ECEN 449-504

Lab 7 Report

William Liu

**Introduction:** The purpose of this lab is to test how to read and process PWM signals from an IR remote. We use an TV remote to send the signal and have the ZYBO board process the signal.

**Procedure:**

1) Build the IR receiver circuit and test that the circuit works by measuring the output on the oscilloscope by sending some messages from the TV remote.

2) Record the amount of time each bit is represented by. The remote uses an active LOW signal to represent the bits. The time interval for start message, 0, and 1 are used to process the signal in the next steps.

3) Create the block diagram for the ZYBO board using the Zynq processing system.

4) Create a new peripheral with 4x32 registers and write the Verilog logic to demodulate the IR signal. Slv\_reg0 is used to hold the message, slv\_reg1 is used to count the number of messages received, slv\_reg2 is used for debugging. Use port T20 as input to the ZYBO board from the IR receiver circuit output.

5) To speed up the process of testing and debugging the Verilog code, a Verilog simulator will help tremendously.

6) Verify that the ZYBO board correctly demodulates the signal by writing a short C program to poll for any changes in slv\_reg0 and output if it detects change. Compare the demodulated message with the sample values in the lab manual.

**Results:**

The ZYBO board demodulates signals as expected with some minor noise. Testing and debugging the Verilog code was made much easier and faster by compiling and running simulations using VCS. After verifying that the logic worked in VCS, the code was put to test on the ZYBO board and worked as the simulation showed.

The following code was pulledfrom **ir\_demod\_v1\_0\_S00\_AXI.v** ‘s user logic section so that a testbench could run a simulation.

module ir\_demod\_simple(input IR\_signal, input S\_AXI\_ACLK, output reg[31:0] slv\_reg0, output reg[31:0] slv\_reg1, output reg[31:0] slv\_reg2);

reg[31:0] counter; //used for counting cycles

reg[11:0] message; //buffer to hold 12-bit messages

reg[4:0] bitCounter; //buffer index

wire posedge\_detected; //watch for posedge IR\_signal

wire negedge\_detected; //watch for negedge IR\_signal

reg signal\_dp; //sample IR\_signal

reg signal\_dn; //sample IR\_signal

initial begin

counter = 0;

message = 0;

bitCounter = 11;

clkCounter = 0;

clkDivider = 0;

end

always@(posedge S\_AXI\_ACLK) begin

signal\_dp <= IR\_signal; //sample the IR\_signal ever posedge clock

signal\_dn <= IR\_signal;

slv\_reg2 <= counter;

counter <= counter + 1;

if(negedge\_detected) begin //negedge detected, beginning of next bit to transmit

counter <= 0;

end

if(posedge\_detected) begin //posedge detected, process bit

if(counter <= 50000) begin //this demodulates to 0 (.6ms LOW)

message[bitCounter] <= 0;

bitCounter <= bitCounter - 1;

if(bitCounter == 0) begin

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

slv\_reg1 <= slv\_reg1 + 1;

end

end

else if(counter > 50000 && counter <= 95000) begin //demodulates to 1 (1.2ms LOW)

message[bitCounter] <= 1;

bitCounter <= bitCounter - 1;

if(bitCounter == 0) begin

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

slv\_reg1 <= slv\_reg1 + 1;

end

end

else if(counter > 95000 && counter <= 187000) begin //every message starts with 2.4ms active LOW

bitCounter <= 11;

message <= 0;

end

else begin

counter <= counter;

end

end

end

assign posedge\_detected = !signal\_dp && IR\_signal; //posedge if last known value was 0

assign negedge\_detected = signal\_dn && !IR\_signal; //negedge if last known value was 1

endmodule

**Test bench - the IR signal simulated in here**

module ir\_demod\_top;

reg clk;

reg[31:0] count;

reg irsignal;

wire[31:0] slv\_reg0, slv\_reg1, slv\_reg2;

ir\_demod\_simple irdemod(.IR\_signal(irsignal), .S\_AXI\_ACLK(clk), .slv\_reg0(slv\_reg0), .slv\_reg1(slv\_reg1), .slv\_reg2(slv\_reg2));

always

#6.52 clk = !clk;

initial begin

clk = 0;

irsignal = 1;

#20 irsignal = 0;

#2400000 irsignal = 1;//posedge startmsg

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#20000000 irsignal = 0;

//next message

#2400000 irsignal = 1;//posedge startmsg

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#20000000 irsignal = 0;

//next message

#2400000 irsignal = 1;//posedge startmsg

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#20000000 irsignal = 0;

//next message

#2400000 irsignal = 1;//posedge startmsg

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#1200000 irsignal = 1;//posedge 1

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#600000 irsignal = 0;//negedge

#600000 irsignal = 1;//posedge 0

#20000000 irsignal = 0;

$finish;

end

endmodule

**C program to read demodulated messages**

#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 10000000

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){

oldmsg = readmsg;

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

if(oldmsg != readmsg){

xil\_printf("slv\_reg0 read is: 0x%x\n\r", readmsg);

}

oldmsgCounter = msgCounter;

msgCounter = IR\_DEMOD\_mReadReg(baseAddr, IR\_DEMOD\_S00\_AXI\_SLV\_REG1\_OFFSET);

if(oldmsgCounter != msgCounter){

xil\_printf("slv\_reg1: %d messages received\n\r", msgCounter);

}

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

if(readStart == 0){

xil\_printf("slv\_reg2 read: %d\n\r", readStart);

xil\_printf("receiving NEW message\n\r");

//delay();

}

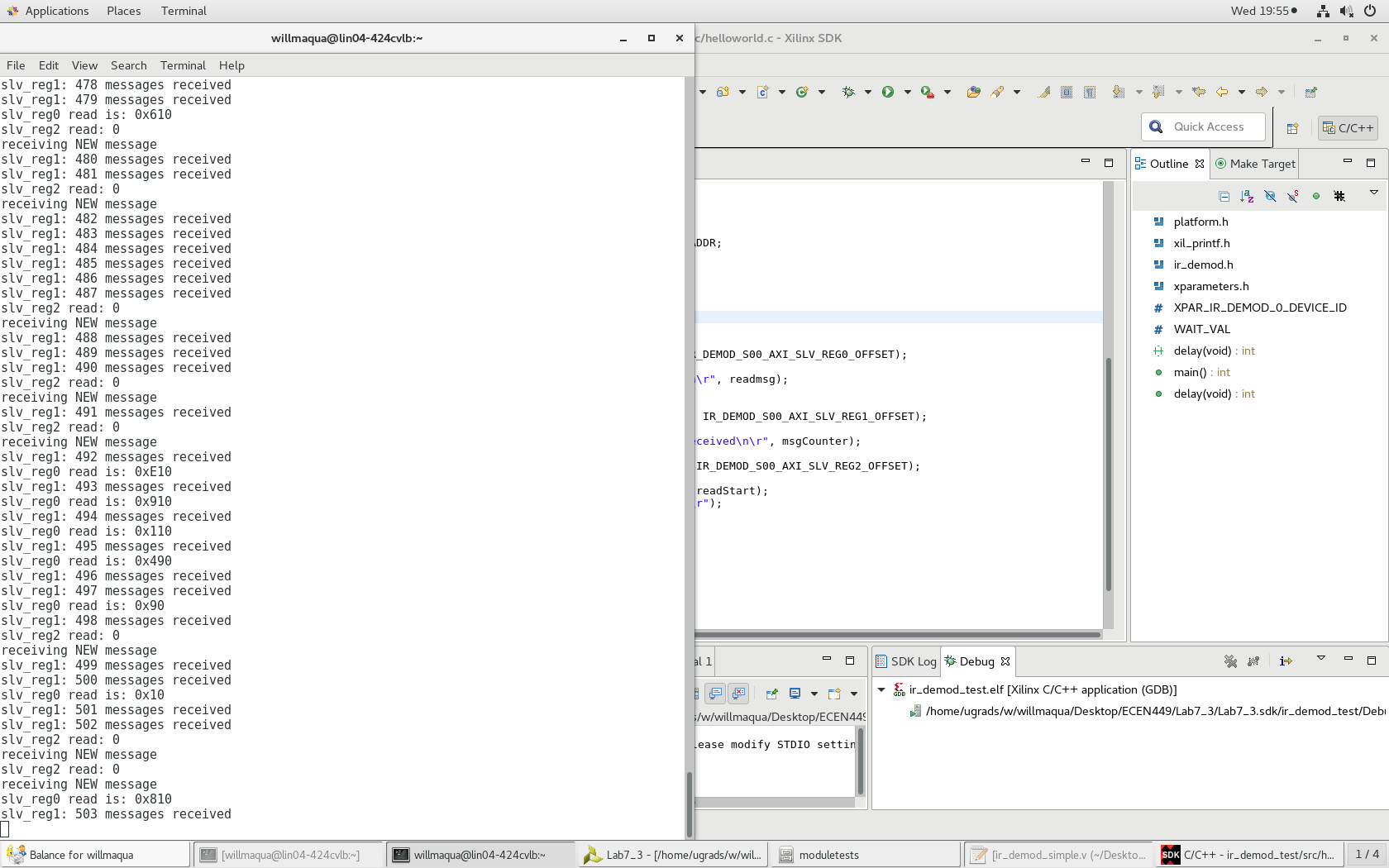
}

cleanup\_platform();

return 0;

}

**Output of picocom when tested with actual remote and ZYBO board**

Volume up, down: 0x490, 0xC90

Channel up, down: 0x90, 0x890

1,2,3,4 respectively: 0x10, 0x810, 0x410, 0xC10

Stop: 0x7B0

Play: 0xFB0

1. About 2 to 5 messages are sent each time the button is pressed. Multiple copies are sent because each message takes around 25ms to transmit. When the button is pushed, it allows the remote to send multiple copies within that small time frame. Multiple messages are sent also because we want to make sure the receiver receives the correct message since noise may affect the demodulation.
2. We can provide an internal signal for receiving new messages by using slv\_reg1 to hold the previous message and compare it with a new message received in slv\_reg0. The signal will go HIGH if they are not equal.

always@(posedge S\_AXI\_ACLK) begin

signal\_dp <= IR\_signal;

signal\_dn <= IR\_signal;

slv\_reg2 <= counter;

counter <= counter + 1;

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;

end

end

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

message[bitCounter] <= 1;

bitCounter <= bitCounter - 1;

if(bitCounter == 0) begin

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

end

end

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

slv\_reg1 <= message;

bitCounter <= 11;

message <= 0;

end

else begin

counter <= counter;

End

if(bitCounter == 0 && slv\_reg0 != slv\_reg1) begin

Receivenew <= 1;

End

Else

Receivenew <= 0;

end

end

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

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

assign receivedNew = (