**Kyle Loyka**

# Lab 7 Report

ECEN 449-503

Due: 28 March 2016

**Introduction**
The purpose of this lab was to create and test hardware that would receive and decode IR signals from a remote.

**Procedure**
A hardware IC chip and circuit was used to capture the IR signal from the remote. The signal was then sent as a stream of bits to the synthesized microblaze processor on the FPGA. The microblaze processor then interpreted the signal from the hardware circuit and decoded it into 0's and 1's.

**Results / Q&A**
The lab worked as expected.

**(a)**

| Control | Hex |
|---|---|
| Volume Up | 490 |
| Volume Down | C90 |
| Stop | 7B0 |
| Play | FB0 |
| 1 | 10 |
| 2 | 810 |
| 3 | 410 |
| 4 | C10 |

**(b)** multiple copies of the same message are sent incase the first message isn't received by the IR receiver. Multiple copies may also be sent for Up/Down commands so that the user can simply press the Up/Down button to change multiple channels without lifting their finger. The remote will repeatedly transmit the signal until the user lets go of the button

**(c)** I would add an internal signal that is set to "1" whenever the counter detects a "start" signal. After the message is sent to slv_reg0, the internal signal would be set back to "0".

**Conclusion**
This lab provided further experience with making hardware interact with software. Its procedure was similar to lab 3.

**C code**

```
//Lab 7


#include<xparameters.h>
#include<ir_demod.h>

int main(void){

      Xuint32 i = 0;
      Xuint32 j = 0;
      Xuint32 k = 0;

      xil_printf("START\n\r**********************************************\n\r"
);

      while(1){
            i = IR_DEMOD_mReadSlaveReg0(XPAR_IR_DEMOD_0_BASEADDR,0);

            j = IR_DEMOD_mReadSlaveReg1(XPAR_IR_DEMOD_0_BASEADDR,0);

            k = IR_DEMOD_mReadSlaveReg2(XPAR_IR_DEMOD_0_BASEADDR,0);
            xil_printf("ir code: %x     j_param: %x
k_param:%x\n\r",i,j,k);

      }

return 0;
}
```

## Verilog

```
module user_logic
(
  // -- ADD USER PORTS BELOW THIS LINE ---------------
  IR_signal,
  // -- ADD USER PORTS ABOVE THIS LINE ---------------

  // -- DO NOT EDIT BELOW THIS LINE ------------------
  // -- Bus protocol ports, do not add to or delete
  Bus2IP_Clk,                     // Bus to IP clock
  Bus2IP_Reset,                   // Bus to IP reset
  Bus2IP_Data,                    // Bus to IP data bus
  Bus2IP_BE,                      // Bus to IP byte enables
  Bus2IP_RdCE,                    // Bus to IP read chip enable
  Bus2IP_WrCE,                    // Bus to IP write chip enable
  IP2Bus_Data,                    // IP to Bus data bus
  IP2Bus_RdAck,                   // IP to Bus read transfer acknowledgement
  IP2Bus_WrAck,                   // IP to Bus write transfer acknowledgement
  IP2Bus_Error                    // IP to Bus error response
  // -- DO NOT EDIT ABOVE THIS LINE ------------------
); // user_logic

// -- ADD USER PARAMETERS BELOW THIS LINE ------------
// --USER parameters added here
// -- ADD USER PARAMETERS ABOVE THIS LINE ------------

// -- DO NOT EDIT BELOW THIS LINE --------------------
// -- Bus protocol parameters, do not add to or delete
parameter C_SLV_DWIDTH                   = 32;
parameter C_NUM_REG                      = 3;
// -- DO NOT EDIT ABOVE THIS LINE --------------------

// -- ADD USER PORTS BELOW THIS LINE -----------------
input IR_signal;
// -- ADD USER PORTS ABOVE THIS LINE -----------------

// -- DO NOT EDIT BELOW THIS LINE --------------------
// -- Bus protocol ports, do not add to or delete
input                                     Bus2IP_Clk;
input                                     Bus2IP_Reset;
input      [0 : C_SLV_DWIDTH-1]           Bus2IP_Data;
input      [0 : C_SLV_DWIDTH/8-1]         Bus2IP_BE;
input      [0 : C_NUM_REG-1]              Bus2IP_RdCE;
input      [0 : C_NUM_REG-1]              Bus2IP_WrCE;
output     [0 : C_SLV_DWIDTH-1]           IP2Bus_Data;
output                                    IP2Bus_RdAck;
output                                    IP2Bus_WrAck;
output                                    IP2Bus_Error;
reg        [0 : C_SLV_DWIDTH-1]           slv_reg0;
  reg        [0 : C_SLV_DWIDTH-1]           slv_reg1;
  reg        [0 : C_SLV_DWIDTH-1]           slv_reg2;
  wire       [0 : 2]                        slv_reg_write_sel;
  wire       [0 : 2]                        slv_reg_read_sel;
  reg        [0 : C_SLV_DWIDTH-1]           slv_ip2bus_data;
  wire                                      slv_read_ack;
  wire                                      slv_write_ack;
  integer                                   byte_index, bit_index;
```

```
  assign
    slv_reg_write_sel = Bus2IP_WrCE[0:2],
    slv_reg_read_sel  = Bus2IP_RdCE[0:2],
    slv_write_ack     = Bus2IP_WrCE[0] || Bus2IP_WrCE[1] || Bus2IP_WrCE[2],
    slv_read_ack      = Bus2IP_RdCE[0] || Bus2IP_RdCE[1] || Bus2IP_RdCE[2];

  // implement slave model register read mux
  always @( slv_reg_read_sel or slv_reg0 or slv_reg1 or slv_reg2 )
    begin: SLAVE_REG_READ_PROC

      case ( slv_reg_read_sel )
        3'b100 : slv_ip2bus_data <= slv_reg0;
        3'b010 : slv_ip2bus_data <= slv_reg1;
        3'b001 : slv_ip2bus_data <= slv_reg2;
        default : slv_ip2bus_data <= 0;
      endcase

    end // SLAVE_REG_READ_PROC


  // ------------------------------------------------------------
  // IR_DEMOD HARDWARE LOGIC

  // slv_reg0 holds latest demodulated message
  // slv_reg1 holds running count of # of messages recieved **TA won't check this
value
  // slv_reg2 debugging

  reg [17:0] counter = 0;  // clock cycle counter to determine IR signal type: Start,
1, or 0
  reg [3:0]  msg_index = 0;      // keeps track of index of the message
  reg [0:11] msg = 12'b0;
  reg [0:11] curr_msg = 12'b0;   // message repeat signal on initial start
  reg buffer = 0;
  reg [3:0] index = 0;
  reg enableCounting = 0;
  reg state = 0;
  reg startFlag = 0;
  reg [17:0] temp = 0;

reg [9:0] slowClkCounter = 0;
reg slowClk = 0;

always@(posedge Bus2IP_Clk) begin
       // this will create a posedge clock signal.
       slowClkCounter <= slowClkCounter + 1;
       if (slowClkCounter == 1000) begin
              slowClk <= 1;
              slowClkCounter <= 0;
       end
       else slowClk <= 0;
end


always@(posedge slowClk) begin

       buffer <= IR_signal;

       slv_reg1 <= index;
       slv_reg2 <= counter;
```

```
	if (buffer && !IR_signal) enableCounting <= 1; //negedge
	else if(!buffer && IR_signal) begin            //posedge
		enableCounting <= 0;
		counter <= 0;
		index <= index + 1;
		if (startFlag) begin
			if (index < 12) begin
				if (index != 0) msg[index-1] <= state;
			end
			else begin
				slv_reg0 <= msg;
				index <= 0;
				startFlag <= 0;
			end
		end
	end
	if(enableCounting && !IR_signal) begin
		counter <= counter + 1;
		if( (counter < 59) && (counter > 29) ) begin
			state <= 0;
		end
		else if( (counter < 104) && (counter > 74) ) begin
			state <= 1;
		end
		else if( (counter < 300) && (counter > 150) ) begin
			startFlag <= 1;
			index <= 0;
		end

	end
end

  assign IP2Bus_Data    = slv_ip2bus_data;
  assign IP2Bus_WrAck   = slv_write_ack;
  assign IP2Bus_RdAck   = slv_read_ack;
  assign IP2Bus_Error   = 0;

endmodule
```

## Modified Verilog

```
reg new_sig = 0;
reg [31:0] prev_signal = 0;

always@(posedge slowClk) begin

	buffer <= IR_signal;

	slv_reg1 <= index;
	slv_reg2 <= counter;

	if (buffer && !IR_signal) enableCounting <= 1; //negedge
	else if(!buffer && IR_signal) begin            //posedge
		enableCounting <= 0;
		counter <= 0;
		index <= index + 1;
		if (startFlag) begin
			if (index < 12) begin
				if (index != 0) msg[index-1] <= state;
			end
			else begin
				if (prev_msg != msg) new_sig <= 1;
				if (prev_msg == msg) new_sig <= 0;
				slv_reg0 <= msg;
				prev_msg <= msg;
				index <= 0;
				startFlag <= 0;

			end
		end
	end
	if(enableCounting && !IR_signal) begin
		counter <= counter + 1;
		if( (counter < 59) && (counter > 29) ) begin
			state <= 0;
		end
		else if( (counter < 104) && (counter > 74) ) begin
			state <= 1;
		end
		else if( (counter < 300) && (counter > 150) ) begin
			startFlag <= 1;
			index <= 0;
		end

	end
end
```

**Kermit OUTPUT**

ir code: 0 j_param: 1 k_param:0
ir code: 0 j_param: 9 k_param:1C
ir code: 10 j_param: 9 k_param:0
ir code: 10 j_param: 0 k_param:0
ir code: 10 j_param: 0 k_param:0
ir code: 10 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 810 j_param: 0 k_param:0
ir code: 410 j_param: 0 k_param:0
ir code: 410 j_param: 0 k_param:0
ir code: 410 j_param: 0 k_param:0
ir code: 410 j_param: 0 k_param:0
ir code: C10 j_param: 0 k_param:0
ir code: C10 j_param: 0 k_param:0
ir code: C10 j_param: 0 k_param:0
ir code: C10 j_param: 0 k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: FFE j_param: F k_param:0
ir code: 890 j_param: B k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 490 j_param: 0 k_param:0
ir code: 490 j_param: 0 k_param:0
ir code: 490 j_param: 0 k_param:0
ir code: 490 j_param: 0 k_param:0

ir code: 490 j_param: 0 k_param:0
ir code: 490 j_param: 0 k_param:0
ir code: 490 j_param: 8 k_param:53
ir code: C90 j_param: 9 k_param:28
ir code: C90 j_param: A k_param:2B
ir code: C90 j_param: B k_param:2C
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: C90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 90 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: 890 j_param: 0 k_param:0
ir code: FB0 j_param: 0 k_param:0
ir code: FB0 j_param: 0 k_param:0
ir code: FB0 j_param: 0 k_param:0
ir code: FB0 j_param: 0 k_param:0

ir code: FB0 j_param: 0 k_param:0
ir code: FB0 j_param: 0 k_param:0
ir code: FB0 j_param: 0 k_param:0
ir code: FB0 j_param: 1 k_param:2
ir code: 7B0 j_param: 1 k_param:10
ir code: 7B0 j_param: 2 k_param:12
ir code: 7B0 j_param: 3 k_param:0
ir code: 7B0 j_param: 3 k_param:0