Lab-11th (6th and 8th November)

ECE270 - Embedded Logic Design

Board: Zedboard

Tasks:

Design and implement an interrupt based design on Zedboard, which displays different numbers on the SSD according to the push button being pressed.

Whenever any of the five push buttons is pressed, an interrupt will be fired. Corresponding to every push button a different value is written into one slave register by PS. The PL reads the value from the register and displays the last four bits (in hexadecimal) on one seven segment of the SSD.

Design Steps:

- Create and Package an IP for the display on SSD. It will be used to generate and send the *cathode* and *anode* signals from PL to the SSD. (Hint: Include the Verilog code which reads the value from one slave register and displays the last four bits equivalent hexadecimal value on one segment of SSD.)
- Create a Block diagram which uses the Zynq Processing System IP, Your IP for SSD and an AXI GPIO IP. The GPIO IP should be interrupt enabled, as these push buttons will be used as interrupts.
- Validate your design, create a wrapper, include the XDC file for SSD, generate bitstream and export the hardware.
- Launch SDK and create an empty application in which different values are written into the register based on an interrupt routine.
- Program FPGA and run the application on Hardware. Verify the display on SSD.

Appendix

HDL Code for SSD:

```
reg [6:0]segOut; //the 7 bit register to hold the data to output

  always @(*) begin
      case (slv_reg0[3:0])
          4'h0 : segOut <= 7'b1000000;  // 0
          4'h1 : segOut <= 7'b1111001;  // 1
          4'h2 : segOut <= 7'b0100100;  // 2
          4'h3 : segOut <= 7'b0110000;  // 3
          4'h4 : segOut <= 7'b0011001;  // 4
          4'h5 : segOut <= 7'b0010010;  // 5
          4'h6 : segOut <= 7'b0000010;  // 6
          4'h7 : segOut <= 7'b1111000;  // 7
          4'h8 : segOut <= 7'b0000000;  // 8
          4'h9 : segOut <= 7'b0010000;  // 9
          4'hA : segOut <= 7'b0001000;    // A
          4'hB : segOut <= 7'b0000011;    // B
          4'hC : segOut <= 7'b1000110;    // C
          4'hD : segOut <= 7'b0100001;    // D
          4'hE : segOut <= 7'b0000110;    // E
          4'hF : segOut <= 7'b0001110;    // F
          default : segOut <= 7'b0111111;

      endcase
  end

    always @(*)
    begin
    cathode = 0;
    anode = ~segOut;
    end
```

Constraint File for SSD mapping:

```
##Pmod Header JA
##Sch name = JA1
set_property PACKAGE_PIN Y11 [get_ports {anode[0]}
set_property IOSTANDARD LVCMOS33 [get_ports {anode[0]}]

##Sch name = JA2
set_property PACKAGE_PIN AA11 [get_ports {anode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[1]}]

##Sch name = JA3
set_property PACKAGE_PIN Y10 [get_ports {anode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[2]}]

##Sch name = JA4
set_property PACKAGE_PIN AA9 [get_ports {anode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[3]}]

##Pmod Header JB
##Sch name = JB1
set_property PACKAGE_PIN W12 [get_ports {anode[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[4]}]

##Sch name = JB2
set_property PACKAGE_PIN W11 [get_ports {anode[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[5]}]

##Sch name = JB3
set_property PACKAGE_PIN V10 [get_ports {anode[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[6]}]

##Sch name = JB4
set_property PACKAGE_PIN W8 [get_ports {cathode}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode}]
```

C Code for SDK:

```
#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_0_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR
#define MY_Count 0x43C00000

#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  button interrupt
//----------------------------------------------------

void BTN_Intr_Handler(void *InstancePtr)
{
        //write the interrupt service routine here
}




//----------------------------------------------------
// MAIN FUNCTION
//----------------------------------------------------
int main (void)
{
        //Write the main function
}
```

```
//--------------------------------------------------
// 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 initialization
	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;
}
```