Sistemas electrónicos para automatización

Memoria de las prácticas de FPGA

Alberto González Isorna

4º GIERM

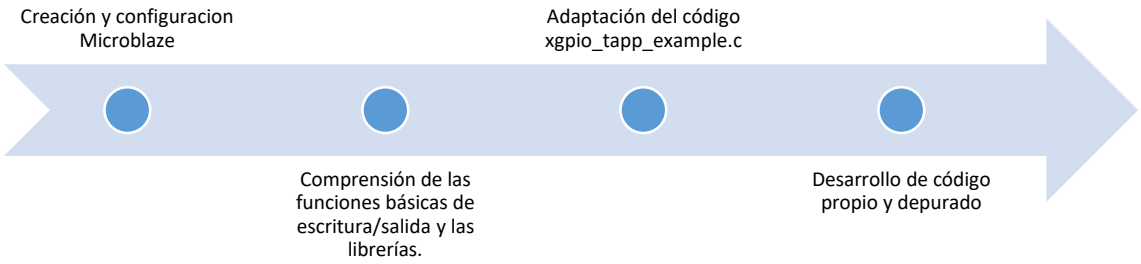
Introducción

En estas prácticas se pretende conocer el entorno de desarrollo EDK para manejar el procesador embebido Microblaze y lograr en última instancia comunicarlo con otras partes de la FPGA, ya sean puertos de entrada/salidas o Vhdl.

El objetivo de esta memoria es explicar cómo hemos procedido en la resolución de las distintas prácticas. Para ello en primer lugar al principio de cada apartado haremos una breve introducción, seguido de un esquema o diagrama de flujo explicativo y analizaremos las partes más importantes del código.

Practica 1 - Introducción al EDK (I) – Manejo de Leds (GPIO)

En esta práctica se pretende que, tras configurar el procesador microblaze, logremos manejar los leds que vienen soldados a la placa según convenga. Para ello se ha seguido el siguiente procedimiento:



Creación y configuracion Microblaze: La configuración del soft-core ha sido la sugerida en clase y la que viene especificada en las diapositivas: Reloj a 66 MHz, los periféricos por defecto y los códigos de test de memoria y de periférico. Tras algunos ajustes se consigue compilar el código que testea la memoria y conectando la placa a la UART vemos como efectivamente pasa los tests.

Comprensión de las funciones básicas de escritura/salida y las librerías: ahora pasamos al test de los periféricos. Al compilar el software, se autogenera una librería donde se alojan las direcciones base y las identificaciones necesarias de los dispositivos declarados para poder hacer referencias a ese dispositivo desde software.

En el ejemplo, xgpio_tapp_example.c, se encienden los leds uno a uno y además se leen sus valores si lo deseamos. Las funciones básicas para el manejo de los leds (o cualquier otro dispositivo conectado al GPIO) son las siguientes:

1. XGpio_Initialize(&GpioOutput, DeviceId): Esta función inicializa el driver del GPIO, si devuelve XST_SUCCESS es que podemos comenzar a usarlo, en caso contrario da un error.

2. XGpio_SetDataDirection(&GpioOutput, LED_CHANNEL, mask): esta función nos permite seleccionar si vamos a tener salidas o entradas en ese canal de ese dispositivo en concreto. Si ponemos un 0, serán salidas, si ponemos un 1 serán entradas.

3. XGpio_DiscreteWrite(&GpioOutput, LED_CHANNEL, mask): nos permite escribir según la máscara que pongamos en el registro asociado al dispositivo especificado. Si por ejemplo ponemos 0xf=se encenderán en nuestro caso todos los bits (00….1111).

4. XGpio_DiscreteRead(&GpioInput, LED_CHANNEL). Nos permite leer el valor del registro asociado al dispositivo.

El primer argumento de estas funciones corresponde a una estructura XGPIO, una estructura propia de Xilinx que contiene algunos argumentos como el baseadress e información de si el canal es dual.

Desarrollo de código propio y depurado del mismo: tras comprender el funcionamiento del código mencionado anteriormente, hemos realizado un código propio para el manejo de los leds, eliminando todas las funciones y código no necesario.

El código principal desarrollado va encendiendo igualmente el los leds uno a uno. En esta ocasión la función creada nos permite introducir un número entero del cero al cuatro y nos enciende el led seleccionado. Para ello nos hemos servido de dos funciones:

- **void enciendeled_concreto (u16 DeviceId,u8 led)**: esta función traduce el numero entero a la máscara necesaria para que se enciendan los leds correspondientes. Por ejemplo, si introducimos un 1, la máscara sería 0….0001. El resto de conversiones las podemos ver en la imagen siguiente:

```
switch (led)
{
    case 0: enciendeleds (GPIO_OUTPUT_DEVICE_ID,0x0); break;
    case 1: enciendeleds (GPIO_OUTPUT_DEVICE_ID,0x1); break;
    case 2: enciendeleds (GPIO_OUTPUT_DEVICE_ID,0x2); break;
    case 3: enciendeleds (GPIO_OUTPUT_DEVICE_ID,0x4); break;
    case 4: enciendeleds (GPIO_OUTPUT_DEVICE_ID,0x8); break;
    default: enciendeleds (GPIO_OUTPUT_DEVICE_ID,0xF); break;
}
```

- **int enciendeleds (u16 DeviceId, u8 mask):** la función anterior hace una llamada a la función enciendeleds, que realiza un procedimiento análogo a la función del ejemplo. Inicializa el GPIO y escribe el valor especificado en la máscara. El código completo se puede encontrar en el anexo.

Adicionalmente hemos hecho dos funciones de espera, una en segundos y otra en milisegundos. Para hallar la equivalencia hemos tenido en cuenta la frecuencia del micro y hemos ido ajustando el número cronometrándolo.

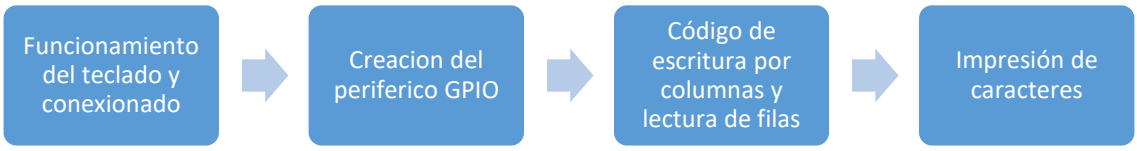
```
void espera_m (int mseg)
{
    volatile int Delay;
    int T;

    T=4500*mseg;

    for (Delay = 0; Delay < T; Delay++)
        asm("nop");
}
```

Practica 2 - Introducción al EDK (II) – Manejo de Keypad (GPIO)

En esta práctica se pretende controlar el Keypad con la placa, a través del GPIO. Para ello hemos seguido el siguiente esquema:



Funcionamiento del teclado y conexionado: para leer un carácter del teclado debemos escribir en las columnas y leer las filas, cruzando los resultados podemos saber que letra se ha pulsado. Tenemos 4 filas y 4 columnas por lo que tenemos 16 caracteres, del 0 al 15 hexadecimal.

Para conectar la placa correctamente a la placa y que la asignación de pines sea correcta debemos mirar el datasheet de la placa y del teclado y hacer una asociación correcta. En nuestro caso hemos conectado el teclado al j4, por lo que el ucf quedaría de la forma:

```
#------------------------------------------
# GPIO KEYPAD
#         INPUT
NET GPIO_KEYPAD_GPIO_IO_I_pin<0> LOC = K12 | IOSTANDARD = LVCMOS33;
NET GPIO_KEYPAD_GPIO_IO_I_pin<1> LOC = K13 | IOSTANDARD = LVCMOS33;
NET GPIO_KEYPAD_GPIO_IO_I_pin<2> LOC = F17 | IOSTANDARD = LVCMOS33;
NET GPIO_KEYPAD_GPIO_IO_I_pin<3> LOC = F18 | IOSTANDARD = LVCMOS33;
#         OUPUT
NET GPIO_KEYPAD_GPIO2_IO_O_pin<0> LOC = H12 | IOSTANDARD = LVCMOS33;
NET GPIO_KEYPAD_GPIO2_IO_O_pin<1> LOC = G13 | IOSTANDARD = LVCMOS33;
NET GPIO_KEYPAD_GPIO2_IO_O_pin<2> LOC = E16 | IOSTANDARD = LVCMOS33;
NET GPIO_KEYPAD_GPIO2_IO_O_pin<3> LOC = E18 | IOSTANDARD = LVCMOS33;
#------------------------------------------
```

Creación del periférico GPIO: para crear el gpio hemos creado un IP XGPIO con dos canales, uno de entrada y otro de salida, cada uno con un ancho de 4 bits ya que tenemos cuatro pines de entrada (filas) y cuatro de salida (columnas). El siguiente paso es establecer los puertos

apropiados como externos a la FPGA y asignarle espacio de memoria en el micro al periférico creado.

Código de escritura por filas y lectura por columnas: en primer lugar, sirviéndonos del ejemplo previo de los leds, se ha realizado una función que inicializa el KEYPAD, para poder llamarlo desde main directamente. La hemos denominado inicia_keypad() y la podemos ver en el código anexo.
Posteriormente dentro de la función leePKMOD (), que es la que lee el KEYPAD, entramos en un bucle que va escribiendo un cero desplazado según la variable del bucle. Esto es, vamos poniendo a nivel bajo las salidas uno a uno. La secuencia sería 1110, 1101, 1011, 0111, 1110, etc. El bucle a cada paso va leyendo las filas con la función num_fila() que explicaremos a continuación. Si hemos pulsado una fila, fila distinta de 0, seguimos leyendo hasta que el valor vuelva a ser cero (para evitar leer repetidas veces) y salimos de la función.

```
// bucle
while (fil==0)
{
for (col=0x0;col<4;col++)
    {
    col_aux=1<<col;
    XGpio_DiscreteWrite(&KYPAD_XGPIO, KEYPAD_CHANNEL_O, (~col_aux));
    // leemos la fila activa
    fil=num_fila();
    if(fil!=0)
        {
       while(num_fila()==fil); //esperamos a soltar el boton (suponem
            print("\n\nfin bucle\n\r fil = ");
            putnum(fil);
            print(" , col = ");
            putnum(col);
            print("\n\r");
            num=MATRIZ[fil-1][col];
            return num;
        }
    }
}
```

Una vez que tenemos que fila y que columna hemos pulsado, extremos el valor de la matriz MATRIZ que contiene todos los caracteres del teclado.

La función num_fila simplemente se dedica a leer el canal de salida y hacer la conversión de la máscara leída a la fila en cuestión. Las filas se han indexado desde la uno, el cero es sin pulsar, es por ello que que al extraer el carácter se ha puesto MATRIZ[fil-1][col].

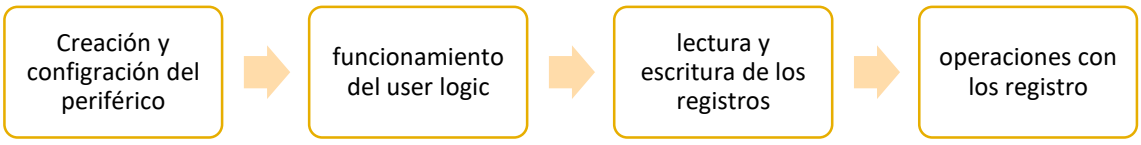
```
// ---------------------------------  Funcion que extrae el numero de fila
int num_fila()
{
u32 dato;
u32 result;
volatile int Delay;
// Lectura de la fila
dato=XGpio_DiscreteRead(&KYPAD_XGPIO,KEYPAD_CHANNEL_I); // ej 0000000 1101(32 bits (26 + 4 de las
// Conversion a entero
switch(dato)
    {
    case 15: result=0; break;
    case 14: result=1; break;
    case 13: result=2; break;
    case 11: result=3; break;
    case 7:  result=4; break;
    default: result=0; break;
    }
return result;
}
```

Por último cabe mencionar que para la impresión de los caracteres pulsados, debemos asignar el valor leído a una cadena de caracteres y añadir el terminador para que podamos verlo con la función print.

```
while(1)
{
mi_char[0]= leePKMOD ();
mi_char[1]='\0';
// imprimimos el numero
print("NUM is :  ");
print(mi_char);
print("\n\r");
}
return 0;
```

Practica 3 - Creación de periféricos personalizados con EDK

El fin de esta práctica es utilizar la herramienta de creación de periféricos personalizados. Esta es una herramienta mucho más potente que el GPIO ya que nos permite trabajar a velocidades mayores, podemos tener registros y podemos tener nuestro propio vhdl en el periférico que se comunique con el microprocesador, lo que da un abanico de posibilidades muy grande. El proceso seguido ha sido el siguiente:



Creación y configuración del periférico: hemos creado el periférico con cuatro registros y la configuración esclavo mínima. Tras conectarlo al plb y asignarle un espacio de memoria en el micro, ya podemos proceder a actuar sobre él.

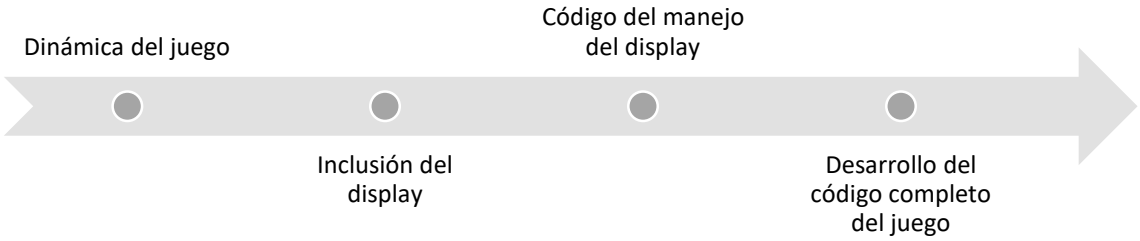
Funcionamiento del user logic y lectura/escritura de los registros: la lógica de nuestro periférico por defecto lee y escribe del registro, lo hemos modificado de forma que el tercer registro sea la suma de los dos primeros.

Para poder operar con estos registros desde el main debemos incluir la librería que se encuentra en los drivers del periférico y utilizar las funciones propias de lectura escritura.

Ya que en la ampliación (II) hemos ahondado más en el periférico PLB, veremos el funcionamiento específico más adelante.

Ampliación (I) – Juego Simón Display – GPIO

El propósito de esta primera ampliación era servirnos de lo que habíamos realizado anteriormente y realizar un juego. Para hacer más visual el juego se ha añadido un diaplay de 7 segmentos. En primer lugar la idea era crear un propio periférico que manejara tanto los leds como el display y el Keypad. Para ello deberíamos sacar pines externos del periférico custom, a través del user logic y el top del periférico. Debido a distintas dificultades al hacerlo, decidimos implementar el juego usando exclusivamente GPIO, e intentar resolver la conexión externa más adelante (ampliación II). El esquema seguido para la realización de esta parte ha sido el siguiente:



Dinámica del juego: inicialmente se muestran por el puerto serie las instrucciones y seguidamente empieza la mecánica del juego. El objetivo es ser capaz de repetir la secuencia aleatoria de números que se muestra por el display. En un principio sólo hay un número a repetir y cada vez que acertamos se incrementa el número de dígitos a repetir. Además, si acertamos el tiempo entre números cada vez es menor, lo que hace que sea más difícil repetir la secuencia. Si fallamos en algún número de la secuencia, se apaga el display y por el puerto serie sale un mensaje de error y se recomienza el juego. El juego tiene un número máximo de victorias tras las cuales el juego se acaba. Si queremos salir en cualquier momento podemos pulsar la tecla "D" e igualmente el juego acaba.

Los leds juegan el papel de advertencia, es decir, cada vez que introducimos una secuencia o acertamos o fallamos parpadean. Utilizan funciones ya vistas en el manejo de leds más la del parpadeo que veremos más adelante.

Inclusión del display: la inclusión del display se hace de forma análoga al KEYPAD. Se crea un GPIO y se añaden los 8 puertos de salida del display. Se asigna la memoria a este GPIO e implementamos una función similar a la que teníamos para inicializar el KEYPAD. Esta función inicializa de paso los leds. Esta función se llama inicia_display_leds y la podemos encontrar en el anexo.

Para asegurarnos del correcto conexionado del display tenemos que usar el datasheet de la placa y el del display. El conexionado sería el siguiente:

LETRA	PIN DISPLAY	NUM. PIN FPGA (J5)	UCF
A	7	9	D17
B	6	10	D18
C	4	3	C17
D	2	2	F16
E	1	1	F15
F	9	8	G14
G	10	7	F14
DP	5	4	C18
VCC	3	5	VCC

Código del manejo del display: para manejar el display de la forma más sencilla posible, se ha pasado de la tabla de equivalencia de números y segmentos en ánodo común (ver anexo) a su valor en hexadecimal. Por ejemplo el 0 que sería el 10000001 (orden DpABCDEF) equivale al 0x81. Todos estos valores decimales se han guardado en un vector de forma que en el número de la posición del vector se encuentra el valor hexadecimal correspondiente.

```
// ---------------------------------- Funcion representacion de DISPLAY
void representa(u32 num)
{
    // conversion a 7 segmenos
    u32 vnum[10]={0x81,0xF3,0x49,0x61,0x33,0x25,0x05,0xF1,0x01,0x31}; //anodo comun

    // escribimos
    XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O,vnum[num]);
}
```

La captura anterior es una versión simplificada, ya que tenemos que tener en cuenta si introducimos número no válidos, además hemos introducido un valor para que podamos apagar el display a nuestro antojo.

Desarrollo del código completo del juego: los pasos que sigue el juego, es decir el programa principal son los siguientes:

1. Inicializaciones: se imprime el inicio por el puerto serie (void imprime_ini()) e iniciamos tanto el keypad como los leds y el display con las funciones anteriormente mencionadas.

2. Parpadeo de los leds: el destello de leds marca el inicio del juego. Este destello se hace a través de una función que utiliza una XOR para ir cambiando el estado de salida. El número de parpadeos y los milisegundos parpadeando se introducen como variables globales.

3. Bucle while: el juego se ejecuta mientras las victorias no sean superior al máximo (Maxvict) o no pulsemos una D como dijimos anteriormente.

4. Generación de números aleatorios: tras observar que la función rand() de C dejaba sin espacio el micro, hemos tenido que implementar nuestra propia función que genere números aleatorios. Nos hemos basado en un algoritmo XOR shift.

```
void alb_rand()
{
    r^=r<<7;
    r^=r<<5;
    r^=r<<3;
}
```

5. Representación por el display: con la función descrita anteriormente, vamos representando estos números por el display (con un bucle for que va desde cero al número de victorias actuales).

6. Lectura por teclado: con el mismo bucle for se van leyendo los caracteres del teclado. En el caso de acertar se incrementa una variable y en caso contrario se sale del bucle, mostrando un mensaje de error.

7. Análisis de los resultados: en este punto pueden haber pasado dos cosas; por un lado si hemos acertado todas, el contador de aciertos será igual al número de dígitos, por lo que pasamos al siguiente nivel (Nvict++). Si hemos fallado ponemos el mensaje de error y recomenzamos (Nvict=0).

8. Fin: cuando se acaba el juego por uno de los dos motivos mencionados, se hace una evaluación de los aciertos totales y el número de veces que hemos pulsados.

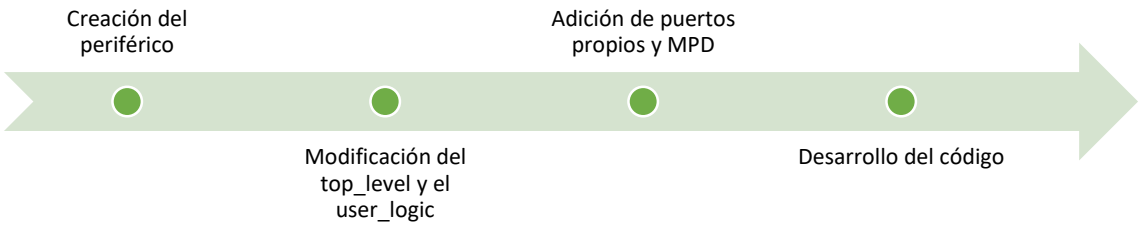
En la carpeta de esta presente memoria se encuentra adjunto un video donde se muestra el funcionamiento completo del juego.

Ampliación (II) – Simón Display – PLB

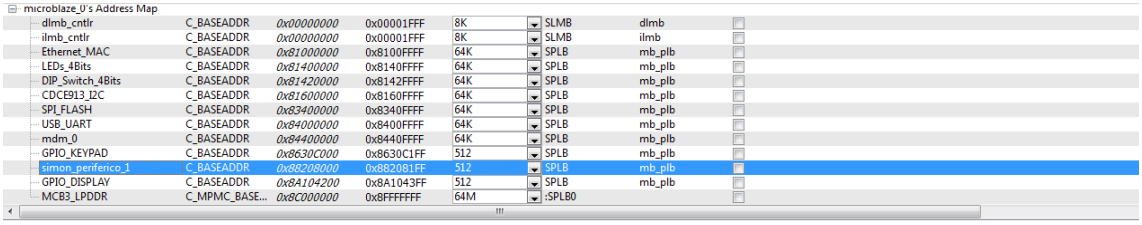
Tras conseguir el juego, se volvió a investigar cómo podríamos hacer lo mismo pero con el PLB. Desgraciadamente, no encontrábamos el fallo y parecía difícil llevarlo a cabo. Tras asistir a tutoría, descubrimos el problema: al poner una salida externa al registro este estaba escribiendo en los bits más significativos en vez de en los menos significativos como cabía esperar. Los bits 0 to 3 correspondían con los primeros bits, más significativos, por lo que había que cambiarlos con los bits 28 to 31.

Debido a falta de tiempo no se ha podido implementa la misma funcionalidad del juego con el periférico custom, aunque si hemos podido leer caracteres del teclado y representar con el periférico propio, por lo que queda resuelto el elemento diferenciador de esta ampliación.

El procedimiento llevado a cabo ha sido el siguiente:



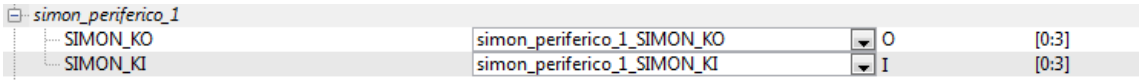
Creación del periférico: dado que ya teníamos creado el periférico no hay que hacer modificaciones en esta parte.



Modificación del top_level y el user_logic: en el top level tenemos que crear y mapear las señales de entrada salida. Además en el user logic tenemos que declararlas y hacer la lógica deseada. Dado que queremos que la salida corresponda al registro 0 y la entrada se guarde en el primer registro la lógica implementada sería la siguiente:

```
153   -- Alberto Logic
154   ---------------------------------------
155   SIMON_KO(0 to 3)    <= slv_reg0(28 to 31);
156   slv_reg1(28 to 31) <= SIMON_KI (0 to 3);
157
```

Adición de puertos propios y MPD: en primer lugar, hay que poner los puertos como externos. Tras esto el siguiente paso es modificar el MPD para añadir los puertos necesarios. Además se ha añadido una opción para que el periférico esté en modo “developpement” y detecte todos los cambios de hardware. A continuación se muestra la adición de puertos en el MPD:




```
37  ## Ports
38  PORT SIMON_KO = "", DIR = O, VEC = [0:3]
39  PORT SIMON_KI = "", DIR = I, VEC = [0:3]
```

Desarrollo del código: el programa detecta el valor leído por teclado, a través de los puertos del periférico custom (simon_periférico_1) y representamos en el display el número leído. Las funciones del keypad son las mismas que la utilizada en la ampliación anterior, salvo que las funciones de lectura y escritura se reemplazan por la escritura y lectura de los registros concretos. Las funciones están disponibles en los drivers del periférico y se pueden ver en el código anexo. Por ejemplo para leer la fila:

```
57  // Lectura de la fila
58  dato=SIMON_PERIFERICO_mReadSlaveReg1(mi_baseaddr, 0);
```

Conclusiones

Las conclusiones que se han podido sacar de la realización tanto de las prácticas como de la ampliación son las siguientes:

- La síntesis de un soft-core como Microblaze, pese a ser algo muy complejo se puede implementar con relativa facilidad con la herramienta EDK y las guías correspondientes.
- El hecho de tener un micro en una FPGA abre un mundo muy amplio de posibilidades, ya que podemos por un lado tener un programa ejecutando procesos o culminándose con el exterior de una forma relativamente sencilla y usar además la parte de vhdl con la que podemos crear el circuito que nos convenga, optimizando así las prestaciones y el tiempo de diseño.
- El manejo del keypad y del display nos ha aportado un conocimiento más amplio acerca de cómo funciona un microprocesador por dentro, el mapeo de direcciones y puertos y como encajan, después de esfuerzo, todas las piezas de esa enorme estructura para que funcione el proyecto completo.
- Sacar pines fuera de la FPGA con un periférico custom es una tarea difícil, ya que aparte de tener que modificar correctamente varios archivos, la asignación se hace en big endian, por lo que hay que prestar atención a los bits que usamos.
- En general, puedo decir que en mi opinión he aprendido bastante con estas prácticas ya que me han ayudado a entender de una forma global y a veces específica como trabajan un microprocesador y una FPGA, tanto por separados como juntos.

Anexo A: Código ampliación (I)

```
-------------------------------------------------------------
----------------- AMPLICACION 1 --------- GPIO ----------------
-------------------------------------------------------------

/*************************** Include Files *******************************/
#include "xparameters.h"
#include "xgpio.h"
#include "stdio.h"
#include "xstatus.h"
#include "simon_periferico.h"
/************************** Constant Definitions *************************/

#define LED_DELAY     1000000

#define LED_CHANNEL 1// parameters.h

#define LED_MAX_BLINK     0x1    /* Number of times the LED Blinks */

#define GPIO_BITWIDTH     4      /* This is the width of the GPIO */

#define printf xil_printf /* A smaller footprint printf */


#define GPIO_OUTPUT_DEVICE_ID    XPAR_LEDS_4BITS_DEVICE_ID
#define GPIO_INPUT_DEVICE_ID     XPAR_LEDS_4BITS_DEVICE_ID

//----------------- KEYPAD ------------------
// un gpio con dos canales
#define KEYPAD_CHANNEL_I 1
#define KEYPAD_CHANNEL_O 2
#define KEYPAD_DEVICE XPAR_GPIO_KEYPAD_DEVICE_ID
XGpio KYPAD_XGPIO;

//----------------- DISPLAY ------------------
#define DISPLAY_CHANNEL_O 1
#define DISPLAY_DEVICE XPAR_GPIO_DISPLAY_DEVICE_ID
XGpio DISPLAY_XGPIO;


//----------------- LEDS ---------------------
#define leds_CHANNEL_O 1
#define leds_DEVICE XPAR_LEDS_4BITS_DEVICE_ID
XGpio leds_XGPIO;


/************************** Function Prototypes **************************/
int GpioOutputExample(u16 DeviceId, u32 GpioWidth);
int GpioInputExample(u16 DeviceId, u32 *DataRead);

void GpioDriverHandler(void *CallBackRef);

// funciones propias
int enciendeleds (u16 DeviceId, u8 mask);
void enciendeled_concreto (u16 DeviceId,char led);
void espera (int n_seg);


// Funciones KEYPAD
int num_fila();
int inicia_keypad();
char leePKMOD ();
char *leePKMOD_str ();
```

```
//Funciones DISPLAY
void representa(u32 num);
int inicia_display_leds();

// Espera
void espera_m (u32 mseg);

// Imprime ini
void imprime_ini();

// parpadeo leds
void parpadeo_leds();
#define Numblinks 3
#define Tblink 100 // ms


// Aleatorio
u8 r=7;

void alb_rand()
{
	r^=r<<7;
	r^=r<<5;
	r^=r<<3;
}

// JUEGO
u8 Nvict=1;
#define Maxvict 10
u8 cont=0;
u8 aciertos=0;
u8 fallos=0;


//*****************************************************************************
**/
char MATRIZ[4][4]={{'1','2','3','A'},{'4','5','6','B'},
{'7','8','9','C'},{'0','F','E','D'}};

int main(void)
{
	int s1,s2;
	u32 InputData,dato;
	char mi_char[2];
	int i=0,j=0;
	volatile int Delay;
	u32 numdisplay[Maxvict];

	print("\tHola Mundo...\n\r");
	imprime_ini();
	// --- Inicio Keypad -----------
	s1=inicia_keypad();
	inicia_display_leds();
	if (s1==XST_FAILURE || s2==XST_FAILURE)
	{
	print("Error INI");
	return 'Z';
	}
	else
	{
	print("-> DISPLAY... OK\n\r");
	print("-> KEYPAD... OK\n\n\r");
	}
	parpadeo_leds();
	// ----------------------------
	while(Nvict<Maxvict && mi_char[0]!='D')
	{
	// numero aleatorio+
```

```
        for (j=0;j<Nvict;j++)
        {
        alb_rand();
        numdisplay[j]=(r*(i++))%10;
        espera_m((u32)100*(Maxvict-Nvict));
        // representamos
        representa(numdisplay[j]);
        }
        parpadeo_leds();
        // leemos el numero
        print("  Introduzca Numeros  ");
        for (j=0;j<Nvict;j++)
        {
        mi_char[0]= leePKMOD();
        if (mi_char[0]=='D')
        continue;
        else
        {
        mi_char[1]='\0';
        print("\n\r");
        // imprimimos el numero
        print("  NUM introducido is :  ");
        print(mi_char);
        print("\n\r");
        if (mi_char[0] == (char)(numdisplay[j]+'0'))
        cont++;
        else
        break;
        }
        }
        /*if (mi_char[0]=='D')
        continue;*/
        // Vemos si coincide
        //numdisplay=(u32)(mi_char[0]-'0');
        if (cont==Nvict)
               {
               print("\a\n");
               print("\t--- OK ----\n\n\r");
               Nvict++;aciertos++;
               SIMON_PERIFERICO_mWriteSlaveReg2(0, 0,aciertos);
               }
        else
               {
               Nvict=1; fallos++;
               SIMON_PERIFERICO_mWriteSlaveReg3(0, 0,fallos);
               print("\t--- ERROR ----\n\n\r");
               representa(0xA);
               }
        cont=0;
        parpadeo_leds();
        }
        print("Se acabó...\n\r");
        dato=SIMON_PERIFERICO_mReadSlaveReg1(0, 0);
        print("exitos: ");
        putnum(aciertos);print("\n\r");
        print("pulsados: ");
        putnum(dato);print("\n\r");

        return 0;
}


// ------------------------------------ Funcion inicializacion de KEYPAD
int inicia_keypad()
{
        int Status;
        // inicializamos inputs FIL y outputs COL
        Status = XGpio_Initialize(&KYPAD_XGPIO, KEYPAD_DEVICE);
```

```
	if (Status != XST_SUCCESS)
	 return XST_FAILURE;
	// FIL
	XGpio_SetDataDirection(&KYPAD_XGPIO,KEYPAD_CHANNEL_I,0xF);
	// COL
	XGpio_SetDataDirection(&KYPAD_XGPIO,KEYPAD_CHANNEL_O,0x0);
	return XST_SUCCESS;
}

// ---------------------------------  Funcion que extrae el numero de fila
int num_fila()
{
u32 dato;
u32 result;
volatile int Delay;
// Lectura de la fila
dato=XGpio_DiscreteRead(&KYPAD_XGPIO,KEYPAD_CHANNEL_I); // ej 0000000 1101(32
bits (26 + 4 de las filas))
// Conversion a entero
switch(dato)
	{
	case 15: result=0; break;
	case 14: result=1; break;
	case 13: result=2; break;
	case 11: result=3; break;
	case 7:  result=4; break;
	default: result=0; break;
	}
return result;
}

// ---------------------------------- Funcion Global que lee el char del
keypad
char leePKMOD ()
{
	char num='X';
	u32 col;
	int fil=0;
	volatile int Delay;
	// bucle
	while (fil==0)
	{
	for (col=0x0;col<4;col++)
		{
		XGpio_DiscreteWrite(&KYPAD_XGPIO, KEYPAD_CHANNEL_O, (~(1<<col)));
		// leemos la fila activa
		fil=num_fila();
		if(fil!=0)
			{
		   while(num_fila()==fil); //esperamos a soltar el boton
(suponemos que estamos manteniendo el mismo)
				num=MATRIZ[fil-1][col];
				break;
			}
		}
	}
	return num;
}



// ---------------------------------- Funcion Global que lee str
char *leePKMOD_str()
{
	char num='X';
	char *mi_str;
	u32 col;
	int fil=0;
```

```
	volatile int Delay;
	// bucle
	while (fil==0)
	{
	for (col=0x0;col<4;col++)
		{
		XGpio_DiscreteWrite(&KYPAD_XGPIO, KEYPAD_CHANNEL_O, (~(1<<col)));
		// leemos la fila activa
		fil=num_fila();
		if(fil!=0)
			{
		   while(num_fila()==fil); //esperamos a soltar el boton
(suponemos que estamos manteniendo el mismo)
				num=MATRIZ[fil-1][col];
				mi_str[0]=num;
				mi_str[1]='\0';
				break;
			}
		}
	}
	return mi_str;
}

// ---------------------------------- Funcion inicializacion de DISPLAY y
Leds
int inicia_display_leds()
{
	int Status;
	//---- display --------
	Status = XGpio_Initialize(&DISPLAY_XGPIO, DISPLAY_DEVICE);
	if (Status != XST_SUCCESS)
	 return XST_FAILURE;
	XGpio_SetDataDirection(&DISPLAY_XGPIO,DISPLAY_CHANNEL_O,0x0);
	XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O, 0xFF);
	// ---- leds --------
	Status = XGpio_Initialize(&leds_XGPIO, leds_DEVICE);
	if (Status != XST_SUCCESS)
	 return XST_FAILURE;
	XGpio_SetDataDirection(&leds_XGPIO,leds_CHANNEL_O,0x0);
	return XST_SUCCESS;
}

// ---------------------------------- Funcion representacion de DISPLAY
void representa(u32 num)
{
	// conversion a 7 segmenos
	u32 vnum[10]={0x81,0xF3,0x49,0x61,0x33,0x25,0x05,0xF1,0x01,0x31};
//anodo comun
	u32 result;

	// borramos lo que haya
	XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O, 0xFF);

	if(num<=9 && num>=0)
	result=(vnum[num]);
	else
	if (num=0xA)
	result=0xFF; //apagamos todos
	else
	result=0x00; //encendemos todos

	// escribimos
	XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O,result);
}

// ---------------------------------- Funcion espera (ms)
void espera_m (u32 mseg)
{
```

```
	volatile int Delay;
	int T;

	T=4500*mseg;

	for (Delay = 0; Delay < T; Delay++)
		asm("nop");
}


void imprime_ini()
{
print("\n\n------- SIMON DISPLAY -------- \n\r");
print("\n\n\t --- Instrucciones ---   \n\n\r");
print("1-Espera el inicio\n\r");
print("2-El parpadeo de leds indica el inicio de la secuencia\n\r");
print("3-Ver la secuencia e intentar memorizarla\n\r");
print("4-Esperar a que termine la secuencia con el parpadeo de leds\n\r");
print("5-Introducir en el KEYPAD la secuencia\n\r");
print("6-Pulsar la tecla [D] que indica fin de la introduccion\n\r");
print("7-Si es correcto, seguir jugando\n\r");
print("8-Si fallamos,  volver a empezar\n\r");
print("\n\n------- SIMON DISPLAY -------- \n\n\n\r");
}

void parpadeo_leds()
{
	u8 out=0x0;
	int i;
	for (i=0;i<(Numblinks<<1);i++)
	{
	out^=0xF;
	XGpio_DiscreteWrite(&leds_XGPIO, leds_CHANNEL_O,out);
	espera_m (Tblink);
	}
}
```

Anexo B: Código Ampliación (II)

```
*******************************/
#include "stdio.h"
#include "alberto_KEYPAD3.h"
#include "alberto_DISPLAY_GPIO.h"


int main()
{
XStatus s1,s2;
u32 num;
char mi_char[2];

s1=inicia_keypad();
s2=inicia_display();

if (s1==XST_FAILURE || s2==XST_FAILURE)
	{
	print("Error INI");
	return 'Z';
	}
else
	{
	print("-> DISPLAY... OK\n\r");
	print("-> KEYPAD... OK\n\n\r");
	}
```

```
while(1)
	{
	mi_char[0]= leePKMOD();
	mi_char[1]='\0';
	// imprimimos el numero
	print("NUM is :  ");
	print(mi_char);
	print("\n\r");
	num=(u32)(mi_char[0]-'0');
	// representamos display
	representa(num);
	}

return 0;
}
```

Anexo C: librería propia “alberto_KEYPAD3.h”

```
/*---------------------------
LIBRERIA .h para el keypad
	Alberto Gonzalez Isorna
	10/01/18
---------------------------*/

// Includes
#include "xstatus.h"
#include "xgpio.h"
#include "xparameters.h"


// Especificaciones periferico
#include "simon_periferico.h"
#define mi_baseaddr XPAR_SIMON_PERIFERICO_1_BASEADDR
#define KEYPAD_DEVICE XPAR_SIMON_PERIFERICO_1_DEVICE_ID
XGpio KYPAD_XGPIO;

// Prototipos Funciones KEYPAD
void GpioDriverHandler(void *CallBackRef);
int num_fila();
int inicia_keypad();
char leePKMOD ();

// Matriz
char MATRIZ[4][4]={{'1','2','3','A'},{'4','5','6','B'},
{'7','8','9','C'},{'0','F','E','D'}};


#define LED_DELAY     10000000



//----------------- keypad ------------------
// un gpio con dos canales
#define KEYPAD_DATA_I *((volatile u32 *)(mi_baseaddr))
#define KEYPAD_TRI_I *((volatile u32 *)(mi_baseaddr+0x4))
#define KEYPAD_DATA_O *((volatile u32 *)(mi_baseaddr+0x8))
#define KEYPAD_TRI_O *((volatile u32 *)(mi_baseaddr+0xC))


// ---------------------------------- Funcion inicializacion de KEYPAD
int inicia_keypad()
{
	// FIL -- INPUT
```

```
	KEYPAD_TRI_I = 0xFFFFFFFF;
	// COL -- OUTPUT
	KEYPAD_TRI_O = 0x00000000;
	return XST_SUCCESS;
}

// ----------------------------------  Funcion que extrae el numero de fila
int num_fila()
{
u32 dato;
u32 result;
volatile int Delay;
// Lectura de la fila
dato=SIMON_PERIFERICO_mReadSlaveReg1(mi_baseaddr, 0);
// Conversion a entero
switch(dato)
	{
	case 15: result=0; break;
	case 14: result=1; break;
	case 13: result=2; break;
	case 11: result=3; break;
	case 7:  result=4; break;
	default: result=0; break;
	}
return result;
}

// ----------------------------------- Funcion Global que lee el char del
keypad
char leePKMOD ()
{
	char num='X';
	u32 col,dato;
	int fil=0;
	volatile int Delay;
	// bucle
	while (fil==0)
	{
	for (col=0x0;col<4;col++)
		{
		SIMON_PERIFERICO_mWriteSlaveReg0(mi_baseaddr, 0, (~(1<<col)));
		// leemos la fila activa
		fil=num_fila();
		if(fil!=0)
			{
		   while(num_fila()==fil); //esperamos a soltar el boton
(suponemos que estamos manteniendo el mismo)
				num=MATRIZ[fil-1][col];
				break;
			}
		}
	}
	return num;
}
```

Anexo D: librería propia “alberto_DISPLAY_GPIO.h”

```
/*----------------------------
LIBRERIA .h para el display
       Alberto Gonzalez Isorna
       10/01/18
----------------------------*/


//----------------- DISPLAY ------------------
#define DISPLAY_CHANNEL_O 1
#define DISPLAY_DEVICE XPAR_GPIO_DISPLAY_DEVICE_ID
XGpio DISPLAY_XGPIO;

//--------------Funciones DISPLAY--------------
void representa(u32 num);
int inicia_display();


// ---------------------------------- Funcion inicializacion de DISPLAY y
Leds
int inicia_display()
{
       int Status;
       //---- display --------
       Status = XGpio_Initialize(&DISPLAY_XGPIO, DISPLAY_DEVICE);
       if (Status != XST_SUCCESS)
        return XST_FAILURE;
       XGpio_SetDataDirection(&DISPLAY_XGPIO,DISPLAY_CHANNEL_O,0x0);
       XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O, 0xFF); //apagado
       return XST_SUCCESS;
}


// ---------------------------------- Funcion representacion de DISPLAY
void representa(u32 num)
{
       // conversion a 7 segmenos
       u32 vnum[10]={0x81,0xF3,0x49,0x61,0x33,0x25,0x05,0xF1,0x01,0x31};
//anodo comun
       u32 result;

       // borramos lo que haya
       XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O, 0xFF);

       if(num<=9 && num>=0)
       result=(vnum[num]);
       else
       if (num=0xA)
       result=0xFF; //apagamos todos
       else
       result=0x00; //encendemos todos

       // escribimos
       XGpio_DiscreteWrite(&DISPLAY_XGPIO, DISPLAY_CHANNEL_O,result);
}
```

Anexo E: conversiones número a segmento (display)

	Catodo Comun							
	Numero	A	B	C	D	E	F	G
Enable	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
0	2	1	1	0	1	1	0	1
0	3	1	1	1	1	0	0	1
0	4	0	1	1	0	0	1	1
0	5	1	0	1	1	0	1	1
0	6	1	0	1	1	1	1	1
0	7	1	1	1	0	0	0	0
0	8	1	1	1	1	1	1	1
0	9	1	1	1	1	0	1	1

	Anodo Comun							
	Numero	A	B	C	D	E	F	G
Enable	0	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	1
1	2	0	0	1	0	0	1	0
1	3	0	0	0	0	1	1	0
1	4	1	0	0	1	1	0	0
1	5	0	1	0	0	1	0	0
1	6	0	1	0	0	0	0	0
1	7	0	0	0	1	1	1	1
1	8	0	0	0	0	0	0	0
1	9	0	0	0	0	1	0	0