Desarrollo de una aplicación para controlar IP propia

Objetivos

- Escribir una aplicación para acceder a un periférico propio
- Configurar un linker script para colocar el software en diferentes zonas de memoria
- Generar una imagen ejecutable de software en formato ELF

Procedimiento

Crear la aplicación

1. Iniciar la herramienta Vivado. Abrir el proyecto **Lab03PS** y guardarlo como **Labo04PS** (verificar que la opcion **Create project subdirectory** este seleccionada)

2. Seleccionar en el menu **File -> Export -> Export Hardware**.

3. En la ventana emergente tildar la opción **Include the bitstream** y presionar **OK**. En la ventana emergente presionar **Yes** (ya que el hardware ha cambiado).

4. Seleccionar el menu **File -> Launch SDK**. En la ventana emergente presionar **OK**.

5. Cerrar los proyectos **TestApp**, **standalonebsp0**,y **system_wrapper_hw_platform_1** presionando el boton derecho del raton en cada proyecto y eligiendo la opcion **Close Project**.

6. Seleccionar el menu **File- > New -> Application Project**.

7. En la entrada **Project name** ingresar **lab4**y en la opcion **Board Support Package** seleccionar **Create New**, con el nombre **lab4_bsp**.

8. Presionar **Next** y en la ventana **Available Templates** seleccionar **Empty Application**. Presionar **Finish**.

9. En **Project Explorer** expandir **lab4** y presionar el boton derecho del raton en la carpeta **src**. En el menu desplegable seleccionar la opcion **Import**.

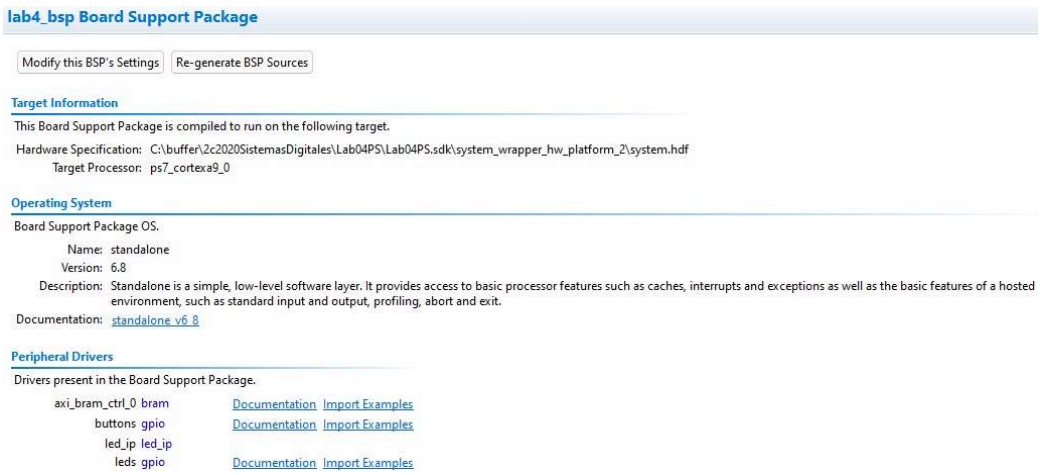
10. En la ventana emergente expandir **General** y dar doble click en **File System**.

11. Navegar a la carpeta donde se encuentra el archivo**lab4.c**.

12. Seleccionar el archivo **lab4.c** y presionar **Finish** para agregarlo al proyecto (ignorar los mensajes de error en la pestaña **Console** en la parte inferior de la pantalla).

13. Expandir el proyecto **lab4_bsp** y abrir el archivo **system.mss**.

14. En la seccion **Peripheral Drivers** buscar el periférico **leds** y presionar su enlace **Documentation** para abrir la documentacion del periferico en un navegador. Como la IP Propia **led_ip** es muy similar, utilizaremos esta documentación como referencia.

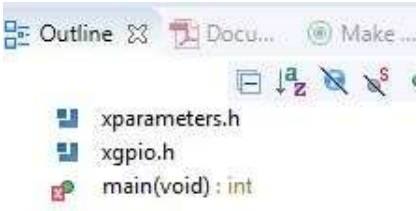


Documentacion de los periféricos

15. Analizar los distintos archivos *.c* y *.h* asociados con el periférico **GPIO** navegando a traves de la documentación.

16. En la herramienta SDK dar doble click al archivo **lab4/src/lab4.c** en el panel **Project Explorer** para abrir el archivo.

17. En la pestaña **Outline** (a la derecha) dar doble click en el archivo **xgpio.h** para ver los prototipos de las funciones disponibles para manejar un periférico **GPIO**.



Vista Outline

Para poder leer/escribir en un dispositivo GPIO se deben realizar los siguientes pasos: 1) inicializar el dispositivo, 2) configurar la direccion de los datos (entrada o salida segun si es lectura o escritura), y 3) escribir/leer los datos

Para ello son necesarias las siguientes funciones (buscar la descripción en la documentación):

XGpio_Initialize (XGpio *InstancePtr, u16 DeviceId)

InstancePtr is a pointer to an XGpio instance. The memory the pointer references must be pre-allocated by the caller. Further calls to manipulate the component through the XGpio API must be made with this pointer.

DeviceId is the unique id of the device controlled by this XGpio component. Passing in a device id associates the generic XGpio instance to a specific device, as chosen by the caller or application developer.

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

InstancePtr is a pointer to the XGpio instance to be worked on.

Channel contains the channel of the GPIO (1 or 2) to operate on.

DirectionMask is a bitmask specifying which bits are inputs and which are outputs. Bits set to 0 are output and bits set to 1 are input.

XGpio_DiscreteRead(XGpio *InstancePtr, unsigned channel)

InstancePtr is a pointer to the XGpio instance to be worked on.

Channel contains the channel of the GPIO (1 or 2) to operate on

18. Abrir el archivo cabecera **xparameters.h** haciendo doble click en **xparameters.h** en la pestaña **Outline**

19. El archivo **xparameters.h** contiene la ubicación de cada periférico dentro del mapa de direcciones del sistema. Este archivo se genera en forma automatica a partir de la descripción de la plataforma de hardware en Vivado. Verificar el siguiente "#define" usado para identificar el periférico **switches**:

```
#define XPAR_SWITCHES_DEVICE_ID 2
```

> (el numero puede ser diferente)

Notar que hay otros #define XPAR_SWITCHES_* en esta sección para el periférico. En particular su dirección de inicio: XPAR_SWITCHES_BASEADDR

15. Modificar la linea 14 de **lab4.c** para usar el #define del periférico SWITCHES en la funcion **Gpio_Initialize()**.
16. Modificar la linea 17 de **lab4.c** para usar el #define del periférico BUTTONS en la funcion **Gpio_Initialize()**.
17. Agregar en la linea 9 la variable tipo **XGpio** *leds*
18. Agregar las lineas 20 y 21 para inicializar y configurar el periférico **LEDS**
19. Agregar en la linea 10 la variable tipo **int** led_out
20. Agregar las lineas 29 a 31 para que los leds definidos como periférico GPIO (no la IP Propia) se enciendan de acuerdo a la configuración de los pulsadores e interruptores

```
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, led_out;
11
12   xil_printf("-- Start of the Program --\r\n");
13
14   XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
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   XGpio_Initialize(&leds, XPAR_LEDS_DEVICE_ID);
21   XGpio_SetDataDirection(&leds, 1, 0x00000000);
22
23   while (1)
24   {
25    psb_check = XGpio_DiscreteRead(&push, 1);
26    xil_printf("Push Buttons Status %x\r\n", psb_check);
27    dip_check = XGpio_DiscreteRead(&dip, 1);
28    xil_printf("DIP Switch Status %x\r\n", dip_check);
29    led_out = 0x00000000;
30    led_out = (dip_check << 4) | psb_check;
```

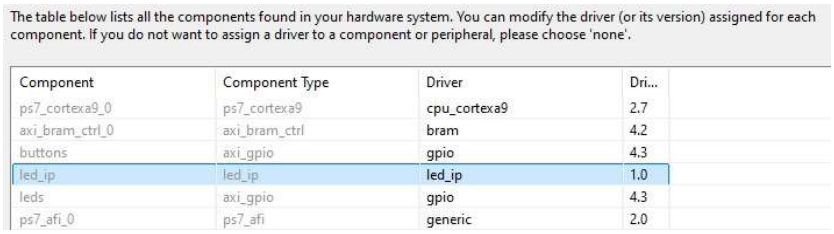
```
31     XGpio_DiscreteWrite(&leds,1,led_out);
32     // output dip switches value on LED_ip device
33
34     for (i=0; i<9999999; i++);
35    }
36 }
```

16. Guardar el archivo **lab4.c*. el proyecto se recompilara y no deberia tener errores de compilación

17. Seleccionar el proyecto **lab4_bsp**, presionar el boton derecho del raton y en el menu desplegable seleccionar la opcion **Board Support Package Settings**.

18. En la ventana emergente seleccionar **Overview -> drivers** (panel a la izquierda de la ventana)

19. Verificar que el Componente **led_ip** tenga asignado el Driver **led_ip**. Presionar **OK**. El proyecto **lab4_bsp** se recopilara. > Notar que los otros periféricos (*buttons*, *switches* y *leds*) tienen asignados un controlador (driver) genérico **gpio**



*la IP Propia tiene asignado el driver **led_ip***

Examinar el código del Controlador

El código del controlador fue generado automáticamente cuando se creo la IP. El controlador incluye funciones de alto nivel que pueden ser llamadas desde la aplicación del usuario e internamente implementa funciones de bajo nivel que interactuan con el hardware 1. En un navegador de archivos, navegar hasta el proyecto **led_ip**, navegar al directorio **led_ip_repo_ip_1.0_ip_v1_0* Verificar los archivos en ese directorio y abrir **led_ip.c**. Este archivo solo inclye al archivo encabezado **led_ip.h**. 2. Cerrar el archivo **led_ip.c** y abrir el archivo **led_ip.h** y verificar las funciones definidas a través de macros:

```
#define LED_IP_mWriteReg( … )
#define LED_IP_mReadReg( … )
```

Ejemplo para la funcion de escritura:

```
/**
 *
 * 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))
```

En este controlador en particular, las funciones de alto nivel accesibles por la aplicación son simplemente alias de las funciones de bajo nivel *Xil_Out32()* y *Xil_In32()*. Si fuera necesario realizar algun procesamiento de los datos de las funciones de alto nivel antes de invocar a las funciones de bajo nivel o viceversa, estas macros permiten incorporar dicho procesamiento antes o despues de invocar a las funciones de bajo nivel.

3. Modificar el codigo C de **lab4.c** para mostrar el valor de los pulsadores en los leds conectados a la IP Propia utilizando las funciones de alto nivel del controlador de la IP. Incluir el archivo de cabecera **led_ip.h**

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

5. Agregar la función para escribir en la IP de acuerdo a los valores de los pulsadores en la linea 33 antes del ciclo for:

```
33    LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, psb_check);
```

El código quedara de esta manera:

```
#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, psb_check);

      for (i=0; i<9999999; i++);
   }
}
```

6. Guardar el archivo. El proyecto se recompilara sin errores.

Analizar la zona de memoria desde donde se ejecuta el software

1. Ejecutar un terminal del SDK. En el menu **Xilinx -> Launch Shell**.
2. Cambiar al directorio **Lab04PS* mediante el comando** cd ./lab4/Debug**.
3. En el terminal ejecutar **arm-none-eabi-objdump –h lab4.elf**. Se veran las distintas secciones del programa y las posiciones de memoria que ocupan:

```
lab4.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00001a84  00100000  00100000  00010000  2**6
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .init         00000018  00101a84  00101a84  00011a84  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .fini         00000018  00101a9c  00101a9c  00011a9c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       0000018c  00101ab4  00101ab4  00011ab4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         000004a8  00101c40  00101c40  00011c40  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  5 .eh_frame     00000004  001020e8  001020e8  000120e8  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
```

Herramienta Object dump: .text, .stack, and .heap estan en la zona de memoria DDR3

> Notar que las secciones .text (codigo), .heap y .stack estan a partir de la direccion 0x00010_0000, que en archivo **xparameters.h** corresponde a la memoria DDR

```
#define XPAR_PS7_DDR_0_S_AXI_BASEADDR 0x00100000
```

Verificación funcional

1. Seleccionar **Xilinx -> Program FPGA**.
2. Presionar el boton **Program** para configurar el PL de la FPGA.
3. En el panel **Project Explorer** seleccionar **lab4**. Presionar el boton derecho del raton y en el menu desplegable seleccionar **Run As -> Launch on Hardware (System Debugger)** para programar el PS de la FPGA y ejecutar la aplicación
4. Seleccionar la pestaña **SDK Terminal**. Presionar el boton + para conectar el terminal a la placa.

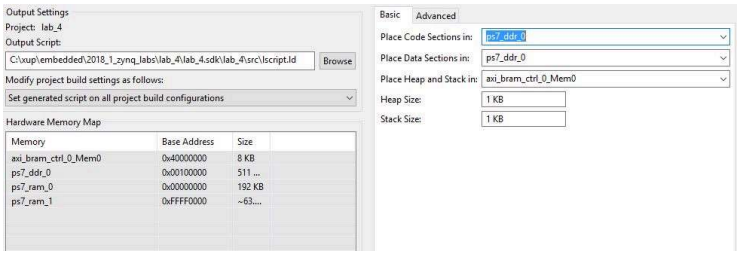
```
DIP Switch Status 1
Push Buttons Status 0
DIP Switch Status 1
Push Buttons Status 0
DIP Switch Status 1
Push Buttons Status 0
DIP Switch Status 3
```

En el terminal se ven los pulsadores e interruptores activados

Verificar que al cambiar los pulsadores cambia la información en pantalla y cambian los leds encendidos sobre los interruptores, verificando el funcionamiento de la IP Propia.

5. Presionar el boton derecho del raton en el proyecto **lab4**. En el menu desplegable elgir la opcion **Generate Linker Script…** Verificar que las 4 secciones de codigo (code, data, stack y heap) estan asignadas al **AXI BRAM Controller**.

6. En la pestaña **Basic** cambiar las entradas **Place Code Sections in:** y **Place Data Sections in:** a **ps7_ddr_0**, dejar la entrada **Place Heap and Stack in:** en **axi_bram_ctrl_0_Mem0** y presionar **Generate**. En la ventana emergente presionar **Yes** para sobreescribir el archivo **lscript.ld**.



Desplazar las secciones Stack y Heap a memoria BockRAM

La aplicación se recompilara..

7. En la terminal del SDK escribir el comando **arm-none-eabi-objdump –h lab4.elf**. Se veran las distintas secciones del programa y las nuevas posiciones de memoria que ocupan:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00001a84  00100000  00100000  00010000  2**6
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .init         00000018  00101a84  00101a84  00011a84  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .fini         00000018  00101a9c  00101a9c  00011a9c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       0000018c  00101ab4  00101ab4  00011ab4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         000004a8  00101c40  00101c40  00011c40  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  5 .eh_frame     00000004  001020e8  001020e8  000120e8  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
```

Las secciones .heap y .stack estan en BlockRAM, el resto esta en DDR

> Notar que la seccion .text (codigo) esta a partir de la direccion 0x00010_0000, que en archivo **xparameters.h** corresponde a la memoria DDR y las secciones .heap y .stack estan a partir de la dirección 0x4000_0000, que en archivo **xparameters.h** corresponde a la memoria BlockRAM

```
#define XPAR_PS7_DDR_0_S_AXI_BASEADDR 0x00100000

#define XPAR_BRAM_0_BASEADDR 0x40000000U
```

1. En el panel **Project Explorer** seleccionar **lab4**. Presionar el boton derecho del raton y en el menu desplegable seleccionar **Run As -> Launch on Hardware (System Debugger)** para programar el PS con la aplicación modificada para trabajar con las

dos zonas de memoria. En la ventana emergente presionar **OK** para detener la ejecución actual.

> El programa comenzará a ejecutarse. Al presionar los interruptores y pulsadores, se notara que la actualización de los leds es mucho mas lenta, lo mismo que la aparicion de los mensajes en la terminal. Esto se debe a que las secciones .stack y .heap estan en memoria sin cache y accedida a traves de transacciones del bus AXI, lo que enlentece la ejecucion del sistema.

2. Cerrar las herramientas SDK y Vivado. Desconectar la placa.

Conclusión

Se uso la herramienta SDK para desarrollar los componentes de software del sistema (controlador y aplicación). Es posible definir un controlador (driver) personalizado para cada periférico del sistema o usar el controlador genérico provisto. Mediante el linker script es posible definir en que zona de memoria se ubican las distintas secciones de un programa