Lab 5

Universal Asynchronous Receiver and Transmitter

# Introduction

Universal Asynchronous Receiver and Transmitter (UART) is an industry standard used in serial communication between electronic devices. The standard is really quite simple and relies heavily on the sender and receiver agreeing on important values. Once we design and implement a working UART system we should be able to expand the design to include a parity check. The parity check will improve the integrity of the data by allowing us to scan for erroneous bit counts.

# Design

For the purposes of this experiment, the communication speed (or baud rate) is defined as 19200 bits per second, the data packets will be 8 bits wide, and we will use odd parity. The receiver and transmitter must agree on these values in order for communication to be successfully established.

Communication begins in an idle state with logic output high. The start state initiates the message by sending a low signal. The next byte of data is sent to the receiver one bit at a time during the data state. The stop state sends the stop bit and returns to the idle state to await the next set of data to send.

The ASM for the UART shows our design methods for parity and no parity:

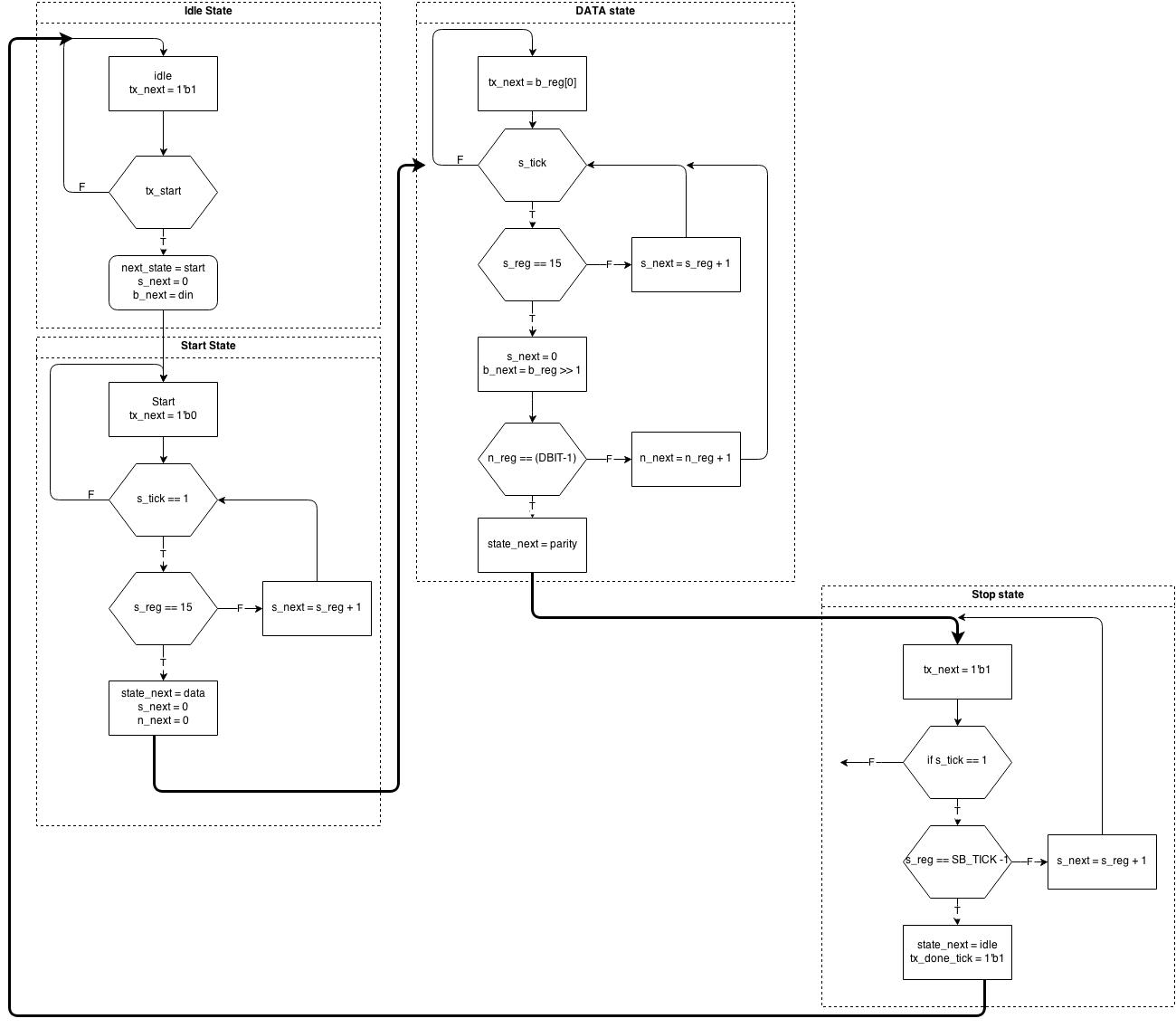


Figure 1: ASM UART w/o parity

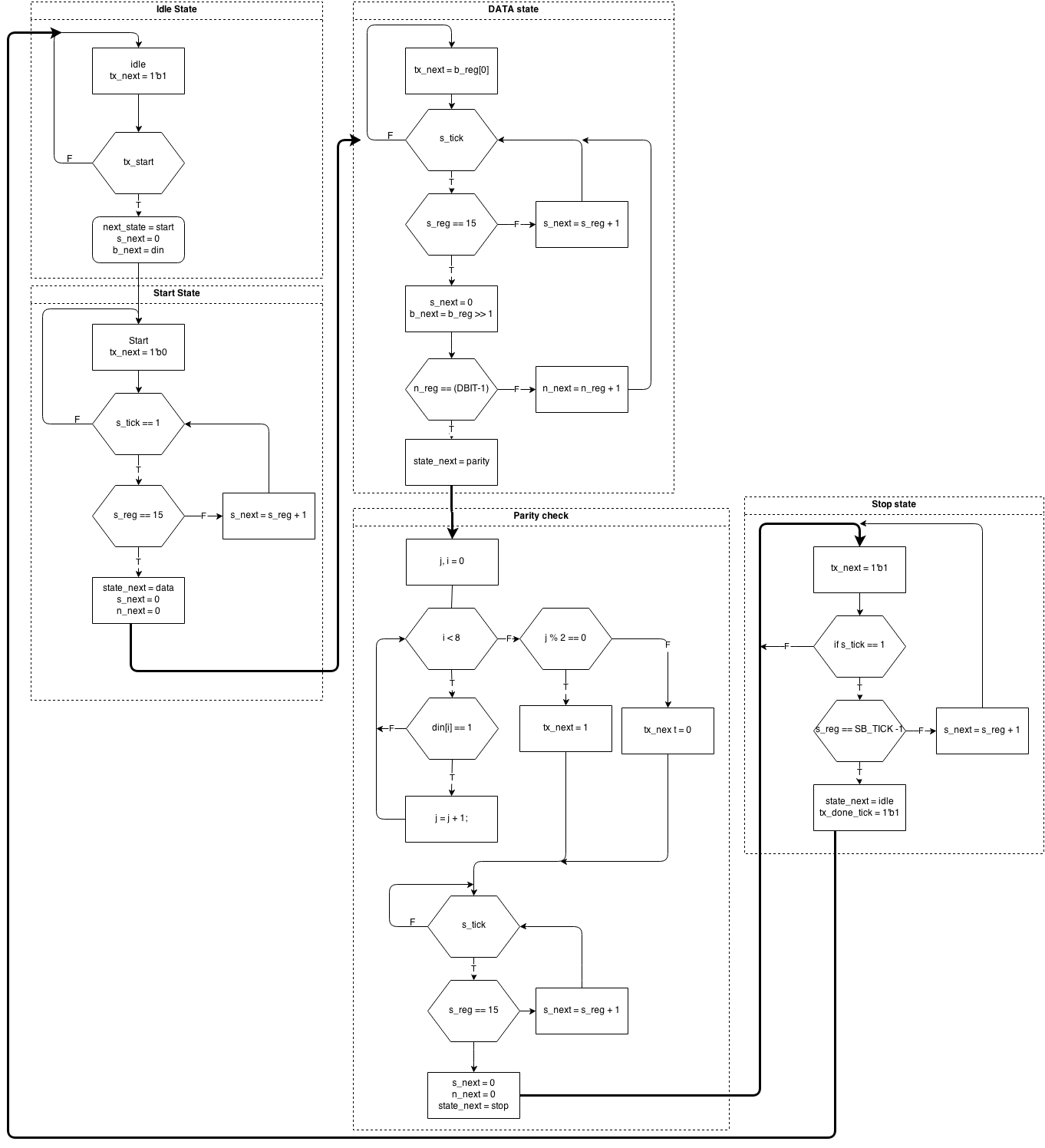


Figure 2: ASM UART w/ parity

# Implementation

A majority of the code for the basic UART was available in the book. We needed to implement the top level block and the modified UART with parity implementation. Below is the code that was not provided for us.

module UART\_controller(

input clk, reset, rx

input [1:0] btn,

input [7:0] sw,

output tx,

output [7:0] sseg,

output [3:0] an, led);

wire button0\_press, button1\_press;

wire alert\_tx\_full, alert\_rx\_empty;

wire [7:0] recieved\_data;

db\_fsm button0(.clk(clk),.reset(reset),.sw(btn[0]),.db(button0\_press));

db\_fsm button1(.clk(clk),.reset(reset),.sw(btn[1]),.db(button1\_press));

uart spartan\_s3\_uart(.clk(clk),.reset(reset),.rd\_uart(button0\_press),

.wr\_uart(button1\_press),.w\_data(sw),.rx(rx),.tx\_full(alert\_tx\_full),

.rx\_empty(alert\_rx\_empty),.tx(tx),.r\_data(recieved\_data));

disp\_hex\_mux display(.clk(clk),.reset(reset),.hex3(sw[7:4]),.hex2(sw[3:0]),

.hex1(recieved\_data[7:4]),.hex0(recieved\_data[3:0]),.dp\_in(4'b1111),

.an(an), .sseg(sseg));

assign led[0] = button0\_press;

assign led[1] = button1\_press;

assign led[2] = alert\_tx\_full;

assign led[3] = alert\_rx\_empty;

endmodule

The following code is for the additional parity check state in the UART Controller.

parity: if (s\_tick)

if (s\_reg==15) begin

s\_next = 0;

state\_next = stop;

j = 0;

for (i=0; i<8; i=i+1) begin

if (b\_reg[i]==1) j=j+1;

end if ((j%2)==rx) begin

parity\_error\_next = 1'b1;

state\_next = idle;

b\_next = 0;

rx\_done\_tick =1'b1;

end else begin

s\_next = 0;

state\_next = stop;

end

end else s\_next = s\_reg + 1;

The parity TX module first find modulo “%” 2 of din. Din contains the complete data packet to be transmitted, and if modulo 2 returns a 1 the packet is odd, if 0 the packet is even. If is even, we want to odd one more bit to the end of the packet indicating that it needs an odd parity bit. After complete, this state passes control over to the end state.

# Results

The image below depicts the simple UART communicating with a Putty session in which Putty is expecting an odd parity bit.

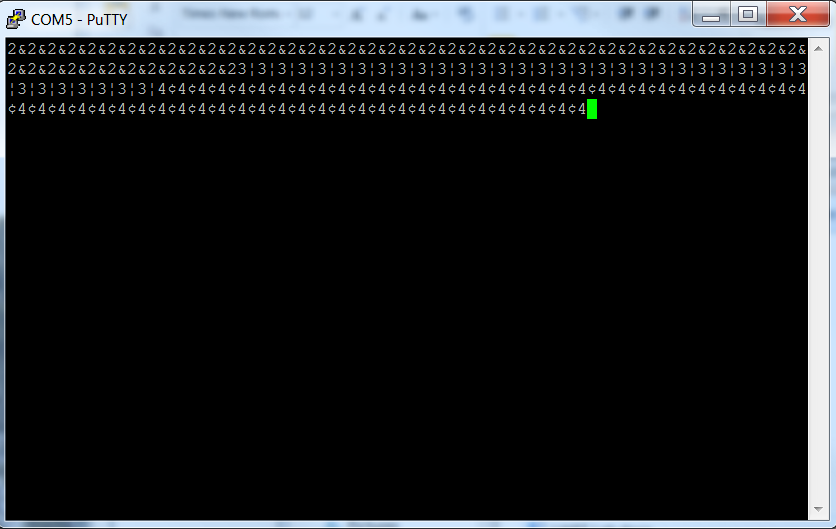


Figure 3: Putty UART Parity Miscommunication

The next image depicts the UART with parity implementation communicating with a Putty session expecting a parity bit.

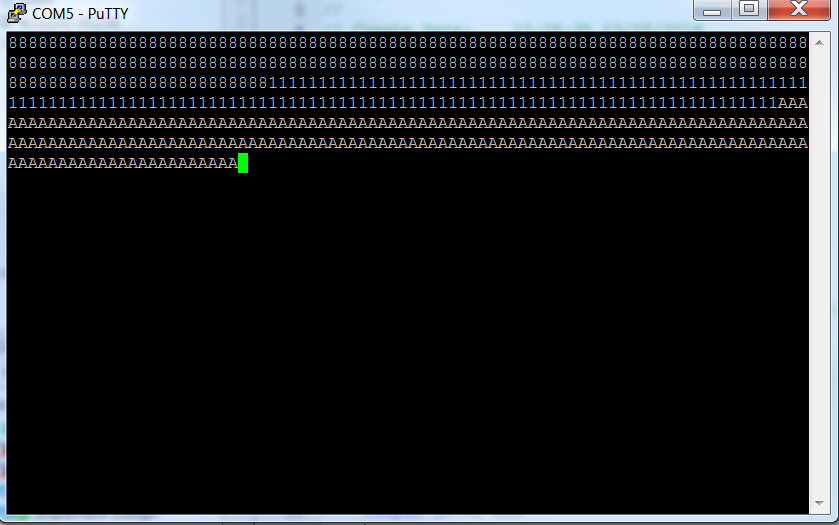
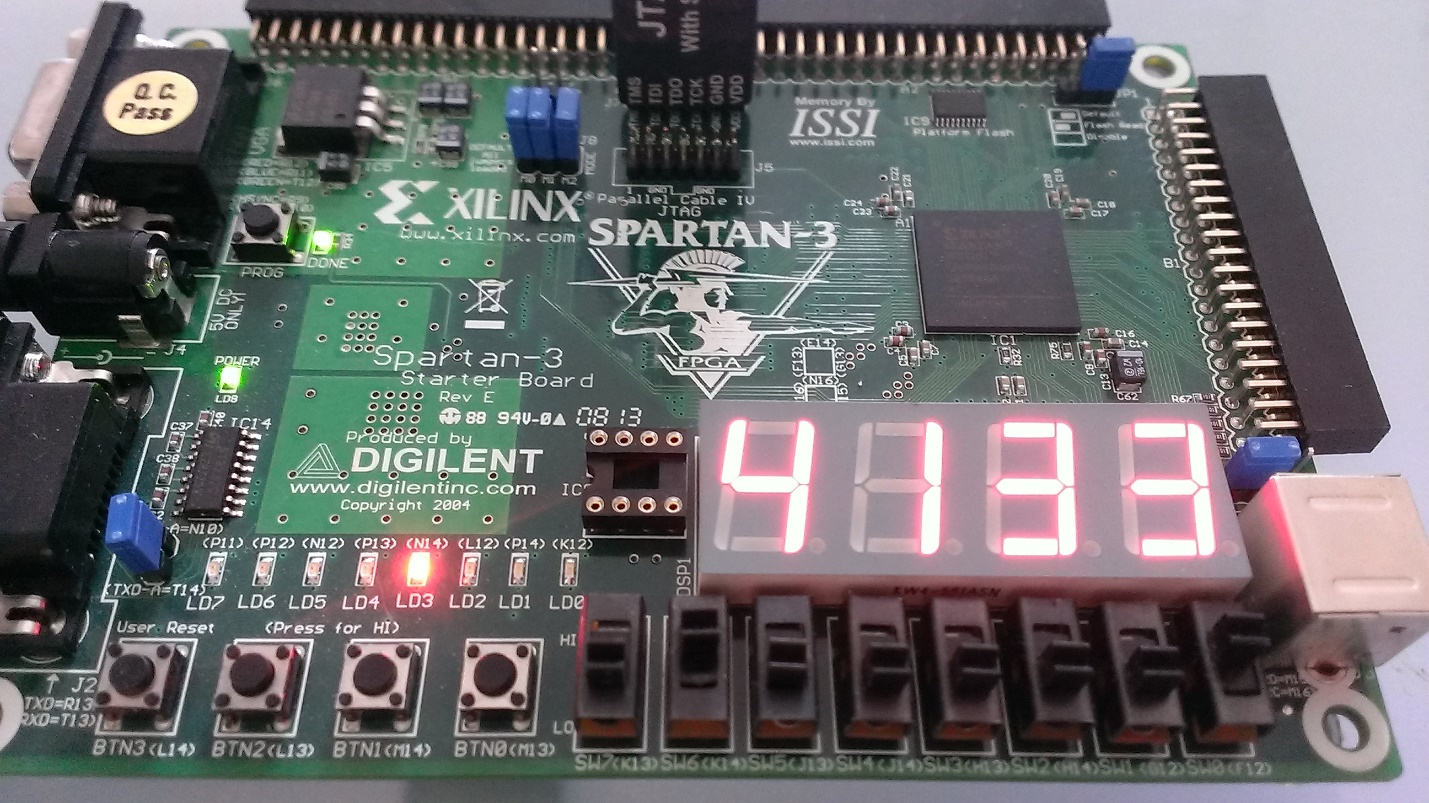


Figure 4: Putty UART Communication

The image below depicts how board resources were utilized for the purposes of this laboratory.



The left two seven segment led sets are utilized for the hex value of the switches (output) and the right two seven segment led sets are used for the display of the input data. The lit led indicates an empty output buffer (all data has been sent). Button zero is for popping data off of the input stack. Button one is for sending the value on the switches to the output.

# Conclusion

# This lab was a success and we were able to implement both a serial TX and RX UART. More importantly though, we learned about the serial protocol and how it operates. That lab worked well, and taught us a lot of about UART’s and serial communication.