Trabajo de Fin de Grado
Grado en Ingenieria de las Tecnologias de Telecomunicación

Intensificación en Sistemas Electrónicos

Uso de Xillybus con ZedBoard

Autor: Daniel Fernández Cuadrado

Tutor: Fernando Muñoz Chavero

Dpto. Ingeniería Eletrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Grado

Grado en Ingeniería en Tecnologías de la Telecomunicación

Uso de Xillybus en ZedBoard

Autor:

Daniel Fernández

Tutor:

Fernando Muñoz Chavero

Profesor titular

Dep. de Teoría de Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo de Fin de Grado: Uso de Xillybus en ZedBoard.

Autor: Daniel Fernández Cuadrado

Tutor: Fernando Muñoz Chavero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

Quiero agradecer a mi tutor Fernando Muñoz Chavero su ayuda y guía a lo largo del proyecto, a todos los profesores a lo largo de la carrera que me han permitido llegar hasta aquí y me han enseñado tanto.

Mencionar a la gente que me ha acogido en Sevilla desde que llegué cuando pensé que no me quedaría, en especial a Rosa Lostes García, y Rafa Martín Ruiz.

También a mis amigos de toda la vida que me han ayudado a aguantar toda la presión y los malos momentos a lo largo de la carrera, Jean Erik Marchal de la Mata, Antonio Rodriguez Trimiño, Cristobal Ismael Montero, Paula María Camino Yuste...

Por último a mi familia y agradecerles permitirme estudiar en Sevilla, estos 5 años.

Resumen

El presente trabajo busca abordar el uso académico de la nueva placa ZedBoard y proporcionar una breve explicación sobre la misma. En un primer momento nos centraremos en la preparación de los elementos de desarrollo para poder realizar un proyecto en vhdl. Dichos elementos serán Vivado 2016.4 y los necesarios para conectarnos a la ZedBoard por puerto Serie (en este caso Ubuntu, la máquina virtual y minicom).

Posteriormente desarrollaremos dos tutoriales con la finalidad de estudiar el empleo de device file, así como la comunicación entre el host y la PL a través de Xillybus. El primer tutorial está destinado a que el usuario se inicie en el uso del device file de lectura del Xillybus, (PL - host); en el segundo se usará el device file de escritura de Xillybus (host -PL), además de incidir en el tema tratado anteriormente.

En ambos tutoriales se incluirá el código, se analizará y se demostrará su funcionamiento mediante test_bench y también en FPGA.

Palabras clave: Device file, Xillybus, tutorial, Zedboard, FPGA.

Abstract

The objective of this work is to tackle the academic use of the new board ZedBoard and provide a brief explanation about it. Firstly, we will focus on the preparation of the developing elements for doing a project in vhdl. These elements will be Vivado 2016.4 and the others necessaries for connecting to ZedBoard by Serie port (in this case Ubuntu, virtual machine and minicom).

Later we will develop two tutorials with the objective of studying the employment of device file, and the communication between host and PL through Xillybus. First tutorial has the purpose of starting users in the use of the reading device file of Xillybus, (PL – host); second tutorial will use writing device file of Xillybus (host – PL), and it will stress in the matter we have explained.

Both tutorials will include the code and we analyze and demonstrate their running by test_bench and also in FPGA.

Keywords: Device file, Xyllibus, tutorial, Zedboard, FPGA.

Índice

ÍNDICE DE FIGURAS

Notación

Ac	Alternanting Current
ADC	Analogic-to-Digital Converter

AXI	Advanced eXtensible Interface
DDR3	Double Data Rate type three
DMA	Direct Memory Acces
EOF	End-Of-File
FIFO	Firs In, Firs Out
FPGA	Field Programmable Gate Array
GAL	Generic Array Logic
GPIO	General-Purpose Input/Output
HDMI	High Definition Multimedia Interface
HW	Hardware
I/O	Input/Output
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
SD	Secure Digital
tb	test bench

1 INTRODUCCIÓN

1.1 Objetivos

El objetivo del trabajo es realizar una serie de tutoriales de vhdl, usando la placa VHDL, desde 0, para que un usuario pueda adquirir cierto conocimiento sobre las interfaces de interconexión de Xillybus con ZedBoard.

Antes de los tutoriales se proporcionan unas guías para preparar todas las herramientas necesarias para su correcto desarrollo y su comprobación:

- Vivado 2016.4
- VMware Workstation 12 Player
- Xillinux
- Image Tool

Los tutoriales que se realizarán consistirán en :

- Tutorial A basado en el device file lectura, viendo distintas opciones que podemos introducir.
- tutorial B usando device file de escritura y de lectura, y procesando los datos que escribimos.

1.2 Estructura del proyecto

La estructura del proyecto consistirá en 3 partes:

- Breve análisis sobre la placa de desarrollo.
- Herramientas de desarrollo.
- Análisis Xillybus.
- Tutoriales A y B.

2 ZEDBOARD

La ZedBoard es una placa de evaluación y desarrollo basada en Xilinx ZynqTM-7000 All Programmable SoC (AP SoC). Combinado con un dual Corex-A9 Processing System (PS) con 85,000 celdas Series-7 Programmable Logic (PL) puede usarse para multitud de aplicaciones. [1]

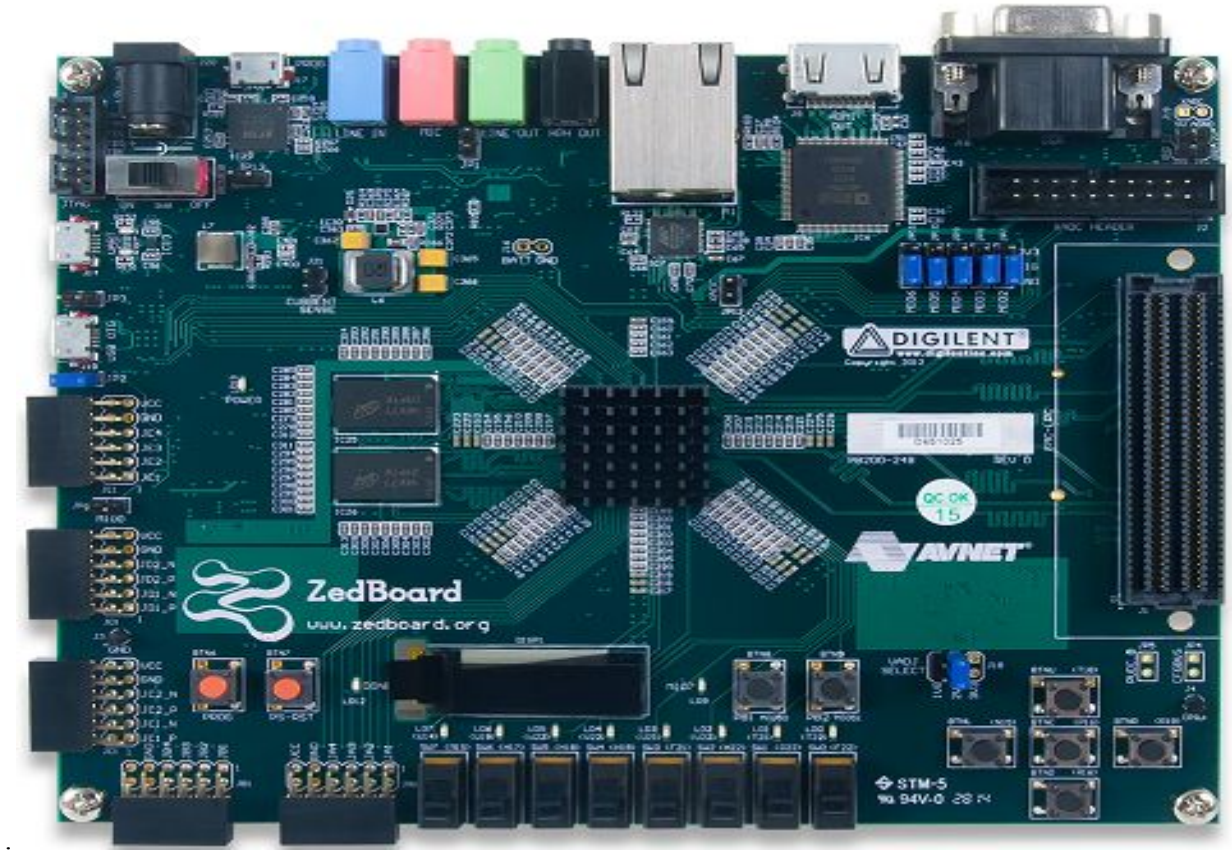

Figura 1. Placa de desarrollo ZedBoard.

2.1 Características

- **Xilinx® XC7Z020-1CLG484C Zynq-7000 AP SoC.**

  Primary configuration = QSPI Flash.

  - Auxiliary configuration options.
    - Cascaded JTAG.
    - SD Card.

- **Memory**.
  - 512 MB DDR3 (128M x 32).
  - 256 Mb QSPI Flash.
- **Interfaces**.
  - USB-JTAG Programming using Digilent SMT1-equivalent circuit.
    - Accesses PL JTAG.
    - PS JTAG pins connected through PS Pmod.
  - 10/100/1G Ethernet.
  - USB OTG 2.0.
  - SD Card .
  - USB 2.0 FS USB-UART bridge .
  - Five Digilent Pmod™ compatible headers (2x6) (1 PS, 4 PL) .
  - One LPC FMC .
  - One AMS Header .
  - Two Reset Buttons (1 PS, 1 PL) .
  - Seven Push Buttons (2 PS, 5 PL).
  - Eight dip/slide switches (PL) .
  - Nine User LEDs (1 PS, 8 PL) .
  - DONE LED (PL) .
- **On-board Oscillators.**
  - 33.333 MHz (PS) .
  - 100 MHz (PL) .
- **Display/Audio .**
  - HDMI Output.
  - VGA (12-bit Color).

- 128x32 OLED Display.
- Audio Line-in, Line-out, headphone, microphone.

- **Power .**
  - On/Off Switch .
  - 12V @ 5A AC/DC regulator .
- **Software** .
  - ISE® WebPACK Design Software .
  - License voucher for ChipScope™ Pro locked to XC7Z020 .

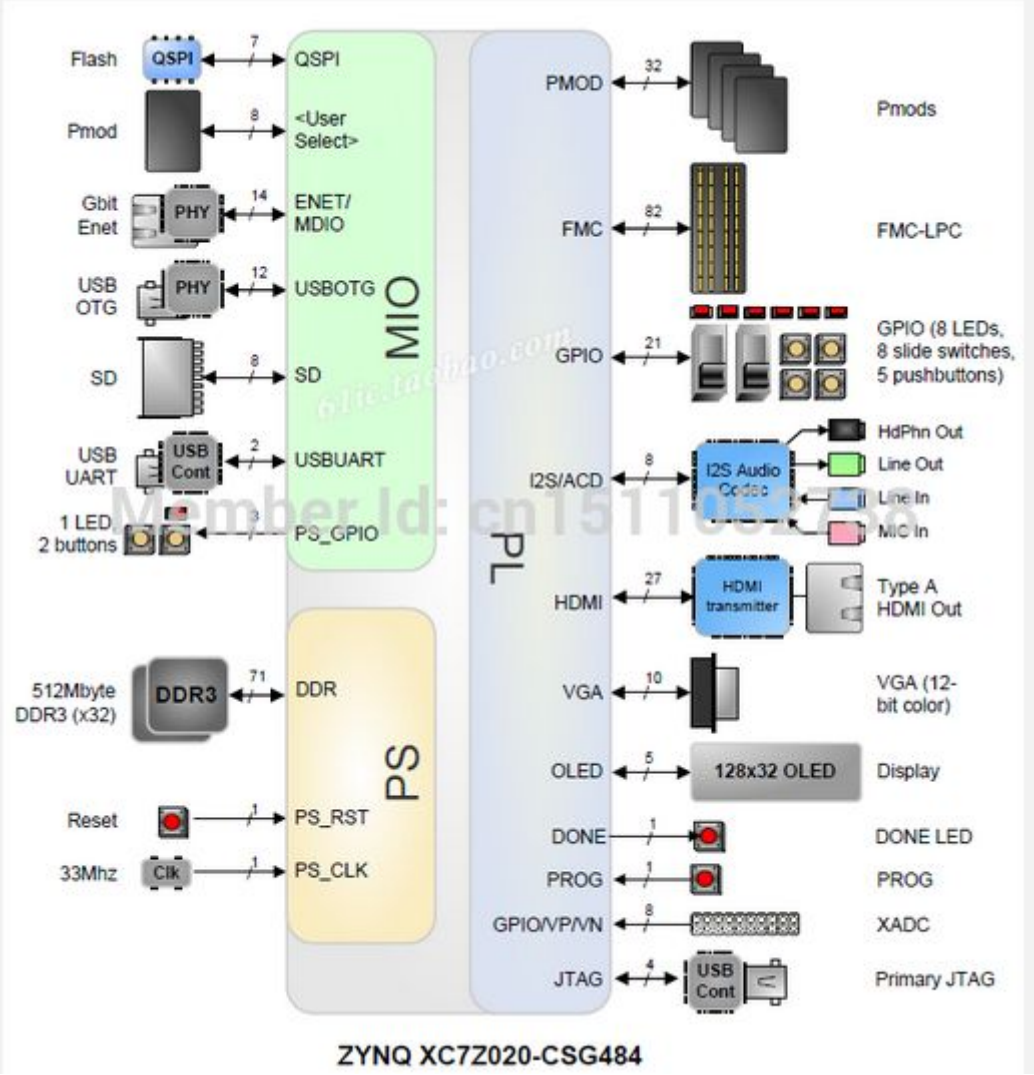

Figura 2. Diagrama de bloques de la ZedBoard.

2.2 Componentes.

2.2.1 Memoria.

2.2.1.1 DDR3.

La ZedBoard incluye dos módulos Microm DDR3 128 Megabit *16, formando una interfaz de 32 bits y una memoria total de 512 MB.[1]

El controlador de memoria DDR multiprotocolo está configurado para accesos amplios de 32 bits a un espacio de direcciones de 512 MB. El PS incorpora tanto el controlador DDR como el PHY asociado, incluyendo su propio conjunto de E / S dedicadas. Las velocidades de interfaz de memoria DDR3 de hasta 533 Mhz (1066 Mbs) son compatibles.

2.2.1.2 QSPI Flash

Se dispone de un quad-SPI NOR flash, el Spansion S25FL256S en la ZedBoard. La memoria Flash Multi-I/O SPI se utiliza para proporcionar código no volátil y almacenamiento de datos. Se puede emplear para inicializar el subsistema PS y configurar el subsistema PL (bitstream). Spansion proporciona Spansion Flash File System (FFS) de cara a su uso después de arrancar el Zynq-7000 AP SoC.[1]

2.2.2 Tarjeta SD

El Zynq PS SD / SDIO controla la comunicación con la tarjeta SD ZedBoard. La tarjeta SD se puede utilizar para almacenar la memoria externa no volátil, así como arrancar el Zynq-7000 AP SoC. El PS periférico sd0 está conectado a través del Banco 1/501 MIO, incluyendo. Detección de tarjetas y protección contra escritura.

La tarjeta SD de ZedBoard se conecta a través de un conector de tarjeta SD estándar de 9 pines, J12, TE 2041021-1. Se recomienda una tarjeta de Clase 4 o mejor (el JP6 debe estar cerrado).[1]

2.2.3 USB

2.2.3.1 USB OTG

ZedBoard implementa una de las dos interfaces USB PS OTG disponibles. Se requiere un PHY externo con una interfaz ULPI de 8 bits. Se utiliza un chip transceptor USB TI TUSB1210 autónomo como PHY. El PHY cuenta con un completo HS-USB Physical front-end de velocidades de apoyo de hasta 480Mbs.

2.2.3.2 Puente USB-UART

El ZedBoard implementa un puente USB-a-UART conectado a un periférico PS UART. Un dispositivo puente Cypress CY7C64225 USB-to-UART permite la conexión a un PC host. El dispositivo USB / UART se conecta al USB Micro B, J14, (TE 1981584-1) de la placa. Sólo se implementa la conexión TXD / RXD básica. Si se requiere control de flujo, se puede añadir a través de MIO extendido en un PL-Pmod ™.[1]

Cypress proporciona controladores virtuales de Puerto COM (VCP) libres que permiten que el puente USB-UART CY7C64225 aparezca como un puerto COM en el pc, para alojar software de comunicaciones.

2.2.3.3 USB-JTAG

El ZedBoard proporciona funcionalidad JTAG basada en el Módulo JTAG de Alta Velocidad USB Digilent, dispositivo SMT1. Este circuito USB-JTAG está totalmente soportado e integrado en las herramientas Xilinx ISE, incluyendo iMPACT, ChipScope y SDK Debugger. El JTAG está disponible a través de un conector USB Micro B, J17, TE 1981568-1.[1]

2.2.4 Audio y Display

2.2.4.1 HDMI

Un Analog Devices ADV 7511 Transmisor HDMI ofrece una interfaz de vídeo digital para el ZedBoard. Este transmisor de 225 MHz es compatible con HDMI 1.4- y DVI 1.0 compatible con 1080p60 con 16 bits, YCbCr, color 4: 2: 2 modo.[1]

2.2.4.2 VGA

El ZedBoard también permite una salida de vídeo en color de 12 bits a través de un conector VGA a través del orificio TE 4-1734682-2. Cada color se crea a partir de la resistencia-escalera de cuatro pines PL.[1]

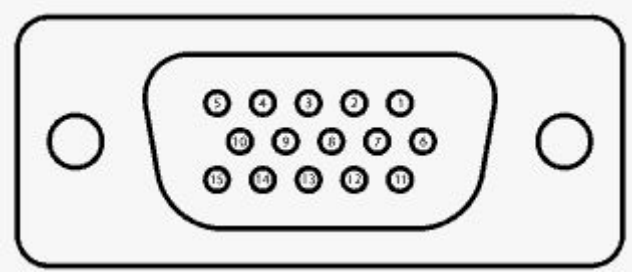

Figura 3. Conector VGA.

2.2.4.3 Codec de Audio I2S

Analog Devices ADAU1761 Audio Codec proporciona procesamiento de audio digital integrado al Zynq-7000 AP SoC. Permite grabación y reproducción estéreo de 48 KHz. Las tasas de muestreo de 8KHz a 96KHz son compatibles. Además, el ADAU1761 ofrece control de volumen digital. El Codec se puede configurar utilizando Analog Devices SigmaStudio ™ para optimizar el audio en aplicaciones acústicas específicas, numerosos filtros, algoritmos y mejoras. Analog Devices dispone de controladores Linux para este dispositivo.

2.2.4.4 Display OLED

Una pantalla OLED Inteltronic / Wisechip UG-2832HSWEG04 se utiliza en el ZedBoard. Esto nos permite usar una pantalla monocromática de matriz pasiva de 128 x 32 píxeles. El tamaño de la pantalla es de 30 mm x 11,5 mm x 1,45 mm.

2.2.5 Fuentes de Relojes

El subsistema PS Zynq-7000 AP SoC utiliza una fuente de reloj dedicada 33.3333 MHz, IC18, Fox 767-33.333333-12, con terminación en serie. La infraestructura de PS puede generar hasta cuatro relojes basados en PLL para el sistema PL. Un oscilador a bordo de 100 MHz, IC17, Fox 767-100-136, suministra la

entrada de reloj del subsistema PL en el banco 13, pin Y9.

2.2.6 Resets

2.2.6.1 Power-on Reset (PS_POR_B)

El Zynq PS admite señales de reinicio de la alimentación externa. El reset de encendido es el reinicio de todo el chip. Esta señal restablece cada registro en el dispositivo . ZedBoard dirige esta señal desde un comparador que mantiene el sistema en reposición hasta que todas las fuentes de alimentación sean válidas. Varios otros ICs en ZedBoard son restablecidos por esta señal también.

2.2.6.2 Reset de pulsador

Un interruptor PROG, BTN6, activa Zynq PROG_B. Esto inicia la reconfiguración de la subsección PL por el procesador.

2.2.6.3 PS Reset

El reinicio de encendido, denominado PS_RST / BTN7, borra todas las configuraciones de depuración. El reset del sistema externo permite al usuario restablecer toda la lógica funcional dentro del dispositivo sin alterar el entorno de depuración. Por ejemplo, los puntos de interrupción anteriores establecidos por el usuario permanecen válidos después del reinicio del sistema. Debido a problemas de seguridad, el reanudamiento del sistema borra todo el contenido de memoria dentro del PS, incluyendo el OCM. El PL también se restablece en el reinicio del sistema. La reinicialización del sistema no vuelve a muestrear los pasadores de flejado del modo de arranque.

2.2.7 Entrada / Salidas

2.2.7.1 Pulsadores

El ZedBoard proporciona 7 botones GPIO de usuario al Zynq-7000 AP SoC; Cinco en el lado PL y dos en el lado PS.

2.2.7.2 Switches

El ZedBoard tiene ocho conmutadores DIP de usuario, SW0-SW7, que proporcionan la entrada del usuario. Los interruptores SPDT conectan la E / S a través de una resistencia de 10kΩ a la fuente de tensión VADJ o GND.

2.2.7.3 LED's

El ZedBoard tiene ocho LEDs de usuario, LD0 – LD7.

2.2.8 10/100/1000 Ethernet PHY

El ZedBoard implementa un puerto Ethernet 10/100/1000 para la conexión de red usando un Marvell 88E1518 PHY. Esta parte funciona a 1.8V. El PHY se conecta al MIO Bank 1/501 (1.8V) e interfiere con el Zynq-7000 AP SoC vía RGMII. El conector RJ-45 es un TE Connectivity 1840750-7 con magnéticos integrados. El RJ-45 tiene dos LED indicadores de estado que indican tráfico y estado de enlace válido.

2.2.9 Conectores expansivos

2.2.9.1 Conectores LPC FMC

En el ZedBoard se proporciona una sola ranura FMC de conector de bajo número (LPC) para soportar un ecosistema grande de módulos plug-in. El LPC FMC expone 68 E / S de un extremo, que pueden configurarse como 34 pares diferenciales. La interfaz FMC se extiende sobre dos bancos PL I / O, bancos 34 y 35. Para cumplir con la especificación FMC, estos bancos se alimentan desde un voltaje ajustable establecido por el puente J18. Los voltajes seleccionables incluyen 1.8V, por defecto, y 2.5V. También es posible ajustar Vadj a 3.3V. Dado que 3,3 V podría ser la configuración de voltaje más dañina para Vadj, esto no está disponible con el hardware de placa predeterminado. Para ajustar Vadj a 3,3 V, se debe soldar un corto a través de las almohadillas 3V3 en J18 o en una cabecera 1x2 adicional. El pin FMC se puede copiar desde el Master UCF.

2.2.9.2 Conectores Diligent Pmod

La ZedBoard tiene cinco cabeceras Digilent Pmod ™ compatibles (2x6). Estos son cabezales de 0.1 hembra que incluyen ocho I/O de usuario más 3.3V y señales de tierra como se muestra en la siguiente figura. Cuatro conectores Pmod conectan al lado PL del Zynq-7000 AP SoC. Un Pmod, JE1, se conecta al lado PS.

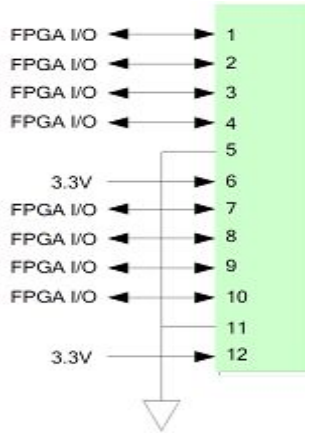

Figura 4. Conectores Diligent P-mod.

2.2.10 XADC

El encabezado XADC proporciona conectividad analógica para diseños de referencia analógicos, incluidas las tarjetas auxiliares AMS de Xilinx.

El encabezado analógico se coloca cerca de la cabecera LPC FMC como se muestra. Tanto el I/O analógico como el digital pueden ser fácilmente soportados para un conector en la tarjeta. Esto permite que la cabecera analógica se conecte fácilmente a la tarjeta FMC utilizando un cable plano corto como se muestra. El encabezado analógico también se puede usar "independientemente" para soportar la conexión de señales analógicas externas.

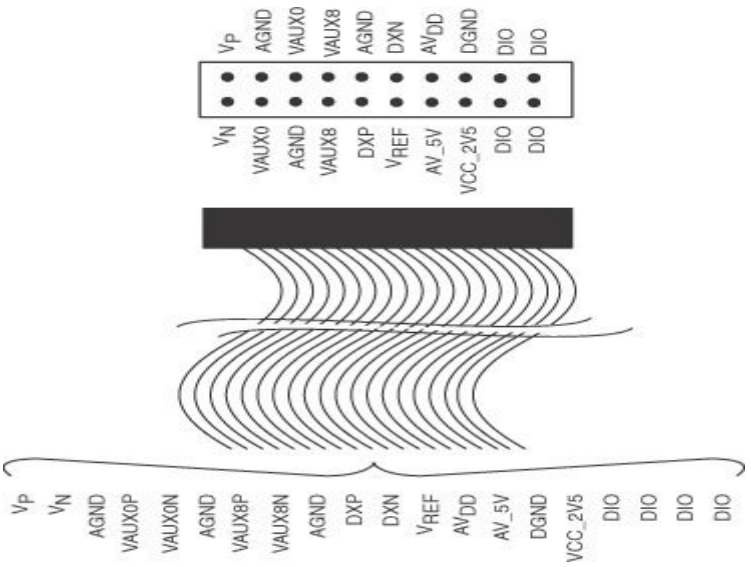

Figura 5. Conector del XADC.

3 HERRAMIENTAS DE DESARROLLO

3.1 Introducción

En este capítulo vamos a representar los programas usados para el desarrollo del proyecto y la interfaz de comunicación con el fin de la realización del mismo.

Los programas usados en el proyecto son:

- Vivado 2016.4 → programa con el objetivo del desarrollo del hardware a fin de las distintas aplicaciones realizadas y el análisis del código y funcionamiento de la interfaz XilliBus.
- VMware Workstation → plataforma con la finalidad de la instalación de una máquina virtual (en este caso un ubuntu 12.4), para la interconexión entre el pc, y la placa de desarrollo ZedBoard.

Además de estos programas, usamos la distribución Xillinux y la interconexión de puerto serie, entre el puerto XX de la ZedBoard y el PC. Donde configuramos la conexión con la distribución montada el VMware Workstation 12 Player.

3.2 Vivado 2016.4

Es un programa proporcionado por la empresa Xilinx, que ofrece el programa de manera gratuita a estudiantes, con una serie de limitaciones en algunos aspectos que no son relevantes para realización del proyecto.

3.2.1 Descarga e instalación

Podemos descargar este programa, desde el enlace siguiente, en la página oficial de Xilinx, [4]. En este enlace de descarga pinchamos en la opción **“Vivado HLx 2016.4: WebPACK and Editions - Windows Self Extracting Web Installer”** . Esto proporcionará un ejecutable, que iniciamos.

En la siguiente figura, vemos la ventana que nos saldrá, introducimos nuestra cuenta en Xilinx (necesaria para la descarga y elegimos la opción de descarga). En este caso introduciremos la cuenta y dejaremos la opción marcada (Download and Install now) y pulsamos *“Next”* .

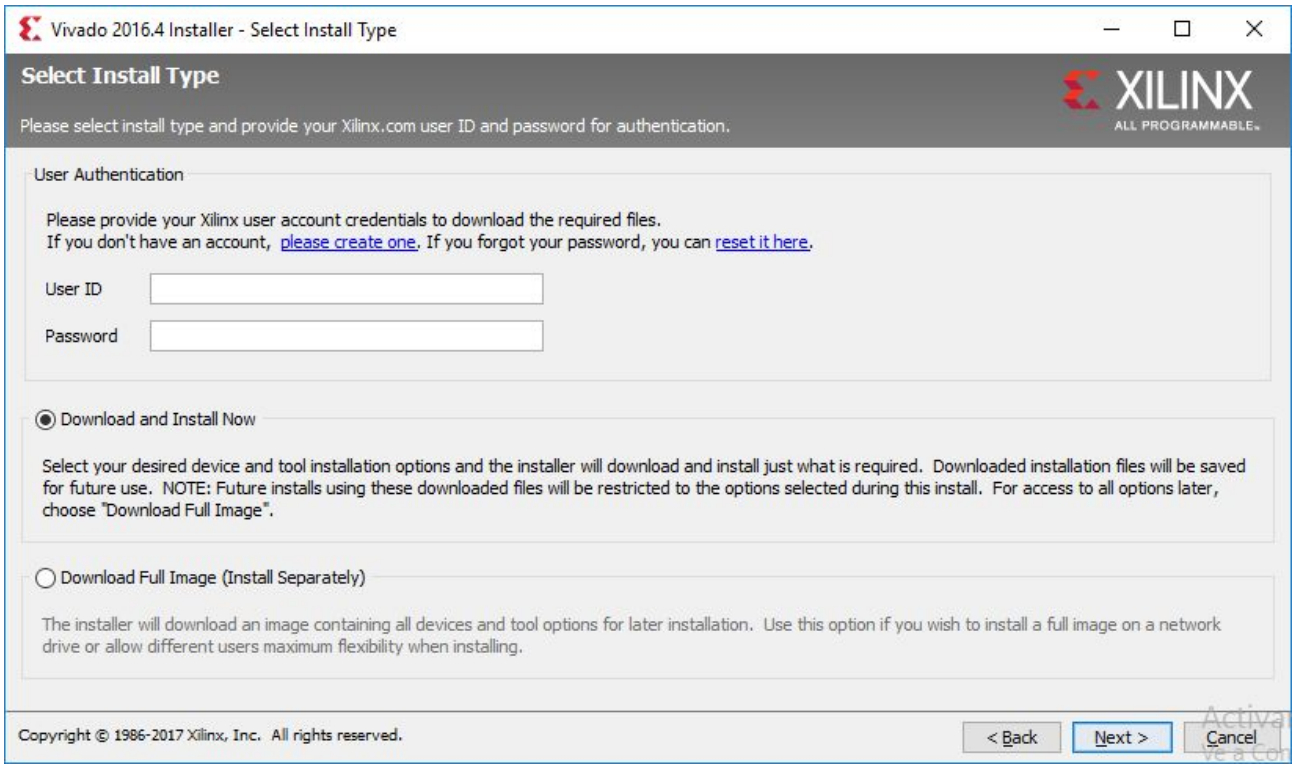

Figura 6. Seleccionar tipo de instalación.

En la siguiente pestaña marcaremos todas las casillas de *"I agree"* y volvemos a pulsar *"Next"*. En la siguiente, como vemos en la figura, seleccionaremos la primera opción, (Vivado HL WebPACK) la más general y sin coste.

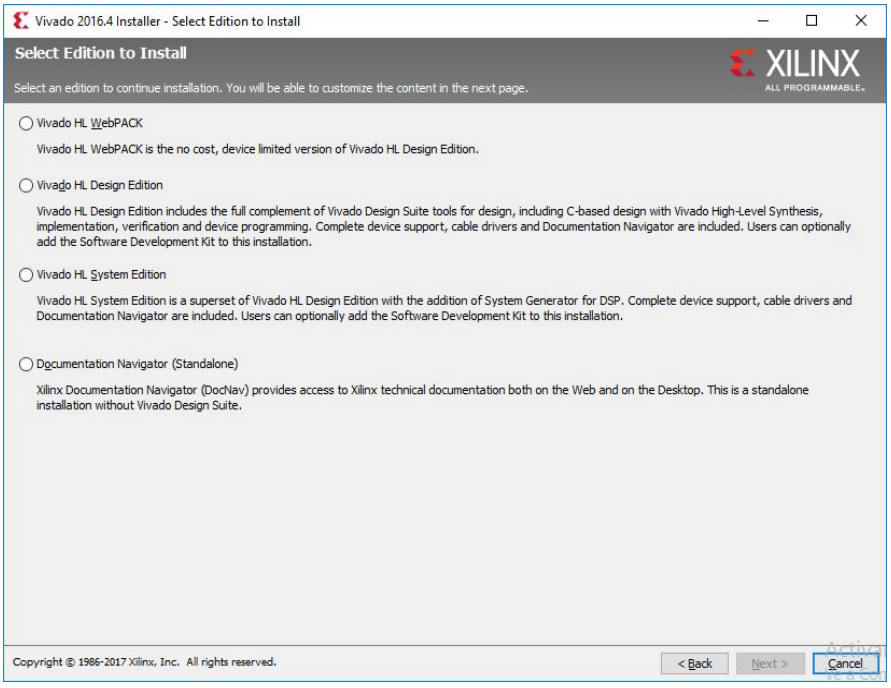

Figura 7. Selección de instalación Vivado.

En la siguiente se indica los elementos que se instalarán, dispositivos, herramientas, disponibles para seleccionar. En este caso dejamos las opciones tal como están seleccionadas por defecto y volvemos a pulsar

"Next" .

Ahora nos aparecerá la siguiente figura, en la que seleccionamos la carpeta donde vamos a instalar el programa, , las opciones de crear iconos, que usuarios del sistema pueden usar …

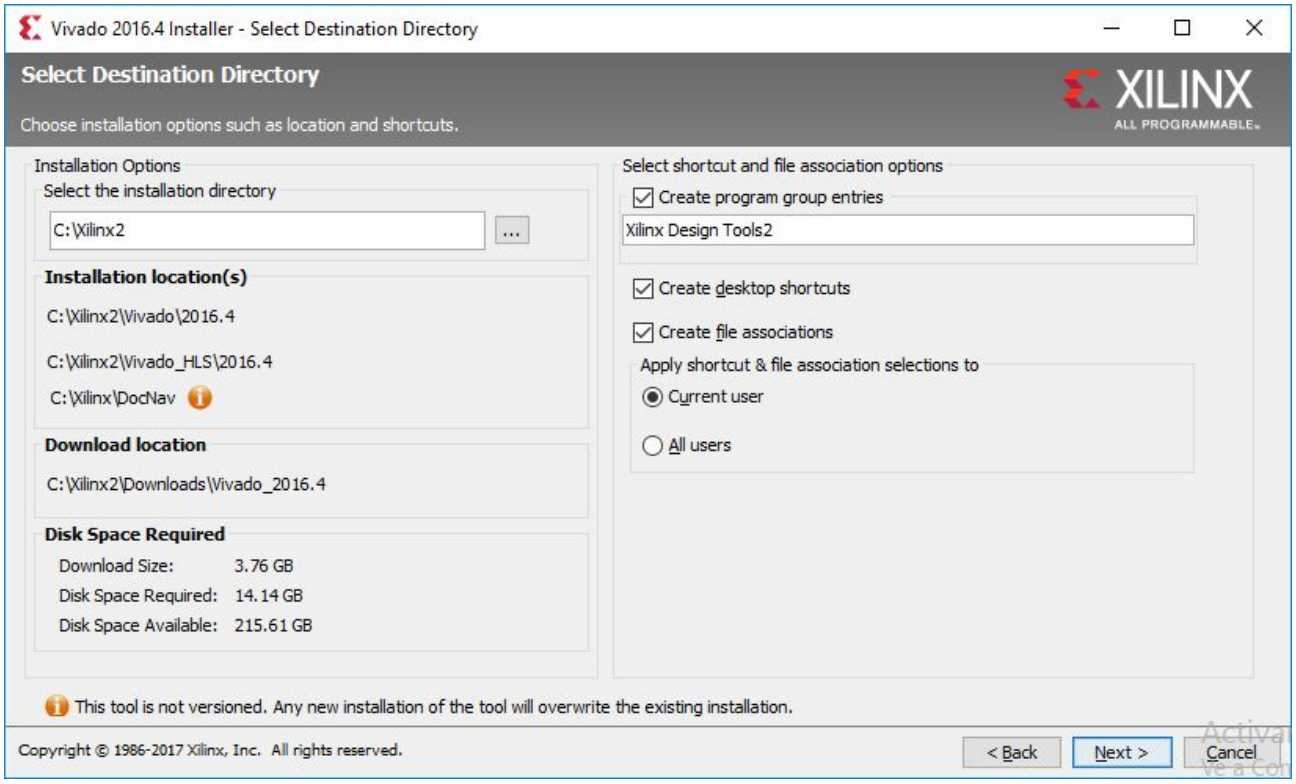

Figura 8. Edición de localización de Vivado.

Después de esta pestaña, nos aparecerá un resumen de las opciones seleccionadas y de lo instalado, en esa pestaña pulsamos en *"Install"* . Nos aparecerá una ventana de espera en la que figura el tiempo hasta que se termine la descarga e instalación del programa.

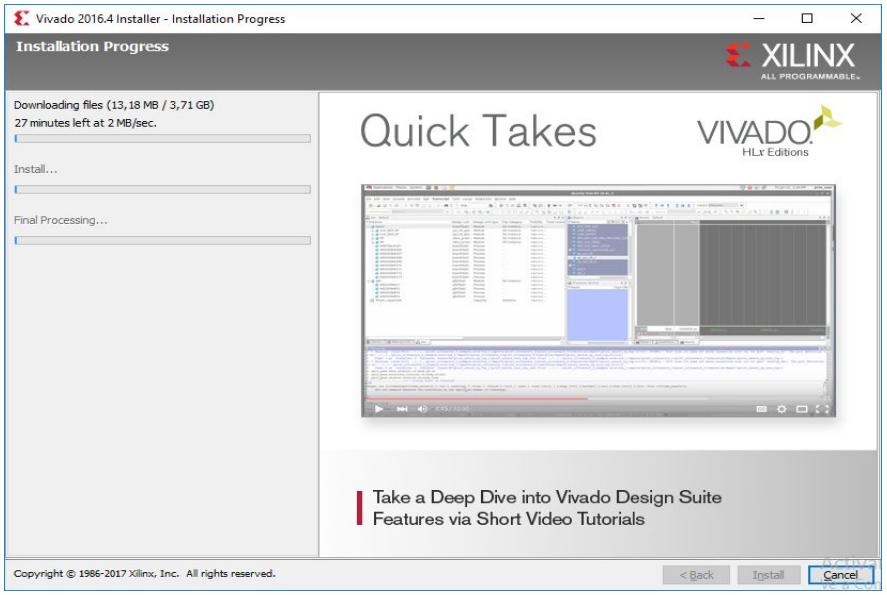

Figura 9. Pantalla de Descarga/Instalación Vivado.

Cuando este proceso termine saldrá una ventana que nos pedirá una licencia para usar el programa, seleccionamos la opción *"Get Vivado or IP Evaluation Licenses"* . Esto nos llevará a un enlace, donde se pedirá nuestra cuenta, iniciamos la sesión, confirmamos datos de la cuenta, y nos aparecerá una página donde podemos generar una licencia, bajo el apartado "*Certificate Based Licenses"*.

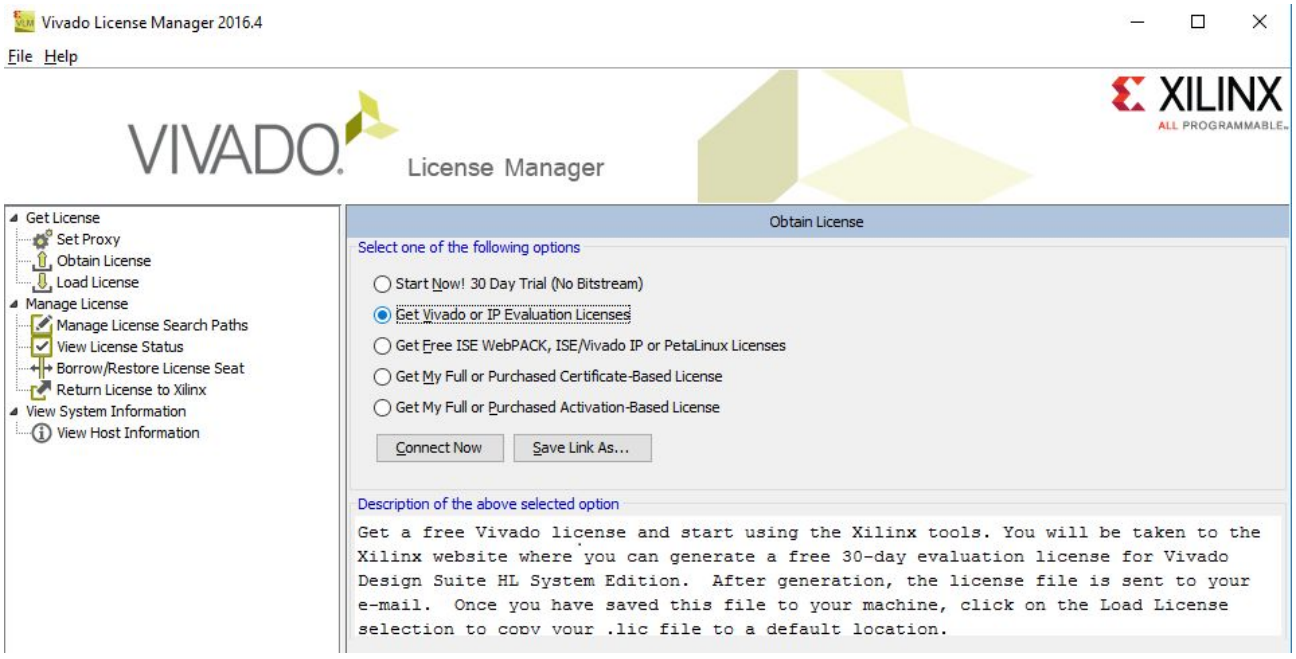

Figura 10. Pantalla de obtención de licencias Vivado.

En este apartado seleccionamos "*Vivado HLS Evaluation License"* y pulsamos *"Generate Node-Locked License"*. Ahora aparecerá una ventana para seleccionar el host donde queremos guardarla, pulsamos *"Next",* y en la siguiente pestaña igual.

Ya se nos habrá enviado la licencia al correo, colocamos la licencia en la carpeta Xilinx, volvemos a la Figura 10, y pulsamos *"Save Link As…"* , buscamos el archivo anterior, y ya lo tenemos listo. Este proceso deberá ejecutarse (el de la licencia) cada 30 días.

3.2.2 Carga del proyecto

Una vez instalado el programa, iniciamos el programa, descargamos el kit de desarrollo para xillibus de ejemplo, que es gratuito en este caso debido a que no vamos a usarlo en ninguna aplicación comercial.

Este kit se descarga en [5] , seleccionando en la parte baja de la página donde esta Download, en el menú *"Download the boot partition kit for your board:"* y seleccionamos la opción *"ZedBoard"* . Cuando se descarga el kit, se descomprime y volvemos al Vivado 2016.4.

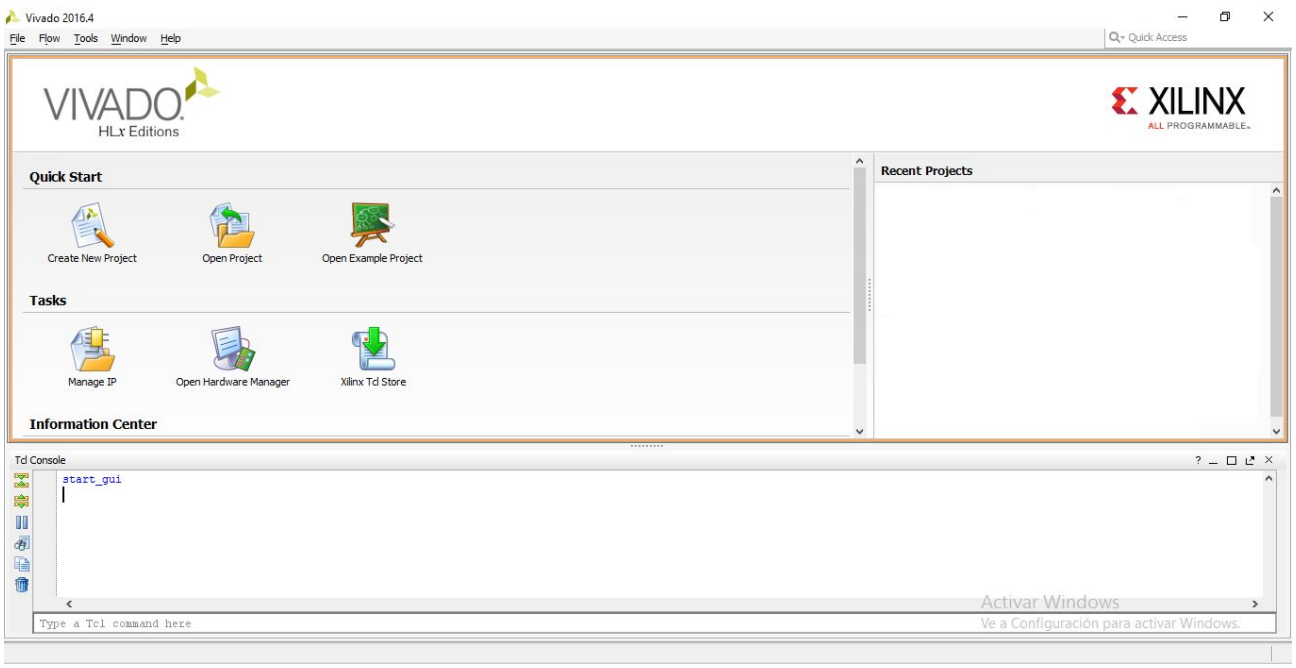

Figura 11. Ventana inicial Vivado.

En la figura XX, seleccionamos el menú *"Tools/ Run Tcl Script…"* Buscamos en el archivo, en este caso está en *"C:\Users\Usuario\...\xillinux-eval-zedboard-2.0a\xillinux-eval-zedboard-2.0a\vhdl"* .

Al hacer esto se produce la carga de un proyecto modelo que permite el desarrollo y ya está disponible el xillybus, en un modelo de ejemplo.

3.2.3 Resumen de las principales ventanas de Vivado 2016.4

Con el proyecto ya cargado, como vemos en la figura 12, tenemos distintos paneles, que explicaremos en esta sección.

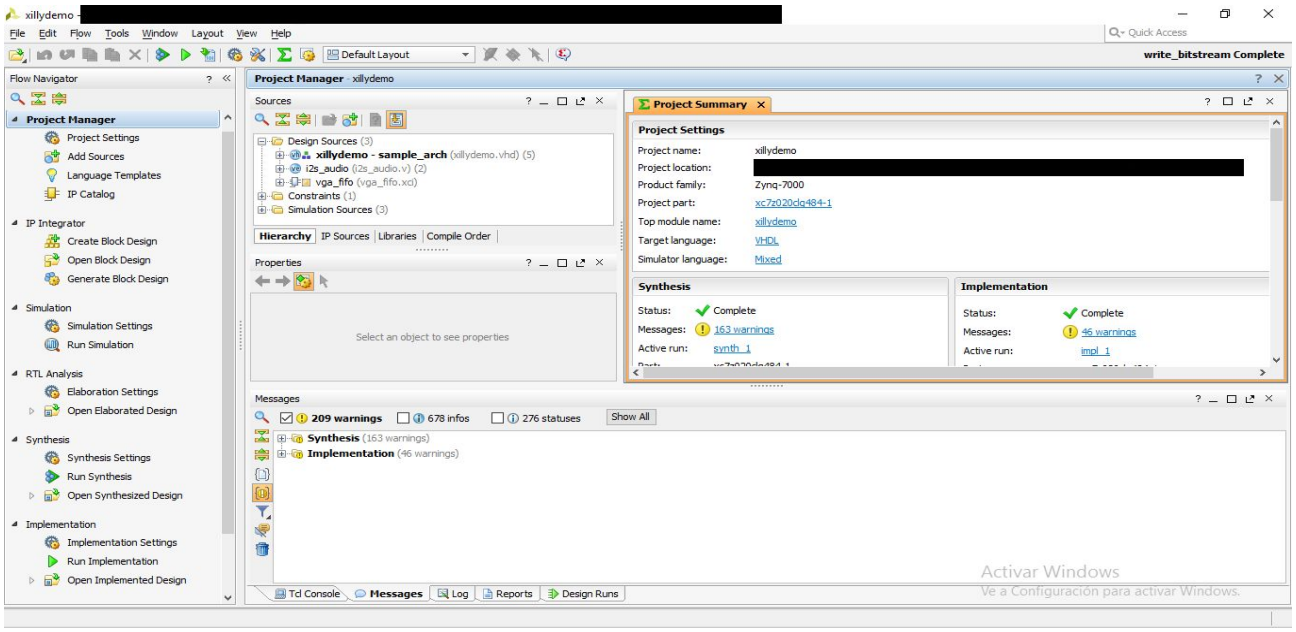

Figura 12. Ventana inicial proyecto de Vivado.

Tenemos disponibles 4 paneles principales:

- Flow Navigator.
- Data Windows Area.
- Results.
- Workspace.

3.2.3.1 Flow Navigator.

En el panel Flow Navigator, disponemos de distintas opciones para la gestión del proyecto. Las opciones disponibles son:

- **Project Manager**:
    - **Project Settings**: para modificar las opciones elegidas en la creación del proyecto (chip a usar, lenguaje de programación, librerías…).
    - **Add Sources**: nos permite añadir nuevos archivos que contengan código al proyecto, ya sean de diseño, simulación...
    - **Language templates**: contiene las distintas plantillas para la hora de generar código, pueden modificarse si necesitamos que siempre se añada algo concreto.
    - **IP Catalog**: contiene un catálogo de los IP que proporciona Xilinx para su uso, con versiones de licencia para empresas es más extenso. Proporciona además un asistente para la creación de estos IP según nuestras necesidades.

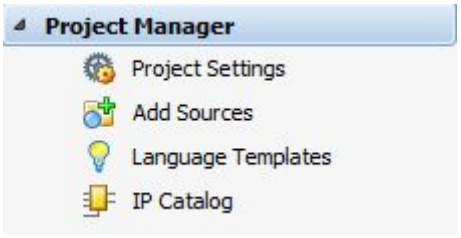

Figura 13. Project Manager Vivado.

- **IP Integrator:**
    - **Create Block Design**: Nos da la posibilidad de crear un diagrama para, mediante la adición de IP, realizar un esquemático con los IP, y su interconexión (archivo .bd) .
    - **Open Block Design**: Nos permite abrir, los esquemáticos guardados.
    - **Generated Block Design**: Esta opción nos ofrece regenerar los IP que se modifiquen fuera del esquemático, para que en el esquemático aparezcan actualizados.

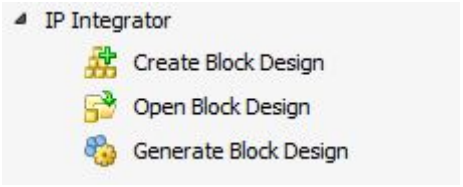

Figura 14. IP Integrator Vivado.

- **Simulation**:
    - **Simulation Settings**: Nos posibilita cambiar las distintas opciones necesarias para la simulación, cuál es el top_module para la simulación, el lenguaje para la simulación...
    - **Run Simulation**: lanza la simulación, y previamente le hace un análisis de síntesis e implementación del test_bench.

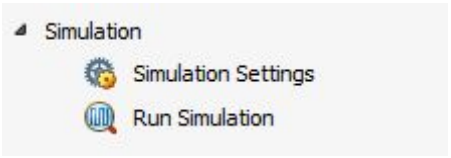

Figura 15. Simulation Vivado.

- **RTL Analysis**:
    - **Elaboration Settings**: podemos seleccionar cómo queremos hacer el analisis de la interconexión de los distintos bloques si en modo caja negra o en netlist. También si queremos generar las restricciones con la netlist.
    - **Open Elaborated Design**:
        - **Report Methodology**:
        - **Report DRC**:
        - **Report Noise**:
        - **Schematic**:

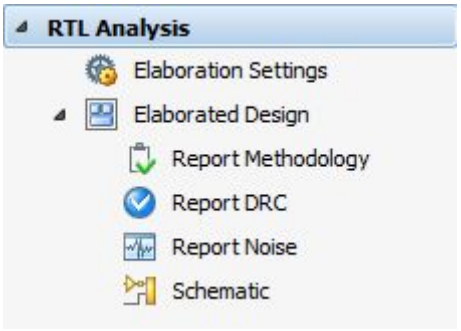

Figura 16, RTL Analysis Vivado.

- **Synthesis**:
    - **Synthesis Settings**: nos permite seleccionar la carpeta de archivos donde ejecutaremos el análisis, ysus opciones.
    - **Run Synthesis**: lanza el análisis sobre el top_module.
    - **Open Synthesized Desing**: contiene los distintos informes sobre tiempo, interconexiones, problemas de reloj, potencia…

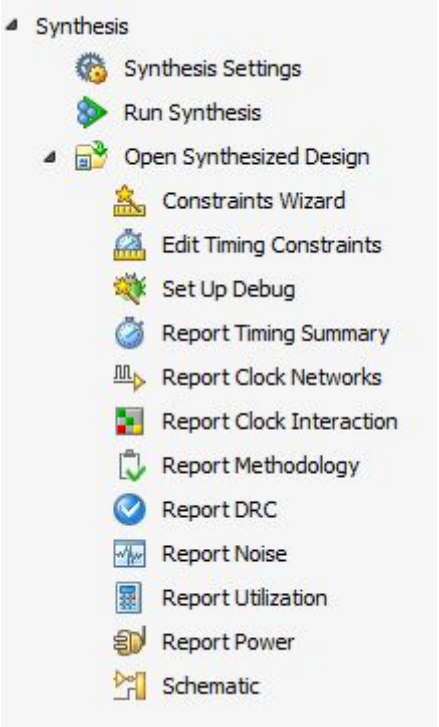

Figura 17. Síntesis Vivado.

- **Implementation:**
    - **Implementation Settings**: permite seleccionar la carpeta de archivos donde ejecutaremos la implementación del código y las opciones del proceso.
    - **Run Implementation**: lanza la implementación (previo Synthesis correcto).
    - **Open Implemented Design**: contiene los distintos informes sobre ruido, interconexiones, problemas de los relojes, potencia…

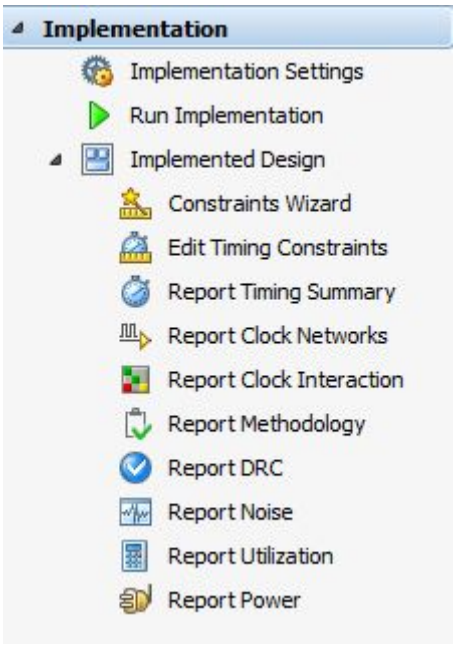

Figura 18. Implementation Vivado.

- **Program and Debug**:
    - **Bitstream Settings**: permite seleccionar algunas opciones para la generación del bitstream.
    - **Generate Bitstream**: genera el .bit después de que la implementación no contenga errores (y en el mejor de los casos sin warnings).
    - **Open Hardware Design**: esta opción da la opción de modificar la tarjeta que elegimos para el hardware, el resto de opciones no se pueden elegir debido a la licencia que estamos usando.

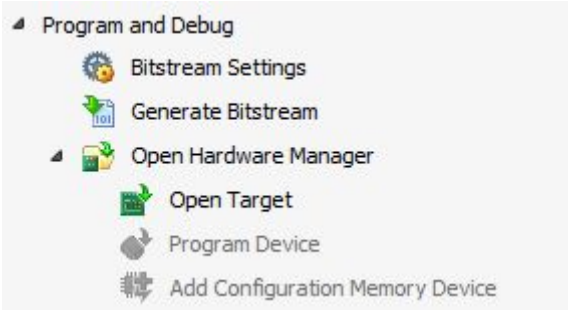

Figura 19. Bitstream Vivado.

3.2.3.2 Data Windows Area

En este panel encontramos los distintos ficheros de código y datos, e información relacionada con los mismos.

Puede tener distintas pestañas, pero las dos principales son:

- **Source**

- **Properties**

La pestaña *"Source"* contiene 4 ventanas internas

- **Hierarchy**: posee los archivos de código fuente, divididos en distintas carpetas, diseño, simulación, archivos con fallos…

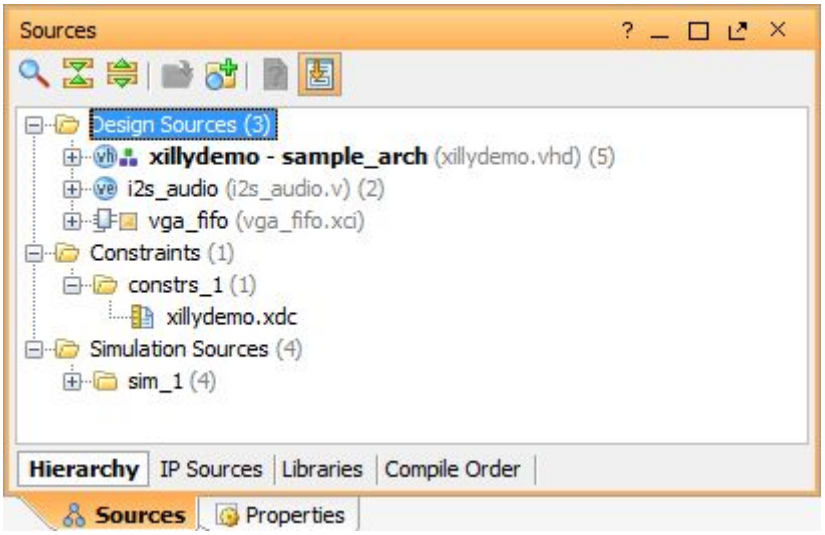

Figura 20. Hierarchy Vivado.

- **IP Source**: alberga los código fuentes de los IP, y de los block design usados en el proyecto.

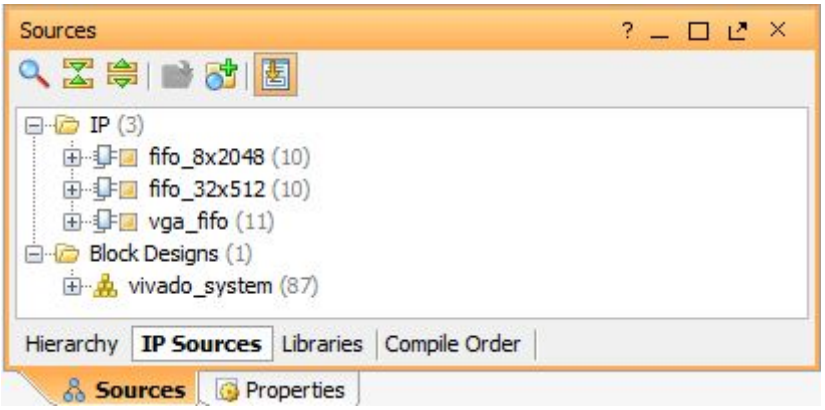

Figura 21. IP Sources Vivado.

- **Libraries**: en esta ventana vemos como están enlazados los distintos archivos. Según su carpeta de la ventana Hierarchy, del lenguaje usado, y en qué librería está cada archivo.

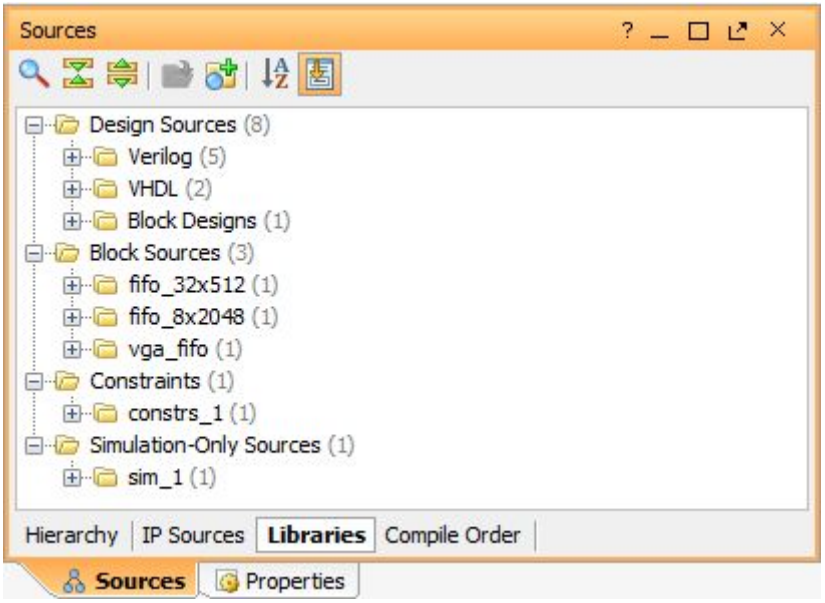

Figura 22. Libraries Vivado.

- **Compile Order**: en esta ventana se observa cuál es el orden de compilación, de los archivos de código fuente, los bloques IP, y las restricciones.

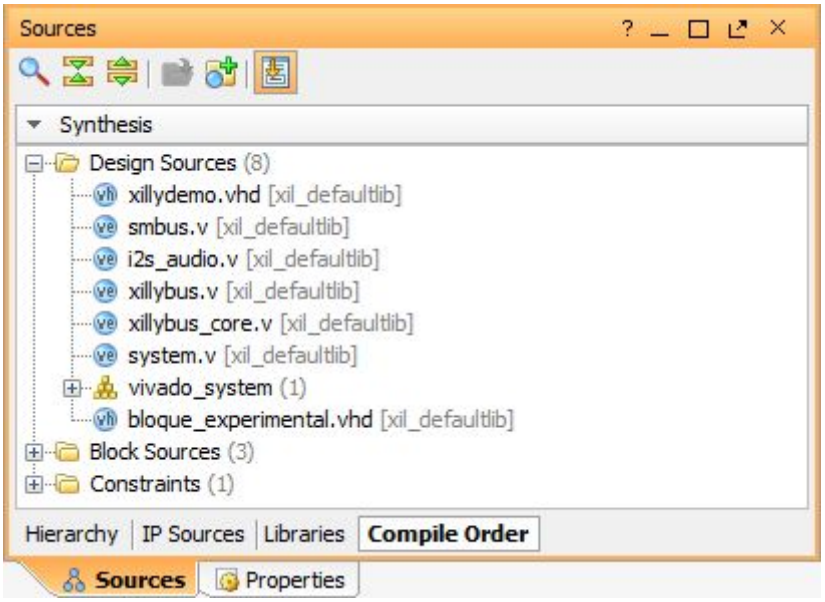

Figura 23. Compile Order Vivado.

En la pestaña *"Properties"* tenemos dos ventanas de información:

- **General**: propiedades generales del archivo de código fuente, si se usarán en los distintos análisis, y si está disponible.

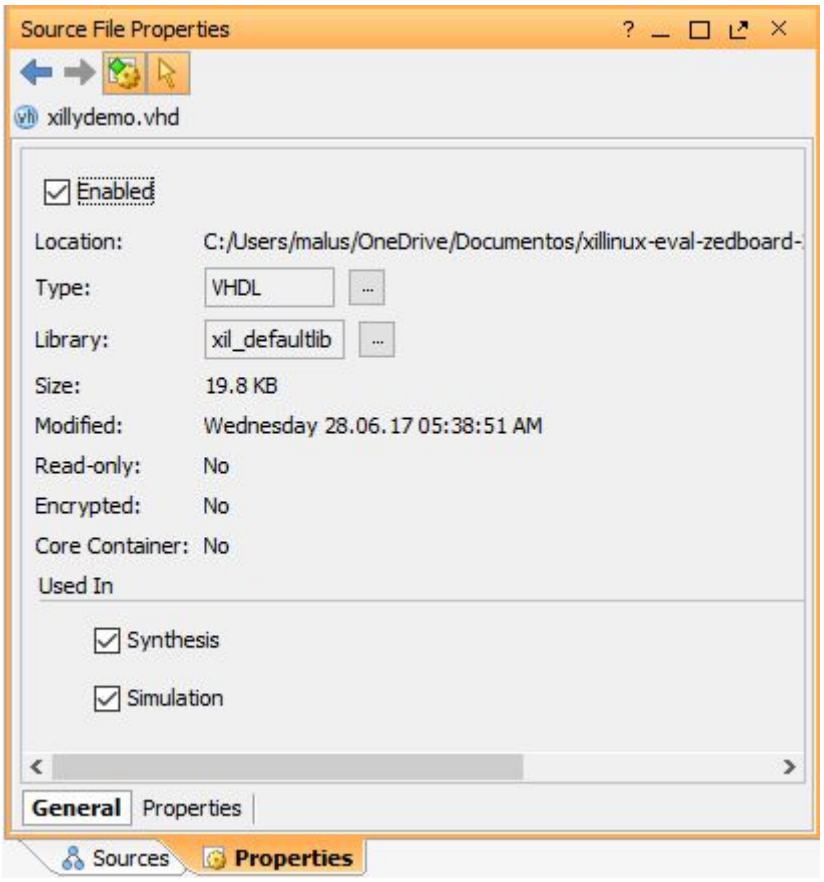

Figura 24. Properties General Vivado.

- **Properties**: en esta ventana, se ven las propiedades individuales de cada archivo de código fuente.

3.2.3.3 Results Area

Este panel incluye todas las ventanas de información referente al proyecto, el estatus y resultado de comandos, los reportes de los distintos procesos (generación de IP, Synthesis…)… y cualquier ventana de información adicional generada por el Flow Navigator. Las principales ventanas de este area son:

- **Tcl Console**: funciona como una consola de comandos, muestra los comandos de Tcl que ejecuta Vivado y sus resultados.

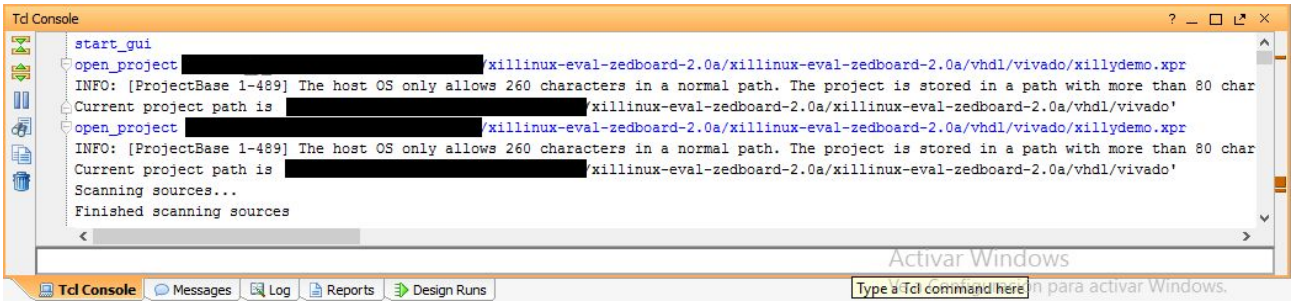

Figura 25. Ventana Tcl Vivado.

- **Messages**: en esta ventana aparecen los distintos mensajes generados, de error, warning, información y status los diferentes procesos de diseño, agrupados en bloques. En esos procesos se concreta en cada código se produce el mensaje de error, warning… Podemos seleccionar qué tipos de mensajes

queremos ver y cuáles no nos interesan.

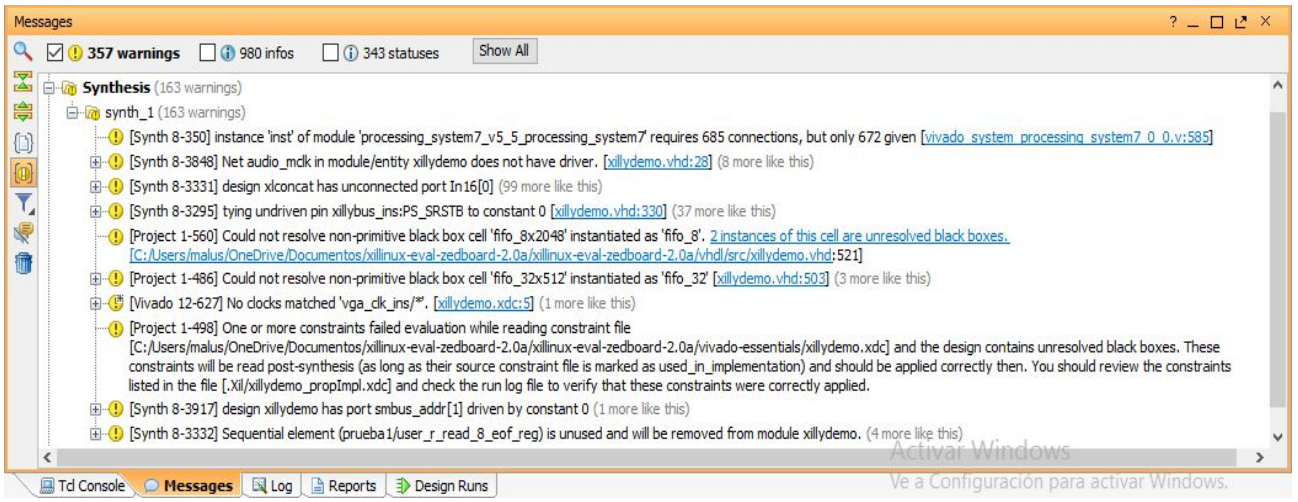

Figura 26. ventana Messages Vivado.

- **Log**: en esta ventana aparecen los archivos .log, que generan los comandos de los procesos de síntesis, implementación y simulación.

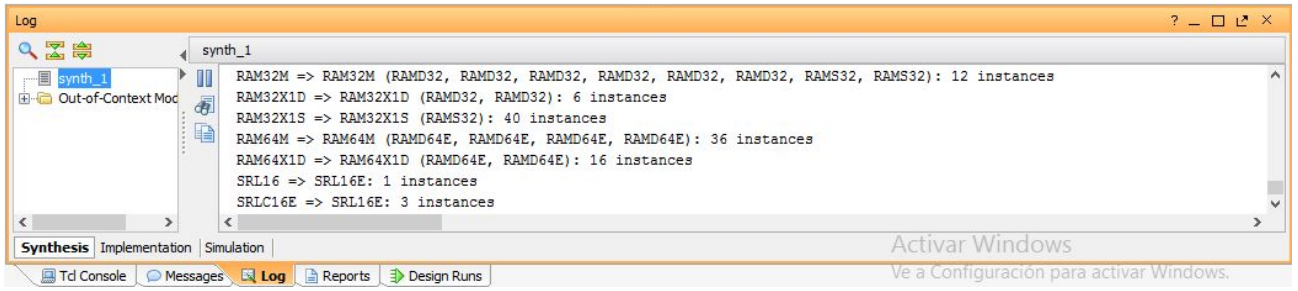

Figura 27. ventana Log Vivado.

- **Reports**: aquí se contienen los distintos informes generados por los procesos de los análisis anteriores realizados.

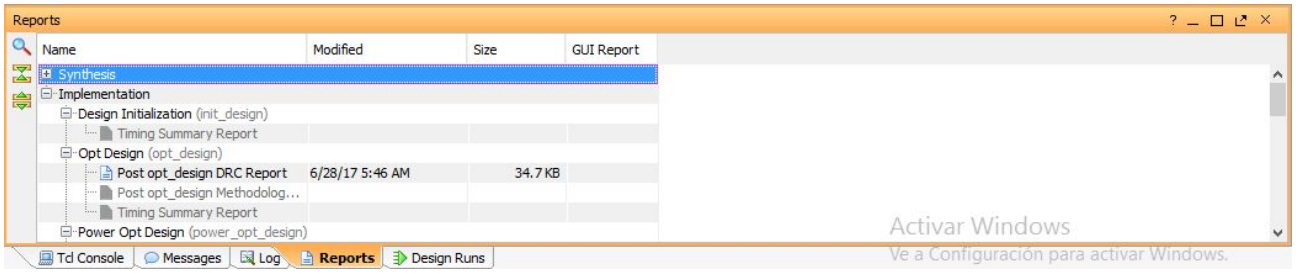

Figura 28. Ventana Report Vivado.

Design Runs: en esta ventana podemos observar distintas propiedades de los procesos e IP generados (como se ve en la figura 29, cuando se empieza y se acaba la generación de IP y los análisis, consumo…) . Y podemos gestionar estos procesos.

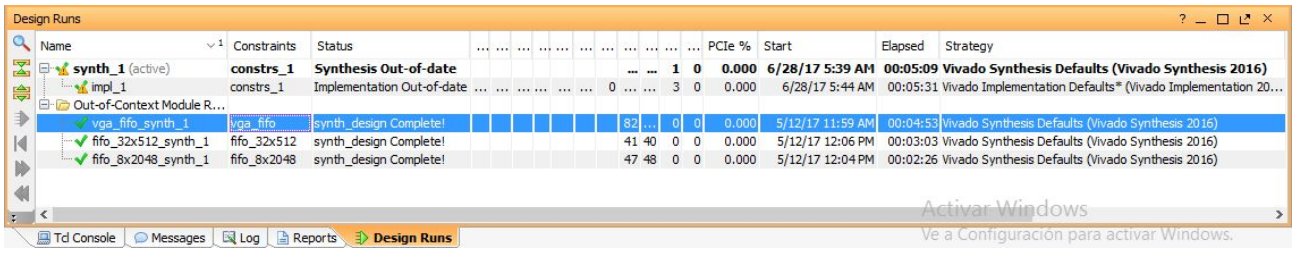

Figura 29. ventana Design Runs Vivado.

3.2.3.4 Workspace

En este panel en primera instancia tenemos una pestaña *"Project Summary"*, donde figura un resumen del proyecto de características como:

- **Project Settings**
- **Synthesis**
- **Implementation**
- **DRC Violations**
- **Timing**
- **Utilization**
- **Power**

Además de esta pestaña, en el panel, tendremos todos los archivos de código fuente que modifiquemos, y toda información que necesite una interfaz gráfica con más tamaño, esquemáticos, bloques de diseño…

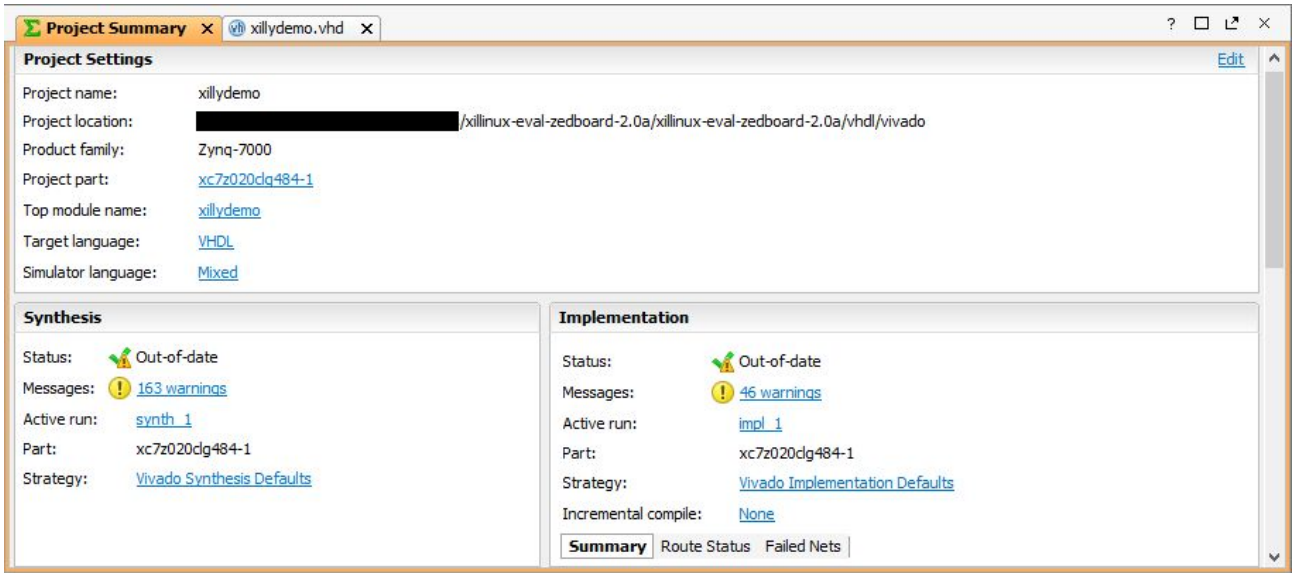

Figura 30. panel Workspace Vivado.

3.3 VMware Workstation 12 Player + Ubuntu 12.04

Para la conexión con la placa de desarrollo, usaremos una máquina virtual que mediante el puerto serie, se hará con una distribución Linux, Ubuntu 12.04, ya que el Xillinux, está basado en esta distribución para evitar problemas de compatibilidad. Se podría usar otro S.O. mientras que se pueda usar el puerto serie para la conexión con Xillinux.

Para esto se instalará la máquina virtual en VMware Workstation 12 Player.

3.3.1 Descarga e instalación

El programa VMware Workstation 12 Player lo descagamos desde el enlace de la página oficial, [6] , seleccionando la opcion *"VMware Workstation 12 Player para Windows de 64 bits"* . Su instalación del programa es intuitiva.

La descarga del S.O. , Ubuntu 12.04 , se realiza desde [7], seleccionando la opción de Ubuntu 12.04 en VMware, descargamos la versión de 64-bits (debido a que el proyecto se realiza en un Windows de 64 bits).

Posteriormente abrimos la imagen vmdk con el VMware y ya tenemos la máquina virtual lista para su uso.

3.4 Xillinux

3.4.1 Introducción

Es un sistema operativo desarrollado por Xillibus para el uso de xillybus basado en Ubuntu 12.04. La distribución Xillinux es un software + kit de código FPGA para ejecutar un escritorio gráfico completo con posibilidad de añadir un teclado y un ratón al hardware de la placa.

Una configuración ejemplo del núcleo Xillybus IP está incluida en la lógica de Xillinux. Se puede configurar y descargar un núcleo IP personalizado de Xillybus en el IP Core Factory que sustituya al que viene por defecto.

Basado en Ubuntu LTS 12.04 para ARM, hace que la ZedBoard se comporte como un PC con la tarjeta SD como su disco duro. Tiene también disponible la conexión de distintos periféricos de audio, teclado y ratón.

3.4.2 Descarga e implementación

La imagen de la distribución la podemos encontrar en [5]. Descargamos esta imagen en el Windows , para montar el xillinux en la tarjeta SD más cómodamente.

Descomprimimos el archivo *"xillinux-1.3.img.gz"* , introducimos la tarjeta SD en el PC. Descargamos USB image tool, desde [7] , descomprimimos y ejecutamos el entorno gráfico del programa.

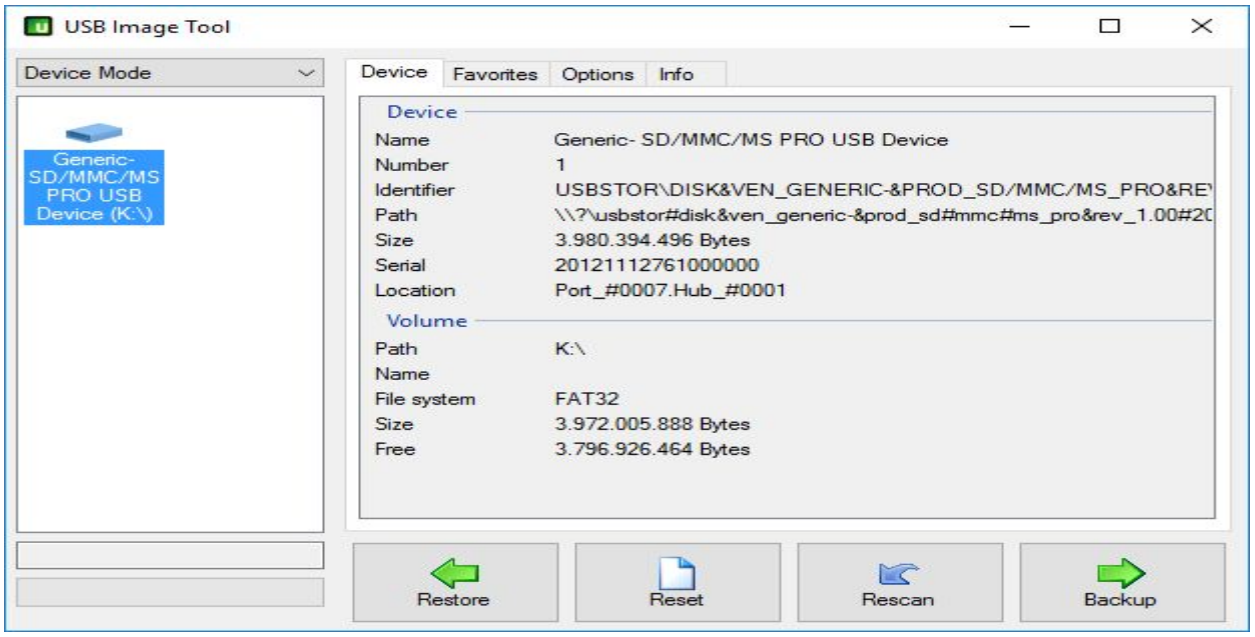

Figura 31. USB Image Tool.

Pulsamos en *"Backup"* , buscamos la imagen descomprimida y esperamos a que se cargue en la tarjeta SD, cuando acabe ya podemos usar la tarjeta como disco duro para la ZedBoard.

3.5 Configuración de puerto serie

Para la configuración del puerto serie, iniciamos la máquina virtual. Cuando cargue abrimos un terminal y entramos en root, si se ha descargado la máquina virtual del enlace anterior, los datos son :

- user: user
- password : password

Tras entrar en modo superusuario, instalamos el paquete minicom:

"apt-get install minicom" (contestamos que sí a los avisos que nos da), y despues añadimos el comando "chmod 666 /dev/ttyACM0".

Conectamos ahora el cable USB-microUSB suministrado por la placa, desde un puerto USB del PC, al puerto serie de la placa y la encendemos.

Figura 32. Barra de iconos de VMware con cable conectado.

En la parte superior derecha de la máquina virtual debe aparecernos ese icono. Hemos de hacer clic derecho y pulsar *"Connect"* . Con esto deberíamos tener ya el cable conectado pero para asegurarnos introduciremos el comando:

"tail -f /var/log/syslog | grep tty"

```
root@ubuntu:/home/user# tail -f /var/log/syslog | grep tty
Jul  3 13:32:52 ubuntu kernel: [ 1907.997351] cdc_acm 1-1:1.0: ttyACM0: USB ACM
device
Jul  3 13:33:20 ubuntu kernel: [ 1935.907046] cdc_acm 1-1:1.0: ttyACM0: USB ACM
device
```

Figura 33. Comprobación de existencia de cable USB-puerto serie.

Ahora pasamos a configurar el minicom:

"minicom -s" Seleccionamos *"Serial port setup"* y pulsamos enter

```
+-----[configuration]------+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl        |
| Save setup as..          |
| Exit                     |
| Exit from Minicom        |
+--------------------------+
```

Figura 34. Menú de configuración minicom.

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyACM0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : Yes                                       |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

Figura 35. Configuración puerto serie, minicom.

En este menú introducimos "*A"* para entrar en *"Serial Device"* y modificamos /dev/tty0, por /dev/ttyACM0, para poner nuestro cable como puerto serie , dejamos el resto igual.

Volvemos al menú de la figura xx, y seleccionamos *"Save setup us dfl"* , y luego salimos de minicom.

Introducimos minicom, y accederemos al Xilinux que estamos corriendo en la FPGA.

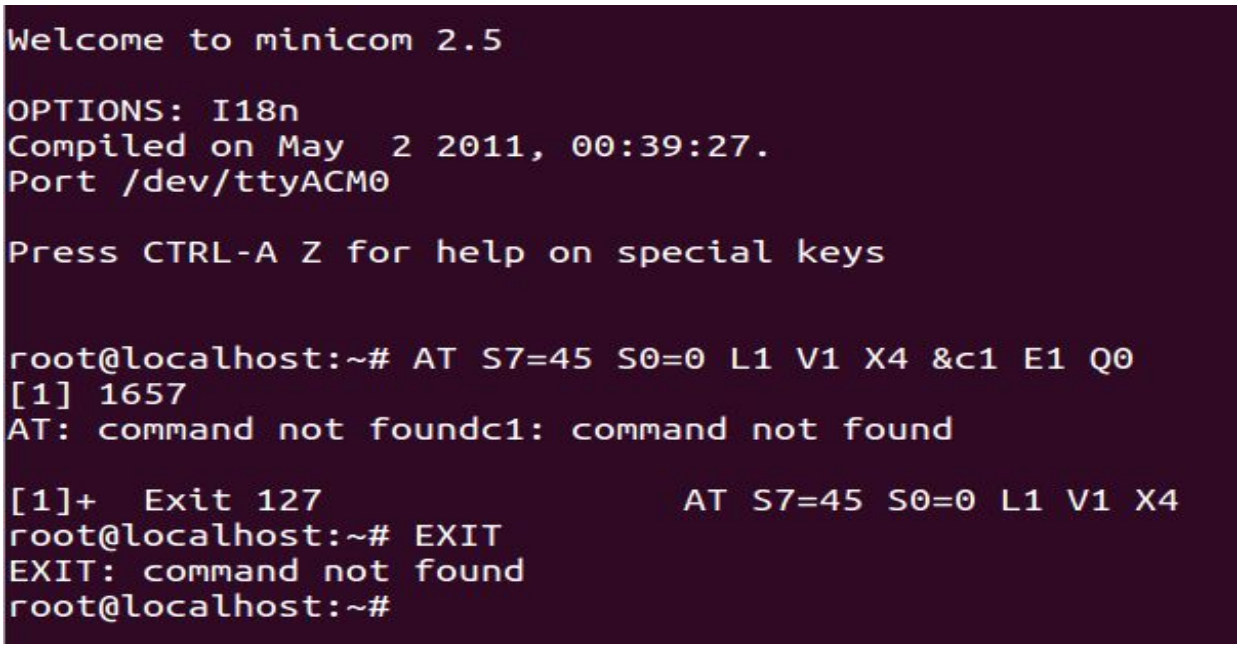

Figura 36. Conexión con la FPGA, por puerto serie.

4 XILLIBUS

4.1 Introducción

Xillibus es una interfaz de interconexión de comunicación entre la FPGA y el host elegido, ya sea un PC (bien Microsoft Windows o Linux), o cualquier sistema embebido que disponga de buses PCI express. Está diseñada para evitar los problemas usuales en el tráfico de datos entre la PL y la PS que surgen a los ingenieros al desarrollar una aplicación en FPGA.

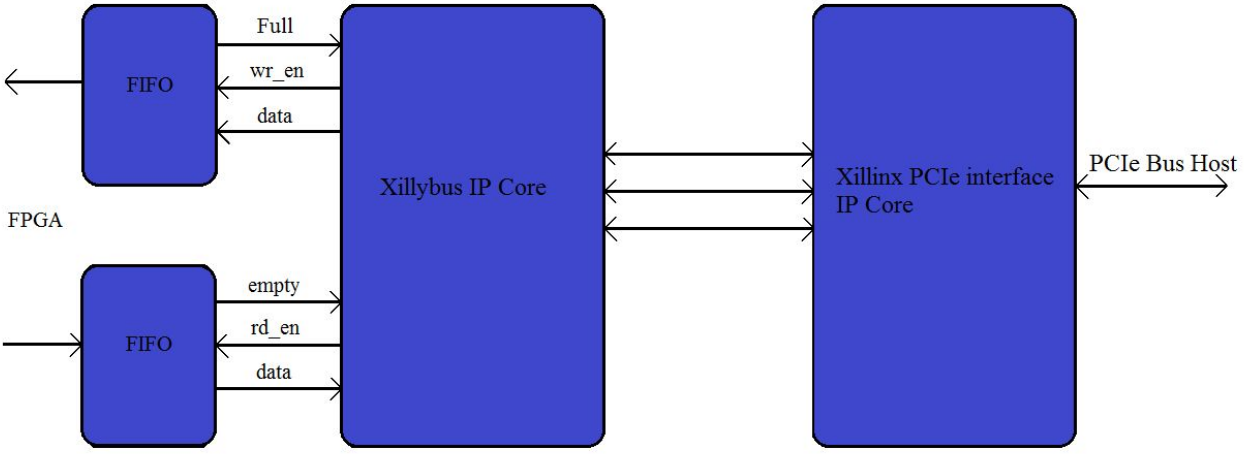

Figura 37. Esquema FPGA-host Xillybus.

El esquema presentado en la figura 37 muestra como Xillibus, mediante el uso de FIFO's, controla el tráfico de datos de la parte de programación lógica. Estas FIFO pueden diseñarse con un ancho y profundidad a discreción del desarrollador. Los datos enviados o recibidos a estas FIFO's se cargan en el host mediante ficheros de datos.

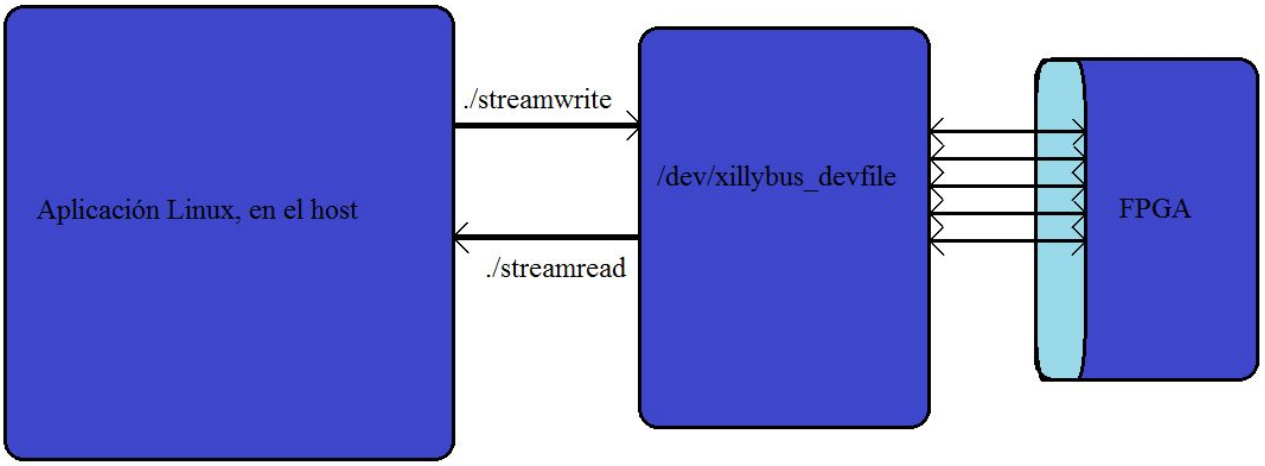

Figura 38. Esquema host-FPGA Xillybus.

En la parte del host, la comunicación con la FPGA se hace escribiendo en ficheros, a los que se accede generalmente /dev/xillibus_devfile. Se realiza desde este fichero a la FPGA de manera similar a cualquier intercambio de datos de dispositivos en Linux, solo que este proceso no entrega datos a otro proceso, sino a una FIFO.

4.2 Descripción de señales

Las señales que se usan en el tráfico de datos siguen las siguientes reglas de nomenclatura:

- Todas las señales de interfaz de usuario empiezan por *"user"*.
- Seguido de una bandera que indica si la señal es de host-FPGA (w), o de FPGA-host (r).
- Posteriormente se añade el nombre del archivo devfile donde estará el archivo origen/destino de la señal en /dev/.
- De manera final se indica la función de la señal.

El formato sería, ***user_r/w_devfile_función*** .

4.2.1 Señales de transmisión host-FPGA

- **user_w_{devfile}_data**: esta señal contiene los datos enviados por el host durante el ciclo de lectura. Su ancho puede variar entre 8, 16 y 32 bits.
- **user_w_{devfile}_wren**: con esta señal se indica cuando el dato de la señal anterior es válido ('1') para su recepción por la entidad receptora.
- **user_w_{devfile}_full**: muestra si se puede seguir escribiendo ('0') o si no ('1') . Suele indicar que la FIFO receptora está llena o no, pero no está controlada por la FIFO, la lógica desarrollada debe controlarla.
- **user_w_{devfile}_open**: se activa ('1') cuando el devfile en el anfitrión está abierto para escritura. Opcionalmente se puede utilizar para restablecer la FIFO u otra lógica entre el archivo abierto. Si un archivo es abierto por varios procesos en el host, esta señal permanece confirmada hasta que todas las instancias abiertas se cierran [3]

4.2.2 Señales de transmisión FPGA-host

- **user_r_{devfile}_data**: contiene los datos enviados hacia el host, durante los ciclos de lectura. También puede ser de ancho 8, 16, y 32 bits. Esta señal no varía mientras no se active, user_r_{devfile}_rden, y el reloj. [3]
- **user_r_{devfile}_rden**: es una señal que se activa ('1') cuando el siguiente dato, a pasar al devfile, está activo y listo, mientras tanto se mantiene a bajo nivel.
- **user_r_{devfile}_empty**: esta señal indica si se puede seguir leyendo ('0') o si no hay más que leer ('1'). Es la equivalente a la user_w_{devfile}_full para ese tráfico, también necesita una lógica adicional para su uso.

- **user_r_{devfile}_ eof**: similar a la user_r_{devfile}_empty, pero cuando se activa no se emiten nuevos ciclos de lectura hasta que se cierra el devfile y se vuelve a abrir. En el lado del host se indica que se ha alcanzado EOF cuando se consumen todos los datos. Esta señal se puede usar para bloquear la lectura. Una vez activada y pase un ciclo de lectura ya no importa que se desactive o se mantenga a '0' hasta que se cierre el devfile. [3]

- **user_r_{devfile}_open**: indica que cuando se activa ('1') el devfile está abierto para lectura. Se puede usar como reset a nivel bajo, para la FIFO u otra lógica entre el devfile y la aplicación receptora. En caso de que el devfile esté abierto por varios procesos, permanece activa hasta que todos los procesos acaben.

4.2.3 Señales de interfaz de memoria

- **user_{devfile}_addr**: dirección de memoria actual. Al activarse una lectura o escritura, esta sería la dirección de memoria donde se realiza la operación. La anchura es configurable hasta 32 bits. Se pone a 0 todos los bits, cuando la operación a realizar sale de la dirección máxima posible por el ancho dado.[3]

- **user_{devfile}_addr_update**: esta señal se activa para indicar que hay que esperar para preparar los datos por la lectura. Se puede usar para pasar a nivel alto la señal user_r_{devfile}_empty después de un ciclo de reloj de que la señal pase a nivel alto.

4.2.4 Señal quiesce

A nivel alto ('1') indica que el host no ha activado la interfaz Xillibus o se ha desactivado, se usa como reset síncrono. Implica en modo activo, que todos los devfile están cerrados, por lo que las *_open se podrían usar sólo como reinicio.

4.3 Flujos Síncronos vs Asíncronos

Cada flujo Xillybus tiene una bandera, que determina si se comporta de forma síncrona o asíncrona. El valor de esta bandera se fija en la lógica de la FPGA. Cuando un flujo está marcado como asíncrono, se le permite comunicar datos entre el FPGA y el software de nivel de kernel del host sin la participación del software del espacio del usuario, siempre y cuando el devfile respectivo esté abierto.[2]

- Los flujos asíncronos tienen un mejor rendimiento, en particular cuando el flujo de datos es continuo.

- Los flujos síncronos son más fáciles de manejar y son la opción preferida cuando se necesita una sincronización estrecha entre las acciones del programa host y lo que ocurre en el FPGA.

En los núcleos IP personalizados que se generan en IP Core Factory, (http://www.xillybus.com/custom-ip-factory) , la selección entre hacer que cada flujo sea síncrono o asincrónico se basa automáticamente en la información sobre el uso previsto de la secuencia, según lo declarado por el usuario de la herramienta cuando está habilitado "autoset internals". Si la opción autoset está desactivada, el usuario hace esta opción explícitamente.[2].

4.3.1 Flujo host-FPGA

4.3.1.1 Flujo Asíncrono

Los flujos asíncronos implican que las llamadas de la aplicación volverán inmediatamente la mayor parte del tiempo, si cuando escriben en el devfile los datos pueden almacenarse en los bufers DMA. Los datos posteriormente se transmiten a la FPGA a la velocidad solicitada por la lógica de la misma, sin implicación de la aplicación del host.

Los datos se envían hacia la FPGA cuando:

- El búfer de DMA actual está lleno.
- El descriptor de archivo se vacía explícitamente.
- El devfile se cierra.
- Un temporizador expira, forzando una descarga automática si no se ha escrito nada en el flujo durante un período específico de tiempo (normalmente 10 ms).

4.3.1.2 Flujos Síncronos

Las llamadas de las funciones de la aplicación de escritura del host a bajo nivel no vuelven hasta que todos los datos hayan alcanzado la lógica de la aplicación de usuario en la FPGA. El retorno suele indicar que todos los datos han llegado a la FIFO, o la lógica de recepción desarrollada.

Los flujos síncronos deben evitarse en aplicaciones de alto ancho de banda, por dos razones:

- El canal físico permanece sin usar durante parte del tiempo. En la mayoría de los casos, esto conduce a un golpe de rendimiento de ancho de banda significativo.
- El FIFO entre la lógica de la aplicación y la lógica del núcleo IP en el FPGA puede desbordarse durante estos intervalos de tiempo.

A pesar de estos inconvenientes, los flujos síncronos son útiles cuando el momento en el que se recogieron los datos en FPGA es importante. En particular, las interfaces tipo memoria requieren un flujo síncrono.[2]

4.3.2 Flujo FPGA-host

4.3.2.1 Flujo Asíncrono

El flujo asíncrono en este caso llena los buffer DMA del host continuamente. Mientras el devfile esté abierto, los datos están disponibles y disponemos de espacio en los buffer.

4.3.2.2 Flujo Síncrono

En este caso, mientras que la aplicación del host no envíe una petición de lectura, la aplicación lógica de la FPGA no envía los datos. Tiene el mismo problema que el flujo inverso con el ancho de banda. También las interfaces tipo memoria en este caso, de la misma manera requieren flujos síncronos.

4.4 IPCore

Xillybus es una plataforma multiusos. Como tal, el núcleo de IP suministrado se configura fácilmente para satisfacer requisitos específicos en términos del número de flujos, su dirección, atributos relacionados con su rendimiento y consumo de recursos.

Para generar fácilmente el IP Core que necesitamos, accedemos al enlace, http://www.xillybus.com/ipfactory/ .

Figura 39. Página inicial de IP Core Factory.

Iniciamos sesión en la página y pulsamos *"Add a new core"* . Volveremos a la página de la figura XX, pero ya podremos crear el IP Core.

En esta página le damos un nombre al IP Core, ya que podemos tener varios, y luego seleccionamos:

- **Target device family**: la placa en la que vamos a usar el IP Core, en nuestro caso *"Xilinx Zynq-7000 (ZedBoard)"*
- **Initial template**: si queremos que tenga una proyecto ejemplo, o esté vacía vamos a dejarla por defecto en este caso
- **Operating system**: con qué sistema operativo va a conectarse, en este proyecto usaremos Linux

Pulsamos *"Create!"* . Y pasaremos a la siguiente página.

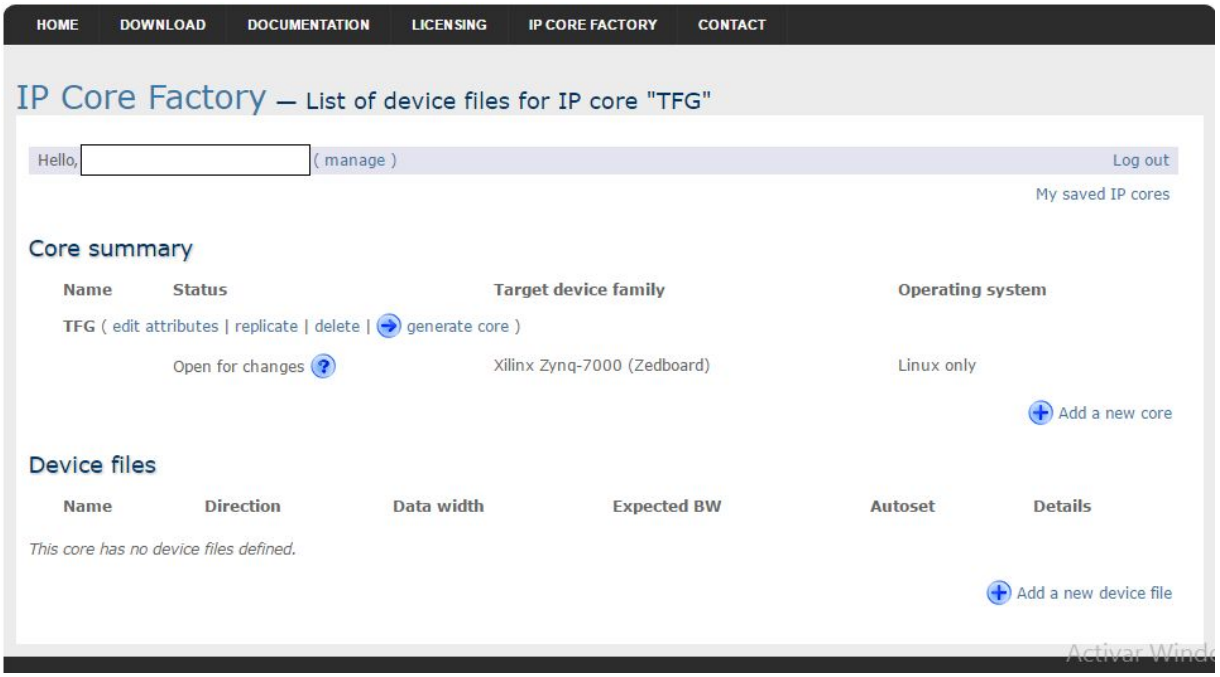

Figura 40. Listado de device files, en el core TFG.

Pulsamos ahora *"Add a new device file"*.

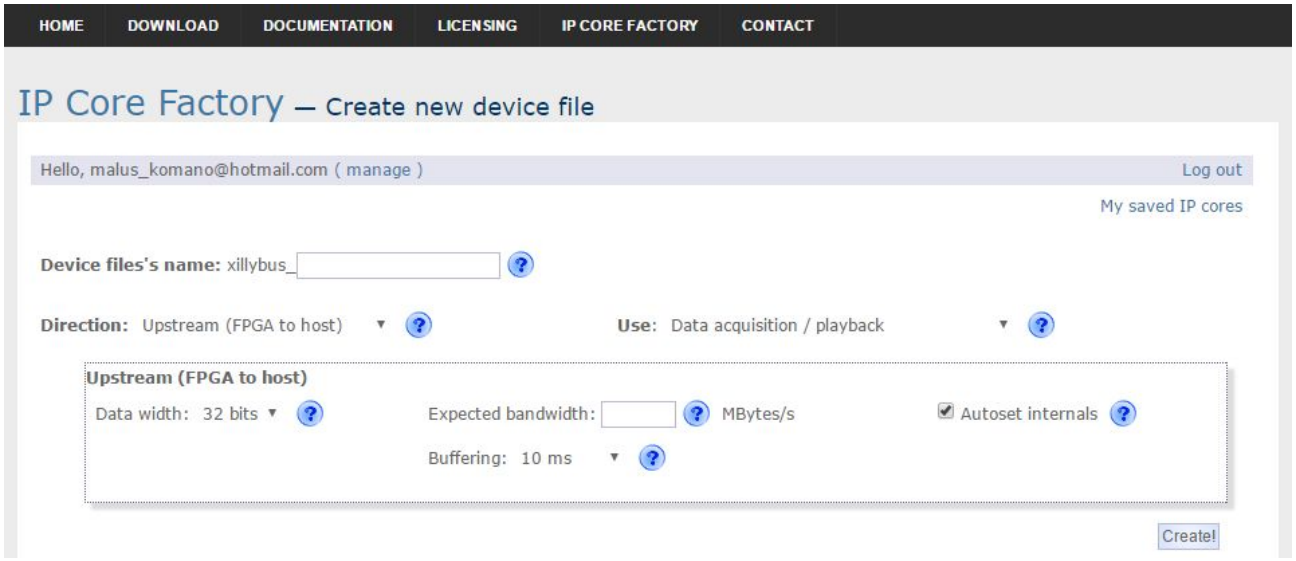

Figura 41. Creación de un device file.

En esta nueva ventana tenemos varias opciones:

- **Device file's name**: nombre que vamos a darle a este devfile, para localizarlo en la lógica que se generará.
- **Direction**: cuál será el sentido del tráfico de datos:
  - Upstream : FPGA-host
  - Downstream: host-FPGA

- Bidirectional

- **Use**: El uso previsto del devfile influye en las asignaciones del buffer y en el control de flujo cuando se comprueba "Autoset internals". También influye en los warnings generados para cada devfile :
  - Frame grabbing / video playback : se elige el tamaño de búfer y control de flujo para facilitar un flujo de datos continuo, por parte del Core
  - Data acquisition / playback : se elige el tamaño de búfer y control de flujo para facilitar un flujo de datos continuo. por parte del Core.
  - Data exchange with coprocessor : exige alto rendimiento pero no un flujo continuo
  - Bridge to external hardware : solo necesitamos saber el régimen binario del periférico y su ancho.
  - Data for in-silicon logic verification:
  - Command and status:
  - Short message transport
  - Address/data interface (5/16/32 address bit) : El devfile es navegable. Las líneas de dirección están expuestas en el lado FPGA y el devfile es síncrono en ambas direcciones
  - General purpose:.
- **Data Width**: El ancho de datos corresponde a la palabra extraída o escrita en las FIFO de la FPGA. Las opciones permitidas son: .
  - 32 bits
  - 16 bits
  - 8 bits

  Los flujos que requieren un alto rendimiento de ancho de banda deben configurarse para utilizar un ancho de datos de 32 bits. La razón es que las palabras se transportan a través de las rutas de datos internas de Xillybus a la velocidad del reloj del bus.

  El transporte de una palabra de 8 bits usa la misma ranura de tiempo que una palabra de 32 bits, haciéndola cuatro veces más lenta.

  Una mala elección del ancho de los datos puede conducir a un comportamiento no deseado (si el ancho del bus es de 32 bits, 4 byte, si enviamos una palabra de 3byte sin rellenar con ‘0’, es posible que se quede esperando continuamente). [ref]

- **Expected bandwidth**: dependiendo de la aplicación a usar tendremos que saber a qué velocidad debemos transmitir los datos. Este valor también se usa para que la página nos dé unos warnings si usamos demasiados recursos.

- **Buffering time**: Se elige una expectativa aproximada del tiempo de almacenamiento en búfer para

los flujos en tiempo real (es decir, que se utilizan para aplicaciones de captura o reproducción). [ref]

- 10 ms
- 20 ms
- 50 ms
- 100 ms
- 200 ms
- 300 ms
- 500 ms
- 1000 ms
- Maximum

En el caso de los que no necesitan transporte en tiempo real (todos, salvo frame grabbing, y data adquisition) tienen un buffer por defecto de 10 ms .

- **Autoset Internals**: Esta opción indica si queremos que se ajusten los buffers DMA internos, y el control de flujo, automáticamente (respecto a las opciones anteriores seleccionadas) o si queremos manejarlos de manera manual (poco recomendable si no se es experto en este tema). [ref]

Seleccionando los parámetros que necesitemos para cada devfile, con esto podremos configurarlos para nuestras necesidades. Al acabar de configurarlo, pulsamos *"Create!"* y ya lo tendremos disponible en el Core.

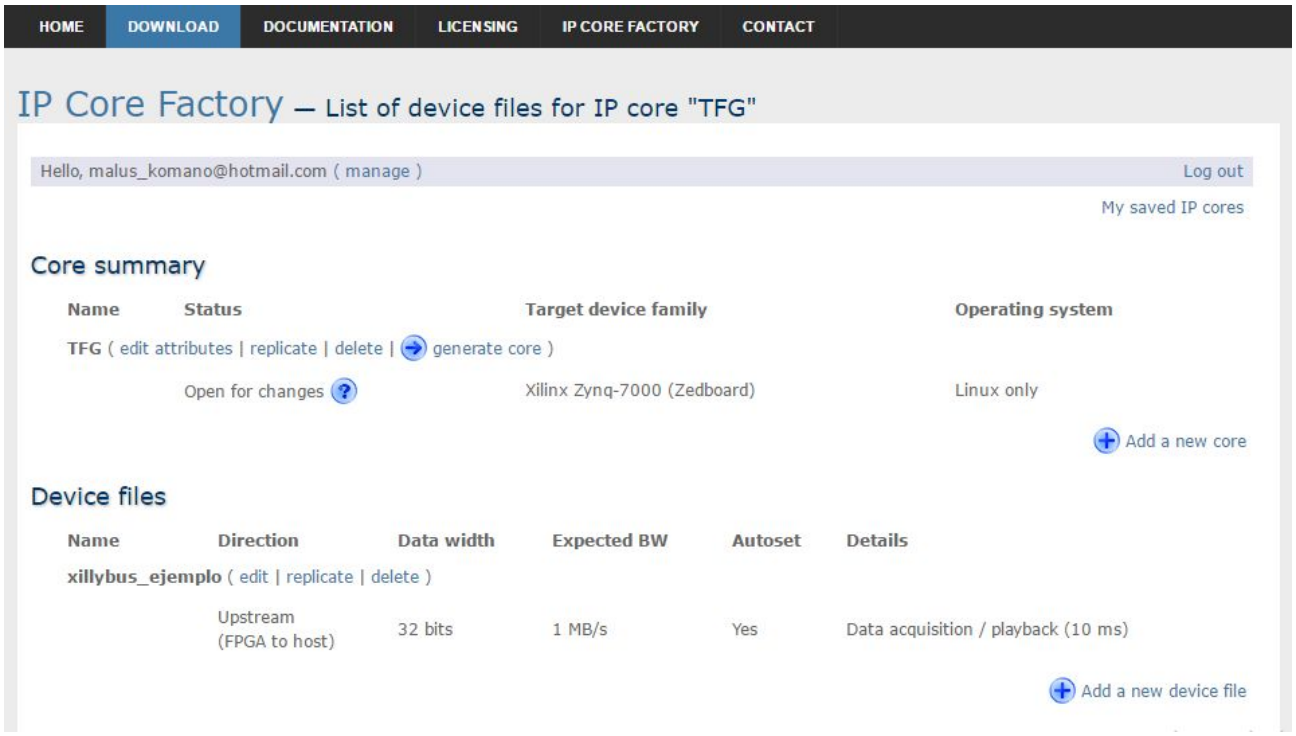

Figura 42. Listado de device files en el core TFG, con un device file de ejemplo.

Como vemos en la figura 42 nos aparece ahora el devfile configurado, (se ha generado uno cualquiera) y nos permite editarlo copiarlo o eliminarlo, según nos interese.

Si se elige la opción de *"Initial template"* al crear el Core, ya tendremos una serie de devfiles creados y configurados que se sabe que funcionan sin problema. Para el desarrollo del proyecto se usará esta plantilla, aunque habrá algunos que no se usen.

5 BLOQUE A

5.1 Introducción

Este primer tutorial consistirá en una aproximación al uso del device file de Xillybus, read_8. Este device file, es de uso general, y de dirección UpStream (FPGA-host). Se ha elegido uno 8 bits para poder representar los resultados en forma de código ASCII y que sea más fácil para el usuario la interpretación de los mismos.

Veremos distintas funcionalidades que podemos darle para que un usuario aprenda el uso de diseño VHDL y el uso del mismo device file, en Xillybus en UpStream.

Como primer bloque realizaremos un módulo que envíe una serie de caracteres ASCII en Upstream. Visualizamos estos caracteres desde el host para comprobar que el funcionamiento es correcto.

Posteriormente, se le añadirá una nueva funcionalidad. Consistirá en la adición de un bloque, que mantendrá los datos un tiempo limitado (en principio 3 segundos).

Una segunda funcionalidad adicional será el uso de botones (los 5 disponibles en PL), para seleccionar el rango de caracteres ASCII que enviaremos al host.

- boton_up = **[** ‘0’ **:** ‘9’ **]**.
- boton_down = **[** ‘:’ **:** ‘@’ **]**.
- boton_right = **[** ‘A’ **:** ‘Z’ **]**.
- boton_left = **[** ‘a’ **:** ‘z’ **]**.
- boton_center = **[** ‘\’ **:** ‘ ` ’ **].**

La última modificación que realizaremos sobre el bloque, será el uso de los Switches disponibles con la función de modificar la velocidad de variación. En la misma se mantienen los datos con el mismo carácter en el host.

- Switch_one → 6 segundos.
- Switch_two → 8 segundos.
- Switch_three → 1 segundo.
- Switch_four → 0,5 segundos.

En este tutorial se pretende que el usuario que lo realice desarrolle capacidades sobre:

- Como se recibir los datos desde el host con xillybus.
- Uso de periféricos de la tarjeta, como seleccionadores de modo de funcionamiento.

5.2 Bloque inicial

5.2.1 Preparación del proyecto

Para iniciar el desarrollo del bloque crearemos un proyecto en Vivado. En la página inicial de Vivado, pulsamos *"Create New Project"* (figura 11), *"Next"* e introducimos el nombre de proyecto y la localización del mismo (figura 43).

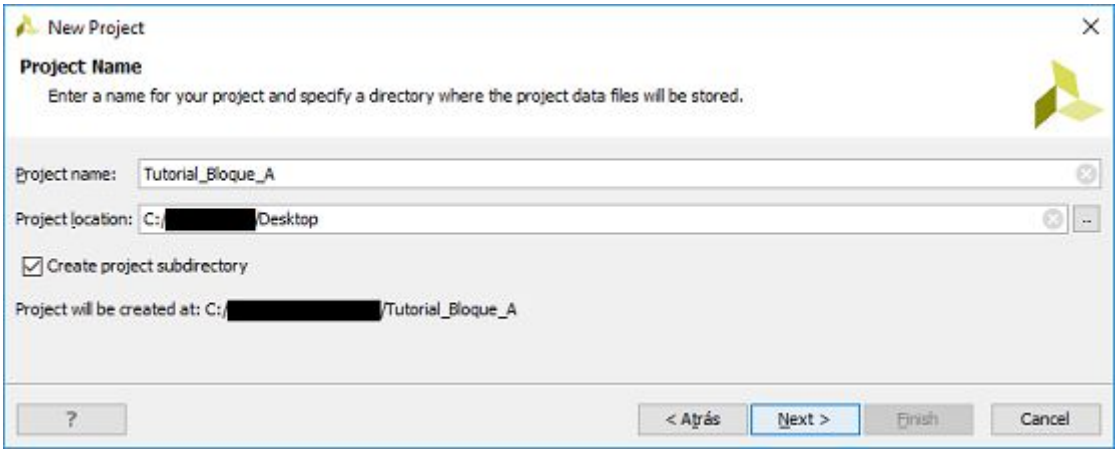

Figura 43. Creación del proyecto Tutorial_Bloque_A.

En la página siguiente seleccionaremos el tipo de proyecto a crear, en este caso concreto elegiremos la opción *"RTL Project"*. En la siguiente página seleccionamos la tarjeta sobre la que se implementará el proyecto. En el menú select, escogemos *"Board"* (figura 44).

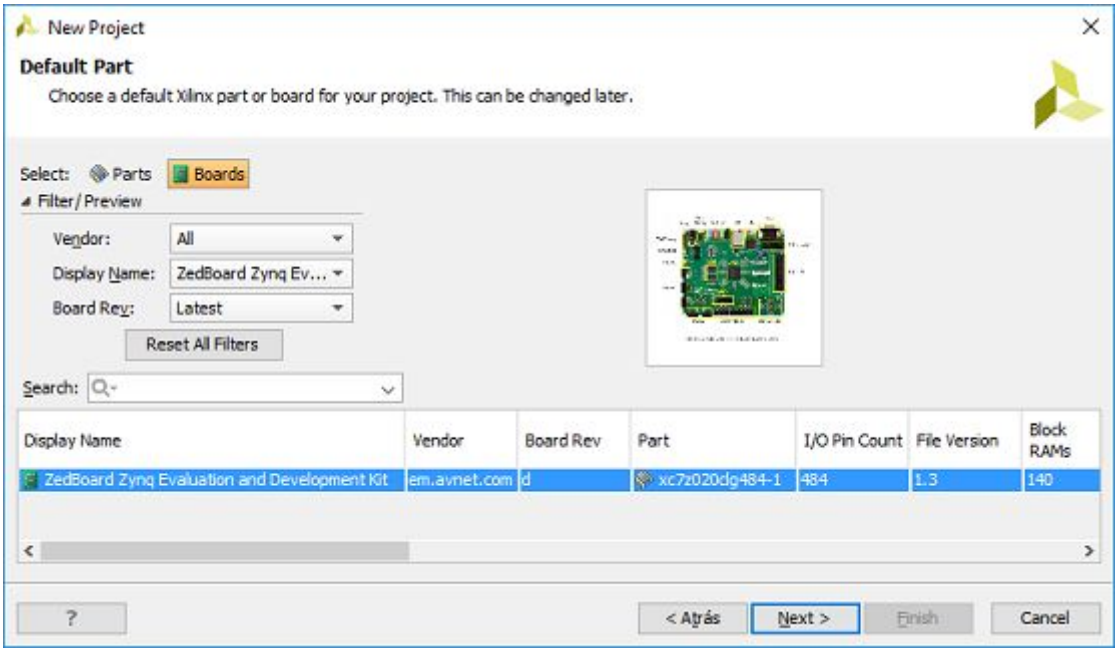

Figura 44. Selección de tarjeta en proyecto Tutorial_Bloque_A.

Entre las posibles tarjetas disponibles, marcamos *"ZedBoard Zynq Evaluation and Development Kit"* que es el kit que disponemos para el desarrollo del proyecto, pulsamos *"Next"* y terminamos el proyecto.

5.2.2 Bloque_A

Se nos abrirá directamente el proyecto. Vamos al *"Flow Navigator"*, en *"Project Manager"* (figura 13) y pulsamos *"Add Sources"*. Ahora aparece la figura 45, donde seleccionamos el tipo de código fuente que necesitamos, en este caso *"Add or create design source"* .

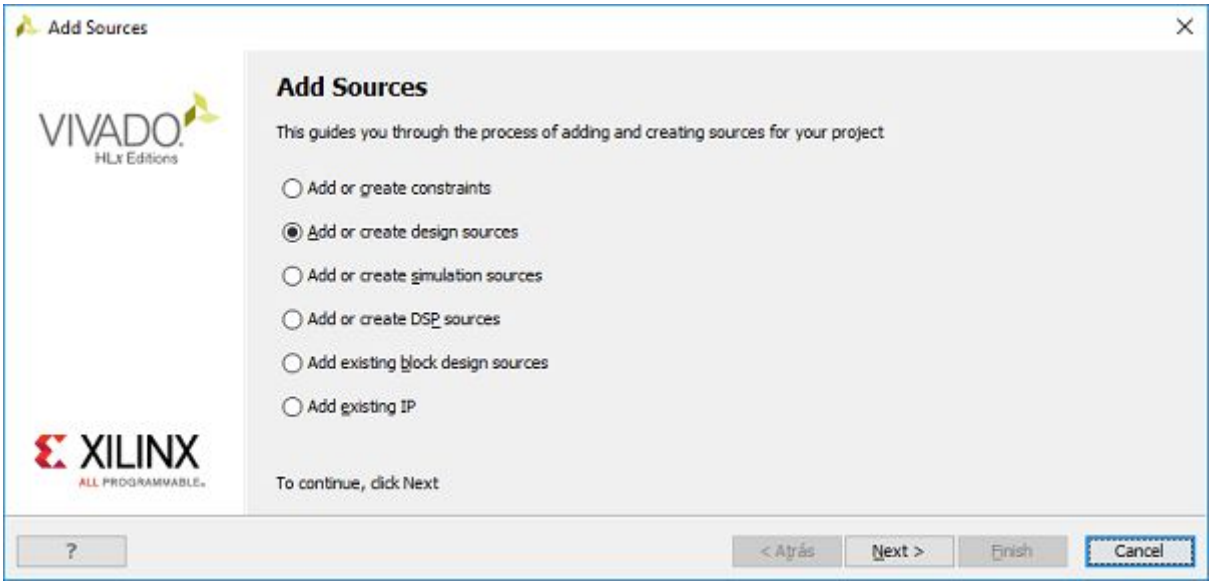

Figura 45. Selección de tipo de código fuente Bloque_A.

En la siguiente página pulsamos *"Create File"* y nos saldrá una ventana en la que se debe seleccionar el lenguaje de diseño en el que se escribirá el archivo, su nombre y localización. Introducimos los datos que aparece en la figura 46.

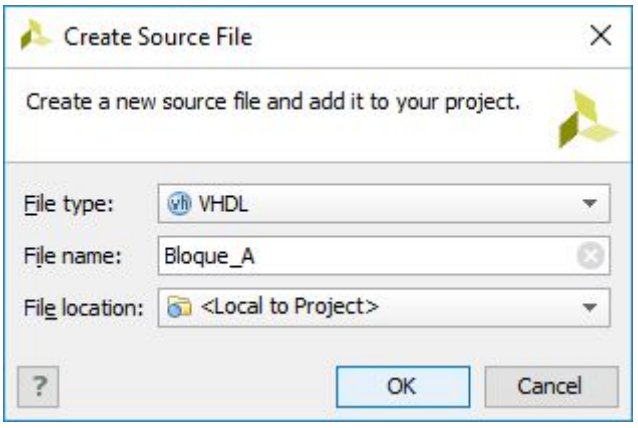

Figura 46. Creación de código fuente Bloque_A.

Cuando ya lo tenemos, pulsamos *"Ok"* y nos aparece una ventana para configurar las señales del Bloque_A generado. Configuraremos estas señales como aparece en la figura 47.

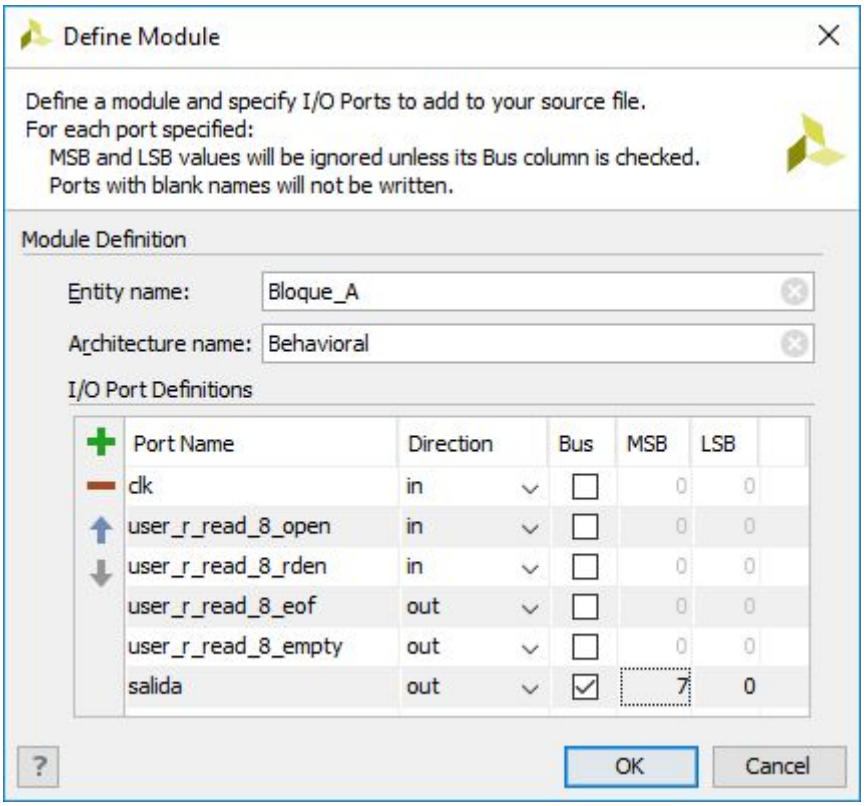

Figura 47. Definición de señales I/O Bloque_A.

Para conseguir la funcionalidad inicial, debemos añadir varias señales auxiliares antes de empezar a diseñar los procesos vhdl. Necesitaremos 2 señales auxiliares para la modificación de la salida y otras dos para el mantenimiento de la salida si no se cumplen las condiciones para modificarla.

Las señales que usaremos son las que vemos en la figura 48.

```
signal salida_sig : unsigned ( 7 downto 0 ) := ( others => '0' );
signal salida_sig_aux : unsigned ( 7 downto 0 ) := ( others => '0' );

signal salida_hold : unsigned ( 7 downto 0 ) := ( others => '0' );
signal salida_hold_sig : unsigned ( 7 downto 0 ) := ( others => '0' );
```

Figura 48. Señales auxiliares Bloque_A.

Para poder usar estas señales, debemos descomentar las líneas 27, 31 y 32 del código generado por la plantilla de Vivado, de esta manera se permite el uso de las librerías IEEE.NUMERIC_STD.ALL y UNISIM.VComponents.all. Estas librerías nos permite añadir componentes del catálogo de Vivado, tipos de señales como unsigned, entre otras, y operaciones matemáticas sobre determinados tipos de señales.

La arquitectura del módulo la dividiremos en 2 procesos:

- **p_seq**
- **p_comb**

El p_seq, es el proceso secuencial del módulo (figura 49). En este proceso actualizaremos las variables de

salida dependiendo de los valores de algunas señales de entrada provenientes de Xillybus.

```
p_seq : process ( clk )
begin
    if rising_edge(clk) then
        if user_r_read_8_open = '1' then
            user_r_read_8_eof <= '0';
            if user_r_read_8_rden = '1' then
                salida <= std_logic_vector( salida_sig );
                salida_sig_aux <= salida_sig;
                salida_hold <= salida_sig;
                user_r_read_8_empty <= '1';
            else
                salida <= std_logic_vector( salida_hold_sig );
                salida_sig_aux <= salida_hold_sig;
                salida_hold <= salida_hold_sig;
                user_r_read_8_empty <= '0';
            end if;
        else
            salida <= "00100001";
            salida_sig_aux <= "00100001";
            salida_hold <= "00100001";
            user_r_read_8_empty <= '0';
            user_r_read_8_eof <= '1';
        end if;
    end if;
end process;
```

Figura 49. Proceso secuencial Bloque_A.

Analizaremos el proceso desde el *"if / end if"* más exterior al más interior. Este proceso se actualizará en cada ciclo de reloj.

El más exterior sirve para diferenciar si el archivo *"/dev/xillybus_read_8"* está abierto o cerrado. Cuando está cerrado la señal *"user_r_read_8_open"* se vuelve '0', y como vemos en la figura 49, mantendremos la salida en el primer carácter '!'. Además de indicar a la interfaz Xillybus que no envíe más datos al host (user_r_read_8_empty = '0') y que el archivo *"/dev/xillybus_read_8"* está cerrado (user_r_read_8_eof = '1').

En caso de que el archivo esté abierto, mantendremos la señal *"user_r_read_8_eof"* = '0', para reflejar al Xillybus que el archivo está abierto. También comprobaremos si el Xillybus puede recibir un nuevo dato con la señal *"user_r_read_8_rden"*. Cuando está activa ('1') indica que puede recibir un nuevo dato, actualizamos las señales de salida, hold y activamos la señal *"user_r_read_8_empty"*. En caso opuesto, se mantienen las señales con el valor anterior y mantenemos *"user_r_read_8_empty"* a '0'.

En la asignación de la señal salida se usa una conversión forzada de las señales *"salida_sig"* y *"salida_hold_sig"* al tipo std_logic_vector. Esto se debe a que como vemos en la figura 48, estas han sido definidas como tipo unsigned, para poder realizar operaciones matemáticas (en este caso la suma) y simplificar el código.

El proceso p_comb, en este primer módulo, es bastante simple como muestra la figura 50.

```
p_comb : process ( salida_sig_aux )
begin
    salida_hold_sig <= salida_sig_aux;
    if salida_sig_aux = "01111110" then
        salida_sig <= "00100001";
    else
        salida_sig <= salida_sig_aux + 1;
    end if;
end process;
```

Figura 50. Proceso combinacional Bloque_A

En este proceso actualizamos el valor de "*salida_sig"*. Dependiendo si se ha llegado al carácter 'z' o no, en caso de llegar al caracter 'z', volvemos al carácter inicial '!'. Si no *"salida_sig"* será el siguiente carácter de la tabla ASCII que corresponda. Además mantendremos el valor de salida en *"salida_hold_sig"*, para usar esta señal en el proceso anterior.

Una vez tenemos los dos procesos vamos al Flow Navigator y, en el menú Synthesis, pulsamos *"Run Synthesis"* y comprobamos que no tenemos Warnings ni Errores.

5.2.3 Test Bench de Bloque_A

Para comprobar que el módulo funciona correctamente, crearemos un archivo de simulación. Para ello seleccionamos la opción *"Add or create simulation source"* de la figura 45.

Seguiremos el mismo proceso que para la creación del código fuente de Bloque_A, solo que cambiaremos el nombre del bloque a *"tb_Bloque_A"* y en la selección de señales (figura 47) lo dejaremos en blanco, pese al mensaje de aviso que nos envía Vivado.

Para evitar posibles problemas también descomentamos las líneas de la plantilla generada, como en la sección anterior. Para simular el Bloque_A, añadiremos el módulo al test_bench, las señales auxiliares necesarias y la constante de tiempo a utilizar para la variación del reloj, usaremos 10 ns pero no tiene que ver con el reloj de la ZedBoard .(figura 51).

```
component Bloque_A
    port (clk   : IN std_logic;
          user_r_read_8_open : in std_logic;
          user_r_read_8_rden : in std_logic;
          user_r_read_8_empty : out std_logic;
          user_r_read_8_eof : out std_logic;
          salida : out std_logic_vector(7 downto 0)
        );
end component;

signal clk : std_logic := '0';

signal user_r_read_8_open : std_logic := '0';
signal user_r_read_8_rden : std_logic := '0';
signal user_r_read_8_empty : std_logic := '0';
signal user_r_read_8_eof : std_logic := '0';

signal salida : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

constant clk_period : time := 10 ns;
```

Figura 51. Inclusión de Bloque_A en test_bench.

Para comprobar el correcto funcionamiento de Bloque_A usaremos tres procesos distintos para simplificar el diseño del test_bench:

- Un proceso para el reloj, que genere una señal de reloj de periodo seleccionado en la constant *"clk_period"* (figura 51).
- Otro proceso para la señal *"user_r_read_8_rden"* que emule los ciclos de lectura de Xillybus variando la señal anterior a un intervalo de arbitrario.
- Un último proceso de estímulo en el que "abrimos" tras un tiempo el devfile (*user_r_read_8_open* = '1'), y posteriormente lo cerramos (*user_r_read_8_open* = '0').

Podemos observar estos procesos en la figura 52.

```
p_clock : process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;
```

```
p_rden : process
begin
    user_r_read_8_rden <= '0';
    wait for clk_period;
    user_r_read_8_rden <= '1';
    wait for clk_period;
end process;
```

```
stim_process: process
begin
    wait for 70 ns;
    user_r_read_8_open <= '1';
    wait for 300 ns;
    user_r_read_8_open <= '0';
    wait;
end process;
```

Figura 52. Procesos tb_Bloque_A.

Además de lo anterior añadimos al component (figura 51) estos procesos (figura 52), y conectarlo a las señales anteriores (figura 51).Tras esto podemos lanzar la simulación. Vamos al Flow Navigator, en el menú Simulation, y clicamos en *"Run Simulation"*.

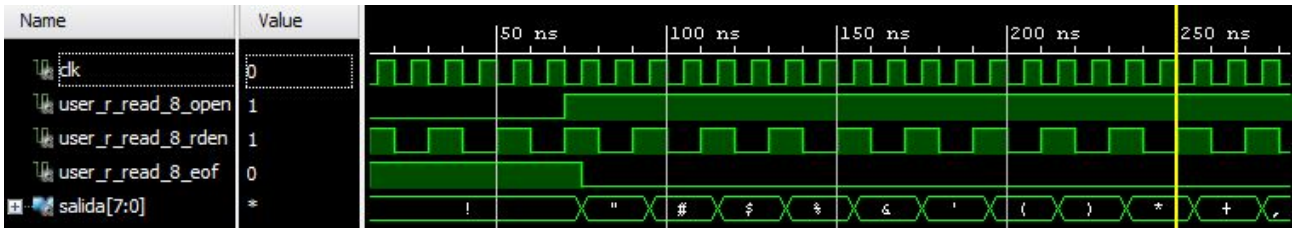

Figura 53. Primera parte simulación Bloque_A.

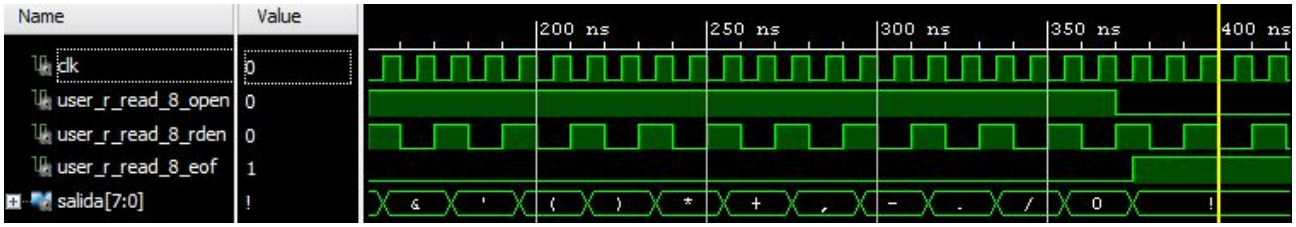

Figura 54. Segunda parte simulación Bloque_A.

En las figuras 53 y 54 se refleja que la funcionalidad que buscamos ha sido implementada con éxito.

Cuando la señal *"user_r_read_8_open"* está a bajo nivel, *"user_r_read_8_eof"* está a nivel alto, y la salida no varía. Pero cuando está a nivel alto, la señal se va actualizado con *"user_r_read_8_rden"*, y cuando vuelve a nivel bajo, se repite el estado anterior. También se observa la variación de *"user_r_read_8_empty"* como se esperaba.

5.2.4 Comprobación en la FPGA.

Para confirmar que la simulación y el bloque se corresponden con la realidad, implementaremos el módulo con la interfaz Xillybus, generamos el .bit y comprobamos en la FPGA la funcionalidad. Como indicamos al final del capítulo Xillybus, habremos generado un core con una plantilla inicial.

Primero accedemos a http://xillybus.com/ipfactory/ nos logueamos y descargamos el core TFG, y el kit de desarrollo desde http://xillybus.com/xillinux (para ZedBoard). Descomprimimos ambos archivos, los pasamos al directorio de desarrollo del proyecto y abrimos el proyecto xillydemo, contenido en la carpeta del kit de desarrollo.

Una vez abierto el proyecto, copiamos el archivo Bloque_A, del anterior proyecto contenido en .../Tutorial_Bloque_A/Tutorial_Bloque_A.srcs/source_1_new y lo pasamos a la carpeta del nuevo proyecto xillydemo .../vhdl/src.

Ahora desde Vivado, vamos a Flow Navigator, menú Project Manager, clicamos en *"Add sources"*. Seleccionamos la opción "*Add or create design source"*, en la siguiente ventana pulsamos *"Add file"*, y buscamos en la dirección anterior el archivo Bloque_A.vhdl.

Para usar esta demo con Vivado, lo primero que debemos hacer es comentar las líneas de código 11, 12 y 13, correspondientes a la declaración de las señales PS_CLK, PS_PORB y PS_SRSTB a la entity de xillydemo. Y descomentar las líneas 273, 274 y 275, que hacen que las señales anteriores sean señales internas, y no señales externas al módulo completo.

Para este módulo en principio también tendremos que comentar la FIFO de 8x2048 que trae la demo, ya que no vamos a usarla. Borramos las líneas 487-501, y el component FIFO 8x2048, y la línea "*attribute syn_black_box of fifo_8x2048: component is true;*".

Una vez realizadas todas las operaciones anteriores, añadimos el componente de Bloque_A, al módulo xillybus, y lo incluimos en el proceso general del archivo.

```
bloq : Bloque_A
port map (clk   => user_clk,
          user_r_read_8_open => user_r_read_8_open,
          user_r_read_8_rden => user_r_read_8_rden,
          user_r_read_8_empty => user_r_read_8_empty,
          user_r_read_8_eof => user_r_read_8_eof,
          salida => user_r_read_8_data
    );
```

Figura 55. Conexión Bloque_A con xillybus.

Después de añadir el Bloque_A, pulsamos *"Run Synthesis"*, cuando acabe *"Run Implementation"* y por último *"Generate Bitstream"* . Tendremos una serie de Warnings, e incluso 1 Warning Crítico, pero debemos ignorarlos, se deben simplemente a que se necesita una licencia específica para que no aparezcan, y el no uso del bloque vga_fifo que en este proyecto no es necesario.

Cuando se termina el proceso de generación del Bitstream, conectamos a nuestro PC la tarjeta SD que configuramos en capítulos anteriores. Buscamos el archivo xillydemo.bit, en la dirección ../vhdl/vivado/xillydemo_runs/impl_1 , copiamos el archivo en la tarjeta y la sacamos del PC de manera segura.

Conectamos la tarjeta SD a la ZedBoard, conectamos el cable USB-microUSB, al puerto UART y a un puerto USB del PC, y encendemos la placa. Abrimos el VMware, entramos en la máquina virtual, tendremos que volver a hacer clic derecho y *"connect"* en el icono correspondiente (figura 32) . Posteriormente accederemos a un terminal y ejecutamos el siguiente comando:

"sudo minicom /var/log/syslog/ttyACM0"

Accedemos de esta manera a la FPGA, estando dentro ejecutamos los siguientes comandos (ya que estaremos solo con visión de terminal):

"cd /xillybus/demoapps"

"make"

"./streamread /dev/xillybus_read_8"

Si se han seguido los pasos correctamente y no hay ningún error, el resultado (remarcado en azul) debe ser el que se muestra en la figura 56. Se muestran también (señalados en rojo) los comandos realizados anteriormente.

```
root@localhost:~# cd xillybus/demoapps/
[1]+  Exit 127                AT S7=45 S0=0 L1 V1 X4  (wd: ~)
(wd now: ~/xillybus/demoapps)
root@localhost:~/xillybus/demoapps# make
make: Warning: File `Makefile' has modification time 1.4e+09 s in the future
gcc  -g -Wall -O3 memwrite.c -o memwrite
gcc  -g -Wall -O3 memread.c -o memread
gcc  -g -Wall -O3 streamread.c -o streamread
gcc  -g -Wall -O3 streamwrite.c -o streamwrite
gcc  -g -Wall -O3 -pthread fifo.c -o fifo
make: warning:  Clock skew detected.  Your build may be incomplete.
root@localhost:~/xillybus/demoapps#  ./streamread /dev/xillybus_read_8
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnop
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-^_`
```

Figura 56. Resultado Bloque_A en ZedBoard.

Los archivos de código fuente completos de Bloque_A.vhdl y tb_Bloque_A que se han usado para la realización de esta sección se añadirán al final del proyecto como Anexos al final de este trabajo.

5.3 Primera modificación

Con esta modificación queremos mostrar cómo podemos variar la velocidad de cambio en la variación de la salida.

Para incluir la nueva funcionalidad al sistema, añadiremos un nuevo bloque al proyecto Tutorial_Bloque_A, Flow Navigator, Project Manager, *"Add Source"* ... En este caso, llamaremos al bloque div_frec, y le pondremos como entrada clk, y enable, y le añadiremos la señales auxiliares necesarias para realizar la funcionalidad que se espera.

```
entity div_frec is
    Port ( clk : in STD_LOGIC;
           enable : out STD_LOGIC);
end div_frec;

architecture Behavioral of div_frec is

signal enable_aux : std_logic := '0';
signal enable_aux_sig : std_logic := '0';

signal cont :  unsigned ( 14 downto 0 ):= ( others => '0' );
signal cont_sig : unsigned ( 14 downto 0 ):= ( others => '0' );
```

Figura 57. Bloque div_frec.

Las señales auxiliares añadidas se necesitan para generar un contador ("*cont*", "*cont_sig*") que marque cuándo debe activarse la salida, mediante las otras dos señales ("*aux*", "*aux_sig*"). El contador llegará hasta 30.000, lo que en última instancia provocará un cambio en la salida cada 3 segundos aproximadamente.

Los procesos que incluiremos en la arquitectura del bloque, consistirán en:

- Un proceso síncrono para la asignación de la salida.

- Y un proceso combinacional que active la señal cuando se llegue a 30.000 y la mantenga dos ciclos de reloj a nivel alto para asegurar que siempre se mantiene activa con user_r_read_8_rden pasa a nivel alto.

La señal "*enable*" la usaremos para conectarla al Bloque_A, al que añadiremos una señal con el mismo nombre para (figura 58) provocar la variación de la salida cada 3 segundos.

```
if ( user_r_read_8_rden = '1' and enable = '1')then
```

Figura 58. Modificación primera Bloque_A.

Para simplificar la unión con el bloque anterior, crearemos un nuevo archivo de código fuente. Lo nombraremos top_Bloque_A_primera_mod, que tendrá las mismas señales que el Bloque_A. Incluimos los dos bloques diseñados anteriormente y una señal auxiliar "*enable_aux*". Esta señal la conectaremos entre los bloques div_frec y Bloque_A (previamente modificado) figura 59.

```
architecture Behavioral of top_Bloque_A_primera_mod is

    component div_frec
        Port ( clk : in std_logic;
               enable : out std_logic);
    end component;

    component Bloque_A_primera_mod
        PORT ( clk : in std_logic;
               enable : in std_logic;
               user_r_read_8_open : in std_logic;
               user_r_read_8_rden : in std_logic;
               user_r_read_8_eof : out std_logic;
               user_r_read_8_empty : out std_logic;
               salida: out std_logic_vector (7 downto 0)
             );
    END component;
```

Figura 59. top_Bloque_A_primera_mod.

Después de tener los 3 bloques preparados, comprobaremos si el módulo funciona correctamente. Añadimos un nuevo fichero de simulación tb_top_Bloque_A_primera_mod, en el que usaremos los mismos procesos de estímulo. Sólo añadiremos más tiempo al nivel alto de "*user_r_read_8_open*" y aumentamos la constante de tiempo.

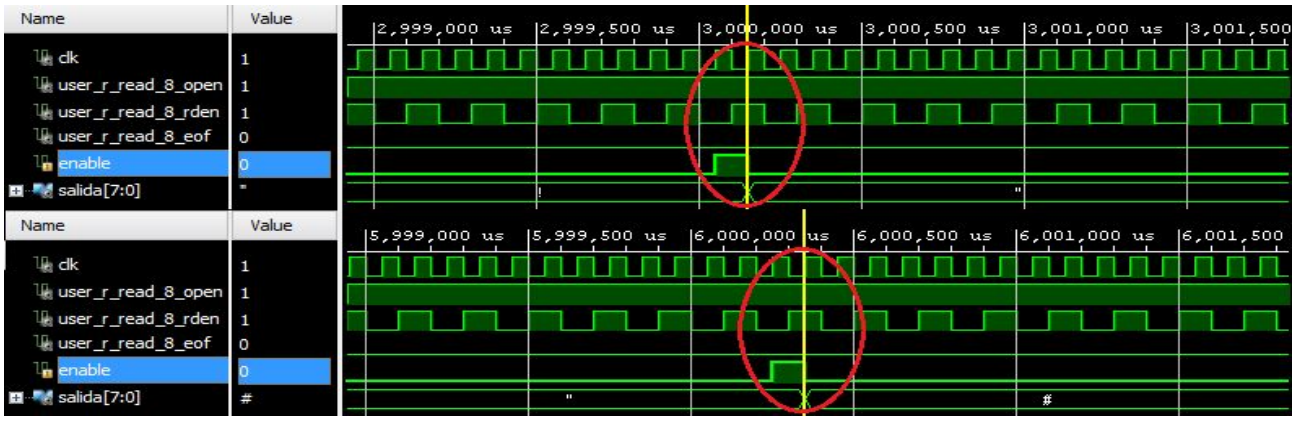

Figura 60. Simulacion tb_top_Bloque_A_primera_mod.

Observamos en la figura 60 que se cumple la funcionalidad que esperamos obtener (concretamente las partes marcadas con sendos círculos rojos).

Tras la comprobación, en la simulación volvemos a abrir el xillydemo, como proyecto nuevo sin el Bloque_A. Y añadimos los 3 bloques diseñados anteriormente al proyecto, e incluimos en la arquitectura xillydemo el bloque top_Bloque_A_primera_mod.

Volvemos a borrar la FIFO de 8 bits de ancho, añadimos el bloque al diseño de la demo proporcionada y conectamos las señales de la misma manera que la sección anterior. Tras esto generamos el .bit, Siguiendo el proceso del apartado anterior, *"Flow Navigator"*, *"Program and Debug"*, y clicamos en "*Generate Bitstream*" que realizará todas las operaciones anteriores de Síntesis e Implementación.

Pasamos el archivo .bit a la tarjeta SD, y conectamos esta a la ZedBoard, además del cable USB-microUSB. Repetimos el proceso de la sección anterior, encendemos la máquina virtual, entramos a la ZedBoard y ejecutamos los comandos descritos en la sección anterior.

"sudo minicom /dev/ttyACM0"

"cd xillybus/demoapps/"

"make"

"./streamread /dev/xillybus_read_8"

Nos saldrá una línea completa del carácter inicial, y cada 3 segundos aproximadamente pulsamos enter y variará según el código ASCII ascendente, obteniendo el resultado que vemos en la figura 61.

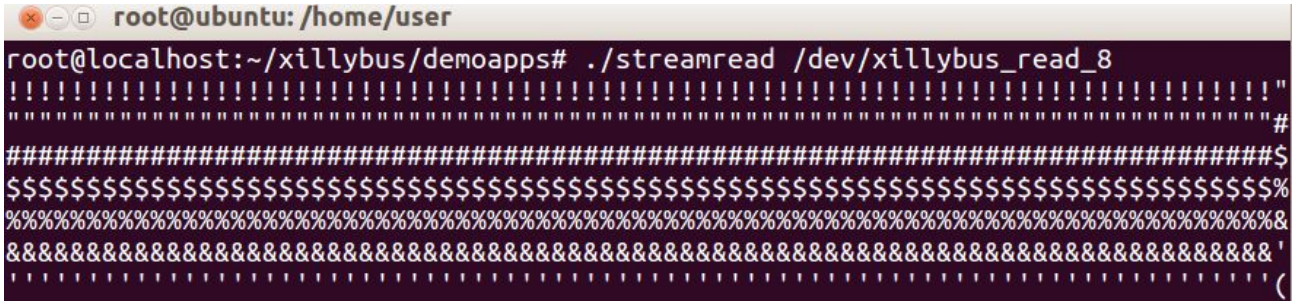

Figura 61. Resultados en FPGA, top_Bloque_A_primera_mod.

Los archivos generados en esta modificación se añadirán en un anexo como Bloque_A_primera_mod.vhdl, div_frec.vhdl, top_Bloque_A_primera_mod.vhdl y tb_Bloque_A_primera-mod.vhdl.

5.4 Segunda modificación

En esta nueva variación partiremos de la modificación anterior. Generamos los archivos de código fuente Bloque_A_segunda_mod y top_Bloque_A_segunda_mod. En principio con las mismas señales que tenían en la sección anterior. Después cambiamos el nombre del bloque top_Bloque_A_primera_mod, por top_Bloque_A_segunda_mod.

Esta modificación consiste en añadir un control de los caracteres que salen de la FPGA hacia el host, para ello usaremos los 5 botones que proporciona la ZedBoard (figura 62).

Figura 62. Botones en la ZedBoard.

En la plantilla de xillydemo vienen añadidos en sus respectivos terminales, en el archivo .xdc. Este archivo nos indica donde están conectados estos botones en xillydemo. Concretamente son los que se muestra en la figura 63.

```
# On-board Left, Right, Up, Down, and Select Pushbuttons

set_property -dict "PACKAGE_PIN N15 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[19]"]
set_property -dict "PACKAGE_PIN R18 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[20]"]
set_property -dict "PACKAGE_PIN T18 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[21]"]
set_property -dict "PACKAGE_PIN R16 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[22]"]
set_property -dict "PACKAGE_PIN P16 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[23]"]
```

Figura 63. Localización de las variables de los botones.

Tras esto, pasamos al archivo Bloque_A_segunda_mod.vhdl, donde copiaremos el contenido del bloque en la anterior sección. Sacamos el código anterior del proyecto y añadimos este. Hacemos la misma operación con el bloque top_Bloque_A_* .

En el archivo top_Bloque_A_segunda_mod.vhdl tenemos que añadir una nueva entrada, a la que llamaremos "*botones*" que será un vector de 5 bits. Esto lo añadimos escribiendo en la declaración de los puertos en el archivo la siguiente línea.

"*botones : in STD_LOGIC_VECTOR (4 downto 0);*"

Actualizamos el xillydemo con la nueva variable del top_Bloque_A_segunda_ mod, y en la asignación de señales al bloque, le conectamos la señal que aparece en la figura (la del xdc), PS_GPIO(23 downto 19).

Ahora vamos al archivo Bloque_A_segunda_mod, y le añadimos 5 señales de entrada de un bit(una por

botón), las llamaremos:

- boton_up
- boton_down
- boton_right
- boton_left
- boton_center

Además de las anteriores, crearemos una serie de variables auxiliares a ellos para mantener el dato de su activación y poder usarlos de manera de diseño en dos procesos. Para mayor facilidad de comprensión, estas señales recibirán los siguientes nombres:

- boton_x_aux
- boton_x_aux_sig

Donde x serán las distintas posiciones como en la lista anterior (up, down…). Estas variables nos servirán para poder recoger el valor de los botones, y poder mantenerlo, en caso necesario.

En el proceso secuencial del bloque sólo añadiremos una asignación de las variables anteriores para asegurarnos que no se producen glitches, (también se añadirán a la lista de sensibilidad del proceso las boton_x_aux_sig) siguiendo este formato:

“boton_x_aux <= boton_x_aux_sig;”

En el combinacional, tenemos que añadir, las entradas del bloque, y las señales auxiliares boton_x_aux, a lista de sensibilidad del proceso combinacional. Ahora diferenciaremos 6 casos distintos en el proceso, según se hayan pulsado o no los botones:

- Si no se ha pulsado ningún botón, mantenemos el caso de la modificación anterior.
- Si se pulsa el boton_up imprimimos el rango **[** ‘0’ **:** ‘9’ **]**.
- Si se pulsa el boton_down imprimimos el rango **[** ‘:’ **:** ‘@’ **]**.
- Si se pulsa el boton_right imprimimos el rango **[** ‘A’ **:** ‘Z’ **]**.
- Si se pulsa el boton_left imprimimos el rango **[** ‘a’ **:** ‘z’ **]**.
- Si se pulsa el boton_center imprimimos el rango **[** ‘\’ **:** ‘`’ **].**

Para implementar esto, usaremos una serie de if/else anidados, el primer caso será simplemente un if que entre solo si no se pulsa ningún botón. Para los siguientes casos, usaremos el formato que vemos en la figura 64.

```
elsif ( ( boton_up = '1' or boton_up_aux = '1' ) and boton_down = '0' and boton_right = '0'
                    and boton_left = '0' and boton_center = '0') then
```

Figura 64. Formato selección de modo, segunda modificación.

La parte de "*(boton_x = '1' or boton_x_aux = '1')*", para entrar, o mantenernos en este modo en caso de que no se pulse ningún otro botón. Como veremos en la siguiente figura,al pulsar el botón se mantendrá a '1' la señal auxiliar del mismo.

```
elsif ( ( boton_up = '1' or boton_up_aux = '1' ) and boton_down = '0' and boton_right = '0'
                    and boton_left = '0' and boton_center = '0') then
    if ( boton_up = '1' ) then
        salida_sig <= "00110000";
        boton_center_aux_sig <= '0';
        boton_up_aux_sig <= '1';
        boton_down_aux_sig <= '0';
        boton_right_aux_sig <= '0';
        boton_left_aux_sig <= '0';
    elsif salida_sig_aux = "00111001" then
        salida_sig <= "00110000";
        boton_center_aux_sig <= boton_center;
        boton_up_aux_sig <= boton_up_aux;
        boton_down_aux_sig <= boton_down;
        boton_right_aux_sig <= boton_right;
        boton_left_aux_sig <= boton_left;
    else
        salida_sig <= salida_sig_aux + 1;
        boton_center_aux_sig <= boton_center;
        boton_up_aux_sig <= boton_up_aux;
        boton_down_aux_sig <= boton_down;
        boton_right_aux_sig <= boton_right;
        boton_left_aux_sig <= boton_left;
    end if;
```

Figura 65. Ejemplo modo de funcionamiento segunda modificación.

Esta figura se repetirá para cada botón, pero analizaremos el funcionamiento de uno como ejemplo. El modo de funcionamiento se divide en 3 niveles:

- Pulsación (boton_x = '1' , '0' en los otros dos):en este if entramos cuando el botón está siendo pulsado, pasamos a la salida el inicio del nuevo rango, activamos la señal auxiliar del modo, para mantenernos en él mientras no se pulse otro botón y el resto de señales auxiliares a '0' para evitar problemas.

- Reinicio de contador: en caso de que se llegue al final del rango, volveremos a empezar por el carácter inicial del rango. También se actualiza el valor de las variables auxiliares, para en caso de que se pulse otro botón salte, y en caso contrario sigamos en nuestro modo actual.

- Mantenedor: en caso de que no ocurra ningún evento, pasar al siguiente valor del rango, y actualizar el valor de las variables como en el caso anterior.

En cualquier otro caso (pulsación de varios botones distintos), se mantendrá la salida y se actualizarán los valores hasta que solo quede un botón o ninguno.

Tras implementar estos modos de funcionamiento en el proceso combinacional, actualizaremos el top_Bloque_A_segunda_mod, el component Bloque_A_segunda_mod.

Ahora pasamos al Tutorial Bloque_A borramos los archivos anteriores y añadimos los archivos de esta modificación. Generamos un archivo test_bench para comprobar esta funcionalidad. Lo llamaremos

tb_Bloque_A_segunda_mod.

Copiaremos el código del test_bench de la sección anterior, y lo pasaremos al nuevo. En este caso ampliaremos la duración del nivel alto de la señal "*user_r_read_8_open*" a 100 segundos después del último estímulo. Actualizaremos el component del test_bench de top_Bloque_A_primera_mod, al de esta sección.

En el proceso de estímulos añadiremos una activación del botón up, a los 7 segundos, (manteniéndolo medio segundo) y una activación del botón left a los 15 siguientes (durante 210 ms). Y obtenemos los siguientes resultados en el caso de la activación del botón left. (figura 65).

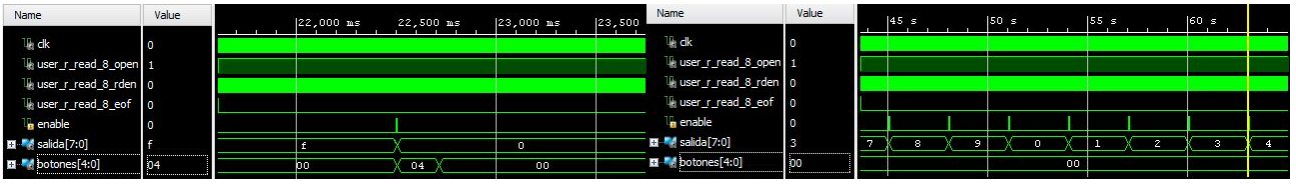

Figura 66. Resultados tb_ Bloque_A_segunda_mod.

Observamos que el resultado en este caso es el esperado, se mantiene el rango, del botón pulsado. Y que el cambio de modo no provoca problemas (figura 66).

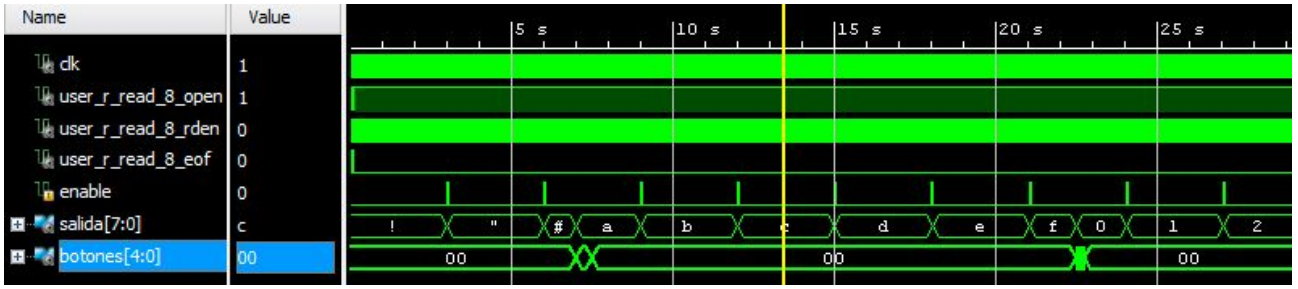

Figura 67. Cambio de modos en tb_Bloque_A_segunda_mod.

A la vista de estos resultados, podemos pasar los dos archivos al proyecto con el xillydemo, en el que ya hemos actualizado el xillydemo.vhdl . Generamos el .bit como en secciones anteriores.

"sudo minicom /dev/ttyACM0"

"cd xillybus/demoapps/"

"make"

"./streamread /dev/xillybus_read_8"

Repetimos el proceso de anteriores secciones, xillydemo.bit a la SD, e inicio de la máquina virtual. Conectamos el cable USB-microUSB, encendemos la tarjeta, y conectamos desde la máquina virtual a la tarjeta. Y ejecutamos los comandos anteriormente descritos para comprobar los resultados en la FPGA.

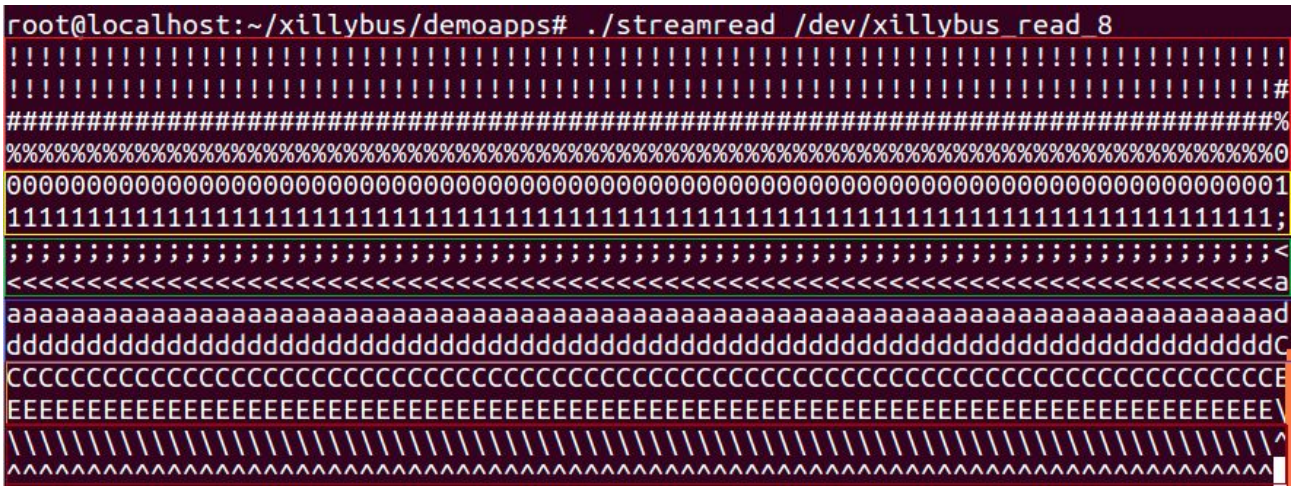

Figura 68 . Resultados top_Bloque_A_segunda_mod en FPGA.

En la figura se observa como el cambio en los botones produce la variación en la salida:

- En el recuadro rojo tenemos el modo por defecto.
- En el recuadro amarillo el modo del boton_up.
- En el recuadro verde el modo del boton_down.
- En el recuadro azul el modo del boton_left.
- En el recuadro naranja el modo del boton_right.
- En el recuadro marrón el modo del boton_center.

Como en las secciones anteriores se proporcionará un Anexo, con los códigos fuente de Bloque_A_segunda_mod.vhdl, top_Bloque_A_segunda_mod.vhdl y div_frec.vhdl.

5.5 Tercera modificación

En esta última modificación sobre el Bloque_A, vamos añadir una funcionalidad para modificar la velocidad a la que la salida anterior varía. Para este desarrollo partiremos de la sección anterior.

Para indicar el modo de velocidad que deseamos, lo haremos mediante unos Switches disponibles en la ZedBoard (figura 68), aunque solo usaremos 4 de ellos.

Figura 69. Switches disponibles en la ZedBoard.

Los Switches que se observan en la figura son los disponibles en la tarjeta y los remarcados en azul, los que usaremos en este desarrollo.

En este caso, como en el anterior, debemos buscar las variables a las que corresponden estos Switches dentro de la lógica programable. También están disponibles en el archivo .xdc del proyecto plantilla xillydemo (

figura 69).

```
# On-board Slide Switches

set_property -dict "PACKAGE_PIN F22 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[11]"]
set_property -dict "PACKAGE_PIN G22 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[12]"]
set_property -dict "PACKAGE_PIN H22 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[13]"]
set_property -dict "PACKAGE_PIN F21 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[14]"]
set_property -dict "PACKAGE_PIN H19 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[15]"]
set_property -dict "PACKAGE_PIN H18 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[16]"]
set_property -dict "PACKAGE_PIN H17 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[17]"]
set_property -dict "PACKAGE_PIN M15 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[18]"]
```

Figura 70. Puertos de los Switches en xillydemo.

Para la realización de esta modificación usaremos por plantilla el archivo div_frec.vhdl. A este archivo le añadimos 4 señales de entrada:

- "*Switch_one : in std_logic;*"
- "*Switch_two : in std_logic;*"
- "*Switch_three : in std_logic;*"
- "*Switch_four : in std_logic;*"

Nos servirán para indicar distintas velocidades a las que se activará la señal "*enable*". Estas señales deben añadirse a la lista de sensibilidad del proceso combinacional, el proceso secuencial no variará en absoluto. Seguiremos un proceso similar al de la sección anterior para implementar los distintos modos de velocidad en esta modificación.

```
if (  Switch_one = '0' and Switch_two = '0' and  Switch_three = '0' and
      Switch_four = '0' ) then
    if cont = "00001110101001100000" then
        enable_aux <= not(enable_aux_sig);
        cont_sig <= cont + 1;
    elsif cont >= "00001110101001100010" then
        enable_aux <= '0';
        cont_sig <= "00000000000000000010";
    else
        enable_aux <= enable_aux_sig;
        cont_sig <= cont + 1;
    end if;
```

Figura 71. Modo default div_frec_tercera_mod.

Como vemos en la figura 71 el modo default estará activo cuando no haya ningún Switch activo, y será el propuesto en la primera modificación.

En el caso de los distintos modos nuevos introducidos, al ser el Switch marcado permanentemente mientras se requiera, no necesitamos variables auxiliares para mantener el modo como la modificación anterior. La variación con el default, consiste solo en la variación del valor de la señal "*cont*" . Esto provocará que la señal enable se active en distintos períodos de tiempo según el modo.

Los distintos modos implementarán los siguientes intervalos de tiempo:

- Modo default: 3 segundos.
- Modo Switch_one: 6 segundos.
- Modo Switch_two: 8 segundos.
- Modo Switch_three: 1 segundo.
- Modo Switch_four: 0,5 segundos.

En caso de introducir más de un Switch, sin importar qué combinación se produzca, el tiempo parará a ser de aproximadamente 10 segundos.

En este caso, los distintos modos se componen en los siguientes niveles:

- Límite alcanzado: en este nivel cambiaremos de valor el nivel de la señal "enable" en la práctica implica ponerlo a nivel alto. Pero no vamos a 0 en el contador debido a que para que sea leído correctamente por el resto de bloques, se mantiene 2 ciclos de reloj más.
- Mantenedor : aquí mantenemos el valor de la señal "enable" durante dos ciclos (la diferencia con el límite de contador del nivel anterior. Tras ese lapso de tiempo pasamos a '0' la señal "*enable*".
- Default: en caso de no estar en ninguno de los dos posibles eventos, se mantiene la señal "*enable*" como esta y se aumenta el contador.

Tras realizar los distintos modos con esos niveles, tenemos completo el archivo div_frec_tercera_mod.vhdl. Ahora pasamos al archivo top_Bloque_A_tercera_mod, al que añadiremos la señal de entrada:

"*Switches : in std_logic_vector (3 downto 0);*"

Además, actualizamos el component div_frec, con el actual y le añadimos la señal correspondiente en el uut_div_frec, conectada a la entrada que se ha añadido al top_Bloque_A_tercera_mod.

Antes de modificar el xillydemo, variamos el test_bench, y realizaremos los siguientes ajustes:

- Le añadiremos la señal correspondiente "*Switches : in std_logic_vector (3 downto 0);*", al component top_Bloque_A_tercera_mod (y le cambiaremos de nombre)
- Añadiremos una "signal" con el mismo nombre y tipo para poder manipular los estímulos de esa señal.
- Cambiaremos el proceso de estímulos proporcionando en vez de variaciones en los botones, variaciones en los Switches (si bien no variaría la comprobación, no modificarlos simplifica el resultado.

Simulamos este proyecto, y obtenemos el siguiente resultado (figura 72).

Figura 72. Resultado simulación tercera modificación.

Como se refleja en la figura con la activación de los distintos Switches, se modifica la frecuencia de activación de la señal "*enable*". Y cuando no hay ningún Switch activo, la señal vuelve a su frecuencia original.

Tras esta comprobación procederemos a la modificación del archivo xillydemo.vhdl, modificamos el nombre del component top_Bloque_A_segunda_mod a top_Bloque_A_tercera_mod, y añadimos la nueva señal, tanto a la declaración como a su uso dentro de la arquitectura del xillydemo. En la arquitectura le asignamos la señal PS_GPIO(14 downto 11), que son los Switches que usaremos, tal y como se observa en la figura.

Añadimos los archivos de código anteriores de esta sección al proyecto xillydemo, y generamos el .bit para su comprobación en la FPGA, el resultado es similar al de la figura pero con variación en la frecuencia de cambio de los datos.

6 BLOQUE B

6.1 Introducción

En este segundo tutorial continuaremos el tema tratado en el el Bloque A. Seguiremos usando el device file de uso general read_8. pero también añadiremos el device file write_8, para incluir la dirección host hacia xillybus (Downstream). Para poder procesar datos que enviemos desde el host hacia la FPGA, y posteriormente leer el resultado como en el bloque anterior.

Se ha elegido la el device file write_8 para, como en el caso anterior, usar los caracteres ASCII. En esta ocasión, además de la lectura, los datos serán obtenidos procesando la información que se enviará a la FPGA por write_8.

En este caso, para que el usuario entienda el funcionamiento del device file de escritura en xillybus, tendremos 2 bloques. Uno inicial más sencillo y otro más elaborado como profundización en el tema.

El primer bloque consistirá en una variación de minusculas a mayusculas y se dejará sin cambios todo lo demás, que se transmita por write_8 a la FPGA.

El segundo bloque, será de mayor complejidad. Consistirá en realizar una pequeña calculadora con 3 operaciones de unidades solo. Tiene una aplicación que posee dos bloques diferenciados:

- Un bloque de lectura donde el formato de introducción de los datos será el siguiente: *"x_x"*, en caso de introducirse caracteres distintos de números o espacio, se escribirá *"Error."* en el device file read_8.
- Un bloque de procesamiento. Dependiendo del Switch activado se transmitirá a la interfaz read_8, con un determinado formato, donde los datos leídos en en el bloque anterior se:
    - sumarán
    - restarán
    - multiplicarán

6.2 Bloque Inicial

6.2.1 Preparación del proyecto

Como en el bloque anterior, lo primero que debemos hacer es generar un proyecto desde Vivado, eligiendo las mismas opciones, que en el Bloque A, pero lo llamaremos Tutorial Bloque B.

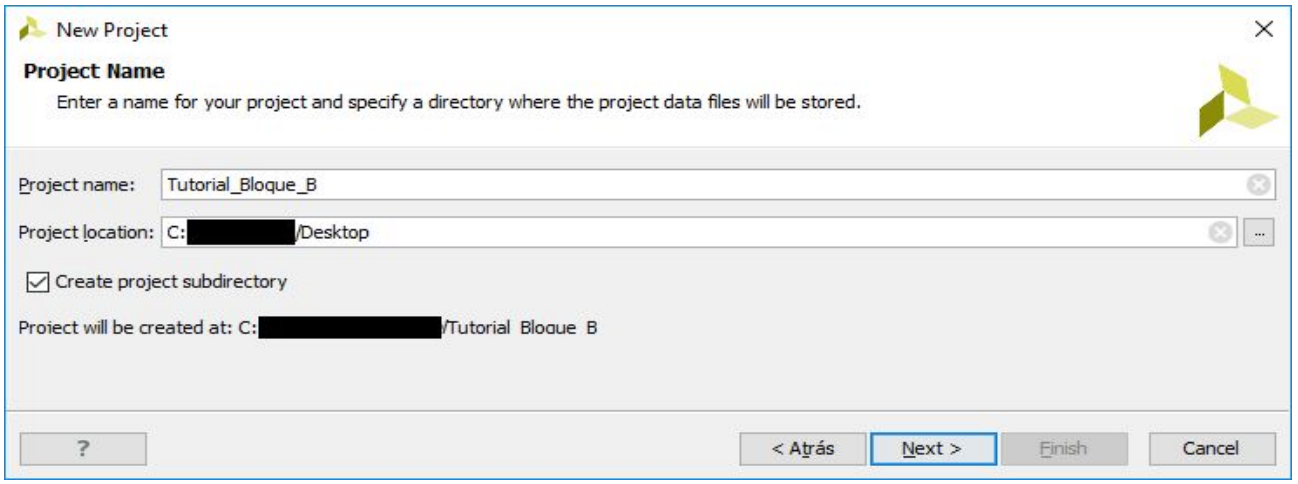

Figura 73. Creación Tutorial Bloque B.

6.2.2 Bloque B

Tras la creación del proyecto, vamos al menú *"Flow Navigator"*, *"Project Manager"*, y pulsamos en la opción *"Add Sources"*. Aparecerá una ventana como la obtenida en la figura 45, seleccionando la misma opción que en la figura.

Pulsaremos *"Create Files"*, lo nombramos Bloque B y definiremos las siguientes señales en la entidad, como veremos en la figura 74.

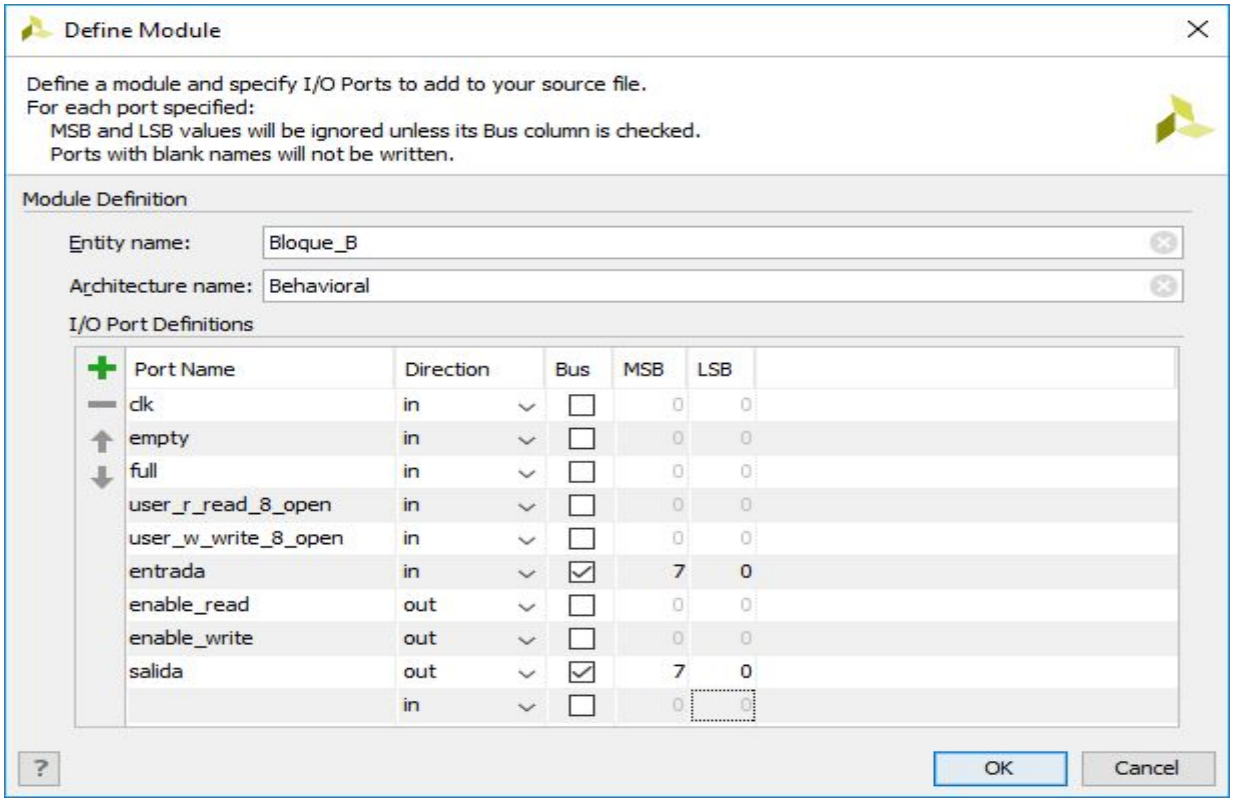

Figura 74. Definición de señales del Bloque B.

Este primer bloque tiene como funcionalidad convertir los caracteres ASCII enviados por el device file write_8, de minúscula a mayúsculas, y mandarlos por device file read_8. Para ello necesitaremos las señales

auxiliares que vemos en la figura 75.

```
signal cont_r : unsigned ( 15 downto 0 ) := ( others => '0' );
signal cont_r_sig : unsigned ( 15 downto 0 ) := ( others => '0' );

signal cont_w : unsigned ( 15 downto 0 ) := ( others => '0' );
signal cont_w_sig : unsigned ( 15 downto 0 ) := ( others => '0' );

signal stop : std_logic := '0';
signal stop_sig : std_logic := '0';

signal enable_read_aux : std_logic := '0';
signal enable_write_aux : std_logic := '0';

signal enable_read_hold : std_logic := '0';
signal enable_write_hold : std_logic := '0';

signal salida_aux : std_logic_vector( 7 downto 0 ) := ( others => '0' );
signal salida_hold : std_logic_vector (7 downto 0 ) := ( others => '0' );
signal salida_aux_hold : std_logic_vector (7 downto 0 ) := ( others => '0' );
```

Figura 75. Señales auxiliares Bloque B.

Las señales *"cont_r"*, *"cont_w"* junto con las *"..._sig"* son dos contadores distintos, el contador *"cont_r"*, para read, y el *"cont_w"* para write.

La señal *"stop"* indica que se ha terminado de escribir.

Las señales *"enables_write/read_aux"* que sirven para indicar cuándo se esta listo para recibir datos o enviarlos,

También procedemos a diseñar este bloque con un diseño en dos procesos:

- **p_seq**
- **p_comb**

```
p_seq: process ( clk )
begin
    if rising_edge(clk) then
        cont_r <= cont_r_sig;
        cont_w <= cont_w_sig;
        stop <= stop_sig;
        if ( enable_write_aux = '1' ) then
            salida <= salida_aux;
            salida_hold <= salida_aux;
        else
            salida <= salida_aux_hold;
            salida_hold <= salida_aux_hold;
        end if;
        if ( user_r_read_8_open = '0' and user_w_write_8_open = '0' and empty = '0'
                and full = '0') then
            enable_write <= enable_write_aux;
            enable_write_hold <= enable_write_aux;
            enable_read <= enable_read_aux;
            enable_read_hold <= enable_read_aux;
        elsif ( user_r_read_8_open = '0' and user_w_write_8_open = '0' and empty = '1'
                        and full = '0') then
            enable_write <= enable_write_aux;
            enable_write_hold <= enable_write_aux;
            enable_read <= '0';
            enable_read_hold <= '0';
        else
            enable_write <= '0';
            enable_write_hold <= '0';
            enable_read <= '0';
            enable_read_hold <= '0';
        end if;
    end if;
end process;
```

Figura 76. Proceso Secuencial Bloque B.

Ahora analizaremos el proceso secuencial del Bloque B. Las variables *"cont_r"*, *"cont_w"* y *"stop"* las asignamos directamente con sus variables de seguimiento.

En el primer *"if / else"* actualizamos el valor de *"salida"* y *"salida_hold"* al nuevo valor, *"salida_aux"*, en caso de que se active la señal *"enable_write_aux"*, de lo contrario se mantendrá, usando la señal *"salida_aux_hold"*.

En el segundo *"if / else"* actualizamos el valor de las variables de enable de escritura y lectura, que enviaremos al top Bloque B, (lo veremos en el epígrafe siguiente). Estas señales sirven para marcar cuando se lee o se escribe, además de otras señales enable hold que usaremos para mantener el estado de dicha señal en caso necesario. Este supuesto se añade para el caso en que se active la señal *"empty"* .

```
p_comb : process (entrada, full,cont_r, cont_w, empty,full, enable_write_hold,
                  enable_read_hold,salida_hold, stop )
begin
    salida_aux_hold <= salida_hold;
    if ( full = '0' and stop = '0') then
        if cont_w = "0111010100110000" then
            enable_write_aux <= '1';
            cont_w_sig <= ( others => '0' );
        else
            enable_write_aux <= '0';
            cont_w_sig <= cont_w +1;
        end if;
        stop_sig <= '0';
    else
        stop_sig <= '1';
        enable_write_aux <= '0';
        cont_w_sig <= ( others => '0' );
    end if;
```

Figura 77. Proceso Combinacional Bloque B, parte 1.

En este proceso combinacional vemos que tenemos en principio la señal de mantenimiento de salida, "salida_aux_hold", se conecta con la señal de *"salida_hold"*, para que en el siguiente ciclo de reloj siga cumpliendo su función correctamente.

Como observamos la figura 77, vemos el primer *"if / else"* que usamos para actualizar la variable *"enable_write_aux"*. Para ello comprobamos en primer lugar que no se ha activado la señal *"stop"* (usada para indicar que se ha terminado de transmitir los datos al device file read_8), ni la *"full"* (que la memoria no esté llena). Si entramos en el if interior, comprobamos el *"cont_w"*, si llegamos al límite marcado, reiniciamos el contador y activamos el *"enable…"*, en caso contrario, dejamos a nivel bajo en *"enable…"*, y sumamos uno al *"cont_w_sig"*.

En caso opuesto, la señal *"stop_sig"* se activará, *"enable_write_aux"* y *"cont_w_sig"* se bajarán a cero. Si *"stop"* y *"full"* están a '0' comprobaremos un contador, que al llegar a un determinado número, se reinicie y se active la señal *"enable_write_aux"*, y *"stop_sig"* pasa a '0'.

```
if ( empty = '0' ) then
    if cont_r = "0111010100110000" then
        enable_read_aux <= '1';
        cont_r_sig <= (others => '0');
    else
        enable_read_aux <= '0';
        cont_r_sig <= cont_r +1;
    end if;
    stop_sig <= '0';
else
    stop_sig <= '1';
    enable_read_aux <= '0';
    cont_r_sig <= ( others => '0' );
end if;
```

Figura 78. Proceso combinacional Bloque B parte 2.

En este caso el siguiente *"if / else"* si la señal empty está a bajo nivel, comprobamos un contador, si se cumple la condición se activa la señal *"enable_read_aux"* y se reinicia el contador, *"cont_r_sig"*, y *"stop_sig"* se pone a bajo nivel.

Si *"empty"* está a nivel alto, se activará la señal *"stop_sig"* y se reiniciará el contador (*"cont_r_sig"*), y se pondrá a bajo nivel *"enable_read_aux"*.

```
case entrada is
when "01000001" => salida_aux <= "01100001"; when "01100001" => salida_aux <= "01000001";
when "01000010" => salida_aux <= "01100010"; when "01100010" => salida_aux <= "01000010";
when "01000011" => salida_aux <= "01100011"; when "01100011" => salida_aux <= "01000011";
when "01000100" => salida_aux <= "01100100"; when "01100100" => salida_aux <= "01000100";
when "01000101" => salida_aux <= "01100101"; when "01100101" => salida_aux <= "01000101";
when "01000110" => salida_aux <= "01100110"; when "01100110" => salida_aux <= "01000110";
when "01000111" => salida_aux <= "01100111"; when "01100111" => salida_aux <= "01000111";
when "01001000" => salida_aux <= "01101000"; when "01101000" => salida_aux <= "01001000";
when "01001001" => salida_aux <= "01101001"; when "01101001" => salida_aux <= "01001001";
when "01001010" => salida_aux <= "01101010"; when "01101010" => salida_aux <= "01001010";
when "01001011" => salida_aux <= "01101011"; when "01101011" => salida_aux <= "01001011";
when "01001100" => salida_aux <= "01101100"; when "01101100" => salida_aux <= "01001100";
when "01001101" => salida_aux <= "01101101"; when "01101101" => salida_aux <= "01001101";
when "01001110" => salida_aux <= "01101110"; when "01101110" => salida_aux <= "01001110";
when "01001111" => salida_aux <= "01101111"; when "01101111" => salida_aux <= "01001111";
when "01010000" => salida_aux <= "01110000"; when "01110000" => salida_aux <= "01010000";
when "01010001" => salida_aux <= "01110001"; when "01110001" => salida_aux <= "01010001";
when "01010010" => salida_aux <= "01110010"; when "01110010" => salida_aux <= "01010010";
when "01010011" => salida_aux <= "01110011"; when "01110011" => salida_aux <= "01010011";
when "01010100" => salida_aux <= "01110100"; when "01110100" => salida_aux <= "01010100";
when "01010101" => salida_aux <= "01110101"; when "01110101" => salida_aux <= "01010101";
when "01010110" => salida_aux <= "01110110"; when "01110110" => salida_aux <= "01010110";
when "01010111" => salida_aux <= "01110111"; when "01110111" => salida_aux <= "01010111";
when "01011000" => salida_aux <= "01111000"; when "01111000" => salida_aux <= "01011000";
when "01011001" => salida_aux <= "01111001"; when "01111001" => salida_aux <= "01011001";
when "01011010" => salida_aux <= "01111010"; when "01111010" => salida_aux <= "01011010";
when "01011011" => salida_aux <= "01111011"; when "01111011" => salida_aux <= "01011011";
when "01011100" => salida_aux <= "01111100"; when "01111100" => salida_aux <= "01011100";
                                             when others => salida_aux <= entrada;
                                             end case;
```

Figura 79. Proceso combinacional Bloque B parte 3.

Esta última parte es donde procesamos los datos que introducimos en device file write_8. En este Bloque, el procesamiento consistirá simplemente en una variación de los caracteres ASCII de letras minúsculas a mayúsculas y viceversa.

Tras terminar de analizar el código volvemos al menú *"Flow Navigator"*, *"Synthesis"*, y *"Run Synthesis"*. En este caso, Vivado no debe darnos ningún Warning.

6.2.3 Test_Bench de Bloque B

Para comprobar que el módulo funciona correctamente, añadimos un archivo de simulación.*"Flow Navigator", "Project Manajer", "Add Sources",* y seleccionamos la opción *"Add or create simulation source"* que se observa en la figura 45.

Seguiremos el mismo proceso que para la creación del test bench del bloque A, solo que cambiaremos el

nombre del bloque a *"tb_Bloque_B"* .

```
architecture Behavioral of tb_Bloque_B is
component Bloque_B
    Port ( clk : in STD_LOGIC;
           entrada : in std_logic_vector(7 downto 0);
           empty : in std_logic;
           full : in std_logic;
           user_r_read_8_open : in std_logic;
           user_w_write_8_open : in std_logic;
           enable_read : out std_logic;
           enable_write : out std_logic;
           salida : out std_logic_vector(7 downto 0) );
end component;

signal clk : STD_LOGIC := '0';
signal entrada : std_logic_vector(7 downto 0) := ( others => '0' );
signal empty : std_logic := '0';
signal full : std_logic := '0';
signal user_r_read_8_open : std_logic := '0';
signal user_w_write_8_open : std_logic := '0';
signal enable_read : std_logic := '0';
signal enable_write : std_logic := '0';
signal salida : std_logic_vector(7 downto 0) := ( others => '0');

constant clk_period : time := 100 ns;
```

Figura 80. Inclusión componente B, y señales auxiliares tb_Bloque_B.

En este caso solo tendremos dos procesos, el proceso de reloj, que es exactamente el mismo que el ejemplo anterior del Bloque A. El proceso de los estímulos consistirá en una apertura del device file write_8 , se introducen una serie de entradas, y posteriormente se cierra el device file write_8. Tras un periodo de tiempo, se abre y se cierra el device file read_8.

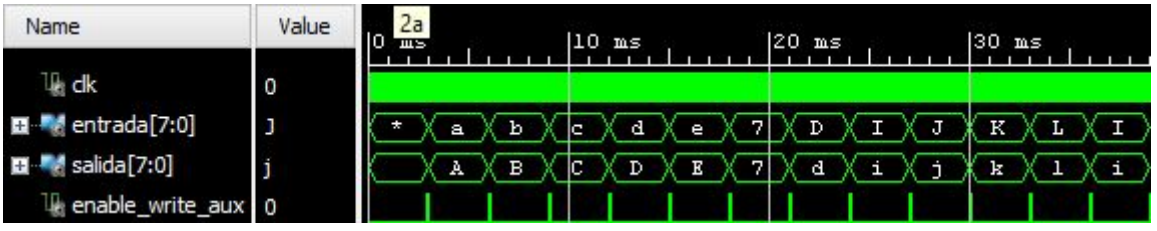

Figura 81. Comprobación Test_Bench Bloque B.

Como se comprueba en el test_bench no hay problema, y se cumplen las funcionalidades, el caso del primer momento de la salida es debido a que hasta que *"enable_write_aux"* no se active, no varía.

6.2.4 Comprobación en la FPGA

Para asegurarnos que la funcionalidad es real, comprobaremos que en la FPGA funciona correctamente.

Seguimos los pasos indicados en la comprobación en la FPGA del caso del Bloque A, hasta el paso donde se comenta la FIFO de 8 bits, pero en esta ocasión no se comenta. Ya que usaremos dos.

En este caso, para poder conectar el bloque correctamente al xillybus, hace falta un bloque de interconexión que llamaremos top_Bloque_B, en el que incluiremos el Bloque B, y dos memorias FIFO 8x2048.

```
entity top_Bloque_B is
  Port (
        clk : in std_logic;
            entrada : in std_logic_vector(7 downto 0);
            user_w_write_8_wren : in std_logic;
            user_r_read_8_rden : in std_logic;
            user_r_read_8_open : in std_logic;
            user_w_write_8_open : in std_logic;
            user_w_write_8_full : out std_logic;
            user_r_read_8_empty : out std_logic;
            user_r_read_8_data : out std_logic_vector(7 downto 0)
            );
end top_Bloque_B;
```

Figura 82. Entidad top_Bloque_B.

Este bloque contiene todas las señales de ambas interfaces usadas en el Bloque (read_8 y write_8), señales que conectaremos a las dos memorias y al bloque anterior para que se procese la entrada de manera correcta.

En primer lugar tendremos una memoria donde almacenaremos los datos que se se introducen en entrada, y se activan su lectura con *"user_w_write_8_wren"*, donde si se llena se activará la señal *"user_w_write_8_full"*.

En segundo lugar, tenemos otra memoria donde la lectura se activa con *"user_r_read_8_rden"*, la salida se conectará a *"user_r_read_8_data"* además se incluirá la señal empty de la interfaz read_8, *"user_r_read_8_empty"*, a esta memoria.

Esto podemos apreciarlo en las siguientes figuras, en las que se observa la definición de las señales auxiliares necesarias para el bloque, y los componentes para el mismo.

```
component Bloque_B
   Port ( clk : in STD_LOGIC;
          entrada : in std_logic_vector(7 downto 0);
          empty : in std_logic;
          full : in std_logic;
          user_r_read_8_open : in std_logic;
          user_w_write_8_open : in std_logic;
          enable_read : out std_logic;
          enable_write : out std_logic;
          salida : out std_logic_vector(7 downto 0) );
end component;
signal full_aux : std_logic := '0';
signal empty_aux : std_logic := '0';
signal enable_read_aux : std_logic := '0';
signal enable_write_aux : std_logic := '0';
signal salida_aux : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal entrada_aux : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
```

```
component fifo_8x2048
    port ( clk: in std_logic;
      srst: in std_logic;
      din: in std_logic_VECTOR(7 downto 0);
      wr_en: in std_logic;
      rd_en: in std_logic;
      dout: out std_logic_VECTOR(7 downto 0);
      full: out std_logic;
      empty: out std_logic);
  end component;
```

Figura 83. Componentes y señales auxiliares top_Bloque_B.

```
uut_bloque_b : Bloque_B
port map ( clk => clk,
           entrada => entrada_aux,
           empty => empty_aux,
           full => full_aux,
           user_r_read_8_open => user_r_read_8_open,
           user_w_write_8_open => user_w_write_8_open,
           enable_read => enable_read_aux,
           enable_write => enable_write_aux,
           salida => salida_aux    );
fifo_8_w : fifo_8x2048
    port map(
        clk        => clk,
        srst       => '0',
        din        => entrada,
        wr_en      => user_w_write_8_wren,
        rd_en      => enable_read_aux,
        dout       => entrada_aux,
        full       => user_w_write_8_full,
        empty      => empty_aux
);
fifo_8_r : fifo_8x2048
    port map(
        clk        => clk,
        srst       => '0',
        din        => salida_aux,
        wr_en      => enable_write_aux,
        rd_en      => user_r_read_8_rden,
        dout       => user_r_read_8_data,
        full       => full_aux,
        empty      => user_r_read_8_empty
    );
```

Figura 84. Interconexión top_Bloque_B.

La primera memoria, se conecta al device file, write_8, para adquirir los datos que provienen del host. Esto se ve como la señal entrada, que se conecta del top, a en xillidemo a *"user_w_write_8_data"*, el enable de escritura con el *"user_w_write_8_wren"* . El enable de lectura proviene del bloque de procesamiento diseñado anteriormente, y la señal de salida, *"dout"* (*"entrada_aux"*) y *"empty"*, de la memoria se conecta también con este para su procesamiento. La señal *"full"* se dirige hacia Xillybus.

La segunda memoria se hace hacia el exterior, recibe la entrada *"din"* (*"salida_aux"*) del bloque de procesamiento, al igual que el enable de escritura, *"wr_en"* (*"enable_write_aux"*), y le envía la señal *"full"* (*"full_aux"*) , al mismo. Y recibe el enable de lectura *"rd_en"* (*"user_r_read_8_rden"*), y envia la señal *"empty"* (*"user_r_read_8_empty"*) y la señal de salida de la memoria, *"dout"* (*"user_r_read_8_data"*), que se envía hacia el device file, read_8 que se observará en el host.

Tras esto añadimos en xillydemo.vhdl, el componente top_Bloque_B. Ahora podemos generar el .bit, vamos

al *"Flow Navigator"*, *"Program and Debug"* y pulsamos *"Generate BitStream"*.

Cuando tengamos el .bit generado, lo cargamos en la tarjeta SD y la introducimos en la ZedBoard. Iniciamos la máquina virtual y ejecutamos los comandos indicados en el apartado anterior:

"sudo minicom /dev/ttyACM0"

"cd xillybus/demoapps/"

"make"

"./streamwrite /dev/xillybus_write_8"

Enviamos una cadena de caracteres ejemplo como se ve en la figura 85.

"./streamread /dev/xillybus_read_8"

Comprobamos que efectivamente se cumple la funcionalidad que se quería implementar, los caracteres en minúscula pasan a mayúscula y viceversa y el resto sigue igual.

```
root@localhost:~/xillybus/demoapps# ./streamwrite /dev/xillybus_write_8
hola esto es una prueba MAYUSCULAS 1994 minusculas^C
root@localhost:~/xillybus/demoapps# ./streamread /dev/xillybus_read_8
HOLA ESTO ES UNA PRUEBA mayusculas 1994 MINUSCULAS^C
root@localhost:~/xillybus/demoapps#
```

Figura 85. Resultados top_Bloque_B, en FPGA.

6.3 Primera modificación

Para profundizar en el tema anterior, pasaremos de una funcionalidad de cambiar sólo los caracteres ASCII a realizar una serie de operaciones dependiendo de los Switches que estén activados.

En primer lugar, como en la tercera modificación del Bloque A usaremos los Switches para seleccionar las 3 operaciones distintas:

- Switch 1 → suma
- Switch 2 → resta
- Switch 3 → multiplicación
- En otro caso, se realizará la suma.

Añadimos un módulo nuevo al Tutorial Bloque B, *"Flow Navigator"*, *"Project Manager"*, *"Add Sources"* y creamos un nuevo archivo de diseño vhdl, como en la figura 45, seleccionando *"Add or create design sources"*, *"Create file"*, con el nombre Bloque_B_primera_mod, con las mismas señales pero añadiendo 3 señales Switch_one, two , three, tipo std_logic.

```
entity Bloque_B_primera_mod is
    Port ( clk : in std_logic;
           Switch_one : in std_logic;
           Switch_two : in std_logic;
           Switch_three : in std_logic;
           entrada : in std_logic_vector(7 DOWNTO 0);
           empty : in std_logic;
           full : in std_logic;
           user_r_read_8_open : in std_logic;
           user_w_write_8_open : in std_logic;
           enable_read : out std_logic;
           enable_write : out std_logic;
           salida : out std_logic_vector(7 DOWNTO 0) );
end Bloque_B_primera_mod;
```

```
entity top_Bloque_B_primera_mod is
  Port ( clk : in std_logic;
           Switch_one : in std_logic;
           Switch_two : in std_logic;
           Switch_three : in std_logic;
           entrada : in std_logic_vector(7 DOWNTO 0);
           user_w_write_8_wren : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_open : in std_logic;
           user_w_write_8_open : in std_logic;
           user_w_write_8_full : out std_logic;
           user_r_read_8_empty : out std_logic;
           user_r_read_8_data : out std_logic_vector(7 DOWNTO 0) );
end top_Bloque_B_primera_mod;
```

Figura 86. Modificaciones entradas, bloque b.

En el top_Bloque_B_primera_mod, aparte de los Switches, no se realiza ninguna otra modificación. En el xillydemo, cambiaremos las líneas del top_Bloque_B, para actualizarlo al top_Bloque_B_primera_mod, en la asignación de señales en los Switches se conectará directamente a los PS_GPIO correspondientes, que se ven la figura 68.

6.3.1 Señales auxiliares

En las siguientes figuras se muestran las distintas señales auxiliares necesarias para el desarrollo de este bloque y una breve descripción de su uso (figuras 87 , 88 , 89).

```
signal start_write : std_logic := '0';
signal start_write_sig : std_logic := '0';

signal start_write_flag : std_logic := '0';
signal start_write_flag_sig : std_logic := '0';

signal letra_result : unsigned (4 downto 0) := ( others => '0' );
signal letra_result_sig : unsigned ( 4 downto 0 ) := ( others => '0' );

signal letra : unsigned ( 2 downto 0 ) := ( others => '0');
signal letra_sig : unsigned ( 2 downto 0 ) := ( others => '0');
```

Figura 87. Señales auxiliares añadidas para la escritura en Bloque_B_primera_mod.

Las señales mostradas en la figura 87, sirven para ayudar en la escritura de los resultados en la memoria correspondiente.

Las señales *"start_write"* y *"start_write_flag"* se usan, como se verá posteriormente, para indicar cuándo se empiezan a transmitir datos UpStream.

En cambio "*letra_result"* y *"letra"* las usaremos en el proceso combinacional para, dependiendo de haber fallo o no, enviar un resultado u otro.

- *"letra_result"* se encarga de marcar los casos donde no hay error, por eso es de 5 bits, ya que el mensaje es: "El resultado es xx" 18 caracteres, superior a los 4 bits que nos daría 16 posibilidades.
- *"letra"* se usa para el caso de error, como simplemente se transmitirá "Error" nos vale con sus 3 bits.

```
signal decenas : std_logic := '0';
signal decenas_sig : std_logic := '0';

signal negativo : std_logic := '0';
signal negativo_sig : std_logic := '0';

signal fallo : std_logic := '0';
signal fallo_sig : std_logic := '0';

signal secure : std_logic := '0';
signal secure_sig : std_logic := '0';
```

Figura 88. Señales auxiliares de signo, decenas, y fallo, Bloque B primera mod.

La señal *"decenas"* (figura 88) indica si en la operación se ha superado, en suma o multiplicación, la cifra de "10", activa en la escritura un slot más y hace que se escriba el sumando correspondiente.

"negativo" (figura 88) con esta señal a nivel bajo la escritura no cambia. A nivel alto implica que en la operación de resta el primer sumando es menor que el segundo, añadiendo un signo "-" a la escritura y habilitando el slot.

Con la señal *"fallo"* (figura 88) se indica cuando se ha inclumpido la condición de formato, principalmente cuando hemos introducido un carácter que no corresponde con número ni espacio.

La última señal "secure" (figura 88) sirve para saber si se ha superado el número de entradas que se esperan.

```
signal cont_sum : unsigned ( 1 downto 0 ) := ( others => '0' );
signal cont_sum_sig : unsigned ( 1 downto 0 ) := ( others => '0' );

signal cont_err : unsigned ( 14 downto 0 ) := ( others => '0' );
signal cont_err_sig : unsigned ( 14 downto 0 ) := ( others => '0' );

signal resultado : unsigned ( 15 downto 0 ) := "0000000000110000";
signal resultado_sig : unsigned ( 15 downto 0 ) := "0000000000110000";

signal resultado_decenas : unsigned ( 7 downto 0 ) := "00110000";
signal resultado_decenas_sig : unsigned ( 7 downto 0 ) := "00110000";

signal sum_sig : unsigned ( 7 downto 0 ) := "00110000";

signal sum_alternative : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal sum_alternative_sig : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

signal sum_alternative_hold : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal sum_alternative_hold_sig : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

signal sumador : unsigned ( 15 downto 0 ) := ( others => '0' );
signal sumador_hold : unsigned ( 7 downto 0 ) := ( others => '0' );

signal sumando : unsigned ( 7 downto 0 ) := ( others => '0' );
signal sumando_sig : unsigned ( 7 downto 0 ) := ( others => '0' );

signal sumando_dec : unsigned ( 7 downto 0 ) := ( others => '0' );
signal sumando_dec_sig : unsigned ( 7 downto 0 ) := ( others => '0' );
```

Figura 89. Señales auxiliares para operaciones y contadores auxiliares, Bloque_B_primera_mod.

En la figura 89, tenemos primero dos contadores:

- *"cont_sum"* → que sirve para ver qué entrada estamos viendo. Son 2 bits y se usan las 4 opciones, como se verá al analizar los procesos.
- *"cont_err"* → usado para dar una variación de tiempo entre la transmisión de un carácter y otro en el caso de que ocurra error.

Después encontramos dos señales de resultado:

- *"resultado"* → esta señal envía el resultado de las operaciones a la transmisión. Tiene 16 bits debido a la operación de multiplicación y señal transmite la unidad de la operación.
- *"resultado_decenas"* → esta señal representa el carácter de las decenas de la operación realizada anteriormente.

Tras esto encontramos las señales *"sum_alternative"* y *"sum_alternative_hold"* que nos servirán para que podamos leer las entradas correctamente, y no tengamos el problema de que la última entrada se mantenga.

En la señal *"sumador"* se realizan las distintas operaciones dependiendo de las condiciones en el proceso combinacional. Su homóloga *"sumador_hold"* sirve para, dependiendo del caso, mantener la señal en el estado anterior.

Con *"sumando"* añadimos el siguiente operando, en función de la condición será entrada o "0".

"sumando_dec" es donde variamos los valores dependiendo de la decena que corresponda al resultado de la operación para enviarlo al resultado.

6.3.2 Proceso Secuencial

Para analizar el proceso secuencial de Bloque_B_primera_mod, lo haremos por partes. Iremos desglosando los distintos elementos del proceso según las figuras que se irán mostrando.

```
if ( sum_alternative /= entrada )
    and user_w_write_8_open = '1' then
    sumando <= unsigned(entrada);
    sumando_dec <= sumando_dec_sig;
    decenas <= decenas_sig;
    negativo <= negativo_sig;
elsif cont_sum = "00" then
    sumando <= "00110000";
    sumando_dec <= "00110000";
    decenas <= decenas_sig;
    negativo <= negativo_sig;
elsif user_r_read_8_open = '1' then
    sumando <= "00110000";
    sumando_dec <= "00110000";
    decenas <= '0';
    negativo <= '0';
else
    sumando <= "00110000";
    sumando_dec <= sumando_dec_sig;
    decenas <= decenas_sig;
    negativo <= negativo_sig;
end if;
```

Figura 90. Primera parte p_seq Bloque_B_primera_mod.

Este if/else sirve para asignar a las señales los valores apropiados, pero teniendo una serie de condiciones.

- La primera condición nos sirve para actualizar los valores cuando varía la entrada del valor anterior.
- La condición cont_sum = "00" hace que en caso del valor remanente se reinicien los valores para evitar que se arrastre en operaciones posteriores.
- user_r_read_8_open = ‘’ la usamos para reiniciar las variables para la siguiente operación
- en otro caso mantendremos todas las variables, y sumador la pondremos a cero en ASCII para que el resultado no varíe.

```
if ( user_r_read_8_open = '0' and user_w_write_8_open = '1' and empty = '0'
        and full = '0') then
    enable_write <= '0';
    enable_write_hold <= '0';
    enable_read <= enable_read_aux;
    enable_read_hold <= enable_read_aux;
elsif (start_write = '1') then
    enable_write <= enable_write_aux;
    enable_write_hold <= enable_write_aux;
    enable_read <= '0';
    enable_read_hold <= '0';
else
    enable_write <= '0';
    enable_write_hold <= '0';
    enable_read <= '0';
    enable_read_hold <= '0';
end if;
```

Figura 91. Asignación enables write/read p_seq Bloque_B_primera_mod.

Si *"user_w_write_8_open"* esta activo y se cumplen el resto de condiciones, se activan las señales de enable de lectura, y se ponen a '0' las de escritura. Si esto no ocurre, revisamos si "start_write" = '1' en ese caso ocurre lo contrario. Si no se cumple ninguna de las dos condiciones, todos los enables se ponen a '0'.

```
if user_w_write_8_open = '1' then
    sum_alternative <= sum_alternative_sig;
    sum_alternative_hold <= sum_alternative_sig;
elsif user_r_read_8_open = '1' then
    sum_alternative <= entrada;
    sum_alternative_hold <= entrada;
else
    sum_alternative <= sum_alternative_hold_sig;
    sum_alternative_hold <= sum_alternative_hold_sig;
end if;
```

Figura 92. Asignación sum_alternative.

Este if/else se usa para que solo se asigne un valor a las señales *"sum_alternative"* y *"sum_alternative_hold"* en caso de que el device file write_8 esté abierta. O en caso de que se abra el device file read_8 , le asignamos *"entrada"* directamente. Hasta el case siguiente, las siguientes líneas consisten en asignaciones directas de señales con sus homónimas con "señal"_sig.

```
case sumador(7 downto 0) is
    when "00000000" =>
        resultado <= "0000000000110000";
    when "00000001" =>
        resultado <= "0000000000110001";
    when "00000010" =>
        resultado <= "0000000000110010";
    when "00000011" =>
        resultado <= "0000000000110011";
    when "00000100" =>
        resultado <= "0000000000110100";
    when "00000101" =>
        resultado <= "0000000000110101";
    when "00000110" =>
        resultado <= "0000000000110110";
    when "00000111" =>
        resultado <= "0000000000110111";
    when "00001000" =>
        resultado <= "0000000000111000";
    when "00001001" =>
        resultado <= "0000000000111001";
    when others =>
        resultado <= "0000000000110000";
end case;
```

Figura 93. Asignación resultado p_seq Bloque_B_primera_mod.

En esta asignación cogemos los 8 bits menos significativos y, dependiendo del valor de la variable sumador, le asignamos a la variable resultado en los 8 bits menos significativos el carácter ASCII correspondiente al valor.

6.3.3 Proceso combinacional

Para analizar el proceso combinacional seguiremos el mismo proceso que en el secuencial. En este hay 3 partes diferenciadas que se añaden al Bloque_B anterior:

- Incremento *"cont_sum"*
- Modificación de valores de operadores.
- Lectura/Escritura

```
if entrada /= sum_alternative and user_w_write_8_open = '1' then
    if cont_sum = "11" then
        cont_sum_sig <= cont_sum;
        secure_sig <= '1';
    else
        cont_sum_sig <= cont_sum +1;
        secure_sig <= '0';
    end if;
elsif user_r_read_8_open = '1' then
    secure_sig <= '0';
    cont_sum_sig <= "00";
else
    cont_sum_sig <= cont_sum;
    secure_sig <= '0';
end if;
```

Figura 94. Incremento cont_sum Bloque_B_primera_mod.

Para aumentar el contador, hay que cambiar el caracter de *"entrada"* y estar el device file write_8 abierto, y no haberse llegado al 3º caracter. En caso de haberse llegado al 3º carácter se activará la señal *"secure"*. Si no se cumple la primera condición del if/else inicial, se comprobará si el device file read_8 está abierto, en ese caso, se reinicia la señal *"secure"*, y el contador, en caso de que no se cumpla ninguna de las dos condiciones se pone a nivel bajo la señal *"secure"*, y se mantiene el contador *"cont_sum"* .

Además, a la señal *"sum_alternative_sig"* se le asigna la señal entrada directamente, y a la señal *"sum_alternative_hold_sig"* la señal *"sum_alternative"*.

Después tenemos la parte de modificación de valores, que puede ser:

- Suma (figura 95)
- Resta (figura 96)
- Multiplicación (figura 97, figura 98)

```
if ( Switch_one = '1' and Switch_two = '0' and Switch_three = '0' ) then
    if sumando > "00110000" and sumando < "00111010" and cont_sum /= "00" and secure = '0'  then
        sumador(7 downto 0) <= sumando + sumador_hold - "00110000" ;
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= sumando_dec;
        decenas_sig <= decenas;
    elsif sumador_hold >= "01011010" then
        sumador(7 downto 0) <= sumador_hold - "01011010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 9;
        decenas_sig <= '1';
```

Figura 95. Algoritmo operación suma, p_comb Bloque_B_primera_mod, primera parte.

```
elsif sumador_hold >= "00001010" then
    sumador(7 downto 0) <= sumador_hold - "00001010";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 1;
    decenas_sig <= '1';
elsif user_r_read_8_open = '1' then
    sumador <= ( others => '0' );
    sumador(15 downto 8) <= ( others => '0' );
    decenas_sig <= '0';
    sumando_dec_sig <= "00110000";
else
    sumador(7 downto 0) <= sumador_hold;
    sumador(15 downto 8) <= sumador_hold;
    decenas_sig <= decenas;
    sumando_dec_sig <= sumando_dec;
end if;
```

Figura 96. Algoritmo operación suma, p_comb Bloque_B_primera_mod, segunda parte.

El algoritmo que vemos en la figura 95, 96, (entramos si se está activo el Switch_one o hay una combinación de Switches no prevista) funciona de la siguiente manera:

- Comprobamos si la entrada es un número distinto a 0 caracter ASCII, y no es el carácter remanente de la operación anterior o el carácter nulo inicial, y "secure" distinto de '0'. En ese caso se añade a *"sumador"* el número que se pasa por entrada, manteniendo las señales de decenas, ya que en caso de suma sí se pueden añadir más de dos entradas.

- Si no comparamos, si la suma arrastrada es superior a, dependiendo de unas decenas depende del, elsif, le restamos las decenas, le sumamos a *"sumando_dec_sig"* el número 1 en caracter ASCII correspondiente y activamos la señal decenas.

- Otra condición que debemos tener en cuenta es si *"user_r_read_8_open"* esta a nivel activo se reinician todas las señales.

- En otro caso se mantienen las señales en el estado anterior.

Después mantenemos la señal *"negativo_sig"* a '0', ya que en la suma no puede darse y mantenemos a '0' el resto de la señal *"sumador"*.

```
elsif ( Switch_one = '0' and Switch_two = '1' and Switch_three = '0' ) then
    if sumando > "00110000" and sumando < "00111010" and cont_sum = "01" and secure = '0'  then
        sumador(7 downto 0) <= sumador_hold + sumando - "00110000" ;
        negativo_sig <= '0';
    elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11"
        and sumador_hold > ( unsigned(entrada) - "00110000" ) and secure = '0'  then
        sumador(7 downto 0) <= sumador_hold - sumando - "00110000";
        negativo_sig <= '0';
    elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11"
        and sumador_hold < ( unsigned(entrada) - "00110000" ) and secure = '0'  then
        sumador(7 downto 0) <=  sumando - sumador_hold - "00110000" ;
        negativo_sig <= '1';
    elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11"
        and sumador_hold = ( unsigned(entrada) - "00110000" ) and secure = '0'  then
        sumador(7 downto 0) <=  "00000000" ;
        negativo_sig <= '0';
    elsif sumador_hold >= "00001010" then
        sumador(7 downto 0) <= sumador_hold - "00001010";
        negativo_sig <= negativo;
    elsif user_r_read_8_open = '1' then
        sumador(7 downto 0) <= ( others => '0' );
        negativo_sig <= '0';
    else
        sumador(7 downto 0) <= sumador_hold;
        negativo_sig <= negativo;
    end if;
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <= sumando_dec;
    decenas_sig <= '0';
```

Figura 97 Algoritmo resta p_comb, Bloque_B_primera_mod.

El algoritmo que vemos en la figura 97 tiene el siguiente funcionamiento (en esta ocasión entra con Switch_two = '1'):

- Comprobamos si la entrada es un número distinto a 0, caracter ASCII , "secure" igual a '0', y es el primer operando, en cuyo caso se añade a *"sumador"* el número que se pasa por entrada, poniendo *"negativo_sig"* a '0' ya que no es un número negativo.

- En otro caso comparamos si el siguiente número es el tercer carácter que introducimos, "secure" igual a '0' y el primer número es mayor que el segundo. Aquí *"negativo_sig"* sigue a '0' y se produce la resta sin más.

- Si el caso es que el segundo número introducido es mayor, se activa la señal *"negativo_sig"* y se resta el al segundo carácter el primero.

- Como en el primer algoritmo tiene la condición de que cuando *"user_r_read_8_open"* pasa a nivel alto, se reinician las variables.

- En otro caso se mantienen las señales en el estado anterior.

Después mantenemos la señal *"decenas_sig"* a '0', ya que en la operación de resta no se va a llegar a las decenas. *"sumando_dec_sig"* se le asigna el caracter 0 en ASCII y se pone a nivel 0 los 8 bits más significativos de la señal *"sumador"*.

```
elsif ( Switch_one = '0' and Switch_two = '0' and Switch_three = '1' ) then
    if sumando > "00110000" and sumando < "00111010" and cont_sum = "01" and secure = '0'  then
        sumador(7 downto 0) <= sumando - "00110000";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= sumando_dec;
        decenas_sig <= decenas;
    elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11" and secure = '0'  then
        sumador <= (sumando - "00110000")* sumador_hold;
        sumando_dec_sig <= sumando_dec;
        decenas_sig <= decenas;
    elsif sumador_hold >= "01011010" then
        sumador(7 downto 0) <= sumador_hold - "01011010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00111001";
        decenas_sig <= '1';
    elsif sumador_hold >= "01010000" then
        sumador(7 downto 0) <= sumador_hold - "01010000";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00111000";
        decenas_sig <= '1';
```

Figura 98. Algoritmo multiplicación p_comb, Bloque_B_primera_mod, primera parte.

```
    elsif sumador_hold >= "00001010" then
        sumador(7 downto 0) <= sumador_hold - "00001010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00110001";
        decenas_sig <= '1';
    elsif user_r_read_8_open = '1' then
        sumador <= ( others => '0' );
        sumador(15 downto 8) <= ( others => '0' );
        decenas_sig <= '0';
        sumando_dec_sig <= "00110000";
    else
        sumador(7 downto 0) <= sumador_hold;
        sumador(15 downto 8) <= sumador_hold;
        decenas_sig <= decenas;
        sumando_dec_sig <= sumando_dec;
    end if;
    negativo_sig <= '0';
```

Figura 99. Algoritmo multiplicación p_comb, Bloque_B_primera_mod, primera dos.

El algoritmo de multiplicación se muestra en las figuras 97 y 98. El funcionamiento consiste en:

- Como en el ejemplo de la resta comprobamos que la entrada es un número y es el primer carácter que introducimos. Además "secure" igual a '0', para asegurarnos que no sobrepasamos los 3 caracteres. En este caso pasamos a *"sumador"* el siguiente carácter en sus 8 bits menos significativos y a 0 los más significativos. Y le pasamos a *"sumando_dec_sig"* 0 en ASCII y *"decenas_sig"* a '0'.

- Si es el 3º carácter introducido (y es un número), se le quita a sumando el carácter 0 de ASCII, y se multiplica por la señal *"sumador_hold"*. También le pasamos a *"sumando_dec_sig"* 0 en ASCII y

"decenas_sig" a '0'.

- Después, dependiendo del resultado de la operación, se restará la decena al resultado y se guardará en los 8 bits menos significativos y se pondrán a 0 los más significativos. Se pondrá el número correspondiente en *"sumando_de_sig"* y se activa *"decenas_sig"*.
- Como en los casos anteriores también se reinician las señales al activar *"user_r_read_8_open"*.
- En caso de no darse ninguna condición anterior, se conservan las variables y la señal *"sumador"* se mantiene a *"sumador"* a 0 para que resultado no varíe.

La señal "negativo_sig" se mantendrá a '0' mientras se mantenga en ese modo.

Ahora analizaremos el bloque que se encarga de comprobar si ha habido un fallo, y transmitir el resultado a la memoria del device file write_8.

```
if user_r_read_8_open = '1' then
  fallo_sig <= '0';
elsif secure = '1' then
    fallo_sig <= '1';
elsif user_w_write_8_open = '1' and cont_sum = "00" then
  fallo_sig <= '0';
elsif ( cont_sum = "11" or cont_sum = "01" or cont_sum = "00" )
    and sum_alternative = "00100000" then
        fallo_sig <= '1';
 elsif ( cont_sum = "10" ) and ( sum_alternative > "00110000"
    and sum_alternative < "00111010" ) then
        fallo_sig <= '1';
elsif ( cont_sum = "00" or cont_sum = "01" or cont_sum = "10" or cont_sum = "11" )
    and ( ( sum_alternative < "00110000" or sum_alternative > "00111010" )
    and sum_alternative /= "000000" and sum_alternative /= "00100000" )  then
    fallo_sig <= '1';
elsif ( cont_sum = "10" or cont_sum = "01" ) and user_w_write_8_open = '0' then
    fallo_sig <= '1';
elsif ( cont_sum = "01" or cont_sum = "10" or cont_sum = "11" )
    and ( ( sum_alternative > "00110000" and sum_alternative < "00111010" )
    or sum_alternative = "000000" or sum_alternative = "00100000" )  then
    fallo_sig <= fallo;
elsif user_w_write_8_open = '0' then
    fallo_sig <= fallo;
else
    fallo_sig <= '1';
end if;
```

Figura 99. Comprobación fallo Bloque_B_primera_mod

En la figura 99 vemos las condiciones para activar la señal "fallo", vamos a analizar las distintas condiciones:

- La primera condición implica que si se abre el device file read_8, se baja la señal fallo, se reinicia.
- Con la señal "secure", se activa la señal "fallo".
- Si el caracter espacio, es introducido en el slot, 1, 3, o 0, en el formato "x_x" se activa la señal

"fallo".

- Si en el slot 2 del formato, se introduce un caracter númerico se activa la señal error.
- Si en algun slot se introduce algun caracter que no sea, caracter nulo, númerico, o espacio, se activa la señal "fallo".
- Si estando con cont_sum = "10" o "01" user_w_write_8_open, baja a 0, se activa la señal fallo, indica que se ha introducido menos datos de los necesarios.
- Si cont_sum es "01", "10" o "11" y tenemos un caracter numerico, nulo o espacio, o user_w_write_8_open = '0' , la señal fallo se mantendrá.
- En otro caso se activará.

Cuando terminamos de leer y comenzamos a escribir, con este último bloque.

```
if fallo = '0' and start_write = '1' then
    letra_sig <= ( others => '0' );
    cont_err_sig <= ( others => '0' );
    if cont_w = "0111010100110000" then
        if letra_result = "00000" then
            salida_aux <= "01000101";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "00001" then
            salida_aux <= "01001100";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "00010" then
            salida_aux <= "00100000";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "00011" then
            salida_aux <= "01010010";
            letra_result_sig <= letra_result + 1 ;
            stop_sig <= '0';
```

Figura 101. Transmisión resultado sin error, Bloque_B_primera_mod, parte 1.

```
    elsif letra_result = "10000" and negativo = '1' then
        salida_aux <= "00101101";
        letra_result_sig <= letra_result + 1;
        stop_sig <= '0';
    elsif ( letra_result = "10001" and decenas = '1' )
        or ( letra_result = "10001" and negativo = '1' )
        or ( letra_result = "10000" and decenas = '0' ) then
        salida_aux <= resultado(7 downto 0);
        letra_result_sig <= letra_result + 1;
        stop_sig <= '0';
    elsif ( letra_result = "10010" or letra_result = "10001"  ) then
        salida_aux <= "00101110";
        letra_result_sig <= letra_result + 1;
        stop_sig <= '0';
    else
        salida_aux <= ( others => '0' );
        letra_result_sig <= letra_result;
        stop_sig <= '1';
    end if;
else
```

Figura 102. Transmisión resultado sin error, Bloque_B_primera_mod, parte 2.

Lo que vemos en las figuras 101 y 102 es el inicio y final de este if/else; en medio de lo que habría entre las dos figuras serían los valores de *"letra_result"* entre "00001" y "10000"/"10001", dependiendo si se ha activado *"decenas"* , *"negativo"* o no.

En cada uno de esos valores se iría actualizando el valor de *"salida_aux"* según el caracter ASCII correspondiente, hasta llegar al carácter 16, cada x tiempo marcado por el contador *"cont_w"* (3ms). Al llegar al carácter 17 comprobamos si *"decenas"* o *"negativo"* están activas. En tal caso, en ese slot se escribirá el *"result_decenas"* o el simbolo negativo. Para *"resultado"* ahora tenemos que comprobar un número más si se han activado las señales anteriores y un número menos de lo contrario, tanto igual para enviar el último carácter '.'.

Si no se ha llegado al final de *"cont_w"*, o condición de entrar para variar *"letra_sig"*, mantenemos las señales al mismo nivel y aumentamos en uno el *"cont_err_sig"*. Cuando llegamos al final se activa la señal *"stop_sig"* para dejar de enviar caracteres a la memoria conectada al device file write_8.

```
else
    letra_result_sig <= ( others => '0' );
    if start_write = '1' then
        if cont_err = "111010100110000" then
            if letra = "0000" then
                salida_aux <= "01000101";
                letra_sig <= letra +1 ;
                stop_sig <= '0';
            elsif letra = "0001" then
                salida_aux <= "01010010";
                letra_sig <= letra +1;
                stop_sig <= '0';
            elsif letra = "0010" then
                salida_aux <= "01010010";
                letra_sig <= letra +1;
                stop_sig <= '0';
            elsif letra = "0011" then
                salida_aux <= "01001111";
                letra_sig <= letra +1;
                stop_sig <= '0';
```

Figura 103. Transmisión resultado con error, Bloque_B_primera_mod, parte 1.

```
            elsif letra = "0110" then
                salida_aux <= "00101110";
                letra_sig <= letra + 1;
                stop_sig <= '0';
            elsif letra = "0111" then
                salida_aux <= "00101110";
                letra_sig <= letra + 1;
                stop_sig <= '0';
            else
                salida_aux <= ( others => '0' );
                letra_sig <= ( others => '0' );
                stop_sig <= '1';
            end if;
            cont_err_sig <= cont_err +1;
        elsif cont_err = "111010100110101" then
            letra_sig <= letra ;
            stop_sig <= stop;
            salida_aux <= salida_aux_hold;
            cont_err_sig <= "000000000000101";
        else
            letra_sig <= letra;
            stop_sig <= stop;
            salida_aux <= salida_aux_hold;
            cont_err_sig <= cont_err +1;
        end if;
    else
        cont_err_sig <= cont_err;
        letra_sig <= (others => '0' );
        stop_sig <= '0';
        salida_aux <= salida_aux_hold;
    end if;
end if;
```

Figura 104. Transmisión resultado con error, Bloque_B_primera_mod, parte 2

En el if/else mostrado en las figuras 103 y 104. ocurre lo mismo que en el anterior, pero se actualiza *"letra_aux"* sin tener en cuenta ninguna variable, pero sí el contador *"cont_err_sig"*. Si no se cumple el final de *"cont_err"*, o condición de entrar para variar *"letra_sig"*, mantenemos las señales al mismo nivel y aumentamos en uno el *"cont_err_sig"*. Cuando llegamos al final se activa la señal *"stop_sig"* para dejar de enviar caracteres a la memoria conectada al device file write_8.

Salvo llegando al último carácter que reinicia, para no tener warnings, ponemos un else conservando las señales. Si no se cumple ninguna de las condiciones de las figuras anteriores se mantienen las señales y *"stop_sig"* se pone a bajo nivel, como se ve al final de la figura 104.

6.3.4 Test bench Bloque B primera mod

Copiamos este archivo Bloque_B_primera_mod.vhdl en el proyecto Tutorial_B, y creamos un documento test_bench tb_Bloque_B_primera_mod. Hacemos lo mismo para el código del anterior test bench y cambiamos el component Bloque_B, por Bloque_B_primera_mod. Finalmente le añadimos las 3 señales Switch_one, Switch_two y Switch_three.

Tras conectar las 3 señales al component modificamos el fichero de estímulos, para provocar:

- Primero una suma
- Segundo una resta
- Tercero una multiplicación
- Cuarto un error

```
stim_process: process
    begin
        Switch_one <= '1';
        Switch_two <='0';
        Switch_three <= '0';
        empty <= '0';
        full <= '0';
        entrada <= "00000000";
        user_w_write_8_open <= '0';
        user_r_read_8_open <= '0';
        wait for 500us;
        entrada <= "00000000";
        user_w_write_8_open <= '1';
        wait for 500 us;
        entrada <= "00110100";
        wait for 500 us;
        entrada <= "00100000";
        wait for 500 us;
        entrada <= "00110101";
        wait for 500 us;
```

Figura 105. Proceso de estímulos, suma, tb_Bloque_B_primera_mod.

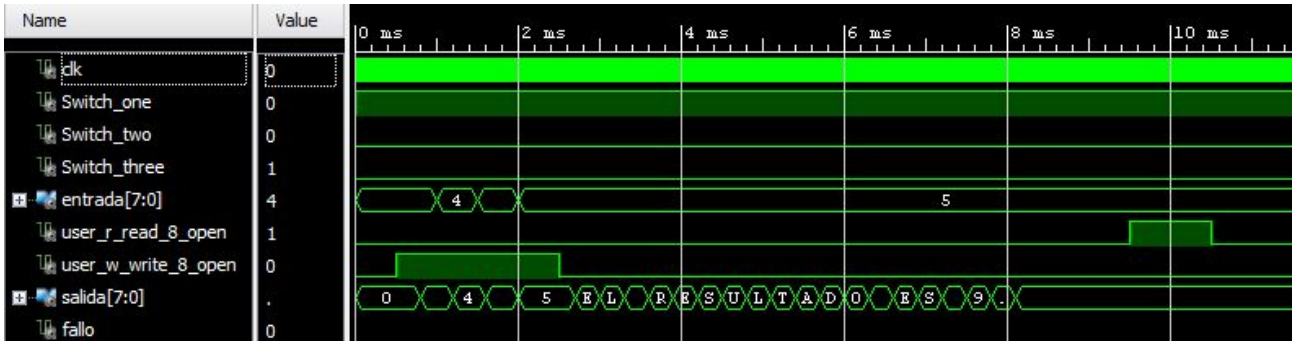

Figura 106. Resultado test_bench suma 4+5, Bloque_B_primera_mod.

```
Switch_one <= '0';
Switch_two <='1';
Switch_three <= '0';
wait for 1000us;
user_w_write_8_open <= '1';
wait for 500ns;
entrada <= "00110011";
wait for 1000 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00110110";
wait for 500 us;
user_w_write_8_open <= '0';
wait for 7000 us;
user_r_read_8_open <= '1';
wait for 6000 us;
user_r_read_8_open <= '0';
wait for 4000us;
```

Figura 107. Proceso estímulos, resta, tb_Bloque_B_primera_mod.

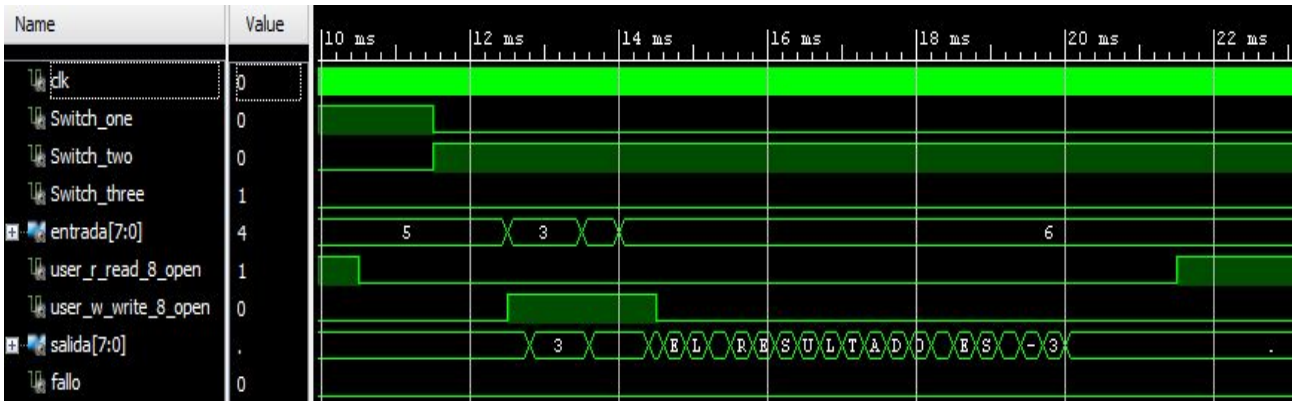

Figura 108. Resultado test_bench 3-6, Bloque_B_primera_mod.

```
Switch_one <= '0';
Switch_two <='0';
Switch_three <= '1';
wait for 1000us;
user_w_write_8_open <= '1';
wait for 500ns;
entrada <= "00111001";
wait for 1000 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00110100";
wait for 500 us;
wait for 300 us;
empty <= '0';
user_w_write_8_open <= '0';

wait for 7000 us;
user_r_read_8_open <= '1';
wait for 6000 us;
user_r_read_8_open <= '0';
wait for 4000us;
```

Figura 109. Proceso estímulos, multiplicación, tb_Bloque_B_primera_mod.

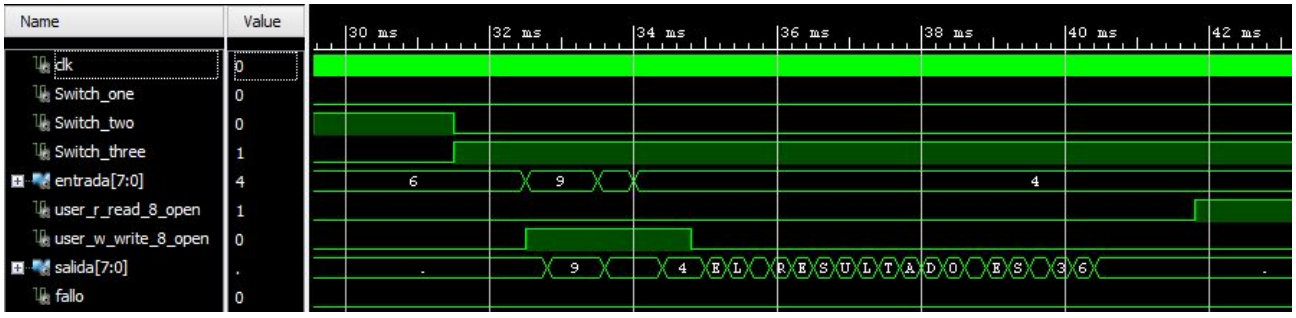

*Figura 110. Resultado test_bench 9*4 Bloque_B_primera_mod.*

```
user_w_write_8_open <= '1';
wait for 500ns;
entrada <= "00110001";
wait for 1000 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00111101";
wait for 500 us;
wait for 300 us;
empty <= '0';
user_w_write_8_open <= '0';

wait for 7000 us;
user_r_read_8_open <= '1';
wait for 6000 us;
user_r_read_8_open <= '0';
wait for 4000us;
```

Figura 111. Proceso estímulos, error , tb_Bloque_B_primera_mod.

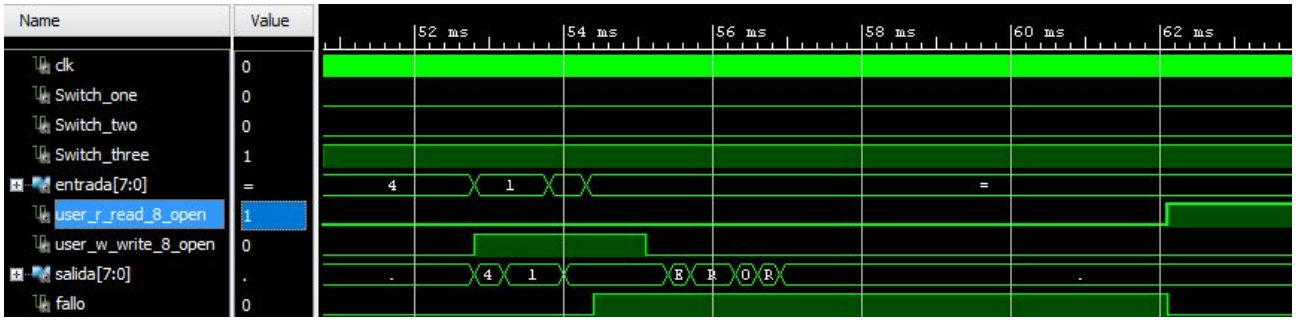

Figura 112. Resultados test_bench, error, Bloque_B_primera_mod.

Vemos que las funcionalidades han sido implementadas con éxito al menos a nivel de simulación de Vivado. Esto significa que en la simulación en Vivado es como esperamos, pero aún debemos comprobarlo en la FPGA. El archivo tb_Bloque_B_primera_mod.vhdl se suministrará con el resto del bloque en archivos anexos, y en la carpeta de proyecto correspondiente.

6.3.5 Comprobación en la FPGA

Para esta modificación cambiaremos el top_Bloque_B.vhdl, a top_Bloque_B_primera_mod.vhdl, pero para este archivo solo se añadirán las entradas Switches y se cambiaran los nombres a los correspondientes components y entity.

En el archivo xillidemo.vhdl modificaremos tambien únicamente el component de top_Bloque_B_primera_mod.vhdl, y se le asignan las señales PS_GPIO(11), PS_GPIO(12), PS_GPIO(13) que corresponden a los Switches.

Una vez tenemos todos los archivos necesarios, procedemos, como en los casos anteriores, a generar el .bit

"Flow Navigator" → "Program and Debug" → "Generate BitStream"

Con el .bit generado , lo cargamos en la SD y conectamos esta a la ZedBoard. Encendemos la máquina

virtual, después la placa y ejecutamos los comandos como en los casos anteriores:

“sudo minicom /dev/ttyACM0”

“cd xillybus/demoapps/”

“make”

“./streamwrite /dev/xillybus_write_8”

```
root@localhost:~/xillybus/demoapps# ./streamwrite /dev/xillybus_write_8
4 5^C
root@localhost:~/xillybus/demoapps# ./streamread /dev/xillybus_read_8
EL RESULTADO ES 9.^C
root@localhost:~/xillybus/demoapps# ./streamwrite /dev/xillybus_write_8
3 6^C
root@localhost:~/xillybus/demoapps# ./streamread /dev/xillybus_read_8
EL RESULTADO ES -3.^C
root@localhost:~/xillybus/demoapps# ./streamwrite /dev/xillybus_write_8
9 4^C
root@localhost:~/xillybus/demoapps# ./streamread /dev/xillybus_read_8
EL RESULTADO ES 36.^C
root@localhost:~/xillybus/demoapps# ./streamwrite /dev/xillybus_write_8
1 =^C
root@localhost:~/xillybus/demoapps# ./streamread /dev/xillybus_read_8
ERROR.^C
root@localhost:~/xillybus/demoapps#
```

113. Comprobación FPGA Bloque_B_primera_mod.

7 CONCLUSIONES Y TRABAJOS FUTUROS

7.1 Conclusiones

En este trabajo se ha abordado el uso de la interfaz Xillybus, para la el envío de datos entre el host y la PL, y se ha visto distintas posibilidades de la funcionalidades:

- DownStream
- UpStream
- O una mezcla de ambos como en el caso del Bloque B

Constituye una primera aproximación al estudio del uso de esta interfaz, como se ha visto muy potente. Ya que pese a que las aplicaciones realizadas en el proyecto son bastantes simples el procesamiento de los datos es de caracter general.

Lo novedoso que proporciona esta interfaz es que proporciona un control desde el procesador a la programación lógica, esto le da una gran potencia. Mientras la programación lógica está ejecutando un determinado algoritmo podemos influir en ella externamente desde el host.

Pese a que las interfaces usadas en el proyecto son de uso general para mayor simplicidad. Hay multitud de interfaces para aplicaciones concretas con características personalizables para el control por parte del usuario como se ve en el capítulo de Xillybus.

7.2 Trabajos Futuros

Ahora podemos optar por dos caminos distintos a la hora de seguir profundizando en el campo:

Profundizando en otros tutoriales para ampliar conocimientos sobre Xillybus:

- Una ampliación sobre las operaciones a realizar en Bloque_B_primera_mod tales como añadir la división, permitir más de unidades en lo que enviamos a la FPGA, o añadir signo a las operaciones. Además de ampliar el rango y no solo tener como máximos los números 99, -8.
- Añadir al Bloque_B_primera_mod una nueva funcionalidad que si se indica mediante un Switch, o de alguna otra manera que estime el diseñador, se permite el uso de operaciones con decimales.
- Un nuevo tutorial en el que monitorizamos de manera continua los sensores de temperatura que tenemos disponibles en la ZedBoard. Y estos datos puedan ser observados en tiempo real por el hos. Pero usando un device file, data adquisition, con un buffer de 1000 ms, con lo que tendríamos datos hasta de 1s antes.
- Un tutorial final, que sería conectando la FPGA a un monitor además del resto de componentes usados en el proyecto, e implementar un juego similar al popular juego “Snake”. El movimiento iría enviado desde le host a la FPGA por las teclas de dirección del teclado en transmisión continua.

Aplicando los conocimientos a aplicaciones reales:

- Una aplicación que recibe datos a partir del XADC por parte de un electrocardiograma, se procesan en la FPGA, y estos son transmitidos hacia el host para su examen por parte del personal pertinente.

ANEXOS

Anexo Bloque_A

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Bloque_A is
    Port ( clk : in std_logic;
           user_r_read_8_open : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_eof : out std_logic;
           user_r_read_8_empty : out std_logic;
           salida: out std_logic_vector (7 downto 0)
           );
end Bloque_A;

architecture Behavioral of Bloque_A is

signal salida_sig : unsigned ( 7 downto 0 ) := ( others => '0' );
signal salida_sig_aux : unsigned ( 7 downto 0 ) := ( others => '0' );

signal salida_hold : unsigned ( 7 downto 0 ) := ( others => '0' );
signal salida_hold_sig : unsigned ( 7 downto 0 ) := ( others => '0' );

begin

    p_seq : process ( clk )
    begin
        if rising_edge(clk) then
            if user_r_read_8_open = '1' then
```

```
            user_r_read_8_eof <= '0';
            if user_r_read_8_rden = '1' then
               salida <= std_logic_vector( salida_sig );
               salida_sig_aux <= salida_sig;
               salida_hold <= salida_sig;
               user_r_read_8_empty <= '1';
            else
               salida <= std_logic_vector( salida_hold_sig );
               salida_sig_aux <= salida_hold_sig;
               salida_hold <= salida_hold_sig;
               user_r_read_8_empty <= '0';
            end if;
         else
            salida <= "00100001";
            salida_sig_aux <= "00100001";
            salida_hold <= "00100001";
            user_r_read_8_empty <= '0';
            user_r_read_8_eof <= '1';
         end if;
      end if;
   end process;

   p_comb : process ( salida_sig_aux )
   begin
      salida_hold_sig <= salida_sig_aux;

      if salida_sig_aux = "01111110" then
         salida_sig <= "00100001";
      else
         salida_sig <= salida_sig_aux + 1;
      end if;
   end process;

end Behavioral;
```

Anexo tb_Bloque_A

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity tb_Bloque_A is
end tb_Bloque_A;

architecture Behavioral of tb_Bloque_A is


component Bloque_A
        port (clk   : IN std_logic;
                        user_r_read_8_open : in std_logic;
                        user_r_read_8_rden : in std_logic;
                        user_r_read_8_empty : out std_logic;
                        user_r_read_8_eof : out std_logic;
                        salida : out std_logic_vector(7 downto 0)
            );
end component;

signal clk : std_logic := '0';

signal user_r_read_8_open : std_logic := '0';
signal user_r_read_8_rden : std_logic := '0';
signal user_r_read_8_empty : std_logic := '0';
signal user_r_read_8_eof : std_logic := '0';

signal salida : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

constant clk_period : time := 10 ns;

begin
```

```
uut_top :  Bloque_A port map (
                clk => clk,
                user_r_read_8_open => user_r_read_8_open,
                user_r_read_8_rden => user_r_read_8_rden,
                user_r_read_8_empty => user_r_read_8_empty,
                user_r_read_8_eof => user_r_read_8_eof,
                salida => salida
          );

        p_clock : process
        begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
        end process;

        p_rden : process
        begin
       user_r_read_8_rden <= '0';
       wait for clk_period;
       user_r_read_8_rden <= '1';
       wait for clk_period;
        end process;

        stim_process: process
        begin
                wait for 70 ns;
                user_r_read_8_open <= '1';
                wait for 300 ns;
                user_r_read_8_open <= '0';
                wait;
        end process;
end Behavioral;
```

Anexo Bloque_A_primera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Bloque_A_primera_mod is
    Port ( clk : in std_logic;
           enable : in std_logic;
           user_r_read_8_open : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_eof : out std_logic;
           user_r_read_8_empty : out std_logic;
           salida: out std_logic_vector (7 downto 0)
           );
end Bloque_A_primera_mod;

architecture Behavioral of Bloque_A_primera_mod is

signal salida_sig : unsigned ( 7 downto 0 ) := "00100001";
signal salida_sig_aux : unsigned ( 7 downto 0 ) := "00100001";

signal salida_hold : unsigned ( 7 downto 0 ) := "00100001";
signal salida_hold_sig : unsigned ( 7 downto 0 ) := "00100001";

signal stop : std_logic := '0';
begin

    p_seq : process ( clk, user_r_read_8_rden, user_r_read_8_open, salida_sig, salida_hold_sig )
    begin
        if rising_edge(clk) then
            if user_r_read_8_open = '1' then
                user_r_read_8_eof <= '0';
                if ( user_r_read_8_rden = '1' and enable = '1')then
                    salida <= std_logic_vector( salida_sig );
```

```
            salida_sig_aux <= salida_sig;
            salida_hold <= salida_sig;
            user_r_read_8_empty <= '1';
         else
            salida <= std_logic_vector( salida_hold_sig );
            salida_sig_aux <= salida_hold_sig;
            salida_hold <= salida_hold_sig;
             user_r_read_8_empty <= '0';
         end if;
      else
         salida <= "00100001";
         salida_sig_aux <= "00100001";
         salida_hold <= "00100001";
         user_r_read_8_eof <= '1';
         user_r_read_8_empty <= '1';
      end if;

   end if;
end process;

p_comb : process ( salida_sig_aux )
begin
   salida_hold_sig <= salida_sig_aux;

   if salida_sig_aux = "01111110" then
      salida_sig <= "00100001";
   else
      salida_sig <= salida_sig_aux + 1;
   end if;
end process;

end Behavioral;
```

Anexo top_Bloque_A_primera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity top_Bloque_A_primera_mod is
    Port ( clk : in std_logic;
           user_r_read_8_open : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_eof : out std_logic;
           user_r_read_8_empty : out std_logic;
           salida : out std_logic_vector (7 downto 0)
           );
end top_Bloque_A_primera_mod;

architecture Behavioral of top_Bloque_A_primera_mod is

    component div_frec
        Port ( clk : in std_logic;
               enable : out std_logic);
    end component;

    component Bloque_A_primera_mod
        PORT ( clk : in std_logic;
               enable : in std_logic;
               user_r_read_8_open : in std_logic;
               user_r_read_8_rden : in std_logic;
               user_r_read_8_eof : out std_logic;
               user_r_read_8_empty : out std_logic;
               salida: out std_logic_vector (7 downto 0)
           );
    END component;
```

```
signal enable_aux : std_logic := '0';

begin

   uut_bloq : Bloque_A_primera_mod
      port map (
            clk => clk,
            enable => enable_aux,
            user_r_read_8_open => user_r_read_8_open,
            user_r_read_8_rden => user_r_read_8_rden,
            user_r_read_8_eof => user_r_read_8_eof,
            user_r_read_8_empty => user_r_read_8_empty,
            salida => salida
            );

   uut_div : div_frec
      port map (
            clk => clk,
            enable => enable_aux
            );

end Behavioral;
```

Anexo tb_Bloque_A_primera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity tb_Bloque_A_primera_mod is
end tb_Bloque_A_primera_mod;

architecture Behavioral of tb_Bloque_A_primera_mod is


component top_Bloque_A_primera_mod
        PORT (clk : in std_logic;
          user_r_read_8_open : in std_logic;
          user_r_read_8_rden : in std_logic;
          user_r_read_8_eof : out std_logic;
          user_r_read_8_empty : out std_logic;
          salida : out std_logic_vector (7 downto 0)
           );
END component;

signal clk : std_logic := '0';

signal user_r_read_8_open : std_logic := '0';
signal user_r_read_8_rden : std_logic := '0';
signal user_r_read_8_empty : std_logic := '0';
signal user_r_read_8_eof : std_logic := '0';

signal salida : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

constant clk_period : time := 10 ns;
```

```
begin

uut_top :  top_Bloque_A_primera_mod port map (
                clk => clk,
                user_r_read_8_open => user_r_read_8_open,
                user_r_read_8_rden => user_r_read_8_rden,
                user_r_read_8_empty => user_r_read_8_empty,
                user_r_read_8_eof => user_r_read_8_eof,
                salida => salida
          );

        p_clock : process
        begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
        end process;

        p_rden : process
        begin
       user_r_read_8_rden <= '0';
       wait for clk_period;
       user_r_read_8_rden <= '1';
       wait for clk_period;
        end process;

        stim_process: process
        begin
                wait for 70 ns;
                user_r_read_8_open <= '1';
                wait for 10000 ms;
                user_r_read_8_open <= '0';
                wait;
        end process;
end Behavioral;
```

Anexo Bloque_A_segunda_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Bloque_A_segunda_mod is
    Port ( clk : in std_logic;
           enable : in std_logic;
           boton_up : in std_logic;
           boton_down : in std_logic;
           boton_right : in std_logic;
           boton_left : in std_logic;
           boton_center : in std_logic;
           user_r_read_8_open : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_eof : out std_logic;
           user_r_read_8_empty : out std_logic;
           salida: out std_logic_vector (7 downto 0)
          );
end Bloque_A_segunda_mod;

architecture Behavioral of Bloque_A_segunda_mod is

signal boton_center_aux : std_logic := '0';
signal boton_up_aux : std_logic := '0';
signal boton_down_aux : std_logic := '0';
signal boton_right_aux : std_logic := '0';
signal boton_left_aux : std_logic := '0';

signal boton_center_aux_sig : std_logic := '0';
signal boton_up_aux_sig : std_logic := '0';
signal boton_down_aux_sig : std_logic := '0';
signal boton_right_aux_sig : std_logic := '0';
```

```
signal boton_left_aux_sig : std_logic := '0';



signal salida_sig : unsigned ( 7 downto 0 ) := "00100001";
signal salida_sig_aux : unsigned ( 7 downto 0 ) := "00100001";

signal salida_hold : unsigned ( 7 downto 0 ) := "00100001";
signal salida_hold_sig : unsigned ( 7 downto 0 ) := "00100001";

signal stop : std_logic := '0';
begin

   p_seq : process ( clk, user_r_read_8_rden, user_r_read_8_open, salida_sig,
                  salida_hold_sig, boton_up_aux_sig, boton_down_aux_sig, boton_right_aux_sig,
                  boton_left_aux_sig, boton_center_aux_sig )
   begin
      if rising_edge(clk) then

         boton_center_aux <= boton_center_aux_sig;
         boton_up_aux <= boton_up_aux_sig;
         boton_down_aux <= boton_down_aux_sig;
         boton_right_aux <= boton_right_aux_sig;
         boton_left_aux <= boton_left_aux_sig;

         if user_r_read_8_open = '1' then
            user_r_read_8_eof <= '0';
            if ( user_r_read_8_rden = '1' and ( enable = '1' or boton_up = '1' or
                  boton_down = '1' or boton_right = '1' or boton_left = '1' or
                        boton_center = '1' ) ) then
               salida <= std_logic_vector( salida_sig );
               salida_sig_aux <= salida_sig;
               salida_hold <= salida_sig;
               user_r_read_8_empty <= '1';
            else
               salida <= std_logic_vector( salida_hold_sig );
               salida_sig_aux <= salida_hold_sig;
```

```
            salida_hold <= salida_hold_sig;
             user_r_read_8_empty <= '0';
          end if;
       else
          salida <= "00100001";
          salida_sig_aux <= "00100001";
          salida_hold <= "00100001";
          user_r_read_8_eof <= '1';
          user_r_read_8_empty <= '1';
       end if;

   end if;
end process;

p_comb : process ( salida_sig_aux, boton_up, boton_down, boton_right, boton_left,
                boton_center, boton_up_aux, boton_down_aux, boton_right_aux,
                boton_left_aux, boton_center_aux )
begin
   salida_hold_sig <= salida_sig_aux;
                if ( boton_up = '0' and boton_down = '0' and boton_right = '0'
          and boton_left = '0' and boton_center = '0' and  boton_up_aux = '0'
                   and boton_down_aux = '0' and boton_right_aux = '0'
                   and boton_left_aux = '0' and boton_center_aux = '0') then

          if salida_sig_aux = "01111110" then
             salida_sig <= "00100001";
          else
             salida_sig <= salida_sig_aux + 1;
          end if;
          boton_center_aux_sig <= boton_center_aux;
          boton_up_aux_sig <= boton_up_aux;
          boton_down_aux_sig <= boton_down_aux;
          boton_right_aux_sig <= boton_right_aux;
          boton_left_aux_sig <= boton_left_aux;
                elsif ( ( boton_up = '1' or boton_up_aux = '1' ) and boton_down = '0' and boton_right = '0'
                    and boton_left = '0' and boton_center = '0') then
```

```
        if ( boton_up = '1' ) then
            salida_sig <= "00110000";
            boton_center_aux_sig <= '0';
            boton_up_aux_sig <= '1';
            boton_down_aux_sig <= '0';
            boton_right_aux_sig <= '0';
            boton_left_aux_sig <= '0';
        elsif salida_sig_aux = "00111001" then
            salida_sig <= "00110000";
            boton_center_aux_sig <= boton_center;
            boton_up_aux_sig <= boton_up_aux;
            boton_down_aux_sig <= boton_down;
            boton_right_aux_sig <= boton_right;
            boton_left_aux_sig <= boton_left;
        else
            salida_sig <= salida_sig_aux + 1;
            boton_center_aux_sig <= boton_center;
            boton_up_aux_sig <= boton_up_aux;
            boton_down_aux_sig <= boton_down;
            boton_right_aux_sig <= boton_right;
            boton_left_aux_sig <= boton_left;
        end if;
                elsif ( boton_down = '1' or boton_down_aux = '1' ) and boton_up = '0' and boton_right = '0'
                    and boton_left = '0' and boton_center = '0' then
    if ( boton_down = '1' ) then
        salida_sig <= "00111010";
        boton_center_aux_sig <= '0';
        boton_up_aux_sig <= '0';
        boton_down_aux_sig <= '1';
        boton_right_aux_sig <= '0';
        boton_left_aux_sig <= '0';
      elsif salida_sig_aux = "01000000" then
        salida_sig <= "00111010";
        boton_center_aux_sig <= boton_center;
        boton_up_aux_sig <= boton_up;
        boton_down_aux_sig <= boton_down_aux;
```

```
            boton_right_aux_sig <= boton_right;
            boton_left_aux_sig <= boton_left;
        else
            salida_sig <= salida_sig_aux + 1;
            boton_center_aux_sig <= boton_center;
            boton_up_aux_sig <= boton_up;
            boton_down_aux_sig <= boton_down_aux;
            boton_right_aux_sig <= boton_right;
            boton_left_aux_sig <= boton_left;
        end if;
    elsif ( boton_right = '1' or boton_right_aux = '1' ) and boton_down = '0' and
                boton_up = '0' and boton_left = '0' and boton_center = '0'then
      if ( boton_right = '1' ) then
          salida_sig <= "01000001";
          boton_center_aux_sig <= '0';
          boton_up_aux_sig <= '0';
          boton_down_aux_sig <= '0';
          boton_right_aux_sig <= '1';
          boton_left_aux_sig <= '0';
      elsif salida_sig_aux = "01011010" then
          salida_sig <= "01000001";
          boton_center_aux_sig <= boton_center;
          boton_up_aux_sig <= boton_up;
          boton_down_aux_sig <= boton_down;
          boton_right_aux_sig <= boton_right_aux;
          boton_left_aux_sig <= boton_left;
      else
          salida_sig <= salida_sig_aux + 1;
          boton_center_aux_sig <= boton_center;
          boton_up_aux_sig <= boton_up;
          boton_down_aux_sig <= boton_down;
          boton_right_aux_sig <=boton_right_aux;
          boton_left_aux_sig <= boton_left;
      end if;
    elsif ( boton_left = '1' or boton_left_aux = '1' ) and boton_down = '0' and
                boton_right = '0' and boton_up = '0' and boton_center = '0' then
```

```
    if ( boton_left = '1' ) then
        salida_sig <= "01100001";
        boton_center_aux_sig <= '0';
        boton_up_aux_sig <= '0';
        boton_down_aux_sig <= '0';
        boton_right_aux_sig <= '0';
        boton_left_aux_sig <= '1';
    elsif salida_sig_aux = "01111010" then
        salida_sig <= "01100001";
        boton_center_aux_sig <= boton_center;
        boton_up_aux_sig <= boton_up;
        boton_down_aux_sig <= boton_down;
        boton_right_aux_sig <= boton_right;
        boton_left_aux_sig <= boton_left_aux;
    else
         salida_sig <= salida_sig_aux + 1;
         boton_center_aux_sig <= boton_center;
         boton_up_aux_sig <= boton_up;
         boton_down_aux_sig <= boton_down;
         boton_right_aux_sig <= boton_right;
         boton_left_aux_sig <= boton_left_aux;
    end if;
elsif ( boton_center = '1' or boton_center_aux = '1' ) and boton_down = '0' and
        boton_right = '0' and boton_left = '0' and boton_up = '0' then
    if ( boton_center = '1' ) then
        salida_sig <= "01011011";
        boton_center_aux_sig <= '1';
        boton_up_aux_sig <= '0';
        boton_down_aux_sig <= '0';
        boton_right_aux_sig <= '0';
        boton_left_aux_sig <= '0';
    elsif salida_sig_aux = "01100000" then
        salida_sig <= "01011011";
        boton_center_aux_sig <= boton_center_aux;
        boton_up_aux_sig <= boton_up;
        boton_down_aux_sig <= boton_down;
```

```
            boton_right_aux_sig <= boton_right;
            boton_left_aux_sig <=  boton_left;
          else
              salida_sig <= salida_sig_aux + 1;
              boton_center_aux_sig <= boton_center_aux;
              boton_up_aux_sig <= boton_up;
              boton_down_aux_sig <= boton_down;
              boton_right_aux_sig <= boton_right;
              boton_left_aux_sig <= boton_left;
          end if;
        else
          salida_sig <= salida_sig_aux;
          boton_center_aux_sig <= boton_center;
          boton_up_aux_sig <= boton_up;
          boton_down_aux_sig <= boton_down;
          boton_right_aux_sig <= boton_right;
          boton_left_aux_sig <= boton_left;
        end if;
    end process;

end Behavioral;
```

Anexo top_Bloque_A_segunda_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity top_Bloque_A_segunda_mod is
    Port ( clk : in std_logic;
           botones : in std_logic_vector ( 4 downto 0 );
           user_r_read_8_open : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_eof : out std_logic;
           user_r_read_8_empty : out std_logic;
           salida : out std_logic_vector (7 downto 0)
           );
end top_Bloque_A_segunda_mod;

architecture Behavioral of top_Bloque_A_segunda_mod is

    component div_frec
        Port ( clk : in std_logic;
               enable : out std_logic);
    end component;

    component Bloque_A_segunda_mod
        PORT ( clk : in std_logic;
               enable : in std_logic;
               boton_up : in std_logic;
               boton_down : in std_logic;
               boton_right : in std_logic;
               boton_left : in std_logic;
               boton_center : in std_logic;
               user_r_read_8_open : in std_logic;
               user_r_read_8_rden : in std_logic;
```

```
            user_r_read_8_eof : out std_logic;
            user_r_read_8_empty : out std_logic;
            salida: out std_logic_vector (7 downto 0)
        );
    END component;

signal enable_aux : std_logic := '0';

begin

    uut_bloq : Bloque_A_segunda_mod
        port map (
            clk => clk,
            enable => enable_aux,
            boton_up => botones(2),
            boton_down => botones(3),
            boton_right => botones(1),
            boton_left => botones(0),
            boton_center => botones(4),
            user_r_read_8_open => user_r_read_8_open,
            user_r_read_8_rden => user_r_read_8_rden,
            user_r_read_8_eof => user_r_read_8_eof,
            user_r_read_8_empty => user_r_read_8_empty,
            salida => salida
            );

    uut_div : div_frec
        port map (
            clk => clk,
            enable => enable_aux
            );

end Behavioral;
```

Anexo tb_Bloque_A_segunda_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity tb_Bloque_A_segunda_mod is
end tb_Bloque_A_segunda_mod;

architecture Behavioral of tb_Bloque_A_segunda_mod is

component top_Bloque_A_segunda_mod
        PORT (clk : in std_logic;
          botones : in std_logic_vector ( 4 downto 0 );
          user_r_read_8_open : in std_logic;
          user_r_read_8_rden : in std_logic;
          user_r_read_8_eof : out std_logic;
          user_r_read_8_empty : out std_logic;
          salida : out std_logic_vector (7 downto 0)
          );
END component;

signal clk : std_logic := '0';

   signal botones : std_logic_vector( 4 downto 0 ) := ( others => '0' );

signal user_r_read_8_open : std_logic := '0';
signal user_r_read_8_rden : std_logic := '0';
signal user_r_read_8_empty : std_logic := '0';
signal user_r_read_8_eof : std_logic := '0';

signal salida : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

constant clk_period : time := 100 us;
```

```
begin

uut_top :  top_Bloque_A_segunda_mod port map (
                clk => clk,
                botones => botones,
                user_r_read_8_open => user_r_read_8_open,
                user_r_read_8_rden => user_r_read_8_rden,
                user_r_read_8_empty => user_r_read_8_empty,
                user_r_read_8_eof => user_r_read_8_eof,
                salida => salida
          );

        p_clock : process
        begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
        end process;

        p_rden : process
        begin
       user_r_read_8_rden <= '0';
       wait for clk_period;
       user_r_read_8_rden <= '1';
       wait for clk_period;
        end process;

        stim_process: process
        begin
                wait for 70 ns;
                user_r_read_8_open <= '1';
                wait for 7000 ms;
                botones <= "00001";
                wait for 500 ms;
```

```
                botones <= "00000";
                wait for 1500ms ;
                botones <= "00100";
                wait for 210 ms;
                botones <= "00000";
                wait for 100000 ms;
                user_r_read_8_open <= '0';
                wait;
        end process;
end Behavioral;
```

Anexo Div_frec

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity div_frec is
    Port ( clk : in std_logic;
           enable : out std_logic);
end div_frec;

architecture Behavioral of div_frec is

signal enable_aux : std_logic := '0';
signal enable_aux_sig : std_logic := '0';

signal cont :  unsigned ( 14 downto 0 ):= ( others => '0' );
signal cont_sig : unsigned ( 14 downto 0 ):= ( others => '0' );

begin

    p_seq : process ( clk )
    begin
        if rising_edge ( clk ) then
            enable <= enable_aux;
            enable_aux_sig <= enable_aux;
            cont <= cont_sig;
        end if;
    end process;

    p_comb : process ( cont, enable_aux_sig)
    begin
        if cont = "111010100110000" then
            enable_aux <= not(enable_aux_sig);
```

```
            cont_sig <= cont + 1;
        elsif cont = "111010100110010" then
            enable_aux <= enable_aux_sig;
            cont_sig <= ( others => '0' );
        else
            enable_aux <= '0';
            cont_sig <= cont + 1;
        end if;
    end process;
end Behavioral;
```

Anexo Bloque_A_tercera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Bloque_A_tercera_mod is
    Port ( clk : in std_logic;
           Switch_one : in std_logic;
           Switch_two : in std_logic;
           Switch_three : in std_logic;
           Switch_four : in std_logic;
          enable : out std_logic);
end div_frec_tercera_mod;

architecture Behavioral of Bloque_A_tercera_mod is

signal enable_aux : std_logic := '0';
signal enable_aux_sig : std_logic := '0';

signal cont :  unsigned ( 18 downto 0 ):= ( others => '0' );
signal cont_sig : unsigned ( 18 downto 0 ):= ( others => '0' );

begin

    p_seq : process ( clk )
    begin
        if rising_edge ( clk ) then
            enable <= enable_aux;
            enable_aux_sig <= enable_aux;
            cont <= cont_sig;
        end if;
    end process;
```

```
p_comb : process ( cont, enable_aux_sig, Switch_one, Switch_two,
                   Switch_three, Switch_four)
begin
   if (  Switch_one = '0' and Switch_two = '0' and  Switch_three = '0' and
         Switch_four = '0' ) then
      if cont = "00001110101001100000" then
         enable_aux <= not(enable_aux_sig);
         cont_sig <= cont + 1;
      elsif cont >= "00001110101001100010" then
         enable_aux <= '0';
         cont_sig <= "00000000000000000010";
      else
         enable_aux <= enable_aux_sig;
         cont_sig <= cont + 1;
      end if;
   elsif ( Switch_one = '1' ) then
      if cont = "00111010100110000" then
         enable_aux <= not(enable_aux_sig);
         cont_sig <= cont + 1;
      elsif cont >= "00111010100110010" then
         enable_aux <= '0';
         cont_sig <= "00000000000000000010";
      else
         enable_aux <= enable_aux_sig;
         cont_sig <= cont + 1;
      end if;
   elsif ( Switch_two = '1' ) then
      if cont = "00010011100010000000" then
         enable_aux <= not(enable_aux_sig);
         cont_sig <= cont + 1;
      elsif cont >= "00010011100010000010" then
         enable_aux <= '0';
         cont_sig <= "00000000000000000010";
      else
         enable_aux <= enable_aux_sig;
         cont_sig <= cont + 1;
```

```
        end if;
    elsif ( Switch_three = '1' ) then
        if cont = "00000010011100010000" then
            enable_aux <= not(enable_aux_sig);
            cont_sig <= cont + 1;
        elsif cont >= "00000010011100010010" then
            enable_aux <= '0';
            cont_sig <= "00000000000000000010";
        else
            enable_aux <= enable_aux_sig;
            cont_sig <= cont + 1;
        end if;
    elsif ( Switch_four = '1' ) then
        if cont = "00000001001110001000" then
            enable_aux <= not(enable_aux_sig);
            cont_sig <= cont + 1;
        elsif cont >= "00000001001110001010" then
            enable_aux <= '0';
            cont_sig <= "00000000000000000010";
        else
            enable_aux <= enable_aux_sig;
            cont_sig <= cont + 1;
        end if;
    else
        if cont = "00011000011010100000" then
            enable_aux <= not(enable_aux_sig);
            cont_sig <= cont + 1;
        elsif cont >= "00011000011010100010" then
            enable_aux <= '0';
            cont_sig <= "00000000000000000010";
        else
            enable_aux <= enable_aux_sig;
            cont_sig <= cont + 1;
        end if;
    end if;
end process;
```

```
end Behavioral;
```

Anexo top_Bloque_A_tercera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity top_Bloque_A_tercera_mod is
    Port ( clk : in std_logic;
           botones : in std_logic_vector ( 4 downto 0 );
           switches : in std_logic_vector ( 3 downto 0 );
           user_r_read_8_open : in std_logic;
           user_r_read_8_rden : in std_logic;
           user_r_read_8_eof : out std_logic;
           user_r_read_8_empty : out std_logic;
           salida : out std_logic_vector (7 downto 0)
           );
end top_Bloque_A_tercera_mod;

architecture Behavioral of top_Bloque_A_tercera_mod is

    component div_frec_tercera_mod
        Port ( clk : in std_logic;
               Switch_one : in std_logic;
               Switch_two : in std_logic;
               Switch_three : in std_logic;
               Switch_four : in std_logic;
               enable : out std_logic
               );
    end component;

    component Bloque_A_segunda_mod
        PORT ( clk : in std_logic;
               enable : in std_logic;
               boton_up : in std_logic;
```

```
            boton_down : in std_logic;
            boton_right : in std_logic;
            boton_left : in std_logic;
            boton_center : in std_logic;
            user_r_read_8_open : in std_logic;
            user_r_read_8_rden : in std_logic;
            user_r_read_8_eof : out std_logic;
            user_r_read_8_empty : out std_logic;
            salida: out std_logic_vector (7 downto 0)
        );
    END component;

signal enable_aux : std_logic := '0';

begin

    uut_bloq : Bloque_A_segunda_mod
        port map (
            clk => clk,
            enable => enable_aux,
            boton_up => botones(2),
            boton_down => botones(3),
            boton_right => botones(1),
            boton_left => botones(0),
            boton_center => botones(4),
            user_r_read_8_open => user_r_read_8_open,
            user_r_read_8_rden => user_r_read_8_rden,
            user_r_read_8_eof => user_r_read_8_eof,
            user_r_read_8_empty => user_r_read_8_empty,
            salida => salida
            );

    uut_div : div_frec_tercera_mod
        port map (
            clk => clk,
            Switch_one => switches(0),
```

```
        Switch_two => switches(1),
        Switch_three => switches(2),
        Switch_four => switches(3),
        enable => enable_aux
        );

end Behavioral;
```

Anexo Bloque_B

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Bloque_B is
    Port ( clk : in std_logic;
           empty : in std_logic;
           full : in std_logic;
           user_r_read_8_open : IN std_logic;
           user_w_write_8_open : IN std_logic;
                  entrada : IN std_logic_vector(7 DOWNTO 0);
           enable_read : out std_logic;
           enable_write : out std_logic;
           salida : OUT std_logic_vector(7 DOWNTO 0) );
end Bloque_B;

architecture Behavioral of Bloque_B is

signal cont_r : unsigned ( 15 downto 0 ) := ( others => '0' );
signal cont_r_sig : unsigned ( 15 downto 0 ) := ( others => '0' );

signal cont_w : unsigned ( 15 downto 0 ) := ( others => '0' );
signal cont_w_sig : unsigned ( 15 downto 0 ) := ( others => '0' );

signal stop : std_logic := '0';
signal stop_sig : std_logic := '0';

signal enable_read_aux : std_logic := '0';
signal enable_write_aux : std_logic := '0';

signal enable_read_hold : std_logic := '0';
signal enable_write_hold : std_logic := '0';
```

```
signal salida_aux : std_logic_vector( 7 downto 0 ) := ( others => '0' );
signal salida_hold : std_logic_vector (7 downto 0 ) := ( others => '0' );
signal salida_aux_hold : std_logic_vector (7 downto 0 ) := ( others => '0' );
begin

   p_seq: process ( clk )
   begin
      if rising_edge(clk) then
         cont_r <= cont_r_sig;
         cont_w <= cont_w_sig;
         stop <= stop_sig;
         if ( enable_write_aux = '1' ) then
            salida <= salida_aux;
            salida_hold <= salida_aux;
         else
            salida <= salida_aux_hold;
            salida_hold <= salida_aux_hold;
         end if;
         if ( user_r_read_8_open = '0' and user_w_write_8_open = '0' and empty = '0'
               and full = '0') then
            enable_write <= enable_write_aux;
            enable_write_hold <= enable_write_aux;
            enable_read <= enable_read_aux;
            enable_read_hold <= enable_read_aux;
         elsif ( user_r_read_8_open = '0' and user_w_write_8_open = '0' and empty = '1'
                     and full = '0') then
            enable_write <= enable_write_aux;
            enable_write_hold <= enable_write_aux;
            enable_read <= '0';
            enable_read_hold <= '0';
         else
            enable_write <= '0';
            enable_write_hold <= '0';
            enable_read <= '0';
            enable_read_hold <= '0';
```

```
        end if;
    end if;
end process;

p_comb : process (entrada, full,cont_r, cont_w, empty,full, enable_write_hold,
                enable_read_hold,salida_hold, stop )
begin
        salida_aux_hold <= salida_hold;
        if ( full = '0' and stop = '0') then
                if cont_w = "0111010100110000" then
                        enable_write_aux <= '1';
                        cont_w_sig <= ( others => '0' );
                else
                        enable_write_aux <= '0';
                        cont_w_sig <= cont_w +1;
                end if;
                stop_sig <= '0';
        else
                stop_sig <= '1';
                enable_write_aux <= '0';
                cont_w_sig <= ( others => '0' );
        end if;
        if ( empty = '0' ) then
                if cont_r = "0111010100110000" then
                        enable_read_aux <= '1';
                        cont_r_sig <= (others => '0');
                else
                        enable_read_aux <= '0';
                        cont_r_sig <= cont_r +1;
                end if;
                stop_sig <= '0';
        else
                stop_sig <= '1';
                enable_read_aux <= '0';
                cont_r_sig <= ( others => '0' );
        end if;
```

```
case entrada is
when "01000001" => salida_aux <= "01100001";
when "01000010" => salida_aux <= "01100010";
when "01000011" => salida_aux <= "01100011";
when "01000100" => salida_aux <= "01100100";
when "01000101" => salida_aux <= "01100101";
when "01000110" => salida_aux <= "01100110";
when "01000111" => salida_aux <= "01100111";
when "01001000" => salida_aux <= "01101000";
when "01001001" => salida_aux <= "01101001";
when "01001010" => salida_aux <= "01101010";
when "01001011" => salida_aux <= "01101011";
when "01001100" => salida_aux <= "01101100";
when "01001101" => salida_aux <= "01101101";
when "01001110" => salida_aux <= "01101110";
when "01001111" => salida_aux <= "01101111";
when "01010000" => salida_aux <= "01110000";
when "01010001" => salida_aux <= "01110001";
when "01010010" => salida_aux <= "01110010";
when "01010011" => salida_aux <= "01110011";
when "01010100" => salida_aux <= "01110100";
when "01010101" => salida_aux <= "01110101";
when "01010110" => salida_aux <= "01110110";
when "01010111" => salida_aux <= "01110111";
when "01011000" => salida_aux <= "01111000";
when "01011001" => salida_aux <= "01111001";
when "01011010" => salida_aux <= "01111010";
when "01011011" => salida_aux <= "01111011";
when "01011100" => salida_aux <= "01111100";

when "01100001" => salida_aux <= "01000001";
when "01100010" => salida_aux <= "01000010";
when "01100011" => salida_aux <= "01000011";
when "01100100" => salida_aux <= "01000100";
when "01100101" => salida_aux <= "01000101";
when "01100110" => salida_aux <= "01000110";
```

```
        when "01100111" => salida_aux <= "01000111";
        when "01101000" => salida_aux <= "01001000";
        when "01101001" => salida_aux <= "01001001";
        when "01101010" => salida_aux <= "01001010";
        when "01101011" => salida_aux <= "01001011";
        when "01101100" => salida_aux <= "01001100";
        when "01101101" => salida_aux <= "01001101";
        when "01101110" => salida_aux <= "01001110";
        when "01101111" => salida_aux <= "01001111";
        when "01110000" => salida_aux <= "01010000";
        when "01110001" => salida_aux <= "01010001";
        when "01110010" => salida_aux <= "01010010";
        when "01110011" => salida_aux <= "01010011";
        when "01110100" => salida_aux <= "01010100";
        when "01110101" => salida_aux <= "01010101";
        when "01110110" => salida_aux <= "01010110";
        when "01110111" => salida_aux <= "01010111";
        when "01111000" => salida_aux <= "01011000";
        when "01111001" => salida_aux <= "01011001";
        when "01111010" => salida_aux <= "01011010";
        when "01111011" => salida_aux <= "01011011";
        when "01111100" => salida_aux <= "01011100";
        when others => salida_aux <= entrada;
        end case;
end process;
end Behavioral;
```

Anexo top_Bloque_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use UNISIM.VComponents.all;

entity top_Bloque_B is
  Port (
           clk : in std_logic;
          entrada : in std_logic_vector(7 downto 0);
          user_w_write_8_wren : in std_logic;
          user_r_read_8_rden : in std_logic;
          user_r_read_8_open : in std_logic;
          user_w_write_8_open : in std_logic;
          user_w_write_8_full : out std_logic;
          user_r_read_8_empty : out std_logic;
          user_r_read_8_data : out std_logic_vector(7 downto 0)
                    );
end top_Bloque_B;

architecture Behavioral of top_Bloque_B is
component Bloque_B
   Port ( clk : in STD_LOGIC;
                                entrada : in std_logic_vector(7 downto 0);
                                empty : in std_logic;
                                full : in std_logic;
                                user_r_read_8_open : in std_logic;
                                user_w_write_8_open : in std_logic;
                                enable_read : out std_logic;
                                enable_write : out std_logic;
                                salida : out std_logic_vector(7 downto 0) );
end component;
component fifo_8x2048
           port ( clk: in std_logic;
             srst: in std_logic;
```

```
            din: in std_logic_VECTOR(7 downto 0);
            wr_en: in std_logic;
            rd_en: in std_logic;
            dout: out std_logic_VECTOR(7 downto 0);
            full: out std_logic;
            empty: out std_logic);
  end component;
signal full_aux : std_logic := '0';
signal empty_aux : std_logic := '0';
signal enable_read_aux : std_logic := '0';
signal enable_write_aux : std_logic := '0';
signal salida_aux : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal entrada_aux : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
begin
   uut_bloque_b : Bloque_B
   port map ( clk => clk,
              entrada => entrada_aux,
              empty => empty_aux,
              full => full_aux,
              user_r_read_8_open => user_r_read_8_open,
              user_w_write_8_open => user_w_write_8_open,
              enable_read => enable_read_aux,
              enable_write => enable_write_aux,
              salida => salida_aux    );
   fifo_8_w : fifo_8x2048
      port map(
         clk       => clk,
         srst      => '0',
         din       => entrada,
         wr_en     => user_w_write_8_wren,
         rd_en     => enable_read_aux,
         dout      => entrada_aux,
         full      => user_w_write_8_full,
         empty     => empty_aux
   );
   fifo_8_r : fifo_8x2048
```

```
    port map(
        clk       => clk,
        srst      => '0',
        din       => salida_aux,
        wr_en     => enable_write_aux,
        rd_en     => user_r_read_8_rden,
        dout      => user_r_read_8_data,
        full      => full_aux,
        empty     => user_r_read_8_empty
    );

end Behavioral;
```

Anexo tb_Bloque_B

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity tb_Bloque_B is
--  Port ( );
end tb_Bloque_B;

architecture Behavioral of tb_Bloque_B is
component Bloque_B
    Port ( clk : in std_logic;
           entrada : in std_logic_vector(7 downto 0);
           empty : in std_logic;
           full : in std_logic;
           user_r_read_8_open : in std_logic;
           user_w_write_8_open : in std_logic;
           enable_read : out std_logic;
           enable_write : out std_logic;
           salida : out std_logic_vector(7 downto 0) );
end component;

signal clk : std_logic := '0';
signal entrada : std_logic_vector(7 downto 0) := ( others => '0' );
signal empty : std_logic := '0';
signal full : std_logic := '0';
signal user_r_read_8_open : std_logic := '0';
signal user_w_write_8_open : std_logic := '0';
signal enable_read : std_logic := '0';
signal enable_write : std_logic := '0';
signal salida : std_logic_vector(7 downto 0) := ( others => '0');

constant clk_period : time := 100 ns;
```

```
begin

uut : Bloque_B
   Port map( clk => clk,
         entrada => entrada,
         empty => empty,
         full => full,
         user_r_read_8_open => user_r_read_8_open,
         user_w_write_8_open => user_w_write_8_open,
         enable_read => enable_read,
         enable_write => enable_write,
         salida => salida );

   p_clock : process
   begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
   end process;

   stim_process: process
      begin
         empty <= '0';
         full <= '0';
         user_r_read_8_open <= '0';
         user_w_write_8_open <= '1';
         entrada <= "01100001";
         wait for 3 ms;
         entrada <= "01100010";
         wait for 3 ms;
         entrada <= "01100011";
         wait for 3 ms;
         entrada <= "01100100";
         wait for 3 ms;
```

```
        entrada <= "01100101";
        wait for 3 ms;
        entrada <= "01100111";
        wait for 3 ms;
        entrada <= "01100100";
        wait for 3 ms;
        entrada <= "01101001";
        wait for 3 ms;
        entrada <= "01101010";
        wait for 3 ms;
        entrada <= "01101011";
        wait for 3 ms;
        entrada <= "01101100";
        wait for 3 ms;
        entrada <= "01101001";
        wait for 3 ms;
        entrada <= "01101010";
        user_r_read_8_open <= '0';
        user_w_write_8_open <= '0';
        wait for 100 ms;
        user_r_read_8_open <= '1';
        user_w_write_8_open <= '0';
        wait;
    end process;

end Behavioral;
```

Anexo Bloque_B_primera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Bloque_B_primera_mod is
    Port ( clk : in std_logic;
           Switch_one : in std_logic;
           Switch_two : in std_logic;
           Switch_three : in std_logic;
           entrada : in std_logic_vector(7 DOWNTO 0);
           empty : in std_logic;
           full : in std_logic;
           user_r_read_8_open : in std_logic;
           user_w_write_8_open : in std_logic;
           enable_read : out std_logic;
           enable_write : out std_logic;
           salida : out std_logic_vector(7 DOWNTO 0) );
end Bloque_B_primera_mod;

architecture Behavioral of Bloque_B_primera_mod is

signal enable_read_aux : std_logic := '0';
signal enable_write_aux : std_logic := '0';

signal enable_read_hold : std_logic := '0';
signal enable_write_hold : std_logic := '0';

signal stop : std_logic := '0';
signal stop_sig : std_logic := '0';

signal cont_r : unsigned ( 31 downto 0 ) := ( others => '0' );
signal cont_r_sig : unsigned ( 31 downto 0 ) := ( others => '0' );
```

```
signal cont_w : unsigned ( 31 downto 0 ) := ( others => '0' );
signal cont_w_sig : unsigned ( 31 downto 0 ) := ( others => '0' );

signal salida_aux : unsigned( 7 DOWNTO 0 ) := "00110000";
signal salida_hold : unsigned (7 downto 0 ) := "00110000";
signal salida_aux_hold : unsigned (7 downto 0 ) := "00110000";

signal start_write : std_logic := '0';
signal start_write_sig : std_logic := '0';

signal start_write_flag : std_logic := '0';
signal start_write_flag_sig : std_logic := '0';

signal letra_result : unsigned (4 downto 0) := ( others => '0' );
signal letra_result_sig : unsigned ( 4 downto 0 ) := ( others => '0' );

signal letra : unsigned ( 3 downto 0 ) := ( others => '0');
signal letra_sig : unsigned ( 3 downto 0 ) := ( others => '0');

signal decenas : std_logic := '0';
signal decenas_sig : std_logic := '0';

signal negativo : std_logic := '0';
signal negativo_sig : std_logic := '0';

signal fallo : std_logic := '0';
signal fallo_sig : std_logic := '0';

signal secure : std_logic := '0';
signal secure_sig : std_logic := '0';

signal cont_sum : unsigned ( 1 downto 0 ) := ( others => '0' );
signal cont_sum_sig : unsigned ( 1 downto 0 ) := ( others => '0' );

signal cont_err : unsigned ( 14 downto 0 ) := ( others => '0' );
signal cont_err_sig : unsigned ( 14 downto 0 ) := ( others => '0' );
```

```
signal resultado : unsigned ( 15 downto 0 ) := "0000000000110000";
signal resultado_sig : unsigned ( 15 downto 0 ) := "0000000000110000";

signal resultado_decenas : unsigned ( 7 downto 0 ) := "00110000";
signal resultado_decenas_sig : unsigned ( 7 downto 0 ) := "00110000";

signal sum_sig : unsigned ( 7 downto 0 ) := "00110000";

signal sum_alternative : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal sum_alternative_sig : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

signal sum_alternative_hold : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal sum_alternative_hold_sig : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

signal sumador : unsigned ( 15 downto 0 ) := ( others => '0' );
signal sumador_hold : unsigned ( 7 downto 0 ) := ( others => '0' );

signal sumando : unsigned ( 7 downto 0 ) := ( others => '0' );
signal sumando_sig : unsigned ( 7 downto 0 ) := ( others => '0' );

signal sumando_dec : unsigned ( 7 downto 0 ) := ( others => '0' );
signal sumando_dec_sig : unsigned ( 7 downto 0 ) := ( others => '0' );

begin

   p_seq: process ( clk)
   begin
      if rising_edge(clk) then

         if ( sum_alternative /= entrada )
                              and user_w_write_8_open = '1' then
            sumando <= unsigned(entrada);
            sumando_dec <= sumando_dec_sig;
            decenas <= decenas_sig;
            negativo <= negativo_sig;
```

```
elsif cont_sum = "00" then
   sumando <= "00110000";
   sumando_dec <= "00110000";
   decenas <= decenas_sig;
   negativo <= negativo_sig;
elsif user_r_read_8_open = '1' then
   sumando <= "00110000";
   sumando_dec <= "00110000";
   decenas <= '0';
   negativo <= '0';
else
   sumando <= "00110000";
   sumando_dec <= sumando_dec_sig;
   decenas <= decenas_sig;
   negativo <= negativo_sig;
end if;

if user_w_write_8_open = '1' then
   sum_alternative <= sum_alternative_sig;
   sum_alternative_hold <= sum_alternative_sig;
elsif user_r_read_8_open = '1' then
   sum_alternative <= entrada;
   sum_alternative_hold <= entrada;
else
   sum_alternative <= sum_alternative_hold_sig;
   sum_alternative_hold <= sum_alternative_hold_sig;
end if;

cont_sum <= cont_sum_sig;
cont_r <= cont_r_sig;
cont_w <= cont_w_sig;
stop <= stop_sig;
start_write <= start_write_sig;
start_write_flag <= start_write_flag_sig;
sumador_hold <= sumador(7 downto 0);
fallo <= fallo_sig;
```

```
secure <= secure_sig;
letra <= letra_sig;
letra_result <= letra_result_sig;
cont_err <= cont_err_sig;
salida <= std_logic_vector(salida_aux);
salida_hold <= salida_aux;
cont_sum <= cont_sum_sig;
        resultado_decenas <= sumando_dec_sig;

        if ( user_r_read_8_open = '0' and user_w_write_8_open = '1' and empty = '0'
    and full = '0') then

   enable_write <= '0';
   enable_write_hold <= '0';
   enable_read <= enable_read_aux;
   enable_read_hold <= enable_read_aux;

elsif (start_write = '1') then
   enable_write <= enable_write_aux;
   enable_write_hold <= enable_write_aux;
   enable_read <= '0';
   enable_read_hold <= '0';
else
   enable_write <= '0';
   enable_write_hold <= '0';
   enable_read <= '0';
   enable_read_hold <= '0';
end if;

        case sumador(7 downto 0) is
                when "00000000" =>
                        resultado <= "0000000000110000";
                when "00000001" =>
                        resultado <= "0000000000110001";
                when "00000010" =>
                        resultado <= "0000000000110010";
```

```
                    when "00000011" =>
                        resultado <= "0000000000110011";
                    when "00000100" =>
                        resultado <= "0000000000110100";
                    when "00000101" =>
                        resultado <= "0000000000110101";
                    when "00000110" =>
                        resultado <= "0000000000110110";
                    when "00000111" =>
                        resultado <= "0000000000110111";
                    when "00001000" =>
                        resultado <= "0000000000111000";
                    when "00001001" =>
                        resultado <= "0000000000111001";
                    when others =>
                        resultado <= "0000000000110000";
                end case;

        end if;




    end process;

    p_comb : process (entrada, letra, letra_result, salida_aux_hold, resultado, sumando_dec,
                      fallo, cont_err, full,cont_r, cont_w, empty, full, enable_write_hold,
                                          enable_read_hold,salida_hold,  stop,  user_r_read_8_open,
cont_sum,
                                          user_w_write_8_open,          start_write_flag,start_write,
sumando, resultado_decenas,
                                          cont_sum,                decenas,                negativo,
sum_alternative,resultado_decenas_sig,
                                          Switch_one,                                      Switch_two,
Switch_three,sum_alternative_hold, sumador_hold )



    begin
```

```
    if entrada /= sum_alternative and user_w_write_8_open = '1' then
        if cont_sum = "11" then
            cont_sum_sig <= cont_sum;
            secure_sig <= '1';
        else
            cont_sum_sig <= cont_sum +1;
            secure_sig <= '0';
        end if;
    elsif user_r_read_8_open = '1' then
        secure_sig <= '0';
        cont_sum_sig <= "00";
    else
        cont_sum_sig <= cont_sum;
        secure_sig <= '0';
    end if;

         sum_alternative_sig <= entrada;
         sum_alternative_hold_sig <= sum_alternative;
         if ( Switch_one = '1' and Switch_two = '0' and Switch_three = '0' ) then
                    if sumando > "00110000" and sumando < "00111010" and cont_sum /= "00" and secure = '0'
then
                                sumador(7 downto 0) <= sumando + sumador_hold - "00110000" ;
            sumador(15 downto 8) <= ( others => '0' );
                                sumando_dec_sig <= sumando_dec;
                                decenas_sig <= decenas;
                    elsif sumador_hold >= "01011010" then
            sumador(7 downto 0) <= sumador_hold - "01011010";
            sumador(15 downto 8) <= ( others => '0' );
            sumando_dec_sig <=  sumando_dec + 9;
            decenas_sig <= '1';
        elsif sumador_hold >= "01010000" then
            sumador(7 downto 0) <= sumador_hold - "01010000";
            sumador(15 downto 8) <= ( others => '0' );
            sumando_dec_sig <=  sumando_dec + 8;
            decenas_sig <= '1';
```

```
elsif sumador_hold >= "01000110" then
    sumador(7 downto 0) <= sumador_hold - "01000110";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 7;
    decenas_sig <= '1';
elsif sumador_hold >= "00111100" then
    sumador(7 downto 0) <= sumador_hold - "00111100";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 6;
    decenas_sig <= '1';
elsif sumador_hold >= "00110010" then
    sumador(7 downto 0) <= sumador_hold - "00110010";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 5;
    decenas_sig <= '1';
elsif sumador_hold >= "00101000" then
    sumador(7 downto 0) <= sumador_hold - "00101000";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 4;
    decenas_sig <= '1';
elsif sumador_hold >= "00011110" then
    sumador(7 downto 0) <= sumador_hold - "00011110";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 3;
    decenas_sig <= '1';
elsif sumador_hold >= "00010100" then
    sumador(7 downto 0) <= sumador_hold - "00010100";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 2;
    decenas_sig <= '1';
elsif sumador_hold >= "00001010" then
    sumador(7 downto 0) <= sumador_hold - "00001010";
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <=  sumando_dec + 1;
    decenas_sig <= '1';
elsif user_r_read_8_open = '1' then
```

```
            sumador <= ( others => '0' );
            sumador(15 downto 8) <= ( others => '0' );
            decenas_sig <= '0';
            sumando_dec_sig <= "00110000";
        else
            sumador(7 downto 0) <= sumador_hold;
            sumador(15 downto 8) <= sumador_hold;
            decenas_sig <= decenas;
            sumando_dec_sig <= sumando_dec;
        end if;
        negativo_sig <= '0';
         elsif ( Switch_one = '0' and Switch_two = '1' and Switch_three = '0' ) then
                if sumando > "00110000" and sumando < "00111010" and cont_sum = "01" and secure = '0'
then
                        sumador(7 downto 0) <= sumador_hold + sumando - "00110000" ;
                        negativo_sig <= '0';
                elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11"
                        and sumador_hold > ( unsigned(entrada) - "00110000" ) and secure = '0'  then
                        sumador(7 downto 0) <= sumador_hold - sumando - "00110000";
                        negativo_sig <= '0';
                elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11"
                        and sumador_hold < ( unsigned(entrada) - "00110000" ) and secure = '0'  then
                        sumador(7 downto 0) <=  sumando - sumador_hold - "00110000" ;
                        negativo_sig <= '1';
        elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11"
            and sumador_hold = ( unsigned(entrada) - "00110000" ) and secure = '0'  then
            sumador(7 downto 0) <=  "00000000" ;
            negativo_sig <= '0';
                elsif sumador_hold >= "00001010" then
                        sumador(7 downto 0) <= sumador_hold - "00001010";
                        negativo_sig <= negativo;
                elsif user_r_read_8_open = '1' then
                        sumador(7 downto 0) <= ( others => '0' );
                        negativo_sig <= '0';
                else
                        sumador(7 downto 0) <= sumador_hold;
            negativo_sig <= negativo;
```

```
                end if;
                sumador(15 downto 8) <= ( others => '0' );
                sumando_dec_sig <= sumando_dec;
        decenas_sig <= '0';
         elsif ( Switch_one = '0' and Switch_two = '0' and Switch_three = '1' ) then
                if sumando > "00110000" and sumando < "00111010" and cont_sum = "01" and secure = '0'
then
                        sumador(7 downto 0) <= sumando - "00110000";
                        sumador(15 downto 8) <= ( others => '0' );
                        sumando_dec_sig <= sumando_dec;
                        decenas_sig <= decenas;
                elsif sumando > "00110000" and sumando < "00111010" and cont_sum = "11" and secure =
'0'  then
                        sumador <= (sumando - "00110000")* sumador_hold;
                        sumando_dec_sig <= sumando_dec;
                        decenas_sig <= decenas;
                elsif sumador_hold >= "01011010" then
           sumador(7 downto 0) <= sumador_hold - "01011010";
           sumador(15 downto 8) <= ( others => '0' );
           sumando_dec_sig <= "00111001";
           decenas_sig <= '1';
        elsif sumador_hold >= "01010000" then
           sumador(7 downto 0) <= sumador_hold - "01010000";
           sumador(15 downto 8) <= ( others => '0' );
           sumando_dec_sig <= "00111000";
           decenas_sig <= '1';
        elsif sumador_hold >= "01000110" then
           sumador(7 downto 0) <= sumador_hold - "01000110";
           sumador(15 downto 8) <= ( others => '0' );
           sumando_dec_sig <= "00110111";
           decenas_sig <= '1';
        elsif sumador_hold >= "00111100" then
           sumador(7 downto 0) <= sumador_hold - "00111100";
           sumador(15 downto 8) <= ( others => '0' );
           sumando_dec_sig <= "00110110";
           decenas_sig <= '1';
        elsif sumador_hold >= "00110010" then
```

```
        sumador(7 downto 0) <= sumador_hold - "00110010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00110101";
        decenas_sig <= '1';
    elsif sumador_hold >= "00101000" then
        sumador(7 downto 0) <= sumador_hold - "00101000";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00110100";
        decenas_sig <= '1';
    elsif sumador_hold >= "00011110" then
        sumador(7 downto 0) <= sumador_hold - "00011110";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00110011";
        decenas_sig <= '1';
    elsif sumador_hold >= "00010100" then
        sumador(7 downto 0) <= sumador_hold - "00010100";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00110010";
        decenas_sig <= '1';
    elsif sumador_hold >= "00001010" then
        sumador(7 downto 0) <= sumador_hold - "00001010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <= "00110001";
        decenas_sig <= '1';
    elsif user_r_read_8_open = '1' then
        sumador <= ( others => '0' );
        sumador(15 downto 8) <= ( others => '0' );
        decenas_sig <= '0';
        sumando_dec_sig <= "00110000";
    else
        sumador(7 downto 0) <= sumador_hold;
        sumador(15 downto 8) <= sumador_hold;
        decenas_sig <= decenas;
        sumando_dec_sig <= sumando_dec;
    end if;
    negativo_sig <= '0';
```

```
 else
    if sumando > "00110000" and sumando < "00111010" and cont_sum /= "00" and secure = '0' then
    sumador(7 downto 0) <= sumando + sumador_hold - "00110000" ;
    sumador(15 downto 8) <= ( others => '0' );
    sumando_dec_sig <= sumando_dec;
    decenas_sig <= decenas;
elsif sumador_hold >= "01011010" then
        sumador(7 downto 0) <= sumador_hold - "01011010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 9;
        decenas_sig <= '1';
    elsif sumador_hold >= "01010000" then
        sumador(7 downto 0) <= sumador_hold - "01010000";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 8;
        decenas_sig <= '1';
    elsif sumador_hold >= "01000110" then
        sumador(7 downto 0) <= sumador_hold - "01000110";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 7;
        decenas_sig <= '1';
    elsif sumador_hold >= "00111100" then
        sumador(7 downto 0) <= sumador_hold - "00111100";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 6;
        decenas_sig <= '1';
    elsif sumador_hold >= "00110010" then
        sumador(7 downto 0) <= sumador_hold - "00110010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 5;
        decenas_sig <= '1';
    elsif sumador_hold >= "00101000" then
        sumador(7 downto 0) <= sumador_hold - "00101000";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 4;
        decenas_sig <= '1';
```

```
    elsif sumador_hold >= "00011110" then
        sumador(7 downto 0) <= sumador_hold - "00011110";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 3;
        decenas_sig <= '1';
    elsif sumador_hold >= "00010100" then
        sumador(7 downto 0) <= sumador_hold - "00010100";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 2;
        decenas_sig <= '1';
    elsif sumador_hold >= "00001010" then
        sumador(7 downto 0) <= sumador_hold - "00001010";
        sumador(15 downto 8) <= ( others => '0' );
        sumando_dec_sig <=  sumando_dec + 1;
        decenas_sig <= '1';
    elsif user_r_read_8_open = '1' then
        sumador <= ( others => '0' );
        sumador(15 downto 8) <= ( others => '0' );
        decenas_sig <= '0';
        sumando_dec_sig <= "00110000";
    else
        sumador(7 downto 0) <= sumador_hold;
        sumador(15 downto 8) <= sumador_hold;
        decenas_sig <= decenas;
        sumando_dec_sig <= sumando_dec;
    end if;
    negativo_sig <= '0';
end if;
salida_aux_hold <= salida_hold;

if user_w_write_8_open = '1' then
        start_write_flag_sig <= '1';
        start_write_sig <= '0';
elsif start_write_flag = '1' and user_w_write_8_open = '0' then
        if fallo = '1' and letra = "1000" then
                start_write_flag_sig <= '0';
```

```
                start_write_sig <= '0';
            elsif ( fallo = '0' and ( letra_result = "10011" and decenas = '0' and negativo = '0' ) )
                or ( fallo = '0' and ( letra_result = "10100" and ( decenas = '1' or negativo = '1' ) ) ) then
                start_write_flag_sig <= '0';
                start_write_sig <= '0';
            else
                start_write_flag_sig <= '1';
                start_write_sig <= '1';
            end if;
        else
            start_write_flag_sig <= '0';
            start_write_sig <= '0';
        end if;

        if ( full = '0' and stop = '0' and start_write = '1') then
            if cont_w >= "00000000000000001000100010111000" then
                enable_write_aux <= '1';
                cont_w_sig <= ( others => '0' );
            else
                enable_write_aux <= '0';
                cont_w_sig <= cont_w +1;
            end if;
        else
                stop_sig <= '1';
            enable_write_aux <= '0';
          cont_w_sig <= ( others => '0' );
        end if;

 if ( empty = '0' ) then
            if cont_r = "0111010100110000" then
                enable_read_aux <= '1';
                cont_r_sig <= (others => '0');
            else
                enable_read_aux <= '0';
                cont_r_sig <= cont_r +1;
            end if;
```

```
                stop_sig <= '0';
        else
                if cont_r = "0111010100110000" then
                        if user_r_read_8_open = '1' then
                                stop_sig <= '1';
                        else
                                stop_sig <= '0';
                        end if;
                        cont_r_sig <= ( others => '0' );
                elsif stop = '1' then
                        stop_sig <= '1';
                        cont_r_sig <= cont_r+1;
                else
                        cont_r_sig <= cont_r+1;
                        stop_sig <= stop;
                end if;
                enable_read_aux <= '0';
        end if;



        if user_r_read_8_open = '1' then
     fallo_sig <= '0';
    elsif secure = '1' then
       fallo_sig <= '1';
    elsif user_w_write_8_open = '1' and cont_sum = "00" then
     fallo_sig <= '0';
    elsif ( cont_sum = "11" or cont_sum = "01" or cont_sum = "00" )
                and sum_alternative = "00100000" then
           fallo_sig <= '1';
     elsif ( cont_sum = "10" ) and ( sum_alternative > "00110000"
                and sum_alternative < "00111010" ) then
           fallo_sig <= '1';
    elsif ( cont_sum = "00" or cont_sum = "01" or cont_sum = "10" or cont_sum = "11" )
                and ( ( sum_alternative < "00110000" or sum_alternative > "00111010" )
       and sum_alternative /= "000000" and sum_alternative /= "00100000" )  then
       fallo_sig <= '1';
```

```
elsif ( cont_sum = "10" or cont_sum = "01" ) and user_w_write_8_open = '0' then
   fallo_sig <= '1';
elsif ( cont_sum = "01" or cont_sum = "10" or cont_sum = "11" )
            and ( ( sum_alternative > "00110000" and sum_alternative < "00111010" )
   or sum_alternative = "000000" or sum_alternative = "00100000" )  then
   fallo_sig <= fallo;
elsif user_w_write_8_open = '0' then
   fallo_sig <= fallo;
else
   fallo_sig <= '1';
end if;



    if fallo = '0' and start_write = '1' then
            letra_sig <= ( others => '0' );
            cont_err_sig <= ( others => '0' );
            if cont_w = "0111010100110000" then
                    if letra_result = "00000" then
                            salida_aux <= "01000101";
                            letra_result_sig <= letra_result + 1;
                            stop_sig <= '0';
                    elsif letra_result = "00001" then
                            salida_aux <= "01001100";
                            letra_result_sig <= letra_result + 1;
                            stop_sig <= '0';
                    elsif letra_result = "00010" then
                            salida_aux <= "00100000";
                            letra_result_sig <= letra_result + 1;
                            stop_sig <= '0';
                    elsif letra_result = "00011" then
                            salida_aux <= "01010010";
                            letra_result_sig <= letra_result + 1 ;
                            stop_sig <= '0';
                    elsif letra_result = "00100" then
                            salida_aux <= "01000101";
                            letra_result_sig <= letra_result + 1;
```

```
            stop_sig <= '0';
        elsif letra_result = "00101" then
            salida_aux <= "01010011";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "00110" then
            salida_aux <= "01010101";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "00111" then
            salida_aux <= "01001100";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "01000" then
            salida_aux <= "01010100";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "01001" then
            salida_aux <= "01000001";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "01010" then
            salida_aux <= "01000100";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "01011" then
            salida_aux <= "01001111";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "01100" then
            salida_aux <= "00100000";
            letra_result_sig <= letra_result + 1;
            stop_sig <= '0';
        elsif letra_result = "01101" then
            salida_aux <= "01000101";
            letra_result_sig <= letra_result + 1;
```

```
                stop_sig <= '0';
            elsif letra_result = "01110" then
                salida_aux <= "01010011";
                letra_result_sig <= letra_result + 1;
                stop_sig <= '0';
            elsif letra_result = "01111" then
                salida_aux <= "00100000";
                letra_result_sig <= letra_result + 1;
                stop_sig <= '0';
            elsif letra_result = "10000" and decenas = '1' then
                salida_aux <= resultado_decenas;
                letra_result_sig <= letra_result + 1;
                stop_sig <= '0';
            elsif letra_result = "10000" and negativo = '1' then
                salida_aux <= "00101101";
                letra_result_sig <= letra_result + 1;
                stop_sig <= '0';
            elsif ( letra_result = "10001" and decenas = '1' )
                or ( letra_result = "10001" and negativo = '1' )
                or ( letra_result = "10000" and decenas = '0' ) then
                 salida_aux <= resultado(7 downto 0);
                 letra_result_sig <= letra_result + 1;
                 stop_sig <= '0';
            elsif ( letra_result = "10010" or letra_result = "10001"  ) then
                salida_aux <= "00101110";
                letra_result_sig <= letra_result + 1;
                stop_sig <= '0';
            else
                salida_aux <= ( others => '0' );
                letra_result_sig <= letra_result;
                stop_sig <= '1';
            end if;
        else
            salida_aux <= salida_aux_hold;
            letra_result_sig <= letra_result;
            stop_sig <= stop;
```

```
            end if;

        else
            letra_result_sig <= ( others => '0' );
            if start_write = '1' then
                if cont_err = "111010100110000" then
                    if letra = "0000" then
                        salida_aux <= "01000101";
                        letra_sig <= letra +1 ;
                        stop_sig <= '0';
                    elsif letra = "0001" then
                        salida_aux <= "01010010";
                        letra_sig <= letra +1;
                        stop_sig <= '0';
                    elsif letra = "0010" then
                        salida_aux <= "01010010";
                        letra_sig <= letra +1;
                        stop_sig <= '0';
                    elsif letra = "0011" then
                        salida_aux <= "01001111";
                        letra_sig <= letra +1;
                        stop_sig <= '0';
                    elsif letra = "0100" then
                        salida_aux <= "01010010";
                        letra_sig <= letra +1;
                        stop_sig <= '0';
                    elsif letra = "0101" then
                        salida_aux <= "00101110";
                        letra_sig <= letra + 1;
                        stop_sig <= '0';
                    elsif letra = "0110" then
                        salida_aux <= "00101110";
                        letra_sig <= letra + 1;
                        stop_sig <= '0';
                    elsif letra = "0111" then
                        salida_aux <= "00101110";
```

```
                                        letra_sig <= letra + 1;
                                        stop_sig <= '0';
            else
               salida_aux <= ( others => '0' );
               letra_sig <= ( others => '0' );
               stop_sig <= '1';
                                end if;
                                cont_err_sig <= cont_err +1;
                        elsif cont_err = "111010100110101" then
                                letra_sig <= letra ;
                                stop_sig <= stop;
                                salida_aux <= salida_aux_hold;
                                cont_err_sig <= "000000000000101";
                        else
                                letra_sig <= letra;
                                stop_sig <= stop;
                                salida_aux <= salida_aux_hold;
                                cont_err_sig <= cont_err +1;
                        end if;
                 else
                        cont_err_sig <= cont_err;
                        letra_sig <= (others => '0' );
                        stop_sig <= '0';
                        salida_aux <= salida_aux_hold;
                end if;
        end if;
end process;
end Behavioral;
```

Anexo top_Bloque_B_pimera_mod

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity top_Bloque_B_primera_mod is
  Port ( clk : in std_logic;
         Switch_one : in std_logic;
         Switch_two : in std_logic;
         Switch_three : in std_logic;
         entrada : in std_logic_vector(7 DOWNTO 0);
         user_w_write_8_wren : in std_logic;
         user_r_read_8_rden : in std_logic;
         user_r_read_8_open : in std_logic;
         user_w_write_8_open : in std_logic;
         user_w_write_8_full : out std_logic;
         user_r_read_8_empty : out std_logic;
         user_r_read_8_data : out std_logic_vector(7 DOWNTO 0) );
end top_Bloque_B_primera_mod;

architecture Behavioral of top_Bloque_B_primera_mod is

component Bloque_B_primera_mod
     Port ( clk : in std_logic;
         Switch_one : in std_logic;
         Switch_two : in std_logic;
         Switch_three : in std_logic;
         entrada : in std_logic_vector(7 DOWNTO 0);
         empty : in std_logic;
         full : in std_logic;
         user_r_read_8_open : in std_logic;
         user_w_write_8_open : in std_logic;
```

```
        enable_read : out std_logic;
        enable_write : out std_logic;
        salida : out std_logic_vector(7 DOWNTO 0) );
end component;

component fifo_8x2048
   port (
     clk: in std_logic;
     srst: in std_logic;
     din: in std_logic_VECTOR(7 downto 0);
     wr_en: in std_logic;
     rd_en: in std_logic;
     dout: out std_logic_VECTOR(7 downto 0);
     full: out std_logic;
     empty: out std_logic);
  end component;

signal full_aux : std_logic := '0';
signal empty_aux : std_logic := '0';
signal reset_8 : std_logic;
signal enable_read_aux : std_logic := '0';
signal enable_write_aux : std_logic := '0';
signal salida_aux : std_logic_vector ( 7 downto 0 ) := ( others => '0' );
signal entrada_aux : std_logic_vector ( 7 downto 0 ) := ( others => '0' );

signal enable_primera_mod : std_logic := '0';
signal enable_default : std_logic := '0';
begin

   uut_bloque_b : Bloque_B_primera_mod
   port map (
            clk => clk,
            Switch_one => Switch_one,
            Switch_two => Switch_two,
            Switch_three => Switch_three,
            entrada => entrada_aux,
```

```
            empty => empty_aux,
            full => full_aux,
            user_r_read_8_open => user_r_read_8_open,
            user_w_write_8_open => user_w_write_8_open,
            enable_read => enable_read_aux,
            enable_write => enable_write_aux,
            salida => salida_aux
    );

    fifo_8_w : fifo_8x2048
        port map(
            clk        => clk,
            srst       => '0',
            din        => entrada,
            wr_en      => user_w_write_8_wren,
            rd_en      => enable_read_aux,
            dout       => entrada_aux,
            full       => user_w_write_8_full,
            empty      => empty_aux
    );
    fifo_8_r : fifo_8x2048
        port map(
            clk        => clk,
            srst       => '0',
            din        => salida_aux,
            wr_en      => enable_write_aux,
            rd_en      => user_r_read_8_rden,
            dout       => user_r_read_8_data,
            full       => full_aux,
            empty      => user_r_read_8_empty
        );


end Behavioral;
```

Anexo tb_Bloque_B_primer_mod

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity tb_Bloque_B is
--  Port ( );
end tb_Bloque_B;

architecture Behavioral of tb_Bloque_B is
component Bloque_B_primera_mod
    Port ( clk : in STD_LOGIC;
           Switch_one : IN std_logic;
           Switch_two : IN std_logic;
           Switch_three : IN std_logic;
           entrada : IN std_logic_vector(7 DOWNTO 0);
           empty : in std_logic;
           full : in std_logic;
           user_r_read_8_open : IN std_logic;
           user_w_write_8_open : IN std_logic;
           enable_read : out std_logic;
           enable_write : out std_logic;
           salida : OUT std_logic_vector(7 DOWNTO 0) );
end component;

signal clk : STD_LOGIC := '0';
signal Switch_one : std_logic := '0';
signal Switch_two : std_logic := '0';
signal Switch_three : std_logic := '0';
signal entrada : std_logic_vector(7 DOWNTO 0) := ( others => '0' );
signal empty : std_logic := '0';
signal full : std_logic := '0';
```

```
signal user_r_read_8_open : std_logic := '0';
signal user_w_write_8_open : std_logic := '0';
signal enable_read : std_logic := '0';
signal enable_write : std_logic := '0';
signal salida : std_logic_vector(7 DOWNTO 0) := ( others => '0');

constant clk_period : time := 10 ns;

begin

uut : Bloque_B_primera_mod
   Port map( clk => clk,
         Switch_one => Switch_one,
         Switch_two => Switch_two,
         Switch_three => Switch_three,
         entrada => entrada,
         empty => empty,
         full => full,
         user_r_read_8_open => user_r_read_8_open,
         user_w_write_8_open => user_w_write_8_open,
         enable_read => enable_read,
         enable_write => enable_write,
         salida => salida );

   p_clock : process
   begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
   end process;

   stim_process: process
      begin
         Switch_one <= '1';
         Switch_two <='0';
```

```
Switch_three <= '0';
empty <= '0';
full <= '0';
entrada <= "00000000";
user_w_write_8_open <= '0';
user_r_read_8_open <= '0';
wait for 500us;
entrada <= "00000000";
user_w_write_8_open <= '1';
wait for 500 us;
entrada <= "00110100";
wait for 500 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00110101";
wait for 500 us;
user_w_write_8_open <= '0';
wait for 7000 us;
user_r_read_8_open <= '1';
wait for 1000 us;
user_r_read_8_open <= '0';
wait for 1000us;


Switch_one <= '0';
Switch_two <='1';
Switch_three <= '0';
wait for 1000us;
user_w_write_8_open <= '1';
wait for 500ns;
entrada <= "00110011";
wait for 1000 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00110110";
wait for 500 us;
```

```
user_w_write_8_open <= '0';
wait for 7000 us;
user_r_read_8_open <= '1';
wait for 6000 us;
user_r_read_8_open <= '0';
wait for 4000us;

Switch_one <= '0';
Switch_two <='0';
Switch_three <= '1';
wait for 1000us;
user_w_write_8_open <= '1';
wait for 500ns;
entrada <= "00111001";
wait for 1000 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00110100";
wait for 500 us;
wait for 300 us;
empty <= '0';
user_w_write_8_open <= '0';

wait for 7000 us;
user_r_read_8_open <= '1';
wait for 6000 us;
user_r_read_8_open <= '0';
wait for 4000us;

user_w_write_8_open <= '1';
wait for 500ns;
entrada <= "00110001";
wait for 1000 us;
entrada <= "00100000";
wait for 500 us;
entrada <= "00111101";
```

```
        wait for 500 us;
        wait for 300 us;
        empty <= '0';
        user_w_write_8_open <= '0';

        wait for 7000 us;
        user_r_read_8_open <= '1';
        wait for 6000 us;
        user_r_read_8_open <= '0';
        wait for 4000us;

        wait ;

    end process;

end Behavioral;
```

REFERENCIAS

[1] ZedBoard, «Hardware User's Guide» [En Linea] Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf .

[2] Xillybus, «Xillybus host application programming guide for Linux» [En Línea] Available: http://xillybus.com/downloads/doc/xillybus_host_programming_guide_linux.pdf

[3] Xyllybus, «Xillybus FPGA designer's guide» [En Linea] Available: http://www.xillybus.com/downloads/doc/xillybus_fpga_api.pdf

[4] Xilinx, «Dowloands» [En Linea] Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2016-4.html

[5] Xillybus «Xillinux, A Linux distribution for ZedBoard, ZyBo, MycroZed and Sockit | xillybus.com» [En Linea] Available: http://xillybus.com/xillinux

[6] VMware, «Dowloads» [En Linea] Available: https://www.vmware.com/latam/products/player/playerpro-evaluation.html .

[7] USB-Image-Tool, «usbit» [En Linea] Available: http://download.cnet.com/USB-Image-Tool/3001-2242_4-75449768.html

[8] Osboxes, «Ubuntu 12.04» [En Linea] Available: http://www.osboxes.org/ubuntu/