**Ajin Sunny &**

**Derek King**
**April 30, 2015**

## Introduction

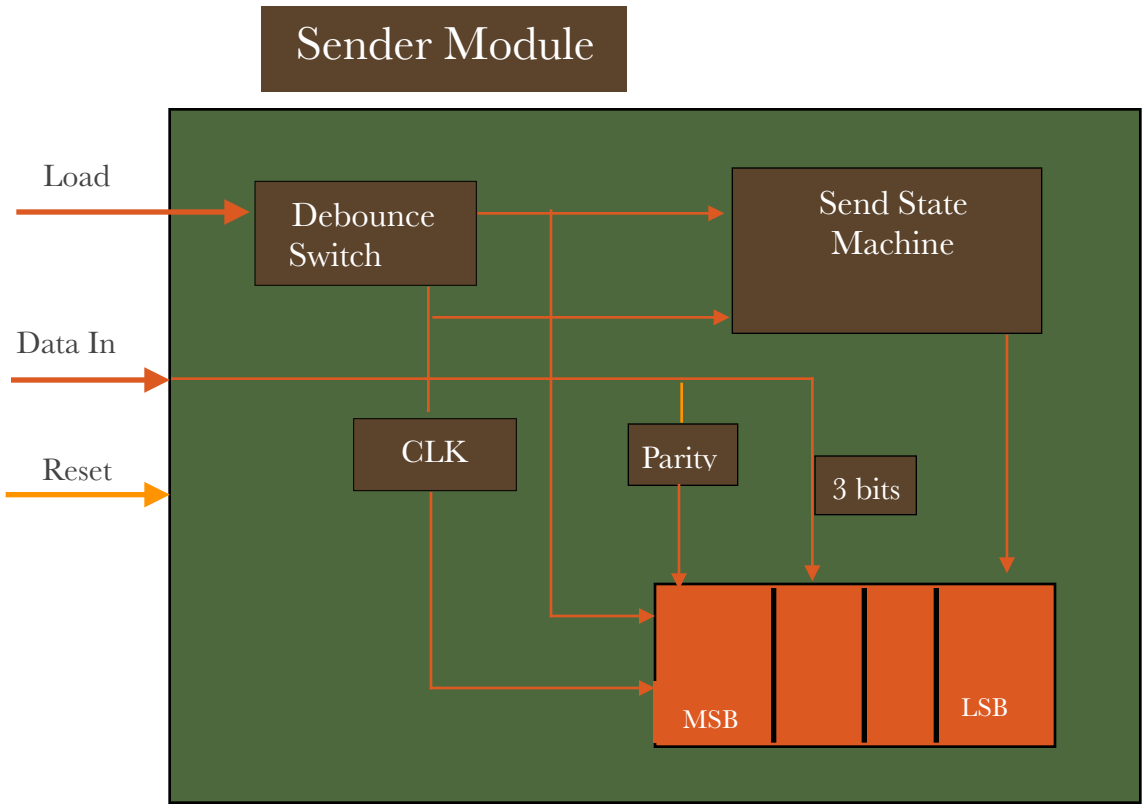
In this Lab we will learn about serial communication and attempt to send a three bit number between two Spartan3 boards. We are asked to write the serial communication code and simulate it. After the simulation confirms the validity of the code, we must built a receiver and sender board and connect them and send the three bit signal between the two. Additionally, we will implement a parity bit check that will help ensure the accuracy of the data being transmitted. Once the data has been received, the three bit binary number will be converted to decimal and displayed on one of the seven segment displays on the receiver board. In this lab, we will learn how to design a system that communicates serially. This is crucial to the field of computer engineering since so many protocols are designed serially, such as USB and Ethernet.

## Experiment Description

### Part 1

In the first part of the experiment we first are required to design and write the code we will use to implement the serial transfer. For good design practice, and to make part two easier, we will design the sender and receiver separately in order to make implementing them on two separate boards much easier.

Below is the Module diagram of the sender module which is used to implement the logic.




********** SENDER MODULE CODE **********

```
module sendermod(input clk, input load, input reset, input [2:0] datain, input ack, output rdy, output data, output shifts, output reg [3:0] y);

    reg [3:0] Q = 0;
    wire shift;

    reg [3:0] Y;
    parameter [3:0] A=4'b0000, B=4'b0001, C=4'b0010, D=4'b0011, E=4'b0100, F=4'b0101, G=4'b0110, H=4'b0111, I=4'b1000;

    always@(ack, load, y)
    begin
        case(y)
            A:    if(load) Y=B;
                  else     Y=A;
```

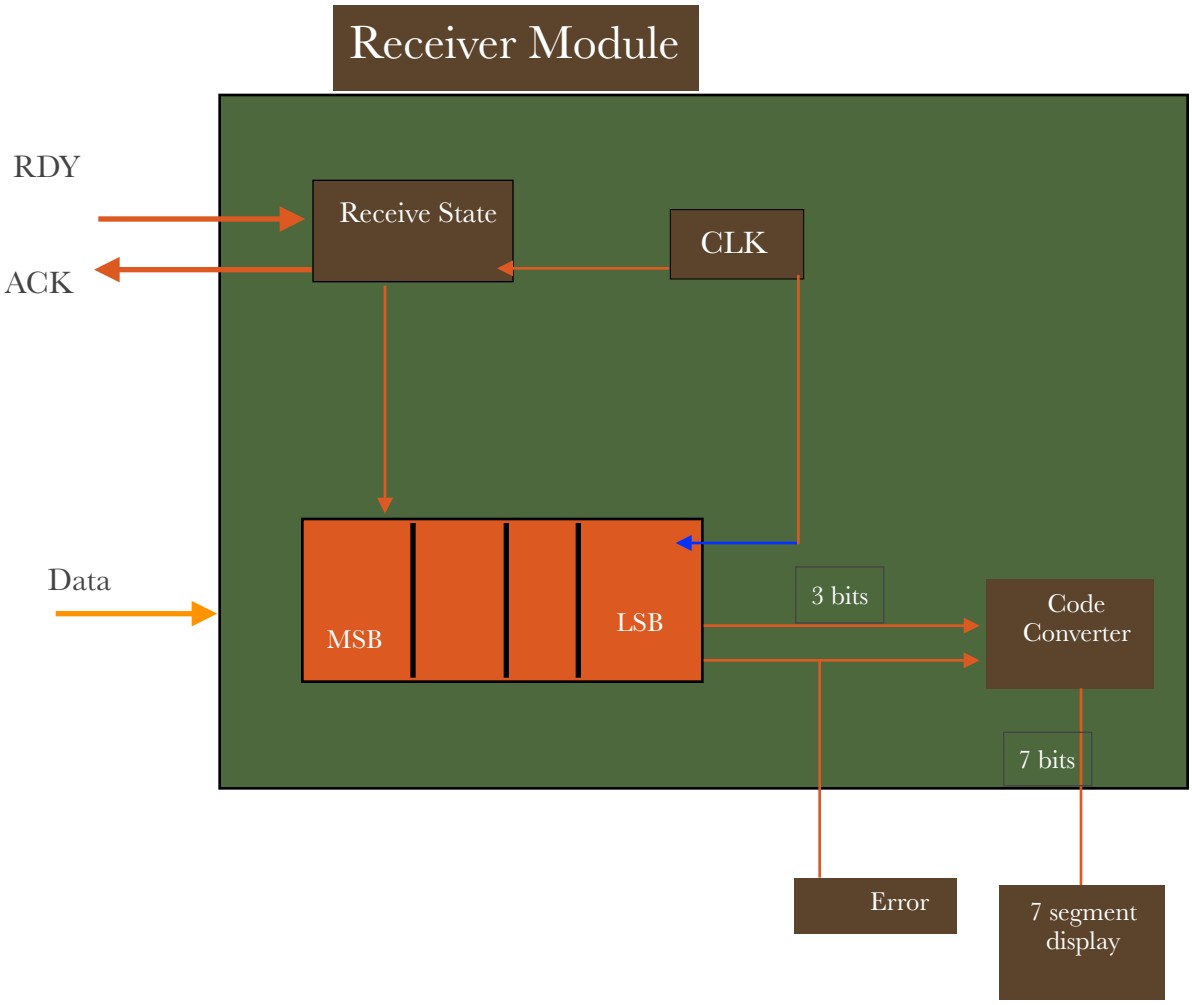
```
                B:      if(load==0) Y=C;
                        else Y=B;
                C:      if(ack==0) Y=D;
                        else Y=C;
                D:      Y=E;
                E:      if(ack) Y=F;
                        else Y=E;
                F:      Y=G;
                G:      if(ack==0)      Y=H;
                        else    Y=G;
                H:      Y=I;
                I: if(ack)      Y=A;
                        else    Y = I;
        endcase
end

always@(posedge clk)
begin
        if(reset) y <= A;
        else y <= Y;
end

assign rdy = (y == A)|(y == B)|(y == E)|(y == F)|(y == I);
assign shift = (y == D)|(y == F)|(y == H);
assign shifts = shift;
assign data = Q[0];

always@(posedge clk)
begin
        if(load)
                Q <= {^datain, datain};
```

```
            else if(shift)
            begin
                    Q[0] <= Q[1];
                    Q[1] <= Q[2];
                    Q[2] <= Q[3];
                    Q[3] <= 0;
            end
    end

endmodule

********** END OF SENDER MODULE ********
```

The sender module mentioned above has 5 inputs. This includes 3 data inputs , Load and a Reset input to the module. The sender module is a 4 bit serial sender which constitutes of a parity bit which will be calculated when the number is loaded into the senders shift register. The Load and reset inputs are used to start the state machine. When the load input is active the , the sender will read and store the value of seven switches. After which it will calculate the value of the even parity bit and start the serial communication using the outputs SRDY and SDATA with an input of SACK. This is how the sender module works. The code written above is self explanatory with different cases from A through I.

In the next part of the experiment we will build the receiver module which has an asynchronous reset input and 4 data outputs to display the seven bits data received. On the serial interface where the handshaking signals takes place, there should be two inputs RRDY and RDATAand one output signal which is RACK. The communication between the sender and the receiver takes place by using the process of handshaking signals.

Depicted below is a module design for the receiver module for the serial communication . If the sender module which is used to implement the logic.

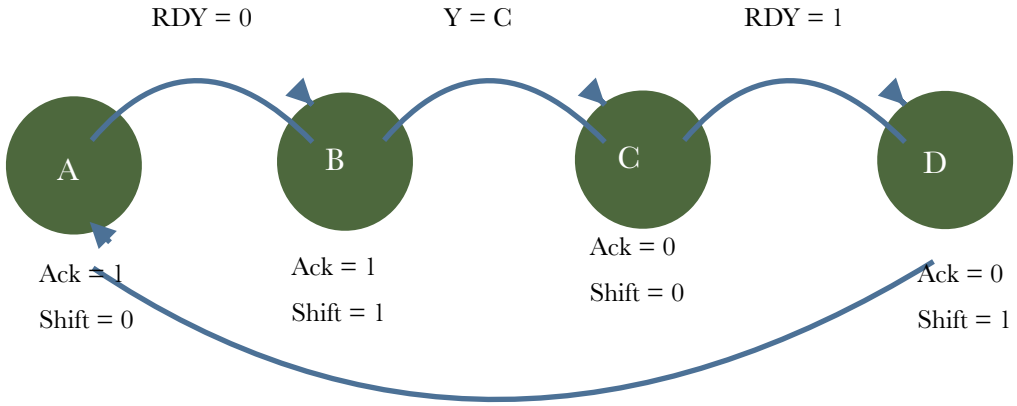



********** RECEIVER MODULE CODE **********

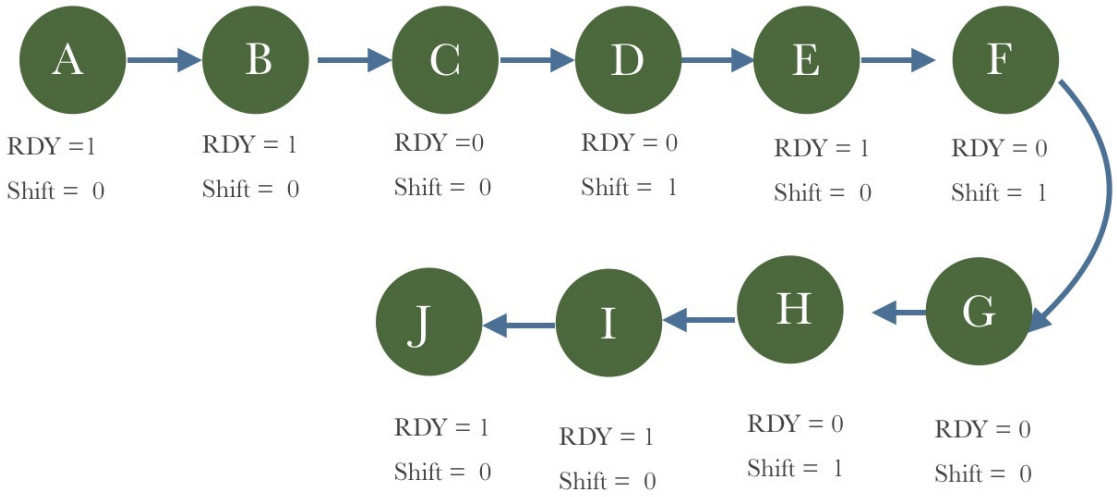
```
module receivermod(input clk, input data, input rdy, input reset, output ack, output error, output [2:0] dataout, output shiftr, output reg [1:0] y);

    reg [3:0] Q = 0;
    wire shift;

    reg [1:0] Y;
    parameter [1:0] A=2'b00, B=2'b01, C=2'b10, D=2'b11;
```

```
always@(rdy, y)
begin
        case(y)
                A:      if(rdy==0) Y=B;
                        else     Y=A;
                B:      Y=C;
                C:      if(rdy) Y=D;
                        else Y=C;
                D:      Y=A;
        endcase
end

always@(posedge clk)
begin
        if(reset) y <= A;
        else y <= Y;
end

assign ack = (y == A)|(y == B);
assign shift = (y == B)|(y == D);
assign error = (y == A)&(^Q==1);
assign dataout = Q;
assign shiftr = shift;

always@(posedge clk)
begin
        if(shift)
                Q <= {data, Q[3:1]};
end
```

RECEIVER STATE MACHINE:




SENDER STATE MACHINE:




On the next page you will find the simulation for the two modules together as a topmodule.

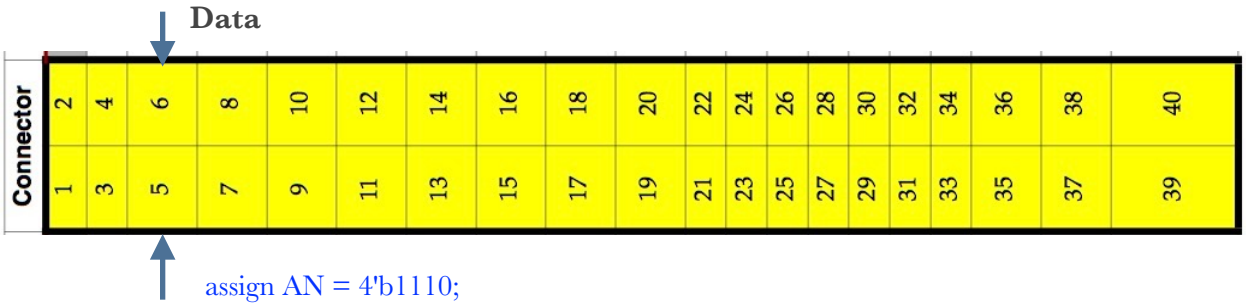
## Part 2

In the second part of the experiment we will implement sender and the receiver modules on two Spartan-3 boards such that we can confirm if the serial communication between the two board works or not. For this process we will first create a top module and use this top module for communication between the two boards. The sender module will consist of a debounce switch which will avoid erratic results as the transmission time is much faster than the time required to stop the debouncing.

The figure shown Below describes the A2 pinout of the Spartan - 3 board which was used for transmitting and receiving signals to and from the two Boards.

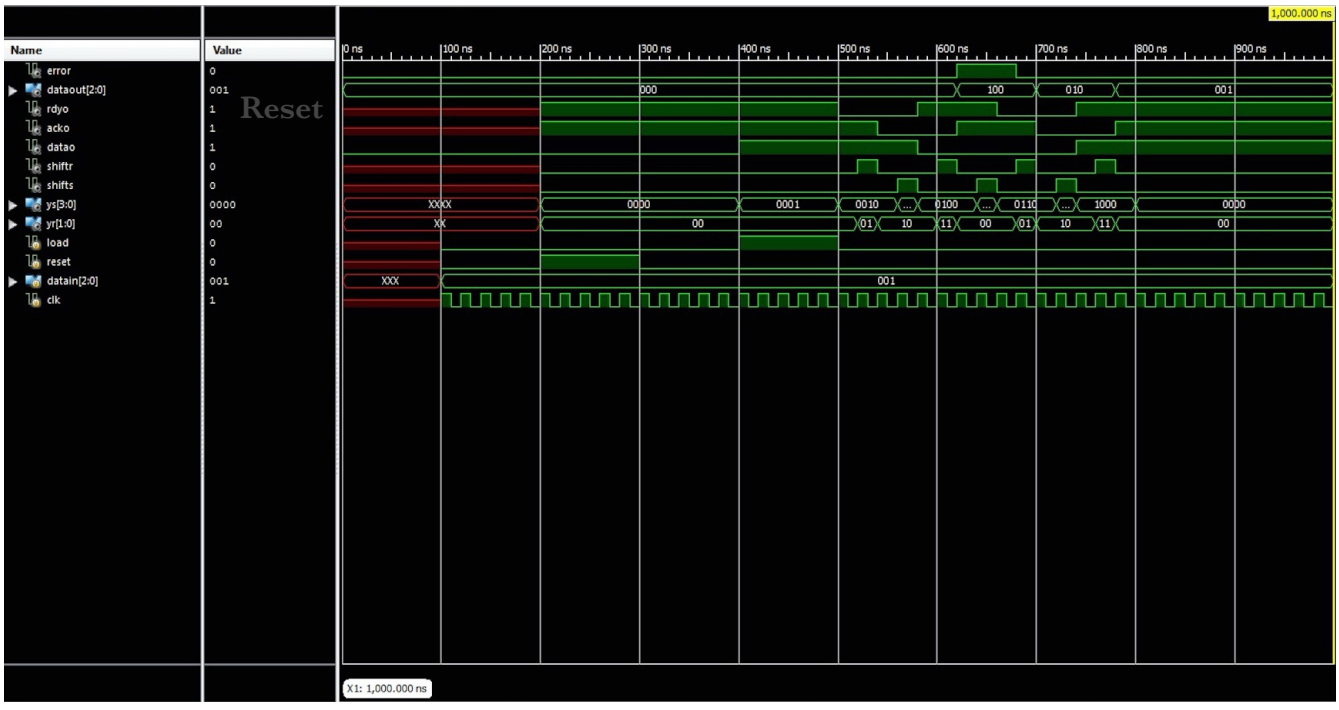
********* TOP MODULE CODE **********

```
module topmod(input load, input reset, input [2:0] datain, output [3:0] AN, output reg [6:0] ssegdisp, output error, output [2:0] dataout, input clk, output rdyo, output acko, output datao, output shiftr, output shifts, output [3:0] ys, output [1:0] yr);

    wire data, rdy, ack;

    assign datao = data;
    assign rdyo = rdy;
    assign acko = ack;

    //module receivermod(input clk, input data, input rdy, input reset, output ack, output error, output [2:0] dataout, output shiftr, output reg [1:0] y);
    receivermod mod1(clk, data, rdy, reset, ack, error, dataout, shiftr, yr);

    //module sendermod(input clk, input load, input reset, input [2:0] datain, input ack, output rdy, output data, output shifts, output reg [3:0] y);
    sendermod mod2(clk, load, reset, datain, ack, rdy, data, shifts, ys);
```




```
assign AN = 4'b1110;
always@(dataout)
begin
case(dataout)
3'b000: ssegdisp = 7'b1000000; //0
3'b001: ssegdisp = 7'b1111001; //1
3'b010: ssegdisp = 7'b0100100; //2
3'b011: ssegdisp = 7'b0110000; //3
3'b100: ssegdisp = 7'b0011001; //4
3'b101: ssegdisp = 7'b0010010; //5
3'b110: ssegdisp = 7'b0000010; //6
3'b111: ssegdisp = 7'b1111000; //7
default: ssegdisp = 7'b1111111; //off
endcase
end

endmodule
```

**********



**SIMULATION RESULTS:**

## RESULTS:

The Lab was able to be completed successfully by this group. Although we did encounter errors while writing the Verilog code for the two modules. We required more than one lab period to successfully complete this lab because our group required further sufficient time to understand the concept of serial communication. The overall results of the experiment had a good turn out. A serial communication was established between the sender Spartan 3 and receiver Spartan 3 which was proved by receiving a successful output on the sevensegment display as well as know that the error LED did not give incorrect information.

## CONCLUSION:

In conclusion from this lab we learned how a unidirectional serial communication works. The lab also helped us understand how to create a state machine and actually implement them by using verilog code. We also learned how to perform an error checking by using the parity bit. All in all we were able to successfully finish this lab with minor errors.