**ECEN 449-504**

**William Liu**

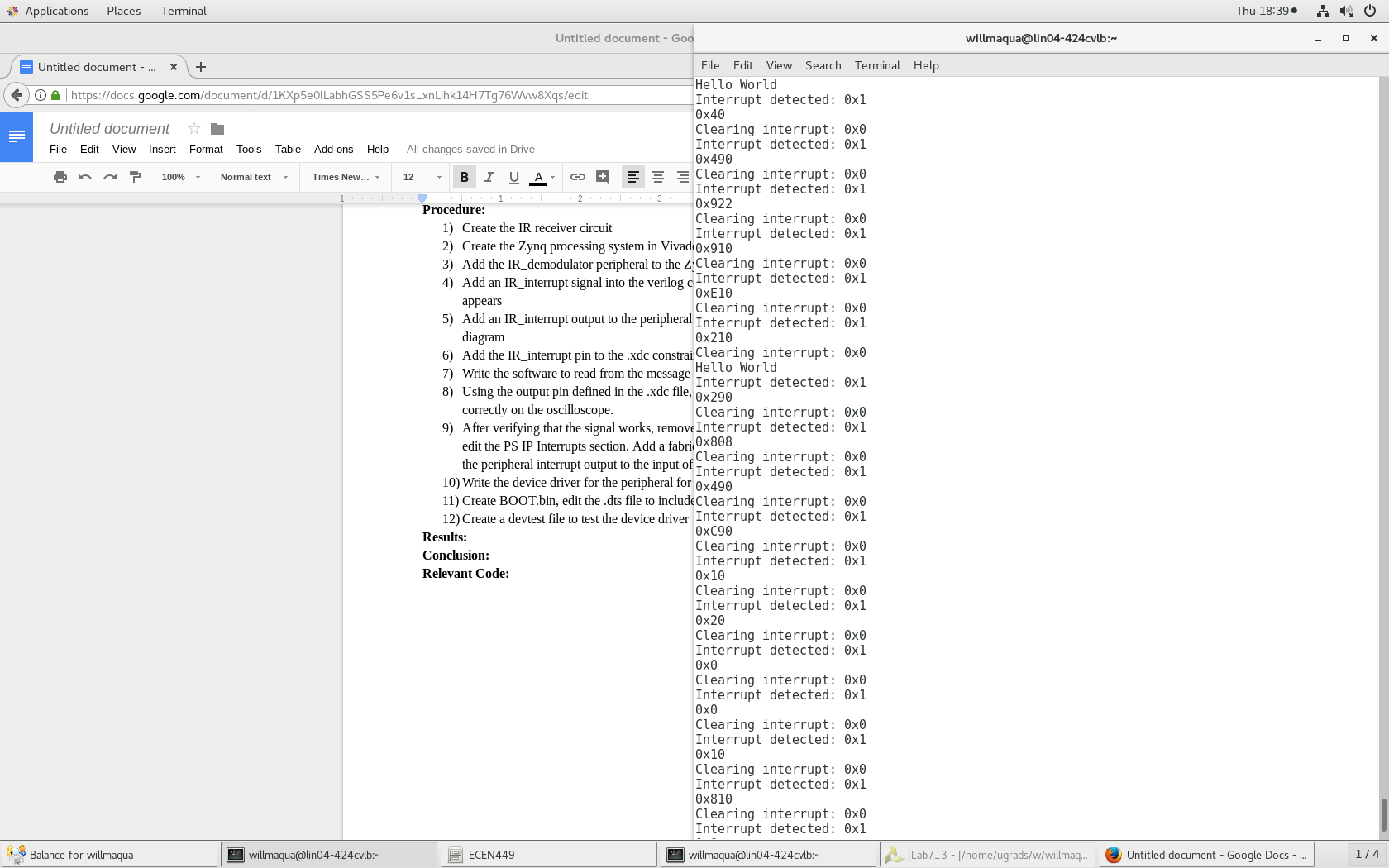
**Lab 8 Report**

**Purpose:** The purpose of this lab is to create a driver for the IR remote receiver by using an interrupt signal instead of polling continuously for the signal to appear.

**Procedure:**

1. Create the IR receiver circuit
2. Create the Zynq processing system in Vivado
3. Add the IR\_demodulator peripheral to the Zynq processing system
4. Add an IR\_interrupt signal into the verilog code for the peripheral for when a message appears
5. Add an IR\_interrupt output to the peripheral and set it to an external port in the block diagram
6. Add the IR\_interrupt pin to the .xdc constraints file
7. Write the software to read from the message register and the interrupt register
8. Using the output pin defined in the .xdc file, check that the interrupt signal shows up correctly on the oscilloscope.
9. After verifying that the signal works, remove the external port from the block diagram and edit the PS IP Interrupts section. Add a fabric interrupt called IRQ\_F2P[15:0] and connect the peripheral interrupt output to the input of the fabric interrupt in the PS block diagram.
10. Write the device driver for the peripheral for Linux
11. Create BOOT.bin, edit the .dts file to include the ir\_demod peripheral
12. Create a devtest file to test the device driver

**Results:**



**Conclusion:**

By using an interrupt signal instead of polling for input continually, more CPU time is made available for use by other processes instead of it being used to wait for empty messages repeatedly. In this situation, an interrupt signal works well because the user will not always be pushing buttons on the remote repeatedly. If data were to come in continuously, polling for input would be a reasonable design.

**Relevant Code:**

**C file**

#include "platform.h"

#include <xil\_printf.h>

#include <ir\_demod.h>

#include <xparameters.h>

#define XPAR\_IR\_DEMOD\_0\_DEVICE\_ID IR\_DEMOD

#define WAIT\_VAL 20000000

int delay(void);

int main()

{

init\_platform();

u32 readmsg = 0;

u32 baseAddr = XPAR\_IR\_DEMOD\_0\_S00\_AXI\_BASEADDR;

u32 oldmsg = 1;

u32 msgCounter = 0;

u32 oldmsgCounter = 1;

u32 readStart = 0;

xil\_printf("Hello World\n\r");

while(1){

//xil\_printf("Message read: 0x%x\n\r", readmsg);

u32 reg2 = IR\_DEMOD\_mReadReg(baseAddr, IR\_DEMOD\_S00\_AXI\_SLV\_REG2\_OFFSET); //read for iterrupt signal

//xil\_printf("reg2: 0x%x\n\r", reg2);

if(reg2 == 1){

xil\_printf("Interrupt detected: 0x%x\n\r", reg2);

//read a message

readmsg = IR\_DEMOD\_mReadReg(baseAddr, IR\_DEMOD\_S00\_AXI\_SLV\_REG0\_OFFSET);

//notify peripheral to clear interrupt bit

IR\_DEMOD\_mWriteReg(baseAddr, IR\_DEMOD\_S00\_AXI\_SLV\_REG2\_OFFSET, 0xFFFF0000);

xil\_printf("0x%x\n\r", readmsg);

//verify the interrupt is cleared

u32 clrreg2 = IR\_DEMOD\_mReadReg(baseAddr, IR\_DEMOD\_S00\_AXI\_SLV\_REG2\_OFFSET);

xil\_printf("Clearing interrupt: 0x%x\n\r", clrreg2);

}

//delay();

}

cleanup\_platform();

return 0;

}

int delay(void){

volatile int delay\_count=0;

while(delay\_count < WAIT\_VAL)

delay\_count++;

return(0);

}

**IR\_demod peripheral**

reg[31:0] counter;

reg[11:0] message;

reg[31:0] recvAmt;

reg[11:0] prevMessage;

reg[4:0] bitCounter;

wire posedge\_detected;

wire negedge\_detected;

reg signal\_dp;

reg signal\_dn;

reg clr\_int\_rst;

initial begin

counter = 0;

message = 0;

bitCounter = 11;

slv\_reg0 = 0;

prevMessage = 0;

slv\_reg2[0] = 0;

recvAmt = 0;

end

always@(posedge S\_AXI\_ACLK) begin

signal\_dp <= IR\_signal;

signal\_dn <= IR\_signal;

counter <= counter + 1;

clr\_int\_rst <= 0; //clear interrupt signal should be 0 by default

if(slv\_reg2[16] == 1) begin

clr\_int\_rst <= 1; //clear interrupt signal when software receives message

slv\_reg2[0] <= 0;

end

if(negedge\_detected) begin

counter <= 0;

end

if(posedge\_detected) begin

if(counter <= 50000) begin

message[bitCounter] <= 0;

bitCounter <= bitCounter - 1;

if(bitCounter == 0) begin

slv\_reg0[11:0] <= message;

recvAmt <= recvAmt + 1;

slv\_reg2[0] <= 1; //set interrupt signal HIGH

end

end

else if(counter > 50000 && counter <= 95000) begin

message[bitCounter] <= 1;

bitCounter <= bitCounter - 1;

if(bitCounter == 0) begin

slv\_reg0[11:0] <= message;

recvAmt <= recvAmt + 1;

slv\_reg2[0] <= 1; //set interrupt signal HIGH

end

end

else if(counter > 95000 && counter <= 187000) begin

bitCounter <= 11;

prevMessage <= message;

message <= 0;

/\*if(slv\_reg2[16] == 1) begin

clr\_int\_rst <= 1;

slv\_reg2[0] <= 0;

end\*/

end

end

end

assign posedge\_detected = !signal\_dp && IR\_signal;

assign negedge\_detected = signal\_dn && !IR\_signal;

assign IR\_interrupt = slv\_reg2[0]; //tie internal interrupt signal to output of peripheral