



PROGRAMA UNIVERSITARIO DE IMAGINATION

Práctica 6 RVfpga

Introducción a Entrada/Salida

1. INTRODUCCIÓN

En las Prácticas 6-10, el usuario aprenderá a utilizar y ampliar el sistema de entrada/salida (I/O) de RVfpga para permitir que el procesador RISC-V interactúe con los dispositivos periféricos. A continuación, se presenta una visión general de los temas tratados en estas prácticas:

- **Práctica 6:** Uso de los pines de la entrada/salida de propósito general (GPIO) conectados a los LEDs, interruptores y pulsadores de la placa Nexys A7
- **Práctica 7:** Uso de los displays de 7 segmentos disponibles en la placa
- **Práctica 8:** Uso de temporizadores
- **Práctica 9:** Uso de las interrupciones para interactuar con dispositivos externos
- **Práctica 10:** Interconexión del Sistema RVfpga con el acelerómetro SPI integrado.

En esta práctica, en primer lugar, se describen las principales características de un sistema de Entrada/Salida de propósito general y del utilizado en el Sistema RVfpga (Sección 2). A continuación, se describe una versión teórica simplificada de un controlador GPIO genérico (Sección 3). Finalmente, nos centraremos en el controlador GPIO utilizado en SweRVolfX: primero se analiza su especificación de alto nivel y se presentan ejercicios básicos (Secciones 4 y 5), para luego finalizar la práctica analizando su implementación de bajo nivel, simulando RVfpgaSim en Verilator y presentando ejercicios avanzados (Secciones 6 y 7).

Se utiliza esta misma estructura general en las prácticas 7-10. En las secciones iniciales se describe la especificación de alto nivel del controlador de E/S (sus características principales, registros y su funcionamiento, y el mapa de memoria) y a continuación se presentan ejercicios básicos para practicar el uso del periférico. En las secciones finales se describe la implementación de bajo nivel del controlador y se proporcionan ejercicios para modificarlo y posteriormente escribir programas que prueben la modificación.

Nota para los docentes: pueden elegir la complejidad de los ejercicios de acuerdo con el nivel de su curso. Por ejemplo, en un curso de primer/segundo año (como Fundamentos de Computadores u Organización de Computadores), los ejercicios básicos - en esta práctica, la Sección 5 - serían adecuados. Sin embargo, en un curso más avanzado (como Arquitectura de Computadoras o Diseño de Sistemas Incrustados), podrían utilizarse tanto los ejercicios básicos como los avanzados - en esta práctica, las secciones 5 y 7 -.

2. ARQUITECTURA DE ENTRADA/SALIDA

La Figura 1 ilustra la estructura de la Arquitectura von Neumann, que se compone de tres bloques principales: la CPU, la Memoria y el Sistema de Entrada/Salida. En las prácticas 6-10 nos centramos en la interacción de la CPU con los dispositivos de Entradas/Salida. Los dispositivos de Entrada/Salida también se denominan periféricos o simplemente dispositivos. A continuación, se analiza el papel de cada unidad principal:

- **CPU:** la CPU es el iniciador de todas las operaciones de Entrada/Salida. Es el *controlador* (históricamente llamado "maestro", pero ese término está obsoleto) de cualquier transacción de Entrada/Salida. Un controlador de acceso directo a memoria (DMA o DMAC por sus siglas en inglés) también podría actuar como controlador, pero no se incluye en esta práctica.
- **Controlador del dispositivo:** El *controlador del dispositivo* espera por las solicitudes de lectura/escritura de un *controlador* para realizar alguna acción. Los

controladores de dispositivos se comportan como *periféricos* (antes llamados "esclavos", pero ese término está obsoleto) en el sistema de Entrada/Salida. Conceptualmente, un controlador de dispositivos consiste en una serie de *registros* accesibles desde el *controlador*. Los valores de estos registros indican al *periférico* qué acción realizar.

- **La interconexión** (bus, crossbar, etc.) establece un camino entre el *controlador* y los *periféricos*, y se suele implementar con varias capas conectadas a través de un *punto de conexión* (bridge) que evita que ciertos dispositivos ralenticen todo el sistema.

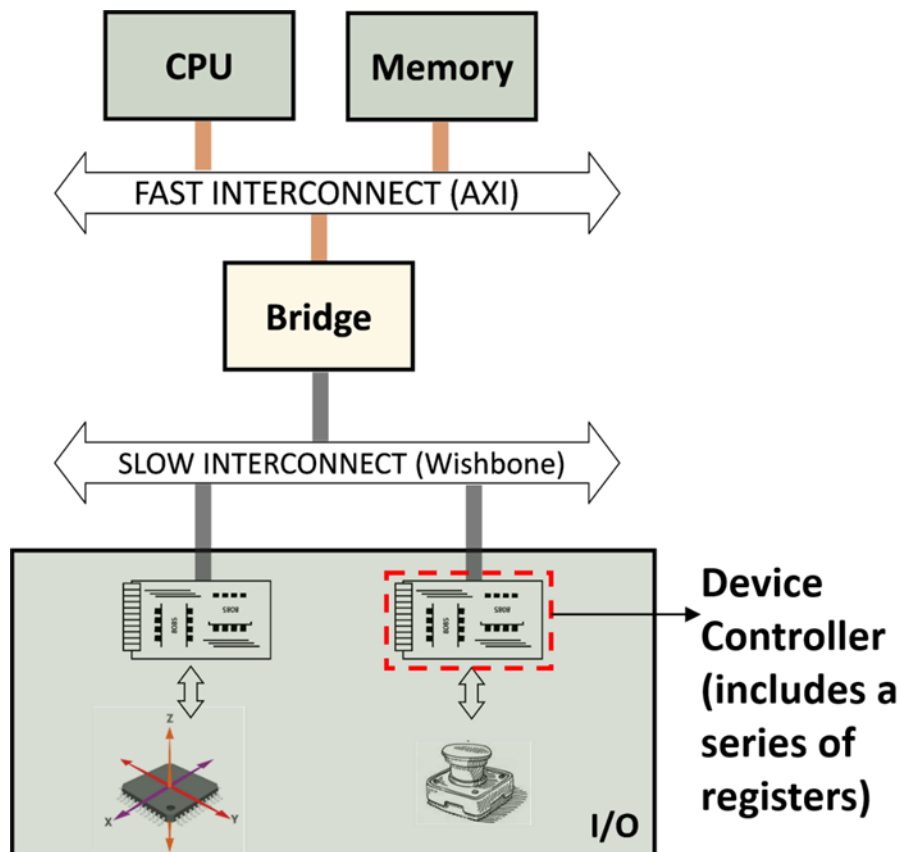


Figura 1. Sistema computador genérico

La Figura 2 muestra el sistema de Entrada/Salida de RVfpga, que incluye los siguientes siete periféricos:

- LEDs e interruptores (considerados como un único periférico), conectados al módulo GPIO1
- Displays de 7 segmentos, conectados al módulo Controlador del Sistema
- Memoria Flash, conectada al módulo SPI1
- Acelerómetro, conectado al módulo SPI2
- Temporizador
- UART
- ROM de arranque

Un multiplexor selecciona un periférico entre las siete posibilidades y lo conecta con la CPU. Observe que es necesaria una interfaz de Wishbone a AXI porque los periféricos utilizan un bus Wishbone (color gris) mientras que el core SweRV EH1 utiliza un bus AXI (color naranja).

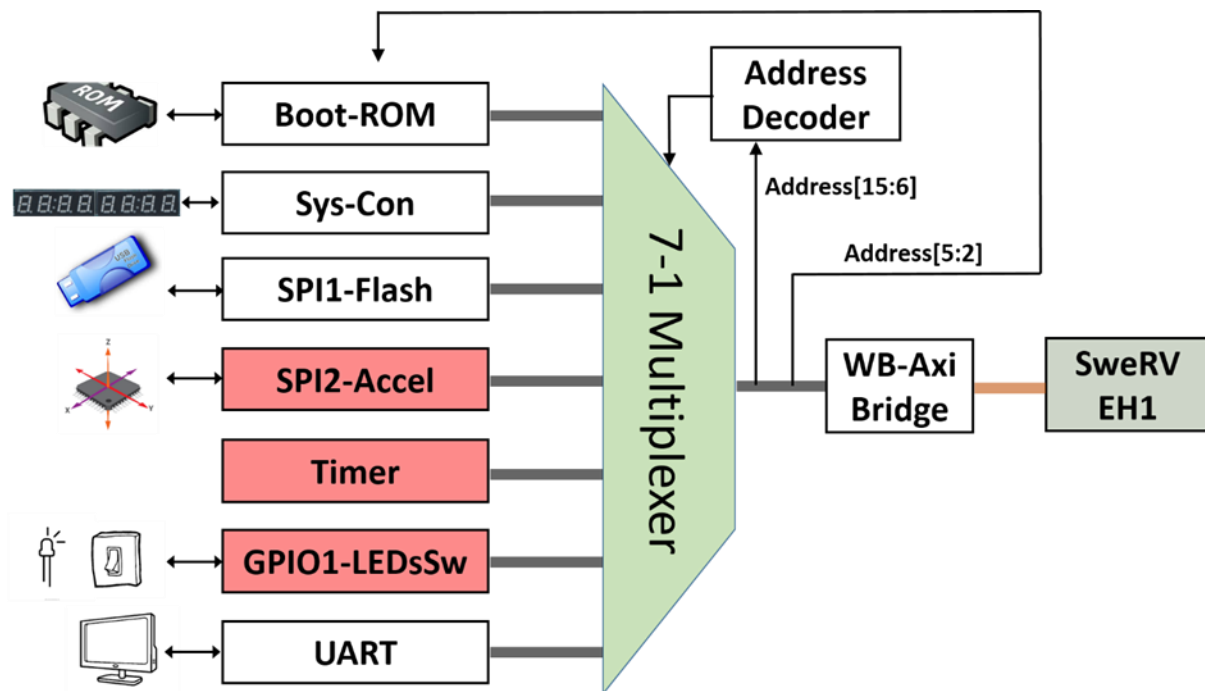


Figura 2. Sistema de Entrada/Salida en el Sistema RVfpga

TAREA: Localizar cada uno de los elementos de la Figura 2 en el SoC. Necesitará examinar los siguientes archivos y directorios:

- [RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v (archivo principal, donde se instancian los elementos de la Figura 2).
- [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals
- [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect
- [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v
- [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v
- [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh

Como se describe en la Guía de Inicio de RVfpga, el SweRVolf original (<https://github.com/chipsalliance/Cores-SweRVolf>) incluye sólo algunos de los periféricos mostrados en la Figura 2: en concreto la ROM de arranque, el controlador del sistema (sin display de 7 segmentos), la memoria flash SPI y la UART (todos ellos en blanco en la Figura 2). Recuerde de la GSG que SweRVolfX amplía el SoC original de SweRVolf con nuevos periféricos: un acelerómetro SPI, un temporizador, un módulo GPIO (en rojo en la Figura 2) y un controlador de display de 7 segmentos (que amplía el actual controlador de sistema de SweRVolf).

Cada periférico recibe valores del procesador y/o envía valores de vuelta al procesador. Se reservan direcciones de memoria para los valores de Entrada/Salida, que se denominan *registros*, *registros de Entrada/Salida mapeados en memoria* o *registros de controlador del dispositivo*. Para enviar un valor a un periférico, la CPU almacena un valor en una dirección de memoria especificada (es decir, un registro mapeado en memoria). Para leer un valor de un periférico, la CPU lee un valor de una dirección de memoria especificada. Así, una simple operación de *load/store* desde la CPU puede configurar un dispositivo, comprobar su estado o leer/escribir datos desde o en él.

El multiplexor de la Figura 2 selecciona el controlador de dispositivo solicitado usando para ello Address[15:6]. Los controladores de dispositivo utilizan Address[5:2] para seleccionar entre varios registros utilizados para controlar el dispositivo.

3. ENTRADA/SALIDA DE PROPÓSITO GENERAL (GPIO)

Un controlador de Entrada/Salida de propósito general (GPIO por sus siglas en inglés) expone al programador pines digitales externos. En cualquier momento del programa, esos pines pueden ser configurados como entradas o salidas. Esa designación es por cada pin y puede cambiar a lo largo del programa, si así se desea. Los pines GPIO pueden conectarse a dispositivos externos como LEDs, interruptores o pulsadores.

Figura 3 muestra un diagrama simplificado de un módulo GPIO genérico que conecta un pin externo a la CPU. El pin se puede conectar a cualquier dispositivo de Entrada/Salida, como un LED, un interruptor, etc. El pin se conecta a un búfer tri-estado, resaltado en verde en la figura. Este búfer permite al programador configurar el pin como una entrada o una salida. Si el búfer tri-estado está activado, el pin actúa como una salida (por ejemplo, para accionar un LED). Si el búfer tri-estado está desactivado, el pin actúa como una entrada (por ejemplo, para leer los valores de los interruptores).

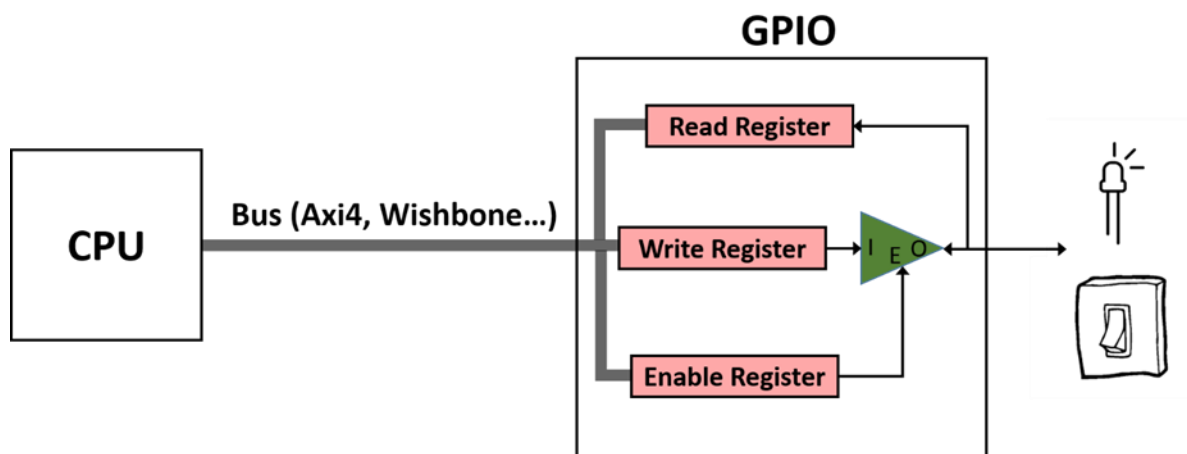


Figura 3. Circuito simplificado de GPIO

Un búfer tri-estado puede actuar como un búfer regular (cuando está activado) o tener una salida flotante (cuando está desactivado). El búfer tri-estado tiene dos entradas, E (enable) e I (input), y una salida, O, y su tabla de verdad se muestra en la Tabla 1. Cuando E es 1, el búfer tri-estado actúa como un búfer regular con la salida (O) y la entrada (I) conectadas. Cuando E es 0, no existe conexión entre la entrada y la salida, y esta última (O) queda en estado de alta impedancia (es flotante). En la Figura 3, para configurar un pin como salida, E=1, lo que permite a la CPU accionar el pin. Cuando se configura un pin como entrada, E=0, lo que impide que la CPU gestione el pin y permite que sea el periférico quien lo haga.

Tabla 1. Tabla de verdad del búfer tri-estado

E	I	O
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

El Sistema RVfpga utiliza Entrada/Salida mapeada en memoria para leer/escribir los valores almacenados en estos registros. Por ejemplo, suponga que el pin de la Figura 3 está conectado a un interruptor y que los tres registros en la GPIO están mapeados de la siguiente manera:

- Read Register = Dirección 0x80001400
- Write Register = Dirección 0x80001404
- Enable Register = Dirección 0x80001408

Para leer el estado del interruptor, se hace lo siguiente:

1. Configurar el pin como una entrada escribiendo un 0 en el *Read Register* (es decir, ejecutando un *store* de 0 a la dirección 0x80001408).
2. Leer el *Read Register* ejecutando una instrucción de *load* de la dirección 0x80001400.

4. ESPECIFICACIÓN DE ALTO NIVEL DE LA GPIO

En esta sección, en primer lugar, se analizará la especificación de alto nivel de la GPIO de SweRVolfX y a continuación se propondrá un ejercicio que utiliza este periférico.

A. Especificación de alto nivel de la GPIO

El módulo GPIO utilizado en SweRVolf es de OpenCores (<https://opencores.org/projects/gpio>). El documento `gpio_spec.pdf` proporcionado con la descarga del módulo GPIO de OpenCores describe la especificación en alto nivel del módulo, y está disponible aquí:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/gpio/docs/gpio_spec.pdf. En esta práctica se resume el funcionamiento básico y las características principales del módulo GPIO. Sin embargo, puede obtener las especificaciones completas en *gpio_spec.pdf*.

El módulo GPIO tiene las siguientes características principales:

- Utiliza una interconexión Wishbone.
- Funciona sólo como un dispositivo periférico.
- El usuario puede usar 1-32 pines GPIO.
- Se pueden usar múltiples módulos GPIO (también llamados cores GPIO) en paralelo para acceder a más de 32 pines GPIO.
- Todos los pines GPIO pueden ser:
 - bidireccionales (en este caso se requieren módulos externos de Entrada/Salida bidireccionales).
 - tri-estado o de drenaje abierto (en este caso se requieren módulos externos de Entrada/Salida tri-estado o de drenaje abierto).
- Los pines GPIO que se programan como entradas:
 - pueden ser registrados.
 - pueden provocar una solicitud de interrupción a la CPU.

La Sección 4 de la especificación del core GPIO describe los registros de control y estado disponibles dentro del módulo GPIO. Cada uno de estos registros se asigna a una dirección diferente como se muestra en la Tabla 2. La dirección base de los registros GPIO es **0x80001400**.

Tabla 2. Registros GPIO

Nombre	Dirección	Ancho	Acceso	Descripción
--------	-----------	-------	--------	-------------

RGPIO_IN	0x80001400	1-32	R	Datos de entrada de GPIO
RGPIO_OUT	0x80001404	1-32	R/W	Datos de salida de GPIO
RGPIO_OE	0x80001408	1-32	R/W	Habilitar el controlador de salida GPIO
RGPIO_INTE	0x8000140C	1-32	R/W	Habilitar interrupción
RGPIO_PTRIG	0x80001410	1-32	R/W	Tipo de evento que provoca una interrupción
RGPIO_AUX	0x80001414	1-32	R/W	Multiplexar las entradas auxiliares a las salidas GPIO
RGPIO_CTRL	0x80001418	2	R/W	Registro de control
RGPIO_INTS	0x8000141C	1-32	R/W	Estado de interrupción
RGPIO_ECLK	0x80001420	1-32	R/W	Habilitar gpio_eclk para acoplar RGPIO_IN
RGPIO_NEC	0x80001424	1-32	R/W	Seleccionar flanco activo de gpio_eclk

Aunque el módulo GPIO de OpenCores es más complejo que la versión simplificada ilustrada en la Figura 3, aún así se pueden identificar los tres registros de la Figura 3: Read (entrada), Write (salida) y Enable. En el módulo GPIO de OpenCores, estos registros se llaman, respectivamente: RGPIO_IN, RGPIO_OUT y RGPIO_OE y están mapeados a las direcciones 0x80001400, 0x80001404, y 0x80001408 respectivamente.

TAREA: Localizar la declaración de los registros RGPIO_IN, RGPIO_OUT y RGPIO_OE en el módulo GPIO, así como la definición de sus direcciones. El módulo GPIO está aquí: *RVfpgaPath]/RVfpga/src/SweRVoltSoC/Peripherals/gpio/gpio_top.v*.

El registro RGPIO_IN guarda las entradas de propósito general. El registro RGPIO_OUT controla las salidas de propósito general. El RGPIO_OE configura cada pin de Entrada/Salida como una entrada o una salida. Cuando el bit de habilitación (dentro de RGPIO_OE) está activo, se habilita el correspondiente controlador de salida de propósito general, y así el pin puede conectarse a un periférico de salida, como un LED. Cuando el bit de habilitación se pone a cero, el controlador de salida funciona en modo de drenaje abierto, también llamado tri-estado o de alta impedancia, y por lo tanto el pin puede conectarse a un periférico de entrada, como un interruptor o un pulsador.

En RVfpgaNexys, los primeros 16 pines GPIO (pines 15:0) del módulo GPIO están conectados a los 16 LEDs de la placa Nexys A7. Los últimos 16 pines GPIO (pines 31:16) del controlador GPIO están conectados a los 16 interruptores de la placa.

5. EJERCICIOS BÁSICOS

Ejercicio 1. Escriba un programa en lenguaje ensamblador de RISC-V y un programa en C que muestren un bloque de cuatro LEDs encendidos que se mueva repetidamente de un lado a otro de los 16 LEDs disponibles en la placa. Se deben incluir también dos interruptores que controlen la velocidad y la dirección. El Switch[0] cambia la velocidad y el Switch[1] cambia la dirección de la siguiente manera:

- Si el Switch[0] está activado (hacia arriba), los LEDs encendidos deben moverse rápido. En caso contrario, los LEDs iluminados deben moverse lentamente. Puede definir lo que significa "rápido" y "lento", pero cualquiera de las dos velocidades debe ser visible, y se debe ser capaz de detectar una diferencia en la velocidad a simple vista.

- Si el Switch[1] está encendido (hacia arriba), los LEDs encendidos deben moverse continuamente de derecha a izquierda (empiezan de nuevo a la derecha cuando llegan al LED de la izquierda). En caso contrario, los LEDs encendidos deben moverse continuamente de izquierda a derecha.

Figura 4 muestra la placa Nexys A7 con los LEDs e interruptores resaltados.

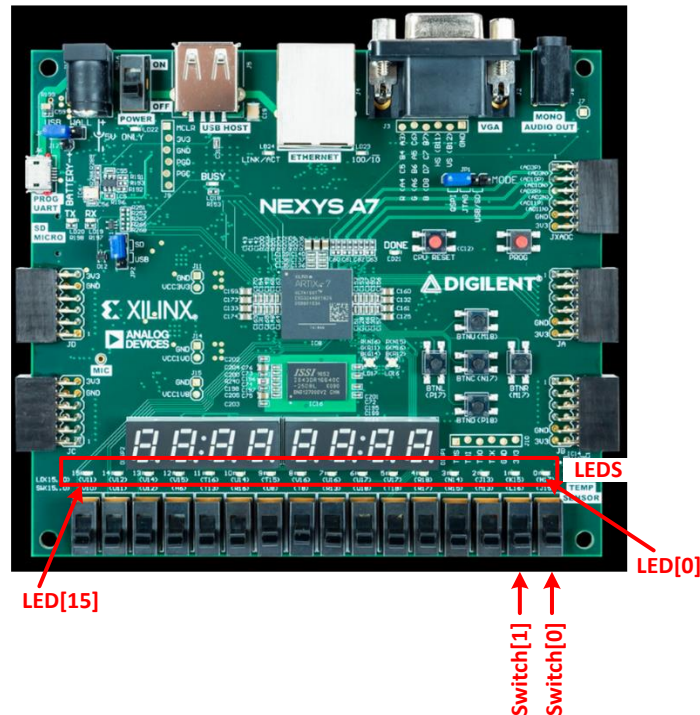


Figura 4. Placa FPGA Nexys A7: LEDs e interruptores

Nota: Recuerde que los interruptores están conectados a los pines 31:16 de los registros de Entrada/Salida mapeados en memoria. Así, para leer el Switch[0], necesitaría escribir 0 en RGPIO_OE[16] y luego leer el valor de RGPIO_IN[16]. Para acceder a los otros LEDs e interruptores será preciso configurar RGPIO_OE adecuadamente.

6. IMPLEMENTACIÓN DE BAJO NIVEL DE GPIO Y SIMULACIÓN

En esta sección se describen en primer lugar los detalles de bajo nivel de la GPIO utilizada en SweRVolfX. Tras ello se modificará RVfpgaSim y se realizará una simulación de ejemplo en Verilator con un sencillo programa en lenguaje ensamblador. Finalmente, se proponen algunos ejercicios en los que primero se simula RVfpgaSim, luego se modifica para añadir un nuevo periférico GPIO y finalmente se escribe un programa que utiliza este nuevo periférico.

A. Implementación de bajo nivel de GPIO

Ahora que ya ha tenido alguna experiencia con el acceso a los pines GPIO usando Entrada/Salida mapeada en memoria, profundizaremos en los detalles de bajo nivel de la GPIO. La GPIO se puede dividir en tres partes principales, como se muestra en la Figura 5: (1) Conexión externa de RVfpgaNexys a los LEDs/interruptores de la placa (región

sombreada a la izquierda en la Figura 5); (2) Integración del módulo GPIO en SweRVolfX (región sombreada en el centro de la Figura 5); (3) Conexión entre la GPIO y el core SweRV EH1 (región sombreada a la derecha en la Figura 5).

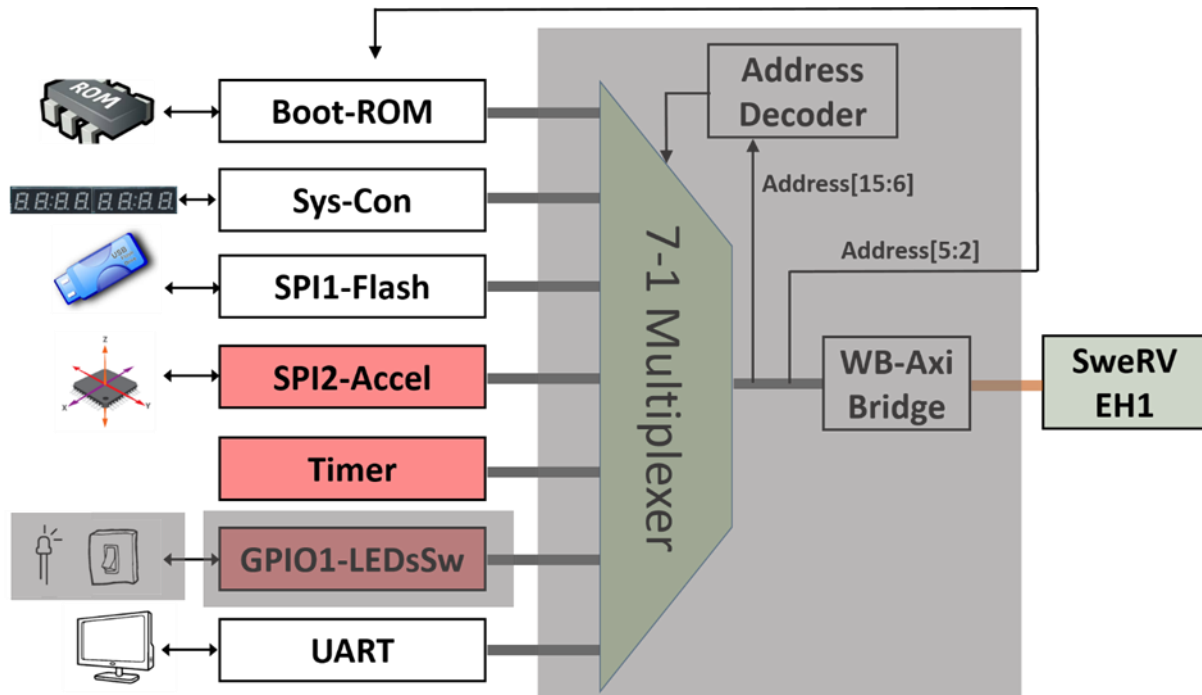


Figura 5. Análisis de GPIO en 3 fases

i. Conexión de los LEDs/interruptores con el SoC

El archivo de restricciones del proyecto (*[RVfpgaPath]/RVfpga/src/rvfpganexys.xdc*) define la conexión entre las señales de entrada/salida del SoC y los dispositivos de la placa. Cada dispositivo de la placa se asocia a un pin determinado de la FPGA. Por ejemplo, Switch[0], el interruptor más a la derecha de la placa, está conectado a través de una pista en la placa de circuito impreso (PCB) al pin J15 de la FPGA.

La placa Nexys A7 incluye 16 LEDs y 16 interruptores. La señal que conecta los 16 LEDs con el módulo superior del SoC (llamado *rvfpganexys*, disponible en el archivo *[RVfpgaPath]/RVfpga/src/rvfpganexys.sv*) se llama *o_led[15:0]*, y la señal que conecta los 16 interruptores con el módulo superior se llama *i_sw[15:0]*. Figura muestra la sección del archivo de restricciones de diseño de Xilinx (xdc), *rvfpganexys.xdc* (disponible en *[RVfpgaPath]/RVfpga/src*) en la que se definen estas 32 conexiones entre señales y pines de la FPGA.

```

26 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[0] }]
27 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[1] }]
28 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[2] }]
29 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[3] }]
30 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { i_sw[4] }]
31 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[5] }]
32 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[6] }]
33 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[7] }]
34 set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[8] }]
35 set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[9] }]
36 set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[10] }]
37 set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[11] }]
38 set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { i_sw[12] }]
39 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { i_sw[13] }]
40 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { i_sw[14] }]
41 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { i_sw[15] }]
42
43 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
44 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { o_led[1] }]
45 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { o_led[2] }]
46 set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { o_led[3] }]
47 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { o_led[4] }]
48 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { o_led[5] }]
49 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { o_led[6] }]
50 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { o_led[7] }]
51 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { o_led[8] }]
52 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { o_led[9] }]
53 set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { o_led[10] }]
54 set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { o_led[11] }]
55 set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { o_led[12] }]
56 set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { o_led[13] }]
57 set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { o_led[14] }]
58 set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { o_led[15] }]

```

Figura 6. Conexión de `i_sw[15:0]` con los interruptores de la placa y `o_led[15:0]` con los LEDs de la placa (archivo `rvfpganexys.xdc`).

Las líneas 48-49 del módulo superior (`rvfpganexys`) muestran estas dos señales conectadas al SoC (parte izquierda de la Figura 7), y el final de ese módulo muestra sus conexiones con el módulo `swervolf_core` (parte derecha de la Figura 7). Obsérvese que las señales `i_sw` y `o_led` se fusionan en la señal `io_data` (línea 257), una señal de entrada/salida de 32 bits conectada con la GPIO en el módulo `swervolf_core` (como se mostrará más adelante, en la Figura 8). Además, tenga en cuenta que la señal `o_led` se maneja a través de una señal intermedia, `gpio_out` (línea 266).

```

25 module rvfpganexys
26     #(parameter bootrom_file = "boot_main.mem")
27     (input wire      clk,
28      input wire      rstn,
29      output wire [12:0] ddram_a,
30      output wire [2:0] ddram_ba,
31      output wire      ddram_ras_n,
32      output wire      ddram_cas_n,
33      output wire      ddram_we_n,
34      output wire      ddram_cs_n,
35      output wire [1:0] ddram_dm,
36      inout wire [15:0] ddram_dq,
37      inout wire [1:0] ddram_dqs_p,
38      inout wire [1:0] ddram_dqs_n,
39      output wire      ddram_clk_p,
40      output wire      ddram_clk_n,
41      output wire      ddram_cke,
42      output wire      ddram_odt,
43      output wire      o_flash_cs_n,
44      output wire      o_flash_mosi,
45      input wire      i_flash_miso,
46      input wire      i_uart_rx,
47      output wire      o_uart_tx,
48      inout wire [15:0] i_sw,
49      output reg [15:0] o_led,

```

```

256 .i ram_init_error (litedram_init_error),
257 .io_data          ({i_sw[15:0], gpio_out[15:0]}),
258 .AN (AN),
259 .Digits Bits ({CA,CB,CC,CD,CE,CF,CG}),
260 .o_accel_sclk      (accel_sclk),
261 .o_accel_cs_n      (o_accel_cs_n),
262 .o_accel_mosi       (o_accel_mosi),
263 .i_accel_miso       (i_accel_miso);
264
265 always @(posedge clk_core) begin
266     o_led[15:0] <= gpio_out[15:0];
267 end
268

```

Figura 7. Conexión de los LED y los interruptores con el módulo superior (`rvfpganexys.sv`)

TAREAS: Siga estas dos señales (`i_sw` y `o_led`) desde el archivo de restricciones hasta el módulo del SoC SweRVolf (donde se fusionan en `io_data`). Necesitará examinar los siguientes archivos:

`[RVfpgaPath]/RVfpga/src/rvfpganexys.xdc`

```
[RVfpgaPath]/RVfpga/src/rvfpganexys.v
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v
```

En la sección anterior se expuso que en RVfpgaNexys los 16 primeros pines GPIO (15 a 0) del módulo GPIO están conectados a los 16 LEDs de la placa, mientras que los 16 últimos pines GPIO (31 a 16) del controlador GPIO están conectados con los 16 interruptores de la placa. ¿Corresponde esto con la implementación descrita en esta sección y en la Figura 8?

ii. Integración del módulo GPIO en el SoC

En las líneas 299-354 del módulo **swervolf_core** (*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v*), se instancia el módulo GPIO y se integra en el SoC (ver Figura 8).

```
299 // GPIO - Leds and Switches
300 wire [31:0] en_gpio;
301 wire      gpio_irq;
302 wire [31:0] i_gpio;
303 wire [31:0] o_gpio;
304
305 bidirec gpio0 (.oe(en_gpio[0]), .inp(o_gpio[0]), .outp(i_gpio[0]), .bidir(io_data[0]));
306 bidirec gpio1 (.oe(en_gpio[1]), .inp(o_gpio[1]), .outp(i_gpio[1]), .bidir(io_data[1]));
307 bidirec gpio2 (.oe(en_gpio[2]), .inp(o_gpio[2]), .outp(i_gpio[2]), .bidir(io_data[2]));
308 bidirec gpio3 (.oe(en_gpio[3]), .inp(o_gpio[3]), .outp(i_gpio[3]), .bidir(io_data[3]));
309 bidirec gpio4 (.oe(en_gpio[4]), .inp(o_gpio[4]), .outp(i_gpio[4]), .bidir(io_data[4]));
310 bidirec gpio5 (.oe(en_gpio[5]), .inp(o_gpio[5]), .outp(i_gpio[5]), .bidir(io_data[5]));
311 bidirec gpio6 (.oe(en_gpio[6]), .inp(o_gpio[6]), .outp(i_gpio[6]), .bidir(io_data[6]));
312 bidirec gpio7 (.oe(en_gpio[7]), .inp(o_gpio[7]), .outp(i_gpio[7]), .bidir(io_data[7]));
313 bidirec gpio8 (.oe(en_gpio[8]), .inp(o_gpio[8]), .outp(i_gpio[8]), .bidir(io_data[8]));
314 bidirec gpio9 (.oe(en_gpio[9]), .inp(o_gpio[9]), .outp(i_gpio[9]), .bidir(io_data[9]));
315 bidirec gpio10 (.oe(en_gpio[10]), .inp(o_gpio[10]), .outp(i_gpio[10]), .bidir(io_data[10]));
316 bidirec gpio11 (.oe(en_gpio[11]), .inp(o_gpio[11]), .outp(i_gpio[11]), .bidir(io_data[11]));
317 bidirec gpio12 (.oe(en_gpio[12]), .inp(o_gpio[12]), .outp(i_gpio[12]), .bidir(io_data[12]));
318 bidirec gpio13 (.oe(en_gpio[13]), .inp(o_gpio[13]), .outp(i_gpio[13]), .bidir(io_data[13]));
319 bidirec gpio14 (.oe(en_gpio[14]), .inp(o_gpio[14]), .outp(i_gpio[14]), .bidir(io_data[14]));
320 bidirec gpio15 (.oe(en_gpio[15]), .inp(o_gpio[15]), .outp(i_gpio[15]), .bidir(io_data[15]));
321 bidirec gpio16 (.oe(en_gpio[16]), .inp(o_gpio[16]), .outp(i_gpio[16]), .bidir(io_data[16]));
322 bidirec gpio17 (.oe(en_gpio[17]), .inp(o_gpio[17]), .outp(i_gpio[17]), .bidir(io_data[17]));
323 bidirec gpio18 (.oe(en_gpio[18]), .inp(o_gpio[18]), .outp(i_gpio[18]), .bidir(io_data[18]));
324 bidirec gpio19 (.oe(en_gpio[19]), .inp(o_gpio[19]), .outp(i_gpio[19]), .bidir(io_data[19]));
325 bidirec gpio20 (.oe(en_gpio[20]), .inp(o_gpio[20]), .outp(i_gpio[20]), .bidir(io_data[20]));
326 bidirec gpio21 (.oe(en_gpio[21]), .inp(o_gpio[21]), .outp(i_gpio[21]), .bidir(io_data[21]));
327 bidirec gpio22 (.oe(en_gpio[22]), .inp(o_gpio[22]), .outp(i_gpio[22]), .bidir(io_data[22]));
328 bidirec gpio23 (.oe(en_gpio[23]), .inp(o_gpio[23]), .outp(i_gpio[23]), .bidir(io_data[23]));
329 bidirec gpio24 (.oe(en_gpio[24]), .inp(o_gpio[24]), .outp(i_gpio[24]), .bidir(io_data[24]));
330 bidirec gpio25 (.oe(en_gpio[25]), .inp(o_gpio[25]), .outp(i_gpio[25]), .bidir(io_data[25]));
331 bidirec gpio26 (.oe(en_gpio[26]), .inp(o_gpio[26]), .outp(i_gpio[26]), .bidir(io_data[26]));
332 bidirec gpio27 (.oe(en_gpio[27]), .inp(o_gpio[27]), .outp(i_gpio[27]), .bidir(io_data[27]));
333 bidirec gpio28 (.oe(en_gpio[28]), .inp(o_gpio[28]), .outp(i_gpio[28]), .bidir(io_data[28]));
334 bidirec gpio29 (.oe(en_gpio[29]), .inp(o_gpio[29]), .outp(i_gpio[29]), .bidir(io_data[29]));
335 bidirec gpio30 (.oe(en_gpio[30]), .inp(o_gpio[30]), .outp(i_gpio[30]), .bidir(io_data[30]));
336 bidirec gpio31 (.oe(en_gpio[31]), .inp(o_gpio[31]), .outp(i_gpio[31]), .bidir(io_data[31]));
337
338 gpio_top gpio_module(
339     .wb_clk_i      (clk),
340     .wb_rst_i      (wb_rst),
341     .wb_cyc_i      (wb_m2s_gpio_cyc),
342     .wb_adr_i      ({2'b0,wb_m2s_gpio_adr[5:2],2'b0}),
343     .wb_dat_i      (wb_m2s_gpio_dat),
344     .wb_sel_i      (4'b1111),
345     .wb_we_i       (wb_m2s_gpio_we),
346     .wb_stb_i      (wb_m2s_gpio_stb),
347     .wb_dat_o      (wb_s2m_gpio_dat),
348     .wb_ack_o      (wb_s2m_gpio_ack),
349     .wb_err_o      (wb_s2m_gpio_err),
350     .wb_inta_o     (gpio_irq),
351     // External GPIO Interface
352     .ext_pad_i     (i_gpio[31:0]),
353     .ext_pad_o     (o_gpio[31:0]),
354     .ext_padoe_o   (en_gpio));
355
```

Figura 8. Integración del módulo GPIO (archivo *swervolf_core.v*).

La interfaz del módulo se puede dividir en dos bloques: señales Wishbone (Tabla 3), que permiten al core SweRV EH1 comunicarse con la GPIO mediante un modelo controlador/periférico, y señales de Entrada/Salida externas (Tabla 4).

Tabla 3. Señales Wishbone

Puerto	Ancho	Dirección	Descripción
wb_cyc_i	1	Entrada	Indica un ciclo de bus válido (selección de core)
wb_adr_i	15	Entradas	Entradas de direcciones
wb_dat_i	32	Entradas	Entradas de datos
wb_dat_o	32	Salidas	Salidas de datos
wb_sel_i	4	Entradas	Indica los bytes válidos en el bus de datos (durante el ciclo válido debe ser 0xf)
wb_ack_o	1	Salida	Salida de <i>acknowledgment</i> (indica la terminación normal de la transacción)
wb_err_o	1	Salida	Salida <i>acknowledgment</i> de error (indica una terminación anormal de la transacción)
wb_rty_o	1	Salida	No se usa
wb_we_i	1	Entrada	Transacción de escritura cuando su valor es 1
wb_stb_i	1	Entrada	Indica un ciclo de transferencia de datos válido
wb_inta_o	1	Salida	Salida de interrupción

Tabla 4. Señales de Entrada/Salida externas

Puerto	Ancho	Dirección	Descripción
en_pad_i	1-32	Entradas	Entradas GPIO
out_pad_o	1-32	Salidas	Salidas GPIO
oen_padoen_o	1-32	Salidas	Habilitación de salidas GPIO (para controladores tri-estado o de drenaje abierto)

Como se muestra en la línea 342 de la Figura 8, los bits 5:2 de la dirección proporcionada por el core en la señal *wb_m2s_gpio_adr[5:2]* del bus Wishbone, se utilizan para seleccionar uno de los 10 registros disponibles mapeados en memoria. Estos cuatro bits se proporcionan al core GPIO a través de la señal *wb_adr_i* (lo que también se muestra en la Figura 8).

La entrada *ext_pad_i* se conecta directamente con el *Read Register* de GPIO (RGPIO_IN). Del mismo modo, la salida *ext_pad_o* conecta directamente con el *Write Register* de GPIO (RGPIO_OUT). Estas dos señales se conectan a los LEDs e interruptores (*i_gpio*, *o_gpio*, *io_data*) a través de 32 módulos de búfer tri-estado (Figura 8, líneas 305-336). De esta manera, los 32 pines se pueden configurar como entradas o salidas. En este caso, los 16 pines inferiores, pines 15:0, están conectados a los LEDs (Figura 7) y por lo tanto deben ser configurados como salidas; los 16 pines superiores, 31:16, están conectados a los interruptores (Figura 7) y por lo tanto deben ser configurados como entradas. Implementamos estos 32 búfers tri-estado incluyendo el siguiente módulo al final del módulo **swervolf_core** (líneas 634-640):

```
module bidirec (input wire oe, input wire inp, output wire outp, inout wire bidir);
    assign bidir = oe ? inp : 1'bZ ;
    assign outp = bidir;
endmodule
```


TAREAS: Los pines GPIO (io_data) están conectados al módulo GPIO a través de búfers tri-estados (ver Figura 8). Analice el búfer tri-estado para los dos posibles estados de la señal de activación ($oe=0$ y $oe=1$).

Teniendo en cuenta la conexión entre el módulo GPIO y los LEDs/interruptores de la placa, ¿qué valores debería asignar el programador a `en_gpio`?

iii. La conexión entre el GPIO y el core SweRV EH1

Como se muestra en la Figura 2, los controladores del dispositivo se conectan al core SweRV EH1 a través de un multiplexor y una interfaz. El multiplexor selecciona uno de los N posibles periféricos (en este caso, $N=7$), dependiendo de la dirección generada por la CPU. La interfaz traduce las señales Wishbone utilizadas por los controladores del dispositivo a señales AXI4 utilizadas por el Core SweRV y viceversa (esta interfaz está implementada en el archivo `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/AxiToWb/axi2wb.v`).

El multiplexor 7:1 (Figura 9) está implementado en el archivo `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v`, y se instancia en las líneas 104-205 del archivo `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh`. Este último archivo está incluido en la línea 168 del módulo `swervolf_core` que se encuentra aquí: `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v`.

```

108 wb_mux
109   #(.num_slaves (7),
110     .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00002000}),
111     .MATCH_MASK ({32'hffffff00, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffff00}))
112   wb_mux_io
113     (.wb_clk_i (wb_clk_i),
114      .wb_rst_i (wb_rst_i),
115      .wbm_adr_i (wb_io_adr_i),
116      .wbm_dat_i (wb_io_dat_i),
117      .wbm_sel_i (wb_io_sel_i),
118      .wbm_we_i (wb_io_we_i),
119      .wbm_cyc_i (wb_io_cyc_i),
120      .wbm_stb_i (wb_io_stb_i),
121      .wbm_cti_i (wb_io_cti_i),
122      .wbm_bte_i (wb_io_bte_i),
123      .wbm_dat_o (wb_io_dat_o),
124      .wbm_ack_o (wb_io_ack_o),
125      .wbm_err_o (wb_io_err_o),
126      .wbm_rty_o (wb_io_rty_o),
127      .wbs_adr_o ({wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o}),
128      .wbs_dat_o ({wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o}),
129      .wbs_sel_o ({wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o}),
130      .wbs_we_o ({wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_uart_we_o}),
131      .wbs_cyc_o ({wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o}),
132      .wbs_stb_o ({wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o}),
133      .wbs_cti_o ({wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o}),
134      .wbs_bte_o ({wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o}),
135      .wbs_dat_i ({wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i}),
136      .wbs_ack_i ({wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i}),
137      .wbs_err_i ({wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i}),
138      .wbs_rty_i ({wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i})),
139   endmodule

```

CPU/Controller Signals

Peripheral Signals

Figura 9. El multiplexor 7-1 selecciona el periférico que se conectará a la CPU (`wb_intercon.v`).

El multiplexor selecciona qué periférico leer o escribir, conectando la CPU (señales `wb_io_*` - líneas 115-126 de la Figura 9) con el Bus Wishbone de un periférico (líneas 127-138 de la Figura 9), dependiendo de la dirección (líneas 110-111). Por ejemplo, si la dirección generada por la CPU está en el rango `0x80001400-0x8000143F`, se selecciona el periférico GPIO, y las señales `wb_io_*` se conectarán con las señales `wb_gpio_*`.

La Figura 10 muestra la implementación Verilog del multiplexor (disponible en el archivo `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon_1.2.2/wb_mux.v`).

TAREA: Analizar en detalle la implementación del multiplexor. Puede centrarse en las señales relacionadas con la GPIO (*wb_gpio_**). Debe examinar los siguientes archivos:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon_1.2.2/wb_mux.v

La comprensión de esta parte del SoC es importante no sólo para esta práctica sino también para las futuras. La simulación que se realiza en la siguiente sección puede ayudar a comprenderla si se amplía la simulación añadiendo nuevas señales relacionadas con el multiplexor.

```

82 //////////////////////////////////////////////////
83 // Master/slave connection
84 //////////////////////////////////////////////////
85
86 //Use parameter instead of localparam to work around a bug in Xilinx ISE
87 parameter slave_sel_bits = num_slaves > 1 ? $clog2(num_slaves) : 1;
88
89 reg wbm_err;
90 wire [slave_sel_bits-1:0] slave_sel;
91 wire [num_slaves-1:0] match;
92
93 genvar idx;
94
95 generate
96   for(idx=0; idx<num_slaves ; idx=idx+1) begin : addr_match
97     assign match[idx] = (wbm_adr_i & MATCH_MASK[idx*aw+:aw]) == MATCH_ADDR[idx*aw+:aw];
98   end
99 endgenerate
100
101 //
102 // Find First 1 - Start from MSB and count downwards, returns 0 when no bit set
103 //
104 function [slave_sel_bits-1:0] ffl;
105   input [num_slaves-1:0] in;
106   integer i;
107
108   begin
109     ffl = 0;
110     for (i = num_slaves-1; i >= 0; i=i-1) begin
111       if (in[i])
112         ffl = i;
113     end
114   end
115 endfunction
116
117 assign slave_sel = ffl(match);
118
119 always @(posedge wb_clk_i)
120   wbm_err <= wbm_cyc_i & !(match);
121
122 assign wbs_adr_o = {num_slaves{wbm_adr_i}};
123 assign wbs_dat_o = {num_slaves{wbm_dat_i}};
124 assign wbs_sel_o = {num_slaves{wbm_sel_i}};
125 assign wbs_we_o = {num_slaves{wbm_we_i}};
126
127 /* verilator lint_off WIDTH */
128 assign wbs_cyc_o = match & (wbm_cyc_i << slave_sel);
129 /* verilator lint_on WIDTH */
130 assign wbs_stb_o = {num_slaves{wbm_stb_i}};
131
132 assign wbs_cti_o = {num_slaves{wbm_cti_i}};
133 assign wbs_bte_o = {num_slaves{wbm_bte_i}};
134
135 assign wbm_dat_o = wbs_dat_i[slave_sel*dw+:dw];
136 assign wbm_ack_o = wbs_ack_i[slave_sel];
137 assign wbm_err_o = wbs_err_i[slave_sel] | wbm_err;
138 assign wbm_rty_o = wbs_rty_i[slave_sel];
139
140
141
142 endmodule

```

Figura 10. Multiplexor Wishbone (archivo *wb_mux.v*).

B. Simulación en Verilator

En esta sección, en primer lugar, se modifica el simulador RVfpgaSim mediante el añadido de una nueva señal de entrada. A continuación, se recompila RVfpgaSim usando Verilator y

se analiza esta nueva señal al ejecutar el simulador un sencillo programa.

i. Modificar y recompilar RVfpgaSim

En la simulación no se dispone de LEDs o interruptores reales. Así, en el programa de pruebas o *testbench* ([RVfpgaPath]/RVfpga/src/rvfpgasim.v), se simula el accionamiento de los interruptores asignando a esa señal (*i_sw*) un valor constante de 0xFE34 (parte izquierda de la Figura 11). Los interruptores se exponen entonces como una entrada al core SweRVolfX (parte derecha de la Figura 11).

```
80    wire [15:0] i_sw;
81    assign i_sw = 16'hFE34; 248    .io_data      ({i_sw,16'bz});
```

Figura 11. Señal *i_sw*, que se asigna y pasa al core SweRVolfX en *rvfpgasim.v*.

Recuerde de la Guía de inicio que el *testbench* (*rvfpgasim.v*) recibe las señales de entrada (*clk*, *rst*, etc.) para RVfpgaSim (parte izquierda de la Figura 12) e instancia el módulo **swervolf_core** (parte derecha de la Figura 12).

```
28    (input wire clk,
29    input wire rst,
30    input wire i_jtag_tck,
31    input wire i_jtag_tms,
32    input wire i_jtag_tdi,
33    input wire i_jtag_trst_n,
34    output wire o_jtag_tdo,
35    output wire o_uart_tx,
36    output wire o_gpio)

190    swervolf_core
191    #(.bootrom_file (bootrom_file))
192    swervolf
193    (.clk (clk),
194    .rstn (!rst),
195    .dmi_reg_rdata (dmi_reg_rdata),
```

Figura 12. Señales de entrada para la instanciar RVfpgaSim y SweRVolfX (archivo *rvfpgasim.v*).

En algunas situaciones puede desear añadir una nueva señal de entrada/salida al simulador. Como ejemplo, a continuación se explica cómo incluir una señal de entrada a RVfpgaSim, llamada *i_sw0*, que proporciona un valor al el interruptor de la derecha.

Siga los siguientes pasos:

1. Modifique el archivo [RVfpgaPath]/RVfpga/src/rvfpgasim.v:

a. Incluya una nueva señal de entrada de 1 bit llamada *i_sw0* (ver Figura 13).

```
28    (input wire clk,
29    input wire rst,
30    input wire i_jtag_tck,
31    input wire i_jtag_tms,
32    input wire i_jtag_tdi,
33    input wire i_jtag_trst_n,
34    output wire o_jtag_tdo,
35    output wire o_uart_tx,
36    output wire o_gpio,
37    input wire i_sw0)
```

Figura 13. Nueva señal de entrada *i_sw0*.

b. Defina esta señal como el interruptor de más a la derecha. Asigne los valores 0xFE34 a los interruptores restantes - excepto el bit 0 - como previamente).

Véase la Figura 14.

```

80
81     wire [15:0] i_sw;
82     // assign i_sw = 16'hFE34;
83     assign i_sw = {15'b111111100011010,i_sw0};
84

```

Figura 14. Definición de i_sw0 como el interruptor de más a la derecha.

2. Modifique el archivo `[RVfpgaPath]/RVfpga/verilatorSIM/tb.cpp`: este es el archivo principal de C++ para Verilator. Al final de este archivo se encuentra un bucle `while` (que se muestra en la Figura 15) en el que cada iteración constituye un pulso de reloj. Obsérvese que la señal de reloj para el SoC se genera dentro de este bucle (línea 178), invirtiendo su valor binario en cada iteración ($1 \rightarrow 0$ ó $0 \rightarrow 1$). Además, el tiempo de simulación se calcula en la variable `main_time` (línea 180) y se mide en nanosegundos (el ciclo de reloj es de 20 ns y por tanto el pulso de reloj de 10 ns). Por último, tenga en cuenta que la simulación finaliza cuando el tiempo de simulación alcanza el valor `timeout` (líneas 173-176).

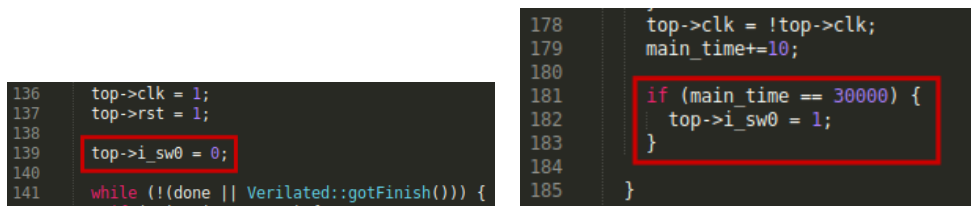
```

136     top->clk = 1;
137     top->rst = 1;
138     while (!done || Verilated::gotFinish()) {
139         if (main_time == 100) {
140             printf("Releasing reset\n");
141             top->rst = 0;
142         }
143         if (main_time == 200)
144             top->i_jtag_trst_n = true;
145
146         top->eval();
147         if (tftp)
148             tftp->dump(main_time);
149         if (baud_rate) do_uart(&uart_context, top->o_uart_tx);
150         if (jtag && (main_time > 300)) {
151             int ret = jtag->doJTAG(main_time/20, //doJtag requires t to only increment by one
152                 &top->i_jtag_tms,
153                 &top->i_jtag_tdi,
154                 &top->i_jtag_tck,
155                 top->o_jtag_tdo);
156             if (ret != VerilatorJtagServer::SUCCESS) {
157                 if (ret == VerilatorJtagServer::CLIENT_DISCONNECTED) {
158                     printf("Ending simulation. Reason: jtag_vpi client disconnected.\n");
159                     done = true;
160                 }
161                 else {
162                     printf("Ending simulation. Reason: jtag_vpi error encountered.\n");
163                     done = true;
164                 }
165             }
166         }
167         if (gpio0 != top->o_gpio) {
168             printf("%lu: gpio0 is %s\n", main_time, top->o_gpio ? "on" : "off");
169             gpio0 = top->o_gpio;
170         }
171         if (timeout && (main_time >= timeout)) {
172             printf("Timeout: Exiting at time %lu\n", main_time);
173             done = true;
174         }
175         top->clk = !top->clk;
176         main_time+=10;
177     }

```

Figura 15. Bucle while para la simulación.

Asigne un valor binario de **0** a la nueva señal `i_sw0` antes de entrar en el bucle (parte izquierda de la Figura 16), y cámbielo a **1** dentro del bucle en el instante de tiempo **30 us** (ver la parte derecha de la Figura 16).



```

136 top->clk = 1;
137 top->rst = 1;
138
139 top->i_sw0 = 0;
140
141 while (!(done || Verilated::gotFinish())) {
142
178 top->clk = !top->clk;
179 main_time+=10;
180
181 if (main_time == 30000) {
182     top->i_sw0 = 1;
183 }
184
185 }

```

Figura 16. Asignación de valores a la señal `i_sw0`.

- Una vez que haya realizado todos estos cambios, recompile RVfpgaSim ejecutando los siguientes comandos (como se explicó en la Guía de Inicio):

```

cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean
make

```

Se debe generar un nuevo archivo *Vrvfpgasim* (el binario de simulación de RVfpgaSim), dentro del directorio *[RVfpgaPath]/RVfpga/verilatorSIM*.

WINDOWS: Tiene que hacer este último paso (paso 3) en la terminal Cygwin (consulte la Sección 6 y el apéndice C de la Guía de Inicio para instrucciones detalladas). Tenga en cuenta que la carpeta *C:\Windows* se puede encontrar dentro de Cygwin en: */cygdrive/c*.

MacOS: Consulte el Apéndice D de la Guía de Inicio para obtener instrucciones detalladas.

ii. Analizar la simulación del programa *LedsSwitches.S*

En esta sección se simulará el programa de ejemplo *LedsSwitches.S* (Figura 17) de la Guía de inicio de RVfpga. Este programa lee los valores de los interruptores y escribe esos valores en los LEDs de la placa Nexys A7. Tenga en cuenta que es necesario configurar el registro de habilitación (*enable register*), de modo que los 32 pines de entrada/salida se configuren como entradas o salidas, según sus conexiones. En particular, los 16 pines inferiores de la GPIO están conectados a los LEDs, por lo que son pines de salida con respecto a la CPU (Enable=1). Los 16 pines superiores de la GPIO están conectados a los interruptores, que son pines de entrada con respecto a la CPU (Enable=0). Dado que los interruptores ocupan los 16 bits superiores del registro de lectura (*read register*), se deben desplazar a la derecha antes de escribir su valor en los LEDs.

```

#define GPIO_SWs      0x80001400
#define GPIO_LEDs     0x80001404
#define GPIO_INOUT    0x80001408

.globl main
main:

li x28, 0xFFFF
li x29, GPIO_INOUT
sw x28, 0(x29)                # Write the Enable Register

next:
    li a1, GPIO_SWs          # Read the Switches
    lw t0, 0(a1)

    li a0, GPIO_LEDs
    srl t0, t0, 16

```

```
sw t0, 0(a0)           # Write the LEDs

beq zero, zero, next
.end
```

Figura 17. LedsSwitches.s para ejecutar en el Core SweRVolfX

Siga los siguientes pasos para ejecutar la simulación.

1. Abra VSCode/PlataformalO en su computadora.
2. En la barra superior, haga clic en *File*→*Open folder...* (Figura 18), y navegue hasta la carpeta *[RVfpgaPath]/RVfpga/examples/*

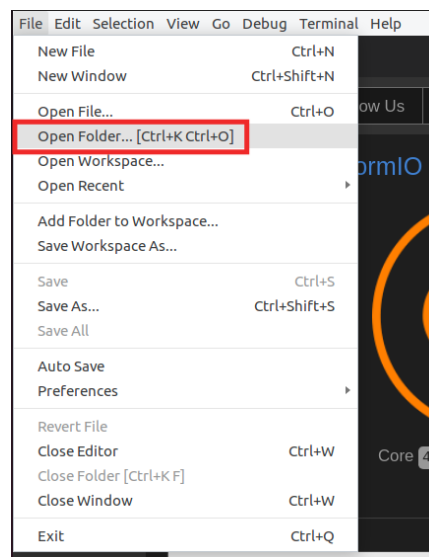


Figura 18. Abra el ejemplo LedsSwitches.S

3. Seleccione el directorio *LedsSwitches* (no lo abra, sólo selecciónelo) y haga clic en OK. El ejemplo se abrirá en PlatformIO.
4. Abra el archivo *platformio.ini* y compruebe si la ruta de acceso al binario de simulación RVfpgaSim (Figura 19) generado anteriormente (paso 3 de la sección anterior) es correcta. Recuerde que conforme a la Guía de Inicio la ruta debe ser:

```
board_debug.verilator.binary =
[RVfpgaPath]/RVfpga/verilatorSIM/Vrvfpgasim
```

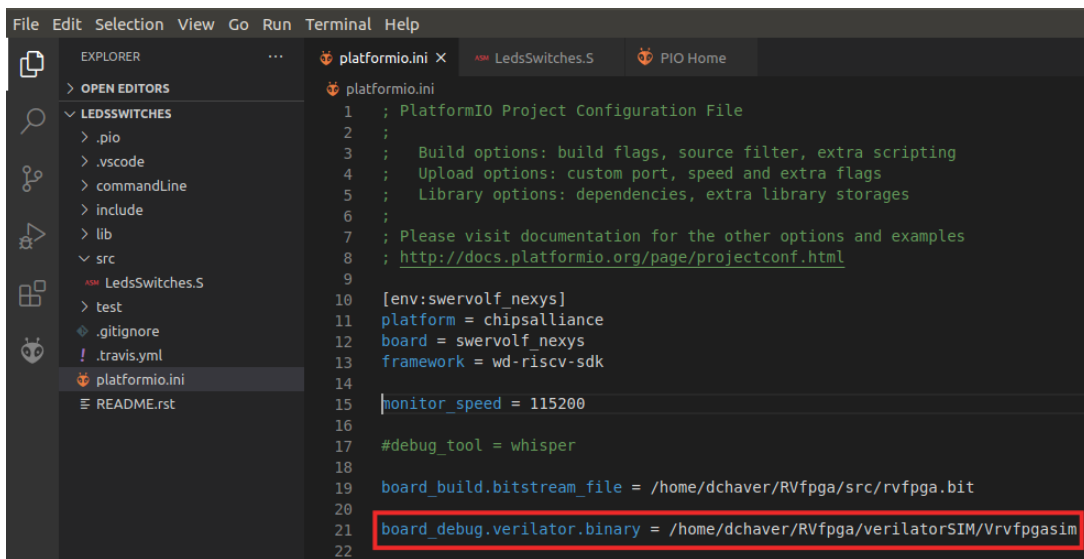



Figura 19. Archivo de inicialización de Platformio: platformio.ini

Windows: El ejecutable de la simulación RVfpgaSim se llama Vrvfpgasim.exe. Por lo tanto:

```
board_debug.verilator.binary = [RVfpgaPath]\RVfpga\verilatorSIM\NVrvfpgasim.exe
```

5. Ejecute la simulación haciendo clic en el icono de PlatformIO en la barra del menú de la izquierda , luego expanda las Project Tasks → env:swervolf_nexys → Platform y haga clic en Generate Trace.

El archivo trace.vcd se debe generar dentro de `[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys`, y lo puede abrir con *GTKWave* escribiendo el siguiente comando en la terminal de PlatformIO.

```
gtkwave [RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/trace.vcd
```

WINDOWS: la carpeta *gtkwave64* que descargó incluye una aplicación llamada *gtkwave.exe* dentro de la carpeta *bin*. Ejecute GTKWave haciendo doble clic en esa aplicación. En la parte superior de la aplicación, haga clic en **File – Open New Tab**, y abra el archivo trace.vcd generado en la carpeta `[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys`.

6. Incluya en la visualización las siguientes señales (entre en el módulo *rvfpgasim-swervolf* para encontrar cada una de estas señales):
 - Añada la señal de reloj: **clk**
 - Añada la señal de entrada GPIO: **i_gpio**
 - Añada la señal de salida GPIO: **o_gpio**

En la gráfica (Figura 20), verá que el valor de los 16 interruptores (16 bits más significativos de la señal **i_gpio**) se copia en los 16 LEDs (16 bits menos significativos de la señal **o_gpio**) con cierto retraso. Además, el interruptor de más a la derecha cambia (0→ 1) en el instante de tiempo 30us, y esto hace que el LED más a la derecha también cambie algún tiempo después.

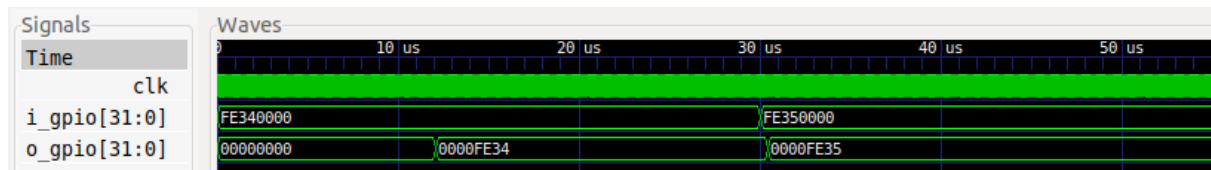


Figura 20. Simulación del programa LedsSwitches

7. EJERCICIOS AVANZADOS

Ejercicio 2. Analice la simulación de la sección anterior con mayor detalle. Figura 21 muestra la versión desensamblada .elf del programa *LedsSwitches* (Figura 17), con las tres instrucciones que acceden a los registros GPIO (*Enable*, *Read* and *Write*) resaltadas. Recuerde de la Guía de Inicio que en PlatformIO puede ver fácilmente la versión desensamblada .elf del programa abriendo el archivo *firmware.dis*, que se genera en tiempo de compilación dentro de la carpeta:

[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf-nexys/ (ver Figura 21).

```

> PSP
> src
  ≡ .sconsign36.dblite
  ≡ firmware.bin
  ≡ firmware.dis
  ≡ firmware.elf
  ≡ LedsSwitches.map
  ≡ libBoardBSP.a
  ≡ libPSP.a
  ≡ project.checksum
> libdeps
> .vscode
> commandLine
> include
> lib
  ≡ src
65 Disassembly of section .text:
66
67 00000090 <main>:
68 90: 00010e37      lui t3,0x10
69 94: fffe0e13      addi t3,t3,-1 # ffff <_sp+0xcebf>
70 98: 80001eb7      lui t4,0x80001
71 9c: 408e8e93      addi t4,t4,1032 # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
72 a0: 01cea023      sw t3,0(t4)
73
74 000000a4 <next>:
75 a4: 800015b7      lui a1,0x80001
76 a8: 40058593      addi a1,a1,1024 # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
77 ac: 0005a283      lw t0,0(a1)
78 b0: 80001537      lui a0,0x80001
79 b4: 40450513      addi a0,a0,1028 # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
80 b8: 0102d293      srli t0,t0,0x10
81 bc: 00552023      sw t0,0(a0)
82 c0: fe0002e3      beqz zero,a4 <next>

```

Figura 21. Versión desensamblada del programa LedsSwitches.S

Simule este programa en RVfpgaSim y analice las señales GPIO durante la ejecución de cada una de las tres instrucciones de acceso a memoria resaltadas en rojo en la Figura 21 (*sw*, *lw* y *sw*). Esto le ayudará a entender la implementación de bajo nivel de GPIO explicada en la Sección A.

Puede comenzar con la simulación de la Sección B y añadir y analizar los valores de las siguientes señales (acceda a los módulos referidos para localizar cada señal):

- rvfpgasim → swervolf → swerv_eh1 → swerv → ifu
 - Reloj: *clk*.
 - Instrucciones buscadas: *ifu_i0_instr* y *ifu_i1_instr*.
- rvfpgasim - swervolf
 - Pines de entrada/salida de 32 bits: *i_gpio* y *o_gpio*.
 - Dirección generada por la CPU: *wb_m2s_io_adr*.
- rvfpgasim - swervolf - gpio_module
 - Interfaz externa de la GPIO: *ext_pad_i*, *ext_pad_o* y *ext_padoe_o*.
- rvfpgasim - swervolf - wb_intercon0

- Dirección de salida y señales de datos para el multiplexor de la Figura 2: **wb_io_adr_i**, **wb_io_dat_i**, **wb_io_dat_o**.
- Señales de datos de la GPIO de entrada para el multiplexor de la Figura 2: **wb_gpio_adr_i**, **wb_gpio_dat_i**, **wb_gpio_dat_o**.
- Señales de selección para el multiplexor de la Figura 2: **wb_*_cyc_o**.
- rvfpgasim - swervolf - wb_intercon0 - wb_mux_io
 - Señal de coincidencia para el multiplexor de la Figura 2: **match**.
- rvfpgasim - swervolf – swerv_eh1 - swerv - dec - arf - gpr_banks(0) - gpr(5) - gprff
 - Valor del registro para t0: **dout**.

Ejercicio 3. Ampliar **RVfpgaNexys** para dar soporte a los cinco pulsadores de la placa. Los pulsadores se muestran en la Figura 22. Los cinco botones se nombran según su ubicación: arriba, abajo, izquierda, derecha y centro - BTNU, BTND, BTNL, BTNR, BTNC.

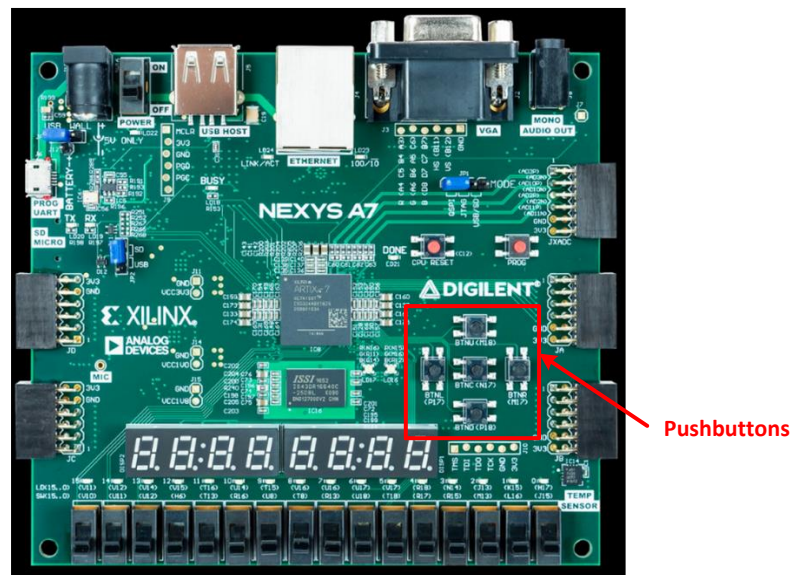


Figura 22. Pulsadores de la placa FPGA Nexys A7

- a. Dado que el tamaño máximo del módulo GPIO que se está usando (`gpio_top`) es 32, que es el número de pines de Entrada/Salida del que se dispone (16 LEDs + 16 Switches), es necesario incluir otra instancia del módulo GPIO en SweRVolfX, así como 5 nuevos búfers tri-estado y todas las señales necesarias.
- b. Utilice las direcciones a partir de 0x80001800 (que están disponibles) para mapear los registros expuestos por el nuevo controlador GPIO. Tenga en cuenta que debe modificar el multiplexor (Figura 7) para incluir el nuevo periférico.
- c. También debe modificar el archivo de restricciones teniendo en cuenta que los cinco pulsadores están conectados a los siguientes pines de la FPGA:
 - i. BTNC está conectado al PIN N17
 - ii. BTNU está conectado al PIN M18
 - iii. BTNL está conectado al PIN P17
 - iv. BTNR está conectado al PIN M17
 - v. BTND está conectado al PIN P18

Ejercicio 4. Diseñe otro controlador en **RVfpgaNexys** para los cinco pulsadores de la placa.

- a. A diferencia del Ejercicio 3, en este caso usted debe implementar en Verilog o SystemVerilog su propio controlador GPIO basado en el esquema de la Figura 3. De hecho, puede incluso simplificar ese circuito e incluir sólo un **Read Register** (es decir, no es necesario incluir los búfers tri-estado ni el **Write Register**).
- b. No es necesario eliminar el controlador del ejercicio anterior porque los pulsadores pueden ser mapeados a direcciones no utilizadas por ese controlador GPIO.
- c. Incluya el nuevo controlador dentro del periférico Controlador del Sistema. Puede utilizar el rango de direcciones 0x8000101C-0x8000101F, que no están siendo utilizadas. Tenga en cuenta que los registros incluidos en el Controlador del Sistema se leen en la CPU conectándolos directamente a la señal de datos del Bus Wishbone (o_wb_rdt) en base a la dirección (i_wb_adr) generada por la CPU. Examine las líneas 234-266 del módulo **swervolf_syscon** (*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v*) como guía para entender cómo proceder.

Ejercicio 5. Escriba un programa en lenguaje ensamblador de RISC-V y un programa en C que muestre una cuenta ascendente en binario, incrementado cada vez un valor y mostrándolo en los LEDs, comenzando en 1. Incluya un bucle de retardo para esperar entre la visualización de cada valor incrementado, de modo que los valores sean visibles por el ojo humano. Lea BTNC a través del periférico OpenCores implementado en el ejercicio 3 y utilícelo para cambiar la velocidad de la cuenta. Lea BTNU a través del periférico ad-hoc implementado en el ejercicio 4 y utilícelo para reiniciar la cuenta cada vez que se presione.