DESARROLLO DE *DRIVERS* PARA PERIFÉRICOS GENÉRICOS IMPLEMENTADOS EN EL SISTEMA DE DESARROLLO SIE.

ARIDES JOSÉ MENESES HERNÁNDEZ

Escuela de Ingenierías
Eléctrica, Electrónica
y de Telecomunicaciones

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Universidad Industrial de Santander
Bucaramanga

Agosto, 2013

DESARROLLO DE *DRIVERS* PARA PERIFÉRICOS GENÉRICOS IMPLEMENTADOS EN EL SISTEMA DE DESARROLLO SIE.

Presentado Por:

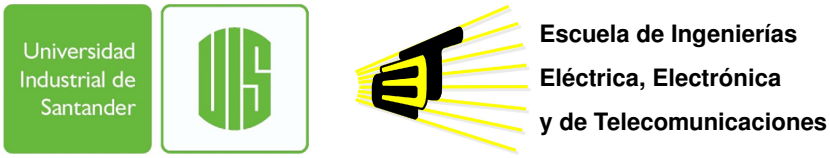
ARIDES JOSÉ MENESES HERNÁNDEZ

Director:

MIE. William A. Salamanca B.

Co-Director:

ING. Carlos A. Angulo J.



Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Universidad Industrial de Santander

Bucaramanga

Agosto, 2013

Dedico este trabajo a mi familia, en especial a mi esposa por ser el complemento perfecto en mi vida, por brindarme la oportunidad de compartir momentos a su lado, por su ayuda incondicional y por regalarme una hija tan hermosa. Katherine Maria Miranda Vanegas y Sharik Milagros Meneses Miranda las AMO demasiado y que Dios bendiga sus vidas hoy y siempre.

Arides José Meneses Hernández

Agradecimientos

Este trabajo investigación ha llegado a feliz término y el mérito no es solo mío sino también de todas las personas que me apoyaron durante todo su desarrollo.

En primer lugar, agradezco a DIOS todo poderoso por haberme regalado la vida, la salud, la sabiduría, el entendimiento, por su acompañamiento y respaldo durante todo este proceso en la Universidad Industrial de Santander. *Te Amo mi Dios*.

En mi familia quiero agradecer de corazón a mis padres Arides José Meneses Torres, Zunith Hernández Turizo y a mi hermano Didier José Meneses Hernández, por que gracias a su apoyo incondicional, esfuerzo, sacrifico y dedicación he podido alcanzar esta meta, por la confianza depositada y por su apoyo en los momentos difíciles que me ayudan a sobrepasar las adversidades que se me van presentando en el camino. Quiero hacer una mención especial a mi tía Maria Esperanza Hernández Turizo quien desde un principio creyó y apoyó mi idea de superación.

Quiero agradecer a la Universidad Industrial de Santander, al grupo de investigación CPS, a mi Director de tesis el profesor William Alexander Salamanca Becerra, a mi Co-director de tesis el profesor Carlos Andres Angulo Julio por su dedicación, disposición y formación durante este proceso. Muchas gracias a ustedes por creer en este trabajo y por su gran compromiso con este proyecto.

Finalmente quiero agradecer a mis demás familiares y a todos mis amigos, por ser ese excelente grupo de personas que me brindo el apoyo necesario para enfrentar este desafío, les quiero compartir este momento y este trabajo.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Lista de Anexos

RESUMEN

TÍTULO: DESARROLLO DE *DRIVERS* PARA PERIFÉRICOS GENÉRICOS IMPLEMENTADOS EN EL SISTEMA DE DESARROLLO SIE.*

AUTOR: ARIDES JOSÉ MENESES HERNÁNDEZ **

PALABRAS CLAVES: Plataforma SIE, *Driver*, periféricos, procesador, FPGA, interfaz de comunicación, sistemas embebidos, transferencia tecnológica.

DESCRIPCIÓN:

El presente proyecto de grado se realizó bajo el apoyo del grupo de investigación Conectividad y Procesamiento de Señales CPS de la Universidad Industrial de Santander y está dirigido a todas las personas interesadas en la temática de este trabajo. El propósito de esta investigación fue realizar un estudio sobre los sistemas que integran un Microprocesador y un FPGA, ventaja que ofrece la plataforma de desarrollo SIE al disponer entre sus recursos estos dos componentes. Como resultado de este estudio se obtuvo un conjunto de herramientas con las cuales se pueden realizar diversas actividades en este tipo de sistemas. El campo de trabajo de esta investigación está relacionado con la transferencia tecnológica en el área de sistemas embebidos. Durante su ejecución se elaboró una interfaz de comunicación entre el Procesador JZ4725 y el FPGA del sistema de desarrollo SIE, para lo cual fue necesario desarrollar *drivers* que administran los recursos de un conjunto de periféricos de hardware. La elaboración de estos *drivers* se centró en la metodología explicada en el libro *Linux Device Drivers, Third Edition*. Al final del trabajo representado en esta investigación, se realizó un proceso de validación mediante la implementación de los diferentes tipos de periféricos con su respectivo *driver* de apoyo funcionando.

*Trabajo de grado

Facultad: Facultad de Ingenierías Físico Mecánicas.**Escuela:** Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones.**Grupo de Investigación:** CPS. **Director:** MIE. William A. Salamanca B.. **Codirector:** ING. Carlos A. Angulo J..

ABSTRACT

TITLE: DRIVERS DEVELOPMENT FOR GENERIC PERIPHERALS IMPLEMENTED ON THE SIE DEVELOPMENT SYSTEM.*

AUTHOR: ARIDES JOSÉ MENESES HERNÁNDEZ **

KEY WORDS: SIE Board, *Driver*, peripherals, processor, FPGA, communication interface, embedded systems, technology transfer.

DESCRIPTION:

This thesis was made with the support of the Connectivity and Signal Processing Research Group (CPS) from the Universidad Industrial de Santander and it is aimed to people interested on topics like; embedded systems, drivers in Linux and FGPAs, among others. The purpose of this research project was to present a study about the systems which are integrated inside a microprocessor and an FPGA. One of the advantages offered by the SIE development board is that it provides these devices. As a result of this study, a tool set has been developed to perform diverse tasks regarding this topics. The work-field of this investigation is related to technological transference on the area of embedded systems. A communication interface between the platform's processor JZ4725 and the Spartan 3E FPGA was made. Drivers development was necessary to manage the resources of a set of hardware peripherals. The drivers development were done upon the methodology explained in the Linux Device Drivers textbook by O'Reilly. To complete this work, a validation process was done implementing the different peripherals with its related support driver.

*Degree project

Faculty: Physico-mechanical Engineering Faculty. School:** School of Electrical, Electronics and Telecommunications Engineering.**Research group:** CPS. **Director:** MIE. William A. Salamanca B.. **Codirector:** ING. Carlos A. Angulo J.

Introducción

Con el transcurrir de los años, los dispositivos de hardware han venido adaptándose al proceso cambiante de la tecnología, sin embargo, cuando adquirimos un nuevo dispositivo es probable que no esté soportado por la distribución del Sistema Operativo que estamos utilizando. En la práctica es común que comencemos en la búsqueda de *drivers* para los dispositivos que no están soportados, pero en la mayoría de los casos tras largos periodos de búsqueda los resultados son negativos. Muchos de los desarrolladores de *drivers* atienden a las solicitudes de los usuarios, pero la respuesta toma largos periodos de tiempo en aparecer. Ésta situación se complica aún más cuando el hardware no es popular, entonces aparecen dos alternativas de solución a este problema. La primera solución es esperar respuesta por parte del fabricante cuya finalidad es proporcionar el producto final terminado, pero se debe estar dispuestos a realizar el proceso de compilado del *driver* y luego añadirlo al Sistema Operativo sin errores para no ocasionar daños irreparables a la máquina. La segunda solución propone como alternativa desarrollar el *driver* y que este cumpla con las restricciones que necesita el dispositivo para su funcionamiento.

El presente proyecto realiza un aporte en la construcción de *drivers* basados en el Sistema Operativo Linux, y en él se resalta la importancia de trabajar con la comunicación directa entre el procesador y el FPGA de la plataforma SIE. Este documento recopila el proceso de diseño, las experiencias, la implementación y está distribuido en capítulos de la siguiente forma:

EL **capítulo uno** documenta todo el proceso de estudio de periféricos en EDK y se presentan las aplicaciones desarrolladas para validar el estudio de periféricos.

El **capítulo dos** explica la metodología empleada para la realización de los *drivers*, el interfaz de periféricos desarrollados en ésta investigación, junto con los diseños finales funcionando en la plataforma de desarrollo SIE.

EL **capítulo tres** presenta la metodología empleada en el proceso de validación de los *drivers*.

Finalmente en el **capítulo cuatro** se presentan las conclusiones obtenidas luego de llevar a cabo la implementación y la continuidad que se le puede dar a esta investigación en el futuro.

CAPÍTULO 1

Embedded Development Kit (EDK)

1.1. Introducción

En este capítulo se presentan los pasos para trabajar con el entorno de trabajo *Embedded Development Kit (EDK)*, se estudiarán cada uno de los periféricos con características [1] comenzando desde la creación del hardware en la herramienta *Xilinx Platform Studio (XPS)* y finalizando el ciclo en las aplicaciones de software que gestionan la información de los periféricos por medio de la herramienta *Software Development Kit (SDK)*. Se validará el estudio realizando pequeños proyectos que involucran el hardware y el software como un conjunto y en la etapa final cada proyecto será implementado en el sistema de desarrollo Spartan-6 SP506 [2] para verificar su funcionamiento y desempeño.

1.2. Sobre EDK

El entorno de trabajo EDK, es un conjunto de herramientas que tiene como función realizar co-diseño de sistemas de procesamiento embebidos basados en los procesadores PowerPC/MicroBlaze y ser implementado en un dispositivo FPGA. EDK es un componente de la herramienta *Integrated Software Environment (ISE)*, éste ultimo permite llevar a cabo el diseño de un circuito digital, desde el ingreso de las fuentes, pasando por el proceso de síntesis y análisis, hasta llegar a su implementación sobre un dispositivo FPGA.

EDK se accede por medio de una interfaz gráfica que permite manipular el software que se ejecuta en el procesador, cómo es la interacción de los elementos del sistema, los buses, interrupciones y demás componentes del hardware. Entre sus características ofrece dos herramientas complementarias que hacen posible el desarrollo, estas son:

- *Xilinx Platform Studio (XPS)*.
- *Software Development Kit (SDK)*.

[1] Estos tipos de periféricos se definen en la sección 1.3.1.
[2] Se puede usar cualquier familia de FPGAs

En la Figura 1.1 se muestran los componentes que interactúan en el proceso del flujo de diseño de EDK.

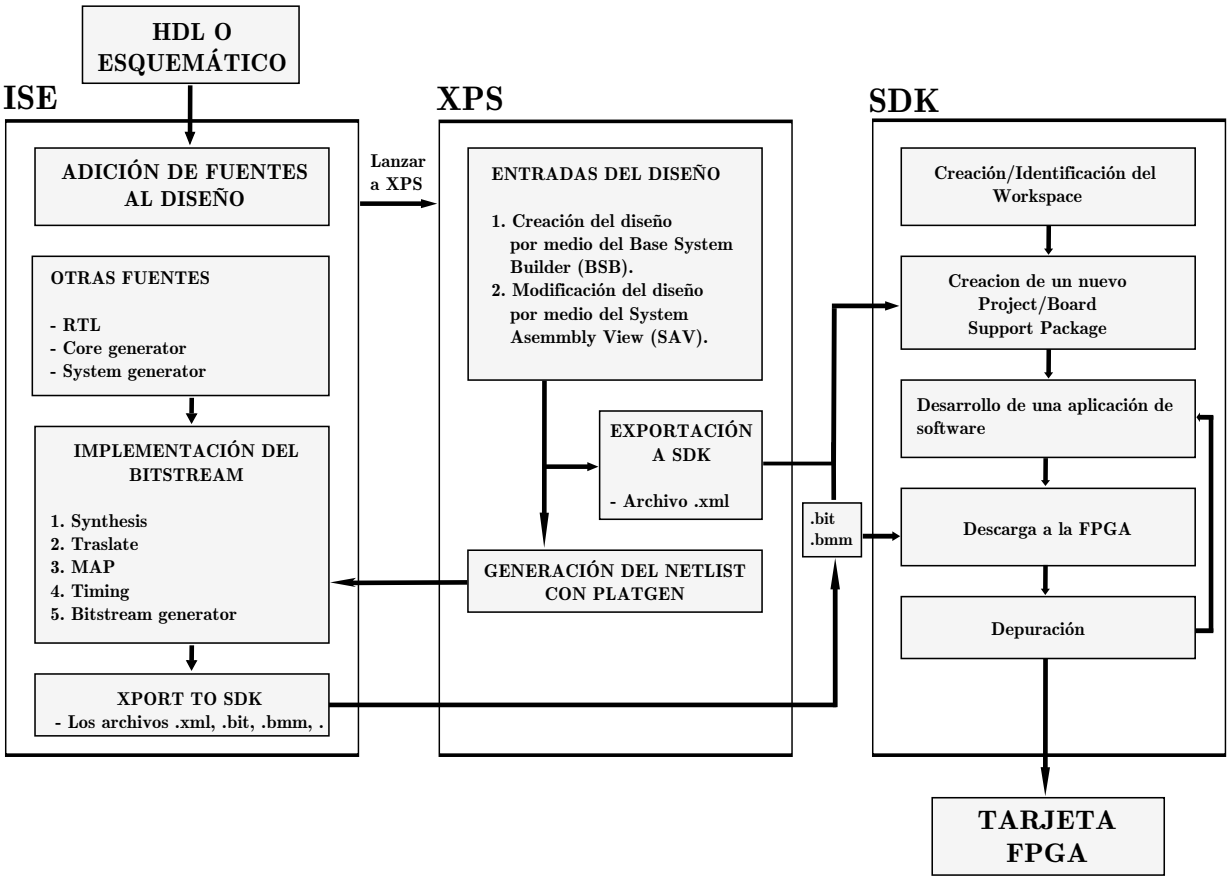


Figura 1.1: Flujo de diseño de EDK.

FUENTE: Adaptado de [2].

A continuación se presentan los procedimientos, pasos y características más importantes que permiten el desarrollo con las herramientas XPS y SDK.

1.3. *Xilinx Platform Studio*

El propósito de trabajar con la herramienta XPS es desarrollar el hardware necesario en el co-diseño. Entre sus características ofrece la posibilidad de:

- Añadir procesadores y periféricos, modificar sus parámetros fundamentales, y hacer las conexiones entre ellos por medio de un bus de comunicación.
- Generar y ver un diagrama de bloques que enseña las conexiones y los elementos que componen el diseño.
- Exportar el hardware a la herramienta de trabajo SDK.

La Figura 1.2 muestra un sistema embebido realizable por la herramienta de trabajo XPS. En ella se identifican los elementos que interactúan en el proceso de diseño.

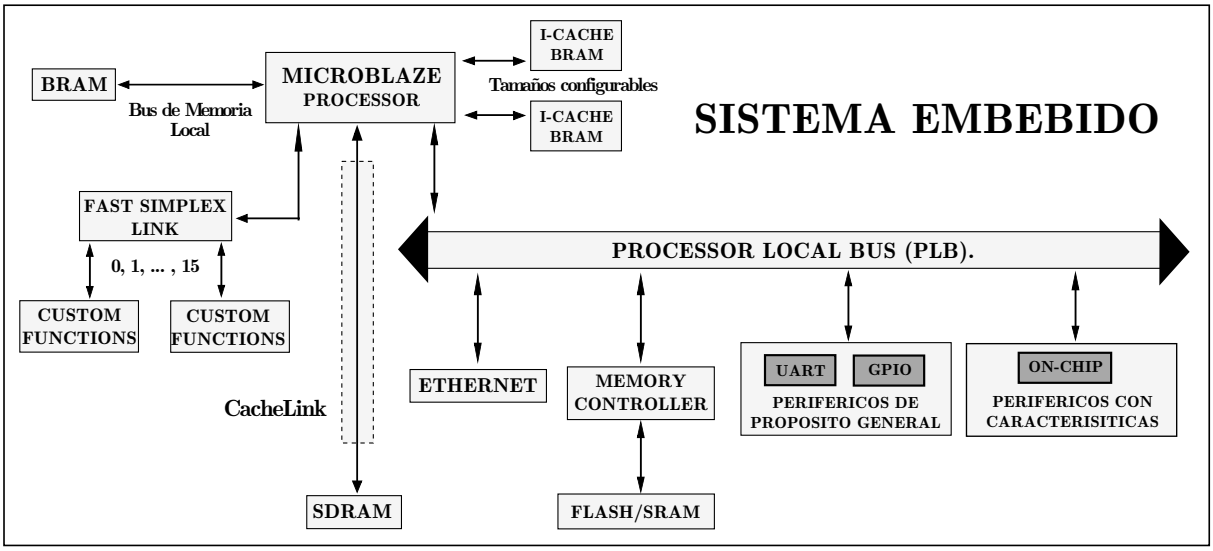


Figura 1.2: Panorama general de un sistema embebido.

FUENTE: Adaptado de [3].

Estos elementos son:

- Procesador Microblaze.
- Bus de comunicación de alta velocidad *Fast Simplex Link (FSL)*.
- Bus de comunicación de baja velocidad *Processor Local Bus (PLB)*.
- Bloques de memoria BRAM.
- Periféricos de propósito general.
- Periféricos con características.
- Periféricos de propósito específico como Ethernet, Flash, Sdram, etc.

Por medio del asistente de configuración *Base System Builder (BSB)* de XPS es posible crear rápidamente un sistema de trabajo. *En el anexo A se describen los pasos para crear un nuevo proyecto en XPS*.

Tipos de periféricos

Los periféricos que ofrece la herramienta de trabajo XPS se pueden clasificar en:

1. Periféricos de propósito general:
   Estos pueden ser utilizados en diversas aplicaciones dentro de un mismo proyecto y se listan a continuación.

- *Dip Switch*
- *Push Button*
- *Leds*

2. Periféricos con características:
   Los periféricos con características reciben este nombre ya que aparte de realizar su labor, pueden ser usados por los otros periféricos.
   - *User Logic Software Register*
   - *Read/Write FIFO*
   - *Software Reset*
   - *Interrup Control*
   - *User Logic Memory Space*
3. Periféricos especiales:
   Son diseños realizados por el usuario cuando se necesita de una aplicación de hardware específica.

1.4. Periféricos con características

Los periféricos con características son el objeto de estudio en esta sección. *En el anexo B se presentan los pasos para crear y configurar un periférico con características en el sistema de trabajo.* Con el propósito de entender el funcionamiento de cada periférico, se realizó una revisión en detalle de los archivos plantilla a disposición del usuario. Estas plantillas están ubicadas en el directorio del proyecto como se muestra en la Figura 1.3.

En la rama de archivos mostrada se identifican los archivos **user_Logic.vhd** y **periferico.vhd**, estos dos se destacan como las más importantes y recopilan toda la descripción de hardware necesaria para entender cómo gestiona la información un periférico.

- user_logic.vhd
  Este es el archivo plantilla en donde el usuario digita la lógica de diseño, contiene todas las funcionalidades que se desean implementar. Esta es una plantilla genérica en la cual el sofware de trabajo muestra ciertos fragmentos de código que son elaborados con fines demostrativos.
- periferico.vhd
  Este es el archivo plantilla de la entidad principal del periférico en diseño. Este archivo es el puente entre el "user_logic.vhd" y el bus de comunicación con el procesador, aquí la herramienta configura los registros y todo lo necesario para establecer la comunicación bus-periférico.

La Figura 1.4 muestra un diagrama de bloques que representa de forma sencilla el conjunto de las plantillas "user_logic.vhd" y periférico.vhd.

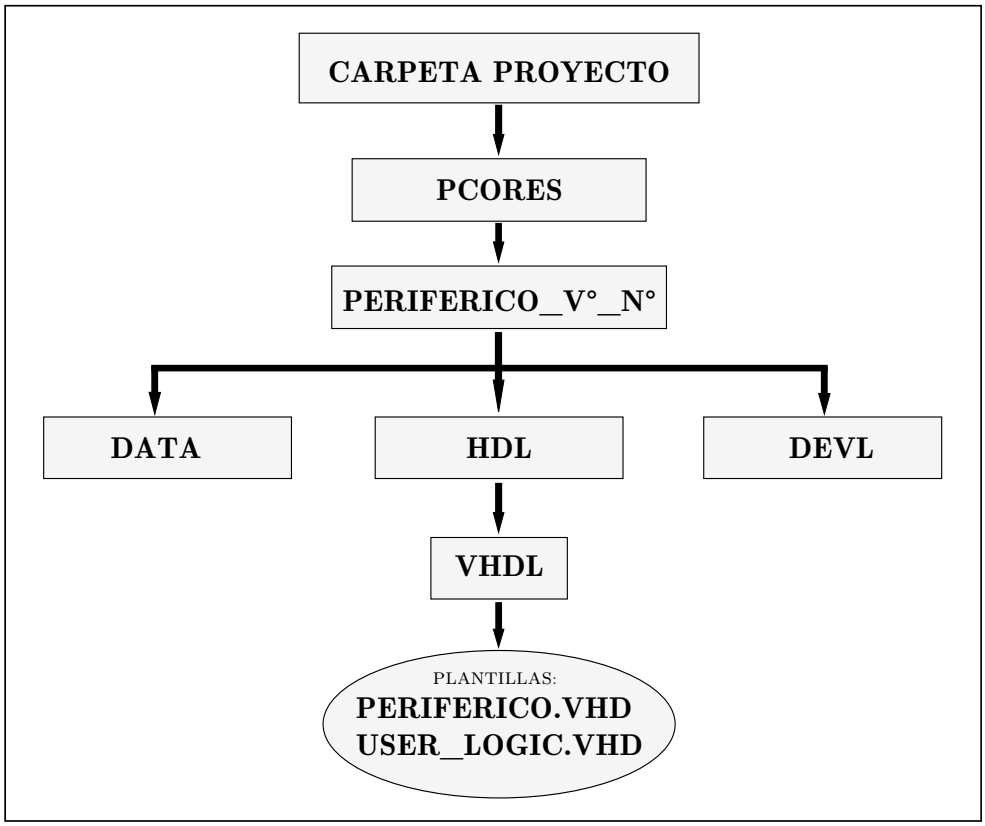


Figura 1.3: Mapa de archivos del hardware.

FUENTE: El autor.

A continuación se presenta la información recopilada en la revisión de las plantillas “user_logic.vhd” y “periférico.vhd”, se dan a conocer los circuitos que influyen en el proceso de lectura/escritura en los periféricos y se muestran los bloques que representan a los periféricos con características.

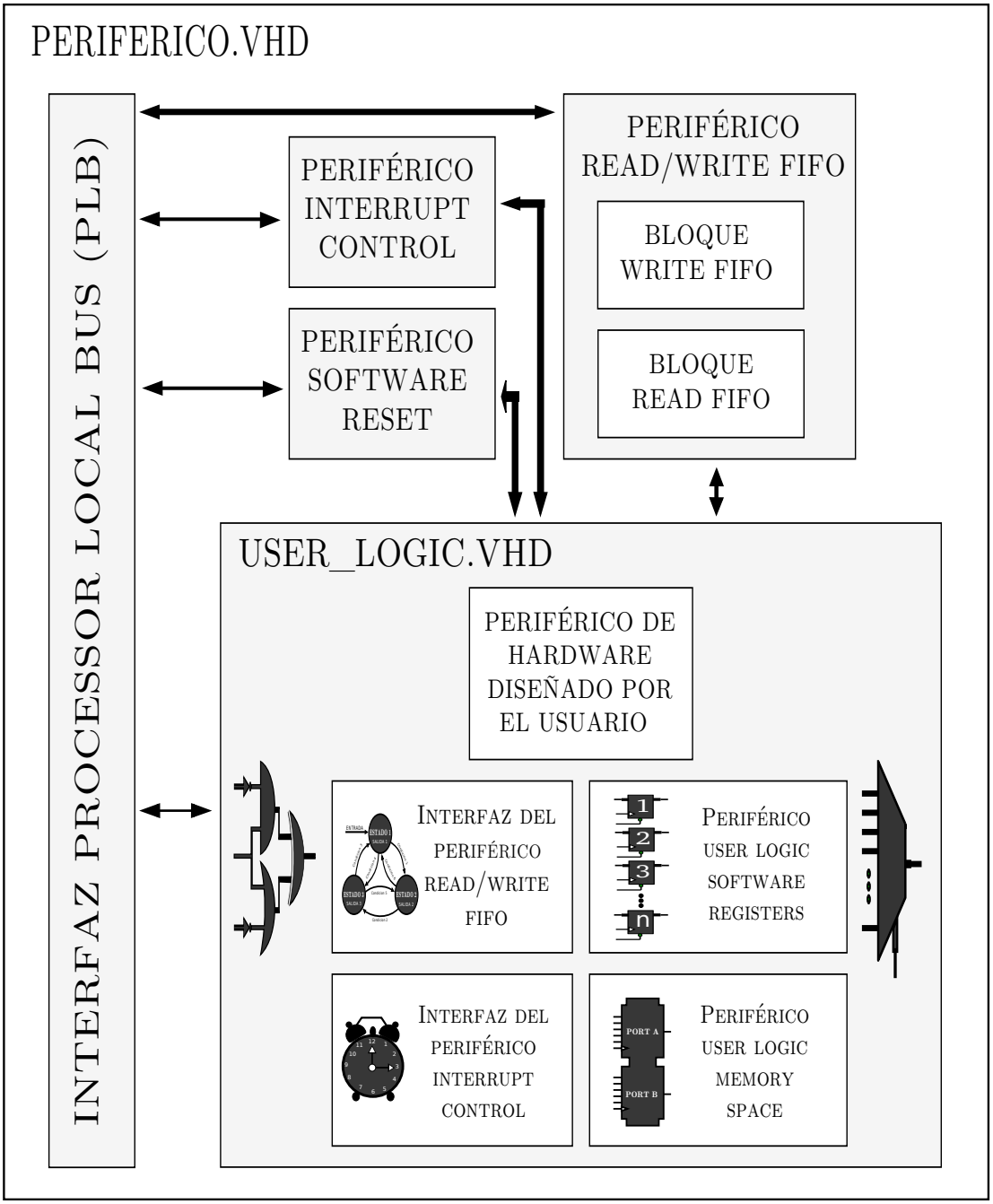


Figura 1.4: Representación en bloques de las plantillas de hardware.

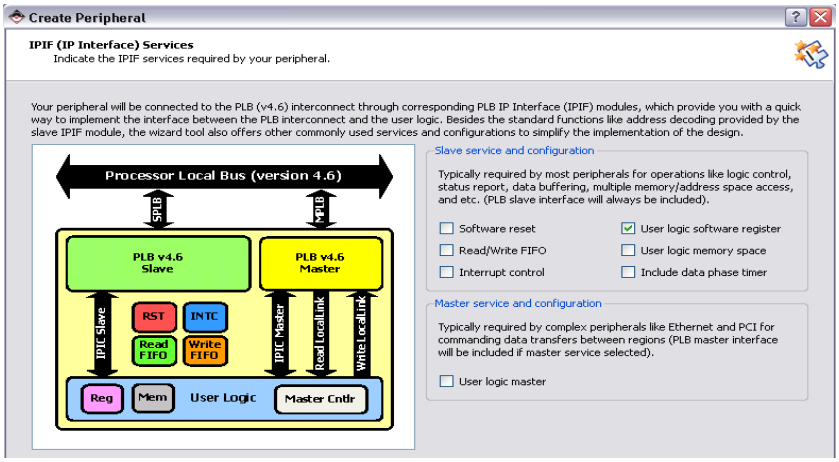
FUENTE: El autor.

Periférico *User Logic Sotfware Register*

El *User Logic Software Register* es un periférico que contiene registros direccionados a través de software. Proporciona al usuario hasta 4096 registros, cada uno de ellos con ancho de palabra de 32 bits. Estos registros se encuentran disponibles al usuario en la plantilla “user_logic.vhd”.

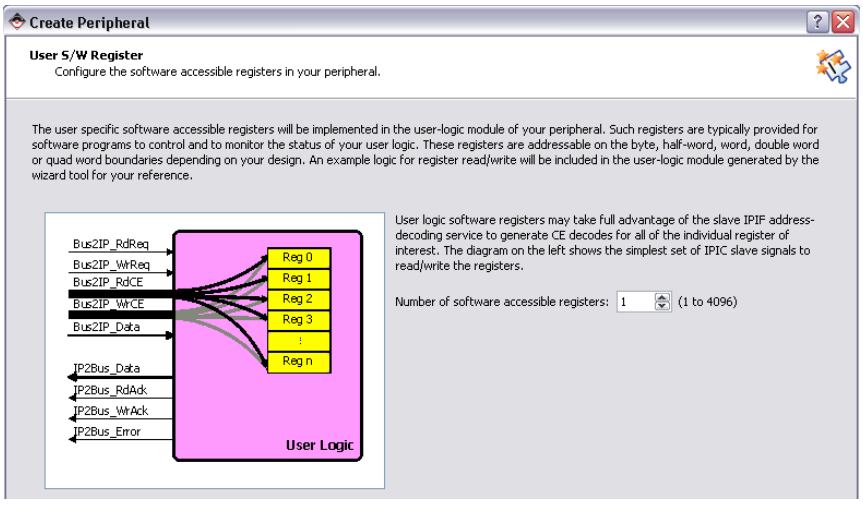
Es necesario generar este periférico al momento de crear el proyecto para poder hacer uso de la plantilla “user_logic.vhd”. Tomando como guía el anexo B, en la ventana 6 se procede como sigue:

1. Seleccionar la casilla del periférico *User Logic Software Register*.



FUENTE: Tomado de [4].

2. Escoger la cantidad de registros a utilizar en la opción *Number of software accessible register*.



FUENTE: Tomado de [4].

3. Continuar con los pasos del anexo B, ventana 7.

De acuerdo al diagrama de bloques de la Figura 1.4, se resaltan los elementos involucrados para este periférico (ver Figura 1.5). Luego de generar el periférico, estamos listos para revisar el contenido de la plantilla "user_logic.vhd", esta se puede encontrar siguiendo la ruta del mapa de archivos mostrado en la Figura 1.3.

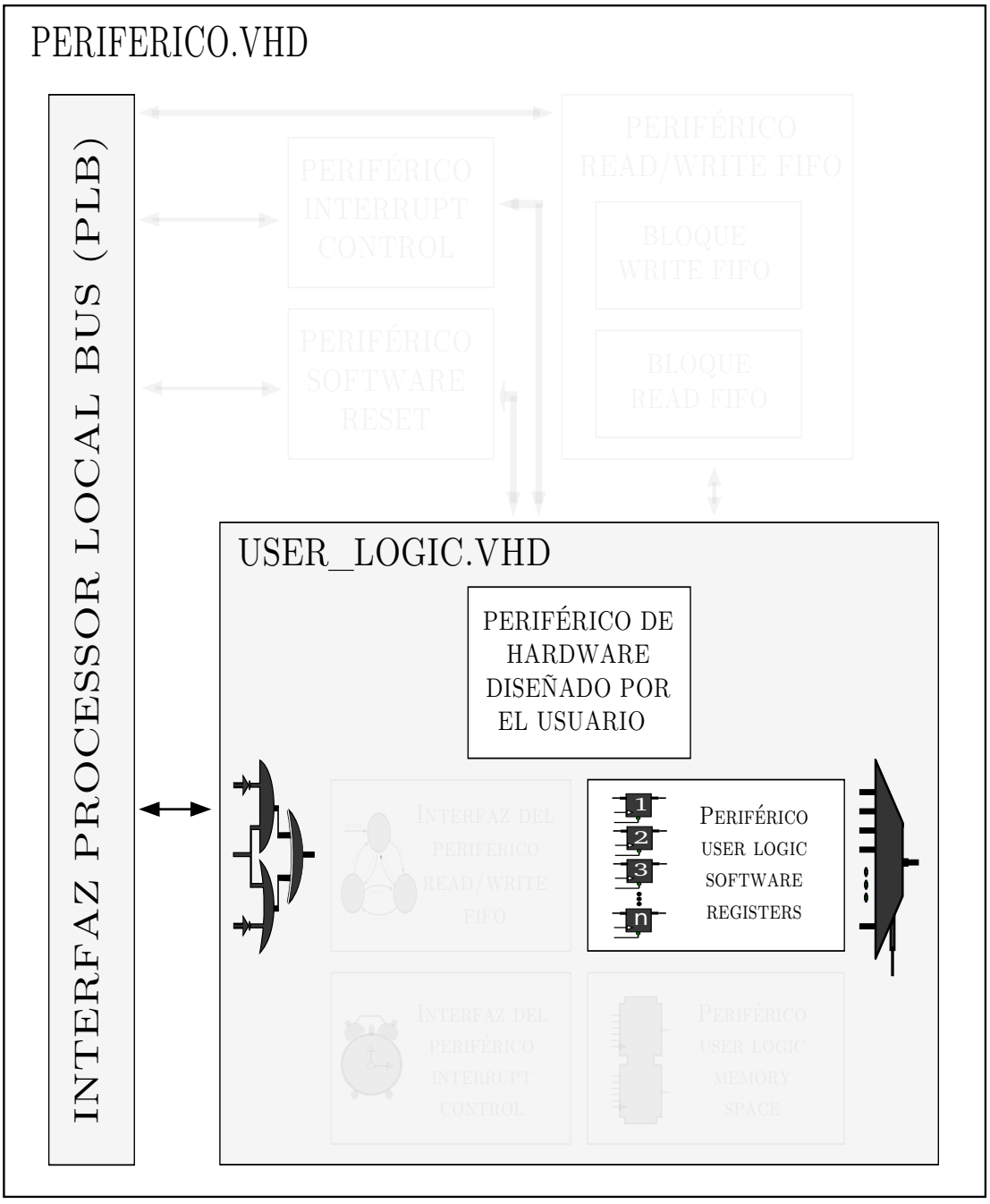


Figura 1.5: Bloques del periférico *user logic software register*.

FUENTE: El autor.

En el proceso de revisión de la plantilla se determinaron los circuitos y elementos que la conforman:

1. Una entidad.
2. Un circuito combinacional a la entrada, para obtener la lectura de datos.
3. Los registros de 32 bits seleccionados para el almacenamiento de los datos.
4. Un Multiplexor que provee la escritura de los datos de salida.
5. Un campo destinado para un periférico especial diseñado por el usuario.

Lo anterior se puede representar mediante un diagrama de bloques como muestra la Figura 1.6. El conjunto de este circuito es indispensable para llevar a cabo la gestión de la información cuando un usuario realiza una operación de lectura o escritura en el periférico.

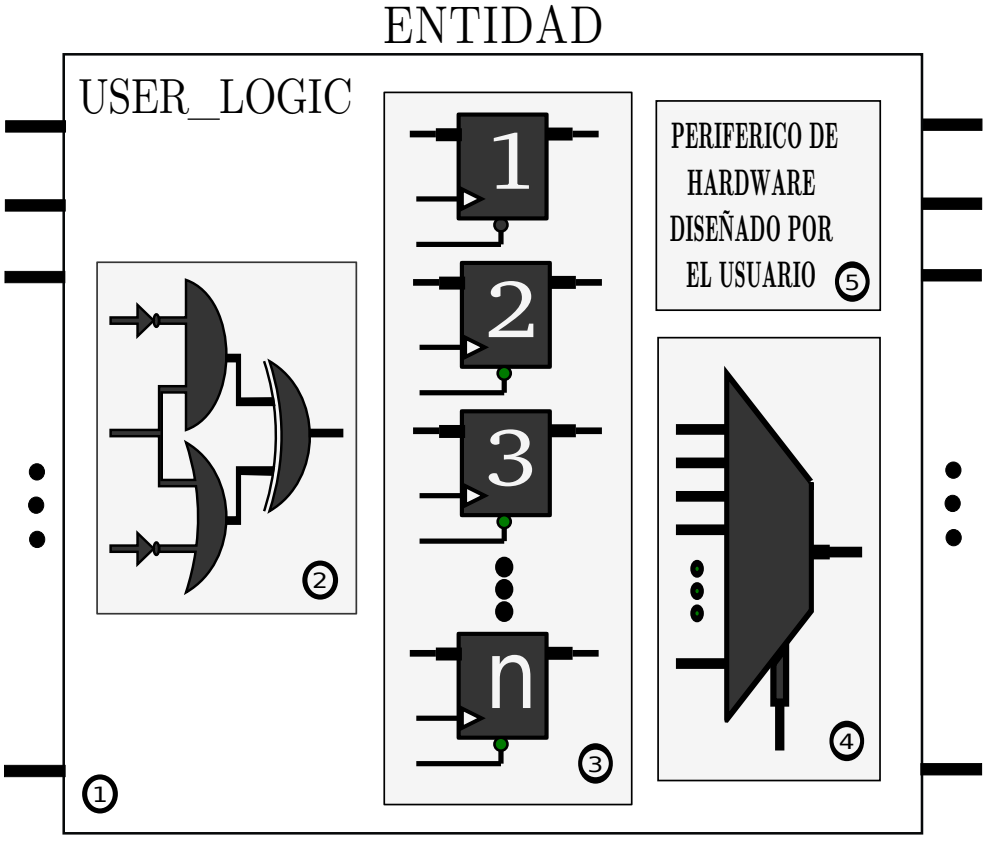


Figura 1.6: user_logic.vhd del priférico *user logic software register*.

FUENTE: El autor.

Este tipo de periférico comúnmente es usado cuando el usuario diseña un periférico especial para una aplicación específica. *El Anexo C corresponde a una guía que describe el procedimiento para adicionar al sistema de trabajo un periférico especial*.

Periférico *Read/Write FIFO*

Este periférico es un bloque de memoria que permite almacenar temporalmente cierto volumen de datos. Su característica principal se relaciona con las iniciales de su nombre, esto es: *First Input - First Output*, el primer dato en entrar es el primer dato en salir, es decir; siempre procesa la información en el orden de llegada. Proporciona al usuario hasta 16384 posiciones de memoria para el almacenamiento de datos y cada posición de memoria es representada como un registro de 32 bits. La Figura 1.7 muestra un arreglo que describe cómo una memoria fifo realiza la gestión de la información.

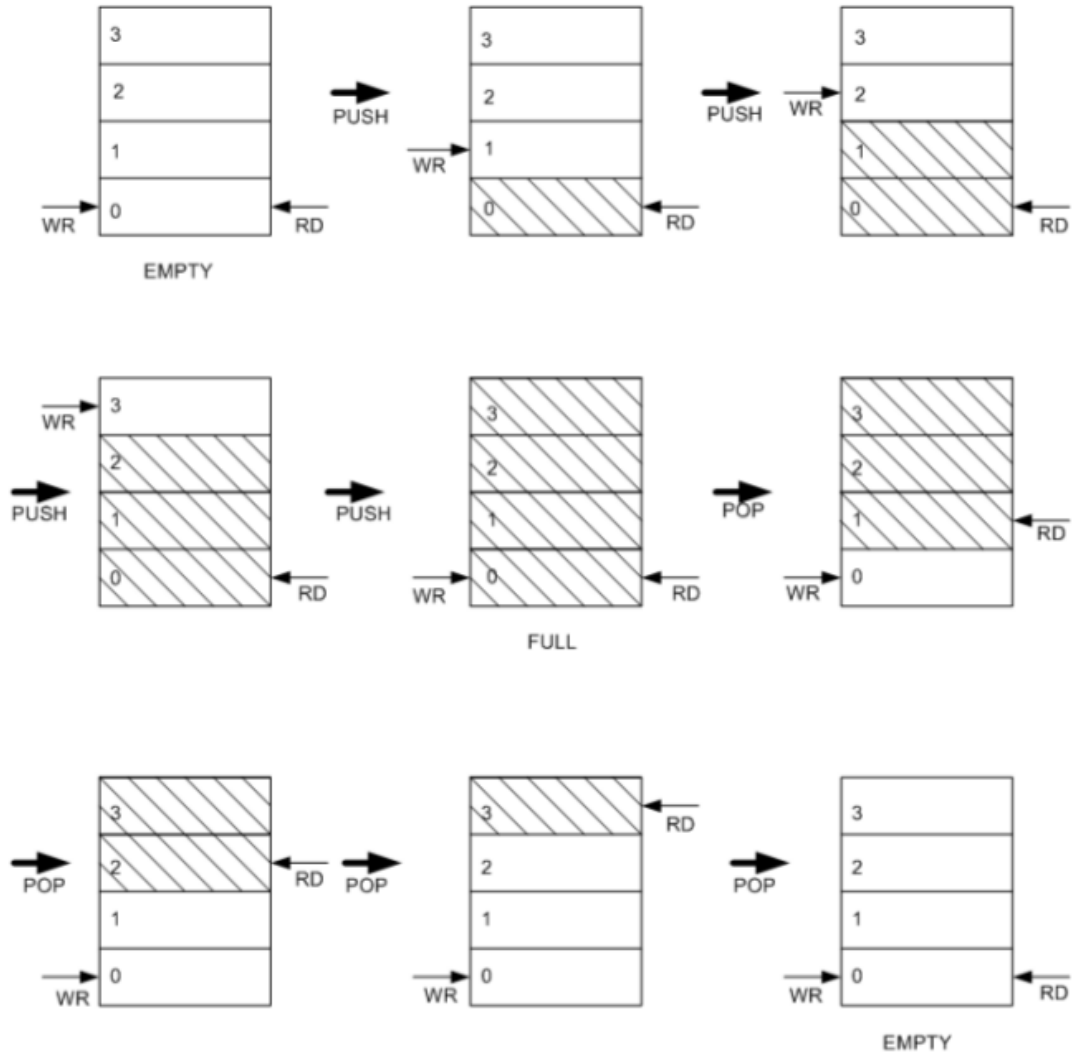
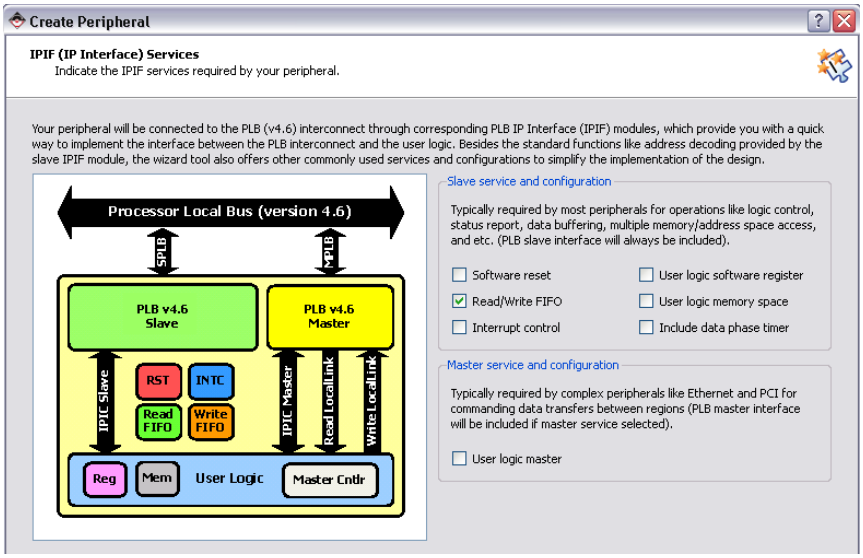


Figura 1.7: Representación del proceso de lectura/escritura en una memoria FIFO.

FUENTE: El autor.

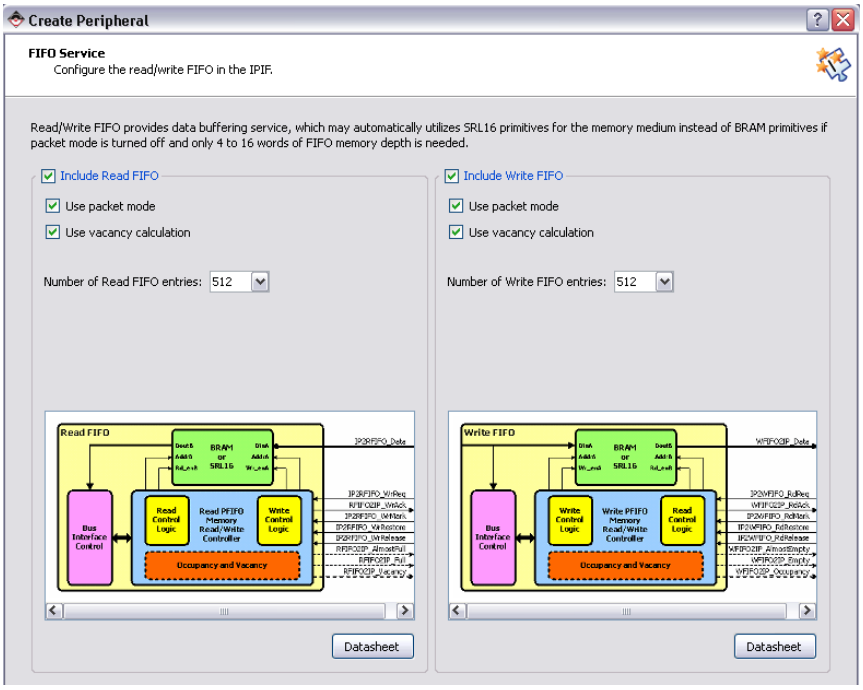
Es necesario generar este periférico al momento de crear el proyecto para poder hacer uso de la plantilla "user_logic.vhd". Tomando como guía el anexo B, en la ventana 6 se procede como sigue:

1. Seleccionar la casilla del periférico *Read/Write FIFO*.



FUENTE: Tomado de [4].

2. Seleccionar la profundidad o número de posiciones de la memoria.



FUENTE: Tomado de [4].

3. Continuar con los pasos del anexo B, ventana 7.

De acuerdo al diagrama de bloques de la Figura 1.4, se resaltan los elementos involucrados para este periférico (ver Figura 1.8). Luego de generar el periférico, estamos listos para revisar el contenido de la plantilla "user_logic.vhd", esta se puede encontrar siguiendo la ruta del mapa de archivos mostrado en la Figura 1.3.

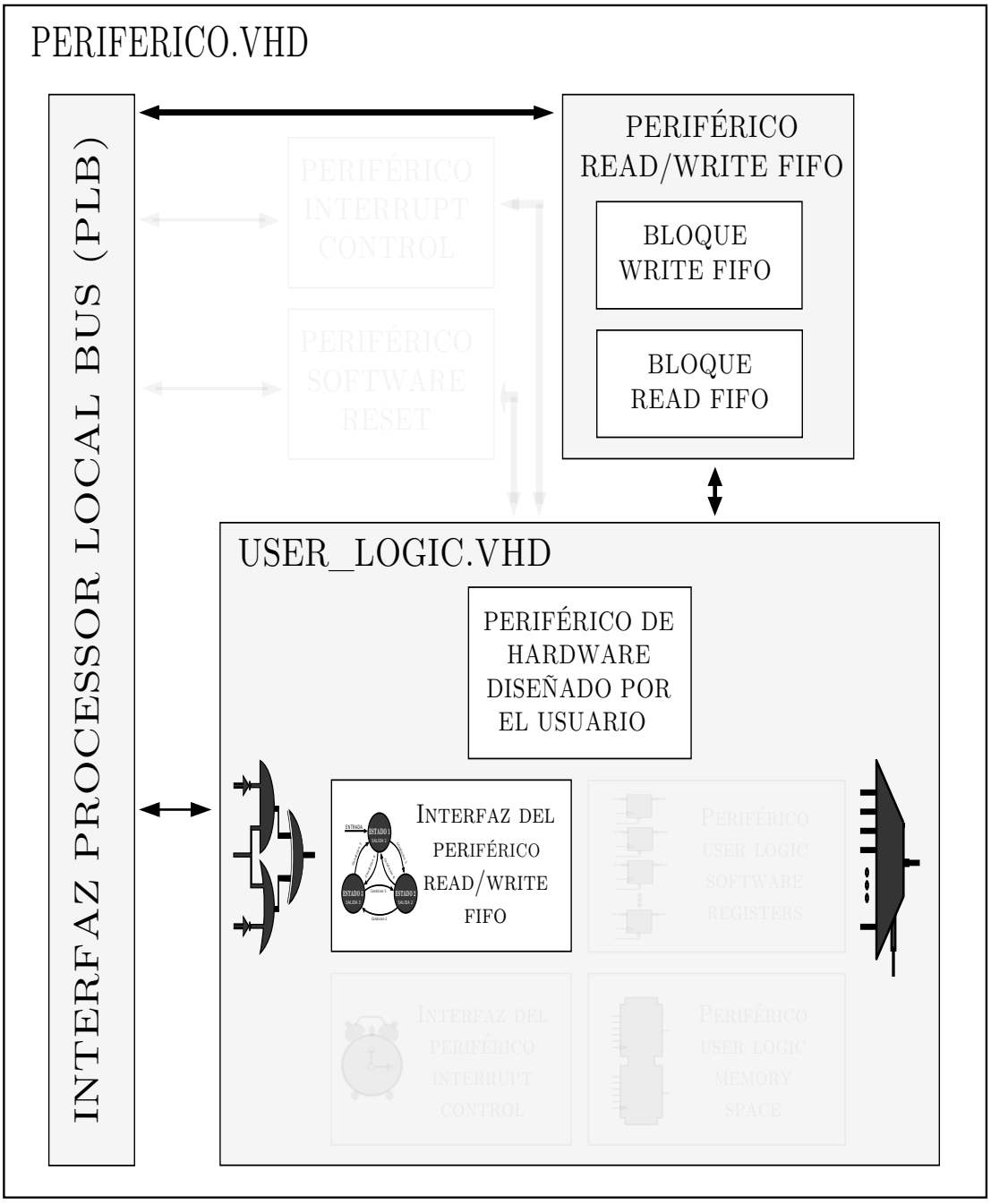


Figura 1.8: Bloques del periférico *read/write fifo*.

FUENTE: El autor.

En el proceso de revisión de la plantilla se determinaron los circuitos y elementos que la conforman:

1. Una entidad.
2. Una máquina de estados FSM que realiza los saltos de posición en la memoria.
3. Un campo destinado para un periférico especial diseñado por el usuario.

Lo anterior se puede representar mediante un diagrama de bloques como se muestra en la Figura 1.9. Si revisamos la Figura 1.8 se puede apreciar que dos nuevos bloques hacen parte del circuito, los cuales son la representación física del periférico creado. Cada uno de ellos realiza una tarea específica, el bloque *Write_FIFO* gestiona los datos que se desean escribir, mientras que el bloque *Read_FIFO* gestiona los datos que se desean leer, por otra parte el circuito del "user_logic.vhd" realiza los saltos de posición en la memoria a medida que ésta se lee o se escribe.

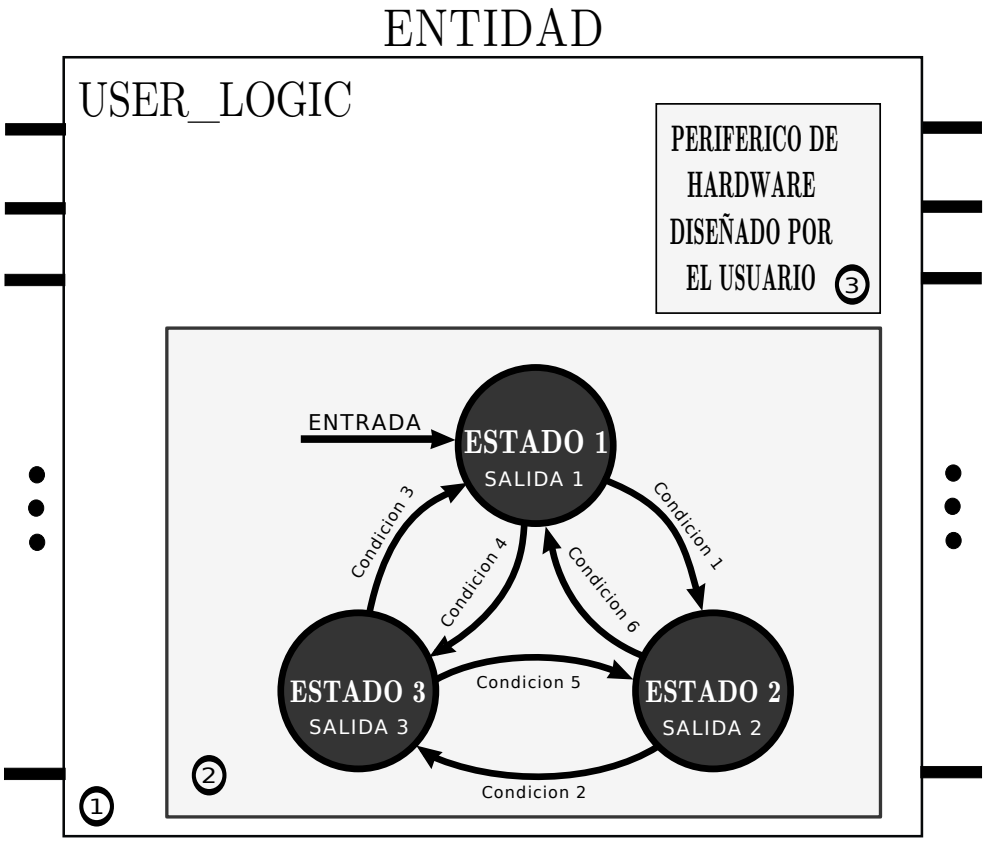


Figura 1.9: user_logic.vhd del priférico *read/write fifo*.

FUENTE: El autor.

Periférico *Interrupt Control*

Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción. Luego de finalizada dicha subrutina, se reanuda la ejecución del programa. Las interrupciones surgen de la necesidad que tienen los dispositivos periféricos de recibir más instrucciones para seguir operando, en la Figura 1.10 se ilustra el ciclo de interrupciones.

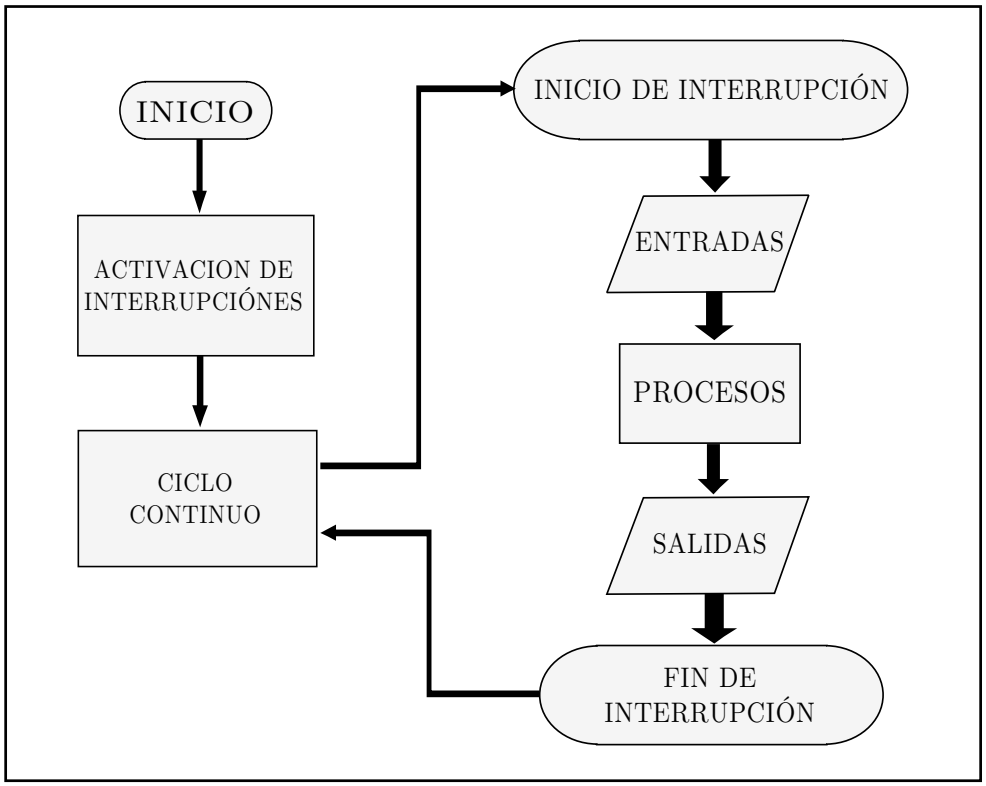
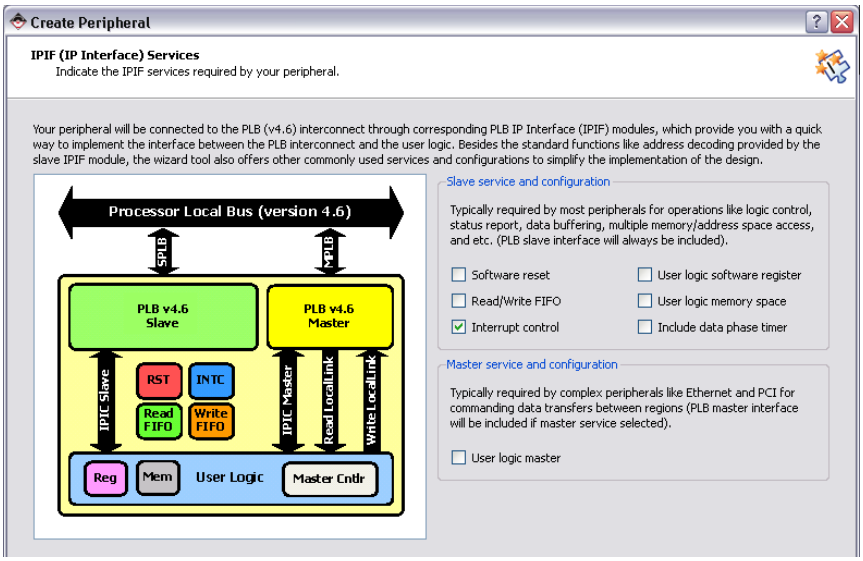


Figura 1.10: Ciclo de interrupción.

FUENTE: Adaptado de [5].

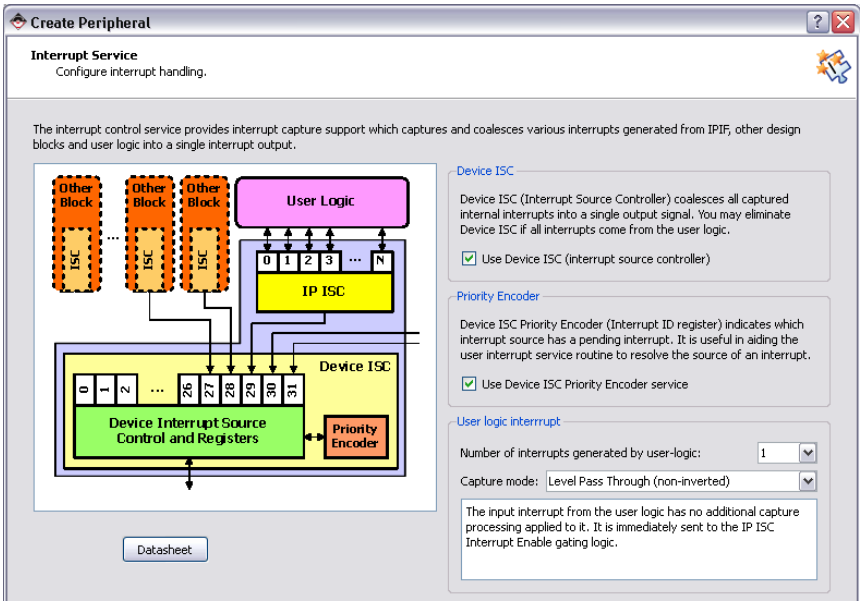
Es necesario generar este periférico al momento de crear el proyecto para poder hacer uso de la plantilla "user_logic.vhd". Tomando como guía el anexo B, en la ventana 6 se procede como sigue:

1. Seleccionar la casilla del periférico *Interrupt Control*.



FUENTE: Tomado de [4].

2. Escoger el número de interrupciones en la opción *Numbers of interrupt generated by user-logic*.



FUENTE: Tomado de [4].

3. Continuar con los pasos del anexo B, ventana 7.

De acuerdo al diagrama de bloques de la Figura 1.4, se resaltan los elementos involucrados para este periférico (ver Figura 1.11). Luego de generar el periférico, estamos listos para revisar el contenido de la plantilla "user_logic.vhd", esta se puede encontrar siguiendo la ruta del mapa de archivos mostrado en la Figura 1.3.

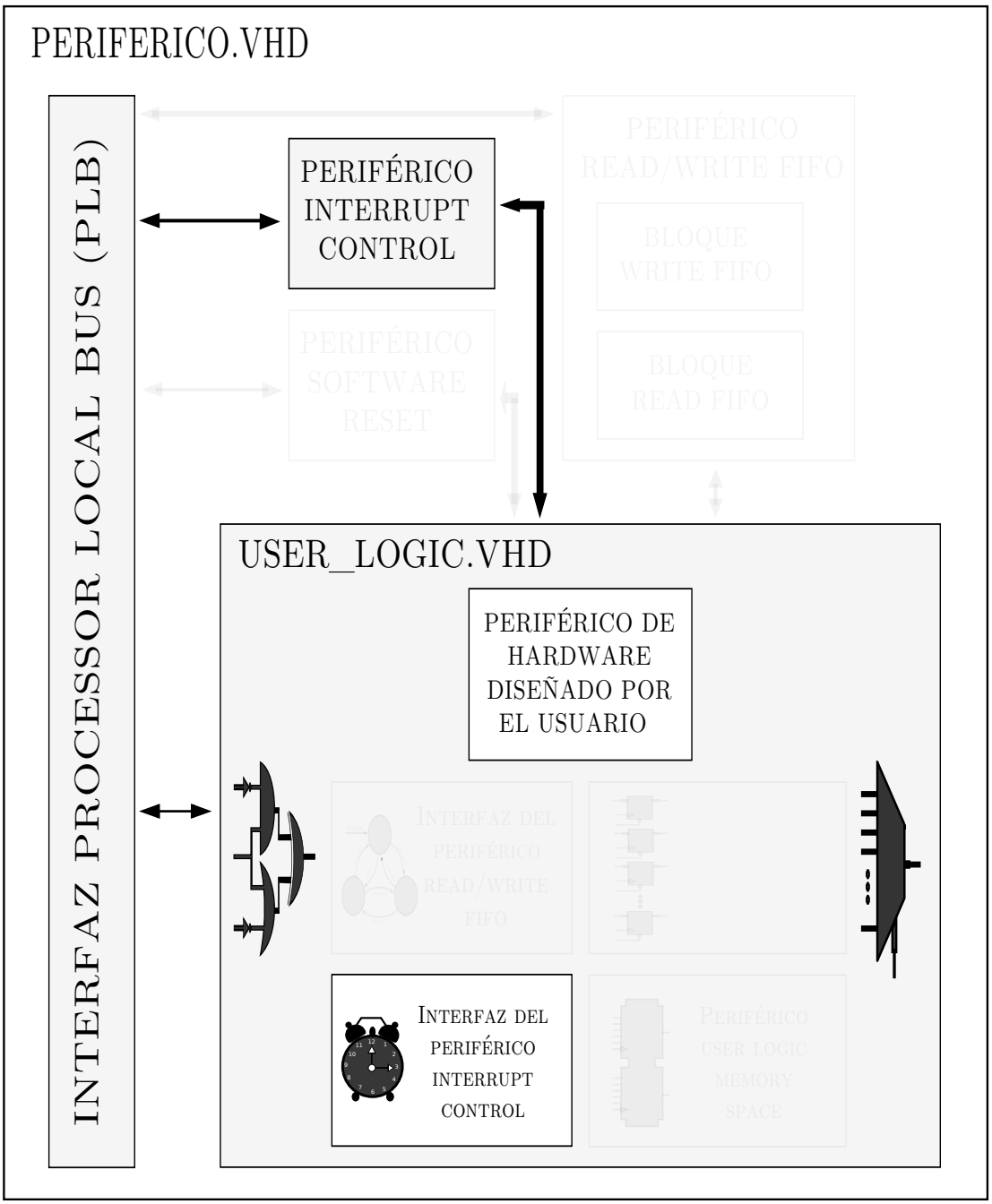


Figura 1.11: Bloques del periférico *interrupt control*.

FUENTE: El autor.

En el proceso de revisión de la plantilla se determinaron los circuitos y elementos que la conforman:

1. Una entidad.
2. Un temporizador encargado de generar interrupciones por intervalos de tiempo.
3. Un campo destinado para un periférico especial diseñado por el usuario.

Lo anterior se puede representar mediante un diagrama de bloques como se muestra en la Figura 1.12. Si revisamos la Figura 1.11 se puede apreciar que un nuevo bloque hace parte del circuito, este circuito es la representación física del periférico creado y es el encargado de recoger las interrupciones generadas por varias fuentes y aplicar priorización.

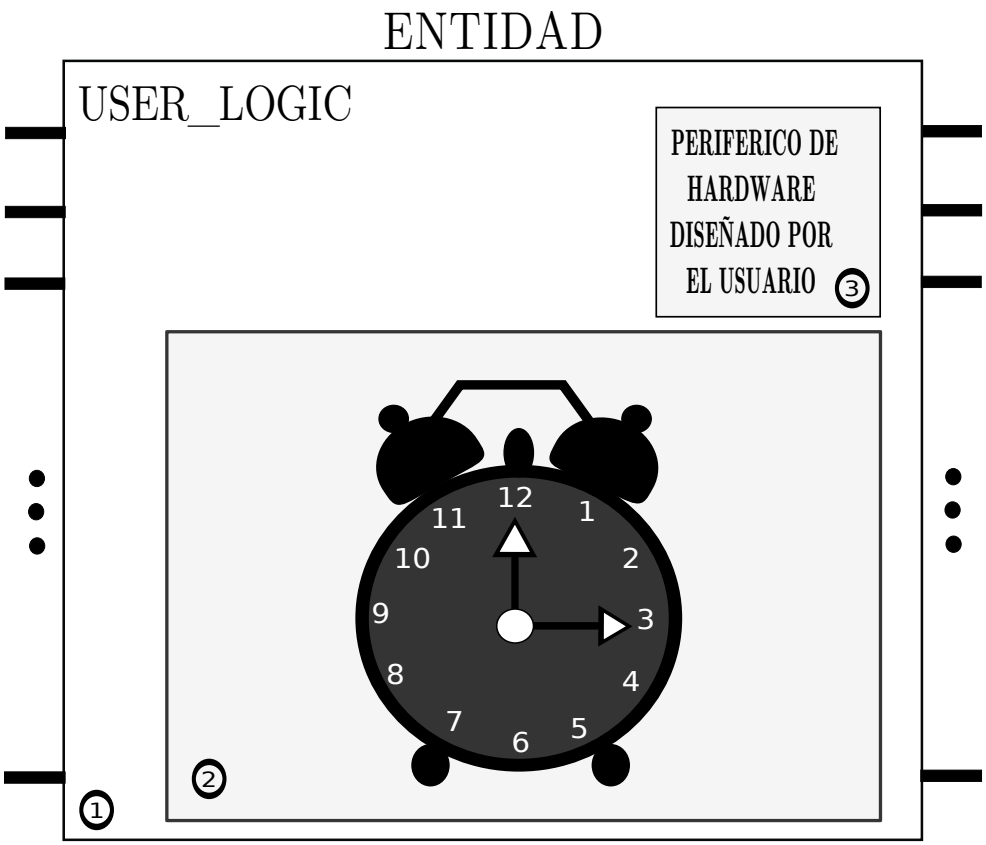


Figura 1.12: user_logic.vhd del priférico *interrupt control*.

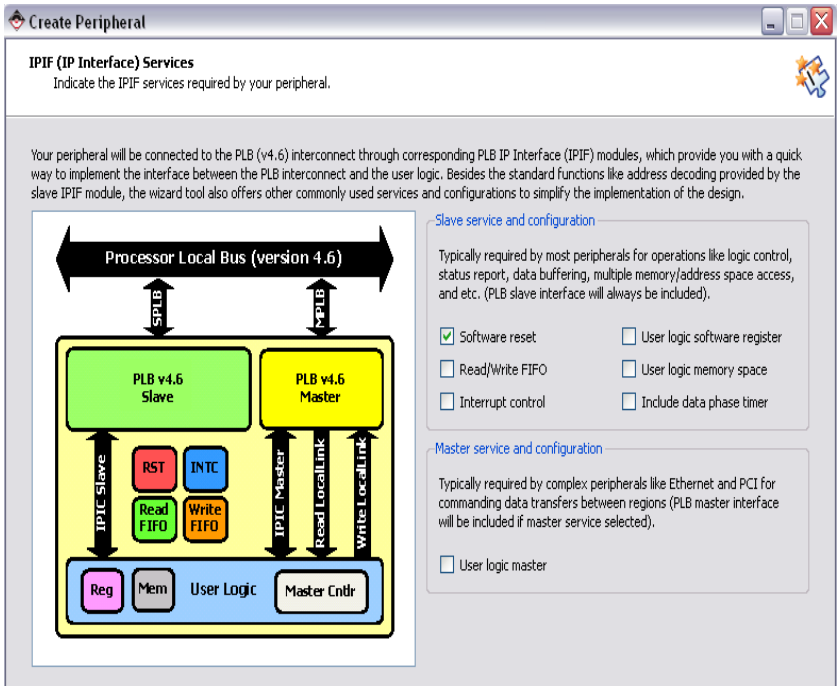
FUENTE: El autor.

Periférico *Software Reset*

A diferencia de los otros periféricos, el *Software Reset* cumple una función especial, ya que posee una dirección de sólo escritura y cuando una palabra específica es escrita en esta dirección, una señal de restablecimiento *reset* es generada para establecer en un valor determinado todos los periféricos que interactúan con él.

Es necesario generar el periférico en el proyecto para poder hacer uso de la plantillas "user_logic.vhd". Tomando como guía el anexo B, ventana 6, se procede como sigue:

1. Seleccionar la casilla del periférico *Software Reset*.



FUENTE: Tomado de [4].

2. Continuar con los pasos del anexo B, ventana 7.

De acuerdo al diagrama de bloques de la Figura 1.4, se resaltan los elementos involucrados para este periférico (ver Figura 1.13). Luego de generar el periférico, estamos listos para revisar el contenido de las plantilla, esta se puede encontrar encontrar siguiendo la ruta del mapa de archivos mostrado en la Figura 1.3.

Nota: *Este periférico no generó ninguna descripción de hardware en la plantilla "user_logic.vhd" de importancia para el usuario.*

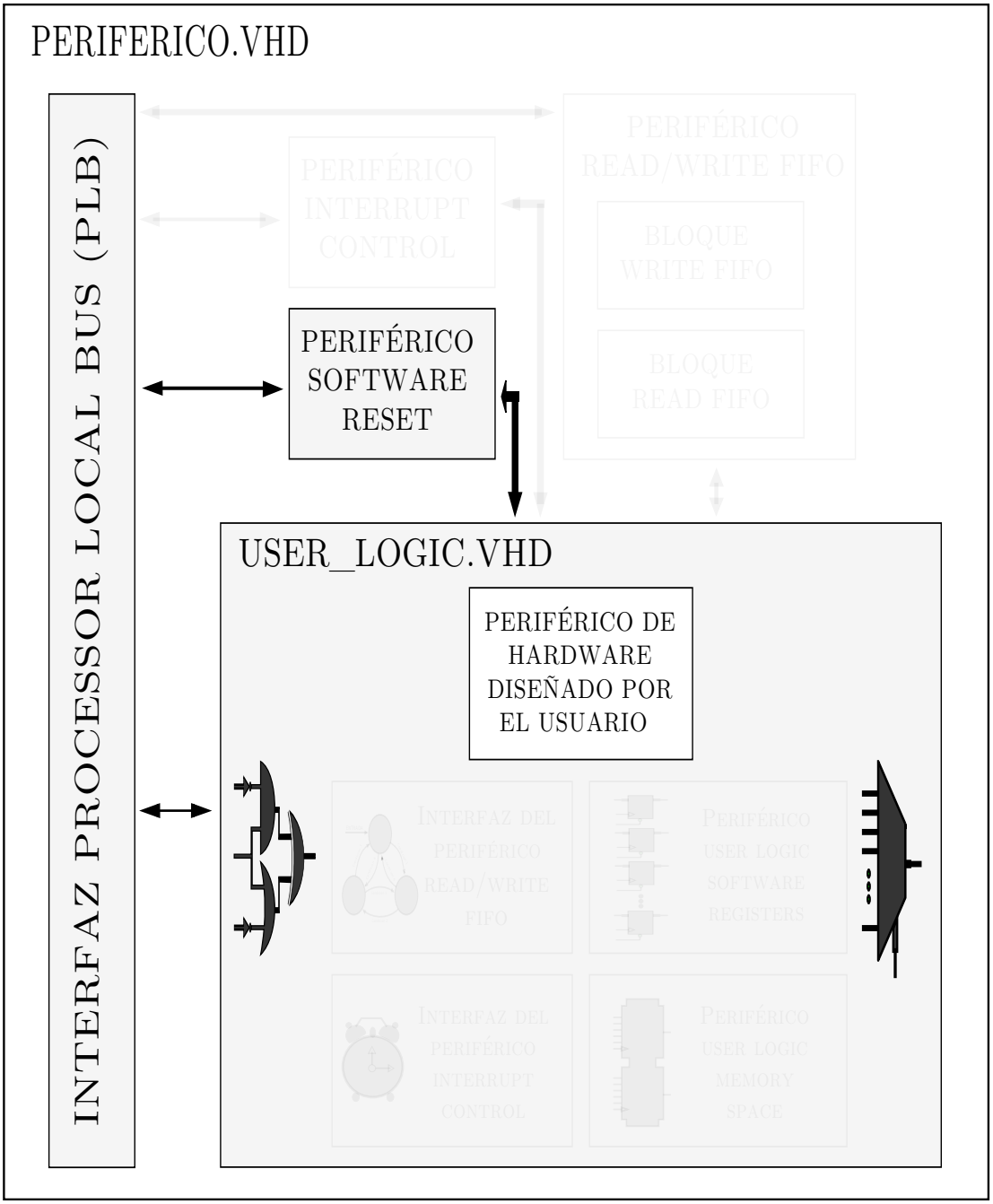


Figura 1.13: Bloques del periférico *software reset*.

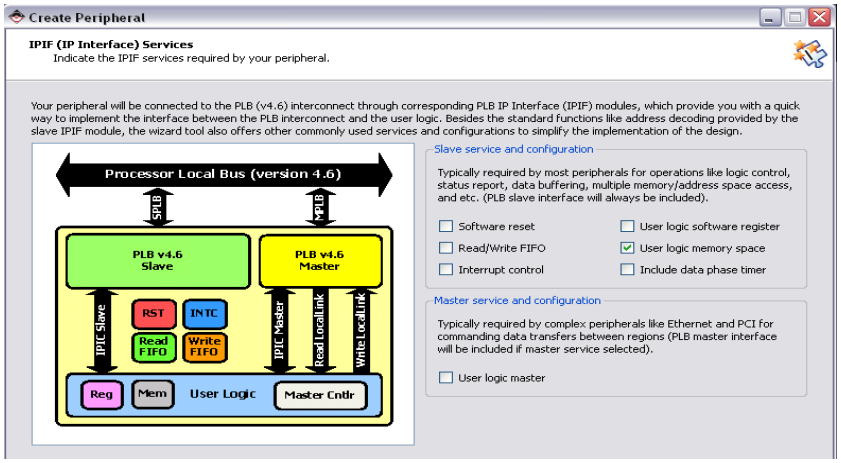
FUENTE: El autor.

Periférico *User Logic Memory Space*

El *User Logic Memory Space* es un bloque de memoria conocido comúnmente como *Random Access Memory Block (BRAM)*. Proporciona hasta 2048 posiciones de memoria con ancho de palabra de 32 bits para almacenamiento de datos. Su nombre se debe a que se puede acceder a cualquier sector de la memoria directamente con una dirección.

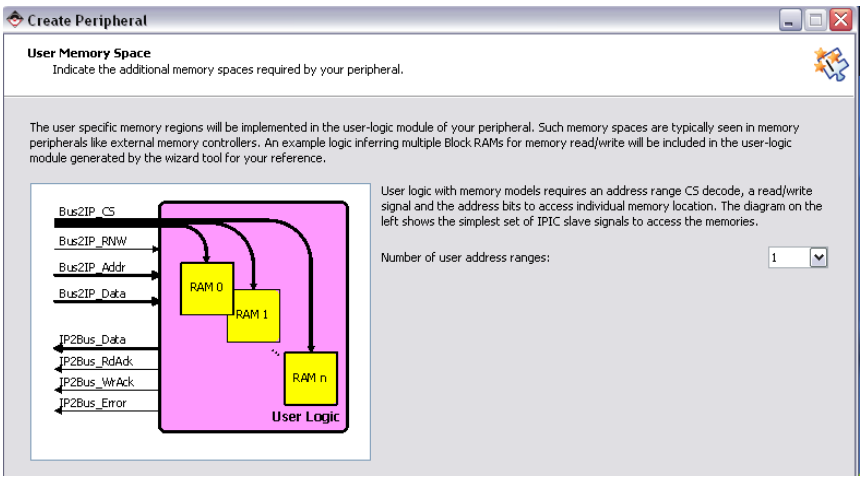
Es necesario generar el periférico en el proyecto para poder hacer uso de la plantillas “user_logic.vhd”. Tomando como guía el anexo B, ventana 6, se procede como sigue:

1. Seleccionar la casilla del periférico *User logic Memory Space*.



FUENTE: Tomado de [4].

2. Escoger las direcciones de memoria en la opción *Number of user address ranges*.



FUENTE: Tomado de [4].

3. Continuar con los pasos del anexo B, ventana 7.

De acuerdo al diagrama de bloques de la Figura 1.4, se resaltan los elementos involucrados para este periférico (ver Figura 1.14). Luego de generar el periférico, estamos listos para revisar el contenido de las plantilla, esta se puede encontrar encontrar siguiendo la ruta del mapa de archivos mostrado en la Figura 1.3.

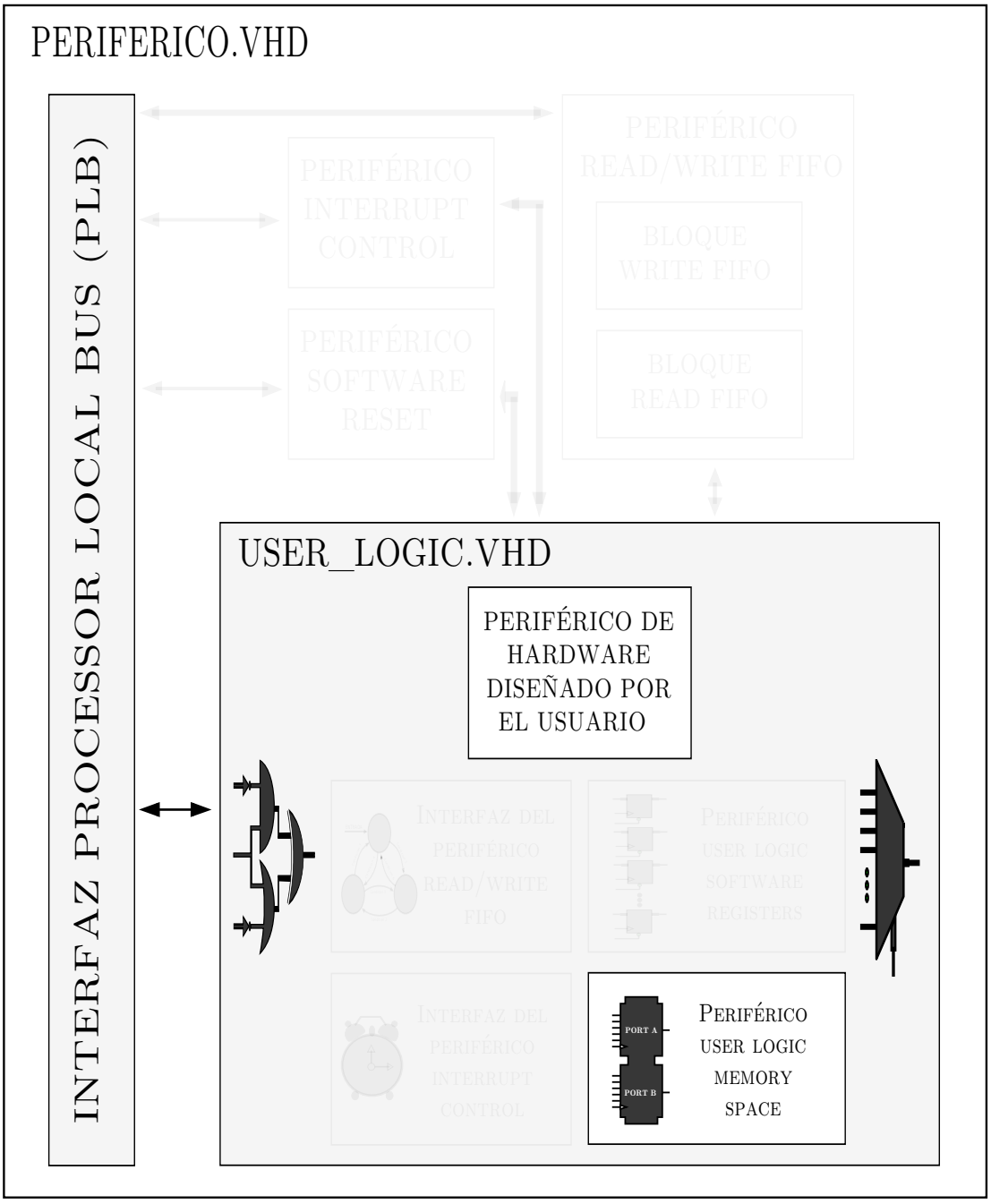


Figura 1.14: Bloques del periférico *user logic memory space*.

FUENTE: El autor.

En el proceso de revisión de la plantilla se determinaron los circuitos y elementos que la conforman:

1. Una entidad.
2. Un circuito combinacional a la entrada, para obtener la lectura de datos.
3. La cantidad BRAM seleccionados, para el almacenamiento de los datos.
4. Un Multiplexor que provee la escritura de los datos de salida.
5. Un campo destinado para un periférico diseñado por el usuario.

Lo anterior se puede representar mediante un diagrama de bloques como se muestra en la Figura 1.15. En esta oportunidad la representación física de este periférico se encuentra en el “user_logic.vhd”.

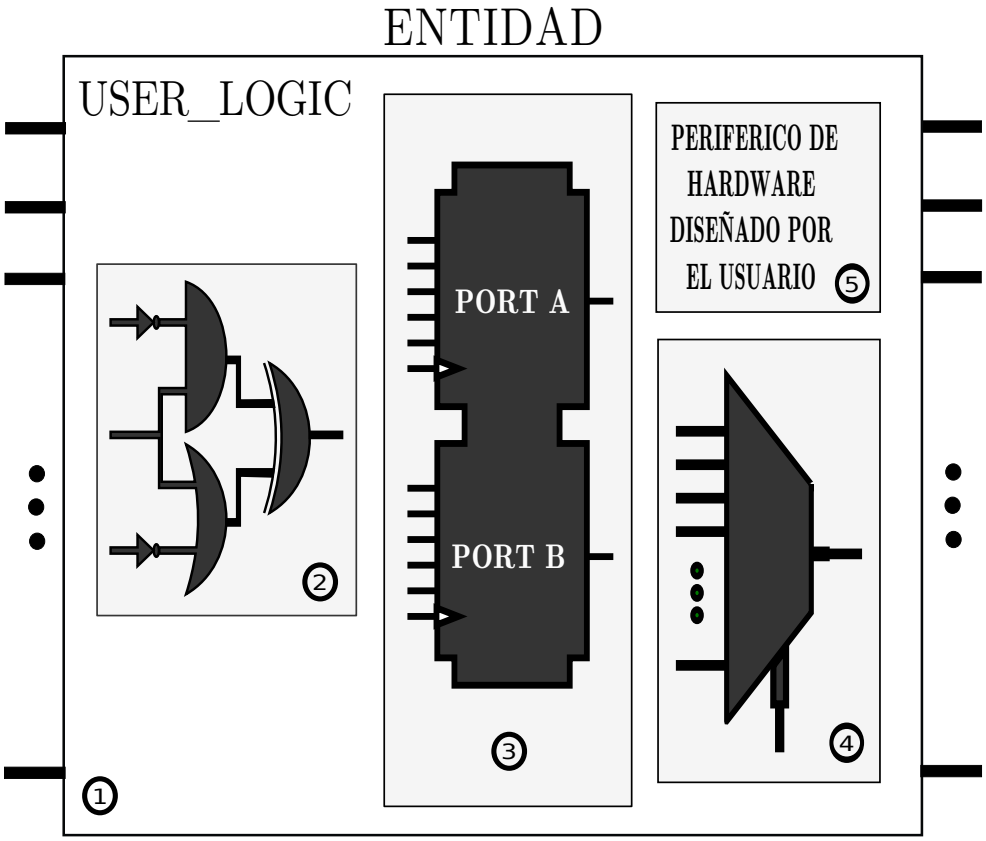


Figura 1.15: user_logic.vhd del priférico *user logic memory space*.

FUENTE: El autor.

1.5. *Software Development Kit*

El propósito de trabajar con la herramienta XPS es desarrollar el software necesario en el co-diseño. Una vez terminado el proceso de crear, adicionar y realizar las conexiones correspondientes de periféricos en XPS, se tiene la opción de comunicar rutinas programación hechas en software con la lógica de usuario desarrollada en hardware. Para este fin, la herramienta SDK permite realizar aplicaciones en lenguaje de programación de alto nivel, en las cuales se utilizan macros [1] que tienen como objetivo manejar la información que se lee o escribe en un periférico.

El anexo D corresponde a una guía que describe el procedimiento para crear un nuevo proyecto en SDK.

En el proceso de creación de un periférico en XPS se tiene la opción de generar las plantillas que contienen los macros, direcciones, desplazamientos, mascaras, etc, esenciales para el usuario a la hora de realizar una aplicación de software. Estas plantillas están ubicadas en el directorio del proyecto como se muestra en la Figura 1.16.

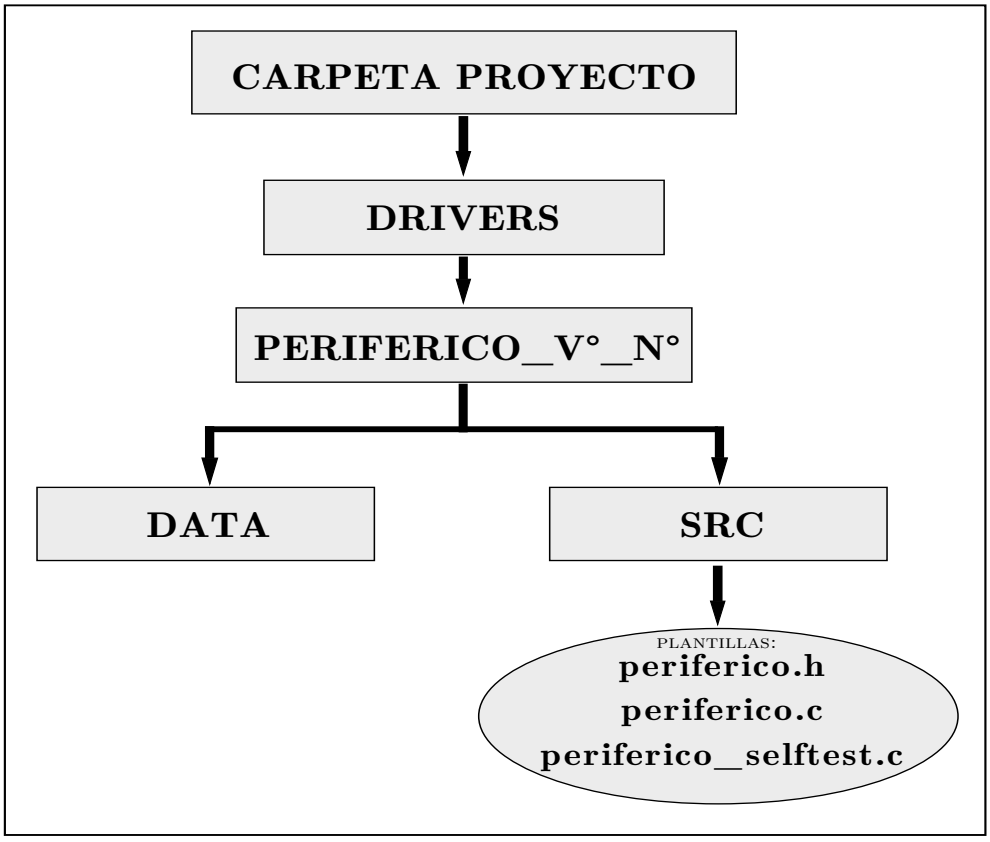


Figura 1.16: Mapa de archivos del software.

FUENTE: El autor.

[1] Funciones definidas por el fabricante o por el usuario.

Cada una de estas plantillas cumple la siguiente función:

- periférico.h
  Este archivo contiene las direcciones de desplazamiento de los registros implementados en el periférico, así como algunas máscaras comunes y sencillos macros o funciones para acceder a los registros.

- periférico.c
  En este archivo se encuentran definidos los cabeceros y funciones necesarias para acceder a los registros del periférico.

- periférico_selftest.c
  Este archivo es un ejemplo para autodiagnostico que contiene un código de prueba que permite verificar las diferentes características del periférico en uso.

Como se mencionó anteriormente, la comunicación entre hardware y el software es posible de realizar por medio de aplicaciones escritas en el lenguaje de programación de preferencia, a través de la herramienta de trabajo SDK, en estas aplicaciones se deben utilizar los macros definidos en la plantilla periferico.h. En la Tabla 1.1 se presenta un resumen de los macros usados en las diferentes aplicaciones de software empleadas en la validación de periféricos.

MACROS	**DESCRIPCIÓN**
XGpio_Initialize() XGpio_SetDataDirection() XGpio_DiscreteWrite() XGpio_DiscreteRead()	Funciones asociadas a los periféricos de propósito general
PERIFERICO_mWriteSlaveReg()	Función asociada al periférico *User Logic Software Register*
PERIFERICO_mResetReadFIFO() PERIFERICO_mReadFIFOEmpty() PERIFERICO_mReadFIFOOccupancy() PERIFERICO_mReadFromFIFO() PERIFERICO_mResetWriteFIFO() PERIFERICO_mWriteFIFOFull() PERIFERICO_mWriteFIFOVacancy() PERIFERICO_mWriteToFIFO()	Funciones asociadas al periférico *Read/Write FIFO*
PERIFERICO_mWriteReg() PERIFERICO_mReadReg() PERIFERICO_EnableInterrupt() PERIFERICO_Intr_DefaultHandler() microblaze_enable_interrupts() XIntc_RegisterHandler() XIntc_MasterEnable() XIntc_EnableIntr()	Funciones asociadas al periférico *Interrupt Control*
PERIFERICO_mReset()	Funciones asociadas al periférico *Software Reset*
PERIFERICO_mWriteMemory() PERIFERICO_mReadMemory()	Función asociada al periférico *User Logic Memory Space*

Tabla 1.1: Macros usados en las aplicaciones de software.

Procedimiento para la validación de periféricos

El proceso de validación se realizó teniendo en cuenta los siguientes pasos:

1. Creación de un nuevo proyecto en la herramienta de trabajo XPS.
2. Adición de periféricos de propósito general, periféricos con características y periféricos especiales al proyecto.
3. Exportación de cada proyecto a la herramienta de trabajo SDK.
4. Creación de la aplicación de software en SDK.
5. Implementación de cada proyecto en el sistema de desarrollo Spartan-6 SP506.

Periférico *User Logic Sotfware Register*

En este proyecto se propone leer el estado de *dip switch* y escribir su valor en *leds*. Como alternativa de solución a este problema se plantean las siguientes fases: En la primera fase de creará un sistema en la herramienta XPS que implementará un registro del periférico *User Logic Sotfware Register* y éste se conectará a los *leds* de propósito general, en la segunda fase se desarrollará en la herramienta SDK una aplicación de software que permitirá leer el estado de los *dip switch*, en la tercera fase se implementará el proyecto en el sistema de desarrollo Spartan SP506 y se verifica su funcionamiento por medio de *leds* y en consola a través del protocolo RS232.

La Figura 1.17 ilustra el flujo del proceso que se realiza con las partes implicadas en el proyecto.

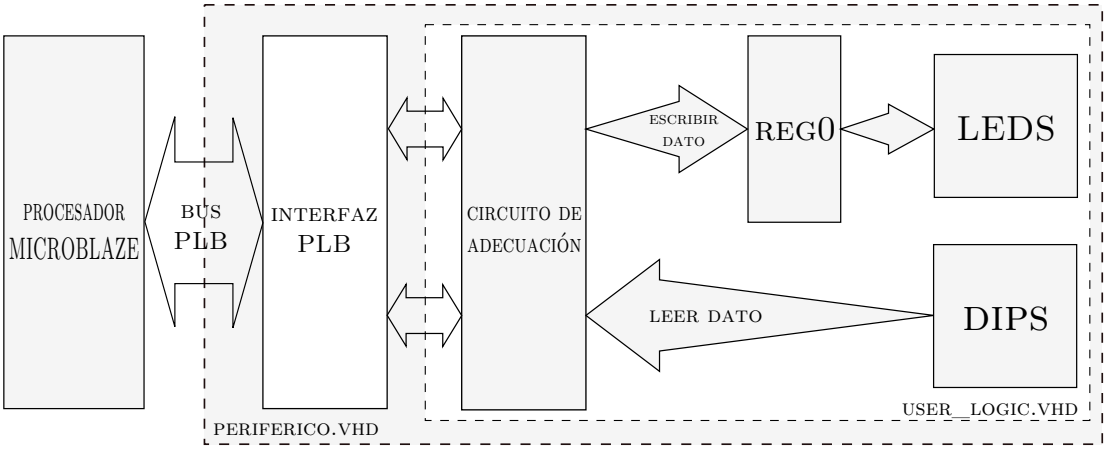


Figura 1.17: Flujo del proceso en la validación del periférico *user logic software register*.

FUENTE: El autor.

Modificaciones:
Estas modificaciones se realizaron con el objetivo de introducir cada uno de los elementos representados en

el flujo de la Figura 1.17, los cambios realizados en el hardware y el software se exponen en el Anexo E, sección E.1.

Implementación:
Como resultado de la implementación de este proyecto, se debe obtener en consola el valor de las entradas en la medida que éstas cambian, similar a como se muestra a continuación:

```
1 Consola@Linux:~$ Aplicacion: INICIO DE LA PRUEBA.
2 Consola@Linux:~$ Aplicacion: Se leyo el valor **dip_switch** = 0x00.
3 Consola@Linux:~$ Aplicacion: Se leyo el valor **dip_switch** = 0x01.
4 Consola@Linux:~$ Aplicacion: Se leyo el valor **dip_switch** = 0x02.
5 Consola@Linux:~$ Aplicacion: Se leyo el valor **dip_switch** = 0x03.
6 Consola@Linux:~$ Aplicacion: Se leyo el valor **dip_switch** = 0x04.
7 Consola@Linux:~$ Aplicacion: FIN DE LA PRUEBA.
```

Script 1.1: Salida en consola de la validación del periférico *user logic software register*.

Periférico *Read/Write FIFO*

En este proyecto se propone realizar una multiplicación entre dos números de 16 bits sin signo, que arrojarán como resultado una salida de 32 bits sin signo. Durante el proceso se hace uso del periférico *Read/Write FIFO* el cual cumple con la siguiente función: El bloque *Write FIFO* se carga con diferentes valores de ancho de palabra de 32 bits, estos valores son extraídos uno a uno de este bloque y se realiza la multiplicación luego de dividir cada valor de 32 bits en dos datos de 16 bits, finalmente los resultados de la multiplicación son almacenados en el bloque *Read FIFO* y se pueden verificar los resultados en consola a través del protocolo RS232.

La Figura 1.18 ilustra el flujo del proceso que se realiza con las partes implicadas en el proyecto.
Modificaciones:
Estas modificaciones se realizaron con el objetivo de introducir cada uno de los elementos representados en el flujo de la Figura 1.18, los cambios realizados en el hardware y el software se exponen en el Anexo E, sección E.2.

Implementación:
Como resultado de la implementación de este proyecto, se debe obtener la siguiente salida en consola:

```
1 Consola@Linux:~$ Aplicacion: INICIO DE LA PRUEBA.
2 Consola@Linux:~$ Aplicacion: Datos a multiplicar = 0x11.
3 Consola@Linux:~$ Aplicacion: Datos a multiplicar = 0x22.
4 Consola@Linux:~$ Aplicacion: Datos a multiplicar = 0x33.
5 Consola@Linux:~$ Aplicacion: Datos a multiplicar = 0x44.
6 Consola@Linux:~$ Aplicacion: Resultado de la multiplicacion = 0x01.
7 Consola@Linux:~$ Aplicacion: Resultado de la multiplicacion = 0x04.
8 Consola@Linux:~$ Aplicacion: Resultado de la multiplicacion = 0x09.
9 Consola@Linux:~$ Aplicacion: Resultado de la multiplicacion = 0x10.
10 Consola@Linux:~$ Aplicacion: FIN DE LA PRUEBA.
```

Script 1.2: Salida en consola de la validación del periférico *read/write fifo*.

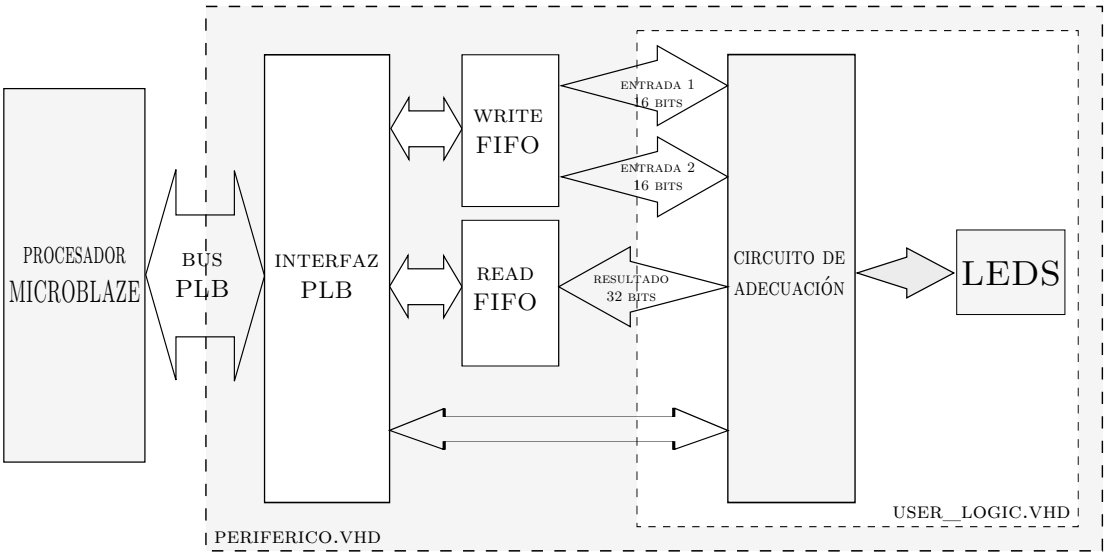


Figura 1.18: Flujo del proceso en la validación del periférico *read/write fifo*.

FUENTE: Adaptado de [6].

Periférico *Interrupt Control*

En este proyecto se propone realizar un temporizador para generar interrupciones. Durante el proceso se hace uso de los periféricos *Interupt Control* y *User Logic Software Register* los cuales cumplen las siguientes funciones: El temporizador utiliza dos registros, uno para almacenar el tiempo de retraso y el otro para iniciar, detener y verificar si el temporizador ha expirado. por otra parte, el modulo de interrupciones será el encargado de verificar si existe una interrupción y así indicar este suceso al usuario por medio de *leds* y en consola a través del protocolo RS232.

La Figura 1.19 ilustra el flujo del proceso que se realiza con las partes implicadas en el proyecto.

Modificaciones:
Estas modificaciones se realizaron con el objetivo de introducir cada uno de los elementos representados en el flujo de la Figura 1.19, los cambios realizados en el hardware y el software se exponen en el Anexo E, sección E.3.

Implementación:
Como resultado de la implementación de este proyecto, se debe obtener la siguiente salida en consola:

```
1 Consola@Linux:~$ Aplicacion: INICIO DE LA PRUEBA.
2 Consola@Linux:~$ Aplicacion: Habilitando interrupciones para el Procesador.
3 Consola@Linux:~$ Aplicacion: Registro del tempotizador en la tabla de vectores.
4 Consola@Linux:~$ Aplicacion: Inicializando la direccion del GPIO conectados a los LEDs.
5 Consola@Linux:~$ Aplicacion: Condicion 1, prender y apagar todos los LEDs.
6 Consola@Linux:~$ Aplicacion: Condicion 2, prender LEDs consecutivos.
7 Consola@Linux:~$ Aplicacion: Inicializando el controlador princpal de interrupcciones.
```

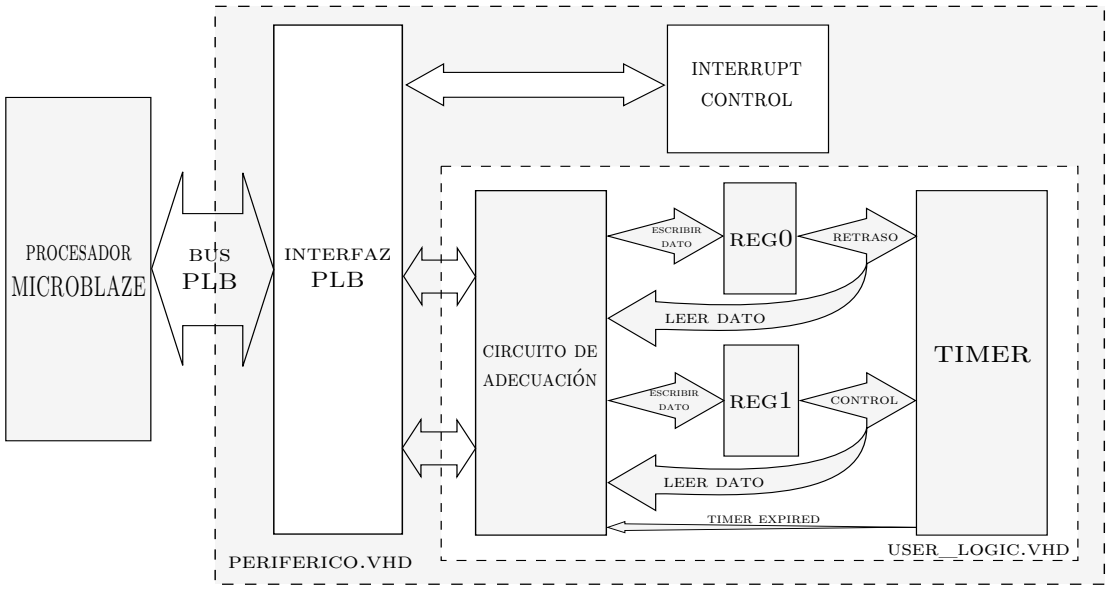


Figura 1.19: Flujo del proceso en la validación del periférico *interrupt control*.

FUENTE: Adaptado de [6].

```
 8 Consola@Linux:~$ Aplicacion: Configurando los ciclos y ejecutando la temporizacion.
 9 Consola@Linux:~$ Aplicacion: Habilitando el temporizador en el periferico.
10 Consola@Linux:~$ Aplicacion: Ejecutando la temporizacion.
11 Consola@Linux:~$ Aplicacion: HE ENTRADO AL MANEJADOR DE INTERRUPCIONES.
12 Consola@Linux:~$ Aplicacion: Obteniendo la interrupcion del temporizador.
13 Consola@Linux:~$ Aplicacion: El valor de la interrupcion es IpStatus = 1.
14 Consola@Linux:~$ Aplicacion: Condicion 1, perender y apagar todos los LEDs.
15 Consola@Linux:~$ Aplicacion: Condicion 2, perender LEDs consecutivos.
16 Consola@Linux:~$ Aplicacion: El valor de contador es = 1.
17 Consola@Linux:~$ Aplicacion: Cargando y ejecutando la temporizacion.
18 Consola@Linux:~$ Aplicacion: Condicion 2, perender LEDs consecutivos.
19 Consola@Linux:~$ Aplicacion: Interrupcion tomada en 1 segundos.
20 Consola@Linux:~$ Aplicacion: FIN DE LA PRUEBA.
```

Script 1.3: Salida en consola de la validación del periférico *interrupt control*.

Periférico *Software Reset*

En este proyecto se propone realizar una máquina de estados (FSM) para una realizar el efecto de un juego de luces.

El juego de luces estará conformado por tres secuencias que se describen a continuación:
Primera secuencia: ⇒ ”00011000”⇒ ”00100100”⇒ ”01000010”⇒ ”100000001”⇒ se repite.
Segunda secuencia: ⇒ ”11111111”⇒ ”00000000”⇒ ”11111111”⇒ ”00000000”⇒ se repite.
Tercera secuencia: ⇒ ”10000001”⇒ ”01000010”⇒ ”00100100”⇒ ”00011000”⇒ se repite.

El usuario puede elegir el orden de las secuencia por medio de la combinación de dos *dip switch*, si la combinación es “00” entonces se ejecuta la primera secuencia, si “01” entonces se ejecuta la segunda secuencia, si “10” entonces se ejecuta la tercera secuencia y si “11” entonces la máquina de estados se restablecerá hasta que nuevamente se introdusca una operación válida.

Durante el proceso se implementará un registro del periférico *User Logic Software Register* del cual se tomarán 2 bits para almacenar el estado de los *dip switch* y se implementará el periférico *Software Reset* con el objetivo de obtener un valor de restablecimiento ejecutado desde el software.

La Figura 1.20 ilustra el flujo del proceso que se realiza con las partes implicadas en el proyecto.

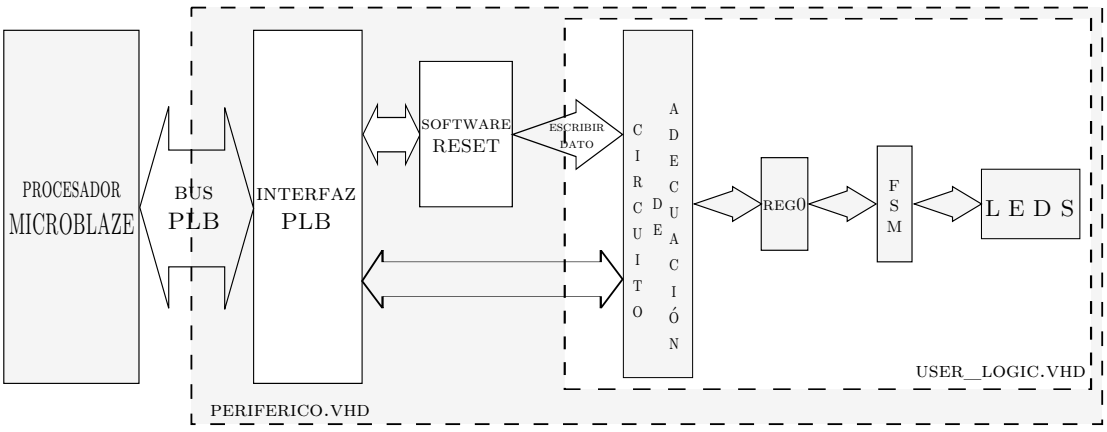


Figura 1.20: Flujo del proceso en la validación del periférico *software reset*.

FUENTE: El autor.

Modificaciones:
Estas modificaciones se realizaron con el objetivo de introducir cada uno de los elementos representados en el flujo de la Figura 1.20, los cambios realizados en el hardware y el software se exponen en el Anexo E, sección E.4.

Implementación:
Como resultado de la implementación de este proyecto, se debe obtener la siguiente salida en consola:

```
1 Consola@Linux:~$ Aplicacion: INICIO DE LA PRUEBA.
2 Consola@Linux:~$ Aplicacion: El estado de los **dip_switch** es = 0x00.
3 Consola@Linux:~$ Aplicacion: Escribiendo el valor 0x00 al registro del periferico.
4 Consola@Linux:~$ Aplicacion: El estado de los **dip_switch** es = 0x01.
5 Consola@Linux:~$ Aplicacion: Escribiendo el valor 0x02 al registro del periferico.
6 Consola@Linux:~$ Aplicacion: El estado de los **dip_switch** es = 0x02.
7 Consola@Linux:~$ Aplicacion: Escribiendo el valor 0x02 al registro del periferico.
8 Consola@Linux:~$ Aplicacion: El estado de los **dip_switch** es = 0x03.
9 Consola@Linux:~$ Aplicacion: Escribiendo el valor 0x03 al registro del periferico.
10 Consola@Linux:~$ Aplicacion: FIN DE LA PRUEBA.
```

Script 1.4: Salida en consola de la validación del periférico *software reset*.

Periférico *User Logic Memory Space*

En este proyecto se propone crear el periférico *user logic memory space* y entre sus características escoger 256 direcciones para almacenamiento. Con el objetivo de verificar su funcionamiento se hace uso de una aplicación de software que lleva a cabo el proceso de lectura/escritura en el Bloque RAM.

La Figura 1.21 ilustra el flujo del proceso que se realiza con las partes implicadas en el proyecto.

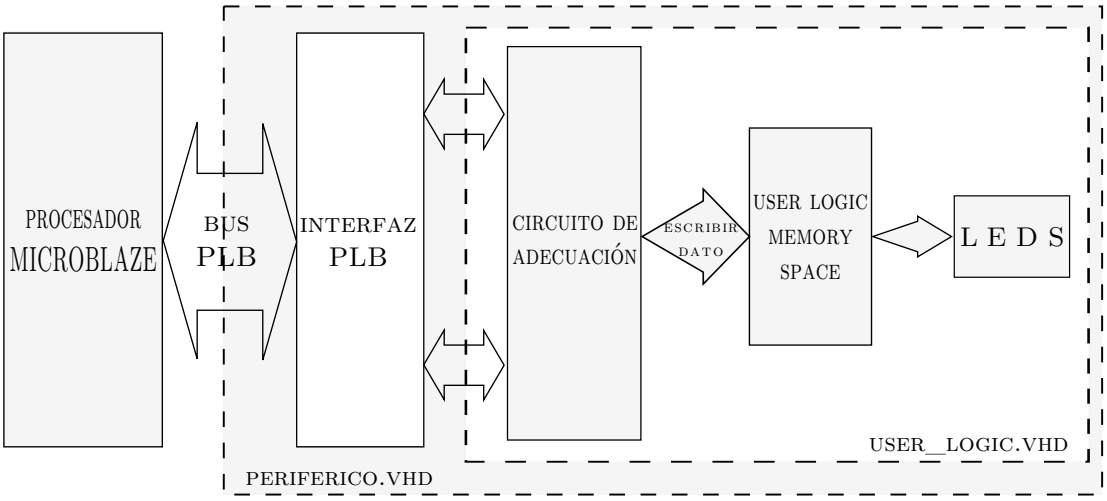


Figura 1.21: Flujo del proceso en la validación del periférico *user logic memory space*.

FUENTE: El autor.

Modificaciones:
Estas modificaciones se realizaron con el objetivo de introducir cada uno de los elementos representados en el flujo de la Figura 1.21, los cambios realizados en el hardware y el software se exponen en el Anexo E, sección E.5.

Implementación:
Como resultado de la implementación de este proyecto, se debe obtener la siguiente salida en consola:

```
1 Consola@Linux:~$ Aplicacion: INICIO DE LA PRUEBA.
2 Consola@Linux:~$ Aplicacion: Escribiendo datos en el BRAMs.
3 Consola@Linux:~$ Aplicacion: La Escritura/Lectura en el BRAM es correcta.
4 Consola@Linux:~$ Aplicacion: FIN DE LA PRUEBA.
```

Script 1.5: Salida en consola de la validación del periférico *user logic memory space*.

CAPÍTULO 2

Desarrollo de *Drivers* en Linux

2.1. Introducción

A lo largo de este capítulo se presenta de forma detallada la estrategia empleada para llevar a cabo el desarrollo de *drivers* en el sistema operativo Linux y con el objetivo de verificar el funcionamiento de cada *driver*, se hará uso de la plataforma de desarrollo SIE. A continuación se exponen algunos detalles de la plataforma y posteriormente se presentan los pasos de la metodología implementada.

2.2. Aporte a la plataforma de desarrollo SIE

Existen en la actualidad un gran número de fabricantes que elaboran tarjetas de desarrollo para uso en sistemas embebidos. Entre los diferentes fabricantes se destaca la empresa Xilinx, ya que esta es la mayor empresa de investigación y desarrollo de FPGAs. Dentro de las familias de dispositivos creados por Xilinx se encuentran las tarjetas de desarrollo Spartan, las cuales son de fácil manejo. Esto se debe a que Xilinx ofrece al usuario un software de trabajo llamado *Embedded Development Kit (EDK)* para el desarrollo de sistemas embebidos. Por medio de esta herramienta es posible de realizar co-diseño de nucleos IP (IP cores) en los lenguajes de descripción de hardware VHDL. Lo anterior es realizable por medio de la interacción de dos grandes elementos que trabajan en conjunto, estos son: el FPGA y microprocesador Microblaze o powerPC en el caso de los sistemas de desarrollo de mayores prestaciones.

En contraste, el sistema de desarrollo SIE frente a las familias de dispositivos ofrecidos por Xilinx para uso en sistemas embebidos, comparte en menor proporción similitudes de diseño. SIE se acomoda a las necesidades del usuario por ser asequible y por su característica filosófica de hardware *copyleft* y software libre, por tanto quien lo desee puede modificarla. La principal falencia de SIE es que esta no cuenta con un software de desarrollo para realizar co-diseño con núcleos IP. Al ser su arquitectura diferente a los dispositivos de Xilinx, es necesario desarrollar una metodología que permita la interacción entre su procesador y el FPGA.

2.3. Sistema de desarrollo SIE

La plataforma de desarrollo SIE es una herramienta elaborada con el propósito de satisfacer la necesidad de los desarrolladores de hardware, permitiendo la creación de aplicaciones comerciales bajo la licencia Creative Commons BY -SA, la cual permite la distribución y modificación del diseño [7][8].

Para llevar a cabo este proyecto se seleccionó el sistema de desarrollo SIE ya que es el principal recurso a disposición en las materias del área de sistemas digitales y por ser una plataforma de propósito general que corre un sistema operativo basado en Linux el cual lo hace apto para poner en marcha aplicaciones que involucren *drivers*, y aplicaciones que hagan uso de FPGAs.

Esta plataforma tiene como componentes los elementos de la Tabla 2.1.

ELEMENTOS	**DESCRIPCIÓN**
XBURST JZ4725	Procesador @336MHz.
SDRAM	Memoria con 64 MBytes de capacidad.
FPGA XC3S100E_VQ100	Proporciona 25 señales digitales de Entrada/Salida para propósito general en un rango digital de 0-3.3V.
ADC	10 canales de entradas analógicas en un rango 0-3.3V.
USB	Este puerto puede ser usado como Ethernet o como dispositivo de consola serial.
Puerto Micro SD	Almacenamiento masivo.
Audio Estéreo	Con líneas de Entrada/Salida.
puerto I2C	Puerto de comunicación en serie.
RS-232	Puerto serial UART.

Tabla 2.1: Componentes principales de SIE.

El diagrama de bloques del sistema de desarrollo SIE se presenta en la Figura 2.1. En él se resumen los recursos del sistema y se resaltan los componentes utilizados en la realización de este proyecto.

Entre los dispositivos usados se encuentra el FPGA, el cual dispone de recursos lógicos con los que se implementaron los periféricos de hardware, el procesador JZ4725 en el cual se implementaron los *drivers* y sus aplicaciones, entre otros elementos como los GPIOs y el puerto USB que permite desplegar información del sistema o hacer pruebas.

2.4. Construcción de Periféricos en SIE

Generalmente algunas aplicaciones necesitan de ciertos periféricos especiales que les permitan cumplir con sus restricciones temporales, es decir, se crean tareas de hardware complementarias a las tareas que realiza el software, para así cumplir con las restricciones de diseño del dispositivo en uso. Estas tareas o funciones son implementadas en periféricos externos al procesador. En muchos casos si no existe un *chip* comercial que pueda realizar la tarea, entonces es necesario implementarla en un dispositivo lógico programable. En el

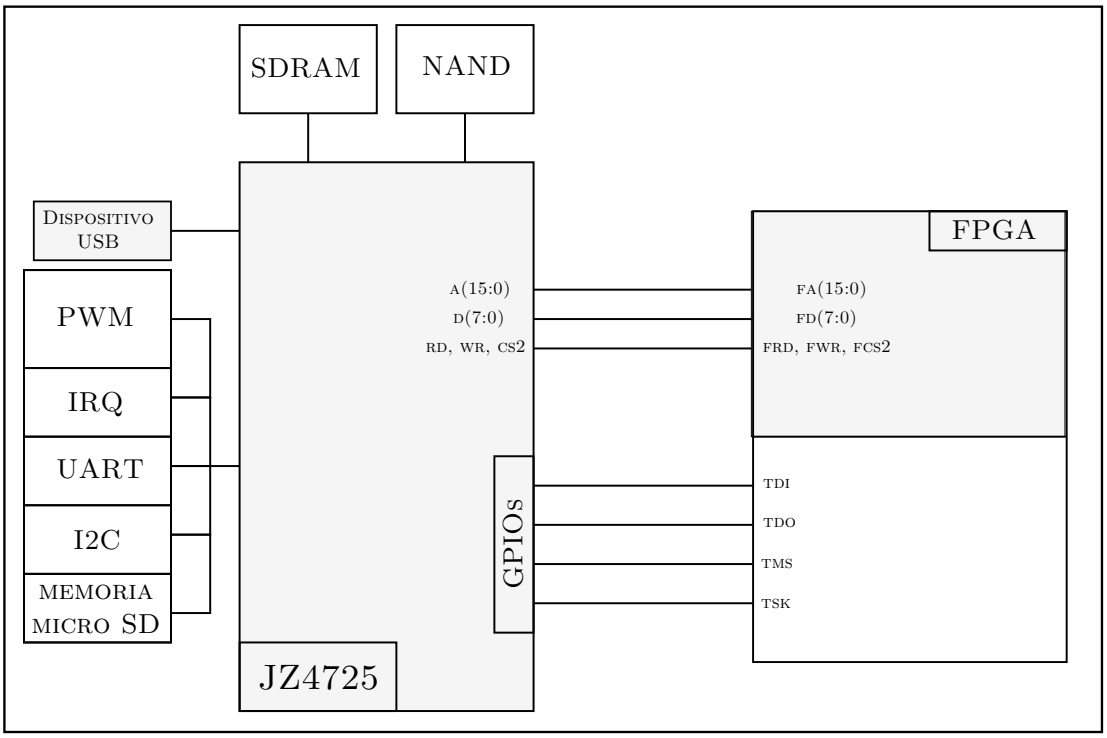


Figura 2.1: Diagrama de bloques SIE.

FUENTE: Adaptado de [8].

trascurso de este capítulo se dará a conocer el protocolo de comunicación entre el procesador y el FPGA de la tarjeta SIE, lo cual a su vez dará las pautas para armar nuestro propio nucleo IP.

Comunicación en el sentido Procesador-Periférico

Inicialmente es necesario comprender el funcionamiento de la arquitectura diseñada para SIE. La Figura 2.2 muestra el diagrama de bloques de la arquitectura básica que lleva a cabo la comunicación de tareas Hardware-Software en el sentido procesador- periférico.En ella se observa que el procesador maneja tres buses de comunicación con el FPGA, estos son:

- **Bus de datos:** Es un bus bidireccional por donde se realiza el intercambio de información.
- **Bus de direcciones:** Este bus controlado por el procesador y es utilizado para direccionar un periférico a una determinada funcionalidad del mismo.
- **Bus de control:** Maneja las señales necesarias para indicarle a los periféricos el tipo de comunicación que se realiza ya sea lectura o escritura).

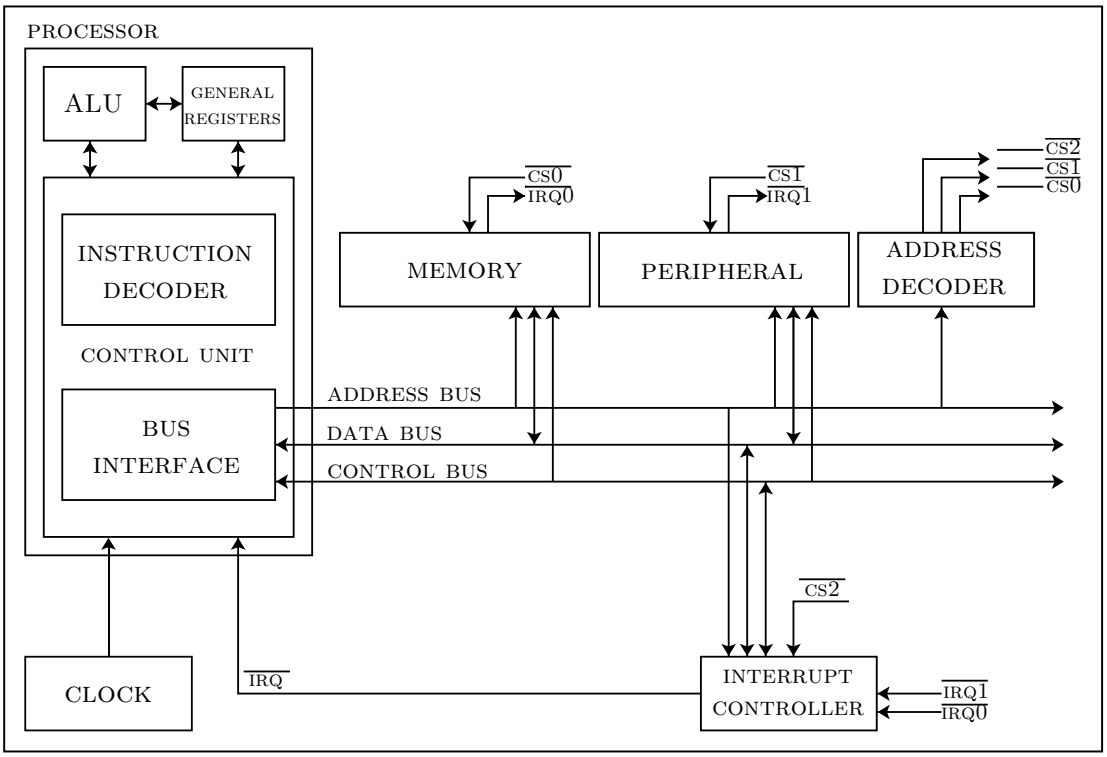


Figura 2.2: Arquitectura básica hardware/software

FUENTE: Adaptado de [8].

Controlador de memoria externa

El procesador posee un bloque interno llamado controlador de memoria externa EMC (*external memory controller*) con el cual se pueden administrar memorias estáticas. Su función es generar las señales CS2, RDWR y WE0 para establecer la comunicación con un periférico. La Figura 2.3 muestra el diagrama de tiempos de las señales del bus de control para un ciclo de lectura y escritura en el procesador.
En esta oportunidad el objetivo de usar el EMC no es para manejar memorias, solo se hará uso de éste para proveer las señales del bus de control a los periféricos que se van a implementar en el FPGA. Su proceso de configuración está explicado en detalle en [9].

Sin embargo, no basta sólo con utilizar estas señales, hay la necesidad de crear un hardware que haga uso de estas señales de forma eficiente para así asegurar una comunicación fiable y libre de errores.

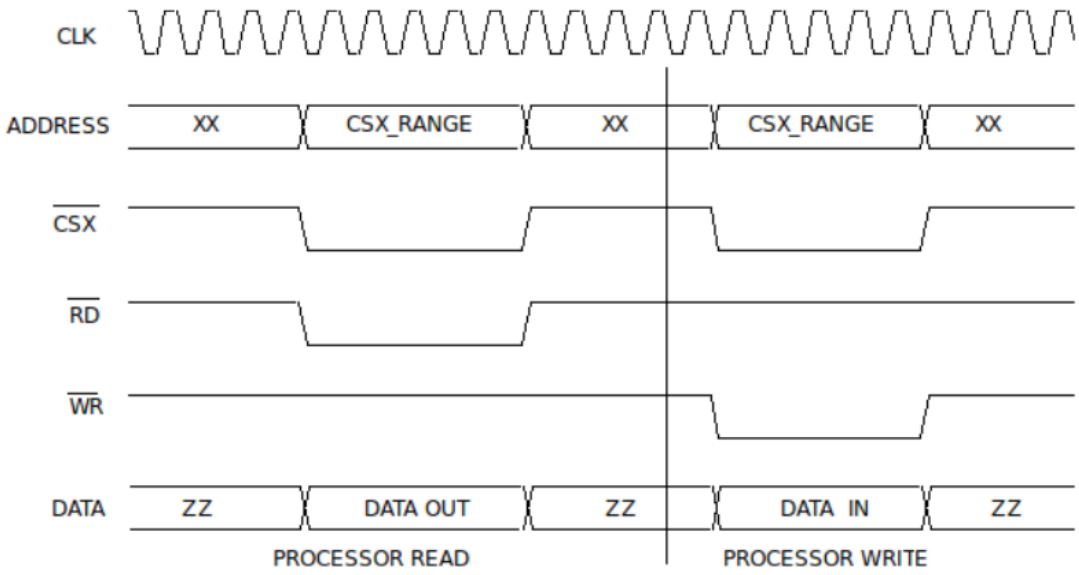


Figura 2.3: ciclo de lectura y escritura.

FUENTE: Tomado de [8].

Implementación de periféricos en el FPGA

La Figura 2.4 muestra el diagrama de bloques de la interfaz necesaria para poder comunicar un grupo de periféricos o tareas de hardware con el procesador. Con el propósito de entender el funcionamiento del hardware representado en el diagrama, se resaltaron por colores los módulos que interactúan en un mismo proceso.

- **Bloque de escritura en color gris:** Conformado por los módulos sincronizador de datos, sincronizador de direcciones, decodificador de direcciones, generador de pulsos de restablecimiento y generador de pulsos de escritura.
- **Bloque de lectura en color verde:** Conformado por los módulos decodificador de direcciones, generador de pulsos de lectura y buffer tercer estado.
- **Bloque periféricos en color azul:** Representa los diferentes tipos de periféricos a implementar.
- **Bloque adicional en color amarillo:** Este bloque es la representación de diferentes módulos que pueden ser añadidos para realizar una tarea específica en el periférico.

Los módulos **sincronizador de datos** y **sincronizador de direcciones** son los encargados de sincronizar la información del bus de datos y el bus de direcciones, este proceso se realiza debido a que el procesador utiliza una frecuencia de trabajo diferente a la frecuencia de trabajo del FPGA y como el objetivo final es el FPGA, hay que garantizar que la información puede ser usada y no se van a producir problemas de metaestabilidad.

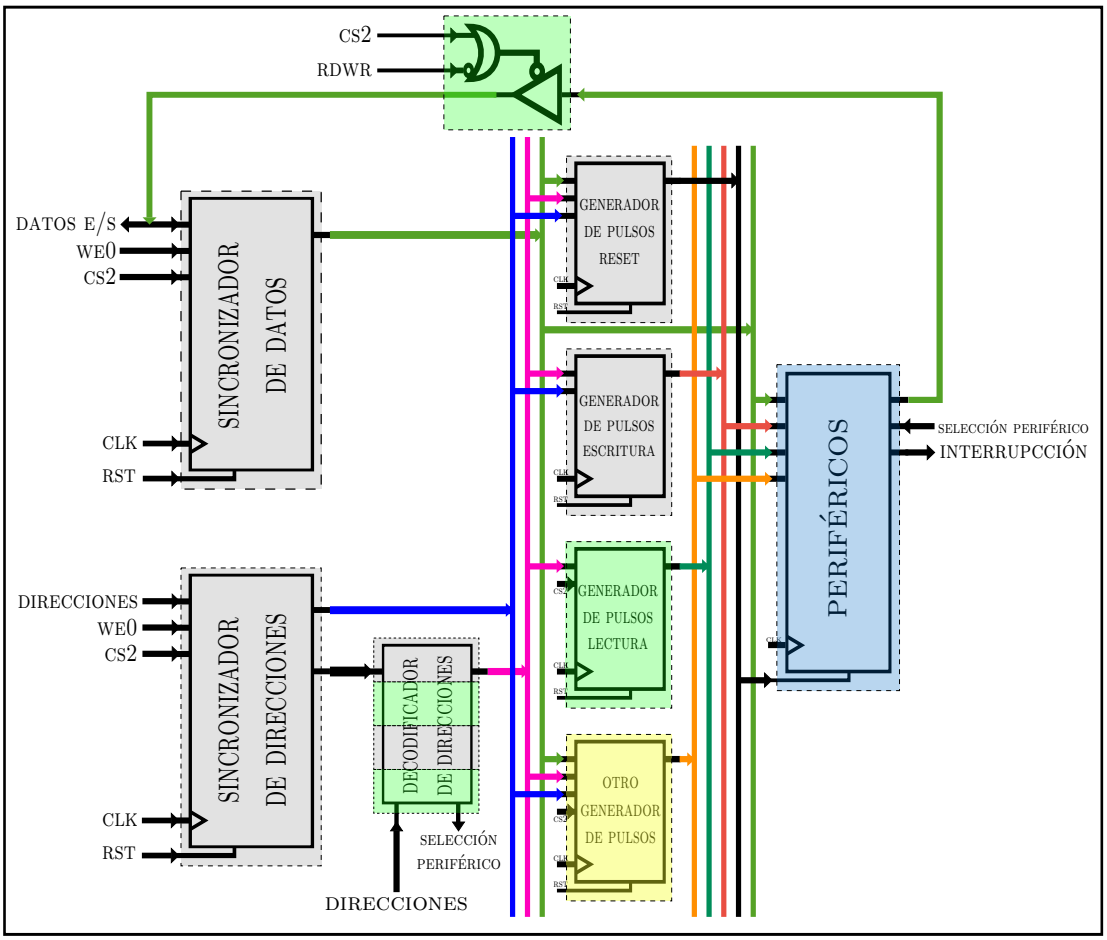


Figura 2.4: Diagrama de bloques para comunicar tareas entre el hardware y el software.

FUENTE: El autor

Los módulos **generador de pulsos de restablecimiento, generador de pulsos de escritura** y **generador de pulsos de lectura**, tienen la función de generar señales de habilitación para efectuar en el periférico un proceso de escritura, lectura o restablecerlo a un valor.

El modulo **buffer tercer estado** es implementado para evitar el choque datos de Entrada/Salida que maneja el bus de datos y garantizar que mientras el procesador no se comunique con los periféricos estos permanezcan en estado de alta impedancia; esto es necesario para evitar corto-circuitos originados por diferentes niveles lógicos en el bus de datos.

Fuentes de Periféricos en VHDL

El proceso de elaboración de periféricos en SIE se realizó teniendo en cuenta la limitación de los recursos disponibles en el FPGA spartan 3E. En el Anexo F se encuentran las fuentes en lenguaje de descripción de

hardware HDL de los periféricos creados, con sus respectivos diagramas de bloques y la información necesaria que permite hacer entendible su funcionamiento.

Dentro del listado de periféricos con características trabajados en el capítulo anterior se encuentra el llamado controlador de interrupciones. Este periférico en SIE se trabaja directamente desde el procesador y será estudiado más adelante en la sección 2.8.

Aplicaciones en el espacio del usuario

Toda aplicación de software creada desde el espacio del usuario cuya finalidad sea la gestión del hardware, debe respetar los niveles de abstracción de la arquitectura de Linux tal y como se observa en la Figura 2.5 Si se desea lograr que una aplicación de software emplee un periférico, es necesario usar un *driver* dentro del sistema operativo que acceda correctamente el mismo.

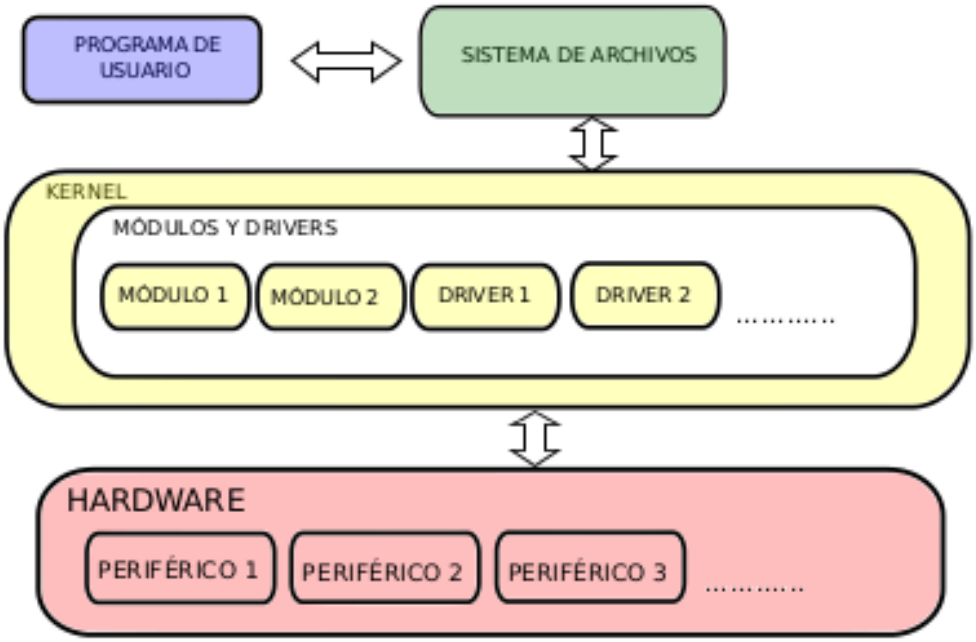


Figura 2.5: Niveles de abstracción.

FUENTE: Tomado de [10].

Las aplicaciones de usuario se realizaron en tres etapas descritas a continuación:

1. En la primera etapa cada aplicación implementa desde el espacio del usuario todas las funciones de bajo nivel necesarias para administrar el hardware. Desde el espacio del usuario la información es manipulada a través de registros que se acceden mediante el sistema operativo que posee el procesador. El sistema operativo administra los recursos del procesador, por tanto para hacer uso de los registros utiliza un *driver* ubicado en la ruta /dev/mem, este archivo administra la memoria física del procesador y por medio de comandos y funciones soportadas por el sistema operativo es posible leer o escribir en

él. El principal inconveniente radica en que el sistema operativo maneja direcciones de memoria virtual que están conectadas con una dirección de memoria física y no se ha establecido alguna relación entre las dos. Una de las soluciones es usar el llamado al sistema *mmap*, que ofrece un sistema de referencia entre las direcciones virtuales y las direcciones físicas. Una vez realizado lo anterior, el usuario que realiza la aplicación es libre de leer o escribir el hardware por medio de punteros al mapa de memoria creado por *mmap*. En [10] se encuentra la explicación en detalle de cómo manipular registros por medio de *mmap*.

2. En la segunda etapa las aplicaciones usan las llamadas al sistema *open*, *closed*, *read*, *write* y *lseek* para escribir los registros del procesador, en esta oportunidad se crea un *driver* que implementa las funciones de bajo nivel en el espacio del *kernel* logrando con esto un mejor desempeño.

3. En la tercera etapa las aplicaciones utilizan funciones. Estas funciones se crearon con el objetivo de mostrarle al usuario de forma sencilla, cómo intercambiar información con el periférico. Para hacer uso de las funciones es necesario trabajar con cuatro archivos, los cuales son:

   - **aplicacion.h:** En este archivo plantilla se definen las direcciones de desplazamiento, así como también las funciones que permiten el acceso al periférico.
   - **aplicacion.c:** En este archivo plantilla se encuentran las rutinas que se ejecutan cuando se hace uso de las funciones definidas en aplicación.c.
   - **prueba.c:** Esta fuente contiene el código de ejemplo para la prueba de las diferentes características del periférico.
   - **Makefile:** Por medio de este archivo se compilan las fuentes y se crea el ejecutable de prueba.

En el Anexo G se encuentra el código fuente final creado en la tercera etapa, para la gestión de los periférico desde el usuario.

2.5. *Driver* en el espacio del *kernel*

La función de un *driver* es generar los mecanismos de acceso a un recurso del sistema de cómputo. En el caso de un periférico, esto implica el acceso a los registros que controlan su funcionamiento y los registros por medio de los cuales se intercambia información. Adicionalmente, dentro del sistema operativo Linux, todos los recursos son abstraídos como archivos, por lo tanto, el *driver* también tiene que llevar a cabo las acciones adecuadas en el momento que un usuario o un proceso acceda a dicho archivo.

Desde el punto de vista del usuario, o de un proceso que corre en el sistema operativo, el periférico aparece como un archivo cuya longitud es la cantidad de registros del periférico y para su acceso se deben ejecutar los llamados al sistema open, close, write, read, lseek, etc. Estos llamados al sistema son funciones que solicitan al *kernel* ejercer una acción sobre los archivos, pero en el caso de un periférico, estas funciones son atendidas por el *driver*. De esta forma el creador del *driver* puede decidir la acción en el periférico más adecuada según la operación solicitada desde el espacio del usuario [11].

Módulos del *kernel*

En Linux los *drivers* son creados como módulos. Estos módulos son pequeñas porciones de código que pueden ser cargados y descargados en el *kernel* en el momento que sea necesario. Ellos extienden la funcionalidad del *kernel*, sin la necesidad de reiniciar el sistema y recompilar el *kernel*.

En Linux existen tres tipos de dispositivos que pueden ser incorporados en el *kernel* como un módulo, estos son: dispositivos de caracteres, dispositivos de bloques y dispositivos de red.

Los *drivers* que se realizaron para los periféricos con características fueron enfocados hacia los dispositivos de caracteres, por tanto son el objetivo de estudio en esta sección.

Dispositivo de Caracteres

Un dispositivo de caracteres es aquel al que se puede acceder a través de un flujo de bytes, este tipo de controlador normalmente incluye llamadas al sistema *open*, *close*, *read*, *write* y la forma de interactuar con él es por medio de nodos en el sistema de archivos. Un ejemplo de este tipo de dispositivo son el puerto serie (/dev/ttyS0) y la consola (/dev/console).

El propósito original de los dispositivos de caracteres es permitir a los procesos comunicarse con los controladores de dispositivos en el *kernel* y, a través de ellos, con los dispositivos físicos (modems, terminales, y en nuestro caso los periféricos con características etc.). Dependiendo del tipo de periférico que se quiera controlar, el *driver* que se emplee siempre será muy dependiente sus características.

Toda aplicación que se ejecuta en el espacio del usuario no puede acceder directamente al área del *kernel*; los dispositivos se acceden a través de archivos de dispositivos, localizados en */dev*. A continuación se muestra la salida del comando *ls -l /dev/*.

```
1 brw-rw---- 1 root disk   3,  0 Nov 27 hda0
2 brw-rw---- 1 root disk   3,  1 Nov 27 hda1
3 crw-rw---- 1 root disk   4, 64 Nov 27 ttyS0
4 crw-rw---- 1 root disk   4, 65 Nov 27 ttyS1
```

Figura 2.6: Salida del comando *ls -l /dev/*.

FUENTE: El autor.

Al observar la salida en la consola se tiene información de los dispositivos que están ejecutándose. En la primera columna una letra identifica al dispositivo, por ejemplo los archivos tipo carácter están definidos por una *"c"* mientras que los tipo bloque por una *"b"*. Se puede observar que existen dos números en la quinta y sexta columna que identifican al *driver*, el número de la quinta columna recibe el nombre de *major number* y el de la sexta *minor number*; estos números son utilizados por el sistema operativo para determinar el dispositivo y el *driver* que debe ser utilizado ante una solicitud a nivel de usuario. El *major number* identifica la clase o grupo del dispositivo, mientras que el *minor number* se utiliza para identificar sub-dispositivos. El *kernel* de

Linux permite que los *drivers* compartan el número mayor, como el caso del disco duro hda que posee dos particiones hda0 y hda1 que son manejados por el mismo *driver*, pero se asigna un número menor único a cada uno; lo mismo sucede con el puerto serie tty.

2.6. *Driver* plantilla

En [12][13] se encuentra una plantilla de un *driver* diseñada para dispositivos de caracteres, la cual es funcional para cualquier periférico basado en registros, cuyos registros deben estar conectados al bus del procesador. Su proceso de elaboración fue basado en lo explicado por el libro [1] y se pudo adaptar a los requerimientos de esta tesis, siguiendo una serie de lineamientos encontrados en el Anexo H o en [12]. Cada porción de código encontrado en esta plantilla esta precedido por una serie de comentarios, con el propósito de hacer entendible su funcionamiento.

Funcionalidades realizadas por el *driver* plantilla

Desde el espacio del usuario a través de líneas de comando en la consola de Linux es posible cargar, realizar tareas, descargar y verificar el funcionamiento del *driver*. A continuación se describen las funcionalidades del *driver* y la forma como este hace uso del *kernel*.

En el script 2.1 se encuentran las líneas de comando a ejecutar, para llevar acabo la interacción del *driver* con el *kernel*. Antes de comenzar a ingresar los comandos, es necesario navegar a la carpeta en donde se encuentran los archivos de alto nivel creados por el usuario los cuales son: Makefile, driver.c, montar.sh, desmontar.sh, aplicaciones.c, etc.

```
1 Consola@Linux:~$ make
2 Consola@Linux:~$ ./instalar.sh
3 Consola@Linux:~$ lsmod
4 Consola@Linux:~$ echo "probando" > /dev/driver
5 Consola@Linux:~$ cat /dev/driver
6 Consola@Linux:~$ ./desinstalar.sh
7 Consola@Linux:~$ lsmod
```

Script 2.1: Uso del *driver*.

Compilación del *driver*

En la primera línea del script 2.1 se encuentra el comando **make**. Este comando hace uso del archivo Makefile mostrado en el script 2.2, el cual se encarga de compilar la fuente del *driver* escrita en código c (driver.c) por medio del compilador **cruzado mipsel-openwrt-linux-gcc**. [1] [2] El resultado de este proceso son una serie

[1] Para que SIE entienda los programas hechos por el usuario mediante el PC, necesita de una herramienta llamada compilador cruzado, el cual permite hacer ejecutables a partir de archivos .c .h para otras arquitecturas diferentes en donde se están creando. Este compilador cruzado hace parte del sistema operativo openwrt que corre en SIE, en especial se escogió una rama de este llamada backfire, en esta rama se puede encontrar los elementos necesarios para producir los binarios del software que posteriormente irán a la tarjeta [10].

[2] En [11] se encuentran los pasos para llevar acabo la instalación de las herramientas de compilación para la tarjeta SIE.

de archivos de los cuales es pieza fundamental el fichero driver.ko. En la Figura 2.7 se puede observar la secuencia de las partes implicadas.

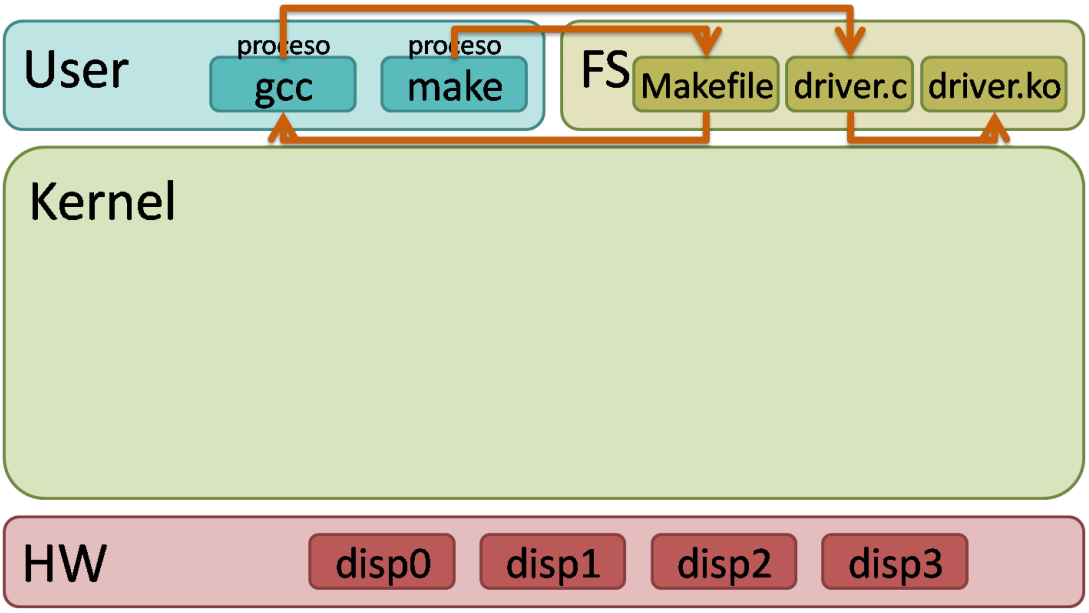


Figura 2.7: Proceso de compilación de la fuente del *driver*.

FUENTE: Tomado de [14].

```
1  EXTRA_CFLAGS += -Wall -I.
2  CC              = mipsel-openwrt-linux-gcc
3  OPENWRT_BASE    = /path/trunk/
4  KERNEL_SRC      = $(OPENWRT_BASE)/build_dir/linux-xburst_qi_lb60/linux-2.6.37.6
5  CROSS_COMPILE   = mipsel-openwrt-linux-
6
7  obj-m +=  driver.o
8
9  all:
10         make -C $(KERNEL_SRC) M=$(PWD) ARCH=mips CROSS_COMPILE=$(CROSS_COMPILE) modules
```

Script 2.2: Makefile.

Carga del *driver*

En la segunda línea del script 2.1 se encuentra el comando **montar.sh**. Este comando hace uso del script 2.3, el cual lleva a cabo las siguientes funciones: Por medio de la función *insmod* se carga el módulo ya creado driver.ko al *kernel* como se muestra en la Figura 2.8, inmediatamente se ejecuta la rutina de inicialización *init* en la cual se desarrollan una serie de eventos.

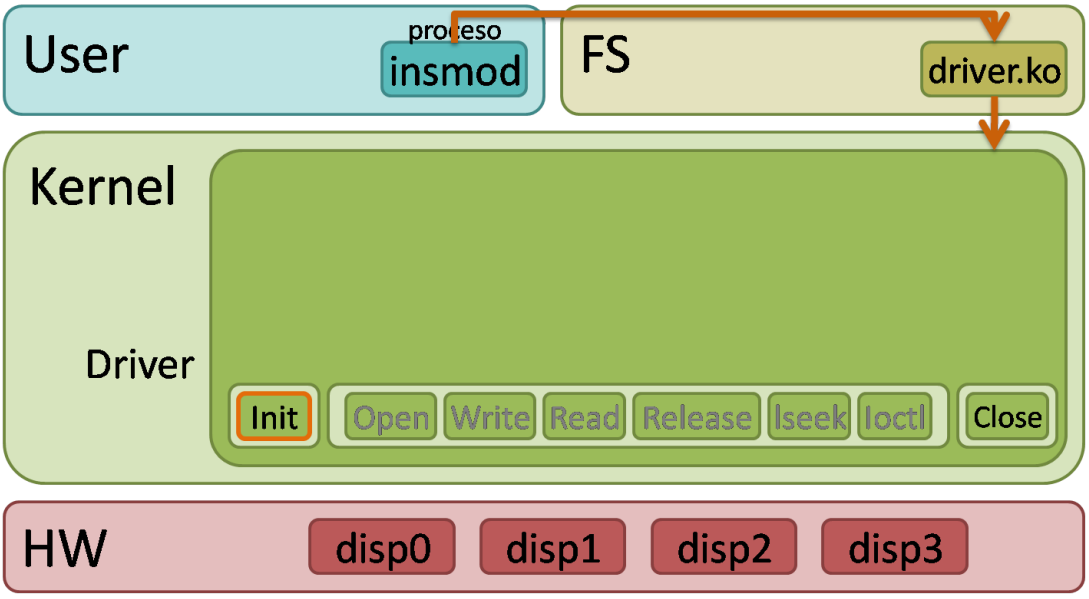


Figura 2.8: Montaje del *driver* en el *kernel* (parte 1).

FUENTE: Tomado de [14].

```
1 #!/bin/sh
2 module="driver"
3 device="periferico"
4 mode="664"
5
6 /sbin/insmod ./$module.ko $* || exit 1
7 major=$(cat /proc/devices | awk '{if($2=="periferico") print $1 ; else print pailas}' \
8 | grep -v pailas)
9 mknod /dev/${device} c $major 0
```

Script 2.3: montar.sh

El primer evento realiza el registro del *driver* a través de nombres dentro del sistema de archivos del *kernel*, estos nombres se denotan como *major number* y *minor number* y se pueden consultar en */proc/devices* como se muestra en la Figura 2.9.

Como el *kernel* de Linux utiliza estructuras para representar a nivel interno dispositivos de caracteres, es necesario crear esta estructura antes de que el *kernel* invoque las operaciones del dispositivo. El segundo evento realiza la asignación e inicialización de esta estructura la cual se denota como *struct cdev*, una vez configurada la estructura *cdev* el siguiente paso es decírselo al *kernel* por medio de un llamado a *cdev_add*. Finalmente, las operaciones de archivo *file_operations* están listas para ser utilizadas tal como se indica en la Figura 2.10. De esta manera tanto la rutina de inicialización *init* y la función *insmod* dan por terminado su participación.

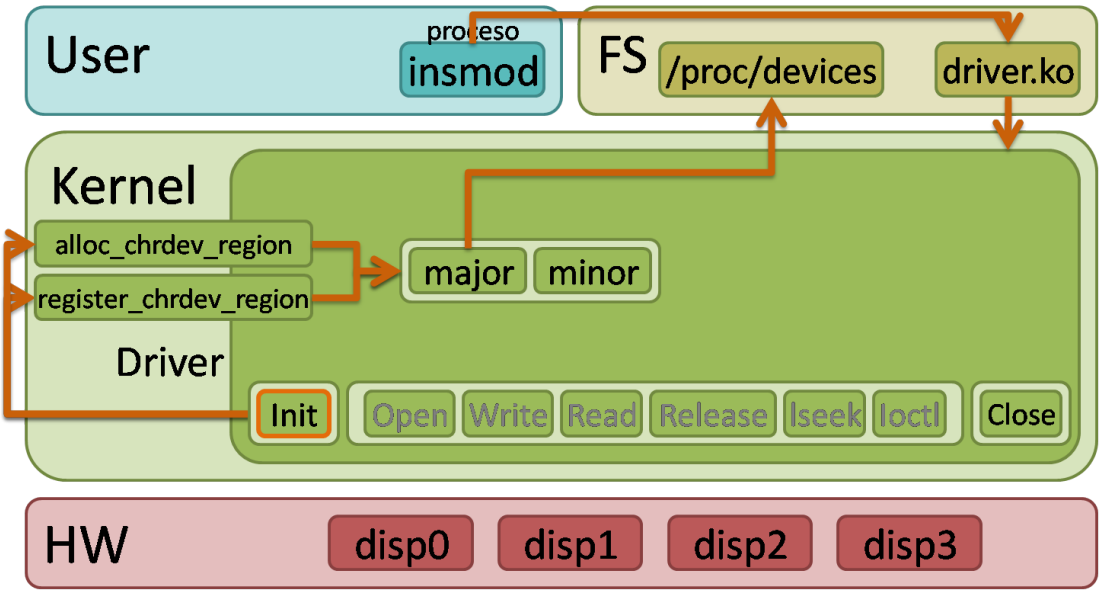


Figura 2.9: Montaje del *driver* en el *kernel* (parte 2).

FUENTE: Tomado de [14].

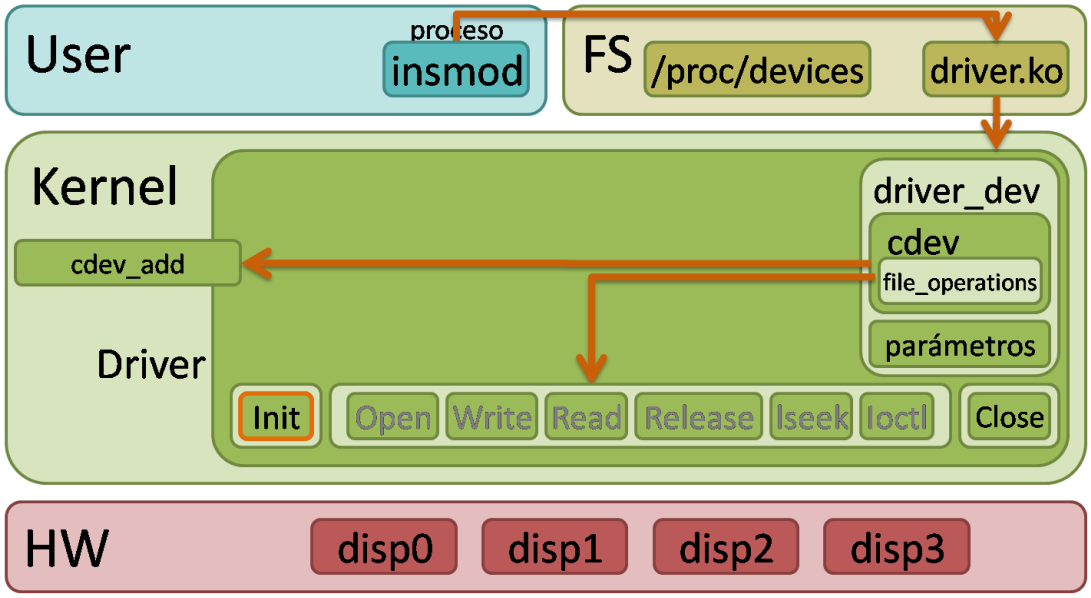


Figura 2.10: Montaje del *driver* en el *kernel* (parte 3).

FUENTE: Tomado de [14].

Luego de la configuración llevada a cabo por la función *insmod*, el *driver* está a la espera de ser usado. Es aquí donde entra en juego la función *mknod*, si bien el *driver* ya se encuentra registrado por medio de su *major number* y *minor munber*, este todavía no es visible como un archivo dentro del *kernel*. Cuando este comando se ejecuta se crea un nodo *inode* [1] que contiene la información de la estructura *cdev*, permitiendo de esta manera poder hacer uso del dispositivo desde el espacio del usuario. En la Figura 2.11 se puede observar la secuencia de las partes implicadas.

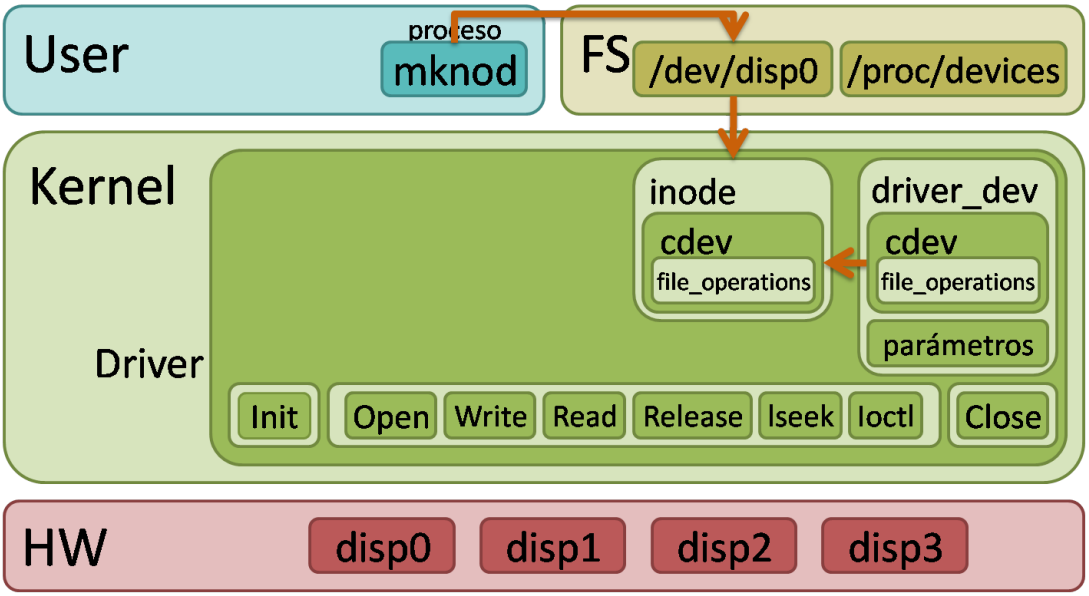


Figura 2.11: Creación del nodo para el uso del *driver*.

FUENTE: Tomado de [14].

En la tercera línea del script 2.1 se encuentra el comando **lsmod**. Este comando se utiliza con el objetivo de verificar si el *driver* quedó correctamente montado dentro del *kernel*.

Operaciones de archivo en el *driver*

Hasta este punto se tiene configurado y montado el módulo dentro del *kernel*, se procederá entonces hacer uso del módulo por medio de las operaciones de archivo. Estas operaciones de archivo se pueden utilizar por medio de la estructura *file_operations* que se encuentra contenida en nuestra estructura *cedv*. Dentro de las operaciones de archivo más comunes se encuentran los llamados al sistema *open*, *write*, *read*, *release* y su funcionamiento se describe a continuación:

En la cuarta línea del script 2.1 se encuentra el comando **echo**. Cuando este comando es ejecutado sucede una serie de eventos que son llevados a cabo por el módulo dentro del *kernel*.

El primer evento realizado es el llamado al sistema *open*, el cual abre el nodo del *driver /dev/disp0*, inmediatamente se crea la estructura *struct_file* la cual representa al *driver* como un archivo abierto, asignándole una zona *private_data* la cual apunta a la estructura *cdev* y así traspasar cualquier función que actúe en el archivo hasta que este se haya cerrado. En la Figura 2.12 se puede observar la secuencia de las partes implicadas.

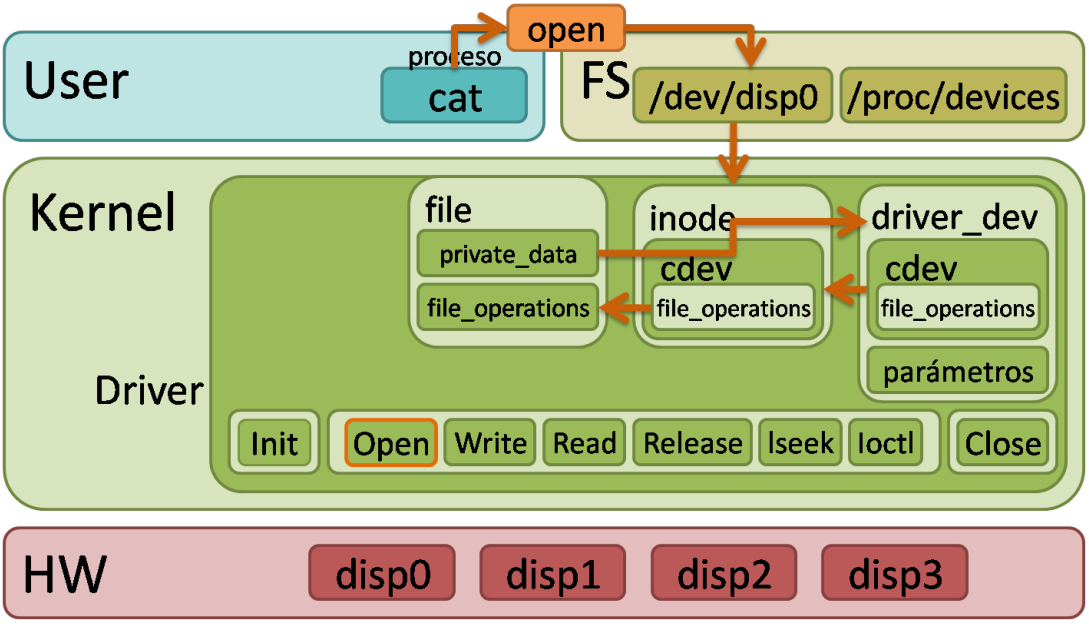


Figura 2.12: Proceso de escritura en el *driver* (parte 1).

FUENTE: Tomado de [14].

El segundo evento realizado es el llamado al sistema *write*, el cual realiza un traspaso de información que va desde el espacio del usuario, pasando por el espacio del *kernel* con destino hacia el dispositivo como se muestra en la Figura 2.13, esto es posible a través de la función *copy_from_user*.

El tercer y último evento realizado es el llamado al sistema *close*, el cual ejecuta la función *release* que se encarga de eliminar la estructura *struct_file* y las asignaciones que se hayan realizado en el llamado al sistema *open*. En la Figura 2.14 se puede observar la secuencia de las partes implicadas.

En la quinta línea del script 2.1 se encuentra el comando **cat**, con el cual se busca leer la información que se encuentra en el dispositivo. Siempre que se realice un llamado al sistema para lectura o escritura los eventos llevados a cabo dentro del *kernel* son similares.

A diferencia de *echo*, *cat* ejecuta el llamado al sistema *read* y accede al dispositivo para realizar el traspaso de información que va desde el dispositivo, pasando por el espacio del *kernel* con destino hacia el espacio del usuario, a través de la función *copy_to_user* como se muestra en la Figura 2.15. El orden de los llamados al sistema utilizados por la función *cat* son: *open*, *read* y *close*.

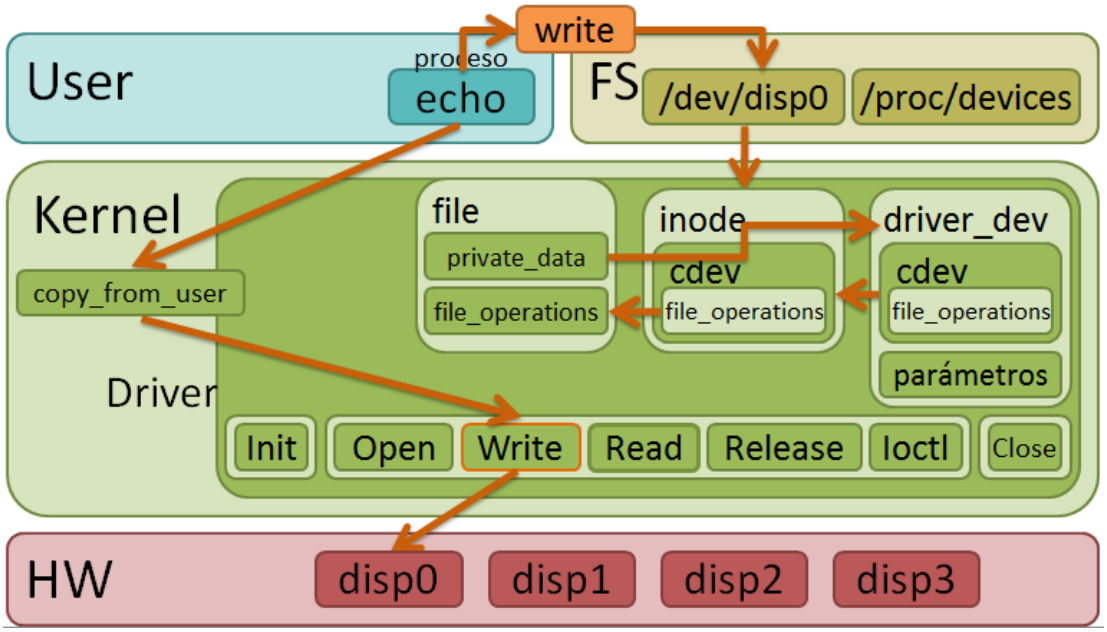


Figura 2.13: Proceso de escritura en el *driver* (parte 2).

FUENTE: Tomado de [14].

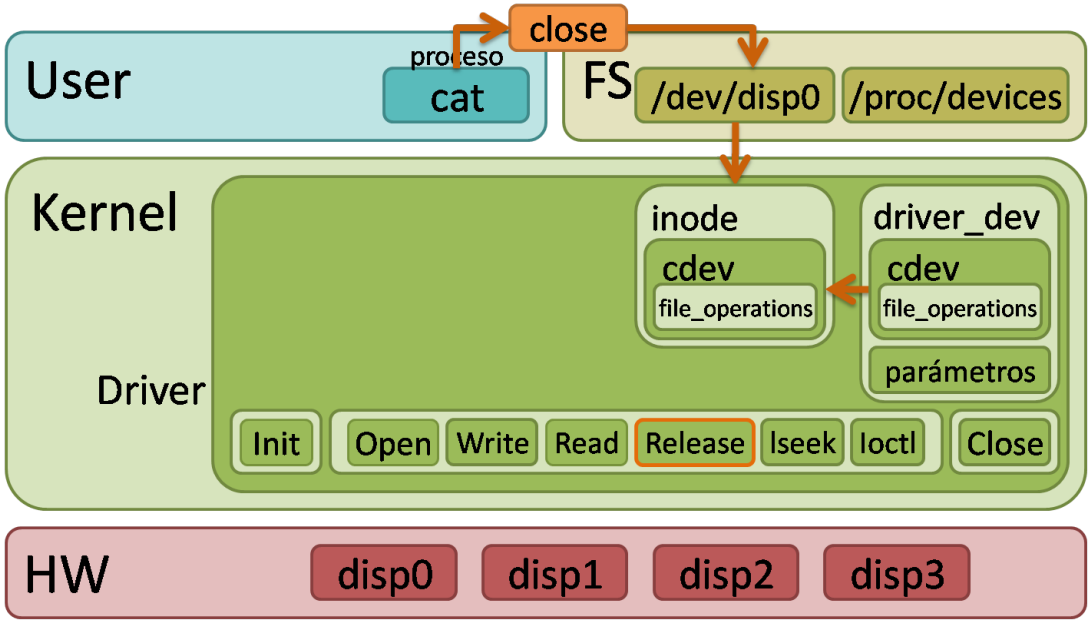


Figura 2.14: Proceso de escritura en el *driver* (parte 3).

FUENTE: Tomado de [14].

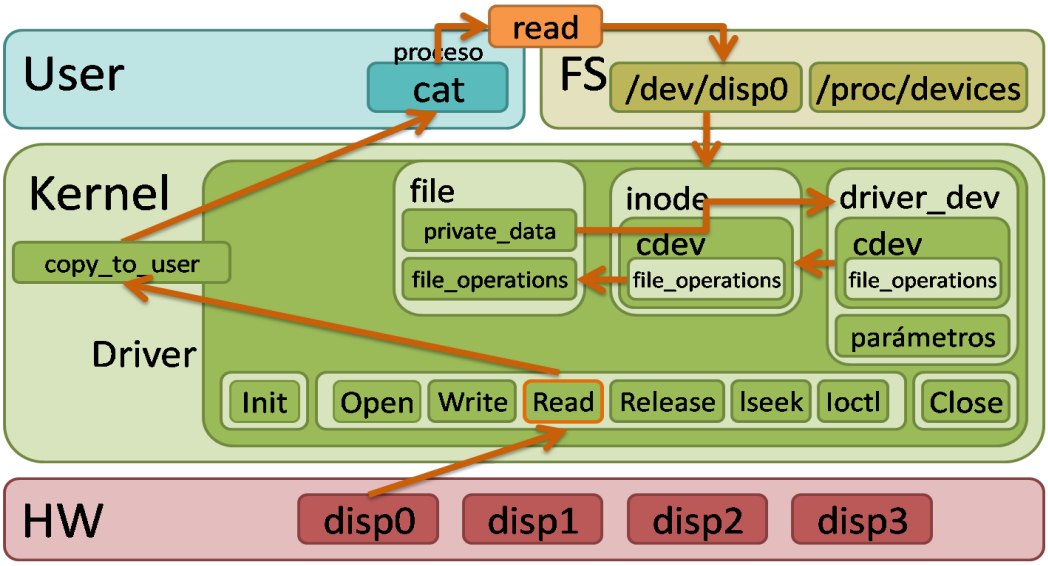


Figura 2.15: Proceso de lectura en el *driver*.

FUENTE: Tomado de [14].

Existen más operaciones de archivos que pueden encontrarse en un *driver* para manejo de un hardware específico, entre las cuales se resaltan los llamados al sistema *LSEEK* y *IOCTL*.

LSEEK es identificado como un puntero de posición, que se mueve con cada lectura o escritura llevada a cabo dentro del fichero abierto. Por otra parte *IOCTL* permite a una aplicación controlar o comunicarse con un *driver* de dispositivo, como una forma alternativa de los llamados al sistema *READ/WITE.*

Descarga del *driver*

En la sexta línea del script 2.1 se encuentra el comando **desmontar.sh**. Este comando hace uso del script 2.4, el cual lleva a cabo la función *rmmod* que se encarga de la liberación del *driver* del *kernel*. Cuando *rmmod* es invocado se realiza el último llamado al sistema de *close*. Antes de que se termine el proceso la estructura *struct_cdev* es liberada y posteriormente se desengancha el *driver* del *kernel*. Una huella del dispositivo es dejada en el sistema de archivos, esta es la imagen del nodo que presentó al *driver* que estaba en uso, la cual se borrará luego de reiniciar el sistema. En las Figuras 2.16, 2.17 y 2.18 se puede observar la secuencia de las partes implicadas.

```
1 #!/bin/sh
2 module="driver"
3 device="periferico"
4
5 /sbin/rmmod ./$module.ko $* || exit 1
6
7 rm -f /dev/${device}
```

Script 2.4: desmontar.sh

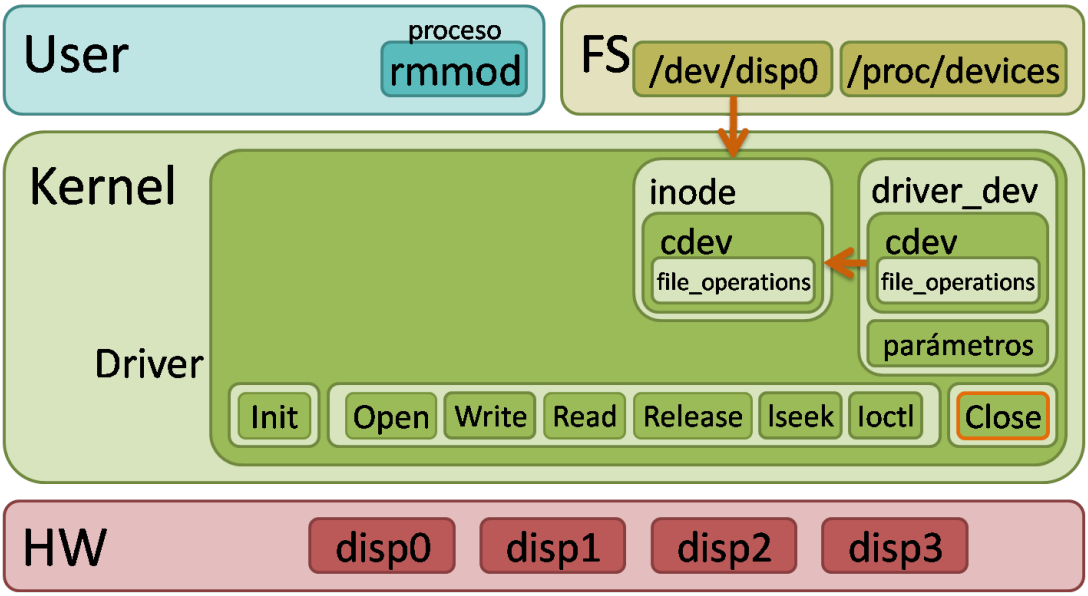


Figura 2.16: Liberación del *driver* del *kernel* (parte 1).

FUENTE: Tomado de [14].

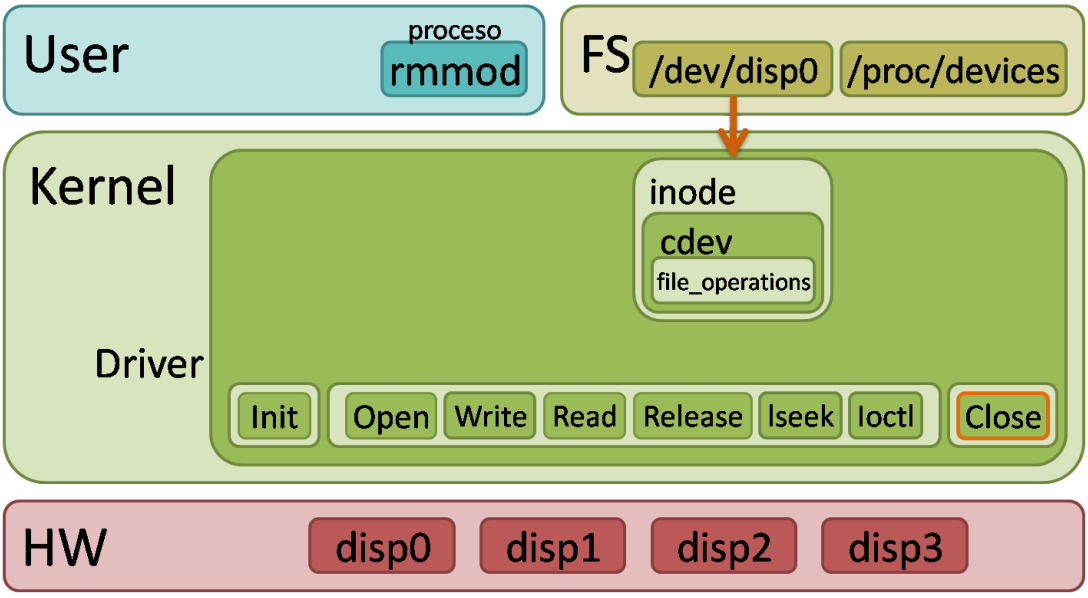


Figura 2.17: Liberación del *driver* del *kernel* (parte 2).

FUENTE: Tomado de [14].

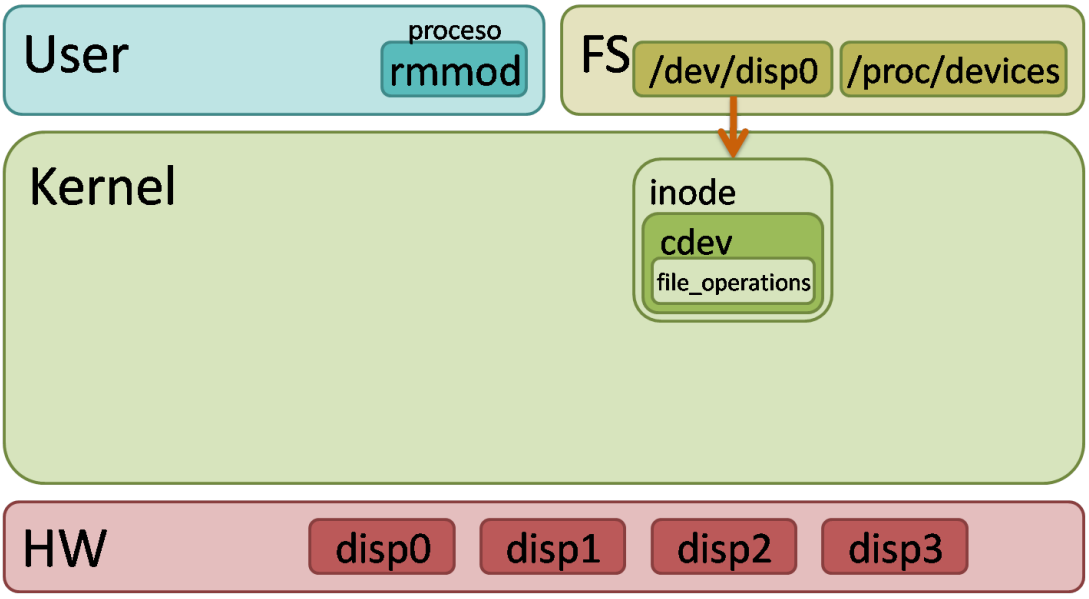


Figura 2.18: Liberación del *driver* del *kernel* (parte 3).

FUENTE: Tomado de [14].

Finalmente, se puede verificar si el *driver* fue removido con éxito del *kernel*, para ello se utiliza el comando *lsmod* como se observa en la séptima línea del script 2.1.

2.7. Adaptación del driver plantilla para el uso de los periféricos

En el anexo H se lista el código fuente del *driver* plantilla adaptado para usar desde el sistema operativo, los periféricos: banco de registros, memoria ram y memoria fifo. En esta nueva versión del *driver* se realizan cambios en la estructura de construcción, cambios en la sección donde se definen las librerías y se agregaron nuevas funciones en las secciones *read* y *write*. Las secciones del *driver* plantilla que no se modificaron fueron *llseek* y *ioctl*, este último no se incluyó en la nueva versión ya que no se necesitaron funciones adicionales en el manejo de los periféricos.

En este mismo anexo se encuentra el archivo Makefile que compila el modulo, los scripts que permiten añadir/quitar el modulo del *kernel* y se explica el proceso de adaptación del modulo a un periférico en particular.

2.8. Comunicación en el sentido Periférico-Procesador

La comunicación en este sentido, es la forma correcta con la cual un periférico debe pedir la atención del procesador. Cuando un periférico a terminado de realizar un proceso y necesita nuevas instrucciones para seguir operando o un nuevo evento originado por un usuario para el uso del periférico necesita ser atendido;

se debe iniciar un proceso para que el procesador se comunique con él. En la Figura 2.4 se observa que cada uno de los periféricos utiliza una señal llamada *interrupcion*, por donde se le informa al procesador que un determinado periférico necesita su atención.

En Linux, las interrupciones se manejan de dos formas: La primera propone instalar el controlador de interrupciones en el momento de la inicialización del *driver* y en la segunda el controlador de interrupciones es instalado cuando el *driver* se abre por primera vez. En esta sección se trabajará con interrupciones administradas en la inicialización del *driver*.

Trabajando con interrupciones

Para manejar interrupciones es necesario adicionar unas líneas a la plantilla del *driver*. A continuación se explicarán los cambios realizados [15].

1. **En la función de inicialización** *PERIFERICO_init* se agregaron este par de comandos con el objetivo de:

   - Definir un pin del procesador como entrada (*PERIFERICO_IRQ_PIN*), y posteriormente utilizarlo como señal de interrupción (*irq_llegada*).
   - Realizar el llamado a la función *request_irq*, la cual asigna recursos a la interrupción, habilita el manejador (*irq_handler*) y la línea de interrupción (*PERIFERICO_IRQ_PIN*).

```
1  irq_llegada = gpio_to_irq(PERIFERICO_IRQ_PIN);
2  equest_irq(irq_llegada, irq_handler, IRQF_DISABLED | IRQF_TRIGGER_RISING, "PERIFERICO-IRQ", NULL);
```

2. **En la función de limpieza** *PERIFERICO_exit* se liberan los recursos asignados a la interrupción por medio del comando:

```
1          free_irq(PERIFERICO_IRQ_PIN, NULL);
```

3. **En La función de interrupción** *irqreturn_t irq_handler*: Esta función se ejecuta cada vez que sucede un evento de interrupción. Si este es el caso y la variable *irq_habilitador* está habilitada, entonces se incrementa el contador *irq_contador* encargado de almacenar la cantidad de interrupciones realizadas por el periférico.

```
1  static irqreturn_t irq_handler(int irq_llegada, void *dev_id)
2  {
3    if(irq_habilitador)
4    {
5     printk(KERN_ALERT "PERIFERICO_irq: La interrupcion ha sido habilitada. \n\r");
6     irq_contador++;
7     return IRQ_HANDLED;
8    }
```

4. **En la función de lectura** *PERIFERICO_read* se traspasa al usuario la cantidad de interrupciones *leido* que se encuentran almacenadas en el contador de interrupciones *irq_contador* por medio del comando *copy_to_user*. Este paso se realiza para verificar si están sucediendo las interrupciones.

```
1  leido = irq_contador;
2  *el_PERIFERICO->intercambio_de_datos = leido;
3  copy_to_user(puntero_usuario, el_PERIFERICO->intercambio_de_datos, retval))
```

5. **En la función de escritura** *PERIFERICO_write* se toman las decisiones de activar o desactivar el nivel de seguridad manejado por el usuario. Este nivel de seguridad permite que en la función de interrupción se incremente el contador de interrupciones *irq_contadoror* sólo cuando la señal *irq_habilitador* se encuentre en 1, si esta es 0, entonces se emite un mensaje al usuario en donde se le indica que la señal *irq_habilitador* no se encuentra activa.

```
1  copy_from_user(el_PERIFERICO->intercambio_de_datos, puntero_usuario, retval))
2  escrito = *el_PERIFERICO->intercambio_de_datos;
3  if(escrito==1)
4  {
5   irq_habilitador = 1;
6  } else {
7   if(escrito==0)
8   {
9    irq_habilitador = 0;
10  }
11 }
```

   El usuario envía la señal de habilitación luego de realizar la llamada al sistema *write*, este valor es recibido en el espacio del *kernel* por medio del comando *copy_from_user* y este almacena la información en la variable *escrito*, luego de un proceso de decisión se asigna un valor a la señal *irq_habilitador*.

Plantilla del *driver* de interrupciones

En el Anexo I se lista el código fuente de la plantilla del *driver* que administra interrupciones y la aplicación desarrollada en el espacio del usuario. Es necesario realizar el proceso de adaptación del driver a un periférico específico y utilizar las fuentes Makefile, instalar.sh, desinstalar.sh, encontradas en el Anexo H, para así llevar a buen termino el uso del *driver*.

CAPÍTULO 3

Pruebas finales

3.1. Funcionamiento del *driver* basado en registros

La estrategia empleada para verificar el funcionamiento del *driver* basado en registros consistió en utilizar la función **echo** que trabaja en el espacio del usuario. Se realizaron modificaciones en el *driver* como se muestra en el cuadro de código 3.1 para poder acceder a la posición de memoria física del *led* de propósito general conectado al procesador de la tarjeta SIE, esta dirección de memoria se estableció luego de revisar la hoja de datos del procesador JZ4725 [9] en donde se encontraron los registros PCDAT, PCDATS y PCDATC que se deben manipular para llevar a cabo la comunicación con el *led*. Desde el espacio del usuario a través de la consola de Linux es necesario ejecutar los pasos del script 3.1 y como resultado se debe observar que el led conectado al procesador prende y apaga cuando se escribe el comando **echo '1' >/dev/Periferico_PERIFERICO** o **echo '0' >/dev/Periferico_PERIFERICO** respectivamente.

```
1  ...
2  /*************************************/
3  /*INFORMACION ASOCIADA AL PERIFERICO*/
4  /*************************************/
5  ...
6  #define PERIFERICO_NUMERO_REGISTROS_DATA 3
7  #define PERIFERICO_BASEADDRESS_DATA 0x10010210
8  #define PERIFERICO_MEMSPACE_SIZE_DATA PERIFERICO_NUMERO_REGISTROS_DATA*4
9  #define PERIFERICO_OFFSET_WRITE_REGISTER 4
10 #define PERIFERICO_OFFSET_CLEAR_REGISTER 8
11 ...
12 /*************************/
13 /*LLAMADA AL SISTEMA write*/
14 /*************************/
15 static ssize_t PERIFERICO_write(struct file *puntero_file, const char __user *puntero_usuario, size_t tamano, \
16 loff_t *puntero_offset)
17 {
18   ...
19   /*Realizando el proceso de escritura desde el espacio del usuario al espacio del nucleo*/
20   printk(KERN_ALERT "PERIFERICO_write: Realizando el traspaso de datos desde el usuario al nucleo. \n\r");
21   if(copy_from_user(el_PERIFERICO->intercambio_de_datos, puntero_usuario, retval))
22   {
23     printk(KERN_ALERT "PERIFERICO_write: No se ha recibido informacion del usuario en el nucleo. \n\r");
```

```
24    } else {
25      printk(KERN_ALERT "PERIFERICO_write: Se recibio informacion del usuario en el nucleo. Esta es: 0x%X. \n\r", \
26      el_PERIFERICO->intercambio_de_datos[0]);
27
28      if(el_PERIFERICO->intercambio_de_datos[0] == '1' || el_PERIFERICO->intercambio_de_datos[0] == '0')
29      {
30        switch(el_PERIFERICO->intercambio_de_datos[0])
31        {
32          case '1':
33            printk(KERN_ALERT "PERIFERICO_write: Dispositivo periferico encendido. \n\r");
34            iowrite32(0x00020000, PERIFERICO_iomem_pointer + PERIFERICO_OFFSET_WRITE_REGISTER );
35            break;
36          case '0':
37            printk(KERN_ALERT "PERIFERICO_write: Dispositivo periferico apagado. \n\r");
38            iowrite32(0x00020000, PERIFERICO_iomem_pointer + PERIFERICO_OFFSET_CLEAR_REGISTER );
39            break;
40          default:
41            return -EINVAL;
42        }
43      } else {
44        printk(KERN_ALERT "PERIFERICO_write: Ha ingresado una orden incorrecta. \n\r");
45        printk(KERN_ALERT "PERIFERICO_write: Las ordenes son: '1' para prender, '0' para apagar. \n\r");
46      }
47      ...
48    }
49    ...
50 }
51 ...
```

Cuadro de código 3.1: Modificaciones en la plantilla del *driver* basado en registros.

```
1 root@BenNanoNote:~# ./instalar.sh
2 root@BenNanoNote:~# echo '1' >/dev/Periferico_PERIFERICO
3 root@BenNanoNote:~# echo '0' >/dev/Periferico_PERIFERICO
4 root@BenNanoNote:~# ./deinstalar.sh
```

Script 3.1: Pasos para la prueba desde consola.

3.2. Funcionamiento de la aplicación del usuario

Desde el espacio del usuario se creó un archivo llamado **aplicacion.c** como se muestra en el cuadro de código 3.2 en donde se realizan las llamadas al sistema *open, lseek, read, write* y *close*, estas llamadas al sistema se hacen al directorio */dev/Periferico_PERIFERICO* que está representando al *driver* que gestiona los registros conectados al *led* del procesador.

En el *driver* basado en registros fue necesario realizar las modificaciones mostradas en el cuadro de código 3.3. En esta oportunidad la aplicación del usuario da las opciones escribir en el registro PCDATS para establecer el led al estado de encendido, escribe en el registro PCDATC para establecer el *led* al estado de apagado y lee el estado del *led* por medio del registro PCDAT.

Para verificar el funcionamiento es necesario ejecutar los pasos del script 3.2 en consola y, luego de escoger alguna de las tres opciones de la aplicación del usuario, se observará el cambio respectivo en *led* del procesador.

```
1  #include <stdio.h>
2  #include <fcntl.h>
3
4  int main()
5  {
6    int file_descriptor;1
7    int opcion;
8    unsigned int prender = 0x00020000;
9    unsigned int apagar = 0x00020000;
10   unsigned int leer[0];
11
12   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
13
14   printf("APLICACION: SELECCIONAR LA OPERACION A RELIZAR. \n");
15   printf("APLICACION: 0. Prender. \n");
16   printf("APLICACION: 1. Apagar. \n");
17   printf("APLICACION: 2. Leer. \n");
18   scanf("%d", &opcion);
19
20   if(opcion == 0)
21   {
22     lseek(file_descriptor, 4, SEEK_SET);
23     write(file_descriptor, &prender, 4);
24   }
25
26   if(opcion == 1)
27   {
28     lseek(file_descriptor, 8, SEEK_SET);
29     write(file_descriptor, &apagar, 4);
30   }
31
32   if(opcion == 2)
33   {
34     lseek(file_descriptor, 0, SEEK_SET);
35     read(file_descriptor, leer, 4 );
36     printf("APLICACION: El dato leido es: 0x%X \n\r", leer[0]);
37   }
38
39   if(opcion > 2)
40   {
41     printf("APLICACION: Esta no es una opccion valida. \n");
42     printf("APLICACION: Vuelva a ejecutar la aplicacion. \n\r");
43   }
44
45   close(file_descriptor);
46
47   return 0;
48 }
```

Cuadro de código 3.2: Primera aplicación del usuario

```
1  ...
2  /*************************************/
3  /*INFORMACION ASOCIADA AL PERIFERICO*/
4  /*************************************/
5  ...
6  #define PERIFERICO_NUMERO_REGISTROS_DATA 3
7  #define PERIFERICO_BASEADDRESS_DATA 0x10010210
8  #define PERIFERICO_MEMSPACE_SIZE_DATA PERIFERICO_NUMERO_REGISTROS_DATA*4
9  ...
10 /*************************/
11 /*LLAMADA AL SISTEMA read*/
12 /*************************/
13 static ssize_t PERIFERICO_read(struct file *puntero_file, char __user *puntero_usuario, size_t tamano, \
14 loff_t *puntero_offset)
15 {
16   ...
17   /*Realizando el proceso de lectura en el sentido periferico-driver-usuario*/
18   printk(KERN_ALERT "PERIFERICO_read: Realizando el traspaso de datos desde el periferico al nucleo. \n\r");
19   if(retval > 0)
20   {
21     /*Leyendo la informacion del periferico desde el espacio del nucleo*/
22     leido = ioread32(PERIFERICO_iomem_pointer_data + (unsigned int)*puntero_offset);
23     *el_PERIFERICO->intercambio_de_datos = leido;
24     printk(KERN_ALERT "PERIFERICO_read: Se recibio informacion del periferico en el nucleo. Esta es: 0x%X. \n\r", \
25     el_PERIFERICO->intercambio_de_datos[0]);
26     ...
27   }
28   ...
29 }
30 ...
31 /***************************/
32 /*LLAMADA AL SISTEMA write*/
33 /***************************/
34 static ssize_t PERIFERICO_write(struct file *puntero_file, const char __user *puntero_usuario, size_t tamano, \
35 loff_t *puntero_offset)
36 {
37   ...
38   /*Realizando el proceso de escritura desde el espacio del usuario al espacio del nucleo*/
39   printk(KERN_ALERT "PERIFERICO_write: Realizando el traspaso de datos desde el usuario al nucleo. \n\r");
40   if(copy_from_user(el_PERIFERICO->intercambio_de_datos, puntero_usuario, retval))
41   {
42     printk(KERN_ALERT "PERIFERICO_write: No se ha recibido informacion del usuario en el nucleo. \n\r");
43   } else {
44     printk(KERN_ALERT "PERIFERICO_write: Se recibio informacion del usuario en el nucleo. Esta es: 0x%X. \n\r", \
45     el_PERIFERICO->intercambio_de_datos[0]);
46
47     /*Realizando el proceso de escritura desde el espacio del nucleo al periferico*/
48     printk(KERN_ALERT "PERIFERICO_write: Realizando el traspaso de datos desde el espacio del nucleo al periferico. \n\r");
49     escrito = *el_PERIFERICO->intercambio_de_datos;
50     iowrite32(escrito, PERIFERICO_iomem_pointer_data + (unsigned int)*puntero_offset);
51     ...
52   }
53   ...
54 }
```

Cuadro de código 3.3: Modificaciones en la plantilla del *driver* basado en registros.

```
root@BenNanoNote:~# ./instalar.sh
root@BenNanoNote:~# ./aplicacion.c
root@BenNanoNote:~# ./deinstalar.sh
```

Script 3.2: Pasos para la prueba desde consola.

3.3. Funcionamiento de los periféricos

Para verificar el funcionamiento de los periféricos se empleó un circuito que permitió observar en ocho *leds* lo que estaba sucediendo en cada uno de ellos. Estos periféricos tienen una característica en particular y es que internamente todos están conformados por registros, esta característica fue aprovechada al conectar cada registro a un *led* de propósito general externo a el FPGA, obteniendo como resultado en los leds la información enviada desde la aplicación del usuario.

Los cambios realizados sobre el hardware se pueden observar en el cuadro de código 3.4, las aplicaciones de usuario empleadas en estas pruebas son las encontradas en el Anexo G y el *driver* usado es la versión final (plantilla del *driver* basado en registros) suministrado en el Anexo H.

Las modificaciones mostradas a continuación se efectuaron a la interfaz y al periférico banco de registros.

```
--************************************--
--MODIFICACIONES EN EL ARCHIVO PRINCIPAL--
--************************************--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity interfaz_periferico_registros is

  port(
    ...
    interruptores : in std_logic_vector(3 downto 0);
    leds_expansion_fpga : out std_logic_vector(7 downto 0)
  );

end interfaz_periferico_registros;

architecture Behavioral of interfaz_periferico_registros is

  ...
  component banco_registros
  port(
    rst : in std_logic;
    clk : in std_logic;
    escribir : in std_logic;
    leer : in std_logic;
    direcciones_escritura : in std_logic_vector(2 downto 0);
    direcciones_lectura : in std_logic_vector(2 downto 0);
    entrada : in std_logic_vector(7 downto 0);
```

```
31      salida: out std_logic_vector(7 downto 0);
32
33      interruptores_prueba : in std_logic_vector(3 downto 0);
34      salida_prueba : out std_logic_vector(7 downto 0)
35    );
36    end component;
37    ...
38
39  begin
40
41    ...
42    instancia_banco_registros: banco_registros
43    port map(
44      rst => senal_6,
45      clk => clk,
46      escribir => senal_7,
47      leer => senal_8,
48      direcciones_escritura => senal_4,
49      direcciones_lectura => senal_5,
50      entrada => senal_1,
51      salida => senal_9,
52
53      interruptores_prueba => interruptores,
54      salida_prueba => leds_expansion_fpga
55    );
56    ...
57
58  end Behavioral;
```

```
1  ----------------------------------
2  --MODIFICACIONES EN EL PEIFERICO--
3  ----------------------------------
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE.STD_LOGIC_ARITH.ALL;
7  use IEEE.STD_LOGIC_UNSIGNED.ALL;
8
9  entity banco_registros is
10
11   ...
12   port(
13     ...
14     interruptores_prueba : in std_logic_vector(3 downto 0);
15     salida_prueba : out std_logic_vector(7 downto 0)
16   );
17
18 end banco_registros;
19
20 architecture Behavioral of banco_registros is
21
22   ...
23   signal habilitador_1 : std_logic;
24   signal habilitador_2 : std_logic_vector(2 downto 0);
25
26 begin
27
28   ...
```

```
29    habilitador_1 <= interruptores_prueba(3);
30    habilitador_2 <= interruptores_prueba(2 downto 0);
31
32    salida_prueba <= "00000" & direcciones_escritura when habilitador_1='0' else
33                     q0 when habilitador_2="000" else
34                     q1 when habilitador_2="001" else
35                     q2 when habilitador_2="010" else
36                     q3 when habilitador_2="011" else
37                     q4 when habilitador_2="100" else
38                     q5 when habilitador_2="101" else
39                     q6 when habilitador_2="110" else
40                     q7 ;
41
42 end Behavioral;
```

```
1 --------------------------------------
2 --MODIFICACIONES EN EL ARCHIVO DE PINES--
3 --------------------------------------
4 ...
5 net "interruptores<0>" loc="p32";
6 net "interruptores<1>" loc="p26";
7 net "interruptores<2>" loc="p18";
8 net "interruptores<3>" loc="p13";
9
10 net "leds_expansion_fpga<0>" loc="p70";
11 net "leds_expansion_fpga<1>" loc="p65";
12 net "leds_expansion_fpga<2>" loc="p62";
13 net "leds_expansion_fpga<3>" loc="p60";
14 net "leds_expansion_fpga<4>" loc="p57";
15 net "leds_expansion_fpga<5>" loc="p53";
16 net "leds_expansion_fpga<6>" loc="p63";
17 net "leds_expansion_fpga<7>" loc="p66";
```

Cuadro de código 3.4: Modificaciones a la interfaz y al periférico banco de registros

3.4. Funcionamiento del *driver* que maneja interrupciones

En la sección 2.8 se hace referencia a una señal llamada **interrupcion** por donde se le informa al procesador que un determinado periférico necesita de su atención. A continuación se presenta la forma más sencilla de generar una interrupción, esta alternativa puede ser empleada en cualquier periférico que necesite la atención por parte del procesador. El código fuente del hardware usado se lista cuadro de código 3.5. La aplicación del usuario y el *driver* empleado se pueden consultar en el Anexo I.

El proceso de interrupción comienza cuando el usuario acciona el pulsador PB2 de la tarjeta SIE, el valor obtenido tras la pulsación es un '1' lógico de duración de un ciclo de reloj y es asignado a la señal **interrupcion**. Este valor es recibido en el espacio del *kernel* por el módulo *driver* que maneja interrupciones, si el nivel de seguridad del usuario está desactivado no se efectúa ninguna operación, en el caso contrario, si el nivel de seguridad está activado entonces el contador de interrupciones almacena un nuevo valor y este valor a su

vez puede ser leído desde el espacio del usuario con el objetivo de verificar si el proceso de interrupción es correcto.

```
1 ----------------------------------
2 --CIRCUITO QUE GENERA INTERRUPCIONES--
3 ----------------------------------
4 entity circuito is
5
6   port(
7     rst : in std_logic;
8     clk : in std_logic;
9     pulsador : in std_logic;
10    interrupcion : out std_logic
11  );
12
13 end circuito;
14
15 architecture Behavioral of circuito is
16
17  signal d0, q0 : std_logic;
18  signal d1, q1 : std_logic;
19
20 begin
21
22  ----------------
23  --Antirrebotes--
24  ----------------
25  process(rst, clk)
26  begin
27    if (rst='0') then
28      q0 <= '0';
29      q1 <= '0';
30    elsif (clk'event and clk='1') then
31      q0 <= d0;
32      q1 <= d1;
33    end if;
34  end process;
35
36  d0 <= not(pulsador);
37  d1 <= q0;
38  interrupcion <= q0 and not(q1);
39
40 end Behavioral;
```

```
1 ----------------------------------------
2 --MODIFICACIONES EN EL ARCHIVO DE PINES--
3 ----------------------------------------
4 ...
5 net "pulsador" loc="p13"; # Pulsador PB2
6 net "interrupcion" loc="p71"
```

Cuadro de código 3.5: Circuito para generar interrupciones.

CAPÍTULO 4

Conclusiones y trabajo futuro

4.1. Conclusiones

Mediante el presente proyecto se logró cumplir los alcances, permitiendo de esta manera dar cumplimiento a los objetivos planteados en la etapa inicial del proyecto. A continuación se presentan las conclusiones más importantes obtenidas durante la investigación:

- **El estudio realizado al entorno de trabajo EDK abarcó una pequeña porción de las funcionalidades ofrecidas por la herramienta, sin embargo, un análisis más profundo conlleva a un codiseño con periféricos aún más robusto aplicable a la industria.**

  Durante el desarrollo del proyecto se definieron los limites de estudio en EDK y se establecieron las pautas que permitieron hacer buen uso de la herramienta. De esta forma se inicia el estudio de los periféricos y se toma como punto de partida la documentación ofrecida por Xilinx en su portal *web*. Este modelo de trabajo fue acertado ya que facilitó el aprendizaje y como resultado quedaron una serie de guías encontradas en los anexos correspondientes al capítulo 1.

- **De los periféricos se logró recopilar la información que define sus características, sobre su protocolo de comunicación y su interacción con los bloques externos que complementan el diseño.**

  Esta particularidad está sujeta a: Siempre que se inicia un nuevo proyecto, el software de trabajo dá la opción crear las fuentes en lenguaje de descripción de hardware *hdl*, para que estas puedan ser utilizadas en el proceso de simulación y síntesis. En este punto se tienen a disposición todos los archivos que componen el hardware y se puede pensar en la opción de revisar como está constituido internamente un periférico, sin embargo, este paso no arroja un resultado concreto ya que la mayor parte de los archivos están escritos en lenguaje de descripción de hardware de bajo nivel y es necesario tener cierto grado de conocimiento en el área.

- **Se realizó un estudio de gran parte de las posibilidades que ofrece la herramienta de computo SIE.**

En el transcurso de este proyecto se abarcaron los niveles: usuario, aplicaciones (software de alto nivel), sistema operativo (software de bajo nivel) y hardware. Se proporcionan los mecanismos para modificar o realizar diseños en los diferentes niveles y finalmente estos son integrados como un conjunto para trabajar tareas específicas.

- **La plataforma de desarrollo SIE proporcionó todos los recursos que hicieron posible la implementación de los códigos fuentes empleados en las pruebas.**

  La importancia de la plataforma SIE en este proyecto se refleja en el uso de su procesador y el FPGA, es a través de la interacción de estos dos componentes que se realizaron las pruebas preliminares y finales. Por medio del dispositivo FPGA se implementaron los diferentes periféricos y el procesador fue configurado para enviar información del usuario a los periféricos utilizando los *drivers* como medio de comunicación. Este conjunto ofrece un amplio posibilidades que van desde una aplicación hardware/-software sencilla, hasta un sistema complejo.

- **Se logró controlar cada uno de los diferentes tipos de periféricos identificados durante el estudio mediante dos plantillas de driver. Así mismo se validó su funcionamiento mediante pruebas de laboratorio.**

- **Se elaboró una interfaz de comunicación implementada en el FPGA, la cual permite administrar un conjunto de periféricos y este conjunto a su vez, puede ser gestionado por el usuario haciendo uso de un solo *driver.***

- **Este proyecto ha realizado un aporte que sirve como base para solucionar diferentes actividades propuestas en los programas académicos de la universidad que tengan relación con el área de trabajo.**

  Un aspecto importante de este proyecto está relacionado con el uso del sistema operativo Linux, en el cual se fundamentan las aplicaciones de usuario y los *drivers* desarrollados. De esta forma se abre una puerta que propone como alternativa el uso del software libre antes de recurrir al software ilegal.

4.2. Trabajo futuro

Con el objetivo de apoyar y orientar los trabajos futuros que busquen complementar el presente proyecto, presentó algunos puntos que surgen como resultado de la experiencia en la realización de este trabajo.

- **Mejoras sobre los *drivers*: basado en registros e interrupciones.**

  El proceso de comunicación con los periféricos por medio de los *drivers*, es posible luego de configurar los registros del procesador conectados a el FPGA, por tanto las aplicaciones desde el espacio del usuario están orientadas a la manipulación de estos registros. Una alternativa de mejora realizable sobre

este estilo de sistemas tiene que ver con la percepción que se tiene del periférico desde el espacio del usuario. Esta perspectiva de los periféricos se puede mejorar si se implementa un estilo que permita una comunicación más eficiente.

- **Sobre la versión de Linux que corre en la tarjeta SIE.**

  El sistema operativo que actualmente está instalado en el sistema de desarrollo SIE corresponde a la versión Linux-2.6.37.6 y es interesante estudiar la viabilidad de implementar este proyecto en una versión de Linux más reciente.

Bibliografía

[1] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, Third Edition*. O'Reilly Media, Inc. 2005.

[2] Xilinx Inc. *UG111: Embedded System Tools Reference Manual EDK*, April 2012.

[3] Xilinx Inc. 13x.plbv46 material técnico, docs_pdf, 12_edk_overview.pptx, 2011.

[4] Xilinx Inc. *Xilinx ISE 13.2 Design Suite software*, 2????.

[5] Pagina de internet. `http://trucoselectronicayprogramacion.blogspot.com/2010_11_01_archive.html`.

[6] Pagina de internet. `http://www.fpgadeveloper.com`

[7] Página de internet Linux en caja. `http://linuxencaja.net/wiki/SIE/es`.

[8] Página de internet qi-hardware. `http://en.qi-hardware.com/wiki/SIE`.

[9] Datasheet JZ4725, *multimedia Application processor*, Ingenic Semiconductor Co. Ltd, Revision: 1.0, May 2009.

[10] Ochoa Gustavo, Mendez Neder. DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: ARQUITECTURA DE COMPUTADORES. Tesis de grado, Universidad Industrial de Santander, 2012.

[11] Uyabán Luis. DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: SISTEMAS EMBEBIDOS. Tesis de grado, Universidad Industrial de Santander, 2012.

[12] Salamanca William. Diseño e implementación, sobre un FPGA, de un sistema reconfigurable dinámicamente instalado como un nodo de un cluster. Tesis de maestría, Universidad Industrial de Santander, Marzo 2011.

[13] Abreo Sergio. Diseño e implementación de un procesador específico en un FPGA para la ejecución del algoritmo de migración 2D de Kirchhoff. Tesis de maestría, Universidad Industrial de Santander, Febrero 2011.

[14] Ramirez Ana, Salamanca William. Diseño e implementación de un sistema embebido dinámicamente reconfigurable sobre un sistema de desarrollo ml507 configurado como un cluster de procesadores de propósito general. Tesis de maestría, Universidad Industrial de Santander, Septiembre 2010.

[15] Camargo Carlos. Plataforma de Desarrollo ECBOOT. Universidad Nacional de Colombia, 2010.

ANEXOS

ANEXO A. Guía para la creación de un nuevo proyecto en XPS

A.1 Introducción

Este Anexo muestra el proceso de creación de un simple diseño de hardware sección 1.3, utilizando la herramienta *Xilinx Platform Studio (XPS)*.

A.2 Objetivos

- Crear un nuevo proyecto utilizando el asistente de configuración Base System Builder (BSB).
- Crear un diseño de hardware simple utilizando los periféricos de propósito general.

A.3 Procedimiento

Los pasos que se seguirán para llevar a buen termino el procedimiento se muestran a continuación:

1. **Abrir la plataforma de trabajo.**
   En el siguiente script se encuentra la línea de comando a ejecutar para lanzar la herramienta de trabajo XPS.

```
1  consola@linux:~$ xps
```

2. **En la barra de menu de XPS.**
   Seleccionar: File ⇒ New Project
3. **Seguidamente se despliegan las siguientes ventanas:**
   - **Ventana 1. Xilinx Plataform Studio.**
     En este cuadro de diálogo es necesario escoger la opción:
     ⇒ Base System Builder Wizard (recommended)
     Nota: Luego de la configuración dar clik en Ok.

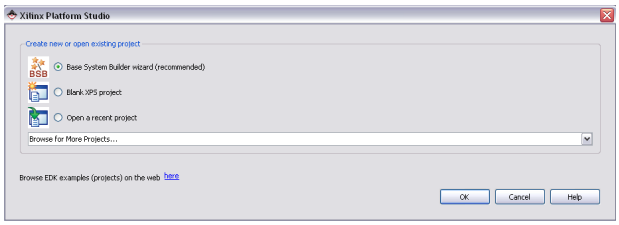


Figura A.1: Creación de un nuevo proyecto usando el BSB.

- **Ventana 2. Create New XPS Project Using BSB Wizard.**
  En este cuadro de diálogo es necesario escoger la opción:
  ⇒ Browse (Ubicarse en la carpeta donde se guardará el proyecto, dar un nombre o dejar por defecto system.xmp.)
  Nota: Después del paso anterior, en esta misma ventana sale la ruta dada del proyecto, es importante que esta dirección no contenga espacios ni tildes, sólo puede tener caracteres alfanuméricos separados por guiones bajos
  Nota: Luego de la configuración dar clik en Ok.

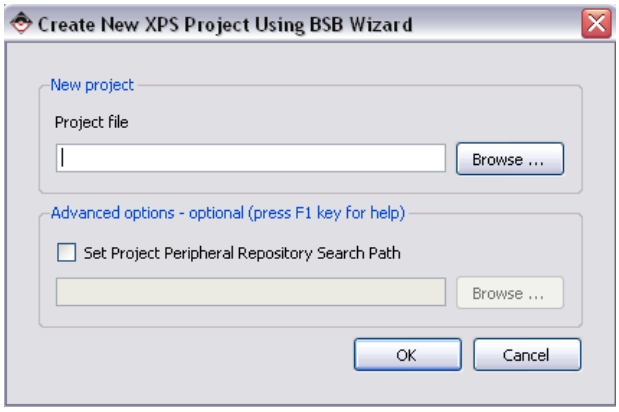


Figura A.2: Asignación del directorio.

- **Ventana 3. Base System Builder.**

  a) **Type Conection:**
  En este cuadro de diálogo es necesario escoger la opción:
  ⇒ PLB System
  Nota: Luego de la configuración dar click en Ok.

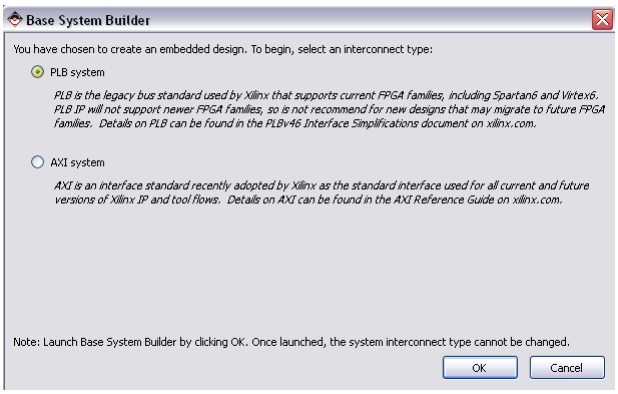


Figura A.3: Cuadro de diálogo para la selección del bus.

b) **Board Selection.**

En este cuadro de diálogo es necesario escoger las opciones:

⇒ I would like to crate a system for the following developent board

⇒ Board Vendor: seleccionar Xilinx.

⇒ Board Name: seleccionar el tipo de tarjeta a utilizar.

⇒ Board Revision: dejar por defecto.

Nota: Deseleccionar Üse Stepping".

Nota: Luego de la configuración dar click en Ok.

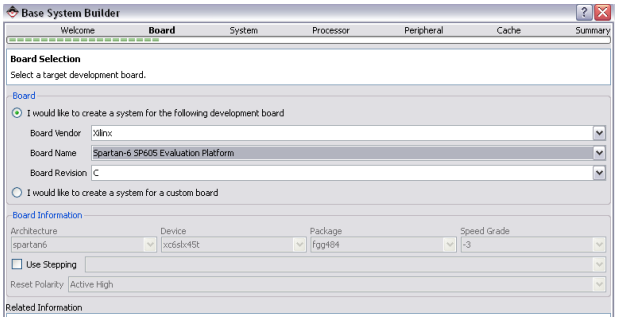


Figura A.4: Cuadro de diálogo para la selección de la tarjeta.

c) **System Configuration.**

En este cuadro de diálogo es necesario escoger la opción:

⇒ Single-Processor System.

Nota: Luego de la configuración dar click en Next.

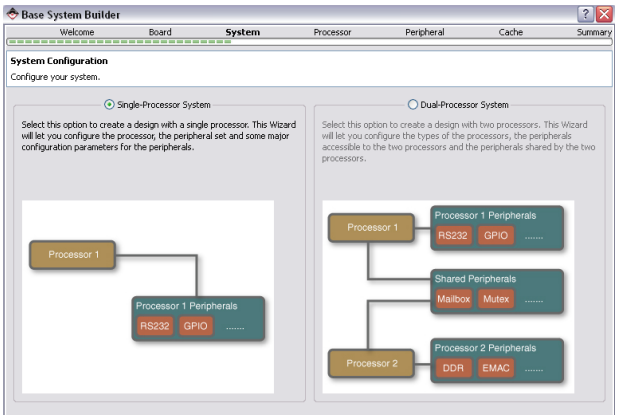


Figura A.5: Cuadro de diálogo para la configuración del sistema.

d) **Processor Configuration.**

En este cuadro de diálogo es necesario escoger la opciones:

⇒ Processor Type: dejar por defecto.

⇒ System Clock Frecuency: 62.50 MHz

⇒ Local Memory : para proyectos pequeños como el de esta guía se puede dejar el valor por defecto, para sistemas más complejos deberá seleccionarse un valor mayor.

Nota: Deseleccionar “Enable Floating Point Unit”.

Nota: Luego de la configuración dar click en Next.

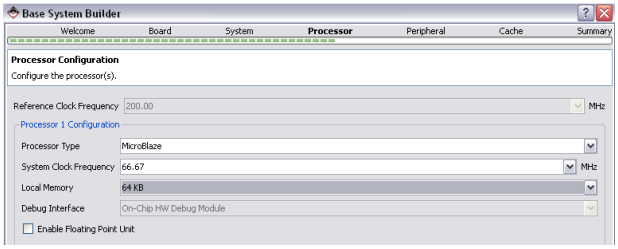


Figura A.6: Cuadro de diálogo para la configuración del procesador.

e) **Peripheral Configuration.**

En este cuadro de diálogo es necesario escoger los periféricos de propósito general a usar.

Nota: Luego de la configuración dar click en Next.

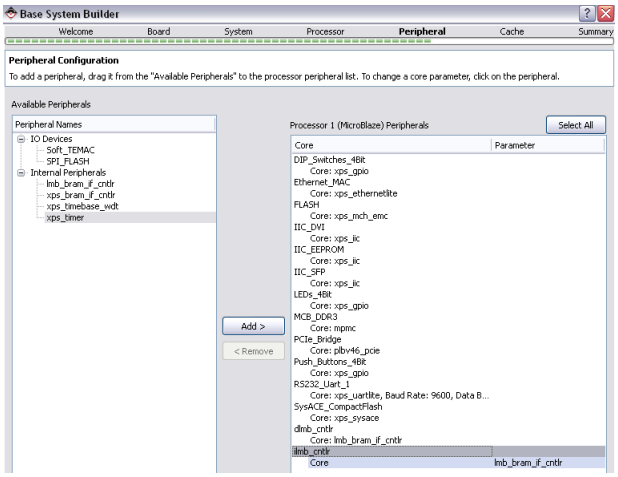


Figura A.7: Cuadro de diálogo para la configuración de periféricos.

f) **Summary.**

En este cuadro de diálogo se resume la configuración del proyecto.

Nota: Luego de verificar el sistema dar click en Finish.

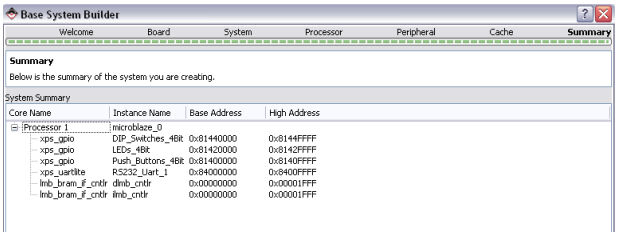


Figura A.8: Resumen del sistema.

De esta forma se realiza la creación de un proyecto usando la herramienta de trabajo XPS. Todas la imágenes utilizadas aquí, fueron tomadas de la versión 13.1 del software de trabajo.

ANEXO B. Guía para la creación de un periférico con característica

B.1 Introducción

Este Anexo muestra el proceso de creación y configuración de un periférico con características sección 1.4, utilizando la opción *Create and Import Peripheral Wizard*.

B.2 Objetivos

- Crear el periférico de hardware.
- Conectar el periférico de hardware al sistema.

B.3 Procedimiento

Los pasos que se seguirán para llevar a buen termino el procedimiento se muestran a continuación:

B.3.1 Creando un periférico de hardware.

1. **En la barra de menú de XPS.**
   Seleccionar: Hardware ⇒ Create or import peripheral.
2. **Seguidamente se despliegan las siguientes ventanas:**
   - **Ventana 1. Welcome to the Create and Import Peripheral Wizard.**
     Nota: En este cuadro de diálogo no se realiza ninguna configuración, dar click en Next.

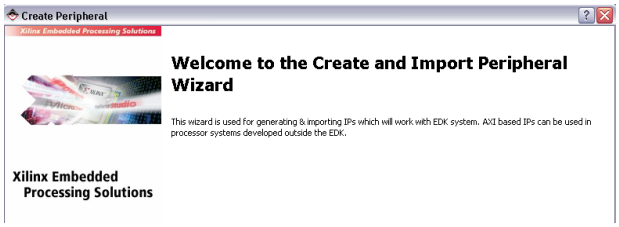


Figura B.1: Asistente para la creación de un periférico de hardware.

- **Ventana 2. Peripheral Flow.**
  En este cuadro de diálogo es necesario escoger la opción:
  ⇒ Create templates for a new peripheral
  Nota: Luego de la configuración dar click en Next.

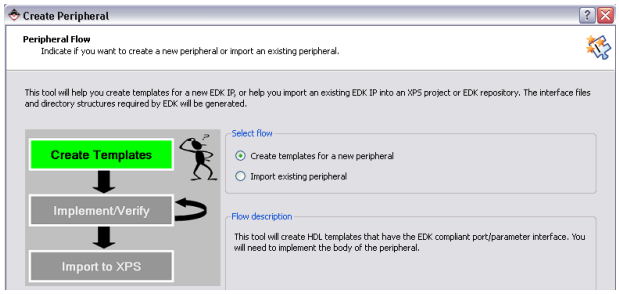


Figura B.2: Cuadro de diálogo *Peripheral Flow*.

- **Ventana 3. Repository or Project.**
  En este cuadro de diálogo es necesario escoger la opción:
  ⇒ To a XPS project
  Nota: Luego de la configuración dar click en Next.

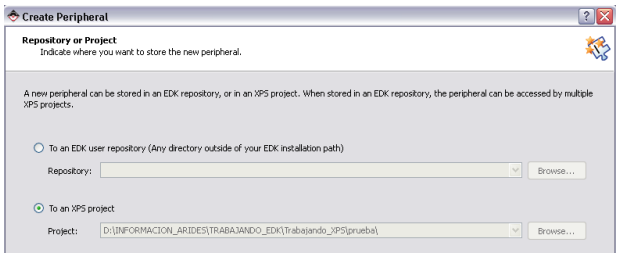


Figura B.3: Cuadro de diálogo *Repository or Project*.

- **Ventana 4. Name and Version.**
  En este cuadro de diálogo es necesario escoger la opción:
  ⇒ Name, Version, Description
  Nota: En el campo “Name” se debe dar el nombre que va a tomar el periférico a crear Se debe tener en cuenta que el este campo sólo permite caracteres alfanuméricos (sin tildes) y guión bajo, el campo “Version” es una etiqueta que se añade al periférico a crear y el campo “Description” contendrá un pequeño resumen de lo que realizará el periférico.
  Nota: Luego de la configuración dar click en Next.

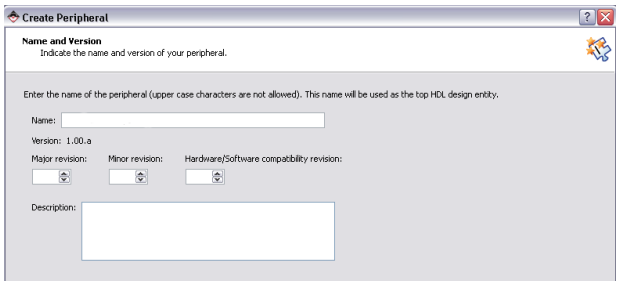


Figura B.4: Cuadro de diálogo *Name and Version*.

- **Ventana 5. Bus Interface.**
  En este cuadro de diálogo es necesario escoger la opción:
  ⇒ Processor Local Bus (PLB v4.6)
  Nota: Luego de la configuración dar click en Next.

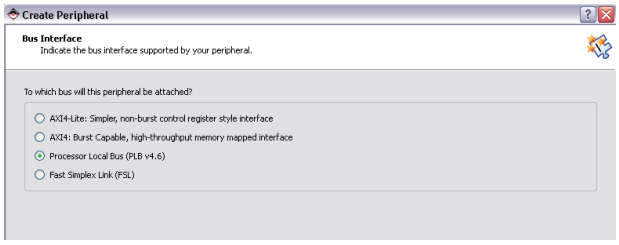


Figura B.5: Cuadro de diálogo *Bus Interface*.

- **Ventana 6. IPIF (IP Interface) Services.**
  En este cuadro de diálogo es necesario escoger entre las opciones:
  ⇒ El tipo de periférico a trabajar
  Nota: Luego de la configuración dar click en Next.

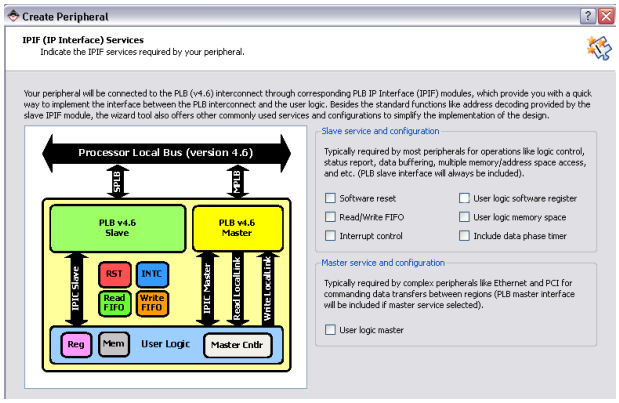


Figura B.6: Cuadro de diálogo *IP Interface*.

- **Ventana 7. Slave interface.**
  Nota: No seleccionar la opción “Burst and cache line-support”.
  Nota: Dar click en Next.

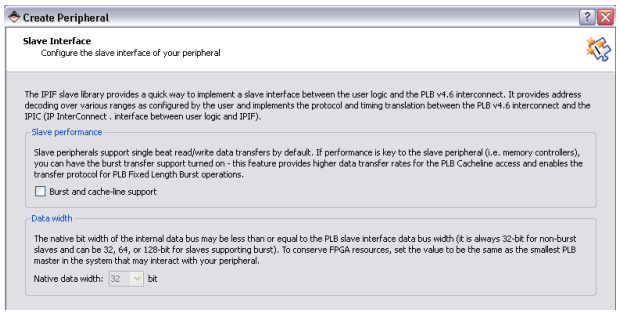


Figura B.7: Cuadro de diálogo *Slave Interface*.

- **Ventana 8. IP Interconnect (IPIC).**
  Nota: En este cuadro de diálogo es necesario dejar las opciones por defecto.
  Nota: Dar click en Next.

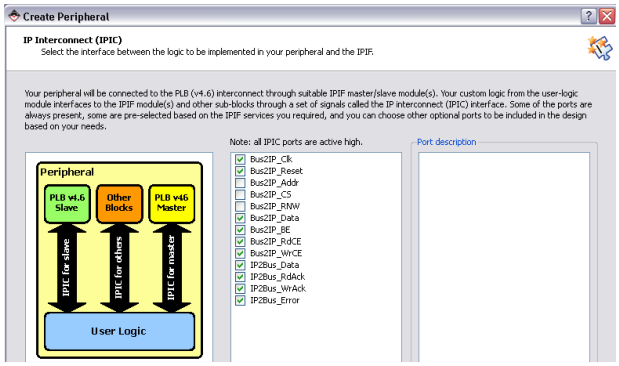


Figura B.8: Cuadro de diálogo *Slave Interface*.

- **Ventana 9. Peripheral simulation support.**
  Nota: No seleccionar la opción “Generate BFM simulation platform for ISim or ModelSim-SE or ModelSim-PE”.
  Nota: Dar click en Next.

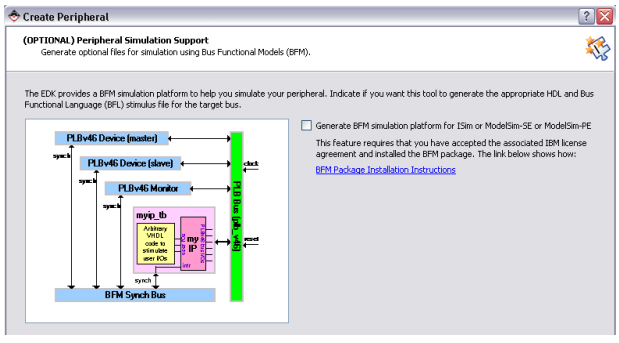


Figura B.9: Cuadro de diálogo Opcional.

- **Ventana 10. Peripheral implementation support.**
  En este cuadro de diálogo es necesario seleccionar las opciones:

⇒ Generate ISE and XST project files to help you implement the peripheral using XST flow
⇒ Generate template driver files to help you implement software interface
Nota: Luego de la configuración dar click en Next.

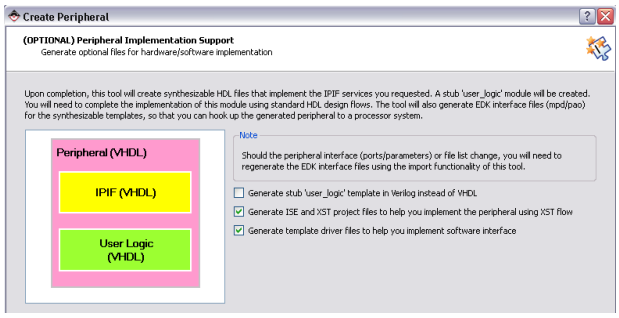


Figura B.10: Cuadro de diálogo en donde se crean las fuentes en VHDL y en C.

- **Ventana 11. Congratulations.**
  En este cuadro de diálogo se resumen todos los cambios realizados en la configuración del proyecto.
  Nota: Luego de verificar el sistema dar click en Finish.

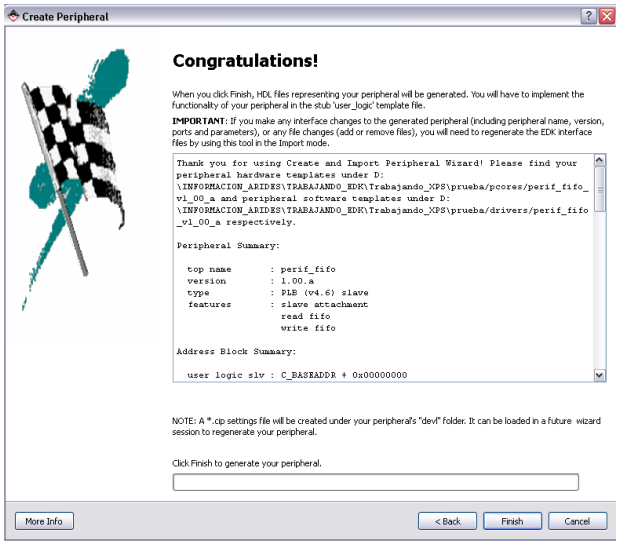


Figura B.11: Cuadro de diálogo *Congrstulstions*.

B.4 Conectando el periférico de hardware al sistema.

1. **IP Catalog.**
   Es Importante realizar el siguiente paso:
   ⇒ USER ⇒ Escoger el periférico creado y arrastrar a ⇒ BusInterfaces .
   Nota: Este mismo paso se puede realizar dando Click derecho agregar al sistema.

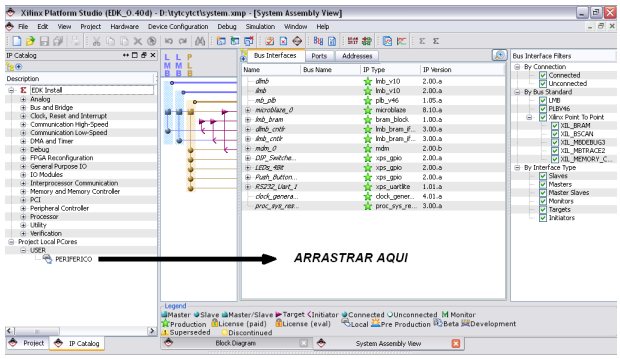


Figura B.12: Conectando las entradas del *IP catalog* en el *Bus Interfaces*.

2. **Bus Interfaces.**
   Es Importante realizar el siguiete paso:
   ⇒ periferico_0,1,n ⇒ SPLB ⇒ mb_plb.

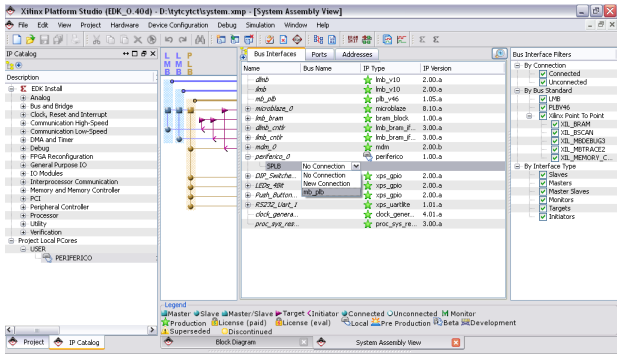


Figura B.13: Conectando las entradas al bus de comunicación.

3. **Address.**
   Es Importante realizar el siguiete paso:
   ⇒ Unmapped Address ⇒ periferico_0,1,n ⇒ en Size: U ⇒ 64K.

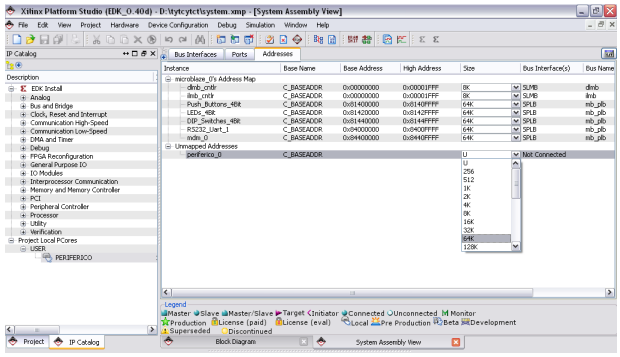


Figura B.14: Configurando el tamaño de memoria del periférico.

4. **Generate Addresses.**
   Nota: Este paso genera de forma automática la dirección de almacenamiento.

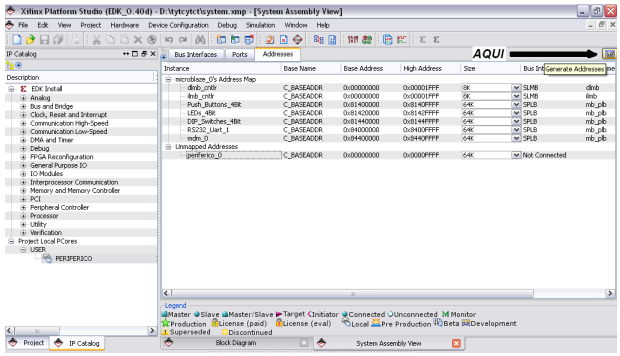


Figura B.15: Generando el rango de direcciones asociadas al periférico.

5. **Actualizar XPS.**
   Nota: Es necesario realizar los siguientes pasos:
   Ir a la barra de menú y seleccionar ⇒ Project ⇒ Rescan User Repositories
   Nota: Se debe generar de inmediato el periférico en el Block Diagram.

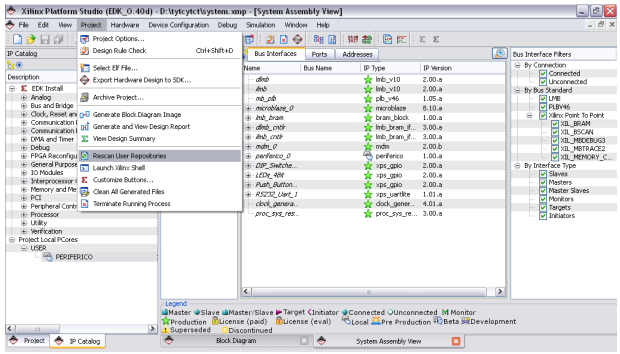


Figura B.16: Diagrama de bloques del sistema.

De esta forma se realiza la creación y conexión de un periférico de hardware con características en la herramienta de trabajo XPS. Todas la imágenes utilizadas aquí, fueron tomadas de la versión 13.1 del software de trabajo.

ANEXO C. Guía para la creación de un periférico especial

C.1 Introducción

Este Anexo muestra el proceso de creación y configuración de un periférico especial sección 1.4.1, para lo cual es necesario modificar las fuentes a disposición del usuario.

C.2 Objetivos

- Realizar la instancia del hardware diseñado a los archivos user_logic.vhd y periferico.vhd.
- Configurar las fuentes .pao, .mpd y .ucf para hacer visible el hardware.

C.3 Procedimiento

Los siguientes archivos que se encuentran al interior de la carpeta del proyecto son los que se modificarán para llevar a buen termino el procedimiento:
⇒ user_logic.vhd
⇒ periferico.vhd
⇒ periferico.pao
⇒ periferico.mpd
⇒ system.ucf

1. **Modificación del archivo user_logic.vhd**
   Este archivo se encuentra siguiendo la ruta:
   ⇒ pcores ⇒ "periférico_diseñado"⇒ hdl ⇒ vhdl ⇒ user_logic.vhd
   Nota: El código fuente se debe incluir en los espacios reservados para le usuario. A continuación se presenta un fragmento de código tomado de la plantilla user_logic.vhd, en donde se muestran los campos que el usuario debe modificar.
   Nota: Luego de la configuración es necesario guardar los cambios realizados.

```
1  ----------------------------------------------------
2  -- FRAGMENTO EXTRAIDO DE UN ARCHIVO user_logi.vhd --
3  ----------------------------------------------------
4  ---------
5  --LIBRERIAS--
6  ---------
7  --USER libraries added here
8    Nota: Debajo de esta linea se copian las librerias correspondientes.
9  -------
10 --ENTIDAD--
11 -------
12 --USER generics added here
13   Nota: Debajo de esta linea se copian las variables genericas correspondientes.
14 --USER ports added here
15   Nota: Debajo de esta linea se copian los puertos I/O correspondientes.
16 -----------
17 --ARQUITECTURA--
18 -----------
19 --USER signal declarations added here, as needed for user logic
20   Nota: Debajo de esta linea se copian las senales correspondientes.
21 -----
22 --BEGIN--
23 -----
24 --USER logic implementation added here
25   Nota: Debajo de esta linea se copia toda la marana de codigo combinacional
26         y secuencial correspondiente.
```

2. **Modificación del archivo periferico.vhd**

   Este archivo se encuentra siguiendo la ruta:

   ⇒ pcores ⇒ ”periférico_diseñado”⇒ hdl ⇒ vhdl ⇒ periferico.vhd

   Nota: El código fuente se debe incluir en los espacios reservados para le usuario. A continuación se presenta un fragmento de código tomado de la plantilla user_logic.vhd, en donde se muestran los campos que el usuario debe modificar.

   Nota: Luego de la configuración es necesario guardar los cambios realizados.

```
1  ----------------------------------------------------
2  -- FRAGMENTO EXTRAIDO DE UN ARCHIVO periferico.vhd --
3  ----------------------------------------------------
4  -------
5  --ENTIDAD--
6  -------
7  --USER generics added here
8    Nota: Debajo de esta linea se copian las variables genericas correspondientes.
9  --USER ports added here
10   Nota: Debajo de esta linea se copian los puertos I/O correspondientes.
11 --------------------------------
12 --BEGIN instancia del user\_logic--
13 --------------------------------
14 --USER generics mapped here
15   Nota: Debajo de esta linea se copian las variables genericas del user\_logic.
16 --USER ports mapped here
17   Nota: Debajo de esta linea se copian los puertos I/O del user\_logic.
```

3. **Modificación del archivo periferico.pao**
   Este archivo se encuentra siguiendo la ruta:
   ⇒ pcores ⇒ ”periférico_diseñado”⇒ data ⇒ periferico.pao
   Nota: Este archivo se modifica si y solo si, existen fuentes .vhd adicionales que complementan el diseño. Si este es el caso, entonces es necesario abrir el archivo periferico.pao y agregar los nombres de estas fuentes como se muestra a continuación.
   Nota: Luego de la configuración es necesario guardar los cambios realizados.

```
1  ##############################################
2  ## FRAGMENTO EXTRAIDO DE UN ARCHIVO pao ##
3  ##############################################
4
5  ##################################################
6  ## Filename:
7  ## Description:
8  ## Date:
9  ##################################################
10
11 lib proc_common_v3_00_a all
12 lib plbv46_slave_single_v1_01_a all
13 lib perif_v1_00_a user_logic vhdl
14 lib perif_v1_00_a periferico vhdl
15
16 ## ADICIONAR ABAJO DE ESTAS LINEAS ##
17 ##          LAS FUENTES VHD          ##
18 --> lib periferico_v1_00_a NOMBRE_DE_LA_FUENTE_1 vhdl
19 --> lib periferico_v1_00_a NOMBRE_DE_LA_FUENTE_2 vhdl
20 --> lib periferico_v1_00_a NOMBRE_DE_LA_FUENTE_n vhdl
```

4. **Modificación del archivo periferico.mpd**
   Este archivo se encuentra siguiendo la ruta:
   ⇒ pcores ⇒ ”periferico_diseñado”⇒ data ⇒ periferico.mpd
   Nota: Este archivo se modifica si y solo si, existen puertos I/O del periférico que se necesiten llevar al exterior. Si este es el caso entonces es necesario abrir el archivo periferico.mpd y realizar las modificaciones como se muestra a continuación.
   Nota: Luego de la configuración es necesario guardar los cambios realizados.

```
1  ##############################################
2  ## FRAGMENTO EXTRAIDO DE UN ARCHIVO mpd ##
3  ##############################################
4
5  ##################################################
6  ## Name :
7  ## Desc :
8  ##################################################
9
10 BEGIN periferico
11
12 ## Peripheral Options
13 ...
14 ## Bus Interfaces
15 ...
```

```
16  ## Ports
17  Nota: Debajo de esta linea se copia la asignacion de puertos.
18
19  Para una entrada std_logic        ==> PORT entrada_1 = "", DIR = I
20  Para una entrada std_logic_vector ==> PORT entrada_2 = "", DIR = I, VEC = [0:N]
21  Para una salida std_logic         ==> PORT salida_1  = "", DIR = O
22  Para una salida std_logic_vector  ==> PORT salida_2  = "", DIR = O, VEC = [0:N]
23
24  END
```

5. **Puertos externos**
   Un vez terminado los pasos anteriores es necesario dirigirse al entorno de XPS y realizar los siguiente:
   ⇒ Ports ⇒ periferico: ⇒ make external
   ⇒ Ports ⇒ External: aqui aparecen los puertos I/O

6. **Modificación del archivo system.ucf**
   Este archivo se encuentra siguiendo la ruta:
   ⇒ data ⇒ system.ucf
   Nota: En este archivo se realiza la correspondiente asignación de pines I/O de acuerdo a la hoja de datos.
   Nota: Luego de la configuración es necesario guardar los cambios realizados y cerrar el archivo.

De esta forma se realiza la creación y conexión de un periférico especial en la herramienta de trabajo XPS.

ANEXO D. Guía para la creación de un nuevo proyecto en SDK

D.1 Introducción

Este Anexo muestra el proceso de creación de un simple diseño de software sección 1.5, utilizando *Software Development Kit (SDK)*.

D.2 Objetivos

- Exportar el hardware a SDK.
- Escribir una aplicación básica que acceda a un periférico en SDK.
- Generar el archivo .bit.
- Programar el sistema de desarrollo.

D.3 Procedimiento

Los pasos que se seguirán para llevar a buen termino el procedimiento se muestran a continuación:

1. **Abrir la plataforma de trabajo.**
   En el siguiente script se encuentra la linea de comando a ejecutar para ejecutar la herramienta de trabajo XPS.

```
1  consola@linux:~$ xsdk
```

2. **Seguidamente se despliega las siguiente ventana:**

   - **ventana 1. Workspace Launcher.**
     En este cuadro de diálogo es necesario escoger la opción:
     ⇒ Workspace ⇒ Browse...
     Nota: Luego de la configuración dar clik en Ok.

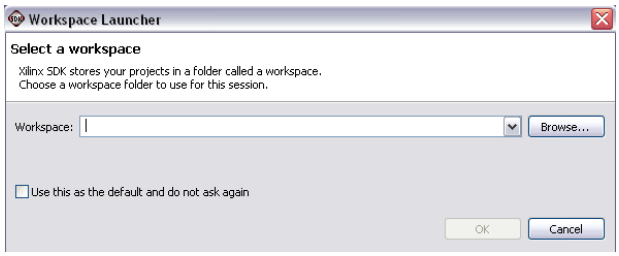


Figura D.1: Cuadro de diálogo *Workspace Launcher*.

3. **En la barra de menu de SDK.**
   Seleccionar la opción:
   ⇒ File ⇒ New Project ⇒ "escoger lenguaje de programación de preferencia"

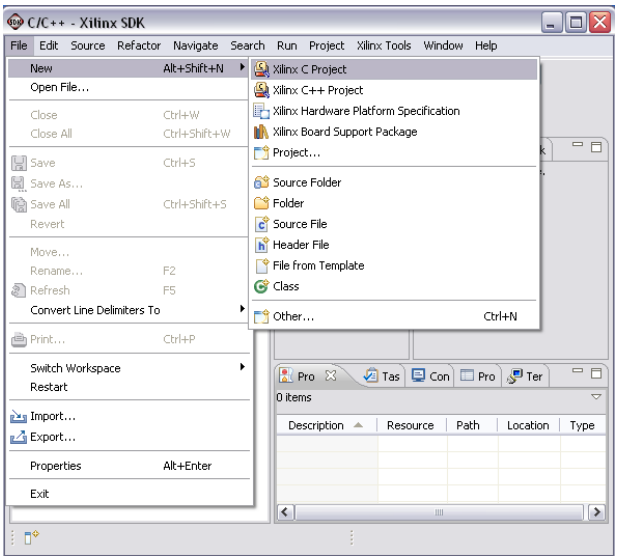


Figura D.2: Escogiendo el lenguaje de preferencia.

4. **Seguidamente se despliegan las siguientes ventanas:**

   - **No Hardware Plataform in the Workspace.**
     En este cuadro de diálogo es necesario escoger la opción:
     ⇒ Specify

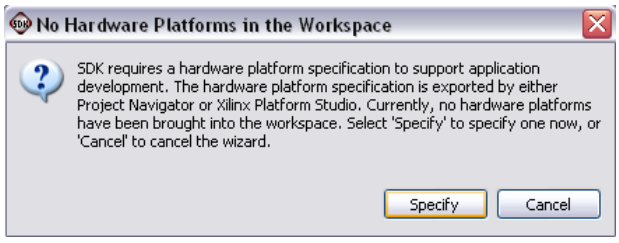


Figura D.3: Cuadro de diálogo *No Hardware Plataform in the Worksapce*.

- **New Hardware Project.**
  En este cuadro de diálogo es necesario escoger las opciónes:
  ⇒ Project name: “aquí dar nombre al proyecto”
  ⇒ Target Hardware Specification ⇒ Browse: “Buscar la carpeta del proyecto XPS” ⇒ _xps ⇒ system.xml
  ⇒ Bitstream and BMM Files ⇒ Browse: “Buscar en la carpeta del proyecto XPS” ⇒ implementación ⇒ “archivo”.bit
  ⇒ BMM File ⇒ Browse: “Buscar en la carpeta del proyecto XPS” ⇒ implementación ⇒ “archivo_bd”.bmm
  Nota: Luego de la configuración dar clik en Finish.

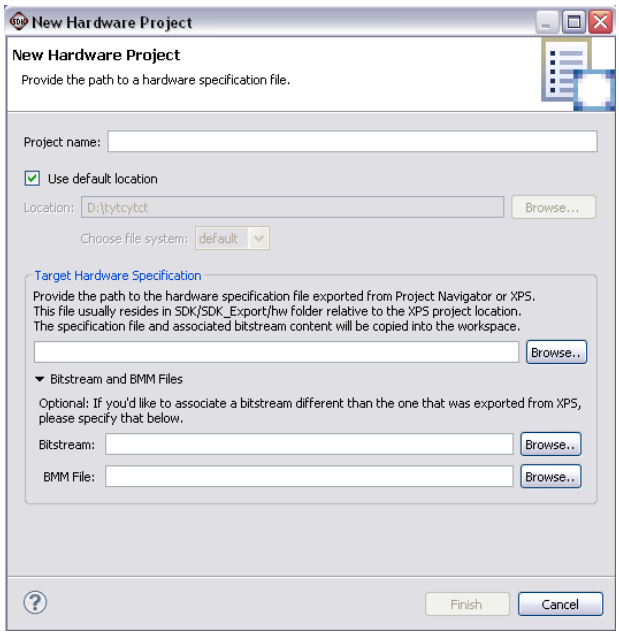


Figura D.4: Cuadro de diálogo *New Hradware Project*.

- **New Project.**
  En este cuadro de diálogo es necesario escoger las opciones:
  ⇒ Slect Project Tempalte: “Escoger algunas de las aplicaciones”
  ⇒ Project name: “Dar un nombre o dejar por defecto”
  Nota: Luego de la configuración dar clik en Next.

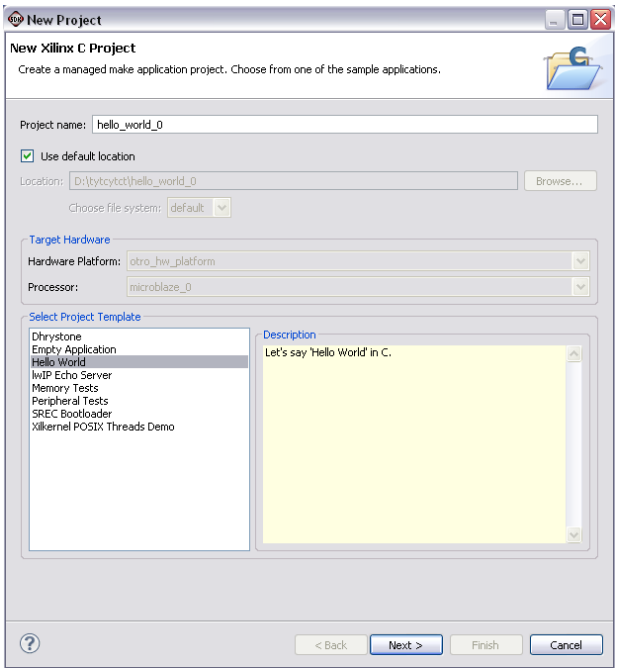


Figura D.5. Cuadro de diálogo *New Project*.

5. **Creación del código según el lenguaje programación a usar.**
   En el entorno de trabajo de SDK seleccionar la opción:
   ⇒ Project Explorer ⇒ src ⇒ “codigo”.c

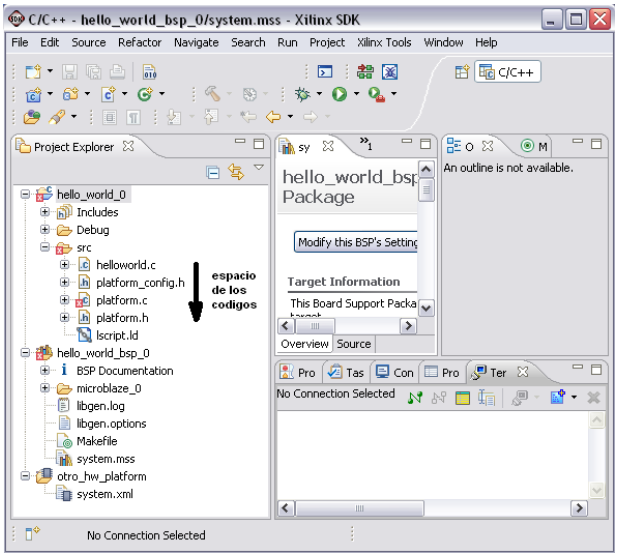


Figura D.6: Cuadro de diálogo para la creación de la aplicación.

6. **implementación.**
   Nota: Luego de haber creado y guardado el código correspondiente se procede como sigue:

- **Programar FPGA, esta opción se encuentra en la barra de menú de SDK.**
  Seleccionar ⇒ Xilinx Tools ⇒ Program FPGA

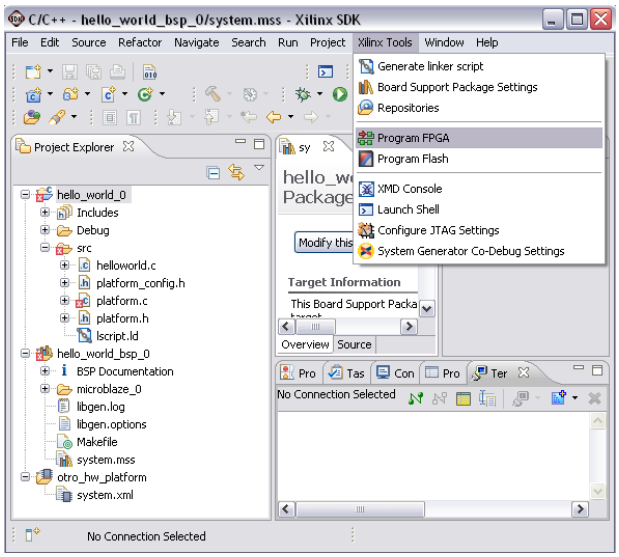


Figura D.7: Cuadro de diálogo para la programación del FPGA.

- **Hacer DEBUG al archivo.elf**
  Nota: Para realizar este paso se debe tener la FPGA correctamente conectada.
  Seleccionar ⇒ Project Explorer ⇒ Binaries ⇒ clik derecho al archivo .elf ⇒ Debug As ⇒ Launch on Hardware.

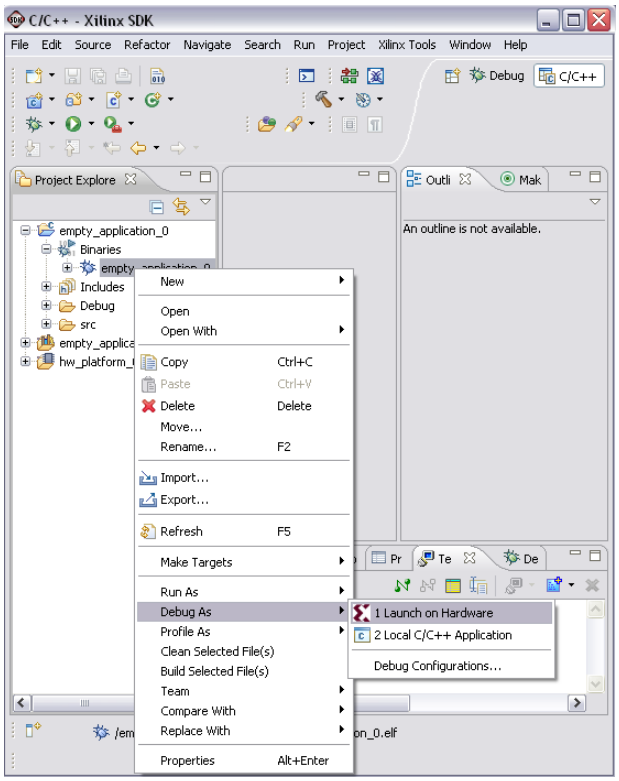


Figura D.8: Cuadro de diálogo para el DEBUG del archivo.elf.

- **En este punto aparece otro entorno de SDK, entonces:**

  ⇒ Dar click izquierdo en el “boton” de play

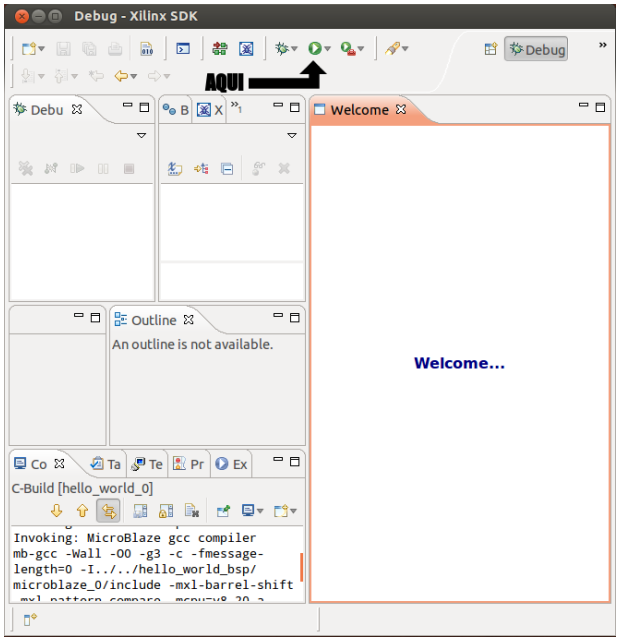


Figura D.9: Cuadro de diálogo para la descarga de la aplicación.

De esta forma se realiza la creación de un proyecto usando la herramienta de trabajo SDK. Todas la imágenes utilizadas aquí, fueron tomadas de la versión 13.1 del software de trabajo.

ANEXO E. Códigos de la validación de periféricos

El presente Anexo muestra el código fuente de los cambios realizados a los archivos de hardware y software empleados en la validación de periféricos.

E.1 Periférico *User Logic Sotfware Register*

E.1.1 Modificando el archivo plantilla user_logic.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  ...
6  entity user_logic is
7  ...
8    port(
9      -- ADD USER PORTS BELOW THIS LINE ---------
10     -- USER ports added here
11     ----------------------------------------
12     ------ ADICIONAR ESTAS DOS LINEAS ------
13     ----------------------------------------
14     dip_switchs : in std_logic_vector(7 downto 0);
15     leds : out std_logic_vector(7 downto 0);
16     ----------------------------------------
17     -- ADD USER PORTS ABOVE THIS LINE ---------
18     ...
19   );
20 ...
21 end entity user_logic;
22 architecture IMP of user_logic is
23 ...
24 begin
25
26   -- USER logic implementation added here
27   ------------------------------------
28   ------ ADICIONAR ESTA LINEA ------
29   ------------------------------------
30   leds <= slv_reg0(24 to 31);
```

```
31   ***********************************
32   ...
33   ---------------------------------------------
34   -- Example code to drive IP to Bus signals --
35   ---------------------------------------------
36   ***********************************
37   ****** REEMPLAZAR ESTA LINEA ******
38   ***********************************
39   IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
40                   (others => '0');
41   ***********************************
42
43   ******************************
44   ****** POR ESTAS LINEAS ******
45   ******************************
46   IP2Bus_Data(0 to 23) <= (others => '0');
47   IP2Bus_Data(24 to 31) <= dip_switchs(7 downto 0);
48   ******************************
49   ...
50 ...
51 end IMP;
```

E.1.2 Modificando el archivo plantilla periferico.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  ...
6  entity periferico is
7  ...
8    port(
9      -- ADD USER PORTS BELOW THIS LINE ---------
10     -- USER ports added here
11     ---------------------------------------
12     ****** ADICIONAR ESTAS DOS LINEAS ******
13     ---------------------------------------
14     dip_switchs : in std_logic_vector(7 downto 0);
15     leds : out std_logic_vector(7 downto 0);
16     ---------------------------------------
17     -- ADD USER PORTS ABOVE THIS LINE ---------
18     ...
19   );
20 ...
21 end entity periferico;
22 architecture IMP of periferico is
23 ...
24 begin
25 ...
26   USER_LOGIC_I : entity periferco.user_logic
27   ...
28   port map (
29     -- MAP USER PORTS BELOW THIS LINE ---------
30     -- USER ports added here
31     ---------------------------------------
```

```
32      ****** ADICIONAR ESTAS DOS LINEAS ******
33      ****************************************
34      dip_switchs => dip_switchs,
35      leds => leds,
36      ****************************************
37      -- MAP USER PORTS ABOVE THIS LINE ---------
38      ...
39    );
40    ...
41  ...
42  end IMP;
```

E.1.3 Aplicación de software

```
1  #include <stdio.h>
2  #include <xparameters.h>
3  #include "periferico.h"
4
5  int main ()
6  {
7    int data_read;
8
9    printf("Aplicacion: INICIO DE LA PRUEBA. \n");
10
11   while(1)
12   {
13     /* Leyendo el estado de los dip switchs. */
14     data_read = PERIFERICO_mReadSlaveReg0(BASSEADDRES, OFFSET);
15
16     /* Impresion de los datos leidos en consola. */
17     printf("Aplicacion: Se leyo el valor **dip_switchs** = 0x%x. \n", data_read);
18
19     /* Configurando las salidas de leds a los valores de los dip switchs. */
20     PERIFERICO_mWriteSlaveReg0(BASSEADDRES, OFFSET, data_read);
21   }
22
23   printf("Aplicacion: FIN DE LA PRUEBA. \n\n\r");
24   return 0;
25 }
```

E.2 Periférico *Read/Write FIFO*

E.2.1 Modificando el archivo plantilla user_logic.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  ...
6  entity user_logic is
7  ...
```

```
8 end entity user_logic;
9 architecture IMP of user_logic is
10
11  --USER signal declarations added here, as needed for user logic
12  ****************************************
13  ****** ADICIONAR ESTAS DOS LINEAS ******
14  ****************************************
15  signal entrada_a       : std_logic_vector(15 downto 0);
16  signal entrada_b       : std_logic_vector(15 downto 0);
17  signal multiplicacion : std_logic_vector(31 downto 0);
18  ****************************************
19  ...
20 ...
21 begin
22
23  -- USER logic implementation added here
24  ***********************************
25  ****** ADICIONAR ESTAS LINEAS ******
26  ***********************************
27  entrada_a       <= WFIFO2IP_Data(0 to 15);
28  entrada_b       <= WFIFO2IP_Data(16 to 31);
29  multiplicacion <= unsigned(entrada_a) * unsigned(entrada_b);
30  IP2RFIFO_Data  <= multiplicacion;
31  ***********************************
32  ...
33 ...
34 end IMP;
```

E.2.2 Aplicación de software

```
1 #include <stdio.h>
2 #include <xparameters.h>
3 #include "periferico.h"
4
5 int main ()
6 {
7   int i;
8   int temp;
9
10  printf("Aplicacion: INICIO DE LA PRUEBA. \n");
11
12  /* Reestableciendo el Read/Write FIFO a su estado inicial. */
13  PERIFERICO_mResetWriteFIFO(BASSEADDRES);
14  PERIFERICO_mResetReadFIFO(BASSEADDRES);
15
16  /* Escribiendo los datos al Write FIFO. */
17  for(i = 1; i <= 4; i++ )
18  {
19    temp = (i << 16) + i;
20    printf("Aplicacion: Datos a multiplicar = 0x%x. \n", temp);
21    PERIFERICO_mWriteToFIFO(BASSEADDRES, OFFSET, temp);
22  }
23
24  /* Guardando los resultados en la Read FIFO. */
25  for(i = 0; i < 4; i++)
```

```
26   {
27     temp = PERIFERICO_mReadFromFIFO(BASSEADDRES, OFFSET);
28     printf("Aplicacion: Resultado de la multiplicacion = 0x%x. \n", temp);
29   }
30
31   /* Reestableciendo el Read/Write FIFO a su estado inicial. */
32   PERIFERICO_mResetWriteFIFO(BASSEADDRES);
33   PERIFERICO_mResetReadFIFO(BASSEADDRES);
34
35   printf("FIN DE LA PRUEBA. \n\n\r");
36   return 0;
37 }
```

E.3 Periférico *Interrupt Control*

E.3.1 Modificando el archivo plantilla user_logic.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  ...
6  entity user_logic is
7  ...
8  end entity user_logic;
9  architecture IMP of user_logic is
10
11   --USER signal declarations added here, as needed for user logic
12   ---------------------------------------
13   ------ ADICIONAR ESTAS DOS LINEAS ------
14   ---------------------------------------
15   signal timer_count   : std_logic_vector(0 to C_SLV_DWIDTH-1);
16   signal timer_expired : std_logic;
17   signal timer_run     : std_logic;
18   senal_1              : std_logic_vector(0 to C_SLV_DWIDTH-1);
19   senal_2              : std_logic;
20   ---------------------------------------
21   ...
22 ...
23 begin
24
25   -- USER logic implementation added here
26   ----------------------------------
27   ------ ADICIONAR ESTAS LINEAS ------
28   ----------------------------------
29   timer_run <= slv_reg1(1);
30
31   process (Bus2IP_Clk, Bus2IP_Reset)
32   begin
33     if (Bus2IP_Reset = '1') then
34       timer_count   <= (others => '0');
35       timer_expired <= '1';
36     elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
37       timer_count   <= senal_1;
```

```
38      timer_expired <= senal_2;
39    end if;
40  end process;
41
42  senal_1 <=  slv_reg0     when (timer_run = '0')             else
43              timer_count  when (timer_count = (others=>'0')) else
44              (timer_count -1);
45
46  senal_2 <=  '0' when (timer_run = '0')             else
47              '1' when (timer_count = (others=>'0')) else
48              '0';
49
50  IP2Bus_IntrEvent(0) <= timer_expired;
51  ***********************************
52
53  ***********************************
54  ****** COMENTAR ESTAS LINEAS ******
55  ***********************************
56  -- ------------------------------------------------
57  -- -- Example code to generate user logic interrupts
58  --
59  -- -- Note:
60  -- -- The example code presented here is to show you one way of generating
61  -- -- interrupts from the user logic. This code snippet infers a counter
62  -- -- and generate the interrupts whenever the counter rollover (the counter
63  -- -- will rollover ~21 sec @50Mhz).
64  -- ------------------------------------------------
65  --
66  -- INTR_PROC : process( Bus2IP_Clk ) is
67  -- constant COUNT_SIZE : integer := 30;
68  -- constant ALL_ONES   : std_logic_vector(0 to COUNT_SIZE-1) := (others => '1');
69  -- variable counter    : std_logic_vector(0 to COUNT_SIZE-1);
70  -- begin
71
72  --  if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
73  --    if ( Bus2IP_Reset = '1' ) then
74  --      counter := (others => '0');
75  --      intr_counter <= (others => '0');
76  --    else
77  --      counter := counter + 1;
78  --      if ( counter = ALL_ONES ) then
79  --        intr_counter <= (others => '1');
80  --      else
81  --        intr_counter <= (others => '0');
82  --      end if;
83  --    end if;
84  --  end if;
85  --
86  -- end process INTR_PROC;
87  --
88  -- IP2Bus_IntrEvent <= intr_counter
89  ***********************************
90  ...
91  ***********************************
92  ****** REEMPLAZAR ESTAS LINEAS ******
93  ***********************************
94  -- implement slave model software accessible register(s) read mux
```

```
95   SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2 ) is
96   begin
97
98     case slv_reg_read_sel is
99       when "00" => slv_ip2bus_data <= slv_reg0;
100      when "01" => slv_ip2bus_data <= slv_reg1;
101      when others => slv_ip2bus_data <= (others => '0');
102    end case;
103
104  end process SLAVE_REG_READ_PROC;
105  ************************************
106
107  ******************************
108  ****** POR ESTAS LINEAS ******
109  ******************************
110   -- implement slave model software accessible register(s) read mux
111  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1 ) is
112  begin
113
114    case slv_reg_read_sel is
115      when "00" => slv_ip2bus_data <= slv_reg0;
116      when "01" => slv_ip2bus_data(1 to C_SLV_DWIDTH-1) <= slv_reg1(1 to C_SLV_DWIDTH-1);
117                   slv_ip2bus_data(0) <= timer_expired;
118      when others => slv_ip2bus_data <= (others => '0');
119    end case;
120
121  end process SLAVE_REG_READ_PROC;
122  ******************************
123 ...
124 end IMP;
```

E.3.2 Aplicación de software

```
1  #include <stdio.h>
2  #include <xparameters.h>
3  #include <xgpio.h>
4  #include <xintc.h>
5  #include "periferico.h"
6
7  #define LEDChan            1
8  #define TIMER_TENSEC       0x3B9ACA00
9  #define TIMER_RUN_LOAD     0x00000000
10 #define TIMER_RUN_RUN      0x20000000
11 #define TIMER_RESET        0x00000000
12
13 unsigned int timer_count = 1;
14 unsigned int count = 1;
15 int one_second_flag = 0;
16
17 XGpio GpioOutput;
18
19 void PERIFERICO_Intr_Handler(void * baseaddr_p)
20 {
21   static   int led_data;
22   unsigned int IpStatus;
```

```
23
24    printf("Aplicacion: HE ENTRADO AL MANEJADOR DE INTERRUPCIONES. \n");
25
26    /* Obteniendo la interrupccion del temporizador. */
27    printf("Aplicacion: Obteniendo la interrupccion del temporizador. \n");
28    IpStatus = PERIFERICO_mReadReg(BASEADDRESS, OFFSET);
29    printf("Aplicacion: El valor de la interrupcion es IpStatus = %x \n", IpStatus);
30
31    if (IpStatus)
32    {
33      /* Condicion 1, perender y apagar todos los LEDs. */
34      printf("Aplicacion: Condicion 1, perender y apagar todos los LEDs. \n");
35      XGpio_DiscreteWrite(&GpioOutput, LEDChan, 0x0000000F);
36
37      /* Si la interrupcion ocurre, entonces se incrementa un contador. */
38      count++;
39      one_second_flag = 1;
40    }
41
42    /* Condicion 2, prender LEDs consecutivos. */
43    printf("Aplicacion: Condicion 2, perender LEDs consecutivos. \n");
44    XGpio_DiscreteWrite(&GpioOutput, LEDChan, count);
45    printf("Aplicacion: El valor de contador es = %x. \n", count);
46
47    /* Cargando y ejecutando la temporizacion. */
48    printf("Aplicacion: Cargando y ejecutando la temporizacion. \n");
49    PERIFERICO_mWriteSlaveReg1(BASEADDRESS, OFFSET, TIMER_RUN_LOAD);
50    PERIFERICO_mWriteSlaveReg1(BASEADDRESS, OFFSET, TIMER_RUN_RUN);
51  }
52
53  int main ()
54  {
55    int count_mod_3;
56
57    printf("Aplicacion: INICIO DE LA PRUEBA \n");
58
59    /* Habilitando interrupcciones para el procesador. */
60    printf("Aplicacion: Habilitando interrupcciones para el Procesador. \n");
61    microblaze_enable_interrupts();
62
63    /* Registro del temporizador en la tala de vectores del manejador de interrupcciones. */
64    printf("Aplicacion: Registro del tempotizador en la tabla de vectores del manejador de interrupcciones.  \n");
65    XIntc_RegisterHandler(BASEADDRESS, ID, (XInterruptHandler)PERIFERICO_Intr_Handler, (void *)BASEADDRESS);
66
67    /* Inicializando y configurando la direccion del GPIO conectados a los LEDs. */
68    printf("Aplicacion: Inicializando y configurando la direccion del GPIO conectados a los LEDs. \n");
69    XGpio_Initialize(&GpioOutput, ID);
70    XGpio_SetDataDirection(&GpioOutput, LEDChan, 0x00000000);
71
72    /* Condicion 1, perender y apagar todos los LEDs. */
73    printf("Aplicacion: Condicion 1, perender y apagar todos los LEDs. \n");
74    XGpio_DiscreteWrite(&GpioOutput, LEDChan, 0x0000000F);
75
76    /* Condicion 2, prender LEDs consecutivos. */
77    printf("Aplicacion: Condicion 2, perender LEDs consecutivos. \n");
78    XGpio_DiscreteWrite(&GpioOutput, LEDChan, count);
79
```

```
80   /* Habilitando el controlador principal de interrupcciones. */
81   printf("Aplicacion: Inicializando el controlador princpal de interrupcciones. \n");
82   XIntc_MasterEnable(XPAR_XPS_INTC_0_BASEADDR);
83   XIntc_EnableIntr(XPAR_XPS_INTC_0_BASEADDR, XPAR_PERIFERICO_0_IP2INTC_IRPT_MASK);
84
85   /* Configurando el numero de ciclos y cargando la temporizacion. */
86   printf("Aplicacion: Configurando el numero de ciclos y ejecutando la temporizacion. \n");
87   PERIFERICO_mWriteSlaveReg0(BASEADDRESS, OFFSET, TIMER_TENSEC);
88   PERIFERICO_mWriteSlaveReg1(BASEADDRESS, OFFSET, TIMER_RUN_LOAD);
89
90   /* Habilitando el temporizador en el periferico de interrupcciones. */
91   printf("Aplicacion: Habilitando el temporizador en el periferico de interrupcciones. \n");
92   PERIFERICO_EnableInterrupt((void *)BASEADDRESS);
93
94   /* Ejecutando la temporizacion. */
95   printf("Aplicacion: Ejecutando la temporizacion. \n");
96   PERIFERICO_mWriteSlaveReg1(BASEADDRESS, OFFSET, TIMER_RUN_RUN);
97
98   while(1)
99   {
100    /* Condicion 2, prender LEDs consecutivos. */
101    printf("Aplicacion: Condicion 2, perender LEDs consecutivos. \n");
102    if(one_second_flag)
103    {
104      count_mod_3 = count %3;
105      if(count_mod_3 == 0)
106      {
107        printf("Aplicacion: Interrupcion tomada en %d segundos. \n", count);
108        one_second_flag=0;
109      }
110    }
111  }
112
113  printf("FIN DE LA PRUEBA. \n\n\r");
114  return 0;
115 }
```

E.4 Periférico *Software Reset*

E.4.1 Modificando el archivo plantilla user_logic.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  ...
6  entity user_logic is
7  ...
8    port(
9      -- ADD USER PORTS BELOW THIS LINE ------------
10     -- USER ports added here
11     ----------------------------------
12     ------ ADICIONAR ESTA LINEA ------
13     ----------------------------------
```

```
14     leds : out std_logic_vector(7 downto 0);
15     ----------------------------------
16     -- ADD USER PORTS ABOVE THIS LINE ---------
17     ...
18   );
19 ...
20 end entity user_logic;
21 architecture IMP of user_logic is
22
23   -- USER signal declarations added here, as needed for user logic
24   ------------------------------------
25   ****** ADICIONAR ESTAS LINEAS ******
26   ------------------------------------
27   -- numero de bist de estado --
28   constant n_state_bits : integer := 4 + 8;
29
30   -- declaracion de los estados --
31   constant S0  : std_logic_vector(n_state_bits-1 downto 0) := "0000" & "00000000";
32   constant S1  : std_logic_vector(n_state_bits-1 downto 0) := "0001" & "00000001";
33   constant S2  : std_logic_vector(n_state_bits-1 downto 0) := "0010" & "00000010";
34   constant S3  : std_logic_vector(n_state_bits-1 downto 0) := "0011" & "00000100";
35   constant S4  : std_logic_vector(n_state_bits-1 downto 0) := "0100" & "00001000";
36   constant S5  : std_logic_vector(n_state_bits-1 downto 0) := "0101" & "00010000";
37   constant S6  : std_logic_vector(n_state_bits-1 downto 0) := "0110" & "00100000";
38   constant S7  : std_logic_vector(n_state_bits-1 downto 0) := "0111" & "01000000";
39   constant S8  : std_logic_vector(n_state_bits-1 downto 0) := "1000" & "10000000";
40   constant S9  : std_logic_vector(n_state_bits-1 downto 0) := "1001" & "11111111";
41   constant S10 : std_logic_vector(n_state_bits-1 downto 0) := "1010" & "00000000";
42   constant S11 : std_logic_vector(n_state_bits-1 downto 0) := "1011" & "10000001";
43   constant S12 : std_logic_vector(n_state_bits-1 downto 0) := "1100" & "01000010";
44   constant S13 : std_logic_vector(n_state_bits-1 downto 0) := "1101" & "00100100";
45   constant S14 : std_logic_vector(n_state_bits-1 downto 0) := "1110" & "00011000";
46
47   -- senales del estado presente y el estado futuro --
48   signal presente, futuro : std_logic_vector(n_state_bits-1 downto 0) := S0;
49   ------------------------------------
50   ...
51 ...
52 begin
53
54   -- USER logic implementation added here
55   ------------------------------------
56   ****** ADICIONAR ESTAS LINEAS ******
57   ------------------------------------
58   dip_switch <= slv_reg0(30 to 31);
59
60   -- logica de registro --
61   process(Bus2IP_Clk, Bus2IP_Reset)
62   begin
63     if (Bus2IP_Reset = '1') then
64       presente <= s0;
65     elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
66       presente <= siguiente;
67     end if;
68   end process;
69
70   -- logica del estado siguiente --
```

```
71   process(presente, Dip_Switch)
72   begin
73     case presente is
74       when s0 => if (dip_switch = "00") then
75                     futuro <= s1;
76                  elsif (dip_switch = "01") then
77                     futuro <= s9;
78                  elsif (dip_switch = "10") then
79                     futuro <= s11;
80                  end if;
81
82       -- primera secuencia --
83       when s1 => futuro <= s2;
84       when s2 => futuro <= s3;
85       when s3 => futuro <= s4;
86       when s4 => futuro <= s5;
87       when s5 => futuro <= s6;
88       when s6 => futuro <= s7;
89       when s7 => futuro <= s8;
90       when s8 => futuro <= s1;
91
92       -- segunda secuencia --
93       when s9 => futuro <= s10;
94       when s10 => futuro <= s9;
95
96       -- tercera secuencia --
97       when s11 => futuro <= s12;
98       when s12 => futuro <= s13;
99       when s13 => futuro <= s14;
100      when s14 => futuro <= s11;
101      when others => futuro <= S0;
102    end case;
103  end process;
104
105  -- logica de salida --
106  leds <= presente(7 downto 0);
107  ********************************
108   ...
109 ...
110 end IMP;
```

E.4.2 Modificando el archivo plantilla periferico.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  ...
6  entity user_logic is
7  ...
8    port(
9      -- ADD USER PORTS BELOW THIS LINE ---------
10     -- USER ports added here
11     ********************************
12     ****** ADICIONAR ESTA LINEA ******
```

```
13     ********************************
14     leds : out std_logic_vector(7 downto 0);
15     ********************************
16     -- ADD USER PORTS ABOVE THIS LINE ---------
17     ...
18   );
19 ...
20 end entity user_logic;
21 architecture IMP of user_logic is
22 ...
23 begin
24 ...
25   USER_LOGIC_I : entity periferco.user_logic
26   ...
27   port map (
28     -- ADD USER PORTS BELOW THIS LINE ---------
29     -- USER ports added heree
30     ********************************
31     ****** ADICIONAR ESTA LINEA ******
32     ********************************
33     leds => leds,
34     ********************************
35     -- MAP USER PORTS ABOVE THIS LINE ---------
36     ...
37   );
38   ...
39 ...
40 end IMP;
```

E.4.3 Aplicación de software

```
1  #include <stdio.h>
2  #include <xparameters.h>
3  #include <xgpio.h>
4  #include <xutil.h>
5  #include "periferico.h"
6
7  int main(void)
8  {
9    int dip_switch;
10   XGpio dip_switch_8bits;
11
12   printf("Aplicacion: INICIO DE L APRUEBA. \n");
13
14   /* inicializando el periferico Dip Switch. */
15   XGpio_Initialize(&dip_switch_8bits, ID);
16
17   /* Esatbleciendo el periferico Dip Switch. ala entrada. */
18   XGpio_SetDataDirection(&dip_switch_8bits, 1, 0x0000001F);
19
20   while(1)
21   {
22     /* Leyendo el periferico Dip Switch. */
23     dip_switch = XGpio_DiscreteRead(dip_switch_8bits, 1);
24     printf("Aplicacion: El estado de los **dip_switchs** = %x. \n", dip_switch);
```

```
25
26     /* Enviando la informacion al periferico. */
27     if (dip_switch == 0)
28     {
29       PERIFERICO_mWriteSlaveReg0(BASEADDRESS, OFFSET, 0x00000000);
30     }
31
32     if (dip_switch == 1)
33     {
34       PERIFERICO_mWriteSlaveReg0(BASEADDRESS, OFFSET, 0x00000001);
35     }
36
37     if (dip_switch == 2)
38     {
39       PERIFERICO_mWriteSlaveReg0(BASEADDRESS, OFFSET, 0x00000002);
40     }
41
42     if (dip_switch == 3)
43     {
44       PERIFERICO_mReset(BASEADDRESS);
45       printf("Aplicacion: Escribiendo el valor 0x%x al registro del periferico. \n", PERIFERICO_mReset(BASEADDRESS));
46     }
47   }
48
49   printf("FIN DE LA PRUEBA \n\n\r");
50   return 0;
51 }
```

E.5 Periférico *User Logic Memory Space*

E.5.1 Aplicación de software

```
1  #include <stdio.h>
2  #include <xgpio.h>
3  #include <xparameters.h>
4  #include "periferico.h"
5
6  int main(void)
7  {
8    int Index, Mem32Value;
9
10   printf("Aplicacion: INICIO DE L APRUEBA. \n");
11
12   /* Escribiendo y leyendo datos en el BRAM. */
13   printf("Aplicacion: Escribiendo datos en el BRAMs. \n");
14   for ( Index = 0; Index < 256; Index++ )
15   {
16     PERIFERICO_mWriteMemory(BASEADDRESS+4*Index, 0xDEADBEEF);
17     Mem32Value = PERIFERICO_mReadMemory(BASEADDRESS+4*Index);
18     if ( Mem32Value != 0xDEADBEEF )
19     {
20       printf("Aplicacion: La Escritura/Lectura en el BRAM fallo en la direccion 0x%x. \n", BASEADDRESS+4*Index);
21     } else {
22       printf("Aplicacion: La Escritura/Lectura en el BRAM es correcta. \n\r");
```

```
23      }
24    }
25
26    printf("Aplicacion: FIN DE LA PRUEBA \n\n\r");
27    return 0;
28 }
```

ANEXO F. Implementación del interfaz de periféricos

Esta implementación utiliza un mínimo de módulos en lenguaje de descripción de hardware HDL, que garantizan el correcto funcionamiento y pueden ser usadas en futuras aplicaciones que involucren los periféricos definidos en este trabajo.

F.1 Fuentes Compartidas

En el diagrama de bloques de la Figura 3.4 se encuentra el hardware diseñado para la interfaz de periféricos. A continuación se presenta el código fuente de los módulos que no cambian su descripción y estructura interna.

F.1.1 Sincronizador de datos

La Figura F.1 muestra el diagrama de bloques del modulo sincronizador de datos.

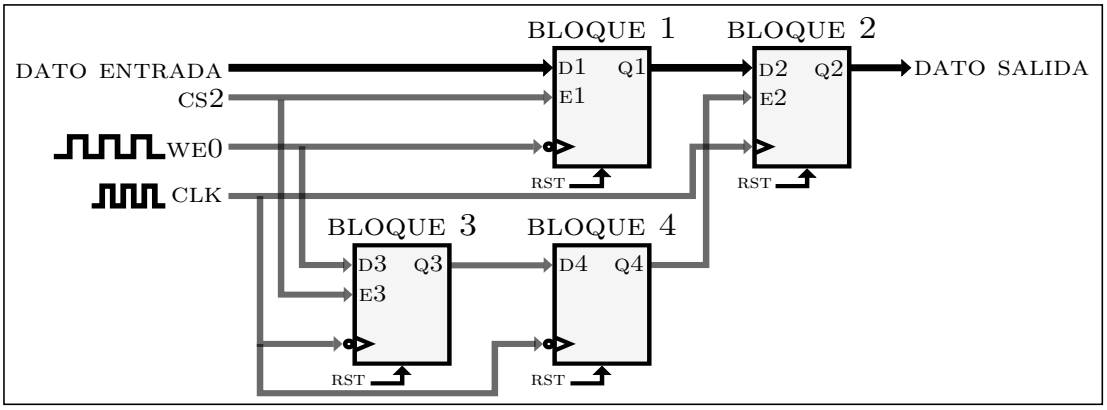


Figura F.1: Sincronizador de datos.
FUENTE: Adaptado de [12].

A continuación se presenta su código en lenguaje de descripción de hardware HDL:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity sincronizador_reloj_bus_datos is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_we0 : in std_logic;
12     entrada_cs2 : in std_logic;
13     entrada_bus_datos : in std_logic_vector(7 downto 0);
14     salida_sincronizador_datos : out std_logic_vector(7 downto 0)
15   );
16
17 end sincronizador_reloj_bus_datos;
18
19 architecture Behavioral of sincronizador_reloj_bus_datos is
20
21   signal d1, q1 : std_logic_vector(7 downto 0);
22   signal d2, q2 : std_logic_vector(7 downto 0);
23   signal d3, q3 : std_logic;
24   signal d4, q4 : std_logic;
25
26 begin
27
28   BLOQUE_1: process(rst, entrada_we0)
29   begin
30      if (rst='0') then
31         q1 <= (others=>'0');
32      elsif (entrada_we0'event and entrada_we0='0') then
33         q1 <= d1;
34      end if;
35   end process;
36
37   d1 <= entrada_bus_datos when entrada_cs2='0' else
38         q1;
39
40   BLOQUE_2: process(rst, clk)
41   begin
42      if (rst='0') then
43         q2 <= (others=>'0');
44      elsif (clk'event and clk='1') then
45         q2 <= d2;
46      end if;
47   end process;
48
49   d2 <= q1 when q4='1' else
50         q2;
51
52   BLOQUE_3: process(rst, clk)
53   begin
54      if (rst='0') then
55         q3 <= '0';
56      elsif (clk'event and clk='0') then
```

```
57          q3 <= d3;
58       end if;
59    end process;
60
61    d3 <= not(entrada_we0) when entrada_cs2='0' else
62          '0';
63
64    BLOQUE_4: process(rst, clk)
65    begin
66       if (rst='0') then
67          q4 <= '0';
68       elsif (clk'event and clk='0') then
69          q4 <= d4;
70       end if;
71    end process;
72
73    d4 <= q3;
74
75    salida_sincronizador_datos <= q2;
76
77 end Behavioral;
```

F.1.2 Sincronizador de direcciones

La Figura F.2 muestra el diagrama de bloques del modulo sincronizador de direcciones.

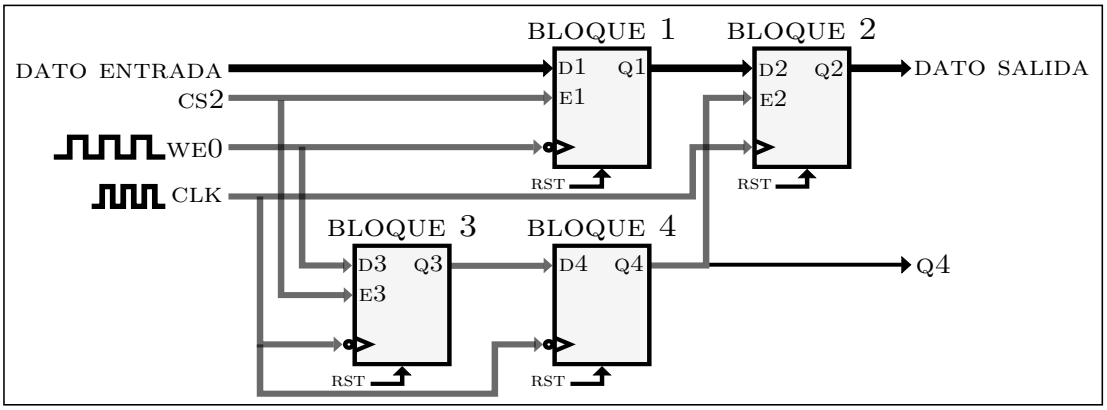


Figura F.2: Sincronizador de direcciones.
FUENTE: Adaptado de [12].

A continuación se presenta su código en lenguaje de descripción de hardware HDL:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity sincronizador_reloj_bus_direcciones is
7
8   port(
9      rst : in std_logic;
```

```
10      clk : in std_logic;
11      entrada_we0 : in std_logic;
12      entrada_cs2 : in std_logic;
13      entrada_bus_datos : in std_logic_vector(7 downto 0);
14      salida_sincronizador_datos : out std_logic_vector(7 downto 0);
15      salida_q4 : out std_logic
16    );
17
18 end sincronizador_reloj_bus_direcciones;
19
20 architecture Behavioral of sincronizador_reloj_bus_direcciones is
21
22    signal d1, q1 : std_logic_vector(7 downto 0);
23    signal d2, q2 : std_logic_vector(7 downto 0);
24    signal d3, q3 : std_logic;
25    signal d4, q4 : std_logic;
26
27 begin
28
29    BLOQUE_1: process(rst, entrada_we0)
30    begin
31       if (rst='0') then
32          q1 <= (others=>'0');
33       elsif (entrada_we0'event and entrada_we0='0') then
34          q1 <= d1;
35       end if;
36    end process;
37
38    d1 <= entrada_bus_datos when entrada_cs2='0' else
39          q1;
40
41    BLOQUE_2: process(rst, clk)
42    begin
43       if (rst='0') then
44          q2 <= (others=>'0');
45       elsif (clk'event and clk='1') then
46          q2 <= d2;
47       end if;
48    end process;
49
50    d2 <= q1 when q4='1' else
51          q2;
52
53    BLOQUE_3: process(rst, clk)
54    begin
55       if (rst='0') then
56          q3 <= '0';
57       elsif (clk'event and clk='0') then
58          q3 <= d3;
59       end if;
60    end process;
61
62    d3 <= not(entrada_we0) when entrada_cs2='0' else
63          '0';
64
65    BLOQUE_4: process(rst, clk)
66    begin
```

```
67      if (rst='0') then
68         q4 <= '0';
69      elsif (clk'event and clk='0') then
70         q4 <= d4;
71      end if;
72   end process;
73
74   d4 <= q3;
75
76   salida_sincronizador_datos <= q2;
77   salida_q4 <= q4;
78
79 end Behavioral;
```

F.1.3 Antirrebotes

La Figura F.3 muestra el diagrama de bloques del modulo antirrebotes.

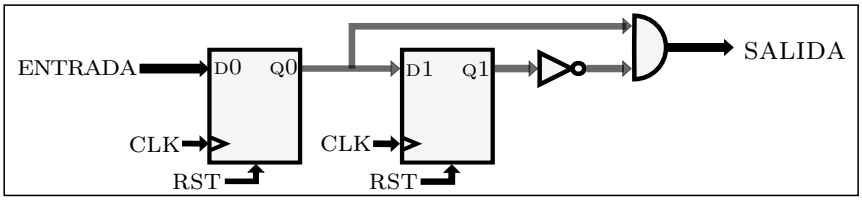


Figura F.3: Antirrebotes.
FUENTE: El autor.

A continuación se presenta su código en lenguaje de descripción de hardware HDL:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity antirrebotes is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada : in std_logic;
12     salida : out std_logic
13   );
14
15 end antirrebotes;
16
17 architecture Behavioral of antirrebotes is
18
19   signal d0, q0 : std_logic;
20   signal d1, q1 : std_logic;
21
22 begin
23
24   process(rst, clk)
25   begin
```

```
26      if (rst='0') then
27         q0 <= '0';
28         q1 <= '0';
29      elsif (clk'event and clk='1') then
30         q0 <= d0;
31         q1 <= d1;
32      end if;
33   end process;
34
35   d0 <= entrada;
36   d1 <= q0;
37   salida <= q0 and not(q1);
38
39 end Behavioral;
```

F.1.4 Buffer tercer estado

La Figura F.4 muestra el diagrama de bloques del modulo tercer estado.

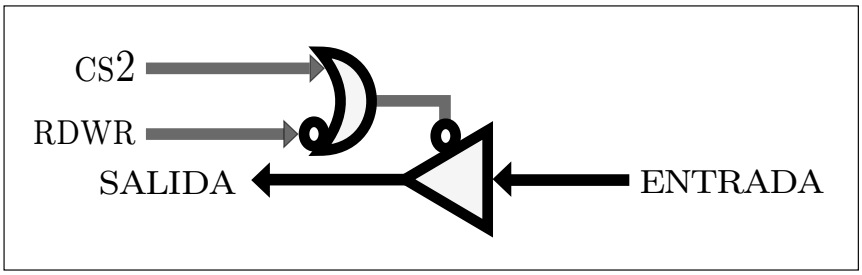


Figura F.4. Buffer tercer estado.
FUENTE: El autor.

A continuación se presenta su código en lenguaje de descripción de hardware HDL:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity tercer_estado is
7
8    port(
9      clk : in std_logic;
10     entrada : in std_logic_vector(7 downto 0);
11     entrada_cs2 : in std_logic;
12     entrada_rdwr : in std_logic;
13     salida : out std_logic_vector(7 downto 0)
14   );
15
16 end tercer_estado;
17
18 architecture Behavioral of tercer_estado is
19
20   signal enable : std_logic;
21
```

```
22 begin
23
24   enable <= not(entrada_rdwr) or entrada_cs2;
25
26   salida <= entrada when enable='0' else
27             (others=>'Z');
28
29 end Behavioral;
```

F.2 Periférico banco de registros

La Figura F.5 muestra el diagrama de bloques de la interfaz diseñada para este periférico.

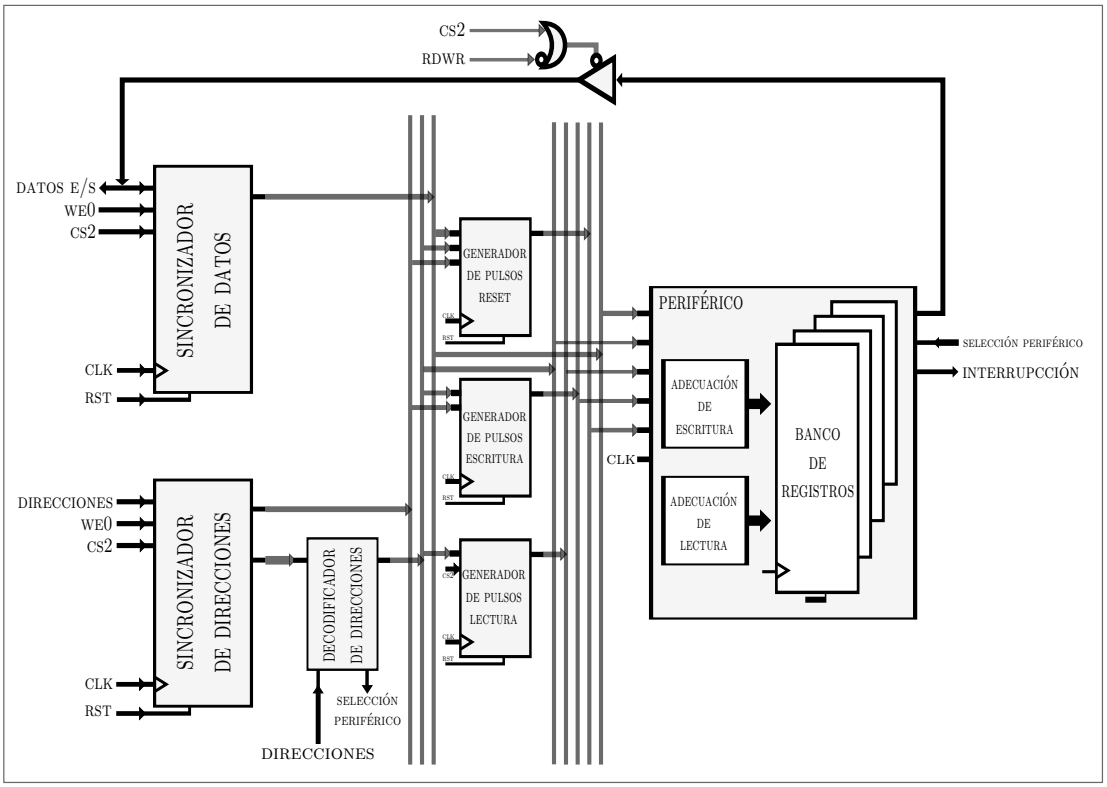


Figura F.5: Interfaz del periférico banco de registros.
FUENTE: El autor.

A continuación se presentan los códigos en lenguaje de descripción de hardware HDL:

F.2.1 Decodificador de direcciones

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity decodificador_direcciones is
7
8    port(
9      entrada_bus_direcciones : in std_logic_vector(12 downto 0);
10     entrada_direcciones_sincronizadas : in std_logic_vector(12 downto 0);
11     salida_direcciones_decodificadas_reset : out std_logic_vector(3 downto 0);
12     salida_direcciones_decodificadas_escritura : out std_logic_vector(2 downto 0);
13     salida_direcciones_decodificadas_lectura : out std_logic_vector(2 downto 0)
14   );
15
16 end decodificador_direcciones;
```

```
17
18 architecture Behavioral of decodificador_direcciones is
19
20 begin
21
22   salida_direcciones_decodificadas_escritura <= "000" when entrada_direcciones_sincronizadas="0000000000001" else
23                                                 "001" when entrada_direcciones_sincronizadas="0000000000010" else
24                                                 "010" when entrada_direcciones_sincronizadas="0000000000011" else
25                                                 "011" when entrada_direcciones_sincronizadas="0000000000100" else
26                                                 "100" when entrada_direcciones_sincronizadas="0000000000101" else
27                                                 "101" when entrada_direcciones_sincronizadas="0000000000110" else
28                                                 "110" when entrada_direcciones_sincronizadas="0000000000111" else
29                                                 "111" when entrada_direcciones_sincronizadas="0000000001000" else
30                                                 (others=>'Z') ;
31
32   salida_direcciones_decodificadas_reset <= "1000" when entrada_direcciones_sincronizadas="0000000001001" else
33                                             (others=>'Z') ;
34
35   salida_direcciones_decodificadas_lectura <= "000" when entrada_bus_direcciones="0000000001010" else
36                                               "001" when entrada_bus_direcciones="0000000001011" else
37                                               "010" when entrada_bus_direcciones="0000000001100" else
38                                               "011" when entrada_bus_direcciones="0000000001101" else
39                                               "100" when entrada_bus_direcciones="0000000001110" else
40                                               "101" when entrada_bus_direcciones="0000000001111" else
41                                               "110" when entrada_bus_direcciones="0000000010000" else
42                                               "111" when entrada_bus_direcciones="0000000010001" else
43                                               (others=>'Z') ;
44
45 end Behavioral;
```

F.2.2 Generador de pulsos de restablecimiento

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_reset is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_q4 : in std_logic;
12     entrada_datos_sincronizados : in std_logic_vector(7 downto 0);
13     entrada_direcciones_reset_decodificadas : in std_logic_vector(3 downto 0);
14     salida_pulso_reset : out std_logic
15   );
16
17 end generador_pulsos_reset;
18
19 architecture Behavioral of generador_pulsos_reset is
20
21   component antirrebotes
22   port(
23     rst : in std_logic;
```

```
24      clk : in std_logic;
25      entrada : in std_logic;
26      salida : out std_logic
27    );
28    end component;
29
30    signal senal_rebote : std_logic;
31
32  begin
33
34    Instancia_antirrebotes: antirrebotes
35    port map(
36      rst => rst,
37      clk => clk,
38      ntrada => entrada_q4,
39      salida => senal_rebote
40    );
41
42    salida_pulso_reset <= '1' when (senal_rebote='1' and
43                                    entrada_datos_sincronizados="00000001" and
44                                    entrada_direcciones_reset_decodificadas="1000") else '0';
45
46  end Behavioral;
```

F.2.3 Generador de pulsos de escritura

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_escritura is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_q4 : in std_logic;
12     entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
13     salida_pulso_escribir : out std_logic
14   );
15
16 end generador_pulsos_escritura;
17
18 architecture Behavioral of generador_pulsos_escritura is
19
20   component antirrebotes
21   port(
22     rst : in std_logic;
23     clk : in std_logic;
24     entrada : in std_logic;
25     salida : out std_logic
26   );
27   end component;
28
29   signal senal_rebote : std_logic;
```

```
30
31 begin
32
33   Instancia_antirrebotes: antirrebotes
34   port map(
35     rst => rst,
36     clk => clk,
37     entrada => entrada_q4,
38     salida => senal_rebote
39   );
40
41   salida_pulso_escribir <= '1' when (senal_rebote='1' and
42                                      (entrada_direcciones_escritura_decodificadas="000" or
43                                       entrada_direcciones_escritura_decodificadas="001" or
44                                       entrada_direcciones_escritura_decodificadas="010" or
45                                       entrada_direcciones_escritura_decodificadas="011" or
46                                       entrada_direcciones_escritura_decodificadas="100" or
47                                       entrada_direcciones_escritura_decodificadas="101" or
48                                       entrada_direcciones_escritura_decodificadas="110" or
49                                       entrada_direcciones_escritura_decodificadas="111" )) else '0';
50
51 end Behavioral;
```

F.2.4 Generador de pulsos de lectura

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_lectura is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_cs2 : in std_logic;
12     entrada_direcciones_lectura_decodificadas: in std_logic_vector(2 downto 0);
13     salida_pulso_leer : out std_logic
14   );
15
16 end generador_pulsos_lectura;
17
18 architecture Behavioral of generador_pulsos_lectura is
19
20 begin
21
22   salida_pulso_leer <= '1' when (entrada_cs2='0' and
23                                  (entrada_direcciones_lectura_decodificadas="000" or
24                                   entrada_direcciones_lectura_decodificadas="001" or
25                                   entrada_direcciones_lectura_decodificadas="010" or
26                                   entrada_direcciones_lectura_decodificadas="011" or
27                                   entrada_direcciones_lectura_decodificadas="100" or
28                                   entrada_direcciones_lectura_decodificadas="101" or
29                                   entrada_direcciones_lectura_decodificadas="110" or
30                                   entrada_direcciones_lectura_decodificadas="111" )) else '0' ;
```

```
31
32 end Behavioral;
```

F.2.5 Banco de registros

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity banco_registros is
7
8    generic(
9      ancho_dato_registros : integer := 8 ;
10     ancho_direcciones_registros : integer := 3
11   );
12
13   port(
14     rst : in std_logic;
15     clk : in std_logic;
16     escribir : in std_logic;
17     leer : in std_logic;
18     direcciones_escritura : in std_logic_vector(ancho_direcciones_registros-1 downto 0);
19     direcciones_lectura : in std_logic_vector(ancho_direcciones_registros-1 downto 0);
20     entrada : in std_logic_vector(ancho_dato_registros-1 downto 0);
21     salida: out std_logic_vector(ancho_dato_registros-1 downto 0)
22   );
23
24 end banco_registros;
25
26 architecture Behavioral of banco_registros is
27
28   signal d0,q0 : std_logic_vector(ancho_dato_registros-1 downto 0);
29   signal d1,q1 : std_logic_vector(ancho_dato_registros-1 downto 0);
30   signal d2,q2 : std_logic_vector(ancho_dato_registros-1 downto 0);
31   signal d3,q3 : std_logic_vector(ancho_dato_registros-1 downto 0);
32   signal d4,q4 : std_logic_vector(ancho_dato_registros-1 downto 0);
33   signal d5,q5 : std_logic_vector(ancho_dato_registros-1 downto 0);
34   signal d6,q6 : std_logic_vector(ancho_dato_registros-1 downto 0);
35   signal d7,q7 : std_logic_vector(ancho_dato_registros-1 downto 0);
36
37 begin
38
39   ----------------------
40   --BANCO DE REGISTROS--
41   ----------------------
42
43   --Logica de registros--
44   REG_0: process(rst, clk)
45   begin
46      if (rst='1') then
47         q0 <= (others=>'0');
48      elsif (clk'event and clk='1') then
49         q0 <= d0;
50      end if;
```

```
51  end process;
52
53  REG_1: process(rst, clk)
54  begin
55     if (rst='1') then
56        q1 <= (others=>'0');
57     elsif (clk'event and clk='1') then
58        q1 <= d1;
59     end if;
60  end process;
61
62  REG_2: process(rst, clk)
63  begin
64     if (rst='1') then
65        q2 <= (others=>'0');
66     elsif (clk'event and clk='1') then
67        q2 <= d2;
68     end if;
69  end process;
70
71  REG_3: process(rst, clk)
72  begin
73     if (rst='1') then
74        q3 <= (others=>'0');
75     elsif (clk'event and clk='1') then
76        q3 <= d3;
77     end if;
78  end process;
79
80  REG_4: process(rst, clk)
81  begin
82     if (rst='1') then
83        q4 <= (others=>'0');
84     elsif (clk'event and clk='1') then
85        q4 <= d4;
86     end if;
87  end process;
88
89  REG_5: process(rst, clk)
90  begin
91     if (rst='1') then
92        q5 <= (others=>'0');
93     elsif (clk'event and clk='1') then
94        q5 <= d5;
95     end if;
96  end process;
97
98  REG_6: process(rst, clk)
99  begin
100    if (rst='1') then
101       q6 <= (others=>'0');
102    elsif (clk'event and clk='1') then
103       q6 <= d6;
104    end if;
105 end process;
106
107 REG_7: process(rst, clk)
```

```
108   begin
109      if (rst='1') then
110         q7 <= (others=>'0');
111      elsif (clk'event and clk='1') then
112         q7 <= d7;
113      end if;
114   end process;
115
116   --proceso de escritura--
117   d0 <= q0 when escribir='0' else
118         q0 when direcciones_escritura/="000" else
119         entrada;
120   d1 <= q1 when escribir='0' else
121         q1 when direcciones_escritura/="001" else
122         entrada;
123   d2 <= q2 when escribir='0' else
124         q2 when direcciones_escritura/="010" else
125         entrada;
126   d3 <= q3 when escribir='0' else
127         q3 when direcciones_escritura/="011" else
128         entrada;
129   d4 <= q4 when escribir='0' else
130         q4 when direcciones_escritura/="100" else
131         entrada;
132   d5 <= q5 when escribir='0' else
133         q5 when direcciones_escritura/="101" else
134         entrada;
135   d6 <= q6 when escribir='0' else
136         q6 when direcciones_escritura/="110" else
137         entrada;
138   d7 <= q7 when escribir='0' else
139         q7 when direcciones_escritura/="111" else
140         entrada;
141
142   --proceso de lectura--
143   salida <= (others=>'Z') when leer='0' else
144             q0 when direcciones_lectura="000" else
145             q1 when direcciones_lectura="001" else
146             q2 when direcciones_lectura="010" else
147             q3 when direcciones_lectura="011" else
148             q4 when direcciones_lectura="100" else
149             q5 when direcciones_lectura="101" else
150             q6 when direcciones_lectura="110" else
151             q7 when direcciones_lectura="111" else
152             (others=>'Z') ;
153
154 end Behavioral;
```

F.2.6 Interfaz principal

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
```

```
6  entity interfaz_periferico_registros is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     cs2 : in std_logic;
12     we0 : in std_logic;
13     rdwr : in std_logic;
14     bus_datos : inout std_logic_vector(7 downto 0);
15     bus_direcciones : in std_logic_vector(12 downto 0)
16   );
17
18 end interfaz_periferico_registros;
19
20 architecture Behavioral of interfaz_periferico_registros is
21
22   component sincronizador_reloj_bus_datos
23   port(
24     rst : in std_logic;
25     clk : in std_logic;
26     entrada_we0 : in std_logic;
27     entrada_cs2 : in std_logic;
28     entrada_bus_datos : in std_logic_vector(7 downto 0);
29     salida_sincronizador_datos : out std_logic_vector(7 downto 0)
30   );
31   end component;
32
33   component sincronizador_reloj_bus_direcciones
34   port(
35     rst : in std_logic;
36     clk : in std_logic;
37     entrada_we0 : in std_logic;
38     entrada_cs2 : in std_logic;
39     entrada_bus_direcciones : in std_logic_vector(12 downto 0);
40     salida_sincronizador_direcciones : out std_logic_vector(12 downto 0);
41     salida_q4 : out std_logic
42   );
43   end component;
44
45   component decodificador_direcciones
46   port(
47     entrada_bus_direcciones : in std_logic_vector(12 downto 0);
48     entrada_direcciones_sincronizadas : in std_logic_vector(12 downto 0);
49     salida_direcciones_decodificadas_reset : out std_logic_vector(3 downto 0);
50     salida_direcciones_decodificadas_escritura : out std_logic_vector(2 downto 0);
51     salida_direcciones_decodificadas_lectura : out std_logic_vector(2 downto 0)
52   );
53   end component;
54
55   component generador_pulsos_reset
56   port(
57     rst : in std_logic;
58     clk : in std_logic;
59     entrada_q4 : in std_logic;
60     entrada_datos_sincronizados : in std_logic_vector(7 downto 0);
61     entrada_direcciones_reset_decodificadas : in std_logic_vector(3 downto 0);
62     salida_pulso_reset : out std_logic
```

```
63   );
64   end component;
65
66   component generador_pulsos_escritura
67   port(
68     rst : in std_logic;
69     clk : in std_logic;
70     entrada_q4 : in std_logic;
71     entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
72     salida_pulso_escribir : out std_logic
73   );
74   end component;
75
76   component generador_pulsos_lectura
77   port(
78     rst : in std_logic;
79     clk : in std_logic;
80     entrada_cs2 : in std_logic;
81     entrada_direcciones_lectura_decodificadas: in std_logic_vector(2 downto 0);
82     salida_pulso_leer : out std_logic
83   );
84   end component;
85
86   component banco_registros
87   port(
88     rst : in std_logic;
89     clk : in std_logic;
90     escribir : in std_logic;
91     leer : in std_logic;
92     direcciones_escritura : in std_logic_vector(2 downto 0);
93     direcciones_lectura : in std_logic_vector(2 downto 0);
94     entrada : in std_logic_vector(7 downto 0);
95     salida: out std_logic_vector(7 downto 0)
96   );
97   end component;
98
99   component tercer_estado
100  port(
101    clk : in std_logic;
102    entrada : in std_logic_vector(7 downto 0);
103    entrada_cs2 : in std_logic;
104    entrada_rdwr : in std_logic;
105    salida : out std_logic_vector(7 downto 0)
106  );
107  end component;
108
109  signal senal_1 : std_logic_vector(7 downto 0);
110  signal senal_2 : std_logic_vector(12 downto 0);
111  signal senal_q4 : std_logic;
112  signal senal_3 : std_logic_vector(3 downto 0);
113  signal senal_4 : std_logic_vector(2 downto 0);
114  signal senal_5 : std_logic_vector(2 downto 0);
115  signal senal_6 : std_logic;
116  signal senal_7 : std_logic;
117  signal senal_8 : std_logic;
118  signal senal_9 : std_logic_vector(7 downto 0);
119
```

```
120 begin
121
122   instancia_sincronizador_reloj_bus_datos: sincronizador_reloj_bus_datos
123   port map(
124     rst => rst,
125     clk => clk,
126     entrada_we0 => we0,
127     entrada_cs2 => cs2,
128     entrada_bus_datos => bus_datos,
129     salida_sincronizador_datos => senal_1
130   );
131
132   instancia_sincronizador_reloj_bus_direcciones: sincronizador_reloj_bus_direcciones
133   port map(
134     rst => rst,
135     clk => clk,
136     entrada_we0 => we0,
137     entrada_cs2 => cs2,
138     entrada_bus_direcciones => bus_direcciones,
139     salida_sincronizador_direcciones => senal_2,
140     salida_q4 => senal_q4
141   );
142
143   instancia_decodificador_direcciones: decodificador_direcciones
144   port map(
145     entrada_bus_direcciones => bus_direcciones,
146     entrada_direcciones_sincronizadas => senal_2,
147     salida_direcciones_decodificadas_reset => senal_3,
148     salida_direcciones_decodificadas_escritura => senal_4,
149     salida_direcciones_decodificadas_lectura => senal_5
150   );
151
152   instancia_generador_pulsos_reset: generador_pulsos_reset
153   port map(
154     rst => rst,
155     clk => clk,
156     entrada_q4 => senal_q4,
157     entrada_datos_sincronizados => senal_1,
158     entrada_direcciones_reset_decodificadas => senal_3,
159     salida_pulso_reset => senal_6
160   );
161
162   instancia_generador_pulsos_escritura: generador_pulsos_escritura
163   port map(
164     rst => rst,
165     clk => clk,
166     entrada_q4 => senal_q4,
167     entrada_direcciones_escritura_decodificadas => senal_4,
168     salida_pulso_escribir => senal_7
169   );
170
171   instancia_generador_pulsos_lectura: generador_pulsos_lectura
172   port map(
173     rst => rst,
174     clk => clk,
175     entrada_cs2 => cs2,
176     entrada_direcciones_lectura_decodificadas => senal_5,
```

```
177     salida_pulso_leer => senal_8
178   );
179
180   instancia_banco_registros: banco_registros
181   port map(
182     rst => senal_6,
183     clk => clk,
184     escribir => senal_7,
185     leer => senal_8,
186     direcciones_escritura => senal_4,
187     direcciones_lectura => senal_5,
188     entrada => senal_1,
189     salida => senal_9
190   );
191
192   instancia_tercer_estado: tercer_estado
193   port map(
194     clk => clk,
195     entrada => senal_9,
196     entrada_cs2 => cs2,
197     entrada_rdwr => rdwr,
198     salida => bus_datos
199   );
200
201 end Behavioral;
```

F.3 Periférico memoria RAM

En la Figura F.6 se encuentra el diagrama de bloques de la interfaz diseñada para este periférico.

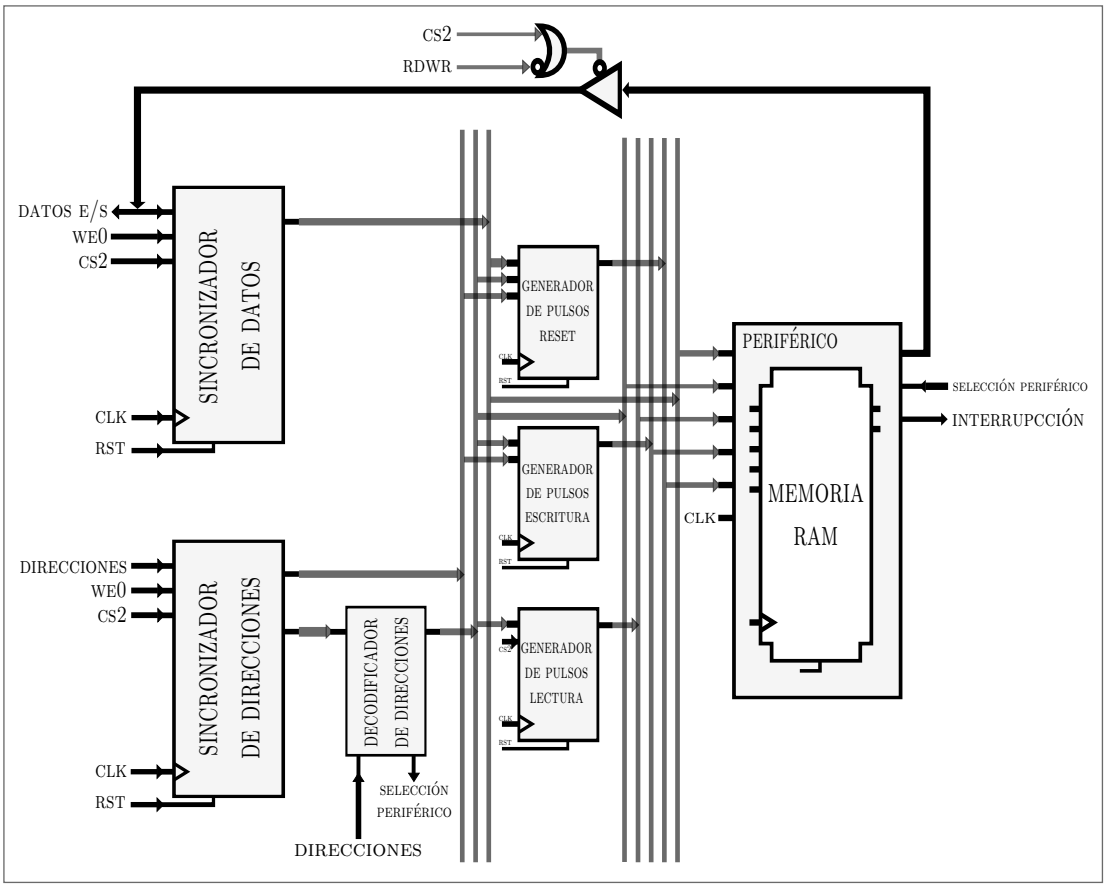


Figura F.6: Interfaz del periférico memoria ram.
FUENTE: El autor.

A continuación se presentan los códigos en lenguaje de descripción de hardware HDL:

F.3.1 Decodificador de direcciones

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity decodificador_direcciones is
7
8    port(
9      entrada_bus_direcciones : in std_logic_vector(12 downto 0);
10     entrada_direcciones_sincronizadas : in std_logic_vector(12 downto 0);
11     salida_direcciones_decodificadas_reset : out std_logic_vector(3 downto 0);
12     salida_direcciones_decodificadas_escritura : out std_logic_vector(2 downto 0);
```

```
13     salida_direcciones_decodificadas_lectura : out std_logic_vector(2 downto 0)
14   );
15
16 end decodificador_direcciones;
17
18 architecture Behavioral of decodificador_direcciones is
19
20 begin
21
22   salida_direcciones_decodificadas_escritura <= "000" when entrada_direcciones_sincronizadas="0000000000001" else
23                                                 "001" when entrada_direcciones_sincronizadas="0000000000010" else
24                                                 "010" when entrada_direcciones_sincronizadas="0000000000011" else
25                                                 "011" when entrada_direcciones_sincronizadas="0000000000100" else
26                                                 "100" when entrada_direcciones_sincronizadas="0000000000101" else
27                                                 "101" when entrada_direcciones_sincronizadas="0000000000110" else
28                                                 "110" when entrada_direcciones_sincronizadas="0000000000111" else
29                                                 "111" when entrada_direcciones_sincronizadas="0000000001000" else
30                                                 (others=>'Z') ;
31
32   salida_direcciones_decodificadas_reset <= "1000" when entrada_direcciones_sincronizadas="0000000001001" else
33                                             (others=>'Z') ;
34
35   salida_direcciones_decodificadas_lectura <= "000" when entrada_bus_direcciones="0000000001010" else
36                                               "001" when entrada_bus_direcciones="0000000001011" else
37                                               "010" when entrada_bus_direcciones="0000000001100" else
38                                               "011" when entrada_bus_direcciones="0000000001101" else
39                                               "100" when entrada_bus_direcciones="0000000001110" else
40                                               "101" when entrada_bus_direcciones="0000000001111" else
41                                               "110" when entrada_bus_direcciones="0000000010000" else
42                                               "111" when entrada_bus_direcciones="0000000010001" else
43                                               (others=>'Z') ;
44
45 end Behavioral;
```

F.3.2 Generador de pulsos de restablecimiento

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_reset is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_q4 : in std_logic;
12     entrada_datos_sincronizados : in std_logic_vector(7 downto 0);
13     entrada_direcciones_reset_decodificadas : in std_logic_vector(3 downto 0);
14     salida_pulso_reset : out std_logic
15   );
16
17 end generador_pulsos_reset;
18
19 architecture Behavioral of generador_pulsos_reset is
```

```
20
21   component antirrebotes
22   port(
23     rst : in std_logic;
24     clk : in std_logic;
25     entrada : in std_logic;
26     salida : out std_logic
27   );
28   end component;
29
30   signal senal_rebote : std_logic;
31
32 begin
33
34   Instancia_antirrebotes: antirrebotes
35   port map(
36     rst => rst,
37     clk => clk,
38     ntrada => entrada_q4,
39     salida => senal_rebote
40   );
41
42   salida_pulso_reset <= '1' when (senal_rebote='1' and
43                                   entrada_datos_sincronizados="00000001" and
44                                   entrada_direcciones_reset_decodificadas="1000") else '0';
45
46 end Behavioral;
```

F.3.3 Generador de pulsos de escritura

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_escritura is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_q4 : in std_logic;
12     entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
13     salida_pulso_escribir : out std_logic
14   );
15
16 end generador_pulsos_escritura;
17
18 architecture Behavioral of generador_pulsos_escritura is
19
20   component antirrebotes
21   port(
22     rst : in std_logic;
23     clk : in std_logic;
24     entrada : in std_logic;
25     salida : out std_logic
```

```
26   );
27   end component;
28
29   signal senal_rebote : std_logic;
30
31 begin
32
33   Instancia_antirrebotes: antirrebotes
34   port map(
35     rst => rst,
36     clk => clk,
37     entrada => entrada_q4,
38     salida => senal_rebote
39   );
40
41   salida_pulso_escribir <= '1' when (senal_rebote='1' and
42                                      (entrada_direcciones_escritura_decodificadas="000" or
43                                       entrada_direcciones_escritura_decodificadas="001" or
44                                       entrada_direcciones_escritura_decodificadas="010" or
45                                       entrada_direcciones_escritura_decodificadas="011" or
46                                       entrada_direcciones_escritura_decodificadas="100" or
47                                       entrada_direcciones_escritura_decodificadas="101" or
48                                       entrada_direcciones_escritura_decodificadas="110" or
49                                       entrada_direcciones_escritura_decodificadas="111" )) else '0';
50
51 end Behavioral;
```

F.3.4 Generador de pulsos de lectura

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_lectura is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_cs2 : in std_logic;
12     entrada_direcciones_lectura_decodificadas: in std_logic_vector(2 downto 0);
13     salida_pulso_leer : out std_logic
14   );
15
16 end generador_pulsos_lectura;
17
18 architecture Behavioral of generador_pulsos_lectura is
19
20 begin
21
22   salida_pulso_leer <= '1' when (entrada_cs2='0' and
23                                  (entrada_direcciones_lectura_decodificadas="000" or
24                                   entrada_direcciones_lectura_decodificadas="001" or
25                                   entrada_direcciones_lectura_decodificadas="010" or
26                                   entrada_direcciones_lectura_decodificadas="011" or
```

```
27                                  entrada_direcciones_lectura_decodificadas="100" or
28                                  entrada_direcciones_lectura_decodificadas="101" or
29                                  entrada_direcciones_lectura_decodificadas="110" or
30                                  entrada_direcciones_lectura_decodificadas="111" )) else '0' ;
31
32 end Behavioral;
```

F.3.5 Memoria RAM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity memoria_ram is
7
8    generic(
9      ancho_dato_ram : integer := 8 ;
10     profundidad_ram : integer := 8;
11     ancho_direcciones_ram : integer := 3
12   );
13
14   port(
15     rst : in std_logic;
16     clk : in std_logic;
17     escribir : in std_logic;
18     leer : in std_logic;
19     direcciones_escritura : in std_logic_vector(ancho_direcciones_ram-1 downto 0);
20     direcciones_lectura : in std_logic_vector(ancho_direcciones_ram-1 downto 0);
21     entrada : in std_logic_vector(ancho_dato_ram-1 downto 0);
22     salida: out std_logic_vector(ancho_dato_ram-1 downto 0)
23   );
24
25 end memoria_ram;
26
27 architecture Behavioral of memoria_ram is
28
29   subtype ancho_dato is std_logic_vector(ancho_dato_ram-1 downto 0);
30   type ram_arreglo is array (0 to profundidad_ram-1) of ancho_dato;
31   signal ram : ram_arreglo;
32
33 begin
34
35   ----------------
36   --MEMORIA RAM--
37   ----------------
38
39   --proceso de escritura--
40   process(rst, clk)
41   begin
42      if (rst='1') then
43         ram <= (others=>(others=>'0'));
44      elsif (clk'event and clk='1') then
45         if (escribir='1') then
46            ram(conv_integer(direcciones_escritura)) <= entrada;
```

```
47          end if;
48       end if;
49    end process;
50
51    --proceso de lectura--
52    process(clk)
53    begin
54       if (clk'event and clk='1') then
55          if (leer='1') then
56             salida <= ram(conv_integer(direcciones_lectura)) ;
57          end if;
58       end if;
59    end process;
60
61 end Behavioral;
```

F.3.6 Interfaz principal

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity interfaz_periferico_ram is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     cs2 : in std_logic;
12     we0 : in std_logic;
13     rdwr : in std_logic;
14     bus_datos : inout std_logic_vector(7 downto 0);
15     bus_direcciones : in std_logic_vector(12 downto 0)
16   );
17
18 end interfaz_periferico_ram;
19
20 architecture Behavioral of interfaz_periferico_ram is
21
22   component sincronizador_reloj_bus_datos
23   port(
24     rst : in std_logic;
25     clk : in std_logic;
26     entrada_we0 : in std_logic;
27     entrada_cs2 : in std_logic;
28     entrada_bus_datos : in std_logic_vector(7 downto 0);
29     salida_sincronizador_datos : out std_logic_vector(7 downto 0)
30   );
31   end component;
32
33   component sincronizador_reloj_bus_direcciones
34   port(
35     rst : in std_logic;
36     clk : in std_logic;
37     entrada_we0 : in std_logic;
```

```
38      entrada_cs2 : in std_logic;
39      entrada_bus_direcciones : in std_logic_vector(12 downto 0);
40      salida_sincronizador_direcciones : out std_logic_vector(12 downto 0);
41      salida_q4 : out std_logic
42    );
43    end component;
44
45    component decodificador_direcciones
46    port(
47      entrada_bus_direcciones : in std_logic_vector(12 downto 0);
48      entrada_direcciones_sincronizadas : in std_logic_vector(12 downto 0);
49      salida_direcciones_decodificadas_reset : out std_logic_vector(3 downto 0);
50      salida_direcciones_decodificadas_escritura : out std_logic_vector(2 downto 0);
51      salida_direcciones_decodificadas_lectura : out std_logic_vector(2 downto 0)
52    );
53    end component;
54
55    component generador_pulsos_reset
56    ort(
57      rst : in std_logic;
58      clk : in std_logic;
59      entrada_q4 : in std_logic;
60      entrada_datos_sincronizados : in std_logic_vector(7 downto 0);
61      entrada_direcciones_reset_decodificadas : in std_logic_vector(3 downto 0);
62      salida_pulso_reset : out std_logic
63    );
64    end component;
65
66    component generador_pulsos_escritura
67    port(
68      rst : in std_logic;
69      clk : in std_logic;
70      entrada_q4 : in std_logic;
71      entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
72      salida_pulso_escribir : out std_logic
73    );
74    end component;
75
76    component generador_pulsos_lectura
77    port(
78      rst : in std_logic;
79      clk : in std_logic;
80      entrada_cs2 : in std_logic;
81      entrada_direcciones_lectura_decodificadas: in std_logic_vector(2 downto 0);
82      salida_pulso_leer : out std_logic
83    );
84    end component;
85
86    component memoria_ram
87    port(
88      rst : in std_logic;
89      clk : in std_logic;
90      escribir : in std_logic;
91      leer : in std_logic;
92      direcciones_escritura : in std_logic_vector(2 downto 0);
93      direcciones_lectura : in std_logic_vector(2 downto 0);
94      entrada : in std_logic_vector(7 downto 0);
```

```
95      salida: out std_logic_vector(7 downto 0)
96    );
97    end component;
98
99    component tercer_estado
100   port(
101     clk : in std_logic;
102     entrada : in std_logic_vector(7 downto 0);
103     entrada_cs2 : in std_logic;
104     entrada_rdwr : in std_logic;
105     salida : out std_logic_vector(7 downto 0)
106   );
107   end component;
108
109   signal senal_1 : std_logic_vector(7 downto 0);
110   signal senal_2 : std_logic_vector(12 downto 0);
111   signal senal_q4 : std_logic;
112   signal senal_3 : std_logic_vector(3 downto 0);
113   signal senal_4 : std_logic_vector(2 downto 0);
114   signal senal_5 : std_logic_vector(2 downto 0);
115   signal senal_6 : std_logic;
116   signal senal_7 : std_logic;
117   signal senal_8 : std_logic;
118   signal senal_9 : std_logic_vector(7 downto 0);
119
120 begin
121
122   instancia_sincronizador_reloj_bus_datos: sincronizador_reloj_bus_datos
123   port map(
124     rst => rst,
125     clk => clk,
126     entrada_we0 => we0,
127     entrada_cs2 => cs2,
128     entrada_bus_datos => bus_datos,
129     salida_sincronizador_datos => senal_1
130   );
131
132   instancia_sincronizador_reloj_bus_direcciones: sincronizador_reloj_bus_direcciones
133   port map(
134     rst => rst,
135     clk => clk,
136     entrada_we0 => we0,
137     entrada_cs2 => cs2,
138     entrada_bus_direcciones => bus_direcciones,
139     salida_sincronizador_direcciones => senal_2,
140     salida_q4 => senal_q4
141   );
142
143   instancia_decodificador_direcciones: decodificador_direcciones
144   port map(
145     entrada_bus_direcciones => bus_direcciones,
146     entrada_direcciones_sincronizadas => senal_2,
147     salida_direcciones_decodificadas_reset => senal_3,
148     salida_direcciones_decodificadas_escritura => senal_4,
149     salida_direcciones_decodificadas_lectura => senal_5
150   );
151
```

```
152   instancia_generador_pulsos_reset: generador_pulsos_reset
153   port map(
154     rst => rst,
155     clk => clk,
156     entrada_q4 => senal_q4,
157     entrada_datos_sincronizados => senal_1,
158     entrada_direcciones_reset_decodificadas => senal_3,
159     salida_pulso_reset => senal_6
160   );
161
162   instancia_generador_pulsos_escritura: generador_pulsos_escritura
163   port map(
164     rst => rst,
165     clk => clk,
166     entrada_q4 => senal_q4,
167     entrada_direcciones_escritura_decodificadas => senal_4,
168     salida_pulso_escribir => senal_7
169   );
170
171   instancia_generador_pulsos_lectura: generador_pulsos_lectura
172   port map(
173     rst => rst,
174     clk => clk,
175     entrada_cs2 => cs2,
176     entrada_direcciones_lectura_decodificadas => senal_5,
177     salida_pulso_leer => senal_8
178   );
179
180   instancia_memoria_ram: memoria_ram
181   port map(
182     rst => senal_6,
183     clk => clk,
184     escribir => senal_7,
185     leer => senal_8,
186     direcciones_escritura => senal_4,
187     direcciones_lectura => senal_5,
188     entrada => senal_1,
189     salida => senal_9
190   );
191
192   instancia_tercer_estado: tercer_estado
193   port map(
194     clk => clk,
195     entrada => senal_9,
196     entrada_cs2 => cs2,
197     entrada_rdwr => rdwr,
198     salida => bus_datos
199   );
200
201 end Behavioral;
```

F.4 Periférico memoria FIFO

En la Figura F.7 se encuentra el diagrama de bloques de la interfaz diseñada para este periférico.

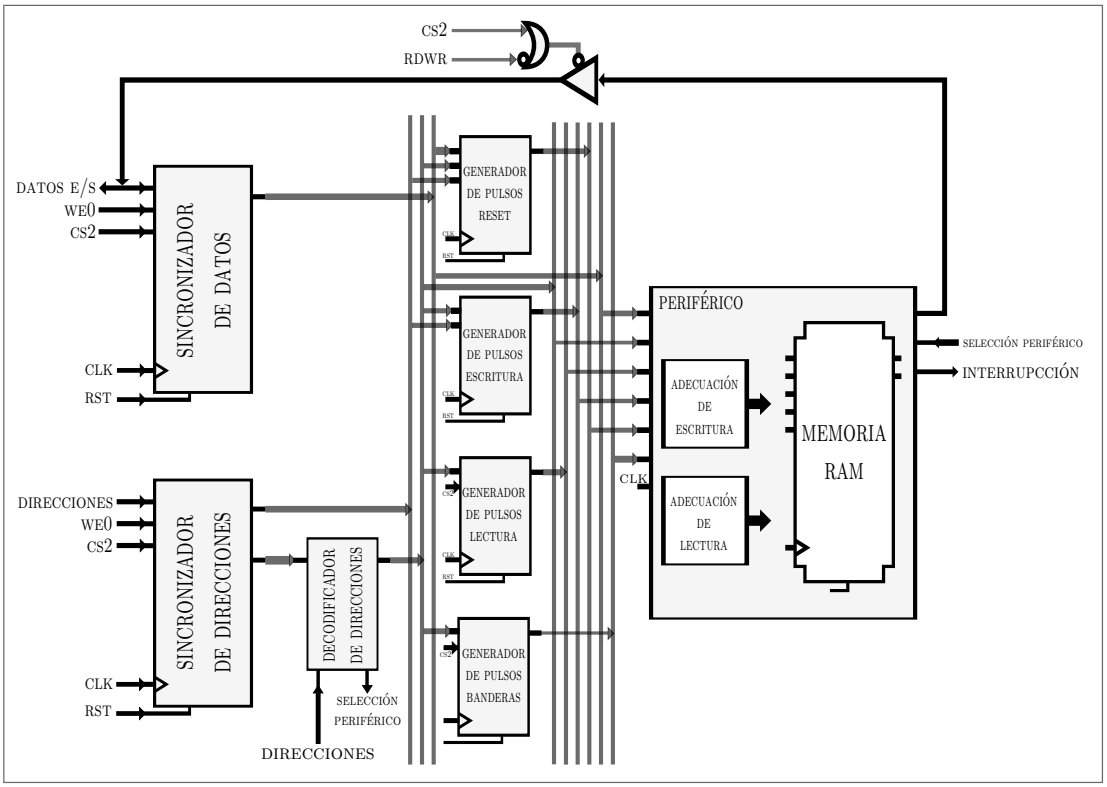


Figura F.7: Interfaz del periférico memoria fifo.
FUENTE: El autor.

A continuación se presentan los códigos en lenguaje de descripción de hardware HDL:

F.4.1 Decodificador de direcciones

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity decodificador_direcciones is
7
8    port(
9      entrada_bus_direcciones : in std_logic_vector(12 downto 0);
10     entrada_direcciones_sincronizadas : in std_logic_vector(12 downto 0);
11     salida_direcciones_decodificadas_escritura : out std_logic_vector(2 downto 0);
12     salida_direcciones_decodificadas_lectura : out std_logic_vector(2 downto 0)
13   );
14
15 end decodificador_direcciones;
16
```

```
17 architecture Behavioral of decodificador_direcciones is
18
19 begin
20
21   salida_direcciones_decodificadas_lectura <= "000" when entrada_bus_direcciones="0000000000000" else
22                                               "001" when entrada_bus_direcciones="0000000000001" else
23                                               "010" when entrada_bus_direcciones="0000000000010" else
24                                               "011" when entrada_bus_direcciones="0000000000011" else
25                                               "100" when entrada_bus_direcciones="0000000000100" else
26                                               (others=>'Z') ;
27
28   salida_direcciones_decodificadas_escritura <= "000" when entrada_direcciones_sincronizadas="0000000000000" else
29                                                 "001" when entrada_direcciones_sincronizadas="0000000000001" else
30                                                 "010" when entrada_direcciones_sincronizadas="0000000000010" else
31                                                 "011" when entrada_direcciones_sincronizadas="0000000000011" else
32                                                 "100" when entrada_direcciones_sincronizadas="0000000000100" else
33                                                 (others=>'Z') ;
34
35 end Behavioral;
```

F.4.2 Generador de pulsos de restablecimiento

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_reset is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_q4 : in std_logic;
12     entrada_datos_sincronizados : in std_logic_vector(7 downto 0);
13     entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
14     salida_pulso_reset : out std_logic
15   );
16
17 end generador_pulsos_reset;
18
19 architecture Behavioral of generador_pulsos_reset is
20
21   component antirrebotes
22   port(
23     rst : in std_logic;
24     clk : in std_logic;
25     entrada : in std_logic;
26     salida : out std_logic
27   );
28   end component;
29
30   signal senal_rebote : std_logic;
31
32 begin
33
```

```
34   Instancia_antirrebotes: antirrebotes
35   port map(
36     rst => rst,
37     clk => clk,
38     entrada => entrada_q4,
39     salida => senal_rebote
40   );
41
42   salida_pulso_reset <= '1' when (senal_rebote='1' and
43                                   entrada_datos_sincronizados="00000001" and
44                                   entrada_direcciones_escritura_decodificadas="000") else '0';
45
46 end Behavioral;
```

F.4.3 Generador de pulsos de escritura

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_escritura is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_q4 : in std_logic;
12     entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
13     salida_pulso_escribir : out std_logic
14   );
15
16 end generador_pulsos_escritura;
17
18 architecture Behavioral of generador_pulsos_escritura is
19
20   component antirrebotes
21   port(
22     rst : in std_logic;
23     clk : in std_logic;
24     entrada : in std_logic;
25     salida : out std_logic
26   );
27   end component;
28
29   signal senal_rebote : std_logic;
30
31 begin
32
33   Instancia_antirrebotes: antirrebotes
34   port map(
35     rst => rst,
36     clk => clk,
37     entrada => entrada_q4,
38     salida => senal_rebote
39   );
```

```
40
41   salida_pulso_escribir <= '1' when (senal_rebote='1' and
42                                      entrada_direcciones_escritura_decodificadas="001") else '0';
43
44 end Behavioral;
```

F.4.4 Generador de pulsos de lectura

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_lectura is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_cs2 : in std_logic;
12     entrada_direcciones_lectura_decodificadas: in std_logic_vector(2 downto 0);
13     salida_pulso_leer : out std_logic
14   );
15
16 end generador_pulsos_lectura;
17
18 architecture Behavioral of generador_pulsos_lectura is
19
20   signal senal_salida : std_logic;
21
22   component antirrebotes
23   port(
24     rst : in std_logic;
25     clk : in std_logic;
26     entrada : in std_logic;
27     salida : out std_logic
28   );
29   end component;
30
31 begin
32
33   senal_salida <= '1' when (entrada_cs2='0' and
34                             entrada_direcciones_lectura_decodificadas="010") else '0';
35
36   Instancia_antirrebotes: antirrebotes
37   port map(
38     rst => rst,
39     clk => clk,
40     entrada => senal_salida,
41     salida => salida_pulso_leer
42   );
43
44 end Behavioral;
```

F.4.5 Generador de pulsos para las banderas

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity generador_pulsos_banderas is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     entrada_cs2 : in std_logic;
12     entrada_direcciones_lectura_decodificadas : in std_logic_vector(2 downto 0);
13     salida_pulso_bandera_lleno : out std_logic;
14     salida_pulso_bandera_vacio : out std_logic
15   );
16
17 end generador_pulsos_banderas;
18
19 architecture Behavioral of generador_pulsos_banderas is
20
21 begin
22
23   salida_pulso_bandera_lleno <= '1' when (entrada_cs2='0' and
24                                       entrada_direcciones_lectura_decodificadas="011") else '0' ;
25
26   salida_pulso_bandera_vacio <= '1' when (entrada_cs2='0' and
27                                       entrada_direcciones_lectura_decodificadas="100") else '0' ;
28
29 end Behavioral;
```

F.4.6 Memoria FIFO

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity memoria_fifo is
7
8    generic(
9      ancho_dato_fifo : integer := 8 ;
10     profundidad_fifo : integer := 8 ;
11     ancho_puntero_fifo : integer := 3
12   );
13
14   port(
15     rst : in std_logic;
16     clk : in std_logic;
17     escribir : in std_logic;
18     leer : in std_logic;
19     entrada : in std_logic_vector(ancho_dato_fifo-1 downto 0);
20     salida : out std_logic_vector(ancho_dato_fifo-1 downto 0);
21     bandera_lleno : out std_logic;
```

```
22      bandera_vacio : out std_logic
23    );
24
25 end memoria_fifo;
26
27 architecture Behavioral of memoria_fifo is
28
29    signal envolvente, De, Qe : std_logic_vector(ancho_puntero_fifo downto 0);
30    signal puntero_escritura, Dpw, Qpw : std_logic_vector(ancho_puntero_fifo-1 downto 0);
31    signal puntero_lectura, Dpr, Qpr : std_logic_vector(ancho_puntero_fifo-1 downto 0);
32
33    subtype ancho_dato_ram is std_logic_vector(ancho_dato_fifo-1 downto 0);
34    type ram_type is array ( 0 to profundidad_fifo-1) of ancho_dato_ram;
35    signal ram : ram_type;
36
37    signal senal_lleno : std_logic;
38    signal senal_vacio : std_logic;
39
40 begin
41
42    ----------------
43    --ENVOLVENTE--
44    ----------------
45    process(rst, clk)
46    begin
47       if (rst='1') then
48          Qe <= (others=>'0');
49       elsif (clk'event and clk='1') then
50          Qe <= De;
51       end if;
52    end process;
53
54    De <= (Qe + 1) when (escribir='1' and leer='0' and senal_lleno='0') else
55          (Qe - 1) when (escribir='0' and leer='1' and senal_vacio='0') else
56          Qe ;
57    envolvente <= Qe;
58
59    ------------------------
60    --PUNTERO DE ESCRITURA--
61    ------------------------
62    process(rst, clk)
63    begin
64       if (rst='1') then
65          Qpw <= (others=>'0');
66       elsif (clk'event and clk='1') then
67          Qpw <= Dpw;
68       end if;
69    end process;
70
71    Dpw <= (Qpw + 1) when (escribir='1' and senal_lleno='0') else Qpw ;
72    puntero_escritura <= Qpw;
73
74    ------------------------
75    --PUNTTERO DE LECTURA--
76    ------------------------
77    process(rst, clk)
78    begin
```

```
79      if (rst='1') then
80         Qpr <= (others=>'0');
81      elsif (clk'event and clk='1') then
82         Qpr <= Dpr;
83   end if;
84   end process;
85
86   Dpr <= (Qpr + 1) when (leer='1' and senal_vacio='0') else Qpr ;
87   puntero_lectura <= Qpr;
88
89   ----------------
90   --MEMORIA RAM--
91   ----------------
92   --proceso de escritura--
93   process(rst, clk)
94   begin
95      if (rst='1') then
96         ram <= (others=>(others=>'0'));
97      elsif (clk'event and clk='1') then
98         if (escribir='1') then
99           if (senal_lleno='0') then
100             ram(conv_integer(puntero_escritura)) <= entrada ;
101          end if;
102        end if;
103     end if;
104  end process;
105
106  --proceso de lectura--
107  process(clk)
108  begin
109     if (clk'event and clk='0') then
110        if (leer='1') then
111           if (senal_vacio='0') then
112              salida <= ram(conv_integer(puntero_lectura)) ;
113           end if;
114        end if;
115     end if;
116  end process;
117
118  ----------------
119  --BANDERAS--
120  ----------------
121  senal_lleno <= '1' when (envolvente=profundidad_fifo) else '0' ;
122  senal_vacio <= '1' when (envolvente=0) else '0' ;
123
124  bandera_lleno <= senal_lleno;
125  bandera_vacio <= senal_vacio;
126
127 end Behavioral;
```

F.4.7 Interfaz principal

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
```

```
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity interfaz_periferico_fifo is
7
8    port(
9      rst : in std_logic;
10     clk : in std_logic;
11     cs2 : in std_logic;
12     we0 : in std_logic;
13     rdwr : in std_logic;
14     bus_datos : inout std_logic_vector(7 downto 0);
15     bus_direcciones : in std_logic_vector(12 downto 0)
16   );
17
18 end interfaz_periferico_fifo;
19
20 architecture Behavioral of interfaz_periferico_fifo is
21
22   component sincronizador_reloj_bus_datos
23   port(
24     rst : in std_logic;
25     clk : in std_logic;
26     entrada_we0 : in std_logic;
27     entrada_cs2 : in std_logic;
28     entrada_bus_datos : in std_logic_vector(7 downto 0);
29     salida_sincronizador_datos : out std_logic_vector(7 downto 0)
30   );
31   end component;
32
33   component sincronizador_reloj_bus_direcciones
34   port(
35     rst : in std_logic;
36     clk : in std_logic;
37     entrada_we0 : in std_logic;
38     entrada_cs2 : in std_logic;
39     entrada_bus_direcciones : in std_logic_vector(12 downto 0);
40     salida_sincronizador_direcciones : out std_logic_vector(12 downto 0);
41     salida_q4 : out std_logic
42   );
43   end component;
44
45   component decodificador_direcciones
46   port(
47     entrada_bus_direcciones : in std_logic_vector(12 downto 0);
48     entrada_direcciones_sincronizadas : in std_logic_vector(12 downto 0);
49     salida_direcciones_decodificadas_escritura : out std_logic_vector(2 downto 0);
50     salida_direcciones_decodificadas_lectura : out std_logic_vector(2 downto 0)
51   );
52   end component;
53
54   component generador_pulsos_reset
55   port(
56     rst : in std_logic;
57     clk : in std_logic;
58     entrada_q4 : in std_logic;
59     entrada_datos_sincronizados : in std_logic_vector(7 downto 0);
60     entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
```

```
61      salida_pulso_reset : out std_logic
62    );
63    end component;
64
65    component generador_pulsos_escritura
66    port(
67      rst : in std_logic;
68      clk : in std_logic;
69      entrada_q4 : in std_logic;
70      entrada_direcciones_escritura_decodificadas : in std_logic_vector(2 downto 0);
71      salida_pulso_escribir : out std_logic
72    );
73    end component;
74
75    component generador_pulsos_lectura
76    port(
77      rst : in std_logic;
78      clk : in std_logic;
79      entrada_cs2 : in std_logic;
80      entrada_direcciones_lectura_decodificadas: in std_logic_vector(2 downto 0);
81      salida_pulso_leer : out std_logic
82    );
83    end component;
84
85    component generador_pulsos_banderas
86    port(
87      rst : in std_logic;
88      clk : in std_logic;
89      entrada_cs2 : in std_logic;
90      entrada_direcciones_lectura_decodificadas : in std_logic_vector(2 downto 0);
91      salida_pulso_bandera_lleno : out std_logic;
92      salida_pulso_bandera_vacio : out std_logic
93    );
94    end component;
95
96    component memoria_fifo
97    port(
98      rst : in std_logic;
99      clk : in std_logic;
100     escribir : in std_logic;
101     leer : in std_logic;
102     entrada : in std_logic_vector(7 downto 0);
103     salida : out std_logic_vector(7 downto 0);
104     bandera_lleno : out std_logic;
105     bandera_vacio : out std_logic
106   );
107   end component;
108
109   component tercer_estado
110   port(
111     entrada : in std_logic_vector(7 downto 0);
112     entrada_cs2 : in std_logic;
113     entrada_rdwr : in std_logic;
114     salida : out std_logic_vector(7 downto 0)
115   );
116   end component;
117
```

```
118   signal senal_1 : std_logic_vector(7 downto 0);
119   signal senal_2 : std_logic_vector(12 downto 0);
120   signal senal_q4 : std_logic;
121   signal senal_3 : std_logic_vector(2 downto  0);
122   signal senal_4 : std_logic_vector(2 downto  0);
123   signal senal_5 : std_logic;
124   signal senal_6 : std_logic;
125   signal senal_7 : std_logic;
126   signal senal_8 : std_logic;
127   signal senal_9 : std_logic;
128   signal senal_10 : std_logic;
129   signal senal_11 : std_logic;
130   signal senal_12 : std_logic_vector(7 downto 0);
131   signal senal_13 : std_logic_vector(7 downto 0);
132
133 begin
134
135   instancia_sincronizador_reloj_bus_datos: sincronizador_reloj_bus_datos
136   port map(
137     rst => rst,
138     clk => clk,
139     entrada_we0 => we0,
140     entrada_cs2 => cs2,
141     entrada_bus_datos => bus_datos,
142     salida_sincronizador_datos => senal_1
143   );
144
145   instancia_sincronizador_reloj_bus_direcciones: sincronizador_reloj_bus_direcciones
146   port map(
147     rst => rst,
148     clk => clk,
149      entrada_we0 => we0,
150     entrada_cs2 => cs2,
151     entrada_bus_direcciones => bus_direcciones,
152     salida_sincronizador_direcciones => senal_2,
153     salida_q4 => senal_q4
154   );
155
156   instancia_decodificador_direcciones: decodificador_direcciones
157   port map(
158     entrada_bus_direcciones => bus_direcciones,
159     entrada_direcciones_sincronizadas => senal_2,
160     salida_direcciones_decodificadas_escritura => senal_3,
161     salida_direcciones_decodificadas_lectura => senal_4
162   );
163
164   instancia_generador_pulsos_reset: generador_pulsos_reset
165   port map(
166     rst => rst,
167     clk => clk,
168     entrada_q4 => senal_q4,
169     entrada_datos_sincronizados => senal_1,
170     entrada_direcciones_escritura_decodificadas => senal_3,
171     salida_pulso_reset => senal_5
172   );
173
174   instancia_generador_pulsos_escritura: generador_pulsos_escritura
```

```
175   port map(
176     rst => rst,
177     clk => clk,
178     entrada_q4 => senal_q4,
179     entrada_direcciones_escritura_decodificadas => senal_3,
180     salida_pulso_escribir => senal_6
181   );
182
183   instancia_generador_pulsos_lectura: generador_pulsos_lectura
184   port map(
185     rst => rst,
186     clk => clk,
187     entrada_cs2 => cs2,
188     entrada_direcciones_lectura_decodificadas => senal_4,
189     salida_pulso_leer => senal_7
190   );
191
192   instancia_generador_pulsos_banderas: generador_pulsos_banderas
193   port map(
194     rst => rst,
195     clk => clk,
196     entrada_cs2 => cs2,
197     entrada_direcciones_lectura_decodificadas => senal_4,
198     salida_pulso_bandera_lleno => senal_8,
199     salida_pulso_bandera_vacio => senal_9
200   );
201
202   instancia_memoria_fifo: memoria_fifo
203   port map(
204     rst => senal_5,
205     clk => clk,
206     escribir => senal_6,
207     leer => senal_7,
208     entrada => senal_1,
209     salida => senal_12,
210     bandera_lleno => senal_10,
211     bandera_vacio => senal_11
212   );
213
214   senal_13 <= "0000000" & senal_10 when senal_8='1' else
215               "0000000" & senal_11 when senal_9='1' else
216               senal_12 ;
217
218   instancia_tercer_estado: tercer_estado
219   port map(
220     entrada => senal_13,
221     entrada_cs2 => cs2,
222     entrada_rdwr => rdwr,
223     salida => bus_datos
224   );
225
226 end Behavioral;
```

F.5 Asignación de puertos

En el siguiente cuadro de código se encuentra la fuente **.ucf** que corresponde a la declaración de puertos para cada interfaz de los periféricos. Esta fuente es igual en todos ya que la entidad principal tiene las mismas entradas y salidas.

F.5.1 Interfaz_principal.ucf

```
net "rst" loc="p30"; # Pulsador PB3
net "clk" loc="p38"; # Reloj 50 [MHz]

net "cs2" loc="p69";
net "we0" loc="p88";
net "rdwr" loc="p86";

net "bus_datos<7>" loc="p4";
net "bus_datos<6>" loc="p5";
net "bus_datos<5>" loc="p9";
net "bus_datos<4>" loc="p10";
net "bus_datos<3>" loc="p11";
net "bus_datos<2>" loc="p12";
net "bus_datos<1>" loc="p15";
net "bus_datos<0>" loc="p16";

net "bus_direcciones<12>" loc = "p90";
net "bus_direcciones<11>" loc = "p91";
net "bus_direcciones<10>" loc = "p85";
net "bus_direcciones<9>"  loc = "p92";
net "bus_direcciones<8>"  loc = "p94";
net "bus_direcciones<7>"  loc = "p95";
net "bus_direcciones<6>"  loc = "p98";
net "bus_direcciones<5>"  loc = "p3";
net "bus_direcciones<4>"  loc = "p2";
net "bus_direcciones<3>"  loc = "p78";
net "bus_direcciones<2>"  loc = "p79";
net "bus_direcciones<1>"  loc = "p83";
net "bus_direcciones<0>"  loc = "p84";
```

ANEXO G. Aplicaciones en el espacio del usuario

En este Anexo se presenta el código fuente de las plantillas obtenidas durante la tercera etapa del desarrollo de aplicaciones desde el espacio del usuario. Antes de hacer uso de las plantillas es necesario añadir el módulo (driver basado en registros) al *kernel* de Linux y programar el FPGA con el interfaz del periférico a usar.

G.1 Plantillas para gestionar el periférico de registros

G.1.1 aplicacion.h

```
1  /**********************/
2  /***DESPLAZAMIENTOS***/
3  /**********************/
4
5  /* Desplazamiento para restablecer el periferico de registros *
6   * desde el espacio del usuario.                               */
7  #define DESPLAZAMIENTO_RESTABLECER_REG (0b00000000001001)
8
9  /* Desplazamiento para leer el periferico de registros desde  *
10  * el espacio del usuario.                                     */
11 #define DESPLAZAMIENTO_LECTURA_REG0 (0b00000000001010)
12 #define DESPLAZAMIENTO_LECTURA_REG1 (0b00000000001011)
13 #define DESPLAZAMIENTO_LECTURA_REG2 (0b00000000001100)
14 #define DESPLAZAMIENTO_LECTURA_REG3 (0b00000000001101)
15 #define DESPLAZAMIENTO_LECTURA_REG4 (0b00000000001110)
16 #define DESPLAZAMIENTO_LECTURA_REG5 (0b00000000001111)
17 #define DESPLAZAMIENTO_LECTURA_REG6 (0b00000000010000)
18 #define DESPLAZAMIENTO_LECTURA_REG7 (0b00000000010001)
19
20 /*Desplazamiento para la escribir el periferico de registros  *
21  * desde el espacio del usuario.                               */
22 #define DESPLAZAMIENTO_ESCRITURA_REG0 (0b0000000000001)
23 #define DESPLAZAMIENTO_ESCRITURA_REG1 (0b0000000000010)
24 #define DESPLAZAMIENTO_ESCRITURA_REG2 (0b0000000000011)
25 #define DESPLAZAMIENTO_ESCRITURA_REG3 (0b0000000000100)
26 #define DESPLAZAMIENTO_ESCRITURA_REG4 (0b0000000000101)
27 #define DESPLAZAMIENTO_ESCRITURA_REG5 (0b0000000000110)
```

```
28 #define DESPLAZAMIENTO_ESCRITURA_REG6 (0b0000000000111)
29 #define DESPLAZAMIENTO_ESCRITURA_REG7 (0b0000000001000)
30
31 /*************/
32 /***TAMANOS***/
33 /*************/
34
35 /*Tamano es la cantidad de bytes a leer/escribir.             */
36 #define TAMANO_BYTES_CHAR    (1)
37 #define TAMANO_BYTES_INT     (4)
38 #define TAMANO_BYTES_DOUBLE (8)
39
40 /*****************************/
41 /***DECLARACION DE FUNCIONES***/
42 /*****************************/
43
44 /*Funcion PERIFERICO_REGISTROS_restablecer
45  * dato : Es el valor necesario para restablecer el periferico.
46  * tamano : Es la cantidad de bytes que se tomaran del dato.
47  * desplazamiento : Es la direccion por donde debe viajar el dato de reset.
48  */
49 int PERIFERICO_REGISTROS_restablecer(unsigned char dato, char tamano, int desplazamiento);
50
51 /*Funcion PERIFERICO_REGISTERS_read
52  * tamano : Es la cantidad de bytes que se tomaran del dato.
53  * desplazamiento : Es la direccion por donde debe llegar el dato a leer.
54  */
55 unsigned char PERIFERICO_REGISTROS_leer(char tamano, int desplazamiento);
56
57 /*Funcion PERIFERICO_REGISTROS_escribir
58  * dato : Es el valor a escribir en el periferico.
59  * tamano : Es la cantidad de bytes que se tomaran del dato.
60  * desplazamiento : Es la direccion por donde debe viajar el dato a escribir.
61  */
62 int PERIFERICO_REGISTROS_escribir(unsigned char dato, char tamano, int desplazamiento);
```

G.1.2 aplicacion.c

```
1 #include <fcntl.h>
2 #include "aplicacion.h"
3
4 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, write y close. */
5 int PERIFERICO_REGISTROS_restablecer(unsigned char dato, char tamano, int desplazamiento)
6 {
7   int file_descriptor;
8
9   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
10
11  lseek(file_descriptor, desplazamiento, SEEK_SET);
12
13  write(file_descriptor, &dato, tamano);
14
15  close(file_descriptor);
16
17  return 1;
```

```
18 }
19
20 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, read y close. */
21 unsigned char PERIFERICO_REGISTROS_leer(char tamano, int desplazamiento)
22 {
23   int file_descriptor;
24
25   unsigned char buffer[0];
26
27   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
28
29   lseek(file_descriptor, desplazamiento, SEEK_SET);
30
31   read(file_descriptor, buffer, tamano);
32
33   close(file_descriptor);
34
35   return buffer[0];
36 }
37
38 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, write y close. */
39 int PERIFERICO_REGISTROS_escribir(unsigned char dato, char tamano, int desplazamiento)
40 {
41   int file_descriptor;
42
43   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
44
45   lseek(file_descriptor, desplazamiento, SEEK_SET);
46
47   write(file_descriptor, &dato, tamano);
48
49   close(file_descriptor);
50
51   return 1;
52 }
```

G.1.3 prueba.c

```
1 #include <stdio.h>
2 #include "aplicacion.h"
3 #include "aplicacion.c"
4
5 int main()
6 {
7   int opcion;
8   unsigned char datoR = 0b00000001;
9   unsigned char datoW1 = 0b11111111;
10  unsigned char datoW2 = 0b11111110;
11  unsigned char datoW3 = 0b11111100;
12  unsigned char datoW4 = 0b11111000;
13  unsigned char datoW5 = 0b11110000;
14  unsigned char datoW6 = 0b11100000;
15  unsigned char datoW7 = 0b11000000;
16  unsigned char datoW8 = 0b10000000;
17  unsigned char leido[7];
```

```
18   int i;
19
20   printf("APLICACION: SELECCIONAR LA OPERACION A RELIZAR. \n");
21   printf("APLICACION: 0. Reestablecer. \n");
22   printf("APLICACION: 1. Escribir. \n");
23   printf("APLICACION: 2. Leer. \n");
24   scanf("%i", &opcion);
25
26   if(opcion == 0)
27   {
28     PERIFERICO_REGISTROS_restablecer(datoR, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_RESTABLECER_REG);
29   }
30
31   if(opcion == 1)
32   {
33     PERIFERICO_REGISTROS_escribir(datoW1, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG0);
34     PERIFERICO_REGISTROS_escribir(datoW2, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG1);
35     PERIFERICO_REGISTROS_escribir(datoW3, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG2);
36     PERIFERICO_REGISTROS_escribir(datoW4, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG3);
37     PERIFERICO_REGISTROS_escribir(datoW5, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG4);
38     PERIFERICO_REGISTROS_escribir(datoW6, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG5);
39     PERIFERICO_REGISTROS_escribir(datoW7, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG6);
40     PERIFERICO_REGISTROS_escribir(datoW8, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_REG7);
41   }
42
43   if(opcion == 2)
44   {
45     for(i=0; i<8; i++)
46     {
47       leido[i] = PERIFERICO_REGISTROS_leer(TAMANO_BYTES_CHAR, DESPLAZAMIENTO_LECTURA_REG0 + i);
48     }
49     for(i=0; i<8; i++)
50     {
51       printf("APLICACION: En la direccion: 0x%x. El dato es: 0x%x. \n", i, leido[i]);
52     }
53   }
54
55   if(opcion > 2)
56   {
57     printf("APLICACION: Esta no es una opccion. \n");
58     printf("APLICACION: Vuelva a ejecutar la aplicacion. \n\r");
59   }
60
61   return 0;
62 }
```

G.2 Plantillas para gestionar el periférico memoria RAM

G.2.1 aplicacion.h

```
1  /*********************/
2  /***DESPLAZAMIENTOS***/
3  /*********************/
4
5  /* Desplazamiento para restablecer el periferico meoria ram   *
6   * desde el espacio del usuario.                              */
7  #define DESPLAZAMIENTO_RESTABLECER_BRAM (0b0000000001001)
8
9  /* Desplazamiento para leer el periferico meoria ram desde    *
10  * el espacio del usuario.                                    */
11 #define DESPLAZAMIENTO_LECTURA_BRAM0 (0b0000000001010)
12 #define DESPLAZAMIENTO_LECTURA_BRAM1 (0b0000000001011)
13 #define DESPLAZAMIENTO_LECTURA_BRAM2 (0b0000000001100)
14 #define DESPLAZAMIENTO_LECTURA_BRAM3 (0b0000000001101)
15 #define DESPLAZAMIENTO_LECTURA_BRAM4 (0b0000000001110)
16 #define DESPLAZAMIENTO_LECTURA_BRAM5 (0b0000000001111)
17 #define DESPLAZAMIENTO_LECTURA_BRAM6 (0b0000000010000)
18 #define DESPLAZAMIENTO_LECTURA_BRAM7 (0b0000000010001)
19
20 /*Desplazamiento para la escribir el periferico meoria ram   *
21  * desde el espacio del usuario.                              */
22 #define DESPLAZAMIENTO_ESCRITURA_BRAM0 (0b0000000000001)
23 #define DESPLAZAMIENTO_ESCRITURA_BRAM1 (0b0000000000010)
24 #define DESPLAZAMIENTO_ESCRITURA_BRAM2 (0b0000000000011)
25 #define DESPLAZAMIENTO_ESCRITURA_BRAM3 (0b0000000000100)
26 #define DESPLAZAMIENTO_ESCRITURA_BRAM4 (0b0000000000101)
27 #define DESPLAZAMIENTO_ESCRITURA_BRAM5 (0b0000000000110)
28 #define DESPLAZAMIENTO_ESCRITURA_BRAM6 (0b0000000000111)
29 #define DESPLAZAMIENTO_ESCRITURA_BRAM7 (0b0000000001000)
30
31 /*************/
32 /***TAMANOS***/
33 /*************/
34
35 /*Tamano es la cantidad de bytes a leer/escribir.            */
36 #define TAMANO_BYTES_CHAR    (1)
37 #define TAMANO_BYTES_INT     (4)
38 #define TAMANO_BYTES_DOUBLE  (8)
39
40 /*******************************/
41 /***DECLARACION DE FUNCIONES***/
42 /*******************************/
43
44 /*Funcion PERIFERICO_BRAM_restablecer
45  * dato : Es el valor necesario para restablecer el periferico.
46  * tamano : Es la cantidad de bytes que se tomaran del dato.
47  * desplazamiento : Es la direccion por donde debe viajar el dato de reset.
48  */
49 int PERIFERICO_BRAM_restablecer(unsigned char dato, char tamano, int desplazamiento);
50
51 /*Funcion PERIFERICO_BRAM_read
52  * tamano : Es la cantidad de bytes que se tomaran del dato.
```

```
53  * desplazamiento : Es la direccion por donde debe llegar el dato a leer.
54  */
55 unsigned char PERIFERICO_BRAM_leer(char tamano, int desplazamiento);
56
57 /*Funcion PERIFERICO_BRAM_escribir
58  * dato : Es el valor a escribir en el periferico.
59  * tamano : Es la cantidad de bytes que se tomaran del dato.
60  * desplazamiento : Es la direccion por donde debe viajar el dato a escribir.
61  */
62 int PERIFERICO_BRAM_escribir(unsigned char dato, char tamano, int desplazamiento);
```

G.2.2 aplicacion.c

```
1  #include <fcntl.h>
2  #include "aplicacion .h"
3
4  /* En esta rutina se ejecutan las llamadas al sistema open, llsek, write y close. */
5  int PERIFERICO_BRAM_restablecer(unsigned char dato, char tamano, int desplazamiento)
6  {
7    int file_descriptor;
8
9    file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
10
11   lseek(file_descriptor, desplazamiento, SEEK_SET);
12
13   write(file_descriptor, &dato, tamano);
14
15   close(file_descriptor);
16
17   return 1;
18 }
19
20 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, read y close. */
21 unsigned char PERIFERICO_BRAM_leer(char tamano, int desplazamiento)
22 {
23   int file_descriptor;
24
25   unsigned char buffer[0];
26
27   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
28
29   lseek(file_descriptor, desplazamiento, SEEK_SET);
30
31   read(file_descriptor, buffer, tamano);
32
33   close(file_descriptor);
34
35   return buffer[0];
36 }
37
38 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, read y close. */
39 int PERIFERICO_BRAM_escribir(unsigned char dato, char tamano, int desplazamiento)
40 {
41   int file_descriptor;
42
```

```
43    file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
44
45    lseek(file_descriptor, desplazamiento, SEEK_SET);
46
47    write(file_descriptor, &dato, tamano);
48
49    close(file_descriptor);
50
51    return 1;
52 }
```

G.2.3 prueba.c

```
1  #include <stdio.h>
2  #include "aplicacion.h"
3  #include "aplicacion.c"
4
5  int main()
6  {
7    int opcion;
8    unsigned char datoR = 0b00000001;
9    unsigned char datoW1 = 0b11111111;
10   unsigned char datoW2 = 0b11111110;
11   unsigned char datoW3 = 0b11111100;
12   unsigned char datoW4 = 0b11111000;
13   unsigned char datoW5 = 0b11110000;
14   unsigned char datoW6 = 0b11100000;
15   unsigned char datoW7 = 0b11000000;
16   unsigned char datoW8 = 0b10000000;
17   unsigned char leido[7];
18   int i;
19
20   printf("APLICACION: SELECCIONAR LA OPERACION A RELIZAR. \n");
21   printf("APLICACION: 0. Reestablecer. \n");
22   printf("APLICACION: 1. Escribir. \n");
23   printf("APLICACION: 2. Leer. \n");
24   scanf("%d", &opcion);
25
26   if(opcion == 0)
27   {
28     PERIFERICO_BRAM_restablecer(datoR, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_RESTABLECER_BRAM);
29   }
30
31   if(opcion == 1)
32   {
33     PERIFERICO_BRAM_escribir(datoW1, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM0);
34     PERIFERICO_BRAM_escribir(datoW2, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM1);
35     PERIFERICO_BRAM_escribir(datoW3, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM2);
36     PERIFERICO_BRAM_escribir(datoW4, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM3);
37     PERIFERICO_BRAM_escribir(datoW5, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM4);
38     PERIFERICO_BRAM_escribir(datoW6, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM5);
39     PERIFERICO_BRAM_escribir(datoW7, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM6);
40     PERIFERICO_BRAM_escribir(datoW8, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_BRAM7);
41   }
42
```

```
43   if(opcion == 2)
44   {
45     for(i=0; i<8; i++)
46     {
47       leido[i] = PERIFERICO_BRAM_leer(TAMANO_BYTES_CHAR, DESPLAZAMIENTO_LECTURA_BRAM0 + i);
48     }
49     for(i=0; i<8; i++)
50     {
51       printf("APLICACION: En la direccion: 0x%x. El dato es: 0x%x. \n", i, leido[i]);
52     }
53   }
54
55   if(opcion > 2)
56   {
57     printf("APLICACION: Esta no es una opccion. \n");
58     printf("APLICACION: Vuelva a ejecutar la aplicacion. \n\r");
59   }
60
61   return 0;
62 }
```

G.3 Plantillas para gestionar el periférico memoria FIFO

G.3.1 aplicacion.h

```
1  /*********************/
2  /***DESPLAZAMIENTOS***/
3  /*********************/
4
5  /* Desplazamiento para restablecer el periferico memoria fifo *
6   * desde el espacio del usuario.                              */
7  #define DESPLAZAMIENTO_RESTABLECER_FIFO (0b0000000000000)
8
9  /* Desplazamiento para leer el periferico memoria fifo desde  *
10  * el espacio del usuario.                                    */
11 #define DESPLAZAMIENTO_LECTURA_FIFO (0b0000000000010)
12
13 /* Desplazamientos para leer las banderas del periferico      *
14  * memoria fifo desdeel espacio del usuario.                  */
15 #define DESPLAZAMIENTO_LECTURA_BANDERA_FIFO_LLENA (0b0000000000011)
16 #define DESPLAZAMIENTO_LECTURA_BANDERA_FIFO_VACIA (0b0000000000100)
17
18 /*Desplazamiento para la escribir el periferico memoria fifo  *
19  * desde el espacio del usuario.                              */
20 #define DESPLAZAMIENTO_ESCRITURA_FIFO (0b0000000000001)
21
22 /*************/
23 /***TAMANOS***/
24 /*************/
25
26 /*Tamano es la cantidad de bytes a leer/escribir.            */
27 #define TAMANO_BYTES_CHAR    (1)
28 #define TAMANO_BYTES_INT     (4)
29 #define TAMANO_BYTES_DOUBLE  (8)
30
31 /*******************************/
32 /***DECLARACION DE FUNCIONES***/
33 /*******************************/
34
35 /*Funcion PERIFERICO_BRAM_restablecer
36  * dato : Es el valor necesario para restablecer el periferico.
37  * tamano : Es la cantidad de bytes que se tomaran del dato.
38  * desplazamiento : Es la direccion por donde debe viajar el dato de reset.
39  */
40 int PERIFERICO_FIFO_restablecer(unsigned char dato, char tamano, int desplazamiento);
41
42 /*Funcion PERIFERICO_BRAM_read
43  * tamano : Es la cantidad de bytes que se tomaran del dato.
44  * desplazamiento : Es la direccion por donde debe llegar el dato a leer.
45  */
46 unsigned char PERIFERICO_FIFO_leer(char tamano, int desplazamiento);
47
48 /*Funcion PERIFERICO_BRAM_escribir
49  * dato : Es el valor a escribir en el periferico.
50  * tamano : Es la cantidad de bytes que se tomaran del dato.
51  * desplazamiento : Es la direccion por donde debe viajar el dato a escribir.
52  */
```

```
53 int PERIFERICO_FIFO_escribir(unsigned char dato, char tamano, int desplazamiento);
```

G.3.2 aplicacion.c

```
1  #include <fcntl.h>
2  #include "aplicacion.h"
3
4  /* En esta rutina se ejecutan las llamadas al sistema open, llsek, write y close. */
5  int PERIFERICO_FIFO_restablecer(unsigned char dato, char tamano, int desplazamiento)
6  {
7    int file_descriptor;
8
9    file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
10
11   lseek(file_descriptor, desplazamiento, SEEK_SET);
12
13   write(file_descriptor, &dato, tamano);
14
15   close(file_descriptor);
16
17   return 1;
18 }
19
20 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, read y close. */
21 unsigned char PERIFERICO_FIFO_leer(char tamano, int desplazamiento)
22 {
23   int file_descriptor;
24
25   unsigned char buffer[0];
26
27   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
28
29   lseek(file_descriptor, desplazamiento, SEEK_SET);
30
31   read(file_descriptor, buffer, tamano);
32
33   close(file_descriptor);
34
35   return buffer[0];
36 }
37
38 /* En esta rutina se ejecutan las llamadas al sistema open, llsek, write y close. */
39 int PERIFERICO_FIFO_escribir(unsigned char dato, char tamano, int desplazamiento)
40 {
41   int file_descriptor;
42
43   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
44
45   lseek(file_descriptor, desplazamiento, SEEK_SET);
46
47   write(file_descriptor, &dato, tamano);
48
49   close(file_descriptor);
50
51   return 1;
```

```
52 }
```

G.3.3 prueba.c

```
1  #include <stdio.h>
2  #include "aplicacion.h"
3  #include "aplicacion.c"
4
5  int main()
6  {
7    int opcion;
8    unsigned char datoR = 0b00000001;
9    unsigned char datoW1 = 0b11111111;
10   unsigned char datoW2 = 0b11111110;
11   unsigned char datoW3 = 0b11111100;
12   unsigned char datoW4 = 0b11111000;
13   unsigned char datoW5 = 0b11110000;
14   unsigned char datoW6 = 0b11100000;
15   unsigned char datoW7 = 0b11000000;
16   unsigned char datoW8 = 0b10000000;
17   unsigned char leido[9];
18   int i;
19
20   printf("APLICACION: SELECCIONAR LA OPERACION A RELIZAR. \n");
21   printf("APLICACION: 0. Reestablecer. \n");
22   printf("APLICACION: 1. Escribir. \n");
23   printf("APLICACION: 2. Leer. \n");
24   printf("APLICACION: 3. Leer bandera lleno. \n");
25   printf("APLICACION: 4. Leer bandera vacio. \n");
26   scanf("%d", &opcion);
27
28   if(opcion == 0)
29   {
30     PERIFERICO_FIFO_restablecer(datoR, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_RESTABLECER_FIFO);
31   }
32
33   if(opcion == 1)
34   {
35     PERIFERICO_FIFO_escribir(datoW1, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
36     PERIFERICO_FIFO_escribir(datoW2, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
37     PERIFERICO_FIFO_escribir(datoW3, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
38     PERIFERICO_FIFO_escribir(datoW4, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
39     PERIFERICO_FIFO_escribir(datoW5, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
40     PERIFERICO_FIFO_escribir(datoW6, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
41     PERIFERICO_FIFO_escribir(datoW7, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
42     PERIFERICO_FIFO_escribir(datoW8, TAMANO_BYTES_CHAR, DESPLAZAMIENTO_ESCRITURA_FIFO);
43   }
44
45   if(opcion == 2)
46   {
47     for(i=0; i<8; i++)
48     {
49       leido[i] = PERIFERICO_FIFO_leer(TAMANO_BYTES_CHAR, DESPLAZAMIENTO_LECTURA_FIFO);
50     }
51     for(i=0; i<8; i++)
```

```
52      {
53        printf("APLICACION: Se leyo el dato nunero %d, su valor es: 0x%x. \n", i, leido[i]);
54      }
55    }
56
57    if(opcion == 3)
58    {
59      leido[8] = PERIFERICO_FIFO_leer(TAMANO_BYTES_CHAR, DESPLAZAMIENTO_LECTURA_BANDERA_FIFO_LLENA);
60      printf("APLICACION: La bandera fifo llena es: 0x%x. \n", leido[8]);
61    }
62
63    if(opcion == 4)
64    {
65      leido[9] = PERIFERICO_FIFO_leer(TAMANO_BYTES_CHAR, DESPLAZAMIENTO_LECTURA_BANDERA_FIFO_VACIA);
66      printf("APLICACION: La bandera fifo vacia es: 0x%x. \n", leido[9]);
67    }
68
69    if(opcion > 4)
70    {
71      printf("APLICACION: Esta no es una opccion. \n");
72      printf("APLICACION: Vuelva a ejecutar la aplicacion. \n\r");
73    }
74
75    return 0;
76  }
```

ANEXO H. Módulo del *kernel*, *driver* basado en registros

El presente Anexo muestra el código fuente del módulo que se añadió al *kernel*, el cual sirvió como puente para efectuar desde el espacio del usuario la escritura y lectura de los de los periféricos. También se presentan los archivos complementarios que llevan a buen termino el uso módulo.

H.1 Plantilla del módulo periferico.c

```
1  /*************************************/
2  /*DECLARACION DE LIBRERIAS Y CABECEROS*/
3  /*************************************/
4  #include<linux/init.h>
5  #include<linux/module.h>
6  #include<linux/fs.h>
7  #include<linux/cdev.h>
8  #include<linux/slab.h>
9  #include<asm/uaccess.h>
10 #include<linux/ioport.h>
11 #include<asm/io.h>
12
13 /*********************************/
14 /*INFORMACION ASOCIADA AL PERIFERICO*/
15 /*********************************/
16 #define PERIFERICO_NAME "Periferico_PERIFERICO"
17
18 #define PERIFERICO_NUMERO_REGISTROS_CS2 3
19 #define PERIFERICO_BASEADDRESS_CS2 0x10010140
20 #define PERIFERICO_MEMSPACE_SIZE_CS2 PERIFERICO_NUMERO_REGISTROS_CS2*4
21
22 #define PERIFERICO_NUMERO_REGISTROS_DATA 20
23 #define PERIFERICO_BASEADDRESS_DATA 0x14000000
24 #define PERIFERICO_MEMSPACE_SIZE_DATA PERIFERICO_NUMERO_REGISTROS_DATA*1
25
26 /*****************************************************************/
27 /*INFORMACION ASOCIADA A LOS PARAMETROS USADOS EN LA FUNCION init*/
28 /*****************************************************************/
29 static char *quien   = "Mundo";
30 static int  cuantos = 1;
31 module_param(quien, charp, S_IRUGO);
```

```
32 module_param(cuantos, int, S_IRUGO);
33
34 /*****************************************************************/
35 /*INFORMACION ASOCIADA A LA RESERVA DE LOS NUMEROS DE DISPOSITIVO*/
36 /*****************************************************************/
37 static dev_t mydev;
38 static unsigned int primermenor = 0;
39 static unsigned int cuenta = 1;
40 static char *nombre = PERIFERICO_NAME;
41
42 /*****************************************/
43 /*OPERACIONES DE ARCHIVO USADAS EN EL DRIVER*/
44 /*****************************************/
45 static int PERIFERICO_open(struct inode *, struct file *);
46 static int PERIFERICO_release(struct inode *, struct file *);
47 static loff_t PERIFERICO_llseek(struct file *, loff_t, int);
48 static ssize_t PERIFERICO_read(struct file *, char __user *, size_t, loff_t *);
49 static ssize_t PERIFERICO_write(struct file *, const char __user *, size_t, loff_t *);
50
51 /*********************************************/
52 /*INICIALIZACION DE LAS OPERACIONES DE ARCHIVO*/
53 /*********************************************/
54 struct file_operations PERIFERICO_fops = {
55   .owner    = THIS_MODULE,
56   .open     = PERIFERICO_open,
57   .release  = PERIFERICO_release,
58   .read     = PERIFERICO_read,
59   .write    = PERIFERICO_write,
60   .llseek   = PERIFERICO_llseek
61 };
62
63 /*****************************************************/
64 /*ESTRUCTURA QUE REPRESENTA AL DISPOSITIVO DE CARACTERES*/
65 /*****************************************************/
66 struct PERIFERICO_dev {
67   char nombre_periferico[21];
68   unsigned int DireccionBase_cs2;
69   unsigned int DireccionBase_datos;
70   unsigned char numero_de_registros_datos;
71   unsigned char *intercambio_de_datos;
72   struct cdev cdev;
73 };
74
75 /*****************************************************************************************/
76 /*SE DECLARA UN PUNTERO HACIA LA ESTRUCTURA DE DISPOSITIVO PARA SER USADO EN init y exit*/
77 /*****************************************************************************************/
78 struct PERIFERICO_dev *my_PERIFERICO_dev;
79
80 /*************************************************************************/
81 /*DECLARACION DE PUNTEROS Y VARIABLES A UTILZAR EN LAS OPERACIONES DE ARCHIVO*/
82 /*************************************************************************/
83 static char bandera;
84 static void *PERIFERICO_iomem_pointer_cs2;
85 static void *PERIFERICO_iomem_pointer_data;
86
87 /*******************************/
88 /*FUNCION DE INICIALIZACION init*/
```

```
89 /*********************************/
90 static int PERIFERICO_init(void)
91 {
92   /*Declaracion de variables usadas en la funcion init*/
93   int k;
94   int resultado;
95   int error;
96
97   /*Mensaje de apertura de la funcion init*/
98   printk(KERN_ALERT "************ PERIFERICO_init ************ \n\r");
99   printk(KERN_ALERT "PERIFERICO_init: Estoy ejecutando la funcion PERIFERICO_init. \n\r");
100
101  /*Parametros que se cargan en la funcion init*/
102  for (k=0; k<cuantos; k++)
103  {
104    printk(KERN_ALERT "PERIFERICO_init: Hola %s. \n\r", quien);
105  }
106
107  /*Reserva dinamica de los numeros Mayores y Menores*/
108  resultado = alloc_chrdev_region(&mydev, primermenor, cuenta, nombre);
109  if(resultado == 0)
110  {
111    printk(KERN_ALERT "PERIFERICO_init: Los numeros reservados fueron: \n# Mayor: %d. \n# Menor: %d. \n\r", MAJOR(mydev), MINOR(mydev));
112  } else {
113    printk(KERN_ALERT "PERIFERICO_init: Hubo un error y los numeros no se reservaron. El error fue: %d. \n\r", resultado);
114  }
115
116  /*Reserva de memoria para la estructura que representa al dispositivo de caracteres*/
117  my_PERIFERICO_dev = kmalloc(sizeof(struct PERIFERICO_dev), GFP_KERNEL);
118  if(my_PERIFERICO_dev==NULL)
119  {
120    printk(KERN_ALERT "PERIFERICO_init: No se puede reservar memoria, toca reiniciar. \n\r");
121  } else {
122    printk(KERN_ALERT "PERIFERICO_init: Se pudo reservar memoria, se reservaron %d bytes. \n\r", (int) sizeof(struct PERIFERICO_dev));
123  }
124
125  /*Inicializacion de la estructura que representa al dispositivo de caracteres*/
126  cdev_init(&my_PERIFERICO_dev->cdev,&PERIFERICO_fops);
127  my_PERIFERICO_dev->cdev.owner = THIS_MODULE;
128  my_PERIFERICO_dev->cdev.ops = &PERIFERICO_fops;
129  error = cdev_add(&my_PERIFERICO_dev->cdev, mydev, cuenta);
130  if(error < 0)
131  {
132    printk(KERN_ALERT "PERIFERICO_init: Hubo un error al registrar el dispositivo. El error fue: %d. \n\r", error);
133  } else {
134    printk(KERN_ALERT "PERIFERICO_init: Se registro exitosamente el dispositivo. \n\r");
135  }
136
137  /*Inicializando bandera a cero, para indicar que el dispositivo no se ha abierto*/
138  bandera = 0;
139  printk(KERN_ALERT "PERIFERICO_init: Colocando bandera = %d, para indicar que el driver no se ha inicializado. \n\r", bandera);
140
141  /*Mensaje de despedida de la funcion init*/
142  printk(KERN_ALERT "PERIFERICO_init: Estoy saliendo de la funcion PERIFERICO_init. \n\n\r");
143  return resultado;
144 }
145
```

```
146 /*************************/
147 /*FUNCION DE LIMPIEZA exit*/
148 /*************************/
149 static void PERIFERICO_exit(void)
150 {
151   /*Menesaje de apertura de la funcion exit*/
152   printk(KERN_ALERT "************ PERIFERICO_exit ************ \n\r");
153   printk(KERN_ALERT "PERIFERICO_exit: Estoy ejecutando la funcion PERIFERICO_exit. \n\r");
154
155   /*Retiro del sistema la estructura que representa al dispositivo de caracteres*/
156   cdev_del(&my_PERIFERICO_dev->cdev);
157   printk(KERN_ALERT "PERIFERICO_release: Retirando del sistema la estructura de dispositivo. \n\r");
158
159   /*Liberacion de la porcion de memoria de la estructura que representa al dispositivo de caracteres*/
160   kfree(my_PERIFERICO_dev);
161   printk(KERN_ALERT "PERIFERICO_release: Liberando el espacio de memoria de la estructura de dspositivo. \n\r");
162
163   /*Liberacion de los numeros Mayores y Menores reservados dinamicamente*/
164   unregister_chrdev_region(mydev,cuenta);
165   printk(KERN_ALERT "PERIFERICO_release: Liberando del sistema los numeros de dispositivo. \n\r");
166
167   /*Liberacion de la porcion de memoria asignada en la funcion open para el intercambio de datos*/
168   if(&my_PERIFERICO_dev->intercambio_de_datos != NULL && bandera==1)
169   {
170     kfree(my_PERIFERICO_dev->intercambio_de_datos);
171     printk(KERN_ALERT "PERIFERICO_release: Liberando el espacio de memoria para el intercambio de datos. \n\r");
172   }
173
174   /*Mensaje de despedidad de la funcion exit*/
175   printk(KERN_ALERT "PERIFERICO_exit: Adios mundo cruel. \n\r ");
176   printk(KERN_ALERT "PERIFERICO_exit: Estoy saliendo de la funcion PERIFERICO_exit. \n\n\r");
177 }
178
179 /*************************/
180 /*LLAMADA AL SISTEMA open*/
181 /*************************/
182 static int PERIFERICO_open(struct inode *puntero_inode, struct file *puntero_file)
183 {
184   /*Declaracion de variables usadas en la funcion open*/
185   int tamano;
186   int k;
187
188   /*Declracion de un punetro que apunta a la estructura que representa al dispositivo de caracteres*/
189   struct PERIFERICO_dev *el_PERIFERICO;
190
191   /*Mensaje de apertura de la funcion open*/
192   printk(KERN_ALERT "************ PERIFERICO_open ************ \n\r");
193   printk(KERN_ALERT "PERIFERICO_open: Estoy ejecutando la funcion PERIFERICO_open. \n\r");
194   printk(KERN_ALERT "PERIFERICO_open: La informacion de puntero_inode es: 0x%X. \n\r", (unsigned int)puntero_inode);
195   printk(KERN_ALERT "PERIFERICO_open: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
196
197   /*Proceso que verifica si la estructura de dispositivo es adecuada*/
198   el_PERIFERICO = container_of(puntero_inode->i_cdev, struct PERIFERICO_dev, cdev);
199   printk(KERN_ALERT "PERIFERICO_open: La informacion de container_of es: 0x%X. \n\r", (unsigned int)puntero_inode->i_cdev);
200
201   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a open*/
202   puntero_file->private_data = el_PERIFERICO;
```

```
203
204   if(bandera==0)
205   {
206     /*Inicializando los campos de la estructura que representa al dispositivo de carcacteres*/
207     printk(KERN_ALERT "PERIFERICO_open: Inicializando los campos de PERIIFERICO_dev. bandera= %d. \n\r", bandera);
208
209     strcpy(el_PERIFERICO->nombre_periferico, PERIFERICO_NAME);
210     el_PERIFERICO->DireccionBase_cs2 = PERIFERICO_BASEADDRESS_CS2;
211     el_PERIFERICO->DireccionBase_datos = PERIFERICO_BASEADDRESS_DATA;
212     el_PERIFERICO->numero_de_registros_datos = PERIFERICO_NUMERO_REGISTROS_DATA;
213     tamano = el_PERIFERICO->numero_de_registros_datos;
214     el_PERIFERICO->intercambio_de_datos = kmalloc(tamano*sizeof(char), GFP_KERNEL);
215
216     for(k=0; k<tamano; k++)
217     {
218       el_PERIFERICO->intercambio_de_datos[k] = 0;
219     }
220
221     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->nombre_periferico: %s. \n\r", el_PERIFERICO->nombre_periferico);
222     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->DireccionBase_cs2: 0x%X. \n\r", el_PERIFERICO->DireccionBase_cs2);
223     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->DireccionBase_data: 0x%X. \n\r", el_PERIFERICO->DireccionBase_datos);
224     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->numero_de_registros_datos: %d. \n\r", el_PERIFERICO->numero_de_registros_datos);
225     printk(KERN_ALERT "PERIFERICO_open: La reserva de memoria para el_PERIFERICO->intercambio_de_datos es de: %u bytes. \n\r", \
226     tamano*sizeof(char));
227
228     /*Colocando la bandera a uno para indicar que el dispositivo ya ha sido abierto*/
229     bandera=1;
230
231   } else {
232     printk(KERN_ALERT "PERIFERICO_open: Ya estaban inicializados los campos de PERIFERICO_dev. bandera= %d. \n\r", bandera);
233   }
234
235   /*Se le indica al nucleo que se se desea hacer uso de puertos de E/S*/
236   printk(KERN_ALERT "PERIFERICO_open: Primera solicitud de region de meoria de puertos E/S. \n\r");
237   if(!request_mem_region(PERIFERICO_BASEADDRESS_CS2, PERIFERICO_MEMSPACE_SIZE_CS2, PERIFERICO_NAME))
238   {
239     printk(KERN_ALERT "PERIFERICO_open: No se pudo realizar la primera asignacion de memoria de puertos E/S. \n\r");
240   } else {
241     printk(KERN_ALERT "PERIFERICO_open: Se realizo la primera asignacion de memoria de puertos E/S. \n\r");
242
243     /*Realizando la asignacion y direccionamiento de los puerto de E/S*/
244     printk(KERN_ALERT "PERIFERICO_open: Realizando el primer mapeo de memoria en bytes para los puertos E/S \n\r");
245     PERIFERICO_iomem_pointer_cs2 = ioremap(PERIFERICO_BASEADDRESS_CS2, PERIFERICO_MEMSPACE_SIZE_CS2);
246     printk(KERN_ALERT "PERIFERICO_open: El primer puntero devuelto es: 0x%X. \n\r", (unsigned int)PERIFERICO_iomem_pointer_cs2);
247   }
248
249   /*Se le indica al nucleo que se se desea hacer uso de puertos de E/S*/
250   printk(KERN_ALERT "PERIFERICO_open: Segunda solicitud de region de meoria de puertos E/S. \n\r");
251   if(!request_mem_region(PERIFERICO_BASEADDRESS_DATA, PERIFERICO_MEMSPACE_SIZE_DATA, PERIFERICO_NAME))
252   {
253     printk(KERN_ALERT "PERIFERICO_open: No se pudo realizar la segunda asignacion de memoria de puertos E/S. \n\r");
254   } else {
255     printk(KERN_ALERT "PERIFERICO_open: Se realizo la segunda asignacion de memoria de puertos E/S. \n\r");
256
257     /*Realizando la asignacion y direccionamiento virtual de los puerto de E/S*/
258     printk(KERN_ALERT "PERIFERICO_open: Realizando el segundo mapeo de memoria en bytes de los puertos E/S. \n\r");
259     PERIFERICO_iomem_pointer_data = ioremap(PERIFERICO_BASEADDRESS_DATA, PERIFERICO_MEMSPACE_SIZE_DATA);
```

```
260     printk(KERN_ALERT "PERIFERICO_open: El segundo puntero devuelto es: 0x%X. \n\r", (unsigned int)PERIFERICO_iomem_pointer_data);
261   }
262
263   /*Mensaje de despedidad de la funcion open*/
264   printk(KERN_ALERT "PERIFERICO_open: Estoy saliendo de la funcion PERIFERICO_open. \n\n\r");
265   return 0;
266 }
267
268 /*****************************/
269 /*LLAMADA AL SISTEMA release*/
270 /*****************************/
271 static int PERIFERICO_release(struct inode *puntero_inode, struct file *puntero_file)
272 {
273   /*Mensaje de apertura de la funcion release*/
274   printk(KERN_ALERT "************ PERIFERICO_release ************ \n\r");
275   printk(KERN_ALERT "PERIFERICO_release: Estoy ejecutando la funcion PERIFERICO_release. \n\r");
276   printk(KERN_ALERT "PERIFERICO_release: La informacion de puntero_inode es: 0x%X. \n\r", (unsigned int)puntero_inode);
277   printk(KERN_ALERT "PERIFERICO_release: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
278
279   /*Liberacion de los puertos E/S virtualizados*/
280   printk(KERN_ALERT "PERIFERICO_release: Liberando el mapeo de memoria. \n\r");
281   iounmap(PERIFERICO_iomem_pointer_data);
282   iounmap(PERIFERICO_iomem_pointer_cs2);
283
284   /*Liberacion de los puertos de E/S desde el nucleo*/
285   printk(KERN_ALERT "PERIFERICO_release: Liberando la solicitud de region de memoria para los puertos E/S. \n\r");
286   release_mem_region(PERIFERICO_BASEADDRESS_DATA, PERIFERICO_MEMSPACE_SIZE_DATA);
287   release_mem_region(PERIFERICO_BASEADDRESS_CS2, PERIFERICO_MEMSPACE_SIZE_CS2);
288
289   /*Mensaje de despedidad de la funcion release*/
290   printk(KERN_ALERT "PERIFERICO_release: Estoy saliendo de la funcion PERIFERICO_release. \n\n\r");
291   return 0;
292 }
293
294 /************************/
295 /*LLAMADA AL SISTEMA read*/
296 /************************/
297 static ssize_t PERIFERICO_read(struct file *puntero_file, char __user *puntero_usuario, size_t tamano, loff_t *puntero_offset)
298 {
299   /*Declaracion de variables usadas en la funcion read*/
300   ssize_t retval = 0;
301   char leido;
302
303   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a read*/
304   struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
305
306   /*Declaracion de una variable que toma la informacion de un campo de la estructura que representa al dispositivo de caracteres*/
307   char cantidad_de_registros_datos = el_PERIFERICO->numero_de_registros_datos;
308
309   /*Mensaje de apertura de la funcion read*/
310   printk(KERN_ALERT "************ PERIFERICO_read ************ \n\r");
311   printk(KERN_ALERT "PERIFERICO_read: Estoy ejecutando la funcion PERIFERICO_read. \n\r");
312   printk(KERN_ALERT "PERIFERICO_read: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
313   printk(KERN_ALERT "PERIFERICO_read: La informacion de puntero_usuario es: 0x%X. \n\r", (unsigned char)puntero_usuario[0]);
314   printk(KERN_ALERT "PERIFERICO_read: La informacion de tamano es: %u bytes. \n\r", (unsigned int)tamano);
315   printk(KERN_ALERT "PERIFERICO_read: La informacion de puntero_offset es: %u bytes. \n\r", (unsigned int)*puntero_offset);
316
```

```
317   /*Configurando tamano al maximo valor de lectura*/
318   printk(KERN_ALERT "PERIFERICO_read: Configurando la cantidad de datos que se pueden leer. \n\r");
319   if(tamano >= PERIFERICO_MEMSPACE_SIZE_DATA)
320   {
321     tamano = PERIFERICO_MEMSPACE_SIZE_DATA;
322   }
323
324   /*Acotando el valor de retorno hasta el maximo de informacion que se puede leer*/
325   if((*puntero_offset + tamano) > cantidad_de_registros_datos*1)
326   {
327     retval = cantidad_de_registros_datos*1 - *puntero_offset;
328     printk(KERN_ALERT "PERIFERICO_read: La lectura que se pide sobrepasa la cantidad de registros del periferico. \n\r");
329     printk(KERN_ALERT "PERIFERICO_read: Leyendo los datos que se pueden hasta llegar al final. La lectura es de = %d bytes. \n\r", \
330     retval);
331   } else {
332     retval = tamano;
333     printk(KERN_ALERT "PERIFERICO_read: La lectura que se pide no sobrepasa la cantidad de registros del periferico \n\r");
334     printk(KERN_ALERT "PERIFERICO_read: Leyendo la cantidad de datos solicitados. La lectura es de = %d bytes. \n\r", retval);
335   }
336
337   /*Realizando el proceso de lectura en el sentido periferico-driver-usuario*/
338   printk(KERN_ALERT "PERIFERICO_read: Realizando el traspaso de datos desde el periferico al nucleo. \n\r");
339   if(retval > 0)
340   {
341     /*Leyendo la informacion del periferico desde el espacio del nucleo*/
342     leido = ioread8(PERIFERICO_iomem_pointer_data + (unsigned int)*puntero_offset);
343     *el_PERIFERICO->intercambio_de_datos = leido;
344     printk(KERN_ALERT "PERIFERICO_read: Se recibio informacion del periferico en el nucleo. Esta es: 0x%X. \n\r", \
345     el_PERIFERICO->intercambio_de_datos[0]);
346
347     /*Envio de la informacion leida desde el espacio del nucleo al espacio del usuario*/
348     printk(KERN_ALERT "PERIFERICO_read: Realizando el traspaso de datos desde el nucleo al usuario. \n\r");
349     if(copy_to_user(puntero_usuario, el_PERIFERICO->intercambio_de_datos, retval))
350     {
351       printk(KERN_ALERT "PERIFERICO_read: No se ha recibido la imformacion desde el nucleo en el usuario. \n\r");
352     } else {
353       printk(KERN_ALERT "PERIFERICO_read: Se recibio la informacion desde el nucleo en el usuario. \n\r");
354
355       /*Actualizacion del puntero del archivo si el traspaso periferico-nucleo-usuario fue exitoso*/
356       printk(KERN_ALERT "PERIFERICO_read: Actualizando el puntero_offset. \n\r");
357       *puntero_offset += retval;
358       printk(KERN_ALERT "PERIFERICO_read: Al salir de la funcion read, puntero_offset: %u bytes. \n\r", (unsigned int)*puntero_offset);
359     }
360   }
361
362   /*Mensaje de despedidad de la funcion read*/
363   printk(KERN_ALERT "PERIFERICO_read: Estoy saliendo de la funcion PERIFERICO_read. \n\n\r");
364   return retval;
365 }
366
367 /**************************/
368 /*LLAMADA AL SISTEMA write*/
369 /**************************/
370 static ssize_t PERIFERICO_write(struct file *puntero_file, const char __user *puntero_usuario, size_t tamano, loff_t *puntero_offset)
371 {
372   /*Declaracion de variables usadas en la funcion write*/
373   ssize_t retval;
```

```
374   unsigned int PBFUN_CS2;
375   unsigned char escrito;
376
377   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a write*/
378   struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
379
380   /*Mensaje de apertura de la funcion write*/
381   printk(KERN_ALERT "************ PERIFERICO_write ************ \n\r");
382   printk(KERN_ALERT "PERIFERICO_write: Estoy ejecutando la funcion PERIFERICO_write. \n\r");
383   printk(KERN_ALERT "PERIFERICO_write: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
384   printk(KERN_ALERT "PERIFERICO_write: La informacion de puntero_usuario es: 0x%X. \n\r", (unsigned char)puntero_usuario[0]);
385   printk(KERN_ALERT "PERIFERICO_write: La informacion de tamano es: %u bytes. \n\r", (unsigned int)tamano);
386   printk(KERN_ALERT "PERIFERICO_write: La informacion de puntero_offset es: %u bytes. \n\r", (unsigned int)*puntero_offset);
387
388   /*Acotando el valor de retorno hasta el maximo de informacion que se puede escribir*/
389   printk(KERN_ALERT "PERIFERICO_write: Configurando la transferencia de datos que se pueden escribir. \n\r");
390   if(tamano > PERIFERICO_MEMSPACE_SIZE_DATA)
391   {
392     retval = PERIFERICO_MEMSPACE_SIZE_DATA;
393   } else {
394     retval = tamano;
395   }
396
397   /*Configurando el registro que permite hacer uso de perifericos en la FPGA*/
398   PBFUN_CS2 = ioread32(PERIFERICO_iomem_pointer_cs2 + 0);
399   printk(KERN_ALERT "PERIFERICO_write: La informacion inicial del registro PBFUN_CS2 es: 0x%X. \n\r", PBFUN_CS2);
400   if (PBFUN_CS2 == 0x87F9FFFF)
401   {
402     printk(KERN_ALERT "PERIFERICO_write: La senal de habilitacion de la FPGA (CS2) ya se encuentra activa. \n\r");
403   }else{
404     printk(KERN_ALERT "PERIFERICO_write: Escribiendo desde el nucleo el dato de activacion de la senal CS2. \n\r");
405     iowrite32(0x04000000, PERIFERICO_iomem_pointer_cs2 + 4);
406     printk(KERN_ALERT "PERIFERICO_write: La senal de habilitacion de la FPGA (CS2) ahora se encuentra activa. \n\r");
407   }
408   PBFUN_CS2 = ioread32(PERIFERICO_iomem_pointer_cs2 + 0);
409   printk(KERN_ALERT "PERIFERICO_write: La informacion final del registro PBFUN_CS2 es: 0x%X. \n\r", PBFUN_CS2);
410
411   /*Realizando el proceso de escritura desde el espacio del usuario al espacio del nucleo*/
412   printk(KERN_ALERT "PERIFERICO_write: Realizando el traspaso de datos desde el usuario al nucleo. \n\r");
413   if(copy_from_user(el_PERIFERICO->intercambio_de_datos, puntero_usuario, retval))
414   {
415     printk(KERN_ALERT "PERIFERICO_write: No se ha recibido informacion del usuario en el nucleo. \n\r");
416   } else {
417     printk(KERN_ALERT "PERIFERICO_write: Se recibio informacion del usuario en el nucleo. Esta es: 0x%X. \n\r", \
418     el_PERIFERICO->intercambio_de_datos[0]);
419
420     /*Realizando el proceso de escritura desde el espacio del nucleo al periferico*/
421     printk(KERN_ALERT "PERIFERICO_write: Realizando el traspaso de datos desde el espacio del nucleo al periferico. \n\r");
422     escrito = *el_PERIFERICO->intercambio_de_datos;
423     iowrite8(escrito, PERIFERICO_iomem_pointer_data + (unsigned int)*puntero_offset);
424
425     /*Actualizacion del puntero del archivo si el traspaso usuario-nucleo-periferico fue exitoso*/
426     printk(KERN_ALERT "PERIFERICO_write: Actualizando el puntero_offset. \n\r");
427     *puntero_offset += retval;
428     printk(KERN_ALERT "PERIFERICO_write: Al salir de la funcion write, puntero_offset: %u bytes. \n\r", (unsigned int)*puntero_offset);
429   }
430
```

```
431   /*Mensaje de despedidad de la funcion write*/
432   printk(KERN_ALERT "PERIFERICO_write: Estoy saliendo de la funcion PERIFERICO_write. \n\n\r");
433   return retval;
434 }
435
436 /***************************/
437 /*LLAMADA AL SISTEMA llseek*/
438 /***************************/
439 static loff_t PERIFERICO_llseek(struct file *puntero_file, loff_t offset, int whence)
440 {
441   /*Declaracion de variables usadas en la funcion llseek*/
442   loff_t retval;
443
444   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a llseek*/
445   struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
446
447   /*Mensaje de apertura de la funcion llseek*/
448   printk(KERN_ALERT "************ PERIFERICO_llsek ************ \n\r");
449   printk(KERN_ALERT "PERIFERICO_llseek: Estoy ejecutando la funcion PERIFERICO_llseek. \n\r");
450   printk(KERN_ALERT "PERIFERICO_llseek: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
451   printk(KERN_ALERT "PERIFERICO_llseek: La informacion de offset es: %u bytes. \n\r", (unsigned int)offset);
452   printk(KERN_ALERT "PERIFERICO_llseek: La informacion de whence es: %d. \n\r", whence);
453
454   /*Especificacion de los diferentes casos para ajustar el puntero del archivo*/
455   switch(whence)
456   {
457     case 0: /* SEEK_SET */
458             retval = offset;
459             printk(KERN_ALERT "PERIFERICO_llseek: Estoy ejecutando SEEK_SET. \n\r");
460             break;
461
462     case 1: /* SEEK_CUR */
463             retval = puntero_file->f_pos + offset;
464             printk(KERN_ALERT "PERIFERICO_llseek: Estoy ejecutando SEEK_CUR. \n\r");
465             break;
466
467     case 2: /* SEEK_END */
468             retval = el_PERIFERICO->numero_de_registros_datos*1 + offset;
469             printk(KERN_ALERT "PERIFERICO_llseek: Estoy ejecutando SEEK_END.\n\r");
470             break;
471
472     default: /* No puede ocurrir */
473              return -EINVAL;
474   }
475
476   /*Se verifica si se hizo uso de los casos de forma correcta*/
477   if (retval < 0)
478   {
479     printk(KERN_ALERT "PERIFERICO_llseek: Hubo un error en llseek, este es retval: %d bytes. \n\r", (unsigned int)retval);
480   } else {
481     printk(KERN_ALERT "PERIFERICO_llseek: Todo salio bien en llseek. \n\r");
482
483     /*Actualizacion del puntero del archivo*/
484     printk(KERN_ALERT "PERIFERICO_llseek: Actualizando el puntero_file->f_pos. \n\r");
485     puntero_file->f_pos = retval;
486     printk(KERN_ALERT "PERIFERICO_llseek: Al salir de la funcion llseek, puntero_file->f_pos: %u bytes. \n\r", (unsigned int)retval);
487   }
```

```
488
489   /*Mensaje de despedidad de la funcion llseek*/
490   printk(KERN_ALERT "PERIFERICO_llseek: Estoy saliendo de la funcion PERIFERICO_llseek. \n\n\r");
491   return retval;
492 }
493
494 /*************************************************************************************/
495 /*FUNCIONES QUE SE EJECUTAN CUANDO SE REALIZA DESDE EL ESPACIO DEL USUARIO insmod y rmmod*/
496 /*************************************************************************************/
497 module_init(PERIFERICO_init);
498 module_exit(PERIFERICO_exit);
499
500 /*********************************/
501 /*INFORMACION ADICIONAL DEL DRIVER*/
502 /*********************************/
503 MODULE_LICENSE("Dual BSD/GPL");
504 MODULE_AUTHOR("Arides Jose Meneses Hernandez <ajomeher@hotmail.com>");
505 MODULE_ALIAS("Plantilla Periferico_PERIFERICO");
506 MODULE_DESCRIPTION("Driver basado en registros. Adaptado de la plantilla de los profesores: William Salamanca \
507 <williamsalamanca@gmail.com> - Sergio Abreo <abreosergio@gmail.com>");
508 MODULE_VERSION("1:0.0");
```

H.2 Archivo Makefile

```
1  EXTRA_CFLAGS += -Wall -I.
2  CC              = mipsel-openwrt-linux-gcc
3  OPENWRT_BASE    = /path/trunk/
4  KERNEL_SRC      = $(OPENWRT_BASE)/build_dir/linux-xburst_qi_lb60/linux-2.6.37.6
5  CROSS_COMPILE   = mipsel-openwrt-linux-
6
7  obj-m +=  periferico.o
8
9  all:
10         make -C $(KERNEL_SRC) M=$(PWD) ARCH=mips CROSS_COMPILE=$(CROSS_COMPILE) modules
```

H.3 Archivo instalar.sh

```
1  #!/bin/sh
2  module="periferico"
3  device="Periferico_PERIFERICO"
4  mode="664"
5
6  /sbin/insmod ./$module.ko $* || exit 1
7
8  major=$(cat /proc/devices | awk '{if($2=="Periferico_PERIFERICO") print $1 ; else print pailas}' | grep -v pailas)
9
10 mknod /dev/${device} c $major 0
```

H.4 Archivo desinstalar.sh

```
1 #!/bin/sh
2 module="periferico"
3 device="Periferico_PERIFERICO"
4
5 /sbin/rmmod ./$module.ko $* || exit 1
6
7 rm -f /dev/${device}
```

H.5 Adaptando el código fuente a un periférico en particular

En el código fuente mostrado, se ha manejado las palabras claves **periferico** y **PERIFERICO**, estos nombres deben ser sustituidos por el nombre del periférico que se esté diseñando, así toman significado los nombres de las funciones, las variables y hasta el id con el que queda registrado el módulo en el *kernel*. Para adaptar estos archivos a un periférico en particular, por ejemplo a uno llamado **leds**, se deben seguir los siguientes pasos:

Nombre del driver: El archivo principal se llama periferico.c, para renombrarlo se debe ejecutar el siguiente comando:

```
1 rename -v 's/periferico/leds/' periferico.*
```

Cambio en el objetivo del Makefile: El Makefile del directorio del driver asume que la fuente se sigue llamando periférico.c, por lo tanto se debe modificar la séptima linea así:

```
1 obj-m += leds.o
```

Modificaciones en el driver: En el contenido de esta fuente se utiliza el nombre PERIFERICO en letras mayúsculas, para renombrarlo se debe abrir el archivo con **vim**, y luego ejecutar:

```
1 :%s/PERIFERICO/LEDS/g
```

Modificaciones en instalar.sh: En este archivo es necesario modificar las lineas dos, tres y ocho como se muestra a continuación:

```
1 module="leds"
2 device="Periferico_leds"
3 major=$(cat /proc/devices | awk '{if($2=="Periferico_PERIFERICO") print $1 ; else print pailas}' | grep -v pailas)
```

Modificaciones en desinstalar.sh: En este archivo se deben modificar las lineas dos y tres como se muestra a continuación:

```
1 module="leds"
2 device="Periferico_leds"
```

ANEXO I. Módulo del *kernel*, *driver* para manejar interrupciones

En este Anexo se lista el código fuente del módulo que se añadió al *kernel*, encargado de administrar las interrupciones de periféricos y se presentan los archivos que componen la aplicación del usuario.

I.1 Plantilla del módulo periferico.c

```
1  /*************************************/
2  /*DECLARACION DE LIBRERIAS Y CABECEROS*/
3  /*************************************/
4  #include<linux/init.h>
5  #include<linux/module.h>
6  #include<linux/fs.h>
7  #include<linux/cdev.h>
8  #include<linux/slab.h>
9  #include<asm/uaccess.h>
10 #include<linux/ioport.h>
11 #include<asm/io.h>
12 #include <linux/irq.h>
13 #include <linux/interrupt.h>
14 #include <linux/gpio.h>
15 #include <asm/mach-jz4740/gpio.h>
16
17 /***********************************/
18 /*INFORMACION ASOCIADA AL PERIFERICO*/
19 /***********************************/
20 #define PERIFERICO_NAME "Periferico_PERIFERICO"
21
22 #define PERIFERICO_NUMERO_REGISTROS_DATA 20
23 #define PERIFERICO_BASEADDRESS_DATA 0x14000000
24 #define PERIFERICO_MEMSPACE_SIZE_DATA PERIFERICO_NUMERO_REGISTROS_DATA*1
25
26 #define PERIFERICO_IRQ_PIN JZ_GPIO_PORTC(15)
27
28 /*************************************************************/
29 /*INFORMACION ASOCIADA A LOS PARAMETROS USADOS EN LA FUNCION init*/
30 /*************************************************************/
31 static char *quien   = "Mundo";
32 static int  cuantos = 1;
33 module_param(quien, charp, S_IRUGO);
```

```
34 module_param(cuantos, int, S_IRUGO);
35
36 /*****************************************************************/
37 /*INFORMACION ASOCIADA A LA RESERVA DE LOS NUMEROS DE DISPOSITIVO*/
38 /*****************************************************************/
39 static dev_t mydev;
40 static unsigned int primermenor = 0;
41 static unsigned int cuenta = 1;
42 static char *nombre = PERIFERICO_NAME;
43
44 /*****************************************/
45 /*OPERACIONES DE ARCHIVO USADAS EN EL DRIVER*/
46 /*****************************************/
47 static int PERIFERICO_open(struct inode *, struct file *);
48 static int PERIFERICO_release(struct inode *, struct file *);
49 static ssize_t PERIFERICO_read(struct file *, char __user *, size_t, loff_t *);
50 static ssize_t PERIFERICO_write(struct file *, const char __user *, size_t, loff_t *);
51
52 /*********************************************/
53 /*INICIALIZACION DE LAS OPERACIONES DE ARCHIVO*/
54 /*********************************************/
55 struct file_operations PERIFERICO_fops = {
56   .owner    = THIS_MODULE,
57   .open     = PERIFERICO_open,
58   .release  = PERIFERICO_release,
59   .read     = PERIFERICO_read,
60   .write    = PERIFERICO_write
61 };
62
63 /*****************************************************/
64 /*ESTRUCTURA QUE REPRESENTA AL DISPOSITIVO DE CARACTERES*/
65 /*****************************************************/
66 struct PERIFERICO_dev {
67   char nombre_periferico[21];
68   unsigned int DireccionBase_datos;
69   unsigned char numero_de_registros_datos;
70   unsigned char *intercambio_de_datos;
71   struct cdev cdev;
72 };
73
74 /*****************************************************************************/
75 /*SE DECLARA UN PUNTERO HACIA LA ESTRUCTURA DE DISPOSITIVO PARA SER USADO EN init y exit*/
76 /*****************************************************************************/
77 struct PERIFERICO_dev *my_PERIFERICO_dev;
78
79 /*************************************************************************/
80 /*DECLARACION DE PUNTEROS Y VARIABLES A UTILZAR EN LAS OPERACIONES DE ARCHIVO*/
81 /*************************************************************************/
82 static int irq_habilitador = 0;
83 static unsigned int irq_contador = 0;
84 static char bandera;
85 static void *PERIFERICO_iomem_pointer_data;
86
87 /*********************/
88 /*FUNCION irq_handler*/
89 /*********************/
90 static irqreturn_t irq_handler(int irq_llegada, void *dev_id)
```

```
91 {
92   /*Menesaje de apertura de la funcion irq_handler*/
93   printk(KERN_ALERT "************ PERIFERICO_irq ************ \n\r");
94   printk(KERN_ALERT "PERIFERICO_open: Estoy ejecutando la funcion PERIFERICO_irq. \n\r");
95
96   /*Rutina que verifica si la interrupccion esta habilitada*/
97   if(irq_habilitador)
98   {
99     /*Cuando sucede la interrupcion se incrementa un contador*/
100    printk(KERN_ALERT "PERIFERICO_irq: La interrupcion ha sido habilitada. \n\r");
101    irq_contador++;
102    printk(KERN_ALERT "PERIFERICO_irq: La cantidad de interrupciones hasta ahora son: %u. \n\r", irq_contador);
103
104    /*En este punto es donde se debe enviar la informacion correspondiente al periferico*/
105    printk(KERN_ALERT "PERIFERICO_irq: ************************************************** \n\r");
106    printk(KERN_ALERT "PERIFERICO_irq: Es el momento de enviar la informacion al periferico. \n\r");
107    printk(KERN_ALERT "PERIFERICO_irq: :) Enviando informacion (: . \n\r");
108    printk(KERN_ALERT "PERIFERICO_irq: ************************************************** \n\r");
109  } else {
110    /*Si no a ocurrido la interrupcion se da un mensaje de aviso*/
111    printk(KERN_ALERT "PERIFERICO_irq: La interrupcion no ha sido habilitada. \n\r");
112    printk(KERN_ALERT "PERIFERICO_irq: ****************************************************************************************** \n\r");
113    printk(KERN_ALERT "PERIFERICO_irq: Esperando que suceda la interrupcion o que el usuario habilite el nivel de seguridad. \n\r");
114    printk(KERN_ALERT "PERIFERICO_irq: ****************************************************************************************** \n\r");
115  }
116
117  /*Mensaje de despedidad de la funcion irq_handler*/
118  printk(KERN_ALERT "PERIFERICO_irq: Estoy saliendo de la funcion PERIFERICO_irq. \n\n\r");
119  return IRQ_HANDLED;
120 }
121
122 /*********************************/
123 /*FUNCION DE INICIALIZACION init*/
124 /*********************************/
125 static int PERIFERICO_init(void)
126 {
127   /*Declaracion de variables usadas en la funcion init*/
128   int k;
129   int resultado;
130   int error;
131   unsigned int irq_llegada;
132   unsigned int irq_resultado;
133
134   /*Mensaje de apertura de la funcion init*/
135   printk(KERN_ALERT "************ PERIFERICO_init ************ \n\r");
136   printk(KERN_ALERT "PERIFERICO_init: Estoy ejecutando la funcion PERIFERICO_init. \n\r");
137
138   /*Parametros que se cargan en la funcion init*/
139   for (k=0; k<cuantos; k++)
140   {
141     printk(KERN_ALERT "PERIFERICO_init: Hola %s. \n\r", quien);
142   }
143
144   /*Reserva dinamica de los numeros Mayores y Menores*/
145   resultado = alloc_chrdev_region(&mydev, primermenor, cuenta, nombre);
146   if(resultado == 0)
147   {
```

```
148     printk(KERN_ALERT "PERIFERICO_init: Los numeros reservados fueron: \n# Mayor: %d. \n# Menor: %d. \n\r", MAJOR(mydev), MINOR(mydev));
149   } else {
150     printk(KERN_ALERT "PERIFERICO_init: Hubo un error y los numeros no se reservaron. El error fue: %d. \n\r", resultado);
151   }
152
153   /*Reserva de memoria para la estructura que representa al dispositivo de caracteres*/
154   my_PERIFERICO_dev = kmalloc(sizeof(struct PERIFERICO_dev), GFP_KERNEL);
155   if(my_PERIFERICO_dev==NULL)
156   {
157     printk(KERN_ALERT "PERIFERICO_init: No se puede reservar memoria, toca reiniciar. \n\r");
158   } else {
159     printk(KERN_ALERT "PERIFERICO_init: Se pudo reservar memoria, se reservaron %d bytes. \n\r", (int) sizeof(struct PERIFERICO_dev));
160   }
161
162   /*Inicializacion de la estructura que representa al dispositivo de caracteres*/
163   cdev_init(&my_PERIFERICO_dev->cdev,&PERIFERICO_fops);
164   my_PERIFERICO_dev->cdev.owner = THIS_MODULE;
165   my_PERIFERICO_dev->cdev.ops = &PERIFERICO_fops;
166   error = cdev_add(&my_PERIFERICO_dev->cdev, mydev, cuenta);
167   if(error < 0)
168   {
169     printk(KERN_ALERT "PERIFERICO_init: Hubo un error al registrar el dispositivo. El error fue: %d. \n\r", error);
170   } else {
171     printk(KERN_ALERT "PERIFERICO_init: Se registro exitosamente el dispositivo. \n\r");
172   }
173
174   /*Verificacion e instalacion del controlador de interrupcciones*/
175   printk(KERN_ALERT "PERIFERICO_init: Realizando el proceso de verificacion de la interrupcion. \n\r");
176   irq_llegada = gpio_to_irq(PERIFERICO_IRQ_PIN);
177   irq_resultado = request_irq(irq_llegada, irq_handler, IRQF_DISABLED | IRQF_TRIGGER_RISING, "PERIFERICO-IRQ", NULL);
178   if(irq_resultado)
179   {
180     printk(KERN_ALERT "PERIFERICO_init: Se instalo correctamente el controlador de interrupciones. **irq_resultado: %u. \n\r", \
181     irq_resultado);
182   } else {
183     printk(KERN_ALERT "PERIFERICO_init: No se pudo instalar el controlador de interrupciones, el error es **irq_resultado: %u. \n\r", \
184     irq_resultado);
185   }
186
187   /*Inicializando bandera a cero, para indicar que el dispositivo no se ha abierto*/
188   bandera = 0;
189   printk(KERN_ALERT "PERIFERICO_init: Colocando bandera = %d, para indicar que el driver no se ha inicializado. \n\r", bandera);
190
191   /*Mensaje de despedida de la funcion init*/
192   printk(KERN_ALERT "PERIFERICO_init: Estoy saliendo de la funcion PERIFERICO_init. \n\n\r");
193   return resultado;
194 }
195
196 /*************************/
197 /*FUNCION DE LIMPIEZA exit*/
198 /*************************/
199 static void PERIFERICO_exit(void)
200 {
201   /*Menesaje de apertura de la funcion exit*/
202   printk(KERN_ALERT "************ PERIFERICO_exit ************ \n\r");
203   printk(KERN_ALERT "PERIFERICO_exit: Estoy ejecutando la funcion PERIFERICO_exit. \n\r");
204
```

```
205   /*Retiro del sistema del controlador de interrupcciones*/
206   free_irq(PERIFERICO_IRQ_PIN, NULL);
207   printk(KERN_ALERT "PERIFERICO_release: Retirando del sistema el controlador de interrupciones. \n\r");
208
209   /*Retiro del sistema la estructura que representa al dispositivo de caracteres*/
210   cdev_del(&my_PERIFERICO_dev->cdev);
211   printk(KERN_ALERT "PERIFERICO_release: Retirando del sistema la estructura de dispositivo. \n\r");
212
213   /*Liberacion de la porcion de memoria de la estructura que representa al dispositivo de caracteres*/
214   kfree(my_PERIFERICO_dev);
215   printk(KERN_ALERT "PERIFERICO_release: Liberando el espacio de memoria de la estructura de dispositivo. \n\r");
216
217   /*Liberacion de los numeros Mayores y Menores reservados dinamicamente*/
218   unregister_chrdev_region(mydev,cuenta);
219   printk(KERN_ALERT "PERIFERICO_release: Liberando los numeros de dispositivo. \n\r");
220
221   /*Liberacion de la porcion de memoria asignada en la funcion open para el intercambio de datos*/
222   if(&my_PERIFERICO_dev->intercambio_de_datos != NULL && bandera==1)
223   {
224     kfree(my_PERIFERICO_dev->intercambio_de_datos);
225     printk(KERN_ALERT "PERIFERICO_release: Liberando el espacio de memoria para el intercambio_de_datos. \n\r");
226   }
227
228   /*Mensaje de despedidad de la funcion exit*/
229   printk(KERN_ALERT "PERIFERICO_exit: Adios mundo cruel. \n\r ");
230   printk(KERN_ALERT "PERIFERICO_exit: Estoy saliendo de la funcion PERIFERICO_exit. \n\n\r");
231 }
232
233 /*************************/
234 /*LLAMADA AL SISTEMA open*/
235 /*************************/
236 static int PERIFERICO_open(struct inode *puntero_inode, struct file *puntero_file)
237 {
238   /*Declaracion de variables usadas en la funcion open*/
239   int tamano;
240   int k;
241
242   /*Declracion de un punetro que apunta a la estructura que representa al dispositivo de caracteres*/
243   struct PERIFERICO_dev *el_PERIFERICO;
244
245   /*Mensaje de apertura de la funcion open*/
246   printk(KERN_ALERT "************ PERIFERICO_open ************ \n\r");
247   printk(KERN_ALERT "PERIFERICO_open: Estoy ejecutando la funcion PERIFERICO_open. \n\r");
248   printk(KERN_ALERT "PERIFERICO_open: La informacion de puntero_inode es: 0x%X. \n\r", (unsigned int)puntero_inode);
249   printk(KERN_ALERT "PERIFERICO_open: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
250
251   /*Proceso que verifica si la estructura de dispositivo es adecuada*/
252   el_PERIFERICO = container_of(puntero_inode->i_cdev, struct PERIFERICO_dev, cdev);
253   printk(KERN_ALERT "PERIFERICO_open: La informacion de container_of es: 0x%X. \n\r", (unsigned int)puntero_inode->i_cdev);
254
255   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a open*/
256   puntero_file->private_data = el_PERIFERICO;
257
258   if(bandera==0)
259   {
260     /*Inicializando los campos de la estructura que representa al dispositivo de carcacteres*/
261     printk(KERN_ALERT "PERIFERICO_open: Inicializando los campos de PERIIFERICO_dev. bandera= %d. \n\r", bandera);
```

```
262
263     strcpy(el_PERIFERICO->nombre_periferico, PERIFERICO_NAME);
264     el_PERIFERICO->DireccionBase_datos = PERIFERICO_BASEADDRESS_DATA;
265     el_PERIFERICO->numero_de_registros_datos = PERIFERICO_NUMERO_REGISTROS_DATA;
266     tamano = el_PERIFERICO->numero_de_registros_datos;
267     el_PERIFERICO->intercambio_de_datos = kmalloc(tamano*sizeof(char), GFP_KERNEL);
268
269     for(k=0; k<tamano; k++)
270     {
271       el_PERIFERICO->intercambio_de_datos[k] = 0;
272     }
273
274     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->nombre_periferico: %s. \n\r", el_PERIFERICO->nombre_periferico);
275     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->DireccionBase_data: 0x%X. \n\r", el_PERIFERICO->DireccionBase_datos);
276     printk(KERN_ALERT "PERIFERICO_open: el_PERIFERICO->numero_de_registros_datos: %d. \n\r", el_PERIFERICO->numero_de_registros_datos);
277     printk(KERN_ALERT "PERIFERICO_open: La reserva de memoria para el_PERIFERICO->intercambio_de_datos es de: %u bytes. \n\r", \
278     tamano*sizeof(char));
279
280     /*Colocando la bandera a uno para indicar que el dispositivo ya ha sido abierto*/
281     bandera=1;
282
283   } else {
284     printk(KERN_ALERT "PERIFERICO_open: Ya estaban inicializados los campos de PERIFERICO_dev. bandera= %d. \n\r", bandera);
285   }
286
287   /*Se le indica al nucleo que se se desea hacer uso de puertos de E/S*/
288   printk(KERN_ALERT "PERIFERICO_open: Solicitud de region de meoria de puertos E/S con request_mem_region(). \n\r");
289   if(!request_mem_region(PERIFERICO_BASEADDRESS_DATA, PERIFERICO_MEMSPACE_SIZE_DATA, PERIFERICO_NAME))
290   {
291     printk(KERN_ALERT "PERIFERICO_open: No se pudo realizar la asignacion de memoria de puertos E/S para el periferico. \n\r");
292   } else {
293     printk(KERN_ALERT "PERIFERICO_open: Se realizo la asignacion de memoria de puertos E/S para el periferico. \n\r");
294
295     /*Realizando la asignacion y direccionamiento virtual de los puerto de E/S*/
296     printk(KERN_ALERT "PERIFERICO_open: Realizando el mapeo de memoria en bytes de los puertos E/S con ioremap. \n\r");
297     PERIFERICO_iomem_pointer_data = ioremap(PERIFERICO_BASEADDRESS_DATA, PERIFERICO_MEMSPACE_SIZE_DATA);
298     printk(KERN_ALERT "PERIFERICO_open: El puntero devuelto por ioremap es: 0x%X. \n\r", (unsigned int)PERIFERICO_iomem_pointer_data);
299   }
300
301   /*Mensaje de despedidad de la funcion open*/
302   printk(KERN_ALERT "PERIFERICO_open: Estoy saliendo de la funcion PERIFERICO_open. \n\n\r");
303   return 0;
304 }
305
306 /*****************************/
307 /*LLAMADA AL SISTEMA release*/
308 /*****************************/
309 static int PERIFERICO_release(struct inode *puntero_inode, struct file *puntero_file)
310 {
311   /*Mensaje de apertura de la funcion release*/
312   printk(KERN_ALERT "************ PERIFERICO_release ************ \n\r");
313   printk(KERN_ALERT "PERIFERICO_release: Estoy ejecutando la funcion PERIFERICO_release. \n\r");
314   printk(KERN_ALERT "PERIFERICO_release: La informacion de puntero_inode es: 0x%X. \n\r", (unsigned int)puntero_inode);
315   printk(KERN_ALERT "PERIFERICO_release: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
316
317   /*Liberacion de los puertos E/S virtualizados*/
318   printk(KERN_ALERT "PERIFERICO_release: Liberando el mapeo de memoria con iounmap. \n\r");
```

```
319   iounmap(PERIFERICO_iomem_pointer_data);
320
321   /*Liberacion de los puertos de E/S desde el nucleo*/
322   printk(KERN_ALERT "PERIFERICO_release: Liberando la solicitud de region de memoria para los puertos E/S con release_mem_region. \n\r");
323   release_mem_region(PERIFERICO_BASEADDRESS_DATA, PERIFERICO_MEMSPACE_SIZE_DATA);
324
325   /*Mensaje de despedidad de la funcion release*/
326   printk(KERN_ALERT "PERIFERICO_release: Estoy saliendo de la funcion PERIFERICO_release. \n\n\r");
327   return 0;
328 }
329
330 /*************************/
331 /*LLAMADA AL SISTEMA read*/
332 /*************************/
333 static ssize_t PERIFERICO_read(struct file *puntero_file, char __user *puntero_usuario, size_t tamano, loff_t *puntero_offset)
334 {
335   /*Declaracion de variables usadas en la funcion read*/
336   ssize_t retval = 0;
337   char leido;
338
339   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a read*/
340   struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
341
342   /*Declaracion de una variable que toma la informacion de un campo de la estructura que representa al dispositivo de caracteres*/
343   char cantidad_de_registros_datos = el_PERIFERICO->numero_de_registros_datos;
344
345   /*Mensaje de apertura de la funcion read*/
346   printk(KERN_ALERT "************ PERIFERICO_read ************ \n\r");
347   printk(KERN_ALERT "PERIFERICO_read: Estoy ejecutando la funcion PERIFERICO_read. \n\r");
348   printk(KERN_ALERT "PERIFERICO_read: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
349   printk(KERN_ALERT "PERIFERICO_read: La informacion de puntero_usuario es: 0x%X. \n\r", (unsigned char)puntero_usuario[0]);
350   printk(KERN_ALERT "PERIFERICO_read: La informacion de tamano es: %u bytes. \n\r", (unsigned int)tamano);
351   printk(KERN_ALERT "PERIFERICO_read: La informacion de puntero_offset es: %u bytes. \n\r", (unsigned int)*puntero_offset);
352
353   /*Configurando tamano al maximo valor de lectura*/
354   printk(KERN_ALERT "PERIFERICO_read: Configurando la cantidad de datos que se pueden leer. \n\r");
355   if(tamano >= PERIFERICO_MEMSPACE_SIZE_DATA)
356   {
357     tamano = PERIFERICO_MEMSPACE_SIZE_DATA;
358   }
359
360   /*Acotando el valor de retorno hasta el maximo de informacion que se puede leer*/
361   if((*puntero_offset + tamano) > cantidad_de_registros_datos*1)
362   {
363     retval = cantidad_de_registros_datos*1 - *puntero_offset;
364     printk(KERN_ALERT "PERIFERICO_read: La lectura que se pide sobrepasa la cantidad de registros del periferico. \n\r");
365     printk(KERN_ALERT "PERIFERICO_read: Leyendo los datos que se pueden hasta llegar al final. La lectura es de = %d bytes. \n\r", \
366     retval);
367   } else {
368     retval = tamano;
369     printk(KERN_ALERT "PERIFERICO_read: La lectura que se pide no sobrepasa la cantidad de registros del periferico \n\r");
370     printk(KERN_ALERT "PERIFERICO_read: Leyendo la cantidad de datos solicitados. La lectura es de = %d bytes. \n\r", retval);
371   }
372
373   /*Realizando el proceso de lectura en el sentido periferico-driver-usuario*/
374   printk(KERN_ALERT "PERIFERICO_read: Realizando el traspaso de datos de interrupciones al usuario. \n\r");
375   if(retval > 0)
```

```
376   {
377     /*Leyendo las interrupciones administradas por la funcion handler*/
378     leido = irq_contador;
379     *el_PERIFERICO->intercambio_de_datos = leido;
380     printk(KERN_ALERT "PERIFERICO_read: Se envio al usuario las interrupciones. Estas son: 0x%X. \n\r", \
381     el_PERIFERICO->intercambio_de_datos[0]);
382 
383     /*Envio de la informacion leida desde el espacio del nucleo al espacio del usuario*/
384     printk(KERN_ALERT "PERIFERICO_read: Realizando el traspaso de datos desde el nucleo al usuario. \n\r");
385     if(copy_to_user(puntero_usuario, el_PERIFERICO->intercambio_de_datos, retval))
386     {
387       printk(KERN_ALERT "PERIFERICO_read: No se ha recibido la imformacion desde el nucleo en el usuario. \n\r");
388     } else {
389       printk(KERN_ALERT "PERIFERICO_read: Se recibio la informacion desde el nucleo en el usuario. \n\r");
390 
391       /*Actualizacion del puntero del archivo si el traspaso nucleo-usuario fue exitoso*/
392       printk(KERN_ALERT "PERIFERICO_read: Actualizando el puntero_offset. \n\r");
393       *puntero_offset += retval;
394       printk(KERN_ALERT "PERIFERICO_read: Al salir de la funcion read, puntero_offset: %u bytes. \n\r", (unsigned int)*puntero_offset);
395     }
396   }
397 
398   /*Mensaje de despedidad de la funcion read*/
399   printk(KERN_ALERT "PERIFERICO_read: Estoy saliendo de la funcion PERIFERICO_read. \n\n\r");
400   return retval;
401 }
402 
403 /*************************/
404 /*LLAMADA AL SISTEMA write*/
405 /*************************/
406 static ssize_t PERIFERICO_write(struct file *puntero_file, const char __user *puntero_usuario, size_t tamano, loff_t *puntero_offset)
407 {
408   /*Declaracion de variables usadas en la funcion write*/
409   ssize_t retval;
410   unsigned char escrito;
411 
412   /*El puntero se lleva hacia el campo private_data para permitir un acceso mas facil en la proxima llamada a write*/
413   struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
414 
415   /*Mensaje de apertura de la funcion write*/
416   printk(KERN_ALERT "************ PERIFERICO_write ************ \n\r");
417   printk(KERN_ALERT "PERIFERICO_write: Estoy ejecutando la funcion PERIFERICO_write. \n\r");
418   printk(KERN_ALERT "PERIFERICO_write: La informacion de puntero_file es: 0x%X. \n\r", (unsigned int)puntero_file);
419   printk(KERN_ALERT "PERIFERICO_write: La informacion de puntero_usuario es: 0x%X. \n\r", (unsigned char)puntero_usuario[0]);
420   printk(KERN_ALERT "PERIFERICO_write: La informacion de tamano es: %u bytes. \n\r", (unsigned int)tamano);
421   printk(KERN_ALERT "PERIFERICO_write: La informacion de puntero_offset es: %u bytes. \n\r", (unsigned int)*puntero_offset);
422 
423   /*Acotando el valor de retorno hasta el maximo de informacion que se puede escribir*/
424   printk(KERN_ALERT "PERIFERICO_write: Configurando la transferencia de datos que se pueden escribir. \n\r");
425   if(tamano > PERIFERICO_MEMSPACE_SIZE_DATA)
426   {
427     retval = PERIFERICO_MEMSPACE_SIZE_DATA;
428   } else {
429     retval = tamano;
430   }
431 
432   /*Realizando el proceso de escritura desde el espacio del usuario al espacio del nucleo*/
```

```
433   printk(KERN_ALERT "PERIFERICO_write: Realizando el traspaso de datos desde el usuario al nucleo. \n\r");
434   if(copy_from_user(el_PERIFERICO->intercambio_de_datos, puntero_usuario, retval))
435   {
436     printk(KERN_ALERT "PERIFERICO_write: No se ha recibido informacion del usuario en el nucleo. \n\r");
437   } else {
438     printk(KERN_ALERT "PERIFERICO_write: Se recibio informacion del usuario en el nucleo. Esta es: 0x%X. \n\r", \
439     el_PERIFERICO->intercambio_de_datos[0]);
440
441     /*Verificando el nivel de seguridad de interrupcion, administrada por el usuario*/
442     printk(KERN_ALERT "PERIFERICO_write: Nivel de seguridad que verifica si la interrupcion esta habilitada/desabilitada. \n\r");
443     escrito = *el_PERIFERICO->intercambio_de_datos;
444     if(escrito==1)
445     {
446       irq_habilitador = 1;
447       printk(KERN_ALERT "PERIFERICO_write: El usuario ha habilitado la interrupcion. \n\r");
448     } else {
449       if(escrito==0)
450       {
451         irq_habilitador = 0;
452         printk(KERN_ALERT "PERIFERICO_write: El usuario ha deshabilitado la interrupcion. \n\r");
453       }
454     }
455
456     /*Actualizacion del puntero del archivo si el traspaso usuario-nucleo-periferico fue exitoso*/
457     printk(KERN_ALERT "PERIFERICO_write: Actualizando el puntero_offset. \n\r");
458     *puntero_offset += retval;
459     printk(KERN_ALERT "PERIFERICO_write: Al salir de la funcion write, puntero_offset: %u bytes. \n\r", (unsigned int)*puntero_offset);
460   }
461
462   /*Mensaje de despedidad de la funcion write*/
463   printk(KERN_ALERT "PERIFERICO_write: Estoy saliendo de la funcion PERIFERICO_write. \n\n\r");
464   return retval;
465 }
466
467 /**************************************************************************************/
468 /*FUNCIONES QUE SE EJECUTAN CUANDO SE REALIZA DESDE EL ESPACIO DEL USUARIO insmod y rmmod*/
469 /**************************************************************************************/
470 module_init(PERIFERICO_init);
471 module_exit(PERIFERICO_exit);
472
473 /*********************************/
474 /*INFORMACION ADICIONAL DEL DRIVER*/
475 /*********************************/
476 MODULE_LICENSE("Dual BSD/GPL");
477 MODULE_AUTHOR("Arides Jose Meneses Hernandez <ajomeher@hotmail.com>");
478 MODULE_ALIAS("Plantilla Periferico_PERIFERICO");
479 MODULE_DESCRIPTION("Driver para administrar Interrupciones. Adaptado de la plnatilla del profesor: Andres Calderon <andresn@emqbit.com>");
480 MODULE_VERSION("1:0.0");
```

I.2 Aplicación en el espacio del usuario

I.2.1 aplicacion.h

```
/*****************************************************************/
/***VALORES PARA HABILITAR Y DESHABILITAR EL NIVEL DE SEGURIDAD***/
/*****************************************************************/

/* Estos valores corresponden a la seguridad que el usuario maneja a *
 * la hora de trabajar con unterrupciones.                           *
 */
#define HABILITA_SEGURIDAD_USUARIO (1)
#define DESHABILITA_SEGURIDAD_USUARIO (0)

/*************/
/***TAMANOS***/
/*************/

/* Tamano es la cantidad de bytes a leer/escribir. *
 */
#define TAMANO_BYTES_CHAR    (1)
#define TAMANO_BYTES_INT     (4)
#define TAMANO_BYTES_DOUBLE (8)

/*******************************/
/***DECLARACION DE FUNCIONES***/
/*******************************/

/* Funcion PERIFERICO_INTERRUPCIONES_leer                    *
 * tamano : Es la cantidad de bytes que se tomaran del dato. *
 */
unsigned char PERIFERICO_INTERRUPCIONES_leer(char tamano);

/* Funcion PERIFERICO_INTERRUPCIONES_habilitar                *
 * dato : Es el valor a escribir en el periferico.           *
 * tamano : Es la cantidad de bytes que se tomaran del dato. *
 */
int PERIFERICO_INTERRUPCIONES_escribir(unsigned char dato, char tamano);
```

I.2.2 aplicacion.c

```
 #include <fcntl.h>
#include "aplicacion.h"

/* En esta rutina se ejecutan las llamadas al sistema open, read y close. */
unsigned char PERIFERICO_INTERRUPCIONES_leer(char tamano)
{
  int file_descriptor;

  unsigned char buffer[0];

  file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);

  read(file_descriptor, buffer, tamano);
```

```
14
15   close(file_descriptor);
16
17   return buffer[0];
18 }
19
20 /* En esta rutina se ejecutan las llamadas al sistema open, write y close. */
21 int PERIFERICO_INTERRUPCIONES_escribir(unsigned char dato, char tamano)
22 {
23   int file_descriptor;
24
25   file_descriptor = open("/dev/Periferico_PERIFERICO", O_RDWR | O_SYNC);
26
27   write(file_descriptor, &dato, tamano);
28
29   close(file_descriptor);
30
31   return 1;
32 }
```

I.2.3 prueba.c

```
1  #include <stdio.h>
2  #include "aplicacion.h"
3  #include "aplicacion.c"
4
5  int main()
6  {
7    int opcion;
8    unsigned char leido[0];
9
10   printf("APLICACION: SELECCIONAR LA OPERACION A RELIZAR. \n");
11   printf("APLICACION: 0. Habilitar el nivel de seguridad de interrupcion. \n");
12   printf("APLICACION: 1. Deshabilitar el nivel de seguridad de interrupcion. \n");
13   printf("APLICACION: 2. Lerr la cantidad de interrupciones hasta el momento. \n");
14   scanf("%d", &opcion);
15
16   if(opcion == 0)
17   {
18     PERIFERICO_INTERRUPCIONES_escribir(HABILITA_SEGURIDAD_USUARIO, TAMANO_BYTES_CHAR);
19   }
20
21   if(opcion == 1)
22   {
23     PERIFERICO_INTERRUPCIONES_escribir(DESHABILITA_SEGURIDAD_USUARIO, TAMANO_BYTES_CHAR);
24   }
25
26   if(opcion == 2)
27   {
28     leido[0] = PERIFERICO_INTERRUPCIONES_leer(TAMANO_BYTES_CHAR);
29     printf("APLICACION: Las interrupciones hasta el momento son: 0x%x. \n", leido[0]);
30   }
31
32   if(opcion > 2)
33   {
```

```
34      printf("APLICACION: Esta no es una opccion. \n");
35      printf("APLICACION: Vuelva a ejecutar la aplicacion. \n\r");
36    }
37
38    return 0;
39 }
```