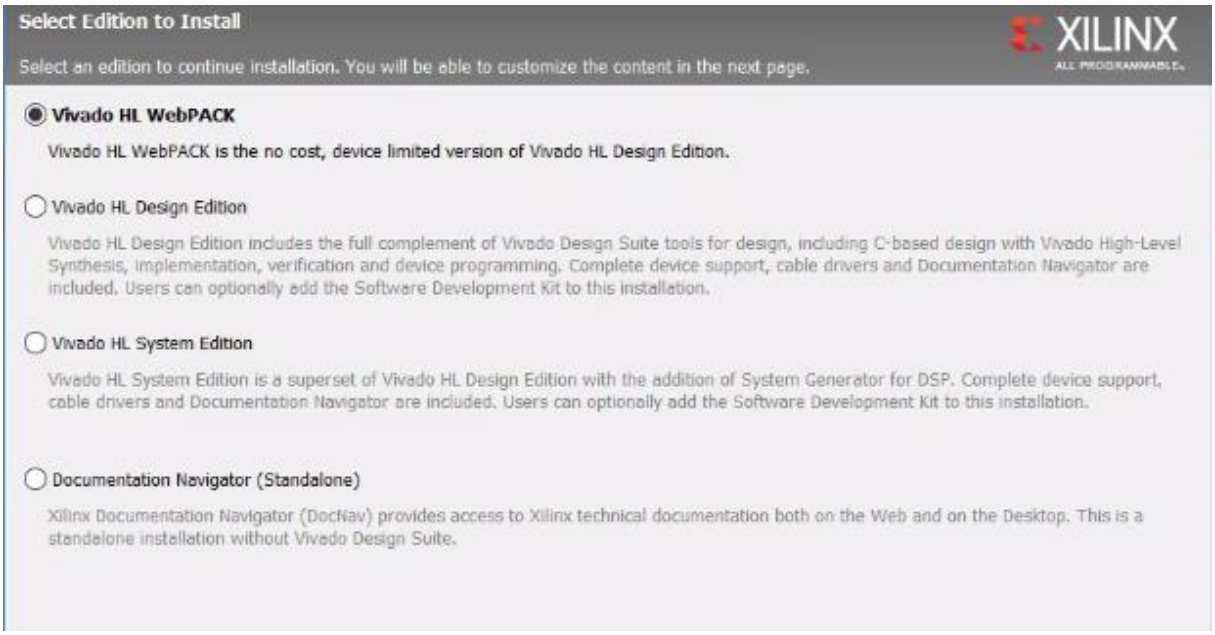


Figura 27. Instalación Vivado HL WebPacK (Autoría propia).

Durante la instalación es importante incluir el ítem “Software Development Kit (SDK)” como se muestra en la figura 28, ya que éste es necesario para crear el código con el cual se programará la Zedboard. También, es relevante seleccionar la casilla “Devices” para incluir las familias de dispositivos disponibles y de esta manera, continuar con los pasos de la instalación como lo recomienda el fabricante.

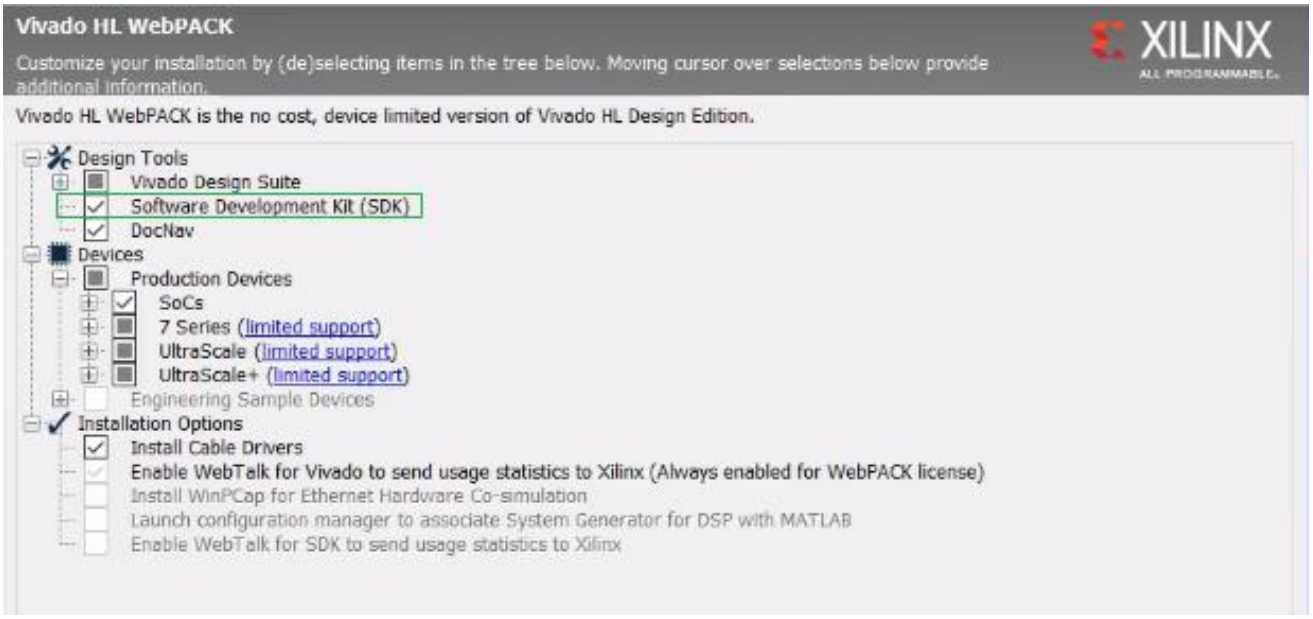


Figura 28. Inclusión de SDK en la instalación (Autoría propia).

3.1.2. Instalación de licencia WebPacK

Cuando se realiza la instalación del software, se abrirá una ventana sobre la administración de la licencia. Para acceder a la licencia gratuita WebPacK se debe hacer click en “Obtain License”, luego en “Get Free Licenses – Vivado WebPacK, SDK, free IP and more” y finalmente en “Connect Now” como se muestra en la figura 29.

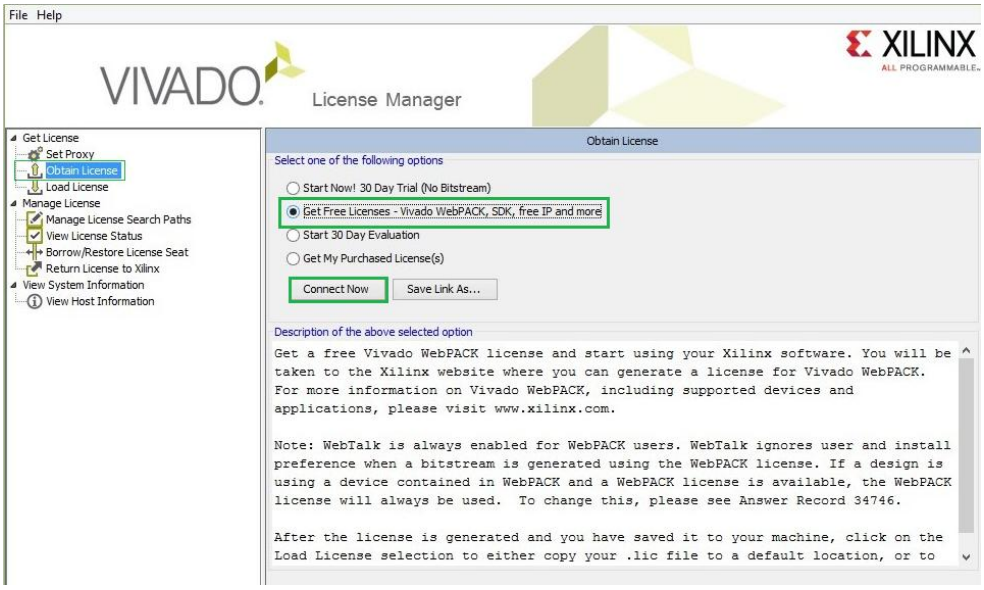


Figura 29. Licencia gratuita (Autoría propia).

Se abrirá la página del fabricante donde se debe iniciar sesión con el usuario y contraseña previamente creados, se selecciona el tipo de licencia requerida para los programas a utilizar y luego se da click en “Generate Node-Locked License”, tal como se muestra en la figura 30.

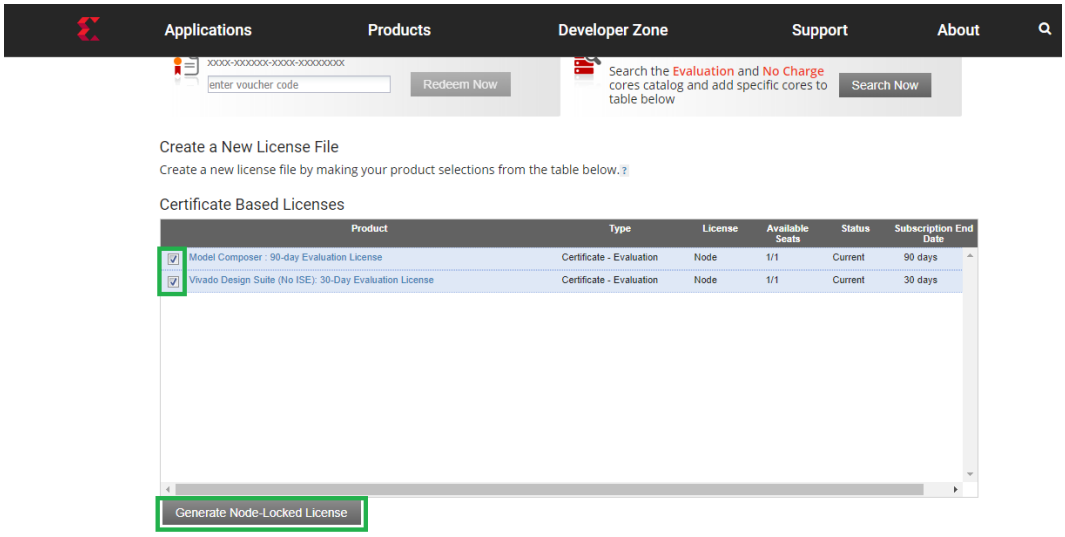


Figura 30. Generación licencia gratuita (Tomado de Xilinx, 2018).

Al correo con cual se hizo el registro, llegará el archivo de licencia. Se debe realizar la activación de la licencia dando click en “Load License”, posteriormente en “Activate License”, por último, se cargará el archivo y quedará listo el software con licencia limitada por un año, tal como se muestra en la figura 31.

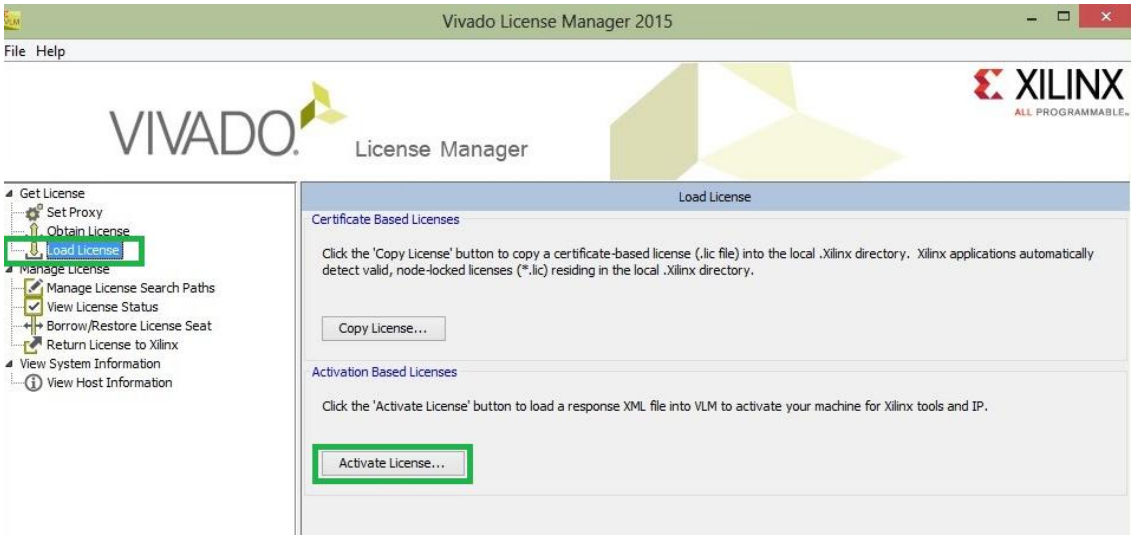


Figura 31. Activación licencia (Autoría propia).

3.1.3. Instalación de librerías

Una vez se tenga instalado y licenciado el software, se procede a descargar las librerías necesarias para la realización de este proyecto.

Guiado por la publicación de Xilinx “Vivado Version 2015.1 and Later Board File Installation”, se procede a descargar e instalar la librería de las familias de tarjetas de Digilent en la carpeta board files, tal como se muestra en la figura 32 y en el siguiente enlace: https://reference.digilentinc.com/reference/software/vivado/board-files?redirect=1

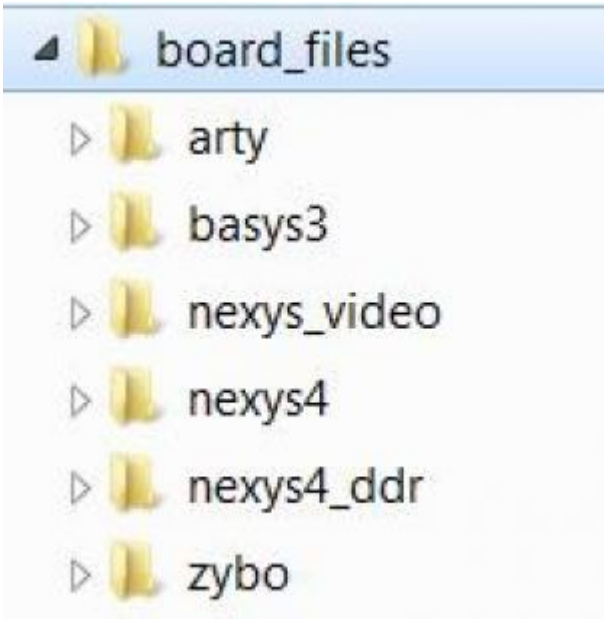


Figura 32. Instalación librería de tarjetas (Tomado de Digilent, 2018).

Después de instalar la librería de las tarjetas, es necesario instalar la librería "Digilent Vivado IP"como se muestra en la figura 33, la cual permite agregar dispositivos periféricos, como el PmodACL, al diseño IP que se realizará en el software Vivado. En el siguiente enlace se encuentra la publicación "Getting Started with Digilent Pmod IPs", donde en el paso 2 "Add the Digilent Library Repository" se indica cómo realizar la instalación de la librería: https://reference.digilentinc.com/learn/programmable-logic/tutorials/pmod-ips/start#add_the_digilent_library_repository

Para la descarga directa de la librería, se puede dirigir al siguiente enlace: https://github.com/Digilent/vivado-library/releases/tag/v2015.4-3

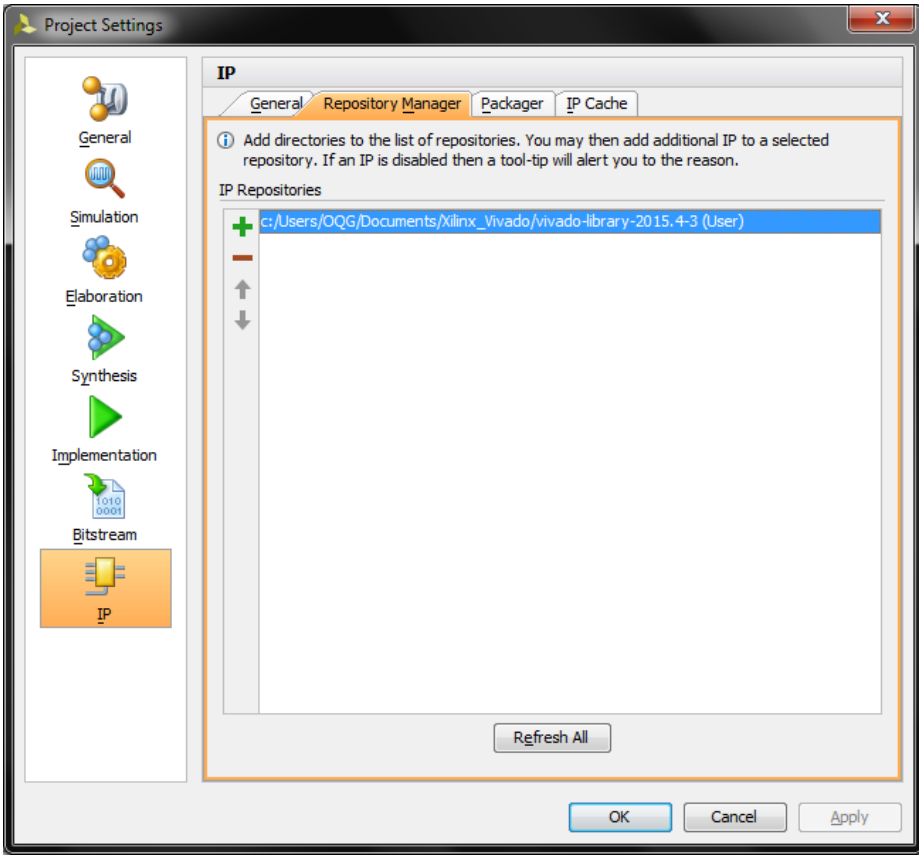


Figura 33. Librería dispositivos periféricos (Autoría propia).

3.2. ETAPA 2

Para comenzar con la creación del proyecto en Vivado, es importante saber cómo se crea un proyecto IP básico desde cero. Para ello, el fabricante recomienda hacer el tutorial "Getting Started with Zynq" para familiarizarse con el entorno de diseño y sus pasos: https://reference.digilentinc.com/learn/programmable-logic/tutorials/zedboard-getting-started-with-zynq/start

Adicionalmente, se puede remitir al siguiente tutorial sobre cómo crear un proyecto IP: https://reference.digilentinc.com/vivado/getting-started-with-ipi/start

3.2.1. Creación de proyecto en vivado (tutorial Xilinx)

Para crear un proyecto Pmod IP en Vivado, el fabricante proporciona un tutorial llamado "Getting Started with Digilent Pmod IPs", en el cual se muestra paso a paso cómo crear un proyecto para ser utilizado con un módulo Pmod de Digilent; en este caso con un módulo PmodACL. El enlace para acceder al tutorial es el siguiente:
https://reference.digilentinc.com/learn/programmable-logic/tutorials/pmod-ips/start

Teniendo en cuenta la anterior información, es importante ilustrar y hacer énfasis en algunas configuraciones necesarias para este proyecto:

El tipo de tarjeta Zedboard es la proporcionada por Digilent y el lenguaje de programación a usar es VHDL, tal como se muestra en la figura 34 y 35 respectivamente.

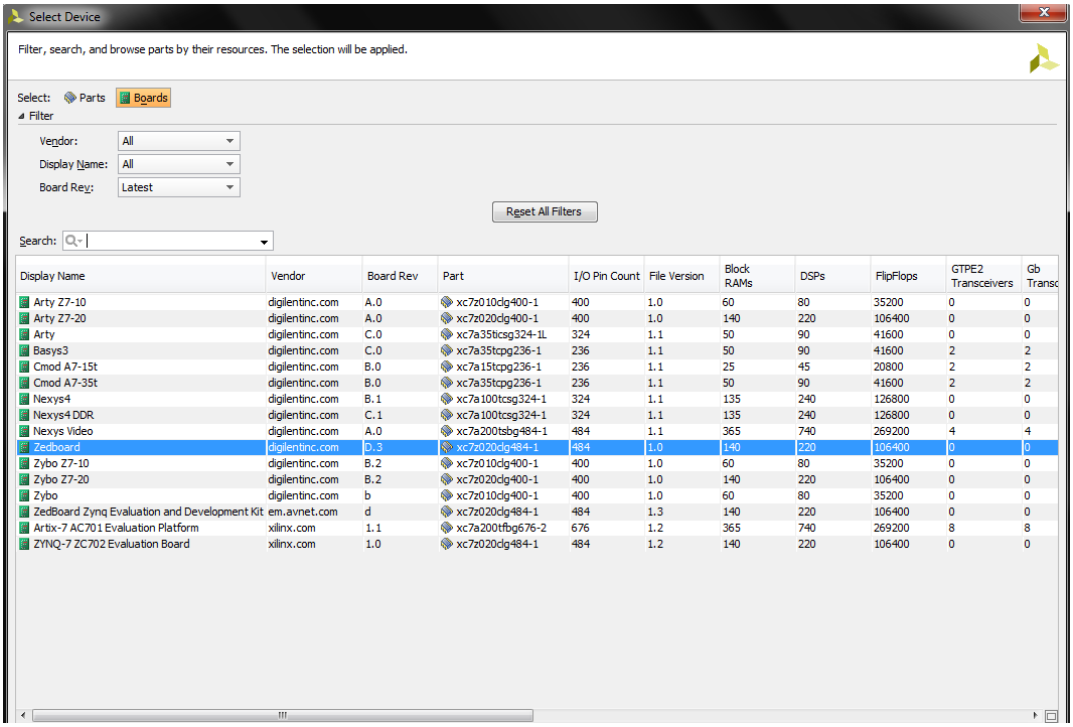


Figura 34. Tarjeta Zedboard Digilent (Autoría propia).

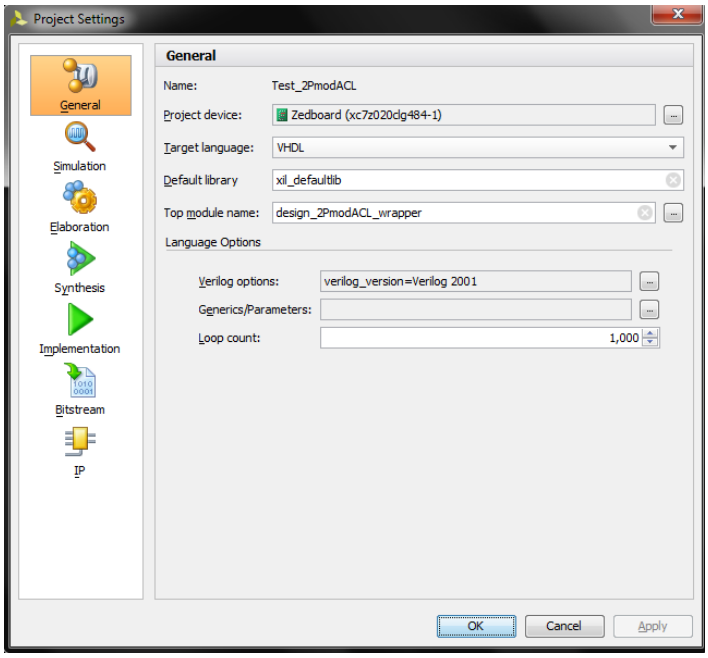


Figura 35. Lenguaje VHDL (Autoría propia).

Al hacer doble click en el procesador ZYNQ7 Processing System, es importante habilitar el reloj FCLK_CLK1 a 80 MHz tal como se muestra en la figura 36 y como lo indica el fabricante en el siguiente enlace:

https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodACL_v1_0

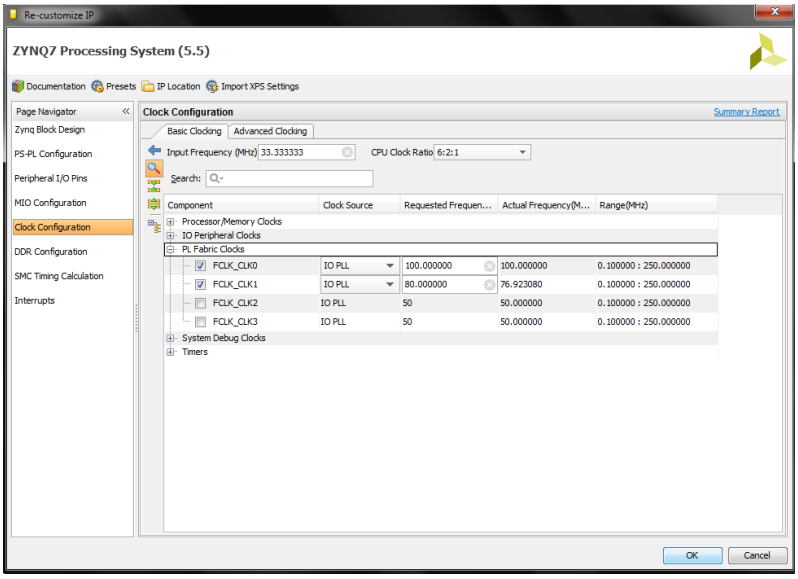


Figura 36. Reloj adicional (Autoría propia).

Para seleccionar el módulo PmodACL, se hace doble click en el conector Pmod de la tarjeta Zedboard como se muestra en la figura 37. En este caso se utilizó el Pmod JA.

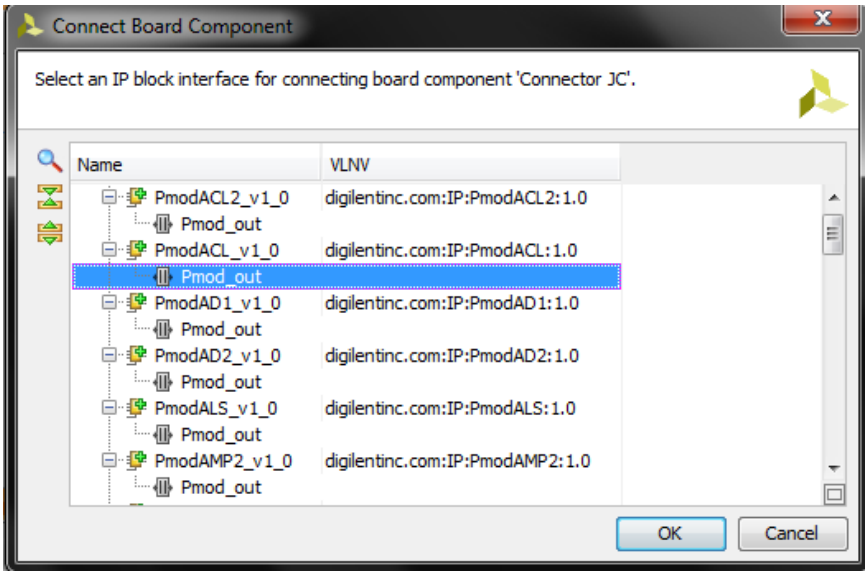


Figura 37. Pmod conector PmodACL (Autoría propia).

Una vez realizada la síntesis del proyecto, se puede verificar la asignación de pines para cada uno de los conectores Pmod, como se muestra en la figura 38.

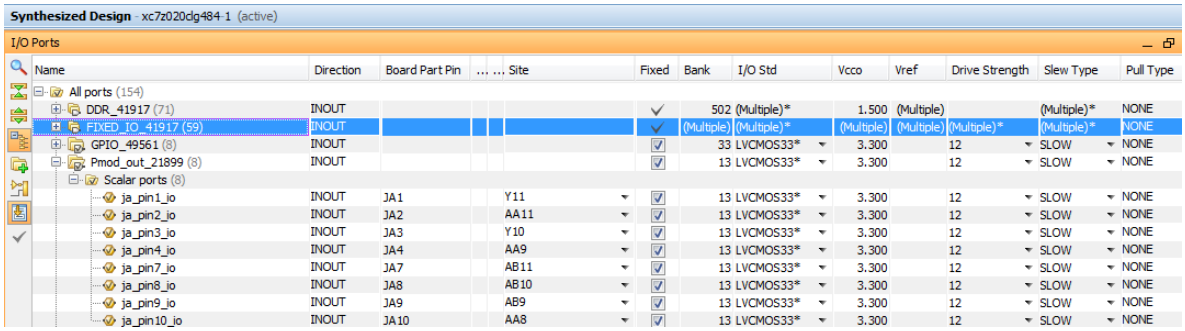


Figura 38. Asignación de pines (Autoría propia).

Culminando el paso 8 del tutorial "Getting Started with Digilent Pmod IPs", se tendrá un diseño IP similar al mostrado en la figura 39.

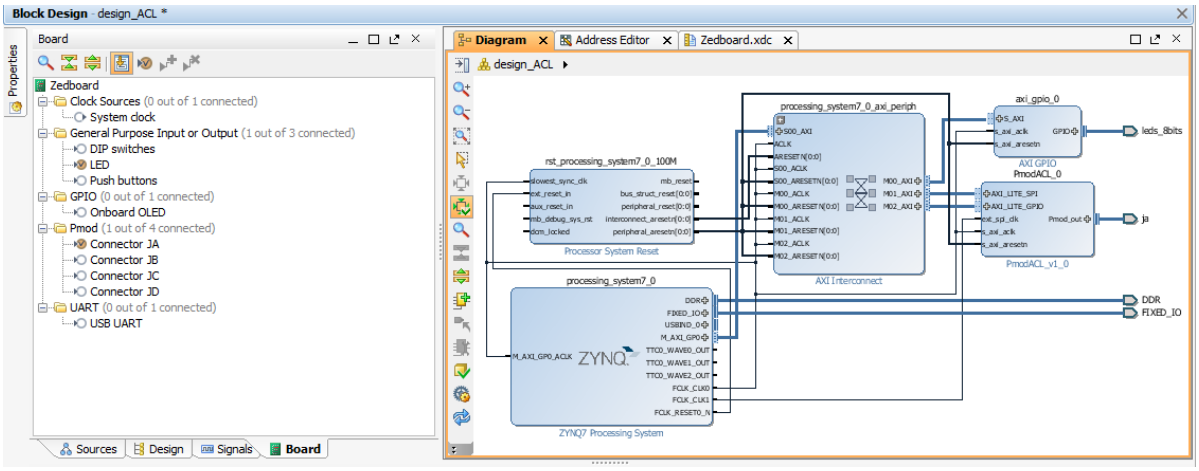


Figura 39. Diseño de bloque IP (Autoría propia).

Una vez se tenga creado el diseño del hardware y el archivo bitstream en Vivado, se siguen los siguientes pasos para exportar el proyecto al SDK, para ello se debe hacer click en File -> Export -> Export Hardware -> Include bitstream -> Ok, tal como se muestra en la figura 40.

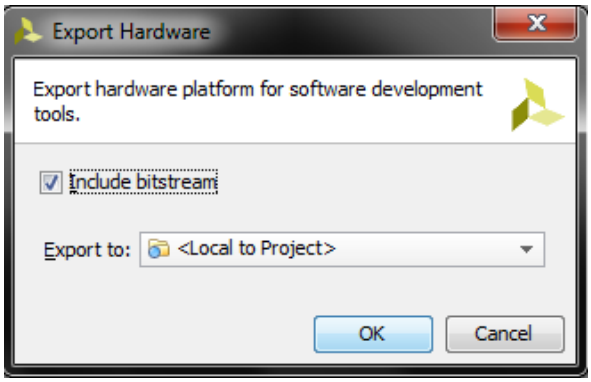


Figura 40. Exportación Hardware a SDK (Autoría propia).

Finalmente hacer click en File -> Launch SDK -> ok

3.3. ETAPA 3

3.3.1. Creación proyecto en SDK y código ejemplo SPI para PmodACL

En este caso, para la creación del proyecto en SDK, se debe importar el código ejemplo que el fabricante brinda para cada módulo Pmod, siguiendo los pasos del 10 al 14 del ya mencionado tutorial “Getting Started with Digilent Pmod IPs”.

Además, existe un tutorial llamado “The Zynq Tutorials for Zybo and ZedBoard”, el cual sirve de gran ayuda para una ilustración más detallada acerca de la creación del proyecto en SDK y su posterior ejecución en la FPGA. Este recurso se puede descargar desde el siguiente enlace:
http://www.zynqbook.com/download-tuts.html

Adicionalmente, es importante tener en cuenta las siguientes configuraciones e información:

Para utilizar el código ejemplo del módulo PmodACL, como se muestra en la figura 41, se copia el archivo main.c de la carpeta “examples” y se pega en la carpeta “src” del proyecto creado.

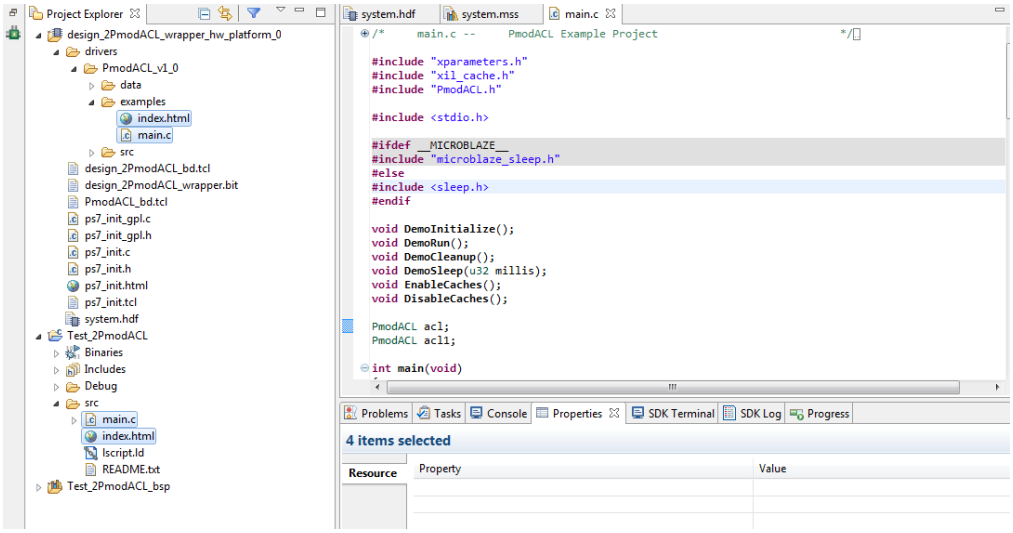


Figura 41. Código ejemplo PmodACL (Autoría propia).

Además, se debe realizar la configuración de los jumpers JP7 al JP11 de la FPGA para trabajar en el modo JTAG, como se muestra en la figura 42. Para este proyecto se requiere tener ambos cables micro USB conectados desde el computador a los puertos UART y JTAG.

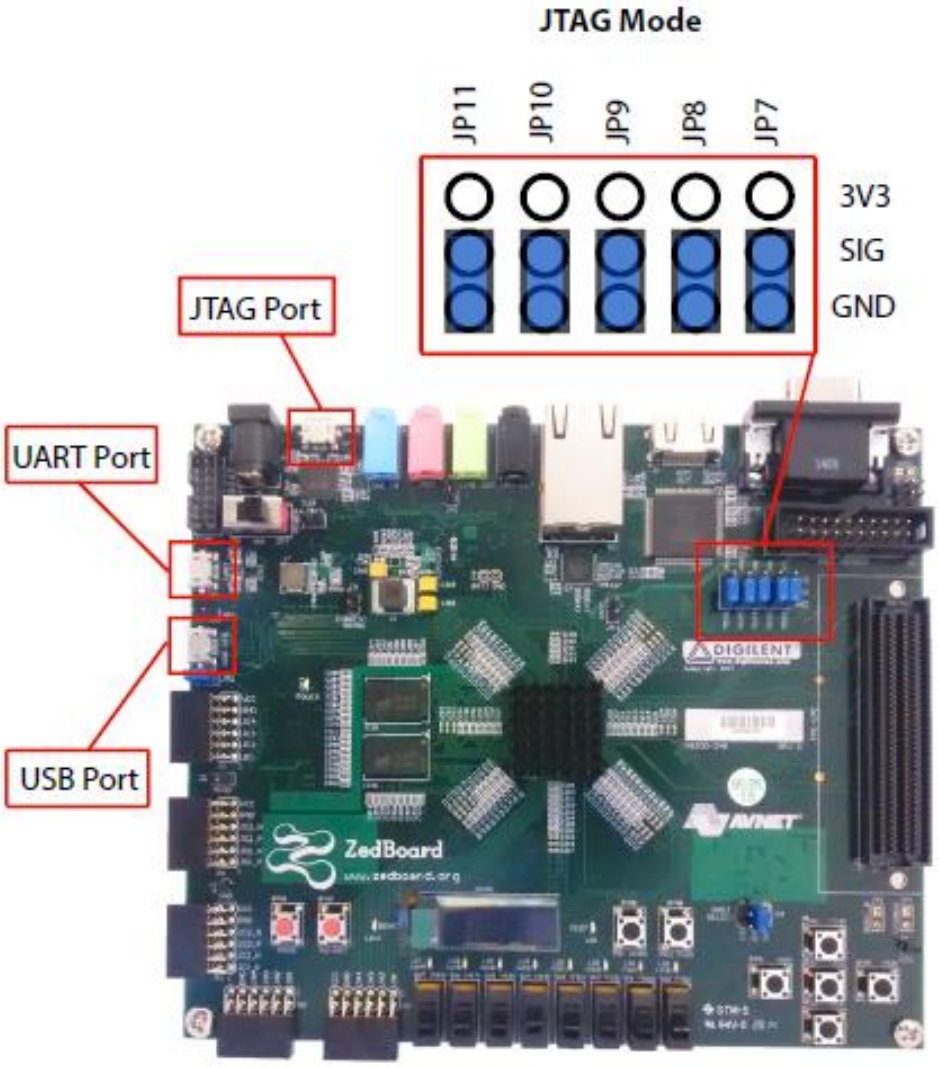


Figura 42. Puertos UART y JTAG (Crockett *et al.,* 2015).

Para cargar el diseño del hardware a la FPGA, es necesario garantizar que esté conectada a la fuente de alimentación y esté encendida. Además, debe estar conectada al computador por medio del cable micro USB al puerto UART. Una vez garantizado esto, hacer click en Xilinx Tools -> Program FPGA -> Program, como se muestra en la figura 43.

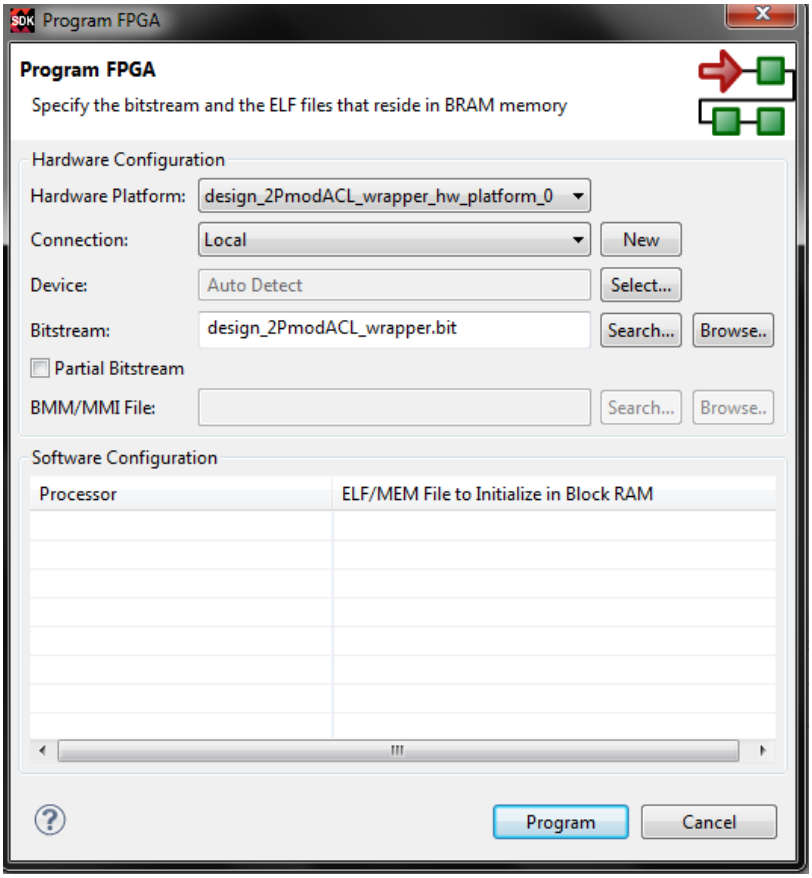


Figura 43. Programación FPGA (Autoría propia).

Para cargar el código creado a la FPGA, es necesario seleccionar la carpeta del proyecto creado, luego dar click en el símbolo Run -> Run as -> Launch on Hardware (System Debugger), tal como se muestra en la figura 44.

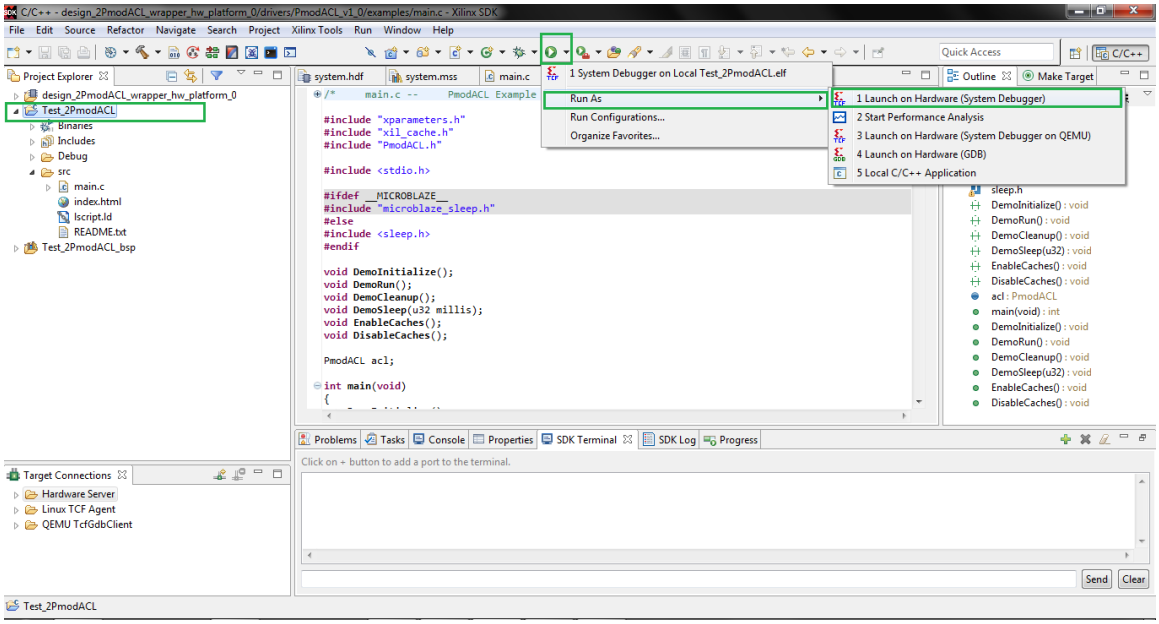


Figura 44. Ejecución código en FPGA (Autoría propia).

Finalmente, para visualizar los datos obtenidos desde el PmodACL, como se muestra en la figura 47, es necesario configurar un terminal serial a 115200 baudios como lo indica el fabricante y como se muestra en las figuras 45 y 46. En este caso se utilizó el terminal serial del SDK.

Demo Program in Xilinx SDK

To set up the demo, a serial terminal, such as TeraTerm, needs to be used to see the data being printed out. Settings for the terminal will vary depending on your board.

For Zynq projects, apply the following settings:

- Baud rate: 115200
- Data bits: 8
- Parity: none
- Stop bits: 1

Figura 45. Configuración terminal serial (Tomado de Digilent, 2018).

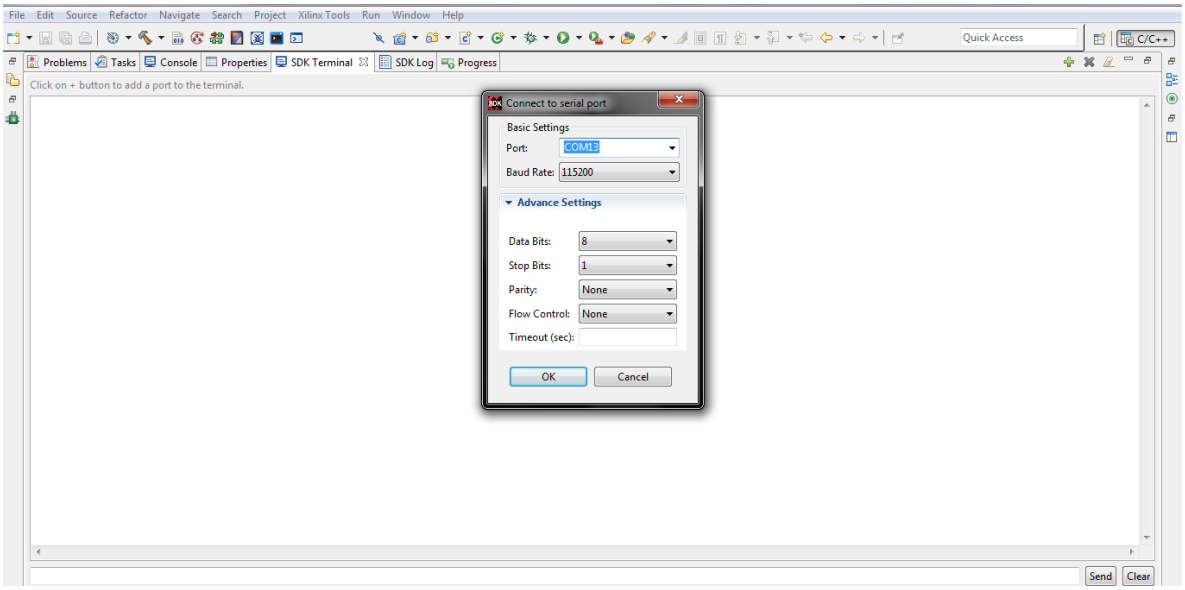


Figura 46. Terminal serial SDK (Autoría propia).

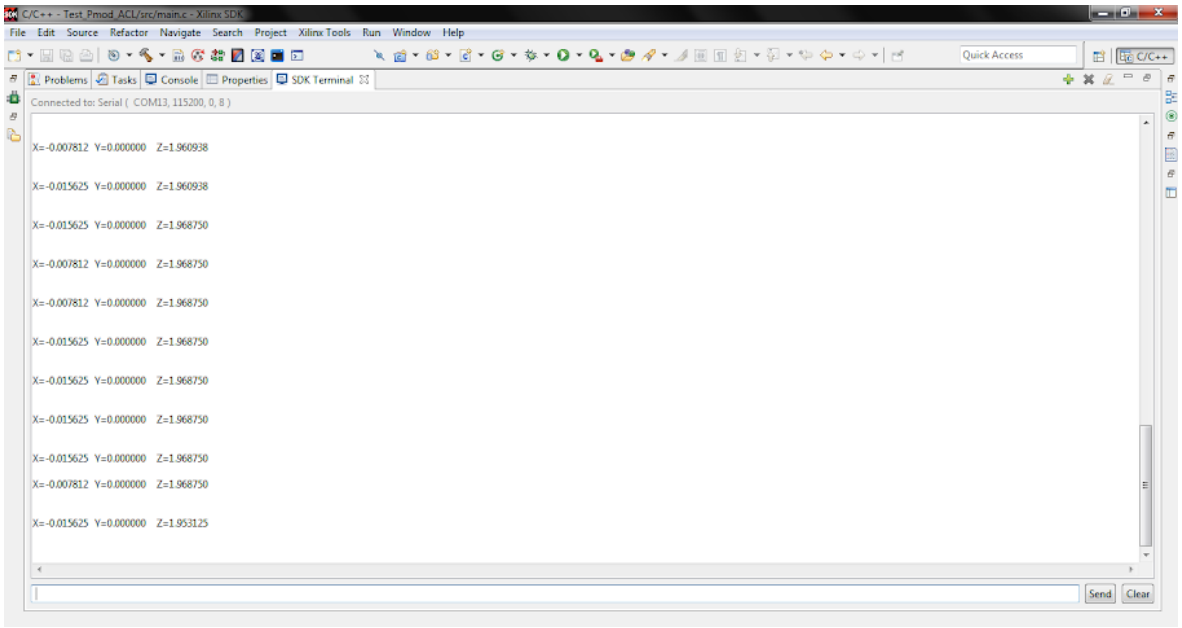


Figura 47. Lectura PmodACL (Autoría propia).

3.4. ETAPA 4

3.4.1. Modificaciones en diseño IP y en SDK

Una vez se tiene establecida la comunicación entre una FPGA Zedboard y un módulo PmodACL mediante el protocolo SPI, se procede a realizar las modificaciones al diseño IP realizado en Vivado y al código ejemplo en el SDK para trabajar en simultáneo con dos módulos PmodACL.

En primera instancia, para modificar el diseño del hardware en Vivado, será necesario ingresar a “IP Integrator”, luego dar click en “Open block design”, después dar click en la pestaña “Board”. Seguido a esto, desplegar el sub-menú “Pmod” y dar doble click en “Connector JB”, para seleccionar el módulo PmodACL. Esto dará como resultado dos PmodACL en el diseño de bloques IP. Para este proyecto se utilizaron los conectores Pmod JA y Pmod JB, tal como se muestra en la figura 48. Para enlazar el nuevo PmodACL al diseño ya realizado, se deben seguir los pasos enunciados en la etapa 2 referentes a la creación de bloques IP y su conexión automatizada.

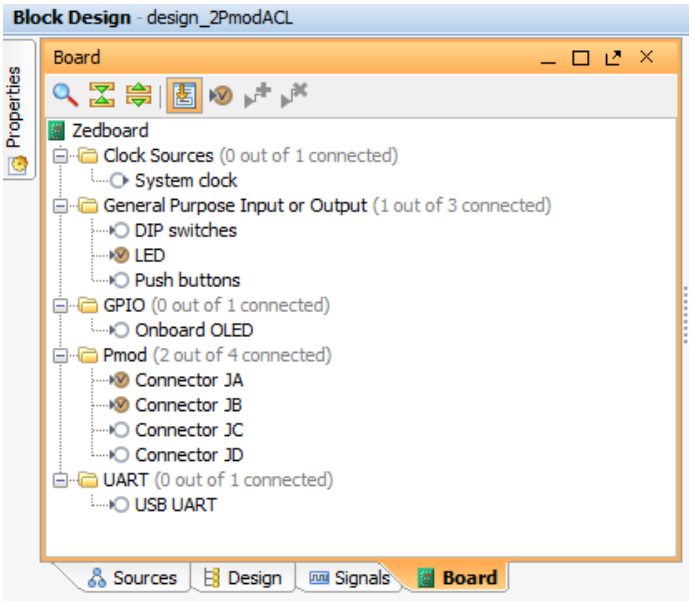


Figura 48. Conectores JA y JB PmodACL (Autoría propia).

Al terminar el diseño IP, así como la etapa de síntesis e implementación, se podrá obtener la nueva asignación de pines en la cual se incluyen los dos PmodACL como se muestra en las figuras 49 y 51.

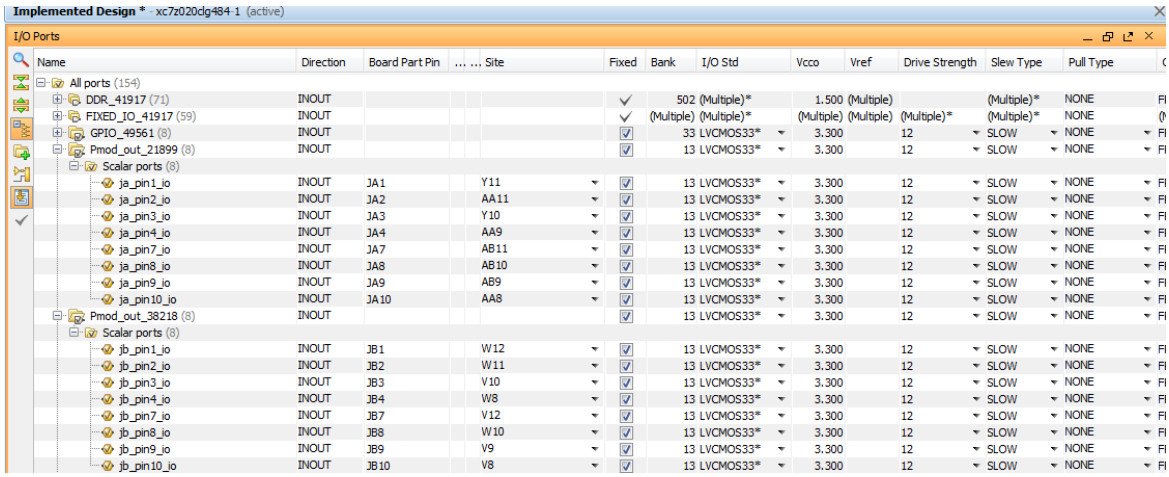


Figura 49. Asignación de pines a dos PmodACL (Autoría propia).

Finalmente, se obtendrá el siguiente diagrama de bloques IP, como se muestra en la figura 50.

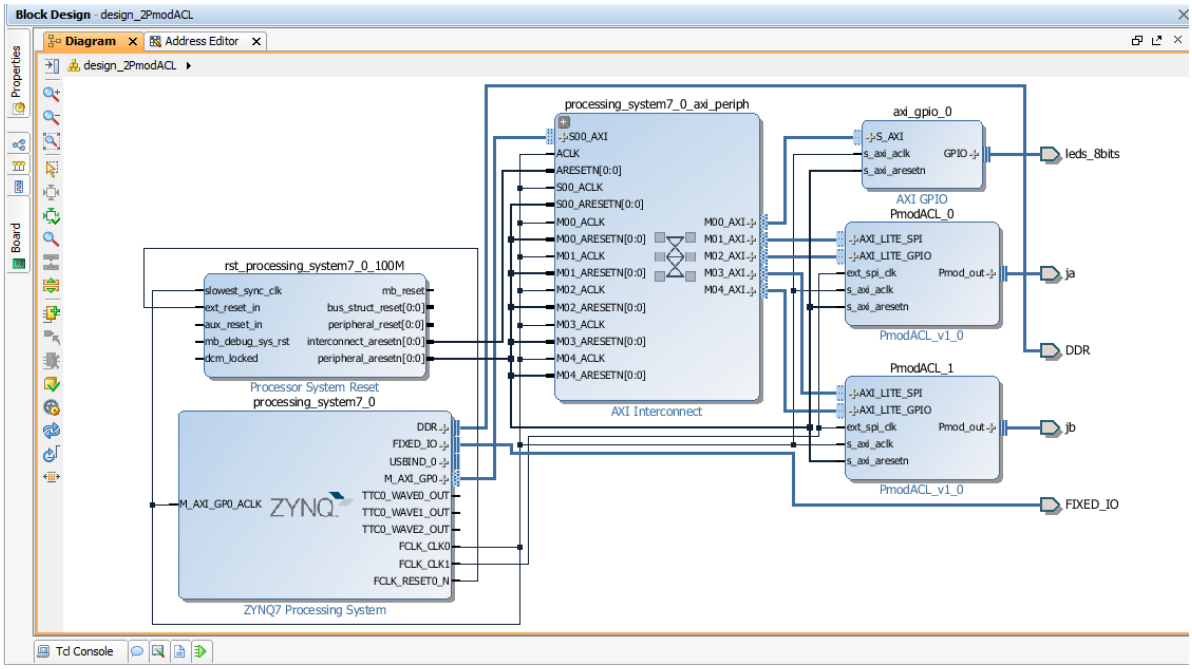


Figura 50. Diseño IP con dos PmodACL (Autoría propia).

Nota: Ver apéndice A para visualizar el código del diseño del hardware en Vivado "design_2PmodACL_wrapper.vhd", donde se especifican las conexiones, mapeo de puertos y entidades a utilizar en el diseño final.

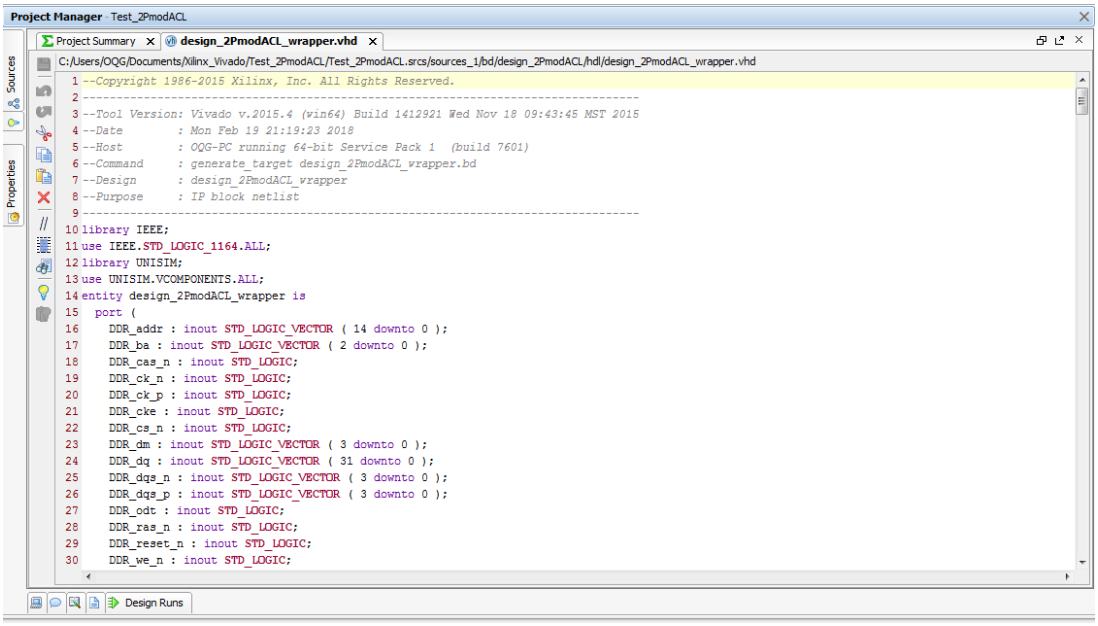


Figura 51. Wrapper diseño final (Autoría propia).

Una vez se tiene el diseño en Vivado, se procede a generar el nuevo archivo bitstream para ser exportado al SDK, donde se siguen los pasos citados en la etapa 3 hasta agregar el código ejemplo “main.c”. Allí se realizarán los cambios pertinentes para leer en simultáneo los dos módulos PmodACL.

Seguidamente, los cambios que se deben realizar en el SDK son los siguientes: incluir en la declaración un segundo PmodACL “acl1”, duplicar las líneas necesarias dentro del “void DemoInitialize” y del “void DemoRun” para reemplazar el “PmoaACL_0” y el “acl0”, por “PmoaACL_1” y “acl1” respectivamente, tal como se muestra en la figura 52:

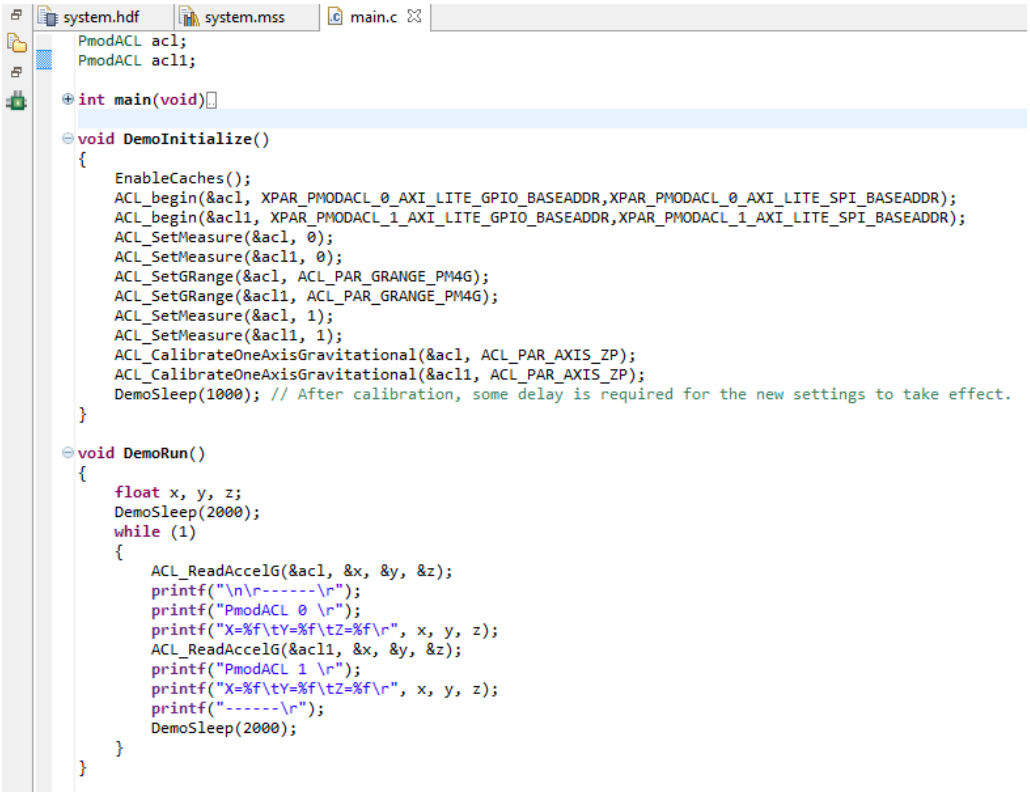


Figura 52. Modificaciones código SDK (Autoría propia).

Nota: Ver apéndice B para visualizar el código final “main.c” del proyecto en SDK.

3.5. ETAPA 5

3.5.1. Montaje físico final

En esta última etapa, en las figuras 53 a la 56, se muestran los movimientos hechos para la prueba y el montaje final en la Zedboard para leer en simultáneo los dos módulos PmodACL.

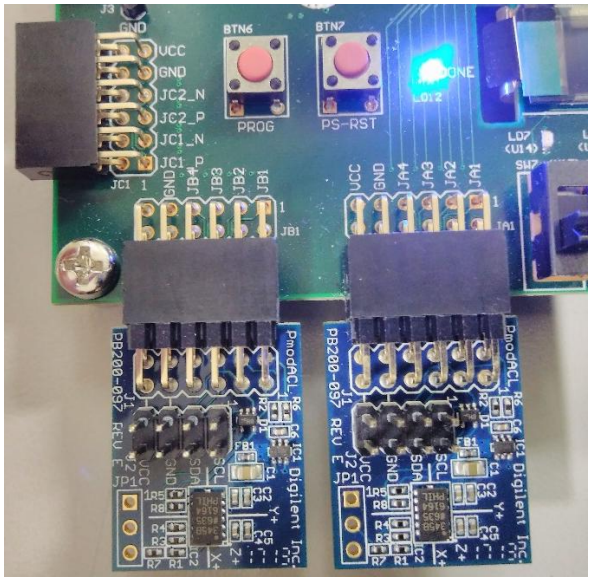


Figura 53. Montaje de dos PmodACL (Autoría propia).

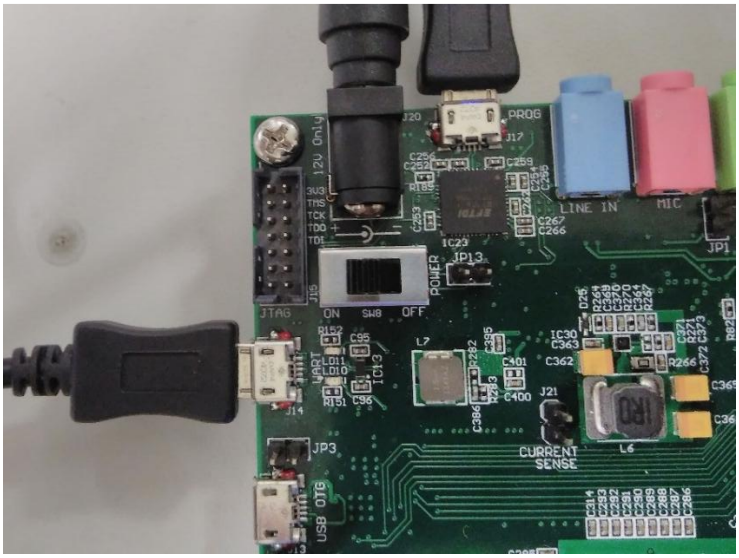


Figura 54. Conexiones USB y alimentación (Autoría propia).

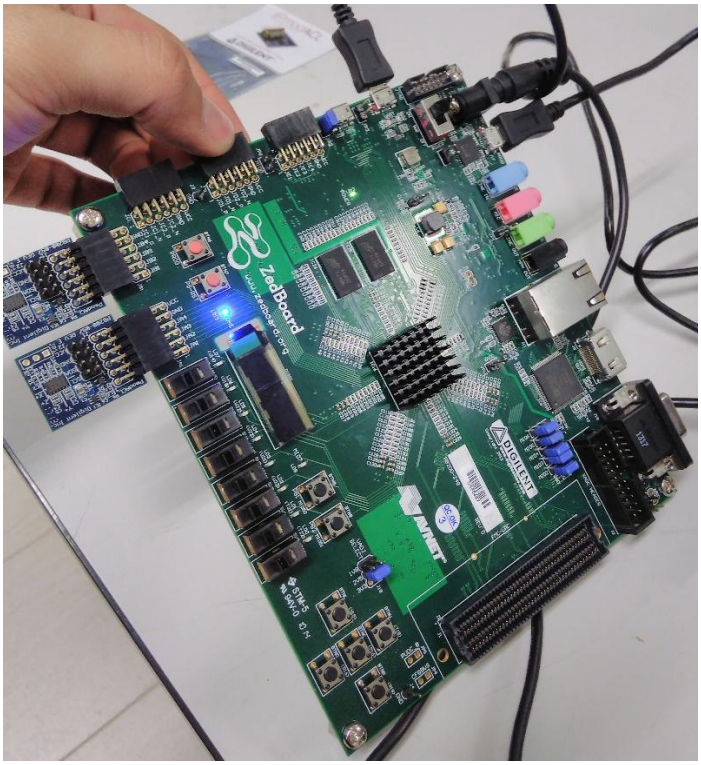


Figura 55. Pruebas de giro y aceleración (Autoría propia).

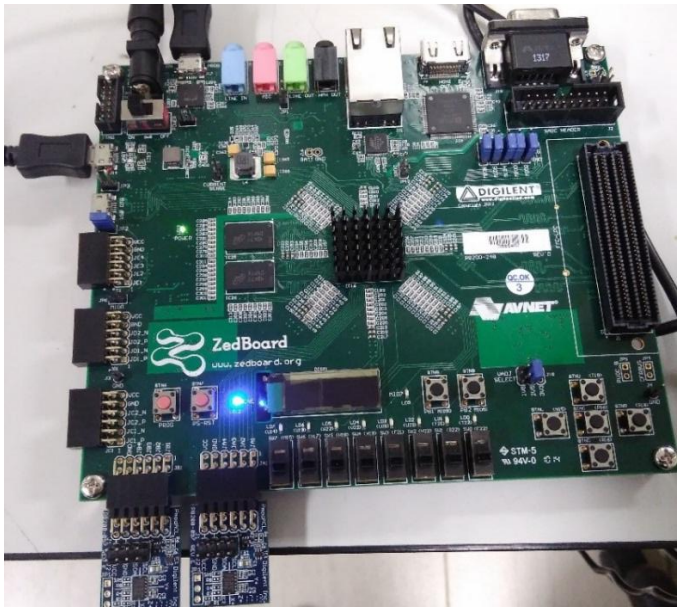


Figura 56. Zedboard con dos PmodACL (Autoría propia).

4. RESULTADOS Y DISCUSIÓN

Se logró establecer una comunicación SPI entre una FPGA Zedboard y un sensor inercial PmodACL, lo cual se basó en el tutorial "Getting Started with Digilent Pmod IPs". A diferencia del tutorial, en este trabajo se realizaron recomendaciones y pasos adicionales que no están explícitos a lo largo del tutorial, los cuales son necesarios para lograr el objetivo de este proyecto, especialmente para aquel que no tenga conocimientos previos relacionados con la programación y manejo del software empleado.

En segundo lugar, gracias a las modificaciones realizadas en este trabajo, se consiguió establecer una comunicación SPI entre una FPGA Zedboard y dos sensores inerciales PmodACL. Aunque este proyecto estuvo guiado por el tutorial anteriormente mencionado, la finalidad de ambos trabajos es diferente, ya que el tutorial brindado por el fabricante se centró en demostrar la compatibilidad y funcionalidad de los dispositivos entre sí, mientras que el presente trabajo tiene como finalidad establecer una comunicación SPI entre una FPGA Zedboard y dos sensores inerciales PmodACL para obtener lecturas en simultáneo.

El resultado obtenido a partir de la lectura simultánea de los dos sensores inerciales PmodACL, se muestra en la figura 57.

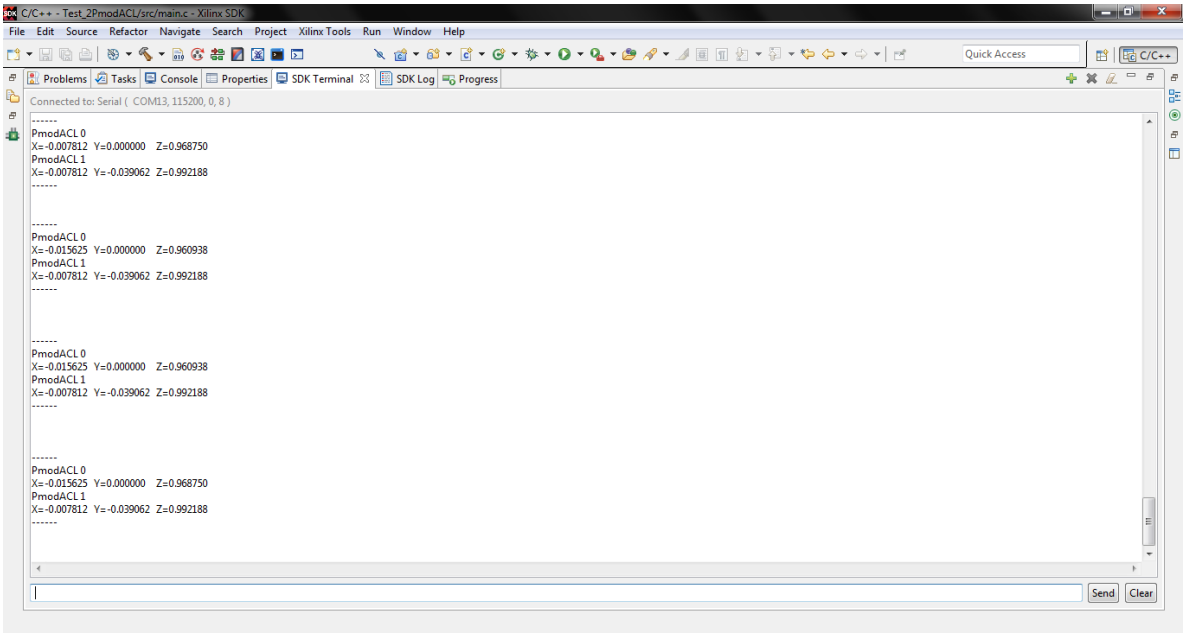


Figura 57. Datos terminal serial de dos PmodACL (Autoría propia).

Nota: Ver apéndice C para visualizar los datos de la lectura de los dos PmodACL en simultáneo.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

- Es posible establecer una comunicación SPI entre una FPGA Zedboard y dos sensores inerciales PmodACL, para ser leídos en simultáneo.
- Para la lectura de dos sensores inerciales PmodACL en simultáneo, se deben realizar modificaciones tanto en el diseño del hardware en el software Vivado, como en el diseño del código en el software SDK del fabricante.
- Se recomienda acceder a la base de datos y foros del fabricante, para fortalecer el conocimiento teórico y profundizar en el contenido de este proyecto.
- En trabajos futuros, se puede continuar con una etapa de acondicionamiento de la información obtenida, con el fin de crear diversas aplicaciones electrónicas.
- Los resultados de este proyecto podrán ser empleados en trabajos futuros, donde se podrán desarrollar nuevas implementaciones electrónicas que permitan incrementar la demanda de estos dispositivos en la industria y dar solución a problemas de diseño electrónico relacionados con la velocidad de procesamiento de información.

REFERENCIAS

Álvarez, A. (2016). *Co-diseño hardware-software de un sistema de posicionamiento basado en procesado de vídeo* (Tesis de pregrado). Universidad de Cantabria, Cantabria, España. Recuperado de https://repositorio.unican.es/xmlui/bitstream/handle/10902/8657/385772.pdf?sequence=1

Analog Devices. (2015). *Digital Accelerometer ADXL345*. Recuperado de http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

AVNET. (2014). *ZedBoard Hardware User's Guide*. Recuperado de http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

Bustos, Á. (2013). *Development of Embedded Linus Applications Using Zedboard* (Tesis de pregrado). Universidad Politécnica de Madrid, Madrid, España. Recuperado de http://oa.upm.es/21488/1/PFC_ALVARO_BUSTOS_BENAYAS.pdf

Crespo, X. (18 de enero de 2017). Qué es un FPGA y por qué jugarán un papel clave en el futuro [Mensaje en un blog]. Recuperado de https://planetachatbot.com/qu%C3%A9-es-una-fpga-y-por-qu%C3%A9-jugar%C3%A1n-un-papel-clave-en-el-futuro-e76667dbce3e

Crockett, L. H., Elliot, R. A., Enderwitz, M. A. y Northcote, D. (2015). *The Zynq Book Tutorials for Zybo and ZedBoard.* Recuperado de http://www.zynqbook.com/download-tuts.html

Digilent. (2018). *About Us*. Recuperado de https://store.digilentinc.com/about-us/

Digilent. (2018). *Pmod ACL: 3 axis Accelerometer*. Recuperado de https://store.digilentinc.com/pmod-acl-3-axis-accelerometer/

Digilent. (2018). *Vivado library/Pmod*ACL. Recuperado de https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodACL_v1_0

Digilent. (2018). *Vivado versión 2015.1 and later board file installation (Legacy)*. Recuperado de https://reference.digilentinc.com/reference/software/vivado/board-files?redirect=1

Digilent. (2018). *ZedBoard Zynq-7000 ARM/FPGA SoC Development Board.* Recuperado de https://store.digilentinc.com/zedboard-zynq-7000-arm-fpga-soc-development-board/

López, E. (s.f.). *Protocolo SPI(serial peripherical interface).* Recuperado de http://www.i-micro.com/pdf/articulos/spi.pdf

López. S. (2007). *Introducción al lenguaje de descripción hardware VHDL.* Recuperado de http://arantxa.ii.uam.es/~jgonzale/fco/curso07-08/download/seminarios-vhdl.pdf

Maiocchi, C. (20 de diciembre de 2013). ¿Qué es una FPGA? [Mensaje en un blog]. Recuperado de http://www.equaphon-university.net/que-es-un-fpga/

Marín, J. M. (s.f.). *Field Programmable Gate Array.* Recuperado de http://bibing.us.es/proyectos/abreproy/11375/fichero/MEMORIA%252FFPGAs.pdf

Martínez, R. y Romero, M. (2013). *Uso de sensores inerciales en la medición y evaluación de movimiento humano para aplicaciones en la salud.* Recuperado de

https://www.researchgate.net/publication/263198667_Uso_de_sensores_inerciales_en_la_medicion_y_evaluacion_de_movimiento_humano_para_aplicaciones_en_la_salud

Naqvi, F. (2015). Design and implementation of serial peripheral interface protocol using Verilog HDL. *IJEDR, 3*(3), 1-5. Recuperado de https://www.ijedr.org/papers/IJEDR1503092.pdf

Sánchez-Élez, M. (2014). Introducción a la programación en VHDL. Recuperado de http://eprints.ucm.es/26200/

Schweers, R. J. (2002). *Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC* (Tesis de grado). Universidad Nacional de la Plata, Buenos Aires, Argentina. Recuperado de http://sedici.unlp.edu.ar/handle/10915/3835

XILINX. (2018). *Descarga Vivado 2015.4.* Recuperado de https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

XILINX. (2018). *Vivado Design Suite Evaluation and WebPack.* Recuperado de https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html

APÉNDICES

APÉNDICE A. DISEÑO DEL HARDWARE EN VIVADO

```
--Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2015.4 (win64) Build 1412921 Wed Nov 18 09:43:45 MST 2015
--Date        : Mon Feb 19 21:19:23 2018
--Command     : generate_target design_2PmodACL_wrapper.bd
--Design      : design_2PmodACL_wrapper
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity design_2PmodACL_wrapper is
  port (
    DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
    DDR_ba : inout STD_LOGIC_VECTOR ( 2 downto 0 );
    DDR_cas_n : inout STD_LOGIC;
    DDR_ck_n : inout STD_LOGIC;
    DDR_ck_p : inout STD_LOGIC;
    DDR_cke : inout STD_LOGIC;
    DDR_cs_n : inout STD_LOGIC;
    DDR_dm : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_dq : inout STD_LOGIC_VECTOR ( 31 downto 0 );
    DDR_dqs_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_dqs_p : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_odt : inout STD_LOGIC;
    DDR_ras_n : inout STD_LOGIC;
    DDR_reset_n : inout STD_LOGIC;
    DDR_we_n : inout STD_LOGIC;
    FIXED_IO_ddr_vrn : inout STD_LOGIC;
    FIXED_IO_ddr_vrp : inout STD_LOGIC;
    FIXED_IO_mio : inout STD_LOGIC_VECTOR ( 53 downto 0 );
    FIXED_IO_ps_clk : inout STD_LOGIC;
    FIXED_IO_ps_porb : inout STD_LOGIC;
    FIXED_IO_ps_srstb : inout STD_LOGIC;
    ja_pin10_io : inout STD_LOGIC;
    ja_pin1_io : inout STD_LOGIC;
    ja_pin2_io : inout STD_LOGIC;
    ja_pin3_io : inout STD_LOGIC;
```

```
    ja_pin4_io : inout STD_LOGIC;
    ja_pin7_io : inout STD_LOGIC;
    ja_pin8_io : inout STD_LOGIC;
    ja_pin9_io : inout STD_LOGIC;
    jb_pin10_io : inout STD_LOGIC;
    jb_pin1_io : inout STD_LOGIC;
    jb_pin2_io : inout STD_LOGIC;
    jb_pin3_io : inout STD_LOGIC;
    jb_pin4_io : inout STD_LOGIC;
    jb_pin7_io : inout STD_LOGIC;
    jb_pin8_io : inout STD_LOGIC;
    jb_pin9_io : inout STD_LOGIC;
    leds_8bits_tri_io : inout STD_LOGIC_VECTOR ( 7 downto 0 )
  );
end design_2PmodACL_wrapper;

architecture STRUCTURE of design_2PmodACL_wrapper is
  component design_2PmodACL is
  port (
    DDR_cas_n : inout STD_LOGIC;
    DDR_cke : inout STD_LOGIC;
    DDR_ck_n : inout STD_LOGIC;
    DDR_ck_p : inout STD_LOGIC;
    DDR_cs_n : inout STD_LOGIC;
    DDR_reset_n : inout STD_LOGIC;
    DDR_odt : inout STD_LOGIC;
    DDR_ras_n : inout STD_LOGIC;
    DDR_we_n : inout STD_LOGIC;
    DDR_ba : inout STD_LOGIC_VECTOR ( 2 downto 0 );
    DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
    DDR_dm : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_dq : inout STD_LOGIC_VECTOR ( 31 downto 0 );
    DDR_dqs_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_dqs_p : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    FIXED_IO_mio : inout STD_LOGIC_VECTOR ( 53 downto 0 );
    FIXED_IO_ddr_vrn : inout STD_LOGIC;
    FIXED_IO_ddr_vrp : inout STD_LOGIC;
    FIXED_IO_ps_srstb : inout STD_LOGIC;
    FIXED_IO_ps_clk : inout STD_LOGIC;
    FIXED_IO_ps_porb : inout STD_LOGIC;
    leds_8bits_tri_i : in STD_LOGIC_VECTOR ( 7 downto 0 );
    leds_8bits_tri_o : out STD_LOGIC_VECTOR ( 7 downto 0 );
    leds_8bits_tri_t : out STD_LOGIC_VECTOR ( 7 downto 0 );
    ja_pin1_o : out STD_LOGIC;
    ja_pin7_i : in STD_LOGIC;
    ja_pin2_o : out STD_LOGIC;
    ja_pin8_i : in STD_LOGIC;
```

```
  ja_pin3_o : out STD_LOGIC;
  ja_pin9_i : in STD_LOGIC;
  ja_pin10_o : out STD_LOGIC;
  ja_pin4_o : out STD_LOGIC;
  ja_pin3_i : in STD_LOGIC;
  ja_pin4_i : in STD_LOGIC;
  ja_pin1_i : in STD_LOGIC;
  ja_pin2_i : in STD_LOGIC;
  ja_pin10_t : out STD_LOGIC;
  ja_pin8_t : out STD_LOGIC;
  ja_pin9_t : out STD_LOGIC;
  ja_pin4_t : out STD_LOGIC;
  ja_pin9_o : out STD_LOGIC;
  ja_pin10_i : in STD_LOGIC;
  ja_pin7_t : out STD_LOGIC;
  ja_pin1_t : out STD_LOGIC;
  ja_pin2_t : out STD_LOGIC;
  ja_pin7_o : out STD_LOGIC;
  ja_pin3_t : out STD_LOGIC;
  ja_pin8_o : out STD_LOGIC;
  jb_pin1_o : out STD_LOGIC;
  jb_pin7_i : in STD_LOGIC;
  jb_pin2_o : out STD_LOGIC;
  jb_pin8_i : in STD_LOGIC;
  jb_pin3_o : out STD_LOGIC;
  jb_pin9_i : in STD_LOGIC;
  jb_pin10_o : out STD_LOGIC;
  jb_pin4_o : out STD_LOGIC;
  jb_pin3_i : in STD_LOGIC;
  jb_pin4_i : in STD_LOGIC;
  jb_pin1_i : in STD_LOGIC;
  jb_pin2_i : in STD_LOGIC;
  jb_pin10_t : out STD_LOGIC;
  jb_pin8_t : out STD_LOGIC;
  jb_pin9_t : out STD_LOGIC;
  jb_pin4_t : out STD_LOGIC;
  jb_pin9_o : out STD_LOGIC;
  jb_pin10_i : in STD_LOGIC;
  jb_pin7_t : out STD_LOGIC;
  jb_pin1_t : out STD_LOGIC;
  jb_pin2_t : out STD_LOGIC;
  jb_pin7_o : out STD_LOGIC;
  jb_pin3_t : out STD_LOGIC;
  jb_pin8_o : out STD_LOGIC
);
end component design_2PmodACL;
component IOBUF is
```

```
port (
  I : in STD_LOGIC;
  O : out STD_LOGIC;
  T : in STD_LOGIC;
  IO : inout STD_LOGIC
);
end component IOBUF;
signal ja_pin10_i : STD_LOGIC;
signal ja_pin10_o : STD_LOGIC;
signal ja_pin10_t : STD_LOGIC;
signal ja_pin1_i : STD_LOGIC;
signal ja_pin1_o : STD_LOGIC;
signal ja_pin1_t : STD_LOGIC;
signal ja_pin2_i : STD_LOGIC;
signal ja_pin2_o : STD_LOGIC;
signal ja_pin2_t : STD_LOGIC;
signal ja_pin3_i : STD_LOGIC;
signal ja_pin3_o : STD_LOGIC;
signal ja_pin3_t : STD_LOGIC;
signal ja_pin4_i : STD_LOGIC;
signal ja_pin4_o : STD_LOGIC;
signal ja_pin4_t : STD_LOGIC;
signal ja_pin7_i : STD_LOGIC;
signal ja_pin7_o : STD_LOGIC;
signal ja_pin7_t : STD_LOGIC;
signal ja_pin8_i : STD_LOGIC;
signal ja_pin8_o : STD_LOGIC;
signal ja_pin8_t : STD_LOGIC;
signal ja_pin9_i : STD_LOGIC;
signal ja_pin9_o : STD_LOGIC;
signal ja_pin9_t : STD_LOGIC;
signal jb_pin10_i : STD_LOGIC;
signal jb_pin10_o : STD_LOGIC;
signal jb_pin10_t : STD_LOGIC;
signal jb_pin1_i : STD_LOGIC;
signal jb_pin1_o : STD_LOGIC;
signal jb_pin1_t : STD_LOGIC;
signal jb_pin2_i : STD_LOGIC;
signal jb_pin2_o : STD_LOGIC;
signal jb_pin2_t : STD_LOGIC;
signal jb_pin3_i : STD_LOGIC;
signal jb_pin3_o : STD_LOGIC;
signal jb_pin3_t : STD_LOGIC;
signal jb_pin4_i : STD_LOGIC;
signal jb_pin4_o : STD_LOGIC;
signal jb_pin4_t : STD_LOGIC;
signal jb_pin7_i : STD_LOGIC;
```

```
  signal jb_pin7_o : STD_LOGIC;
  signal jb_pin7_t : STD_LOGIC;
  signal jb_pin8_i : STD_LOGIC;
  signal jb_pin8_o : STD_LOGIC;
  signal jb_pin8_t : STD_LOGIC;
  signal jb_pin9_i : STD_LOGIC;
  signal jb_pin9_o : STD_LOGIC;
  signal jb_pin9_t : STD_LOGIC;
  signal leds_8bits_tri_i_0 : STD_LOGIC_VECTOR ( 0 to 0 );
  signal leds_8bits_tri_i_1 : STD_LOGIC_VECTOR ( 1 to 1 );
  signal leds_8bits_tri_i_2 : STD_LOGIC_VECTOR ( 2 to 2 );
  signal leds_8bits_tri_i_3 : STD_LOGIC_VECTOR ( 3 to 3 );
  signal leds_8bits_tri_i_4 : STD_LOGIC_VECTOR ( 4 to 4 );
  signal leds_8bits_tri_i_5 : STD_LOGIC_VECTOR ( 5 to 5 );
  signal leds_8bits_tri_i_6 : STD_LOGIC_VECTOR ( 6 to 6 );
  signal leds_8bits_tri_i_7 : STD_LOGIC_VECTOR ( 7 to 7 );
  signal leds_8bits_tri_io_0 : STD_LOGIC_VECTOR ( 0 to 0 );
  signal leds_8bits_tri_io_1 : STD_LOGIC_VECTOR ( 1 to 1 );
  signal leds_8bits_tri_io_2 : STD_LOGIC_VECTOR ( 2 to 2 );
  signal leds_8bits_tri_io_3 : STD_LOGIC_VECTOR ( 3 to 3 );
  signal leds_8bits_tri_io_4 : STD_LOGIC_VECTOR ( 4 to 4 );
  signal leds_8bits_tri_io_5 : STD_LOGIC_VECTOR ( 5 to 5 );
  signal leds_8bits_tri_io_6 : STD_LOGIC_VECTOR ( 6 to 6 );
  signal leds_8bits_tri_io_7 : STD_LOGIC_VECTOR ( 7 to 7 );
  signal leds_8bits_tri_o_0 : STD_LOGIC_VECTOR ( 0 to 0 );
  signal leds_8bits_tri_o_1 : STD_LOGIC_VECTOR ( 1 to 1 );
  signal leds_8bits_tri_o_2 : STD_LOGIC_VECTOR ( 2 to 2 );
  signal leds_8bits_tri_o_3 : STD_LOGIC_VECTOR ( 3 to 3 );
  signal leds_8bits_tri_o_4 : STD_LOGIC_VECTOR ( 4 to 4 );
  signal leds_8bits_tri_o_5 : STD_LOGIC_VECTOR ( 5 to 5 );
  signal leds_8bits_tri_o_6 : STD_LOGIC_VECTOR ( 6 to 6 );
  signal leds_8bits_tri_o_7 : STD_LOGIC_VECTOR ( 7 to 7 );
  signal leds_8bits_tri_t_0 : STD_LOGIC_VECTOR ( 0 to 0 );
  signal leds_8bits_tri_t_1 : STD_LOGIC_VECTOR ( 1 to 1 );
  signal leds_8bits_tri_t_2 : STD_LOGIC_VECTOR ( 2 to 2 );
  signal leds_8bits_tri_t_3 : STD_LOGIC_VECTOR ( 3 to 3 );
  signal leds_8bits_tri_t_4 : STD_LOGIC_VECTOR ( 4 to 4 );
  signal leds_8bits_tri_t_5 : STD_LOGIC_VECTOR ( 5 to 5 );
  signal leds_8bits_tri_t_6 : STD_LOGIC_VECTOR ( 6 to 6 );
  signal leds_8bits_tri_t_7 : STD_LOGIC_VECTOR ( 7 to 7 );
begin
design_2PmodACL_i: component design_2PmodACL
     port map (
      DDR_addr(14 downto 0) => DDR_addr(14 downto 0),
      DDR_ba(2 downto 0) => DDR_ba(2 downto 0),
      DDR_cas_n => DDR_cas_n,
      DDR_ck_n => DDR_ck_n,
```

```
DDR_ck_p => DDR_ck_p,
DDR_cke => DDR_cke,
DDR_cs_n => DDR_cs_n,
DDR_dm(3 downto 0) => DDR_dm(3 downto 0),
DDR_dq(31 downto 0) => DDR_dq(31 downto 0),
DDR_dqs_n(3 downto 0) => DDR_dqs_n(3 downto 0),
DDR_dqs_p(3 downto 0) => DDR_dqs_p(3 downto 0),
DDR_odt => DDR_odt,
DDR_ras_n => DDR_ras_n,
DDR_reset_n => DDR_reset_n,
DDR_we_n => DDR_we_n,
FIXED_IO_ddr_vrn => FIXED_IO_ddr_vrn,
FIXED_IO_ddr_vrp => FIXED_IO_ddr_vrp,
FIXED_IO_mio(53 downto 0) => FIXED_IO_mio(53 downto 0),
FIXED_IO_ps_clk => FIXED_IO_ps_clk,
FIXED_IO_ps_porb => FIXED_IO_ps_porb,
FIXED_IO_ps_srstb => FIXED_IO_ps_srstb,
ja_pin10_i => ja_pin10_i,
ja_pin10_o => ja_pin10_o,
ja_pin10_t => ja_pin10_t,
ja_pin1_i => ja_pin1_i,
ja_pin1_o => ja_pin1_o,
ja_pin1_t => ja_pin1_t,
ja_pin2_i => ja_pin2_i,
ja_pin2_o => ja_pin2_o,
ja_pin2_t => ja_pin2_t,
ja_pin3_i => ja_pin3_i,
ja_pin3_o => ja_pin3_o,
ja_pin3_t => ja_pin3_t,
ja_pin4_i => ja_pin4_i,
ja_pin4_o => ja_pin4_o,
ja_pin4_t => ja_pin4_t,
ja_pin7_i => ja_pin7_i,
ja_pin7_o => ja_pin7_o,
ja_pin7_t => ja_pin7_t,
ja_pin8_i => ja_pin8_i,
ja_pin8_o => ja_pin8_o,
ja_pin8_t => ja_pin8_t,
ja_pin9_i => ja_pin9_i,
ja_pin9_o => ja_pin9_o,
ja_pin9_t => ja_pin9_t,
jb_pin10_i => jb_pin10_i,
jb_pin10_o => jb_pin10_o,
jb_pin10_t => jb_pin10_t,
jb_pin1_i => jb_pin1_i,
jb_pin1_o => jb_pin1_o,
jb_pin1_t => jb_pin1_t,
```

```
      jb_pin2_i => jb_pin2_i,
      jb_pin2_o => jb_pin2_o,
      jb_pin2_t => jb_pin2_t,
      jb_pin3_i => jb_pin3_i,
      jb_pin3_o => jb_pin3_o,
      jb_pin3_t => jb_pin3_t,
      jb_pin4_i => jb_pin4_i,
      jb_pin4_o => jb_pin4_o,
      jb_pin4_t => jb_pin4_t,
      jb_pin7_i => jb_pin7_i,
      jb_pin7_o => jb_pin7_o,
      jb_pin7_t => jb_pin7_t,
      jb_pin8_i => jb_pin8_i,
      jb_pin8_o => jb_pin8_o,
      jb_pin8_t => jb_pin8_t,
      jb_pin9_i => jb_pin9_i,
      jb_pin9_o => jb_pin9_o,
      jb_pin9_t => jb_pin9_t,
      leds_8bits_tri_i(7) => leds_8bits_tri_i_7(7),
      leds_8bits_tri_i(6) => leds_8bits_tri_i_6(6),
      leds_8bits_tri_i(5) => leds_8bits_tri_i_5(5),
      leds_8bits_tri_i(4) => leds_8bits_tri_i_4(4),
      leds_8bits_tri_i(3) => leds_8bits_tri_i_3(3),
      leds_8bits_tri_i(2) => leds_8bits_tri_i_2(2),
      leds_8bits_tri_i(1) => leds_8bits_tri_i_1(1),
      leds_8bits_tri_i(0) => leds_8bits_tri_i_0(0),
      leds_8bits_tri_o(7) => leds_8bits_tri_o_7(7),
      leds_8bits_tri_o(6) => leds_8bits_tri_o_6(6),
      leds_8bits_tri_o(5) => leds_8bits_tri_o_5(5),
      leds_8bits_tri_o(4) => leds_8bits_tri_o_4(4),
      leds_8bits_tri_o(3) => leds_8bits_tri_o_3(3),
      leds_8bits_tri_o(2) => leds_8bits_tri_o_2(2),
      leds_8bits_tri_o(1) => leds_8bits_tri_o_1(1),
      leds_8bits_tri_o(0) => leds_8bits_tri_o_0(0),
      leds_8bits_tri_t(7) => leds_8bits_tri_t_7(7),
      leds_8bits_tri_t(6) => leds_8bits_tri_t_6(6),
      leds_8bits_tri_t(5) => leds_8bits_tri_t_5(5),
      leds_8bits_tri_t(4) => leds_8bits_tri_t_4(4),
      leds_8bits_tri_t(3) => leds_8bits_tri_t_3(3),
      leds_8bits_tri_t(2) => leds_8bits_tri_t_2(2),
      leds_8bits_tri_t(1) => leds_8bits_tri_t_1(1),
      leds_8bits_tri_t(0) => leds_8bits_tri_t_0(0)
    );
ja_pin10_iobuf: component IOBUF
     port map (
      I => ja_pin10_o,
      IO => ja_pin10_io,
```

```
      O => ja_pin10_i,
      T => ja_pin10_t
    );
ja_pin1_iobuf: component IOBUF
     port map (
      I => ja_pin1_o,
      IO => ja_pin1_io,
      O => ja_pin1_i,
      T => ja_pin1_t
    );
ja_pin2_iobuf: component IOBUF
     port map (
      I => ja_pin2_o,
      IO => ja_pin2_io,
      O => ja_pin2_i,
      T => ja_pin2_t
    );
ja_pin3_iobuf: component IOBUF
     port map (
      I => ja_pin3_o,
      IO => ja_pin3_io,
      O => ja_pin3_i,
      T => ja_pin3_t
    );
ja_pin4_iobuf: component IOBUF
     port map (
      I => ja_pin4_o,
      IO => ja_pin4_io,
      O => ja_pin4_i,
      T => ja_pin4_t
    );
ja_pin7_iobuf: component IOBUF
     port map (
      I => ja_pin7_o,
      IO => ja_pin7_io,
      O => ja_pin7_i,
      T => ja_pin7_t
    );
ja_pin8_iobuf: component IOBUF
     port map (
      I => ja_pin8_o,
      IO => ja_pin8_io,
      O => ja_pin8_i,
      T => ja_pin8_t
    );
ja_pin9_iobuf: component IOBUF
     port map (
```

```
      I => ja_pin9_o,
      IO => ja_pin9_io,
      O => ja_pin9_i,
      T => ja_pin9_t
    );
jb_pin10_iobuf: component IOBUF
     port map (
      I => jb_pin10_o,
      IO => jb_pin10_io,
      O => jb_pin10_i,
      T => jb_pin10_t
    );
jb_pin1_iobuf: component IOBUF
     port map (
      I => jb_pin1_o,
      IO => jb_pin1_io,
      O => jb_pin1_i,
      T => jb_pin1_t
    );
jb_pin2_iobuf: component IOBUF
     port map (
      I => jb_pin2_o,
      IO => jb_pin2_io,
      O => jb_pin2_i,
      T => jb_pin2_t
    );
jb_pin3_iobuf: component IOBUF
     port map (
      I => jb_pin3_o,
      IO => jb_pin3_io,
      O => jb_pin3_i,
      T => jb_pin3_t
    );
jb_pin4_iobuf: component IOBUF
     port map (
      I => jb_pin4_o,
      IO => jb_pin4_io,
      O => jb_pin4_i,
      T => jb_pin4_t
    );
jb_pin7_iobuf: component IOBUF
     port map (
      I => jb_pin7_o,
      IO => jb_pin7_io,
      O => jb_pin7_i,
      T => jb_pin7_t
    );
```

```
jb_pin8_iobuf: component IOBUF
     port map (
      I => jb_pin8_o,
      IO => jb_pin8_io,
      O => jb_pin8_i,
      T => jb_pin8_t
    );
jb_pin9_iobuf: component IOBUF
     port map (
      I => jb_pin9_o,
      IO => jb_pin9_io,
      O => jb_pin9_i,
      T => jb_pin9_t
    );
leds_8bits_tri_iobuf_0: component IOBUF
     port map (
      I => leds_8bits_tri_o_0(0),
      IO => leds_8bits_tri_io(0),
      O => leds_8bits_tri_i_0(0),
      T => leds_8bits_tri_t_0(0)
    );
leds_8bits_tri_iobuf_1: component IOBUF
     port map (
      I => leds_8bits_tri_o_1(1),
      IO => leds_8bits_tri_io(1),
      O => leds_8bits_tri_i_1(1),
      T => leds_8bits_tri_t_1(1)
    );
leds_8bits_tri_iobuf_2: component IOBUF
     port map (
      I => leds_8bits_tri_o_2(2),
      IO => leds_8bits_tri_io(2),
      O => leds_8bits_tri_i_2(2),
      T => leds_8bits_tri_t_2(2)
    );
leds_8bits_tri_iobuf_3: component IOBUF
     port map (
      I => leds_8bits_tri_o_3(3),
      IO => leds_8bits_tri_io(3),
      O => leds_8bits_tri_i_3(3),
      T => leds_8bits_tri_t_3(3)
    );
leds_8bits_tri_iobuf_4: component IOBUF
     port map (
      I => leds_8bits_tri_o_4(4),
      IO => leds_8bits_tri_io(4),
      O => leds_8bits_tri_i_4(4),
```

```
      T => leds_8bits_tri_t_4(4)
    );
leds_8bits_tri_iobuf_5: component IOBUF
     port map (
      I => leds_8bits_tri_o_5(5),
      IO => leds_8bits_tri_io(5),
      O => leds_8bits_tri_i_5(5),
      T => leds_8bits_tri_t_5(5)
    );
leds_8bits_tri_iobuf_6: component IOBUF
     port map (
      I => leds_8bits_tri_o_6(6),
      IO => leds_8bits_tri_io(6),
      O => leds_8bits_tri_i_6(6),
      T => leds_8bits_tri_t_6(6)
    );
leds_8bits_tri_iobuf_7: component IOBUF
     port map (
      I => leds_8bits_tri_o_7(7),
      IO => leds_8bits_tri_io(7),
      O => leds_8bits_tri_i_7(7),
      T => leds_8bits_tri_t_7(7)
    );
end STRUCTURE;
```

APÉNDICE B. CÓDIGO SDK

```
/************************************************************************/
/*                                                                      */
/*     main.c --     PmodACL Example Project                            */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/*     Author: Thomas Kappenman                                         */
/*     Copyright 2016-2017, Digilent Inc.                               */
/*                                                                      */
/************************************************************************/
/* Module Description:                                                 */
/*          This file contains code for running a demonstration of the  */
/*          PmodACL2 when used with the PmodACL2 IP core.               */
/*                                                                      */
/*          X, Y, and Z acceleration data in units of g are printed to  */
/*          a serial terminal application ten times per second.         */
/*          Baud rates to set up this connection should be 115200 for   */
/*          a Zynq device and whatever the AXI UART LITE IP is          */
/*          configured for for a Microblaze device - typically 9600 or  */
/*          115200 baud.                                                */
/*                                                                      */
/************************************************************************/
```

```
/*  Revision History:                                                     */
/*                                                                        */
/*      03/23/2016(TKappenman): Created                                   */
/*      08/15/2016(jpeyron): Sleep and Zynq include fixes                 */
/*      08/11/2017(artvvb): Validated for Vivado 2015.4                   */
/*                                                                        */
/**************************************************************************/

#include "xparameters.h"
#include "xil_cache.h"
#include "PmodACL.h"

#include <stdio.h>

#ifdef __MICROBLAZE__
#include "microblaze_sleep.h"
#else
#include <sleep.h>
#endif

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void DemoSleep(u32 millis);
void EnableCaches();
void DisableCaches();

PmodACL acl;
PmodACL acl1;

int main(void)
{
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize()
{
    EnableCaches();
    ACL_begin(&acl,
XPAR_PMODACL_0_AXI_LITE_GPIO_BASEADDR,XPAR_PMODACL_0_AXI_LITE_SPI_BASEADDR);
    ACL_begin(&acl1,
XPAR_PMODACL_1_AXI_LITE_GPIO_BASEADDR,XPAR_PMODACL_1_AXI_LITE_SPI_BASEADDR);
    ACL_SetMeasure(&acl, 0);
    ACL_SetMeasure(&acl1, 0);
    ACL_SetGRange(&acl, ACL_PAR_GRANGE_PM4G);
    ACL_SetGRange(&acl1, ACL_PAR_GRANGE_PM4G);
    ACL_SetMeasure(&acl, 1);
    ACL_SetMeasure(&acl1, 1);
    ACL_CalibrateOneAxisGravitational(&acl, ACL_PAR_AXIS_ZP);
    ACL_CalibrateOneAxisGravitational(&acl1, ACL_PAR_AXIS_ZP);
```

```
    DemoSleep(1000); // After calibration, some delay is required for the new settings to
take effect.
}

void DemoRun()
{
    float x, y, z;
    DemoSleep(2000);
    while (1)
    {
        ACL_ReadAccelG(&acl, &x, &y, &z);
        printf("\n\r------\r");
        printf("PmodACL 0 \r");
        printf("X=%f\tY=%f\tZ=%f\r", x, y, z);
        ACL_ReadAccelG(&acl1, &x, &y, &z);
        printf("PmodACL 1 \r");
        printf("X=%f\tY=%f\tZ=%f\r", x, y, z);
        printf("------\r");
        DemoSleep(2000);
    }
}

void DemoCleanup()
{
    DisableCaches();
}

void DemoSleep(u32 millis)
{
#ifdef __MICROBLAZE__
    MB_Sleep(millis);
#else
    usleep(1000 * millis);
#endif
}

void EnableCaches()
{
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches()
{
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_DCACHE
  Xil_DCacheDisable();
#endif
```

```
#ifdef XPAR_MICROBLAZE_USE_ICACHE
  Xil_ICacheDisable();
#endif
#endif
}
```

APÉNDICE C. RESULTADOS LECTURA SIMULTÁNEA DE DOS PMODACL

```
Connected to COM13 at 115200


------
PmodACL 0
X=-0.007812    Y=0.000000    Z=0.960938
PmodACL 1
X=-0.007812    Y=-0.046875    Z=1.000000
------
------
PmodACL 0
X=-0.015625    Y=0.000000    Z=0.960938
PmodACL 1
X=-0.007812    Y=-0.046875    Z=0.992188
------
------
PmodACL 0
X=-0.007812    Y=0.000000    Z=0.960938
PmodACL 1
X=-0.007812    Y=-0.046875    Z=0.992188
------
------
PmodACL 0
X=-0.015625    Y=0.000000    Z=0.960938
PmodACL 1
X=-0.007812    Y=-0.046875    Z=0.992188
------
------
PmodACL 0
X=0.039062     Y=-0.171875    Z=0.843750
PmodACL 1
```

```
X=0.062500     Y=-0.210938    Z=0.945312
------
------
PmodACL 0
X=0.257812     Y=-0.835938    Z=0.476562
PmodACL 1
X=0.273438     Y=-0.859375    Z=0.523438
------
------
PmodACL 0
X=0.265625     Y=-0.960938    Z=0.085938
PmodACL 1
X=0.273438     Y=-0.992188    Z=0.132812
------
------
PmodACL 0
X=0.125000     Y=-0.992188    Z=-0.015625
PmodACL 1
X=0.140625     Y=-1.015625    Z=0.015625
------
------
PmodACL 0
X=0.117188     Y=-0.984375    Z=-0.078125
PmodACL 1
X=0.132812     Y=-1.015625    Z=-0.023438
------
------
PmodACL 0
X=0.117188     Y=-0.898438    Z=0.375000
PmodACL 1
X=0.125000     Y=-0.937500    Z=0.414062
------
------
PmodACL 0
X=-0.070312    Y=-0.289062    Z=0.906250
PmodACL 1
```

```
X=-0.062500    Y=-0.335938    Z=0.953125
------
------
PmodACL 0
X=-0.281250    Y=0.890625     Z=0.351562
PmodACL 1
X=-0.304688    Y=0.851562     Z=0.406250
------
------
PmodACL 0
X=-0.109375    Y=1.000000     Z=0.078125
PmodACL 1
X=-0.132812    Y=0.953125     Z=0.117188
------
------
PmodACL 0
X=0.617188     Y=0.234375     Z=0.765625
PmodACL 1
X=0.632812     Y=0.226562     Z=0.757812
------
------
PmodACL 0
X=0.984375     Y=-0.078125    Z=-0.039062
PmodACL 1
X=0.976562     Y=-0.109375    Z=-0.015625
------
------
PmodACL 0
X=0.992188     Y=-0.039062    Z=-0.015625
PmodACL 1
X=0.992188     Y=-0.070312    Z=0.015625
------
------
PmodACL 0
X=1.000000     Y=-0.007812    Z=0.132812
PmodACL 1
```

```
X=1.007812    Y=-0.039062    Z=0.132812
------
------
PmodACL 0
X=0.171875    Y=-0.546875    Z=0.750000
PmodACL 1
X=0.179688    Y=-0.578125    Z=0.812500
------
------
PmodACL 0
X=0.015625    Y=0.750000    Z=0.679688
PmodACL 1
X=0.000000    Y=0.726562    Z=0.703125
------
------
PmodACL 0
X=0.054688    Y=0.375000    Z=0.843750
PmodACL 1
X=0.046875    Y=0.312500    Z=0.921875
------
------
PmodACL 0
X=0.023438    Y=-0.031250    Z=1.117188
PmodACL 1
X=-0.015625    Y=-0.109375    Z=1.312500
------
------
PmodACL 0
X=-0.015625    Y=0.000000    Z=0.960938
PmodACL 1
X=-0.007812    Y=-0.039062    Z=0.984375
------
------
PmodACL 0
X=-0.015625    Y=0.000000    Z=0.960938
PmodACL 1
```

```
X=-0.007812    Y=-0.046875    Z=0.992188
```

Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

FIRMA ESTUDIANTE

Oscar Quintero

FIRMA ASESOR

Luis Fernando Cortés Londoño

FECHA ENTREGA: 27/08/18

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

CHA ENTREGA: _____