Faculty of Electrical and Electronic Engineering
Semester II Session 2022/2023

Computer Engineering Laboratory BEJ42001

Section 1

Embedded Systems Design

Topic 4 – Input/Output Interfacing
Topic 5 – Memory Access
Topic 6 – Custom IP Integration

A. Goal:

   i. To build an embedded system with simple input and output, memory block and custom IP components.

B. Requirements:

   i. Design software – VIVADO, SDK from AMD XILINX.
   ii. Hardware – ZC702 or Zybo board.

C. Tasks (6 hours):

   i. Create hardware architecture in VIVADO with Zynq processing system with input/output components (LED, switch, etc.), a BRAM and a custom IP (choose two or more). Create new software application that execute on the components.
   ii. General design steps to complete each tasks are given in the **Appendix A**, **Appendix B** and **Appendix C.** refer also to the tutorial videos in AUTHOR.

D. Resources:

   i. Refer to XILINX documentation portal (https://docs.xilinx.com/) for procedures to use VIVADO and SDK.
   ii. Other sources can be used to complete the tasks but must be cited in the report.

E. Deliverable/elements for evaluation:

   i. Laboratory report (as per table below) – 10 pages maximum, font type: arial, font size: 11. Laboratory report contents are as in the evaluation rubric elements. Upload the laboratory report in author 1 week after the laboratory session for topic no. 6. Use the given laboratory report cover page. Plagiarised reports will be penalized with 0 marks.
   ii. Demonstration on design software usage to complete the tasks in the laboratory.
   iii. Demonstration on work integrity during laboratory work (respectful, dedicated, reliable, professional, honest, ethical).

F. Evaluation rubrics:

   i. Rubric for laboratory report, demonstration and ethics are shown in the **Appendices**.
   ii. Report and demonstration marks are for all group members while ethics marks are for individual member.

Appendix A
General Design Steps For Integration Input/Output (LED, switch)

VIVADO design steps:

1. Create **new project**, specify ZC702 or Zybo board for the target device.
2. Create **block design**, add Zynq processing system and AXI GPIO (2 blocks) using IP Integrator.
3. Run **block automation** and **connection automation** as needed.
4. Configure 1 AXI GPIO block to be the 4 bits LED and another AXI GPIO block to be the switch with interrupt enable.
5. Customize the Zynq processing system to enable the IRQ2F2P[15:0] interrupt signal.
6. Run **block automation** and **connection automation** as needed.
7. Run “**Create HDL Wrapper**” for the block design.
8. Run “**Generate Bitstream**”.
9. Run “**Export Hardware to SDK**”, include bitstream and launch SDK.

SDK design steps:

1. Create **new application project**, write the project name, use **blank template**.
2. Write the software application code as given in **Appendix A-1.**
3. **Connect the ZC702 or Zybo board** to the PC and power it on.
4. Run **Program FPGA** to download the design into the FPGA device.
5. Configure the **SDK Terminal connection**.
6. Select the application project folder in the project explorer, run “**Run As → Launch on Hardware** (**System Debugger**)”.
7. **Operate/check the relevant input/output components** on the board.
8. In the **Console** window, click **square red button** to terminate the execution.

Appendix A-1

```
#include <stdio.h>
#include "platform.h"

#include "xparameters.h"
#include "xgpio.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"

// parameter definitions
#define INTC_DEVICE_ID      XPAR_PS7_SCUGIC_0_DEVICE_ID
#define BTNS_DEVICE_ID      XPAR_AXI_GPIO_1_DEVICE_ID
#define LEDS_DEVICE_ID      XPAR_AXI_GPIO_0_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID          XPAR_FABRIC_AXI_GPIO_1_IP2INTC_IRPT_INTR

#define BTN_INT        XGPIO_IR_CH1_MASK

XGpio LEDInst, BTNInst;
XScuGic INTCInst;
static int led_data;
static int btn_value;
```

```
// prototype functions
static void BTN_Intr_Handler(void *baseaddr_p);
static int InterruptSystemSetup(XScuGic *XScuGicInstancePtr);
static int IntcInitFunction(u16 DeviceId, XGpio *GpioInstancePtr);

// interrupt handler functions
// - called by the timer, button interrupt, performs
// - LED flashing

void BTN_Intr_Handler(void *InstancePtr) {

	// Disable GPIO interrupts
	XGpio_InterruptDisable(&BTNInst, BTN_INT);

	// Ignore additional button presses
	if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) != BTN_INT) {
		return;
	}

	btn_value = XGpio_DiscreteRead(&BTNInst, 1);

	// Increment counter based on button value
	// Reset if centre button pressed
	if (btn_value != 1)
		led_data = led_data + btn_value;
	else
		led_data = 0;

	XGpio_DiscreteWrite(&LEDInst, 1, led_data);
	(void) XGpio_InterruptClear(&BTNInst, BTN_INT);

	// Enable GPIO interrupts
	XGpio_InterruptEnable(&BTNInst, BTN_INT);
}

// initial setup functions

int InterruptSystemSetup(XScuGic *XScuGicInstancePtr) {
	// Enable interrupt
	XGpio_InterruptEnable(&BTNInst, BTN_INT);
	XGpio_InterruptGlobalEnable(&BTNInst);

	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT, (Xil_ExceptionHandler)
		XscuGic_InterruptHandler, XScuGicInstancePtr);
	Xil_ExceptionEnable();

	return XST_SUCCESS;

}

int IntcInitFunction(u16 DeviceId, XGpio *GpioInstancePtr) {

	XScuGic_Config *IntcConfig;
	int status;
	// Interrupt controller initialisation
	IntcConfig = XScuGic_LookupConfig(DeviceId);
	status = XScuGic_CfgInitialize(&INTCInst, IntcConfig,
		IntcConfig->CpuBaseAddress);
	if (status != XST_SUCCESS)
		return XST_FAILURE;

	// Call to interrupt setup
	status = InterruptSystemSetup(&INTCInst);
	if (status != XST_SUCCESS)
		return XST_FAILURE;

	// Connect GPIO interrupt to handler
	status = XScuGic_Connect(&INTCInst, INTC_GPIO_INTERRUPT_ID,
			(Xil_ExceptionHandler) BTN_Intr_Handler, (void *) GpioInstancePtr);
	if (status != XST_SUCCESS)
		return XST_FAILURE;

	// Enable GPIO interrupts interrupt
	XGpio_InterruptEnable(GpioInstancePtr, 1);
	XGpio_InterruptGlobalEnable(GpioInstancePtr);

	// Enable GPIO and timer interrupts in the controller
	XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID);

	return XST_SUCCESS;
}

int main() {

	init_platform();

	int status;

	// initialize the peripherals & set directions of gpio

	// initialise LEDs
	status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
	if (status != XST_SUCCESS)
		return XST_FAILURE;

	// initialise push buttons
	status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);
	if (status != XST_SUCCESS)
		return XST_FAILURE;

	// set LEDs direction to outputs
	XGpio_SetDataDirection(&LEDInst, 1, 0x00);

	// set all buttons direction to inputs
	XGpio_SetDataDirection(&BTNInst, 1, 0xFF);

	// initialize interrupt controller
	status = IntcInitFunction(INTC_DEVICE_ID, &BTNInst);
	if (status != XST_SUCCESS)
		return XST_FAILURE;
```

```
        while (1)
                ;
        return 0;
}
```

Appendix B
General Design Steps For Writing and Reading To/From Memory

VIVADO design steps:

1. Create **new project**, specify ZC702 or Zybo board for the target device.
2. Create **block design**, add Zynq processing system, **AXI BRAM Controller** and **block memory generator** using IP Integrator.
3. Configure the **AXI BRAM Controller** to use only 1 BRAM interface.
4. Run **block automation** and **connection automation** as needed.
5. Run “**Create HDL Wrapper**” for the block design.
6. Run “**Generate Bitstream**”.
7. Run “**Export Hardware to SDK**”, include bitstream and launch SDK.

SDK design steps:

1. Create **new application project**, write the project name, choose “**blank template**”.
2. Write the software application code as given in Appendix B-1 below.
3. **Connect the ZC702 or Zybo board** to the PC and power it on.
4. Run **Program FPGA** to download the design into the FPGA device.
5. Configure the **SDK Terminal connection.**
6. Select the application project folder in the project explorer, “**Run As → Lunch on Hardware (System Debugger)**”.
7. Check the **terminal output**.
8. In the **Console** window, click **square red button** to terminate the execution.

Appendix B-1

```
#include <stdio.h>
#include “platform.h”
#include”xio_io.h”
#include “xparameters.h”

int main() {

        int word1, word2, word3, word4;

        Xil_Out8(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 0, 0xAB);
        Xil_Out8(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 1, 0xCD);
        Xil_Out8(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 2, 0x12);
        Xil_Out8(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 3, 0x34);

        Xil_Out8(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 10, 0x56);
        Xil_Out8(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 12, 0x78);

        Xil_Out32(0xE000A244, 0x00);

        word1 = Xil_In32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR);
        word2 = Xil_In32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 4);
        word3 = Xil_In32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 8);
        word4 = Xil_In32(0xE000A244);
        printf(“Word1 = 0x%08x\n\r”, word1);
        printf(“Word2 = 0x%08x\n\r”, word2);
        printf(“Word3 = 0x%08x\n\r”, word3);
        printf(“Word4 = 0x%08x\n\r”, word4);

        return 0;

}
```

Appendix C
General Design Steps For Custom IP Integration

VIVADO design steps:

1. Create **new project**, specify ZC702 or Zybo board for the target device.
2. Create new design source using "**Add Sources**" and write the verilog code in **Appendix C-1**.
3. Run **synthesis** to check the code has no errors.
4. Run"**Create and Package IP**" menu, choose any name (for e.g – custom_ip_adder) for the IP, "**Create New AXI4 Peripheral**" and "**Edit IP**".
5. Find "// **Add user logic here**" statement in verilog file named xxx_v1_0_S00_AXI.v (xxx is the name of your IP specify in step 4 – custom_ip_adder), insert the code in **Appendix C-2**.
6. Find "//**-- Number of slave registers 4**" in the same verilog file, insert the code in **Appendix C-3**.
7. Find "// **Address decoding for reading registers**" in the same verilog file, modify the code as in **Appendix C-4**.
8. Update the IP configuration using "**Re-package IP**". Then close the project.
9. Create **new project**, specify ZC702 or Zybo board for the target device.
10. **Refresh IP repository** in project setting.
11. Create **block design**, add Zynq processing system, AXI interconnect and the custom IP created previously using IP Integrator (step 4).
12. Run **block automation** and **connection automation** as needed
13. Run "**Create HDL Wrapper**" for the block design.
14. Run "**Generate Bitstream**".
15. Run "**Export Hardware to SDK**", include bitstream and launch SDK.

SDK design steps:

1. Create **new application project**, write the project name, choose "**blank template**".
2. Write the software application code as given in **Appendix C-5**.
3. **Connect the ZC702/Zybo board** to the PC and power it on.
4. Run **Program FPGA** to download the design into the FPGA device.
5. Configure the **SDK Terminal connection.**
6. Select the application project folder in the project explorer, "**Run As → Lunch on Hardware (System Debugger)**".
7. Check the **terminal output**.
8. In the **Console** window, click **square red button** to terminate the execution.

Appendix C-1

```
‘timescale 1ns / 1ps

module custom_ip_adder (
        input clk,
        input [7:0] a,
        input [7:0] b,
        output reg [8:0] sum
);

always @ (posedge clk) begin
        sum <= a + b;
end

endmodule
```

Appendix C-2

```
custom_ip_adder custom_ip_adder_instance
(.clk(S_AXI_ACLK), .a(slv_reg0[7:0]), .b(slv_reg0[15:8]), .sum(adder_out));
```

Appendix C-3

```
wire [8:0] adder_out;
```

Appendix C-4

2’h1: reg_data_out <= **adder_out;**

Appendix C-5

```
#include “xparameters.h”
#include “xil_io.h”
#include “xbasic_types.h”

int main () {

        u32 adder_out;

        Xil_Out32(XPAR_CUSTOM_IP_ADDER_0_S_AXI_BASEADDR, 0x00000301);
        adder_out = Xil_In32( XPAR_CUSTOM_IP_ADDER_0_S_AXI_BASEADDR + 4);
        xil_printf(“A=1; B=3, SUM=%d\n\r”, adder_out);

        Xil_Out32(XPAR_CUSTOM_IP_ADDER_0_S_AXI_BASEADDR, 0x0000FF13);
        adder_out = Xil_In32( XPAR_CUSTOM_IP_ADDER_0_S_AXI_BASEADDR + 4);
        xil_printf(“A=19; B=255, SUM=%d\n\r”, adder_out);


        return 0;

}
```