

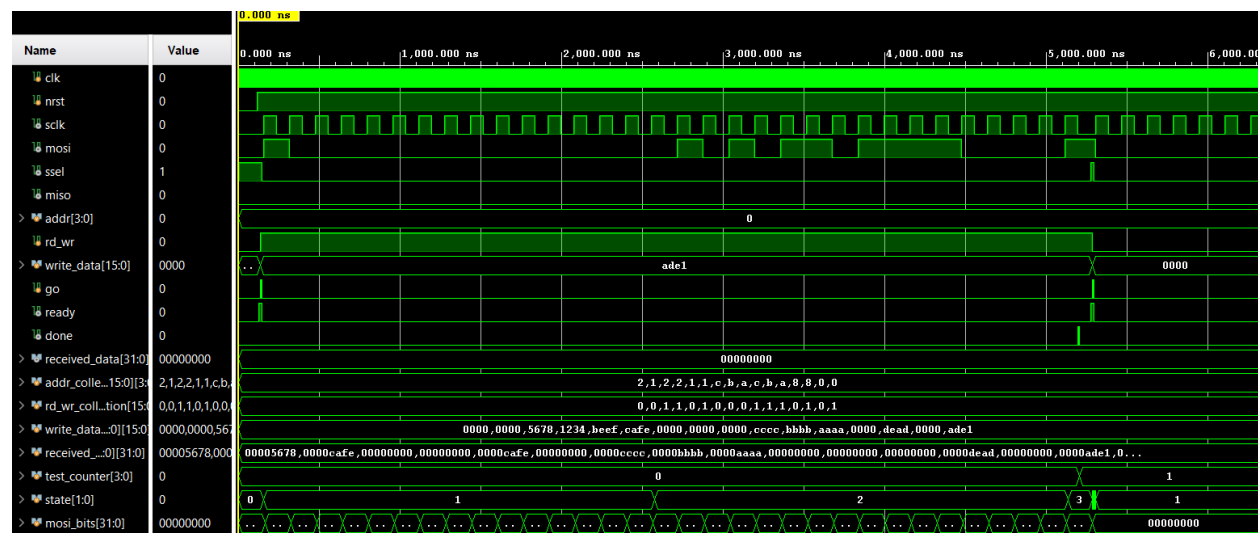
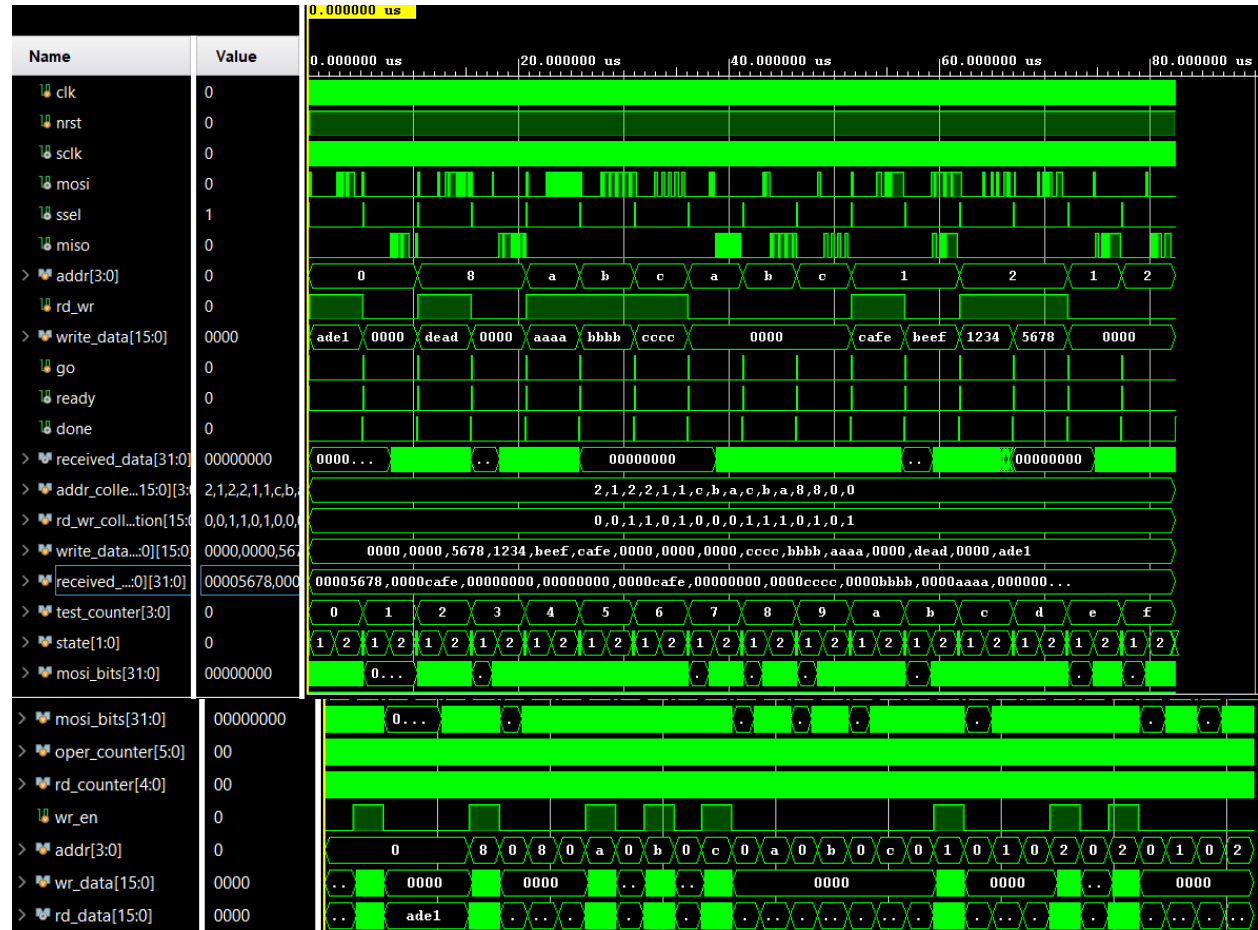
John Danielle T Castor

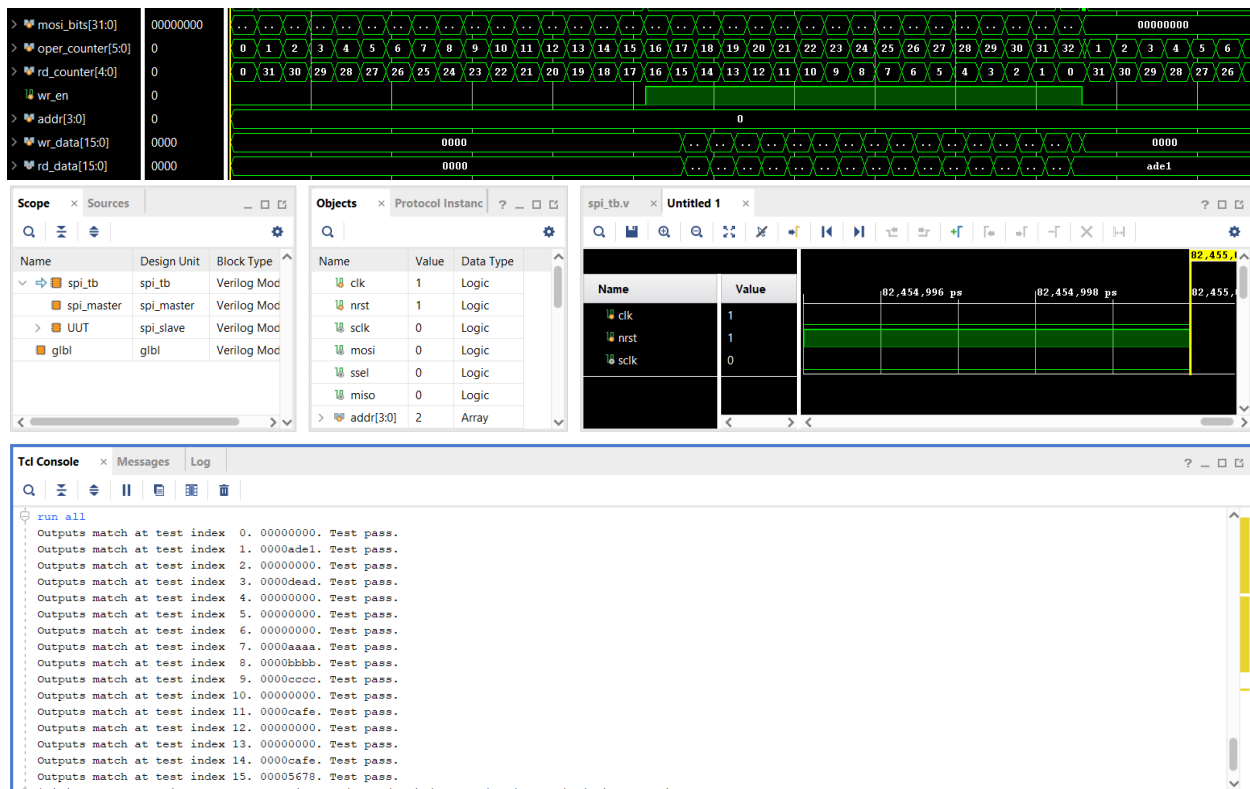
2020-

11-24-23

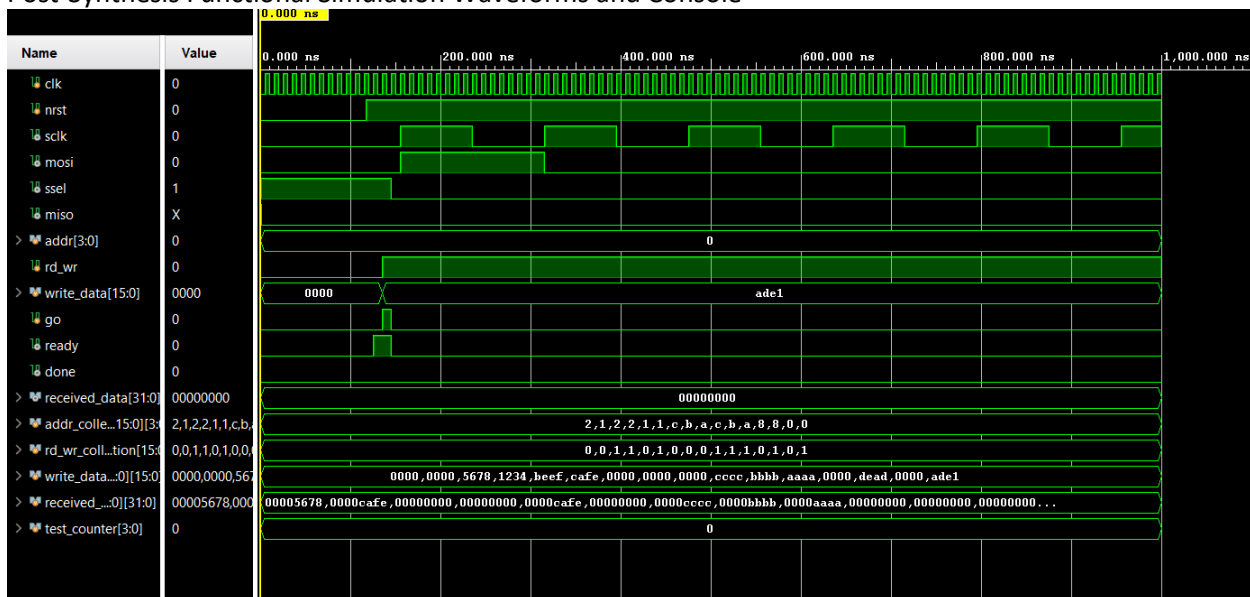
Project 4

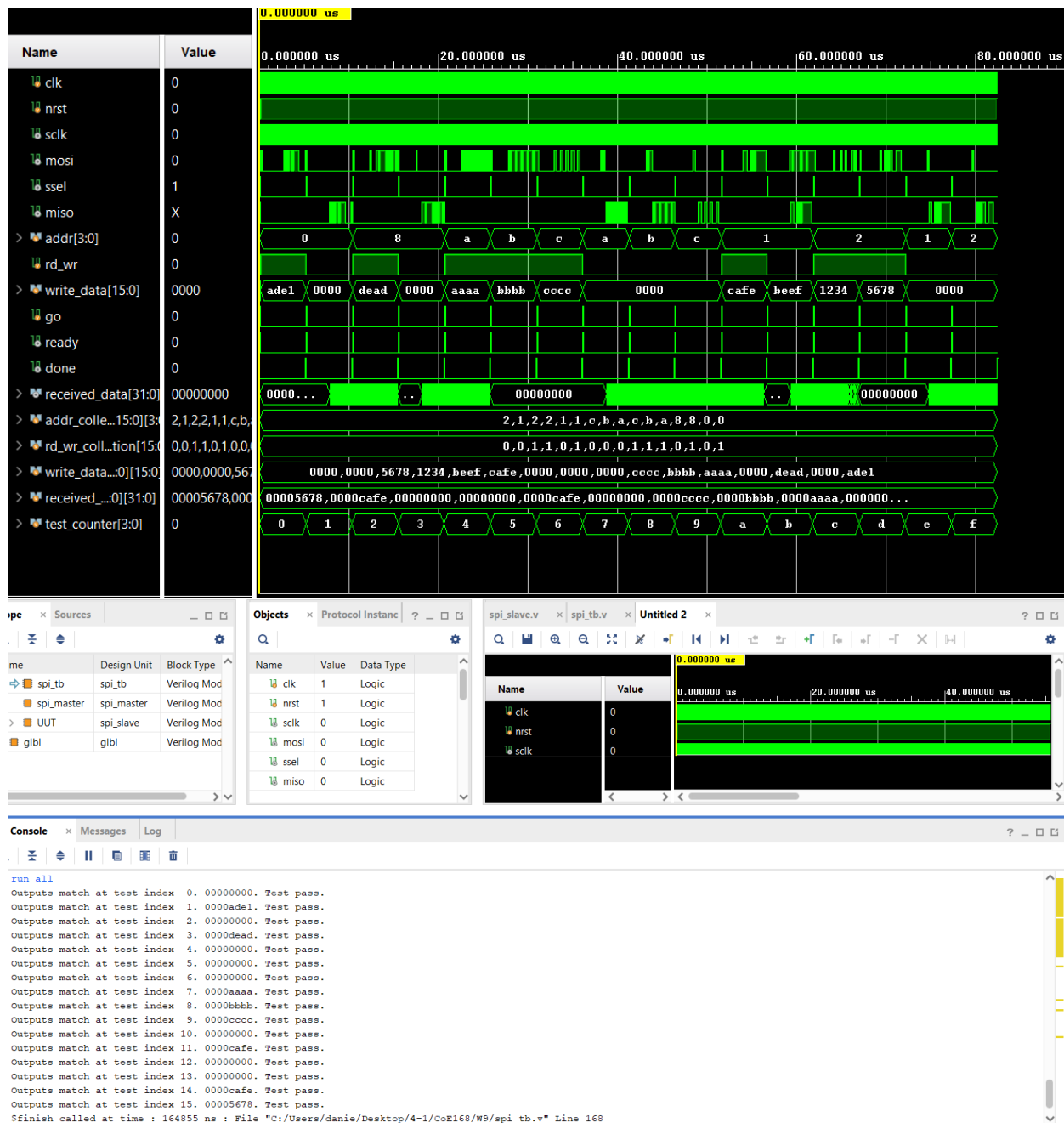
Behavioral Simulation Waveforms and Console





Post-Synthesis Functional Simulation Waveforms and Console





spi_slave.v code
`timescale 1ns / 1ps

module spi_slave
(

input clk,
input nrst,
input sclk,
input mosi,
input ssel,

```

    output reg miso
);

reg [15:0] wr_data;
wire [15:0] rd_data;
reg [3:0] addr;
reg wr_en;

// write your other internal signals here
reg [1:0] state;
reg sclk_prev;
reg ssel_prev;
reg [31:0] mosi_bits;
reg [5:0] oper_counter;
reg [4:0] rd_counter;
reg wr_checker;

// register file instantiation
regfile regfile(
    .clk(clk),
    .nrst(nrst),
    .wr_en(wr_en),
    .addr(addr),
    .wr_data(wr_data),
    .rd_data(rd_data)
);

// you should only use posedge clk. Don't use posedge sclk, or negedge sclk!
always@(posedge clk or negedge nrst) begin
    if (!nrst) begin
        miso <= 0;
        wr_data <= 16'b0000000000000000;
        addr <= 4'b0000;
        wr_en <= 0;

        state <= 2'b00;
        sclk_prev <= 0;
        ssel_prev <= 0;
        mosi_bits <= 32'h00000000;
        oper_counter <= 6'b000000;
        rd_counter <= 6'b100000;
        wr_checker <= 0;
    end else begin
        sclk_prev <= sclk; //01- posedge, won't update right away
        if (sclk_prev == 0 & sclk == 1) begin //01
            mosi_bits <= {mosi_bits[30:0], mosi};
            oper_counter <= oper_counter + 1;
        end
    end
end

```

```

    rd_counter <= rd_counter - 1;
end

if (state == 2'b00) begin
    ssel_prev <= ssel;
    if (ssel_prev == 1 & ssel == 0) begin //Ssel becomes 0 from 1
        state <= 2'b01;
        mosi_bits <= 32'h00000000;
        oper_counter <= 6'b000000;
        rd_counter <= 6'b100000;
    end
    else
        state <= 2'b00;
    end
end

else if (state == 2'b01) begin
    if (oper_counter < 6'b010000)
        state <= 2'b01;
    else if (oper_counter == 6'b010000) begin //Counter = 16, 16th bit
        state <= 2'b10;
        wr_en <= mosi_bits[15];
        addr <= mosi_bits[3:0];
        if (sclk_prev == 0 & sclk == 1 & mosi_bits[15] == 1)
            wr_data <= {wr_data[14:0], mosi};
        else if (sclk_prev == 0 & sclk == 1 & mosi_bits[15] == 0)
            miso <= rd_data[rd_counter];

    end
end

else if (state == 2'b10) begin //Obtain 17th bit
    if (wr_en == 1) begin //Write
        if (oper_counter < 6'b100000 & oper_counter > 6'b010000) begin
            if (sclk_prev == 1 & sclk == 0)
                wr_data <= {wr_data[14:0], mosi};
            state <= 2'b10;
        end
        else if (oper_counter == 6'b100000 & oper_counter > 6'b010000) begin //Counter = 32, 32nd bit,
bit31
            if (sclk_prev == 0 & sclk == 1)
                wr_data <= {wr_data[14:0], mosi};
            ssel_prev <= ssel;
            state <= 2'b11;
        end
    end
    else if (wr_en == 0) begin //Read
        if (oper_counter < 6'b100000 & oper_counter > 6'b010000) begin
            //if (sclk_prev == 1 & sclk == 0)

```

```

        miso <= rd_data[rd_counter];
        state <= 2'b10;
    end
    else if (oper_counter == 6'b100000 & oper_counter > 6'b010000) begin
        //if (sclk_prev == 0 & sclk == 1)
            miso <= rd_data[rd_counter];
            ssel_prev <= ssel;
            state <= 2'b11;
        end
    end
end

else if (state == 2'b11) begin
    ssel_prev <= ssel;
    if (wr_en == 1) begin
        if (sclk_prev == 1 & sclk == 0)
            wr_data <= {wr_data[14:0], mosi};
        end
    else if (wr_en == 0) begin
        if (sclk_prev == 1 & sclk == 0)
            miso <= rd_data[rd_counter];
        end
    end

    if (ssel_prev == 0 & ssel == 1) begin //Ssel becomes 1 from 0
        state <= 2'b00;
        miso <= 0;
        wr_data <= 16'b0000000000000000;
        addr <= 4'b0000;

        wr_en <= 0;
        state <= 2'b00;
        sclk_prev <= 0;
        mosi_bits <= 32'h00000000;
        oper_counter <= 6'b000000;
        rd_counter <= 6'b100000;
    end
    else
        state <= 2'b11;
    end
end

end
end

endmodule

```