System on Chip Lab

Report of the practices
Platform Studio and Vivado

SoC Course ENSEEIHT – December 2019

Summited by:

Juan Valverde
ESECA M2

This lab session aims at designing and implementing some practices in a FPGA SPARTAN3 or in a Nexys 4 DDR in order to:

- Learn how to use the low-level libraries includes on the IDE to program Microblaze microcontroller and their peripherals.
- Use interruptions in order to periodize some fast signals that needs a quick response or analysis.
- Design the systems in IPE, the old IDE for Spartan3 and in the new one IDE for the Nexys 4 DDR.

Platform Studio for SPARTAN3

Practice 1

The hardware is designed in Platform Studio and has the next IP in order to control a set of leds outputs.

We can see all the IPs used to develop the hardware level included the Microblaze, bram block, the data control, the instruction control, the general-purpose input / output, the clock generator and the reset.

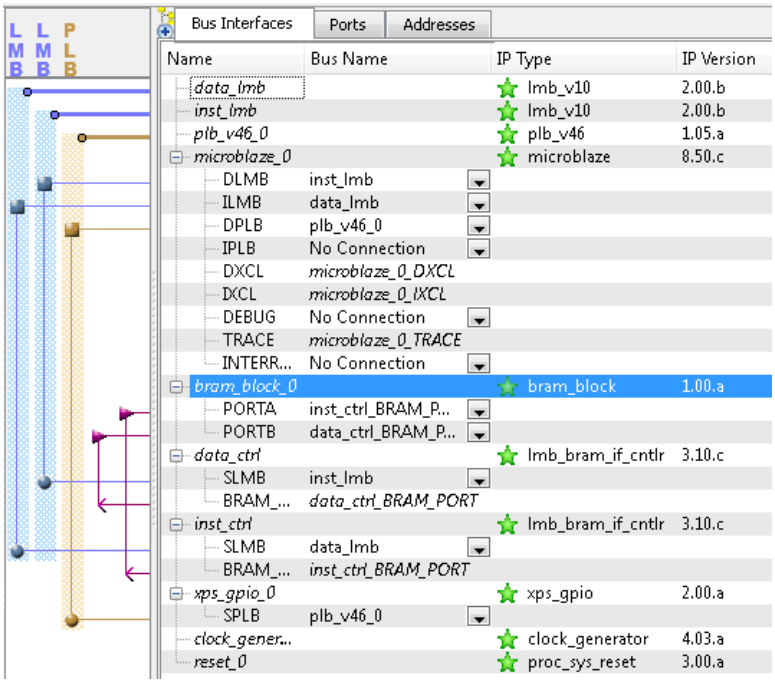


Figure 1 Bus Interfaces

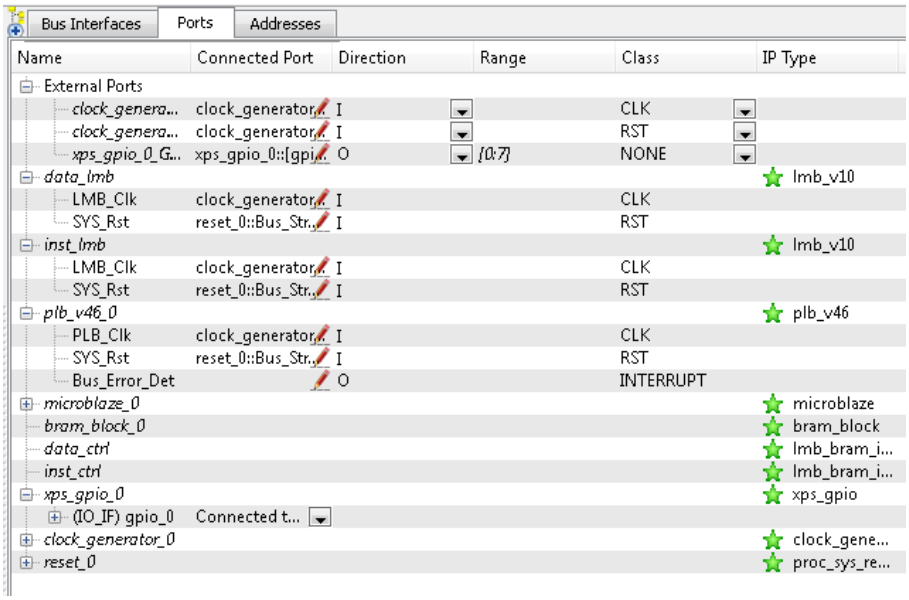


Figure 2. Ports

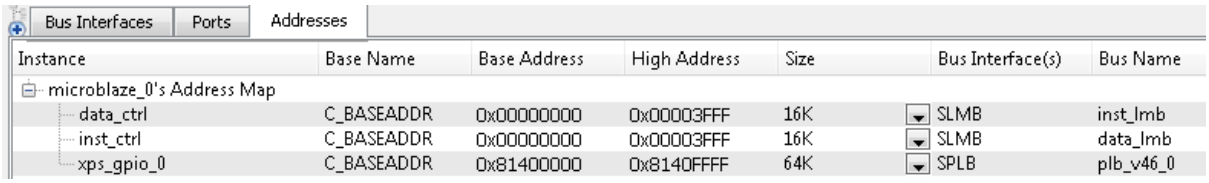


Figure 3. Addresses

The code on the SDK is below:

For the practice 1 I use the simple code in order to control the input and output channels. Next to it I will present the code used to develop this practice. I use simple functions to control the output for the synthetized device:

```
/*
 * led_bliking.c
 *
 *  Created on: 13 nov. 2018
 *      Author: jvalverd
 */

#include "xparameters.h"
#include "xgpio_l.h"

int main(void)
{

    int count =0;

    while (1)
    {

        for (count = 0 ; count < 100000 ; count++)
        {
            XGpio_WriteReg(XPAR_GPIO_0_BASEADDR, XGPIO_DATA_OFFSET, 0x02);
//XGpio_WriteReg(BaseAddress, RegOffset, Data)
        }

        for (count = 0 ; count < 100000 ; count++)
        {
            XGpio_WriteReg(XPAR_GPIO_0_BASEADDR, XGPIO_DATA_OFFSET, 0x00);
//XGpio_WriteReg(BaseAddress, RegOffset, Data)
        }
    }
```

```
  return 0;
}
```

Practice 2

I need to add another GPIO in order to read the switches and write to the leds:

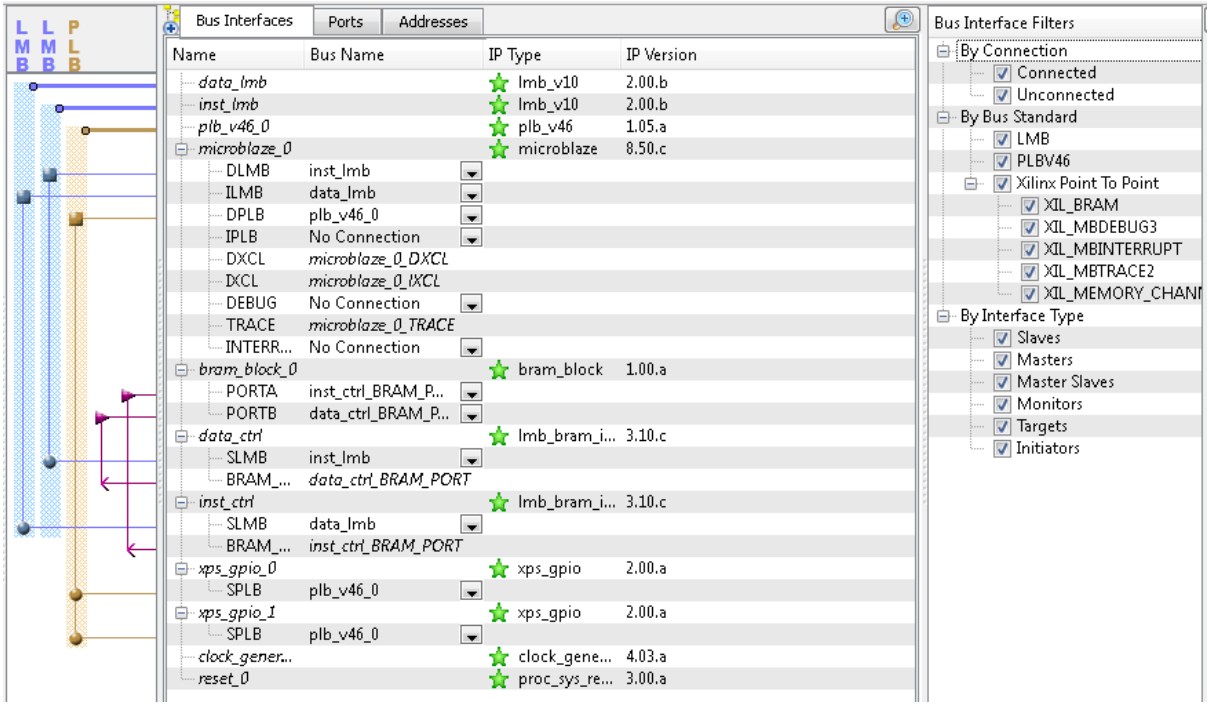


Figure 4 Bus interfaces

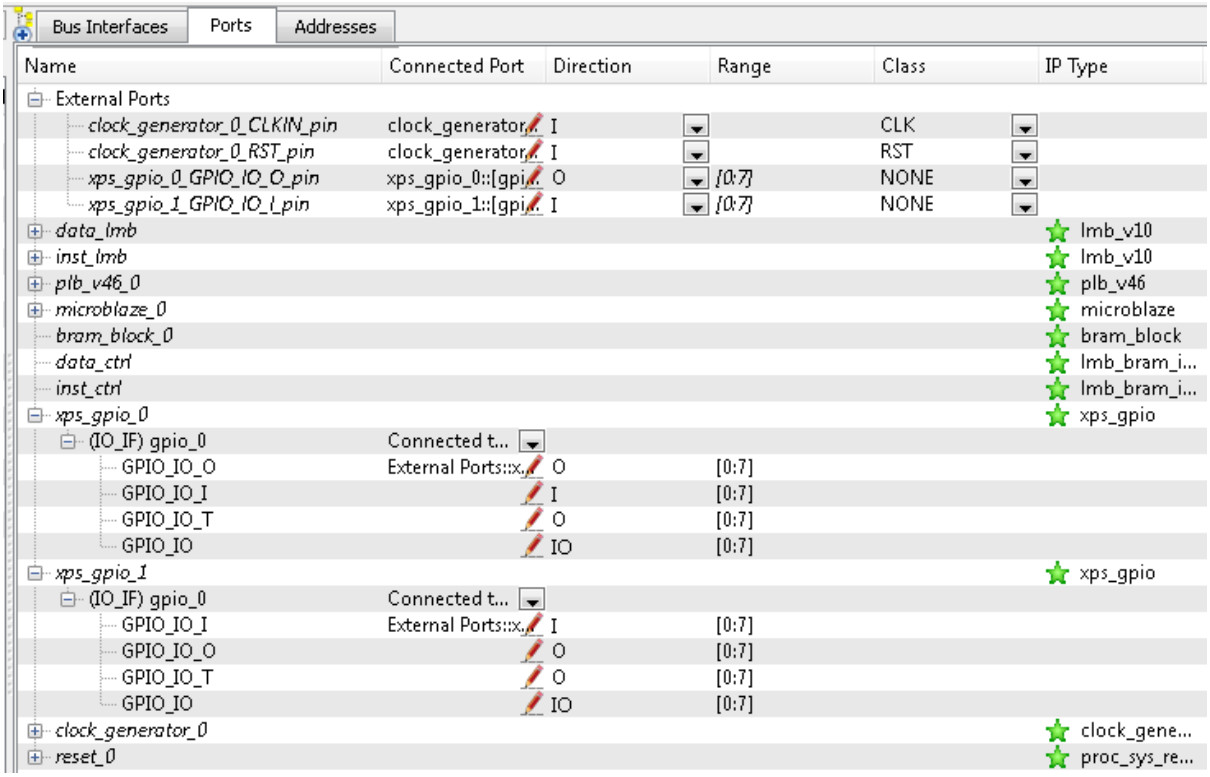


Figure 5 Ports

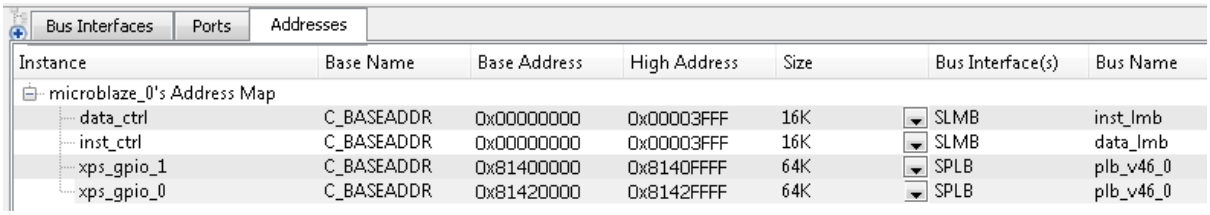


Figure 6. Addresses

For the practice 2 I use more complex functions in order to control the GPIO. Next to it I will present the code used to develop this practice

```
/*
 * led_dir.c
 *
 *  Created on: 13 nov. 2018
 *      Author: jvalverd
 */

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

int main(void)
{
    int flag=0;
    int count=0;
    int count1=1;
    int leds=0;

    XGpio led, sw;  // defined gpio variables

    XGpio_Initialize (&led, XPAR_GPIO_0_DEVICE_ID);
    XGpio_SetDataDirection (&led,1,0); // set display as output ports

    XGpio_Initialize (&sw,XPAR_GPIO_1_DEVICE_ID);
    XGpio_SetDataDirection (&sw,1,1); // set digit as input ports
```

```
    Xuint8 data = 0;
    Xuint8 data0 = 0;

    data0=XGpio_DiscreteRead(&sw, 1);
    while (1)
    {
        data=XGpio_DiscreteRead(&sw, 1);

        /*if((data)==data0)
        {
            for (count = 0 ; count < 100000 ; count++)
            {
            }

            count1=count1+count1;
            if((count1)>129)
            {
                count1=1;
            }
            XGpio_DiscreteWrite(&led, 1, data);
        }
        else
        {
            for (count = 0 ; count < 100000 ; count++)
            {
                XGpio_DiscreteWrite(&led, 1, 0x00);
            }
        } */


        XGpio_DiscreteWrite(&led, 1, data);

    }

  return 0;
}
```

Practice 3

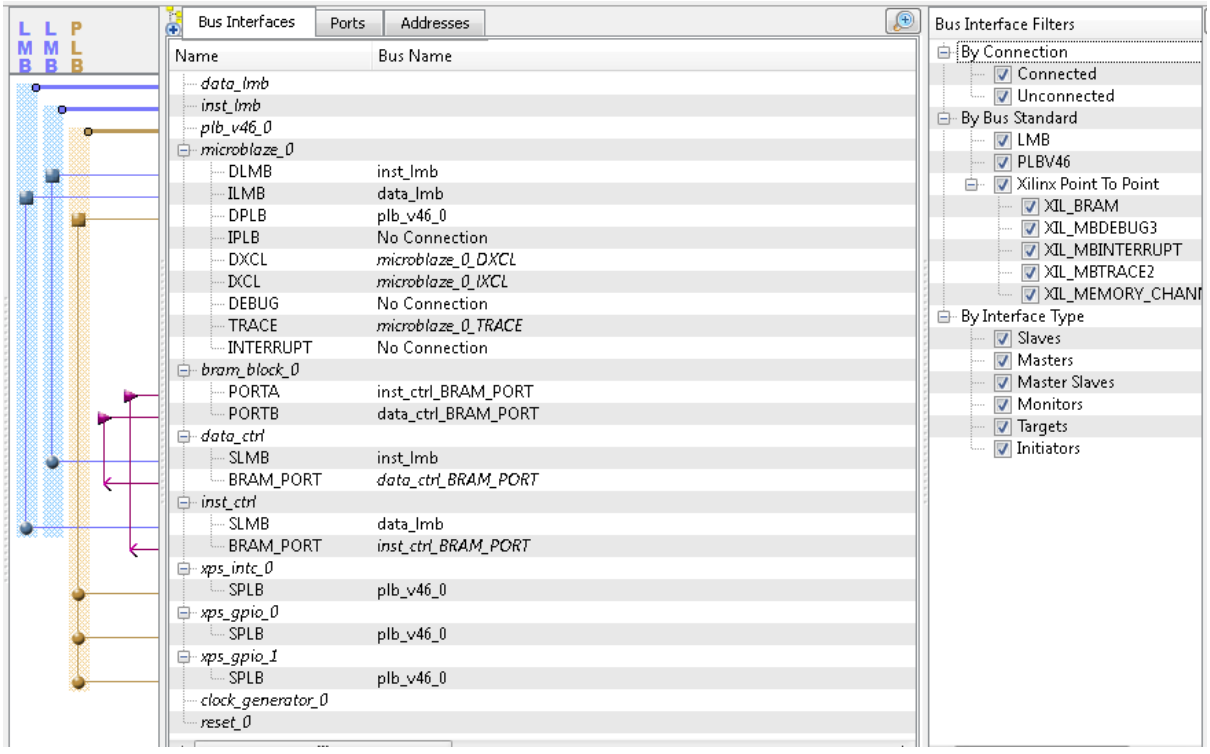


Figure 7. Bus Interfaces

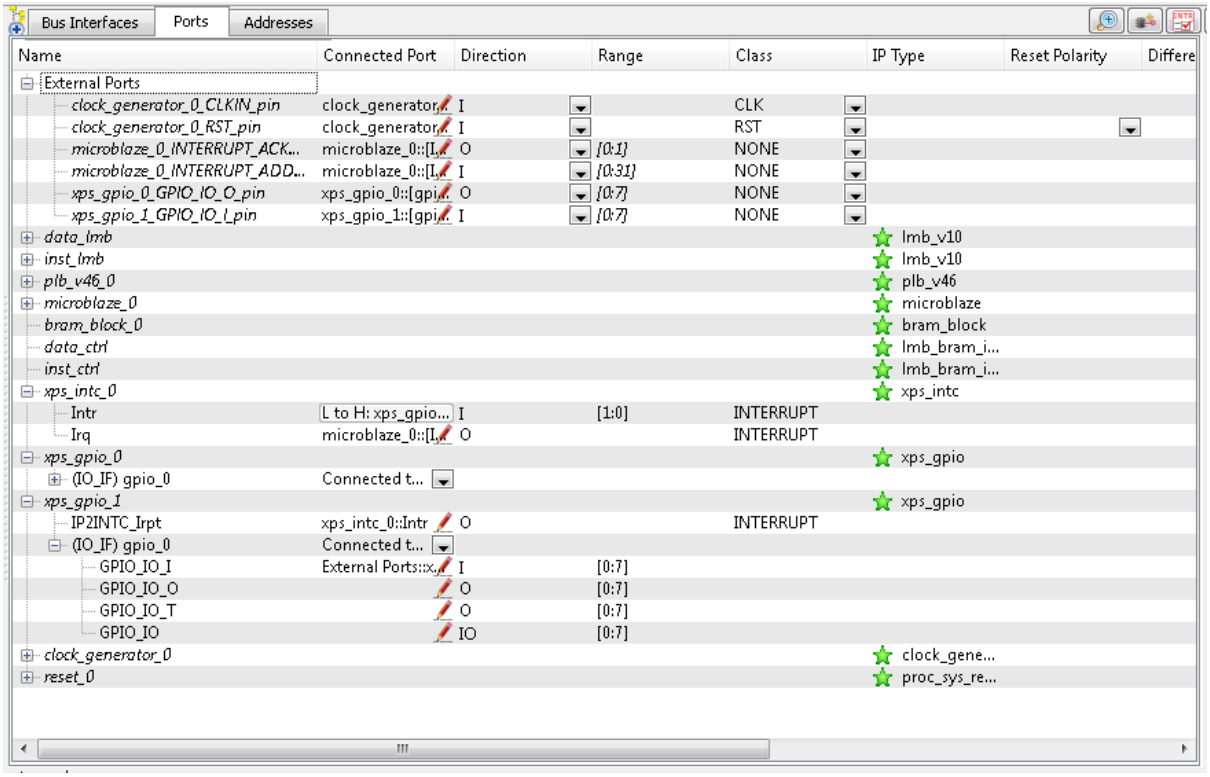


Figure 8. Ports

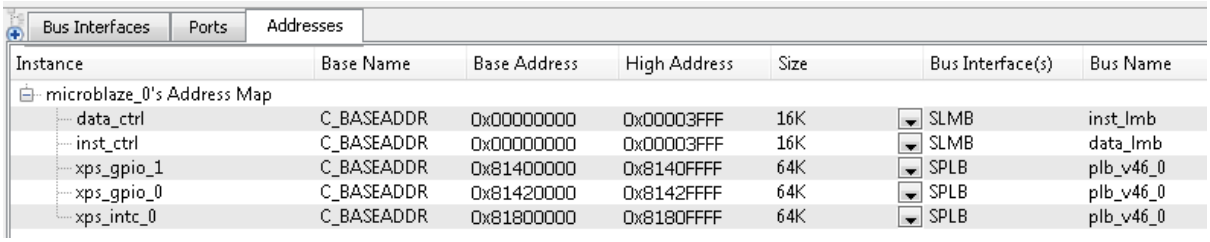


Figure 9. Addresses

For the practice 2 I use the next code in order to control the input and output channels with interruption from the push buttons.

```
#include "xparameters.h"
#include "xgpio.h"
#include "xintc.h"

XGpio Switches, Leds;
XIntc Interrupt;
int inputs;

const int switchs_ch = 1;
const int leds_ch    = 1;

void SwitchInterruptHandler(void *pointer)
{
  //XGpio_InterruptDisable(&Switches, XGPIO_IR_CH1_MASK);
  inputs = XGpio_DiscreteRead(&Switches, switchs_ch);
  XGpio_DiscreteWrite(&Leds, leds_ch, inputs);
  XGpio_InterruptClear(&Switches, XGPIO_IR_CH1_MASK);
  //XGpio_InterruptEnable(&Switches, XGPIO_IR_CH1_MASK);
}
```

```
void SoC_Configuration()
{
  // Initialize Level 1 of IO Interfaces
  XGpio_Initialize(&Switches, XPAR_GPIO_0_DEVICE_ID);
  XGpio_Initialize(&Leds, XPAR_GPIO_1_DEVICE_ID);
  // Enable Interruptions in GPIO Devices
  XGpio_InterruptEnable(&Switches, XGPIO_IR_CH1_MASK);
  XGpio_InterruptGlobalEnable(&Switches);
  // Initialize Interruption on Interruption Controller
  XIntc_Initialize(&Interrupt, XPAR_XPS_INTC_0_DEVICE_ID);//XPAR_INTC_CONTROLLER_DEVICE_ID
  XIntc_SelfTest(&Interrupt);
  // Define Interruption function and start functioning of interruption
  XIntc_Connect(&Interrupt, XPAR_XPS_INTC_0_XPS_GPIO_1_IP2INTC_IRPT_INTR,
(XInterruptHandler)SwitchInterruptHandler, (void
*)0);//XPAR_INTC_CONTROLLER_GPIO_SWITCHES_IP2INTC_IRPT_INTR
  XIntc_Start(&Interrupt, XIN_REAL_MODE);
  XIntc_Enable(&Interrupt,
XPAR_XPS_INTC_0_XPS_GPIO_1_IP2INTC_IRPT_INTR);//XPAR_INTC_CONTROLLER_GPIO_SWITCHES_IP2INTC_IRP
T_INTR
  // Define direction of inputs and outputs
  XGpio_SetDataDirection(&Switches, switchs_ch, 1);
  XGpio SetDataDirection(&Leds, leds ch, 0);
  // Enable Microblaze interruptions
  microblaze_enable_interrupts();
}

int main()
{
  SoC_Configuration();
  // PROGRAM START
  while(1){


  }
  return 0;
}
```

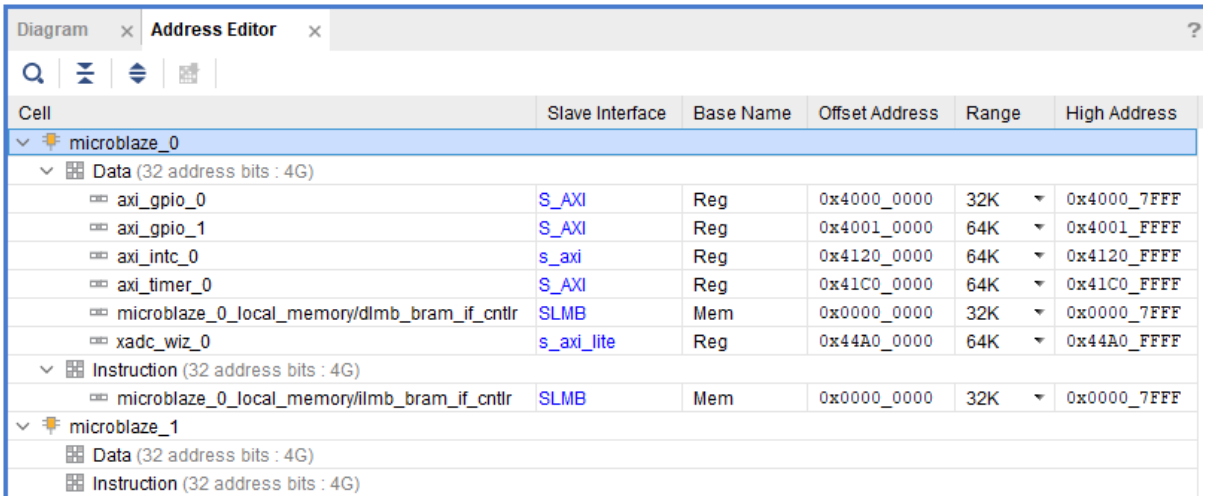
VIVADO

For the Nexys 4 DDR part, we need to use VIVADO IDE in order to syntheses the hardware and program de firmware.

I do all the practices in one design in order to simplify the explanation.

BLOCK DESIGN

First, I need to design the hardware, I need to put the next IPs on the Block Design Editor:

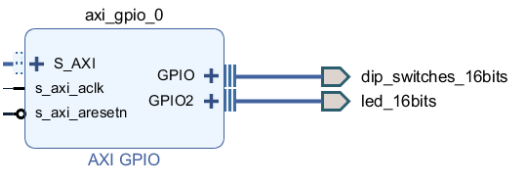


Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x4000_0000	32K	0x4000_7FFF
axi_gpio_1	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
axi_intc_0	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
xadc_wiz_0	s_axi_lite	Reg	0x44A0_0000	64K	0x44A0_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
microblaze_1					
Data (32 address bits : 4G)					
Instruction (32 address bits : 4G)					

microblaze_0

When I choose this IP and place into the Block Editor, a wizard gives me some options in order to configure the system, MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are added and connected as needed. A preset MicroBlaze configuration can also be selected. Here we configure the local and the instruction memory.

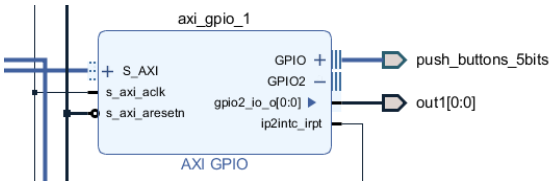
axi_gpio_0



This block is an IP AXI General Purpose Input/output (GPIO) core provides a general-purpose input/output interface to the AXI interface. We have two buses, one for the 16 dip switch inputs and another for the 16 leds outputs.

With this block the first practice was developed.

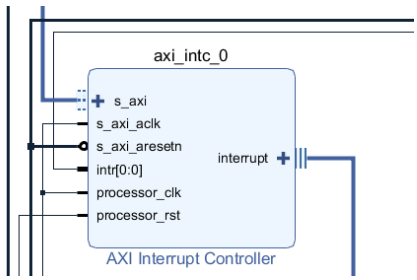
axi_gpio_1



This block is an IP AXI General Purpose Input/output (GPIO). We have two connections, one for the 5 push buttons inputs and another for the one led output.

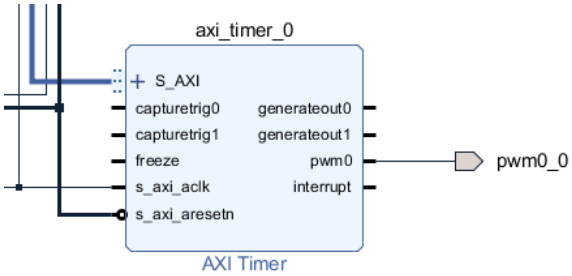
With this block the second practice was developed in order to make interruptions I use the AXI_INTC like an interface between the Micro Blaze and the GPIO module interrupt part.

axi_intc_0



The AXI Interrupt Controller (INTC) core receives multiple interrupt inputs from peripheral devices and merges them into an interrupt output to the system processor. The registers used for storing interrupt vector addresses, checking, enabling and acknowledging interrupts are accessed through the AXI4-Lite interface.

axi_timer_0

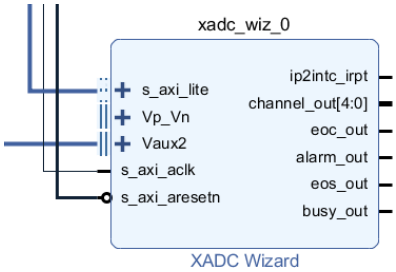


The IP AXI Timer/Counter is a 32/64-bit timer module that interfaces to the AXI4-Lite interface. This module has activated the Timer1 and the Timer2 in order to use on the last practice in PWM mode to control a servomotor.

I use this block in order to understand the configuration and the SDK interface programing, I think that I could use my own IP or I need to use another interface, but for student purposes, it is enough because my objective is learn the basics about VIVADO and next I could move to the advanced level.

If I was working on a company probably, I choose an IP from this company or I will need to talk with the developer boss in order to understand the objectives of this project.

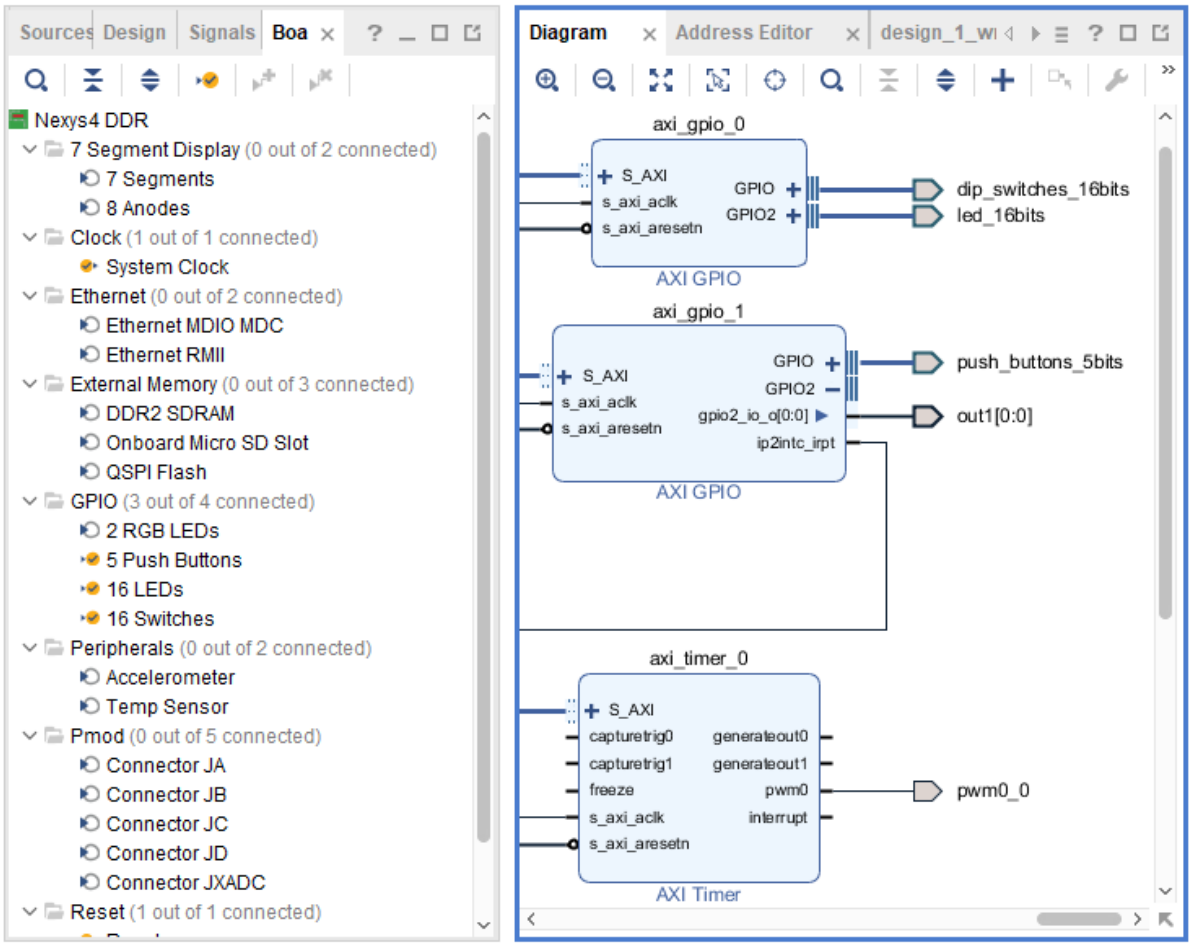
xadc_wiz_0



The IP Analog-to-Digital Converter (XADC) Wizard generates an HDL wrapper to configure the XADC primitive for user-specified external channels, internal sensor channels, modes of operation, and alarms. This module has activated the single channel: auxiliary channel 2 (VAUX2P and VAUX2N) in continuous mode in order to control on the last practice the PWM high period of the signal to control a servomotor.

I use the library supplied by Digilent in order to simplify the connections (Implementation) from the synthetized device to the Nexys 4 DDR pins. The figure below shows the library used.

I

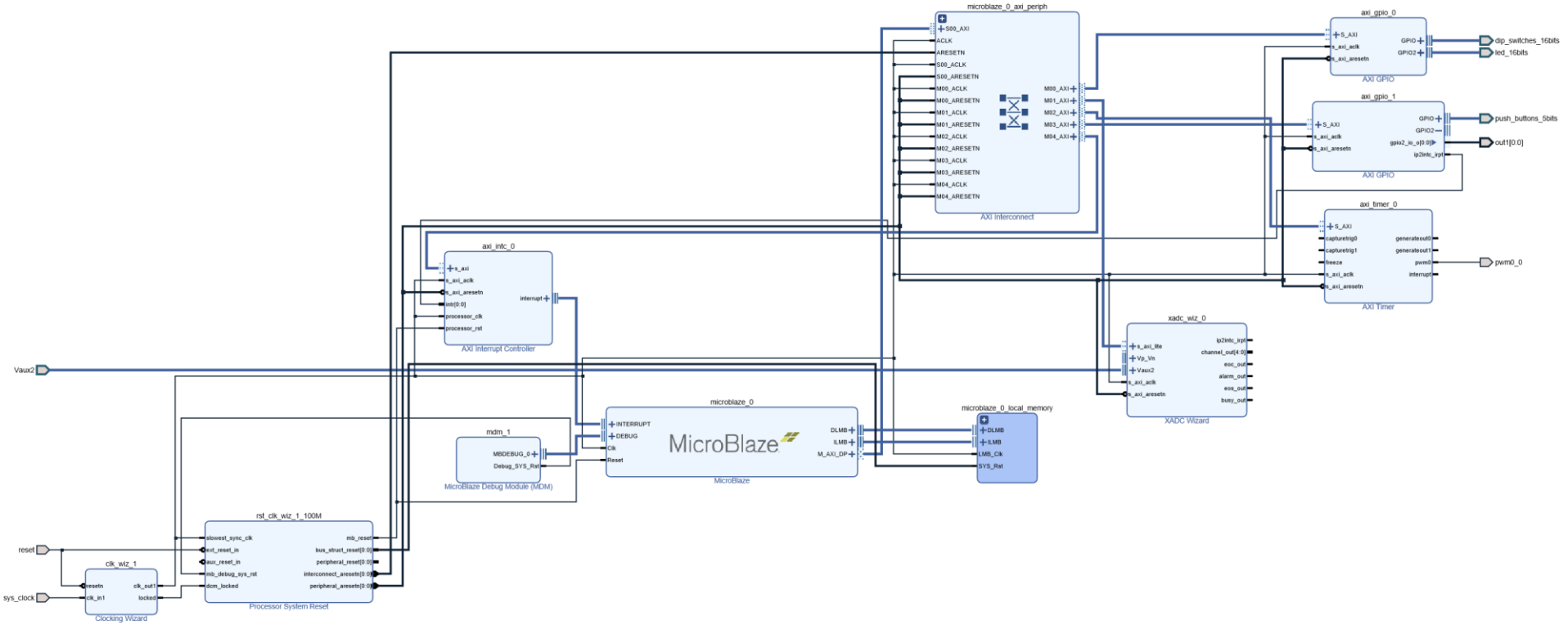


For other connections like the PWM signal, the interruption signal and the ADC signal, I use the constrains in order to configure the implementation:

```
set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { pwm0_0 }];
set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { out1 }];

set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { Vaux2_v_n }];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports { Vaux2_v_p }];
```

The complete block diagram is on the figure below, we can see the different type of connections, the order and the configurations (ports activates, and ports disables, etc.).


microblaze_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M01_ACLK
M01_ARESETN
M02_ACLK
M02_ARESETN
M03_ACLK
M03_ARESETN
M04_ACLK
M04_ARESETN
M00_AXI
M01_AXI
M02_AXI
M03_AXI
M04_AXI
AXI Interconnect
axi_gpio_0
S_AXI
s_axi_aclk
s_axi_aresetn
GPIO
GPIO2
AXI GPIO
dip_switches_16bits
led_16bits
axi_gpio_1
gpio2_io_o[0:0]
ip2intc_irpt
push_buttons_5bits
out1[0:0]
axi_timer_0
capturetrig0
capturetrig1
freeze
generateout0
generateout1
pwm0
interrupt
AXI Timer
pwm0_0
axi_intc_0
s_axi
irq[0:0]
processor_clk
processor_rst
AXI Interrupt Controller
xadc_wiz_0
s_axi_lite
Vp_Vn
Vaux2
ip2intc_irpt
channel_out[4:0]
eoc_out
alarm_out
eos_out
busy_out
XADC Wizard
Vaux2
microblaze_0
INTERRUPT
DEBUG
Clk
Reset
DLMB
ILMB
M_AXI_DP
MicroBlaze
MicroBlaze
microblaze_0_local_memory
DLMB
ILMB
LMB_Clk
SYS_Rst
mdm_1
MBDEBUG_0
Debug_SYS_Rst
MicroBlaze Debug Module (MDM)
rst_clk_wiz_1_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
reset
clk_wiz_1
resetn
clk_in1
clk_out1
locked
Clocking Wizard
sys_clock

SDK:

In order to program the MicroBlaze IP, we need to export the hardware to the SDK and create the application project to configure and program the firmware that can control the different IP synthetized on the Nexys 4 DDR.

In the C code, I need to add a library for each IP used on hardware, we need to take reference on the library xparameters.h in order to use the constants included here, because this file contains the memory directions to each module.

I need to read the Xilinx documentations in order to know the basics principles to program this type of microcontrollers.

Practice 1

For the practice 1 I use the simple code in order to control the input and output channels. Nest to it I will present the code used to develop this practice.

Libraries:

I need to include these libraries in order to use the configuration and control functions on the code.

```
#include "xgpio.h"  //AXI GPIO driver
```

Configuration Functions:

This function helps me to configure the IP how I want:

```
    XGpio_Initialize(&gpio0, XPAR_GPIO_0_DEVICE_ID);
    XGpio_SetDataDirection(&gpio0, 2, 0x00000000); // set LED GPIO channel tristates to All
Output
    XGpio_SetDataDirection(&gpio0, 1, 0xFFFFFFFF); // set BTN GPIO channel tristates to All
Input
```

Control Functions:

These functions helps me to control the IP:

```
btn = XGpio_DiscreteRead(&gpio0, 1);
XGpio_DiscreteWrite(&gpio0, 2, led);
```

Practice 2

For the practice 2 I use the next code in order to control the input and output channels with interruption from the push buttons.

Libraries:

I need to include these libraries in order to use the configuration and control functions on the code.

```
#include "xgpio.h"  //AXI GPIO driver

#include "xintc.h" //XINTC driver
```

Configuration Functions:

This function helps me to configure the IP how I want, these includes the interrupt configuration for the module:

```
XGpio_Initialize(&gpio1, XPAR_GPIO_1_DEVICE_ID);
    XGpio_SetDataDirection(&gpio1, 2, 0x00000000); // set LED GPIO channel tristates to All
Output
    XGpio_SetDataDirection(&gpio1, 1, 0xFFFFFFFF); // set BTN GPIO channel tristates to All
Input

    XGpio_InterruptEnable(&gpio1, XGPIO_IR_CH1_MASK);
    XGpio_InterruptGlobalEnable(&gpio1);
    // Initialize Interruption on Interruption Controller
    XIntc_Initialize(&Interrupt, XPAR_INTC_0_DEVICE_ID);//XPAR_INTC_CONTROLLER_DEVICE_ID
    XIntc_SelfTest(&Interrupt);
    // Define Interruption function and start functioning of interruption
    XIntc_Connect(&Interrupt, XPAR_AXI_INTC_0_AXI_GPIO_1_IP2INTC_IRPT_INTR,
(XInterruptHandler)SwitchInterruptHandler, (void
*)0);//XPAR_INTC_CONTROLLER_GPIO_SWITCHES_IP2INTC_IRPT_INTR
    XIntc_Start(&Interrupt, XIN_REAL_MODE);
    XIntc_Enable(&Interrupt,
XPAR_AXI_INTC_0_AXI_GPIO_1_IP2INTC_IRPT_INTR);//XPAR_INTC_CONTROLLER_GPIO_SWITCHES_IP2INTC_IRP
T_INTR

    microblaze_enable_interrupts();// Enable Microblaze interruptions
```

Control Functions:

This function helps me to control the IP:

For the GPIO interruption I use this function:

```
void SwitchInterruptHandler(void *pointer)

XGpio_InterruptClear(&gpio1, XGPIO_IR_CH1_MASK);// Clear the interrupt flag
```

Practice 3

For the practice 3 I use the XADC and the XTMR in order to control the position of a servomotor, the XADC reads a 12 bits value and I need to change it for to use in a fixed period of 20mS for the PWM and a variable high Pulse duration from 0.9mS to 2.1mS with 1.5mS as center. The next simple code is explained in order to control the IPs.

Libraries:

I need to include these libraries in order to use the configuration and control functions on the code.

```
#include "xsysmon.h" //XADC WIZ driver
#include "xtmrctr.h" //XTMR driver
```

Configuration Functions:

This function helps me to configure the IP how I want:

For the XADC configured in simple channel and continuous mode:

```
    //XADC
    u32 VccPdroRawData;
    XSysMon_Config *SysMonConfigPtr;
    XSysMon *SysMonInstPtr = &SysMonInst;
    SysMonConfigPtr = XSysMon_LookupConfig(XPAR_SYSMON_0_DEVICE_ID);

    XSysMon_CfgInitialize(SysMonInstPtr, SysMonConfigPtr,SysMonConfigPtr->BaseAddress);
    XSysMon_GetStatus(SysMonInstPtr); // Clear the old status
```

For the XTMR configured in 32-bit PWM mode:

- Configured in count down I need to use this formula:
-

```
PWM_PERIOD = (TLR0 + 2) * AXI_CLOCK_PERIOD
PWM_HIGH_TIME = (TLR1 + 2) * AXI_CLOCK_PERIOD
```

Where:

$$TLR0 = \frac{PWM\_PERIOD}{AXI\_CLOCK\_PERIOD} - 2 \approx \frac{20mS}{100M^{-1}} = 2M$$

```
    //XTMR
    XTmrCtr TimerInstancePtr;
    XTmrCtr_Initialize(&TimerInstancePtr,XPAR_TMRCTR_0_DEVICE_ID);

    XTmrCtr_SetOptions(&TimerInstancePtr, 0,
XTC_ENABLE_ALL_OPTION|XTC_DOWN_COUNT_OPTION|XTC_AUTO_RELOAD_OPTION);
//XTmrCtr_SetOptions(&TimerInstancePtr, 0, (XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION |
XTC_DOWN_COUNT_OPTION));

    u32 CounterControlReg = Xil_In32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[0] +
XTC_TCSR_OFFSET);
    CounterControlReg = CounterControlReg | XTC_CSR_ENABLE_PWM_MASK |
XTC_CSR_EXT_GENERATE_MASK;
    Xil_Out32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[0] + XTC_TCSR_OFFSET,
CounterControlReg);

    XTmrCtr_SetOptions(&TimerInstancePtr, 1,
XTC_ENABLE_ALL_OPTION|XTC_DOWN_COUNT_OPTION|XTC_AUTO_RELOAD_OPTION);
//XTmrCtr_SetOptions(&TimerInstancePtr, 1, (XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION |
XTC_DOWN_COUNT_OPTION));

    CounterControlReg = Xil_In32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[1] +
XTC_TCSR_OFFSET);
    CounterControlReg = CounterControlReg | XTC_CSR_ENABLE_PWM_MASK |
XTC_CSR_EXT_GENERATE_MASK;
    Xil_Out32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[1] + XTC_TCSR_OFFSET,
CounterControlReg);

    XTmrCtr_SetResetValue(&TimerInstancePtr, 0, 2000000);
//XTmrCtr_SetResetValue(&TimerInstancePtr, 0, 0x5f5e100);
    XTmrCtr_SetResetValue(&TimerInstancePtr, 1, 150000);
//XTmrCtr_SetResetValue(&TimerInstancePtr,1, 0x1f78a40);

    XTmrCtr_Start(&TimerInstancePtr,0);
    XTmrCtr_Start(&TimerInstancePtr,1);
```

Control Functions:

These functions helps me to control the IP:

For read the XADC configured in simple channel and continuous mode:

```
VccPdroRawData = XSysMon_GetAdcData(SysMonInstPtr,XSM_CH_AUX_MIN+2);
```

The XADC has 12 bits, then I have 4096 values with a max number of 4095.

For the refresh of the high pulse duration of the XTMR configured in 32-bit PWM mode:

```
XTmrCtr_SetResetValue(&TimerInstancePtr, 1, 90000+VccPdroRawData*27);
```

Where:

```
PWM_PERIOD = (TLR0 + 2) * AXI_CLOCK_PERIOD
PWM_HIGH_TIME = (TLR1 + 2) * AXI_CLOCK_PERIOD
```

$$TLR1 = \frac{PWM\_HIGH\_TIME}{AXI\_CLOCK\_PERIOD} - 2 \approx \frac{0.9mS}{100M^{-1}} = 90K$$

$$TLR1 = \frac{PWM\_HIGH\_TIME}{AXI\_CLOCK\_PERIOD} - 2 \approx \frac{1.5mS}{100M^{-1}} = 150K$$

$$TLR1 = \frac{PWM\_HIGH\_TIME}{AXI\_CLOCK\_PERIOD} - 2 \approx \frac{2.1mS}{100M^{-1}} = 210K$$

For the high pulse duration, I need to change the TLR1from 90K for the minimum to 210K to the maximum, hence the variation:

Variation: 210K – 90 K = 110 K

This variation in time (0.9mS to 2.1mS) corresponds a version on TLR1 (90K to 210K), I need to control this with the 12bit read from the XADC, hence:

$$\frac{110K}{4095} \approx 27$$

This corresponds to the formula used for the previous function to refresh the high pulse duration:

$$TLR1 = 90000 + XADC\_VALUE * 27$$

```
/*
 * main.c
 *
 *  Created on: Dec 25, 2018
 *      Author: juandres666
 */

#include "xparameters.h" //information about AXI peripherals
#include "xgpio.h"  //AXI GPIO driver
#include "xsysmon.h" //XADC WIZ driver
#include "xtmrctr.h" //XTMR driver
#include "xintc.h" //XINTC driver

static XSysMon SysMonInst; //sysmon instance

XGpio gpio1;
XIntc Interrupt;
u32 led1=0x00000000;

void SwitchInterruptHandler(void *pointer)
{
    //XGpio InterruptDisable(&gpio1, XGPIO IR CH1 MASK);

    if (led1 != 0x00000000) // turn all LEDs on when any button is pressed
    {
        //XGpio_DiscreteWrite(&gpio1, 2, 0xFFFFFFFF);
        led1=0xFFFFFFFF;
    }
    else
    {
        //XGpio_DiscreteWrite(&gpio1, 2, 0x00000000);
        led1=0x00000000;
    }

    XGpio_InterruptClear(&gpio1, XGPIO_IR_CH1_MASK);
    //XGpio_InterruptEnable(&gpio1, XGPIO_IR_CH1_MASK);
}

int main()
{
    //XADC
    u32 VccPdroRawData;
    XSysMon_Config *SysMonConfigPtr;
    XSysMon *SysMonInstPtr = &SysMonInst;
    SysMonConfigPtr = XSysMon_LookupConfig(XPAR_SYSMON_0_DEVICE_ID);

    XSysMon_CfgInitialize(SysMonInstPtr, SysMonConfigPtr,SysMonConfigPtr->BaseAddress);
    XSysMon_GetStatus(SysMonInstPtr); // Clear the old status

    //XTMR
    XTmrCtr TimerInstancePtr;
    XTmrCtr_Initialize(&TimerInstancePtr,XPAR_TMRCTR_0_DEVICE_ID);

    XTmrCtr_SetOptions(&TimerInstancePtr, 0,
XTC_ENABLE_ALL_OPTION|XTC_DOWN_COUNT_OPTION|XTC_AUTO_RELOAD_OPTION);
//XTmrCtr_SetOptions(&TimerInstancePtr, 0, (XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION |
XTC_DOWN_COUNT_OPTION));

    u32 CounterControlReg = Xil_In32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[0] +
XTC_TCSR_OFFSET);
    CounterControlReg = CounterControlReg | XTC_CSR_ENABLE_PWM_MASK |
XTC_CSR_EXT_GENERATE_MASK;
    Xil_Out32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[0] + XTC_TCSR_OFFSET,
CounterControlReg);

    XTmrCtr_SetOptions(&TimerInstancePtr, 1,
XTC_ENABLE_ALL_OPTION|XTC_DOWN_COUNT_OPTION|XTC_AUTO_RELOAD_OPTION);
//XTmrCtr_SetOptions(&TimerInstancePtr, 1, (XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION |
XTC_DOWN_COUNT_OPTION));

    CounterControlReg = Xil_In32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[1] +
XTC_TCSR_OFFSET);
    CounterControlReg = CounterControlReg | XTC_CSR_ENABLE_PWM_MASK |
XTC CSR EXT GENERATE MASK;
    Xil_Out32(TimerInstancePtr.BaseAddress + XTmrCtr_Offsets[1] + XTC_TCSR_OFFSET,
CounterControlReg);
```

```
    XTmrCtr_SetResetValue(&TimerInstancePtr, 0, 2000000);
//XTmrCtr_SetResetValue(&TimerInstancePtr, 0, 0x5f5e100);
    XTmrCtr_SetResetValue(&TimerInstancePtr, 1, 150000);
//XTmrCtr_SetResetValue(&TimerInstancePtr,1, 0x1f78a40);

    XTmrCtr_Start(&TimerInstancePtr,0);
    XTmrCtr_Start(&TimerInstancePtr,1);

    //XGPIO
    XGpio gpio0;
    u32 btn, led;

    XGpio_Initialize(&gpio0, XPAR_GPIO_0_DEVICE_ID);
    XGpio_SetDataDirection(&gpio0, 2, 0x00000000); // set LED GPIO channel tristates to All
Output
    XGpio_SetDataDirection(&gpio0, 1, 0xFFFFFFFF); // set BTN GPIO channel tristates to All
Input

    XGpio_Initialize(&gpio1, XPAR_GPIO_1_DEVICE_ID);
    XGpio_SetDataDirection(&gpio1, 2, 0x00000000); // set LED GPIO channel tristates to All
Output
    XGpio SetDataDirection(&gpio1, 1, 0xFFFFFFFF); // set BTN GPIO channel tristates to All
Input

    XGpio_InterruptEnable(&gpio1, XGPIO_IR_CH1_MASK);
    XGpio_InterruptGlobalEnable(&gpio1);
    // Initialize Interruption on Interruption Controller
    XIntc_Initialize(&Interrupt, XPAR_INTC_0_DEVICE_ID);//XPAR_INTC_CONTROLLER_DEVICE_ID
    XIntc_SelfTest(&Interrupt);
    // Define Interruption function and start functioning of interruption
    XIntc_Connect(&Interrupt, XPAR_AXI_INTC_0_AXI_GPIO_1_IP2INTC_IRPT_INTR,
(XInterruptHandler)SwitchInterruptHandler, (void
*)0);//XPAR_INTC_CONTROLLER_GPIO_SWITCHES_IP2INTC_IRPT_INTR
    XIntc_Start(&Interrupt, XIN_REAL_MODE);
    XIntc_Enable(&Interrupt,
XPAR_AXI_INTC_0_AXI_GPIO_1_IP2INTC_IRPT_INTR);//XPAR_INTC_CONTROLLER_GPIO_SWITCHES_IP2INTC_IRP
T_INTR

    microblaze_enable_interrupts();// Enable Microblaze interruptions

    while (1)
    {
        btn = XGpio_DiscreteRead(&gpio0, 1);
        if (btn != 0) // turn all LEDs on when any button is pressed
            led = 0xFFFFFFFF;
        else
            led = 0x00000000;
        XGpio_DiscreteWrite(&gpio0, 2, led);

        VccPdroRawData = XSysMon_GetAdcData(SysMonInstPtr,XSM_CH_AUX_MIN+2);//Read the
external Vaux2 Data

        XTmrCtr_SetResetValue(&TimerInstancePtr, 1, 90000+VccPdroRawData*27);
    }
}
```

After to complete this practices I need to say for VIVADO I don't have so much information about the libraries, but I think this subject is important in order to design systems where I need to improve a lot of characteristics like a time control or real time manipulation, I was reading some articles and I would like to learn something about operating systems mounted on this type of devices, like Free RTOS, In order to develop some electronic devices that needs to control some sources and sends data to a network.

I would like to learn about the development of devices and firmware in equipment, what software can be used, what organizational strategies, and what forms to organize the source code, so that it can be reused by the team, and what standards are used in the companies . Also, some testing and the techniques that can be used in the code when it is produced in large quantities.