Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

Configuración y puesta en marcha de módulos de comunicación inalámbrica en un sistema reconfigurable

Autor: Rodrigo Llorente Aragón

Tutor: Alfredo Gardel Vicente

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

Configuración y puesta en marcha de módulos de comunicación inalámbrica en un sistema reconfigurable

Autor: Rodrigo Llorente Aragón

Tutor: Alfredo Gardel Vicente

TRIBUNAL:

Presidente: Raúl Mateos Gil

Vocal 1º: Pablo Ramos Sainz

Vocal 2º: Alfredo Gardel Vicente

FECHA: 14 de julio de 2017

Agradecimientos

En primer lugar quería agradecer a mi tutor Alfredo Gardel y a Ignacio Bravo por su interés y su ayuda en la realización de este trabajo.

También quisiera agradecer a mi familia, en especial a mis padres, por su confianza y apoyo incondicional a lo largo de todos mis estudios universitarios.

Por último, a todos mis amigos y compañeros con los que he compartido multitud de momentos y experiencias durante estos años universitarios.

Resumen

El presente TFG se engloba dentro de la temática de los sistemas electrónicos reconfigurables. Partiendo de una placa de evaluación ZedBoard, se ha planteado como principal objetivo dotar a esta tarjeta de una conectividad inalámbrica (WiFi).

A lo largo de este trabajo se explicarán detalladamente todas las fases de diseño, tanto hardware como software, necesarias para la puesta en marcha de un módulo de comunicaciones WiLink8. Para llevar a cabo dicho diseño, se emplearán las herramientas incluidas en Xilinx Vivado Design Suite así como el software PetaLinux SDK.

De forma adicional y para comprobar las distintas opciones disponibles, se empleará un módulo WiFi ESP8266 que permite una comunicación inalámbrica más sencilla a cambio de unas prestaciones más modestas.

En el desarrollo de este trabajo, se realizarán distintas aplicaciones que hagan uso de las diferentes capacidades de comunicación inalámbrica. De esta manera, se podrá establecer una conexión con una red ya configurada o generar nuestra propia subred inalámbrica en la cual conectar múltiples nodos reconfigurables como el desarrollado.

Todo lo anterior se ha probado con varias placas ZedBoard, comparando las velocidades alcanzadas en las comunicaciones y la viabilidad del diseño para una posterior implementación o despliegue en una aplicación real.

Palabras clave: SoC – System on Chip, Hardware reconfigurable, Redes inalámbricas, ZedBoard, Sistemas embebidos.

Abstract

The main aim of this project is to use a ZedBoard evaluation board as environment to provide wireless communications (WiFi).

In order to set up a WiLink8 communication module there are some design stages that are necessary to bring all the pieces together, as well as specific hardware and software that will be thoroughly explained along this work. Xilinx Vivado Design Tools and PetaLinux SDK software platform will be the main arms to implement the design.

In addition to the main design, another system will be deployed using an ESP8266 WiFi module that will allow to establish a simpler wireless communication keeping a reasonable performance.

This work will involve various applications and ways of using wireless communications, such as establishing a connection with an already created network or setting up a wireless subnet to connect more reconfigurable boards.

Several tests have been run with ZedBoard devices, comparing connection speeds and thinking about the possibilities of including this kind of system in a finished application ready to work.

Keywords: SoC - System on Chip, Reconfigurable Hardware, Wireless networks, ZedBoard, embedded systems.

Resumen extendido

En la actualidad, la sociedad demanda dispositivos con altas capacidades de procesamiento, bajo consumo y tamaño reducido. Los dispositivos diseñados para realizar un conjunto de tareas específicas y que cumplen con las características anteriores son denominados sistemas empotrados.

Para realizar el diseño de los sistemas empotrados, una opción es emplear un sistema reconfigurable que permite diseñar la parte hardware adaptándolo a las necesidades propias de una determinada aplicación.

Con el objetivo de implementar las distintas aplicaciones que se van a diseñar en este trabajo, se va a utilizar una tarjeta de desarrollo ZedBoard. El principal elemento que compone esta placa es un dispositivo Zynq 7000 diseñado por Xilinx. Este dispostivo puede ser clasificado como *All Programmable SoC* ya que incluye en un único chip una zona de silicio dedicada a hardware reconfigurable (similar al disponible en una FPGA) que se encuentra acoplada a un procesador ARM de doble núcleo.

De esta forma, el diseño de un sistema desarrollado en la tarjeta ZedBoard debe ser dual. Por un lado se debe diseñar la parte hardware del sistema deseado para, posteriormente, llevar a cabo el desarrollo de la parte software.

El diseño hardware se lleva a cabo con la herramienta Vivado 2014.4 de Xilinx. En esta herramienta se configurará el dispositivo Zynq de forma que este tenga habilitadas las características necesarias para que la aplicación que se está desarrollando funcione correctamente. Además, con este programa se añaden los componentes hardware para posteriormente realizar la programación de la parte reconfigurable.

Partiendo del diseño hardware, se debe elaborar el diseño software en el cual se va a programar el comportamiento de la aplicación. Para ello, se emplea un segundo programa denominado Xilinx SDK en el cual se realiza el diseño empleando lenguaje de alto nivel C o C++. Este software además dispone de las herramientas necesarias para llevar a cabo la programación de la parte reconfigurable y la ejecución de la aplicación sobre la tarjeta ZedBoard.

El diseño software descrito anteriormente se corresponde con un sistema denominado bare-metal o standalone que permite ejecutar la aplicación sin ningún sistema operativo. Sin embargo, algunas aplicaciones necesitan un sistema más complejo en el que ejecutarse. En este caso, la herramienta Xilinx SDK genera el FSBL o *First Stage BootLoader* que es el primer ejecutable necesario para el arranque del sistema operativo sobre la tarjeta de desarrollo ZedBoard.

Para generar el resto de archivos que componen el sistema operativo, existen varias posibilidades según las necesidades de la aplicación que se está desarrollando. En este trabajo se emplea la herramienta PetaLinux SDK que permite generar un sistema operativo de reducido tamaño basado en el kernel de Linux. Así, este software genera el SSBL o *Second Stage BootLoader* que se encarga de realizar el arranque del sistema operativo así como su núcleo y árbol de directorios necesarios para su ejecución.

La tarjeta ZedBoard dispone de una gran variedad de periféricos que le permiten comunicarse con otros sistemas. De esta forma, el dispositivo Zynq dispone de controladores para algunas de las tecnologías más utilizadas como SPI, I2C, UART o SDIO entre otros.

Sin embargo, todos los periféricos de los que dispone la placa de desarrollo son cableados. Por ello, en este trabajo se pretende dotar a la tarjeta ZedBoard de conectividad WiFi ya que esta tecnología inalámbrica es una de las más empleadas en la actualidad.

Para ello, se van a diseñar diferentes aplicaciones que emplearán dos módulos WiFi, cada uno de los cuales tienen diferentes características y formas de implementación.

El primero de los módulos que se emplean en este trabajo es el WiLink8 de TI. Para poder conectar este módulo, es necesario emplear el WiLink8 Pmod Adaptor que convierte las 100 conexiones disponibles en el módulo WiFi a dos interfaces Pmod de forma que pueda ser fácilmente conectado a la tarjeta ZedBoard.

Este módulo necesita una serie de drivers para funcionar correctamente. Por ello, se debe generar el sistema operativo, a partir de la herramienta PetaLinux SDK, añadiendo estas bibliotecas. De esta forma, al arrancar el sistema los drivers serán cargados pudiéndose emplear la conectividad inalámbrica.

Para comprobar el comportamiento del módulo WiFi, se van a desarrollar una serie de aplicaciones que hacen uso de las capacidades inalámbricas de las que dispone la tarjeta ZedBoard con el módulo WiLink8.

La primera de las aplicaciones diseñadas permite la visualización y el control del estado de los LEDs disponibles en la tarjeta ZedBoard de forma inalámbrica a través de un dispositivo con capacidad WiFi. Para ello, se emplea una página web ubicada en un servidor ejecutándose en la tarjeta de desarrollo. Para el control y la generación de la página web, se emplea un CGI que adapta la web enviada al navegador según el estado de los distintos elementos del sistema.

La segunda aplicación diseñada consiste en el control de un conjunto de sistemas conectados entre sí por una red local inalámbrica. Un sistema concentrador es el encargado de generar dicha red WiFi y, además, conectar el sistema a una red local externa desde la cual se podrá controlar los diodos LEDs de la tarjeta ZedBoard. Al igual que en la aplicación anterior, el usuario podrá acceder a una página web desde la cual conocer el estado y actuar sobre los distintos LEDs de todas las ZedBoard que conforman el sistema. Cabe destacar que, para establecer la conexión entre el concentrador y el resto de tarjetas de desarrollo, se empleará el protocolo SSH que permite ejecutar de forma remota y segura todos los comandos necesarios.

Una vez desarrolladas las aplicaciones anteriormente descritas con el módulo WiLink8, se va a diseñar una aplicación con un dispositivo WiFi diferente. En este caso, se va a emplear un dispositivo WiFi de bajo coste ESP8266 para desarrollar una aplicación *bare-metal* de forma que se pueda controlar el estado de los LEDs disponibles en la tarjeta ZedBoard de forma inalámbrica. De esta forma, se integra la conectividad WiFi a la tarjeta ZedBoard de forma más sencilla que en el caso del módulo anterior pero con unas capacidades más reducidas.

Por último, se realizarán un conjunto de transferencias con el objetivo de medir la velocidad obtenida empleando distintas aplicaciones y protocolos.

Índice general

Índice de figuras

Capítulo 1: Introducción

En la actualidad, la sociedad demanda sistemas que realicen tareas cada vez más complejas. Además, se desea que estos sistemas consuman la menor energía posible y tengan un tamaño reducido para que puedan ser transportados y empleados en cualquier momento. Esto implica que todas las conexiones que permiten la comunicación del sistema deben ser inalámbricas, evitando de esta forma depender de cables que disminuyan la movilidad del dispositivo. Todos estos dispositivos forman parte de un grupo de sistemas conocidos como sistemas embebidos o empotrados.

Para realizar el diseño de los sistemas empotrados, una opción es emplear sistemas reconfigurables que permiten diseñar diferentes sistemas en una misma plataforma. Estos sistemas permiten diseñar el hardware adaptándolo a las necesidades propias de la aplicación que se desea implementar. Además, una vez realizado el diseño Hardware, se debe llevar a cabo el diseño de la parte Software con la realización de aplicaciones que implementen la funcionalidad del sistema.

1.1 Sistemas empotrados

Un sistema empotrado (también denominado coloquialmente como sistema embebido derivado de la palabra inglesa *embedded*) es un sistema diseñado para realizar un conjunto de tareas específicas al contrario de un sistema de propósito general, como un ordenador personal, que se diseñan con el objetivo de realizar un amplio abanico de tareas.

Los sistemas empotrados deben cumplir una serie de características básicas:

- Deben ser confiables, es decir, deben funcionar sin errores de forma continuada en el tiempo.
- Deben ser fácilmente reparables de forma que el sistema vuelva rápidamente a funcionar correctamente después de un fallo.
- Deben tener una disponibilidad elevada, es decir, deben funcionar la mayor cantidad de tiempo posible.
- Deben establecer comunicaciones confidenciales y autenticadas.
- Deben ser eficientes en relación a la energía que emplean, su tamaño, su peso y su coste.

Los sistemas embebidos obtienen información del exterior a partir de sensores. A partir de esta información ejecutan respuestas por medio de actuadores.

Disponen de distintas interfaces de comunicación que se pueden clasificar en:

- Interfaz con el operador: A través de esta interfaz se establece una relación con los humanos. A este tipo de interfaz pertenecen los botones, interruptores, LEDs, entre otros muchos.
- Interfaz con otros componentes del sistema: Esta interfaz se encarga de obtener y enviar información a partir de las señales que recibe o emite a los diferentes componentes que componen el sistema. A esta interfaz pertenecen las comunicaciones UART, I2C, SPI, entre otras.
- Interfaz con componentes externos: Se encarga del control de las comunicaciones con el resto de sistemas externos. En esta interfaz se incluyen Ethernet, WiFi, Bluetooth, etcétera.

Existe una gran cantidad de áreas donde se emplean sistemas empotrados: industria de transporte como la automovilística, ferroviaria o aeronáutica, industria de telecomunicaciones, medica, militar, y de electrónica de consumo como teléfonos móviles, cámaras de video y fotográficas, etc.

1.2 Sistemas integrados en un único chip (System on Chip – SoC)

Se engloban dentro de la categoría *System on Chip* a todos los sistemas que disponen dentro de un único circuito integrado todos o gran parte de los elementos necesarios para su funcionamiento.

Con la mejora de los procesos de fabricación, cada vez es más fácil integrar más elementos dentro de un mismo espacio. Esta gran integración permite obtener un mayor rendimiento que el obtenido si todos los elementos fueran fabricados de forma independiente y posteriormente conectados entre sí. Esto provoca un menor consumo de energía y una reducción considerable del tamaño del sistema.

Los elementos más comunes que constituyen un SoC son los siguientes:

- **Microprocesador**: Es el elemento más importante del sistema ya que se encarga de ejecutar todos los programas que, a su vez, emplean el resto de elementos que constituyen el sistema. Su función es ejecutar las instrucciones de bajo nivel que conforman un programa de forma secuencial, realizando operaciones de acceso a memoria, operaciones binarias y operaciones aritméticas simples. El núcleo de un SoC también puede estar formado por un microcontrolador que, además de todo lo anterior, dispone de una serie de elementos integrados al procesador como periféricos o memoria. Por último, el núcleo de un SoC también puede estar compuesto por un sistema especializado en el procesamiento de señales digitales o DSP.
- **Módulos de memoria**: Existe una gran cantidad de tipos de memoria diferentes que pueden formar parte de un SoC: ROM (memoria de solo lectura), RAM (memoria de acceso aleatorio), EEPROM (memoria de solo lectura programable y borrable de forma electrónica) y Flash (memoria formada por puertas NAND).
- **Interfaces de comunicación**: Para establecer comunicación con otros dispositivos externos suelen incluir estándares de comunicación como USB, Ethernet, UART, SPI, entre otros.
- **Módulos conversores digital - analógico**: Una gran variedad de sensores devuelven los valores medidos con una señal analógica. Sin embargo, los datos solo pueden ser almacenados y tratados cuando estos tienen un formato digital. Para llevar a cabo esta transformación, un SoC puede disponer de un conversor analógico digital o ADC. De la misma forma, hay actuadores que necesitan una entrada analógica que debe ser convertida desde el formato digital que emplea el sistema. Para ello, se emplea un conversor digital analógico o DAC.
- **Reguladores de voltaje**: Para el correcto funcionamiento de los elementos que conforman un SoC, es importante que la alimentación de estos sea constante y adaptada a las necesidades de cada uno de ellos. Para lograr esta estabilidad de tensión, un SoC incluye reguladores de voltaje en el circuito integrado que lo forma.

- **Generadores de frecuencia**: Para disponer de diferentes relojes en el sistema, es importante la existencia de un generador de frecuencia a partir del cual se puedan generar los relojes necesarios.
- **Buses de comunicación**: para la interconexión de los diferentes elementos anteriores, se emplean buses de comunicación que emplean distintas especificaciones y estándares.

Existen SoC más complejos que, además de lo anterior, incluyen una FPGA o *Field Programmable Gate Array*. Estos dispositivos son denominados *All Programmable* SoC ya que permiten realizar un diseño hardware necesario para programar la FPGA e incorporarlo al diseño software necesario para la programación del microprocesador.

Este tipo de dispositivos reprogramables dan una gran libertad en el diseño de un sistema, permitiendo el desarrollo de multitud de componentes hardware que se adaptan a las necesidades del sistema.

1.3 Objetivos

Como se desprende de lo comentado en las secciones previas de la introducción, la tarjeta ZedBoard dispone de multitud de interfaces y periféricos que permiten su comunicación con otros sistemas. Sin embargo, todos estos periféricos son cableados lo que disminuye considerablemente la capacidad de movimiento del sistema empotrado que se desea diseñar.

El objetivo de este trabajo es incluir conectividad inalámbrica a la tarjeta de desarrollo ZedBoard. Para ello, se desarrollarán una serie de puntos enumerados a continuación.

- Análisis y desarrollo de un sistema basado en el dispositivo Zynq.
- Estudio de las distintas posibilidades de conectividad que presenta ZedBoard. Para ello, se enumerarán los distintos periféricos de los que dispone el sistema Zynq-7000 y las distintas conexiones que se encuentran disponibles en la tarjeta ZedBoard.
- Descripción de algunas de las tecnologías inalámbricas más empleadas que permiten proveer a la tarjeta de desarrollo de conectividad sin cables o *Wireless*.
- Estudio de las distintas posibilidades disponibles para añadir la conectividad WiFi al sistema. Para ello, se estudian las diferentes interfaces y tecnologías que emplean los módulos WiFi disponibles en el mercado para comunicarse con la placa ZedBoard.
- Descripción de los módulos WiFi que se emplearán posteriormente a lo largo del trabajo.
- Explicación detallada de los pasos que se deben dar para obtener un correcto funcionamiento del sistema con conectividad inalámbrica. Para comprobar el funcionamiento del módulo se hará uso del sistema SSH que permite ejecutar comandos en ZedBoard a través de una conexión en red.
- Desarrollo de una aplicación que permita controlar, a través de un dispositivo conectado a una red WiFi, distintos elementos que se encuentran en la tarjeta de desarrollo ZedBoard.
- Desarrollo de una aplicación *bare-metal* sencilla en la cual se emplee un segundo dispositivo WiFi que se comunica con la tarjeta ZedBoard por medio de una interfaz UART.

Capítulo 2: Desarrollo de un sistema basado en Zynq

En el desarrollo de este trabajo se va a emplear la tarjeta de desarrollo ZedBoard basada en el dispositivo Zynq. El dispositivo Zynq es un sistema reconfigurable que puede ser clasificado como un *All Programmable* SoC ya que está formado por una FPGA acoplada a un procesador.

Para desarrollar un sistema basado en Zynq se debe realizar un diseño dual. Primero se debe diseñar la parte hardware donde se indica cómo se debe configurar la parte reconfigurable. Una vez que la parte hardware está correctamente diseñada, se procede a diseñar la parte software del proyecto.

2.1 Tarjeta de desarrollo ZedBoard

Para diseñar e implementar un sistema empotrado, existen multitud de herramientas que facilitan el trabajo proporcionando un completo entorno de desarrollo.

La placa de desarrollo ZedBoard (*Zynq Evaluation & Development Board*) es una tarjeta de desarrollo que emplea Zynq-7000 All Programmable Soc de Xilinx. Además, dispone de una gran cantidad de conexiones que facilitan la comunicación con otros dispositivos: micro-USB, HDMI, VGA, entre otros muchos.

El sistema Zynq-7000 combina hardware, software y la facilidad de programación de dispositivos de entrada y salida en un único dispositivo. Zynq-7000 está formado por una FPGA de la serie 7 de 28 nm acoplada a un procesador ARM Dual-Core Cortex-A9.

En la Figura 2.1 se pueden observar todos los elementos que forman el dispositivo Zynq-7000 AP Soc.

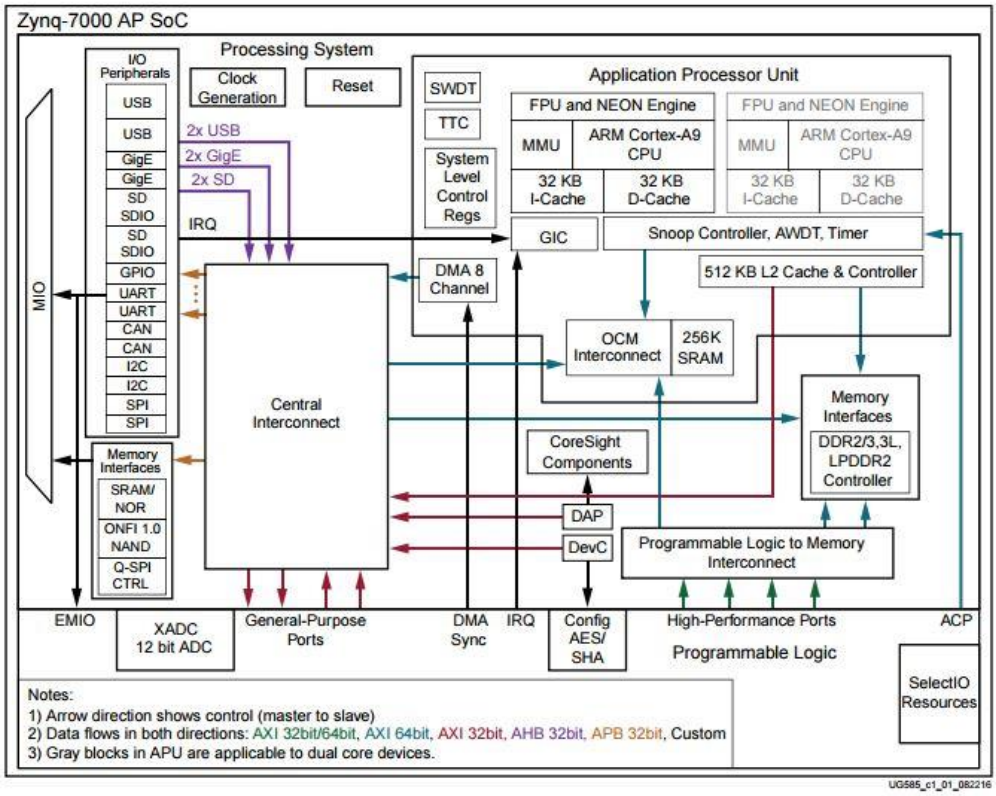

Figura 2.1: ZYNQ 7000 AP SoC [1]

Todo el sistema está dividido en dos partes claramente diferenciadas: la parte de procesamiento (*Processing System* o PS) y la parte de lógica programable (*Programmable Logic* o PL).

2.1.1 Sistema de procesamiento

La parte PS está formada principalmente por la APU (*Application Processor Unit*), la memoria caché, memoria ROM y RAM, interfaces de memoria externa, controlador DMA y los periféricos e interfaces de entrada y salida.

El elemento más importante del sistema de procesamiento es la APU en la cual se encuentran los dos núcleos ARM Cortex-A9. En este mismo bloque se encuentra una memoria caché de 512 KB junto con una memoria RAM de 512 KB. De forma transparente para el usuario, el sistema dispone también de un bloque de memoria ROM de 128 KB empleada para el arranque. Dentro de este elemento se encuentra también el controlador DMA (*Direct Memory Access*) que dispone de 8 canales. El acceso directo a memoria permite que se realicen transferencias desde o hacia memoria sin intervención del procesador. De esta forma, las transferencias se realizan de forma más eficiente al provocar un menor número de interrupciones a la unidad de procesamiento.

En el sistema de procesamiento también se encuentran diferentes controladores de memoria: NOR/NAND/SRAM Flash y DRAM.

Por último, la parte PS también incluye una serie de periféricos de entrada y salida que permiten la comunicación con otros sistemas. Los controladores de periféricos que componen el sistema Zynq son: SPI, I2C, CAN, UART, GPIO, SDIO, USB y Ethernet Gigabit. Estos periféricos se conectan con el exterior a través de la interfaz MIO (*Multiplexed I/O*). Debido al limitado número de pines MIO, los periféricos se pueden conectar con la parte PL a través de la interfaz EMIO (*Extended MIO*). De esta forma, se pueden emplear los recursos de la parte de lógica programable como salidas del sistema.

Todos los elementos que componen el sistema de procesamiento están intercomunicados entre sí con buses de datos. La arquitectura de buses que emplea Zynq es una tecnología desarrollada por ARM denominada AMBA.

2.1.2 Lógica programable

Existen tres versiones diferentes del dispositivo Zynq que se diferencian únicamente en la parte de lógica programable. La tarjeta ZedBoard emplea el dispositivo Z-7020 cuya FPGA Artix-7 está formada por 85000 células lógicas, 53200 LUTs, 106400 biestables, 4,9 Mb de RAM y 220 DSP *Slices*.

En la PL también se encuentra un módulo XADC formado por 2 conversores analógico digital de 12 bits cada uno. Este conversor tiene como entradas 17 canales externos, un sensor de temperatura y 6 sensores de voltaje.

La parte de procesamiento y la parte lógica deben poder conectarse entre sí para obtener sacar el máximo partido al sistema. Para ello, se emplea el protocolo AXI (*Advanced extensible interfaz*) que forma parte de la especificación AMBA.

2.2 Diseño Hardware

El diseño hardware se realiza con la herramienta de diseño Vivado. Este software ha sido desarrollado por Xilinx y puede ser utilizado tanto por su interfaz gráfica como a través del terminal de comandos que incorpora. Existen multitud de versiones que no son directamente compatibles entre sí, es decir, no se puede abrir un proyecto creado con una versión diferente a la instalada.

En este trabajo se empleará la versión 2014.4 instalada sobre el sistema operativo Windows. Además, en todo momento se va a emplear el interfaz gráfico que incluye Vivado debido a su mayor facilidad de uso.

El desarrollo de un diseño Hardware se puede observar resumido en el diagrama de flujo mostrado en la Figura 2.2.

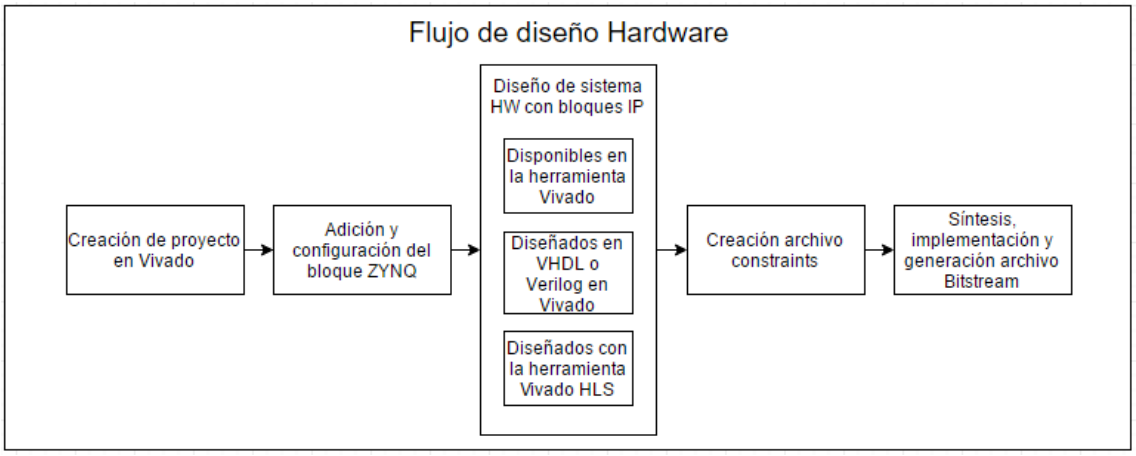

Figura 2.2: Diagrama de flujo de diseño Hardware

El primer paso que se debe dar es crear un nuevo proyecto. Tras indicar un nombre y la ubicación donde se desea guardar, se debe señalar que placa de desarrollo se va a emplear. En este trabajo se empleará en todo momento la tarjeta ZedBoard.

Una vez creado el proyecto correctamente, se va a crear un diagrama de bloques que representará todo el sistema. Los bloques que se van a emplear son llamados *IP Core* e implementan la funcionalidad hardware.

Existe una gran cantidad de *IP Core* disponibles en el entorno Vivado. Estos bloques han sido diseñados por Xilinx y, la mayoría de ellos, pueden ser empleados de forma libre con la licencia del software. Sin embargo, también existen *IP Core* que solo se pueden emplear un número determinado de veces antes de necesitar obtener una licencia para seguir usándolos. En este trabajo únicamente se emplearán los *IP Core* que vienen incluidos en la licencia del software.

Además de los *IP Core* diseñados por Xilinx, se pueden diseñar otros por parte del usuario desde la propia herramienta Vivado. Para ello, se debe emplear la herramienta "*Create and package IP*". Tras asignar un nombre que identifique el bloque IP, se deben generar todos los archivos que definan su comportamiento. Para ello, se debe emplear un lenguaje de descripción hardware que, en Vivado, debe ser VHDL o Verilog.

Además de poder diseñar los *IP Core* con un lenguaje de descripción de hardware también se puede emplear otra herramienta de Xilinx llamada Vivado HLS. La herramienta *Vivado High-Level Synthesis* se emplea para describir hardware usando el lenguaje de programación C, C++ o

System C. De esta forma, no es necesario utilizar un lenguaje de descripción hardware sino que la herramienta se encarga de convertir el lenguaje de alto nivel C o C++ a VHDL o Verilog. Una vez sintetizado el código deseado, se puede exportar el resultado desde Vivado HLS a Vivado como un *IP Core* que debe ser añadido al proyecto Vivado para poder ser utilizado.

Todos los diseños hardware en Vivado deben contener el bloque IP llamado *ZYNQ7 Processing System*. Este bloque representa el sistema de procesamiento Zynq-7000. Este sistema está compuesto por una FPGA Serie 7 de lógica programable y un microprocesador ARM Dual-Core Cortex-A9.

Una vez añadido el bloque que representa al Zynq7, se debe llevar a cabo su configuración para que se adapte a las necesidades del sistema que se desea diseñar. Existe una gran cantidad de parámetros que se pueden configurar: relojes del sistema, interrupciones, tipo de memoria que se desea emplear, buses AXI, entre otros muchos. Cabe destacar que en este bloque se configura también los periféricos disponibles que se van a emplear en el sistema. Una descripción detallada de los controladores disponibles se explicará posteriormente en el capítulo 3.

Tras llevar a cabo la configuración del bloque Zynq7, se pueden añadir otros *IP Core* para que formen parte del proyecto. Todos los bloques deben estar correctamente conectados entre sí. El software facilita este trabajo con una herramienta que automatiza este proceso.

Para poder llevar a cabo la implementación del sistema, se deben definir las *constraints* del diseño. El archivo que define las *constraints* relaciona los componentes de ZedBoard y los pines de Zynq.

Una vez realizados todos los pasos anteriores, se debe realizar la síntesis e implementación del sistema. Si no existe ningún error, se debe proceder a generar el archivo *bitstream*. Este archivo es el que emplea la tarjeta ZedBoard para configurar la parte reconfigurable.

Una vez se ha terminado de diseñar la parte hardware, se debe realizar el diseño de la parte software.

2.3 Diseño Software

Una vez diseñada la parte de lógica reconfigurable, se debe llevar a cabo el diseño de la parte software. Este diseño se realiza con otra herramienta llamada Xilinx SDK (*Software Development Kit*). Esta herramienta está basada en Eclipse y proporciona un entorno completo para el desarrollo de aplicaciones software.

El desarrollo de un diseño Software se puede ver en el diagrama de flujo mostrado en la Figura 2.3.

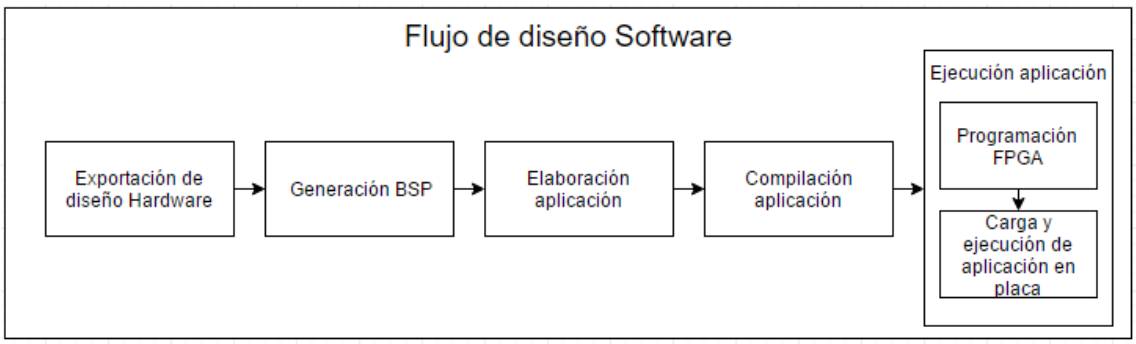

Figura 2.3: Diagrama de flujo de diseño Software

Para realizar el diseño software se parte del proyecto hardware realizado en Vivado. Para poder emplear todos los archivos generados en Vivado, se debe exportar el proyecto previamente. Una vez realizada la exportación hardware todos los archivos se encuentran añadidos al proyecto software en Xilinx SDK.

El siguiente paso es generar el *Board Support Package* o BSP. El BSP es un conjunto de bibliotecas y drivers que forman parte de la capa más baja de la arquitectura software. Ofrece servicios software basados en el procesador y los periféricos que forman la parte PS de Zynq. Por esto, está asociado a una plataforma hardware concreta.

Cuando se genera el BSP, se incluye de forma automática la biblioteca "libxil.a" que contiene los controladores de todos los periféricos que se hayan configurado en el bloque Zynq y en la parte reconfigurable. Además, se pueden incluir una serie de bibliotecas que proporciona Xilinx y que son necesarias únicamente si se hace uso de características como el tratamiento de archivos FAT, control de redes TCP/IP o encriptación RSA.

Una vez generado el BSP, el siguiente paso es elaborar una aplicación que sea ejecutada en el sistema. Esta aplicación se debe programar en lenguaje C o C++. Desde esta aplicación se pueden controlar los dispositivos hardware por medio de los drivers que se han incluido en el BSP.

Tras generar la aplicación deseada, esta se debe compilar y, posteriormente, generar el ejecutable que se cargará en la placa de desarrollo.

Antes de poder ejecutar la aplicación diseñada, se debe programar la parte reconfigurable de la tarjeta ZedBoard con el archivo *Bitstream* generado tras la implementación hardware. Cuando el proceso de programado concluye correctamente, se enciende el LED "Done" de la placa de desarrollo.

En este momento se puede ejecutar la aplicación diseñada cargando el archivo .elf obtenido tras la compilación.

Esta forma de ejecución de una aplicación es denominada *bare-metal* o *Standalone*. Este tipo de ejecución provee un sistema sencillo que proporciona características básicas de procesamiento como entrada y salida estándar y acceso a las características hardware del procesador.

2.4 Sistema operativo sobre ZedBoard

Además del diseño y ejecución de una aplicación *bare-metal* como la explicada en el apartado anterior, es posible cargar un sistema operativo sobre ZedBoard. El sistema operativo está basado en Linux debido al procesador ARM que incorpora el SoC Zynq.

En este caso, es necesario generar una serie de archivos, mostrados en la Figura 2.4, que permitan un correcto arranque del sistema operativo.

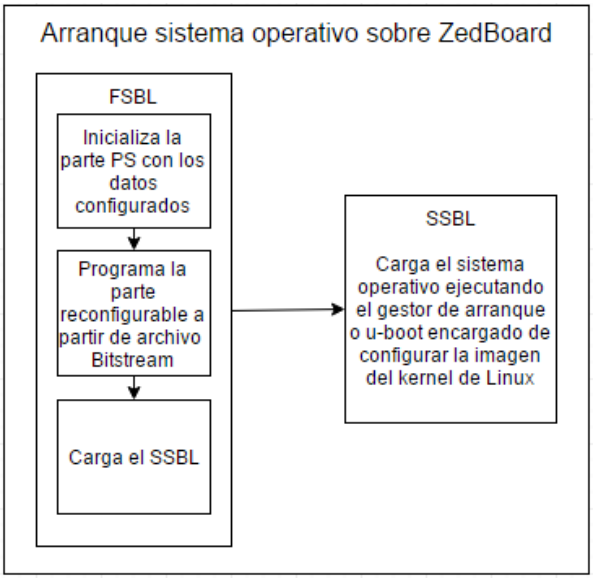

Figura 2.4: Arranque sistema operativo en ZedBoard

El primero de los archivos necesarios es el FSBL o *First Stage BootLoader*. El FSBL es el encargado de configurar el arranque del sistema preparándolo para cargar archivos de arranque más complejos como los que son necesarios para un sistema operativo.

Antes de que se ejecute el archivo de arranque del sistema operativo, se debe programar la parte reconfigurable de la tarjeta ZedBoard. El archivo encargado es el denominado *BitStream*, generado al finalizar el diseño de la parte hardware.

Para generar el resto de archivos que componen un sistema operativo basado en Linux, hay una gran variedad de distribuciones disponibles como PetaLinux, Xillinux o Android.

En este trabajo se va a emplear la herramienta PetaLinux SDK [2] de Xilinx para generar el resto de archivos que forman el sistema operativo que se va a ejecutar sobre ZedBoard. El software PetaLinux SDK genera un sistema operativo básico que es fácilmente configurable según las necesidades del sistema que se desea diseñar.

Entre los diferentes archivos que genera PetaLinux SDK, el siguiente que se emplea en el arranque del sistema es el SSBL. El *Second Stage BootLoader* es el gestor de arranque del sistema operativo que se ejecuta sobre la tarjeta de desarrollo.

PetaLinux SDK también genera el archivo imagen donde se encuentra el *kernel* del sistema operativo así como su árbol de directorios.

Por último, para llevar a cabo un correcto arranque del sistema, es necesario un archivo donde se definan todos los dispositivos hardware accesibles. Este archivo es denominado *device tree* [3] y en él deben estar incluidos todos los dispositivos que conforman el hardware del sistema.

Todos los archivos descritos anteriormente se ubican en una tarjeta SD desde la cual, la placa ZedBoard va ejecutándolos.

Por defecto, PetaLinux SDK viene configurado para que el sistema operativo se ejecute en memoria RAM de forma que, si bien la ejecución es más rápida, al apagar el sistema, todos los cambios realizados se pierden.

Sin embargo, PetaLinux SDK puede ser configurado para que todos los archivos sean almacenados en una segunda partición de la tarjeta SD. De esta forma, el árbol de directorios del sistema operativo se almacena en esta segunda partición permitiendo que todos los cambios realizados se guarden tras apagar el sistema.

Capítulo 3: Conectividad disponible en ZedBoard

La tarjeta ZedBoard dispone de multitud de posibilidades para la conexión de periféricos y dispositivos que permiten aumentar la capacidad de interacción con el mundo exterior a través de distintos protocolos e interfaces. Todas estas posibilidades de conexión se encuentran señaladas en la Figura 3.1.

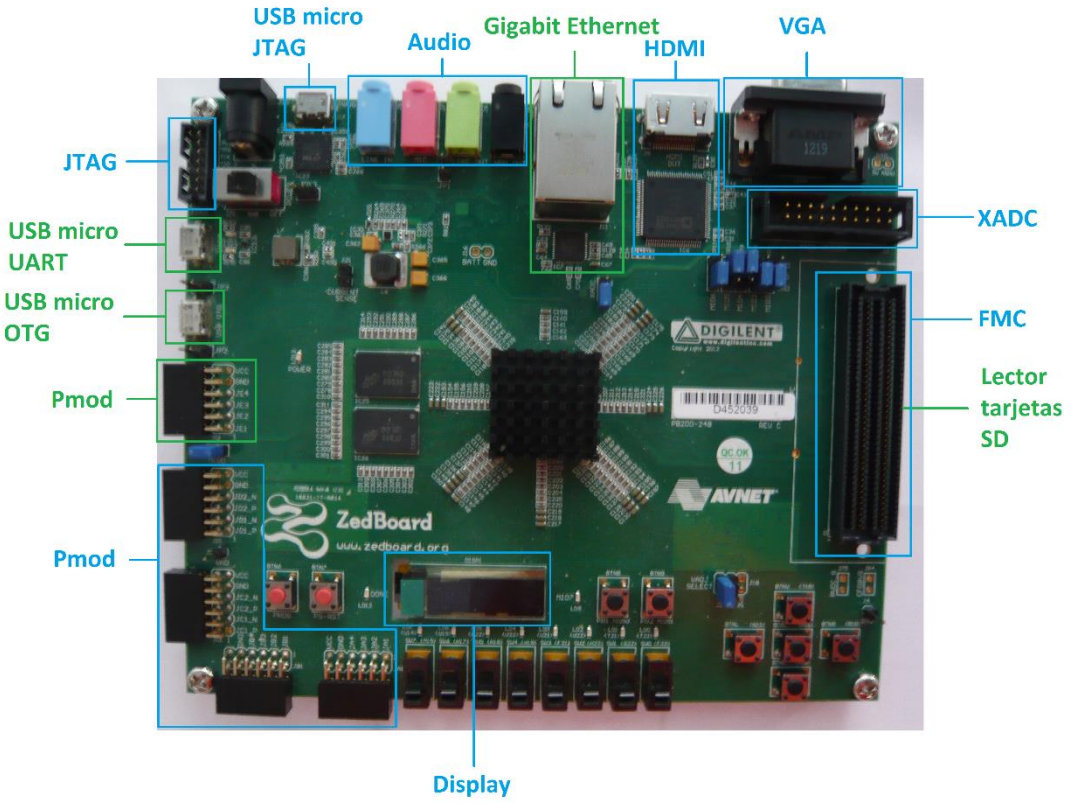

Figura 3.1: Placa de desarrollo ZedBoard con conexiones disponibles

3.1 Periféricos en ZYNQ

En la parte PS del dispositivo Zynq, existen una serie de periféricos que se encuentran precableados y que pueden ser configurados según sea necesario. A continuación, se describen las diferentes tecnologías que se encuentran disponibles en el dispositivo Zynq y que pueden emplearse para ampliar las capacidades de conectividad de un sistema desarrollado con este dispositivo.

3.1.1 Comunicación SPI

El dispositivo Zynq dispone de dos puertos *Serial Peripheral Interface* de tipo *full-duplex*. SPI es un estándar síncrono que emplea 4 señales para llevar a cabo la comunicación entre un dispositivo maestro y otro esclavo:

- SCLK (*Clock*): transmite la señal de reloj para sincronizar el envío y recepción de bits.
- MOSI (*Master Output Slave Input*): Señal que por la que se envían datos desde el maestro al dispositivo esclavo.
- MISO (*Master Input Slave Output*): Señal que emplea un dispositivo esclavo para transferir datos al dispositivo maestro.

- SS (*Select*): Señal empleada para indicar a un dispositivo esclavo que debe activarse. Para cada dispositivo esclavo debe existir una señal SS única. En ZedBoard existen 3 señales para este propósito por lo que se pueden conectar 3 dispositivos esclavos pudiéndose expandir si fuera necesario.

Existen multitud de dispositivos que, a partir del estándar de comunicaciones SPI, se pueden relacionar con un sistema aportando diferentes datos o servicios. Algunos ejemplos de dispositivos que empleen esta tecnología son: sensores (acelerómetros, sensor de luz ambiental, micrófonos, giroscopios, barómetros, entre otros muchos), pantallas OLED y LCD, ADC, memorias flash, etcétera.

3.1.2 Comunicación I2C

Zynq dispone de dos interfaces *Inter-Integrated Circuit* que pueden funcionar como maestro o como esclavo en la comunicación serie. La comunicación es bidireccional y el dispositivo maestro es el encargado de seleccionar que dispositivo esclavo debe activarse. Para ello, cada dispositivo tiene asignada una dirección que el maestro envía a través de la línea de datos. En la tarjeta ZedBoard se encuentra implementada la versión 2 de este protocolo. El protocolo I2C emplea únicamente dos señales para llevar a cabo la comunicación:

- CLK (*Serial Clock*): es la señal de reloj que permite realizar la comunicación de forma síncrona.
- SDA (*Serial Data*): es la línea que se emplea para el envío y recepción de datos entre el maestro y los esclavos.

Al igual que en el caso de comunicación SPI, a través de la especificación I2C se pueden ampliar las capacidades de un sistema conectando diferentes dispositivos como: sensores (giroscopios, acelerómetros, sensores de temperatura, botones capacitivos, sensores de humedad, entre otros), convertidores analógico digital, relojes de tiempo real (RTC), pantallas LCD, etcétera.

3.1.3 Comunicación CAN

El dispositivo Zynq dispone de dos interfaces *Controller Area Network* compatibles con los estándares CAN 2.0A y CAN 2.0B. Soporta una velocidad de transferencia máxima de 1 Mbps. El protocolo CAN emplea dos líneas de transmisión para llevar a cabo la comunicación. Se soportan los siguientes modos de operación:

- *Configuration*: Este modo de operación es empleado para establecer las propiedades de comunicación que se van a emplear en la comunicación a través del bus CAN.
- *Normal*: Es el modo definido por la especificación IEEE para la transmisión y recepción de mensajes.
- *Sleep*: Este modo es usado para ahorrar energía en los momentos que no se lleva a cabo ninguna comunicación.
- *Loop Back*: Es el modo empleado para diagnosticar problemas en el bus CAN
- *Snoop mode*: Al igual que el anterior, se emplea para diagnosticar problemas en la comunicación.

El bus CAN está muy extendido en la industria, especialmente en el área de la automoción, debido a su sencillez y robustez. La mayoría de automóviles actuales emplean CAN como protocolo para el control de sus sistemas internos. Por tanto, a través de esta especificación y

con la conexión disponible en los automóviles, se puede conocer una gran cantidad de información captada por los distintos sensores que se encuentran repartidos por el vehículo.

3.1.4 Comunicación UART

Zynq dispone de dos interfaces *Universal Asynchronous Receiver-Transmitter* configurables de forma individual o conjunta para su funcionamiento a diferentes velocidades de transmisión y con distintas configuraciones de señal como, por ejemplo, el número de bits de datos que se envían, el número de bits que se emplean para la señalización de parada y la configuración de bit de paridad. Se emplean colas FIFO de 64 bytes para transmitir y recibir los datos. El protocolo de comunicación UART emplea 2 señales para llevar a cabo la comunicación:

- TRX: es la señal empleada para enviar los datos que se desea transmitir.
- RCX: es la señal por la que se esperan los datos que se envían desde el dispositivo con el que se pretende llevar a cabo la comunicación.

Dependiendo del uso que se vaya a dar al controlador UART, este puede comportarse como modem permitiendo generar y responder las señales propias de ese sistema de comunicación.

A través de una interfaz UART, al igual que con SPI y I2C, se pueden conectar multitud de dispositivos que amplíen las capacidades de un sistema. Algunos ejemplos de dispositivos que emplean la especificación UART son: receptores GPS, sensores de ultrasonidos, dispositivos Bluetooth, pantallas LCD o adaptadores a USB o RS232.

3.1.5 Comunicación SDIO

El dispositivo Zynq dispone de dos controladores SD/SDIO *Secure Digital Input Output* para establecer comunicación con dispositivos SDIO, tarjetas de memoria SD y tarjetas MMC. Para llevar a cabo la transferencia se pueden emplear una única línea o cuatro líneas en paralelo. El controlador es compatible con la versión 2 del estándar SD pero no soporta el modo SPI. Se soportan las tarjetas de alta capacidad SDHC *Secure Digital High Capacity* lo que permite mayor velocidad de escritura y capacidad. Se puede seleccionar la velocidad de funcionamiento:

- *Low-speed*: en la opción de baja velocidad se emplean entre 1 KHz y 400 KHz.
- *Full-speed*: la opción de alta velocidad permite emplear desde 1 MHz a 50 MHz.

Los controladores SDIO pueden hacer uso del DMA para que la transmisión de datos se realice de una forma mucho más rápida provocando muchas menos interrupciones a la unidad de procesamiento.

3.1.6 Comunicación Gigabit Ethernet

Zynq dispone de dos controladores Gigabit Ethernet compatibles con el estándar IEEE 802.3-2008. Tienen capacidad de operación tanto en modo *half duplex* como *full duplex*. Soportan tres velocidades: 10, 100 y 1000 Mbps y cada controlador puede ser configurado de forma independiente. Esta configuración permite elegir la dirección MAC, programar el DMA y seleccionar la velocidad y el modo de trabajo. Los controladores Gigabit Ethernet hacen uso de la interfaz RGMII cuando se emplea MIO y de la interfaz GMII cuando se usa EMIO. Los controladores Gigabit Ethernet pueden realizar las transferencias de datos empleando el acceso directo a memoria (DMA).

3.1.7 Comunicación USB

El dispositivo Zynq dispone de dos controladores USB iguales que pueden ser configurados de forma independiente. Cada controlador USB emplea el protocolo ULPI para establecer una comunicación con la interfaz física ULPI PHY usando MIO. La interfaz ULPI emplea 8 bits en paralelo y usa un reloj a una frecuencia de 60 MHz. Cada controlador puede controlar hasta 12 dispositivos y emplea la versión USB 2.0. Para mejorar la velocidad de transferencia, los controladores USB pueden emplear el DMA. Hay tres modos de operación:

- *USB Host*: El software dispone de la capacidad de controlar el estado de la comunicación en todo momento. Este modo soporta una velocidad máxima de 480 Mbps.
- *USB Device*: En este modo, el controlador responde a los comandos enviados por el *Host*.
- *USB OTG* (*On-the-go*): El software cambia entre los modos *Host* y *Device* empleando los protocolos *Host Negotiated Protocol* (HNP) y *Session Request Protocol* (SRP). De esta forma, el controlador actúa como *USB Host* o *USB Device* dependiendo del modo en el que se encuentre.

3.1.8 Interfaz GPIO

Los *General purpose I/O* permiten observar y controlar el estado de 54 pines conectados al módulo MIO. También se puede emplear la interfaz EMIO permitiendo controlar hasta 64 entradas y 128 salidas entre la parte PS y PL. Cada GPIO puede ser programado individualmente o de forma grupal. Además, pueden ser usados como interrupciones en las cuales se puede elegir el tipo: sensible por nivel (nivel alto o nivel bajo) o sensible por flanco (flanco de subida, flanco de bajada o ambos). El módulo GPIO está dividido en 4 bancos de 32 bits cada uno:

- Banco 0: Controla los pines MIO[31:0].
- Banco 1: Controla los pines MIO[53:32].
- Banco 2: Controla las señales EMIO [31:0].
- Banco 3: Controla las señales EMIO [63:32].

3.2 Conexiones en ZedBoard

La tarjeta ZedBoard dispone de multitud de conexiones que permiten que esta tenga una gran conectividad con multitud de dispositivos distintos.

Como se podrá observar en los apartados desarrollados a continuación, solo algunos de los periféricos enumerados anteriormente disponen de una conexión específica para su uso. A modo de ejemplo, uno de los controladores SDIO está asociado al lector de tarjetas SD disponible en la ZedBoard.

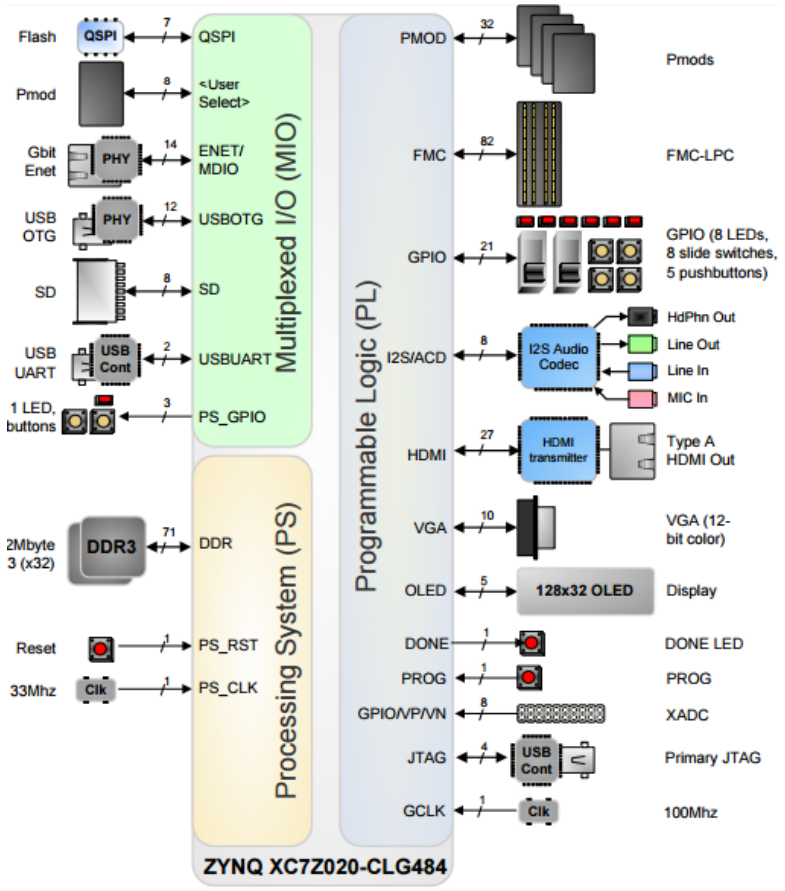

Figura 3.2: Diagrama ZedBoard [4]

Debido a la dualidad en el funcionamiento de ZedBoard, se van a indicar las distintas conexiones de las que dispone esta tarjeta divididos en dos grandes grupos, las conectadas directamente a la parte PS y las que pertenecen a la parte PL.

3.2.1 Conexiones PS

Los siguientes dispositivos se encuentran conectados directamente a los periféricos disponibles en el sistema Zynq que se han descrito en el apartado anterior.

- **USB-micro UART**: El chip Cypress CY7C64225 [5] sirve como puente entre el USB y la interfaz UART1 de la parte PS. Este chip permite la conexión con un ordenador que disponga de software capaz de interpretar la información enviada a través de este puerto serie. Ejemplos de estos programas serían TeraTerm, PuTTY, minicom o HyperTerm, entre otros muchos. Únicamente está implementada la comunicación básica TXD/RXD. Si se requieren otras funcionalidades como el control de flujo, se debe añadir empleando la interfaz EMIO de la parte PL.
- **USB-micro OTG**: Implementa uno de los dos USB disponibles en la configuración de Zynq, denominado USB0. La parte física es implementada mediante el chip TI TUSB1210 [6] que está conectado al banco 1/501 de MIO. Mediante los jumpers JP2 y JP3 se selecciona el modo de funcionamiento del controlador USB: Host, Device y OTG. Cuando se encuentra configurado con los modos Host u OTG, ZedBoard proporciona 5 voltios.
- **Gigabit Ethernet**: ZedBoard dispone de un puerto Ethernet 10/100/1000 para tener capacidades de trabajo en red a través de un conector RJ-45. Emplea un chip Marvell 88E1518 PHY [7] que trabaja con un voltaje de 1,8 voltios y se comunica con el Zynq-

7000 a través de un interfaz RGMII. El controlador conectado a este puerto es el denominado ENET0 en la configuración del procesador Zynq.

- **PMOD**: Se dispone de 5 puertos Pmod. La interfaz Pmod [8] fue desarrollada por Digilent como un *open standard* para la conexión de periféricos en FPGAs y microcontroladores. Esta interfaz dispone de 12 pines, de los cuales 8 son entradas y salidas y los 4 restantes son empleados para alimentación, 3,3 voltios, y tierra. De los 5 puertos Pmod, únicamente uno de ellos se encuentra conectado a la parte PS. Este puerto es denominado JE1 y tiene asignado los puertos MIO 0 y del 9 al 15.
- **Lector de tarjetas SD:** El controlador SD0 está conectado al lector de tarjetas disponible en la placa ZedBoard. Está conectada a los puertos MIO[40-47] pertenecientes al banco 1. Para poder hacer uso del lector de tarjetas, el *jumper* 6 debe estar cerrado, cortocircuitado de esta forma los pines que lo forman.

3.2.2 Conexiones PL

Al contrario que los dispositivos anteriores, estos dispositivos pertenecen a la parte PL y son conectados a los controladores a partir de la interfaz EMIO.

- **USB-micro JTAG**: Este puerto USB es empleado para cargar y depurar los programas ya que es soportado por las herramientas de diseño de Xilinx.
- **VGA**: La placa ZedBoard dispone de un conector VGA como salida de vídeo en color con 12 bits.
- **HDMI**: El chip Analog Devices ADV7511 [9] permite disponer de una salida digital HDMI. Los drivers para su funcionamiento se pueden encontrar en la página web del fabricante.
- **Audio**: El chip Analog Devices ADAU1761 [10] proporciona el procesamiento de audio digital en Zynq. Soporta frecuencia de muestras comprendidas entre 8 y 96 KHz. En ZedBoard se dispone de 4 conectores Jack de 3,5 mm: micrófono (rosa), línea de entrada (azul), línea de salida (verde) y micrófono (negro). Los drivers de Linux para el correcto funcionamiento del chip se pueden encontrar en la página web del fabricante.
- **Pmods**: En la parte PL se encuentran 4 de las 5 interfaces Pmod explicadas anteriormente. Dos de estas conexiones Pmod pueden configurarse de forma diferencial (JC1 y JD1).
- **XADC**: Módulo de expansión que permite conectar diseños analógicos.
- **FMC (FPGA Mezzanine Card)**: Es un estándar para la conexión de módulos en FPGAs y otros dispositivos con lógica reconfigurable.
- **Display**: ZedBoard dispone de una pantalla OLED UG-2832HSWEG04 de tamaño 128x32 píxeles.

Como se ha comprobado en este capítulo, no existe ninguna conectividad inalámbrica implementada en la tarjeta ZedBoard. Esto dificulta su empleo en determinadas aplicaciones que requieran que el sistema tenga movilidad o que este se ubique en un lugar de difícil acceso.

Por ello, para disponer de una conectividad inalámbrica se deben añadir módulos adicionales que implementen alguna de las tecnologías sin cables existentes y que se comuniquen con la tarjeta ZedBoard a través de alguno de los periféricos y conexiones de los que esta dispone.

En el siguiente capítulo se expondrán algunas de las tecnologías inalámbricas disponibles así como las posibilidades de su implementación sobre la tarjeta de desarrollo Zedboard. En

capítulos posteriores se implementarán algunos de estos módulos inalámbricos y se comprobará su correcto funcionamiento a partir de aplicaciones prácticas.

Capítulo 4: Conectividad WiFi en ZedBoard

Como se ha visto en el capítulo anterior, todos los mecanismos de conectividad de los que dispone la placa de desarrollo ZedBoard son cableados. Por ello, para poder interactuar con otros dispositivos, el sistema diseñado debe estar ubicado en un lugar fácilmente accesible donde se puedan tender los cables necesarios para llevar a cabo la comunicación. Este hecho provoca una limitación en relación a la ubicación donde se pueden instalar los sistemas diseñados con esta tecnología.

Sin embargo, en ciertas ocasiones los sistemas deben ser ubicados en lugares de difícil acceso o donde no es posible utilizar cables por motivos económicos, logísticos o incluso estéticos. En estos casos, una solución puede ser emplear tecnologías que proporcionan mecanismos de conectividad inalámbrica.

4.1 Tipos de conectividades inalámbricas

Hay una gran variedad de tecnologías inalámbricas que, en función de las necesidades del sistema, presentan diferentes características. A continuación, se enumeran algunas de las tecnologías inalámbricas que existen en el mercado indicando sus principales características:

- **Infrarrojos**: Es el tipo de red inalámbrica más limitado ya que los dos equipos que se van a comunicar deben tener visión directa sin obstáculos puesto que la comunicación se hace de forma direccional. Además, debido a la longitud de onda de los rayos infrarrojos que se emplean, 850-900 nm, los dispositivos deben estar muy próximos entre sí. Una de las ventajas que presenta la comunicación por infrarrojos es el bajo coste de los elementos necesarios para llevar a cabo el intercambio de información.
- **ZigBee**: El protocolo ZigBee [11] fue desarrollado con el objetivo de disponer de una tecnología inalámbrica fiable pero con una reducida tasa de transferencia de datos que no supera los 250 kbps. Tiene un consumo muy reducido y los equipos necesarios para su uso tienen un coste muy bajo. Sin embargo, existe un menor número de equipos disponibles en el mercado lo que provoca que su uso sea mucho más reducido en la actualidad.
- **Redes de telefonía móvil**: Existen diferentes tecnologías desplegadas que permiten disponer de conectividad a Internet de forma inalámbrica. Actualmente, se complementan las redes de diferentes generaciones, 2G (GSM/GPRS), 3G (UMTS) y 4G (LTE), donde cada generación tiene una tasa de transmisión creciente con su antecesor y una cobertura decreciente. Para poder emplear esta tecnología, se debe tener contratado el servicio con un operador de Telecomunicaciones por lo que el coste de esta solución es elevado. Además, este sistema tiene un gran consumo de energía y, para su implementación y funcionamiento, es necesario que el lugar donde se desee instalar el sistema disponga de cobertura.
- **Bluetooth**: La tecnología Bluetooth [12] permite establecer una comunicación entre dispositivos que se encuentren a una distancia reducida entre sí. Hay diferentes clases de dispositivos según la distancia máxima y la tasa de transferencia máxima que permiten. La mayoría de dispositivos actuales disponen de módulos Bluetooth que tienen una distancia máxima de funcionamiento de 10 metros y una tasa de transferencia de 3 Mbps. Además, a partir de la versión 4 del protocolo, existe una versión de bajo consumo (Bluetooth Low Energy, BLE) que reduce considerablemente el consumo disminuyendo el radio de cobertura y la tasa de transmisión. Existen en el

mercado una gran cantidad de dispositivos con un coste reducido que implementan el protocolo Bluetooth.

- **WiFi**: El protocolo WiFi [13] permite establecer comunicación entre distintos dispositivos que se encuentran separados a una distancia máxima que varía desde los 20 metros en interiores hasta los 100 metros sin obstáculos. Tiene un mayor consumo que otras tecnologías como ZigBee o Bluetooth pero logra una mayor tasa de transferencia que puede llegar, e incluso superar, los 100 Mbps. La especificación WiFi permite emplear dos frecuencias: 2,4 GHz y 5 GHz. El protocolo WiFi es una de las tecnologías más ampliamente utilizadas en la actualidad y se encuentra en dispositivos como ordenadores, *smartphones*, *tablets* y multitud de equipos que se conectan a la red y que pueden clasificar dentro del conjunto IoT (*Internet of things*). Por todo ello, existe una gran cantidad de dispositivos que implementan este protocolo en el mercado a un bajo precio.

A partir de las características indicadas anteriormente para cada una de las tecnologías, en este trabajo se ha optado por implementar el protocolo WiFi en la placa de desarrollo ZedBoard.

4.2 Opciones de implementación de la conectividad WiFi

Existe una gran variedad de dispositivos en el mercado que permiten añadir una conectividad WiFi con diferentes características y conexiones. En la Figura 4.1 se muestran algunas de las distintas opciones que podrían implementarse en un sistema reconfigurable como ZedBoard.

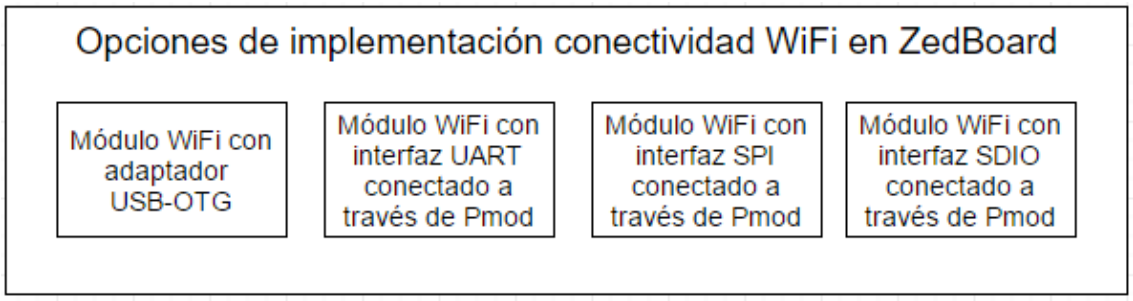

Figura 4.1: Opciones de implementación conectividad WiFi en ZedBoard

- **A través de PMOD-SPI**: la interfaz WiFi se implementa a través de una conexión SPI. En este caso, son necesarias 4 señales para llevar a cabo el intercambio de información entre el controlador del dispositivo WiFi y la placa de desarrollo. El dispositivo implementa la pila de protocolos TCP/IP y dispone de una antena para la emisión y recepción a la frecuencia de funcionamiento. Existe una gran cantidad de dispositivos con diferentes características en el mercado. Un ejemplo de dispositivo es el MRF24WG0MA [14] desarrollado por Microchip y que se muestra en la Figura 4.2.

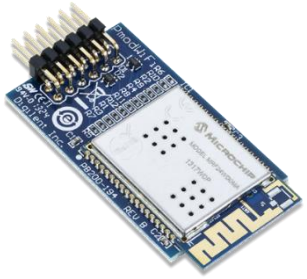

Figura 4.2: Módulo WiFi MRF24WG0MA [15]

- **Adaptadores USB-OTG**: estos dispositivos se conectan a un puerto USB-OTG (USB – *On The Go*) y, a través de él, implementan todas las funciones necesarias para el correcto funcionamiento de la interfaz inalámbrica. Este es el tipo de dispositivo más abundante en el mercado ya que una gran cantidad de dispositivos disponen de un interfaz de comunicación USB. En el caso de funcionamiento con un sistema operativo, muchos de estos adaptadores instalan los drivers necesarios automáticamente sin necesitar la intervención del usuario siendo, por tanto, sistemas "Plug and play". Un ejemplo de adaptador USB-OTG es el dispositivo DWA-140 de D-Link mostrado en la Figura 4.3.

Figura 4.3: Adaptador USB-OTG DWA-140 [16]

- **A través de PMOD-SDIO**: en este caso, la funcionalidad WiFi se implementa a partir de un módulo que se comunica con una tarjeta de desarrollo a través de la interfaz SDIO con 4 hilos. Un dispositivo que funciona con esta tecnología es el WL1835MOD que incluye un módulo WiLink8 de Texas Instruments. Sin embargo, este dispositivo dispone de una conexión con 100 pines que no se puede conectar directamente a la tarjeta ZedBoard. Para llevar a cabo la conexión, se necesita un adaptador que convierta las 100 conexiones que tiene el módulo a una conexión Pmod. Para ello, se emplea el módulo WiLink8 Pmod Adaptor diseñado por Avnet. Estos módulos serán los utilizados para proveer de conectividad inalámbrica a la tarjeta ZedBoard tal y como se explicará en apartados posteriores.
- **A través de PMOD-UART**: en estos dispositivos, la conectividad WiFi se implementa a través de un controlador UART. De esta forma, con únicamente dos hilos se produce el intercambio de datos entre el dispositivo WiFi y el sistema reconfigurable. Estos dispositivos incluyen toda la electrónica necesaria para implementar la pila de protocolos TCP/IP y una antena necesaria para llevar a cabo la comunicación inalámbrica. Existe una gran variedad de dispositivos con diferentes características. Por ejemplo, uno de estos módulos es el ESP8266 que se empleará posteriormente en este trabajo.

4.3 Dispositivo WiFi seleccionado

En este trabajo se va a emplear el dispositivo WL1835MOD WiLink8 [17] con el adaptador WiLink8 Pmod Adaptor conectados tal y como puede verse en la parte superior de la Figura 4.4.

Además, en el capítulo 8 de este trabajo, se va a emplear el dispositivo ESP8266 para proporcionar conectividad WiFi a la tarjeta ZedBoard de forma más sencilla pero también más limitada. En este caso, la comunicación se establece por medio de una de las interfaces UART de las que dispone el dispositivo Zynq. La conexión física se llevará a cabo a través de un puerto Pmod del cual se emplearán únicamente 2 pines, uno para transmitir y otro para recibir, tal y como puede verse en la parte inferior de la Figura 4.4.

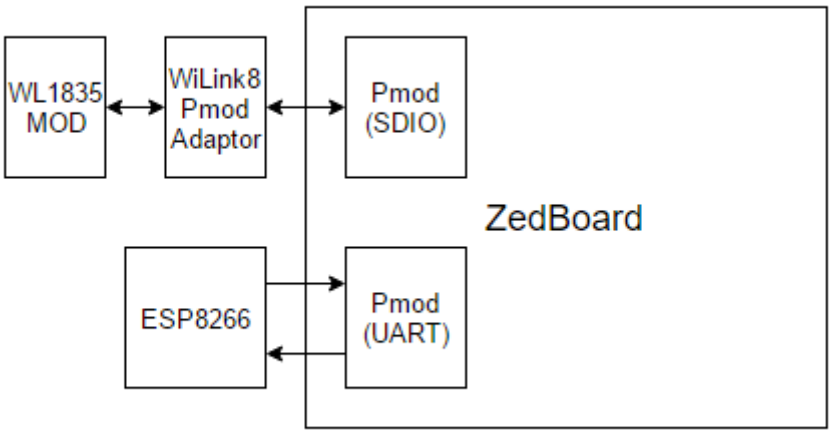

Figura 4.4: Diagrama de conexión dispositivo WiFi

4.3.1 WL1835MOD WiLink8

Este dispositivo ha sido diseñado por Texas Instruments con el objetivo de proveer de conectividad inalámbrica a diferentes sistemas reconfigurables que se pueden encontrar en el mercado.

El módulo elegido forma parte de una familia de dispositivos WL18xx que se diferencian entre sí por las funcionalidades que implementan [18]:

- **WL1801MOD**: Este dispositivo es el más básico de la familia. Implementa únicamente la conectividad WiFi empleando para ello la frecuencia de 2,4 GHz. Además, es de tipo SISO, *single input single output*. Este dispositivo cumple el estándar 802.11bgn.
- **WL1805MOD**: Al igual que el anterior, dispone únicamente de conectividad WiFi a la frecuencia de 2,4 GHz pero en este caso, además de SISO, soporta MIMO (*Multiple input multiple output*) y MRC (*Maximum ratio combining*). Este dispositivo cumple el estándar 802.11bgn. Este dispositivo cumple el estándar 802.11abgn.
- **WL1807MOD**: Complementa al dispositivo anterior haciendo uso también de la frecuencia de 5 GHz que soporta la especificación WiFi.
- **WL1831MOD**: Este dispositivo es igual que el dispositivo WL1801MOD pero con el añadido de soportar también conectividad Bluetooth. Este dispositivo cumple el estándar 802.11bgn.
- **WL1835MOD**: Al igual que el caso anterior, este modelo es igual al WL1805MOD pero implementa el protocolo Bluetooth. Este dispositivo cumple el estándar 802.11bgn.
- **WL1837MOD**: Este dispositivo es el más completo de la familia ya que implementa conectividad WiFi y Bluetooth. Además, en el caso del WiFi puede operar a las dos frecuencias que indica la especificación, 2,4 y 5 GHz, y soporta SISO, MIMO y MRC. Este dispositivo cumple el estándar 802.11abgn.

El dispositivo que se va a emplear en este trabajo es el WL1835MOD [19], mostrado en la Figura *4.5*, del cual únicamente se va a hacer uso de la conectividad WiFi.

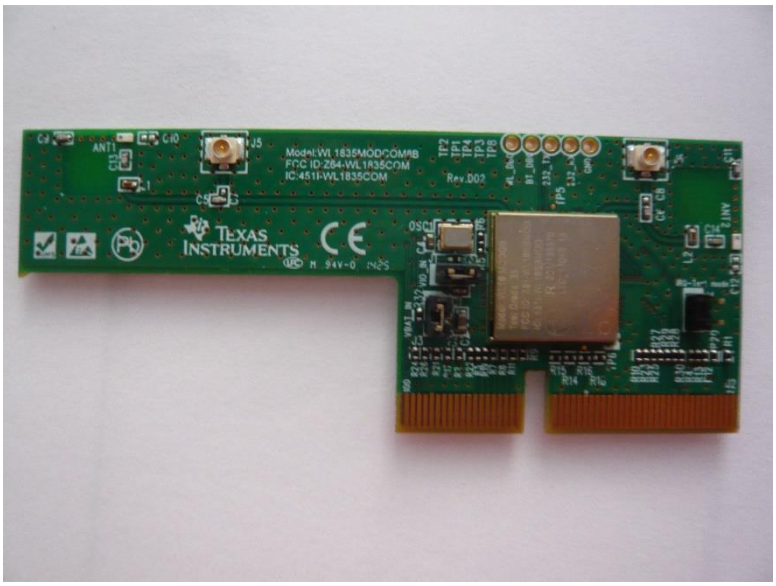

Figura 4.5: Módulo WiLink8 WL1835MOD

El dispositivo tiene un tamaño de 76 mm de largo por 31 mm de ancho. Dispone de dos antenas integradas en el propio dispositivo que permiten el empleo del módulo sin necesidad de añadir antenas externas. Sin embargo, se pueden emplear antenas externas ya que dispone de dos conectores U.FL Coaxial ultra miniatura.

Para llevar a cabo la conexión con una placa de desarrollo, dispone de una conexión COM8 formada por 100 pines que se encuentran divididos en dos grupos.

Para llevar a cabo la comunicación entre la placa de desarrollo y el módulo WiFi se emplea la interfaz SDIO. La interfaz SDIO emplea 4 señales de datos y soporta una frecuencia máxima de reloj de 50 MHz. La velocidad máxima (teórica) de transferencia es de 80 Mbps si se trata de comunicación TCP y asciende a 100 Mbps si se emplea UDP como protocolo de transporte [20].

4.3.2 WiLink8 Pmod Adaptor

Debido al modo de conexión que dispone el módulo WL1835MOD, este dispositivo no puede ser conectado directamente a la tarjeta de desarrollo ZedBoard. Para poder llevar a cabo la conexión, se debe emplear el módulo WiLink8 Pmod Adaptor [21], mostrado en la Figura 4.6, diseñado por Avnet.

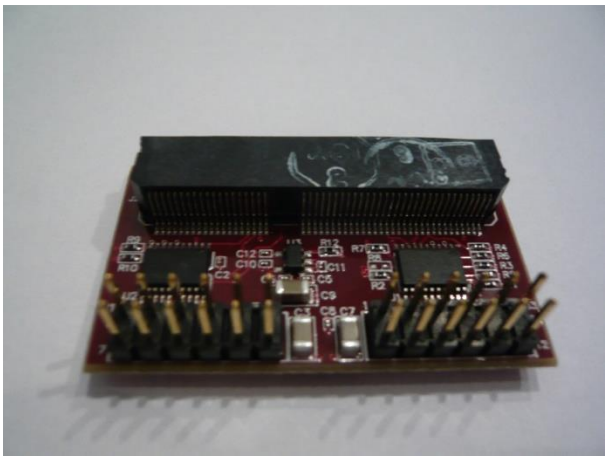

Figura 4.6: Módulo WiLink8 Pmod Adaptor

Este módulo sirve de adaptador entre la conexión COM8 que dispone el módulo WiFi y uno de los puertos Pmod de los que dispone ZedBoard. La adaptación de las señales se logra mediante el chip TXS0108E [22] de Texas Instruments.

Además, este módulo permite llevar a cabo la adaptación de tensiones necesarias para lograr un correcto funcionamiento del sistema ya que los puertos Pmod entregan una tensión de 3,3 voltios y el módulo WL1835MOD emplea 3,3 voltios y 1,8 voltios. Para llevar a cabo la regulación de voltaje, emplea el chip TPS73618DBVR [23] de Texas Instruments.

En la Figura 4.7 se muestra el módulo y el adaptador ya conectados a la placa de desarrollo ZedBoard.

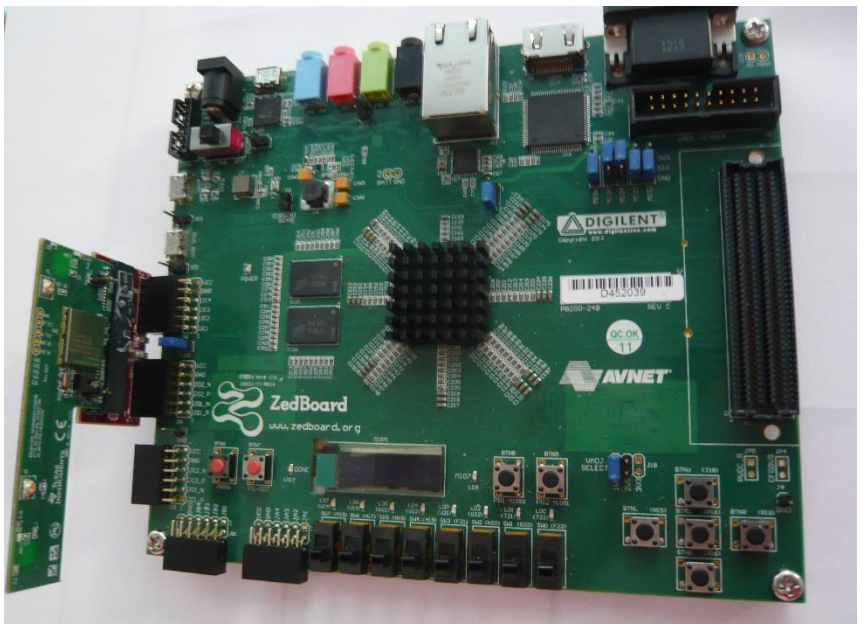

Figura 4.7: Tarjeta ZedBoard con módulo WiFi conectado

Capítulo 5: Utilización del módulo WiFi en ZedBoard

En este capítulo se explicarán detalladamente todos los pasos que se deben seguir para dotar a la tarjeta ZedBoard de una interfaz inalámbrica WiFi. Todos estos pasos se pueden observar esquematizados en la Figura 5.1.

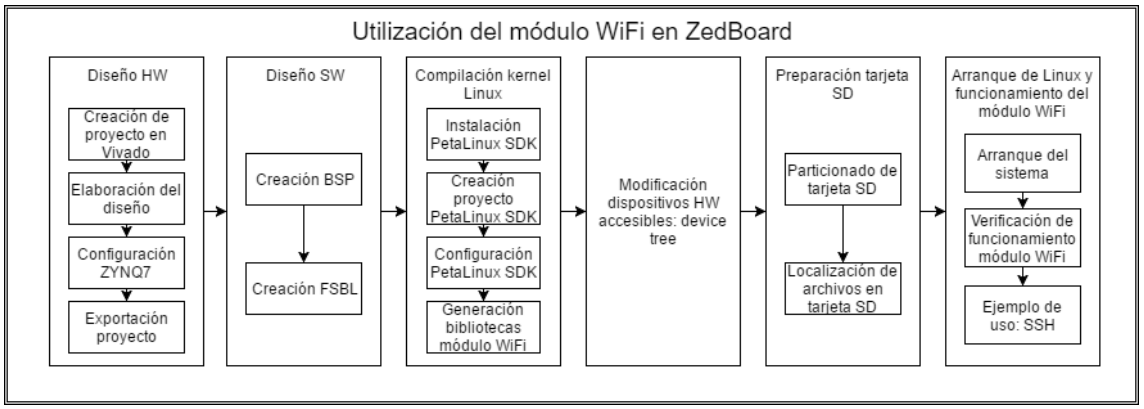

Figura 5.1: Diagrama de flujo de diseño

Para emplear el módulo WiFi en ZedBoard, primero se debe realizar el diseño hardware del sistema. En este diseño, únicamente hará falta configurar el dispositivo Zynq activando la interfaz SDIO que empleará el módulo WiFi.

Una vez realizado el diseño Hardware, se llevará a cabo el diseño Software. Se va a emplear un sistema operativo que se ejecutará sobre la tarjeta de desarrollo ZedBoard. Por ello, el primer paso es crear el archivo FSBL que se empleará como primer sistema de arranque.

Posteriormente, se generará el kernel del sistema Linux que se ejecutará en ZedBoard. Para ello, se empleará la herramienta PetaLinux SDK de Xilinx que genera todos los archivos del sistema operativo necesarios para su correcto funcionamiento.

El último archivo necesario para el correcto funcionamiento del sistema es el *device tree* en el cual se deben indicar todos los dispositivos Hardware accesibles.

Todos los archivos generados anteriormente se ubicarán en una tarjeta SD desde la cual se realizará el arranque del sistema en ZedBoard.

Por último, se comprobará el correcto funcionamiento del sistema y se realizará un ejemplo de uso con el empleo de la herramienta SSH.

5.1 Creación del diseño hardware

Para realizar el diseño hardware se va a emplear la herramienta Vivado 2014.4 de Xilinx. Este software se puede descargar de la página web de Xilinx, siendo necesario estar registrado en la web para llevar a cabo la descarga. Junto a Vivado se instalará también el software Xilinx SDK que se empleará en siguientes apartados. Una vez instaladas las herramientas, se creará un nuevo proyecto con las especificaciones necesarias para el correcto funcionamiento del módulo WiLink8.

5.1.1 Creación de nuevo proyecto en Vivado

El primer paso que se debe llevar a cabo es crear un nuevo proyecto en Vivado 2014.4. Para ello, se hace doble *click* en el icono del programa para que se ejecute el *wizard* de configuración de un nuevo proyecto. Una vez que ha arrancado, aparece una ventana con varias opciones disponibles. Se debe elegir la opción *Create New Project* y, posteriormente, *Next* en la ventana que se abre. A continuación, se introduce el nombre del proyecto en el campo *Project name* y la localización donde se quiere guardar el proyecto en *Project location*. Además, se debe dejar seleccionada la opción de crear un subdirectorio donde se guardarán todos los archivos que componen el proyecto y pulsamos en *Next*. En la nueva pantalla que se nos presenta dejamos seleccionada la opción por defecto *RTL Project*, señalamos la opción *Do not specify sources at this time* ya que no se va a añadir ningún archivo al proyecto y se pulsa nuevamente en *Next*. Todas las ventanas indicadas anteriormente se pueden observar en la Figura 5.2.

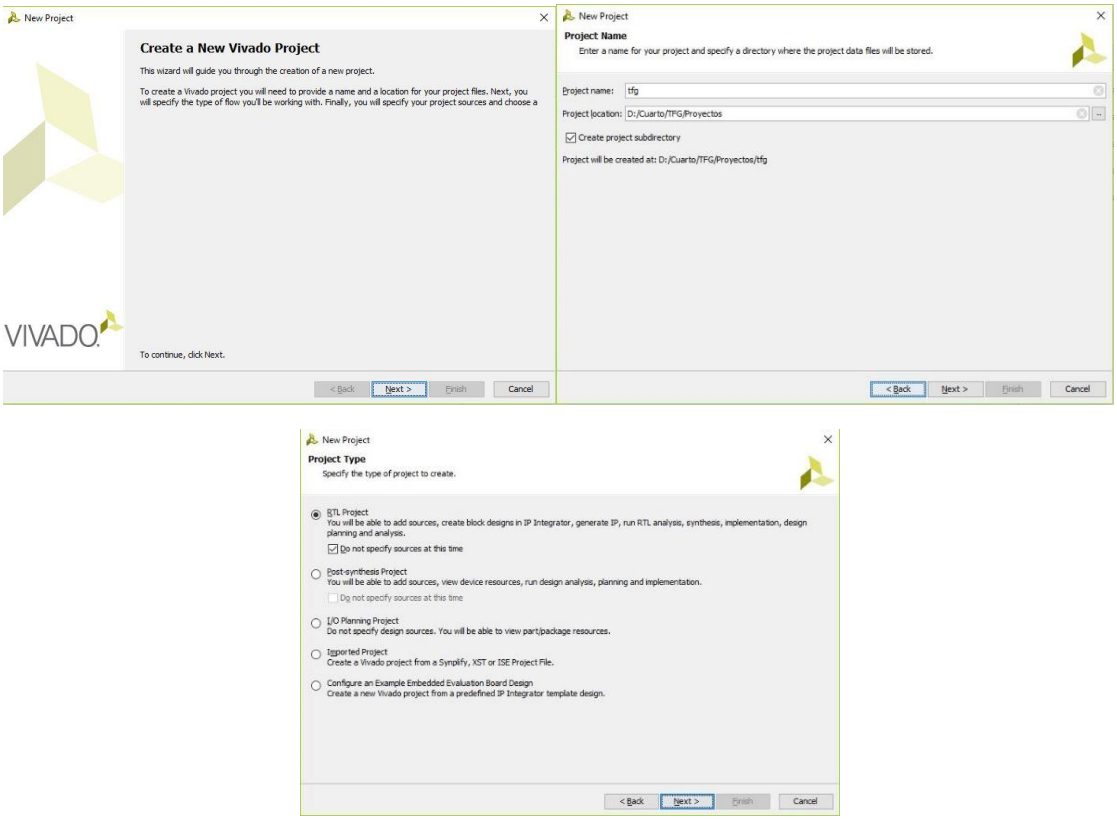

Figura 5.2: Pantallas de creación de un nuevo proyecto en Vivado

En la ventana *Default Part* que se muestra en la Figura 5.3, se debe indicar cuál es la placa de desarrollo que se está empleando. Para ello, se selecciona *Boards* y, posteriormente, en el campo *Vendor* se elige *em.avnet.com*. En la parte inferior de la ventana aparecen dos placas de desarrollo debiendo seleccionar *ZedBoard Zynq Evaluation and Development Kit*.

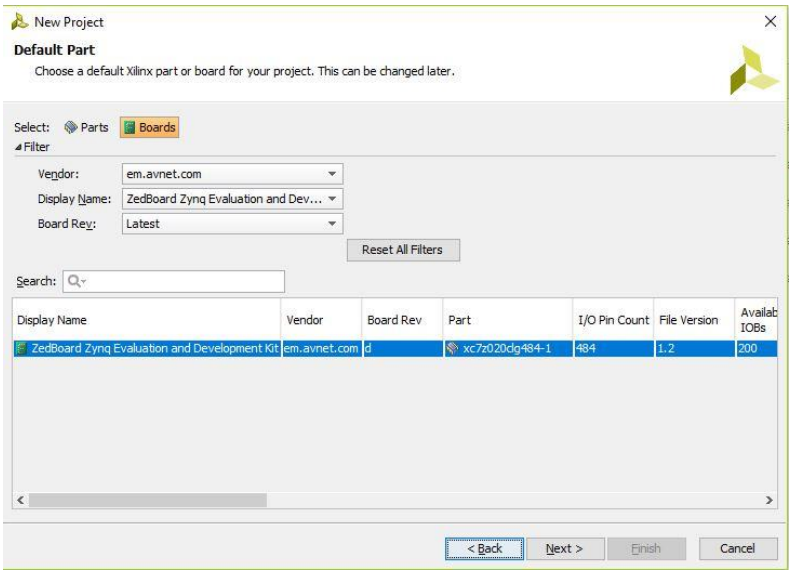

Figura 5.3: Selección de dispositivo hardware en Vivado

Tras pulsar *Next*, se muestra un resumen de las características del proyecto que se va a crear. Por último, se pulsa en *Finish*.

5.1.2 Elaboración del diseño

Una vez creado el proyecto se selecciona la opción *Create Block Design* que se encuentra en el menú *Flow Navigator -> IP Integrator*. En la ventana que se abre se pulsa *Ok* dejando las opciones por defecto. Este paso genera una nueva ventana llamada *Diagram*, como la mostrada en la Figura 5.4, donde se realizará el diseño.

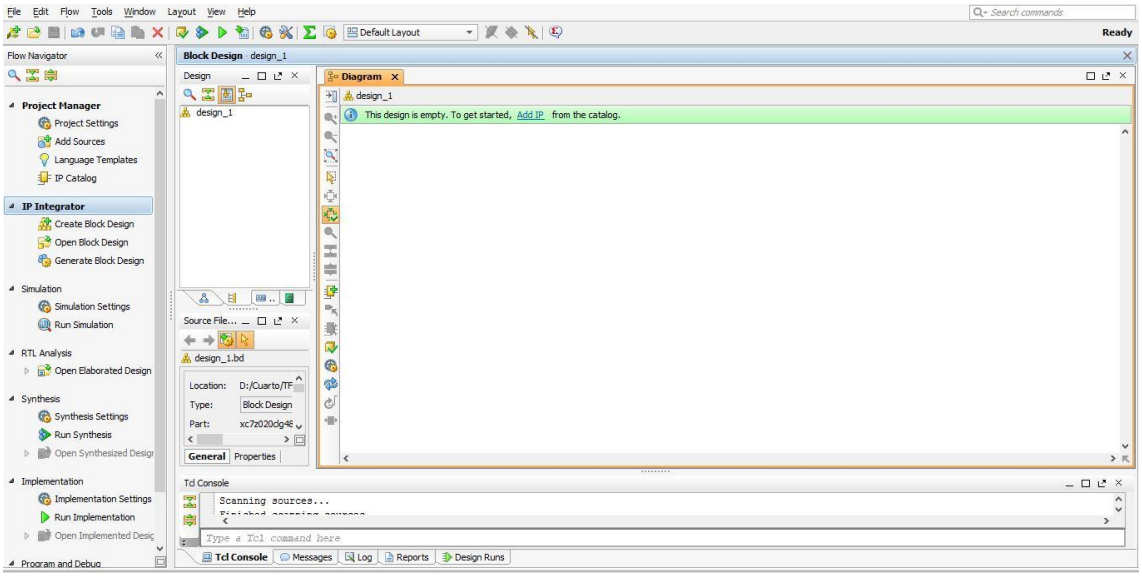

Figura 5.4: Pantalla inicial Diagram

El primer paso es añadir el *IP Core* que representa la arquitectura ZYNQ7. Para ello, se pulsa en el icono *Add IP* que se puede encontrar a la izquierda de la ventana *Diagram*. En la ventana que se abre, mostrada en la Figura 5.5, se debe buscar el componente *ZYNQ7 Processing System* y hacer doble *click* sobre él para añadirlo al diseño.

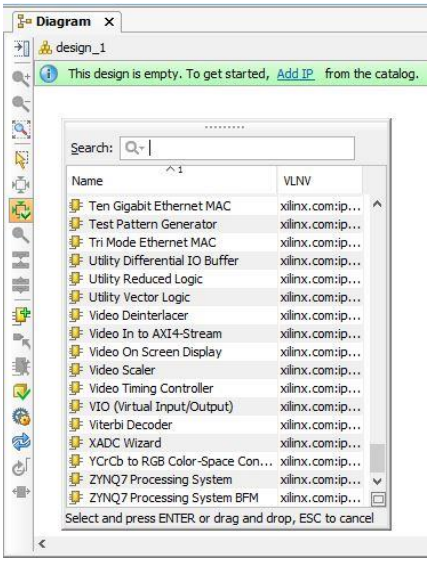

Figura 5.5: Ventana Add IP

Cuando el proceso ha terminado, en la parte superior de la ventana *Diagram* aparece un asistente que permite automatizar algunas conexiones necesarias. Se selecciona *Run Block Automation* y, en la ventana que surge, se selecciona el componente *processing_system7_0* y se pulsa *Ok*. Este proceso crea las conexiones DDR y FIXED_IO. En la Figura 5.6 se puede observar el diseño antes y después de la aplicación del asistente de conexión.

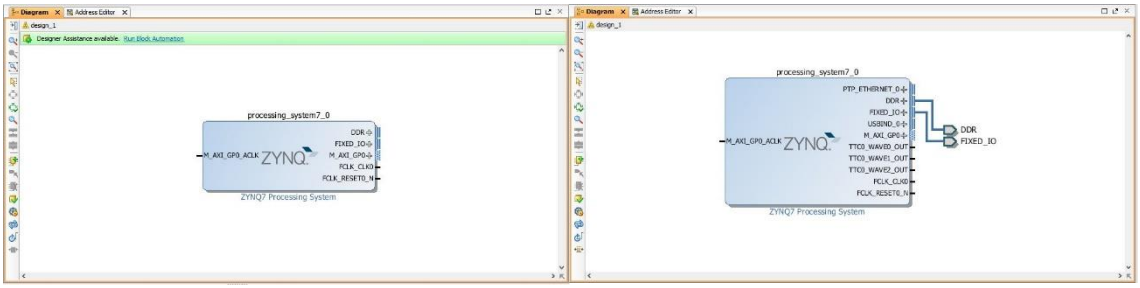

Figura 5.6: Conexiones antes y después de aplicar el asistente de conexión

5.1.3 Configuración de ZYNQ7

Para continuar con el diseño, se debe modificar la configuración del componente que se acaba de agregar, añadiendo las características que el módulo WiFi necesita para funcionar. Para ello, se hace doble click sobre el módulo *processing_system7_0*. En la ventana que se abre, mostrada en la Figura 5.7, se pueden ver los módulos que son configurables de color verde. En el caso de los *I/O Peripherals* se marcan con un *tick* los que se encuentran activos actualmente. Para modificar las características de dichos periféricos, se debe hacer *click* sobre el periférico que se quiera modificar o hacer *click* en *MIO Configuration* que se encuentra en la parte izquierda de la ventana y, a continuación, desplegar *I/O Peripherals*.

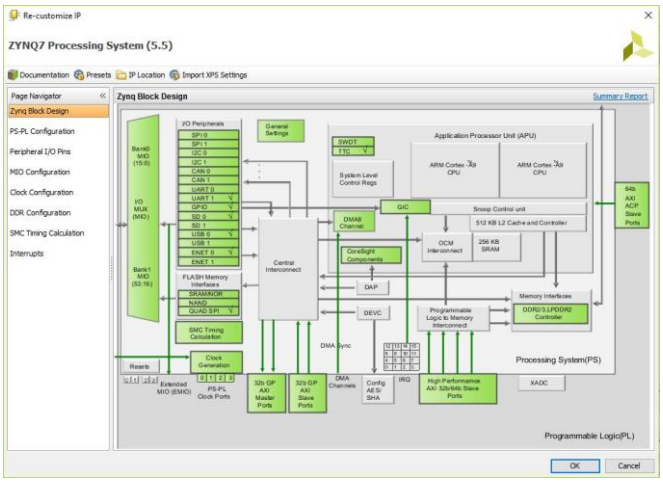

Figura 5.7: Configuración por defecto del bloque ZYNQ7 Processing System

Uno de los periféricos más importantes que se va a utilizar es la UART1 ya que es el encargado de activar la conexión serie a través del puerto micro-usb de la ZedBoard. A través de este puerto se podrá conocer el estado de ejecución del sistema y enviar comandos para provocar diferentes acciones. Para activar este módulo, se hace *click* sobre *UART 1* y se selecciona los puertos *MIO 48 .. 49*. La configuración de la UART1 se puede observar en la Figura 5.8.

UART 1	MIO 48 .. 49					
Modem signals						
UART 1	MIO 48	tx	LVCMOS 1.8V	slow	disabled	out
UART 1	MIO 49	rx	LVCMOS 1.8V	slow	disabled	in

Figura 5.8: Configuración de UART1 en ZYNQ7 Processing System

Además, se debe indicar la velocidad de trabajo de la UART1. Esta velocidad se selecciona en el apartado *PS-PL Configuration* en la opción *General*. En este caso, se va a dejar el valor que por defecto está seleccionado de 115200 baudios tal y como se puede observar en la Figura 5.9.

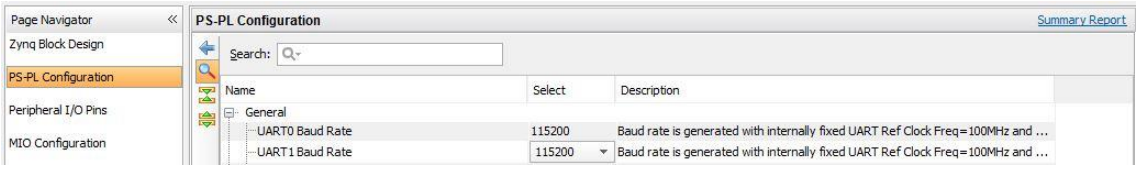

Figura 5.9: Configuración velocidad de UART en ZYNQ7 Processing System

Otro de los periféricos que se va a emplear es el lector de tarjetas SD. Para el funcionamiento del módulo WiFi, se va a ejecutar un sistema operativo en la placa de desarrollo ZedBoard. Todos los archivos necesarios para ello se almacenan en una tarjeta SD que debe ser leída con el periférico SD 0. Para activar este módulo se debe hacer *click* en *SD 0* y seleccionar los puertos *MIO 40 .. 45*. Además, debe estar activa la señal *Write Protect* o *WP* en el puerto *MIO 46* y la señal *Card Detect* o *CD* en el puerto *MIO 47*. La configuración del periférico SD 0 se puede observar en la Figura 5.10.

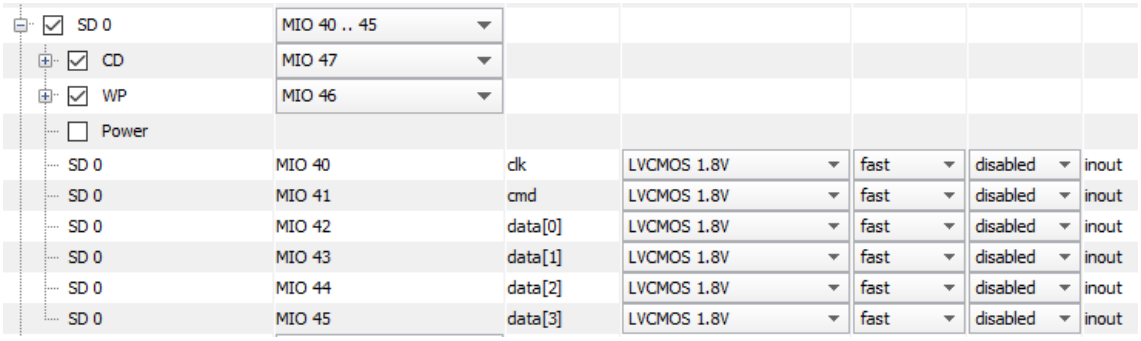

Figura 5.10: Configuración SD 0 en ZYNQ7 Processing System

El módulo WiFi se conecta a la placa ZedBoard a través de la interfaz PS Pmod y, a su vez, al periférico SDIO de 4 bits. Este periférico debe ser activado en la configuración del módulo ZYNQ7. Tras desplegar *I/O Peripherals* se tiene que activar el periférico *SD 1* y seleccionar en la columna *IO* los puertos *MIO 10 .. 15*. Además, es necesario activar la señal *CD* del periférico *SD 1* y conectarlo a la interfaz EMIO. Esta opción *Card Detect* debe estar habilitada y debe tener un nivel bajo para obtener un correcto funcionamiento del módulo WiFi, tal y como posteriormente se configurará en el archivo de *constraints*. La configuración del periférico SD 1 se puede observar en la Figura 5.11.

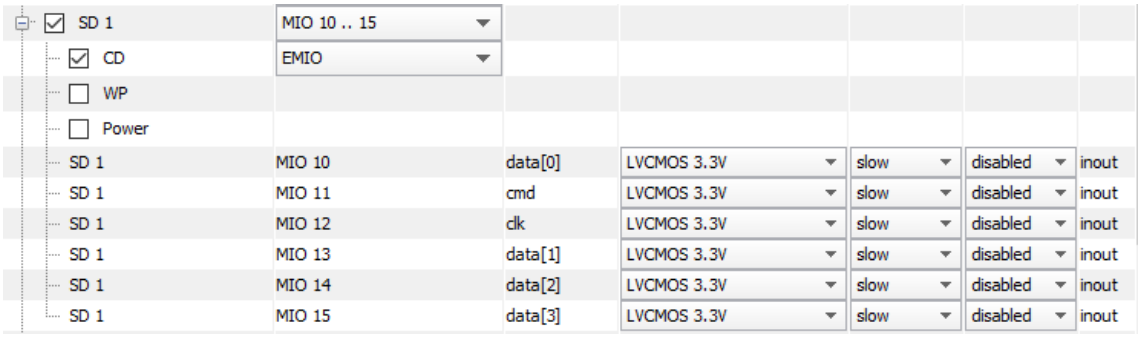

Figura 5.11: Configuración SD 1 en ZYNQ7 Processing System

Por último, se debe configurar la interfaz GPIO seleccionando GPIO MIO tal y como se puede observar en la Figura 5.12.

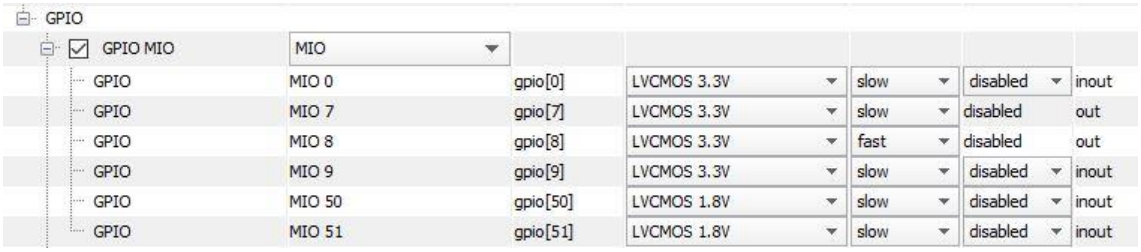

Figura 5.12: Configuración de GPIO en ZYNQ7 Processing System

En este mismo apartado, se despliegan las diferentes opciones del apartado *Application Processor Unit* y se deselecciona la opción *Timer 0* ya que no se va a utilizar.

En el apartado *PS-PL Configuration* se despliega *GP Master AXI Interface* y se deselecciona la opción *M AXI GP0 interface*.

En el apartado *Clock Configuration* se despliega *PL Fabric Clocks* y se desactiva *FCLK_CLK0*.

Tras realizar todos los cambios anteriores, se hace *click* en *Ok* para que la configuración se guarde. El resultado obtenido se puede observar en la Figura 5.13.

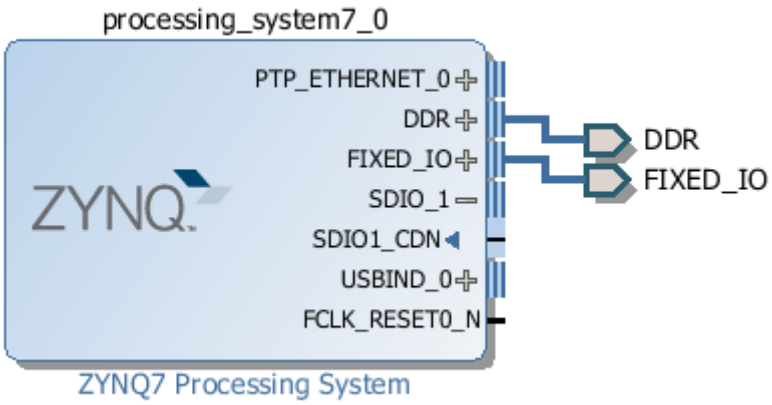

Figura 5.13: Resultado tras la configuración del bloque ZYNQ7 Processing System

Antes de generar el archivo de configuración Hardware, hay que indicar que el puerto SDIO1_CDN es externo. Para ello, se selecciona la conexión y, tras pulsar el botón derecho, se selecciona la opción *Make External*. El resultado puede observarse en la Figura 5.14.

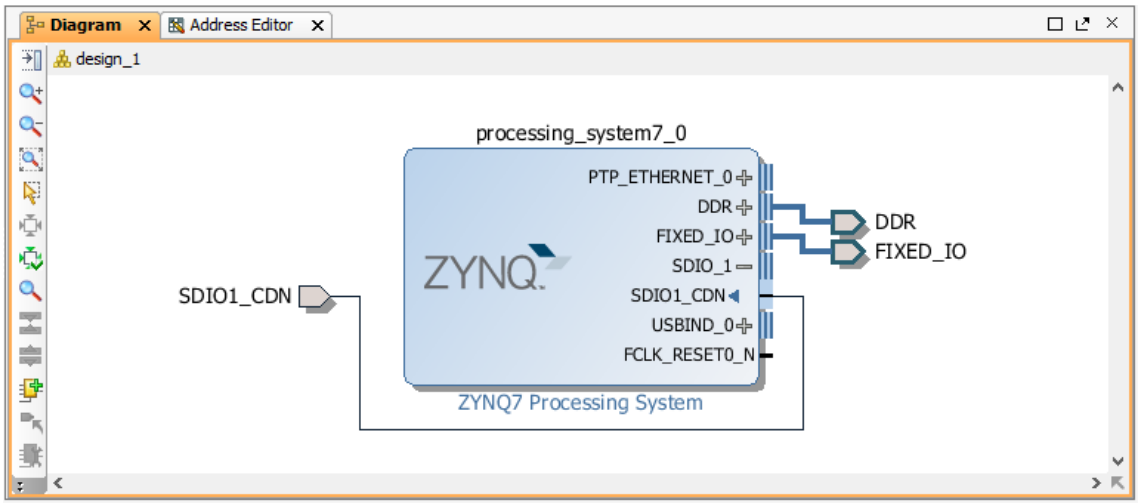

Figura 5.14: Resultado tras configurar el puerto como externo

5.1.4 Generación archivo .bit

Una vez reconfigurado el módulo ZYNQ, se debe guardar el proyecto. Antes de realizar la síntesis, sobre la ventana *sources*, se hace *click* con el botón derecho y se selecciona *Create HDL Wrapper…* En la ventana que aparece, mostrada en la Figura 5.15, se deja seleccionada la opción por defecto *Let Vivado manage wrapper and auto-update* y se hace *click* en *Ok*.

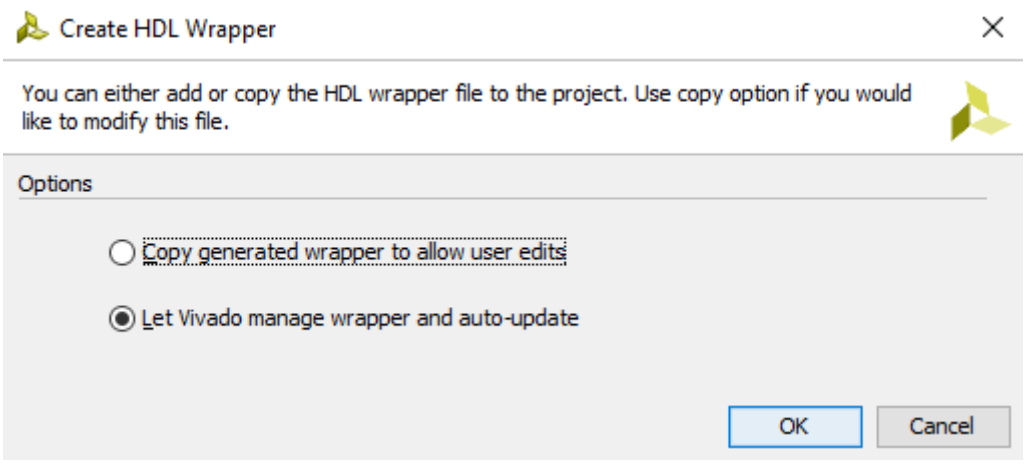

Figura 5.15: Creación de top level HDL Wrapper

Antes de generar el archivo de configuración de la parte reconfigurable, es necesario generar el archivo de *constraints*. En este archivo se establece la relación entre el puerto EMIO y la conexión en la placa ZedBoard. Además, se puede seleccionar el nivel de tensión en cada pin así como si dispone de resistencia de *pull-up* o *pull-down*.

Para crear el archivo de *constraints* se selecciona la opción *constraints wizard* que se encuentra en el menú *Flow Navigator -> Synthesis -> Synthesized Design*. Para que esta opción esté habilitada es necesario realizar una primera síntesis del sistema descrito. En la ventana que se abre, mostrada en la Figura 5.16, pulsamos la opción *Cancel*.

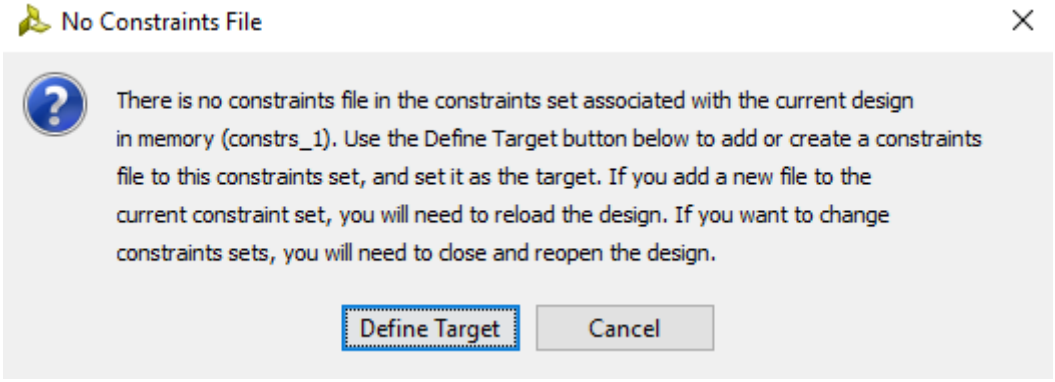

Figura 5.16: Ventana informativa archivo constraints

Se necesita tener un pin a nivel bajo que haga de *Card Detect* para la interfaz SD1. Para ello, se va a configurar un pin de una conexión Pmod al cual no se va a conectar nada. Sin embargo, se va a configurar con una resistencia de *pull-down* por lo que el nivel de dicho pin va a ser un nivel bajo. En este caso, se ha elegido el pin denominado JB1 en la placa ZedBoard que se corresponde con el pin W12. La configuración explicada anteriormente se puede observar en la Figura 5.17.

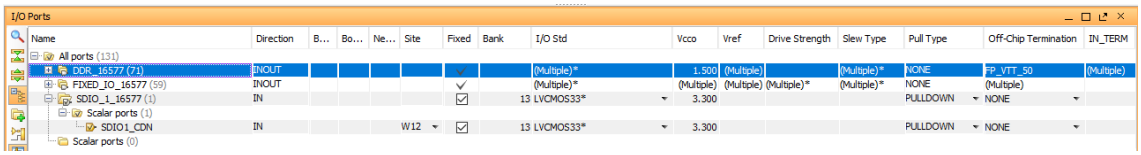

Figura 5.17: Configuración puerto SDIO1_CDN

Una vez definido el archivo de *constraints* se debe guardar. Para ello, se selecciona la opción *File -> Save Constraints*. Tras asignarle un nombre, tal y como se puede ver en la Figura 5.18, se selecciona la opción *Ok*.

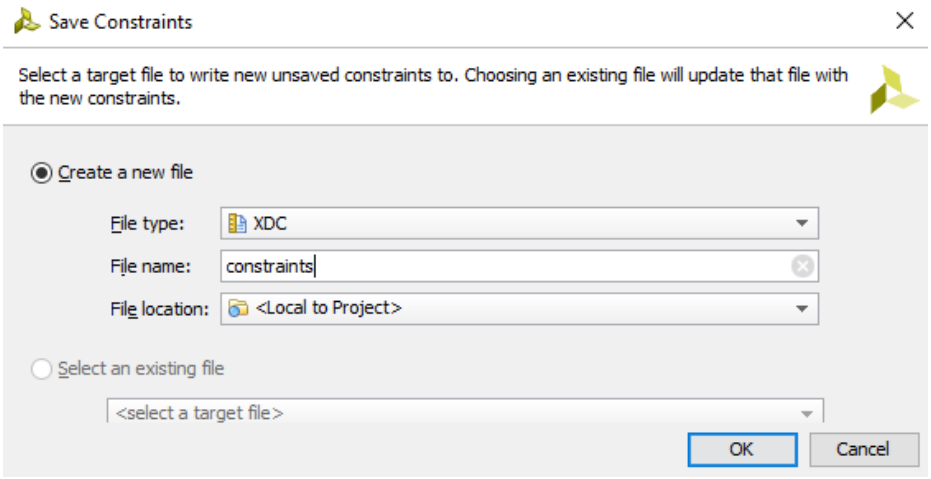

Figura 5.18: Guardado del archivo constraints

Por último, se selecciona la opción *Generate Bitstream* que se encuentra en la ventana *Flow Navigator* en el apartado *Program and Debug*. En la ventana que se abre, mostrada en la Figura 5.19, se hace *click* en la opción *Yes* para que, antes de generar el archivo .bit se realice la síntesis y la implementación del proyecto.

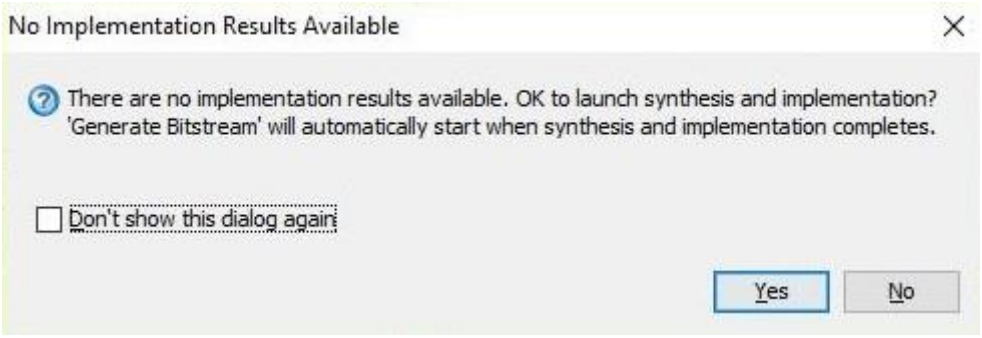

Figura 5.19: Realización de síntesis e implementación previa a la generación del archivo .bit

El diseño descrito anteriormente no hace uso de la parte lógica programable de la que dispone la placa ZedBoard.

5.1.5 Exportación del diseño hardware a Xilinx SDK

Una vez que el proceso termina, se hace *click* en *File -> Export -> Export Hardware…* En la ventana que se abre, que se muestra en la Figura 5.20, se selecciona *Include bitstream* y se hace *click* en *Ok*. Por último, se hace *click* en *File -> Launch SDK* y en la ventana que se abre se pulsa en *Ok*.

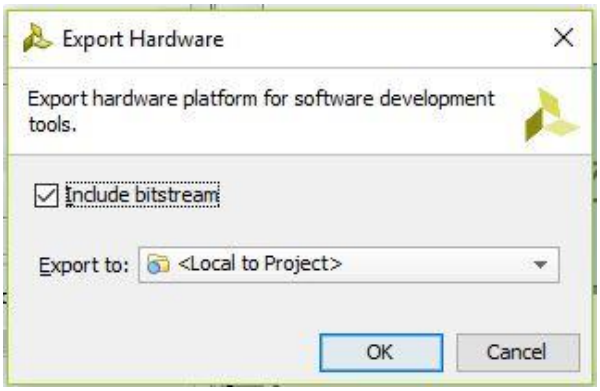

Figura 5.20: Exportación a Xilinx SDK del proyecto HW

5.2 Creación del diseño software

Para la creación del diseño software se va a emplear la herramienta Xilinx SDK. Tras realizar la exportación del diseño hardware en el apartado anterior, la herramienta carga estos archivos a partir de los cuales se realiza el diseño software.

5.2.1 Board Support Package (BSP)

El *Board Support Package* es un conjunto de bibliotecas y drivers que forman parte de la capa más baja de la arquitectura software. Ofrece servicios software basados en el procesador y los periféricos que forman el PS de Zynq. Por esto, está asociado a una plataforma hardware.

Una vez que el programa se está ejecutando, se debe crear el *Board Support Package* (BSP). Para ello, se selecciona *File -> New -> Board Support Package*. En la ventana que se abre, mostrada en la Figura 5.21, se dejan todos los valores por defecto y se selecciona *Finish*.

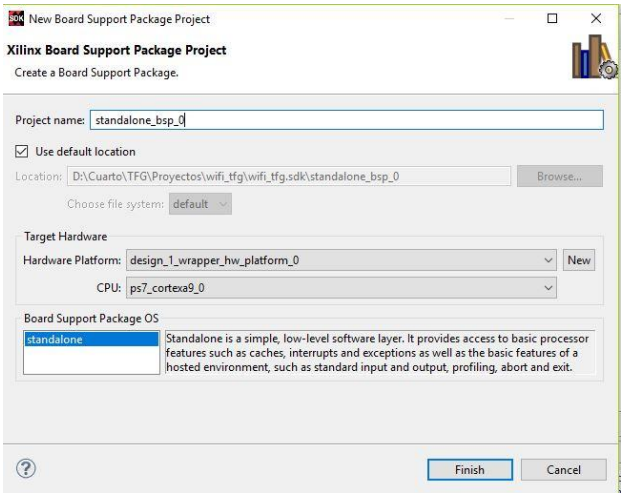

Figura 5.21: Creación de BSP en Xilinx SDK

A continuación, en la ventana *Board Support Package Settings* se debe seleccionar las bibliotecas *xilffs* y *xilrsa* y se pulsa en *Ok*.

La biblioteca *xilffs* es una biblioteca genérica de archivos FAT que debe ser añadida para el empleo de proyectos que empleen el controlador de tarjetas SD.

La biblioteca *xilrsa* proporciona algoritmos de encriptación y desencriptación RSA y un algoritmo de generación de *hash* SHA2.

5.2.2 First Stage BootLoader (FSBL)

Para terminar, se debe crear el archivo FSBL. Para ello, se selecciona *File -> New -> Application Project*. En la ventana que surge se debe introducir *zynq_fsbl_0* en el campo *Project name* y se selecciona *Use existing* en el campo *Board Support Package*. Esta ventana se puede observar en la imagen de la izquierda de la Figura 5.22.

Se pulsa *Next* y en la siguiente ventana, mostrada en la imagen de la derecha de la Figura 5.22, se selecciona *Zynq FSBL* y se pulsa *Finish*. En la ventana *Project Explorer*, en el proyecto de aplicación dentro de la carpeta *Debug* puede encontrarse el archivo *zynq_fsbl_0.elf* necesario para el correcto arranque del sistema operativo en la placa de desarrollo ZedBoard.

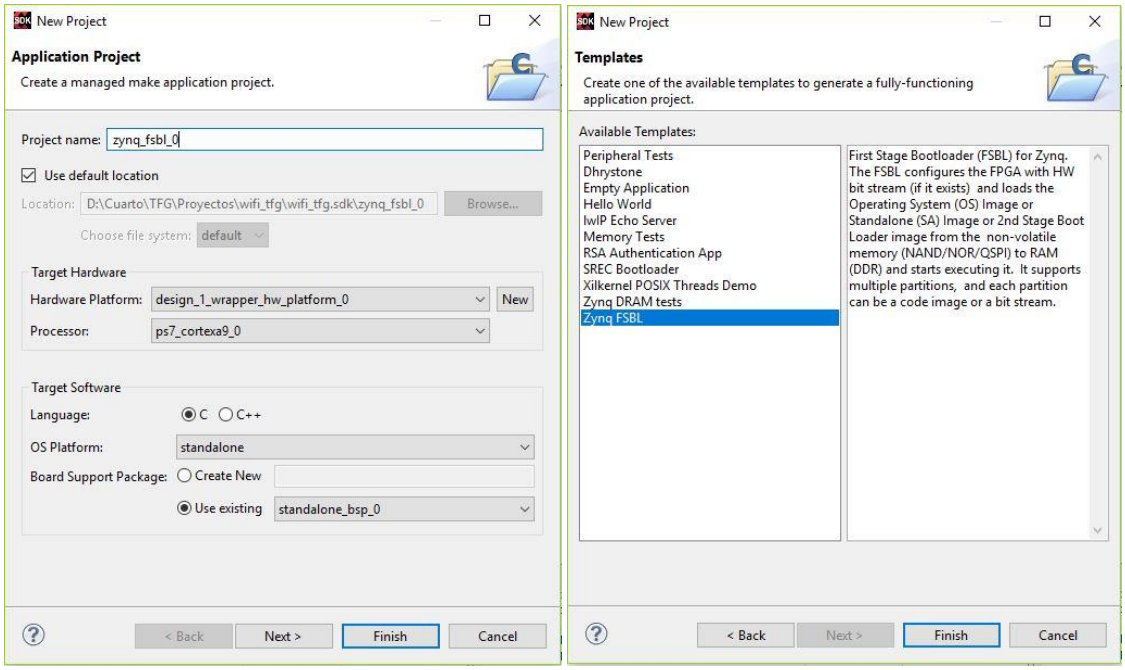

Figura 5.22: Creación de FSBL en Xilinx SDK

Una vez realizados los pasos anteriores, ya se han creado dos de los archivos que se necesitan para la creación del archivo imagen de arranque del SO Linux BOOT.bin:

- El archivo design_1_wrapper.bit se encuentra en la ruta de directorios: <directorio del proyecto Vivado>/sdk/design_1_wrapper_hw_platform_0/design_1_wrapper.bit
- El archivo zynq_fsbl_0.elf se encuentra en: <directorio del proyecto Vivado>/sdk/zynq_fsbl_0/Debug/zynq_fsbl_0.elf

5.3 Compilación kernel de Linux

En esta sección se va a generar un sistema Linux adaptado de forma que este tenga un tamaño reducido incluyendo únicamente las características básicas necesarias para un funcionamiento óptimo en la tarjeta ZedBoard. Para ello, se va a seguir un desarrollo que se muestra esquematizado en la Figura 5.23.

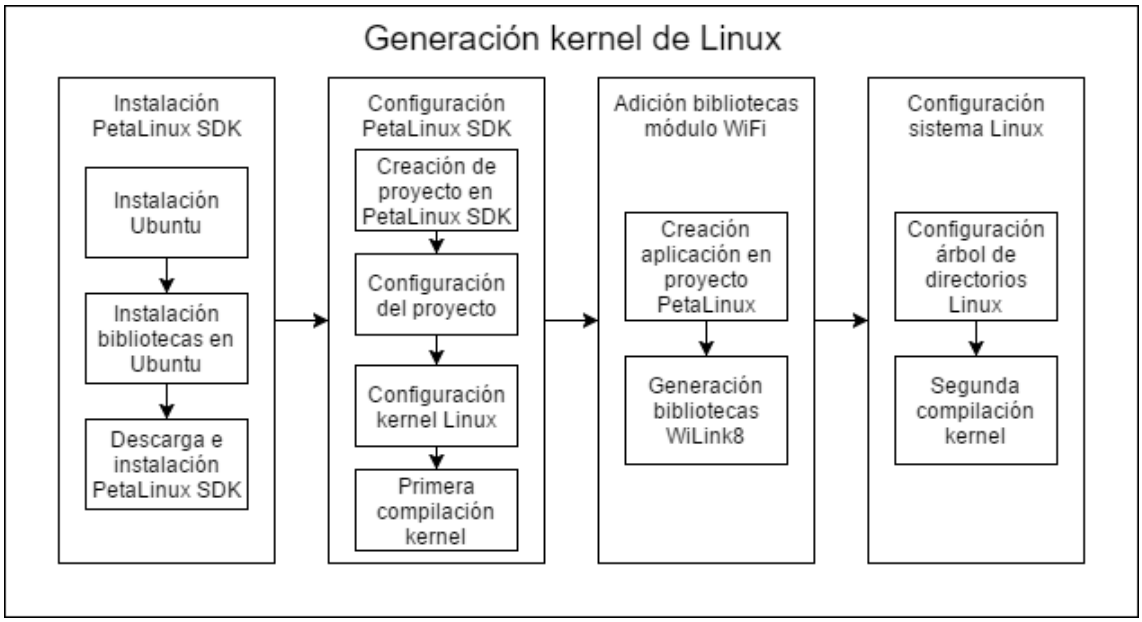

Figura 5.23: Diagrama de flujo generación del kernel de Linux

Para la compilación del kernel de Linux se va a utilizar un sistema operativo distinto al empleado en los capítulos anteriores. En este caso, se va a emplear una máquina virtual en la cual se instalará Ubuntu, un sistema operativo de código abierto basado en Linux. Para llevar a cabo la compilación del kernel, se va a utilizar la herramienta PetaLinux SDK de Xilinx instalada en dicho sistema operativo.

El software PetaLinux SDK generará los archivos necesarios para el correcto arranque del sistema Linux. Estos archivos son el *Second Stage Boot Loader* y el archivo imagen del sistema.

El SSBL o *Second Stage Boot Loader* es el segundo archivo de arranque que emplea el sistema para ejecutar el sistema operativo. Este archivo es denominado *u-boot* y es un ejecutable que gestiona el arranque del sistema operativo Linux.

El segundo de los archivos generado por PetaLinux SDK es el archivo imagen del sistema. Tiene una extensión .ub y en él se incluye el sistema Linux propiamente dicho.

Ambos archivos están muy ligados entre sí de forma que cualquier modificación del sistema hace necesario la actualización de los dos archivos tras una nueva compilación.

5.3.1 Instalación de PetaLinux SDK sobre Ubuntu 64 bits

Lo primero que se debe hacer es instalar la máquina virtual para lo cual existen diferentes alternativas como VMWare o VirtualBox. En este trabajo se va a emplear el software VMWare para ejecutar el sistema operativo Ubuntu sobre Windows. Además, se recomienda habilitar el portapapeles compartido ya que esto simplificará la compartición de archivos entre los dos sistemas operativos.

Una vez instalado el programa VMWare se debe descargar el sistema operativo que se va a instalar. En este caso se va a utilizar Ubuntu 16.04.2 [24] de 64-bits ya que esta es la versión estable más actual, pero, en cualquier caso, se podría emplear cualquier otra versión anterior.

Una vez instalado Ubuntu, es necesario instalar PetaLinux SDK ya que es el programa que se empleará para compilar el núcleo de Linux que se ejecutará en la tarjeta de desarrollo ZedBoard.

Para que el programa funcione correctamente necesita una serie de bibliotecas que deben ser instaladas previamente (en modo administrador). Para ello, se abre un terminal y se introducen los siguientes comandos:

- sudo apt-get install tofrodos
- sudo apt-get install iproute
- sudo apt-get install gawk
- sudo apt-get install gcc
- sudo apt-get install git-core
- sudo apt-get install make
- sudo apt-get install net-tools
- sudo apt-get install ncurses-dev
- sudo apt-get install tftpd
- sudo apt-get install zlib1g-dev
- sudo apt-get install flex
- sudo apt-get install bison
- sudo apt-get install libtool
- sudo apt-get install python-m2crypto
- sudo apt-get install autoconf
- sudo apt-get install device-tree-compiler

Como en este caso se está empleando un sistema operativo de 64 bits, una vez instaladas las bibliotecas anteriores se deben instalar también las bibliotecas compatibles con la versión de 32 bits necesarias para el correcto funcionamiento de PetaLinux SDK. Por tanto, introducimos en un terminal los siguientes comandos:

- sudo apt-get install ncurses-dev:i386
- sudo apt-get install libstdc++6:i386
- sudo apt-get install libselinux1:i386
- sudo apt-get install lib32ncurses5-dev
- sudo apt-get install lib32ncurses5
- sudo apt-get install lib32ncursesw5

Una vez que están todas las bibliotecas necesarias correctamente instaladas, se procede a la instalación del programa PetaLinux SDK. En este caso se va a instalar la versión 2014.4 ya que esta versión debe coincidir con la versión instalada de las herramientas de Xilinx: Vivado 2014.4 y Xilinx SDK 2014.4. Para ello, se abre el navegador web y se introduce en la barra de direcciones la web de descargas de Xilinx. En la página web se debe acceder a la sección *Embedded Development* y seleccionar la versión 2014.4 tal y como se muestra en la Figura 5.24. Una vez seleccionada la versión, se debe descargar *PetaLinux 2014.4 Installation archive for Zynq and MicroBlaze*. Para poder realizar la descarga es necesario registrarse como usuario de la web. Una vez introducidos nuestros datos, comenzará la descarga.

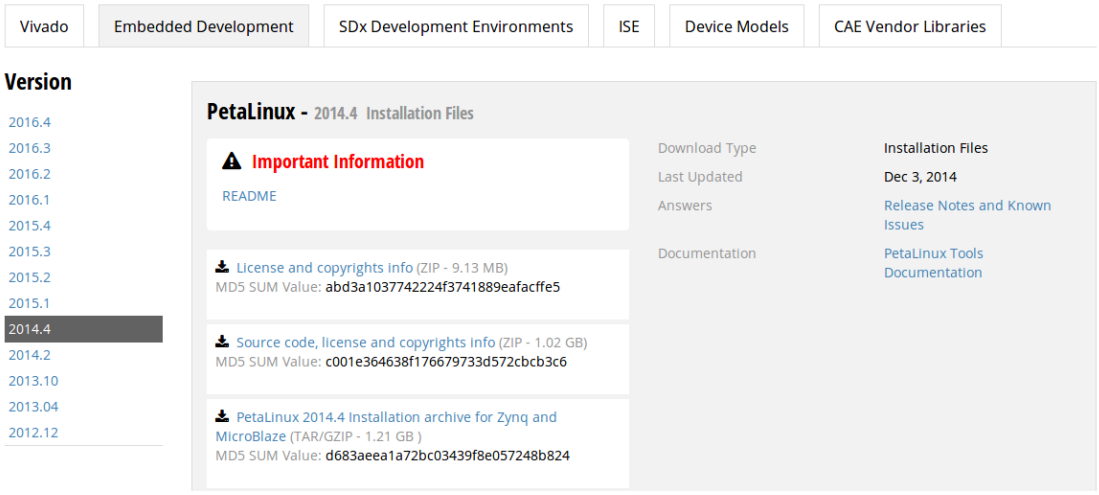

Figura 5.24: Página de descarga PetaLinux 2014.4 [25]

Una vez concluida la descarga, se debe proceder a instalar PetaLinux SDK [26]. Para ello, se introduce en el terminal:

- cd Descargas
- chmod +x petalinux-v2014.4-final-installer.run
- sudo ./petalinux-v2014.4-final-installer.run /opt/pkg
- Para concluir la instalación, debemos aceptar las licencias. Para ello, primero se debe pulsar *enter* dos veces cuando se indica y, a continuación, *q* para salir del texto. Después, se debe pulsar *y* cuando se pregunte si aceptamos la licencia. Posteriormente se debe repetir el proceso para aceptar otra licencia.

En la Figura 5.25 se muestra el terminal de Ubuntu en el cual se puede observar el proceso de instalación de la herramienta PetaLinux SDK.

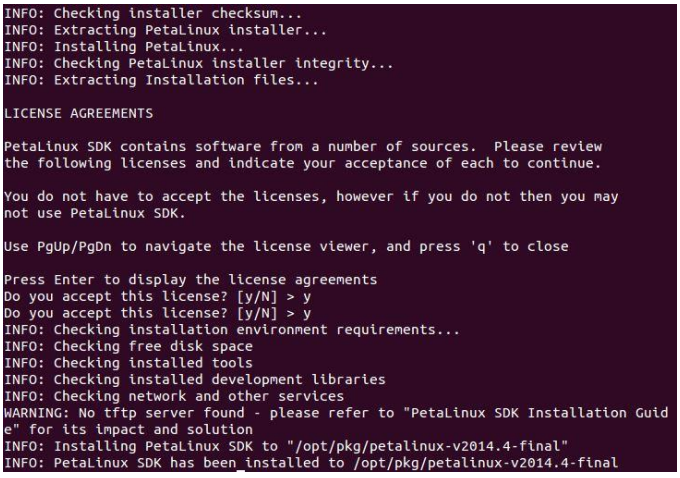

Figura 5.25: Instalación de PetaLinux en un sistema Ubuntu

Una vez que la instalación se ha realizado con éxito, se debe cambiar el intérprete de comandos de Ubuntu para que PetaLinux SDK funcione correctamente. Para ello, se debe ejecutar el comando:

- sudo dpkg-reconfigure dash

La ejecución del comando anterior mostrará una ventana como la mostrada en la Figura 5.26 en la cual, se debe seleccionar la opción *No*.

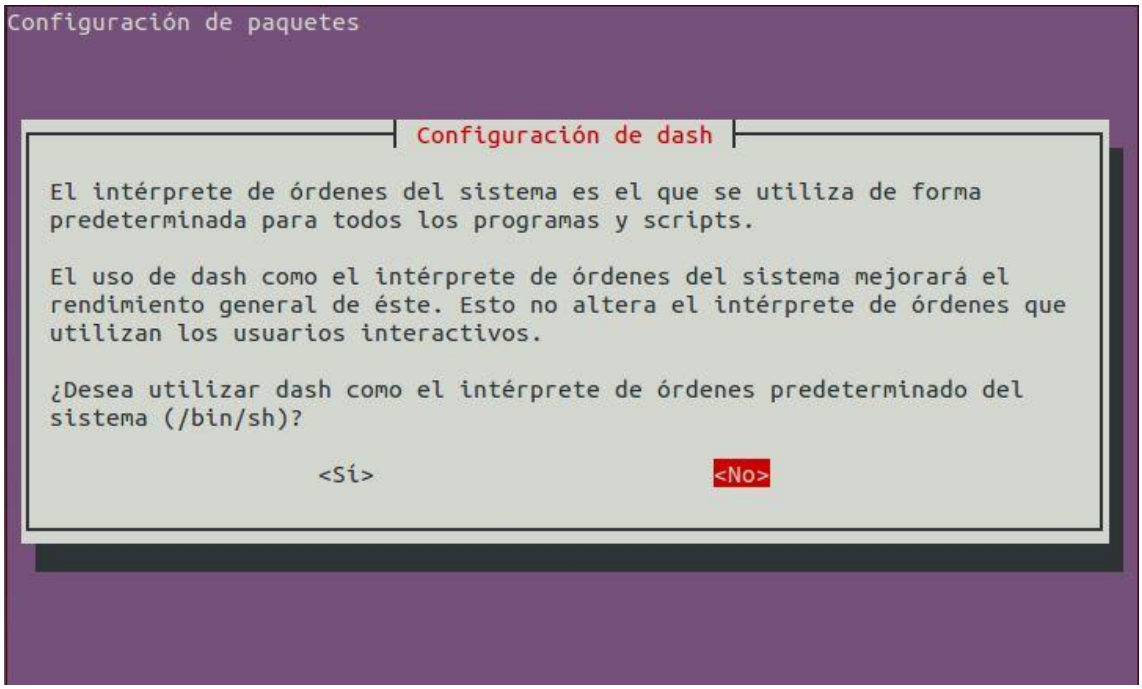

Figura 5.26: Modificación del intérprete de comandos de Ubuntu

Por último, para poder usar las herramientas de PetaLinux SDK, cada vez que se deseen emplear, se debe indicar donde se encuentran las variables del entorno con la siguiente instrucción:

- source /opt/pkg/petalinux-v2014.4-final/settings.sh

La ejecución del comando anterior devuelve un resultado similar al mostrado en la Figura 5.27 si todos los elementos han sido instalados correctamente. Cabe destacar que el aviso sobre la falta de un servidor tftp instalado no tiene ningún efecto ya que no se va a emplear en ningún momento a lo largo de este trabajo.

```
PetaLinux environment set to '/opt/pkg/petalinux-v2014.4-final'
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide
" for its impact and solution
```

Figura 5.27: Resultado devuelto al indicar las variables del entorno PetaLinux

5.3.2 Creación de un proyecto en PetaLinux SDK

Para la creación de un nuevo proyecto en PetaLinux SDK se emplea la siguiente instrucción:

- petalinux-create –t project –n <nombre del proyecto PetaLinux> --template zynq

La ejecución de la orden anterior creará una carpeta en el directorio padre donde se haya ejecutado con el nombre indicado.

En la carpeta del proyecto creado en Vivado en Windows, una vez realizados los pasos indicados por el *wizard* del SDK explicados en el apartado 5.2, existe un directorio llamado <nombre del proyecto Vivado>.sdk. Esta carpeta se debe copiar al nuevo directorio creado por PetaLinux en Ubuntu.

Para añadir la descripción hardware y software creada por Vivado y Xilinx SDK que se encuentra en la carpeta <nombre del proyecto Vivado>.sdk al proyecto PetaLinux, se debe ejecutar:

- cd <nombre del proyecto PetaLinux>
- petalinux-config --get-hw-description=<nombre del proyecto Vivado>.sdk

En la ventana que se abre al ejecutar la anterior instrucción se deben realizar los siguientes cambios:

- Como el tamaño del archivo image.ub ocupará más o menos 20 MB, se debe aumentar el offset en la configuración del SSBL. Esta opción se encuentra en la opción *u-boot configuration -> netboot offset 0x4000000*. La configuración de esta opción se puede observar en la Figura 5.28.

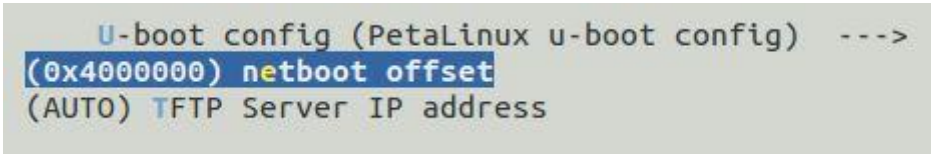

Figura 5.28: Modificación del offset en configuración del SSBL

- Para poder modificar el archivo de dispositivos hardware accesibles de forma más rápida, se va a almacenar el *device tree* directamente en la partición FAT de la tarjeta SD en vez de que este se incluya en el archivo imagen del sistema. Esto debe ser configurado en la opción: *Subsystem AUTO Hardware Settings -> Advanced bootable images storage settings -> dtb image settings -> image storage media (from boot image) -> primary sd*, tal y como puede observarse en la Figura 5.29.

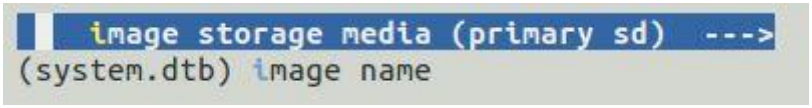

Figura 5.29: Ubicación de Device Tree en tarjeta SD

- Se deselecciona la compresión del kernel que se encuentra en la opción *Image Packaging Configuration -> Compress kernel image*, tal y como puede observarse en la Figura 5.30. La compresión del kernel no se lleva a cabo en esta ocasión porque, tras la realización de diversas pruebas, se detectó un problema en el momento de arranque del sistema que tenía lugar en el momento de descompresión del kernel. Por ello, y debido a que el conjunto de archivos necesarios no ocupan una gran cantidad de espacio en la tarjeta SD, se ha decidido evitar la compresión del kernel en este capítulo.

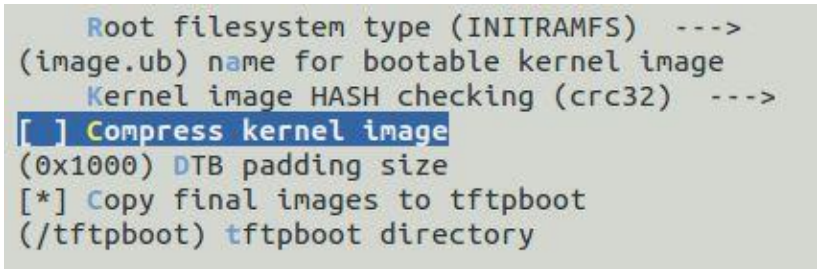

Figura 5.30: Configuración compresión de kernel

Una vez realizados los cambios, se selecciona *Save* y se pulsa *Ok*. Para salir de la ventana de configuración se debe pulsar repetidas veces en *Exit*.

5.3.3 Configuración del kernel

El siguiente paso que se debe dar es la configuración del kernel habilitando algunas características que permitan el correcto funcionamiento del módulo WiFi.

Una de las características que se debe activar está oculta, por lo que para hacerla accesible, se debe modificar el archivo *Kconfig*. Este archivo se encuentra en la ruta de directorios /opt/pkg/petalinux-v2014.4-final/components/linux-kernel/xlnx-3.17/net/wireless.

Es necesario tener permisos de administrador para poder realizar este cambio, por lo que, empleando el terminal, el comando que se debe introducir es el siguiente:

- sudo gedit /opt/pkg/petalinux-v2014.4-final/components/linux-kernel/xlnx-3.17/net/wireless/Kconfig

Una vez abierto, se debe modificar el documento añadiendo las siguientes líneas de forma que la opción *Enable Wireless Extension* quede visible para posteriormente poderla habilitar:

```
config WIRELESS_EXT
bool "Enable Wireless Extension"
default n
```

Cuando los cambios se hayan realizado, se guarda el documento. El archivo *Kconfig,* tras realizar las modificaciones anteriores, debe ser similar al mostrado en la Figura 5.31.

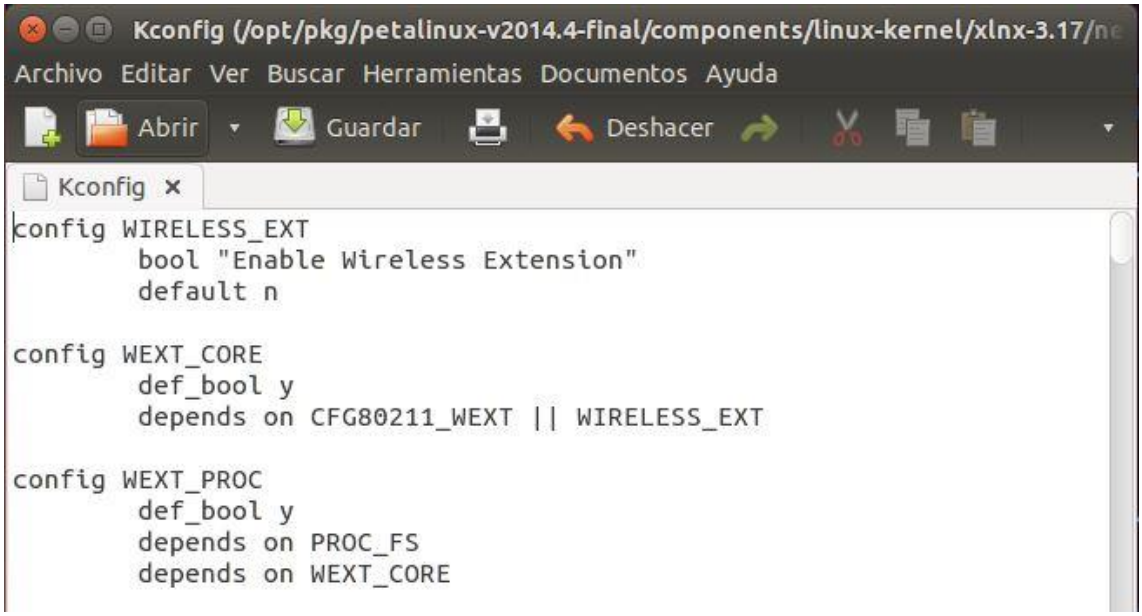

Figura 5.31: Fragmento archivo Kconfig

Los demás cambios necesarios se deben realizar desde la ventana que se abre al ejecutar el comando:

- petalinux-config –c kernel

Primero se debe habilitar la opción *WIRELESS_EXT*. Para ello, se va a emplear la herramienta de búsqueda a la que se accede con */*. En la nueva ventana se busca *WIRELESS_EXT* y se pulsa *OK*. El resultado de la búsqueda puede observarse en la Figura 5.32.

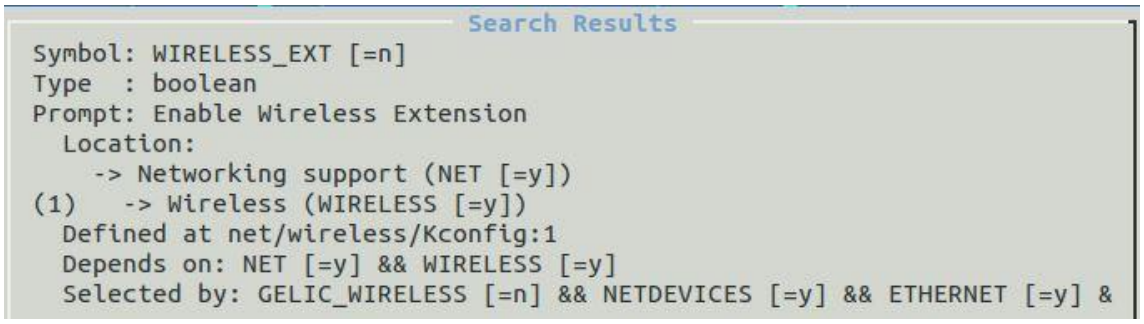

Figura 5.32: Resultado de búsqueda en la configuración de kernel

Para habilitar la opción buscada, se pulsa *1* y, posteriormente, *y* en el campo *Enable Wireless Extension*. De esta forma, esta opción quedará marcada, tal y como puede observarse en la Figura 5.33.

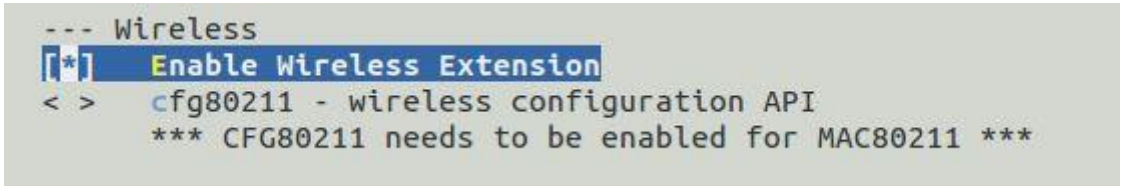

Figura 5.33: Activación de la configuración "Wireless Extension"

Para realizar los demás cambios necesarios, hay que desplazarse por los menús activando diferentes parámetros relacionados con diferentes sistemas de encriptación, códigos contra errores, sistemas de seguridad y soporte para comunicaciones inalámbricas. Las opciones que se deben activar son las siguientes:

- Cryptographic API
  - CCM Support
  - GCM Support
  - ECB Support
  - Michael Mic keyed digest algorithm
  - Arc4 cipher algorithm
- Library routines
  - CRC7 functions
- Device Drivers
  - Input device support
    - Miscellaneous devices
  - Userspace I/O drivers
- Networking support
  - RF switch subsystem support
- Security option
  - Enable access key retention support
  - Enable different security models

Para terminar de configurar el kernel, se debe buscar las siguientes configuraciones, empleando el operador / indicando:

- *CMDLINE_FROM_BOOTLOADER* y elegir la opción *Extend bootloader kernel arguments*.
- *INPUT_UINPUT* y activar la opción pulsando *y* (*User drivers support*)

Una vez que todas las opciones anteriores estén activadas, se selecciona *Save*, a continuación, *Ok* y, por último, *Exit*.

Para salir de la ventana de configuración de kernel, se deben pulsar repetidas veces *Exit* hasta que se salga de todos los menús previos.

Antes de proceder a la primera compilación del kernel, se debe aplicar un parche que se encuentra entre los archivos necesarios para la creación de las bibliotecas del módulo WiFi. Todos los archivos necesarios para la generación de las bibliotecas se encuentran en un repositorio git. Para descargarlo, en una carpeta externa a la creada anteriormente para el proyecto PetaLinux, ejecutamos el siguiente comando en un terminal:

- git clone git://git.ti.com/wilink8-wlan/build-utilites.git

Cuando termina la descarga, se habrá creado una carpeta llamada *build-utilites*. Para aplicar el parche, se debe copiar el archivo ubicado en build-utilites/patches/kernel_patches/imx-3.14.28/0010-mmc-Add-SDIO-function-devicetree-subnode-parsing.patch en la carpeta /opt/pkg/petalinux-v2014.4-final/components/linux-kernel/xlnx-3.17. Una vez ubicado en el directorio anterior se debe aplicar el parche. Los pasos anteriormente descritos se pueden ejecutar con los siguientes comandos:

- sudo cp build-utilites/patches/kernel_patches/imx-3.14.28/0010-mmc-Add-SDIO-function-devicetree-subnode-parsing.patch /opt/pkg/petalinux-v2014.4-final/components/linux-kernel/xlnx-3.17
- sudo patch -p1 < /opt/pkg/petalinux-v2014.4-final/components/linux-kernel/xlnx-3.17/0010-mmc-Add-SDIO-function-devicetree-subnode-parsing.patch

Una vez seleccionadas todas las configuraciones necesarias en el kernel y aplicado el parche anterior, se procede a compilar el kernel de PetaLinux. Para ello, se ejecuta la siguiente instrucción en el terminal en el directorio donde se encuentre el proyecto:

- petalinux-build

Una vez que el proceso haya terminado, en el directorio del proyecto se habrá creado una nueva carpeta llamada *images* donde se encuentran todos los archivos necesarios para el correcto funcionamiento de Linux en ZedBoard. Sin embargo, para el correcto funcionamiento del módulo WiFi se necesita añadir una serie de bibliotecas en dicho sistema operativo.

Lo primero que se debe hacer es crear una aplicación Petalinux llamada, por ejemplo, *wireless*:

- petalinux-create -t apps -n wireless

Esta instrucción creará un nuevo directorio en la carpeta *components* llamado *apps*. En esta carpeta se ubicarán todas las aplicaciones que se creen. Actualmente existirá otra carpeta llamada *wireless* donde se encuentran los archivos que esa aplicación usará. La ejecución del comando anterior devuelve como resultado algo similar a lo que se muestra en la Figura 5.34.

```
INFO: Create apps: wireless
INFO: New apps successfully created in /home/rodrigo/Documentos/wifi_tfg/componen
ts/apps/wireless
```

Figura 5.34: Resultado devuelto al crear una aplicación en PetaLinux SDK

De los archivos creados, se debe borrar el denominado *wireless.c* puesto que no se va a necesitar.

Además, se debe crear dentro de la carpeta *wireless* un directorio llamado *fs*. Esto se puede hacer con el terminal o con el botón derecho y, a continuación, seleccionando *Carpeta nueva* y, como nombre, *fs*.

Por último, se debe modificar el archivo *Makefile*. Para ello, se abre dicho archivo con un editor de texto como gedit y se modifica para que únicamente contenga el siguiente código:

```
ifndef PETALINUX
$(error "Error: PETALINUX environment variable not set. Change to the root of
your PetaLinux install, and source the settings.sh file")
endif
include apps.common.mk
APP = settings
all: build install
.PHONY: build
build:
install:
	cp -r ./fs/* $(TARGETDIR)
clean:
```

Una vez modificado, se guarda y se cierra el editor de textos.

En la carpeta anteriormente creada, *fs*, se debe crear otro directorio llamado *etc* y, contenido dentro de este último, un archivo llamado *modules*. En este archivo se debe añadir, en líneas distintas, los distintos *drivers* que el módulo WiFi necesita:

- compat
- cfg80211
- mac80211
- wlcore
- wlcore_sdio
- wl18xx

5.3.4 Generación bibliotecas módulo WiFi

En la carpeta *build-utilites* generada anteriormente, exite un archivo denominado *setup-env.sample* que permite automatizar el proceso de compilado de las bibliotecas del módulo WiFi [27]. Este documento debe ser modificado correctamente para que todos los archivos que se van a crear se ubiquen en el lugar adecuado. En este caso, se debe modificar de la siguiente forma:

- export TOOLCHAIN_PATH=/opt/pkg/petalinux-v2014.4-final/tools/linux-i386/arm-xilinx-linux-gnueabi/bin
- export ROOTFS =/<dirección aplicación PetaLinux>/components/apps/wireless/fs

- export KERNEL_PATH=/<dirección aplicación PetaLinux>/build/linux/kernel/xlnx-3.17
- export KERNEL_VARIANT=xlnx-3.17
- export CROSS_COMPILE=arm-xilinx-linux-gnueabi-

Una vez realizadas las modificaciones, se guarda el documento con el nombre *setup-env*.

Ahora se ejecuta el comando siguiente para descargar todos los archivos necesarios para la correcta generación de las bibliotecas:

- cd build-utilites
- ./build_wl18xx.sh init

Cuando termine de ejecutarse la orden anterior, se debe ejecutar:

- ./sudo_build_wl18xx.sh update R8.7_SP1

Para ejecutar la instrucción anterior se debe tener permisos de administrador por lo que se solicitará la contraseña. Cuando el proceso acabe, se mostrará un resultado similar al obtenido en la Figura 5.35.

```
******  *******
Installing bluetooth init script: /home/rodrigo/Documentos/build-utilites/src/bt-
firmware/initscripts/TIInit_12.8.32.bts
******  *******
Installing bluetooth init script: /home/rodrigo/Documentos/build-utilites/src/bt-
firmware/initscripts/TIInit_6.12.26.bts
******  *******
Installing bluetooth init script: /home/rodrigo/Documentos/build-utilites/src/bt-
firmware/initscripts/TIInit_6.7.16.bts
******  *******
Installing bluetooth init script: /home/rodrigo/Documentos/build-utilites/src/bt-
firmware/initscripts/TIInit_7.6.15.bts
******  *******
Verifying filesystem skeleton...
ERROR  /home/rodrigo/Documentos/wifi_tfg/components/apps/wireless/fs/lib/modules/
20144317..*/updates/drivers/net/wireless/ti/wl18xx/wl18xx.ko  Not found !
md5sum: /home/rodrigo/Documentos/wifi_tfg/components/apps/wireless/fs/lib/modules
/20144317..*/updates/drivers/net/wireless/ti/wl18xx/wl18xx.ko: No existe el archi
vo o el directorio
./_build_with_sudo.sh: línea 815: [: !=: se esperaba un operador unario
ERROR  /home/rodrigo/Documentos/wifi_tfg/components/apps/wireless/fs/lib/modules/
20144317..*/updates/drivers/net/wireless/ti/wlcore/wlcore.ko  Not found !
md5sum: /home/rodrigo/Documentos/wifi_tfg/components/apps/wireless/fs/lib/modules
/20144317..*/updates/drivers/net/wireless/ti/wlcore/wlcore.ko: No existe el archi
vo o el directorio
./_build_with_sudo.sh: línea 815: [: !=: se esperaba un operador unario
Database signature verification failed.
Please update your public key used to verify the DB
Wifi Package Build Successful
```

Figura 5.35: Finalización proceso de generación de bibliotecas del módulo WiFi

Una vez que el proceso ha acabado, el siguiente paso es cambiar la configuración de PetaLinux SDK para que incluya la aplicación que se ha creado. Además, en esta ventana de configuración, se muestran diferentes aplicaciones que pueden ser activadas para que se incluyan en el sistema Linux. Una de estas aplicaciones es SSH que se va a emplear en este ejemplo.

Para acceder a la ventana de configuración, se ejecuta en la carpeta del proyecto PetaLinux:

- petalinux-config –c rootfs

En la ventana que se abre, se debe habilitar la opción *wireless* que se encuentra en el menú *Apps*, tal y como se puede observar en la Figura 5.36. De esta forma, la aplicación creada anteriormente se encentra disponible en el sistema Linux que se ejecutará en la tarjeta ZedBoard.

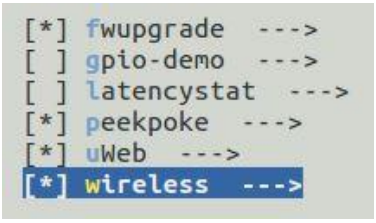

Figura 5.36: Activación de la aplicación "wireless" en PetaLinux SDK

Para poder hacer uso de la herramienta *Secure Shell* o SSH, se debe activar la aplicación *dropbear*. Esta aplicación incluye un servidor SSH que emplea pocos recursos de procesador y tiene un tamaño reducido lo que la hace ideal para sistemas embebidos.

La aplicación *dropbear* se encuentra en el menú *Filesystem Packages* -> *Console/network*. Se deben activar las opciones *dropbear* y *dropbear-openssh-sftp-server*.

Una vez realizados las dos modificaciones, se selecciona *Save*, *Ok* y, finalmente, *Exit* para guardar los cambios efectuados. Para salir de la ventana de configuración, se selecciona *Exit*.

Se debe añadir el archivo *wl18xx-conf.bin* en la siguiente ruta de directorios *fs/lib/firmware/ti-connectivity*. Este archivo se encuentra con el nombre *wl18xx-conf-default.bin* en la carpeta *build-utilites* en la ruta *build-utilites/src/ti_utils/wlconf*. Se debe copiar dicho archivo y pegarlo en la ruta anteriormente indicada. Posteriormente, se debe modificar el nombre del archivo a *wl18xx-conf.bin*.

En el archivo *WL1835MOD_INI.ini* se debe comentar, poniendo *#*, todas las líneas en blanco que contiene el archivo ya que estas producen errores. Para ello, se emplea el comando:

- sudo gedit <directorio del proyecto PetaLinux> /components/apps/wireless/fs/usr/sbin/wlconf/official_inis/WL1835MOD_INI.ini

Una vez que se ha configurado completamente PetaLinux, se debe recompilar el kernel con la instrucción:

- petalinux-build

Una vez que la instrucción anterior ha terminado de ejecutarse, en la carpeta *images* se encuentran los archivos que se van a necesitar para ejecutar Linux en la placa ZedBoard.

5.4 Modificación de los dispositivos HW accesibles: Device Tree

En la carpeta *images* que se acaba de crear, existe un archivo llamado *system.dtb*. Este es el archivo binario del *device tree* que se ha creado por defecto al compilar el kernel de Linux. El *device tree* identifica los diferentes elementos que conforman el hardware. Por ello, se debe añadir el nuevo hardware que se va a usar a este archivo.

El primer paso que se debe dar para poder modificar el *device tree* binary es convertirlo a texto. Para ello, se debe ejecutar el siguiente comando en un terminal de Ubuntu:

- dtc –I dtb –O dts –o system.dts system.dtb

Esta instrucción generará un nuevo archivo llamado *system.dts* que es el que se debe modificar. Para ello, se debe abrir este archivo y modificar lo siguiente:

- Las líneas correspondientes a la SD 1 deben modificarse de la siguiente forma, indicando la frecuencia de trabajo, el *driver* que se debe emplear y el dispositivo conectado:

```
sdhci@e0101000
{
        clock-frequency = <0x989680>;
        compatible = "arasan,sdhci-8.9a";
        clock-names = "clk_xin", "clk_ahb";
        clocks = <0x1 0x16 0x1 0x21>;
        interrupt-parent = <0x3>;
        interrupts = <0x0 0x2f 0x4>;
        reg = <0xe0101000 0x1000>;
        bus-width = <0x4>;
        ti,non-removable;
        ti,needs-special-hs-handling;
        cap-power-off-card;
        keep-power-in-suspend;
        vmmc-supply = <0x6>;
        #address-cells = <0x1>;
        #size-cells = <0x0>;
        wlcore@0
        {
                compatible = "ti,wl1835";
                interrupt-parent = <0x5>;
                interrupts = <0x0 0x8>;
                reg = <0x2>;
                platform-quirks = <0x1>;
                board-ref-clock = <0x4>;
        };
};
```

- Añadir en gpio@e000a000

```
#interrupt-cells = <0x2>;
linux,phandle = <0x5>;
phandle = <0x5>;
```

- Añadir fixedregulator@1

```
fixedregulator@1
{
        compatible = "regulator-fixed";
        regulator-name = "wlan-en-regulator";
        regulator-min-microvolt = <0x325aa0>;
        regulator-max-microvolt = <0x325aa0>;
        gpio = <0x5 0x9 0x4>;
        startup-active-us = <0x11170>;
        enable-active-high;
        linux,phandle = <0x6>;
        phandle = <0x6>;
};
```

Una vez realizados los cambios anteriores, se debe guardar el archivo para que los cambios queden modificados. A continuación, se debe regenerar el *device tree* binario introduciendo el siguiente comando en un terminal de Ubuntu:

- dtc –I dts –O dtb –o system.dtb system.dts

Tras la ejecución de la instrucción anterior, se ha generado el archivo *system.dtb* que es otro de los archivos que se deben ubicar en la tarjeta SD para poder ejecutar un sistema operativo sobre la tarjeta ZedBoard.

5.5 Generación archivo BOOT.bin

Se debe generar el archivo llamado *BOOT.bin* que está formado por otros tres archivos que ya se han generado.

Dos de estos archivos fueron obtenidos por Vivado 2014.4 y Xilinx SDK y se encuentran en las carpetas de dichos programas:

- zynq_fsbl_0.elf: se encarga de configurar el sistema hardware programando la PL con el archivo *BitStream* y prepara al sistema para cargar códigos de arranque más complejos. Se encuentra en la ruta de direcciones: <directorio del proyecto Vivado>/sdk/zynq_fsbl_0/Debug/zynq_fsbl_0.elf
- design_1_wrapper.bit: Es el archivo de descripción de hardware creado por Vivado 2014.4. Se encuentra en la ruta de directorios: <directorio del proyecto Vivado>/sdk/design_1_wrapper_hw_platform_0/design_1_wrapper.bit

El último de los archivos que conforma el *BOOT.bin* es el archivo *u-boot.elf* que ha sido generado por PetaLinux SDK. El archivo se encuentra en la carpeta *images* dentro de la carpeta del proyecto de PetaLinux. Este archivo se encuentra en Ubuntu y la herramienta que se va a emplear para generar el *BOOT.bin* se encuentra instalada en Windows 10 por lo que es necesario copiar el archivo *u-boot.elf* en un directorio de este último sistema operativo.

Para la creación del archivo *BOOT.bin* se emplea una herramienta integrada en Xilinx SDK llamada *Create Zynq Boot Image*. Para ejecutar la herramienta, se pulsa en *Xilinx Tools* y, a continuación, en *Create Zynq Boot Image*. Esta herramienta puede observarse en la Figura 5.37.

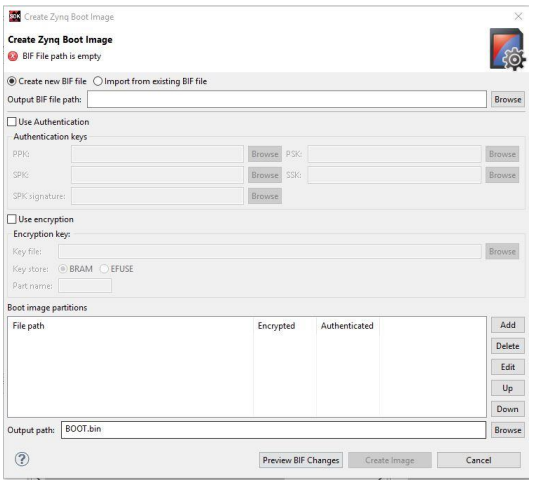

Figura 5.37: Aplicación "Create Zynq Boot Image"

Se selecciona la opción *Create new BIF file* y se pulsa sobre *Browse*. En la ventana que se abre, se elige el directorio donde se va a guardar el archivo .bif y su nombre.

Este archivo es el que emplea la herramienta para generar el archivo *BOOT.bin* y permite modificar de una forma más rápida dicho archivo ya que, si se necesita generar un nuevo *BOOT.bin* modificando alguno de los archivos que lo forman, se puede importar el archivo .bif y en la ventana *Boot image partitions* se indicarán los archivos que forman el archivo.

En la zona inferior de la ventana se deben indicar los archivos que van a componer el archivo *BOOT.bin*. Los archivos deben añadirse en el orden indicado a continuación:

- Se pulsa en *Add* y se abre una ventana llamada *Add new boot image partition*. El primer archivo que se debe añadir es el llamado *zynq_fsbl_0.elf*. Además, en el campo *Partition type* se debe seleccionar la opción *bootloader* ya que este es el primer archivo necesario en el arranque del sistema. Por último, se pulsa *Ok*.
- Una vez añadido el primer archivo, se añade el siguiente pulsando de nuevo en *Add*. En la ventana *Add new boot image partition* se selecciona *Browse* y, en esta ocasión, se añade el archivo *design_1_wrapper.bit*. En este caso, en el campo *Partition type* se selecciona la opción *datafile* y se pulsa *Ok*.
- Por último, se vuelve a pulsar *Add* y en la ventana que se abre se selecciona *Browse* y se añade el archivo *u-boot.elf*. En este caso también se debe seleccionar en *Partition type* la opción *datafile* y se pulsa en *Ok*.

El proceso que se debe seguir según los pasos anteriores se puede ver en la Figura 5.38.

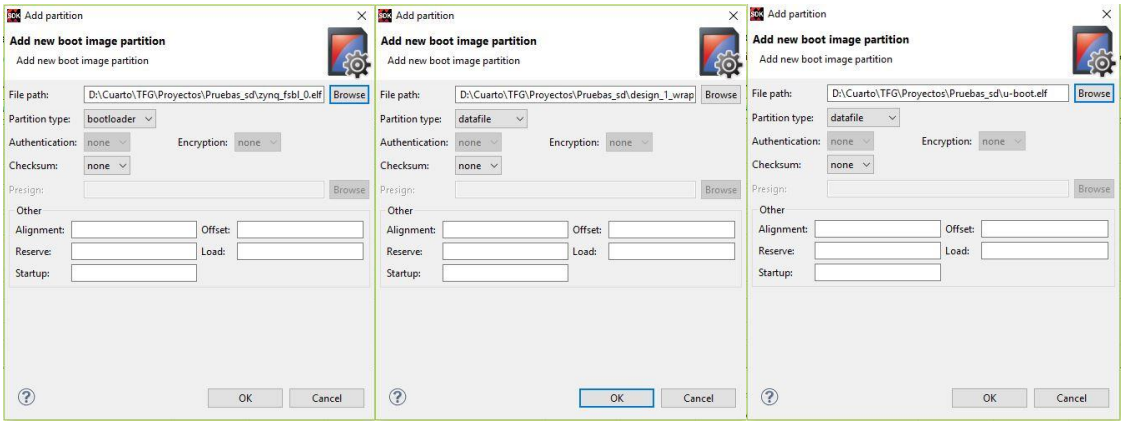

Figura 5.38: Proceso de adición de los tres archivos que conforman el archivo BOOT.bin

Una vez añadidos todos los archivos anteriores, en el campo *Output path* se debe seleccionar la ruta donde se generará el archivo *BOOT.bin*. Tras realizar los pasos anteriores, el aspecto de la herramienta debe ser similar al mostrado en la Figura 5.39. Por último, se selecciona la opción *Create Image*. Tras terminar de ejecutarse, en el directorio elegido se creará el archivo *BOOT.bin*.

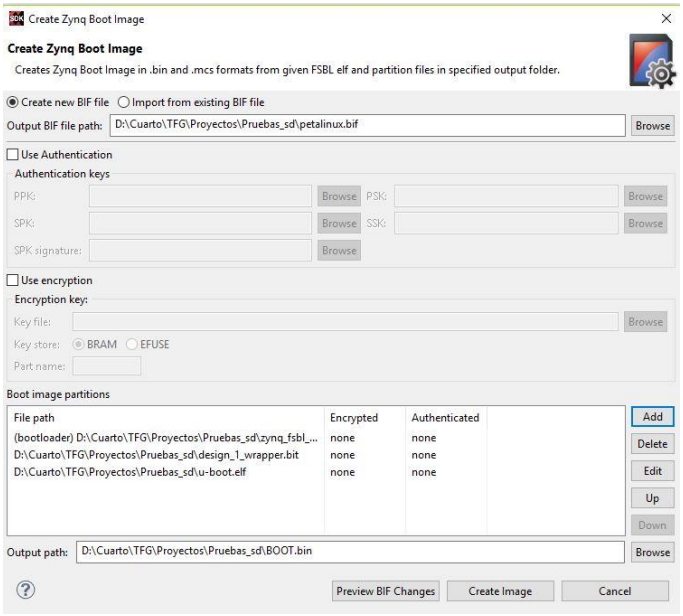

Figura 5.39: Archivos añadidos en la aplicación "Create Zynq Boot Image"

5.6 Preparación de la tarjeta SD

Todos los archivos generados en apartados anteriores deben ubicarse en una tarjeta SD desde donde serán leídos y ejecutados. Antes de poder ubicar dichos archivos, la tarjeta debe tener un formato que pueda ser leído posteriormente por la placa de desarrollo. Por tanto, el proceso que se va a seguir en este apartado se puede observar en la Figura 5.40.

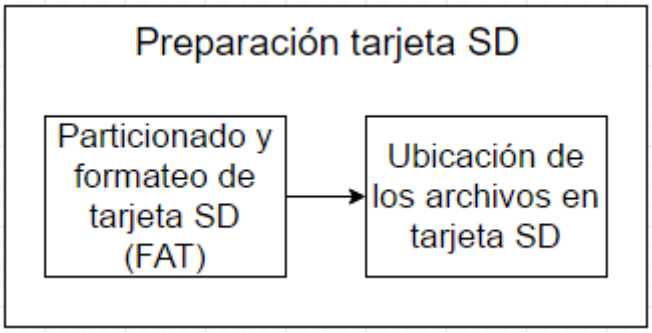

Figura 5.40: Preparación tarjeta SD

5.6.1 Particionado de la tarjeta SD

Para poder ubicar los archivos generados anteriormente, se debe dar formato a la tarjeta SD que se empleará para arrancar el sistema operativo en la tarjeta ZedBoard.

Para ello, se pueden usar distintos programas como por ejemplo GParted [28]. La instalación de este programa puede realizarse ejecutando en un terminal el siguiente comando:

- sudo apt-get install gparted

Al ejecutar el programa GParted se debe introducir la contraseña de administrador ya que, con este programa, se pueden producir importantes cambios en el sistema operativo. Una vez que la herramienta ha arrancado se debe seleccionar la tarjeta SD en la parte superior derecha. Esto

muestra las diferentes particiones y formatos que actualmente tiene la tarjeta que se desea formatear.

Como primer paso, se deben desmontar todas las particiones que existan. Para ello, se selecciona cada partición y con el botón derecho se selecciona *Desmontar*. A continuación, se debe borrar las particiones y crear una nueva.

Una de ellas debe tener un tamaño de aproximadamente 1GB y tener un formato FAT32. A esta partición se le asignará el nombre "*BOOT*".

Una vez que se han seleccionado todas las características anteriores, se debe pulsar sobre el *tick* que se encuentra en la parte superior. A continuación, se debe aceptar el aviso de que la operación no se puede deshacer. Cuando el proceso termina, la tarjeta SD tendrá el formato adecuado para ubicar los archivos necesarios para el correcto arranque del sistema operativo. En la Figura 5.41 se muestra el formato final de la tarjeta SD que se va a emplear para almacenar los archivos necesarios para el arranque del sistema operativo en ZedBoard.

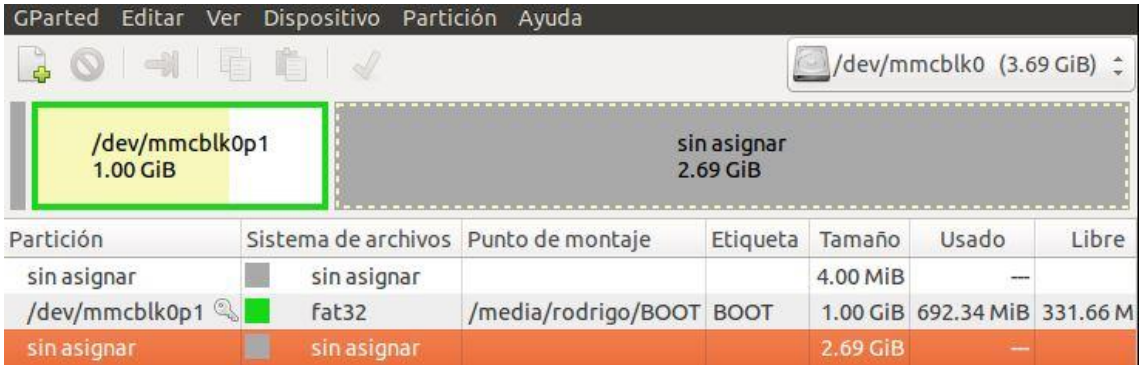

Figura 5.41: Partición FAT creada con aplicación GParted

5.6.2 Localización de archivos en la tarjeta SD

En la partición creada anteriormente denominada *BOOT* se deben ubicar los siguientes archivos copiándolos de los directorios donde han sido generados:

- BOOT.bin
- image.ub
- system.dtb

5.7 Arranque y funcionamiento del SO Linux en ZedBoard

Una vez que todos los archivos necesarios se encuentran guardados en la tarjeta SD, se puede proceder al arranque del sistema operativo en ZedBoard. En la Figura 5.42 se pueden observar todas las conexiones necesarias para poder llevar a cabo el arranque del sistema.

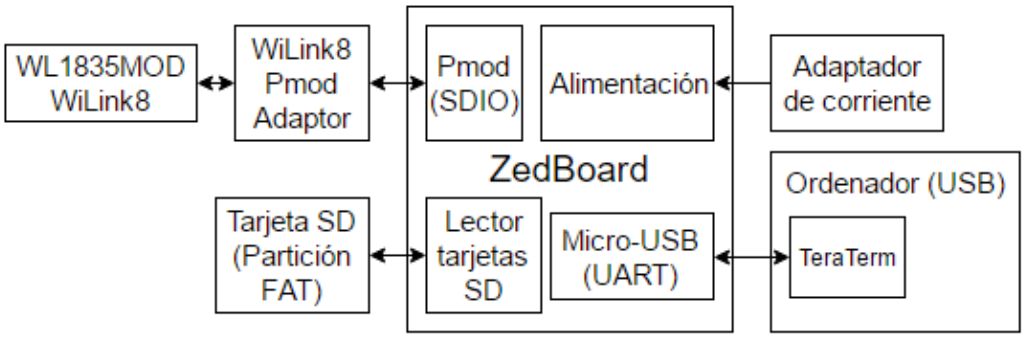

Figura 5.42: Conexiones necesarias para el arranque del sistema

Primero se debe conectar el adaptador de corriente a la ZedBoard y un cable micro-usb a la conexión UART que se usará para la comunicación con el sistema operativo. Esta comunicación puede realizarse con cualquiera de los sistemas operativos que se están usando. En este caso se empleará el sistema operativo Windows pero se podría emplear Ubuntu con, por ejemplo, el programa minicom.

Para llevar a cabo la comunicación, se debe emplear un programa que permite la comunicación serie como por ejemplo, TeraTerm [29]. Este programa se debe configurar para funcionar con el puerto serie que se está usando y una velocidad de 115000 baudios. Para configurar esta velocidad, se debe acceder a la opción *Serial Port…* que se encuentra en el menú *Setup* ubicado en la parte superior de la ventana. En la opción *Baud rate* se debe seleccionar 115000 baudios dejando las demás opciones en su valor por defecto.

También se deben conectar el módulo inalámbrico, a través del adaptador, a los puertos Pmod de la tarjeta ZedBoard.

Por último, se debe introducir la tarjeta SD en la ranura de lectura de SD disponible en la placa. Para poder trabajar con la tarjeta SD en la placa ZedBoard es necesario que el jumper JP6 se encuentre cerrado. Además, para que el arranque se produzca desde los archivos ubicados en la tarjeta SD, se deben colocar los *jumpers* JP8 a GND y JP9 y JP10 a 3,3V tal y como se muestra en la Figura 5.43.

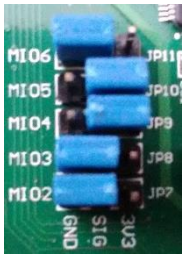

Figura 5.43: Posición de los jumpers en ZedBoard

Una vez realizadas todas las conexiones anteriores, se puede proceder a encender la tarjeta ZedBoard con el interruptor POWER.

Tras esto, comienza el proceso de arranque de Linux. Este proceso se puede seguir a través de la información que se muestra a través de la UART.

La primera información que se muestra indica la elección de la interfaz serie para el envío y recepción de información en el proceso de arranque. El siguiente paso es la obtención de los datos relacionados con la tarjeta SD como su velocidad de transferencia, su versión y su capacidad. Finalmente, se leen dos de los archivos que esta contiene: *image.ub* y *system.dtb*.

```
U-Boot 2014.07
DRAM:  ECC disabled 512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total
32 MiB
In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
Device: zynq_sdhci
Manufacturer ID: 2
OEM: 544d
Name: SD04G
Tran Speed: 50000000
Rd Block Len: 512
SD version 2.0
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 4-bit
reading image.ub
3472088 bytes read in 306 ms (10.8 MiB/s)
(…)
reading system.dtb
10254 bytes read in 16 ms (625 KiB/s)
```

Una vez leídos los archivos anteriores y según su contenido, el siguiente paso es la configuración y localización del sistema en memoria, comprobando la integridad de los datos. En este momento, se descomprime el *kernel* si este ha sido comprimido en el momento de su generación en PetaLinux SDK.

```
## Loading kernel from FIT Image at 04000000 ...
   Using 'conf@1' configuration
   Trying 'kernel@1' kernel subimage
     Description:  PetaLinux Kernel
     Type:         Kernel Image
     Compression:  gzip compressed
     Data Start:   0x040000f0
     Data Size:    3457535 Bytes = 3.3 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x00008000
     Entry Point:  0x00008000
     Hash algo:    crc32
     Hash value:   df6fda78
   Verifying Hash Integrity ... crc32+ OK
## Flattened Device Tree blob at 0a800000
   Booting using the fdt blob at 0xa800000
   Uncompressing Kernel Image ... OK
   Loading Device Tree to 07ffa000, end 07fff80d ... OK
```

A partir de este momento, el *kernel* del sistema operativo comienza su arranque comprobando el correcto funcionamiento de todos los sistemas que va a emplear. El momento en el que se comprueba la existencia de las dos interfaces SDIO empleadas se muestra en el siguiente fragmento del arranque. Las líneas precedidas por *mmc0* se refieren a la propia tarjeta SD donde

se encuentran todos los elementos que conforman el sistema. Por el contrario, las líneas que comienzan por *mmc1* están referidas al módulo WiFi conectado por medio de la interfaz SDIO1.

```
mmc0: new high speed SDHC card at address 57b8
mmcblk0: mmc0:57b8 SD04G 3.69 GiB
 mmcblk0: p1 p2
sdhci-arasan e0101000.sdhci: card claims to support voltages below defined range
mmc1: queuing unknown CIS tuple 0x91 (3 bytes)
mmc1: new high speed SDIO card at address 0001
```

El siguiente paso dado por el sistema es la carga e inicialización de todos los *drivers* que este necesita. En el siguiente fragmento de código se pueden observar los controladores necesarios para el correcto funcionamiento del módulo WiLink8 empleado en este trabajo.

```
Loading modules backported from Linux version R8.7_SP1-0-g13c25bc
Backport generated by backports.git R8.7_SP1-0-gd4777ef
(…)
wlcore: wl18xx HW: 183x or 180x, PG 2.2 (ROM 0x11)
wlcore: loaded
wlcore: driver version: R8.7_SP1
```

Por último, se ejecutan los programas configurados para su arranque en el momento del encendido del sistema. En el siguiente fragmento se puede observar el arranque de la aplicación *dropbear* que permite el empleo de SSH.

```
Starting Dropbear SSH server: Generating key, this may take a while...
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQ(…)+tkxhSKG1QQXeJLwO7YCRzEaCtF root@wifi_tfg_2
Fingerprint: md5 0c:3c:70:4c:30:a9:02:4c:36:bc:cf:b0:37:59:db:99
NET: Registered protocol family 10
dropbear.
```

Una vez que ha terminado de arrancar el sistema operativo, se pide introducir el usuario (“root”) y la contraseña (“root”). Tras introducir los datos anteriores, el sistema operativo ha terminado de arrancar y se encuentra a la espera de recibir comando por parte del usuario.

```
Built with PetaLinux v2014.4 (Yocto 1.7) wifi_tfg_2 /dev/ttyPS0
wifi_tfg_2 login: root
Password:
login[803]: root login on 'ttyPS0'
root@wifi_tfg_2:~#
```

5.8 Verificación del funcionamiento del módulo WiFi

Para verificar el correcto funcionamiento del módulo WiFi, lo primero que se debe comprobar es la correcta instalación de los módulos que permiten disponer de la interfaz WiFi en Linux. Una vez que la interfaz se encuentra disponible, se comprobará su conectividad estableciendo conexión con una red WiFi disponible.

5.8.1 Comprobación interfaz WiFi

Al iniciarse el sistema operativo, se deben cargar los módulos necesarios para el correcto funcionamiento del hardware WiFi. Los módulos requeridos son los indicados anteriormente en apartados previos: *compat, cfg80211, mac80211, wlcore, wlcore_sdio y wl18xx*. Para saber que módulos se han cargado en un sistema Linux se ejecuta el comando *lsmod*. El resultado obtenido debe ser semejante al que se puede observar en la Figura 5.44.

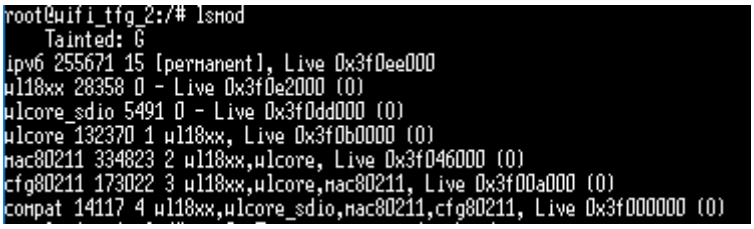

Figura 5.44: Módulos cargados en Linux

Para comprobar que todas las bibliotecas se han instalado correctamente, se ejecuta el comando siguiente:

- ifconfig -a

Esta instrucción muestra los diferentes interfaces disponibles en el sistema operativo que se está ejecutando. Se deben mostrar 3 interfaces: eth0, lo y wlan0, tal y como se muestra en la Figura 5.45. Esta última interfaz es la que se emplea para realizar la conexión inalámbrica. Para cada interfaz se muestra diferente información como el número de paquetes transmitidos, recibidos y erróneos. Además, aquí también se muestra la dirección MAC y IP, si ésta ha sido asignada.

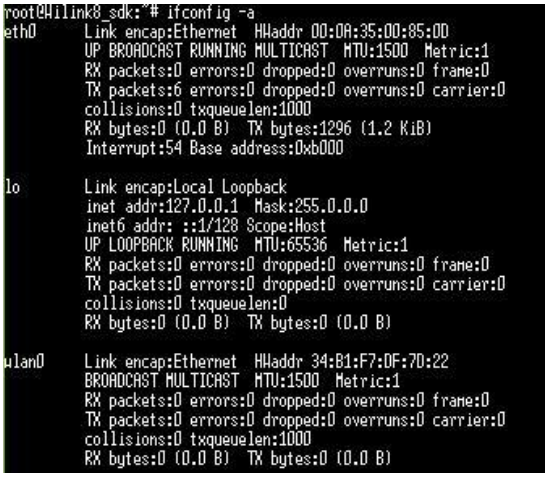

Figura 5.45: Interfaces de comunicación en Linux

5.8.2 Conexión a una red inalámbrica WiFi desde la Zedboard

En este apartado se explica cómo conocer las redes WiFi que se encuentran disponibles y como establecer conexión con alguna de ellas.

5.8.2.1 Escaneado de las redes WiFi disponibles

Para poder realizar la búsqueda de las redes WiFi disponibles, es necesario activar la interfaz. Para ello se debe ejecutar el siguiente comando:

- ifconfig wlan0 up

Al ejecutar la instrucción anterior se debe obtener una respuesta similar a la que se puede observar en la Figura 5.46.

Figura 5.46: Respuesta activación red inalámbrica

Para mostrar las redes WiFi disponibles, se debe ejecutar la instrucción:

- iw wlan0 scan

De esta forma, se muestra toda la información disponible de cada red WiFi: nombre de la red, potencia de recepción, tipo de encriptado y otros parámetros de la red. Un ejemplo de esta información se puede observar en la Figura 5.47.

Figura 5.47: Ejemplo de información de red WiFi

Si únicamente se desea conocer el nombre de las redes disponibles, se debe ejecutar el comando:

- iw wlan0 scan | grep SSID

De esta forma, solo se mostrará el nombre de cada red WiFi a las que se puede conectar la tarjeta ZedBoard, como se muestra en la Figura 5.48.

Figura 5.48: Ejemplo de redes WiFi mostradas por SSID

5.8.2.2 Conexión a una red WiFi con contraseña tipo WEP

El procedimiento a seguir para la conexión con una red WiFi con protección WEP es el siguiente:

- wpa_supplicant –d –Dnl80211 –c/etc/wpa_supplicant.conf –iwlan0 –B
- wpa_cli –iwlan0 add_network
- wpa_cli –iwlan0 set_network 0 auth_alg OPEN
- wpa_cli –iwlan0 set_network 0 wep_key0 ‘”<contraseña de red>”’
- wpa_cli –iwlan0 set_network 0 key_mgmt NONE
- wpa_cli –iwlan0 set_network 0 mode 0
- wpa_cli –iwlan0 set_network 0 ssid ‘”<nombre de red>”’
- wpa_cli –iwlan0 select_network 0
- wpa_cli –iwlan0 enable_network 0
- wpa_cli –iwlan0 reassociate
- wpa_cli –iwlan0 status

Un ejemplo de conexión a una red con protección WEP se puede apreciar en la Figura 5.49.

Figura 5.49: Configuración de conexión a una red WEP

Una vez que la conexión se ha llevado a cabo, si la interfaz wireless no tiene asignada una dirección IP, se debe asignar una con la instrucción:

- udhcpc –i wlan0

5.8.2.3 Conexión a una red WiFi con contraseña tipo WPA-PSK

Para conectarse a una red con contraseña del tipo WPA existen dos métodos:

El primer de los métodos para conectarse a una red WiFi con contraseña de tipo WPA consiste únicamente en ejecutar 3 comandos:

- Primero es necesario ubicarse en el directorio donde se encuentran los ejecutables del módulo WiFi. Para ello, se debe cambiar el directorio con el comando:
  cd /usr/share/wl18xx
- Una vez en ese directorio se debe ejecutar:
  ./sta_start.sh
  Si la ejecución se realiza sin errores, la respuesta devuelta se puede observar en la Figura 5.50.

```
root@Hilink8_sdk:/usr/share/wl18xx# ./sta_start.sh
root@Hilink8_sdk:/usr/share/wl18xx# Successfully initialized wpa_supplicant
```

Figura 5.50: Ejecución comando sta_start.sh

- Para llevar a cabo la conexión con la red se ejecuta la siguiente instrucción siguiente cambiando <SSID_red> por el nombre de la red a la que se desea conectar y <contraseña> por la clave de dicha red:
  ./sta_connect-ex.sh <SSID_red> WPA-PSK <contraseña>
  Un ejemplo de conexión a una red WiFi con protección WPA-PSK empleando el script anterior puede observarse en la Figura 5.51.

```
root@Hilink8 sdk:/usr/share/wl18xx# ./sta_connect-ex.sh WLAN_A789 WPA-PSK xxxxxxxxxxxxxxxxxx

netid=1
========================
OK
OK
OK
OK
wlan0: SME: Trying to authenticate with dc:0b:1a:a5:a7:8a (SSID='wlan0: authenticate with dc:0b:1a:a5:a7:8a
WLAN_A789' freq=2412 MHz)
wlan0: send auth to dc:0b:1a:a5:a7:8a (try 1/3)
wlcore: ERROR SW watchdog interrupt received! starting recovery.
wlcore: Hardware recovery in progress. FW ver: Rev 8.9.0.0.70
wlcore: pc: 0x11c32, hint_sts: 0x00000000 count: 1
wlcore: down
wlcore: down
ieee80211 phy0: Hardware restart was requested
wlan0: send auth to dc:0b:1a:a5:a7:8a (try 2/3)
wlcore: using inverted interrupt logic: 8
wlcore: PHY firmware version: Rev 8.2.0.0.237
wlcore: firmware booted (Rev 8.9.0.0.70)
wlan0: authenticated
wlan0: Trying to associate with dc:0b:1a:a5:a7:8a (SSID='WLAN_A789' freq=2412 MHz)
wlan0: associate with dc:0b:1a:a5:a7:8a (try 1/3)
wlan0: RX AssocResp from dc:0b:1a:a5:a7:8a (capab=0x411 status=0 aid=1)
wlan0: associated
IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
wlan0: Associated with dc:0b:1a:a5:a7:8a
wlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
wlan0: WPA: Key negotiation completed with dc:0b:1a:a5:a7:8a [PTK=Culcore: Association completed.
CMP GTK=TKIP]
wlan0: CTRL-EVENT-CONNECTED - Connection to dc:0b:1a:a5:a7:8a completed [id=1 id_str=]
```

Figura 5.51: Ejecución comando sta_connect-ex.sh

- Una vez establecida la conexión, si la interfaz inalámbrica no tiene una dirección IP asignada, se debe asignar una dirección IP que esté disponible en la red local. Para ello, se emplea el protocolo DHCP con el siguiente comando:
  udhcpc –i wlan0
  El comando anterior se encarga de encontrar un servidor DHCP al cual pide una dirección IP de la red. Si el proceso se realiza correctamente, la instrucción devuelve la dirección IP asignada, tal y como puede observarse en la Figura 5.52.

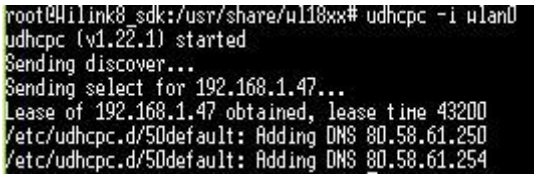

Figura 5.52: Asignación dirección IP por DHCP

El segundo de los métodos es semejante al empleado en el caso de la conexión a una red WiFi WEP. En este caso, el procedimiento a seguir es el siguiente:

- wpa_supplicant –d –Dnl80211 –c/etc/wpa_supplicant.conf –iwlan0 –B
- wpa_cli –iwlan0 add_network
- wpa_cli –iwlan0 set_network 1 auth_alg OPEN
- wpa_cli –iwlan0 set_network 1 key_mgmt WPA-PSK
- wpa_cli –iwlan0 set_network 1 psk ‘”<contraseña de red>”’
- wpa_cli –iwlan0 set_network 1 pairwise CCMP TKIP
- wpa_cli –iwlan0 set_network 1 group CCMP TKIP
- wpa_cli –iwlan0 set_network 1 mode 0
- wpa_cli –iwlan0 set_network 1 ssid ‘”<nombre de red>”’
- wpa_cli –iwlan0 select_network 1
- wpa_cli –iwlan0 enable_network 1
- wpa_cli –iwlan0 reassociate
- wpa_cli –iwlan0 status

Si se opta por emplear el segundo de los métodos de conexión, es necesario indicar todos los parámetros de la conexión, al igual que se hizo en la conexión a una red con protección WEP. Un ejemplo de asociación a una red WiFi indicando todas las características de la red puede observarse en la Figura 5.53.

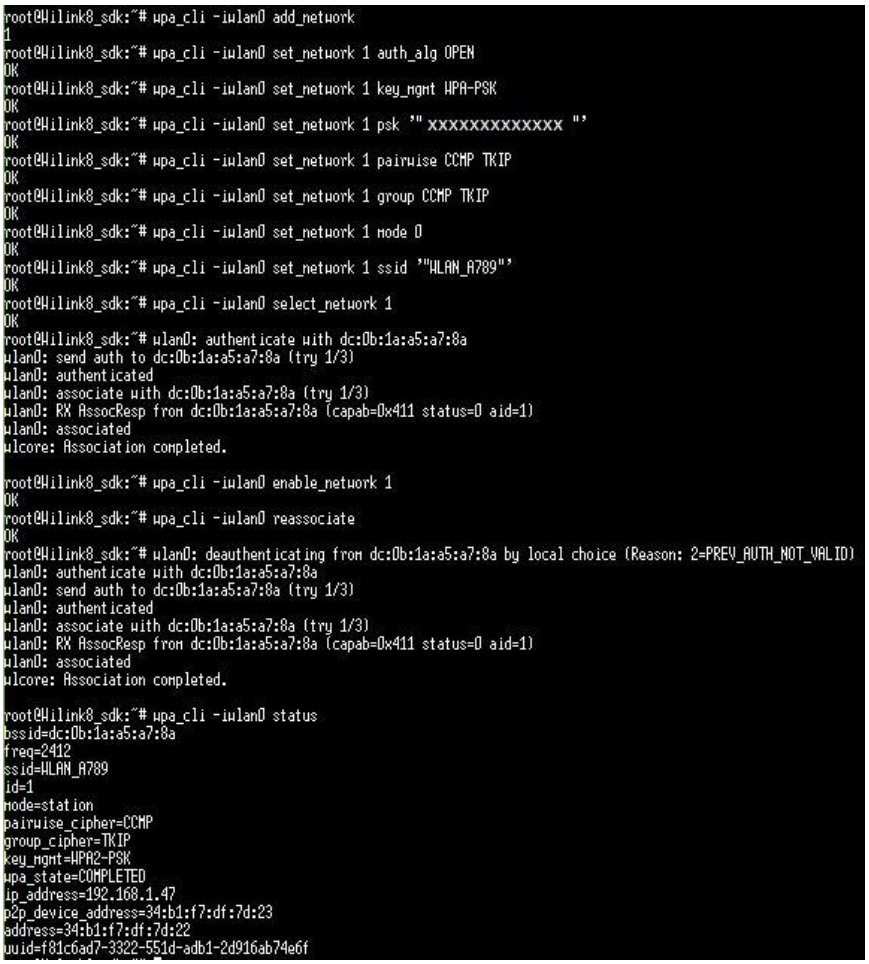

Figura 5.53: Configuración de conexión a una red WPA-PSK

Una vez que se ha establecido la conexión, si la interfaz WiFi no tiene asignada una dirección IP, se debe asignar una con el comando:

- udhcpc –iwlan0

5.8.2.4 Comprobación de la conexión establecida

Para comprobar el estado de la conexión establecida anteriormente se debe ejecutar el comando:

- iw wlan0 link

Este comando devuelve el nombre de la red inalámbrica a la que se está conectado, la frecuencia de funcionamiento, el número de paquetes enviados y recibidos y la potencia de la señal recibida. Un ejemplo de ejecución se puede observar en la Figura 5.54.

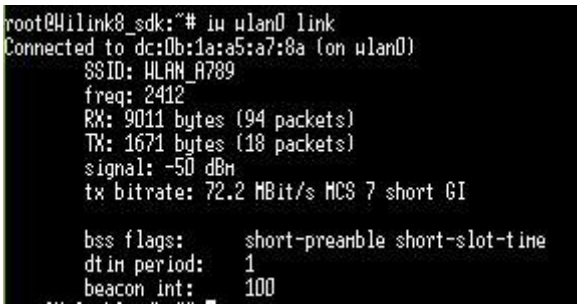

Figura 5.54: Comprobación estado de la conexión

Otro método para comprobar el estado de la conexión es con el empleo del comando "ping". Con esta instrucción se puede comprobar el tiempo que tarda en responder un determinado dispositivo.

Para comprobar el tiempo de respuesta del punto de acceso se debe ejecutar el siguiente comando:

- ping 192.168.1.1

Si la conexión está correctamente establecida, el dispositivo que genera la red WiFi responderá a los *ping* quedando constancia de esas respuestas en los resultados devueltos por el comando. Un ejemplo de *ping* a un dispositivo de la red local puede observarse en la Figura 5.55.

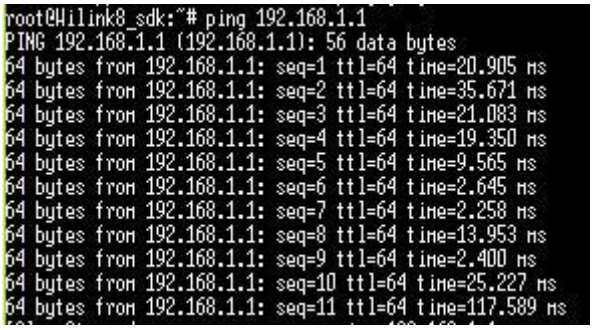

Figura 5.55: Ping a dirección local

Si el punto WiFi tiene conexión a Internet, se puede comprobar si desde la tarjeta ZedBoard se puede acceder a la red. Para ello, se puede ejecutar el comando *ping* con cualquier dirección web, obteniéndose respuesta únicamente si la tarjeta ZedBoard puede acceder a Internet a través del punto de acceso al que está conectado.

- ping google.es

En este caso, el *ping* se realiza a una dirección web cuya dirección IP asociada se obtiene tras consultar al servicio DNS. En este caso, dicha dirección se encuentra fuera de la red local a la que la tarjeta ZedBoard está conectada. Si la placa recibe respuesta del servidor al que se ha realizado el *ping*, se obtendrá un resultado similar al mostrado en la Figura 5.56.

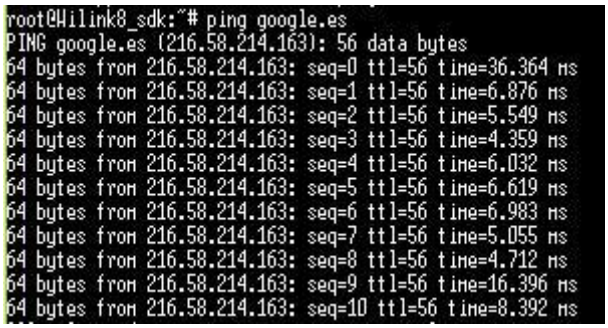

Figura 5.56: Ping a dirección de Internet

5.8.2.5 Otros comandos

Para desconectarse de una red WiFi se debe ejecutar el comando:

- wpa_cli –iwlan0 disconnect

Para mostrar las distintas redes configuradas se ejecuta la instrucción:

- wpa_cli –iwlan0 list_networks

Con el resultado obtenido con esta instrucción, se puede seleccionar una red WiFi configurada anteriormente observando el número asignado a cada configuración. Con dicho número, se puede proceder a establecer la comunicación WiFi ejecutando únicamente los siguientes dos comandos:

- wpa_cli –iwlan0 select_network <número de red>
- wpa_cli –iwlan0 enable_network <número de red>

Por último, se puede borrar la configuración de una red añadida anteriormente con la instrucción:

- wpa_cli –iwlan0 remove_network <SSID_red>

5.9 Acceso al sistema a través de SSH

Gracias a la conectividad inalámbrica que se ha añadido a la tarjeta ZedBoard, se puede tener un control remoto del sistema Linux que se está ejecutando. De esta forma, aunque el sistema se encuentre instalado en un lugar poco accesible, a través de una red WiFi se puede tener un control total del sistema. Con la herramienta SSH [30] se puede ejecutar cualquier comando en el terminal del sistema Linux que se está ejecutando en la placa ZedBoard desde otro dispositivo que se encuentre conectado a la misma red WiFi de una forma sencilla y segura.

El servidor SSH arranca automáticamente al encender el sistema por lo que únicamente es necesario que éste se encuentre conectado a una red para que se pueda acceder a él de forma remota. La información que se muestra por el terminal se puede observar en la Figura 5.57. Cabe destacar que se genera y se muestra una clave pública que se puede emplear para la conexión remota SSH sin necesidad de emplear una contraseña. Esta clave es útil para trabajar con *scripts* que se conectan a sistemas remotos de forma automática.

Figura 5.57: Ejecución en el arranque del servidor SSH

Una vez que la tarjeta ZedBoard está correctamente conectada a una red WiFi, únicamente hace falta conectarse a su servidor SSH para poder controlar de forma remota el sistema.

Para ello, es necesario disponer de un dispositivo que tenga capacidad de conectarse a la misma red local a la que se encuentre conectada la placa ZedBoard. Además, este sistema debe ser capaz de ejecutar un cliente SSH. En este trabajo, se va a emplear el sistema operativo Ubuntu que se ha empleado anteriormente para generar el sistema Linux que se está ejecutando en la placa de desarrollo.

El cliente SSH viene instalado por defecto en este sistema operativo por lo que, en este caso, no es necesario instalar ningún *software* adicional.

En el terminal del equipo que se desea emplear para controlar el sistema que está corriendo en ZedBoard, se debe ejecutar el siguiente comando:

- ssh root@<dirección IP asignada a la tarjeta ZedBoard>

Cuando se ejecuta por primera vez el comando anterior, se notifica este hecho teniendo que indicar *yes* para poder llevar a cabo la sesión SSH.

Por último, se debe indicar la contraseña del sistema que se está ejecutando en ZedBoard. En el caso de este trabajo, la contraseña es "*root*". Una vez indicada la contraseña correcta, el terminal Ubuntu indica el nombre del equipo al que se está conectado. A partir de este momento, se tiene control remoto del sistema operativo que se está ejecutando en ZedBoard.

Un ejemplo de primera conexión a un servidor SSH puede observarse en la Figura 5.58.

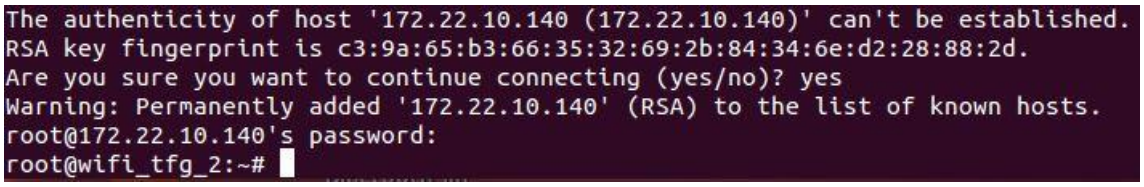

Figura 5.58: Establecimiento sesión SSH

De esta forma, se puede ejecutar cualquier comando como si se estuviera introduciendo directamente a través del interfaz UART como se había hecho anteriormente. A modo de ejemplo, en la Figura 5.59 se muestra lo que devuelve la ejecución del comando *ifconfig*:

```
root@wifi_tfg_2:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:85:0D
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:129 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:43362 (42.3 KiB)
          Interrupt:54 Base address:0xb000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 34:B1:F7:DF:7D:22
          inet addr:172.22.10.140  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:77 errors:0 dropped:0 overruns:0 frame:0
          TX packets:75 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17836 (17.4 KiB)  TX bytes:12159 (11.8 KiB)
```

Figura 5.59: Ejecución del comando ifconfig con SSH

Una vez se han ejecutado todos los comandos deseados, se debe cerrar la conexión SSH con el comando *exit*, obteniéndose una respuesta similar a la mostrada en la Figura 5.60.

```
Connection to 172.22.10.140 closed.
```

Figura 5.60: Cierre de sesión SSH

Capítulo 6: Aplicación E/S remota en plataforma reconfigurable

Como ejemplo de una aplicación real en la que se haga uso del módulo WiFi en una plataforma de hardware reconfigurable, se pretende diseñar un sistema de control remoto de algunos de los elementos disponibles en la tarjeta ZedBoard.

Concretamente, en este capítulo se indicarán todos los pasos que se deben dar para poder encender y apagar los LEDs LD0 a LD7 y leer el estado de los interruptores que se encuentran en la parte PL de la tarjeta ZedBoard, así como el LED MIO7 conectado a la parte PS. La aplicación web mostrará si dichos LEDs se encuentran apagados o encendidos y el estado de los interruptores. Para poder llevar a cabo el control de los distintos elementos, se empleará un servidor web implementado en la ZedBoard al que se obtendrá acceso a través de una red local inalámbrica generada por el módulo WiLink8.

Un esquema de la aplicación que se va a desarrollar en este capítulo puede verse en la Figura 6.1.

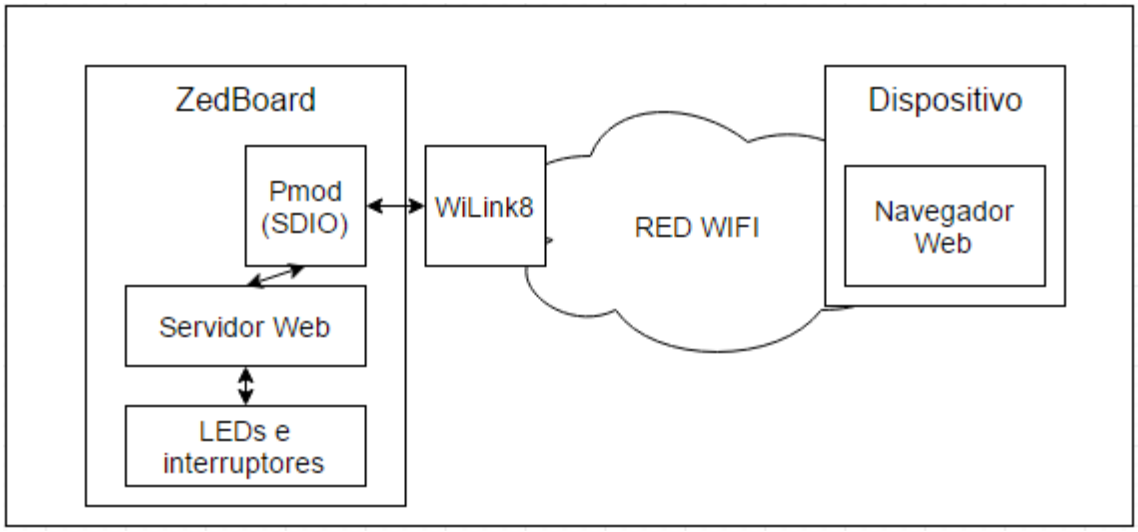

Figura 6.1: Esquema de aplicación para el control de elementos de la tarjeta ZedBoard

De esta forma, un usuario conectado a la red WiFi generada por el módulo WiLink8 podrá acceder al servidor web desde el cual se descargará y visualizará, por medio de un navegador web, una página desde donde podrá consultar y cambiar el estado de los LEDs que están presentes en la tarjeta ZedBoard.

Se deben seguir una serie de pasos para lograr el objetivo propuesto. Primero se creará la parte hardware necesaria con Vivado 2014.4 y, a continuación, la parte software con la herramienta Xilinx SDK. Después, partiendo del kernel de Linux creado en apartados anteriores, se realizarán los cambios necesarios para que las modificaciones que se realicen en dicho sistema queden grabadas y no se pierdan tras apagar el sistema. El siguiente paso será la instalación de un servidor en el cual se ubicarán el archivo HTML y CGI necesario para llevar a cabo la interacción con el sistema. Por último, se mostrará el funcionamiento del sistema completo.

Todos los pasos que se van a seguir en el desarrollo del sistema descrito anteriormente pueden verse resumidos en el diagrama de flujo mostrado en la Figura 6.2.

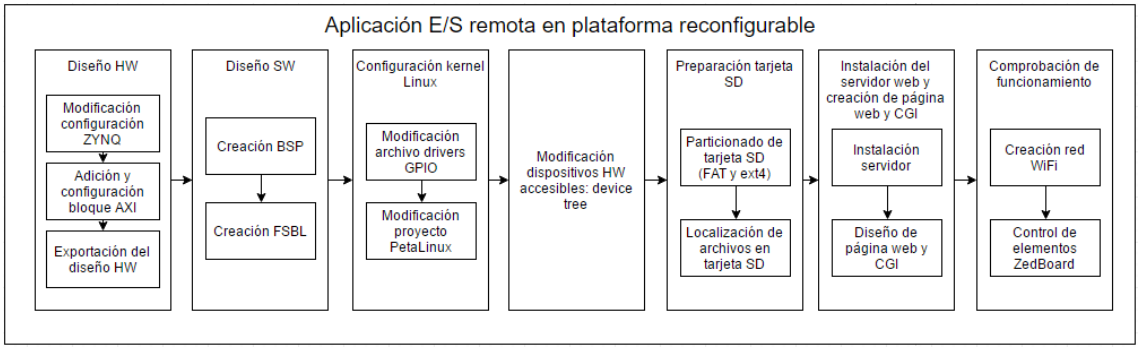

Figura 6.2: Diagrama de flujo de diseño

6.1 Diseño hardware

Para la realización de este apartado se va a partir del diseño hardware realizado anteriormente que se muestra en la Figura 6.3.

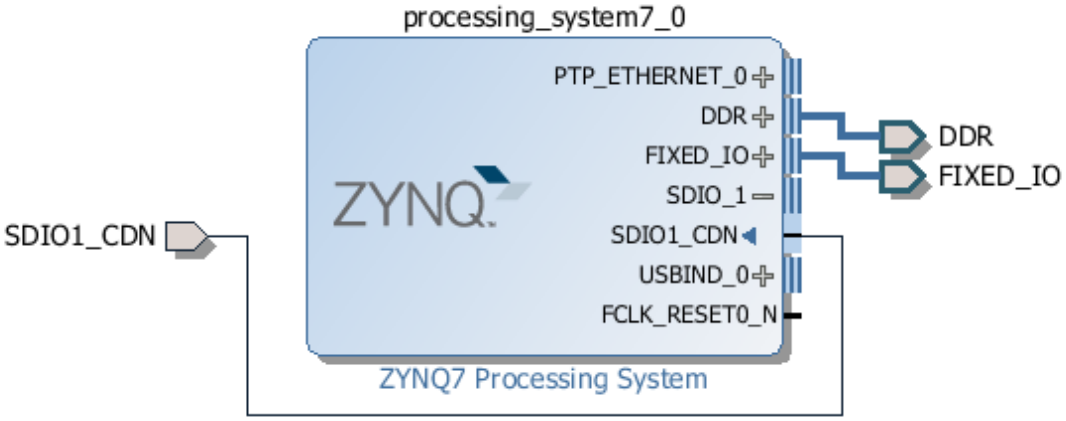

Figura 6.3: Punto de partida HW

6.1.1 Modificación de la configuración de ZYNQ7

Para llevar a cabo la comunicación con los LEDs y los interruptores que se encuentran en la parte PL, se va a emplear una interfaz AXI [31]. Por ello, se deben modificar algunos aspectos del componente *processing_system_0* haciendo doble *click* sobre él.

En la ventana que se abre, se selecciona la opción *PS-PL configuration* que se encuentra en la parte izquierda. De las opciones disponibles, se debe desplegar la opción *GP Master AXI Interface* y, posteriormente seleccionar la opción *M AXI GP0 interface* tal y como se muestra en la Figura 6.4.

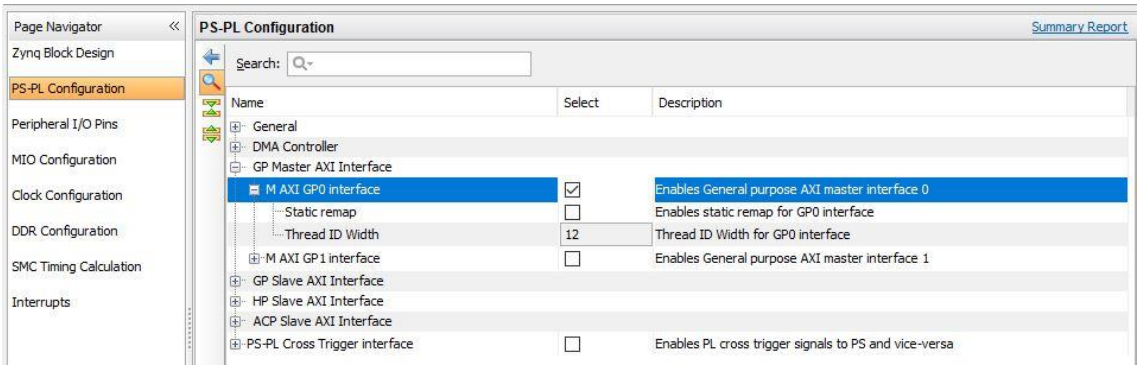

Figura 6.4: Configuración interfaz AXI en ZYNQ7 Processing System

Una vez realizado el cambio anterior, en la parte izquierda de la ventana, se selecciona la opción *Clock Configuration*. De las opciones disponibles, se debe desplegar la opción *PL Fabric Clocks* y se selecciona *FCLK_CLK0* dejando la frecuencia por defecto de 100 MHz. Esta configuración puede observarse en la Figura 6.5.

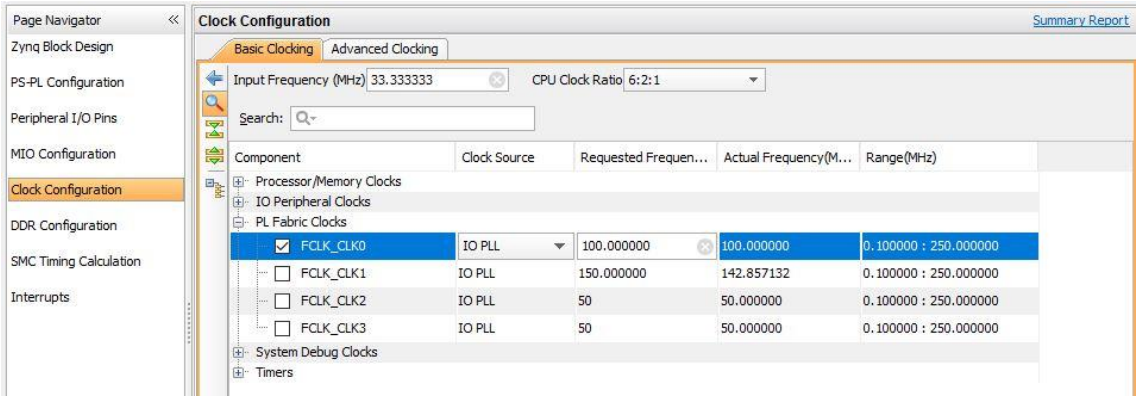

Figura 6.5: Configuración clock en ZYNQ7 Processing System

Una vez se han realizado las modificaciones anteriores, se pulsa *Ok* para que los cambios queden guardados. Tras procesar los cambios realizados, la herramienta Vivado regenera el aspecto del bloque que representa el sistema ZYNQ. El resultado puede observarse en la Figura 6.6.

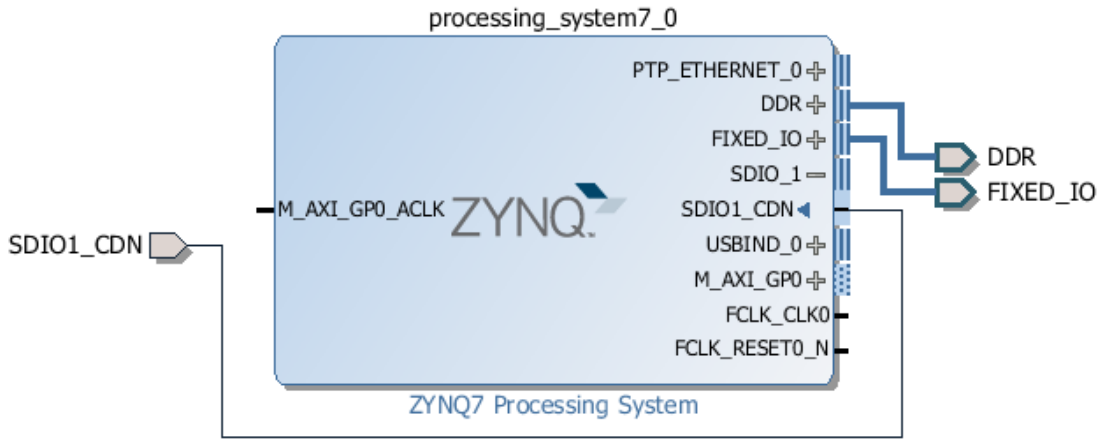

Figura 6.6: Resultado configuración ZYNQ7 Processing System

6.1.2 Adición y configuración bloque IP AXI GPIO

A continuación, se debe añadir un bloque nuevo al diseño. Para ello, se pulsa en *Add IP* y en la ventana que se abre se debe buscar *AXI GPIO*. El resultado de la búsqueda se puede ver en la Figura 6.7. Para añadir el bloque, se debe hacer doble *click* sobre el resultado que se obtiene de la búsqueda.

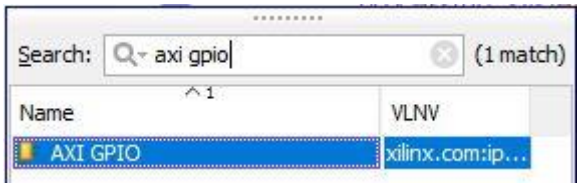

Figura 6.7: Busqueda del bloque AXI GPIO

Una vez el bloque se encuentra añadido al diseño, se hace doble *click* para modificar sus características. En la ventana que se abre, se debe seleccionar *leds 8bits* en la interfaz GPIO y *sws 8bits* en la interfaz GPIO2, según se puede apreciar en la Figura 6.8. Una vez se han realizado los cambios anteriores, se pulsa *Ok* para que guardar los cambios.

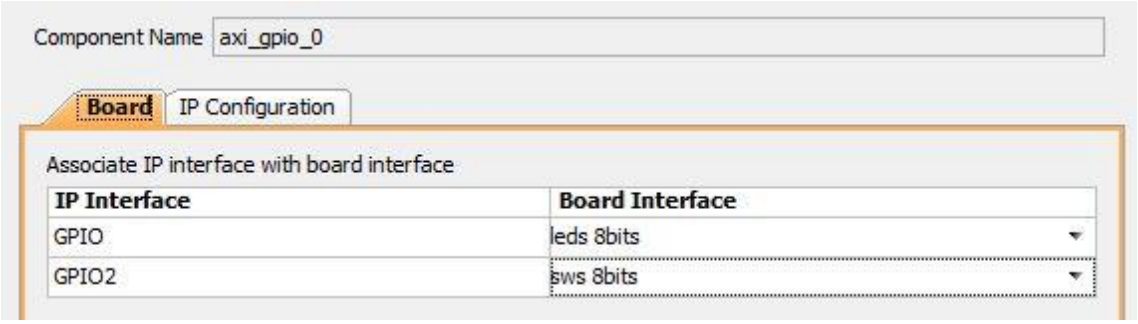

Figura 6.8: Configuración AXI GPIO

Tras añadir el componente anterior, en la parte superior de la ventana de diseño aparece un mensaje que facilita la conexión automática de los bloques. Se hace *click* en *Run Connection Automation* y en la ventana que se abre, mostrada en la Figura 6.9, se deben seleccionar las tres opciones disponibles haciendo *click* sobre *GPIO*, *GPIO2* y *S_AXI*. Al seleccionar la opción *S_AXI*, se debe seleccionar la opción */processing_system7_0/FCLK_CLK0 (100 MHz)* en el campo *Clock connection*. Por último, se selecciona *Ok*.

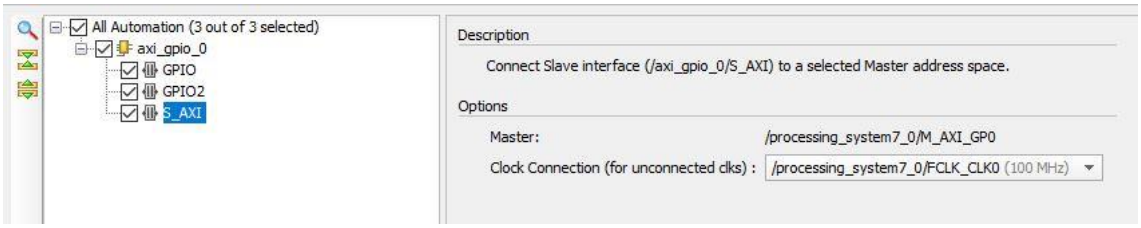

Figura 6.9: Configuración asistente de conexion

El resultado final de la implementación hardware puede observarse en la Figura 6.10.

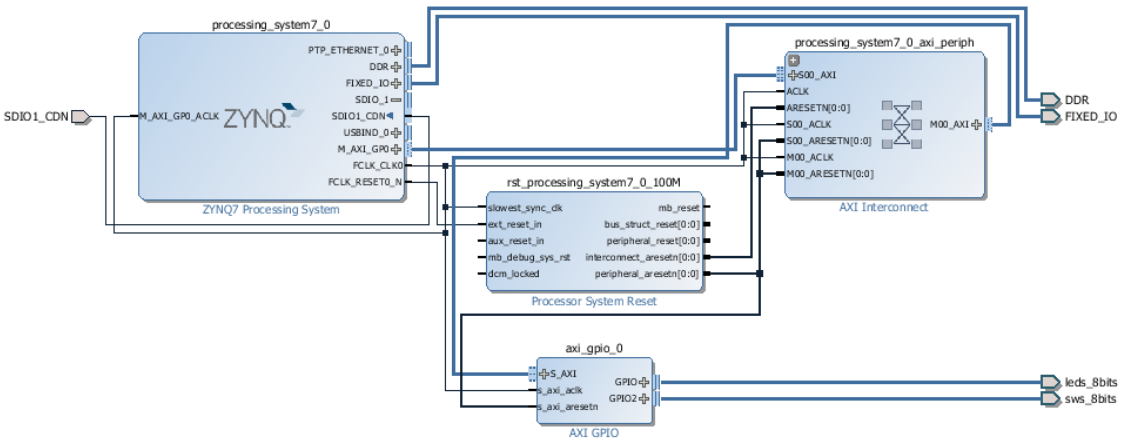

Figura 6.10: Implementación hardware

6.1.3 Generación y exportación de la configuración hardware

Una vez se han realizado todos los cambios anteriores, se guardan los cambios y se selecciona la opción *Generate Bitstream* del menú *Flow Navigator*. En la ventana que se abre, se selecciona *Yes* para realizar la síntesis e implementación antes de la generación del archivo .bit.

Cuando termine todo el proceso de sintetizado, implementación y creación del archivo .bit del diseño elaborado, se mostrará una ventana como la mostrada en la figura siguiente en la cual se seleccionará la opción *Cancel*.

Una vez finalizado el diseño hardware en Vivado 2014.4, es necesario volver a elaborar el archivo FSBL necesario para el correcto arranque del sistema. Para ello, se empleará, al igual que se explicó anteriormente, el software Xilinx SDK.

El primer paso que se debe dar es la exportación del proyecto hardware a Xilinx SDK. Para ello, seleccionamos la opción *Exportar Hardware* que se encuentra en *File -> Export -> Export Hardware*. En la ventana que se abre, se debe seleccionar la opción *Include bitstream* dejando la opción *Export to* por defecto, tal y como se puede observar en la Figura 6.11. A continuación, se pulsará *Ok*.

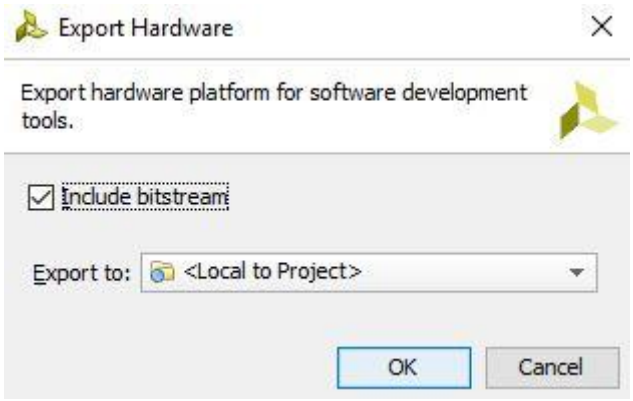

Figura 6.11: Exportación HW a Xilinx SDK

En este caso, como se está empleando el mismo proyecto creado anteriormente, se indica que ya existe un archivo exportado anteriormente, tal y como se muestra en la figura siguiente. Se debe seleccionar la opción *Yes* para que se pueda trabajar sobre el hardware modificado.

6.2 Diseño software

Una vez se ha realizado la exportación del hardware diseñado, se debe ejecutar la herramienta Xilinx SDK. Para ello, se selecciona la opción *Launch SDK* que se encuentra en el menú *File* de Vivado 2014.4.

Cuando el programa haya terminado de arrancar, se mostrará en la ventana *Project Explorer* los archivos generados anteriormente (*design_1_wrapper_hw_platform_0*, *standalone_bsp_0* y *zynq_fsbl_0*) y un nuevo diseño llamado *design_1_wrapper_hw_platform_1*. Para trabajar solo con el nuevo diseño, se deben cerrar los archivos que se habían generado anteriormente. Para ello, se seleccionan con el botón derecho y se elige la opción *Close Project*.

En la Figura 6.12 se pueden observar los archivos antes y después de cerrar los diseños empleados en el capítulo anterior.

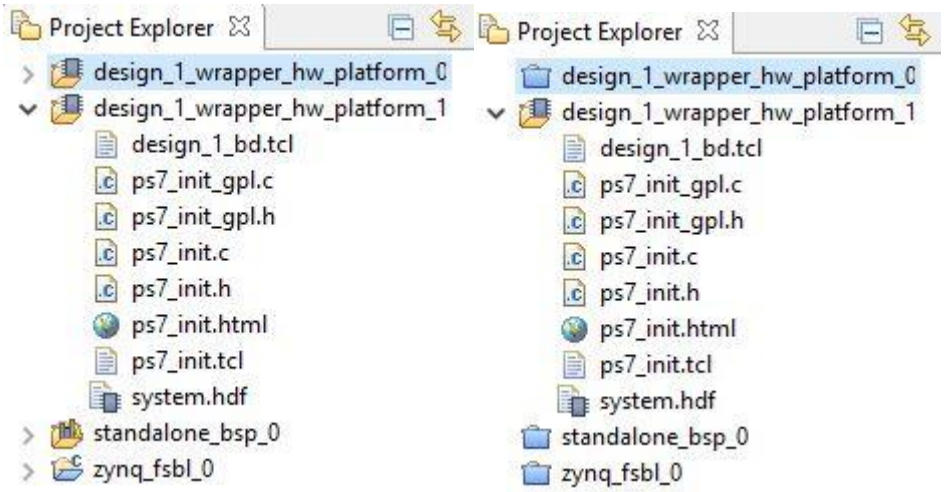

Figura 6.12: Proyecto antes y después de cerrar los archivos antiguos

A partir de este momento se van a seguir los mismos pasos explicados anteriormente en *Creación de diseño software*.

6.2.1 Board Support Package (BSP)

Para la creación del BSP, se elige la opción *File -> New -> Board Support Package*. En la ventana que se abre, mostrada en la parte izquierda de la Figura 6.13, se dejan los valores por defecto y se elige la opción *Finish*. A continuación, en la ventana *Board Support Package Settings*, mostrada en la parte derecha de la Figura 6.13, se debe seleccionar las bibliotecas *xilffs* y *xilrsa* y se pulsa en *Ok*.

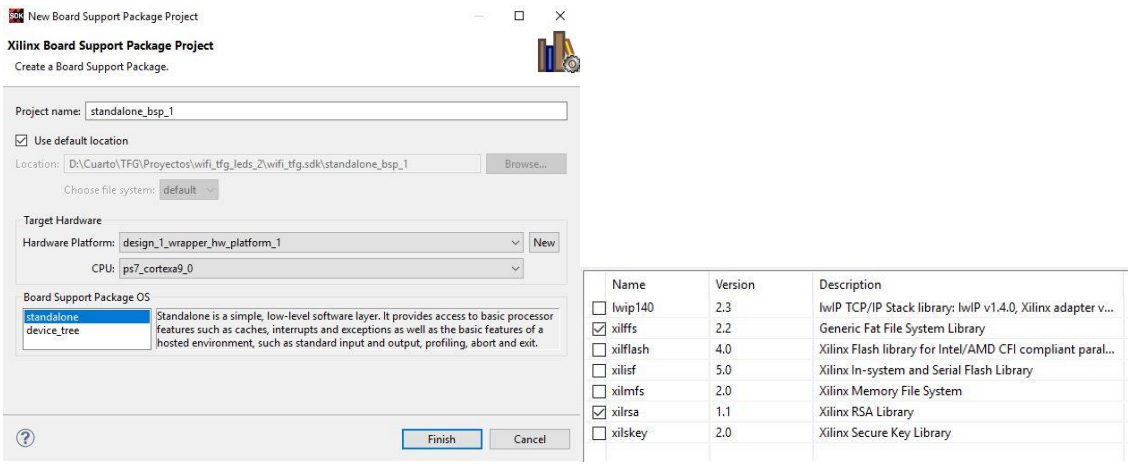

Figura 6.13: Creación BSP y adición de bibliotecas

6.2.2 First Stage BootLoader (FSBL)

Por último, se debe generar el archivo FSBL. Para ello, se selecciona *File -> New -> Application Project*. En la ventana que se abre, se debe elegir un nombre de proyecto, por ejemplo *zynq_fsbl_1*. Además, en *Hardware Platform* se debe elegir *design_1_wrapper_hw_platform_1* para emplear el hardware diseñado en Vivado. En esta misma ventana, en la opción *Board Support Package* se debe seleccionar *Use existing* y *standalone_bsp_1*. Una vez se han cambiado las opciones anteriores, mostradas en la Figura 6.14, se pulsa en la opción *Next*.

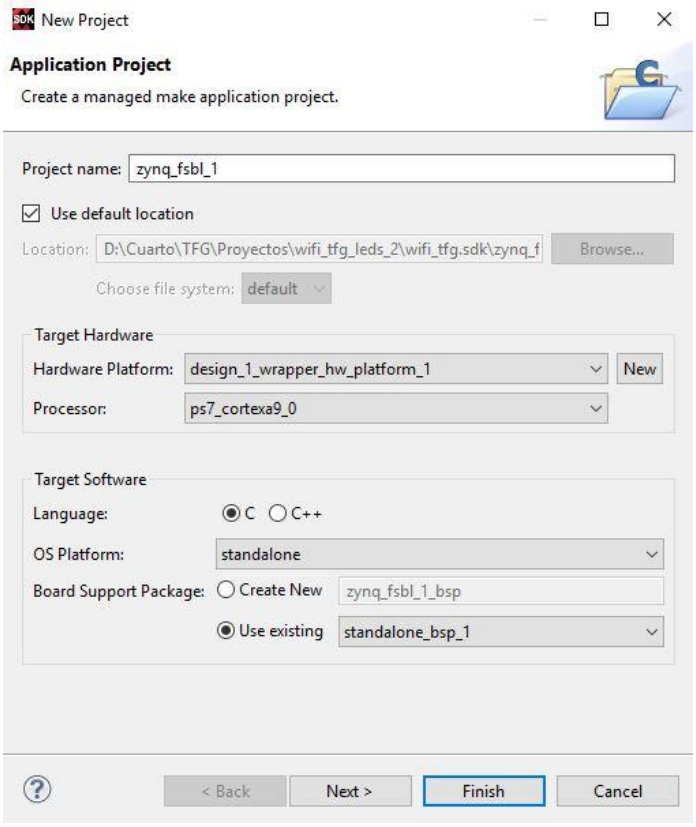

Figura 6.14: Opciones de creación de proyecto de aplicación

En la nueva ventana, mostrada en la Figura 6.15, se debe seleccionar en la parte izquierda la opción *Zynq FSBL* y, por último, *Finish*.

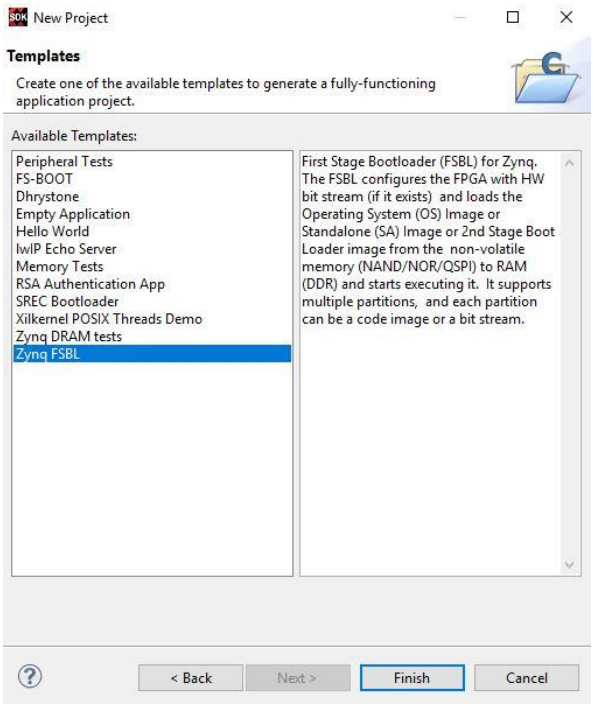

Figura 6.15: Elección de plantilla "Zynq FSBL" para el proyecto de aplicación

Una vez realizados todos los pasos anteriores, se ha generado el archivo *zynq_fsbl_1.elf* que es el que se empleará como FSBL en el sistema. Este archivo se encuentra en la ruta de archivos: <directorio del proyecto Vivado>/sdk/zynq_fsbl_0/Debug/zynq_fsbl_0.elf.

6.3 Modificación y compilación del kernel de Linux

Al igual que en los apartados anteriores, se va a partir de la configuración de PetaLinux SDK empleada anteriormente. A esta configuración, se realizarán los cambios necesarios para que el sistema de archivos se almacene en otra partición de la tarjeta SD con el objetivo de poder guardar los cambios realizados en dicho sistema operativo aun cuando se apague el sistema. Además, de esta forma, se pueden instalar algunas aplicaciones que se van a emplear como el servidor web y configurarlo para que arranque al iniciar el sistema.

En este capítulo, se van a emplear diferentes GPIO para controlar los LEDs de la placa ZedBoard. Por defecto, en la configuración de Linux que genera PetaLinux SDK el acceso a los GPIO tiene un offset de 138. Esto significa que, por ejemplo, para modificar el estado del LED conectado a la parte PS que tiene asignado el GPIO número 7, si no se realiza ninguna modificación, habría que referirse a él como si fuera el GPIO número 145. El procedimiento para modificar el valor de un pin se explicará detalladamente en el apartado 6.7.2.

Para eliminar este offset, se debe modificar el archivo *gpio-zynq.c* que se encuentra en el directorio donde está instalado PetaLinux SDK. El archivo indicado anteriormente se encuentra en la ruta de directorios /opt/pkg/petalinux-v2014.4-final/components/linux-kernel/xlnx-3.17/drivers/gpio/gpio-zynq.c. La línea que se debe modificar es la número 606 donde se debe sustituir:

```
chip-> base = -1;
```

por:

```
chip-> base = 0;
```

Para poder modificar el archivo anterior es necesario tener permisos de administrador del sistema.

Una vez modificado el archivo *gpio-zynq.c*, para poder emplear desde un terminal el entorno PetaLinux SDK, se debe indicar donde se encuentran las variables de dicho entorno con la instrucción:

- source /opt/pkg/petalinux-v2014.4-final/settings.sh

El primer paso que se debe dar es configurar PetaLinux SDK para que el sistema de archivos de Linux se ubique en una segunda partición de la tarjeta SD en lugar de en memoria RAM. Esto permite que se puedan instalar otras aplicaciones que no vienen instaladas en los archivos de Linux que genera PetaLinux SDK y que estos cambios efectuados en el sistema no se borren tras apagarlo.

Para configurar PetaLinux SDK de forma que el sistema de archivos se ubique en una segunda partición de la tarjeta SD se debe ejecutar la siguiente instrucción:

- petalinux-config

En la ventana de configuración que se abre, se selecciona la opción *Image packaging configuration*. De las opciones disponibles se debe modificar *root filesystem* eligiendo *sd card*. Por último, se habilita la compresión de kernel que se encuentra en esa misma ventana de configuración.

Al realizar el cambio anterior, el sistema de archivos que se debe ubicar en la segunda partición no se genera al ejecutar la instrucción *petalinux-build*. Para poderlo generar se deben realizar algunas modificaciones adicionales.

En la ventana de configuración anterior se debe desmarcar la opción *kernel autoconfig* que se encuentra en la opción *Auto config settings*.

Tras realizar el cambio se debe seleccionar *Save* y posteriormente *Ok*.

También es necesario realizar algunas modificaciones en el kernel de Linux. Para ello, se debe ejecutar la siguiente instrucción:

- petalinux-config –c kernel

En la ventana que se abre, se debe activar la opción *initial RAM filesystem and RAM disk (initramfs/initrd) support* que se encuentra dentro del menú *general setup*. Además, se debe dejar vacío el campo *initramfs source file(s)* que se encuentra en el mismo menú. Antes de salir de esta ventana se deben guardar los cambios seleccionando *Save* y posteriormente *Ok*.

Por último, es necesario deshabilitar la aplicación uWeb que viene por defecto habilitada ya que se va a utilizar otro servidor web más sencillo de utilizar que posteriormente se instalará. Para ello, se debe ejecutar:

- petalinux-config –c rootfs

En las opciones de configuración se debe deshabilitar la aplicación *uWeb* que se encuentra en el menú *Apps*. Una vez realizado el cambio se selecciona *Save* y posteriormente *Ok* para que los cambios queden guardados.

Una vez que se han realizado los cambios anteriores se deben generar los archivos necesarios ejecutando:

- petalinux-build

Cuando la instrucción anterior termina de ejecutarse, en la carpeta *images* que se encuentra en el directorio del proyecto se encuentran dos de los archivos que se van a emplear: *image.ub* y *boot.elf*.

6.4 Modificación de los dispositivos HW accesibles: Device Tree

Se deben realizar algunas modificaciones en el *device tree* que emplea el sistema. Igual que en los casos anteriores, se va a partir del *device tree* que se empleó en el apartado anterior añadiendo los elementos necesarios para el correcto funcionamiento del sistema.

El primer paso es disponer del archivo *system.dts* que se utilizó para generar el *device tree* binario en los apartados anteriores. El archivo binario *system.dtb* se encuentra en la partición FAT de la tarjeta SD. Para poder realizar las modificaciones, se debe convertir a formato *dts* con la siguiente instrucción:

- dtc –I dtb –O dts –o system.dts system.dtb

Tras ejecutar la instrucción anterior, se habrá creado un nuevo archivo llamado *system.dts* que es el que se va modificar. Este archivo se puede modificar con cualquier editor de textos como, por ejemplo, *gedit*.

Lo primero que se debe modificar es el código dentro de *chosen* [32]. Esto nodo, a diferencia del resto de nodos del *device tree*, no representa un dispositivo real del sistema, sino que se emplea para pasar información entre el *firmware* y el sistema operativo que se está ejecutando. La etiqueta *bootargs* es una propiedad del nodo *chosen* que se emplea para pasar argumentos al *kernel* en el momento de arranque del sistema operativo.

En este caso, se desea que se emplee la UART0 de Zynq como terminal serie del sistema operativo. Esto se indica con el parámetro *console* igualado a *ttyPS0* ya que la UART0 es denominada *ttyPS0* por el sistema. Además, se debe indicar la velocidad a la que se va a llevar a cabo la comunicación serie que, en este caso, es 115200 baudios.

Por otra parte, se va a emplear la segunda partición de la tarjeta SD para almacenar todo el árbol de directorios del sistema operativo. Esto se debe indicar con el parámetro *root* igualado a */dev/mmcblk0p2* que representa la segunda partición de la tarjeta SD.

Por último, se van a emplear otros parámetros que indican diferentes características del sistema:

- El parámetro *earlyprintk* se emplea para indicar al sistema que envíe información desde el primer momento del arranque. De esta forma, se puede obtener más información en el caso de que el sistema no arranque correctamente.
- El parámetro *rw* se emplea para montar el sistema de archivos del sistema operativo en el arranque de forma que se pueda tanto leer como escribir en él.
- El parámetro *rootwait* se emplea para que el sistema espere a que el árbol de directorios del sistema esté montado correctamente.

Todas las opciones anteriores se indican en el nodo *chosen* del *device tree* de la siguiente forma:

```
chosen {
       bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk0p2 rw
       rootwait";
};
```

También se debe añadir el dispositivo hardware que controla los LEDs e interruptores a través de la interfaz AXI4-Lite diseñado en Vivado 2014.4. Este código debe ser añadido dentro de la etiqueta *amba* del *device tree*.

Entre las etiquetas más importantes está *compatible* que define el modelo de programación del dispositivo facilitando al sistema operativo la identificación del *driver*. La etiqueta *reg* indica la dirección del registro y su longitud. Por último, las etiquetas *xlnx,gpio-width* y *xlnx,gpio2-width* indican el número de pines que se van a emplear. En este caso, en ambos casos es 8 ya que se dispone de 8 LEDs y 8 interruptores en la tarjeta ZedBoard..

```
axi_gpio_0@41200000 {
       #gpio-cells = <0x2>;
       compatible = "xlnx,xps-gpio-1.00.a";
       gpio-controller;
       reg = <0x41200000 0x1000>;
       xlnx,all-inputs = <0x0>;
       xlnx,all-inputs-2 = <0x0>;
       xlnx,dout-default = <0x0>;
       xlnx,dout-default-2 = <0x0>;
       xlnx,gpio-width = <0x8>;
       xlnx,gpio2-width = <0x8>;
       xlnx,interrupt-present = <0x1>;
       xlnx,is-dual = <0x1>;
       xlnx,tri-default = <0xffffffff>;
       xlnx,tri-default-2 = <0xffffffff>;
};
```

Una vez se han añadido las dos modificaciones anteriores, se crea el *device tree* en formato binario que se añadirá a la partición FAT de la tarjeta SD. Este archivo se consigue con el comado siguiente:

- dtc –I dts –O dtb –o system.dtb system.dts

Una vez realizado este paso, ya se dispone de todos los archivos que se deben ubicar en la partición *BOOT* de la tarjeta SD desde donde se produce el arranque del sistema operativo Linux en ZedBoard.

6.5 Generación archivo BOOT.bin

Antes de proceder a copiar todos los archivos a la partición FAT, se debe generar el archivo *BOOT.bin* de la misma forma que se explicó anteriormente y aquí se indica de forma resumida.

Para la generación del archivo *BOOT.bin* se emplea la herramienta *Create Zynq Boot Image* disponible en Xilinx SDK, que se encuentra instalado en Windows. En dicha herramienta se debe indicar la ruta de los tres archivos que forman el archivo *BOOT.bin*. Estos archivos se deben añadir con el orden que se indica a continuación:

- El primer archivo que se debe añadir es el FSBL o *First Stage BootLoader* generado con la herramienta Xilinx SDK. Este archivo se encuentra en la ruta de directorios: <directorio del proyecto Vivado>/sdk/zynq_fsbl_1/Debug/zynq_fsbl_1.elf
- El segundo de los archivos que se debe añadir es el archivo de configuración hardware creado por la herramienta Vivado 2014.4. El archivo tiene una extensión .bit y se encuentra en la ruta de directorios: <directorio del proyecto Vivado>/sdk/design_1_wrapper_hw_platform_1/design_1_wrapper.bit
- El tercer y último archivo es el SSBL o *Second Stage Boot Loader*. Este archivo ha sido generado por PetaLinux SDK y se encuentra en el directorio *images* del proyecto con el nombre *u-boot.elf*.

Cuando se han añadido los archivos anteriores, se selecciona la opción *Create Image* que genera el archivo *BOOT.bin*.

6.6 Preparación de la tarjeta SD

En este capítulo, el árbol de directorios de Linux se almacenará en la tarjeta SD en vez de en memoria RAM. De esta forma, cuando el sistema se reinicie, todos los cambios realizados quedarán guardados en el árbol de directorios que se ubicará en una segunda partición con formato ext4 de la tarjeta SD. El proceso que se va a seguir en este apartado se muestra en la Figura 6.16.

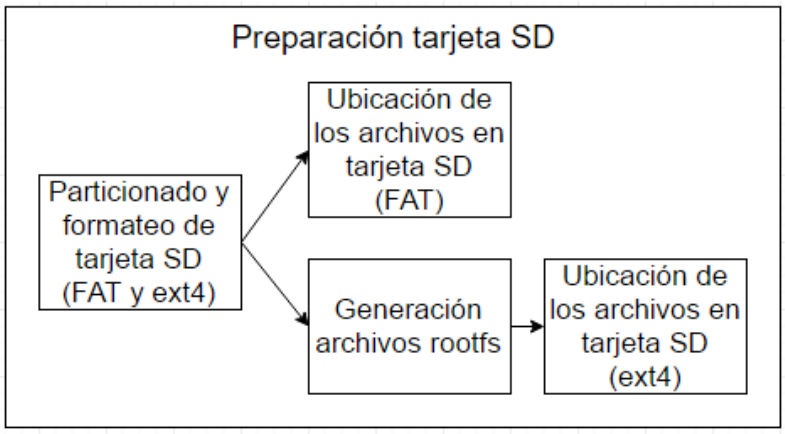

Figura 6.16: Preparación tarjeta SD

6.6.1 Particionado de la tarjeta SD

Para crear la segunda partición se va a emplear el mismo programa empleado anteriormente para dar formato a la tarjeta SD. Partiendo de la partición que se llevó a cabo anteriormente en formato FAT32 y de 1 GB de tamaño, se va a formatear el espacio restante con un formato ext4.

El primer paso es ejecutar el programa GParted y seleccionar la tarjeta SD. La ventana que se abre se puede observar en la Figura 6.17 y en ella se puede observar cómo está distribuido el espacio disponible en la tarjeta SD.

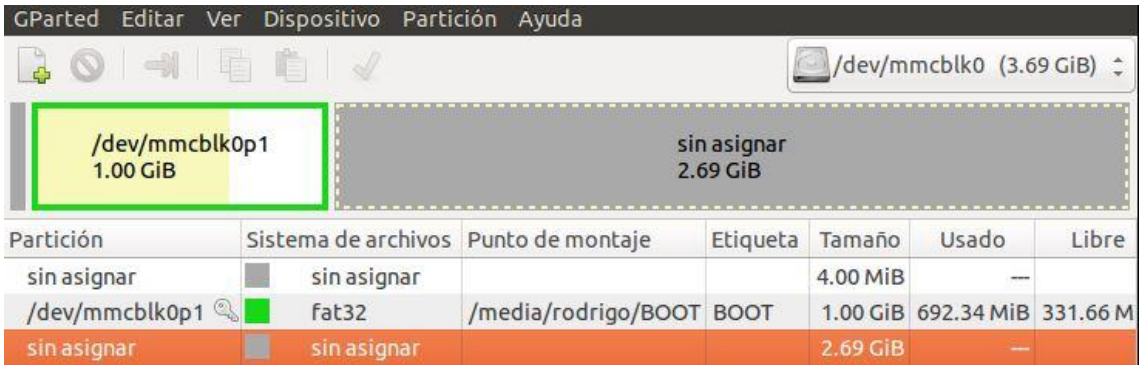

Figura 6.17: Formato inicial tarjeta SD

En este caso se va a dar formato al espacio sin asignar que se dejó anteriormente. Para ello, se hace *click* con el botón derecho sobre "sin asignar" y se selecciona *nuevo*. En la ventana que se abre se debe dar nombre a la partición indicando en el campo etiqueta "*rootfs*", indicar el tipo de partición ext4 y seleccionar el tamaño. Cuando se han realizado todos los cambios, se selecciona *Ok* y, a continuación, se debe pulsar sobre el *tick* que se encuentra en la parte superior de la ventana. Tras realizar esta última acción, aparece una ventana nueva donde se debe aceptar el aviso de que la operación no se puede deshacer. Cuando el proceso termina, la tarjeta dispondrá de dos particiones, una en FAT32 y de un tamaño de 1 GB y una segunda partición en formato ext4 que ocupa el resto del espacio disponible. Esta distribución puede observarse en la Figura 6.18.

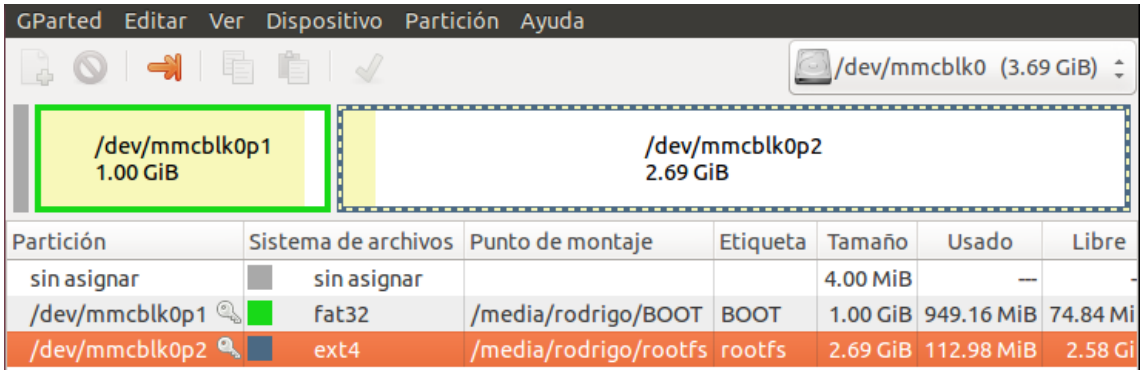

Figura 6.18: Formato final tarjeta SD

6.6.2 Localización de archivos en partición FAT

Una vez que la tarjeta SD tiene el formato adecuado, se deben ubicar los siguientes archivos en la partición denominada *BOOT* que tiene un formato FAT32:

- image.ub: este archivo es creado por PetaLinux y contiene toda la configuración del kernel de Linux. El archivo se puede encontrar en el directorio *images* en la carpeta del proyecto.
- system.dtb: es el *device tree* en formato binario generado anteriormente a partir del archivo *system.dts* y en él se incluyen los dispositivos hardware accesibles.
- BOOT.bin: este archivo incluye el FSBL, el archivo de configuración hardware y el SSBL.

6.6.3 Generación y localización de archivos en partición ext4

Se debe generar el archivo *rootfs.cpio* donde se encuentra todo el sistema de archivos de Linux. Para ello, se debe ejecutar la siguiente instrucción:

- petalinux-package --image –c rootfs --format initramfs

Tras terminar de ejecutarse la instrucción anterior, cuyo resultado se puede observar en la Figura 6.19, en la carpeta *images* se encontrará el archivo *rootfs.cpio* que debe ser descomprimido en la segunda partición de la tarjeta SD.

```
[INFO ] package rootfs.cpio to /home/rodrigo/Documentos/wifi_tfg_2/images/linux
Generating filesystem description file for gen_init_cpio...
Creating /init symlink if required...Done.
directories...files...symlinks...device nodes...done.
Generating CPIO archive /home/rodrigo/Documentos/wifi_tfg_2/images/linux/rootfs.
cpio ...done.
Image Name:
Created:      Mon Jun 12 18:05:19 2017
Image Type:   ARM Linux RAMDisk Image (gzip compressed)
Data Size:    17694773 Bytes = 17280.05 kB = 16.88 MB
Load Address: 00000000
Entry Point:  00000000
```

Figura 6.19: Creación árbol directorios Linux

En la partición *rootfs* se deben ubicar todos los directorios necesarios para el correcto funcionamiento de Linux que se encuentran comprimidos en el archivo *rootfs.cpio*. Para llevar a cabo la descompresión de dicho archivo se va utilizar una herramienta llamada *pax* que se debe instalar antes de poderla utilizar. Las instrucciones que se deben ejecutar son:

- sudo apt-get install pax
- cp <directorio PetaLinux>/images/linux/rootfs.cpio /media/rootfs/
- cd /media/rootfs
- sudo pax –rvf rootfs.cpio

Tras la ejecución de los comandos anteriores, en la partición *rootfs* de la tarjeta SD se encontrarán todas las carpetas que necesita Linux para arrancar correctamente.

6.7 Instalación de servidor y página web en árbol de directorios de Linux

En esta sección, se va a llevar a cabo una explicación detallada del procedimiento que se debe seguir para la instalación y puesta en funcionamiento de un servidor web que se ejecuta en la tarjeta de desarrollo ZedBoard. Además, se llevará a cabo el desarrollo de una página web y un CGI, que se ubicarán en dicho servidor, con los cuales se podrá controlar algunos de los elementos que forman la placa de desarrollo utilizada.

6.7.1 Instalación y configuración del servidor web

Para alojar la página web que se empleará para controlar los LEDs y saber el estado de los interruptores disponibles en la placa Zedboard se va a emplear un servidor web llamado Boa. Este servidor se caracteriza por su pequeño tamaño y su fácil configuración.

Para su instalación y puesta en funcionamiento se debe descargar la versión 0.94.13 disponible en [33]. Una vez descargado, se procede a descomprimir el archivo con el comando:

- sudo tar –xzvf boa-0.94.13.tar.gz

Una vez descomprimido, de debe ejecutar:

- cd /boa-0.94.13/src
- ./configure

Tras ejecutar lo anterior, se ha generado un *makefile* en la carpeta *src*. Se deben realizar varias modificaciones para el correcto funcionamiento del servidor.

El primer cambio que se debe realizar es en el archivo *makefile* recién creado modificando el compilador. Para ello, se modifica las siguientes líneas:

```
CC = arm-xilinx-linux-gnueabi-gcc

CPP = arm-xilinx-linux-gnueabi-gcc –E
```

También se deben realizar algunas modificaciones en diferentes archivos.

En el archivo “boa.c” se deben borrar las líneas 225, 226 y 227:

```
if (setuid(0) != -1) {
       DIE(“icky Linux kernel bug!”);
}
```

En el archivo “compat.h” se debe reemplazar la línea 120:

```
#define TIMEZONE_OFFSET(foo) foo##->tm_gmtoff
```

Por:

```
#define TIMEZONE_OFFSET(foo) (foo)->tm_gmtoff
```

En el archivo “config.c” se debe borrar las líneas de la 266 a la 286.

```
if (!server_name) {
       …
}
```

La última modificación que se debe hacer es borrar las líneas 72, 73 y 74 del archivo “log.c”:

```
if (dup2(error_log, STDERR_FILENO == -1) {
       DIE(“unable to dup2 the error log”);
}
```

Una vez realizadas todas las modificaciones anteriores, se debe ejecutar el comando *make* en el directorio *src*.

Esta última instrucción genera dos binarios llamados “boa” y “boa_indexer” que se deberán emplazar en el sistema de archivos de Linux tal y como luego se indicará.

Por último, se debe ejecutar:

- arm-xilinx-linux-gnueabi-strip boa

En la partición *rootfs* de la tarjeta SD se deben añadir algunos directorios y archivos:

- /mnt/log/
- /mnt/log/boa/
- /mnt/www/
- /mnt/www/cgi_bin/
- /etc/boa/

En el directorio /bin se debe copiar los binarios “boa” y “boa_indexer” que se encuentran en la carpeta boa-0.94.13/src.

La carpeta /etc debe contener tres archivos: “passwd”, “group” y “mime.types”. Los primeros dos archivos ya se encuentran en el sistema de archivos que se genera al descomprimir *rootfs.cpio*. El archivo *mime.types* relaciona las extensiones de archivos con los tipos MIME. Un ejemplo de este archivo se puede descargar de [34].

En el directorio */etc/boa/* se debe ubicar el archivo *boa.conf* que se encuentra en */boa-0.94.13*. Una vez copiado dicho archivo, se debe modificar su contenido. A continuación se muestran los elementos más importantes que debe contener el archivo *boa.conf*:

- User 0 (línea 48)
- Group 0 (línea 49)
- ErrorLog /mnt/log/boa/error_log (línea 62)
- AccessLog /mnt/log/boa/access_log (línea 74)
- DocumentRoot /mnt/www (línea 111)
- DirectoryIndex index.html (línea 123)
- Se debe comentar DirectoryMarker (línea 130)
- AddType application/x-httpd-cgi cgi (línea 173)
- ScriptAlias /cgi_bin / mnt/www/cgi_bin/ (línea 193)

El último paso que se va a dar para la configuración del servidor, es la automatización de su arranque cuando se enciende el sistema. Para ello, se crea un script, *boa.sh*, dentro del directorio */etc/init.d/* de la partición ext4 que contenga el siguiente código:

```
#!/bin/sh
cd /bin
./boa
```

Una vez creado, hay que darle permisos de ejecución con la instrucción:

- sudo chmod +x <partición ext4>/etc/init.d/boa.sh

Para terminar de configurar que el servidor arranque automáticamente al iniciar el sistema, se debe ejecutar una última instrucción una vez que el sistema haya arrancado. Esto solo es necesario realizarlo una vez.

Para arrancar el sistema se deben seguir los mismos pasos indicados anteriormente. Se debe introducir la tarjeta SD en la ZedBoard y conectar el cable MicroUSB-USB al ordenador y la conexión UART de la ZedBoard. Una vez conectado el adaptador de alimentación se cambia el interruptor de encendido a “ON”.

Una vez que el sistema haya arrancado, se pedirá el usuario (“root”) y la contraseña (“root”). Tras introducir los datos anteriores correctamente el sistema se encuentra a la espera de recibir comandos.

Para terminar de automatizar el arranque del servidor se debe introducir, a través de TeraTerm, la siguiente instrucción:

- update-rc.d boa.sh defaults

Esta instrucción genera los enlaces simbólicos que necesita el sistema para arrancar el servidor Boa automáticamente cuando se enciende el sistema, según se indica en la Figura 6.20.

Figura 6.20: Adición del servidor Boa al arranque de Linux

Para comprobar el correcto funcionamiento, se reinicia el sistema, apagando y volviendo a encender la placa. Si todo ha funcionado correctamente, justo antes de pedir el usuario y la contraseña, aparecerá la información que devuelve el servidor Boa cuando arranca. Esto se puede observar en la Figura 6.21.

Figura 6.21: Ejecución servidor Boa

6.7.2 Diseño de página web y CGI

Una vez diseñado el sistema hardware y software para el control de los elementos de la ZedBoard e instalado y configurado el servidor en el sistema operativo que se ejecuta sobre ella, se debe crear la página web que permita el control de dichos elementos.

En el directorio */mnt/www/* se debe crear un archivo llamado "index.html" que es el que se mostrará al acceder a la dirección web. En este caso, este código HTML mostrará una pantalla de acceso que permitirá acceder, a través de un botón, a la página desde donde se podrá ver y controlar el estado de los LEDs y conocer la posición de los interruptores.

El archivo debe ser llamado "index.html" y debe contener el siguiente código html:

```
<!DOCTYPE html>
<html>
	<head>
		<title>Control elementos ZedBoard desde servidor Web</title>
	</head>
	<body>
		<center><font size="10">Servidor Web para el control de elementos
			de la placa ZedBoard</font></center><br>
		<form action="/cgi_bin/control.cgi" method="get">
			<center><input type="submit" value="ACCEDER"
style='width:300px; height:70px'></center>
		</form>
	</body>
</html>
```

En la carpeta */mnt/www/cgi_bin/* se debe crear un archivo CGI el cual permite intercambiar información entre el servidor y el usuario desde la página web. De esta forma, al usuario se le presentará una página web donde se indicará en todo momento que LEDs están encendidos y

cuales apagados y la posición de los interruptores. Además, el usuario podrá modificar el estado de los LEDs desde esa misma página web.

Por tanto, se debe crear un nuevo archivo llamado "control.cgi" que debe contener un código similar al mostrado en el Anexo I.

A continuación, se mostrarán los fragmentos más representativos de dicho CGI con una explicación de su funcionamiento.

En las primeras líneas del CGI se debe indicar que el CGI debe ser ejecutado con el intérprete de comandos *sh* y que incluye código HTML. Estas primeras líneas del archivo son imprescindibles para que el CGI se interprete de forma correcta y, por tanto, tenga el funcionamiento esperado.

```
#!/bin/sh
echo -e "Content-type: text/html\r\n\r\n"
echo ""
```

Para poder controlar el estado de los LEDs y los interruptores, lo primero que se debe hacer es exportar el pin GPIO que controla cada componente. Esto se hace con la instrucción:

- echo <número GPIO> > /sys/class/gpio/export

El GPIO del LED conectado a la PS es el número 7 de forma predeterminada. En el caso de los LEDs e interruptores, conectados a la parte PL, son del 240 al 247 para los interruptores y del 248 al 255 para los LEDs.

El siguiente paso es indicar si los pines son de entrada o de salida. Para ello, se ejecuta el comando:

- echo in > /sys/class/gpio/gpio<número GPIO>/direction
- echo out > /sys/class/gpio/gpio<número GPIO>/direction

En el caso de los LEDs se indica que son de salida ya que se puede realizar modificaciones de su estado de forma software. Por el contrario, los interruptores deben ser configurados como entrada ya que los cambios no se pueden producir a nivel software.

En el fragmento mostrado a continuación, se puede observar la configuración de los GPIO que se van a emplear en esta aplicación. En este fragmento se muestran, a modo de ejemplo, la configuración de cada uno de los GPIO diferentes que se usan en el sistema: el LED conectado a la parte PS, uno de los interruptores conectados a la parte PL y uno de los LEDs pertenecientes a la parte PL. Cabe destacar que la dirección del GPIO únicamente se indica cuando es de salida ya que, por defecto, los pines están configurados como entradas.

La configuración de los pines se realiza dentro de un *if* que comprueba si la configuración se ha realizado anteriormente. De esta forma, se evita que la modificación del estado de uno de los LEDs implique volver a realizar su configuración.

```
if [ ! -d /sys/class/gpio/gpio7 ];
then
       echo 7 > /sys/class/gpio/export
       echo out > /sys/class/gpio/gpio7/direction

       echo 240 > /sys/class/gpio/export
       (…)
       echo 248 > /sys/class/gpio/export
       (…)
       echo out > /sys/class/gpio/gpio248/direction
```

```
        (…)
fi
```

Para realizar el control de los LEDs, se va a emplear el método *GET*, tal y como se explicará posteriormente. Al emplear este método, la información que se envía al servidor se incluye en la barra de direcciones del navegador. En el CGI se debe extraer dicha información para posteriormente interpretarlos y realizar los cambios adecuados. La extracción de esta información se realiza con las líneas mostradas a continuación.

```
LED=`echo $QUERY_STRING | sed -n 's/^.*led=\([^&]*\).*$/\1/p' | sed "s/%20/ /g"`

LED0=`echo $QUERY_STRING | sed -n 's/^.*led0=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
(…)
LED7=`echo $QUERY_STRING | sed -n 's/^.*led7=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
```

Una vez leída la información anterior, se puede generar la página web adaptada a las condiciones reales del estado de los LEDs y los interruptores. De esta forma, desde la página se podrá conocer el estado de todos estos elementos sin necesidad de observar directamente estos elementos sobre la ZedBoard.

El siguiente código muestra las etiquetas HTML básicas que debe contener el inicio de cualquier página web.

```
Echo "<html><head><title>Control elementos ZedBoard desde servidor
Web</title></head><body>"
echo "<center><font size="20">Servidor Web para el control de elementos de la
placa ZedBoard</font></center><br>"
```

La página web va a estar dividida en dos partes claramente diferenciables: una parte desde la cual el usuario podrá conocer y cambiar el estado de los LEDs y una segunda parte donde se mostrará el estado de los interruptores.

En la parte relacionada con los LEDs, el usuario debe poder interactuar con la página web. Para ello, se va a emplear la etiqueta HTML *form* y el método GET.

La etiqueta *form* permite emplear diferentes tipos de *input* según las necesidades del elemento que se desee controlar. En este CGI se van a emplear dos tipos de *input* diferentes que se explicarán posteriormente.

El envío de la información al servidor se realizará a través del método *GET*. El empleo de este método supone que toda la información se envía en la barra de direcciones junto a la dirección del servidor. Todos los datos se codifican después del símbolo *"?"* que separa la dirección web de la información. Los diferentes datos enviados se separan entre sí por medio del símbolo *&*.

```
Echo "<form action="/cgi_bin/control.cgi" method="get">"
```

Para controlar el estado del LED conectado a la parte PS, se va a emplear la etiqueta *input* con el tipo *radio*. Además, la opción seleccionada cuando se carga la página web será la que indica el estado real del LED. De esta forma, cuando se desee cambiar el estado únicamente será necesario seleccionar la opción no marcada.

En el caso de los GPIO configurados como salida, como los pines que controlan los LEDs, para modificar el valor que presentan se ejecuta:

- Para poner a nivel alto: echo 1 > /sys/class/gpio/gpio<número GPIO>/value
- Para poner a nivel bajo: echo 0 > /sys/class/gpio/gpio<número GPIO>/value

Cabe destacar que los GPIO configurados como salidas no pueden ser leídos para conocer el estado actual que presentan.

```
echo "CONTROL LED PS<br>"
if [ $LED = "encender" ];
then
       echo 1 > /sys/class/gpio/gpio7/value
       echo      "<input      type="radio"      name="led"      value="encender"
checked="checked"><font size="5">Encendido</font><br>"
       echo     "<input     type="radio"     name="led"     value="apagar"><font
size="5">Apagado</font><br>"
else
       echo 0 > /sys/class/gpio/gpio7/value
       echo    "<input    type="radio"    name="led"    value="encender"><font
size="5">Encendido</font><br>"
       echo      "<input      type="radio"      name="led"      value="apagar"
checked="checked"><font size="5">Apagado</font><br>"
fi
```

Para controlar el estado de los 8 LEDs conectados a la parte PL, se va a emplear la etiqueta *input* de tipo *checkbox*. De esta forma, cada LED dispondrá de un *checkbox* que controlará su estado permaneciendo marcado cuando el LED esté encendido y desmarcado cuando se encuentre apagado. El usuario podrá controlar el estado de cada LED marcando o desmarcando su correspondiente *checkbox*.

En el fragmento siguiente se muestra el código necesario para el control de uno de los LED de la parte PL. Este código debe ser replicado un total de ocho veces para lograr el control de todos los LEDs disponibles en la parte PL.

```
echo "CONTROL LEDS PL<br>"
if [ $LED0 = "led0" ];
then
       echo 1 > /sys/class/gpio/gpio248/value
       echo      "<input      type="checkbox"      name="led0"      value="led0"
checked="checked"><font size="5">LED 0</font><br>"
else
       echo 0 > /sys/class/gpio/gpio248/value
       echo "<input type="checkbox" name="led0" value="led0" <font size="5">LED
0</font><br>"
fi
```

Para que los cambios surtan efecto, la información debe ser enviada al servidor. Este envío se realizará cuando el usuario pulse el botón ideado para ello.

```
echo "<input type="submit" value="ACTUALIZAR">"
```

Desde la página web también se puede consultar el estado de los ocho interruptores ubicados en la parte PL de la tarjeta ZedBoard.

Los GPIO que se encuentren configurados como entradas, como los pines que están conectados a los interruptores, pueden ser leídos para conocer su estado. Para ello, se ejecuta la instrucción:

- cat /sys/class/gpio/gpio<número GPIO>/value

La instrucción anterior devuelve como resultado un 1 o un 0 dependiendo del estado del GPIO consultado.

En el caso de la página web que se está diseñando, la información del estado de los interruptores se mostrará a partir de una tabla compuesta por 8 columnas. De esta forma cuando un

interruptor se encuentre a nivel alto, su columna se mostrará de color verde. En caso contrario, la columna se mostrará de color rojo.

Esta información se actualizará con los valores actuales de los pulsadores cada vez que se actualice la página web.

```
echo "ESTADO DE LOS INTERRUPTORES<br>"
echo "<table><tr>"
i=240
max=248
while [ $i -lt $max ]
do
       VALOR=`cat /sys/class/gpio/gpio$i/value`
       true $(( i++ ))
       if [ $VALOR = "1" ];
       then
              echo "<th bgcolor="00FF00">ON</th>"
       else
              echo "<th bgcolor="FF0000">OFF</th>"
       fi
done
echo "</tr></table>"
echo "</form></body></html>"
```

Se debe dar permisos de ejecución al CGI creado anteriormente con la instrucción siguiente:

- chmod +x control.cgi

Una vez realizado el paso anterior ya se ha terminado de configurar el sistema y se puede poner en funcionamiento.

6.8 Comprobación de funcionamiento

Una vez el sistema ha arrancado correctamente, se va a crear una red WiFi a la cual se pueda conectar cualquier dispositivo que disponga de esa conectividad inalámbrica. Una vez los dispositivos se han conectado, podrán acceder a la página web que se encuentra ubicada en el servidor de la placa ZedBoard. Desde dicha página web, los usuarios podrán controlar el estado de los distintos LEDs que se encuentran disponibles en la tarjeta ZedBoard. Además, también podrán ver el estado de los interruptores que incluye la placa.

6.8.1 Creación de un punto de acceso WiFi

Para tener acceso al servidor web instalado en la ZedBoard, se puede configurar el módulo WiFi como un punto de acceso al cual se pueden conectar diferentes dispositivos que soporten este tipo de conectividad inalámbrica. Para ello, primero se debe configurar todas las características que tendrá dicha red WiFi tales como su identificador o SSID o su tipo de seguridad. Posteriormente, se activará la red para que se encuentre disponible para que se conecten otros dispositivos.

6.8.1.1 Configuración de las propiedades del punto de acceso WiFi

Se puede configurar el punto de acceso que se va a crear modificando un archivo que se encuentra ubicado en el directorio *usr/share/wl18xx/hostapd.conf*. En este archivo se puede

modificar el nombre de la red, el tipo de protección de la red y la contraseña de red entre otros parámetros.

Para modificar dicho archivo, se deben ejecutar los siguientes comandos:

- cd usr/share/wl18xx
- vi hostapd.conf

Este último comando ejecuta el programa de edición de texto "*vi*" y abre el archivo que se debe modificar.

El programa "*vi*" tiene dos modos de funcionamiento: modo comandos y modo insertar.

Cuando el programa se inicia se encuentra en el modo comandos. En este modo se pueden realizar distintas acciones como copiar una línea (yy), pegar una línea (p), borrar una o varias líneas (dd) y salir del programa. Para volver a este modo desde el modo insertar se debe pulsar la tecla *esc*.

Para pasar al modo insertar hay diferentes opciones dependiendo del cambio que se desea hacer en el documento abierto. Pulsando la letra "*i*", se pasa al modo insertar y queda a la espera de introducir caracteres que se insertarán antes de donde se encontrara el cursor.

Se va a crear una red WiFi con el nombre "ZedBoard" y una protección de red tipo WEP. Para modificar el SSID de la red, se debe colocar el cursor en la línea 94 del archivo *hostapd.conf*. Una vez en esa línea, se debe pulsar la tecla "*i*" y, a continuación, modificar el nombre por "ZedBoard". Para activar la contraseña WEP, se debe descomentar la línea 366 del archivo que se está modificando. Para ello, se debe borrar la almohadilla (#) de esa línea. Por último, se debe descomentar también la línea 375, eliminando la almohadilla (#). La contraseña que tendrá la red será la indicada tras el igual. La parte del archivo relacionada con el tipo de seguridad de la red WiFi descrita anteriormente puede observarse en la Figura 6.22.

Figura 6.22: Configuración del archivo hostapd.conf

Una vez realizados los cambios anteriores, se deben guardar los cambios. Para ello, primero se debe pulsar la tecla *esc* para pasar al modo comando y, a continuación, pulsar ":" y "wq". Este comando guarda los cambios y cierra el archivo.

6.8.1.2 Activación del punto de acceso

Antes de activar el punto de acceso, se debe configurar la dirección IP del punto de acceso y el archivo de configuración del DHCP. Es posible ver las características configuradas por defecto ejecutando el comando:

- vi ap_start.sh

En el documento que se abre, se puede observar la dirección IP, denominada "IP_ADDR" en la Figura 6.23, que se va utilizar para comprobar el correcto funcionamiento del punto de acceso. Una vez consultada la dirección IP, se cierra el archivo con ":q".

Figura 6.23: Configuración del archivo ap_start.sh

Para crear el punto de acceso, se debe ejecutar el archivo ap_start.sh:

- ./ap_start.sh

La ejecución de la instrucción anterior debe devolver un resultado similar a lo que se muestra en la Figura 6.24 donde se indica que la red WiFi ha sido creada:

Figura 6.24: Activación de red WiFi

6.8.2 Conexión a la red WiFi y control de elementos de ZedBoard

Con un dispositivo con conectividad WiFi escaneamos las redes inalámbricas disponibles y nos conectamos a la red que tenga el nombre dado en la configuración de la red, introduciendo la contraseña si es necesario. Con la configuración indicada anteriormente, el nombre de la red será "ZedBoard" y la contraseña "123456789a".

Si la conexión es correcta, en el terminal se mostrará la dirección MAC del equipo que se ha conectado a la red, tal y como se puede observar en la Figura 6.25.

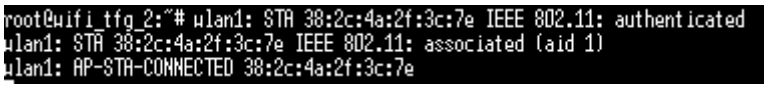

Figura 6.25: Información del dispositivo conectado

Una vez que el terminal está correctamente conectado a la ZedBoard a través de la red WiFi, puede acceder a la página web alojada en el servidor. La dirección IP que hay que introducir en el navegador será la indicada en el archivo *ap_start.sh* que, en la configuración indicada anteriormente, es la dirección 192.168.43.1.

Cuando se accede a esa dirección desde el navegador, el servidor responde con el código HTML que muestra una página de presentación con un botón para acceder a la página desde la cual se podrá modificar el estado de los LEDs y comprobar el nivel de los interruptores. La página de acceso se puede observar en la Figura 6.26.

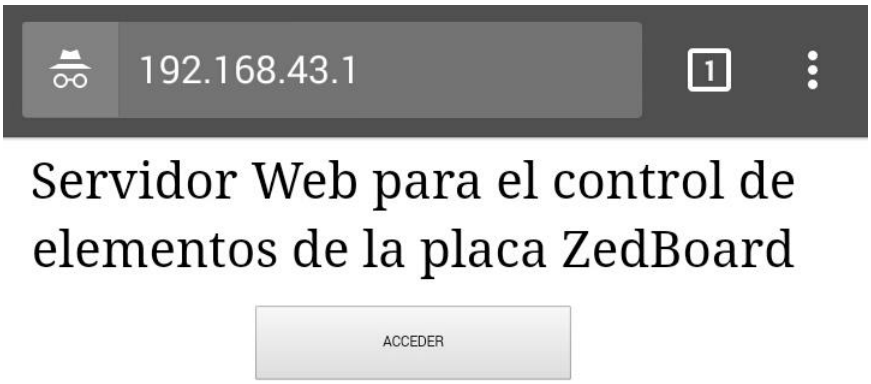

Figura 6.26: Página web de acceso

Una vez que se pulsa el botón de acceder, se redirige al usuario a la página desde la cual podrá interactuar con el servidor. Esta página es la controlada por el CGI, que permite que el código HTML necesario para la generación de la página web sea dinámico, es decir, que se adapte al estado actual de los LEDs y los interruptores.

Por un lado, cuando el usuario quiere encender o apagar un LED, debe cambiar a la opción elegida en el caso del LED conectado a la parte PS o seleccionar/deseleccionar el *tick* correspondiente al LED deseado si este es uno de los 8 conectados a la parte PL de la ZedBoard. Cuando se produce un cambio y se pulsa el botón de actualizar, se envía, a través del método *get* por medio de la barra de direcciones los cambios realizados al servidor. En el servidor se ejecuta el CGI y lleva a cabo las acciones necesarias en la ZedBoard.

Por otro lado, el CGI se encarga de mantener la página web actualizada de forma que, tras realizar cada cambio, se muestre el estado actual de los LEDs indicando para cada uno de ellos si se encuentra encendido o apagado. Además, el CGI se encarga de leer el estado de los interruptores cada vez que se actualiza la página indicando si estos se encuentran en posición ON u OFF.

Un ejemplo del funcionamiento explicado anteriormente se puede observar en la Figura 6.27.

Figura 6.27: Página web de control y resultado en ZedBoard

6.8.3 Desconexión de la red WiFi

Cuando un dispositivo conectado a la red se desconecta de ella, se muestra la dirección MAC del dispositivo que se ha desconectado, tal y como se muestra en la Figura 6.28.

```
wlan1: AP-STA-DISCONNECTED 38:2c:4a:2f:3c:7e
wlan1: STA 38:2c:4a:2f:3c:7e IEEE 802.11: disassociated
wlan1: STA 38:2c:4a:2f:3c:7e IEEE 802.11: deauthenticated due to inactivity (timer DEAUTH/REMOVE)
```

Figura 6.28: Desconexión de un equipo de la red WiFi creada

Para finalizar la creación del punto de acceso, se debe ejecutar el comando:

- ./ap_stop.sh

La ejecución de la instrucción anterior devuelve una respuesta como la mostrada en la Figura 6.29.

```
root@Wilink8_sdk:/usr/share/wl18xx# ./ap_stop.sh
Terminating DHCP
Terminating hostapd
wlan1: interface state ENABLED->DISABLED
wlan1: AP-DISABLED
nl80211: deinit ifname=wlan1 disabled_11b_rates=0
root@Wilink8_sdk:/usr/share/wl18xx# wlcore: down
```

Figura 6.29: Deshabilitación de la red WiFi creada

De esta forma, la red WiFi creada por el módulo dejará de estar disponible para que se conecten dispositivos.

Capítulo 7: Aplicación E/S remota con red de sistemas reconfigurables

En este capítulo se va desarrollar un sistema más complejo que aúne los desarrollos llevados a cabo en los capítulos anteriores. De esta forma, se empleará el sistema desarrollado en el capítulo anterior en el cual se hace uso de un servidor web y, además, se utilizará el protocolo SSH para establecer comunicaciones seguras entre los distintos sistemas que componen la aplicación.

El objetivo de la aplicación es controlar un conjunto de sistemas conectados entre sí por una red inalámbrica local. La generación de dicha red así como su conexión con otra red local externa, se logra a través de un sistema concentrador. Este concentrador es el encargado de albergar un servidor web al cual cualquier dispositivo de la red local externa puede acceder para controlar, desde una página web, los dispositivos ubicados en la red local inalámbrica.

Un esquema de la aplicación descrita anteriormente puede observarse en la Figura 7.1.

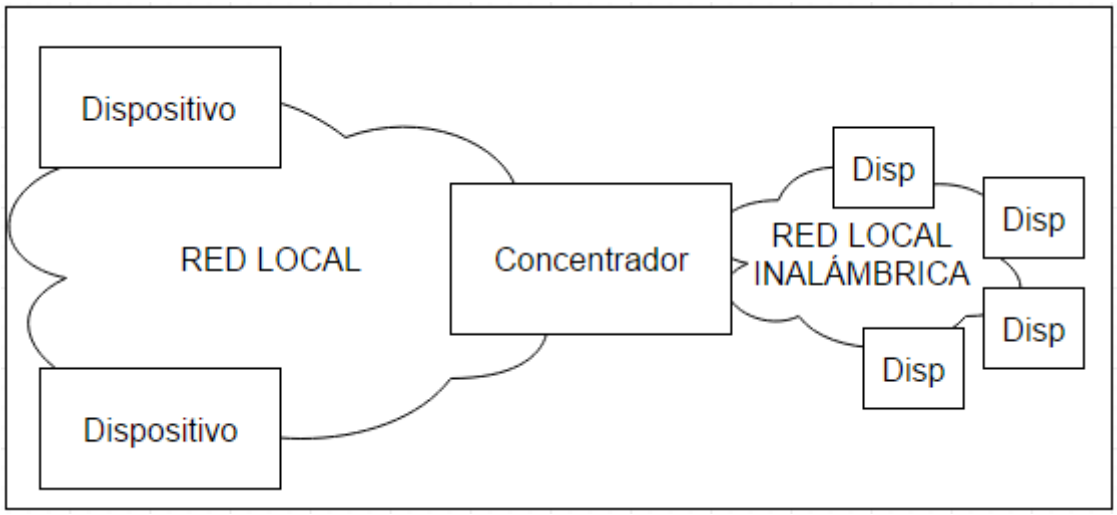

Figura 7.1: Esquema general de aplicación

Para implementar la aplicación anterior, se van a emplear dos tarjetas de desarrollo ZedBoard donde cada una de ellas cumplirá un papel diferente en el sistema.

Por un lado, una de las placas ZedBoard cumplirá el papel del sistema concentrador el cual estará conectado a la red externa a través de una conexión Ethernet. El sistema concentrador también es el encargado de generar una red local WiFi a la cual se conectará la segunda de las tarjetas ZedBoard. A través de la conexión Ethernet, los dispositivos conectados a la misma red externa podrán acceder a una página web albergada en un servidor que se ejecuta en el sistema concentrador. Desde esta página, se puede observar y controlar el estado de todos los LEDs disponibles tanto en esta como en la segunda de las tarjetas ZedBoard. Asimismo, desde la misma página se puede conocer el estado de todos los interruptores del sistema.

Por otro lado, una segunda tarjeta ZedBoard será el dispositivo que se conectará a la red inalámbrica creada por el sistema concentrador y que, por tanto, será accedido de forma remota. La conexión desde el sistema concentrador se establecerá a partir del protocolo SSH

asegurando de esta forma un intercambio seguro de información. Por ello, este sistema debe ejecutar un servidor SSH al cual se pueda conectar el concentrador actuando de cliente SSH.

En la Figura 7.2 se puede observar un esquema del sistema que se va a implementar a lo largo de este capítulo.

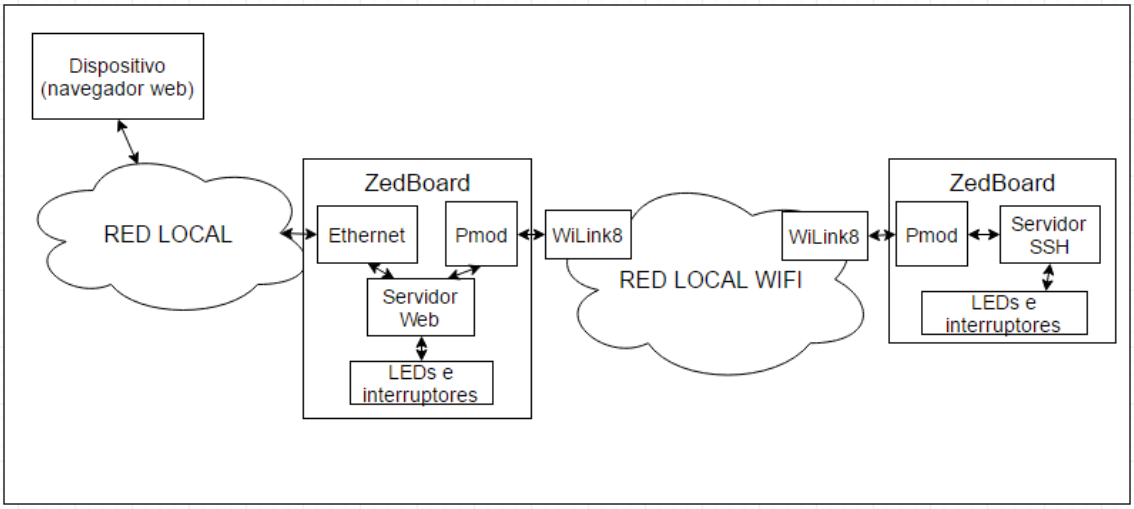

Figura 7.2: Esquema aplicación

7.1 Diseño del sistema

La aplicación que se va a desarrollar requiere, por tanto, dos tipos de sistemas que cumplen diferentes cometidos.

7.1.1 Sistema concentrador

En primero de los sistemas es el que hace de nexo entre una red local externa en la cual se encuentran otros dispositivos y una segunda red local a la cual se conecta la segunda de las tarjetas ZedBoard.

Este sistema va a utilizar el mismo diseño realizado en el capítulo anterior al cual únicamente se deben llevar a cabo dos modificaciones. Por un lado es necesario modificar el *device tree* para poder emplear la conexión Ethernet disponible. Por otro lado, se debe adaptar el CGI encargado de controlar los diferentes elementos accesibles y generar la página web que se muestra al usuario.

7.1.1.1 Modificación *device tree*

Para poder hacer uso de la conexión Ethernet, es necesario realizar una pequeña modificación en el archivo *system.dts* que contiene el *device tree* del sistema.

El nodo que se debe modificar es el que identifica el dispositivo Ethernet denominado *ethernet@e000b000*. Una de las etiquetas más importantes es *compatible* que indica el nombre del *driver* que controla el dispositivo, en este caso *xlnx,ps7-ethernet-1.00.a*. Además, se debe indicar la dirección del registro y su tamaño en la etiqueta *reg*. Otra de las etiquetas es *local-mac-address* que permite seleccionar la dirección MAC que identificará al sistema.

Dentro del nodo *ethernet* se debe indicar también el dispositivo que se encarga del control de la capa física. En la tarjeta ZedBoard, este dispositivo es el 88e1510 de Marvell.

Una posible configuración del nodo *Ethernet* se puede observar en el código mostrado a continuación.

```
ethernet@e000b000 {
	compatible = "xlnx,ps7-ethernet-1.00.a";
	reg = <0xe000b000 0x1000>;
	status = "okay";
	interrupts = <0x0 0x16 0x4>;
	clocks = <0x1 0xd 0x1 0x1e>;
	clock-names = "ref_clk", "aper_clk";
	local-mac-address = [00 0a 35 00 b0 b5];
	xlnx,has-mdio = <0x1>;
	#address-cells = <0x1>;
	#size-cells = <0x0>;
	phy-mode = "rgmii-id";
	xlnx,ptp-enet-clock = <0x69f6bcb>;
	xlnx,enet-clk-freq-hz = <0x17d7840>;
	xlnx,eth-mode = <0x1>;
	phy-handle = <0x4>;
	mdio {
		#address-cells = <0x1>;
		#size-cells = <0x0>;
		phy0:phy@0{
			compatible="marvell,88e1510";
			device_type="ethernet-phy";
			marvel,reg-ini=<3 16 0xff00 0x1e 3 17 0xfff0 0x00>;
			reg=<0>;
			linux,phandle=<0x4>;
			phandle=<0x4>;
		};
	};
};
```

Una vez realizada la modificación anterior, se debe convertir el *device tree* a formato binario de forma que pueda ser leído en el arranque del sistema. Para ello, se debe ejecutar el comando siguiente en un terminal de Ubuntu:

- dtc –I dts –O dtb –o system.dtb system.dts

De esta forma, se creará un archivo llamado *system.dtb* que contiene en formato binario el *device tree* del sistema.

7.1.1.2 Modificación CGI

El CGI empleado en el capítulo anterior sirve como punto de partida para la elaboración del CGI que realice todas las funciones necesarias en esta aplicación. Así, se va a mantener todo el código explicado en el capítulo 6 ya que este va a permitir controlar los LEDs y ver el estado de los interruptores de la tarjeta ZedBoard que alberga el servidor web.

Sin embargo, para poder controlar el estado de los LEDs de la tarjeta ZedBoard que se encuentra conectada a la red inalámbrica, se debe ejecutar el código en ella de forma remota. Para ello, se va a emplear el protocolo SSH que permite la conexión con estos sistemas de forma segura.

Para que las instrucciones del CGI se ejecuten en otro sistema a través de SSH, cada una de ellas debe estar precedida por el comando:

- ssh root@<dirección IP servidor SSH> “<instrucción que se desea ejecutar>”

A diferencia de la conexión SSH realizada en el capítulo 5, en este caso esta será llevada a cabo de forma automática desde el CGI. Por esto, es necesario que no haya que introducir la contraseña cada vez que se establece la conexión. Tal y como se explicará posteriormente en este mismo capítulo, se emplearán claves RSA siendo necesario indicar la ubicación de dichas claves con la opción *–i* del comando SSH. Por ello, las instrucciones del archivo CGI tendrán la estructura siguiente:

- ssh root@<dirección IP servidor SSH> -i<ubicación claves RSA> “<instrucción que se desea ejecutar>”

7.1.2 Sistemas pertenecientes a la red local WiFi

La tarjeta ZedBoard conectada a la red inalámbrica, no necesita ejecutar un servidor web ya que sus elementos serán controlados desde el servidor ubicado en el sistema concentrador.

Por ello, se va a emplear el sistema generado en el capítulo 6 hasta el apartado 6.6 incluido. De esta forma, el sistema dispondrá de todas las capacidades que el sistema concentrador a excepción del servidor web.

Por tanto, en este caso no es necesario realizar ninguna modificación adicional al sistema ya descrito anteriormente.

7.2 Puesta en marcha y configuración inicial

Antes de poder utilizar el sistema, es necesario la generación y conexión a la red local inalámbrica. Además, la primera ejecución requiere una configuración adicional de forma que la aplicación SSH no requiera de la introducción de una contraseña para establecer la comunicación. Para ello, y sustituyendo la contraseña, se emplearán un sistema de criptografía de clave pública denominado RSA.

7.2.1 Conexionado

El sistema que se va a emplear está compuesto por dos tarjetas ZedBoard conectadas independientemente a un ordenador a través de una conexión UART por medio de un cable USB. Además, ambas disponen de un módulo WL1835MOD conectado a un puerto Pmod que les proporciona conectividad WiFi.

La tarjeta ZedBoard que ejerce las funciones de concentrador, además, se encuentra conectada a una red local a través de una conexión Ethernet.

Todas las tarjetas que se emplean en esta aplicación almacenan los archivos necesarios en una tarjeta SD con dos particiones tal y como se explicó anteriormente en el capítulo 6 de este documento. Además, para que el arranque se lleve a cabo desde esta tarjeta, los *jumpers* deben estar colocados tal y como se muestran en la Figura 7.3.

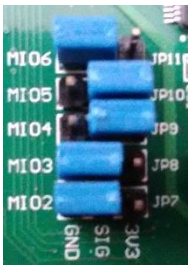

Figura 7.3: Jumpers arranque desde tarjeta SD

Todas las conexiones anteriores necesarias para un correcto del sistema se pueden observar en el diagrama mostrado en la Figura 7.4.

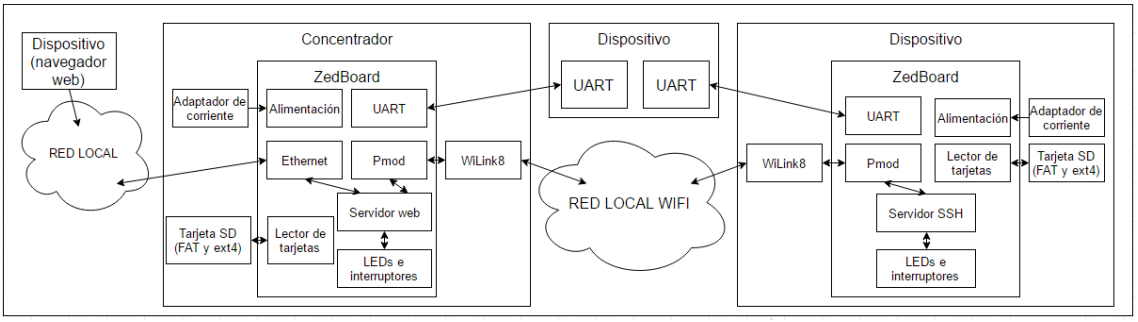

Figura 7.4: Conexionado completo para la ejecución de la aplicación

Una vez realizadas las conexiones anteriores, se encienden ambos sistemas para llevar a cabo la creación y conexión de una red WiFi así como la configuración de la aplicación SSH.

7.2.2 Creación y conexión de red local WiFi

A diferencia de la red creada en el capítulo anterior, en este caso se ha configurado el archivo *hostapd.conf* de forma que la red WiFi esté abierta y no se necesite una contraseña para conectarse a ella.

Para que se habilite la red únicamente es necesario ejecutar el *script* llamado *ap_start.sh* que se encuentra en el directorio *usr/share/wl18xx*. Por tanto, a través de la UART conectada al sistema concentrador se deben ejecutar los siguientes comandos:

- cd /usr/share/wl18xx
- ./ap_start.sh

Si todo funciona correctamente, el resultado devuelto será similar al mostrado en la Figura 7.5.

```
root@wifi_tfg_2:/usr/share/wl18xx# ./ap_start.sh
adding wlan1 interface
Configuration file: /usr/share/wl18xx/hostapd.conf
wlcore: using inverted interrupt logic: 8
wlcore: PHY firmware version: Rev 8.2.0.0.237
wlcore: firmware booted (Rev 8.9.0.0.70)
IPv6: ADDRCONF(NETDEV_UP): wlan1: link is not ready
wlan1: interface state UNINITIALIZED->COUNTRY_UPDATE

./ap_start.sh: line 73: iptables: not found
root@wifi_tfg_2:/usr/share/wl18xx# Using interface wlan1 with hwaddr 34:b1:f7:df
:7d:23 and ssid "ZedBoard"
IPv6: ADDRCONF(NETDEV_CHANGE): wlan1: link becomes ready
wlan1: interface state COUNTRY_UPDATE->ENABLED
wlan1: AP-ENABLED
```

Figura 7.5: Creación red WiFi

Una vez disponible la red WiFi, la placa ZedBoard debe conectarse a dicha red. Para ello, el primer paso es ejecutar el comando:

- wpa_supplicant –d –Dnl80211 –c/etc/wpa_supplicant.conf –iwlan0 –B

Una vez se ha terminado de ejecutar la instrucción anterior, se puede emplear el comando *wpa_cli* para establecer comunicación con la red inalámbrica.

- wpa_cli –iwlan0 add_network
- wpa_cli –iwlan0 set_network 0 ssid "'<nombre de red WiFi>"'
- wpa_cli –iwlan0 set_network 0 key_mgmt NONE
- wpa_cli –iwlan0 enable_network 0

Si todo ha funcionado correctamente, se mostrará un resultado similar al que se puede observar en la Figura 7.6.

Figura 7.6: Asociación a red inalámbrica

Con el objetivo de que en otras ocasiones el sistema se conecte automáticamente a la red WiFi, se puede ejecutar el comando siguiente que guarda los parámetros de configuración de la red introducidos anteriormente:

- wpa_cli –iwlan0 save_config

De esta forma, y tras la ejecución del comando *wpa_supplicant*, automáticamente el sistema intentará conectarse a la red inalámbrica.

Una vez establecida la conexión, es necesario asignar una dirección IP a la tarjeta que se conecta a la red WiFi. Para ello, se ejecuta el comando:

- udhcpc –iwlan0

La respuesta incluye, tal y como se puede apreciar en la Figura 7.7, la dirección IP que le ha sido asignada a la ZedBoard que ejecuta el comando. Esta dirección IP es la que debe ser indicada en el CGI ubicado en el servidor web.

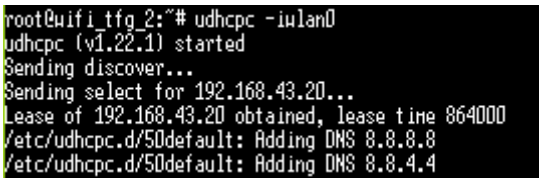

Figura 7.7: Asignación de dirección IP

Antes de que un dispositivo conectado a la red local externa a la que está conectado el concentrador acceda a la web de este, es necesario configurar la aplicación SSH para que se emplee una autenticación basada en claves RSA en vez de en el uso de una contraseña.

7.2.3 Configuración SSH con claves RSA

En este capítulo, las sesiones SSH son establecidas desde el CGI que se ejecuta de forma transparente al usuario. Por ello, no es posible introducir la contraseña cada vez que sea necesario establecer una sesión SSH.

Para sustituir el empleo de una contraseña, se va a emplear el sistema criptográfico de clave pública RSA [35]. Con este sistema, se emplean dos claves relacionadas entre sí. Una de las claves es pública y se emplea para cifrar los archivos que se envían desde el sistema emisor al sistema receptor. La segunda de las claves es privada y se emplea para el descifrado de los archivos que recibe el sistema receptor y que se encuentran codificados con la clave pública. De esta forma, se puede obtener un cifrado extremo a extremo que asegure la confidencialidad y autenticación de los sistemas que establezcan la comunicación.

El protocolo SSH dispone de la posibilidad de emplear el sistema RSA de forma que no sea necesario el empleo de una contraseña. Para utilizar esta forma de autenticación SSH, el cliente SSH debe generar un par de claves RSA. La clave pública se debe enviar al servidor SSH y esta debe ser almacenada en un archivo denominado *authorized_keys*. Así, cuando el cliente SSH desee establecer comunicación con el servidor SSH este comprobará si tiene su clave pública y, en ese caso, no demandará una contraseña para establecer la comunicación.

En el caso de la aplicación que se está diseñando, el sistema concentrador es el dispositivo que necesita conectarse al dispositivo conectado a su red local inalámbrica. De esta forma, el concentrador ejercerá de cliente SSH y el dispositivo conectado de servidor SSH.

Cuando el sistema arranca por primera vez, las claves RSA se crean y se almacenan en el directorio */etc/dropbear* con el nombre *dropbear_rsa_host_key*. Para obtener la clave pública y poderla enviar al servidor SSH, se ejecuta el comando siguiente en el cliente SSH:

- dropbearkey –y –f /etc/dropbear/dropbear_rsa_host_key | grep "^ssh-rsa " >> /etc/dropbear/authorized_keys

De esta forma, se ha creado un archivo llamado *authorized_keys* con la clave pública del cliente SSH.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCIpuq56v7uNOetcI2pTy8LEF2jl0GRomnqBAEY7gOQa3okHC
mPnIY3OI9dkupBSFLpj3/QDPayFaSHJJKTNaP2v3aGA3Orggwf2xlrQVFV5fass//Yr7cvjNee96z1
wjIVT5cX9NHIKt9/XLyjvO47Bvk9G7NDVHRXqTnxyl8Hd/ZA0n1QipSimoG21CBIWj5VuQNGuj2Zz9
oq+QCaaKDS7BjE1qW5xc8zrB0mmlxxIOOKq82u2ZDoI1TsD7vGB3E1dK/6dA/12xp60lnPYnBot2QH
9HgaldtTaQTOV0Y7mU5R5TSJX76Hst/Vf+tkxhSKG1QQXeJLwO7YCRzEaCtF root@wifi_tfg_2
```

Este archivo debe ser enviado al servidor SSH y ubicarlo en el directorio */home/root/.ssh/* Para enviar el archivo se va a emplear la herramienta *scp* que permite, a partir de una conexión SSH, enviar de forma segura un archivo. Así, el comando que se debe ejecutar en el cliente SSH es:

- scp /etc/dropbear/authorized_keys root@<dirección IP ZedBoard destino>:/etc/dropbear/

La ejecución del comando anterior pide la contraseña del sistema remoto que, en este caso, es *root*. Si todo ha funcionado correctamente, se habrá realizado el envío del archivo *authorized_keys* que, además, habrá sido ubicado en la carpeta */etc/dropbear/*. En la Figura 7.8 se puede observar el resultado devuelto por la aplicación *scp* cuando el envío se ha realizado correctamente.

Figura 7.8: Envío archivo authorized_keys con scp

En el servidor SSH, es necesario ubicar el archivo *authorized_keys* en el directorio */home/root/.ssh/*.Para ello, el primer paso es crear dicho directorio y posteriormente copiar el archivo:

- mkdir /home/root/.ssh
- cp /etc/dropbear/authorized_keys /home/root/.ssh

Una vez realizado el proceso anterior, el establecimiento de la conexión SSH no demandará la contraseña de acceso pudiendo ser ejecutada desde un *script*. Para comprobar el correcto funcionamiento, se va a establecer una sesión SSH entre las dos tarjetas ZedBoard ejecutando el siguiente comando:

- ssh root@<dirección IP servidor SSH> –i/etc/dropbear/dropbear_rsa_host_key

Cabe destacar que la opción *–i* indica a la aplicación donde se encuentra almacenado el archivo que contiene las claves RSA del sistema cliente.

Si todo funciona correctamente, la conexión se realizará automáticamente sin necesidad de introducir ninguna contraseña, tal y como puede verse en la Figura 7.9.

Figura 7.9: Conexión SSH sin contraseña

Para cerrar la sesión SSH se debe ejecutar el comando *exit*, volviendo así al terminal del dispositivo concentrador.

7.3 Prueba de funcionamiento

Una vez realizada la configuración anterior, el sistema está preparado para que, desde un dispositivo conectado a la red local externa, se pueda controlar el estado de todos los LEDs y conocer la posición de los interruptores de las dos tarjetas ZedBoard que conforman el sistema.

Así, para acceder al servidor web ubicado en la tarjeta ZedBoard, solamente es necesario introducir la dirección IP asignada a esta.

La página web, mostrada en la Figura 7.10, está dividida en dos partes diferenciadas. En la parte superior aparece la información relacionada con la tarjeta ZedBoard que hace de concentrador en el sistema. En la parte inferior, se muestra toda la información relacionada con la tarjeta ZedBoard que forma parte de la red local creada por el sistema concentrador.

Para cada tarjeta ZedBoard, la página web incluye un conjunto de *checkbox* que indican el estado actual de los LEDs y, además, por medio de estos *checkbox*, el usuario puede modificar su valor. Finalmente, el estado de los interruptores se representa organizado en una tabla.

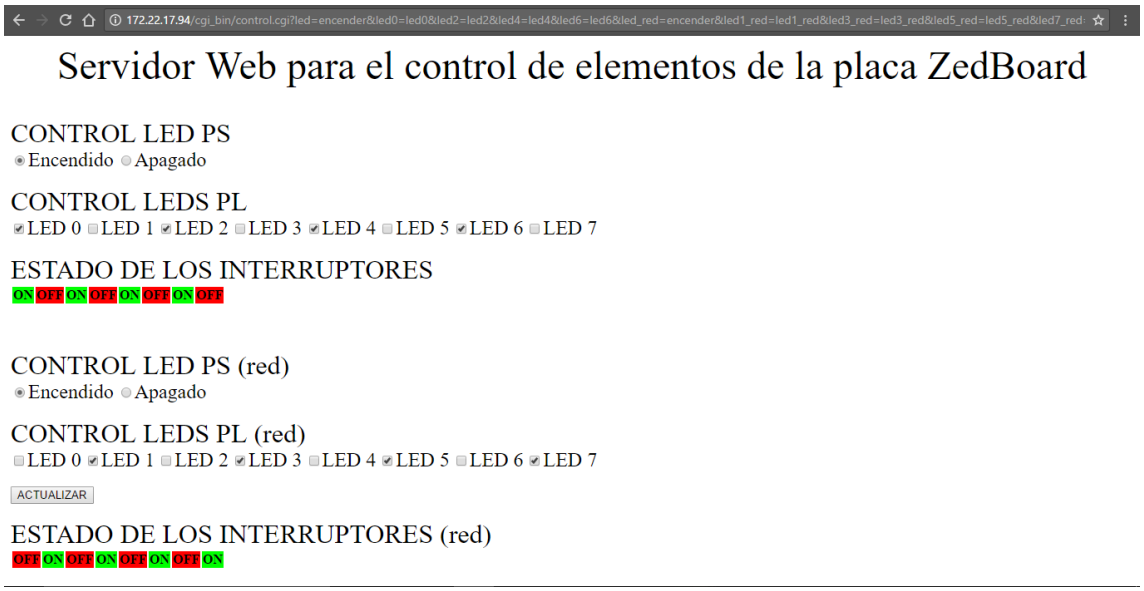

Figura 7.10: Página web de control de los elementos de ámbas tarjetas ZedBoard

El estado de los LEDs e interruptores mostrado en la Figura 7.10 se puede observar en las imágenes de la Figura 7.11. En la parte izquierda, se muestra el estado de los LEDs e interruptores de la tarjeta ZedBoard que cumple con el papel de concentrador en el sistema. En la parte derecha, se muestra el estado de estos elementos en la segunda tarjeta ZedBoard que conforma la aplicación.

Figura 7.11: Estado elementos tarjetas ZedBoard

Capítulo 8: Diseño de una aplicación bare-metal para el control inalámbrico de una ZedBoard

El objetivo de la aplicación que se va a desarrollar es controlar, mediante un cliente TCP y a través de una comunicación inalámbrica, elementos hardware que están incluidos en la tarjeta de desarrollo ZedBoard. En este caso, por simplicidad, se controlará el estado de cada uno de los 8 LEDs de la placa.

Para dar conectividad inalámbrica a la tarjeta ZedBoard se va a emplear un módulo WiFi diferente al empleado en los capítulos anteriores de este trabajo. En este caso, se va a emplear el dispositivo ESP8266 que permite su utilización en sistemas más simples que no ejecutan un sistema operativo.

De esta forma, esta aplicación se ejecutará de modo *standalone* sin necesidad de utilizar un sistema operativo, como el empleado en capítulos anteriores, sobre el que se ejecute la aplicación.

Al igual que en el resto de aplicaciones desarrolladas en este trabajo, es necesario realizar un diseño dual dividido en una parte Hardware, encargada de configurar la parte reprogramable del dispositivo Zynq, y otra parte Software, necesaria para la elaboración del programa que controla el funcionamiento del sistema.

Un esquema general de funcionamiento de la aplicación se puede observar en la Figura 8.1.

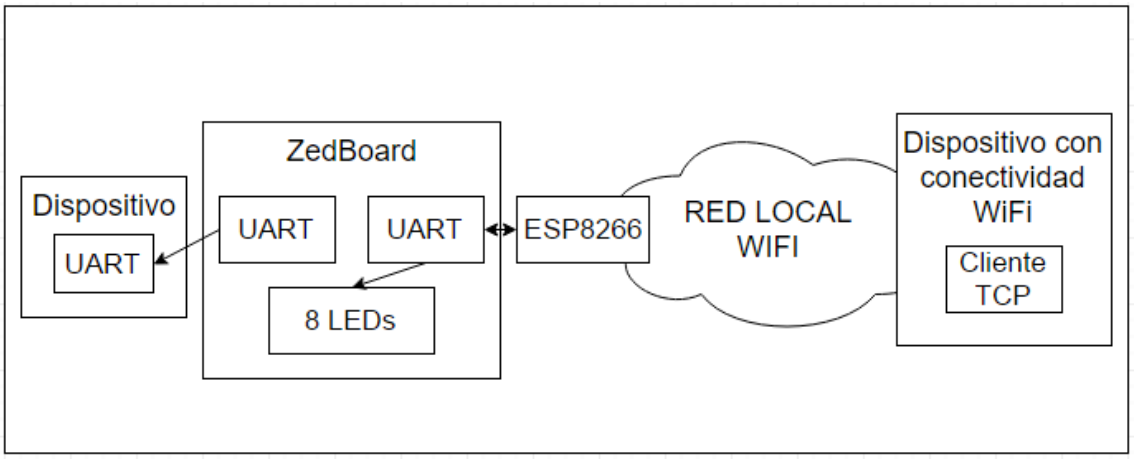

Figura 8.1: Esquema de aplicación desarrollada

8.1 Dispositivo ESP8266

El módulo ESP8266 [36], mostrado en la Figura 8.2, es un pequeño dispositivo de bajo coste que incluye una pila de protocolos TCP/IP completa y un microprocesador que permiten conectarse a una red WiFi mediante comandos de texto AT. Debido al diseño, el módulo puede funcionar de dos formas diferentes. Por un lado, es posible alojar la aplicación directamente en el módulo y, por otro lado, y tal y como se va a emplear en este trabajo, usarlo como adaptador WiFi para realizar comunicaciones de forma inalámbrica ejecutando la aplicación en otro dispositivo. Este módulo está diseñado únicamente para trabajar en la banda de frecuencias de 2,4 GHz empleando el protocolo 802.11 b/g/n. Una de las ventajas que tiene este dispositivo es su bajo

consumo, haciéndolo ideal para su empleo en sistemas móviles donde el gasto de energía está más limitado.

Para llevar a cabo la comunicación entre el módulo y la placa de desarrollo a la que se quiere dar conectividad inalámbrica, se emplea una UART que se puede configurar a diferentes velocidades.

El dispositivo ESP8266 puede trabajar de dos formas complementarias: en modo AP (*Access Point*) en el cual el dispositivo crea su propia red inalámbrica y en modo STA (*Station*) en el cual el módulo se conecta a una red WiFi ya existente.

En este trabajo, únicamente se va a emplear el método AP de forma que el dispositivo cree una red WiFi a la cual se pueda conectar cualquier dispositivo con este tipo de conectividad.

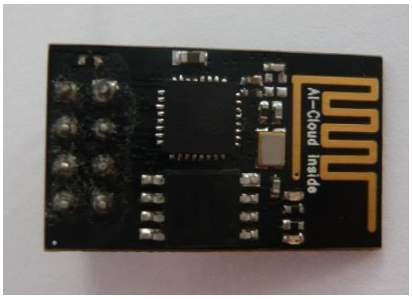

Figura 8.2: Módulo WiFi ESP8266

8.2 Diseño Hardware

Para llevar a cabo el diseño Hardware, se va a emplear la herramienta Vivado al igual que en los capítulos anteriores de este trabajo. Debido a que este diseño es muy similar al elaborado en dichos capítulos, únicamente se indicarán los aspectos más importantes para el correcto funcionamiento de la aplicación.

El primer paso, una vez creado el proyecto, es ubicar y configurar el *IP Core* que representa al dispositivo Zynq. De los controladores disponibles, únicamente se van a utilizar las dos UART disponibles configuradas a 115200 baudios.

La UART0 es la comunicación serie que se empleará para el intercambio de información con el módulo ESP8266. Para realizar el conexionado, compuesto únicamente de dos hilos, se emplearán dos pines de uno de los puertos Pmod conectados a la parte PL de la tarjeta ZedBoard. Por ello, es necesario elegir la interfaz EMIO en el momento de activar esta característica en el sistema Zynq.

La UART1 se empleará para enviar información a un dispositivo desde el cual se podrá controlar la diferente información intercambiada por el cliente TCP y la tarjeta ZedBoard. La UART1 está conectada con uno de los conectores micro-USB disponibles en la placa de desarrollo que se encuentra asociado a los pines MIO 48 y MIO 49.

En la configuración del bloque Zynq es necesario también la activación de *M AXI GP0 interface* que se empleará para la conexión con el siguiente de los bloques *IP Core* que se debe configurar.

Para el empleo de los 8 LEDs disponibles en la tarjeta ZedBoard, es necesario añadir y configurar el *IP Core* denominado *AXI_GPIO*. Una vez añadido dicho bloque, este se debe configurar seleccionando la opción *leds8_bits* en la interfaz *GPIO*.

Una vez añadidos y configurados los dos bloques anteriores, estos deben conectarse entre sí. Para ello, se va a emplear la herramienta *Run Connection Automation* que se encarga de interconectar los bloques que componen el sistema añadiendo todos los *IP Core* necesarios para ello.

Es importante destacar la necesidad de hacer externos los puertos de la UART0 conectada a la interfaz EMIO. Para ello, sobre cada uno de los puertos denominados *UART0_TX* y *UART0_RX*, se hace *click* derecho y se selecciona la opción *Make External*.

El diseño obtenido debe ser similar al mostrado en la Figura 8.3.

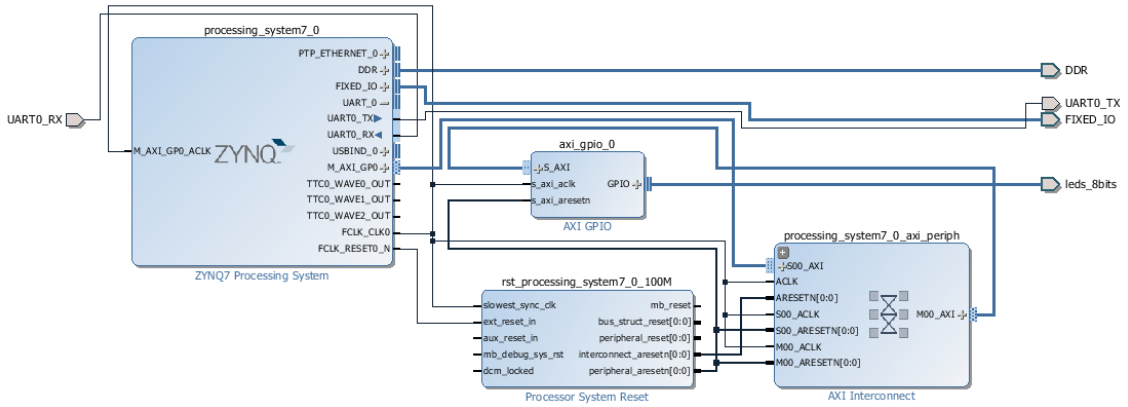

Figura 8.3: Diseño Hardware módulo ESP8266

Una vez realizado el diseño y antes de generar el archivo *Bitstream* encargado de programar la parte reconfigurable del dispositivo Zynq, es necesario configurar a que pines de la placa ZedBoard se va conectar el módulo WiFi. Esto, junto con la tensión y el tipo de resistencia del pin, se debe indicar en el archivo de *constraints*.

De esta forma, se va a configurar que el pin transmisor de la UART se corresponde con el pin JA2 perteneciente al puerto Pmod denominado JA. De igual forma, el pin receptor se corresponderá con el pin JA1. Según la información proporcionada por el fabricante, estos pines son denominados respectivamente AA11 y Y11 en la tarjeta ZedBoard.

Por último, ambas conexiones se establecerán con una tensión de 3,3 voltios y dispondrán de una resistencia de *pull-up*.

La configuración de las conexiones anteriormente descritas se puede observar en la Figura 8.4.

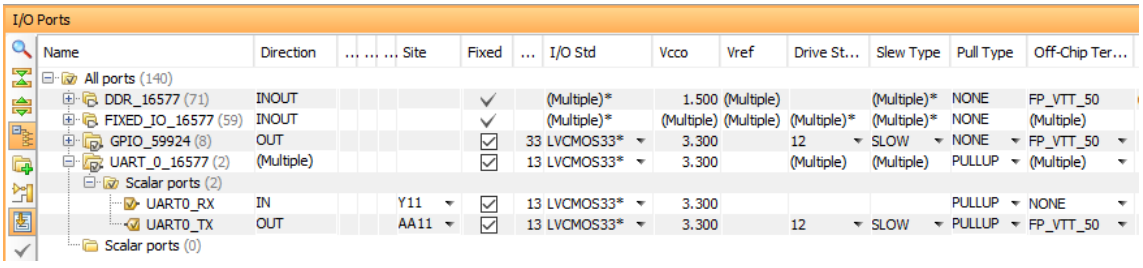

Figura 8.4: Configuración puertos UART

Una vez configurado el archivo de *constraints* se puede realizar la síntesis, implementación y generación del archivo *Bitstream*.

Una vez generado el archivo de configuración de la parte reconfigurable del dispositivo Zynq, se debe continuar con el diseño del sistema realizando la parte Software de la aplicación.

8.3 Diseño Software

Para llevar a cabo el diseño Software se va a emplear el software Xilinx SDK utilizado anteriormente. Sin embargo, en este capítulo la aplicación es *bare-metal* y, por tanto, no necesita un sistema operativo en el cual ejecutarse.

Para comenzar el diseño Software se parte del diseño Hardware explicado en el apartado anterior. Por ello, el primer paso es exportar dicho diseño para que pueda ser utilizado desde la herramienta Xilinx SDK. Una vez exportado, se ejecuta la herramienta y se comienza el diseño Software.

El primer paso es la creación del *Board Support Package* o BSP. Para ello, se selecciona la opción *New -> Board Support Package* y, tras indicar un nombre y el proyecto hardware asociado, se selecciona la opción *Finish*. En este caso, no es necesario añadir ninguna biblioteca adicional tal y como se hacía en el caso de los capítulos anteriores.

El último paso a dar en el diseño Software es la creación de la aplicación que cumpla los objetivos propuestos en este capítulo. Para ello, se selecciona la opción *File -> New -> Application Project*.

En la ventana que se abre, se indica un nombre, el lenguaje de programación y el empleo del BSP creado anteriormente, tal y como se muestra en la Figura 8.5.

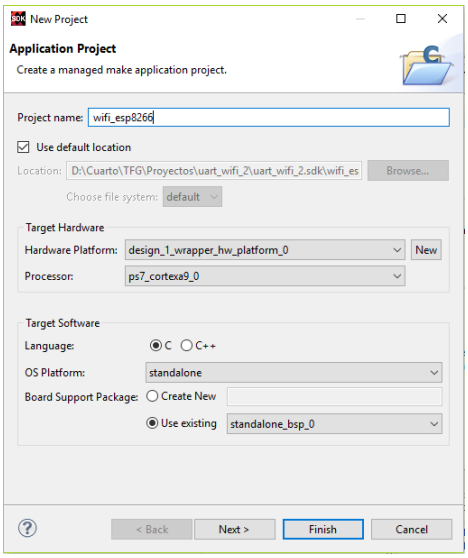

Figura 8.5: Creación aplicación

Tras seleccionar la opción *Next* se elige el *template* denominado *Empty Application* como se indica en la Figura 8.6. Esta opción crea una aplicación vacía a la cual se deben añadir los archivos fuente que definan el funcionamiento del sistema.

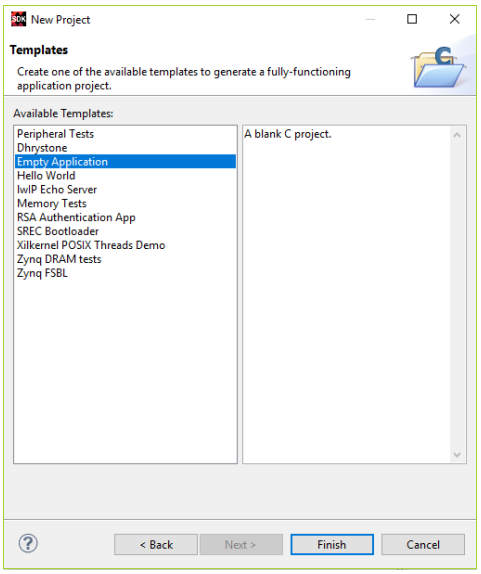

Figura 8.6: Elección template para aplicación

Para crear el archivo fuente, en el cual se añadirá el programa descrito en lenguaje C, se selecciona *File -> New -> Source File*. En la ventana que se abre, mostrada en la Figura 8.7, se debe asignar un nombre al archivo y su ubicación dentro del proyecto.

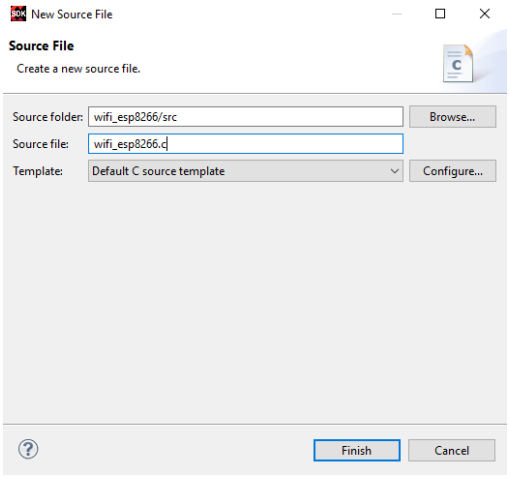

Figura 8.7: Creación archivo fuente

Para emplear los distintos elementos que forman el sistema, tales como UART0, UART1 o GPIO, se deben emplear los *drivers* disponibles en el BSP para cada uno de ellos. A través de las funciones que proporcionan dichos *drivers*, se puede interactuar con los distintos elementos de forma sencilla.

Para poder utilizar los LEDs disponibles en la tarjeta ZedBoard, el primer paso es inicializar la interfaz GPIO que los controla. Posteriormente, se debe indicar que son elementos actuadores, es decir, que el sentido de los datos es desde el procesador a los LEDs. Además, como inicialmente se desea que todos los LEDs se encuentren apagados, se emplea la función *XGpio_DiscreteWrite* pasando como parámetro el valor 0.

```
XGpio_Initialize(&leds, XPAR_GPIO_0_DEVICE_ID);
XGpio_SetDataDirection(&leds, 1, 0x0);
XGpio_DiscreteWrite(&leds, 1, 0x0);
```

De la misma forma, es necesario inicializar cada UART empleada en la aplicación de forma individual. En el siguiente fragmento de código se muestra únicamente una de las dos inicializaciones necesarias.

```
config_wifi = XUartPs_LookupConfig(XPAR_XUARTPS_0_DEVICE_ID);
    if(NULL == config_wifi)
    {
       return XST_FAILURE;
    }
    status = XUartPs_CfgInitialize(&uart_wifi, config_wifi, config_wifi-
>BaseAddress);
    if(status != XST_SUCCESS)
    {
       return XST_FAILURE;
    }
    status = XUartPs_SelfTest(&uart_wifi);
    if(status != XST_SUCCESS)
    {
       return XST_FAILURE;
    }
```

Una vez llevadas a cabo las inicializaciones anteriores, es posible emplear la comunicación UART con dos funciones.

La función *XUartPs_Send* se emplea para enviar datos a través de la interfaz UART. Como parámetros, esta función necesita conocer por cuál de las dos UART se desea enviar los datos, los propios datos y el número de bytes que estos ocupan.

La función empleada para leer los datos recibidos por la UART es *XUartPs_Recv*. Al igual que la función anterior, necesita conocer cuál es la UART que se desea leer, donde debe almacenar la información leída y cuantos bytes debe leer.

Un ejemplo de uso de las dos funciones anteriores se puede observar en el siguiente fragmento de código de la aplicación desarrollada.

```
XUartPs_Send(&uart_wifi, "Introduzca un numero de LED\n\r", 29);
ReceivedCount = 0;
while(ReceivedCount < BUFFER_SIZE)
{
ReceivedCount += XUartPs_Recv(&uart_wifi, &RecvBuffer[ReceivedCount],
(BUFFER_SIZE - ReceivedCount));
}
```

El código completo de la aplicación desarrollada para este capítulo puede observarse en el Anexo II.

8.4 Comprobación de funcionamiento

Para comprobar el funcionamiento del sistema, es necesario primero realizar todas las conexiones que comunican los distintos componentes. Posteriormente, se debe programar la FPGA incluida en el dispositivo Zynq con el archivo *Bitstream*. Finalmente, se podrá ejecutar la aplicación desarrollada sobre la tarjeta ZedBoard para verificar su funcionamiento.

8.4.1 Conexionado

Antes de poder comprobar el correcto funcionamiento del sistema desarrollado, es necesario realizar una serie de conexiones que permitan el intercambio de información entre los diferentes dispositivos que forman el sistema.

Por un lado, es necesario conectar la ZedBoard a través de un puerto JTAG micro-USB por el cual se llevará a cabo la programación de la placa.

Además, desde otro de los puertos micro-USB se llevará a cabo la comunicación UART con un ordenador que mostrará estos mensajes a través de un programa que permita la conexión serie como, por ejemplo, TeraTerm.

Por último, en los dos primeros pines del puerto Pmod JB se deben conectar los dos hilos que necesita el módulo WiFi para llevar a cabo la comunicación con el controlador UART de Zynq. Según el diseño realizado en este trabajo, en el pin denominado JA2 conectado al hilo transmisor de la UART de Zynq, se debe conectar el cable receptor del módulo WiFi. De igual forma, el pin JA1 conectado internamente al receptor UART de Zynq, se debe conectar al pin transmisor del módulo ESP8266.

El módulo ESP8266 necesita ser conectado, a través de dos pines diferentes, a una tensión continua de 3,3 voltios así como a masa. En el mismo puerto Pmod al que se ha conectado el módulo WiFi, están disponibles estos valores de tensión que pueden ser empleados para alimentarlo. En la Figura 8.8 se pueden observar todas las conexiones de las que dispone el módulo ESP8266 y cuáles de ellas deben ser conectadas para esta aplicación.

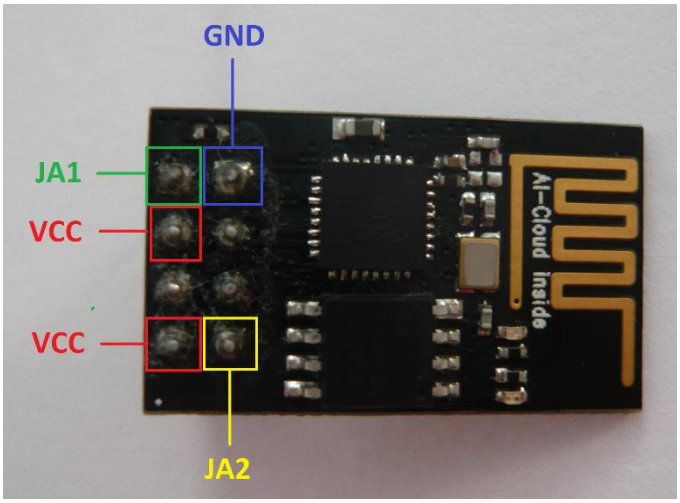

Figura 8.8: Conexionado módulo ESP8266

Antes de accionar el interruptor de encendido de la tarjeta ZedBoard, se deben conectar los *jumpers* JP8, JP9 y JP10 según como se muestra en la Figura 8.9, de forma que el arranque del sistema tenga lugar a través del puerto JTAG.

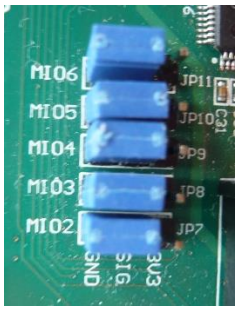

Figura 8.9: Colocación jumpers en aplicación bare-metal

Una vez realizadas todas las conexiones anteriores, mostradas en la Figura 8.10, se puede proceder a encender la tarjeta ZedBoard.

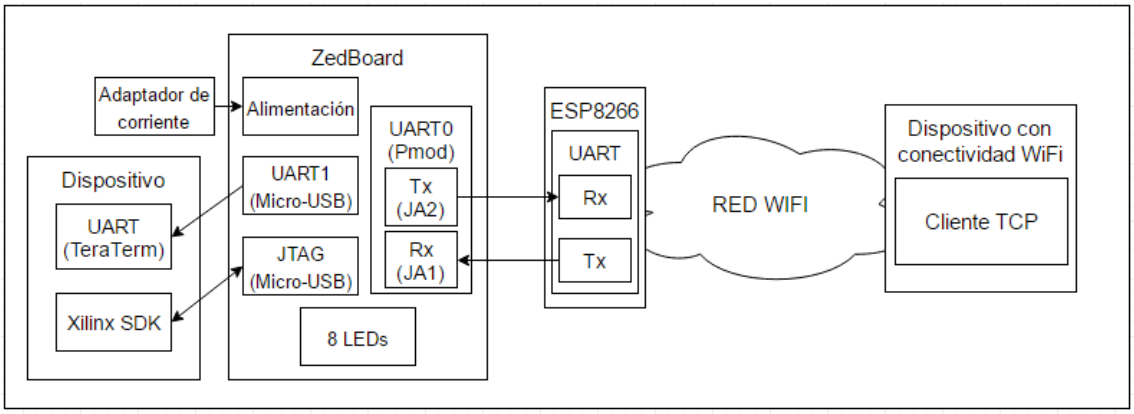

Figura 8.10: Conexionado aplicación ESP8266

8.4.2 Programación FPGA

El primer paso tras el encendido de la tarjeta de desarrollo es la programación de la parte reconfigurable del dispositivo Zynq. Para ello, se emplea una herramienta denominada *Program FPGA*, mostrada en la Figura 8.11, disponible en el software Xilinx SDK en el menú *Xilinx Tools*. Esta herramienta se encarga de cargar el archivo *Bitstream* generado en Vivado en la tarjeta de desarrollo. Cuando el proceso de programación de la FPGA ha concluido, se indica al usuario con el encendido del LED *Done* disponible en la tarjeta ZedBoard.

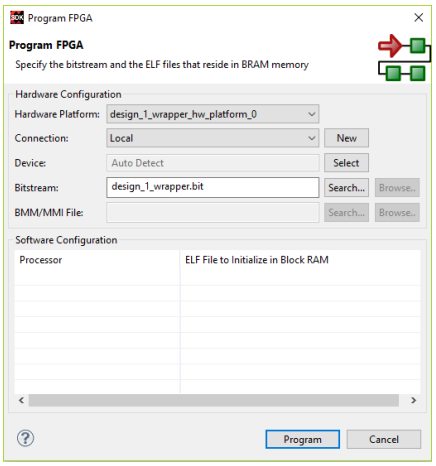

Figura 8.11: Herramienta para la programación de la FPGA

8.4.3 Ejecución de la aplicación desarrollada

Antes de ejecutar la aplicación, un dispositivo con conectividad WiFi y con un cliente TCP instalado, debe conectarse a la red creada por el módulo ESP8266. A través de este cliente, el usuario podrá interactuar con la aplicación indicando cual es el diodo LED que desea encender.

Por otra parte, en el ordenador conectado por cable se debe ejecutar un programa como TeraTerm configurado a 115200 baudios. A través de este software, se recibirán los distintos mensajes que envíe la aplicación por la UART1.

El último paso es la ejecución, sobre la tarjeta ZedBoard, de la aplicación desarrollada. Para ello, únicamente es necesario seleccionar la opción *Run -> Run As -> Launch on Hardware*.

Una vez se ha terminado de cargar completamente sobre la tarjeta ZedBoard, la aplicación se ejecuta de forma automática mandando un mensaje de texto a través de las dos conexiones UART.

Desde el terminal inalámbrico, el usuario podrá indicar numéricamente el LED que desea encender. Tras ello, el sistema le preguntará si desea introducir otro número o finalizar la ejecución de la aplicación.

Un ejemplo de ejecución de la aplicación se puede observar en la Figura 8.12.

Figura 8.12: Ejecución aplicación ESP8266

Capítulo 9: Resultados obtenidos

En este capítulo se muestran los resultados obtenidos en relación a la velocidad de transferencia lograda para cada uno de los dos dispositivos empleados en este trabajo.

9.1 Velocidad de transferencia del módulo WiLink8

Para comprobar el comportamiento del módulo WiLink8 se han realizado diferentes transferencias a través de distintas aplicaciones y protocolos.

Para poder medir la velocidad de transferencia del módulo WiFi, en todos los casos mostrados a continuación se ha realizado la transferencia de un mismo archivo de tamaño 100 MB. Para comprobar el comportamiento del dispositivo se han realizado distintas pruebas que permiten evaluar la velocidad de transferencia obtenida para diferentes protocolos.

9.1.1 ZedBoard conectada a red WiFi existente

Las primeras pruebas realizadas se han llevado a cabo conectando una tarjeta de desarrollo ZedBoard a una red local WiFi tal y como se puede apreciar en el esquema mostrado en la Figura 9.1.

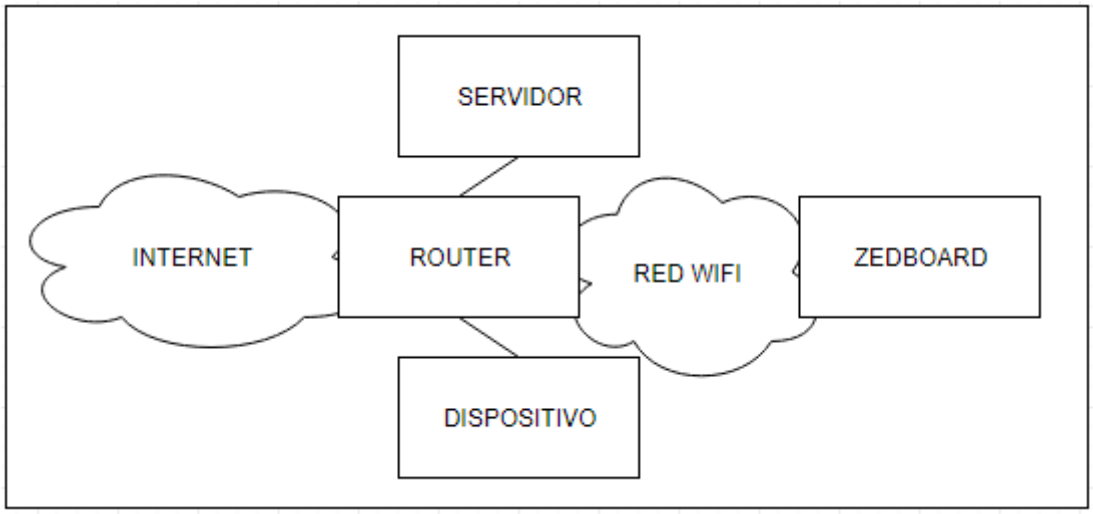

Figura 9.1: ZedBoard actuando como cliente en red WiFi

La primera transferencia realizada, mostrada en la Figura 9.2, ha consistido en la descarga desde un servidor en Internet de un archivo de tamaño 100 MB. En este caso, la velocidad obtenida en la descarga ha sido de 15,38 Mbps. Esta velocidad está condicionada por el retardo que pueden sufrir los paquetes por la red. Este retardo dependerá principalmente de la congestión que la red soporte en el momento de la descarga.

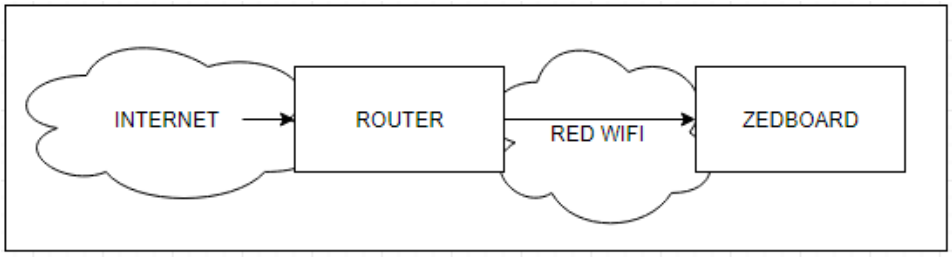

Figura 9.2: Transferencia desde servidor en Internet

Para comprobar la velocidad de transferencia empleando el protocolo TCP pero sin influencia de la velocidad de conexión a Internet, se ha descargado el archivo desde un servidor ubicado en la misma red local y conectado por medio de un cable Ethernet, tal y como se puede observar en la Figura 9.3. En este caso, la velocidad obtenida ha sido de 15,69 Mbps. Esta velocidad depende únicamente de los dos dispositivos inalámbricos que permiten la comunicación sin que sea condicionada por factores externos como en el apartado anterior.

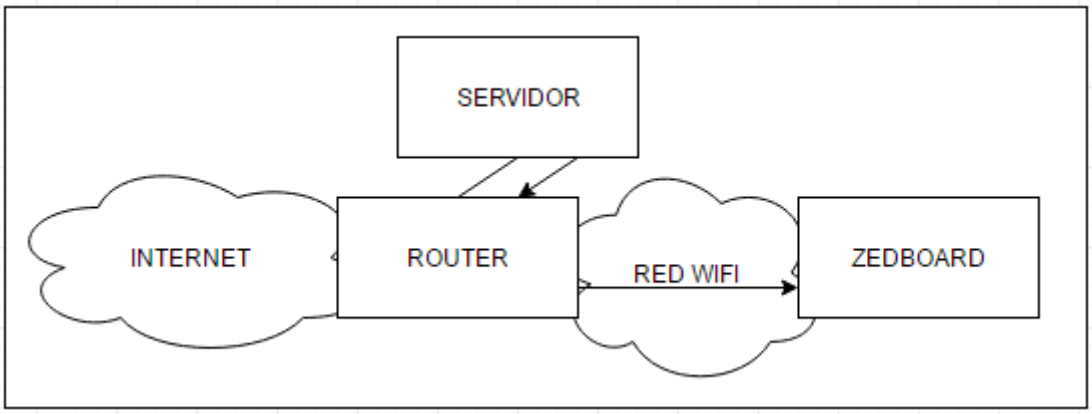

Figura 9.3: Transferencia desde servidor ubicado en red local

La siguiente prueba realizada a consistido en la transferencia del mismo archivo empleando el protocolo *scp*. Para ello, desde un ordenador conectado a la misma red local a través de un cable Ethernet se ha enviado el archivo a la tarjeta ZedBoard, tal y como puede observarse en la Figura 9.4. La velocidad obtenida en este caso ha sido de 7,21 Mbps.

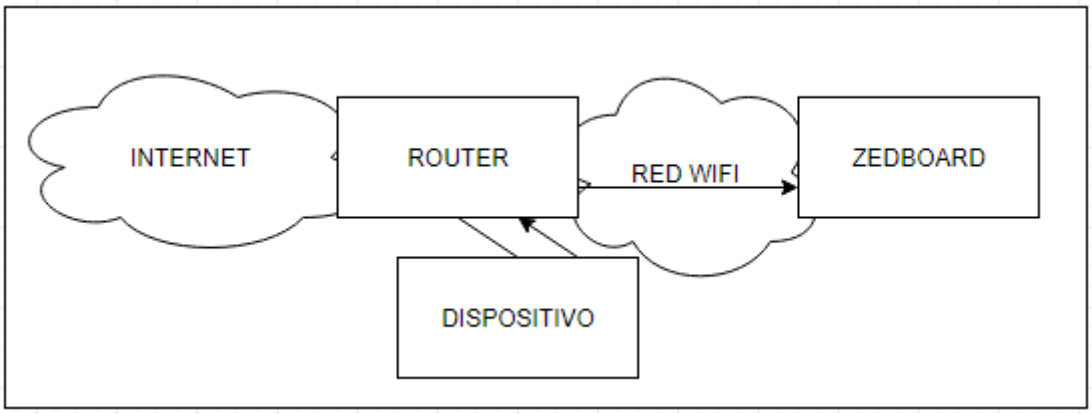

Figura 9.4: Transferencia scp

En la tabla siguiente se muestra la velocidad obtenida en las tres transferencias realizadas en este apartado.

Transferencia	Velocidad de transferencia
Servidor Internet -> Router -> ZedBoard (TCP)	15,38 Mbps
Servidor red local -> Router -> ZedBoard (TCP)	15,69 Mbps
Dispositivo local -> Router -> ZedBoard (SCP)	7,21 Mbps

Cabe destacar que en las tres pruebas realizadas anteriormente, la velocidad de transferencia se ha podido ver limitada por la velocidad máxima de la red local WiFi a la que se ha conectado

la tarjeta ZedBoard. Por ello, en los apartados posteriores se emplearán dos tarjetas ZedBoard donde cada una de ellas dispone de un módulo de comunicaciones inalámbricas WiLink8.

9.1.2 ZedBoard actuando como AP

En este apartado, se emplearán dos tarjetas ZedBoard equipadas con el módulo de comunicaciones inalámbricas WiLink8. Una de las ZedBoard generará una red local WiFi a la cual se conectará la segunda de las tarjetas ZedBoard tal y como se muestra en la Figura 9.5. De esta forma, la transferencia se realizará dependiendo únicamente de la velocidad de transferencia del módulo en estudio, no viéndose influido por elementos externos como el router WiFi empleado en el apartado anterior.

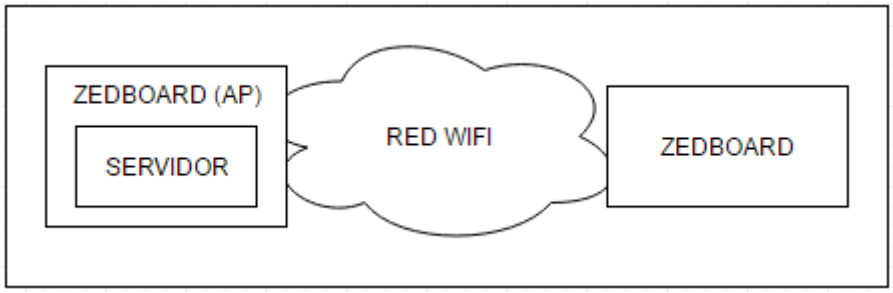

Figura 9.5: ZedBoard actuando como AP

La tarjeta ZedBoard que genera la red local inalámbrica dispone de un servidor en el cual se ubica el archivo de 100 MB que se está empleando para realizar todas las transferencias de archivos. Desde la ZedBoard conectada a la red local se descarga dicho archivo empleando para ello el protocolo TCP. La velocidad de transferencia obtenida en este caso ha sido de 38,1 Mbps.

En la siguiente prueba llevada a cabo se ha realizado el envío del archivo empleando la aplicación *scp* actuando una de las tarjetas Zedboard como servidor y la otra como cliente. En este caso, la velocidad de transferencia obtenida ha sido de 4,44 Mbps.

Transferencia	Velocidad de transferencia
Servidor ZedBoard AP -> ZedBoard (TCP)	38,1 Mbps
Zedboard AP -> ZedBoard (SCP)	4,44 Mbps

En este capítulo se ha estudiado el comportamiento de los módulos WiFi siendo estos empleados como generadores de la red local o como clientes de una red. Sin embargo, los módulos WiLink8 pueden actuar simultáneamente de ambas formas.

9.1.3 ZedBoard conectada a red WiFi existente y como AP

En este apartado, una de las tarjetas ZedBoard generará una red local inalámbrica al mismo tiempo que está conectada a una red WiFi externa. La segunda de las tarjetas ZedBoard únicamente estará conectada a la red local creada por la otra ZedBoard. De esta forma, se puede estudiar si varía el comportamiento de los módulos cuando estos son empleados de forma simultánea como AP y clientes en una red local.

El esquema de red empleado en este apartado se representa en la Figura 9.6.

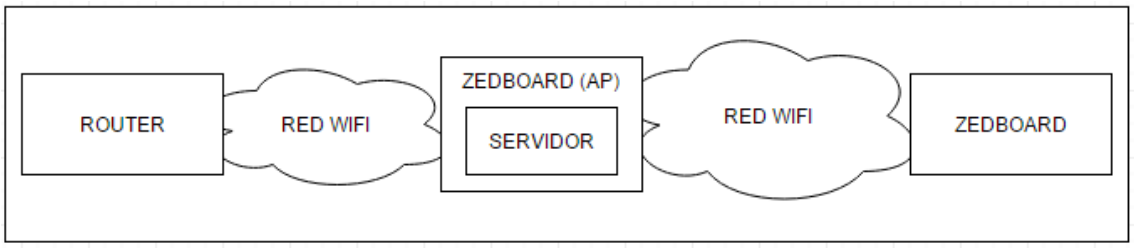

Figura 9.6: ZedBoard actuando como AP y como cliente

En este caso se desea conocer el impacto en la velocidad de transferencia empleando TCP cuando la tarjeta ZedBoard conectada a una red local externa no realiza ninguna transferencia sobre dicha red. De esta forma, únicamente está establecida la conexión pero no se lleva a cabo ningún envío por medio de la red WiFi externa. En este caso, la velocidad de transferencia obtenida es de 33,33 Mbps siendo ésta ligeramente inferior a la obtenida anteriormente.

Para comprobar la influencia del empleo simultaneo del módulo WiFi como AP y cliente de una red local externa, se va a realizar el envío del archivo a la segunda ZedBoard empleado TCP mientras se realiza un uso intensivo de la conexión con la red externa. En este caso, la velocidad de transferencia obtenida ha sido de 17,02 Mbps. De esta forma, se puede observar que la velocidad obtenida se ha reducido a la mitad comparada con el resultado anterior.

Transferencia	Velocidad de transferencia
Servidor ZedBoard AP + STA -> ZedBoard (TCP)	33,33 Mbps
Servidor ZedBoard AP + STA (uso intensivo) -> ZedBoard (TCP)	17,02 Mbps

Para comparar los resultados de una forma más sencilla, a continuación se muestran todas las velocidades obtenidas en las pruebas anteriores.

En la siguiente tabla se muestran los resultados obtenidos empleando el protocolo TCP, ordenados según la velocidad de transferencia lograda.

Transferencia	Velocidad de transferencia
Servidor Internet -> Router -> ZedBoard (TCP)	15,38 Mbps
Servidor red local -> Router -> ZedBoard (TCP)	15,69 Mbps
Servidor ZedBoard AP + STA (uso intensivo) -> ZedBoard (TCP)	17,02 Mbps
Servidor ZedBoard AP + STA -> ZedBoard (TCP)	33,33 Mbps
Servidor ZedBoard AP -> ZedBoard (TCP)	38,1 Mbps

Se puede observar que las pruebas realizadas en el apartado 9.1.1 obtienen velocidades similares entre sí pero más reducidas que en el resto de transferencias. Por tanto, se puede concluir que en dichas pruebas el router es el dispositivo que limita la velocidad de transferencia.

En el resto de las pruebas realizadas empleando el protocolo TCP, se puede observar que el empleo del módulo WiLink8 de forma simultánea como AP y como cliente solo afecta a la velocidad cuando se emplean activamente ambas conexiones.

Por último, cabe destacar que la mayor velocidad obtenida empleando el módulo WiLink8 y el protocolo TCP es de 38,1 Mbps.

En la siguiente tabla se recogen los resultados obtenidos empleando el protocolo SCP.

Transferencia SCP	Velocidad de transferencia
Zedboard AP -> ZedBoard (SCP)	4,44 Mbps
Dispositivo local -> Router -> ZedBoard (SCP)	7,21 Mbps

En el caso del protocolo SCP la velocidad obtenida es sustancialmente menor a la obtenida empleando el protocolo TCP. Esto es debido a la mayor cantidad de cabeceras que emplea el protocolo SCP para lograr seguridad e integridad en la transferencia.

Por último, cabe destacar que en este caso la velocidad obtenida es mayor en el caso de la transferencia desde un ordenador a una tarjeta ZedBoard que en la transferencia entre dos tarjetas ZedBoard. Esto difiere con los resultados obtenidos para el protocolo TCP y puede ser debido a la mayor capacidad de procesamiento del ordenador que envía el archivo frente a la capacidad de procesamiento de una tarjeta ZedBoard.

9.2 Velocidad de transferencia del módulo ESP8266

Para conocer la velocidad de transferencia del módulo ESP8266, se ha realizado un programa que envía una cierta cantidad de datos por la UART conectada al dispositivo WiFi. Para ello, un dispostivo se conecta a la red inalámbrica generada por el módulo WiFi y con un cliente TCP, se observa cuanto se tarda en recibir todos los datos enviados. En la Figura 9.7 se muestra el esquema de red que se va a utilizar en este apartado.

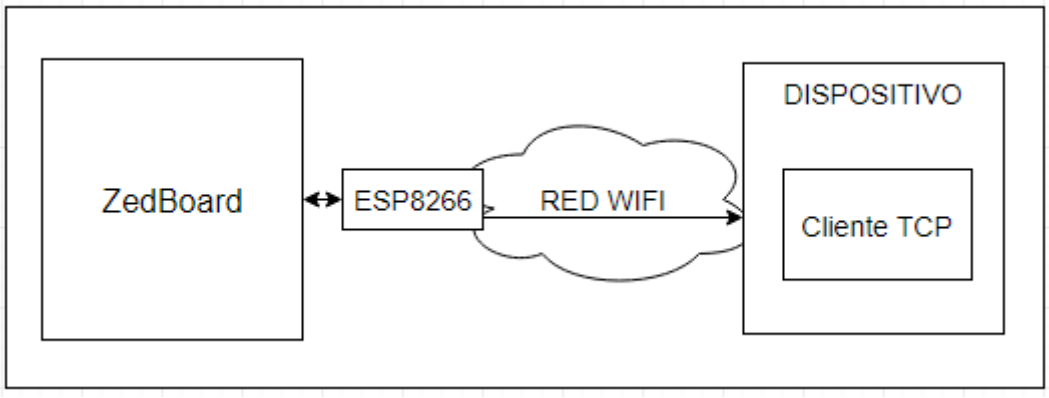

Figura 9.7: Transferencia con módulo ESP8266

Se han realizado envíos de diferentes cantidades de datos, desde pocos bytes a 1 MB, de forma que se pudiera comprobar el comportamiento del dispositivo en diferentes situaciones. En todos los casos se han contabilizado unos tiempos proporcionales por lo que se considera que la velocidad obtenida es prácticamente estable para cualquier cantidad de datos enviados.

La velocidad de transferencia conseguida por el módulo ESP8266 ha sido de 93 Kbps. Cabe destacar que la UART estaba configurada para realizar la transferencia a 115200 baudios por lo que, teniendo en cuenta que es necesario añadir a los datos recibidos las cabeceras necesarias propias del protocolo WiFi, la velocidad obtenida no se encuentra muy por debajo de la velocidad máxima ideal.

Como se ha podido observar, la velocidad máxima conseguida con el módulo ESP8266 (93 Kbps) es muy inferior a la velocidad de transferencia obtenida con el módulo WiLink8 (38 Mbps). Este resultado es el esperado debido a las diferentes características y capacidades con las que han sido diseñados. Además, los módulos emplean diferentes formas de comunicación con la tarjeta

ZedBoard. Así, el módulo ESP8266 emplea una comunicación UART con dos únicos hilos mientras que el módulo WiLink8 utiliza una comunicación SDIO de 4 hilos.

Capítulo 10: Conclusiones y líneas futuras

10.1 Conclusiones

En la realización de este trabajo, se han obtenido una serie de conclusiones y resultados que se resumen a continuación:

- Se ha señalado la gran importancia para la sociedad de hoy en día de los sistemas empotrados, ya que estos están presentes en todos los elementos que conforman esta sociedad. Además, se ha comentado la relevancia de los sistemas *System on Chip* que incluyen dentro de un único chip todos los componentes que estos necesitan para realizar todas las funciones para lo que han sido diseñados.
- Se ha presentado el dispositivo Zynq, un *SoC* compuesto por dos partes claramente independientes: un sistema de procesamiento y un sistema reconfigurable. Además, se ha detallado las herramientas y el procedimiento dual que se debe seguir para llevar a cabo su programación.
- Se ha caracterizado la tarjeta de desarrollo ZedBoard, que incluye un dispositivo Zynq, sobre la que se han implementado todos los diseños realizados a lo largo de este trabajo. De forma específica, se han comentado los diferentes elementos de los que dispone esta tarjeta para su conectividad con otros dispositivos y sistemas.
- Se han analizado algunas de las tecnologías inalámbricas más utilizadas para su posible implementación en la tarjeta de desarrollo ZedBoard y se ha implementado la tecnología WiFi en la tarjeta ZedBoard. Para ello, y tras conocer algunas de las posibilidades para llevar a cabo esta implementación, se ha elegido un módulo WiLink8 de TI.
- Se ha explicado detalladamente todos los pasos que se deben dar para lograr un correcto funcionamiento de dicho módulo, desde el diseño Hardware hasta la configuración de un sistema operativo que se ejecuta sobre la tarjeta ZedBoard.
- Se han diseñado un conjunto de aplicaciones que ejemplifican algunos de los posibles usos de la conectividad WiFi, como el control inalámbrico de algunos de los elementos que incluye la tarjeta de desarrollo, el control remoto del sistema con el empleo del protocolo SSH o el empleo de un servidor web.
- Para comparar el comportamiento del módulo WiLink8 con otros dispostivios WiFi disponibles en el mercado, se ha empleado un módulo WiFi de bajo coste ESP8266 para el diseño de una aplicación *bare-metal* que permite el encendido de los LEDs disponibles en la tarjeta ZedBoard de forma inalámbrica.
- Se ha realizado un estudio de la velocidad de transferencia máxima obtenida empleando diferentes protocolos y configuraciones de los módulos estudiados. Para el módulo WiLink8 se ha conseguido una velocidad máxima de transferencia de 38,1 Mbps empleando el protocolo TCP y de 7,21 Mbps emplenado el protocolo SCP. Para el módulo ESP8266 se ha obtenido una velocidad máxima de 93 Kbps.

10.2 Futuras líneas de investigación

A lo largo de este trabajo, se ha implementado una única tecnología inalámbrica en la tarjeta de desarrollo ZedBoard. Sin embargo, existen multitud de tecnologías inalámbricas disponibles en el mercado que tienen diferentes características y, por tanto, son adecuadas para su uso en

distintas aplicaciones. Por ello, se propone la implementación de algunas de estas tecnologías como Bluetooth, infrarrojos, ZigBee o las redes móviles como 2G y 3G, entre otras muchas.

En relación al sistema operativo empleado, se ha usado PetaLinux pero existen otras distribuciones, basadas igualmente en Linux, como *Zynq-Linux*, *Xillinux* o *Android*, cada una con diferentes características, que también pueden ser empleadas en un sistema desarrollado en la tarjeta ZedBoard.

Por último, el empleo de una conectividad inalámbrica permite el desarrollo de infinidad de aplicaciones que hagan uso de ella. Por ejemplo, una posible aplicación es el desarrollo de un sistema de vigilancia formado por una cámara que capta imágenes de forma periódica y, posteriormente, estas son enviadas, de forma inalámbrica a través de una red WiFi, a otro sistema desde el cual el usuario puede visualizarlas.

Pliego de condiciones

Para la realización de las distintas aplicaciones desarrolladas a lo largo de este trabajo, se han empleado una serie de equipos y dispositivos así como un conjunto de herramientas software.

Recursos hardware

- Ordenador portátil Asus F555L
  - Procesador Intel Core i7-5500U 2,4 GHz
  - 8 GB de memoria RAM
- 2x Tarjeta de desarrollo ZedBoard
- 2x Módulo WiFi WL1835MOD WiLink8 de TI
- 2x Adaptador WiLink8 Pmod
- Módulo WiFi ESP8266
- Cables: Ethernet y USB a USB-micro

Recursos software

- Sistema operativo Windows 10 de 64-bits
- Xilinx Vivado Design Suite
- VMware Workstation Pro
- Sistema operativo Ubuntu 16.04.2 de 64-bits
- PetaLinux SDK
- TeraTerm
- Cliente TCP
- Suite Microsoft Office

Presupuesto

El presupuesto de este trabajo se desgrana en varios apartados mostrados a continuación.

Recursos hardware

En la tabla mostrada a continuación se resume el presupuesto necesario en relación a dispositivos y equipos necesarios para el desarrollo de este trabajo.

Material	Precio unidad	Unidades	Precio total	Amortización	Tiempo de uso	Subtotal
Ordenador	800 €	1	800 €	3 años	5 meses	111,11 €
ZedBoard	475 €	2	950 €	-	-	950 €
Módulo WL1835MOD	19,02 €	2	38,04 €	-	-	38,04 €
Adaptador WiLink8 Pmod	47,50 €	2	95 €	-	-	95 €
Módulo ESP8266	8 €	1	8 €	-	-	8 €
Total						252,15 €

Recursos software

El presupuesto necesario en concepto de software se puede observar en la tabla mostrada a continuación.

Software	Precio	Amortización	Tiempo de uso	Subtotal
Xilinx Vivado Design Suite	2995 €	3 años	5 meses	416 €
Suite Microsoft Office	149 €	3 años	5 meses	20,69c€
Total				436,69 €

Mano de obra

En este apartado se muestra, resumido en una tabla, el presupuesto destinado a la mano de obra en la realización de este trabajo.

Tarea	Salario (Euros/hora)	Horas	Subtotal
Diseño del sistema	25 €/hora	270	6750 €
Redacción del libro	8 €/hora	180	1440 €
Total			8190 €

Coste total

El coste total será la suma de los tres apartados anteriores al cual se debe añadir el 21% en concepto de IVA.

Concepto	Total
Recursos hardware	252,15 €
Recursos software	436,69 €
Mano de obra	8190 €
Coste total	**8878,84 €**
IVA (21%)	1864,56 €
Importe total	**10743,40 €**

Bibliografía

[1] Xilinx, Zynq-7000 All Programmable Soc Technical Reference Manual.

[2] Xilinx, «PetaLinux SDK,» [En línea]. Disponible: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[3] Linux Foundation, «Device Tree,» [En línea]. Disponible: https://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf.

[4] ZedBoard.org, «ZedBoard Hardware User's Guide,» [En línea]. Disponible: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[5] Cypress, «Cypress.com,» [En línea]. Disponible: http://www.cypress.com/file/139421/download.

[6] Texas Instruments, «TI TUSB1210,» [En línea]. Disponible: http://www.ti.com/lit/ds/symlink/tusb1210.pdf.

[7] Marvell, «Marvell 88E1518,» [En línea]. Disponible: http://www.marvell.com/transceivers/assets/Marvell_Alaska_88E1510_18-002_product_brief.pdf.

[8] Digilent, «Digilent Pmod,» [En línea]. Disponible: https://reference.digilentinc.com/reference/pmod/specification?redirect=1.

[9] Analog Devices, «ADV7511 Analog Devices,» [En línea]. Disponible: http://www.analog.com/media/en/technical-documentation/user-guides/ADV7511_Hardware_Users_Guide.pdf.

[10] Analog Devices, «ADAU1761 Analog Devices,» [En línea]. Disponible: http://www.analog.com/media/en/technical-documentation/data-sheets/ADAU1761.pdf.

[11] ZigBee Alliance, «ZigBee Alliance,» [En línea]. Disponible: www.zigbee.org.

[12] Bluetooth Alliance, «Bluetooth Technology,» [En línea]. Disponible: https://www.bluetooth.com.

[13] WiFi Alliance, «WiFi Alliance,» [En línea]. Disponible: wi-fi.org.

[14] Microchip, «MRF24WG0MA Microchip,» [En línea]. Disponible: http://ww1.microchip.com/downloads/en/DeviceDoc/70686B.pdf.

[15] Digilent, «Pmod WiFi SPI,» [En línea]. Disponible: https://reference.digilentinc.com/doku.php?id=pmod/pmod/wifi/old_ref_manual.

[16] D-Link, «DWA-140 Wireless Adaptor,» [En línea]. Disponible: http://www.dlink.com/es/es/products/dwa-140-wireless-n-usb-adapter.

[17] Texas Instruments, «WiLink8,» [En línea]. Disponible: www.ti.com/lit/ds/symlink/wl1835mod.pdf.

[18] Texas Instruments, «WL18xx Overview,» [En línea]. Disponible: http://processors.wiki.ti.com/index.php/WL18xx_Overview.

[19] Texas Instruments, «WL1835MOD,» [En línea]. Disponible: http://www.ti.com/lit/ug/swru359e/swru359e.pdf.

[20] Texas Instruments, «WiLink8 WLAN Features,» [En línea]. Disponible: http://www.ti.com/lit/ug/swru423a/swru423a.pdf.

[21] Avnet, «WiLink 8 Pmod Adaptor,» [En línea]. Disponible: http://zedboard.org/sites/default/files/documentations/WiLink8_HW_UG_1.0.pdf.

[22] Texas Instruments, «TXS0108E,» [En línea]. Disponible: http://www.ti.com/lit/ds/symlink/txs0108e.pdf.

[23] Texas Instruments, «TPS736,» [En línea]. Disponible: http://www.ti.com/lit/ds/sbvs038u/sbvs038u.pdf.

[24] Ubuntu, «Ubuntu,» [En línea]. Disponible: https://www.ubuntu.com.

[25] Xilinx, «PetaLinux SDK 2014.4,» [En línea]. Disponible: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/archive.html.

[26] Xilinx, «PetaLinux Installation Guide,» [En línea]. Disponible: https://www.xilinx.com/support/documentation/sw_manuals/petalinux2014_2/ug976-petalinux-installation.pdf.

[27] Texas Instruments, [En línea]. Disponible: http://processors.wiki.ti.com/index.php/WL18xx_System_Build_Scripts.

[28] GParted. [En línea]. Disponible: gparted.org.

[29] TeraTerm. [En línea]. Disponible: https://ttssh2.osdn.jp/index.html.en.

[30] SSH, «SSH Protocol,» [En línea]. Disponible: https://www.ssh.com/ssh/protocol.

[31] Xilinx, «AXI4-Lite IP Interface,» [En línea]. Disponible: https://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif/v3_0/pg155-axi-lite-ipif.pdf.

[32] L. Torvalds, «Kernel Linux,» [En línea]. Disponible: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/kernel-parameters.txt?id=refs/tags/v3.17-rc1 .

[33] Boa.org, «boa-0.94.13,» [En línea]. Disponible: www.boa.org/boa-0.94.13.tar.gz.

[34] Xilinx, «mime.types,» [En línea]. Disponible: https://www.xilinx.com/Attachment/mime.types.

[35] R. Rivest, A. Shamir y L. Adleman, «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,» 1977.

[36] «ESP8266 datasheet,» [En línea]. Disponible: http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf.

Anexos

Anexo I

En este anexo, se muestra el código completo que forma el archivo CGI empleado en el capítulo 6.

```
#!/bin/sh
echo -e "Content-type: text/html\r\n\r\n"
echo ""
if [ ! -d /sys/class/gpio/gpio7 ];
then
       echo 7 > /sys/class/gpio/export
       echo out > /sys/class/gpio/gpio7/direction

       echo 240 > /sys/class/gpio/export
       echo 241 > /sys/class/gpio/export
       echo 242 > /sys/class/gpio/export
       echo 243 > /sys/class/gpio/export
       echo 244 > /sys/class/gpio/export
       echo 245 > /sys/class/gpio/export
       echo 246 > /sys/class/gpio/export
       echo 247 > /sys/class/gpio/export

       echo 248 > /sys/class/gpio/export
       echo 249 > /sys/class/gpio/export
       echo 250 > /sys/class/gpio/export
       echo 251 > /sys/class/gpio/export
       echo 252 > /sys/class/gpio/export
       echo 253 > /sys/class/gpio/export
       echo 254 > /sys/class/gpio/export
       echo 255 > /sys/class/gpio/export

       echo out > /sys/class/gpio/gpio248/direction
       echo out > /sys/class/gpio/gpio249/direction
       echo out > /sys/class/gpio/gpio250/direction
       echo out > /sys/class/gpio/gpio251/direction
       echo out > /sys/class/gpio/gpio252/direction
       echo out > /sys/class/gpio/gpio253/direction
       echo out > /sys/class/gpio/gpio254/direction
       echo out > /sys/class/gpio/gpio255/direction
fi

LED=`echo $QUERY_STRING | sed -n 's/^.*led=\([^&]*\).*$/\1/p' | sed "s/%20/ /g"`

LED0=`echo $QUERY_STRING | sed -n 's/^.*led0=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED1=`echo $QUERY_STRING | sed -n 's/^.*led1=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED2=`echo $QUERY_STRING | sed -n 's/^.*led2=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED3=`echo $QUERY_STRING | sed -n 's/^.*led3=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED4=`echo $QUERY_STRING | sed -n 's/^.*led4=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED5=`echo $QUERY_STRING | sed -n 's/^.*led5=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED6=`echo $QUERY_STRING | sed -n 's/^.*led6=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
LED7=`echo $QUERY_STRING | sed -n 's/^.*led7=\([^&]*\).*$/\1/p' | sed "s/%20/
/g"`
```

```
echo "<html><head><title>Control elementos ZedBoard desde servidor Web</title></head><body>"
echo "<center><font size="20">Servidor Web para el control de elementos de la placa ZedBoard</font></center><br>"

echo "<form action="/cgi_bin/control.cgi" method="get">"

echo "<br>"
echo "CONTROL LED PS<br>"
if [ $LED = "encender" ];
then
      echo 1 > /sys/class/gpio/gpio7/value
      echo "<input type="radio" name="led" value="encender" checked="checked"><font size="5">Encendido</font><br>"
      echo "<input type="radio" name="led" value="apagar"><font size="5">Apagado</font><br>"
else
      echo 0 > /sys/class/gpio/gpio7/value
      echo "<input type="radio" name="led" value="encender"><font size="5">Encendido</font><br>"
      echo "<input type="radio" name="led" value="apagar" checked="checked"><font size="5">Apagado</font><br>"
fi

echo "<br>"
echo "<br>"

echo "CONTROL LEDS PL<br>"
if [ $LED0 = "led0" ];
then
      echo 1 > /sys/class/gpio/gpio248/value
      echo "<input type="checkbox" name="led0" value="led0" checked="checked"><font size="5">LED 0</font><br>"
else
      echo 0 > /sys/class/gpio/gpio248/value
      echo "<input type="checkbox" name="led0" value="led0" <font size="5">LED 0</font><br>"
fi

if [ $LED1 = "led1" ];
then
      echo 1 > /sys/class/gpio/gpio249/value
      echo "<input type="checkbox" name="led1" value="led1" checked="checked"><font size="5">LED 1</font><br>"
else
      echo 0 > /sys/class/gpio/gpio249/value
      echo "<input type="checkbox" name="led1" value="led1" <font size="5">LED 1</font><br>"
fi

if [ $LED2 = "led2" ];
then
      echo 1 > /sys/class/gpio/gpio250/value
      echo "<input type="checkbox" name="led2" value="led2" checked="checked"><font size="5">LED 2</font><br>"
else
      echo 0 > /sys/class/gpio/gpio250/value
      echo "<input type="checkbox" name="led2" value="led2" <font size="5">LED 2</font><br>"
fi

if [ $LED3 = "led3" ];
```

```
then
       echo 1 > /sys/class/gpio/gpio251/value
       echo      "<input      type="checkbox"      name="led3"      value="led3"
checked="checked"><font size="5">LED 3</font><br>"
else
       echo 0 > /sys/class/gpio/gpio251/value
       echo "<input type="checkbox" name="led3" value="led3" <font size="5">LED
3</font><br>"
fi

if [ $LED4 = "led4" ];
then
       echo 1 > /sys/class/gpio/gpio252/value
       echo      "<input      type="checkbox"      name="led4"      value="led4"
checked="checked"><font size="5">LED 4</font><br>"
else
       echo 0 > /sys/class/gpio/gpio252/value
       echo "<input type="checkbox" name="led4" value="led4" <font size="5">LED
4</font><br>"
fi

if [ $LED5 = "led5" ];
then
       echo 1 > /sys/class/gpio/gpio253/value
       echo      "<input      type="checkbox"      name="led5"      value="led5"
checked="checked"><font size="5">LED 5</font><br>"
else
       echo 0 > /sys/class/gpio/gpio253/value
       echo "<input type="checkbox" name="led5" value="led5" <font size="5">LED
5</font><br>"
fi

if [ $LED6 = "led6" ];
then
       echo 1 > /sys/class/gpio/gpio254/value
       echo      "<input      type="checkbox"      name="led6"      value="led6"
checked="checked"><font size="5">LED 6</font><br>"
else
       echo 0 > /sys/class/gpio/gpio254/value
       echo "<input type="checkbox" name="led6" value="led6" <font size="5">LED
6</font><br>"
fi

if [ $LED7 = "led7" ];
then
       echo 1 > /sys/class/gpio/gpio255/value
       echo      "<input      type="checkbox"      name="led7"      value="led7"
checked="checked"><font size="5">LED 7</font><br>"
else
       echo 0 > /sys/class/gpio/gpio255/value
       echo "<input type="checkbox" name="led7" value="led7" <font size="5">LED
7</font><br>"
fi

echo "<br>"
echo "<br>"

echo "<input type="submit" value="ACTUALIZAR">"

echo "<br>"
echo "<br>"

echo "ESTADO DE LOS INTERRUPTORES<br>"
```

```
echo "<table><tr>"
i=240
max=248
while [ $i -lt $max ]
do
       VALOR=`cat /sys/class/gpio/gpio$i/value`
       true $(( i++ ))
       if [ $VALOR = "1" ];
       then
              echo "<th bgcolor="00FF00">ON</th>"
       else
              echo "<th bgcolor="FF0000">OFF</th>"
       fi
done
echo "</tr></table>"
echo "</form></body></html>"
```

Anexo II

En este anexo se incluye el código completo empleado en la aplicación desarrollada en el Capítulo 8 basado en el módulo WiFi ESP8266.

```
#include "xparameters.h"
#include "xuartps.h"
#include "xgpio.h"

#define BUFFER_SIZE 2

XUartPs uart_wifi;
XUartPs uart_terminal;
XGpio leds;

int main()
{
       XUartPs_Config *config_wifi;
       XUartPs_Config *config_terminal;
       int status;
       unsigned int SentCount;
       unsigned int ReceivedCount;
       u32 LoopCount = 0;

       u8 SendBuffer[BUFFER_SIZE];
       u8 RecvBuffer[BUFFER_SIZE];

       XGpio_Initialize(&leds, XPAR_GPIO_0_DEVICE_ID);
       XGpio_SetDataDirection(&leds, 1, 0x0);
       XGpio_DiscreteWrite(&leds, 1, 0x0);

    config_wifi = XUartPs_LookupConfig(XPAR_XUARTPS_0_DEVICE_ID);
    if(NULL == config_wifi)
    {
       return XST_FAILURE;
    }
    status = XUartPs_CfgInitialize(&uart_wifi, config_wifi, config_wifi-
>BaseAddress);
    if(status != XST_SUCCESS)
    {
       return XST_FAILURE;
    }
    status = XUartPs_SelfTest(&uart_wifi);
    if(status != XST_SUCCESS)
```

```
    {
       return XST_FAILURE;
    }


    config_terminal = XUartPs_LookupConfig(XPAR_XUARTPS_1_DEVICE_ID);
    if(NULL == config_terminal)
    {
       return XST_FAILURE;
    }
    status     =     XUartPs_CfgInitialize(&uart_terminal,     config_terminal,
config_terminal->BaseAddress);
    if(status != XST_SUCCESS)
    {
       return XST_FAILURE;
    }
    status = XUartPs_SelfTest(&uart_terminal);
    if(status != XST_SUCCESS)
    {
       return XST_FAILURE;
    }

    XUartPs_Send(&uart_wifi, "\nControl de LEDs WiFi\n\r", 22);
    XUartPs_Send(&uart_terminal, "\nControl de LEDs Terminal\n\r", 27);

    do{

              XUartPs_Send(&uart_wifi, "Introduzca un numero de LED\n\r", 29);

              ReceivedCount = 0;
              while(ReceivedCount < BUFFER_SIZE)
              {
                     ReceivedCount          +=          XUartPs_Recv(&uart_wifi,
&RecvBuffer[ReceivedCount], (BUFFER_SIZE - ReceivedCount));
              }

              if(atoi(&RecvBuffer[0]) < 9 && atoi(&RecvBuffer[0]) > 0)
              {
                     XUartPs_Send(&uart_terminal, RecvBuffer, 2);
                     XUartPs_Send(&uart_wifi, RecvBuffer, 2);
                     XGpio_DiscreteWrite(&leds, 1, (1 << (atoi(&RecvBuffer[0])-
1)));
              }
              else
              {
                     XUartPs_Send(&uart_terminal, "El numero debe estar entre 1
y 8\n\r", 34);
                     XUartPs_Send(&uart_wifi, "El numero debe estar entre 1 y
8\n\r", 34);
              }
              XUartPs_Send(&uart_wifi, "\nSeguir? (s/n)\n\r", 15);

              ReceivedCount = 0;
              while(ReceivedCount < BUFFER_SIZE)
              {
                     ReceivedCount          +=          XUartPs_Recv(&uart_wifi,
&RecvBuffer[ReceivedCount], (BUFFER_SIZE - ReceivedCount));
              }
    }while(strncmp(RecvBuffer, "s", 1) == 0);

    XUartPs_Send(&uart_wifi, "\nTerminado\n\r", 12);
    XUartPs_Send(&uart_terminal, "\nTerminado\n\r", 12);
```

```
    return 0;
}
```

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR