Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

SISTEMAS ELECTRÓNICOS AVANZADOS

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN ANALIZADOR LÓGICO EMBEBIDO EN DISPOSITIVOS FPGA CON INTERFAZ DE COMUNICACIÓN ETHERNET

Estudiante *Murillo, Serrano, Dayanna E.*
Director *Zuloaga, Izaguirre, Aitzol*
Departamento
Curso académico *2020/2021*

Bilbao, 24 de junio de 2021

Resumen

El continuo desarrollo social y el avance tecnológico, han generado una fuerte y creciente necesidad de desarrollar sistemas hardware cada vez más complejos, que se ajusten a las características de los nuevos sistemas dentro de la industria, y que sean capaces de proporcionar todas las prestaciones que dichos sistemas requieren.

Gracias a los dispositivos FPGA (Field Programmable Gate Arrays), los diseñadores hardware son capaces de programar, probar, corregir y ampliar sus diseños, antes de ser fabricados y puestos en marcha, lo que a su vez proporciona mayor seguridad y garantía de un correcto funcionamiento.

Para facilitar la tarea de los diseñadores, los proveedores de estos dispositivos no solo han desarrollado varias herramientas software, a través de las cuales se puede realizar la programación y depuración del diseño. Sino que también han diseñado varias placas de desarrollo, sobre las cuales se pueden realizar las pruebas y ensayos, tanto de un diseño hardware como un diseño Software.

Una de las herramientas más utilizadas a la hora de realizar dichas pruebas en dispositivos FPGAs, son los analizadores lógicos embebidos o ILA (Integrated logic Analyzer). Estos analizadores lógicos permiten crear una conexión con un sistema de control para realizar la monitorización y depuración de las señales internas del diseño.

La conexión con dicho sistema de control se realiza a través de la interfaz JTAG Boundary Scan del dispositivo FPGA. Esta interfaz de comunicación serie es de corto alcance y requiere cortas distancias entre las placas de desarrollo y los sistemas de control. Esto supone un gran problema dentro de las necesidades de la sociedad actual, ya que impide a los diseñadores realizar pruebas y ensayos de forma remota desde cualquier parte del mundo y los limita también en aspectos como el tiempo y recursos.

Partiendo de esta base, en el presente proyecto se busca eliminar dichos problemas añadiendo una interfaz de comunicación Ethernet dentro de un diseño en FPGA, que sea capaz de conectarse al núcleo del depurador ILA para recoger y trasmitir los datos adquiridos, a un sistema de control de forma remota.

Para ello, se ha realizado un análisis e investigación de las herramientas de depuración que hay en el mercado actual, haciendo hincapié principalmente en las que ofrece el proveedor Xilinx: los núcleos de depuración LogiCore IP ILA y el LocgiCore IP Debug Bridge y la aplicación software xvcServer dentro de las herramientas de Petalinux.

Se han estudiado sus características y limitaciones, sus funcionalidades, las configuraciones y requerimientos necesarios, etc. para posteriormente combinarlas de tal forma que pueda crearse un

diseño sencillo en FPGA, capaz de cumplir con la necesidad de monitorizar y depurar el sistema de forma remota.

Del diseño obtenido se han realizado tres pruebas de conexión en entornos diferentes: Conexión directa a través de cable Ethernet, conexión remota por wifi en una misma red de área local y conexión remota a través de una red VPN (Virtual Private Network) por internet.

De cada prueba se han hecho varias depuraciones para observar y comparar las principales diferencias de cada entorno, teniendo en cuenta sobre todo, la estabilidad de la conexión, las velocidades de transmisión de datos y los posibles problemas que puedan llegar a ocurrir dentro de cada red.

Palabras Clave: FPGA, interfaz Ethernet, JTAG Boundary Scan, LogiCore IP ILA, LogiCore IP Debug Bridge, Petalinux, xvcServer, kernel de Linux, VPN.

Laburpena

Etengabeko garapen sozialak eta aurrerapen teknologikoak hardware sistema gero eta konplexuagoak garatzeko premia gero eta handiagoa sortu dute, industria barruko sistema berrien ezaugarrietara egokituko direnak eta sistema horiek eskatzen dituzten prestazio guztiak emateko gai izango direnak.

FPGA (Field Programmable Gate Arrays) gailuei esker, hardware diseinatzaileak gai dira beren diseinuak programatu, probatu, zuzendu eta zabaltzeko, fabrikatu eta martxan jarri aurretik, eta horrek, aldi berean, segurtasun eta berme handiagoa ematen du behar bezala funtzionatzeko.

Diseinatzaileen lana errazteko, gailu horien hornitzaileek software-tresna bat baino gehiago garatu dituzte, eta horien bidez, diseinua programatu eta araztu daiteke. Horrez gain, hainbat garapen-plaka diseinatu dituzte, eta horien gainean egin daitezke probak eta saiakuntzak, bai hardware-diseinu zein software-diseinu batean.

Proba horiek FPGAetan egiteko gehien erabiltzen diren tresnetako bat; analizatzaile logiko txertatuak edo ILA (Integrated logic Analizer) dira. Analizatzaile logiko horiei esker, konexio bat sor daiteke kontrol-sistema batekin, diseinuaren barne-seinaleak monitorizatzeko eta arazteko.

Kontrol-sistema horrekiko konexioa FPGA gailuaren JTAG Boundary Scan interfazearen bidez egiten da. Serieko komunikazio-interfaze hori irismen laburrekoa da, eta garapen-plaken eta kontrol-sistemen arteko konexioak distantzia laburrak eskatzen ditu. Hori arazo handia da gaur egungo gizartearen premien barruan, diseinatzaileei munduko edozein leku urrunetik probak eta saiakuntzak egitea eragozten dielako, eta denbora eta baliabide aldetik ere mugatzen dituelako.

Egoera horretatik abiatuz, proiektu honetan arazo horiek ezabatu nahi dira Ethernet interfazea gehituz FPGAko diseinu baten barruan, ILA araztegira konektatzeko eta eskuratutako datuak bildu eta urrutiko kontrol-sistema batera transmititzeko gai izan dadin.

Horretarako, gaur egungo merkatuan dauden arazketa-tresnak ikertu eta aztertu dira: ezaugarriak, mugak, funtzionalitateak, beharrezko konfigurazioak, eskakizunak, eta abar. Horrela, FPGAn diseinu erraz bat sortu ahal izango da, sistema urrutitik monitorizatzeko eta arazteko beharra betetzeko gai izango dena.

Lortutako diseinutik hiru konexio-proba egin dira ingurune desberdinetan: zuzeneko konexioa, Ethernet kablearen bidez; wifi bidezko urruneko konexioa, eremu lokaleko sare berean eta urruneko konexioa, Internet bidezko VPN sare baten bidez (Virtual Private Network).

Proba bakoitzetik hainbat arazketa egin dira ingurune bakoitzaren desberdintasun nagusiak behatzeko eta alderatzeko, batez ere kontuan hartuta konexioaren egonkortasuna, datuak transmititzeko abiadurak eta gerta daitezkeen arazoak.

Hitz Gakoak: FPGA, Ethernet interfazea, JTAG Boundary Scan, LogiCore IP ILA, LogiCore IP Debug Bridge, Petalinux, xvcServer, kernel de Linux, VPN.

Summary

The continuous social development and technological advancement has generated a strong and growing need to develop more and more complex hardware systems, which can adjust to the characteristics of one or more systems within the industry and are capable of providing all the features that such systems require.

Thanks to Field Programmable Gate Arrays devices or FPGAs, hardware designers are able to program, test, correct and expand their designs, even before being manufactured and commissioned, which in turn provides greater security and guarantee of correct operation.

To facilitate the task of designers, the suppliers of these devices have not only developed various software tools through which programming and debugging of the design can be carried out, but also they have designed several development boards, on which tests and trials can be carried out, both of a hardware design and a Software design.

One of the most used tools when performing such tests on FPGAs devices are the Integrated logic Analyzer, or ILA. These logic analyzers allow you to create a connection with a control system to perform the monitoring and debugging of the internal signals of the design.

The connection with this control system is made through the JTAG Boundary Scan interface of the FPGA device. This serial communication interface has a short-range and requires short distances between development boards and control systems. This is a big problem within the needs of nowadays society, since it prevents designers from carrying out tests and trials remotely from anywhere in the world and also limits them in aspects such as time or resources.

Based on this panorama, this project seeks to eliminate these problems by adding an Ethernet interface within an FPGA design, which can connect to the ILA debugger and collecting and transmitting the acquired data to a control system remotely.

For this purpose, an analysis and investigation of the debugging tools available in the current market has been carried out, emphasizing mainly those offered by the supplier Xilinx: the debugging cores LogiCore IP ILA and LogiCore IP Debug Bridge and the software application xvcServer within the Petalinux tools.

Their characteristics and limitations, functionalities, necessary configurations and requirements have been studied in order to later combine them in such a way that a simple FPGA design capable of fulfilling the need to monitor and debug the system remotely can be created.

From the design obtained, three connection tests have been carried out in different environments: Direct connection through Ethernet cable, remote connection by Wi-Fi in the same local area network and remote connection through a Virtual Private Network, VPN, by Internet.

Several debugging tests have been made to observe and compare the main differences of each environment, considering above all, the stability of the connection, the speed of data transmission and the different possible problems that may occur within each network.

Keywords: FPGA, Ethernet interface, JTAG Boundary Scan, LogiCore IP ILA, LogiCore IP Debug Bridge, Petalinux, xvcServer, Linux kernel, VPN.

Índice

Índice de tablas

Índice de Figuras

1. CONTEXTO

Una de las partes más importantes en un diseño electrónico, son las pruebas y ensayos que permitan comprobar el correcto funcionamiento del circuito diseñado.

Para la simulación y depuración de un diseño en FPGAs, existen varias herramientas que ayudan a localizar posibles errores dentro de la descripción HDL o en el propio diseño. Por un lado, se utilizan los módulos Testbench para realizar una simulación de las señales del circuito y, por otra parte, se usan los analizadores lógicos embebidos, ILA (Integrated Logic Analyzer), para capturar y mostrar señales reales durante la ejecución del diseño.

En ambos casos se puede obtener información del comportamiento del sistema para su posterior análisis [1], sin embargo, esta última herramienta tiene una ventaja frente a la primera, ya que facilita la tarea de encontrar y corregir posibles fallos que solo son visibles cuando el diseño se enfrenta a un entorno real.

Hoy en día, debido a la complejidad cada vez mayor de los chips, la depuración de diseños FPGA es una tarea cada vez más difícil y, en consecuencia, los proveedores de estos sistemas han desarrollado varias herramientas de depuración para entornos reales, los cuales están conformados por analizadores lógicos integrados, combinados con una herramienta de análisis de software [1]. Entre los analizadores lógicos más conocidos se tienen: SignalTap II [2] y FPGA View [3] para los dispositivos FPGA de Altera, FS2 Logic Navigator para las FPGAs de Actel [13] y, para los dispositivos de Xilinx, las herramientas de depuración más utilizadas son: ChipScope [4] en el entorno de ISE e Integrated logic Analyzer [5] para el entorno de Vivado.

Debido a la facilidad de uso y gracias a su entorno de diseño, el Integrated logic Analyzer, ILA, es el depurador más extendido en los últimos años. Esta herramienta se trata de un LogiCore IP de Xilinx que es proporcionado a los diseñadores junto con el software de Vivado [5], y cuya función es realizar la monitorización y depuración de las señales internas que el diseñador le indique [6].

A nivel de diseño, el ILA puede ser incorporado de forma muy sencilla, ya que se introduce al diseño como un Core o núcleo independiente y únicamente se requiere conectar todas las señales a depurar. Una vez la FPGA es programada, este Core establece una conexión automática entre la FPGA y un PC donde este instalado el software de control, en este caso, Vivado. El nivel de software también es de fácil uso, ya que se habilita de forma automática un panel de control a través del cual el usuario puede visualizar, controlar y configurar las opciones de depuración más adecuadas para su diseño [6].

A pesar de estas ventajas, el ILA cuenta con una limitación importante con respecto a su interfaz de comunicaciones. La conexión con el PC de control, creada por el ILA, se realiza a través de la interfaz

JTAG Boundary Scan del dispositivo FPGA [6]. Esta interfaz física usa un protocolo de comunicación serial y requiere cables de corta longitud para acceder al bus de transmisión JTAG [8].

Esta última característica impide realizar depuraciones de forma remota, limitando a los diseñadores en aspectos como el tiempo, lugar o recursos. Como problema añadido, la situación de la Covid 19 agrava estas limitaciones provocando muchas veces que, proyectos y e investigaciones se hayan visto interrumpidas debido a la imposibilidad de acceder a los equipos de trabajo.

Este problema no solo se aplica al ILA de Xilinx, sino que también a la gran mayoría de dispositivos FPGA, puesto que es muy común que la arquitectura de estos sistemas tenga una única interfaz para la comunicación y la depuración que por lo general es JTAG.[8]

La Implementación de un IP Core que cuente con una interfaz de comunicación de largo alcance, como por ejemplo Ethernet, y que sea capaz de acceder a los datos del analizador lógico y enviarlos al PC de control, eliminaría la necesidad de un conector JTAG y permitiría a los diseñadores utilizar una única interfaz para la comunicación y la depuración mientras se mantiene la accesibilidad a través de una red. Esto supondría una gran ventaja y comodidad para los diseñadores y permitiría acelerar el avance tecnológico con dispositivos FPGA.

2. SOLUCION PLANTEADA

Como se explicó anteriormente, problema de las limitaciones en la interfaz de comunicación no solo se aplica al ILA, sino que también a varias otras herramientas de depuración. Esto se debe a que JTAG es la interfaz más utilizada y extendida tanto en la programación como en la depuración de sistemas embebidos [8] y la gran mayoría de herramientas de depuración lo utilizan como interfaz de comunicación [8].

Teniendo en cuenta lo anterior, en este documento se buscará una solución en concreto para el analizador lógico en auge actualmente, el LogiCore IP ILA de Xilinx.

Al igual que el software de Vivado, el ILA de Xilinx es una herramienta gratuita, sin embargo, su diseño interior no está disponible a los usuarios y, por tanto, no es posible modificarlo o añadirle nuevas funcionalidades o interfaces de comunicación. Sin embargo, Xilinx ofrece el LogiCore IP Debug Bridge como solución para este problema.

Debug Bridge es un IP Core gratuito que funciona a modo de controlador y ofrece múltiples opciones para comunicarse con uno o varios núcleos de depuración de Xilinx dentro de un diseño [5]. En resumen, este Core puede ser utilizado a modo de puente para conectar las FPGA de Xilinx y el PC de control, de tal forma que sea posible depurar diseños remotamente a través de Ethernet u otras interfaces sin la necesidad de un cable JTAG [5].

Como es obvio, para la puesta en marcha de este Core, es necesario un software o driver para comunicaciones Ethernet, que controle los paquetes de transmisión TCP/IP entre ambos dispositivos, este software estaría corriendo dentro del subsistema de procesamiento, PS, de las placas, mientras se realiza la depuración del diseño en la FPGA. Este software recibe el nombre de “XVC Server app” [12] y se trata de una aplicación desarrollada con las herramientas de Petalinux y es compatible con todas las familias de FPGA de Xilinx. A continuación, en Figura 1 se muestra un diagrama general de todas las conexiones, tanto dentro de la placa FPGA como con el PC de control.

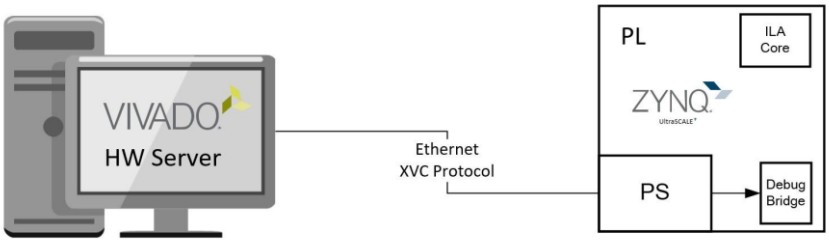

Figura 1: Diagrama de conexión PC Control – FPGA [12]

A pesar de la gran ventaja que Debug Bridge ofrece, este no es muy conocido y su uso no está muy extendido entre los diseñadores. Esto probablemente se debe a que no se tiene mucha documentación

de los pasos a seguir y de las configuraciones necesarias para poder utilizar este IP, además de esto, esta herramienta únicamente es compatible con ciertas familias de placas. Por esta razón, en el presente documento, se realiza un estudio de esta herramienta y se datan todos los requerimientos y configuraciones necesarias para poner a prueba su funcionamiento en diferentes entornos.

3. OBJETIVO PRINCIPAL

El objetivo principal del presente trabajo es **diseñar un analizador lógico que pueda ser embebido en un dispositivo FPGA, que permita capturar y extraer las señales lógicas internas del diseño, para posteriormente enviarlas a través de una interfaz Ethernet** con la finalidad de observar las mismas en sistemas remotos.

Para cumplir con este objetico se utilizará la herramienta LogiCore IP Debug Bridge, para establecer el enlace entre el analizador lógico de Xilinx, LogiCore IP ILA y un PC de control. Se analizarán las limitaciones tanto del Analizador lógico como del Debug Bridge y se pondrá a prueba el funcionamiento de los mismos en una plaza FPGA Zybo 7000. Se realizarán depuraciones de un mismo diseño de pruebas en varios entornos: con conexión directa de los dispositivos a través de cable Ethernet, con conexión remota por wifi en una misma red local y por conexión remota a través de Internet.

4. INVESTIGACION Y ANALISIS DEL ESTADO DEL ARTE

En el mercado actual se tienen diferentes opciones de analizadores lógicos para FPGAs que han sido lanzados por los proveedores de estos dispositivos y servirían como alternativa a las herramientas de Xilinx.

El competidor principal de Xilinx es Altera Corporation (desde 2015 parte de Intel) [10], cuya herramienta de depuración es el analizador lógico integrado SignalTap II.

SignalTap II es gratuito y está disponible a los diseñadores como un paquete independiente o incluido con el software Quartus II [2]. Esta herramienta de depuración tiene una gran ventaja añadida frente a sus competidores, ya que captura y muestra el comportamiento de la señal en tiempo real [24], ya sea para sistemas en un chip programable (SOPC) o para cualquier diseño de FPGA [2] [24]. Además de esto, el analizador lógico SignalTap II admite la mayor cantidad de canales, la mayor profundidad de muestra y las velocidades de reloj más rápidas de cualquier analizador lógico en el mercado de la lógica programable [2][24]. La Figura 2 muestra un diagrama de bloques de los componentes que componen el analizador lógico SignalTap II.

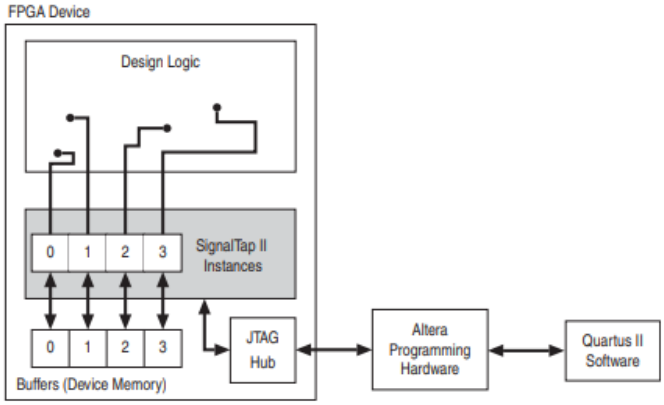

Figura 2: Diagrama de bloque de SignalTap II [2]

Tal y como se observa en la Figura 1, SignalTap II también utiliza una interfaz JTAG para la comunicación entre el PC de control y la FPGA teniendo el mismo problema de la conectividad remota. La solución de Altera es muy parecida al Debug Bridge de Xilinx, la cual se puede observar en la Figura 3.

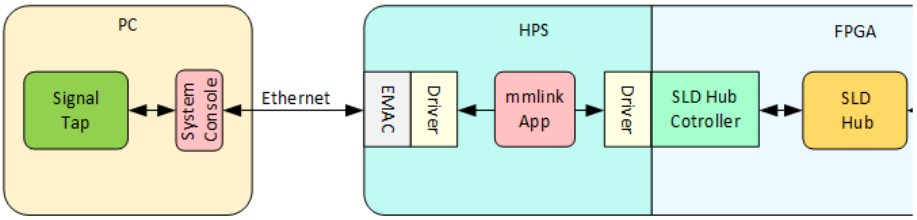

Figura 3: Solución de Altera para las depuraciones remotas con SignalTap [11]

Altera proporciona el "SLD Hub Controller" como adaptador o puente para que el analizador lógico pueda soportar conexiones Ethernet. A esta herramienta se le suma el conjunto HPS que es un

controlador software para el kernel de Linux, que se encarga de la trasmisión de los paquetes TCP/IP entre ambos dispositivos [11].

Otra alternativa para la depuración remota en dispositivos de Altera es el software FPGA View. La ventaja de FPGAView es que se pueden medir señales rápida y fácilmente dentro del diseño de la FPGA y seleccionar qué grupo de señales internas sondear sin tener que volver a compilar el diseño [3]. Por otro lado, la desventaja de esta herramienta es que no utiliza analizadores lógicos integrados, sino que funciona con la serie de analizadores lógicos Tektronix TLA y la serie MSO4000 de osciloscopios de señal mixta, para permitir la depuración en tiempo real de los dispositivos FPGA [3]. La depuración remota es posible utilizando los equipos TLA, pero eso supone añadir más recursos y nuevos gastos que los diseñadores deben soportar.

Tal como se observa, Altera ofrece diferentes herramientas para solventar el problema de las depuraciones remotas. Sin embargo, para este proyecto se han descartado estas opciones teniendo en cuenta que:

- La herramienta FPGAView tiene como ventaja el ahorro de tiempo ya que elimina la tarea de recompilar el diseño y reprogramar las placas cada vez que se desee añadir nuevas señales de depuración. Sin embargo, para su funcionamiento requiere de equipos electrónicos innecesarios y de gastos añadidos que podrían evitarse utilizando analizadores lógicos integrados.
- La herramienta SDL Hub Controller es la opción más lógica teniendo en cuenta las ventajas de tiempo real que tiene el analizador lógico de Altera. Sin embargo, el uso del SDL Hub Controller eliminaría dichas ventajas, ya que se añadirían más elementos al diseño que retrasarían la velocidad de depuración. Así mismo, esta solución es muy parecida a la herramienta Debug Bridge de Xilinx y, si se tiene en cuenta el hecho de que Xilinx es la compañía más grande y más extendida en el mercado de las FPGA, sería preferible utilizar las herramientas de Xilinx para acceder y comprender mejor el entorno de las familias de dispositivos que esta compañía ofrece.

Por estas razones, se ha optado por utilizar las herramientas de Xilinx para realizar el diseño de este proyecto. A continuación, se analizan y describen las 3 herramientas principales que se utilizaran para el diseño del circuito: El LogiCore IP ILA, LogicCore IP Debug Bridge, y la aplicación software XVC server.

4.1. LogiCore IP ILA

El analizador lógico integrado ILA es un IP Core diseñado por Xilinx y dirigido a sus dispositivos PFGA, que permite depurar y monitorear las señales internas de un diseño, activar eventos hardware y capturar datos a velocidad del sistema [5].

La última versión de este Core es la V6.2 [6] que fue lanzada en octubre de 2016 [5]. Entre las principales características que tiene este depurador están:

- El número y el tamaño de las señales monitorizadas son configurables por el usuario. [6] pudiendo monitorizar hasta 1024 señales, cada una con una amplitud de bits que va desde 1 a 4096 [6].
- Se puede combinar múltiples señales monitorizadas en una sola condición de disparo [6]
- Permite depurar IP Cores AXI [6]

Se puede usar este Core añadiéndolo a un diseño RTL o se puede insertado luego de la síntesis en diagrama de flujo de diseño de vivado (Vivado desing Flow) [5]. La comunicación con el ILA se realiza automáticamente gracias al centro de depuración de instancia automático que conecta el Core a la interfaz JTAG de la FPGA [6], siendo esta la única interfaz de comunicaciones que tiene.

Las señales atribuidas con depuración se conectan al ILA tal como se muestran en la figura 1. Internamente estas señales se conectan al comparador de disparo (Trigger Comparator) del ILA, el cual es capaz de realizar diferentes operaciones [6]. La velocidad de muestreo de las señales dependerá de la configuración de reloj que se tenga en el diseño ya que esta se aplicará también al reloj interno del ILA [5]. Las muestras se almacenan en un bloque de RAM en chip (BRAM) hasta que el software los cargue [6] para ser visualizados por el usuario a través de Vivado logic Analyzer.

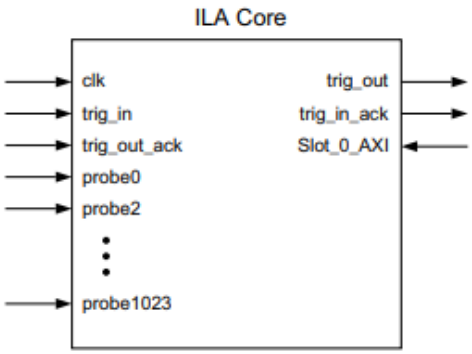

Figura 4: ILA CORE [6]

La configuración del comparador de disparo se realiza durante el tiempo de ejecución [6] y las opciones de captura van desde: detección de transiciones de flanco, como flanco ascendente (R), flanco descendente (F), flanco (B) o sin transición (N). Comparaciones de valor como: mayor que (>), menor que (<), mayor o igual que (≥), menor o igual que (≤) y comparaciones más complejas que incluyen patrones como, por ejemplo: igual que (=) X0XX100 o diferente que (!=) X0XX101.

Adicionalmente, se pueden realizar las capturas combinando diferentes señales, mediante la configuración de la condición de disparo (ILA Trigger Condition) [6]. Esta condición es el resultado de un cálculo booleano "AND" u "OR" de cada uno de los resultados del comparador de disparo del ILA. El ajuste "AND" provoca un disparo de activación cuando se satisfacen todas las comparaciones, mientras que el ajuste "OR" provoca un disparo de activación cuando se satisface cualquiera de las comparaciones [1].

A continuación, se muestra la tabla 1 de parámetros configurables dada por Xilinx de este IP Core, en la que también se describe brevemente la función de cada uno de ellos.

Tabla 1 Parámetros configurables del LogiCore IP ILA [6]

Parameter Name	Allowable Values	Default Value	Description
Component_Name	String with A–Z, 0–9, and _ (underscore)	ila_0	Name of instantiated component.
C_NUM_OF_PROBES	1–1,024	1	Number of ILA probe ports.
C_DATA_DEPTH	1,024, 2,048, 4,096, 8,192, 16,384, 32,768, 65,536, 131,072	1,024	Probe storage buffer depth. This number represents the maximum number of samples that can be stored at run time for each probe input.
C_PROBE<n>_WIDTH	1–4,096	1	Width of probe port <n>. Where <n> is the probe port having a value from 0 to 1,023.
C_TRIGOUT_EN	True/False	FALSE	Enables the trig out functionality. Ports trig_out and trig_out_ack are used.
C_TRIGIN_EN	True/False	FALSE	Enables the trig in functionality. Ports trig_in and trig_in_ack are used
C_INPUT_PIPE_STAGES	0–6	0	Add extra flops to the probe ports. One parameter applies to all of the probe ports.
C_EN_STRG_QUAL	0, 1	0	Enable the Capture (Storage) Qualifier. By enabling this you can specify the capture condition in Vivado Logic Analyzer thus capture the probes selectively. This takes one extra compare values (match) unit. This means if advance trigger (C_ADV_TRIGGER) option is enabled, the maximum number of match units per probes reduces to three from four.
C_ADV_TRIGGER	True/False	FALSE	Enables the advance trigger option. This enables trigger state machine and you can write your own trigger sequence in Vivado Logic Analyzer.
ALL_PROBE_SAME_MU	True/False	TRUE	This forces the same compare value units (match units) to all of the probes.
C_PROBE<n>_MU_CNT	1–16	1	Number of Compare Value (Match) units per probe. This is valid only if ALL_PROBE_SAME_MU is FALSE.

Tabla 1 Parámetros configurables del LogiCore IP ILA [6]

Parameter Name	Allowable Values	Default Value	Description
C_PROBE<n>_TYPE	DATA and TRIGGER, TRIGGER, DATA	DATA and TRIGGER	To choose a selected probe for specifying trigger condition or for data storage purpose or for both.

En cuanto a las limitaciones de este Core, las más destacables son:

a) **Reloj:** Es recomendable utilizar el mismo reloj tanto para el diseño como para el ILA. Esto implica que velocidad de muestreo se ve afectada por las configuraciones que el diseñador aplique al reloj del sistema implementado [6]. En caso de que no se use el mismo reloj para el diseño y para el ILA, el reloj debe cumplir ciertos requisitos

- Debe estar a la misma frecuencia que el reloj del sistema. De lo contrario puede causar diversos fallos como violaciones de retención [6][9].
- El reloj usado para el ILA debe ser un reloj "LIBRE". Un reloj libre es un reloj que no deja de funcionar (es decir, el reloj no está bloqueado en fase para otras fuentes de reloj). Ejemplos de relojes libres son GT TXOUTCLK, RXOUTCLK, RXRECCLK. [6]

b) **Limitaciones de velocidad:** las limitaciones del reloj, sumado al tiempo que el software tarda en cargarlos datos, impiden que el ILA pueda ser utilizado en aplicaciones que requieren depuraciones en Real Time.

El problema se agrava cuando se trata de sumarle una interface de comunicaciones Ethernet como el Debug Bridge ya que esto agregaría un paso más (y otro retraso) en la depuración del diseño.

Por este motivo, actualmente existen proyectos de investigación y desarrollo de analizadores lógicos embebidos ultrarrápidos, que cuenten con interface de comunicaciones Ethernet y sean capaces de cubrir las necesidades de Real Time. Un ejemplo de proyecto es *"Time-multiplexed 10Gbps Ethernet-based Integrated Logic Analyzer for FPGAs" [9]* el cual comenzó en la cuarentena del 2020 y continua en desarrollo.

c) **Numero de Señales y amplitudes limitados:** El LogiCore IP ILA actualmente pueden muestrear ventanas relativamente pequeñas de un número limitado de señales del diseño implementado. Como antes se vio, únicamente puede tener hasta 1024 (2^{10}) señales a

depurar, cada una de 1 a 4096 (2^{12}) bits de ancho, equivalente a un número máximo de 2^{22} señales de 1 bit [5][8].

Para diseños sencillos y con pocas señales esta capacidad sería suficiente, pero, los diseños más grandes y complejos pueden llegar a necesitar más de in ILA, con lo que también se complicaría un poco más las configuraciones durante la depuración.

4.2. LogiCore IP Debug Bridge

El LogiCORE IP Debug Bridge es un controlador proporcionado por Xilinx, que permite establecer un canal de comunicación con los IP Cores de depuración (ILAs) de un diseño [7].

Debug Bridge puede funcionar de dos formas: *Tandem with Field Updates* y Xilinx Virtual Cable *XVC* [7]. Tandem with Field Updates permite agregar nuevas funcionalidades a un diseño a través de un enlace PCIe. Está destinado a diseños con reconfiguraciones parciales, y se usa para agregar una instancia de Debug Bridge, en cada módulo reconfigurable del diseñador que estén conectados a algún IP core de depuración de Xilinx como: ILA, VIO, Memory IP y JTAG2AXI [7].

El modo Xilinx Virtual cable, XVC, permite utilizar dichos IP Cores de depuración y realizar la depuración del diseño a través de una interfaz que no es JTAG (por ejemplo, Ethernet / PCIe). Este modo de funcionamiento tiene 5 opciones de configuración, las cuales se clasifican en:

- **From_AXI_to_BSCAN:** Conecta el Debug Bridge con un maestro Ethernet/PCIe dentro del diseño, a su vez, se conecta a través de BSCAN con los depuradores que haya en el diseño como: ILA, Memory IP, JTAG2AXI. Este modo está pensado para realizar las depuraciones en la misma placa [7].

- **From_AXI_to_JTAG:** Este modo está pensado para depurar un diseño en otra placa FPGA sobre VXC. Para se conecta el Debug bridge con un algún maestro Ethernet/PCIe, dentro del diseño, a su vez, el Debug bridge ira sacando los pines JTAG a través de los pines E/S para que puedan conectarse a los pines JTAG de otra placa. [7]

- **From_JTAG_to_BSCAN:** Permite depurar los diseños a través del controlador Soft Test Access Port (TAP).[7]

- **From_PCIE_to_BSCAN** y **From_PCIE_to_JTAG:** (solo para la familia UltraScale+™ y UltraScale™). Cumple la misma función que From_AXI_to_BSCAN y From_AXI_to_BSCAN

respectivamente. Con la diferencia que conecta el Debug Bridge a maestro PCIe dentro del diseño.[7]

En este caso, para realizar la depuración remota a través de una interfaz Ethernet se utiliza la configuración From_AXI_to_BSCAN. Cuyos pines de entrada se aprecian en la Figura 5:

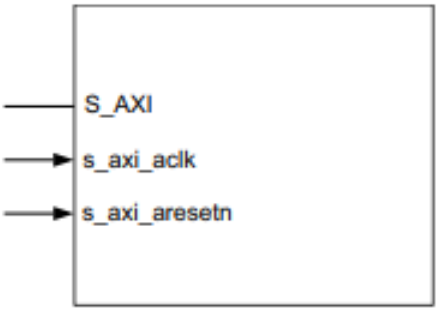

Figura 5: Debug Bridge configuración modo From AXI to BSCAN [7]

Así mismo, en la Figura 6 se puede observar cual sería el diagrama de comunicaciones completo en este modo de configuración. En resumen, la parte PS de la placa y el depurador ILA estarán conectados y comunicándose a través del puente de depuración o Debug Bridge. Por un lado, PS establecerá una interconexión AXI con el Debug Bridge a través de un controlador o AXI Master. De igual forma, Debug Bridge establecerá otra rama de interconexión con el depurador ILA a través de BSCAN. Como se observa en la imagen, el diseño puede contar con más de un Core de depuración, en cuyo caso se utiliza un IP Core Debug Hub como centro de conexión de los ILAs y al que estaría conectado el puente de depuración.

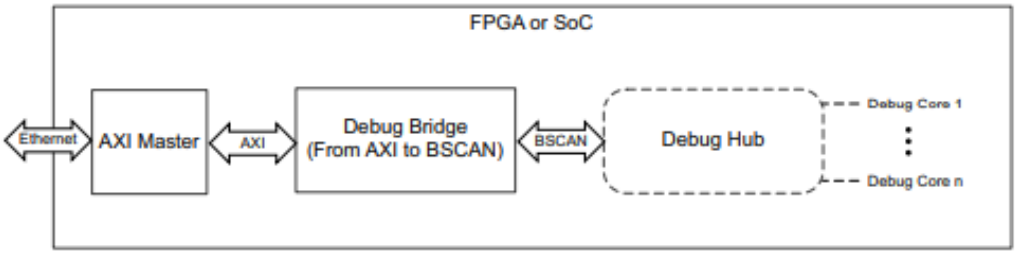

Figura 6: Diagrama de comunicaciones con Debug Bridge [7]

4.3. Linux Kernel - App xvcServer

El kernel Linux es el núcleo del sistema operativo GNU/Linux creado originalmente como un clon del sistema operativo Unix. [19] Se trata de un programa modular que proporciona un entorno multitarea y multiusuario con características como:

- Capacidad para funcionar sobre múltiples arquitecturas incluyendo las arquitecturas ARM de 32-bits y 64-bits. [18][19]
- Capacidad de funcionamiento sobre plataformas con poca memoria y poca capacidad de almacenamiento, dependiendo de los servicios y programas de usuario quiera usar. [18][19]

- Cuenta con varios drivers para una gran cantidad de dispositivos hardware y tiene posibilidad de añadir drivers propios de diseñadores para para manejar IPs en PL.
- Soporta multitud de buses de comunicación: I2C, SPI, CAN, USB… así como redes de ordenadores: ethernet, wifi, bluetooth… y otras capacidades de red como: TCP/IP, Firewall, Advanced Routing… [18][19]
- Proporciona a los programas de usuario un conjunto de APIs o llamadas al sistema independientes del hardware.

xvcServer es una aplicación para los sistemas Linux embebidos dirigidos a los dispositivos FPGA de Xilinx. Esta aplicación utiliza el protocolo Xilinx Virtual Cable (XVC) (basado en TCP/IP), para actuar como un cable JTAG y proporcionar un medio para acceder y depurar un diseño de FPGA o SoC de forma remota. [14]

A continuación, se describen las principales características de esta aplicación:

a. Establecimiento de comunicación:

El protocolo XVC es utilizado por un cliente y servidor TCP/IP para transferir vectores JTAG de bajo nivel, desde una aplicación de alto nivel (por ejemplo, las herramientas de diseño de Xilinx como Vivado o ISE) a un dispositivo FPGA o SoC. [14]

EL protocolo de comunicación inicia cuando el cliente se conecta a un servidor XVC utilizando métodos de conexión cliente/servidor TCP/IP estándar, una vez conectado, el cliente emite un mensaje XVC (getinfo:) al servidor solicitando la versión del servidor.

Una vez que el cliente valida la versión del protocolo, se envía un nuevo mensaje al servidor XVC para establecer la tasa de tck de JTAG para futuras operaciones de turno.

El mensaje de cambio, “shift”, es el comando principal utilizado entre el cliente y el servidor para transferir vectores JTAG de bajo nivel. El cliente emitirá operaciones shift para determinar la composición de la cadena JTAG y luego ejecutará varias instrucciones JTAG, por ejemplo, conducir pines o programar un dispositivo.[14]

b. Protocolo

El protocolo de comunicación XVC 1.0 consta de tres mensajes, getinfo, settck y shift. Para cada mensaje, se espera que el cliente envíe el mensaje y espere una respuesta del servidor. El servidor debe procesar cada mensaje en el orden en que los recibió y proporcionar una respuesta de inmediato.

Para evitar problemas de bloqueo y entrelazado que pueden ocurrir con la comunicación simultánea del cliente, el protocolo XVC solo asume una conexión.[14]

Estos tres mensajes principales contienen la siguiente información:

- **getinfo:** Mensaje que el cliente envía para obtener la versión del servidor XVC. La versión del servidor proporciona al cliente una forma de determinar las capacidades de protocolo del servidor.

  El mensaje de respuesta que el servidor enviará será ***<xvc_vector_len>\n"*** que es el ancho máximo del vector que se puede shiftear al servidor.[14]

- **settck:<periodo en ns>** configura el período TCK del servidor. Al enviar vectores JTAG, es posible que sea necesario variar la velocidad TCK para adaptarse a las condiciones de integridad de la señal del cable y la placa. Los clientes utilizan este comando para ajustar la tasa TCK con el fin de ralentizar o acelerar el desplazamiento de los vectores JTAG.[14]

- **shift:<num bits> <vector tms > <vector tdi>** El mensaje "shift:" se usa para cambiar los vectores JTAG dentro y fuera de un dispositivo. El número de bits a desplazar se especifica como el primer parámetro de comando de desplazamiento seguido de los vectores de datos TMS y TDI. Los vectores TMS y TDI se dimensionan de acuerdo con el número de bits a desplazar, redondeados al byte más cercano. Por ejemplo, si se desplaza en 13 bits, los vectores de bytes se redondearán a 2 bytes. Una vez completada la operación de cambio de JTAG, el servidor devolverá un vector de tamaño de byte que contiene el valor de TDO objetivo muestreado para cada reloj TCK cambiado.[14]

En cuanto a la versatilidad de esta aplicación, xvcServer está diseñada para ser compatible con casi todos los dispositivos FPGA y SoC de Xilinx, y únicamente requiere la instalación y ejecución de un sistema Linux en PS de las placas [14] de modo que xvcServer pueda aprovechar el stack TCP/IP incluida en el kernel de Linux [14]. Sin embargo, dependiendo de la placa que tenga el diseñador, se tendrían que realizar unos pequeños cambios en el código main del archivo principal c.

En resumen, el protocolo XVC define un método de comunicación JTAG simple con capacidades suficientes para clientes de alto nivel, como Xilinx Vivado y herramientas SDK, para realizar funciones complejas como programación y depuración de dispositivos.[14]

5. DISEÑO Y MONTAJE

En este punto se describen de forma detalla los pasos seguidos para completar un diseño en FPGA, que pueda ser depurado remotamente a través de una interfaz Ethernet. Para ello, el proyecto se ha dividido en dos niveles: Nivel Hardware, en donde se plantea las configuraciones necesarias tanto de la placa y de los IP Cores que se utilizarán, así como las características del diseño propio que se quiere depurar. Nivel Software, en donde se describen los requerimientos de la herramienta Petalinux y los comandos necesarios para crear y construir un proyecto Petalinux compatible con la placa de pruebas.

5.1. Nivel Hardware

Para el diseño hardware se ha realizado utilizando las siguientes herramientas:

- **Zybo Board** cuya familia FPGA es Zynq 7000
- **Board Files** para Zybo de Diligent Reference https://reference.digilentinc.com/software/vivado/board-files?redirect=1 [15]
- **Vivado Desing Suite v2018.3**

En la Figura 7 se observa el diagrama de bloques del diseño completo. Se podría dividir el diseño en tres partes importantes: PS desde donde se realizará la comunicación con el PC de control, Debug Cores que engloba al depurador ILA y al puente de depuración o Debug Bridge y, por último, User-Core que sería el diseño propio que se quiere depurar.

A continuación, se describen los pasos seguidos en cada una de las partes del proyecto.

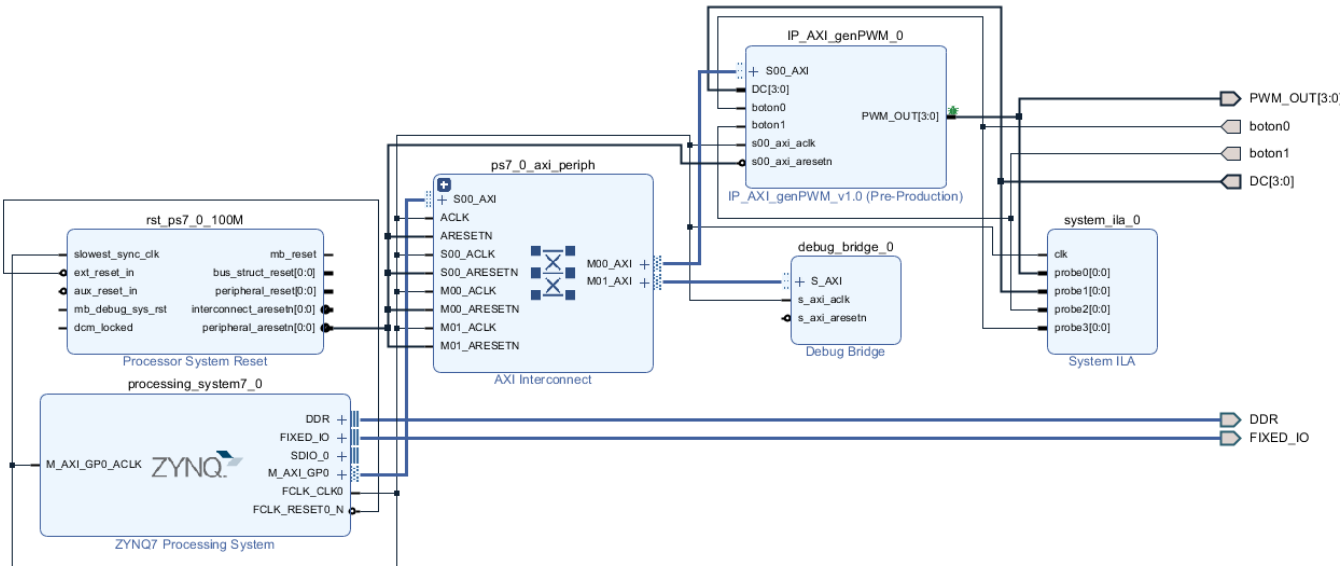

Figura 7: Diagrama de Bloques completo del diseño

Como se muestra en la Figura 8, el proyecto en Vivado se ha se ha creado desde cero usando como base los “boards files” de Diligent que contienen las características de la placa Zybo, estos archivos fueron descargados desde la página oficial de Diligent Referece.

Cabe destacar que es importante generar el proyecto en base a estos “board files” de modo que, cuando se exporte la plataforma de hardware para herramientas de desarrollo de software, se envíen todos los datos y características de la placa que se está utilizando. Esto servirá posteriormente para la creación del proyecto Petalinux asegurando una mayor compatibilidad con la placa.

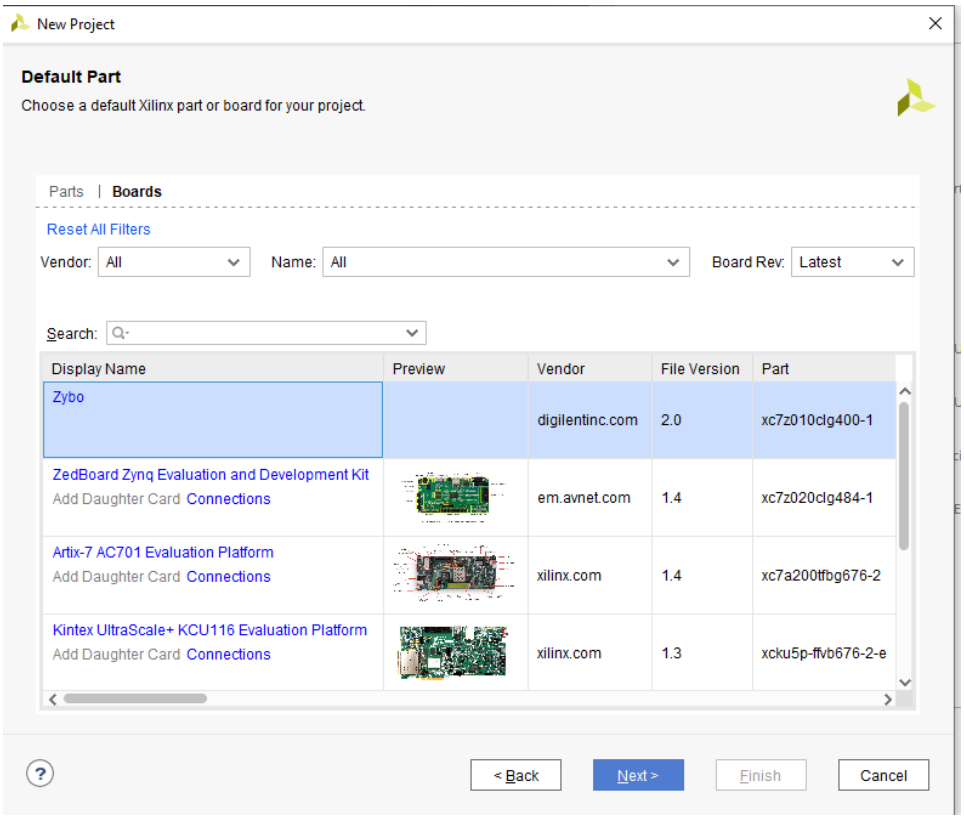

Figura 8: Creación de proyecto con "board files" de Diligent Reference

Con el proyecto ya creado se utilizan las opciones del IP integrador de Vivado para añadir y configurar todos los Cores que se observan en el diseño completo (Figura 7). Los pasos seguidos en cada una de las tres partes son:

a. PS - ZYNQ7 Processing System

En primer lugar, se añade ZYNQ7 Processing System o PS. Aquí irá alojado el sistema Linux sobre el cual correrá la aplicación XVC-Server para conectarse y comunicarse con el PC de control.

Las configuraciones que requiere son:

- Activar la interfaz Ethernet 0 a través de la cual se establecerá la conexión remota.
- Activar el puerto SD1 para la detección de tarjetas memoria SD en la placa, esto se necesitará

para la posterior programación de la placa a través de una imagen Petalinux la cual cargará tanto el sistema Linux en PS como el diseño hardware (archivo bitstream), en PL.

- La activación de los demás puertos, como por ejemplo el UART, USB, SPI, etc., o la configuración de relojes, resets, etc dependerá del diseño del usuario y sus requerimientos, y no interfieren en la realización la depuración remota.

La activación de dichos puertos se realiza desde la pestaña "*Peripherial I/O Pins*" de las opciones de configuración de PS. En la Figura 9 se puede observar los puertos activados para este diseño.

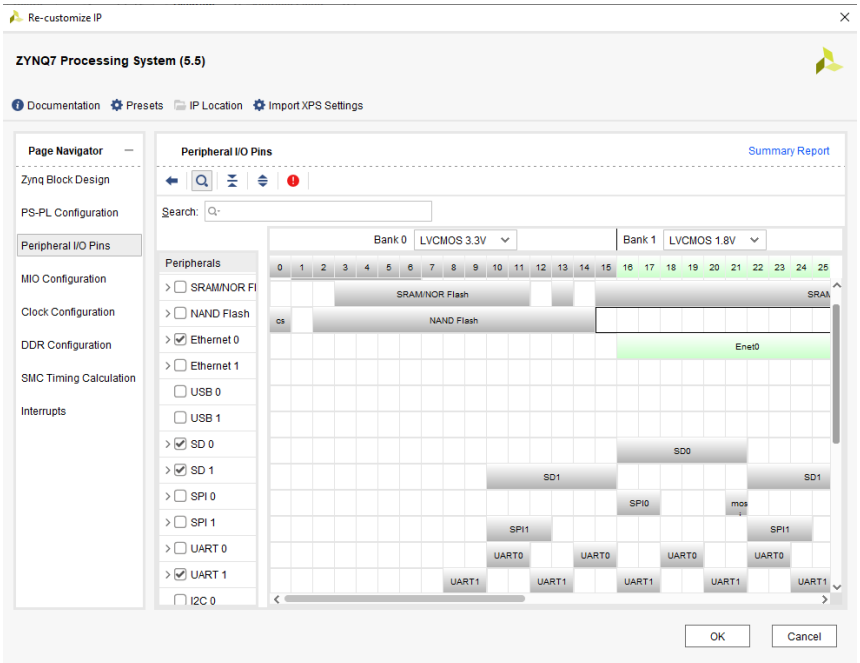

Figura 9 Configuración de I/O Peripherials

Una vez realizadas las configuraciones:

- Click en OK
- Click en Run Block Automation

En la Figura 10 se muestra como que diagrama bloque de PS para este diseño.

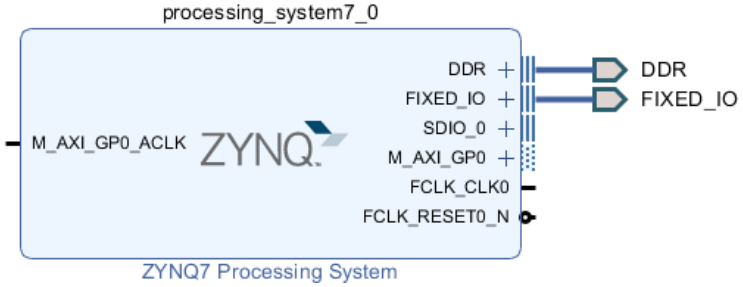

Figura 10: Zynq Block Desing-Configuración de I/O Peripherials

Tal como se muestra en la Figura 11, en la consola TCL aparecen las siguientes advertencias críticas:

- *CRITICAL WARNING: [PSU-1] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_0 has negative value -0.073 . PS DDR interfaces might fail when entering negative DQS skew values.*

- *CRITICAL WARNING: [PSU-2] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_1 has negative value -0.034 . PS DDR interfaces might fail when entering negative DQS skew values.*

- *CRITICAL WARNING: [PSU-3] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_2 has negative value -0.03 . PS DDR interfaces might fail when entering negative DQS skew values.*

- *CRITICAL WARNING: [PSU-4] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_3 has negative value -0.082 . PS DDR interfaces might fail when entering negative DQS skew values.*

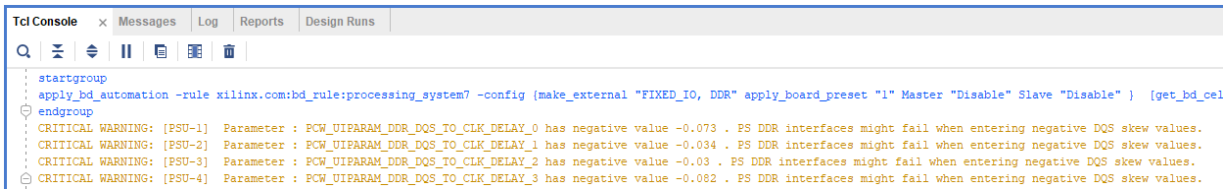

Figura 11: Advertencias Críticas por incompatibilidad de versiones

Eso se debe a una “errata” en los “board files” de Diligent, que causa un problema de incompatibilidad de versiones entre los ajustes preestablecidos de la placa y la versión de Vivado que se está utilizando. [16]

La advertencia crítica aparece en las versiones de Vivado a partir de 2017.4 y está relacionada con que los retrasos de CK a DQS son negativos [16]. Estos valores negativos se deben a que la traza de CK es más corta que cualquiera de las cuatro trazas de DQS [16] y los requerimientos para las interfaces de memoria Zynq-7000 es que el CK sea igual o más largo que el DQS.[17]

Para silenciar las advertencias, se pueden establecer estos retrasos a 0 en la configuración de PS, las herramientas de calibración de Vivado parecen estar usando valores de inicio cero de todos modos cuando se dan retrasos negativos.[17].

- Como se observa en la Figura 12, en la pestaña *“DDR Configuration”* poner a 0 los retrasos de DQS a CLK
- OK

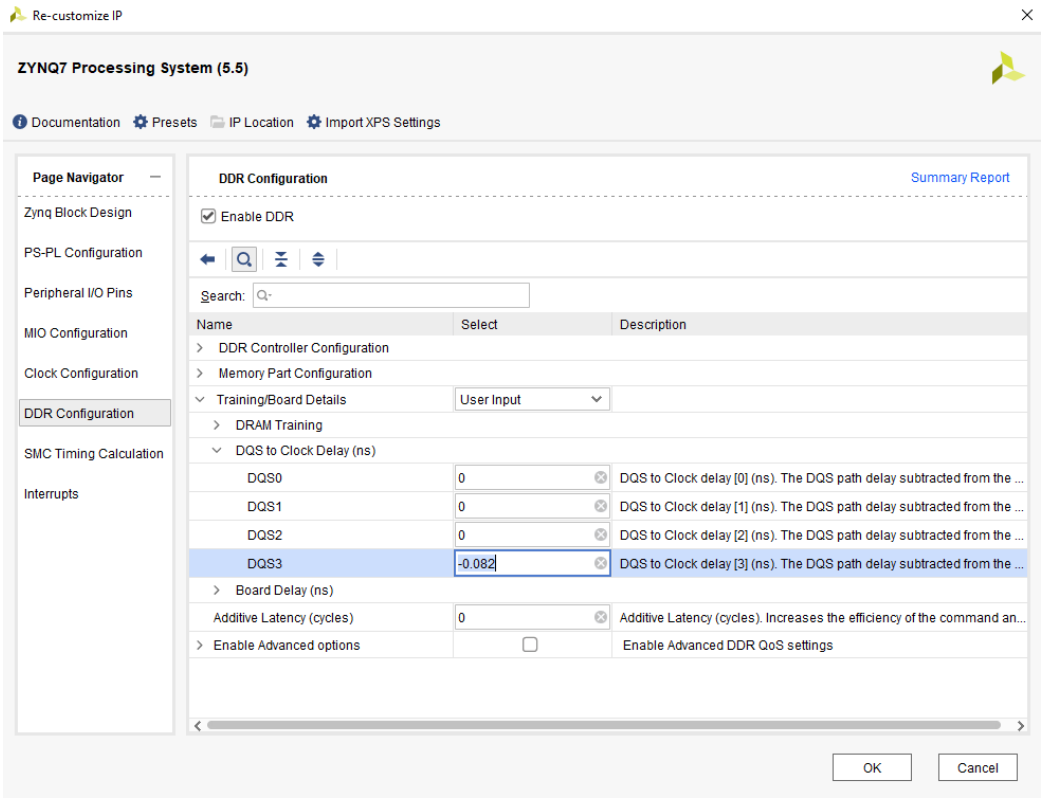

Figura 12: Configuración DDR

b. User-Core

Para las pruebas de depuración se ha realizado se ha diseñado un IP Core "IP_AXI_genPWM" con el que se podrá testear el proyecto. Se trata de un generador de señales PWM, que utiliza dos botones y 4 switches para genera una señal de 4 bits (uno por cada led) para controlar la iluminación de los leds de la placa mediante la variación del ciclo de trabajo o DC.

Para agregar el Core de usuario al repositorio de IPs:

- Project Manager/settings
- Project Settings/IP/Repository/Add
- Seleccionar la carpeta que contiene el IP
- Apply y Ok

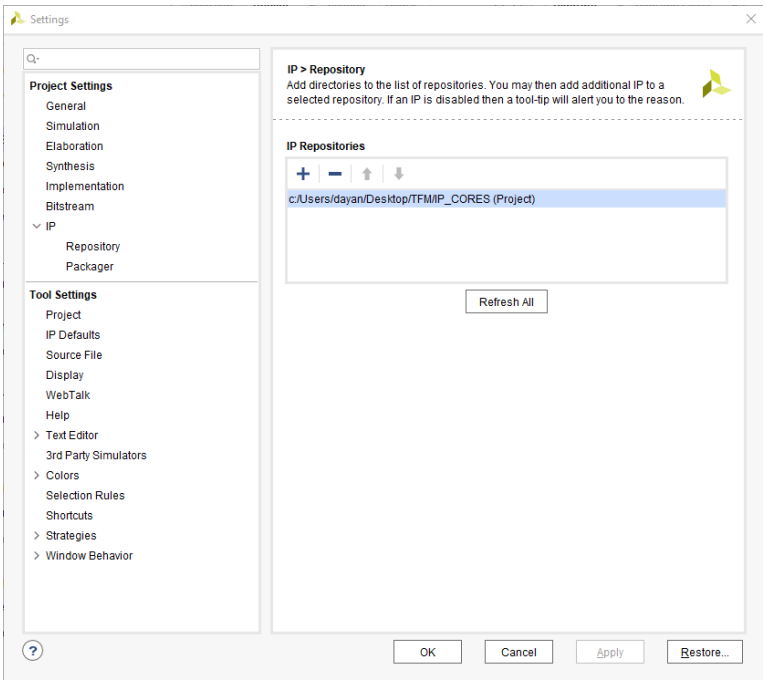

Figura 13: Añadir repositorio de IP Cores de usuario

Ya con el repositorio agregado, en el diagrama de bloques se puede añadir el IP al diseño.

- Add IP
- Seleccionar el nombre del IP creado
- Run Connection Automation
- Click derecho sobre el IP y seleccionar “Make external” para sacar los puertos de salida

En la Figura 14 se observa cómo queda el diagrama de bloques con el IP Core a depurar

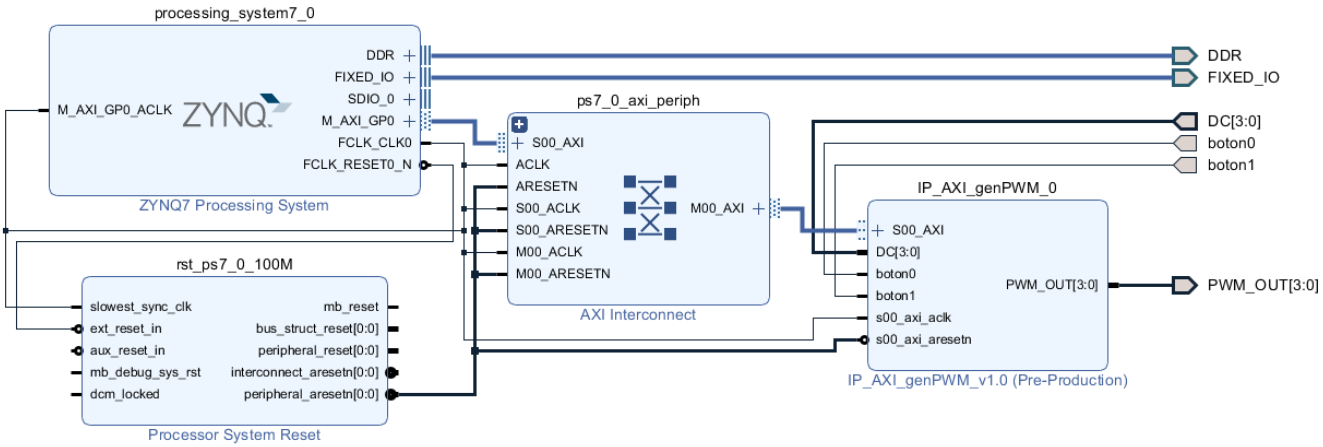

Figura 14: Diagrama de bloques con IP de usuario

En resumen, el funcionamiento de este Core es el siguiente:

Cada bit de la señal PWM de salida tiene su propio ciclo de trabajo. Para la seleccionar que led se quiere encender y cuál va a ser su DC, se utiliza una máquina de estados que gestiona dos botones y los cuatro switches de la siguiente forma:

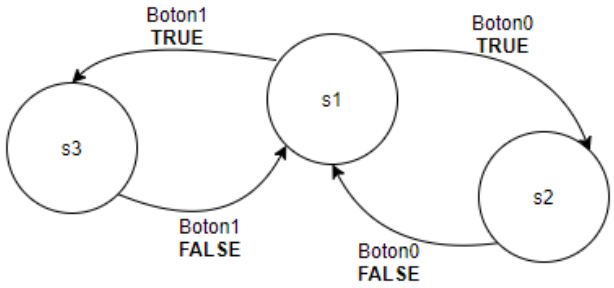

Figura 15 Maquina de estados de IP_AXI_genPWM

- **S1 Estado de espera**: La máquina espera en este estado hasta que se pulse alguno de los dos botones. Si se pulsa el botón 0 se pasa al estado S2, selección de led y si se aplasta el botón 1 se pasa al estado S3.

- **S2 Selección de led**: En este estado, se toma el valor que hay en los switches 0 y 1 y se guarda en un registro con el que se va a seleccionar el led equivalente a dicho valor (0, 1, 2 o 3).

- **S3 variación del DC**: En este estado se toma el valor de los switches y establece el DC en el led que se seleccionó previamente. El valor de DC va de 0 a 15 (4 switches $\rightarrow 2^4$- 1) ciclos de reloj.

Un ejemplo de uso es el siguiente: se quiere seleccionar el led 2 para establecer un DC de 10.

En primer lugar, se escribe 2 en los switches (0010) y en se pulsa el botón 0 para guardar la selección. Ahora se puede variar los switches para establecer el DC, para establecer un DC de 10 se coloca 1010 en los switches y se pulsa el botón 1 para guardar el nuevo DC. Si se quiere cambiar de led se repite nuevamente el proceso.

En la Figura 16 se muestra las pruebas realizadas para comprobar el funcionamiento del IP. Aquí se puede observar el ejemplo dado anteriormente.

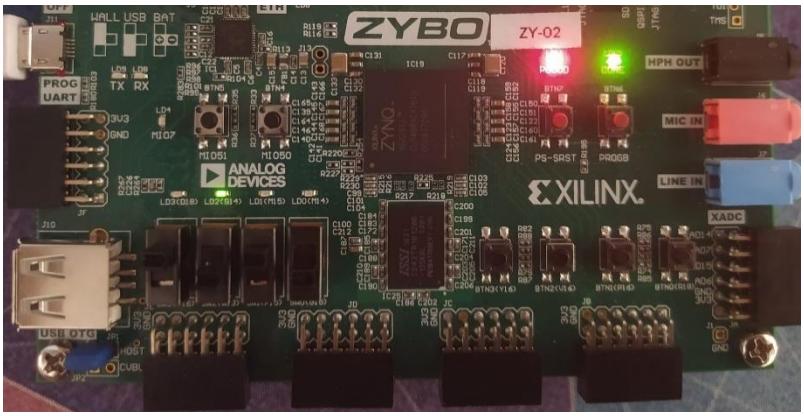

Figura 16: Pruebas de IP_AXI_genPWM

C. Debug Cores (ILA y Debug Bridge)

Con el Core de usuario ya introducido al diseño se pasa a seleccionar las señales que se quieren depurar.

En este caso se ha seleccionado las señales de entrada, boton0 y boton1 y CD (conectado a los switches) así como la señal de salida PWM_OUT que se conecta a los leds.

- Seleccionar todas las señales
- Click derecho/ debug

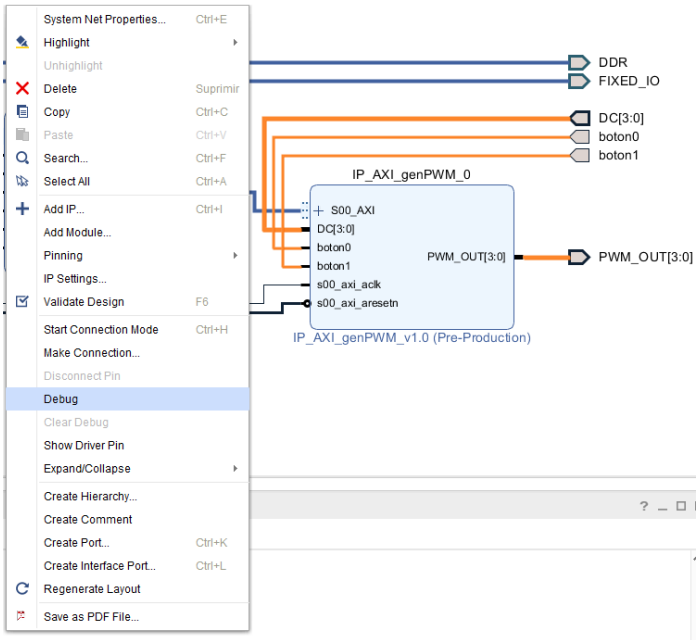

Figura 17: Selección de señales a depurar

- Run Connection Automation
- Ok

Automáticamente se añade el depurador ILA al diseño tal y como se observa en la Figura 18

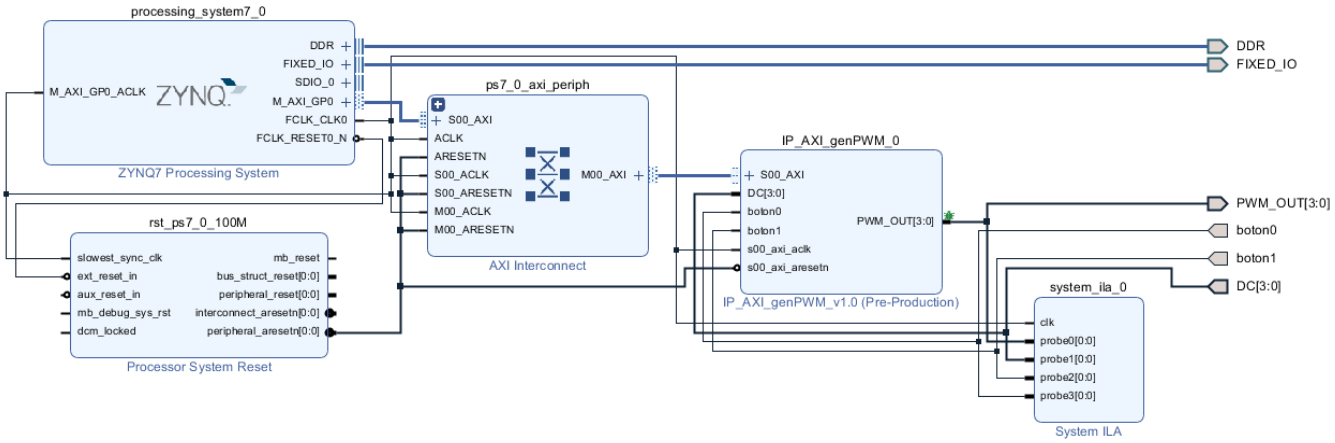

Figura 18: Diagrama de bloques con depurador ILA

Por último, se pasa añadir el puente de depuración o Debug Bridge

- Add IP
- Buscar el IP Debug Bridge v3.0
- Ok

Como se muestra en la Figura 19 se configurará a Debug Bridge en modo “From AXI to BSCAN”. En este modo de configuración, Debug Bridge funcionara con un puente que conecta PS con el depurador ILA. Por un lado, estará conectado a PS a través de un controlador AXI y, a su vez se conectará con el ILA a través de BSCAN.

En resumen, se establecerá una conexión cliente servidor TCP/IP entre el PC de control y PS. Los comandos XVC serán recibidos por PS en protocolo TCP/IP y transmitidos a PL a través de una interfaz AXI4-Lite. Los comandos serán recibidos por Debug Bridge y enviados al ILA a por BSCAN.

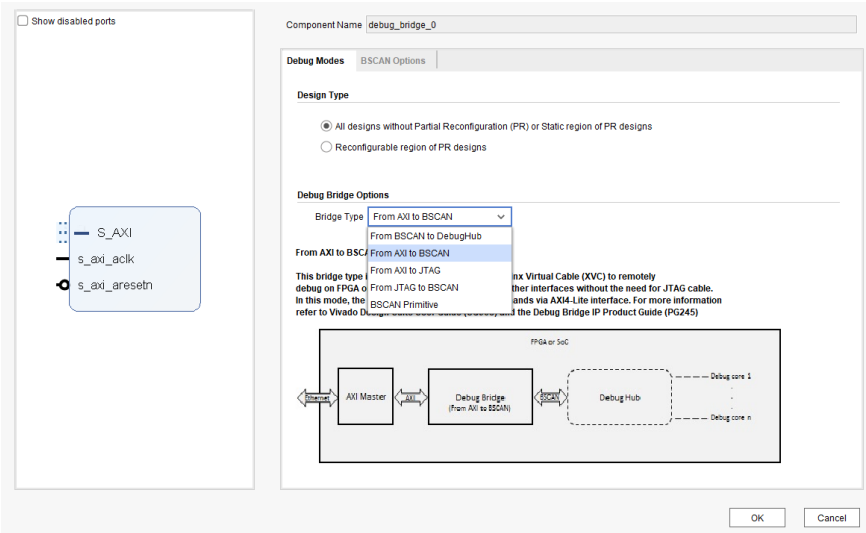

Figura 19: Configuraciones de Debug Bridge

En la pestaña “*BSCAN Options*” la configuración será:

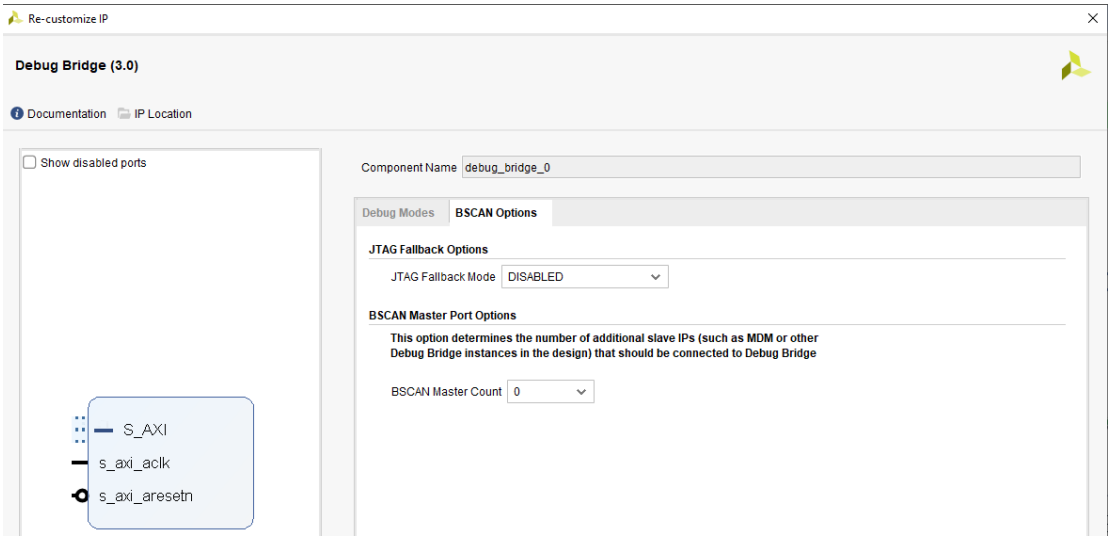

Figura 20: BSCAN Options

Una vez realizada la configuración:

- Run Connection Automation
- Ok

El diagrama final del diseño se muestra en la Figura 21, a partir de aquí ya se podría validar el diseño y generar el wrapper.vhd con la descripción del diseño. En las opciones de Tools click en *“Validate dessing”*.

- En la pestaña Sources seleccionar system.bd / botón derecho / Create HDL Wrapper
- En la pestaña Sources seleccionar system.bd / botón derecho / Generate Output Products

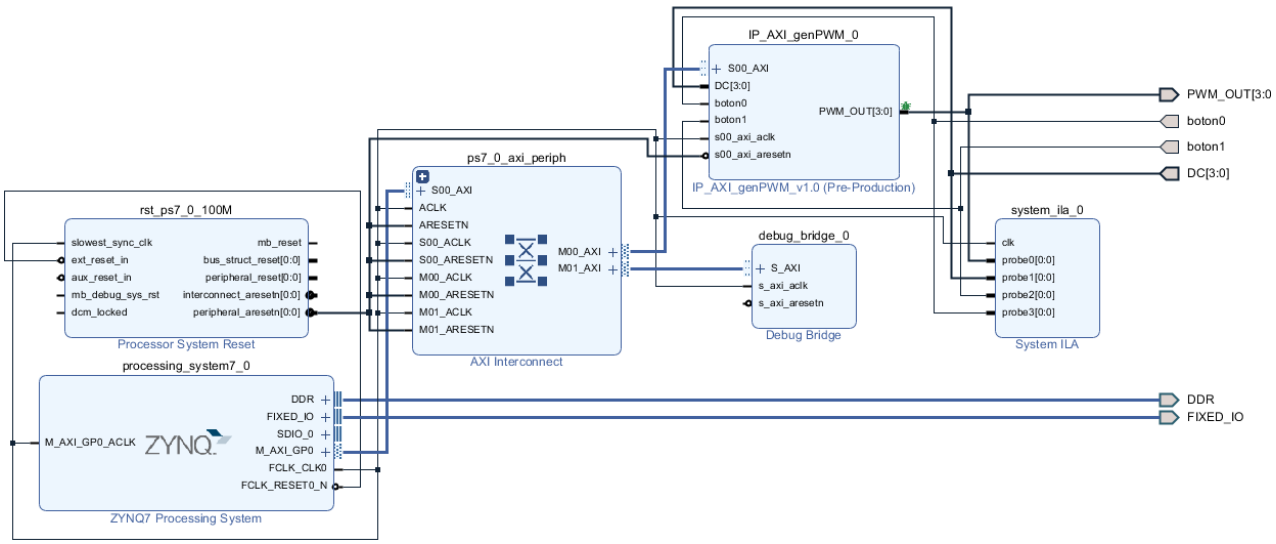

Figura 21: Diagrama de bloques completo

Antes de sintetizar e implementar el proyecto se añade el Constraints con la descripción de las conexiones de las señales a los puertos de la placa. En la Figura 22 se muestra las conexiones realizadas.

```
set_property IOSTANDARD LVCMOS33 [get_ports {DC[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DC[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DC[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DC[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWM_OUT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWM_OUT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWM_OUT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWM_OUT[0]}]
set_property PACKAGE_PIN T16 [get_ports {DC[3]}]
set_property PACKAGE_PIN W13 [get_ports {DC[2]}]
set_property PACKAGE_PIN P15 [get_ports {DC[1]}]
set_property PACKAGE_PIN G15 [get_ports {DC[0]}]
set_property PACKAGE_PIN D18 [get_ports {PWM_OUT[3]}]
set_property PACKAGE_PIN G14 [get_ports {PWM_OUT[2]}]
set_property PACKAGE_PIN M15 [get_ports {PWM_OUT[1]}]
set_property PACKAGE_PIN M14 [get_ports {PWM_OUT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports boton0]
set_property IOSTANDARD LVCMOS33 [get_ports boton1]
set_property PACKAGE_PIN R18 [get_ports boton0]
set_property PACKAGE_PIN P16 [get_ports boton1]
```

Figura 22: Constraints

A partir de aquí ya se puede generar el Bitstream

- Program and Debug/Generate Bitstream/ok

Una vez generado se exporta el hardware junto con el Bitstream para generar el archivo "*.hdf*". Este archivo será utilizado posteriormente para crear el proyecto Petalinux con las configuraciones realizadas anteriormente.

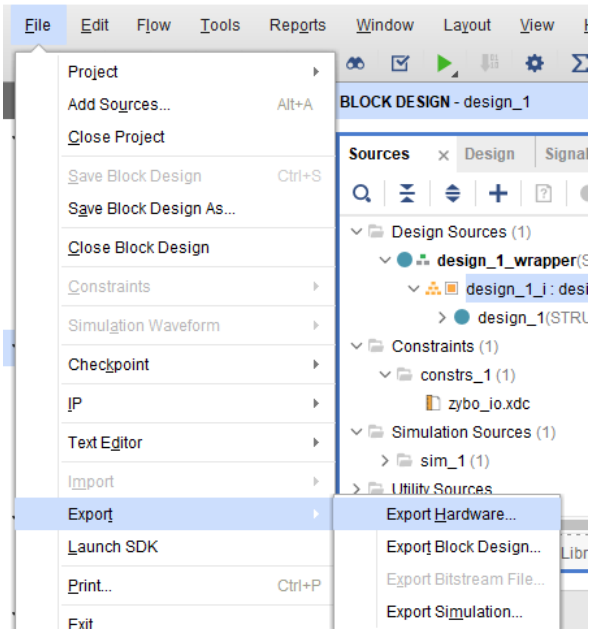

Figura 23: Export Hardware

Para terminar tal y como se muestra en la Figura 24, se buscan los archivos ".hdf" y ".bit" y se guardan en una carpeta aparte para usarlos en la parte software del proyecto.

La ubicación del archivo ".hdf" está en la carpeta sdk generada en el proyecto, mientras que el archivo *".bit"* estará la carpeta "tester_VXC.runs\impl_1".

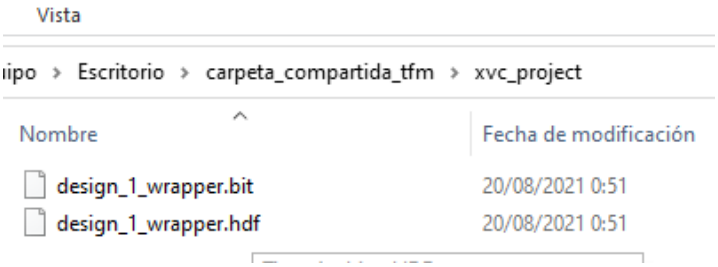

Figura 24: Archivos hdf y bit

5.2. Nivel Software

Para establecer la comunicación entre PS y el PC de control se utilizará la aplicación xvcServer. Como se mencionó anteriormente en el punto 4.3, esta aplicación utiliza el protocolo Xilinx Virtual Cable (XVC) (basado en TCP/IP), para actuar como un cable JTAG y proporcionar un medio para acceder y depurar un diseño de FPGA o SoC de forma remota. [14]

En resumen, el protocolo XVC es utilizado por un cliente y servidor TCP/IP para transferir vectores JTAG de bajo nivel, desde una aplicación de alto nivel (en este caso desde Vivado) a un dispositivo FPGA o SoC (en este caso PS de Zybo).

Esta aplicación correrá sobre un sistema Linux instalado en PS. Para ello se creará un proyecto de Petalinux que cuente con las características y configuraciones de la placa realizadas durante el diseño hardware en Vivado. Las herramientas que se utilizarán son:

- Máquina Virtual **Ubuntu 16.02**
- **Petalinux v2018.2**
- **Archivo xvcServer.c** descargado del repositorio github de Xilinx https://github.com/Xilinx/XilinxVirtualCable [14]
- Archivos ".hdf" y ".bit" generados anteriormente en el diseño hardware en Vivado

Para comenzar, en la máquina virtual de Ubuntu se creará una carpeta vacía en donde se alojará el proyecto. Una vez creada se debe abrir una terminal de la carpeta y habilitar las herramientas de Petalinux.

Para habilitar las herramientas se debe conocer previamente la carpeta en donde se instaló Petalinux 2018.2, en este caso, la carpeta es /opt/petalinux. Desde la terminal de la carpeta ejecutar el siguiente comando:

- Source /opt/petalinux/settings.sh

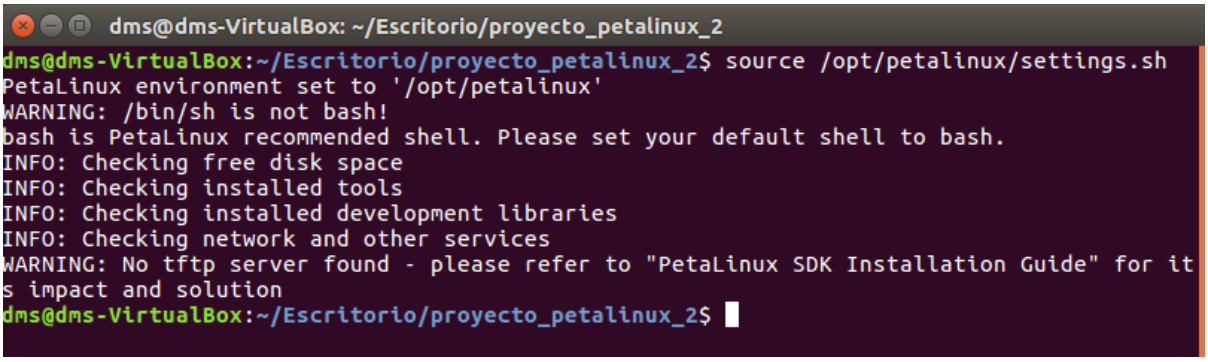

Figura 25: Habilitación de herramientas Petalinux

Una vez habilitadas las herramientas de Petalinux se pasa a crear el proyecto inicial.

Al igual que los "boards files" de Vivado, por lo general Xilinx ofrece "Board Support Packages" o ficheros ".bsp" contienen las características de las placas de desarrollo, como por ejemplo Zedboard, Microblaze, etc. Estos ficheros pueden servir de base a los diseñadores para la creación de sus propios proyectos Petalinux.

En caso de que no se cuente con ficheros bsp para una placa en específico, se puede utilizar las plantillas de Petalinux con las características de FPGAs con diferentes arquitecturas como Zynq-7000, Zynq-UltraScale+, etc.

Para el caso de la placa Zybo no se cuenta con estos ficheros, por lo que se crea el proyecto en base a la plantilla de su FPGA con arquitectura Zynq.

En la terminal se ejecutar el comando:

- Petalinux-create -t Project -n (nombre del proyecto) –template zynq

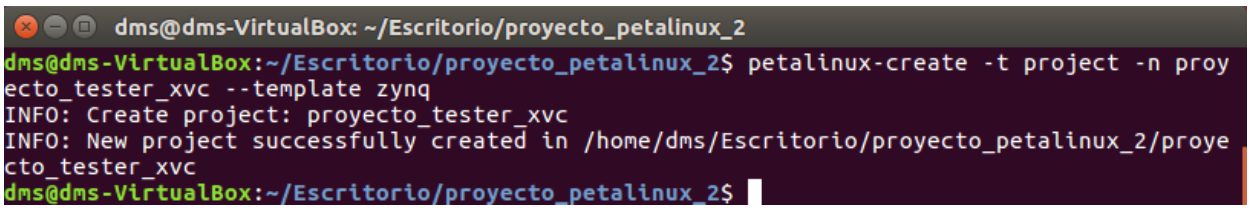

Figura 26: Creación de proyecto Petalinux

Como se observa en la Figura 27: Carpeta de Proyecto dentro de la carpeta creada anteriormente se genera una carpeta con el nombre que se le dio al proyecto, aquí se copiaran los archivos bit y hdf que se generaron en el proyecto de Vivado.

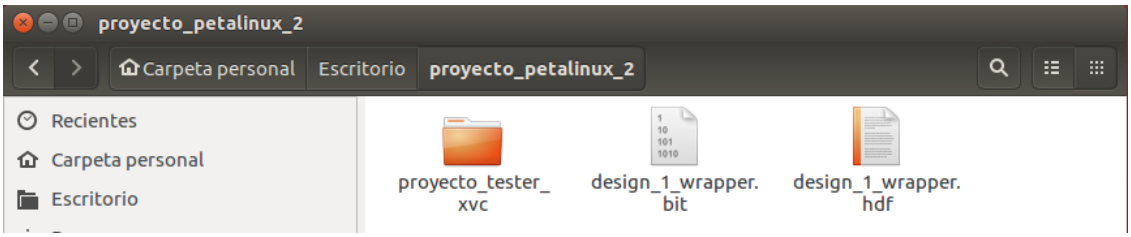

Figura 27: Carpeta de Proyecto

Una vez copiados los archivos, se vuelve a la terminal y se accede a la carpeta que contiene el proyecto inicial con el comando cd:

```
- cd (nombre de proyecto)
```

Ya dentro del proyecto se puede pasar la configuración hardware de la placa (archivo .hdf) para que el proyecto se ajuste al diseño de Vivado.

```
- petalinux-config --get-hw-configuration=..
```

Aparece una ventana de configuración para Zynq que se desean para el proyecto:

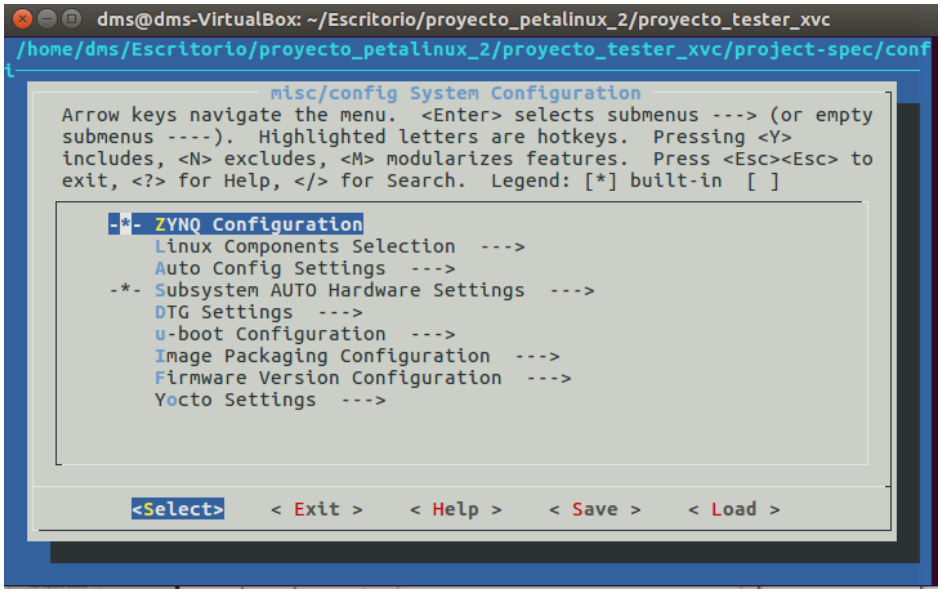

Figura 28: Ventana de configuración de Zynq

- *DTG Settings/ Kernel Bootargs*
- Deseleccionar la opción "*generate boot args automatically*"
- Seleccionar "*user set kernel bootargs"* y escribir la siguiente configuración:

```
earlycon console=ttyPS0,115200 clk_ignore_unused uio_pdrv_genirq.of_id=generic-uio
```

Esto establecerá el baud rate del Uart1 en PS a 115200. (La comunicación serie con PS a través de la interfaz Uart, será necesaria para iniciar sesión en el sistema Linux más adelante). En la Figura 29: configuración de kernel-Bootargs se puede observar cómo quedaría la pantalla de configuración.

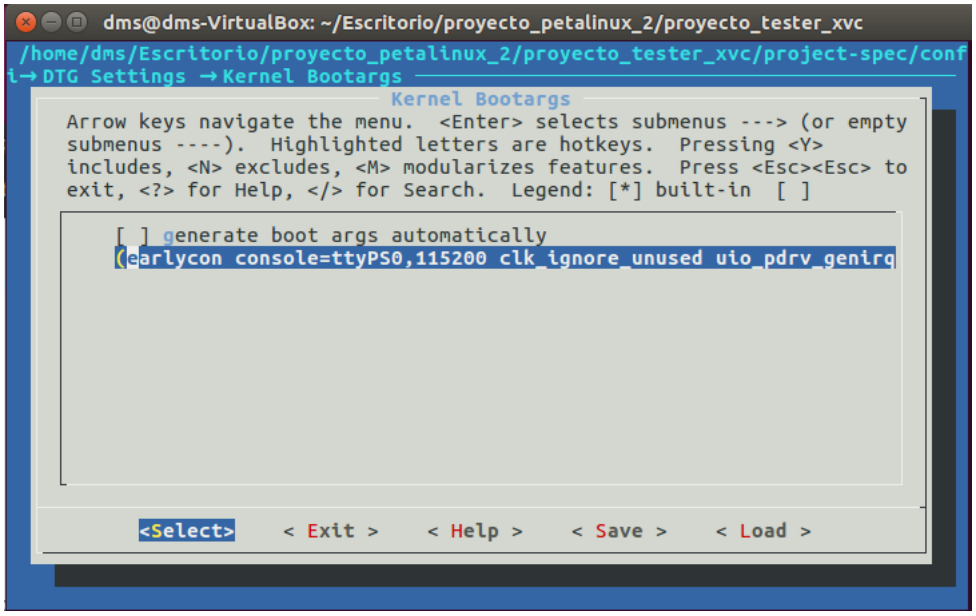

Figura 29: configuración de kernel-Bootargs

Una vez hecho esto, se guarda la configuración y se sale de la ventana. En la terminal aparecerán varios mensajes de aviso mientras Petalinux construye el proyecto. En la se observa los mensajes finales una vez acaba de compilar la configurar hardware.

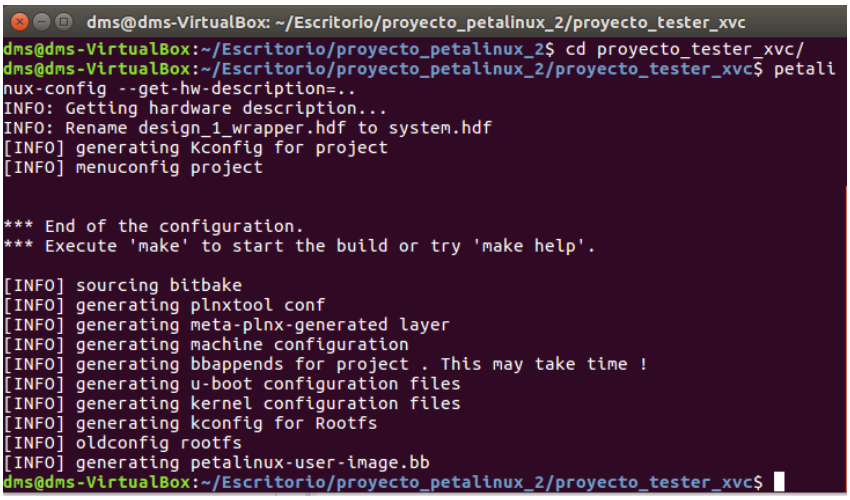

Figura 30: Configuración de la descripción Hardware

Ahora se pasa a la configuración del Kernel de Linux, en el terminal se escribe el comando:

```
-    petalinux-config -c kernel
```

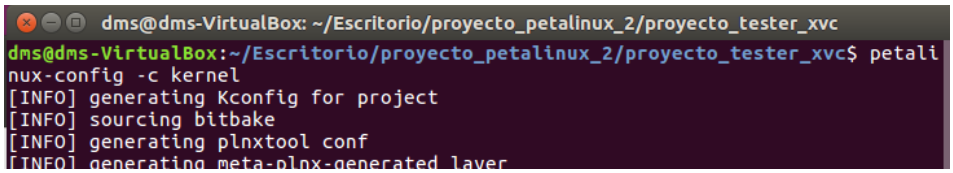

Figura 31: Configuración Kernel de Linux

Aparecerá la siguiente ventana de configuración

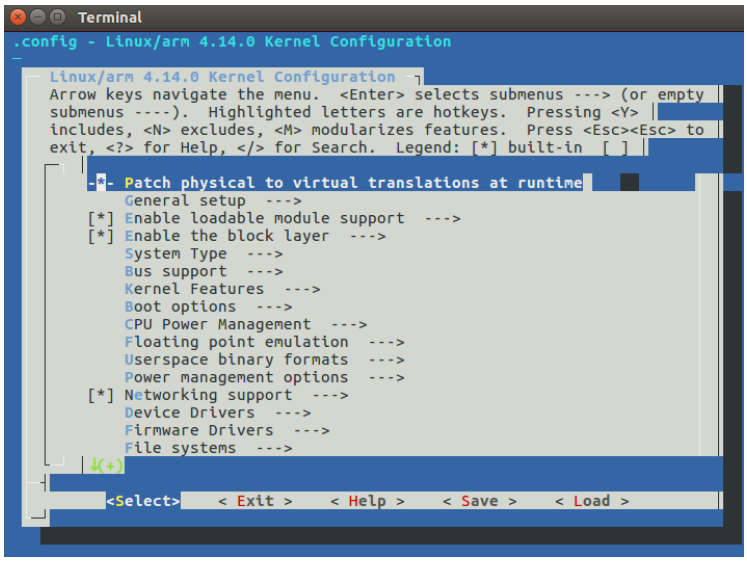

Figura 32: Ventana de Configuración de Kernel

- *Device Drivers -> Userspace I/O drivers*

Las opciones de configuración se muestran en la Figura 33 , Esto permitirá que los controladores UIO se incluyan en el kernel.

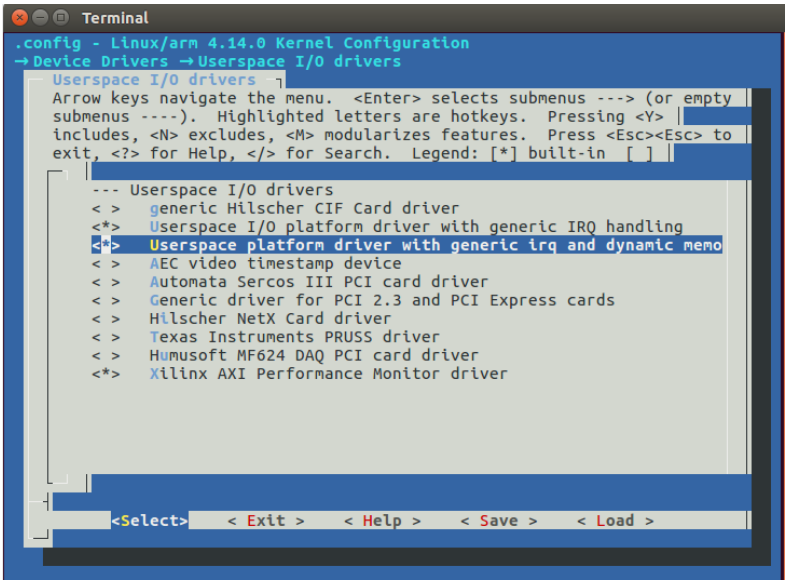

Figura 33: Opciones de configuracion Kernel

- Se guardan los cambios y se vuelve a la ventana inicial
- *Power Management -> CPU Idle*

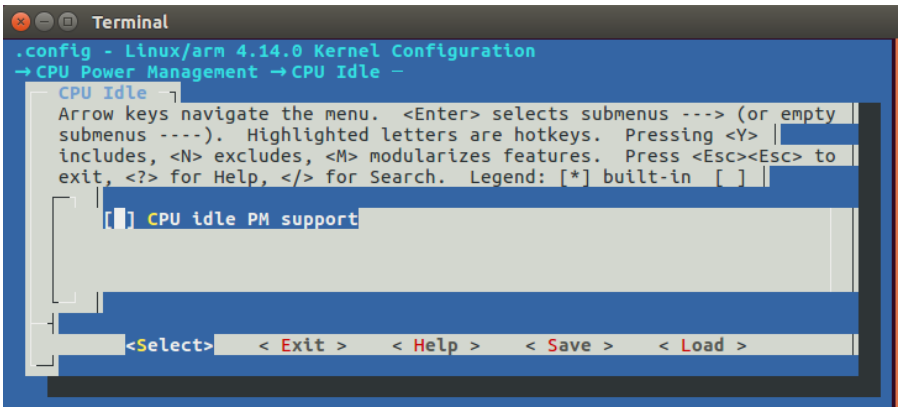

Figura 34: CPU Power Managment

Guardar y salir de las configuraciones. De igual forma que antes, en el terminal aparecerán varios mensajes mientras se compilan las configuraciones del Kernel. En la Figura 35 se muestran los mensajes finales una vez haya terminado.

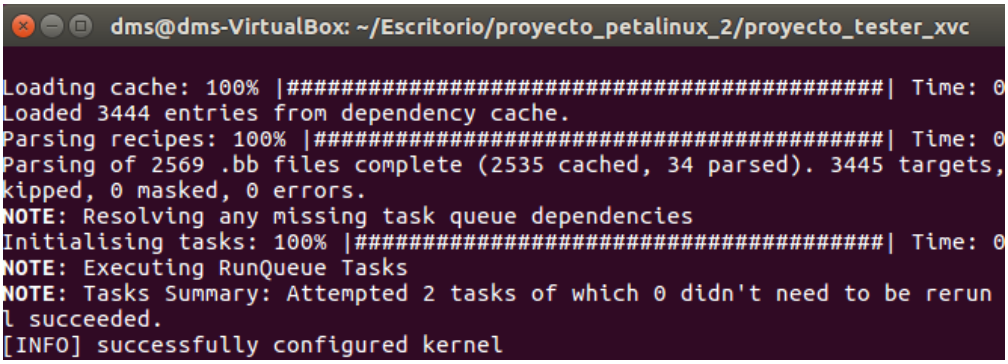

Figura 35: Compilación de configuraciones Kernel

Una vez que se completado el proceso, se necesita modificar el device-tree para permitir que Debug Bridge se comunique en el espacio uio de Linux.

El device-tree es una estructura de datos en formal de árbol que utiliza el kernel durante el arranque, para saber el hardware que está bajo control del sistema operativo [19]. En este caso, PS enviara los comandos XVC al core Debug Bridge, luego se tiene que configurar el fichero para que controle dicho core, esto se realiza de la de la siguiente forma:

El fichero a modificar se encuentra en la ruta: *project-spec / meta-user / recipes-bsp / device-tree / files / system-user.dtsi*

- Agregar el driver de Debug bridge justo al final tal como se observa en la Figura 36

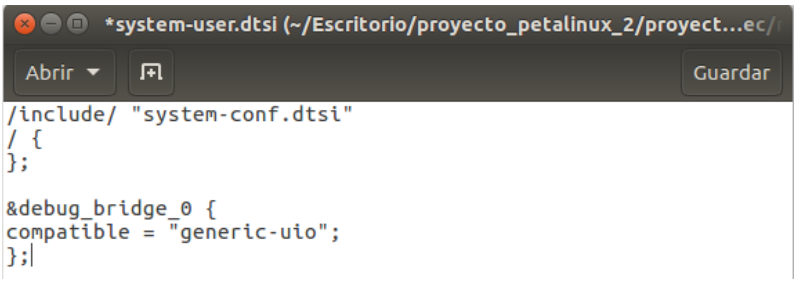

Figura 36: Configuración del Device-tree

Una vez hecho esto ya se puede agregar la aplicación de xvcServer

```
-    petalinux-create -t apps -n xvcserver --template c --enable
```

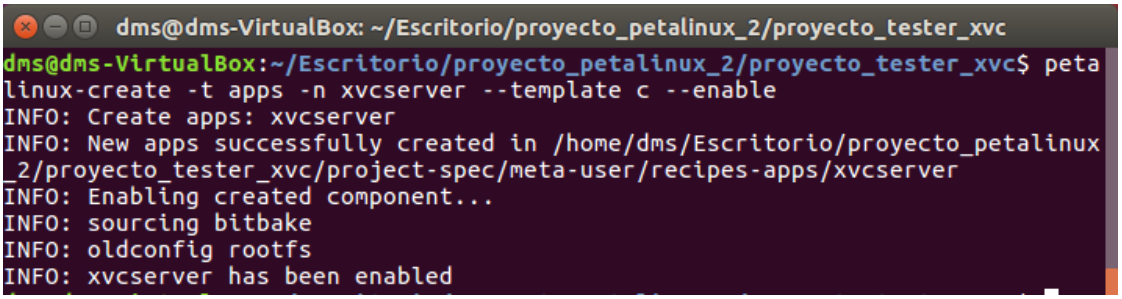

Figura 37: Aplicación xvcServer

Este comando creará la plantilla para la aplicación xvcServer, incluida una plantilla de "hola mundo" en c.

En el archivo c se deberá cambiar el código de “Hola mundo”, por el código de xvcServer descargado desde el repositorio github de Xilinx https://github.com/Xilinx/XilinxVirtualCable [14].

Este código contiene las bases del protocolo XVC que permitirá a la aplicación actuar como un driver TCP/IP para conectar a PS con el software de Vivado para realizar la depuración.

En resumen, el funcionamiento del protocolo XVC es el siguiente:

Se tienen 3 mensajes principales:

- getinfo:
- settck:<period in ns>
- shift:<num bits><tms vector><tdi vector>

EL protocolo de comunicación iniciara cuando el cliente, en este caso Vivado, se conecta a un servidor XVC (PS) a través de una dirección IP.

Una vez conectado, el cliente emitirá un mensaje inicial “getinfo” al servidor solicitando la versión del servidor. La versión del servidor proporciona al cliente una forma de determinar las capacidades de

protocolo del servidor. A su vez, el servidor le responderá enviando un mensaje tal que <xvc_vector_len>\n" que es el ancho máximo del vector que se puede shiftear al servidor.[14]

Cuando cliente valida la versión del protocolo, se envía un nuevo mensaje al servidor XVC "*settck*" para establecer la tasa de tck de JTAG para futuras operaciones de turno. Esto se realiza ya que, al enviar vectores JTAG, es posible que sea necesario variar la velocidad TCK para adaptarse a las condiciones de integridad de la señal del cable y la placa. El cliente utiliza este comando para ajustar la tasa TCK con el fin de ralentizar o acelerar el desplazamiento de los vectores JTAG.[14]

Realizados estos ajustes, comienza el tráfico de datos de forma normal. Para enviar los mensajes de cambio se utiliza el comando"shift". Este es el comando principal utilizado entre el cliente y el servidor para transferir vectores JTAG de bajo nivel. El cliente emitirá operaciones shift para determinar la composición de la cadena JTAG y luego ejecutará varias instrucciones JTAG durante la depuración de la placa.

Para copiar el código dentro de la aplicación se debe buscar el archivo c en la ruta project-spec / meta-user / recipes-apps / xvcserver / files/xvcserver.c

Adicionalmente, dentro del código xvcserver es posible que se deban realizar algunos cambios dependiendo de la placa de desarrollo que se valla a utilizar. Por ejemplo, en los dispositivos Zinq, el kernel de Linux utiliza uio0 (Userspace I / O framework) para controlar operaciones de E/S de bajo nivel, por otro lado, en la arquitectura de los dispositivos UltraScale+, se utiliza uio1.

El código original de xcvserver se preparó para ser usado en dispositivos Zynq [14] por lo que, en este caso, la placa Zybo es compatible, pero en el caso de usar dispositivos MPSoC, se deberá cambiar la dirección de los archivos de espacio de usuario, uio0 a uio1 tal en la línea de condigo que se observa en la Figura 38.

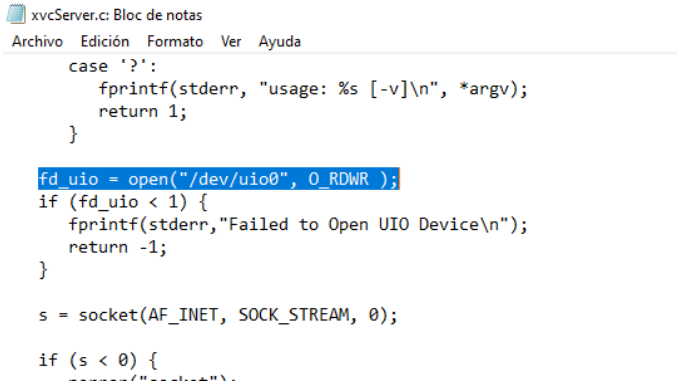

Figura 38: Dirección de archivos de espacio de usuario UIO0

Otra cosa a tener en cuenta en el código es que el Listen() backlog puede llegar a dar problemas para establecer la comunicación. Básicamente, Listen() backlog afecta al número de conexiones entrantes que pueden hacer cola si la aplicación no acepta conexiones tan pronto como llegan. Es decir, si durante la primera llamada de establecimiento de red la aplicación no escucha, se pone a la cola otra llamada para volver a intentar la comunicación.

En el código xvcServer el Listen() backlog está configurado a 0 (lo que significa solo escucha la primera llamada). Se puede configurar este valor para evitar posibles problemas de conexión, en este caso, se a establecido su valor a 5 tal como se observa en la Figura 39.

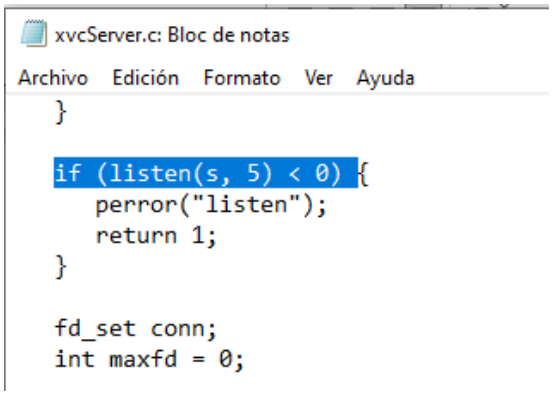

Figura 39: Configuración del Listen() backlog

Una vez realizados estos cambios en el código, se puede construir el proyecto completo. En el terminal se escribe el comando:

```
-    Petalinux-build
```

El proceso de compilación puede tardar varios minutos y emitirá varios mensajes como se muestra en la Figura 40.

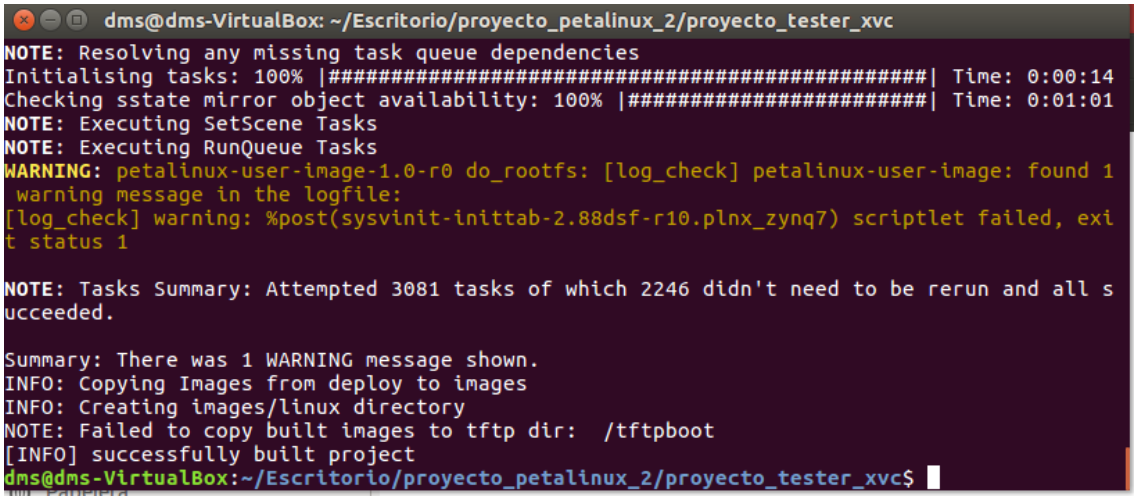

Figura 40: Proyecto Compilado

Cuando se haya completado el proceso, en la carpeta "*images*" dentro del proyecto, se crearán varios archivos mediante los cuales se podrá crear una imagen de arranque. Esta imagen servirá para programar tanto PS como PL durante la secuencia de arranque de la placa. Los archivos creados se pueden observar en la Figura 41.

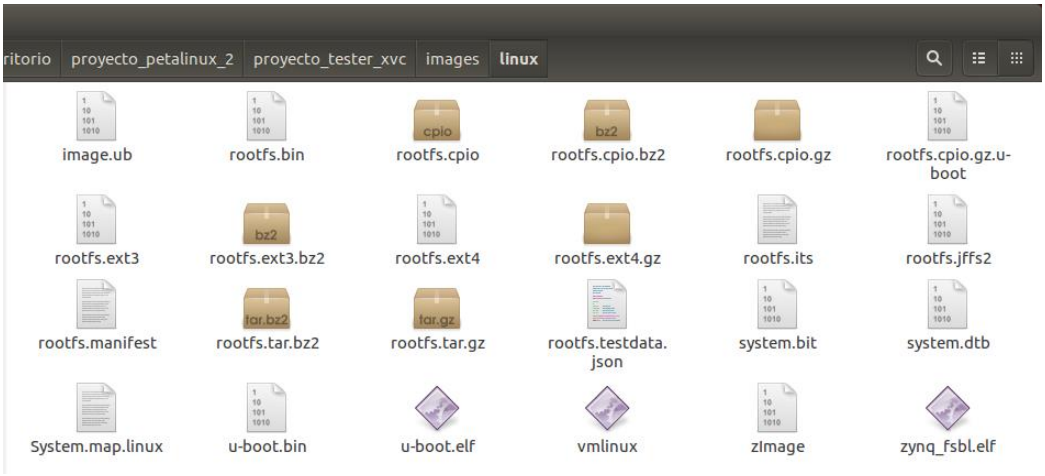

Figura 41: Archivos creados tras la compilación del proyecto

La imagen creada deberá incluir un programa de inicio FSBL o First Stage Bootloader (zynq_fsbl.elf), la aplicación xvcserver junto con el kernel de Linux (u-boot.elf), y el bitstream (system.bit).

Como se comentó antes, esta imagen de arranque servirá para cargar la programación de PS y el diseño en PL, en la Figura 42 se observa como seria la secuencia de arranque del dispositivo.

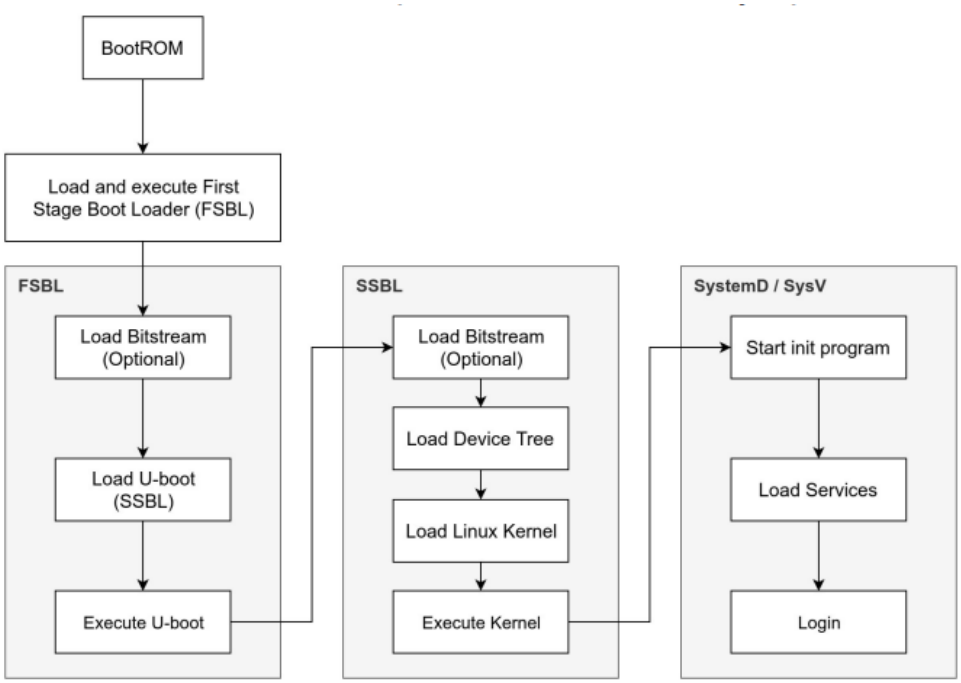

Figura 42: Secuencia de arranque de linux en Zynq

Cuando el dispositivo se enciende, la PS comienza a ejecutar un código de arranque contenido en una memoria ROM integrada dentro del propio ZYNQ (BootROM).

Duante este proceso se inicializa la APU y una cantidad mínima de periféricos y se leen los pines *BOOT_MODE* que determinan el dispositivo principal desde el que arrancar: NOR, NAND, Quad-SPI o SD. [19]

Luego de esto se pasa a ejecutar el programa First State Boot Loader, FSBL. Este programa ejecuta 3 principales tareas:

1) Se configura la PS de acuerdo a la información exportada desde Vivado (Inicialización de los registros de configuración).
2) Si se encuentra un bitstream en el dispositivo de arranque se configura la PL.
3) Se carga en la memoria DDR la aplicación en standalone (si no hay sistema operativo) o un SSBL, Second Stage Bootloader, en caso de usar Linux.
4) Se salta a ejecutar la aplicación o el SSBL (U-boot en Linux).

U-boot es un gestor de arranque universal para Linux diseñado para utilizarse en sistemas embebidos. El flujo de ejecución de U-boot:

1) U-boot configura los periféricos necesarios.
2) Se cargan las variables de entorno.
3) Si “bootdelay” es mayor que cero se muestra una cuenta atrás dando oportunidad al usuario a detener el arranque y entrar a la consola de u-boot.
4) Si el usuario no para el arranque se ejecuta “bootcmd”.
5) “bootcmd” carga en memoria RAM, desde memoria no volátil el kernel, device tree, y opcionalmente una imagen de RAM disk.
6) Se ejecuta el kernel de Linux

Para crear la Imagen de arranque, en la terminal se escribe el siguiente comando:

```
-   petalinux-package --boot --fsbl images/linux/zynq_fsbl.elf –-fpga images/li
    nux/system.bit –-u-boot --force
```

```
dms@dms-VirtualBox: ~/Escritorio/proyecto_petalinux_2/proyecto_tester_xvc
dms@dms-VirtualBox:~/Escritorio/proyecto_petalinux_2/proyecto_tester_xvc$ petali
nux-package --boot --fsbl images/linux/zynq_fsbl.elf --fpga images/linux/system.
bit --u-boot --force
INFO: File in BOOT BIN: "/home/dms/Escritorio/proyecto_petalinux_2/proyecto_test
er_xvc/images/linux/zynq_fsbl.elf"
INFO: File in BOOT BIN: "/home/dms/Escritorio/proyecto_petalinux_2/proyecto_test
er_xvc/images/linux/system.bit"
INFO: File in BOOT BIN: "/home/dms/Escritorio/proyecto_petalinux_2/proyecto_test
er_xvc/images/linux/u-boot.elf"
INFO: Generating Zynq binary package BOOT.BIN...


****** Xilinx Bootgen v2018.3
  **** Build date : Nov 15 2018-19:22:29
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
```

Figura 43: Creación de la imagen de arranque

La imagen creada BOOT.BIN se guardará en la misma carpeta Images dentro del proyecto. Los archivos "*BOOT.BIN*" e "*image.ub*" se guardan en una SD que posteriormente se introducirá en la placa Zybo.

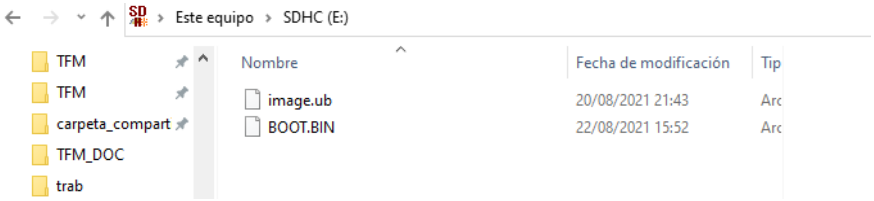

Figura 44: Imagen de arranque en SD

Para finalizar, se debe colocar los pines *BOOT_MODE* de la placa para configurar el arranque desde la SD. En el manual de Zybo proporcionado por de Dilingent Reference [20] se observa cómo deben colocarse los pines para obtener el microSD Boot Mode.

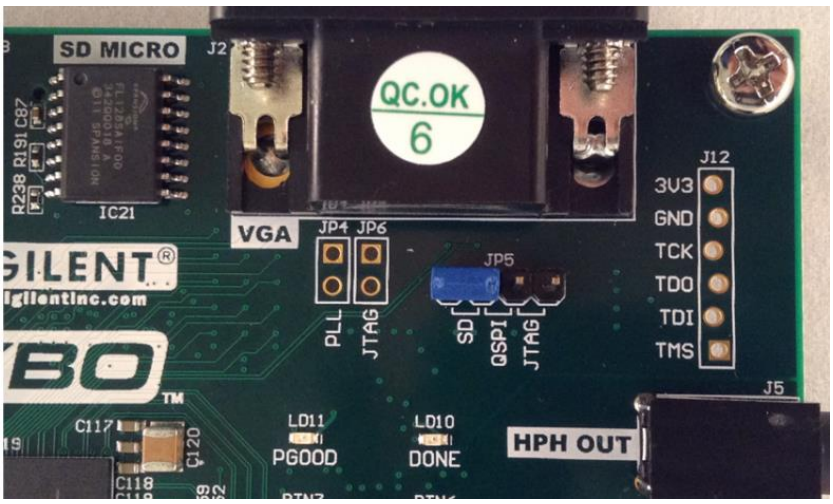

Figura 45: Configuración de pines BOOT-MODE para microSD Boot Mode [20]

6. PRUEBAS Y ENSAYOS

Para comenzar se enciende el dispositivo y se conecta a él mediante comunicación serie. En este caso se ha utilizado la aplicación Putty para realizar la conexión.

Como se mostró en el punto anterior, se ha configurado la interfaz UART para tener un Baud Rate de 115200 bps.

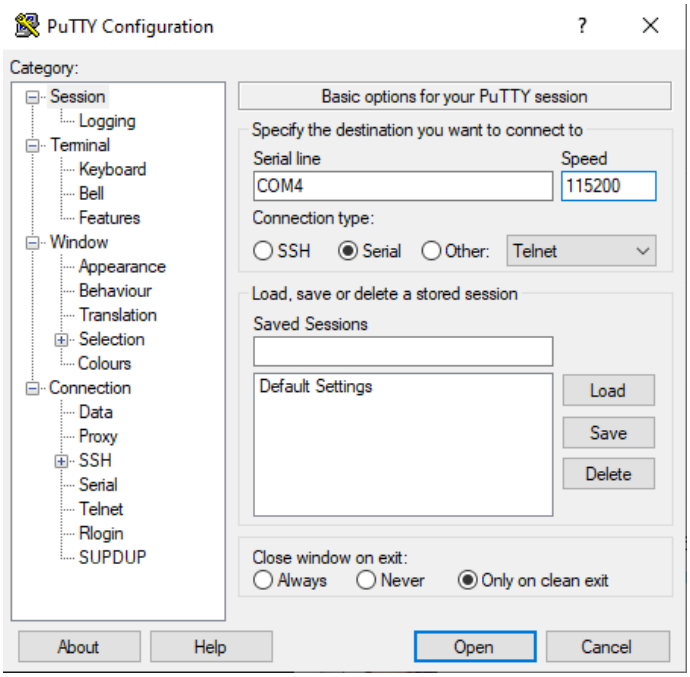

Figura 46: Configuración comunicación serie

En la Figura 47 se observa los mensajes de inicio una vez se arranca la placa. En la figura se observa cómo se ha leído la imagen.ub que se cargó en la SD y posteriormente empieza a cargar el Kernel de Linux.

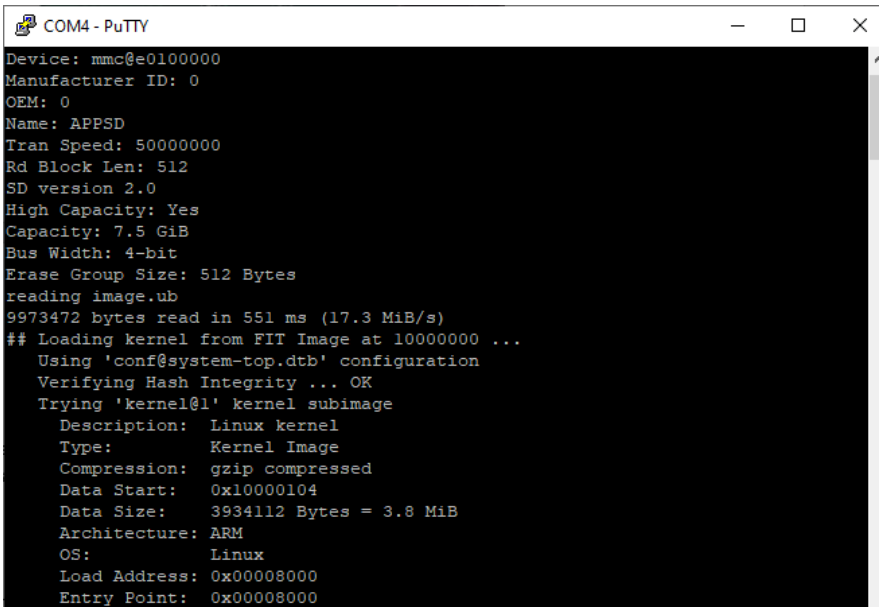

Figura 47:Arranque de la placa

Una vez se haya cargado los datos y configurado PS, la placa pide un inicio de sesión. Las credenciales son:

Login: *root*

Contraseña: *root*

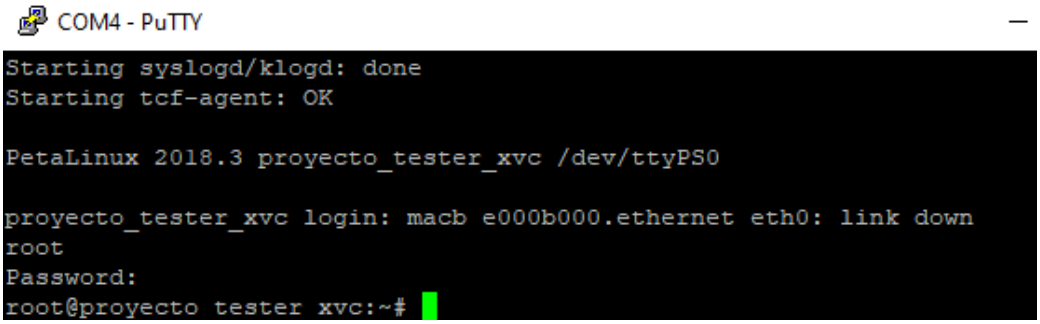

Figura 48: Inicio de sesión en kernel de Linux

Antes de ejecutar la aplicación xvcServer se deberá obtener o configurar la dirección IP de la interfaz Ethernet mediante la cual se establecerá la conexión con la placa.

En la terminal de comunicación serie se escribe el comando:

```
-   ifconfig
```

Aparecerán los datos de configuración Ethernet, en el este caso de que no se haya establecido una dirección IP específica para el dispositivo y habrá que configurarlo manualmente.

En el terminal se escribe:

```
-   Ifconfig eth0 192.168.xx.xx (dirección IP deseada)
```

Para comprobar que se configuró correctamente se ejecuta nuevamente el comando

```
-   ifconfig
```

En este caso, se ha establecido una dirección IP de 192.168.10.10 para la placa.

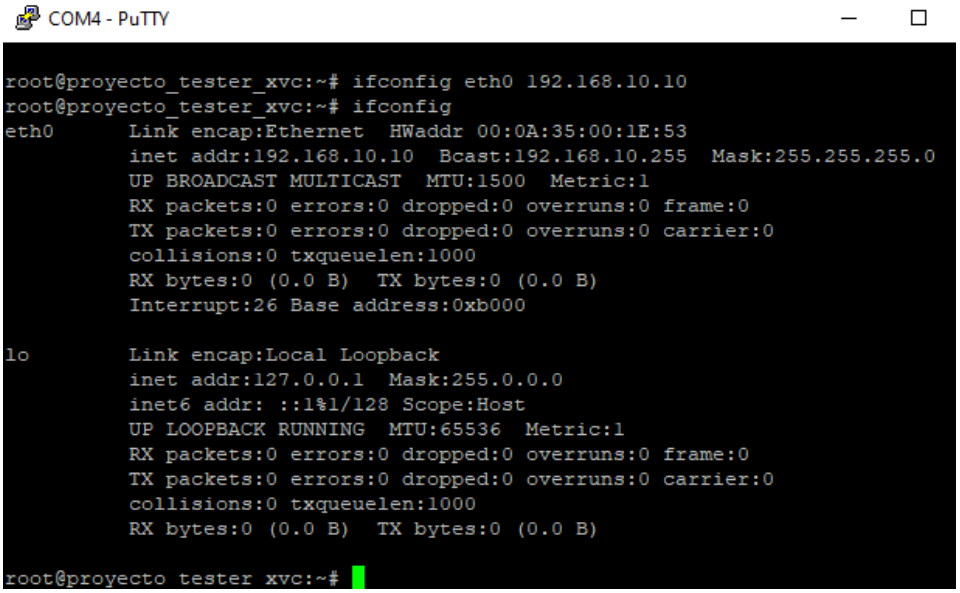

Figura 49: Configuración de dirección IP de Ethernet

Hecho esto se puede ejecutar la aplicación xvcserver, en la terminal escribir:

```
- xvcserver
```

A partir de aquí ya se podría conectar el software de Vivado con la placa a través de una interfaz ethernet.

Se realizarán tres pruebas de depuración, la primera conectando directamente el PC de control a la placa Zybo a través de un cable Ethernet. La segunda prueba se realizará a través de una red de área local, para ello se conectará la placa Zybo a un Router, el cual emitirá una señal Wifi para que el PC pueda conectarse y acceder remotamente a la red. En la tercera y última prueba, se buscará imitar un entorno más real, en el que la placa Zybo y el PC de control estén muy separados y no se encuentran conectados a una misma red local, en este caso, se realizará la conexión entre ambos dispositivos se realizara a través de internet.

6.1. Primera Prueba: conexión directa con cable Ethernet

En este caso se busca establecer una conexión directa a través de un cable Ethernet, para ello se deberá configurar las opciones del adaptador Ethernet del PC de control, para que sea compatible y pueda conectarse a la dirección IP de la placa.

En el PC de control acceder a:

- Configuraciones/Red e Internet/Ethernet/cambiar opciones del adaptador

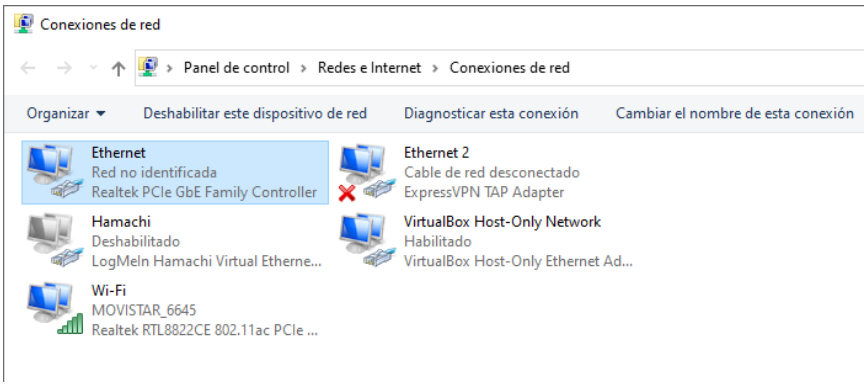

Figura 50: Opciones de red del PC de control

En el adaptador Ethernet

- Click derecho: propiedades/Protocolo de internet versión 4 (TCP/IPv4)
- usar la siguiente dirección IP:

Se deberá introducir una dirección IP compatible con la IP configurada en la placa. En este caso Los compatibles son 192.168.10.xx (xx no puede ser 10, ya que 10 es el identificador de la placa).

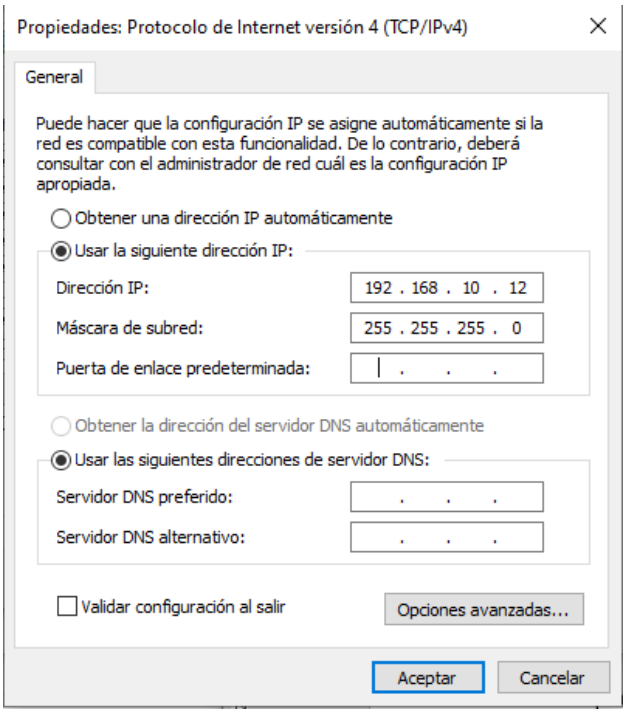

Figura 51: Dirección IP configurada en el PC de control

Una vez hecho esto se inicializa el software de Vivado y se abre las opciones Hardware manager dentro de TASK. (Otra opción es abrir cualquier proyecto a travez de Quiz Start)

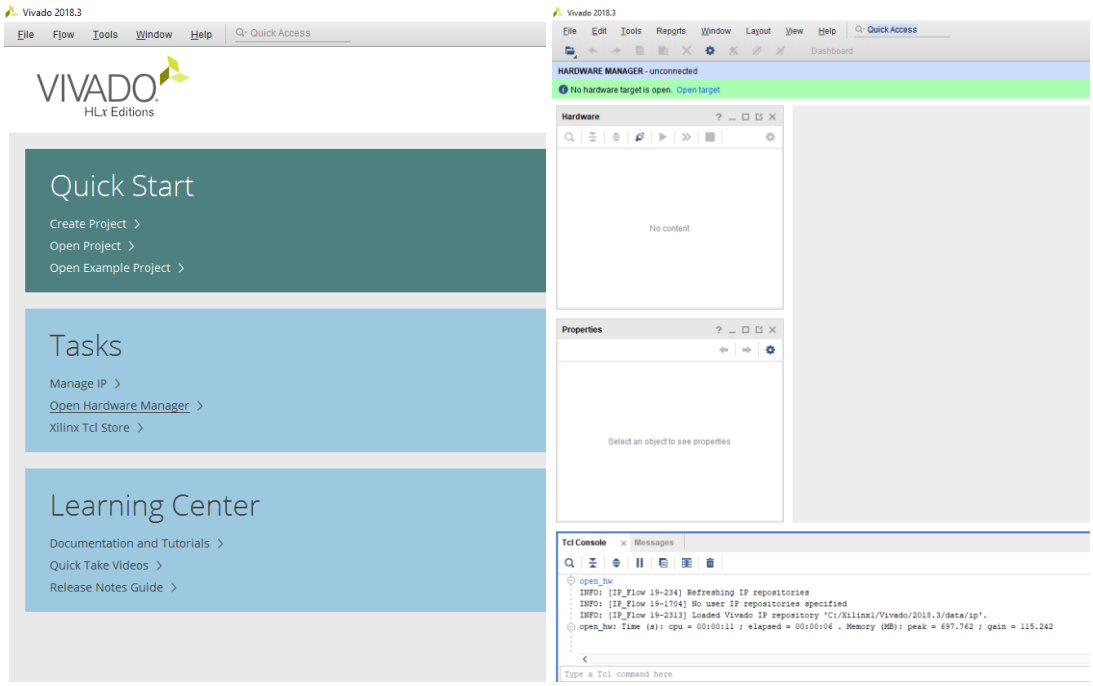

Figura 52: Inicio de software Vivado

En las opciones de abrir tarjeta, se selecciona Open New Target tal y como se observa en la Figura 53

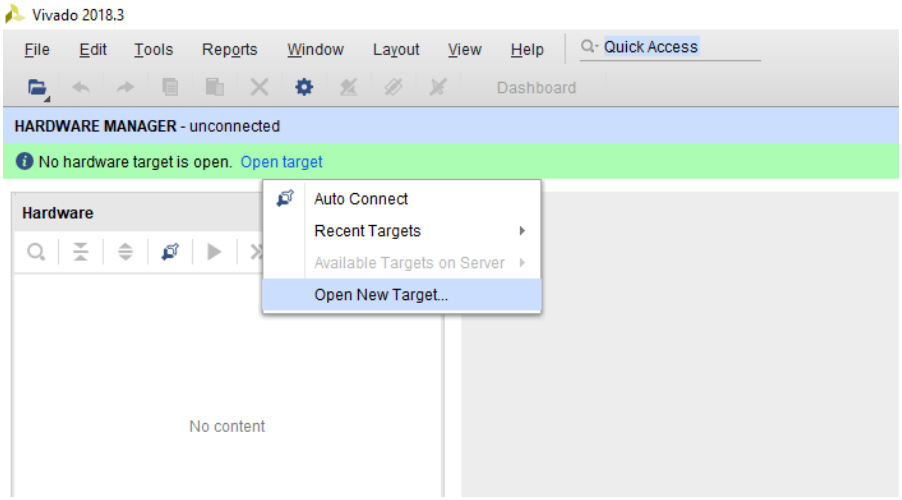

Figura 53: Open New Target

- Seleccionar next
- En la opción Conect to, seleccionar local server (target on local machine)
- Next

Aparecerán las opciones que se muestran en la Figura 54, se deberá seleccionar la opción *Add Xilinx Virtual Cable (XVC).* Tras esto, tal como se observa en la Figura 55, se deberá introducir la dirección IP de la tarjeta que se configuro con anterioridad, en este caso es: 192.168.10.10

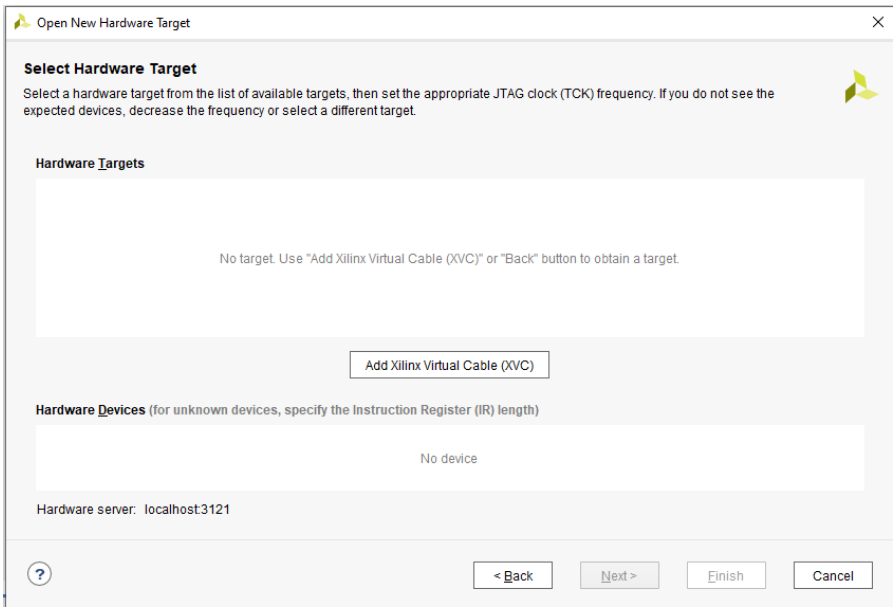

Figura 54: Opciones de selección de tarjeta

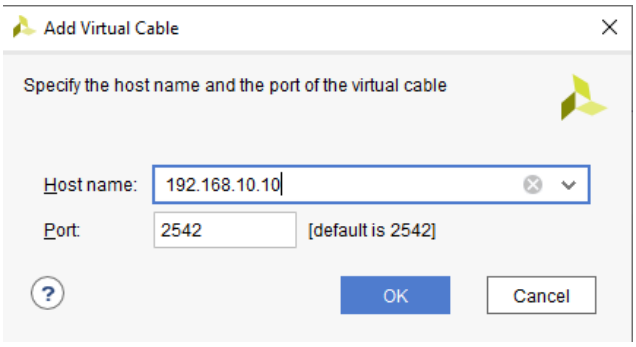

Figura 55: Dirección IP de tarjeta PS

Una vez se introduce la dirección y de da a ok, aparecerá en las opciones la tarjeta de pruebas junto con su dirección IP. De igual forma, en la Figura 57 se observan los mensajes enviados a través de la conexión serie por la UART en la que se ve que PS ha aceptado la conexión TCP.

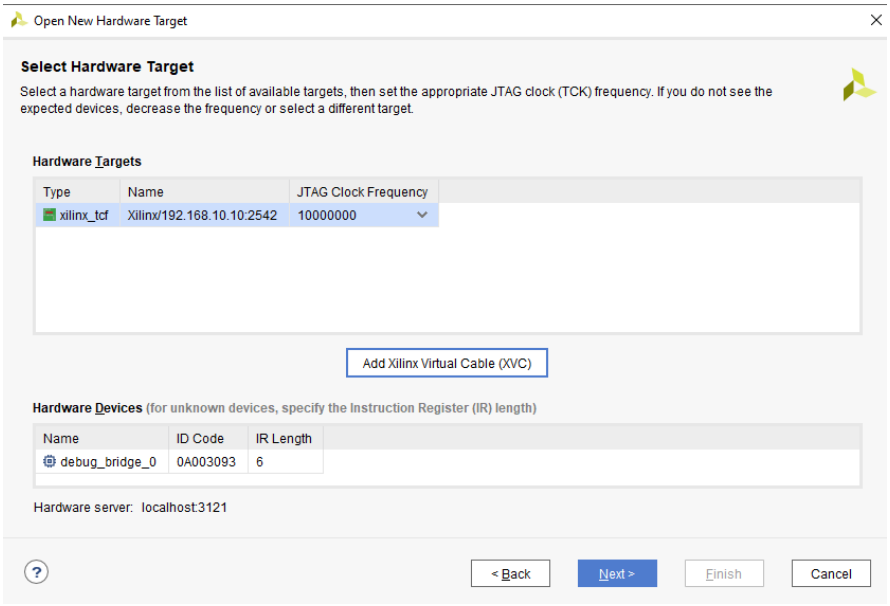

Figura 56: Tarjeta de pruebas

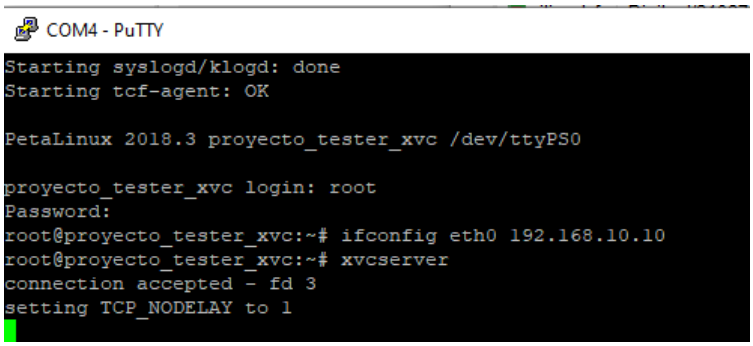

Figura 57: Mensajes en la terminal Serie de la placa

Finalmente, cuando se ha logrado establecer la conexión TCP/IP con la a la placa. En la ventana de hardware se puede ver que el ILA aparece como un Core que está bajo el control del Debug bridge.

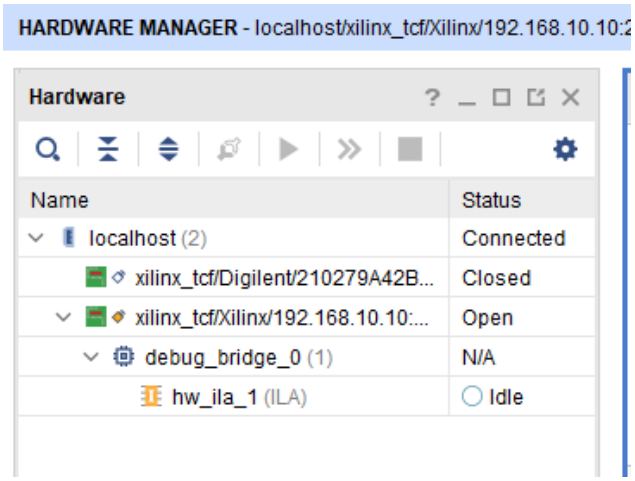

Figura 58:Conexion TCP/IP con la placa

De igual forma, aparecerá una ventana con el Dashboard del ILA.

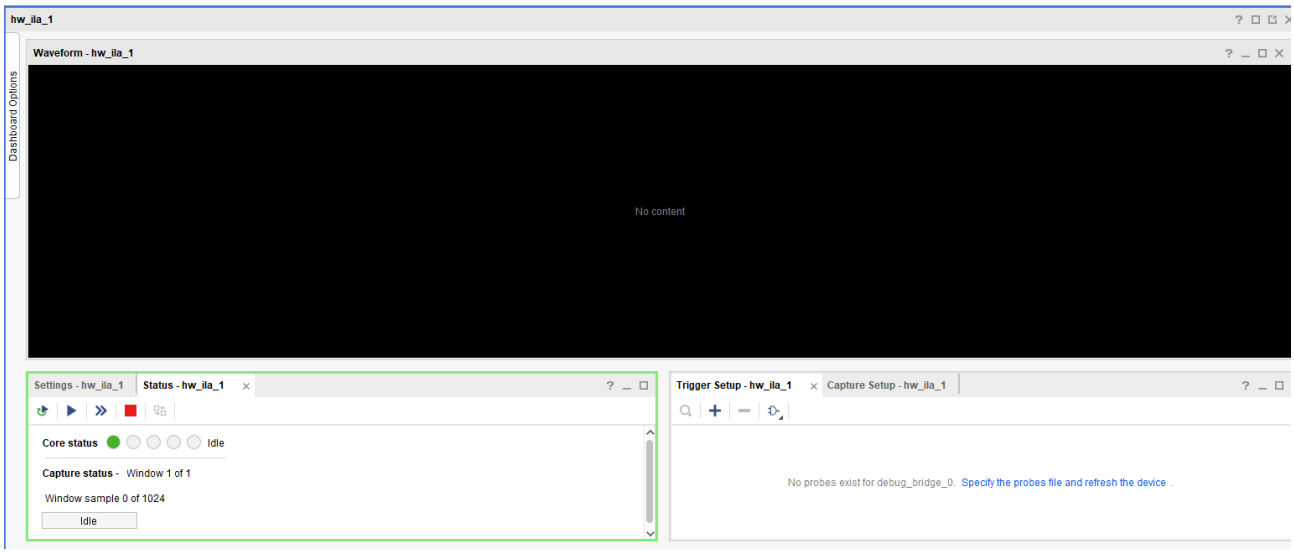

Figura 59: Dashboard de depurador ILA

Como se observa en la Figura 59, el ILA no tiene información de que señales se desea depurar, esto se debe a que no se le ha proporcionado la lista de señales conectadas al ILA.

Como se vio en el punto DISEÑO Y MONTAJE5, durante la creación de la imagen de arranque se proporcionó la descripción de PL (bitstream) mediante el archivo system.bit. Sin embargo, este archivo no contiene la información de las señales que se eligieron para la depuración, la información de las señales a depurar se guarda en un archivo *".xlt"* el cual se genera durante la implementación del diseño en Vivado.

Este archivo se encuentra dentro del proyecto de Vivado en la carpeta run/impl *"nombre-proyecto_wrapper.ltx"* y para que el ILA obtenga esta información se debe pasar la ruta del archivo tal que:

- En la Ventana Triger Setup-hw_ila_1 seleccionar la *"Specify the probes file and refresh the device"*

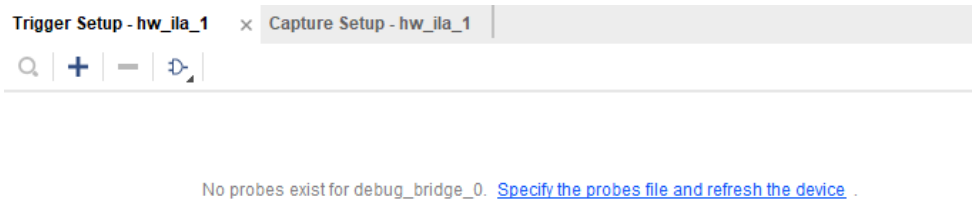

Figura 60: Ventana Trigger Setup-hw_ila_1

- dentro de la carpeta del proyecto seleccionar la ruta *…\project_1.runs\impl_1/desing_1_wrapper.ltx*

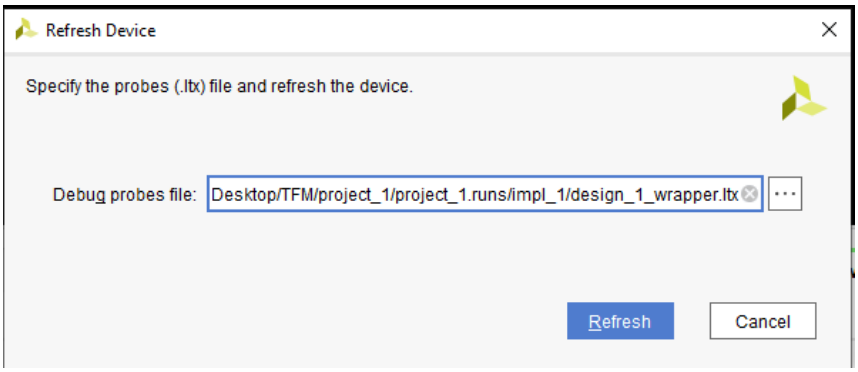

Figura 61: ruta de archivo ltx

Cuando se haya refrescado aparecerán las señales y el sistema está listo para iniciar la depuración.

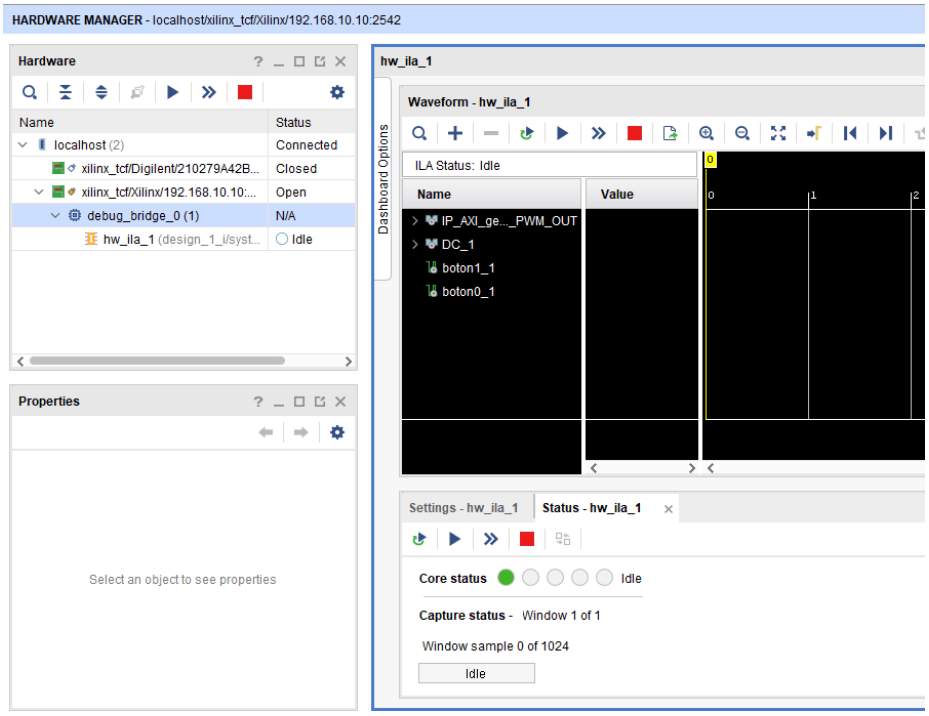

Figura 62: Señales de depuración

Se realizará varias pruebas modificando la configuración del ILA para evaluar diferentes señales, de cada una de las pruebas se tomarán 10 muestras.

- **<u>Depuración 1. Botones de selección</u>**:

  Para esta prueba se ha seleccionado las señales de entrada de los botones 0 y 1 como controles. Se quiere capturar el momento justo en el que se asigna un valor de DC para la señal PWM de uno leds. Siguiendo el funcionamiento del Core de usuario, ese momento seria cuando boton0=0 y boto1 para de 0 a 1. En la Figura 71 se muestra como quedaría la configuración del ILA.

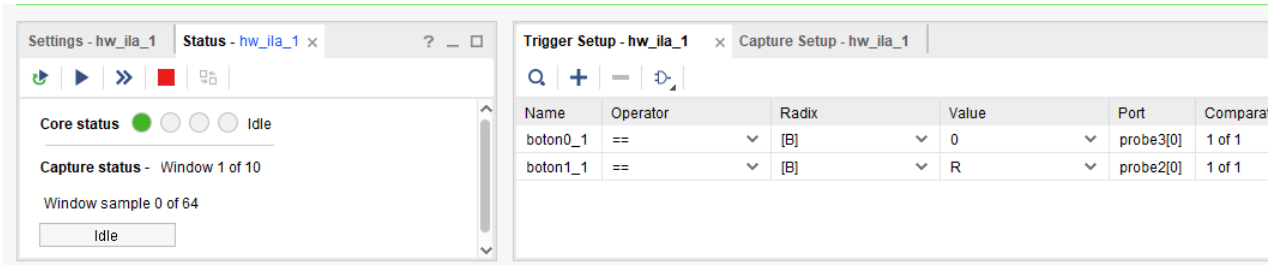

Figura 63: Configuración ILA depuración 1

A continuación, se muestran las muestras obtenidas tras la depuración:

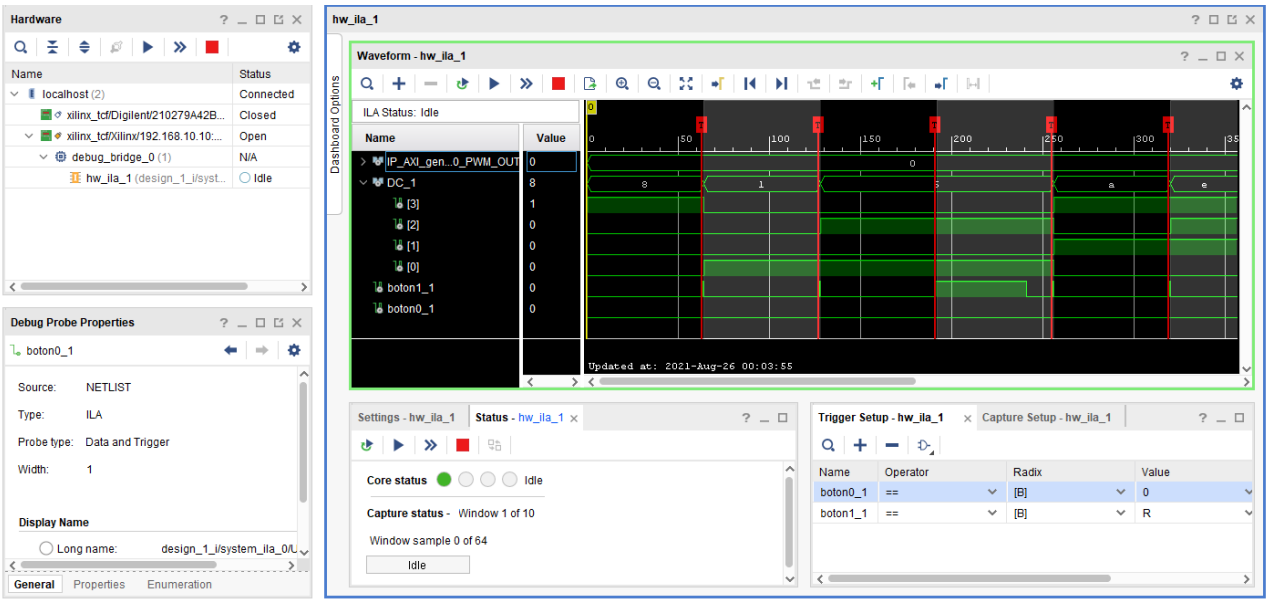

Figura 64: Resultados prueba 1-1

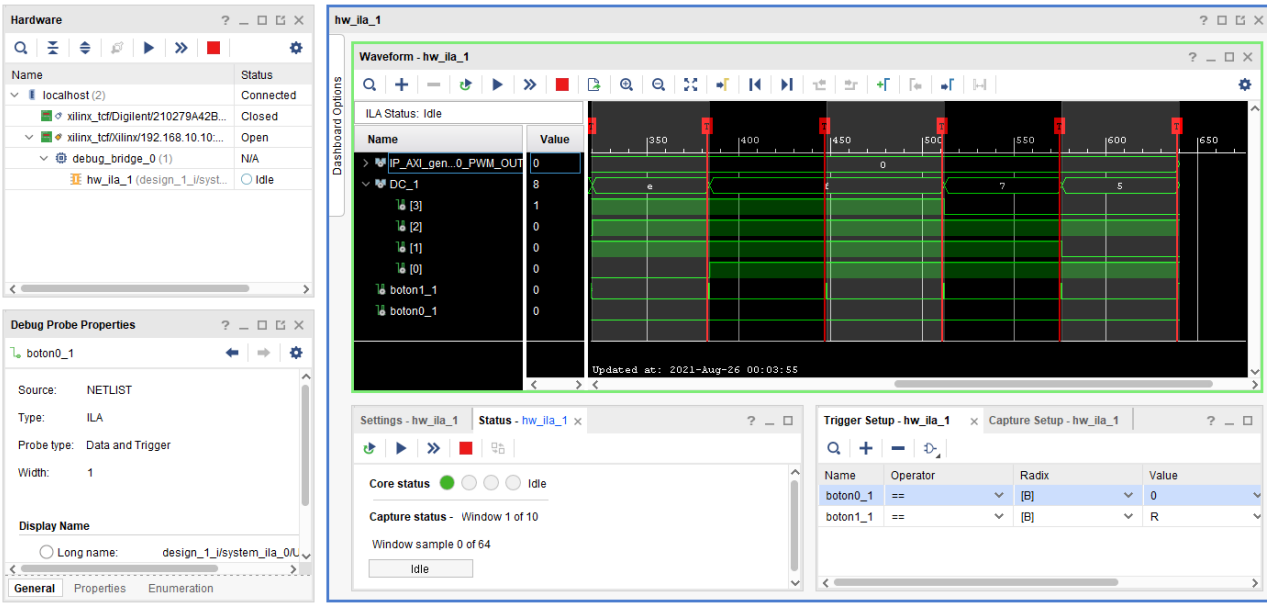

Figura 65: Resultados prueba 1-2

- **<u>Depuración 2. Pruebas de comparación DC</u>**:

Para esta prueba se ha seleccionado la entrada de los switches y el boton1 como controles de muestras. Se quiere observar cómo aumenta el DC de 0 a F para uno de los Leds. En la placa se irá aumentando de 1 en 1 el DC, la muestra se capturará cuando se presione el boton1 o (OR) el valor de DC es F.

La configuración del ILA es: DC=F(1111) o (trigger= OR) transición de boton1 de 1 a 0

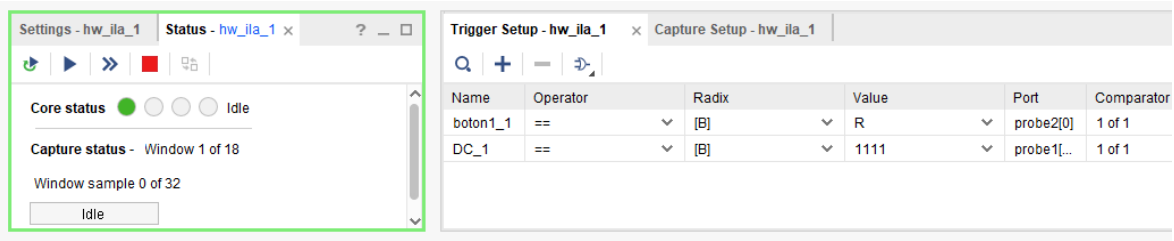

Figura 66: Configuración ILA depuración 2

A continuación, se muestran los resultados

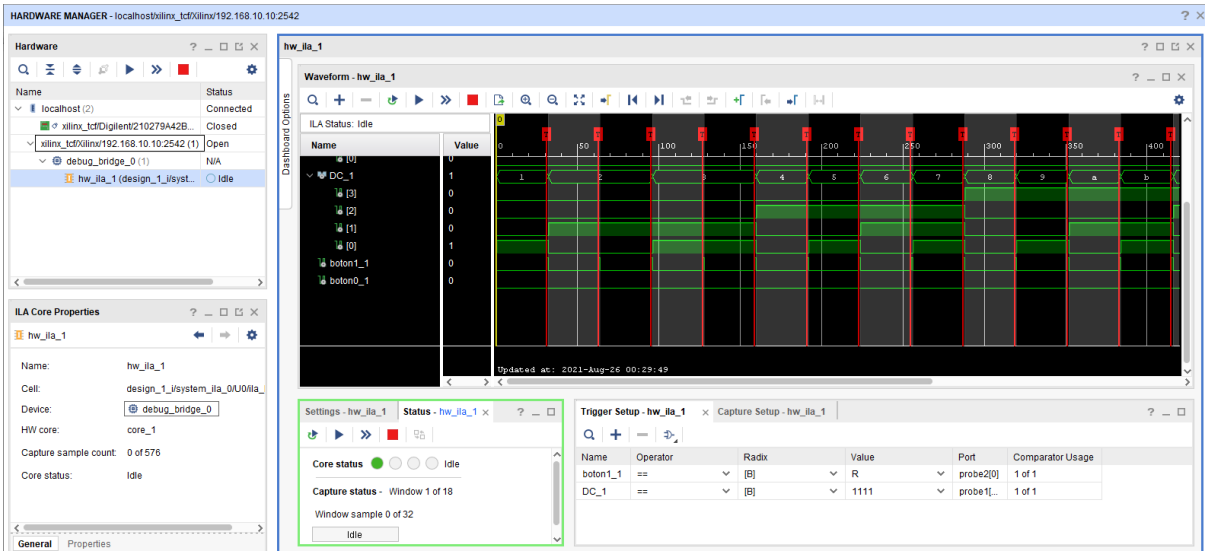

Figura 67: Resultados depuración 2-1

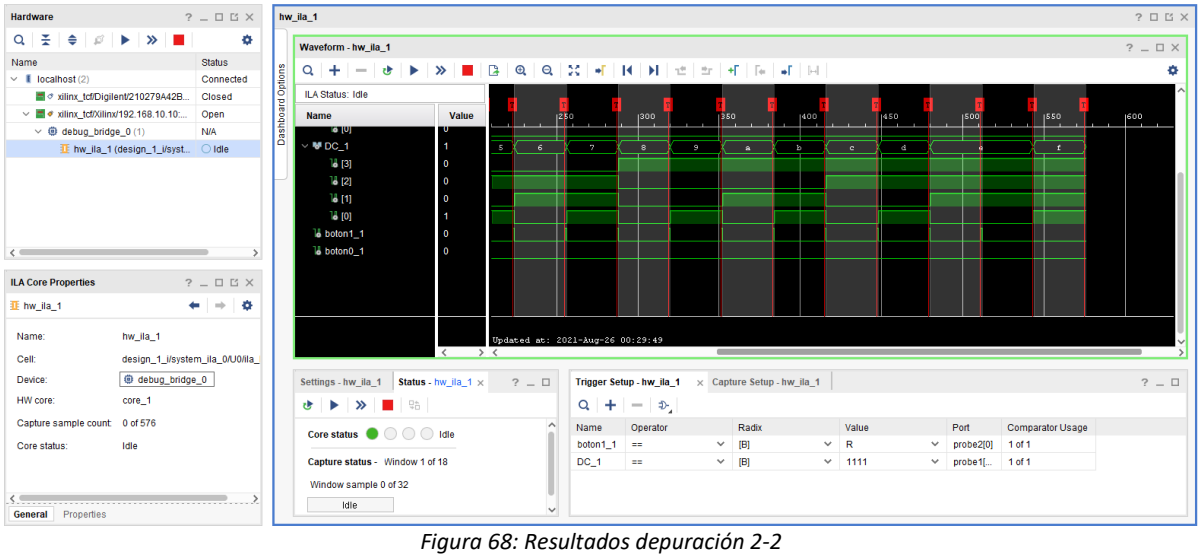

Figura 68: Resultados depuración 2-2

- **<u>Depuración 3. Salida PWM</u>**: para esta prueba se buscará comparar el valor de la señal de salida PWM. Se quiere comparar únicamente 2 de los 4 leds, de modo que se configurara al ILA para que tome muestras cuando PWM_OUT=X1X0. En las siguientes figuras se observan los resultados de ir variando el DC de los 4 leds.

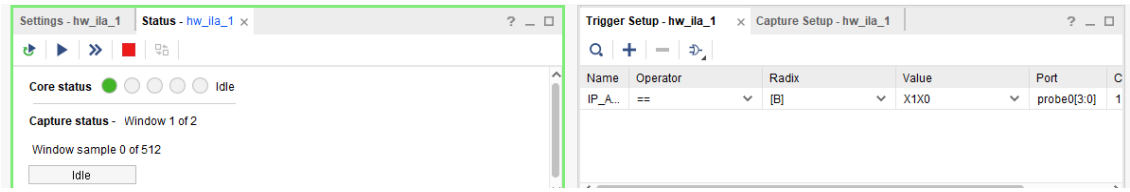

Figura 69: Configuración ILA depuración 3

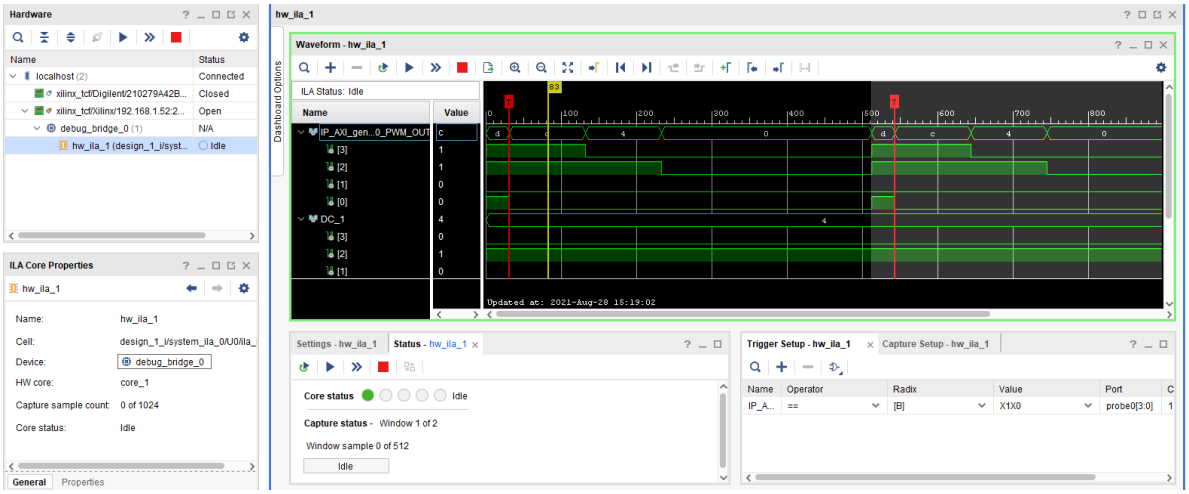

Figura 70: Resultados depuración 3-1

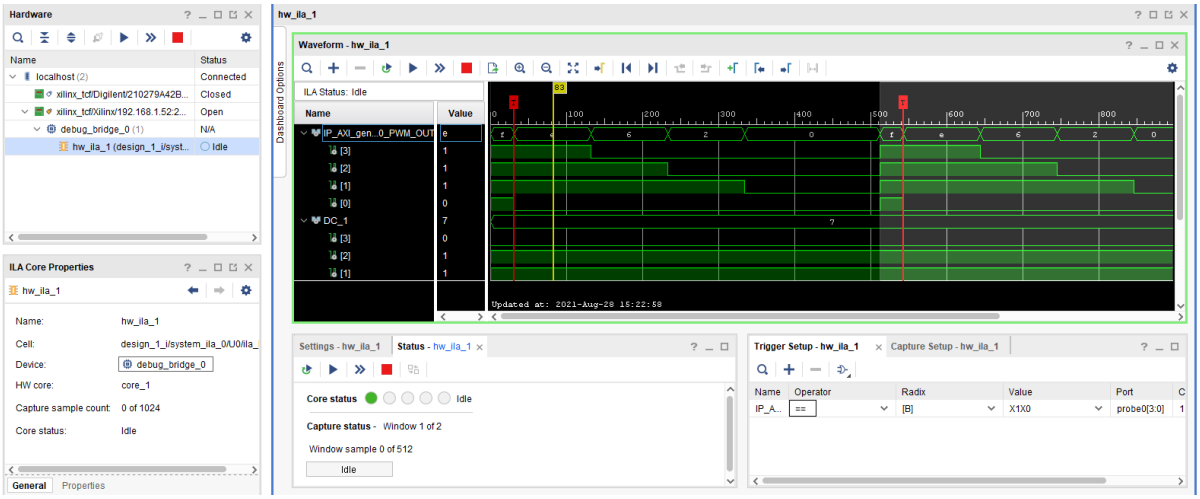

Figura 71: Resultados depuración 3-2

Durante las pruebas no hubo ningún problema de desconexión o velocidad o perdidas de datos. Se puede decir que la conexión fue muy estable ya que la comunicación no se interrumpió en ningún

momento y las depuraciones se realizaron normalmente a gran velocidad y no parecía haber perdido ninguna muestra.

6.2. Segunda Prueba: Conexión remota por red de área local

En resumen, en esta prueba se busca imitar un entorno parecido al de un centro de trabajo o de investigación en el que existe una única red local a la que están conectados la gran mayoría de equipos. Para esta prueba se crea una red de área local mediante un rúter. La placa se conectará al rúter a través de un cable Ethernet, y este último emitirá una señal Wifi a la que el pc de control podrá conectarse remotamente.

Figura 72: Conexión ethernet entre placa Zybo y Rúter

En este caso, cuando la placa se reinicia y carga el kernel de Linux, el rúter asigna una dirección IP al dispositivo dentro de su red local. Por lo que no es necesario configurar los datos eth0 dentro del kernel Linux de la placa

Luego de introducir las credenciales de administrador y la contraseña, se puede obtener la dirección IP del dispositivo enviando el comando "*ifconfig*" a través del terminal serie. En la se observa que la dirección IP asignada para este caso es 192.168.1.52

```
root@proyecto_tester_xvc:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:1E:53
          inet addr:192.168.1.52  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe00:1e53%lo/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1991 (1.9 KiB)  TX bytes:3068 (2.9 KiB)
          Interrupt:26 Base address:0xb000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1%1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 73: Asignación de IP para Zybo

Por otro lado, el PC de control se conecta a la misma red por wifi

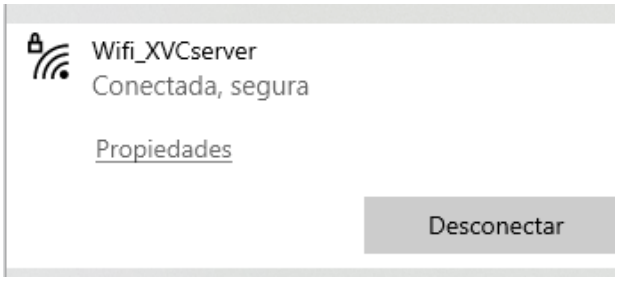

Figura 74:Conexión a red de área local por wifi

Una vez la placa zybo y el PC de control están conectados a la misma red, se puede abrir el software de Vivado para conectar la placa y realizar las pruebas de depuración.

Al igual que en la primera prueba, en el terminal serie de la placa se deberá ejecutar previamente el comando xvcserver para lanzar la aplicación.

```
-   xvcserver
```

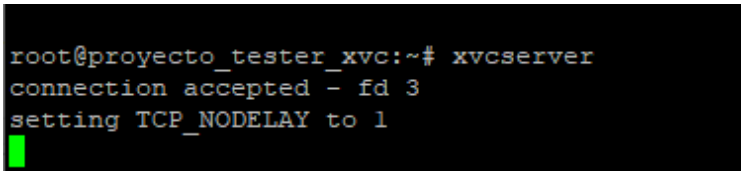

Figura 75: Comando xvcserver

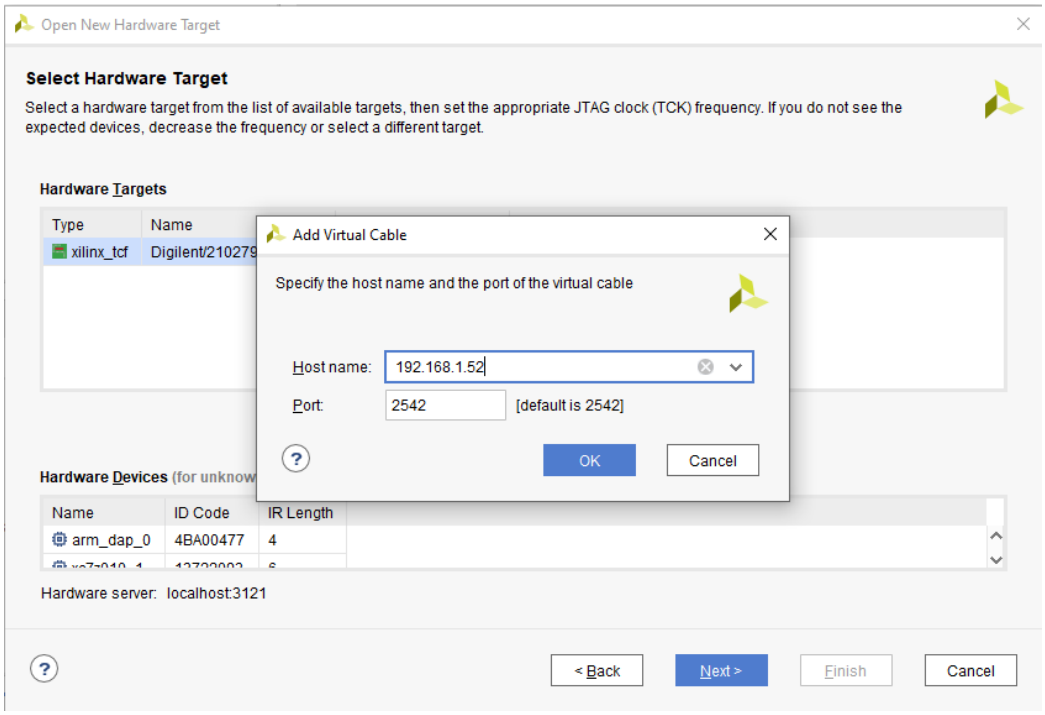

Figura 76: IP DE Zybo

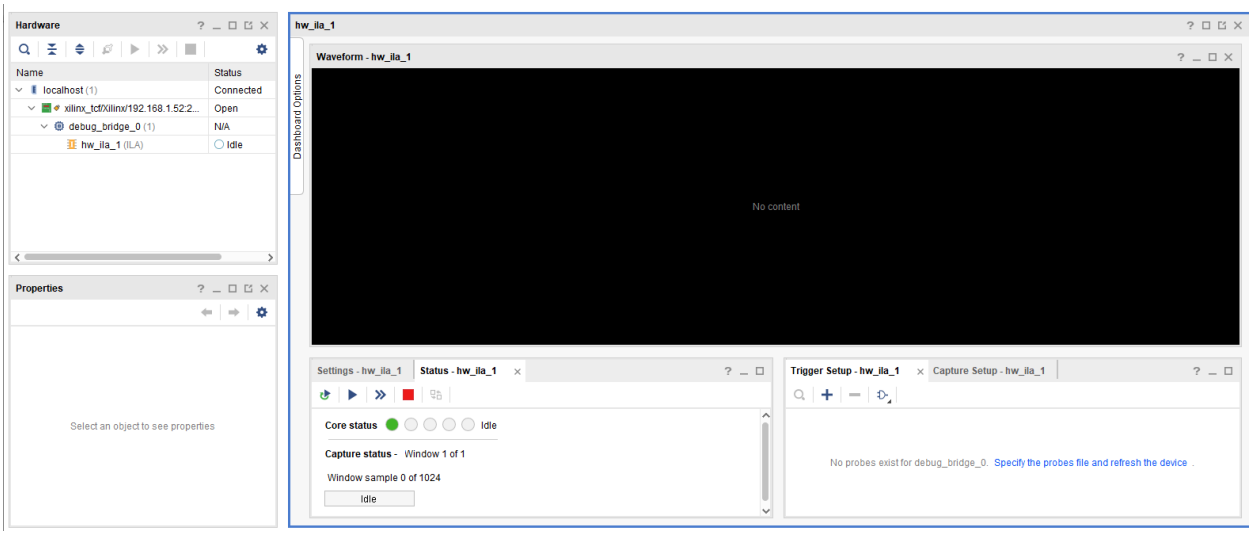

Figura 77: Establecimiento de conexión remota

Del mismo modo que en la prueba anterior, se debe pasar el archivo ".ltx" con la información de las señales que se desean depurar.

Para este caso se realizaron las pruebas de depuración con las mismas configuraciones que antes para observar el comportamiento del sistema y verificar si tiene la misma estabilidad que la prueba anterior. A continuación, se presentan los resultados de las depuraciones.

- **<u>Depuración 4. Botones de selección</u>**:

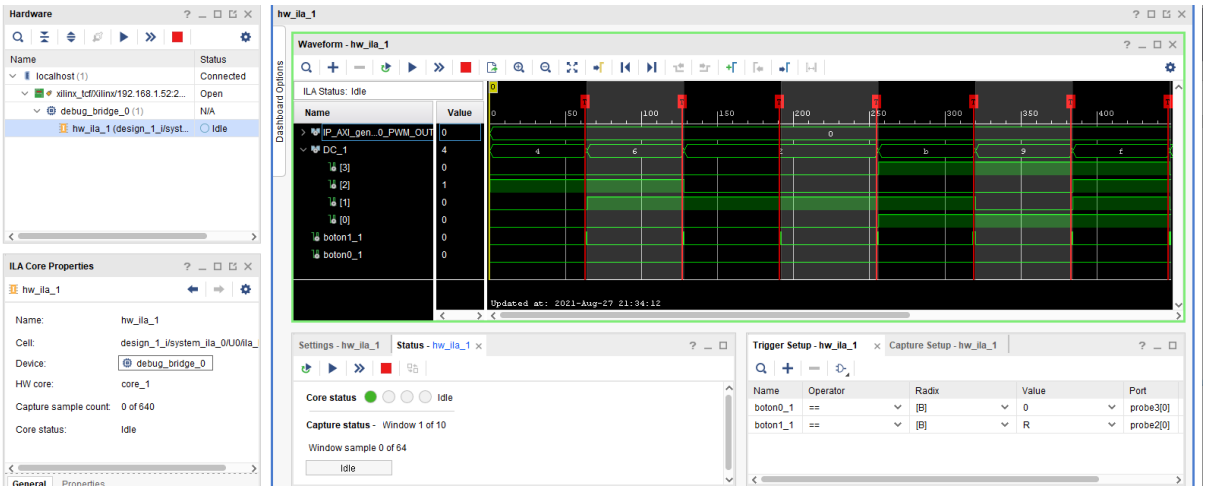

Figura 78: Resultados depuración 4-1

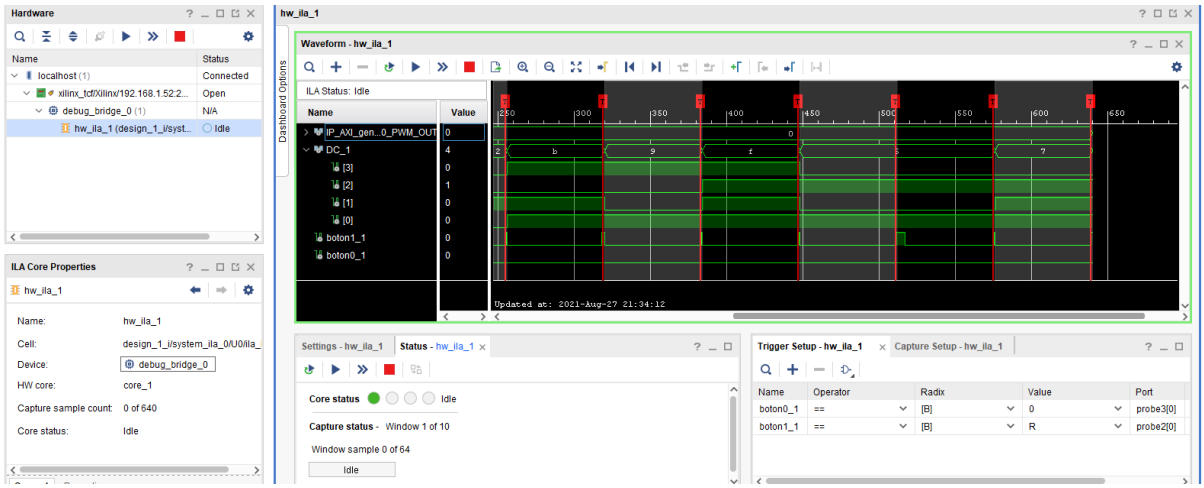

Figura 79:Resultados depuración 4-2

- **<u>Depuración 5. Pruebas de comparación DC</u>**:

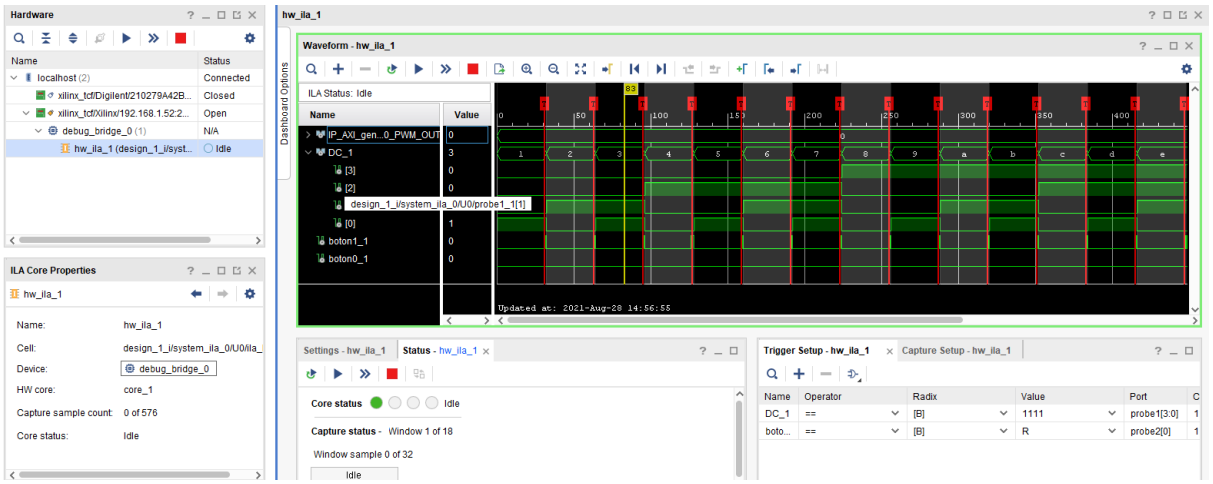

Figura 80:Resultados depuración 5-1

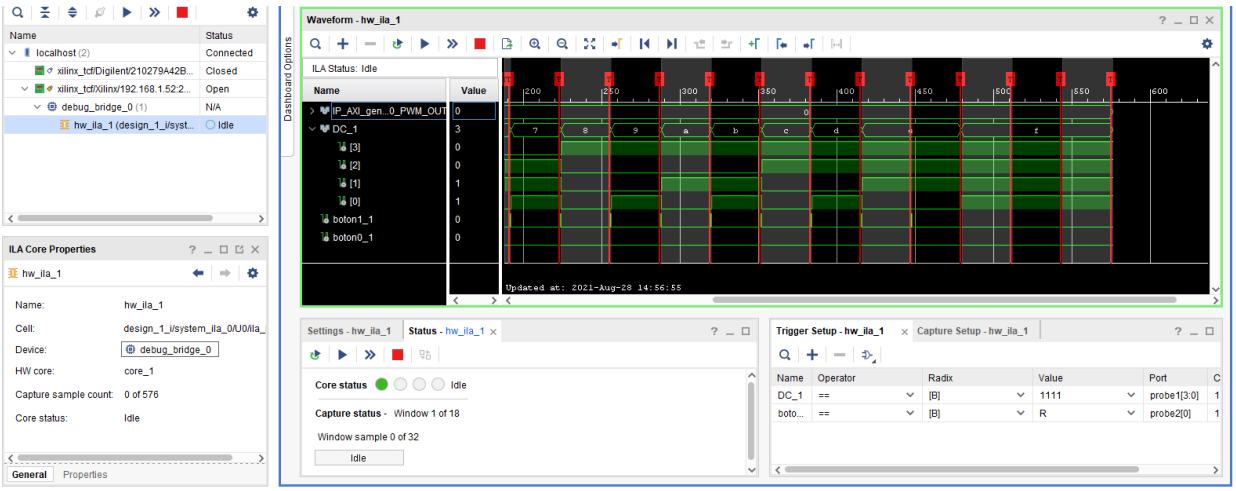

Figura 81: Resultados depuración 5-2

- **<u>Depuración 6. Comparación Salida PWM</u>**:

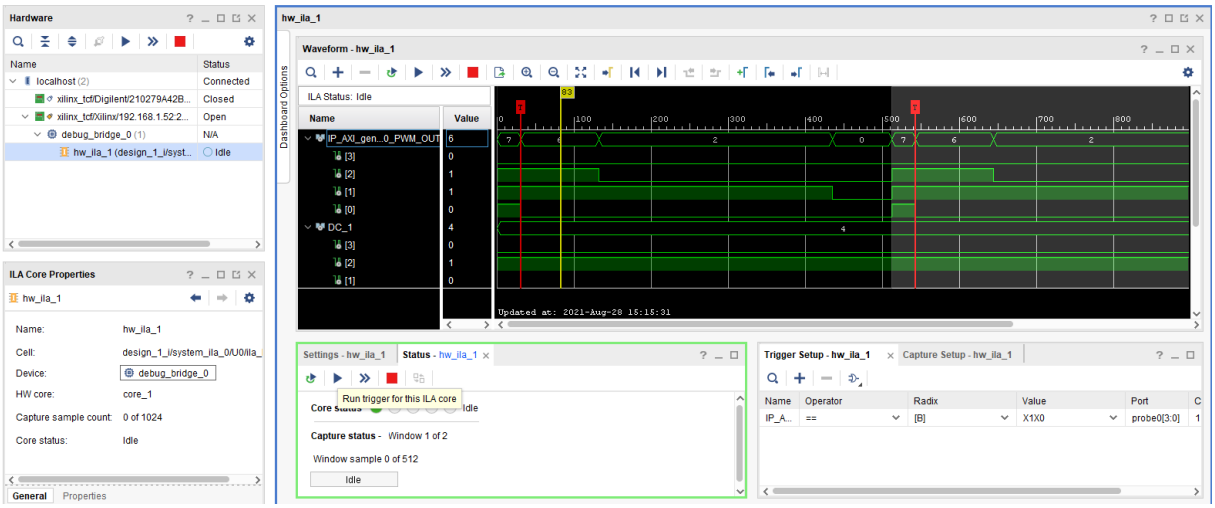

Figura 82: Resultado depuración 6-1

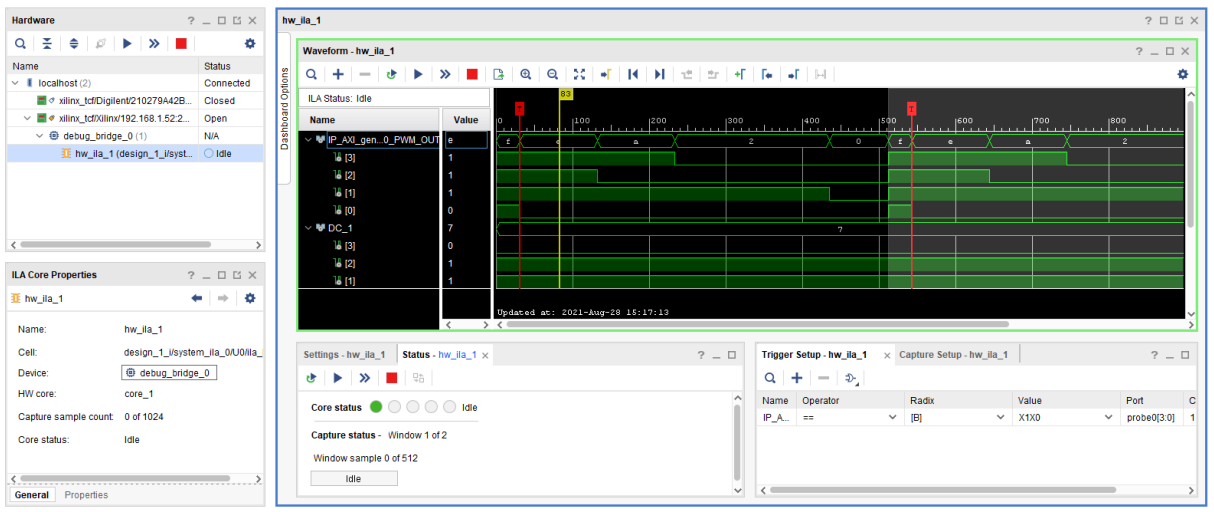

Figura 83: Resultados depuración 6-2

Al igual que la primera prueba, en la que se realizó una conexión directa con el cable Ethernet, durante estas depuraciones se tuvo gran estabilidad y no hubo ninguna interrupción durante la comunicación. En este caso, como la conexión entre los dispositivos era a través de Wifi, probablemente la velocidad de transmisión de datos se vio reducida, pero esto no era perceptible y la sensación era la misma que realizando depuraciones con conexión directa.

6.3. Tercera Prueba: Conexión remota por Internet (VPN)

En esta prueba se busca imitar un entorno más real, en el que la placa Zybo y el PC de control se encuentren en áreas separadas y no estén conectados a la misma red local, en este caso, la conexión entre ambos dispositivos se realizará a través de internet.

Por un lado, la placa Zybo se conectará a una red de área local con acceso a internet, para ello se utilizará el router de la prueba anterior. Por otro lado, se conectará el PC a otra red con acceso a internet ya sea a través de Ethernet, wifi o cualquier otro medio.

Para conectar el PC de control a la red local de la placa se ha creado un servidor VPN, Virtual Private Network, que daría acceso a la red mediante un usuario y contraseña.

Para crear el servidor VPN se ha utilizado las herramientas de Windows de un PC local conectado a la red de la placa. En resumen, este PC local serviría de punto de acceso a la red de la placa, y hará las funciones de enrutamiento de tramas TCP entre el PC de control y la placa Zybo.

En la Figura 84 se observa el diagrama general de la conexión entre la placa Zybo y el PC de control a través de una red virtual privada en Internet.

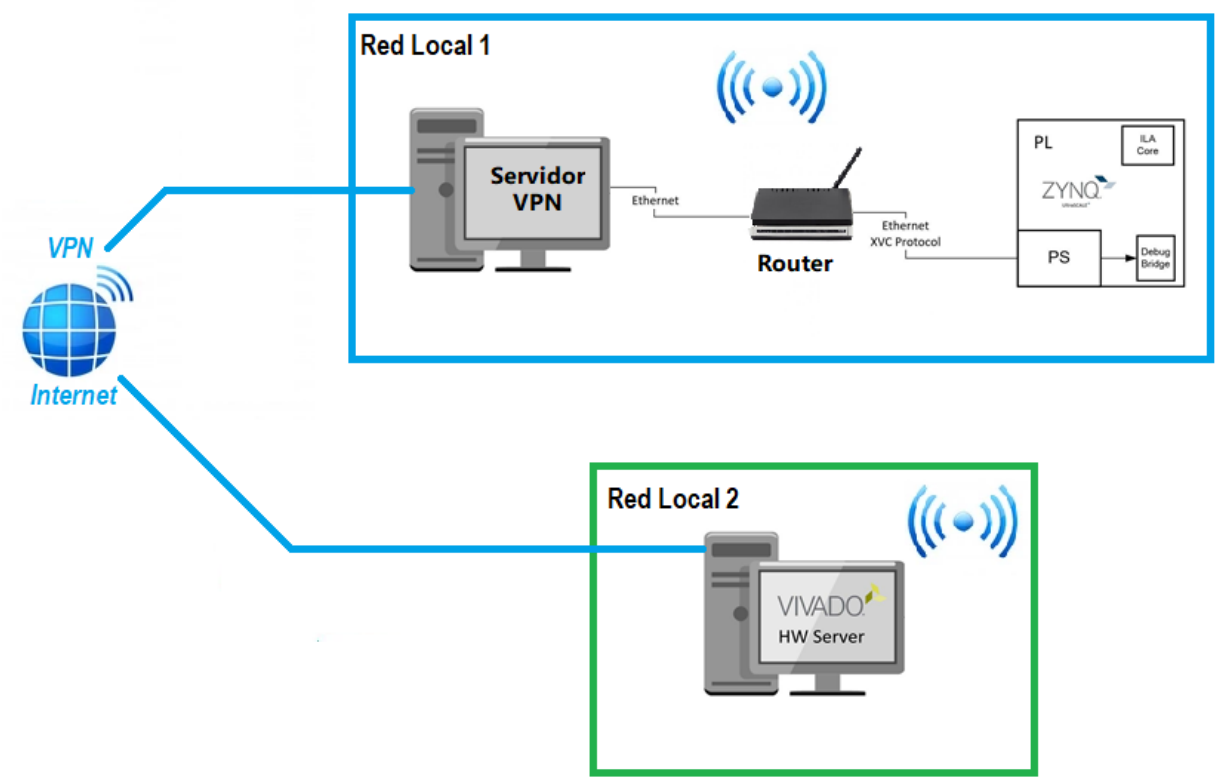

Figura 84: Diagrama b de conexiones mediante internet-VPN

Tanto la placa Zybo como el PC local estarán conectados al Router de la prueba anterior, este último provee de internet a la red local 1.

En la red local 2 se utiliza un teléfono móvil para actuar como antena Wifi, de este modo se puede conectar el PC de control a internet utilizando los datos móviles. En la Figura 85 se puede observar al

PC conectado a la red wifi “Redmi” (red móvil) y a la red “xvc_vpn” que sería la red privada entre el PC local y de control.

Figura 85: Conexiones del PC de control

En la Figura 86 se puede observan las configuraciones de red del PC de control. Se puede comprobar que la red “xvc_vpn” ha concedido acceso a la red local de la placa *192.168.1.XX*. Para este PC se ha establecido la dirección IP 192.168.1.235 y ya se podría comunicar con la placa Zybo cuya dirección IP asignada era 192.168.1.52.

```
Adaptador PPP xvc_vpn:

   Sufijo DNS específico para la conexión. . :
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.1.235
   Máscara de subred . . . . . . . . . . . . : 255.255.255.255
   Puerta de enlace predeterminada . . . . . :

Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . :
   Vínculo: dirección IPv6 local. . . : fe80::d556:f86d:9397:b977%4
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.43.13
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 192.168.43.1
```

Figura 86: Configuraciones de red de PC de control

Una vez se ha conectado la VPN, los pasos a seguir son exactamente los mismos que antes. En el software de Vivado se abre el Hardware manager y se abre una nueva conexión local, se introduce la dirección IP de la placa y se espera a al establecimiento de la comunicación.

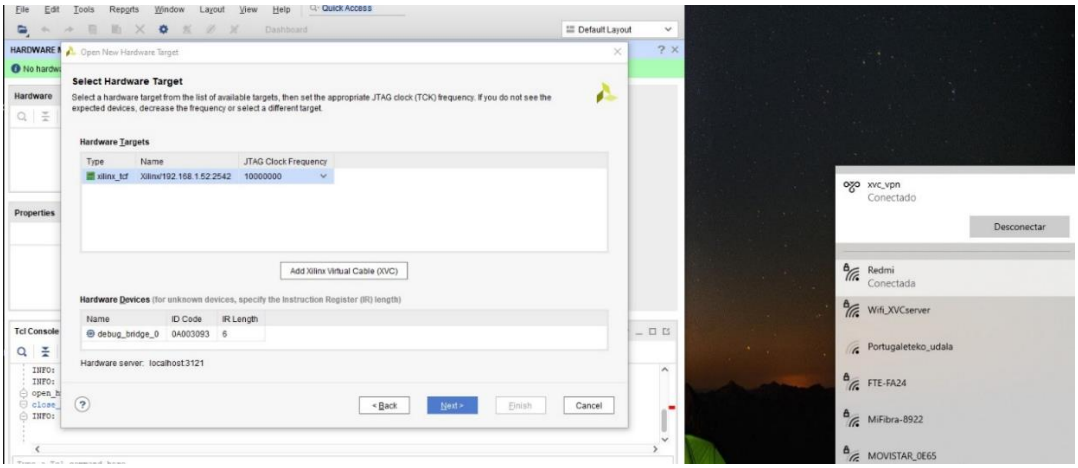

Figura 87: Software de depuración de Vivado conectado a Zybo a través de Internet

A continuación, se muestran las pruebas de depuración realizadas a través de internet.

- **<u>Depuración 7. Botones de selección</u>**:

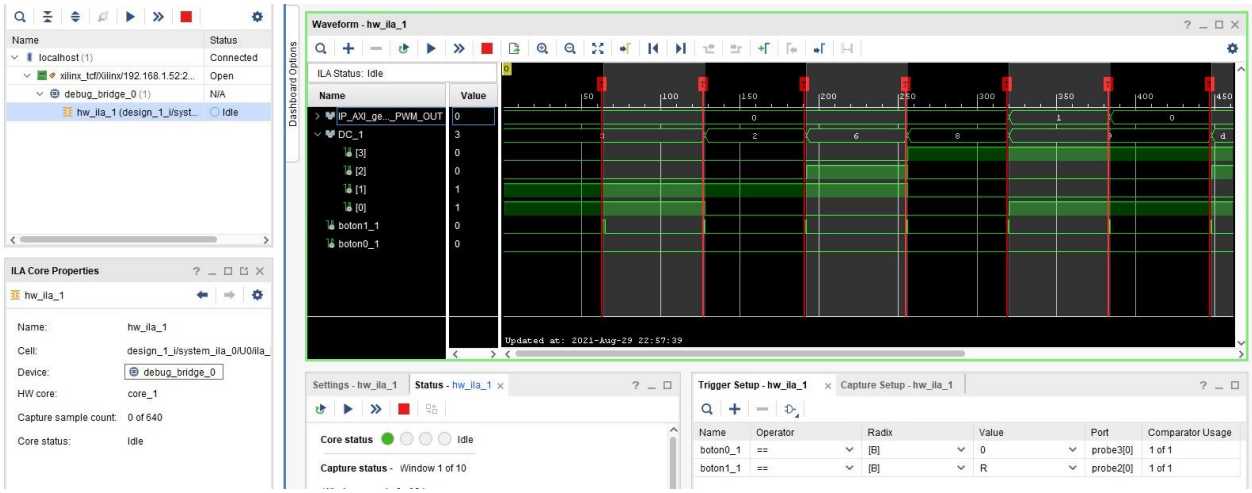

Figura 88: Resultados depuración 7-1

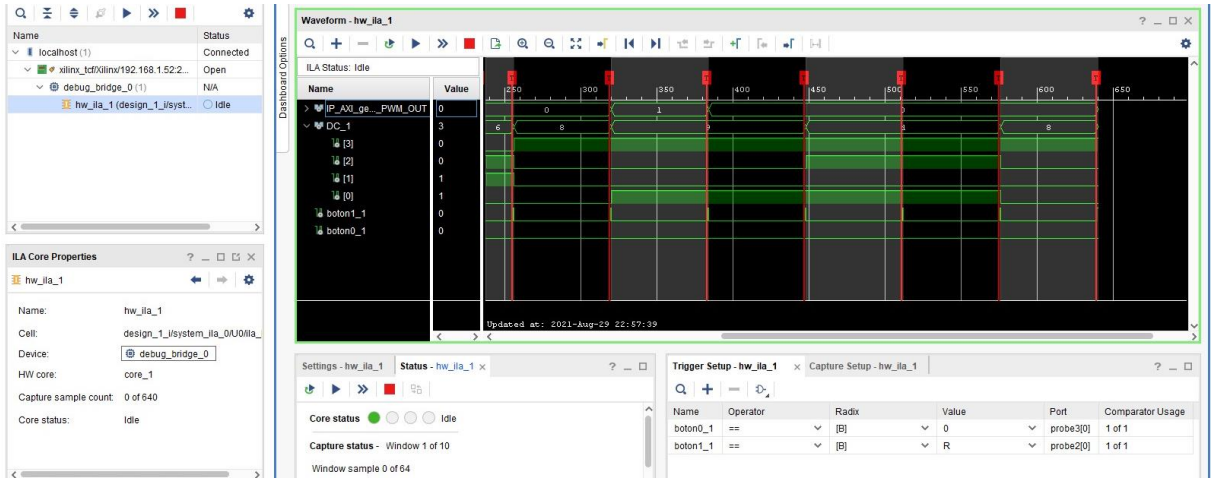

Figura 89:Resultados depuración 7-2

- **<u>Depuración 8. Pruebas de comparación DC</u>**:

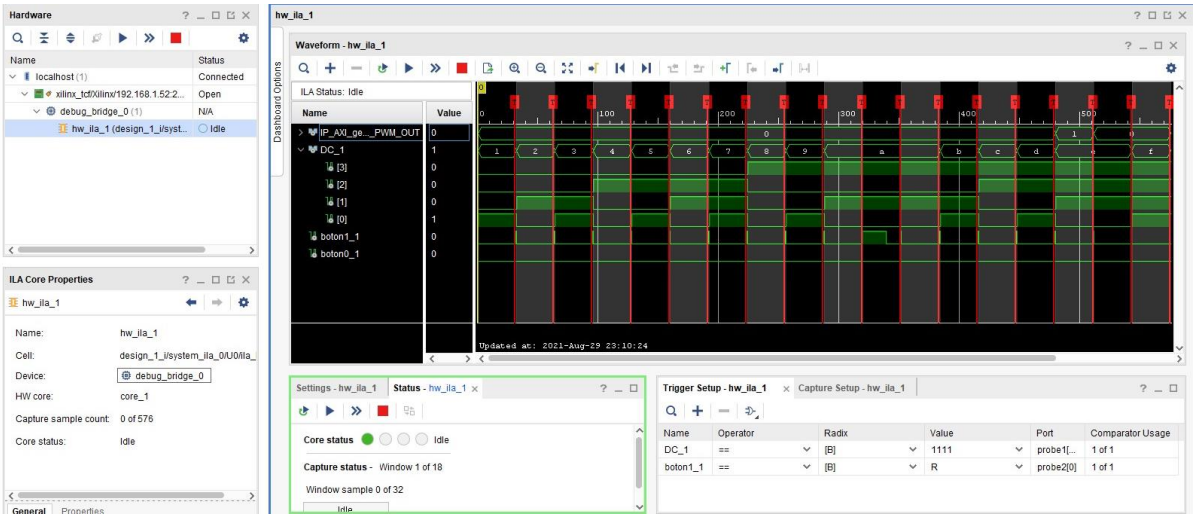

Figura 90:Resultados depuración 8-1

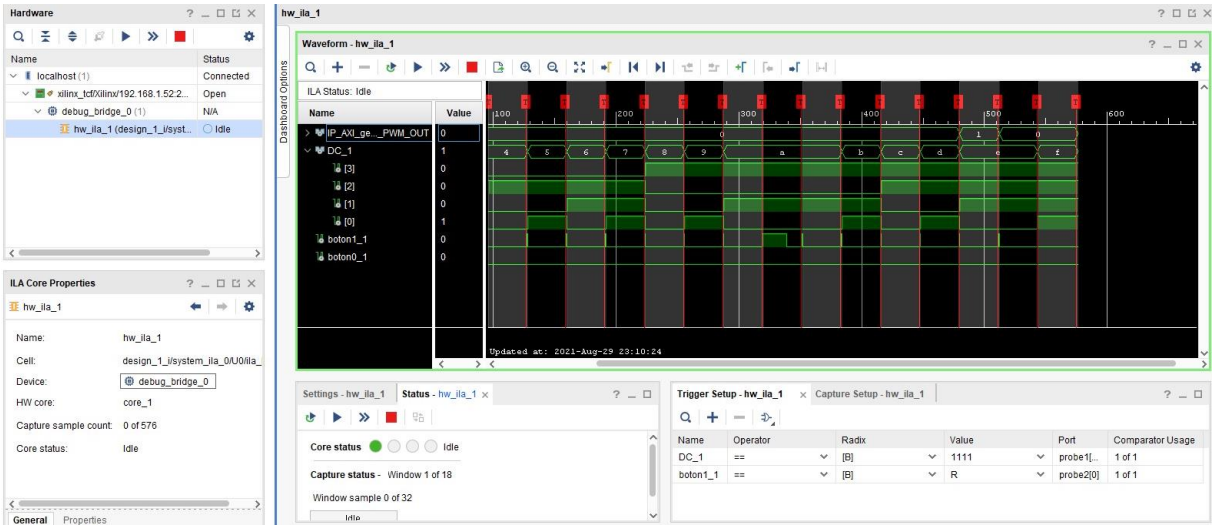

Figura 91: Resultados depuración 8-2

- **<u>Depuración 9. Comparación Salida PWM</u>**:

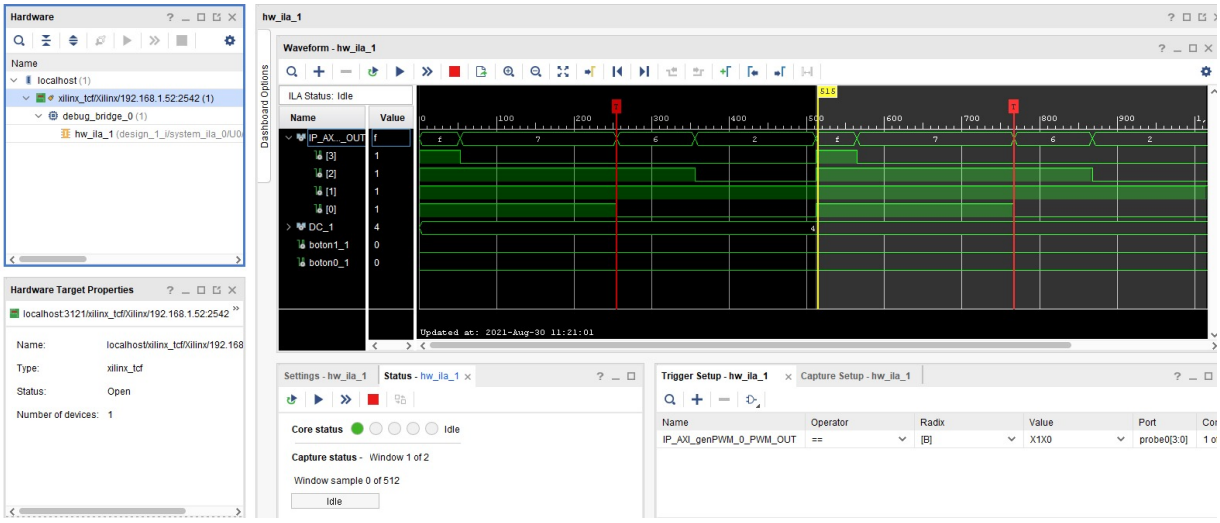

Figura 92: Resultado depuración 9-1

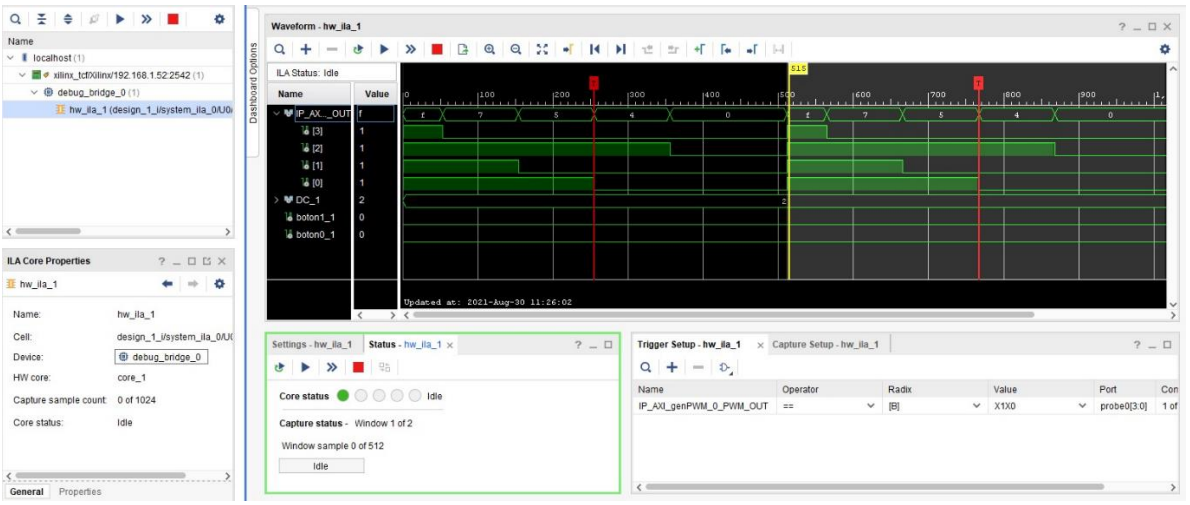

Figura 93: Resultados prueba 9-2

Durante estas últimas depuraciones sí que hubo grandes diferencias respecto de las anteriores pruebas anteriores. Por un lado, la conexión tenía gran estabilidad, no hubo interrupciones ni fallos de comunicación.

Por otra parte, la velocidad de transmisión de datos se redujo en gran medida, en el primer caso, las muestras tardaron en llegar y cargar al software de Vivado y entre 5 y 7 segundos, en este caso se tenían 10 muestras tomadas. En la siguiente depuración se tomaron 18 muestras, el tiempo que tardaron en llegar los datos fue de aproximadamente 12 segundos, de igual forma, la tercera y última depuración, donde se tomaron 2 muestras, el tiempo de espera fue de aproximadamente 2 segundos.

7. CONCLUCIONES

El presente proyecto tenía como objetivo principal el diseño de un analizador lógico que pueda ser embebido en un dispositivo FPGA y que permita capturar y extraer las señales lógicas internas del diseño, para posteriormente enviarlas a través de una interfaz Ethernet y obsérvalas en sistemas remotos.

Se ha logrado diseñar un sistema sencillo que no solo es capaz de realizar depuraciones remotas, sin la necesidad de equipos externos, sino que además es compatible con un gran número de dispositivos FPGA disponibles en el mercado. Basándose en las pruebas de depuración realizadas no en uno, si no en varios entornos reales, se puede decir que el dicho objetivo principal se ha cumplido.

En cuanto a la funcionalidad y la estabilidad del diseño, la conexión entre ambos dispositivos era muy buena y no se tuvieron interrupciones en ninguna de las depuraciones realizadas. Por otra parte, durante las pruebas de depuración se encontró que la principal diferencia entre cada uno de los entornos de depuración, es la latencia o el tiempo que tarda en llegar los datos que envía PS al software de Vivado.

En la primera prueba se realizó una conexión directa mediante un cable Ethernet, la depuración realizada fue prácticamente igual a las depuraciones realizadas mediante cables JTAG, los tiempos de comunicación son imperceptibles para las personas.

En la segunda prueba se intercalo un rúter conectado a la placa, y el PC conectado a Wifi. En este caso pasaba exactamente lo mismo que la prueba anterior, es decir la latencia no se percibía en lo absoluto. Sin embargo, esto puede variar dependiendo del uso que se dé a la señal wifi (N.º de equipos conectados y ancho de banda que utilice cada uno), la intensidad de señal y la frecuencia de la banda. Hay que tener en cuenta que, para poder conectarse al rúter por Wifi, es necesario el protocolo CSMA/CA esto permite al rúter, antes de que establezca la conexión y emitir la señal, comprobar si el canal está libre, mientras más equipos haya conectados, más lleno estará ese canal, esto se traduce en una mayor lentitud a la hora de enviar la señal y por tanto mayor latencia.

Así mismo, el estándar Wifi IEEE 802.11g (la que se usó para esta prueba) opera en la banda de 2,4 GHz y es capaz de alcanzar una velocidad de transferencia máxima de 54 Mbps. Para un diseño como el generador de señales PWM, esa frecuencia es más que suficiente para que la latencia no sea perceptible, pero en diseños más complejos, si la cantidad de datos que se quiere enviar durante la depuración es muy elevada, necesitaría más tiempo para cargar todos los datos.

En la última prueba se añadió dos dispositivos más a la red, un PC local como servidor VPN y un teléfono móvil con datos móviles GSM como emisor de señal Wifi. En este caso, la latencia era muy elevada y

se notaba mucho el tiempo que tardaban en cargar los datos al software de Vivado (alrededor de 8 a 12 segundos, dependiendo de la cantidad de muestras que se hayan tomado).

Este problema seguramente se debe a las características de los servidores utilizados para realizar la conexión a internet. Por una parte, el PC local utilizado no tiene un rendimiento adecuado para actuar como servidor VPN y el tiempo que tarda en realizar el enrutamiento de datos es, con diferencia, mucho mayor que los servidores comunes. Por otro lado, el teléfono móvil usado sí que cuenta con la tecnología adecuada para este servicio, sin embargo, este dispositivo utiliza datos móviles GSM cuyo ancho de banda es más pequeño y es más propensa a causar interferencias, esto conlleva una ralentización en el tráfico de datos.

En un principio, se realizaron las pruebas utilizando el teléfono como emisor wifi, a lo que se añaden los inconvenientes mencionados en la prueba anterior. La latencia era muy elevada, llegado incluso a los 12 segundos en cargar los datos durante la segunda depuración (se tomaron 18 muestras).

En un intento de mejorar los tiempos, se realizó nuevamente otra prueba de depuración, esta vez con una conexión directa del teléfono móvil con el PC de control, en este caso los datos móviles se transferían a través de un cable USB. Este cambio redujo el tiempo de carga de datos a 10 segundos, se puede decir con seguridad que el problema de la latencia puede desaparecer o mejoraría en gran medida si se utilizan los equipos y recursos adecuados.

Con respecto al diseño y montaje del proyecto, las herramientas que ofrece Xilinx como el LogiCore IP ILA y LogiCore IP Debug Bridge para el diseño hardware, así como también las herramientas de Petalinux para el diseño software, han supuesto una gran ayuda y han facilitado el diseño del proyecto, en donde la mayor parte del trabajo ha sido la búsqueda de las herramientas adecuadas, estudiarlas y combinarlas para lograr el objetivo.

Sin embargo, la falta de información de cómo usar, configurar y combinar dichas herramientas, han generado grandes dificultades y tropiezos durante el desarrollo del mismo. Por un lado, la información que Xilinx proporciona sobre el Core Debug Bridge se limita a únicamente los modos de configuración que tiene, sin especificar como se tiene que poner en marcha el sistema, ni la necesitad de software de control. Salvo por algún foro de soporte técnico de Xilinx, prácticamente no se tiene documentación alguna, de cómo utilizar y las posibilidades que tiene esta herramienta.

Lo mismo pasa con la aplicación xvcServer, esta aplicación software fue diseñada para ser utilizada con un diseño hardware distinto al Core Debug Bridge [12], con lo que tampoco se tiene mucha información de como configurar kernel del Linux para que la aplicación se compatible con el Core.

A pesar de todos estos inconvenientes, el trabajo realizado de investigación, análisis y estudio, ha permitido adquirir el conocimiento adecuado para cumplir el objetivo final del proyecto. Como logro añadido, se ha plasmado en un solo documento, las herramientas, recursos y los pasos necesarios para establecer un diseño como esté, que puede servir de guía o base para diseñadores o investigadores en el futuro.

Por último, con respecto a las líneas futuras de estudio, se puede decir que quedan varios frentes abiertos que pueden llegar a sumar más prestaciones además de las depuraciones remotas. Las dos líneas de estudio principales que se consideran son:

- Por una parte, una de las crecientes necesidades dentro de las depuraciones, es que estas puedan ser realizadas en tiempo real. En este caso, aunque se utilicen los equipos y recursos adecuados para realizar las conexiones entre los dispositivos de control y PS, que mejoren por mucho los tiempos de transmisión de datos, no sería suficiente para cumplir con los requerimientos de tiempo real.

- También queda pendiente plantear un sistema que, además de realizar depuraciones remotas, pueda usarse también para la programación de la placa. En el sistema de este proyecto, no es necesario utilizar el puerto JTAG para realizar la programación, sino que esta se realiza desde una memoria SD insertada en la placa. Sin embargo, el sistema sería todavía más útil, si se logra incorporar esta funcionalidad también de forma remota.

8. REFERENCIAS

[1] G. Knittel, S. Mayer and C. Rothlaender, "Integrating Logic Analyzer Functionality into VHDL Designs," 2008 International Conference on Reconfigurable Computing and FPGAs, 2008, pp. 127-132, doi: 10.1109/ReConFig.2008.77.

[2] Altera Corporation, "SignalTap II Embedded Logic Analyzer Documentation", May 2008, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii53009.pdf

[3] Farnel, "FPGAView™Software for Debugging Altera FPGA Devices with Tektronix Logic Analyzers and Mixed Signal Scopes" http://www.farnell.com/datasheets/1673509.pdf

[4] Xilinx, "LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v1.04a)" June 22, 2011 https://www.xilinx.com/support/documentation/ip_documentation/chipscope_ila/v1_04_a/chipscope_ila.pdf

[5] Xilinx, "Vivado Design Suite User Guide Programming and Debugging October 30, 2019 https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug908-vivado-programming-debugging.pdf

[6] Xilinx, "Integrated Logic Analyzer v6.2, LogiCORE IP Product Guide", PG172 October 5, 2016 https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_2/pg172-ila.pdf

[7] Xilinx, "Debug Bridge v3.0 LogiCORE IP Product Guide Vivado Design Suite" December 5, 2018 https://www.xilinx.com/support/documentation/ip_documentation/debug_bridge/v3_0/pg245-debug-bridge.pdf

[8] Be Van Ngo, P. Law and A. Sparks, "Use of JTAG boundary-scan for testing electronic circuit boards and systems," 2008 IEEE AUTOTESTCON, 2008, pp. 17-22, doi: 10.1109/AUTEST.2008.4662576.

[9] S. Popa, M. Ivanovici and R. -M. Coliban, "Time-multiplexed 10Gbps Ethernet-based Integrated Logic Analyzer for FPGAs," 2020 International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania, 2020, pp. 1-4, doi: 10.1109/ISETC50328.2020.9301115.

[10] Intel, "Intel FPGAs and Programmable Devices" https://www.intel.com/content/www/us/en/products/programmable.html

[11] RocketBoards.org, "Remote System Debug, Replace JTAG connection with Ethernet to enable remote hardware debug" 2016 https://rocketboards.org/foswiki/Projects/RemoteSystemDebug

[12] Xilinx," Xilinx Virtual Cable Running on Zynq-7000 Using the PetaLinux Tools" Abril 2015, https://www.xilinx.com/support/documentation/application_notes/xapp1251-xvc-zynq-petalinux.pdf

[13] Testech, "Logic NavigatorTM for Actel FPGAs Logic Analyzer and Debug Tool for Actel Programmable Logic" http://www.testech-elect.com/fs2/lnav-actel.htm

[14] Xilinx, "XilinxVirtualCable" Febrero 2021 https://github.com/Xilinx/XilinxVirtualCable

[15] "Diligent Reference" "Vivado Version 2015.1 and Later Board File Installation" https://reference.digilentinc.com/software/vivado/board-files?redirect=1

[16] Diligent Reference, "Hardware errata" https://reference.digilentinc.com/programmable-logic/zybo-z7/reference-manual?redirect=1#hardware_errata

[17] Xilinx, "Zynq-7000 SoC PCB Design Guide" Marzo 2019

https://www.xilinx.com/support/documentation/user_guides/ug933-Zynq-7000-PCB.pdf

[18] Kernel, "The Linux kernel user's and administrator's guide" https://www.kernel.org/doc/html/latest/admin-guide/index.html

[19] "Sistemas SoPC con Sistema Operativo Linux" Diseño de un Sistema SoPC para comunicaciones Ethernet, Master en Sistemas Electrónicos Avanzados, Universidad del País Vasco.

[20] Diligent Reference, "Zybo Reference Manual" https://digilent.com/reference/programmable-logic/zybo/reference-manual

[21] Sumiła M., Miszkiewicz A. (2015) Analysis of the Problem of Interference of the Public Network Operators to GSM-R. In: Mikulski J. (eds) Tools of Transport Telematics. TST 2015. Communications in Computer and Information Science, vol 531. Springer, Cham. https://doi.org/10.1007/978-3-319-24577-5_25

[22] M. Petrova, L. Wu, P. Mahonen and J. Riihijarvi, "Interference Measurements on Performance Degradation between Colocated IEEE 802.11g/n and IEEE 802.15.4 Networks," Sixth International Conference on Networking (ICN'07), 2007, pp. 93-93, doi: 10.1109/ICN.2007.53.

[23] G. Knittel, S. Mayer and C. Rothlaender, "Integrating Logic Analyzer Functionality into VHDL Designs," 2008 International Conference on Reconfigurable Computing and FPGAs, 2008, pp. 127-132, doi: 10.1109/ReConFig.2008.77.

[24] C. Yajun, C. Qingshan, Z. Lianqing, G. Yangkuan and P. Zhikang, "Signal Tap-II Based Debugging Approach for the Data Acquisition System of Multi-joint Coordinate Measuring Machine," 2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control, 2012, pp. 1182-1184, doi: 10.1109/IMCCC.2012.277.

[25] A. Sukhanov, I. Sukhanov, S. Kim, A. Shutov and S. Bazylev, "Online Monitoring and Remote FPGA Configuration Using JTAG Over Ethernet," 2007 15th IEEE-NPSS Real-Time Conference, 2007, pp. 1-2, doi: 10.1109/RTC.2007.4382778.