Embedded Systems (CSCE 4114)

Lab 4

Zack Fravel

10/20/16

zpfravel@uark.edu

Abstract

The purpose of this lab was to design and test a serial bit receiver. UART, or Universal Asynchronous Receiver/Transmitter, is the serial interface we're designing with in mind. This is "asynchronous" because it isn't dependent on a system clock speed but rather an agreed upon baud rate. After following the lab I was successful in designing a VHDL Moore finite state machine that handles receiving a UART signal.

Introduction

To be able to transmit and receive data asynchronously, there needs to be an agreed upon interface speed, or baud rate. The baud rate is the number of bits per second that a device transmits or receives. Specifically, for our purposes, we want to design a UART receiver that receives data at a baud rate of 115,200 bits/s. This means that for a 50 MHz clock rate, every bit has a duration of 8.68 microseconds or 434 clock cycles. Below is a diagram that shows exactly how the data is sent. The line is kept high ('1') until a START bit is detected and then the 8 bits of data is able to be read until the STOP ('1') 9th bit.

UART Transmission Diagram

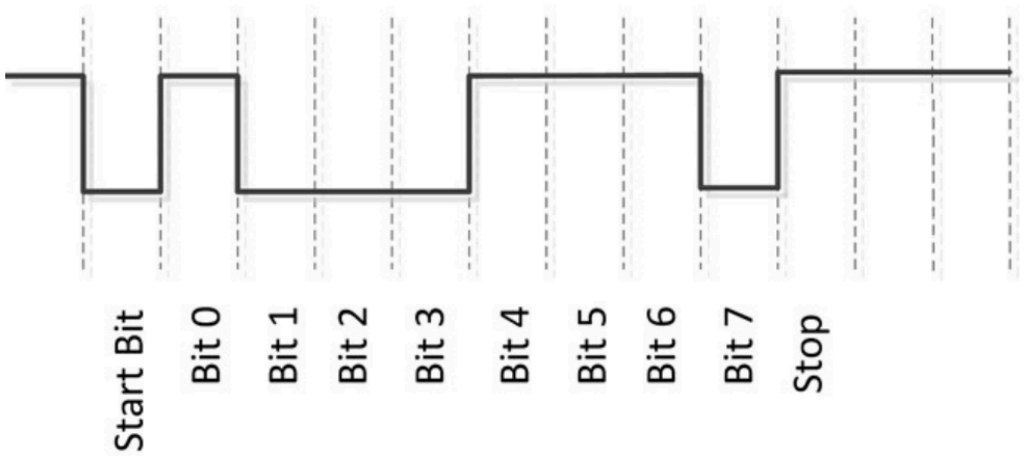


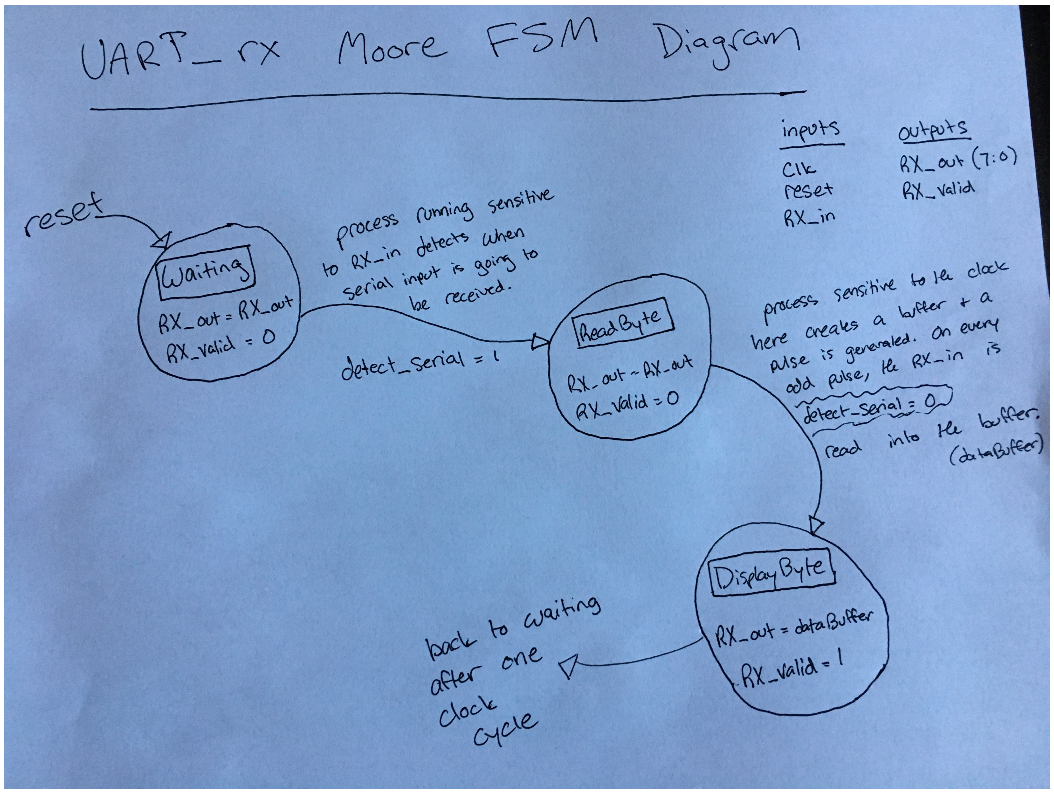
Figure 1

It can be seen that the data transmission begins with the least significant bit and ends with the most significant bit. We need to take this into account when we read data into our device. For my design of the receiver I went with a simple finite state machine that detects the start bit, generates a pulse at twice the speed of the baud rate, reads in the data and displays it on the output as well as indicates when a valid bit has been received.

Design and Implementation

As described previously, I designed a finite state machine that has three unique states. These states are "Waiting," "ReadByte," and "DisplayByte." Below is the state transition diagram for my FSM.

FSM State Diagram



UART_rx Moore FSM Diagram
inputs
Clk
reset
RX_in
outputs
RX_out (7:0)
RX_valid
reset
Waiting
RX_out = RX_out
RX_valid = 0
process running sensitive to RX_in detects when serial input is going to be received.
detect_serial = 1
ReadByte
RX_out = RX_out
RX_valid = 0
process sensitive to the clock here creates a buffer + a pulse is generated. On every odd pulse, the RX_in is read into the buffer. (dataBuffer)
detect_serial = 0
DisplayByte
RX_out = dataBuffer
RX_valid = 1
back to waiting after one clock cycle

Figure 2

The whole module is broken up into six different processes. Before the processes however, I initialize some signals that I'll be needing in the design. First is Pulse_s, which is a signal to be used to generate a pulse every 217 clock cycles. Others are a clock counter (clk_c), detect_serial, bitCount, dataTemp, and the dataBuffer along with the current state and next state signals. The first process handles the current state transition. If the reset signal goes high, the current state is set to "Waiting," all other times the current state is set to the next state signal on every clock cycle. The second process handles the next state transition using a case statement on current state. In the "Waiting" state, the counters are reset and the next state is only set to "ReadByte" whenever the detect_serial signal goes high.

The detect_serial signal is handled by the third process and sets the signal to 1 whenever the process detects a change in the RX_SERIAL_in signal and bitCount is less than 19. Once the second process sets the next state to "ReadByte," on every clock cycle the clk_c signal starts counting. With a case statement, I set the Pulse_s signal to go high ('1') every 217 clock cycles, then clk_c is set back to 0 and bitCount is incremented. All other times Pulse_s = '0.' The other two processes that are running during the "ReadByte" state create two buffers. One is a temporary buffer that reads in the RX_SERIAL_in and appends it to the end along with its previous value. Then, the other process creates a buffer the size of the output (8 bits) and one clock cycle at a time the dataBuffer takes on the value of the dataTemp buffer and right shifts its value over. Once bitCounter exceeds 19, enough to take in 8 bits, detect_serial is set to '0' and the next state is "DisplayByte;" this state is only one clock cycle long then the machine is back in the "Waiting" state. The final process handles all the outputs based on the current state, "Waiting" and "ReadByte" set RX_valid = '0' and "DisplayByte" sets the RX_out to the dataBuffer as well as RX_valid = '1.' With this design, we have a machine that sets its output and indicates when its output is valid on one clock cycle, however when it returns to the waiting state the output stays whatever value was read in. All VHDL files are included at the end of the report.

Results

Once I had my design working as intended I designed a test bench that would suitably show all the functionality of the design. In my testbench I instantiate a 20 ns clock cycle (50 MHz) and create one process. In this process, I wait for 8680 ns, or the length of one bit in the UART baud rate, set reset to '1,' wait 20 ns, and set reset to '0.' I wait for one more bit length, set RX_in to '0' (START bit) and with nine more "wait for 8680 ns" and setting the RX_in to a different value each time, I'm able to simulate a whole byte transmission in UART. Then finally I make the circuit wait for 3 bit lengths and connect my entity. The byte I send in to the RX_in over the whole time is "01100101." Below is the simulation waveform of my testbench.

Simulation Waveform

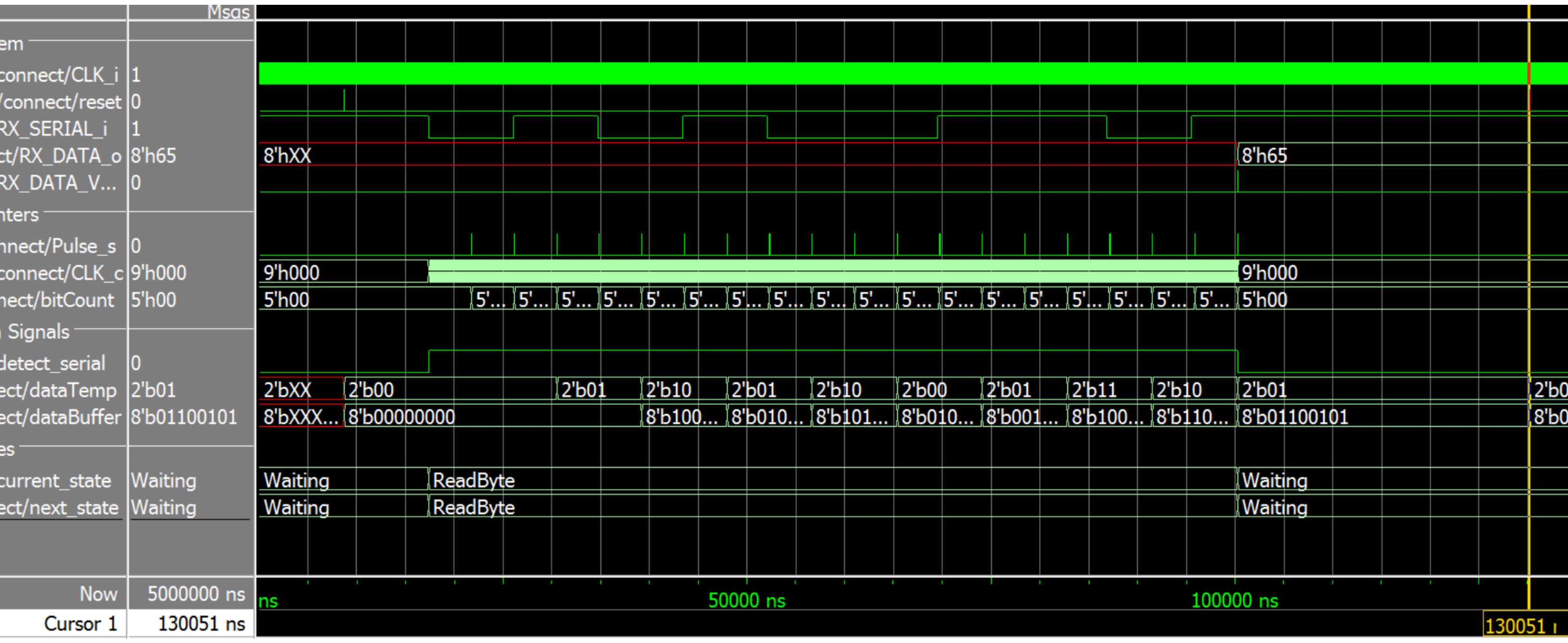
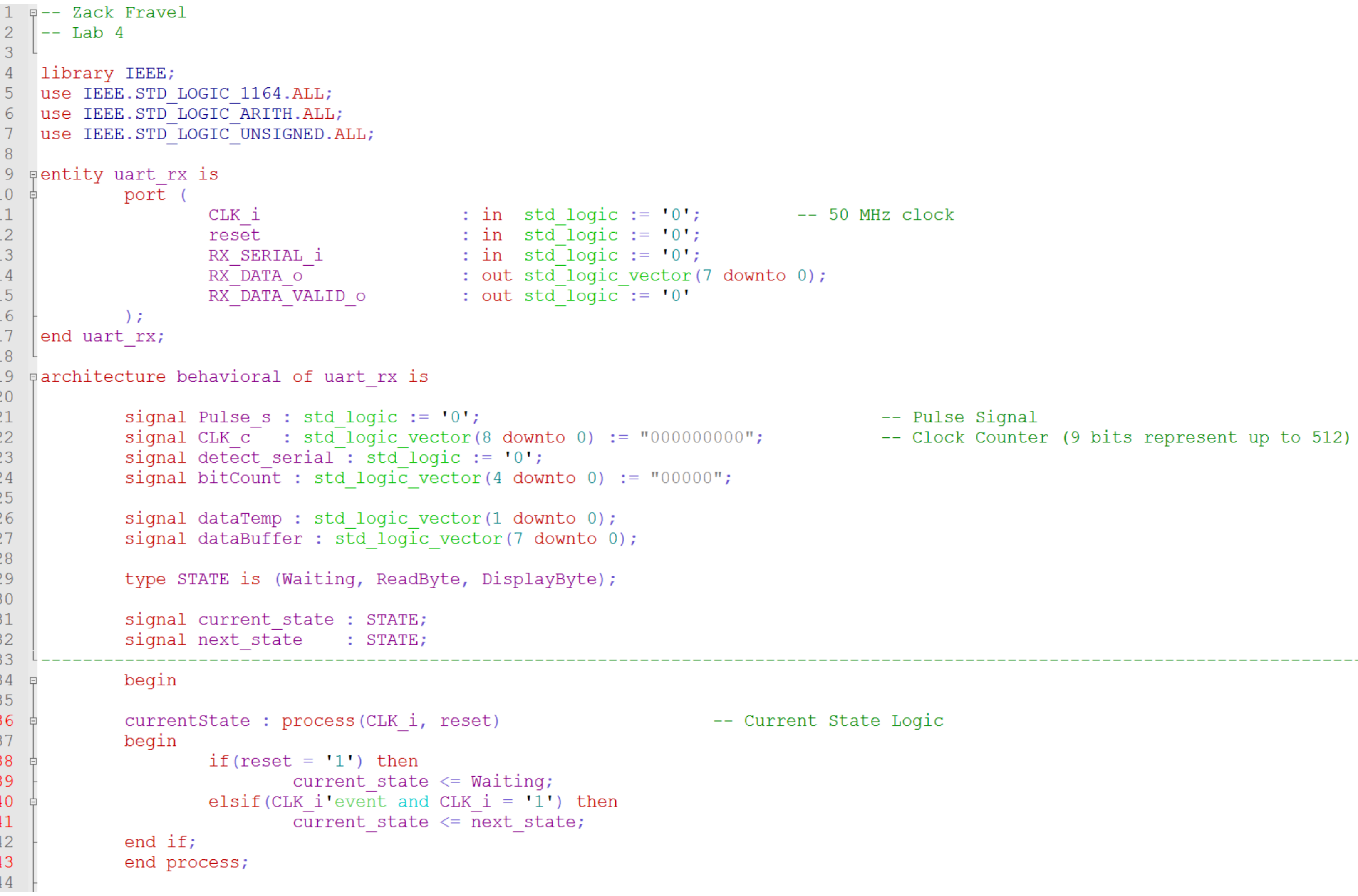


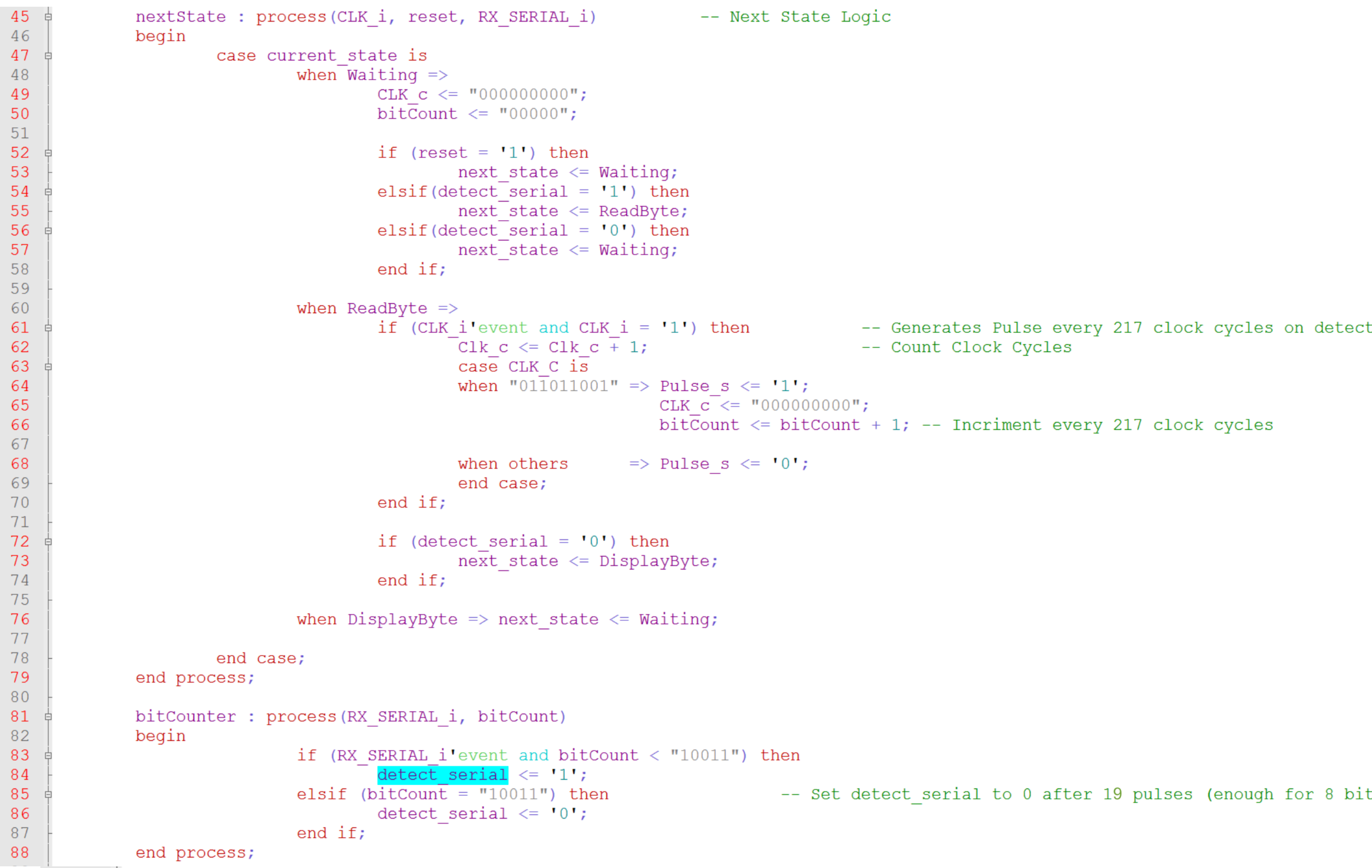
Figure 3

It can be seen on the simulation waveform that the circuit works exactly as described. Once the start bit is detect on the serial input, the counters start spinning up and a pulse is generated every 217 clock cycles. On each odd pulse, the value of RX_in is read into the dataTemp buffer and subsequently read in and right shifted into the dataBuffer. After 19 pulses, the state is set to "DisplayByte" where the output is set to the dataBuffer and RX_valid = 1. Following that, the circuit is set back into the "Waiting" state and the reset signal is asserted after a time. In total, I spend probably at least six hours designing the module.

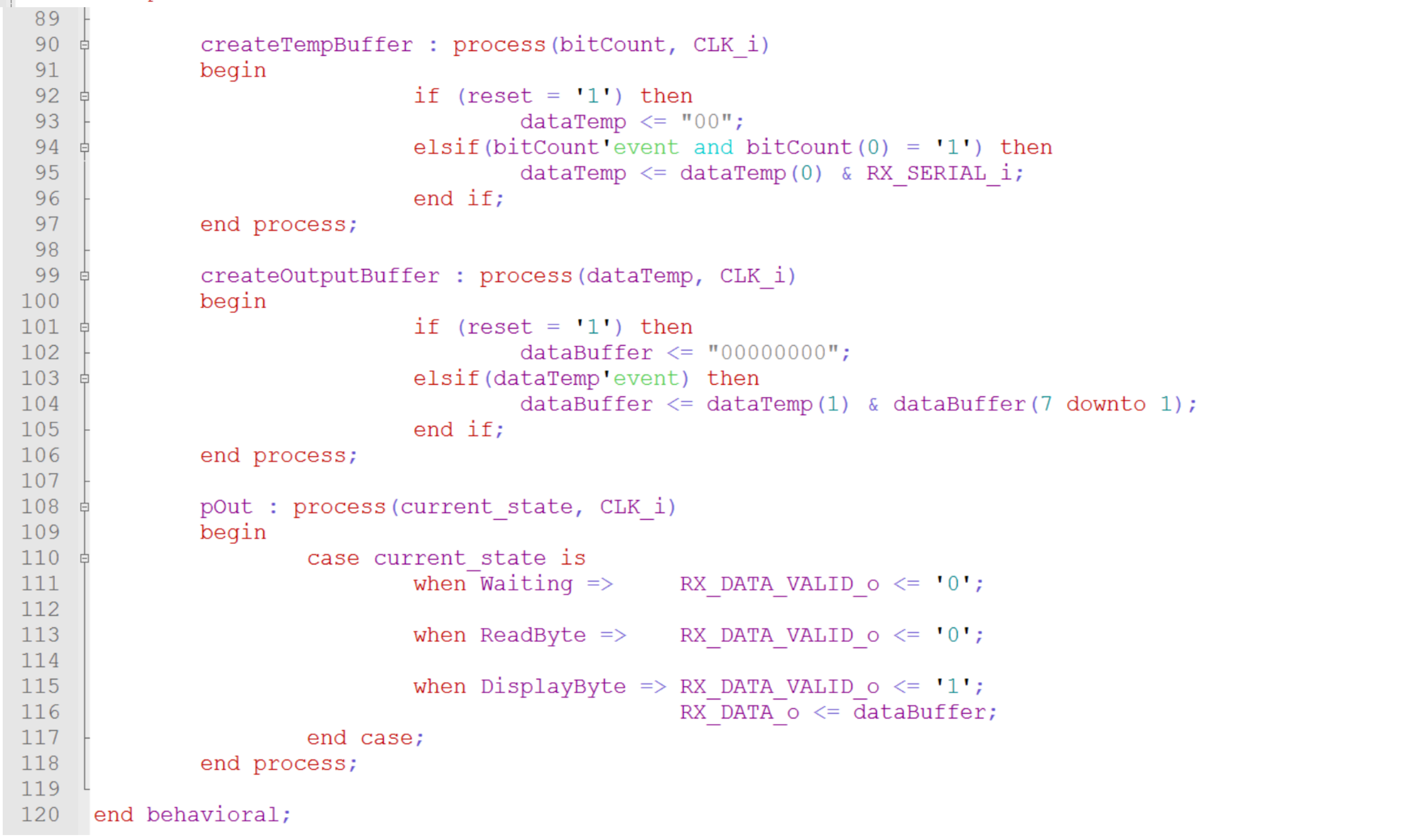
Source Code



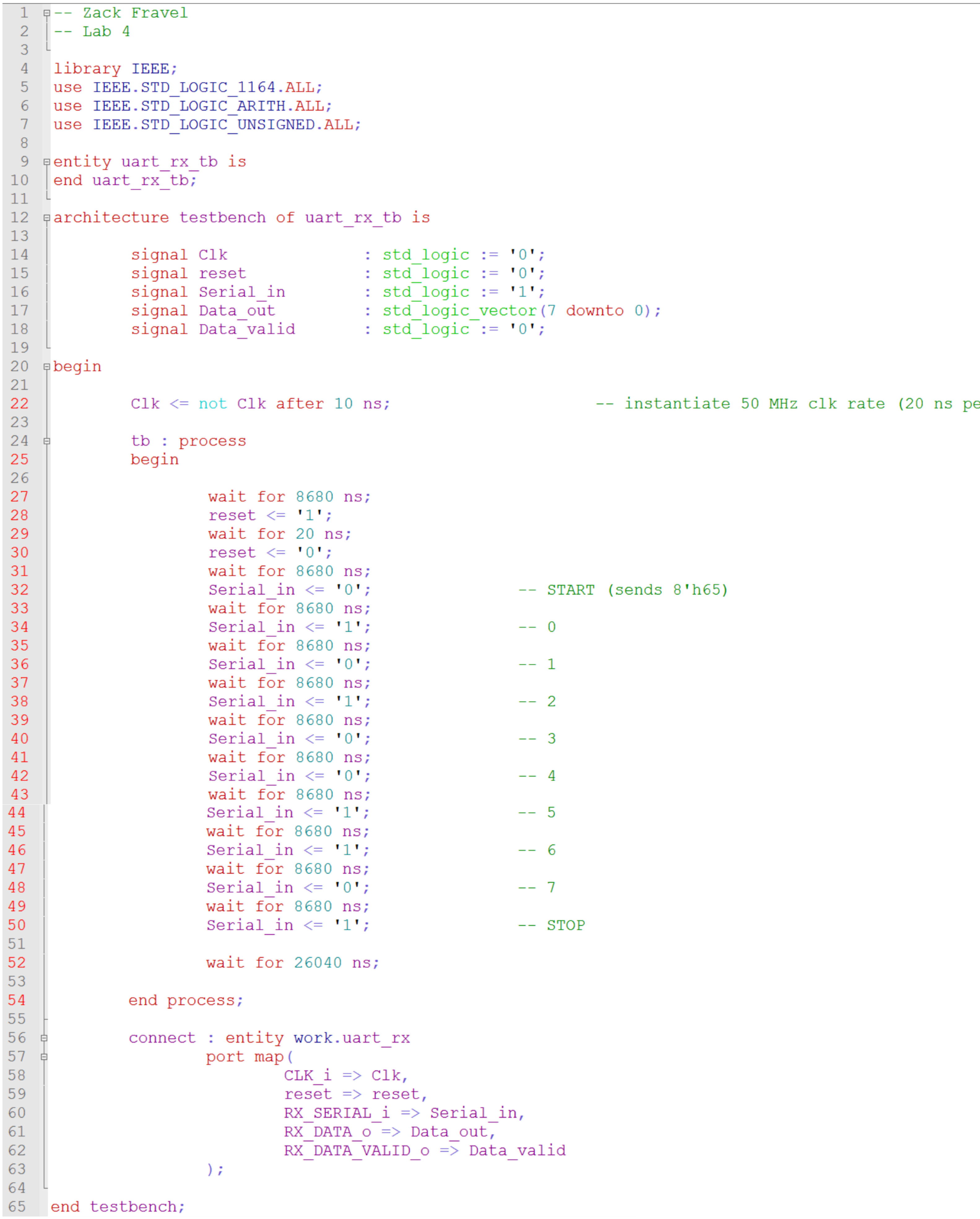
```
-- Zack Fravel
-- Lab 4

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity uart_rx is
        port (
                CLK_i                   : in  std_logic := '0';         -- 50 MHz clock
                reset                   : in  std_logic := '0';
                RX_SERIAL_i             : in  std_logic := '0';
                RX_DATA_o               : out std_logic_vector(7 downto 0);
                RX_DATA_VALID_o         : out std_logic := '0'
        );
end uart_rx;

architecture behavioral of uart_rx is

        signal Pulse_s : std_logic := '0';                                      -- Pulse Signal
        signal CLK_c   : std_logic_vector(8 downto 0)  := "000000000";          -- Clock Counter (9 bits represent up to 512)
        signal detect_serial : std_logic := '0';
        signal bitCount : std_logic_vector(4 downto 0) := "00000";

        signal dataTemp : std_logic_vector(1 downto 0);
        signal dataBuffer : std_logic_vector(7 downto 0);

        type STATE is (Waiting, ReadByte, DisplayByte);

        signal current_state : STATE;
        signal next_state    : STATE;
-----------------------------------------------------------------------------------------------------------------------------
        begin

        currentState : process(CLK_i, reset)                    -- Current State Logic
        begin
                if(reset = '1') then
                        current_state <= Waiting;
                elsif(CLK_i'event and CLK_i = '1') then
                        current_state <= next_state;
        end if;
        end process;
```



```vhdl
        nextState : process(CLK_i, reset, RX_SERIAL_i)          -- Next State Logic
        begin
                case current_state is
                        when Waiting =>
                                CLK_c <= "000000000";
                                bitCount <= "00000";

                                if (reset = '1') then
                                        next_state <= Waiting;
                                elsif(detect_serial = '1') then
                                        next_state <= ReadByte;
                                elsif(detect_serial = '0') then
                                        next_state <= Waiting;
                                end if;

                        when ReadByte =>
                                if (CLK_i'event and CLK_i = '1') then           -- Generates Pulse every 217 clock cycles on detect
                                        Clk_c <= Clk_c + 1;                     -- Count Clock Cycles
                                        case CLK_C is
                                        when "011011001" => Pulse_s <= '1';
                                                            CLK_c <= "000000000";
                                                            bitCount <= bitCount + 1; -- Incriment every 217 clock cycles

                                        when others      => Pulse_s <= '0';
                                        end case;
                                end if;

                                if (detect_serial = '0') then
                                        next_state <= DisplayByte;
                                end if;

                        when DisplayByte => next_state <= Waiting;

                end case;
        end process;

        bitCounter : process(RX_SERIAL_i, bitCount)
        begin
                        if (RX_SERIAL_i'event and bitCount < "10011") then
                                detect_serial <= '1';
                        elsif (bitCount = "10011") then                 -- Set detect_serial to 0 after 19 pulses (enough for 8 bit
                                detect_serial <= '0';
                        end if;
        end process;
```



```vhdl

        createTempBuffer : process(bitCount, CLK_i)
        begin
                        if (reset = '1') then
                                dataTemp <= "00";
                        elsif(bitCount'event and bitCount(0) = '1') then
                                dataTemp <= dataTemp(0) & RX_SERIAL_i;
                        end if;
        end process;

        createOutputBuffer : process(dataTemp, CLK_i)
        begin
                        if (reset = '1') then
                                dataBuffer <= "00000000";
                        elsif(dataTemp'event) then
                                dataBuffer <= dataTemp(1) & dataBuffer(7 downto 1);
                        end if;
        end process;

        pOut : process(current_state, CLK_i)
        begin
                case current_state is
                        when Waiting =>     RX_DATA_VALID_o <= '0';

                        when ReadByte =>    RX_DATA_VALID_o <= '0';

                        when DisplayByte => RX_DATA_VALID_o <= '1';
                                            RX_DATA_o <= dataBuffer;
                end case;
        end process;

end behavioral;
```



```vhdl
-- Zack Fravel
-- Lab 4

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity uart_rx_tb is
end uart_rx_tb;

architecture testbench of uart_rx_tb is

        signal Clk                  : std_logic := '0';
        signal reset                : std_logic := '0';
        signal Serial_in            : std_logic := '1';
        signal Data_out             : std_logic_vector(7 downto 0);
        signal Data_valid           : std_logic := '0';

begin

        Clk <= not Clk after 10 ns;                     -- instantiate 50 MHz clk rate (20 ns pe

        tb : process
        begin

                wait for 8680 ns;
                reset <= '1';
                wait for 20 ns;
                reset <= '0';
                wait for 8680 ns;
                Serial_in <= '0';               -- START (sends 8'h65)
                wait for 8680 ns;
                Serial_in <= '1';               -- 0
                wait for 8680 ns;
                Serial_in <= '0';               -- 1
                wait for 8680 ns;
                Serial_in <= '1';               -- 2
                wait for 8680 ns;
                Serial_in <= '0';               -- 3
                wait for 8680 ns;
                Serial_in <= '0';               -- 4
                wait for 8680 ns;
                Serial_in <= '1';               -- 5
                wait for 8680 ns;
                Serial_in <= '1';               -- 6
                wait for 8680 ns;
                Serial_in <= '0';               -- 7
                wait for 8680 ns;
                Serial_in <= '1';               -- STOP

                wait for 26040 ns;

        end process;

        connect : entity work.uart_rx
                port map(
                        CLK_i => Clk,
                        reset => reset,
                        RX_SERIAL_i => Serial_in,
                        RX_DATA_o => Data_out,
                        RX_DATA_VALID_o => Data_valid
                );

end testbench;
```