APÉNDICE B

En este apéndice se va a crear un tutorial para crear una aplicación muy simple para poder controlar los leds integrados en la PYNQ – Z2. Se usarán los dos switches de la placa, junto con los 4 botones disponibles, 4 leds normales, y un led rgb. Los leds se van a encender de 4 formas distintas:

- Config sw 0x00: Al pulsar los diferentes botones, el led RGB se encenderá de diferentes colores según el botón que se pulse:

Botón	Color Led RGB
BTN0	Rojo
BTN1	Verde
BTN2	Azul
BTN3	Blanco

Tabla B. 1: Función botones (1) tutorial.

- Config sw 0x01: Al pulsar los diferentes botones se encenderá uno de los 4 leds normales.

Botón	Led Encendido
BTN0	LD0
BTN1	LD1
BTN2	LD2
BTN3	LD3

Tabla B. 2: Función botones (2) tutorial.

- Config sw 0x10: Al pulsar los diferentes botones, se encenderán uno o más leds normales.

Botón	Nº Leds encendidos
BTN0	1
BTN1	2
BTN2	3

BTN3	4

Tabla B. 3: Función botones (3) tutorial.

- Config sw 0x11: Al pulsar los diferentes botones, el led RGB se encenderá de diferentes colores según el botón que se pulse, de forma similar a la *Config sw 0x00.* La única diferencia, es que el led se mantendrá encendido durante 1 segundo.

B.1 Diseño hardware en Vivado (Tutorial)

Para comenzar con el diseño crearemos un nuevo proyecto en Vivado. Para ello haremos clic en **Quick Start >> Create Project** en la pantalla inicial de Vivado (*Figura B. 1*)**.**

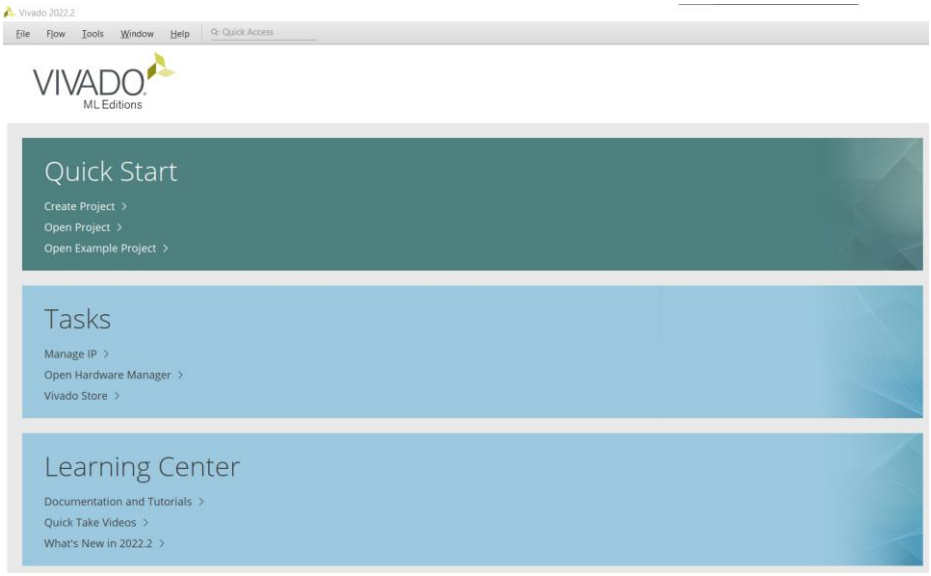

Figura B. 1: Pantalla de inicio de Vivado

Una vez hecho esto, aparecerá el asistente de creación de proyecto de Vivado, donde se le dará un nombre al proyecto, una ubicación y se elegirá el tipo de proyecto (*Figura B. 2*). Se elegirá **RTL Project,** que permitirá crear o añadir un diseño de bloques.

En las siguientes pestañas del asistente (*Figura B. 3*) será posible añadir algún diseño existente y el fichero de restricciones de, y se podrá configurar el lenguaje de programación que se va a emplear tanto para el diseño como para las simulaciones, VHDL en este caso. Esto se selecciona en el campo **Target language** y **Simulator language.**

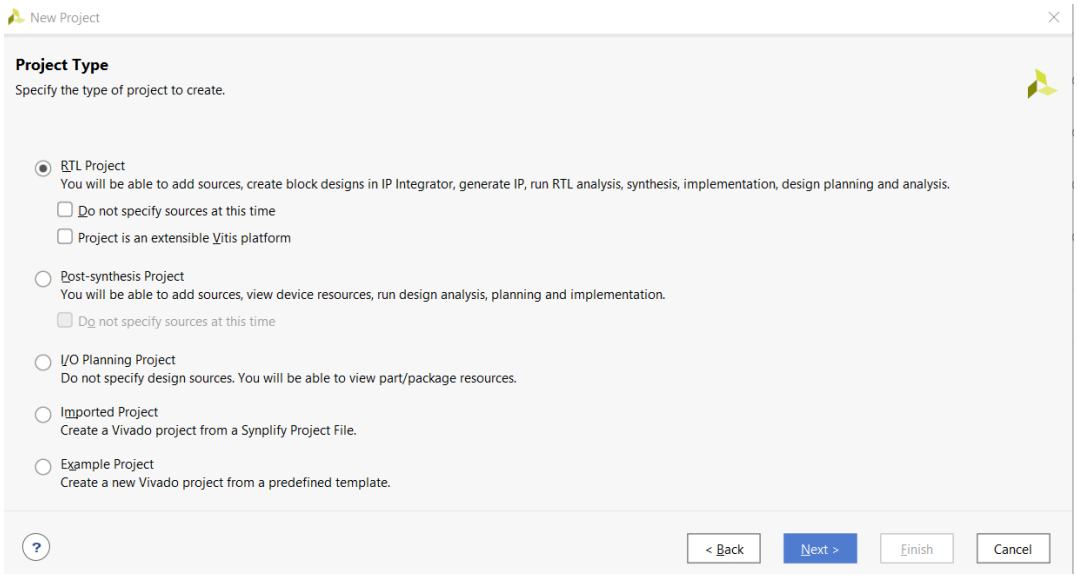

Figura B. 2: Elección de tipo de proyecto en Vivado.

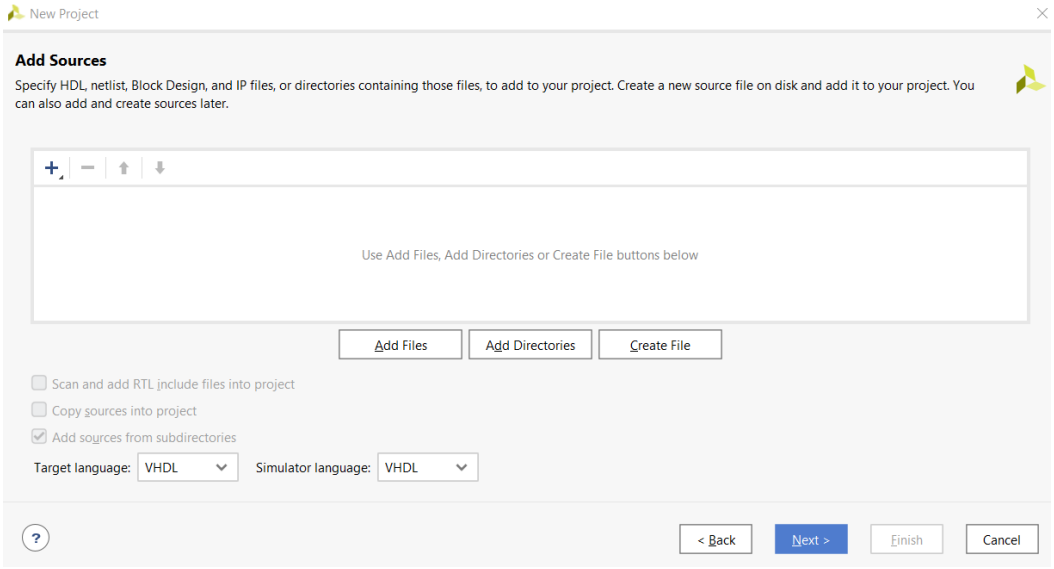

Figura B. 3: Página para configurar lenguaje y añadir fuentes en Vivado.

Seguimos pasando las páginas del asistente haciendo clic en **Next,** hasta llegar a la última página, en la que seleccionaremos la placa a la que está destinada el proyecto. Para ello en el buscador, en la pestaña **Boards** (*Figura B. 4*)**,** escribimos "pynq" y aparecerá nuestra placa PYNQ-Z2. La seleccionamos y pulsamos **Next** y **Finish** en la siguiente página, y el proyecto ya estaría creado.

NOTA: La placa Pynq-Z2 no está en la instalación por defecto de Vivado. Hay que hacerlo manualmente previamente a la creación del proyecto.

Una vez se haya creado el proyecto, en el menú de la izquierda (*Figura B. 5*), seleccionaremos **Create Block Design.** Aparecerá un asistente en el que se podrá añadirle un nombre al diseño.

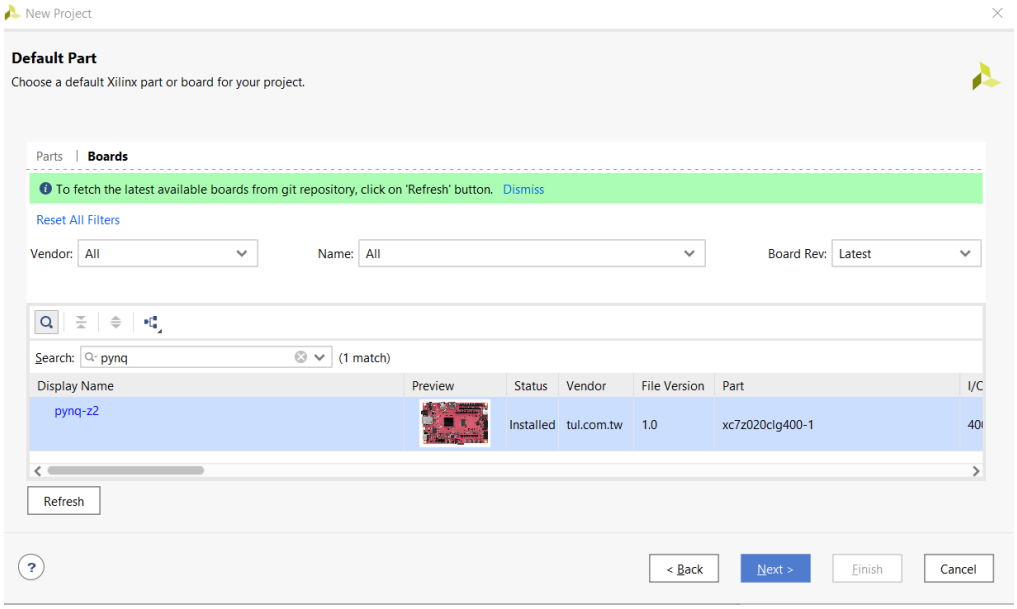

Figura B. 5: Página para elegir placa en Vivado.

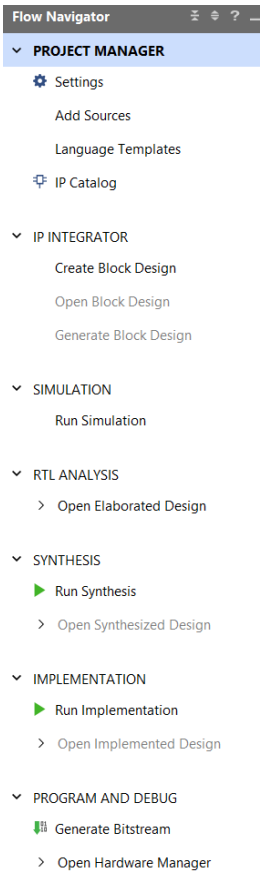

Figura B. 4: Menú izquierdo de Vivado

Al hacer esto, aparecerá una pestaña en blanco, en la que podremos añadir los diferentes bloques IP. Lo primero es añadir el ZYNQ7 que es nuestro microprocesador, y para ello buscamos “Zynq” en el buscador de bloques (marcado con el símbolo “+” en la barra superior de la pestaña creada).

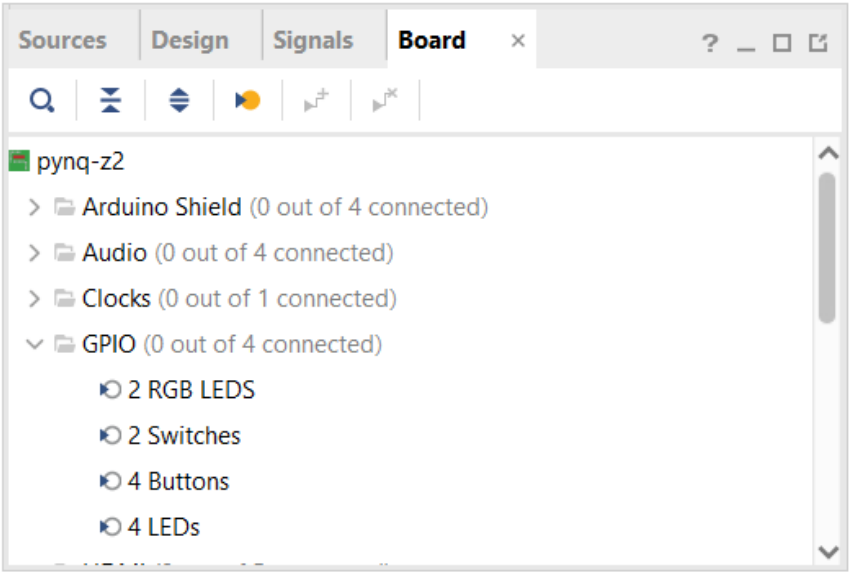

Figura B. 6: Pestaña para añadir bloques IP relacionados con la placa en Vivado.

Posteriormente, se añaden los bloques GPIO, necesarios para controlar los switches, los botones y todos los leds. Esto se hace, haciendo clic en la pestaña **Board** (*Figura B. 6*)**,** y en el apartado **GPIO,** hacemos clic sobre: **2 RGB LEDS, 2 Switches, 4 Buttons y 4 LEDs.** Al hacer clic automáticamente se generarán los bloques ip necesarios para isntanciar estos componentes.

El resultado será el siguiente (Ver *Figura B. 7*):

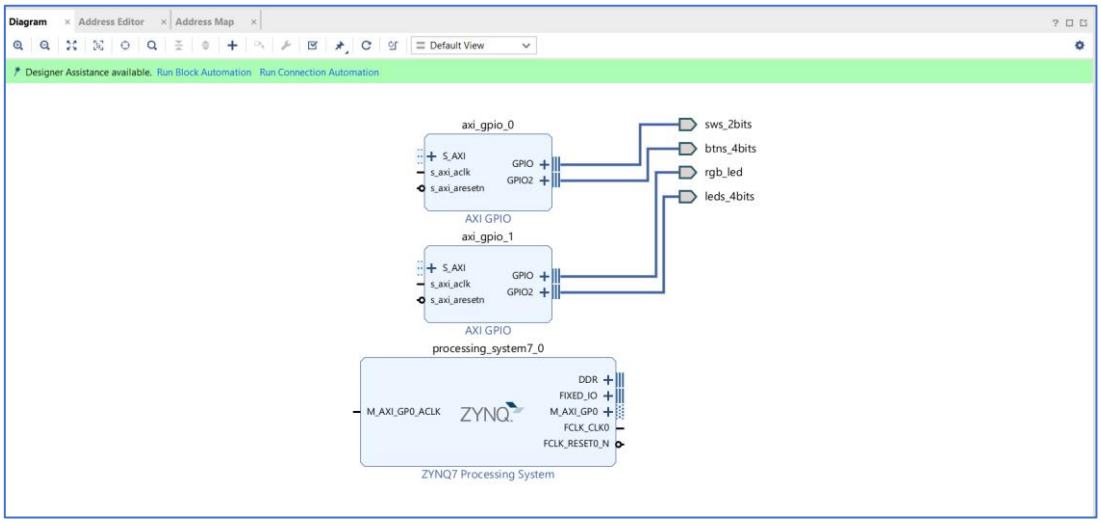

Figura B. 7: Diagrama de bloques previo.

IMPORTANTE: El puerto asociado a los switches (sws_2bits) debe ir conectado al canal 0 del bloque GPIO_0, el puerto asociado a los botones (btns_4bits) debe ir conectado al canal 1 del bloque GPIO_0. El puerto asociado al LED RGB (rgb_led), debe ir asociado al canal 0 del bloque GPIO_1 y el puerto asociado a los 4 LEDs verdes (leds_4bits) debe ir conectado al canal 1 del bloque GPIO_1. Tomar la referencia de la *Figura B. 7*.

Después de tener los 3 bloques en el diagrama, faltaría conectar todos los bloques entre sí. Para ello, pulsamos en **Run Block Automation,** y marcamos la opción "All Automation" (*Figura B. 8*).

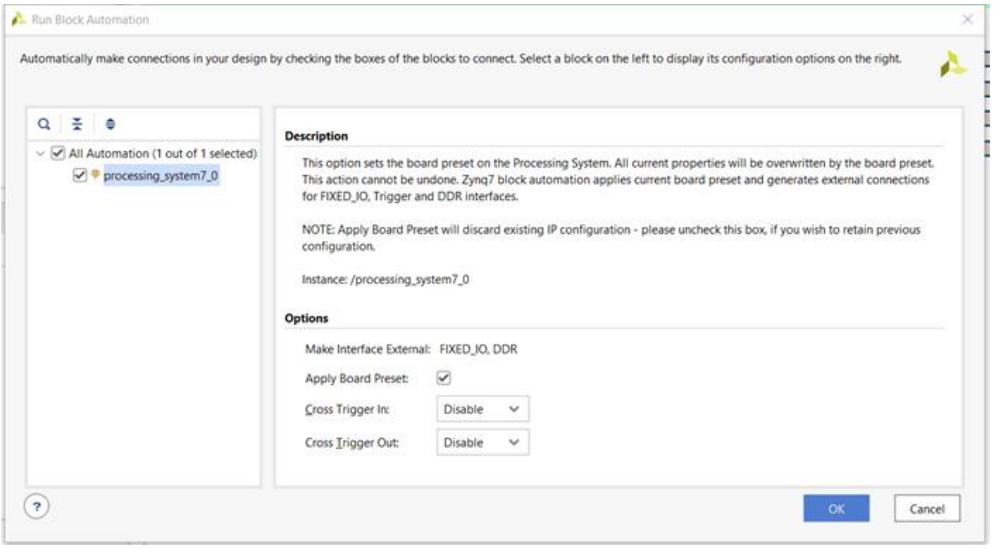

Figura B. 8: Asistente Run Block Automation.

Y a continuación, pulsamos en **Run Connection Automation,** y marcamos de nuevo la opción "All Automation" (*Figura B. 9*).

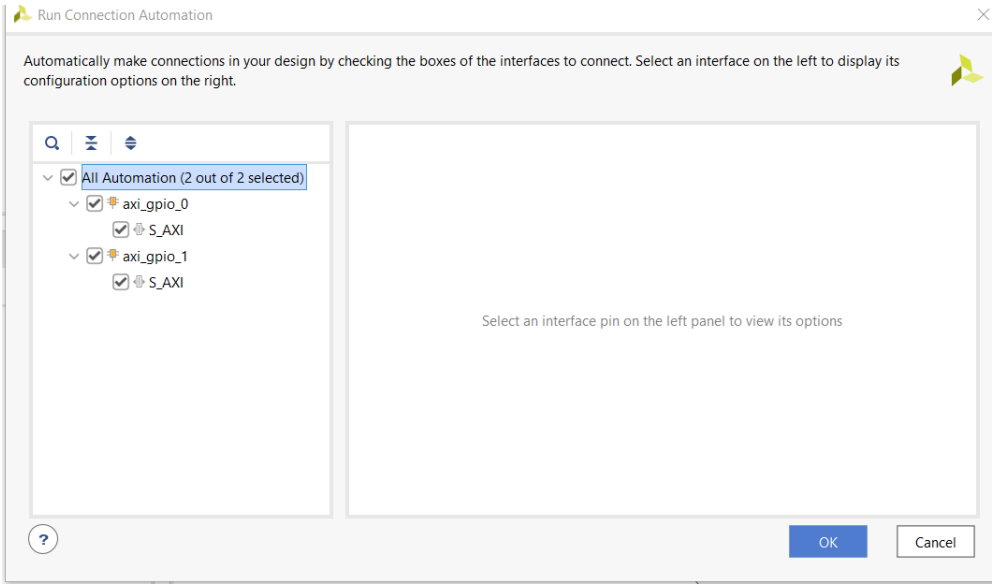

Figura B. 9: Asistente Run Connection Automation

Una vez hecho esto, el diagrama final quedaría como el de la *Figura B. 10*.

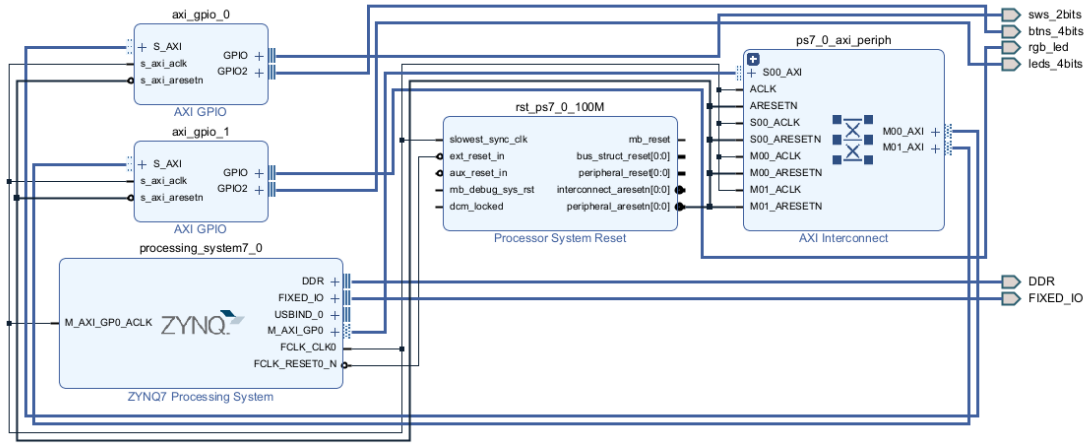

Figura B. 10: Diagrama completo del tutorial.

Ahora hay que añadir el fichero de restricciones de pines, que se puede encontrar en la web: https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html como "Master XDC". Para añadir el fichero, en la pestaña **Sources,** haremos clic en el símbolo "+" y a continuación se abrirá un asistente en el que tendremos que seleccionar la opción **Add or create constraints,** y añadimos el fichero descargado.

Este fichero aparecerá en la pestaña **Sources >> Constraints >> constrs_1.** Antes de editar el fichero de restricciones de pines, cambiaremos los nombres de los puertos de salidas de los leds, botones y switches.

Para ello hacemos doble clic en los puertos del diagrama y aparecerá el siguiente cuadro (*Figura B. 11*):

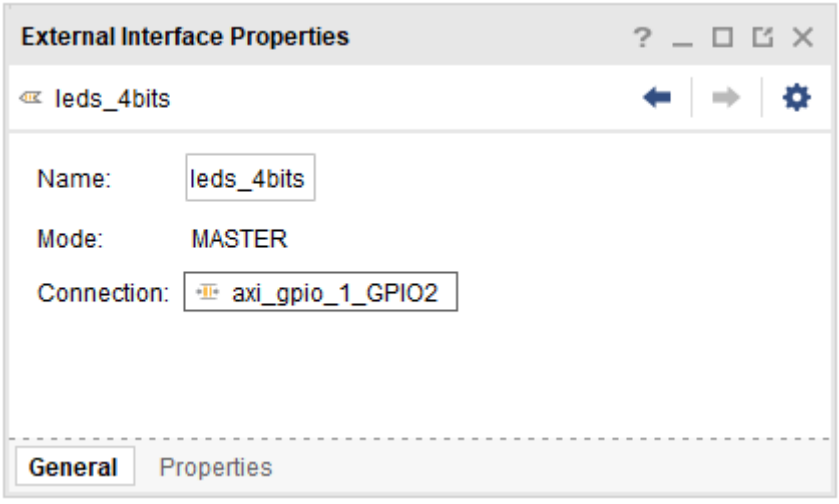

Figura B. 11: Propiedades de puerto (Vivado).

Se cambiarán los nombres acorde a la *Tabla B. 4*:

Nombre predeterminado	Nombre nuevo
sws_2bits	sw
btns_4bits	btns
rgb_led	rgb
leds_4bits	leds

Tabla B. 4: Nombres para los puertos E/S (tutorial).

Una vez hecho esto, abrimos el fichero de restricciones de pines y quitamos los comentarios de las líneas de código referidas a los puertos, tal y como en la *Figura B. 12*:

```
##Switches

set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { sws[0] }]; #IO_L7N_T1_AD2N_35 Sch=sw[0]
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { sws[1] }]; #IO_L7P_T1_AD2P_35 Sch=sw[1]

##RGB LEDs

set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { rgb[2] }]; #IO_L22N_T3_AD7N_35 Sch=led4_b
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { rgb[1] }]; #IO_L16P_T2_35 Sch=led4_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { rgb[0] }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led4_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led5_b }]; #IO_0_35 Sch=led5_b
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { led5_g }]; #IO_L22P_T3_AD7P_35 Sch=led5_g
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led5_r }]; #IO_L23N_T3_35 Sch=led5_r

##LEDs

set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { leds[0] }]; #IO_L6N_T0_VREF_34 Sch=led[0]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { leds[1] }]; #IO_L6P_T0_34 Sch=led[1]
set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { leds[2] }]; #IO_L21N_T3_DQS_AD14N_35 Sch=led[2]
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { leds[3] }]; #IO_L23P_T3_35 Sch=led[3]

##Buttons

set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { btns[0] }]; #IO_L4P_T0_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { btns[1] }]; #IO_L4N_T0_35 Sch=btn[1]
set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { btns[2] }]; #IO_L9N_T1_DQS_AD3N_35 Sch=btn[2]
set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { btns[3] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=btn[3]

##PmodA
```

Figura B. 12: Fichero de restricciones de pines PYNQ-Z2.

Ya tendríamos todo el diseño creado, ahora solo hay que crear el wrapper, haciendo clic derecho en el archivo del diagrama de bloques en la pestaña **Sources,** y **Create HDL Wrapper…** Al hacer esto se creará un archivo por encima del diagrama, y a este archivo hay que generarle el Bitstream y será el que hay que exportar. Seleccionamos el archivo, y en el menú izquierdo de Vivado hacemos clic en **Generate Bitstream.**

Cuando el bitstream esté generado, podremos exportar el diseño en **File >> Export >> Export Hardware...**

Se abrirá otro asistente, y haremos clic en **Include bitstream,** se le dará un nombre al fichero, y se elegirá la ubicación de destino. Este archivo exportado, de extensión ".xsa" se usará en el siguiente apartado, por tanto, es importante recordar en que carpeta se ha guardado.

B.2 Creación del Proyecto de aplicación en Vitis (Tutorial)

Para crear un proyecto de aplicación, abriremos Vitis IDE, que debe descargarse desde la web de Xilinx. Al abrir el programa tendremos que elegir un espacio de trabajo, que será donde se almacenen todos los proyectos creados. Una vez abierto, los pasos son:

- Crear la plataforma hardware
- Crear el proyecto de aplicación

Para crear la plataforma hardware, usaremos el fichero ".xsa" mencionado al final del apartado anterior. Para ello nos vamos al menú superior de Vitis, **File >> New >> Platform Project.** Al hacer esto aparecerá un asistente para crear la plataforma. Primero escribiremos el nombre que le vayamos a asignar a la plataforma, y posteriormente elegimos el fichero base desde el que se va a crear. Para ello en la siguiente página del asistente (*Figura B. 13*), en la pestaña **Create a new platform from hardware (XSA),** hacemos clic en **Browse…** y seleccionamos el fichero ".xsa" creado.

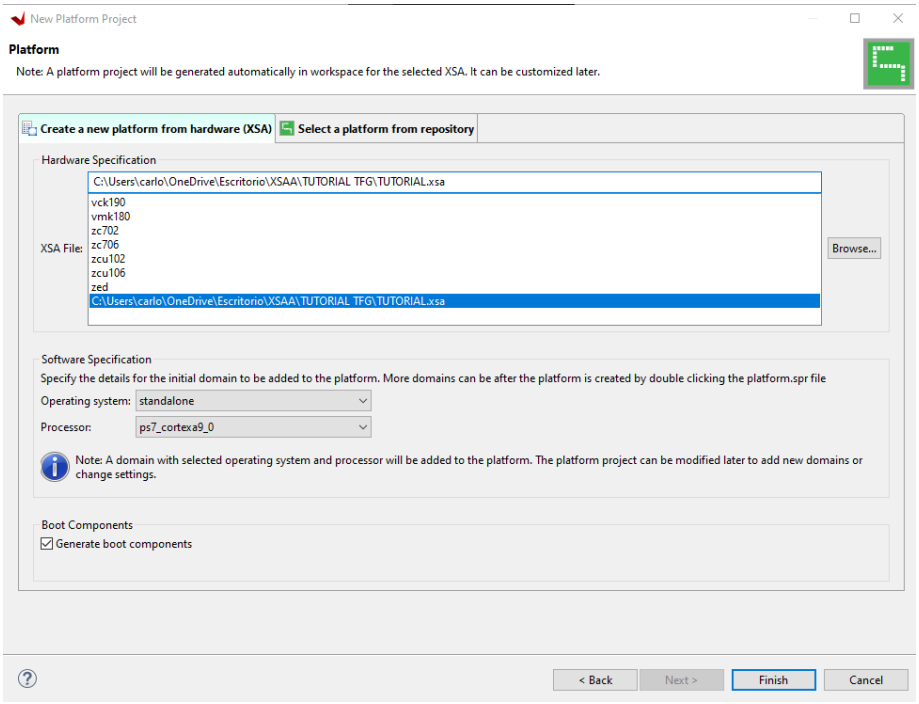

Figura B. 13: Elección de plataforma.

En el campo "Operating system" seleccionaremos **standalone,** lo que quiere decir que no se usará sistema operativo, y que el sistema será independiente. Se hace clic en **Finish** y ya estaría creada la plataforma.

La plataforma aparecerá en el menú izquierdo de Vitis con el texto "(Out of date)" después del nombre. Esto quiere decir que la plataforma no está compilada, por tanto, la seleccionamos, hacemos clic derecho y marcamos **Build Project.** Tras esto, debería de compilar el programa y finalizar sin errores, y el estado "(Out of date)" ya no aparecerá.

Una vez creada la plataforma, ya podemos crear el proyecto de aplicación. Para ello, en la barra superior de nuevo marcamos: **File >> New >> Application Project.** Se mostrará otro asistente (*Figura B. 14*), en el que solo tendremos que escribir el nombre para la aplicación y seleccionar la plataforma base. Vamos a la pestaña **Select a platform from repository,** y elegimos la plataforma que hemos creado. En la siguiente pestaña ponemos un nombre a la aplicación. Pasamos las siguientes páginas haciendo clic en **Next,** hasta llegar a la pestaña **Templates,** donde se eligirá la plantilla “helloworld” que incluye un código de prueba. Ahora, se habrá generado el sistema software, con todos los elementos necesarios, y ya se podría editar el código de nuestro programa.

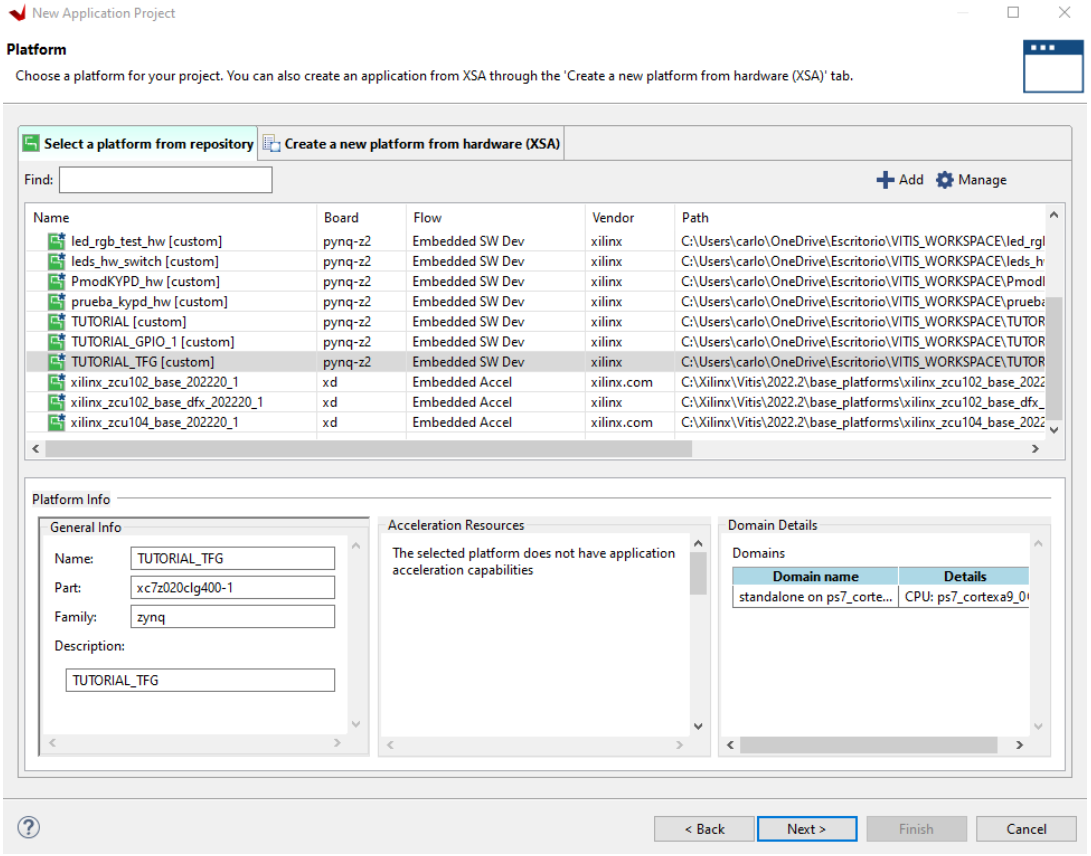

Figura B. 14: Selección de plataforma para la aplicación

Al crear todo, debería aparecer algo similar a lo que muestra la *Figura B. 15.*

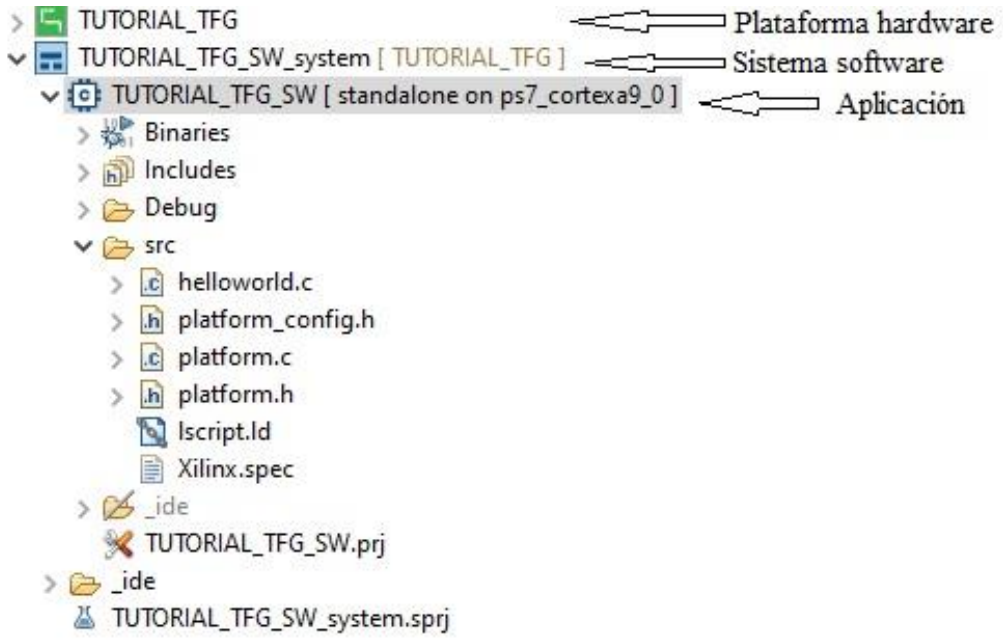

Figura B. 15: Proyecto completo en Vitis.

El programa en C (aplicación), se compone básicamente de 5 secciones de código:

- Librerías
- Definiciones
- Funciones de configuración: Son funciones que se usan para manejar más fácilmente los parámetros asociados a los bloques ip de la plataforma hardware a través de estructuras.
- Funciones externas: Entre ellas están las funciones de inicialización de puertos u otros parámetros, y las funciones complementarias para el programa principal.
- Función main: Esta es la función principal de la aplicación, desde ella se hace una llamada al resto de funciones que se deben ejecutar.

El código del programa será el siguiente:

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
// Librerias incluidas
#include "xparameters.h"
#include "xgpio.h"
#include "sleep.h"

#define R 0x04 //0100
#define G 0x02 //0010
#define B 0x03 //0011

//Declaración de estructura para GPIO
XGpio gpio_bt_sw, gpio_leds;
```

```
void GPIO_BTNS_SWS_Init(){//inicializar GPIO
	int status;
	status = XGpio_Initialize(&gpio_bt_sw, XPAR_AXI_GPIO_0_DEVICE_ID);
	if(status != XST_SUCCESS){
		print("Err: Gpio Initialization failed\n\r");
	}
	else{
		print("Info: Gpio Initialization successful\n\r");
	}
}

void GPIO_LEDS_Init(){//inicializar GPIO
	int status;
	status = XGpio_Initialize(&gpio_leds, XPAR_AXI_GPIO_1_DEVICE_ID);
	if(status != XST_SUCCESS){
		print("Err: Gpio Initialization failed\n\r");
	}
	else{
		print("Info: Gpio Initialization successful\n\r");
	}
}

void ConfigGpio() {//configurar mapeo

	XGpio_SetDataDirection(&gpio_bt_sw, 1, 0);//SWITCHES
	XGpio_SetDataDirection(&gpio_bt_sw, 2, 0);//BUTTONS
	XGpio_SetDataDirection(&gpio_leds, 1, 0);//RGB LEDS
	XGpio_SetDataDirection(&gpio_leds, 2, 0);//LEDS
	XGpio_DiscreteSet(&gpio_leds, 1, 0);
	XGpio_DiscreteSet(&gpio_leds, 2, 0);
}


void runProject(){
	int button, switches;
	while(1){
		switches = XGpio_DiscreteRead(&gpio_bt_sw, 1);
		button = XGpio_DiscreteRead(&gpio_bt_sw, 2);
		switch(button)
		{
		case 0x1://RGB CONFIG 1
			switch (switches){
					case 0x0://0001
						xil_printf("%d: ");
						XGpio_DiscreteWrite(&gpio_leds, 1, R);
						xil_printf("\tRED\n\r");
						break;
					case 0x1://0010
						xil_printf("%d: ");
						XGpio_DiscreteWrite(&gpio_leds, 2, 0x1);
						xil_printf("\tLD0\n\r");
						break;
					case 0x2://0100
						xil_printf("%d: ");
						XGpio_DiscreteWrite(&gpio_leds, 2, 0x1);
						xil_printf("\t1LED\n\r");
						break;
					case 0x3://1000
						xil_printf("%d: ");
						XGpio_DiscreteWrite(&gpio_leds, 1, R);
						xil_printf("\tRED\n\r");
						sleep(1);
```

```
                        break;
                    default://Apagar
                        XGpio_DiscreteClear(&gpio_leds, 1, 0x7);
                        XGpio_DiscreteClear(&gpio_leds, 2, 0xF);
                        break;
            }
            break;
        case 0x2://LED CONFIG 1
            switch (switches){
                    case 0x0://0001
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 1, G);
                        xil_printf("\tGREEN\n\r");
                        break;
                    case 0x1://0010
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 2, 0x2);
                        xil_printf("\tLD1\n\r");
                        break;
                    case 0x2://0100
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 2, 0x3);
                        xil_printf("\t2LEDS\n\r");
                        break;
                    case 0x3://1000
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 1, G);
                        xil_printf("\tGREEN\n\r");
                        sleep(1);
                        break;
                    default://Apagar
                        XGpio_DiscreteClear(&gpio_leds, 1, 0x7);
                        XGpio_DiscreteClear(&gpio_leds, 2, 0xF);
                        break;
            }
            break;
        case 0x4://LED CONFIG 2
            switch (switches){
                    case 0x0://0001
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 1, B);
                        xil_printf("\tBLUE\n\r");
                        break;
                    case 0x1://0010
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 2, 0x4);
                        xil_printf("\tLD2\n\r");
                        break;
                    case 0x2://0100
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 2, 0x7);
                        xil_printf("\t3LEDS\n\r");
                        break;
                    case 0x3://1000
                        xil_printf("%d: ");
                        XGpio_DiscreteWrite(&gpio_leds, 1, B);
                        xil_printf("\tBLUE\n\r");
                        sleep(1);
                        break;
                    default://Apagar
                        XGpio_DiscreteClear(&gpio_leds, 1, 0x7);
                        XGpio_DiscreteClear(&gpio_leds, 2, 0xF);
                        break;
```

```
                }
                break;
        case 0x8://RGB CONFIG 2
                switch (switches){
                        case 0x0://0001
                                xil_printf("%d: ", button);
                                xil_printf("\tWHITE\n\r");
                                XGpio_DiscreteWrite(&gpio_leds, 1, 0x7);
                                break;
                        case 0x1://0010
                                xil_printf("%d: ", button);
                                XGpio_DiscreteWrite(&gpio_leds, 2, 0x8);
                                xil_printf("\tLD3\n\r");
                                break;
                        case 0x2://0100
                                xil_printf("%d: ", button);
                                XGpio_DiscreteWrite(&gpio_leds, 2, 0xF);
                                xil_printf("\t4LEDs\n\r");
                                break;
                        case 0x3://1000
                                xil_printf("%d: ", button);
                                xil_printf("\tWHITE\n\r");
                                XGpio_DiscreteWrite(&gpio_leds, 1, 0x7);
                                sleep(1);
                                break;
                        default://Apagar
                                XGpio_DiscreteClear(&gpio_leds, 1, 0x7);
                                XGpio_DiscreteClear(&gpio_leds, 2, 0xF);
                                break;
                }
                break;
        default:
                XGpio_DiscreteClear(&gpio_leds, 1, 0x7);
                XGpio_DiscreteClear(&gpio_leds, 2, 0xF);
                break;

        }

    }

}


int main()
{
    init_platform();
    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    GPIO_BTNS_SWS_Init();
    GPIO_LEDS_Init();
       ConfigGpio();
    runProject();
    return 0;
}
```

Las funciones `GPIO_BTNS_SWS_Init` y `GPIO_LEDS_Init` son funciones de configuración, la función `ConfigGpio` se usa para configurar los puertos de los GPIO, como entradas y salidas según corresponda. Y la función `runProject` será un bucle infinito, que contendrá la funcionalidad principal del programa. Por último, la función `main`

hace una llamada al resto de funciones necesarias para el programa. Las funciones `init_platform` y `cleanup_platform` se crean automáticamente al generar la plantilla "helloworld.c".

Las funciones "XGpio_..." son propias de la librería "xgpio.h" que viene incluida en la plataforma, al haber usado bloques GPIO de la librería de Vivado en el diseño hardware. Basicamente, las funciones que se usan en este programa, son para escribir en la memoria asociada a los modulos GPIO. Esto permitirá alterar el valor de las salidas de los leds. La función `XGpio_DiscreteRead()` hace justamente lo contrario; lee los valores de la memoria y los almacena en una variable. Esto nos permite conocer el estado de los botones y switches, es decir, leer entradas.

Es fundamental incluir las librerías: `"xparameters.h"`, `"xgpio.h"` y `"sleep.h"`.

- `xparameters`: Esta librería es creada por la plataforma y contiene parámetros de direcciones de memoria y otros datos relacionados con el hardware, que serán de utilidad en los diferentes proyectos.
- `xgpio.h` : Permitirá usar las funciones de lectura y escritura en memoria, para los GPIO.
- `sleep.h`: Permitirá usar la función *sleep,* que crea un delay del tiempo que se le ordene.

Cuando el código esté finalizado, la aplicación se compilará y se depurará para comprobar su correcto funcionamiento. Los pasos a seguir son los siguientes:

- Para compilar la aplicación, hacemos click derecho en el icono azul de la "Aplicación" representado en la *Figura B. 15* y seleccionamos **Build Project**, si el código se ha copiado correctamente, aparecerá en la consola "**Build Finished**" y no aparecerá ningún error.
- Para depurar la aplicación, primero hay que cambiar el jumper (1) a la posición JTAG, después conectamos la placa al PC, a través del cable USB (4), y la encendemos a través del interruptor de alimentación (6). Ver *Figura B. 17.*
- Tras encender la placa, volvemos a Vitis, y hacemos clic derecho en la Aplicación (icono azul mencionado) y seleccionamos **Debug As** >> **Debug configurations.** Se abrirá un asistente, en el que habrá que seleccionar en el menú izquierdo: "**Single Application Debug**". Posteriormente aparecerán varias pestañas en el asistente.
- En la pestaña main (*Figura B. 16*) habrá que elegir la aplicación que se va a depurar en el campo **Project**. Hacemos clic en **Browse** y elegimos el nombre de la aplicación que queremos depurar. En el campo **Configuration** marcamos **Debug,** y hacemos clic en la opción inferior: **Debug.**

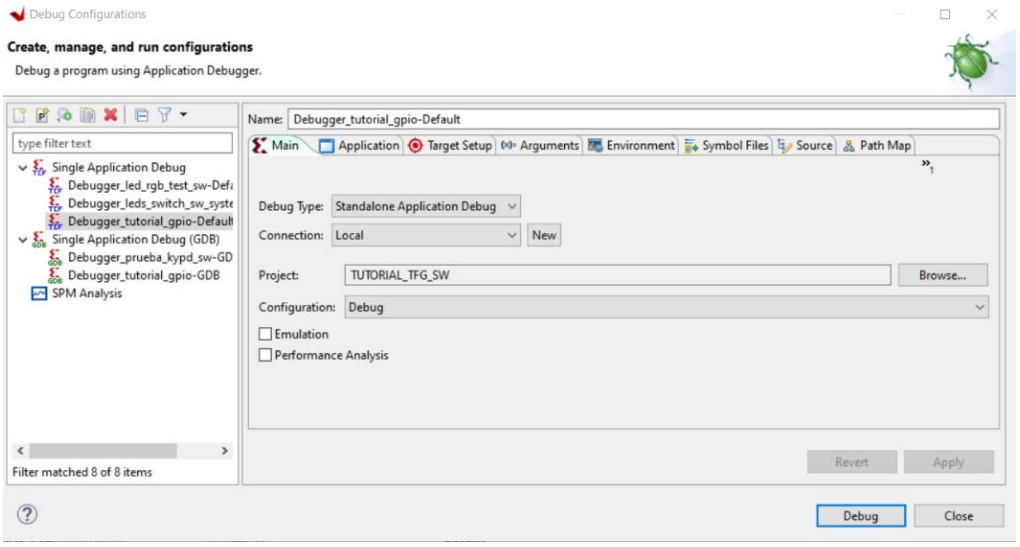

Figura B. 16: Asistente de depuración.

- Se abrirá el modo depuración, y sería hora de probar el programa. Este modo es similar al modo de depuración de cualquier entorno de programación para sistemas embebidos, por lo que no debería suponer mucho problema aprender a usarlo. Si hubiese algún problema, también se puede consultar el apartado 3.8 de la memoria de este proyecto.

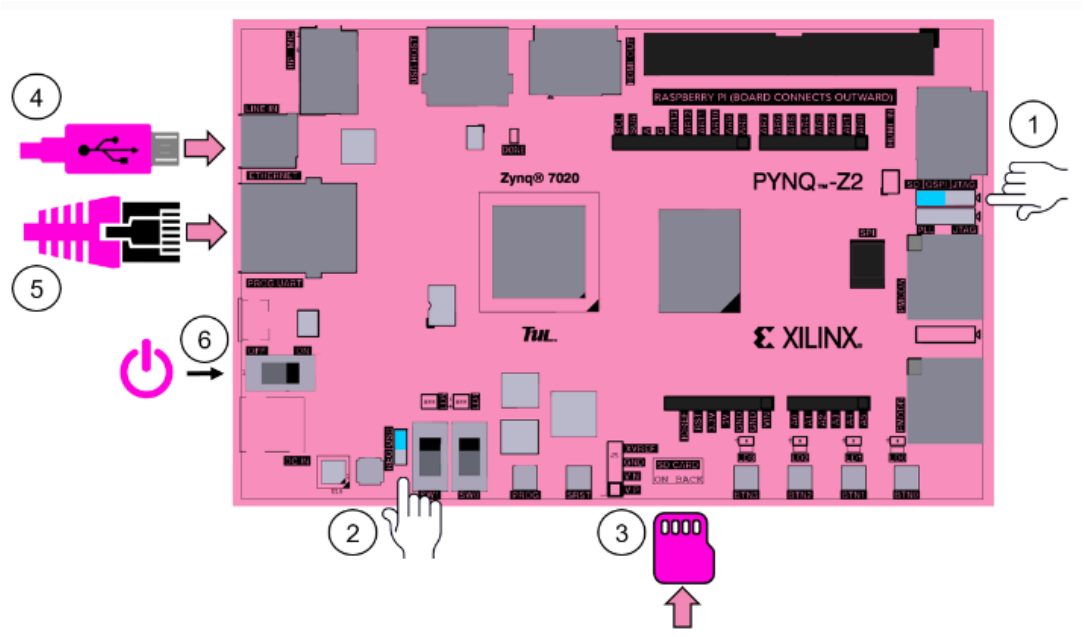

Figura B. 17: Puesta en marcha PYNQ – Z2. Fuente: xilinx.com