

Sistemas Digitales

Práctica 4

Escribiendo una Aplicación de Software Básica

(Adaptado del curso “Embedded System Design Flow” de Xilinx)

Introducción

Esta práctica lo guiará a través del proceso de de escribir una aplicación de software básica. El software que desarrollará escribirá sobre los LEDs de la placa. Un controlador AXI BRAM y una BRAM de 8KB asociada fueron agregados en el laboratorio anterior. La aplicación correrá desde la BRAM modificando el linker script para el proyecto para colocar la sección text de la aplicación dicha memoria. Verificará que el diseño opera como se espera que lo haga, probándolo en hardware.

Objetivos

Después de completar esta práctica será capaz de:

- Escribir una aplicación básica para acceder a un periférico IP en el SDK
- Desarrollar un linker script
- Particionar las secciones del ejecutable dentro de los espacios de ambas memorias, la DDR3 y BRAM
- Generar un archivo ejecutable ELF
- Descargar el bitstream y la aplicación y verificar en la placa

Procedimiento

Esta práctica está separada en pasos que consisten en sentencias generales que proveen información sobre las instrucciones detalladas que le siguen. Siga estas instrucciones detalladas para avanzar dentro de esta práctica.

Esta práctica está compuesta por 4 pasos principales: Usted abrirá el proyecto Vivado, exportará el hardware e invocará el SDK, creará un proyecto de software, analizará los archivos objeto ensamblados y verificará el diseño en hardware.

Descripción del Diseño

El diseño fue extendido al final del laboratorio anterior para incluir un controlador de memoria. El bitstream ya fue generado. Será desarrollada una aplicación de software básica para acceder a los LEDs de la placa.

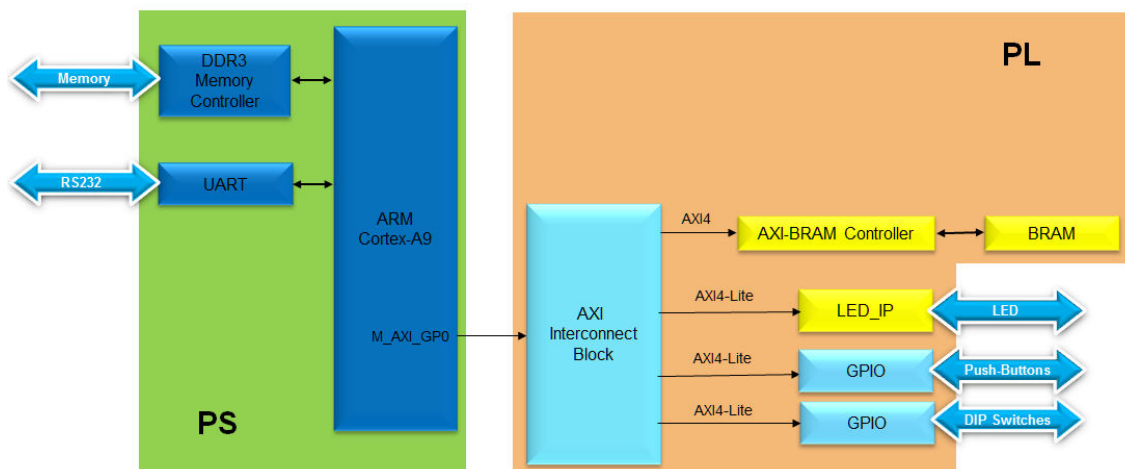
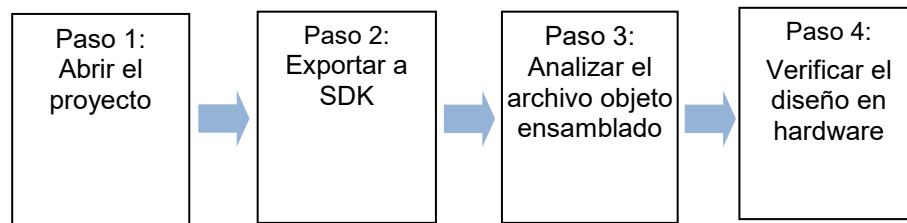


Figura 1. Diseño usado de la práctica anterior

Flujo General para esta práctica



Abrir el proyecto

Paso 1

1-1. Usar el proyecto de la práctica pasada.

1-1-1. Abrir Vivado seleccionando **Start ► Xilinx Design Tools ► Vivado 2018.1** y abrir la práctica 3.

1-1-2. Seleccionar **File ► Project ► Save As ...** para abrir el cuadro de diálogo *Save Project As*. Introduzca **lab4** como el nombre del proyecto. Asegúrese que esté marcada la opción *Create Project Subdirectory*. Esto creará el directorio lab4 y guardará el proyecto y el directorio asociado con este nombre.

Exportar a SDK y crear un Proyecto de Aplicación

Paso 2

2-1. Exportar el hardware al SDK junto con el bitstream generado.

2-1-1. Hacer click en **File ► Export ► Export Hardware**. Hacer click en el casillero *Include the bitstream* y luego hacer click en **Yes** para sobrescribir.

2-1-2. Seleccionar **File ► Launch SDK** y hacer click sobre **OK**.

2-2. Cerrar todos los proyectos previamente creados. Crear un proyecto vacío llamado lab4. Importar el archive lab4.c provisto en el campus

2-2-1. Para limpiar el workspace hacer botón derecho sobre los proyectos **TestApp**, **standalone_bsp_0**, y **system_wrapper_hw_platform_0** pertenecientes a la práctica anterior, y hacer click en **Close Project**, ya que estos proyectos no serán usados en esta práctica. Llegado el caso, si son necesarios en algún momento, se los puede reabrir.

2-2-2. Seleccionar **File ► New ► Application Project**.

2-2-3. Ingresar **lab4** como nombre del proyecto, y para *Board Support Package*, elegir **Create New lab4_bsp** (debería ser la única opción).

SDK New Project

Application Project

Create a managed make application project.

Project name:

☒ Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Hardware Platform:

Processor:

Target Software

Language: ☒ C ☐ C++

Compiler:

Hypervisor Guest:

Board Support Package: ☒ Create New ☐ Use existing

Figura 2. Creando un nuevo proyecto de aplicación

2-2-4. Hacer click en **Next**, y seleccionar *Empty Application* y hacer click en **Finish**.

2-2-5. Expandir **lab4** en la vista de proyecto y hacer botón-derecho en la carpeta *src* y seleccionar **Import**.

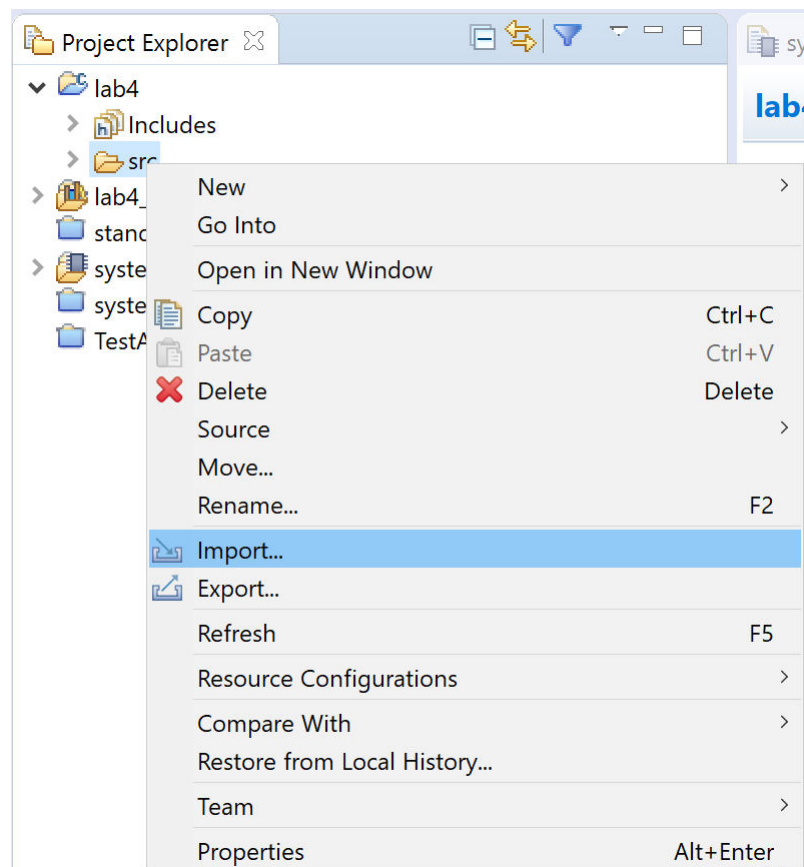


Figura 3. Importando código existente

2-2-6. Expandir la categoría **General** y hacer doble click **File System**.

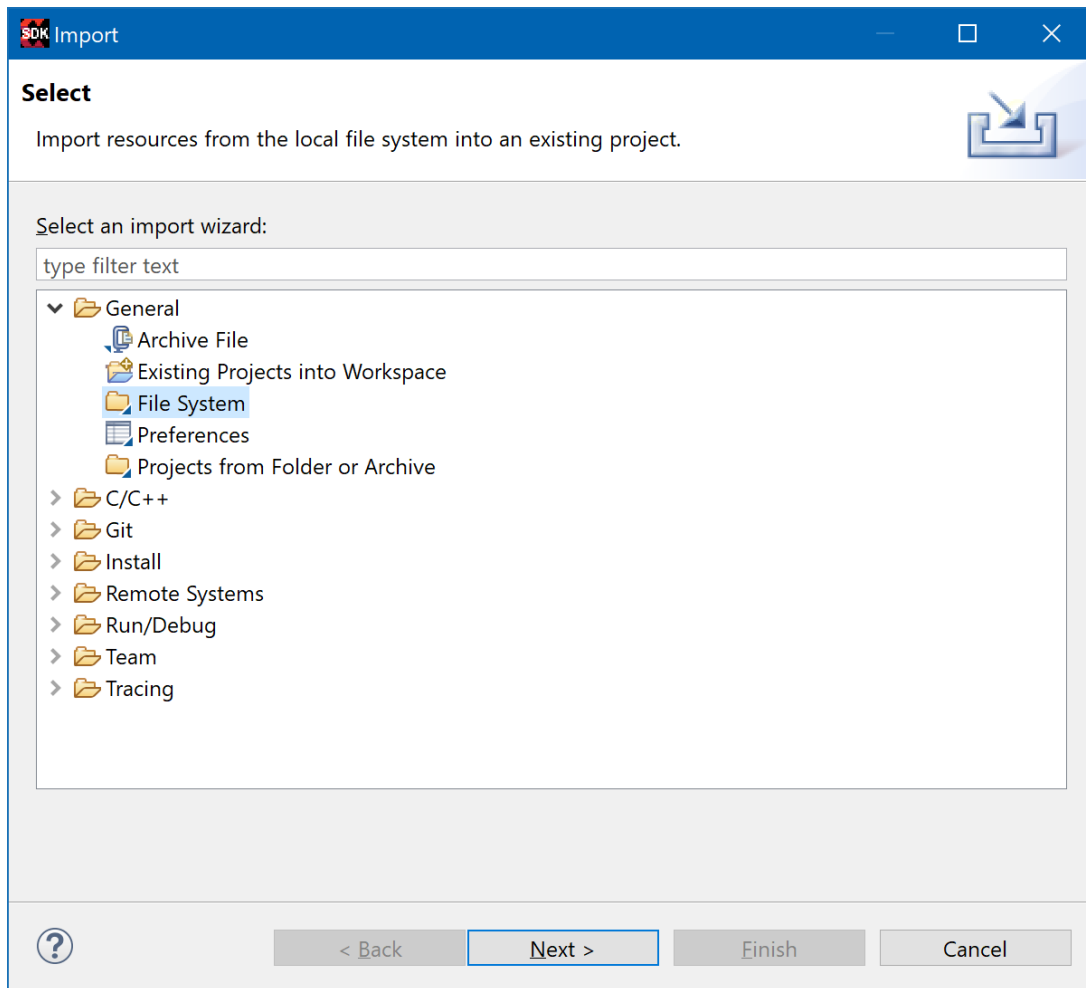


Figura 4. Importando código existente desde el sistema de archivos

- 2-2-7.** Navegar hasta la ubicación de la carpeta donde se encuentra el archivo **lab4.c** y hacer click en **OK**.
- 2-2-8.** Seleccionar el archivo **lab4.c** y hacer click en **Finish** para agregarlo al proyecto (por el momento, ignorar cualquier error).
- 2-2-9.** Expandir **lab4_bsp** y abrir **system.mss**.
- 2-2-10.** Hacer click sobre el link **Documentation** correspondiente al periférico **buttons** debajo de la sección **Peripheral Drivers** para abrir la documentación en un navegador. Ya que nuestro **led_ip** es muy similar al **GPIO**, miramos en la mencionada documentación.

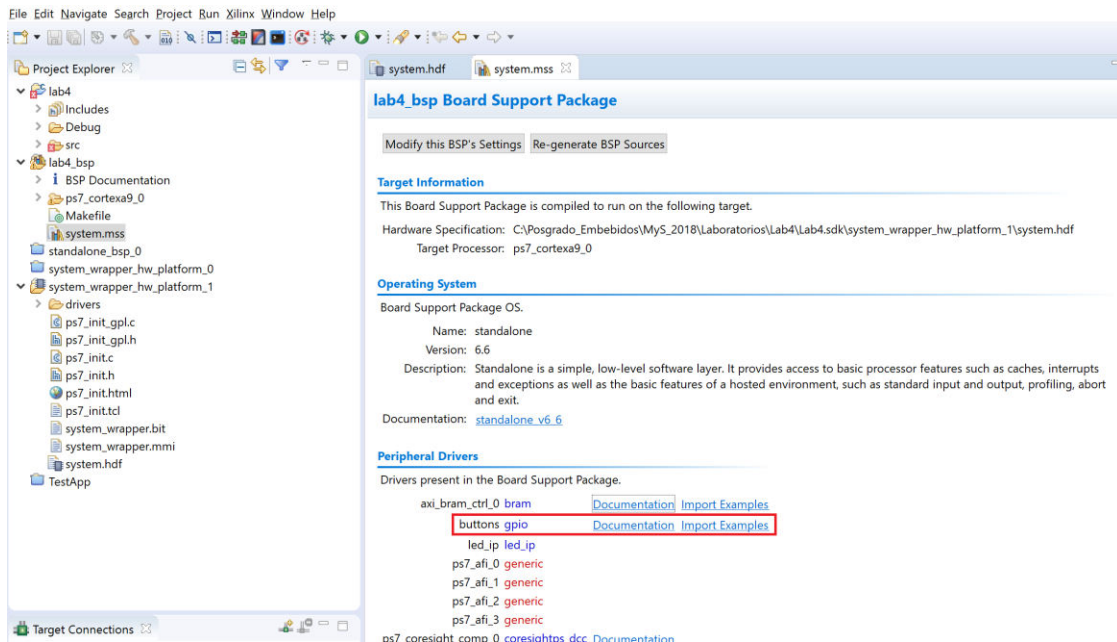


Figura 5

2-2-11. Ver los varios archivos C y Headers asociados con el GPIO haciendo click sobre la pestaña **File List** en la parte superior de la página.

2-2-12. Hacer doble click sobre **lab4.c** en la vista *Project Explorer* para abrir el archivo. Esto hará que se muestren en la ventana **Outline** xparameters.h, xgpio.h.

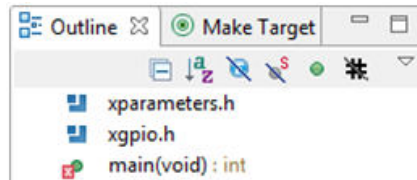


Figura 6. Pestaña Outline

2-2-13. Hacer doble click en xgpio.h en la vista *Outline* y revisar el contenido del archivo para ver las llamadas a función disponibles para el GPIO.

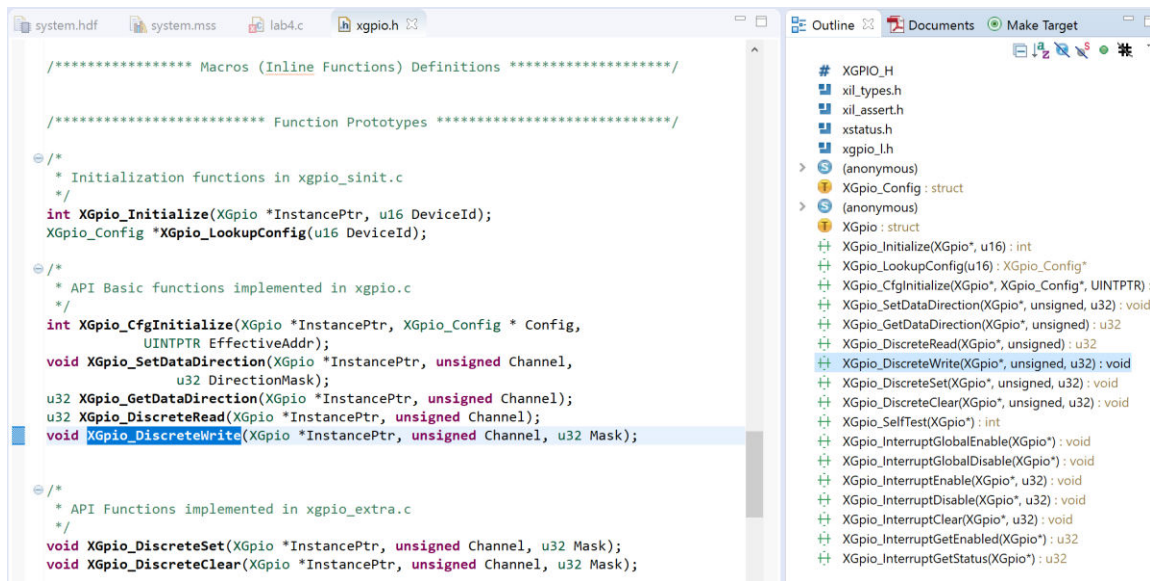


Figura 7

Para habilitar la lectura desde el GPIO se deben realizar los siguientes pasos en su aplicación de software:

- 1) Inicializar el GPIO
- 2) Establecer la dirección de los datos
- 3) Leer el dato

Encontrar las descripciones de las siguientes funciones:

XGpio_Initialize (XGpio *InstancePtr, u16 DeviceId)

InstancePtr es un puntero a una instancia XGpio. Las llamadas posteriores para manipular el componente a través del API XGpio debe ser hecho con este puntero.

DeviceId es la identificación única del dispositivo controlado por este componente XGpio. Al pasar un *device id* se asocia la instancia al genérico XGpio a un dispositivo específico, elegido por el caller o el desarrollador de la aplicación.

XGpio_SetDataDirection (XGpio * InstancePtr, unsigned Channel, u32 DirectionMask)

InstancePtr es un puntero a una instancia de XGpio para trabajar con él.

Channel contiene el canal del GPIO (1 or 2) para operar con él.

DirectionMask es una máscara que especifica qué bits son entradas y cuales salidas. Los bits colocados en 0 son salidas y los colocados en 1 entradas.

XGpio_DiscreteRead(XGpio *InstancePtr, unsigned channel)

InstancePtr es un puntero a la instancia de XGpio para trabajar con él.

Channel contiene el canal del GPIO (1 or 2) para operar con él.

2-2-14. Abrir el archivo **xparameters.h** haciendo doble clic sobre **xparameters.h** en la pestaña **Outline**

El archivo xparameters.h contiene el mapa de direcciones para los periféricos en el sistema. Este archivo es generado desde la descripción de la plataforma de hardware desde Vivado. Encuentre el siguiente #define usado para identificar el periférico **switches**:

```
#define XPAR_SWITCHES_DEVICE_ID 1 (Notar que el número puede ser diferente)
```

Notar las otras sentencias #define XPAR_SWITCHES* en esta sección para los periféricos switches, y en particular la dirección del periférico definida por: XPAR_SWITCHES_BASEADDR

2-2-15. Modificar la línea 14 del lab4.c para usar esta macro (#define) en la funciónin XGpio_Initialize.

En el caso en que no aparezca la numeración de las líneas de código presionar botón derecho sobre el código y elegir **Preferences ...** En la ventana siguiente expandir General ► Editors ► Text Editors y seleccionar *Show line numbers*

```

1  #include "xparameters.h"
2  #include "xgpio.h"
3
4  //=====
5
6  int main (void)
7  {
8
9      XGpio dip, push;
10     int i, psb_check, dip_check;
11
12     xil_printf("-- Start of the Program --\r\n");
13
14     XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID); // Modify this
15     XGpio_SetDataDirection(&dip, 1, 0xffffffff);
16
17     XGpio_Initialize(&push, XPAR_PUSH_DEVICE_ID); // Modify this
18     XGpio_SetDataDirection(&push, 1, 0xffffffff);
19
20
21     while (1)
22     {
23         psb_check = XGpio_DiscreteRead(&push, 1);
24         xil_printf("Push Buttons Status %x\r\n", psb_check);
25         dip_check = XGpio_DiscreteRead(&dip, 1);
26         xil_printf("DIP Switch Status %x\r\n", dip_check);
27
28         // output dip switches value on LED_ip device
29
30         for (i=0; i<99999999; i++);
31     }
32 }
```

Figura 8. Código fuente importado, resaltando las líneas para inicializar los switches como entradas, y leer desde ellos

- 2-2-16.** Hacer lo mismo para *BUTTONS*; encontrar la macro (`#define`) para el periférico *BUTTONS* en `xparameters.h`, y modificar la línea 17 en `lab4.c`, y guardar el archivo.

El proyecto será recompilado. Si hay algún error, verifique y corrija su código. Su código C leerá eventualmente el valor de los switches y lo sacará por `led_ip`.

2-3. Asignar el driver `led_ip` desde el directorio *driver* a la instancia `led_ip`.

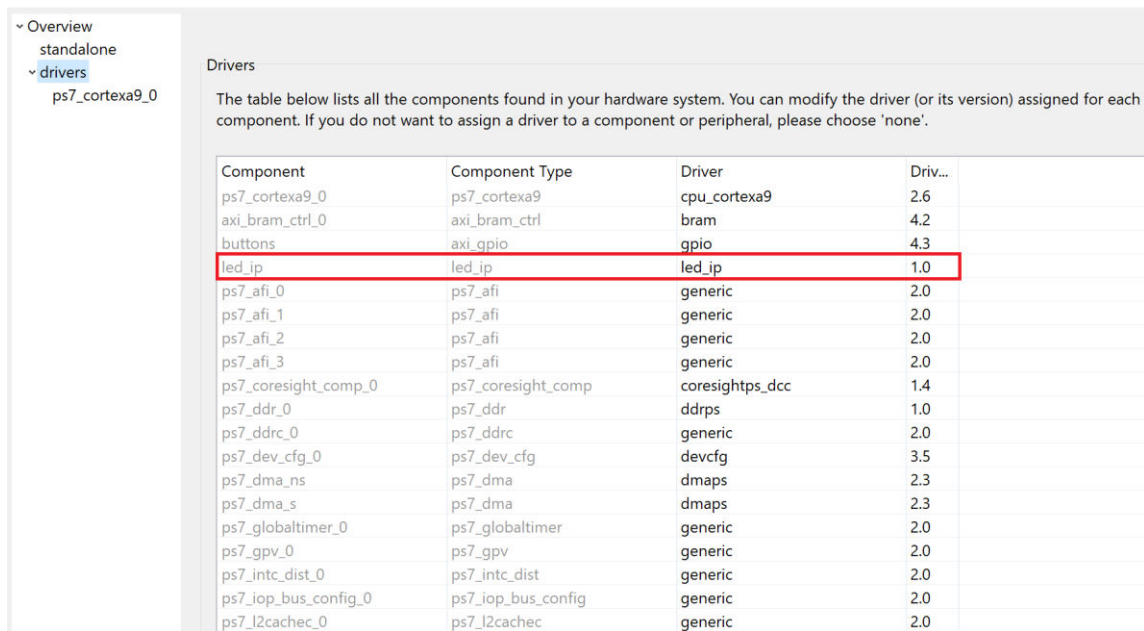
- 2-3-1.** Seleccionar `lab4_bsp` en la vista de proyecto, hacer click derecho, y seleccionar **Board Support Package Settings**.

- 2-3-2.** Seleccionar *drivers* a la izquierda (debajo de *Overview*)

- 2-3-3.** Si el driver `led_ip` no ha sido todavía seleccionado, seleccionar *Generic* debajo de la columna *Driver* para `led_ip` para acceder al menu. En el menú dropdown, seleccionar `led_ip`, y hacer click en **OK**.

Board Support Package Settings

Control various settings of your Board Support Package.



Component	Component Type	Driver	Driv...
ps7_cortexa9_0	ps7_cortexa9	cpu_cortexa9	2.6
axi_bram_ctrl_0	axi_bram_ctrl	bram	4.2
buttons	axi_gpio	gpio	4.3
led_ip	led_ip	led_ip	1.0
ps7_afi_0	ps7_afi	generic	2.0
ps7_afi_1	ps7_afi	generic	2.0
ps7_afi_2	ps7_afi	generic	2.0
ps7_afi_3	ps7_afi	generic	2.0
ps7_coresight_comp_0	ps7_coresight_comp	coresightps_dcc	1.4
ps7_ddr_0	ps7_ddr	ddrps	1.0
ps7_ddrc_0	ps7_ddrc	generic	2.0
ps7_dev_cfg_0	ps7_dev_cfg	devcfg	3.5
ps7_dma_ns	ps7_dma	dmaps	2.3
ps7_dma_s	ps7_dma	dmaps	2.3
ps7_globaltimer_0	ps7_globaltimer	generic	2.0
ps7_gpv_0	ps7_gpv	generic	2.0
ps7_intc_dist_0	ps7_intc_dist	generic	2.0
ps7_iop_bus_config_0	ps7_iop_bus_config	generic	2.0
ps7_l2cachec_0	ps7_l2cachec	generic	2.0

Figura 9. Asignar el driver `led_ip`

2-4. Examinar el código del Driver

El código del driver fue generado automáticamente cuando la plantilla de la IP fue creada. El driver incluye funciones de alto nivel que pueden ser llamadas desde la aplicación de usuario. El driver implementará la funcionalidad de bajo nivel usada para controlar su periférico.

- 2-4-1.** En un navegador de archivos, ir hasta el directorio `led_ip\ip_repo\led_ip_1.0\drivers\led_ip_v1_0\src`

Observar los archivos en este directorio y abrir `led_ip.c`. Este archivo solo incluye el archivo header para la IP.

2-4-2. Cerrar `led_ip.c` y abrir el archivo header `led_ip.h` y notar las macros:

```
LED_IP_mWriteReg( ... )
LED_IP_mReadReg( ... )
```

Por ejemplo: buscar la macro llamada `LED_IP_mWriteReg`:

```
/**
 *
 * Write a value to a LED_IP register. A 32 bit write is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is written.
 *
 * @param BaseAddress is the base address of the LED_IP device.
 * @param RegOffset is the register offset from the base to write to.
 * @param Data is the data written to the register.
 *
 * @return None.
 *
 * @note
 * C-style signature:
 * void LED_IP_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset,
 * Xuint32 Data)
 */
#define LED_IP_mWriteReg(BaseAddress, RegOffset, Data) \
Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))
```

Para este driver, puede verse que las macros son alias de las funciones de bajo nivel `Xil_Out32( )` y `Xil_Out32( )`. Las macros en este archivo arman el nivel alto de la API del driver `led_ip`. Si está escribiendo su propio driver para su propia IP, necesitará usar funciones de bajo nivel como estas para leer y escribir desde su IP. Las funciones de acceso al hardware de bajo nivel son encapsuladas en su driver haciendo más fácil usar su IP en un proyecto de aplicación.

2-4-3. Modificar su código C (ver la figura que sigue, o puede usar el archivo modificado provisto **lab4_sol.c**), para realizar el eco de los switches en los LEDs por medio de las macros de la API del driver `led_ip`, y guardar la aplicación.

2-4-4. Incluir el archivo `.h`:

```
#include "led_ip.h"
```

2-4-5. Incluir la función para escribir a la IP (insertarla antes del loop *for*):

```
LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, dip_check);
```

Recordar que la dirección de hardware para un periférico (por ej. la macro **XPAR_LED_PIN_S_AXI_BASEADDR** en la línea de arriba) puede ser encontrada en `xparameters.h`

```

#include "xparameters.h"
#include "xgpio.h"
#include "led_ip.h"

//=====

int main (void)
{
    XGpio dip, push;
    int i, psb_check, dip_check;

    xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID); // Modify this
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID); // Modify this
    XGpio_SetDataDirection(&push, 1, 0xffffffff);

    while (1)
    {
        psb_check = XGpio_DiscreteRead(&push, 1);
        xil_printf("Push Buttons Status %x\r\n", psb_check);
        dip_check = XGpio_DiscreteRead(&dip, 1);
        xil_printf("DIP Switch Status %x\r\n", dip_check);

        // output dip switches value on LED_ip device
        LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, dip_check);

        for (i=0; i<99999999; i++);
    }
}

```

Figura 10. Archivo C completo

2-4-6. Guardar el archivo y el programa volverá a compilarse.

Analizar los Archivos Objeto Ensamblados

Paso 3

3-1. Lanzar el Shell y objdump lab4.elf y mirar las secciones que ha creado.

3-1-1. Lanzar el shell desde el SDK seleccionando **Xilinx ► Launch Shell**.

3-1-2. Cambiar el directorio a **lab4\Debug** usando el comando **cd** en el shell.

3-1-3. Tipear **arm-none-eabi-objdump -h lab4.elf** en el shell para listar las diferentes secciones del programa, junto con la dirección de inicio y el tamaño de cada una de ellas.

Debería observar resultados similares a los que muestra la imagen siguiente:

Sections:						
Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	000019c4	00100000	00100000	00010000	2**6
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.init	00000018	001019c4	001019c4	000119c4	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
2	.fini	00000018	001019dc	001019dc	000119dc	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
3	.rodata	0000018c	001019f4	001019f4	000119f4	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
4	.data	00000498	00101b80	00101b80	00011b80	2**3
	CONTENTS, ALLOC, LOAD, DATA					
5	.eh_frame	00000004	00102018	00102018	00012018	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
6	.mmu_tbl	00004000	00104000	00104000	00014000	2**0
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
7	.init_array	00000004	00108000	00108000	00018000	2**2
	CONTENTS, ALLOC, LOAD, DATA					
8	.fini_array	00000004	00108004	00108004	00018004	2**2
	CONTENTS, ALLOC, LOAD, DATA					
9	.ARM.attributes	00000033	00108008	00108008	00018008	2**0
	CONTENTS, READONLY					
10	.bss	00000030	00108008	00108008	00018008	2**2
	ALLOC					
11	.heap	00002008	00108038	00108038	00018008	2**0
	ALLOC					
12	.stack	00003800	0010a040	0010a040	00018008	2**0
	ALLOC					

Figura 11. Resultados del dump - .text, .stack, y .heap en el espacio de DDR3

Verificar en Hardware

Paso 4

4-1. Conectar la placa con el cable micro-usb. Establecer una comunicación serie con alguna terminal (puede usar la terminal que brinda el SDK).

4-1-1. Asegurarse de que el cable micro-USB esté conectado entre la placa y la PC.

4-1-2. Configurar la terminal con 115200 como velocidad. Conectarse al puerto correspondiente.

4-2. Programar la FPGA seleccionando Xilinx ► Program FPGA y asignando el archivo system_wrapper.bit. Ejecutar la aplicación TestApp y verificar la funcionalidad.

4-2-1. Seleccionar **Xilinx ► Program FPGA**.

4-2-2. Hacer click en el botón **Program** para programar la FPGA.

- 4-2-3.** Seleccionar **lab4** en el *Project Explorer*, hacer click derecho y seleccionar **Run As ► Launch on Hardware (GDB)** para descargar la aplicación, ejecutar `ps7_init`, y ejecutar `lab4.elf`.

Mover los DIP switches y verificar que los LEDs reflejan los cambios. Verificar también que se observan los resultados de los switches DIP y los botones Push en la terminal.

```
DIP Switch Status C
Push Buttons Status 0
DIP Switch Status C
Push Buttons Status 0
DIP Switch Status C
Push Buttons Status 0
DIP Switch Status C
Push Buttons Status 0
DIP Switch Status C
```

Figura 12. Visualización de los switches DIP y los botones Push en la terminal

- 4-3. Cambiar el linker script para mover las secciones de código al controlador de BRAM y hacer un objdump lab4.elf y ver las secciones que se han creado.**

- 4-3-1.** Hacer click derecho sobre `lab4` y hacer click en **Generate Linker Script...**

Notar que las cuatro principales secciones, `code`, `data`, `stack` y `heap` deben ser asignadas al controlador BRAM.

- 4-3-2.** En la pestaña *Basic* cambiar las secciones *Code* y *Data* a **ps7_dds_0**, dejando la sección *Heap and Stack* en memoria **axi_bram_ctrl_0** y hacer click en **Generate**, y click nuevamente en **Yes** para sobrescribir.

Basic	Advanced
Place Code Sections in:	axi_bram_ctrl_0_Mem0
Place Data Sections in:	axi_bram_ctrl_0_Mem0
Place Heap and Stack in:	axi_bram_ctrl_0_Mem0
Heap Size:	1 KB
Stack Size:	1 KB

Basic	Advanced
Place Code Sections in:	ps7_dds_0
Place Data Sections in:	ps7_dds_0
Place Heap and Stack in:	axi_bram_ctrl_0_Mem0
Heap Size:	1 KB
Stack Size:	1 KB

Figura 13. Colocando las secciones Stack/Heap en BRAM

El programa compilará nuevamete.

- 4-3-3.** Tipear nuevamente en el shell **arm-none-eabi-objdump -h lab4.elf** para listar las distintas secciones del programa, como así también la dirección de inicio y el tamaño de cada una de ellas.

Debería observar resultados similares a los mostrados por la siguiente figura:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	000019c4	00100000	00100000	00010000	2**6
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.init	00000018	001019c4	001019c4	000119c4	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
2	.fini	00000018	001019dc	001019dc	000119dc	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
3	.rodata	0000018c	001019f4	001019f4	000119f4	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
4	.data	00000498	00101b80	00101b80	00011b80	2**3
	CONTENTS, ALLOC, LOAD, DATA					
5	.eh_frame	00000004	00102018	00102018	00012018	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
6	.mmu_tbl	00004000	00104000	00104000	00014000	2**0
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
7	.init_array	00000004	00108000	00108000	00018000	2**2
	CONTENTS, ALLOC, LOAD, DATA					
8	.fini_array	00000004	00108004	00108004	00018004	2**2
	CONTENTS, ALLOC, LOAD, DATA					
9	.ARM.attributes	00000033	00108008	00108008	00018008	2**0
	CONTENTS, READONLY					
10	.bss	00000030	00108008	00108008	00018008	2**2
	ALLOC					
11	.heap	00000400	40000000	40000000	00020000	2**0
	ALLOC					
12	.stack	00001c00	40000400	40000400	00020000	2**0
	ALLOC					

Figura 14. Las secciones .heap y .stack colocadas en BRAM mientras que el resto de la aplicación está en DDR

4-4. Ejecutar la aplicación lab4.elf y observar la aplicación trabajando aún cuando varias secciones están en memorias diferentes.

- 4-4-1.** Seleccionar **lab4** en *Project Explorer*, hacer botón derecho y seleccionar **Run As ► Launch on Hardware (GDB)** para descargar la aplicación, ejecutar ps7_init, y ejecutar lab4.elf

Hacer click en **Yes** si se lo solicita para parar la ejecución y correr la nueva aplicación.

Observar la terminal durante la ejecución del programa. Modificar el estado de los switches y observar los LEDs. Notar que el sistema es relativamente lento al mostrar los mensajes en la terminal y al cambiar el estado de los switches ya que el stack y el heap están en una memoria BRAM no-cacheda.

- 4-4-2.** Al terminar, hacer click en el botón **Terminate** en la pestaña *Console*.

4-4-3. Salir del SDK y Vivado.

4-4-4. Desconectar la placa.

Conclusión

Se usó el SDK para definir, desarrollar, e integrar los componentes de software del sistema embebido. Usted puede definir una interfaz de driver de dispositivo por cada uno de los periféricos y el procesador. El SDK importa un archivo hdf (Archivo de descripción de hardware), crea el correspondiente archivo MSS y le permite actualizar la configuración de tal manera que usted puede desarrollar el software del sistema de procesamiento. Usted puede desarrollar y compilar software funcional para un periférico específico y generar el archivo ejecutable desde el código objeto compilado y las librerías. Si es necesario, usted también puede usar un linker script para acceder a diferentes segmentos en varias memorias. Cuando la aplicación es demasiado grande para caber en la BRAM interna, puede descargar la aplicación en una memoria externa y luego ejecutar el programa.