ECE 271, Design Project, Group 18

Matthew Miller, Matthew Lohr, Nick Varnum, Brett Stoddard, Brogan Miner

June 9, 2017

Contents

1 Project Description

The purpose of this project was to design digital logic that will be able to take input from various types of controllers and use this input to control an NES game console. The controllers that were offered for this project were an 8-input button board, an N64 controller, and a TV remote that sends a 38kHz wave signal. The chosen controllers were the button board and the TV remote, as there was not enough time to do all three. The general overview for the project can be seen below.

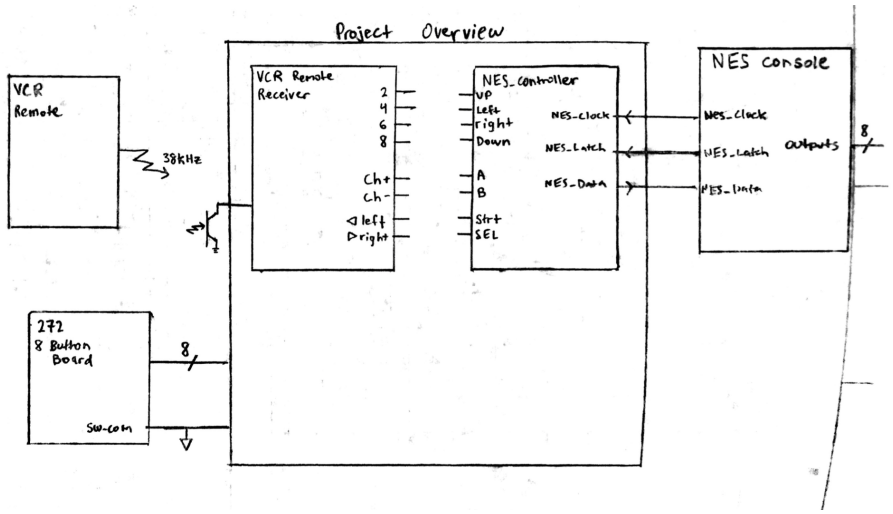

Figure 1: Project Overview

As can be seen from Figure 1, the data inputs for this design are the TV remote signal and the button board inputs. These inputs are processed using digital logic programmed onto a Lattice MachXO3LF FPGA, and the corresponding button assignments on an NES controller are determined. This data is put in a form that the NES will expect, which is an 8-bit input representing the 8 buttons on the NES controller, and the data is outputted to teh NES console data line. For this design, we are assuming that the NES console will correctly process the data as it needs. The job of the design project was merely to get the data in a form that could be read by the NES data line, and this is what was done both with the TV remote and the button board.

Although this project was just a design project, the hardware connections were designed should one want to physically create the design. This hardware diagram can be seen in Figure 2 below. The only connections that need to be made are the button board to the FPGA, the VCR receiver to the FPGA, and the NES console to the FPGA.

As an additional feature for this project we added the ability to chose between using the button board and the TV remote. This is useful if a player had a certain preference between these two practically unusable controllers. This is instantiated in the form of a mux that sits between the button board and the TV remote modules. Holding inputs on an external button board selects inputs from that comes from another.

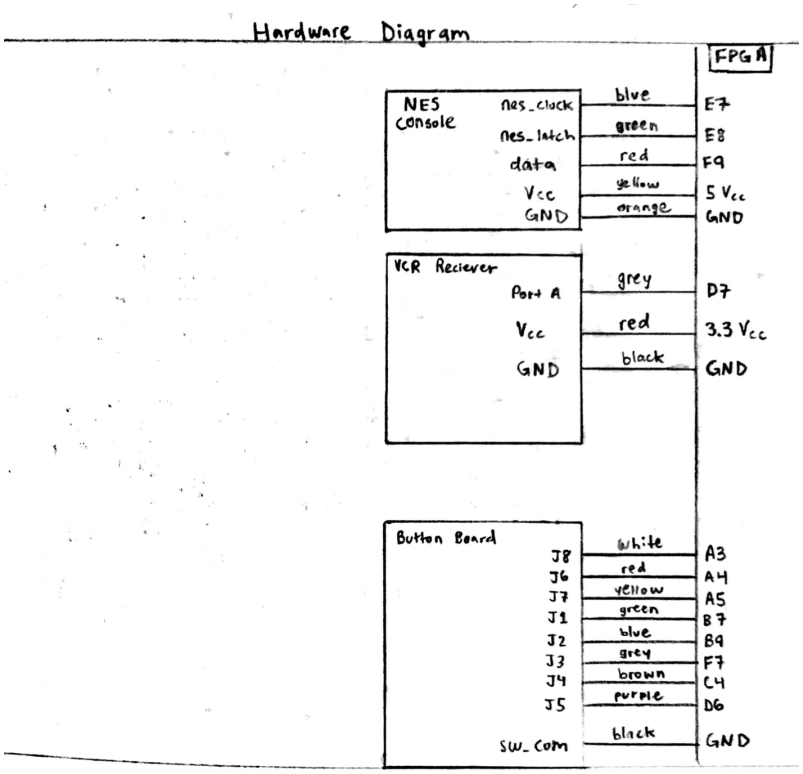

Figure 2: Hardware Diagram

2 High Level Description

This design project was created entirely using SystemVerilog. SystemVerilog is a hardware description language (HDL) that can be synthesized and programmed onto a specified piece of hardware, in this case the MachXO3LF FPGA. The SystemVerilog code is broken up into modules. Each module represents a single task that has inputs and outputs. Below is a breakdown of the SystemVerilog modules that were created for this design project. Figure 3 shows the Most Top module, which is the highest level of our design. The inputs and outputs to the Most Top module are the inputs and outputs to the entire system.

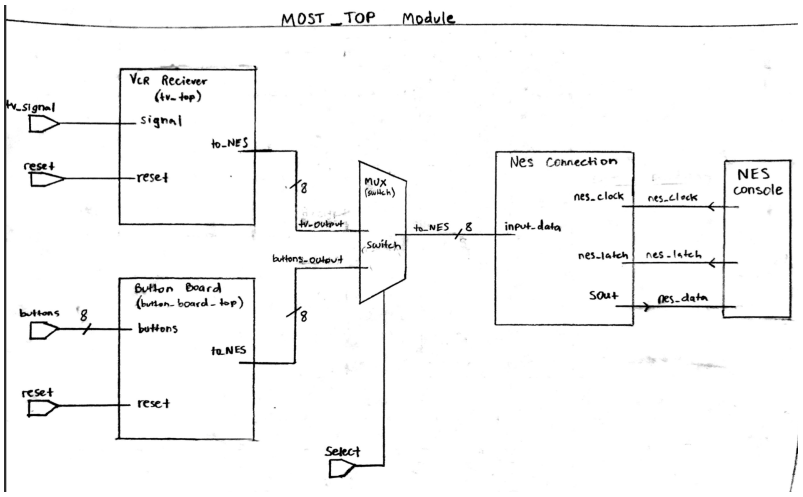

Figure 3: Most Top Module

Inputs: The 8-bit button board input, the serial TV remote input signal, reset signals for both the Button Board Top and TV Remote Top, a switch to decide which controller will be used, the NES Clock, and the NES Latch

Outputs: The NES Data line

The button board buttons are active low, meaning they are represented as a logic "1" when not pressed, and a logic "0" when pressed. The input from the button board is simply the 8-bit combination of which buttons are pressed. For example, if button 4 was pressed, the 8-bit input would be represented as "11101111". The TV remote signal comes in serially, which means one bit at a time. The TV Top module takes care of translating this serial data into a form that can be handled with digital logic. Both the TV Top module and Button Board Top module output an 8-bit number that represents buttons on an NES controller being pressed. A multiplexer uses a the switch input to determine which controller is to be used, and then gives that chosen controller's output to the NES Connection module. The NES Clock and NES Latch signals are what the NES console uses to read the data, and the NES Connection module uses them to parse through the data given from the multiplexer and send the 8-bit data serially to the NES console through the NES Data line.

The Most Top module was tested as a functional unit and the simulation results can be seen below.

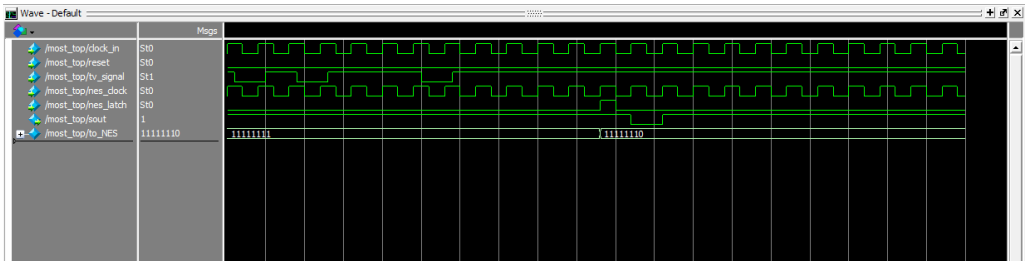

Figure 4: Most Top Simulation Results

The NES Connection module was also tested as a functional unit and the simulation results can be seen below.

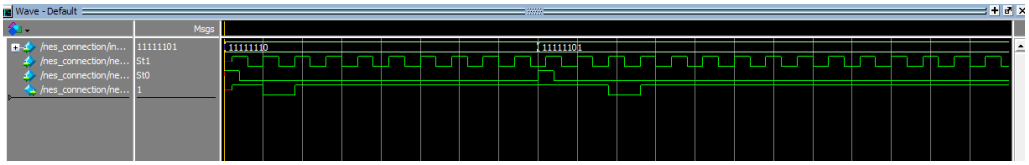

Figure 5: NES Connection Simulation Results

2.1 Functional Unit: TV Remote

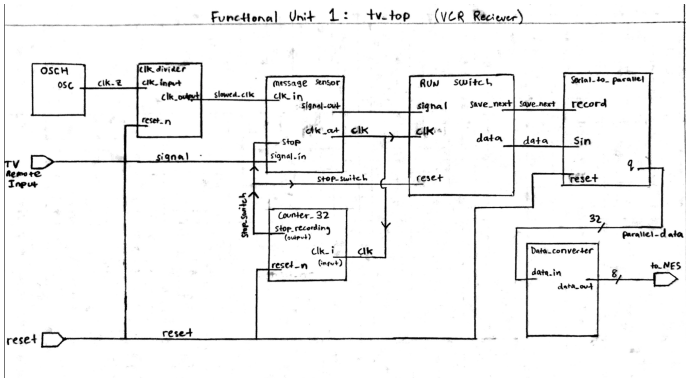

Figure 6: TV Top Module

Inputs: The serial TV remote signal, and reset

Outputs: The 8-bit data representation of buttons pressed on an NES controller

The TV Top module deals entirely with taking in the TV remote wave signal serially and converting into an 8-bit output. First, the data is read serially and converted into a parallel 32-bit value. This 32-bit value is then converted into an 8-bit value that is outputted from the module.

The TV Top module was tested as a functional unit and the simulation results can be seen below.

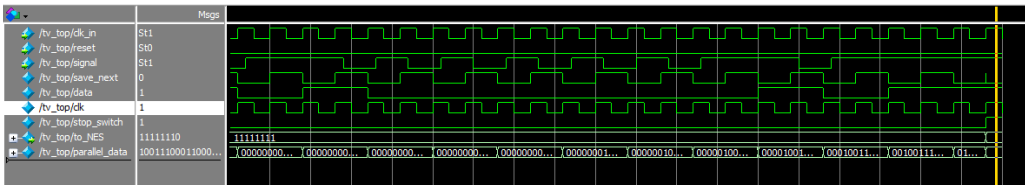

Figure 7: TV Top Simulation Results

2.1.1 Individual Block: Oscillator

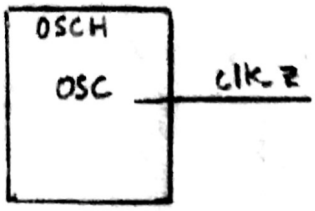

Figure 8: Clock Oscillator Module

Inputs: None

Outputs: The internally generated 2.08MHz clock

Description: This module outputs the clock that is internally generated inside the MachXO2 FPGA, which we chose to run at 2.08MHz.

2.1.2 Individual Block: Clock Divider

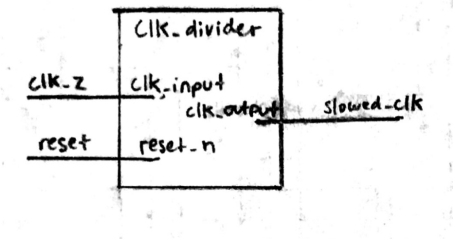

Figure 9: Clock Divider Module

Inputs: The internally generated 2.08MHz clock, and reset

Outputs: A slowed clock running at 1645Hz

Description: This module creates a clock signal based on the internal clock by oscillating an output bit between 0 and 1 after a certain period of the internal clock's oscillation, so that the desired frequency of 1654Hz is achieved.

2.1.3 Individual Block: Counter 32

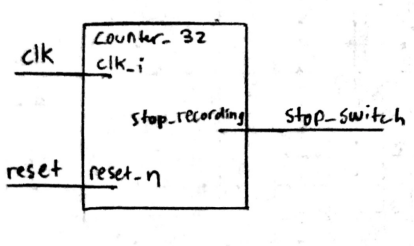

Figure 10: Counter 32 Module

Inputs: The 1645Hz clock, and reset

Outputs: Control that tells to stop reading once 32 bits have been read

Description: This module is a 32-bit counter that counts at 1645Hz. The signal the module outputs becomes a 1 when 32 bits of the TV signal have been read.

2.1.4 Individual Block: Message Sensor

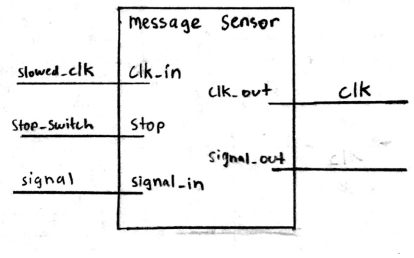

Figure 11: Message Sensor Module

Inputs: The 1645Hz clock, the stop control that says when to stop reading, and the serial signal from the TV remote

Outputs: Output clock signal, and the serial TV remote signal that was read

Description: This module determines when the signal should start being read as data, based on the incoming signal.

2.1.5 Individual Block: Run Switch

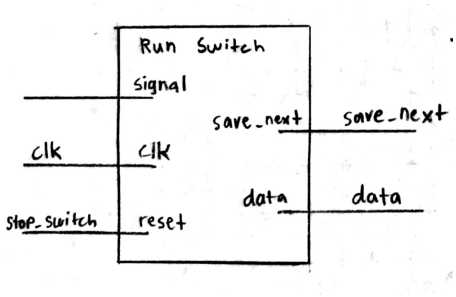

Figure 12: Run Switch Module

Inputs: The 1645Hz clock, the serial signal read from Message Sensor, and a reset signal which is the stop control from Counter 32

Outputs: The current bit read and the next bit that will be read

Description: This module looks at the current bit that is read from the TV signal to determine whether or not data should be read. If the current bit is a logic 1, the following bit is read as part of the 32-bit data, and the signal is read further.

2.1.6 Individual Block: Serial to Parallel

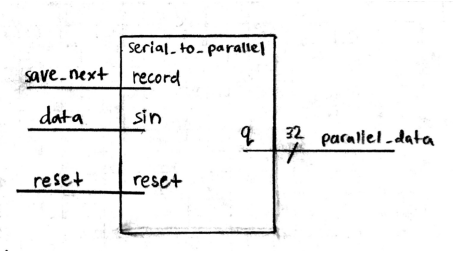

Figure 13: Serial to Parallel Module

Inputs: The data that was read, the data to save, and a reset

Outputs: The 32-bit TV signal

Description: The serial data that is read from Run Switch is all given to this module, which compounds them all into one 32-bit register.

2.1.7 Individual Block: Data Converter

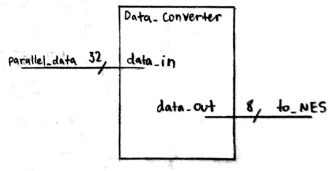

Figure 14: Data Converter Module

Inputs: The 32-bit TV signal

Outputs: The 8-bit representation of NES buttons being pressed, to be sent to NES Connection

Description: This module takes the 32-bit register of TV remote signal data, determines which button is being pressed, and outputs an 8-bit register that correctly corresponds to an NES controller button being pressed.

2.2 Functional Unit 2: Button Board Top

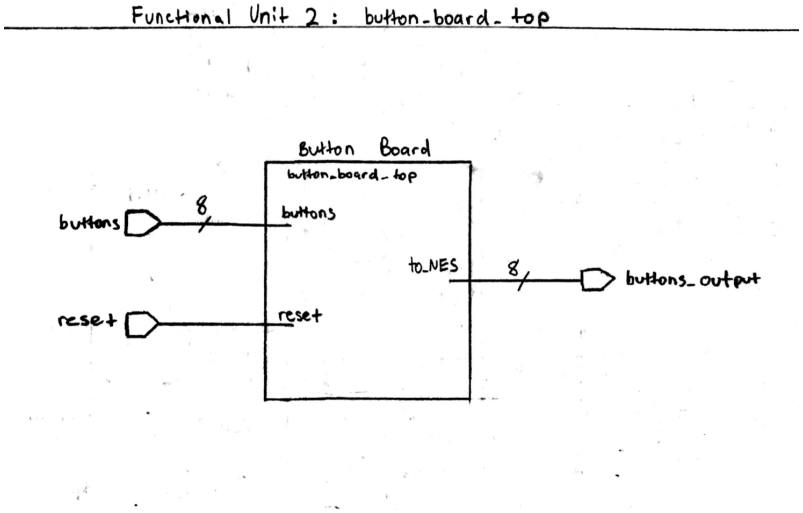

Figure 15: Button Board Top Module

Inputs: The 8-bit data which represents which button is pressed, and a reset

Outputs: An 8-bit register that represents NES controller buttons being pressed

Description: This module is very simple, all it does is take in the button input, and feed it through to NES Connection. Becuase the NES controller has 8 buttons, and the button board has 8 buttons, no change needs to be done to get the input from the button board to the NES Connection shift register.

Because the Button Board Top module did not have any digital logic that needed to be tested, as all it does is feed data from an input to an output (see Appendix A.11), it was decided that testing was not necessary so no simulation was done on this functional unit.

A SystemVerilog Files

A.1 Most Top Module

```
1  module most_top(
2                  input logic reset,
3          input logic tv_signal,
4          input logic [7:0] buttons,
5          input logic switch,
6          input logic nes_clock,
7          input logic nes_latch,
8          output logic sout
9          );
10
11     logic [7:0] to_NES;
12     logic [7:0] tv_output;
13     logic [7:0] buttons_output;
14     logic clock_in;
15     logic  fClock;
16     OSCH #("2.08") osc_int (
17                            .STDBY(1'b0),
18                            .OSC(fClock),
19                            .SEDSTDBY()
20         );
21     clock_divider  cd(
22         .clk_input(fClock),
23                  .reset(reset),
24                  .clk_output(clock_in)
25     );
```

```
26
27          tv_top tv_receiver(
28                  .reset(reset),
29                  .clk_in(clock_in),
30              .sig(tv_signal),
31              .to_NES(tv_output)
32              );
33
34      button_board_top button_receiver(
35              .buttons(buttons),
36              .to_NES(buttons_output)
37              );
38
39      assign to_NES = switch ? tv_output : buttons_output;
40
41      nes_connection connection(
42                  .nes_clock(nes_clock),
43              .nes_latch(nes_latch),
44              .input_data(to_NES),
45              .nes_data(sout)
46              );
47
48  endmodule
```

A.2 TV Top Module

```
1   module tv_top(
2           input logic reset,signal,
3           input logic clk_in,
4           output logic  [7:0] to_NES       //change 8 to how long the messages
5           );
6
7   //TODO: create module that senses when new inputs come in
8
9                   logic clk, clk_z;
10                  logic save_next;
11                  logic data;
12                  logic stop_switch;
13                  logic [31:0] parallel_data;
14
15
16                  message_sensor m_s (
17                          .signal(signal),
18                          .stop(stop_switch),
19                          .clk_in(clk_in),
20                          .clk_out(clk),
21                          .reset(reset)
22                          );
23
24                  run_switch con_1(
25                          .signal(signal),
26                          .clk(clk),
27                          .reset(stop_switch),
28                          .data(data),
29                          .save_next(save_next)
30                          );
31
32  //TODO: change the clock speed to sample rate
33
34
35
36                  counter_32 count_32(
37                          .clk_i(save_next),                          //often, "tags" are added to variables
                                to denote what they do for the user
38                          .reset_n(reset),                                    //an active high signal ("resets
                                when high"); later will come from message detector
39                          .stop_recording(stop_switch)    //high when full
40                          );
41
42                  data_converter d_con(
43                          .data_in(parallel_data),
44                          .data_out(to_NES)                   //to NES converter module
45                          );
46
47                  serial_parallel_shiftreg bit_reg(
48                          .record(save_next),                 //save_next acts as a clock on negedge; negedge
                                save_next shifts bits over
49                          .reset(reset),
50                          .sin(data),                                 //from run_switch
51                          .q(parallel_data)
52                          );
53
54  endmodule
```

A.3 Clock Divider Module

```
1   module clock_divider(
2                   input logic clk_input,
3                   input logic reset,
4                   output logic clk_output
5   );
6
7           logic [13:0] count;
8
9           always_ff @ (posedge clk_input, posedge reset_n)
10                  begin
11                          count <= count + 1;
12                          if(reset)
13                                  begin
14                                          clk_output <= 0;
15                                          count <= 0;
16                                  end
17                          else
18                                  if(count >= 1264)
19                                          begin
20                                                  clk_output <= ~clk_output;
21                                                  count <= 0;
22                                          end
23                  end
24
25  endmodule
```

A.4 Counter 32 Module

```
module counter_32(
        input logic clk_i,              //often, "tags" are added to variables to denote what they do
             for the user
        input logic reset_n,    //here, 'i' is used for input and 'o' for the output, while 'n'
             specifies
                                                //an active low signal ("not")
        output logic stop_recording  //from 0-30 low, when 31 high; will connect to run_switch
                );

                logic [5:0]count;       //register stores the counter value so that it can be modified
                                        //on a clock edge. Register size needs to store as large of a
                                        //number as the counter reaches. Here, 2^(5) = 16,384

                always_ff @ (posedge clk_i, posedge reset_n)    //counts up to 32 then stops
                        if(reset_n)
                                begin
                                count <= 0;             //active high
                                stop_recording <=0;
                                end
                        else
                                if(count >= 31)  //POTENTIAL PROBLEM
                                        stop_recording<=1;
                                else
                                        count <= count+1;
endmodule
```

A.5 Start/Stop Module Module

```
module message_sensor (
        input logic clk_in, signal,stop,reset,
        output logic clk_out
        );

        logic signal_going; //1 for recording, 0 for not recording
        logic signal_end;

        start_sensor s_s( //short bitregister that checks for two rising edges
                .signal(signal),
                .signal_end(signal_end), //resets when signal ends
                .signal_start(signal_going)
                );

        end_sensor e_s(
                .end_(signal_end),
                .reset(reset),
                .stop(stop)                     //from 32 bit counter
                );


        always_comb
                if(signal_going)
                        clk_out = clk_in;
                else
                        clk_out = 0;


endmodule
```

A.6 Start Message Module

```
module start_sensor (
        input logic signal,signal_end,
        output logic signal_start
        );
        logic edges;

        always@(posedge signal) //counts 2 posedges
                if (!signal_end) //
                        begin
                        edges <=0; //reset after signal ends
                        signal_start <=0;
                        end
                else if (!edges)
                        begin
                        edges <= edges+1;
                        signal_start<=0;
                        end
                else
                        begin
                        signal_start<=1;
                        edges <= edges+1;
                        end
endmodule
```

A.7 End Message Module

```
module end_sensor (
        input logic stop,reset,
        output logic end_
        );

        always @ (posedge stop,posedge reset)

                if(reset) end_<=1;
                else if (!end_) end_<=1;
                else if(stop) end_<= 0;
                else        end_<=0;

endmodule
```

A.8 Run Switch Module

```
module run_switch (
        input logic clk, signal, reset,
        output logic data, save_next
        );
        //logic save_next;
        always@(posedge clk, posedge reset)
                if(reset)
                        save_next = 0;
                else
                        case (signal)
                        0:
                                begin
                                if(save_next)
                                        begin
                                        data = 0;
                                        save_next = 0;
                                        end
                                end
                        1:
                                if(save_next)
                                        begin
                                        data = save_next;
                                        save_next = 0;
                                        end
                                else
                                        save_next = 1;
                endcase

endmodule
```

A.9 Serial to Parallel Module

```
module serial_parallel_shiftreg #(parameter N=32)
                (input logic record, reset, sin,
                output logic [N-1:0] q
                );

                logic sout;

        always_ff@(negedge record, posedge reset)
                if (reset)              q<=0;
                else                    q<={q[N-2:0],sin};

        assign sout = q[N-1];
endmodule
```

A.10 Data Converter Module

```
module data_converter(
        input logic [31:0]data_in,
    output logic [7:0]data_out
    );
    always_comb
    case(data_in)     //4'h specifies hex numbers
        32'h9C63E817: //Channel + on tv remote
                data_out = 8'b11111110;
        32'h9C6318E7: //Channel - on tv remote
                data_out = 8'b11111101;
        32'h9C63F807: //Right on tv remote
                data_out = 8'b11111011;
        32'h9C637887: //Left on tv remote
                data_out = 8'b11110111;
        32'h9C63807F: //2 on tv remote
                data_out = 8'b11101111;
        32'h9C63906F: //8 on tv remote
                data_out = 8'b11011111;
        32'h9C6320DF: //4 on tv remote
                data_out = 8'b10111111;
        32'h9C63609F: //6 on tv remote
                data_out = 8'b01111111;
        //TODO: add case for holding button
        default:
                data_out = 8'b11111111;
    endcase
endmodule
```

A.11 Button Board Top Module

```
module button_board_top(
                input logic [7:0] buttons,
        output logic [7:0] to_NES
        );

        always_comb
        to_NES <= buttons;

endmodule
```

A.12 NES Connection Module

```
 module nes_connection(
                input logic [7:0] input_data,
                input logic nes_clock,
                input logic nes_latch,
                output logic nes_data
);

        reg [4:0]count;

        always_ff@(posedge nes_clock)
                begin
                        if(nes_latch)
                                count <= 0;
                        else
```

```
                                count <= count + 1;
                        case(count)
                                0:
                                        nes_data <= input_data[0];
                                1:
                                        nes_data <= input_data[1];
                                2:
                                        nes_data <= input_data[2];
                                3:
                                        nes_data <= input_data[3];
                                4:
                                        nes_data <= input_data[4];
                                5:
                                        nes_data <= input_data[5];
                                6:
                                        nes_data <= input_data[6];
                                7:
                                        nes_data <= input_data[7];
                                default:
                                        nes_data <= 1;
                        endcase
                end

endmodule
```

B Simulation Files (Do scripts)

B.1 Top Module

```
add wave clock_in
add wave reset
add wave tv_signal
add wave nes_clock
add wave nes_latch
add wave sout
add wave to_NES

force switch 1 0

force clock_in 1 10, 0 30 -repeat 40
force nes_clock 1 10, 0 30 -repeat 40

force nes_latch 0 0
force nes_latch 1 2530
force nes_latch 0 2550

force reset 1 0
force reset 0 20

force tv_signal 1 20
force tv_signal 1 60
force tv_signal 1 100
force tv_signal 0 140
force tv_signal 1 180
force tv_signal 0 220
force tv_signal 1 260
force tv_signal 1 300
force tv_signal 1 340
force tv_signal 1 380
force tv_signal 1 420
force tv_signal 1 460
force tv_signal 1 500
force tv_signal 0 540
force tv_signal 1 580
force tv_signal 0 620
force tv_signal 1 660
force tv_signal 0 700
force tv_signal 1 740
force tv_signal 1 780
force tv_signal 1 820
force tv_signal 1 860
force tv_signal 1 900
force tv_signal 0 940
force tv_signal 1 980
force tv_signal 0 1020
force tv_signal 1 1060
force tv_signal 0 1100
force tv_signal 1 1140
force tv_signal 1 1180
force tv_signal 1 1220
force tv_signal 1 1260
force tv_signal 1 1300
force tv_signal 1 1340
force tv_signal 1 1380
force tv_signal 1 1420
force tv_signal 1 1460
force tv_signal 1 1500
force tv_signal 1 1540
force tv_signal 0 1580
force tv_signal 1 1620
force tv_signal 1 1660
force tv_signal 1 1700
force tv_signal 0 1740
force tv_signal 1 1780
force tv_signal 0 1820
force tv_signal 1 1860
force tv_signal 0 1900
force tv_signal 1 1940
force tv_signal 0 1980
force tv_signal 1 2020
force tv_signal 0 2060
force tv_signal 1 2100
force tv_signal 0 2140
force tv_signal 1 2180
force tv_signal 1 2220
force tv_signal 1 2260
force tv_signal 0 2300
force tv_signal 1 2340
force tv_signal 1 2380
force tv_signal 1 2420
force tv_signal 1 2460
```

```
force tv_signal 1 2500
force tv_signal 1 2540

run 3000 ps
```

B.2 NES Connection Functional Unit

```
add wave input_data
add wave nes_clock
add wave nes_latch
add wave nes_data

force nes_clock 1 10, 0 30 -repeat 40

force nes_latch 1 0
force nes_latch 0 20

force input_data 10#254# 0
force nes_latch 1 400
force nes_latch 0 420
force input_data 10#253# 400

run 800 ps
```

B.3 VCR Receiver Functional Unit

```
add wave clk_in
add wave reset
add wave signal
add wave save_next
add wave data
add wave clk
add wave stop_switch
add wave to_NES
add wave parallel_data

force clk_in 1 10, 0 30 -repeat 40

force reset 1 0
force reset 0 20

force signal 1 20
force signal 1 60
force signal 1 100
force signal 0 140
force signal 1 180
force signal 0 220
force signal 1 260
force signal 1 300
force signal 1 340
force signal 1 380
force signal 1 420
force signal 1 460
force signal 1 500
force signal 0 540
force signal 1 580
force signal 0 620
force signal 1 660
force signal 0 700
force signal 1 740
force signal 1 780
force signal 1 820
force signal 1 860
force signal 1 900
force signal 0 940
force signal 1 980
force signal 0 1020
force signal 1 1060
force signal 0 1100
force signal 1 1140
force signal 1 1180
force signal 1 1220
force signal 1 1260
force signal 1 1300
force signal 1 1340
force signal 1 1380
force signal 1 1420
force signal 1 1460
force signal 1 1500
force signal 1 1540
force signal 0 1580
force signal 1 1620
force signal 1 1660
force signal 1 1700
force signal 0 1740
force signal 1 1780
force signal 0 1820
force signal 1 1860
force signal 0 1900
force signal 1 1940
force signal 0 1980
force signal 1 2020
force signal 0 2060
force signal 1 2100
force signal 0 2140
force signal 1 2180
force signal 1 2220
force signal 1 2260
force signal 0 2300
force signal 1 2340
force signal 1 2380
force signal 1 2420
force signal 1 2460
force signal 1 2500
force signal 1 2540

run 2550 ps
```

C Synthesized Design

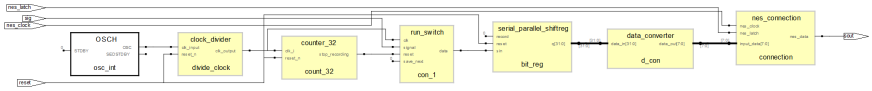

Figure 16: TV Top Synthesized

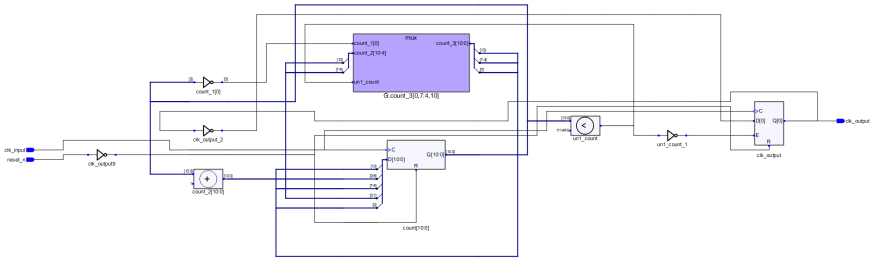

Figure 17: Clock Divider Synthesized

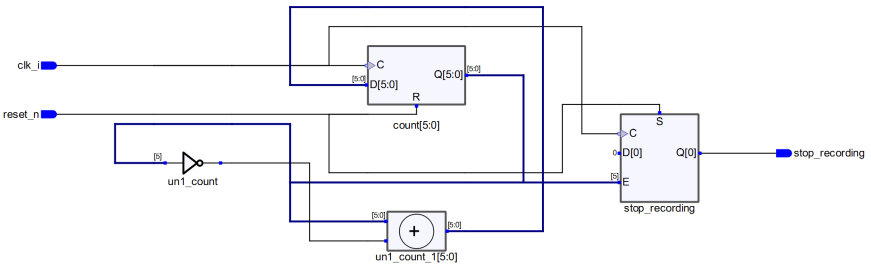

Figure 18: Counter 32 Synthesized

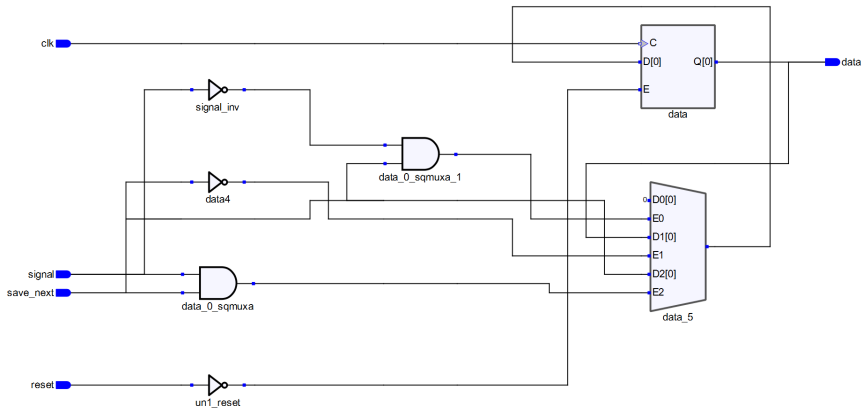

Figure 19: Run Switch Synthesized

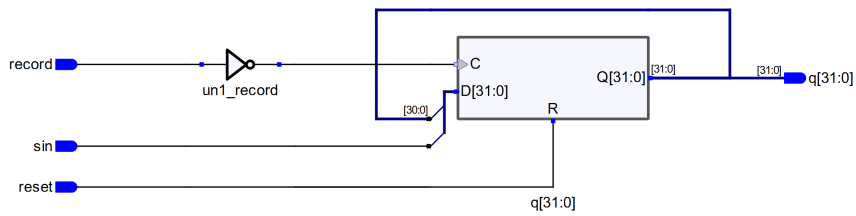

Figure 20: Serial to Parallel Synthesized

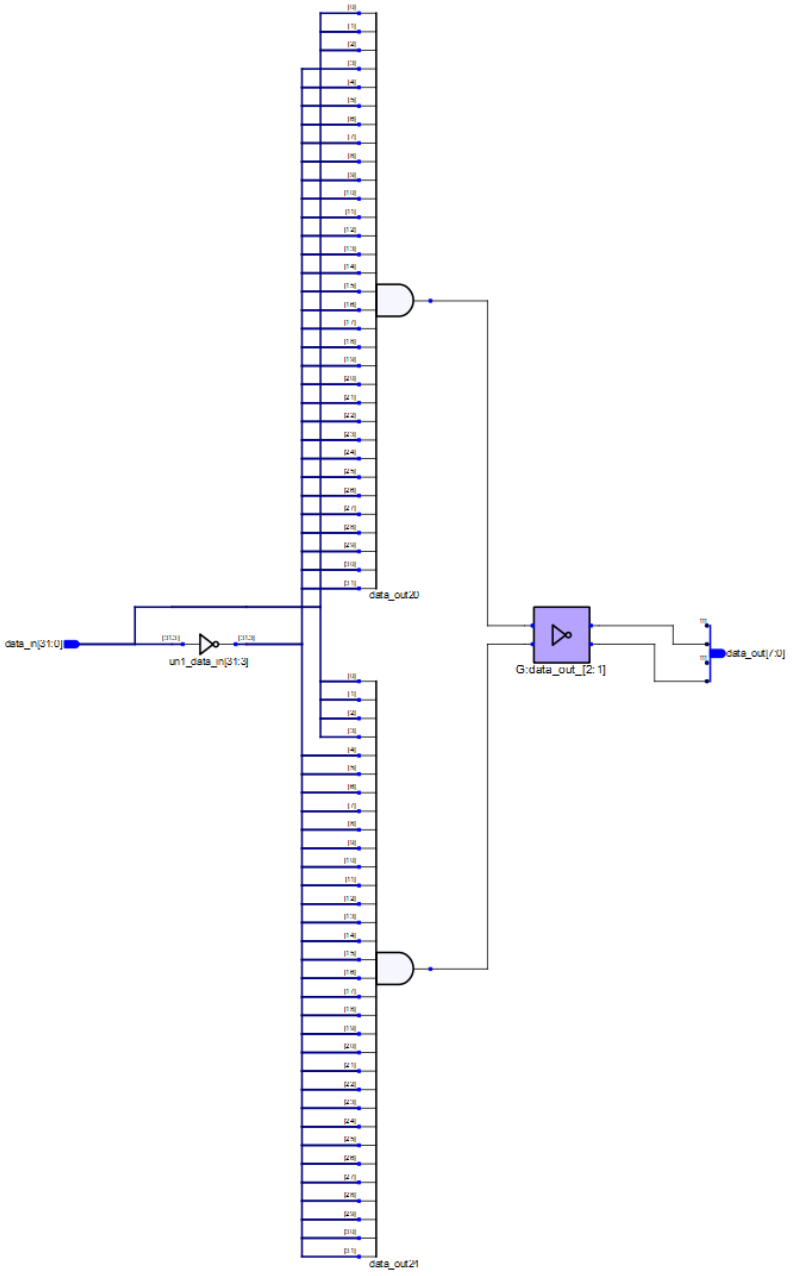

Figure 21: Data Converter Synthesized

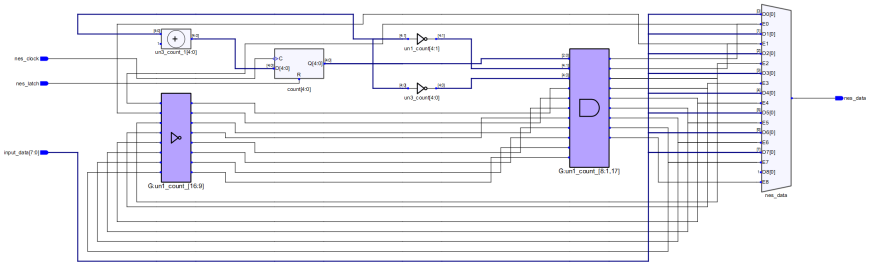

Figure 22: NES Connection Synthesized